Виктор Осокин Эльяр Гасанов

ДВИНЕМНАУКУ

[система ведения научно-образовательных процессов]

геолокация
интерфейс **iOS**
база данных гистограмма
КЛАССИФИКАЦИЯ
tf-idf поиск схожих изображений MFCC
ПОИСКОВЫЕ СИСТЕМЫ
индекс цитирования HITS инвертированный индекс сессии
PageRank css apache **ВИКИПЕДИЯ**
ссылочное ранжирование MySQL html php
DVINEMNAUKU.RU
одностраничные веб-приложения javascript json
AJAX **ОНЛАЙН-КАРТЫ** rest api
агрегатор новостей метод расстояний
КЛАСТЕРИЗАЦИЯ
метод k-средних точки на карте
авторизация фото активити
ANDROID
фрагменты

ver 1.0/09/15

Корпоративная информационная система крупной компании - это живой организм, непродуманное добавление новых модулей в который может легко сломать его функционирование. Программы Виктора Осокина отличаются тем, что они реализуют ровно тот функционал, который нужен для корректной автоматизации необходимых бизнес-процессов компании, но при этом естественным образом интегрируются в существующую корпоративную информационную систему.

HYUNDAI

Александр Нефёдов,
директор по развитию дилерской сети Хендэ Мотор СНГ

Одним из результатов Виктора Осокина стала созданная разработчиками компании «Два Облака» в сотрудничестве с R&D департаментом ТПХ «Русклимат» уникальная программа подбора климатического оборудования. Интуитивно понятный интерфейс, кроссплатформенность и совместимость с AutoCAD, возможность выбора систем вентиляции и кондиционирования из широчайшего ассортимента ТПХ «Русклимат» из любой точки мира, огромный потенциал внутреннего развития программы подбора климатического оборудования - все эти возможности вывели процесс формирования технико-коммерческого предложения на качественно новый уровень.

РУСКЛИМАТ

Алексей Николаев,
финансовый директор Русклимат Вент

В непростых экономических условиях на первое место выходят показатели эффективности ведения бизнеса, а если предприятие выпускает инновационное инженерное оборудование - то качественная функция IT становится залогом успешной работы. Благодаря Виктору Осокину и компании «Два Облака» нам удалось минимизировать затраты на расчет и подбор производимого нами оборудования, а профессионализм его команды позволил реализовать самые сложные с инженерной точки зрения алгоритмы. Благодаря Виктору нашей компании удалось максимизировать эффективность взаимодействия между нами и нашими клиентами.

НОВЫЕ ПРОМЫШЛЕННЫЕ ТЕХНОЛОГИИ
ДИВИЗИОН КЛИМАТИКА

Михаил Марьяхин,
генеральный директор НПТ-Климатика

ISBN 978-5-9907270-0-7

Виктор Осокин

Окончил механико-математический факультет МГУ (2007). Кандидат физико-математических наук (2011).

С 2011 года работает на кафедре Математической теории интеллектуальных систем механико-математического факультета МГУ. В Московском университете читает курсы по веб-технологиям, мобильным технологиям, информационному поиску, машинному обучению.

С 2009 года руководит разработкой корпоративных систем на веб и мобильных технологиях. Разработанные им системы ежедневно используют более сотни российских компаний.

Основатель системы ведения научно-образовательных процессов ДвинемНауку.

ДВИНЕМНАУКУ [система ведения научно-образовательных процессов]

Виктор Осокин Эльяр Гасанов

ver 1.0/09/15

Эльяр Гасанов

Окончил факультет вычислительной математики и кибернетики МГУ (1982). Кандидат физико-математических наук (1986), доктор физико-математических наук (1999).

Профессор кафедры Математической теории интеллектуальных систем механико-математического факультета МГУ (2003).

В Московском университете читает курсы "Теория интеллектуальных систем", "Теория баз данных и информационного поиска", "Дополнительные главы дискретной математики и математической кибернетики" и ведет спецсеминар "Вопросы сложности алгоритмов поиска."

Подготовил 14 кандидатов наук. Опубликовал более 150 научных работ, среди которых 3 монографии, 4 учебных пособия и более 50 патентов США.

Московский Государственный Университет
имени М.В. Ломоносова

ДВИНЕМНАУКУ [система ведения научно-образовательных процессов]

Москва 2015

УДК 004.42
ББК 32.973.26-018
О75

ДВИНЕМНАУКУ [система ведения научно-образовательных процессов] / В.В. Осокин, Э.Э. Гасанов. — М.: Интеллектуальные системы, 2015. — 272 с. — ISBN 978-5-9907270-0-7

Данная книга посвящена описанию некоторых научно-образовательных процессов, проводимых авторами на механико-математическом факультете и в филиалах МГУ им. М.В. Ломоносова. Для онлайн-сопровождения этих процессов используется система ДвинемНауку, которая и дала название книге.

Описывается создание базовых веб-сайтов, одностраничных веб-приложений на примере онлайн-карт, рассматривается устройство поисковых систем, методы кластеризации точек на карте и новостей, методы классификации картинок и музыкальных файлов, приводятся примеры создания iOS и Android приложений.

Рецензент: доктор физико-математических наук, академик, профессор В.Б. Кудрявцев.

ISBN 978-5-9907270-0-7

Содержание

Предисловие от Виктора Осокина

Еще не прошло и 10 лет с тех пор, как интернет стал массовым и доступным в России, что, по моему мнению, принципиально изменило подходы к процессу обучения. Дело в том, что сейчас благодаря поисковым системам и википедии заинтересованный студент, независимо от уровня знаний, может сам начать свой процесс познания интересующей его тематики. Для этого ему достаточно набрать интересующий вопрос в поисковой строке, после чего ему доступны и текстовые, и видеоматериалы. Единственная проблема — таких материалов слишком много, в них легко потеряться. Задача обучения, по моему мнению — научить студента легко путешествовать по этому морю знаний. Все, что для этого нужно — это пройти совместно с каждым студентом по каким-то путям в этом море, разобрать примеры, которые в дальнейшем позволят ему двигаться самому. Именно этим мы руководствовались как при создании системы ДвинемНауку, так и при создании рассмотренных в данной книге задач. Мы не пытались создать справочник по веб, мобильным и поисковым технологиям, мы хотели показать варианты полноценного решения известных прикладных задач, оставив читателю разбираться с конкретными особенностями тех или иных языков и технологий.

Я посвящаю данную книгу своим родителям, Владимиру Викторовичу и Алле Васильевне Осокиным. Папа, мама, каждую секунду своей жизни я чувствовал вашу неиссякаемую поддержку, заботу и желание помочь.

Я невероятно рад, что наконец-то получил возможность написать книгу с моим Учителем Эльяром Эльдаровичем Гасановым.

Я искренне благодарен бессменному руководителю и символу нашей кафедры Валерию Борисовичу Кудрявцеву за возможность развивать и продвигать в МГУ интересные мне темы и подходы.

Есть еще три человека, без которых данная книга не появилась бы на свет. Это Рустам Фахриддинович Алимов, Виталий Анатольевич Бендик и Замира Анваровна Ниязова. Многие годы мы работаем с вами бок о бок, спасибо, что оказали неоценимую помощь при подготовке материалов для книги.

Посвящается моим родителям
Владимиру Викторовичу Осокину
и
Алле Васильевне Осокиной

Предисловие от Эльяра Гасанова

Написанию этой книги способствовали три счастливых события, без которых эта книга не состоялась бы.

Первым событием является факт создания на механико-математическом факультете МГУ в 1991 году кафедры математической теории интеллектуальных систем. В свою очередь кафедра возникла на базе лаборатории проблем теоретической кибернетики, которая занималась прикладными исследованиями для оборонной промышленности. Поэтому с самого своего появления кафедру определяли 3 основных вектора развития: фундаментальные исследования, свойственные всем кафедрам мех-мата, прикладные разработки – наследие лаборатории ПТК, и подготовка студентов математиков с навыками инженеров IT технологий. Последнее является главной особенностью и предназначением кафедры МаТИС.

Вторым счастливым событием явилось то, что в 2004 году ко мне подошел студент-второкурсник и попросил быть его научным руководителем. Этим студентом был Виктор Осокин. Когда мы с ним стали придумывать задачу для его студенческой научной работы, я узнал о его страстном увлечении веб-программированием. В то время на мех-мате не было курса по веб-технологиям, и встреча с Виктором зародила во мне надежду, что со временем вокруг Виктора можно будет сформировать новое направление для кафедры и факультета – веб-технологии. К счастью этим надеждам суждено было сбыться. Виктор успешно закончил университет. Его дипломная работа была опубликована в реферируемом журнале. Он поступил в аспирантуру и через 3 года успешно ее завершил. Его кандидатская диссертация по машинному обучению, которую он защитил в 2011 году, стала победителем конкурса «Ломоносов-2011». По сложившейся на кафедре МаТИС традиции Виктор, как победитель конкурса «Ломоносов», был оставлен работать на кафедре. Первым серьезным делом Виктора на кафедре были подготовка и чтение курса «Современные технологии проектирования и продвижения сайтов». Тогда же начала создаваться онлайн система поддержки учебного процесса dvinemnauku.ru.

Третьим счастливым фактором, способствовавшим рождению этой книги, явилось появление в 2006 году филиала МГУ в г. Ташкенте, и то, что в 2008 году филиал возглавил заведующий кафедрой МаТИС академик В.Б.Кудрявцев. Так в сфере влияния нашей кафедры оказал-

ся ташкентский филиал, и наша кафедра стала базовой для факультета прикладной математики и информатики. Кафедра стала целенаправленно изменять программу факультета ПМиИ, делая ее отвечающей самым современным тенденциям науки и технологий, сбалансировала объемы непрерывной и дискретной математики, создала новые курсы по самым последним достижениям науки и инженерии. В том числе создан курс «Технологии проектирования сайтов». К счастью, студенты ташкентского филиала оказались жадными до знаний, учились с увлечением и добросовестностью. Вокруг Виктора сформировалась большая группа студентов, увлеченная данным направлением. Через некоторое время веб-программирование, а затем и программирование для мобильных устройств, было включено в программу практикума по программированию. Сейчас в филиале МГУ в г. Ташкенте практикум по программированию присутствует в каждом из 8 семестров, и в каждом семестре часть тем отдана веб-технологиям. По сути данная книга составлена из тем, изучаемых в рамках практикума, и каждая глава книги содержит материал, изучаемый в соответствующем семестре.

И практикум, и новое направление по веб-технологиям активно развиваются, поэтому будет развиваться и эта книга. Самую актуальную версию этой книги вы всегда сможете найти на сайте dvinemnauku.ru.

Хочется пожелать успехов всем, кто станет изучать этот курс.

С уважением, Эльяр Гасанов.

Введение

Предлагаемая читателю книга посвящена описанию некоторых научно-образовательных процессов, проводимых авторами на механико-математическом факультете и в филиалах МГУ им. М.В. Ломоносова. Поскольку данные научно-образовательные процессы ведутся во многом удаленно, сложно представить их без онлайн-сопровождения в некоторой системе. Для этих целей используется система онлайн-сопровождения научно-образовательных процессов ДвинемНауку [1], которая и дала название книге.

В первой главе мы напишем простейшее веб-приложение, аналог Википедии. Текст каждой страницы нашей версии Википедии хранится в базе данных. Текст содержит ссылки на другие страницы. Мы реализуем функционал редактирования своих страниц авторизованными пользователями системы, как и функционал добавления новых страниц.

Во второй главе мы напишем так называемое одностраничное веб-приложение. Известными примерами одностраничных приложений являются ВКонтакте, Яндекс-карты, Яндекс-музыка, Gmail и многие другие. Особенностью одностраничных приложений является то, что основная работа по созданию интерфейса пользователя возлагается в таких приложениях не на серверный язык (такой, как PHP), а на клиентский (например, JavaScript). Сервер выполняет только роль звена, которое в нужный момент отдает клиенту (браузеру) данные. В ходе работы будет реализован функционал веб-карт. Будет рассмотрен метод отрисовки цельной карты из маленьких картинок (тайлов), подгрузка необходимых участков по мере передвижения по карте и изменения масштаба отображения карты. В разработанной веб-карте будет выведена карта Москвы. Также на данной карте будут отображены торговые точки Москвы. Кроме отображения на карте, точки будут выведены в виде таблицы. По мере скроллинга, в таблицу будут подгружаться данные еще не отображенных в ней точек. В таблице можно будет редактировать данные о точках и переходить из нее к отображению карты с центрированием конкретных точек.

В третьей главе будут рассмотрены поисковые системы. Их задачей является осуществление полнотекстового поиска среди веб-страниц (далее просто страницы). Необходимость в них возникла сравнительно недавно, с появлением и очень стремительным разви-

тием интернета в 90-х годах XX века. Большинство поисковых систем состоят из следующих подсистем: кроулер — сбор страниц из интернета с целью дальнейшей работы с ними и сохранения связей, которые состоят из ссылок между ними; индексер — индексация страниц с целью получения информации о том, какие слова содержатся на каких страницах; поисковый интерфейс — веб-интерфейс поисковой системы, позволяющий вводить поисковые запросы и получать в ответ подходящие для них страницы (поисковую выдачу). В дальнейшем мы рассмотрим каждую подсистему подробнее и реализуем их. Далее будет рассмотрено применение алгоритмов ранжирования в генерации поисковой выдачи. После прохождения главы у нас будет реализованная нами поисковая система, которую мы сможем протестировать на реальных данных.

В четвертой главе будут рассмотрены методы кластеризации точек на карте и методы кластеризации новостей. Для кластеризации точек на карте будут реализованы два алгоритма: алгоритм кластеризации методом расстояний и алгоритм k-средних. Для кластеризации новостей будет реализована иерархическая кластеризация снизу-вверх.

В пятой главе рассмотрим задачу классификации изображений и музыкальных файлов. Имеется изначальное множество изображений (музыкальных файлов). Некоторый файл из данного множества был подвержен различным искажениям. Требуется правильно отнести искаженный файл к исходному. Для решения этой задачи будет необходимо изучить работу с музыкальными файлами и изображениями, а также реализовать алгоритмы, оценивающие различия между файлами. В результате будет написано веб-приложение, решающее поставленные задачи.

В шестой главе мы напишем мобильное приложение под платформу iOS, которое позволит мерчендайзерам производить посещения торговых точек, фиксировать выкладки, производить аудит цен, а также фиксировать наличие товара на полке и/или на складе. Для реализации описанного функционала нам понадобится реализовать авторизацию пользователей в приложении, построить пользовательский интерфейс для отображения списка торговых точек и других фрагментов с целью реализации пользовательского взаимодействия с приложением. Далее мы разберем работу с картой, выставление маркеров и обработку действий над ними. Рассмотрим работу с камерой и об-

работку полученного с нее изображения. В результате мы получим работающее мобильное приложение для платформы iOS с описанным выше функционалом.

В седьмой главе мы напишем мобильное приложение под платформу Android, которое повторит функционал, реализованный в предыдущей главе. Будет реализован пользовательский интерфейс, взаимодействие с сетью, работа с БД и камерой с учетом специфики платформы Android.

Основой для данной книги послужил цикл статей [1, 2, 3, 4, 5, 6, 7].

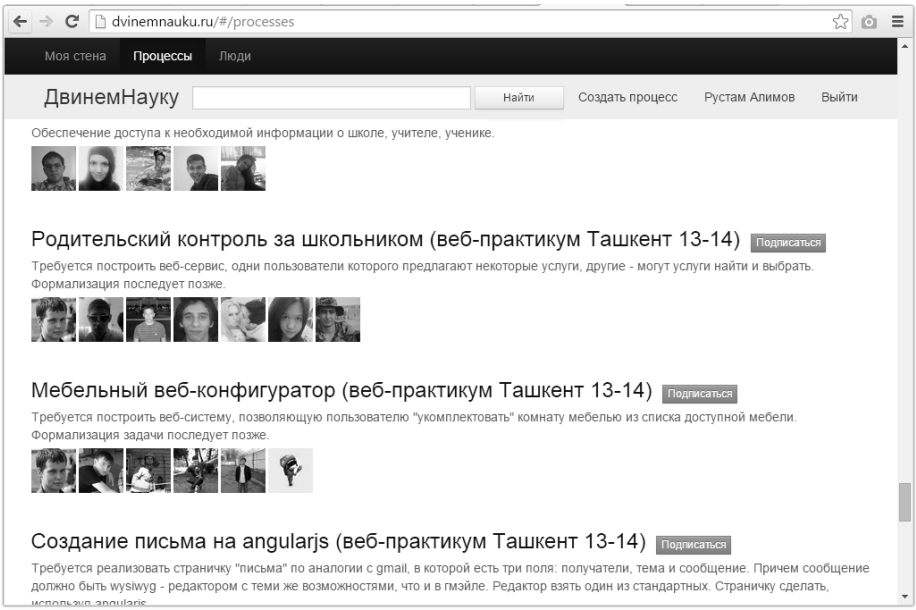

Рис. 1: Список процессов

Система ДвинемНауку

С появлением и развитием интернета знания стали действительно общедоступными. Больше не стоит проблема найти ту или иную статью, книгу, узнать информацию по интересующей тематике. Каждый может познавать, не выходя из дома, лишь бы компьютер или телефон были подключены к интернету. Но появилась другая проблема – информации стало слишком много, в ней стало слишком просто запутаться. Кроме очевидной причины этого, состоящей в том, что опубликовать в интернете информацию может абсолютно каждый, независимо от уровня компетенции, есть и другая, не менее важная. Состоит она в том, что устаревшая информация зачастую не удаляется со страниц интернета, продолжая выдаваться в результатах поиска. В случае с пограммированием эта проблема особенно актуальна, поскольку технологии устаревают крайне быстро. Решением этой проблемы в какой-то степени являются блоги экспертов, следящих и повествующих об актуальных событиях в их тематиках.

В созданной нами системе ДвинемНауку мы обратились к клю-

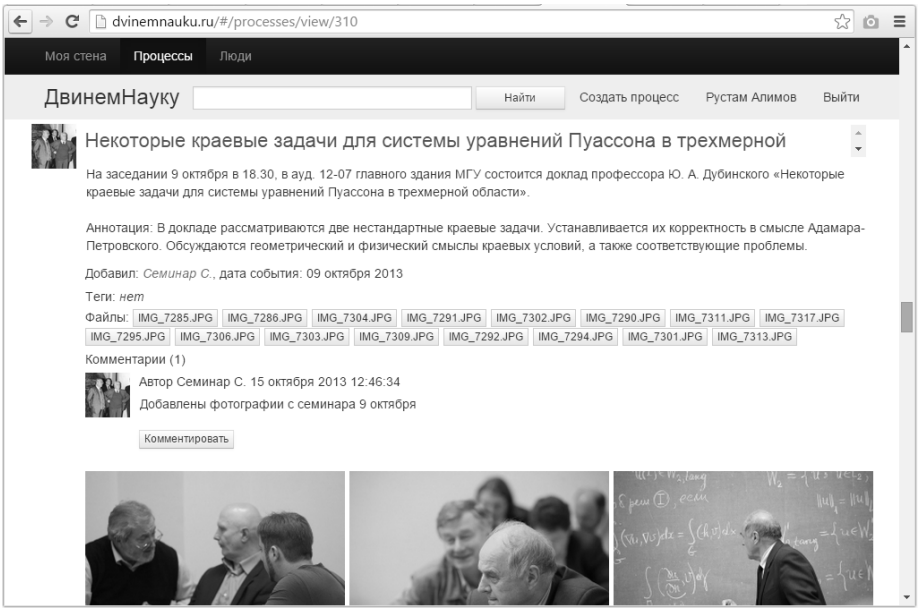

Рис. 2: Процесс

чевому понятию в образовании – научно-образовательному процессу. Учебный план любого университета на самом деле состоит из набора научно-образовательных процессов. Это лекции, семинары, практикумы, и если речь идет об обучении программированию, то материал, разбираемый на них, меняется из года в год – вместе с развитием соответствующих языков и технологий. На самом деле, научную работу любого студента можно считать научно-образовательным процессом. Совместную работу студентов в рамках практикума можно считать научно-образовательным процессом. Таких примеров можно привести много. Так вот суть системы ДвинемНауку состоит в том, чтобы каждому научно-образовательному процессу сопоставить его онлайн-отображение, считайте, блог, в котором можно и нужно оповещать о новых событиях этого процесса.

Если мы рассматриваем научную работу студента, то событием соответствующего процесса можно считать любой полученный им результат. Или новую найденную статью. Или участие в конференции. Или общение с научным руководителем. Это удобно и для студента, и для его научного руководителя. При таком подходе сложно потерять

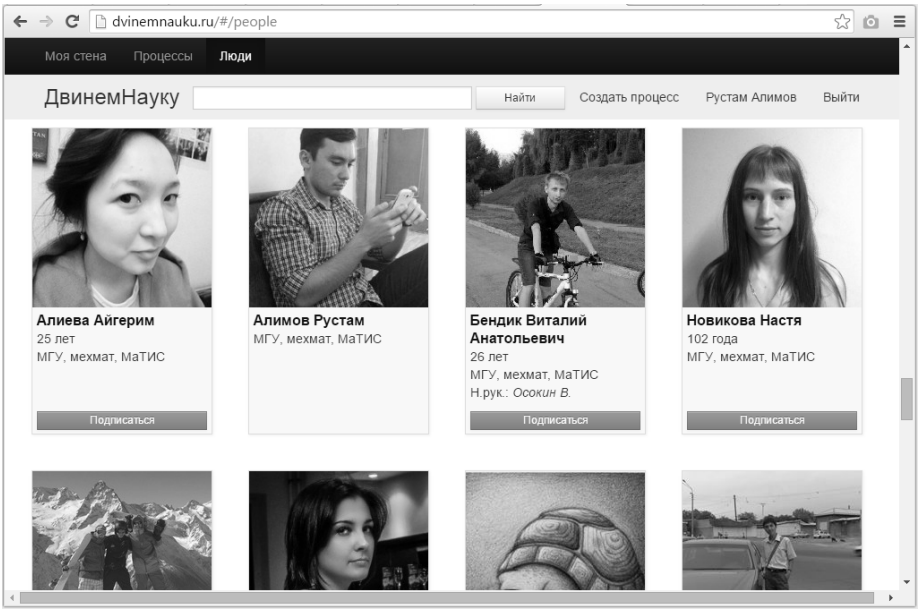

Рис. 3: Список пользователей

статью, которую когда-то находил, но забыл, куда сохранил. При таком подходе легко отследить тупиковые направления исследований и потраченное на них время. При таком подходе можно в любой момент подключить к процессу новых людей, которые легко изучат историю вопроса. В конце концов, при таком подходе становится куда менее важным физическое расположение студента и его научного руководителя.

В случае семинаров и лекций многие преподаватели уже давно сопровождают соответствующие процессы блогами или сайтами, размещая на них информацию об очередных заседаниях. Преимуществом ДвинемНауку является связь между процессами. Так, практикум могут вести несколько преподавателей, ему соответствует один процесс в системе и в этом поцессе оповещается о любых общих вопросах практикума. А вот в рамках практикума каждый студент или группа студентов могут решать свои задачи, сопровождая это отдельным процессом в ДвинемНауку. Преподаватели подписаны на процессы студентов, которых они проверяют, тем самым получая информацию о решенных задачах или вопросах на своей стене. Это позволяет более

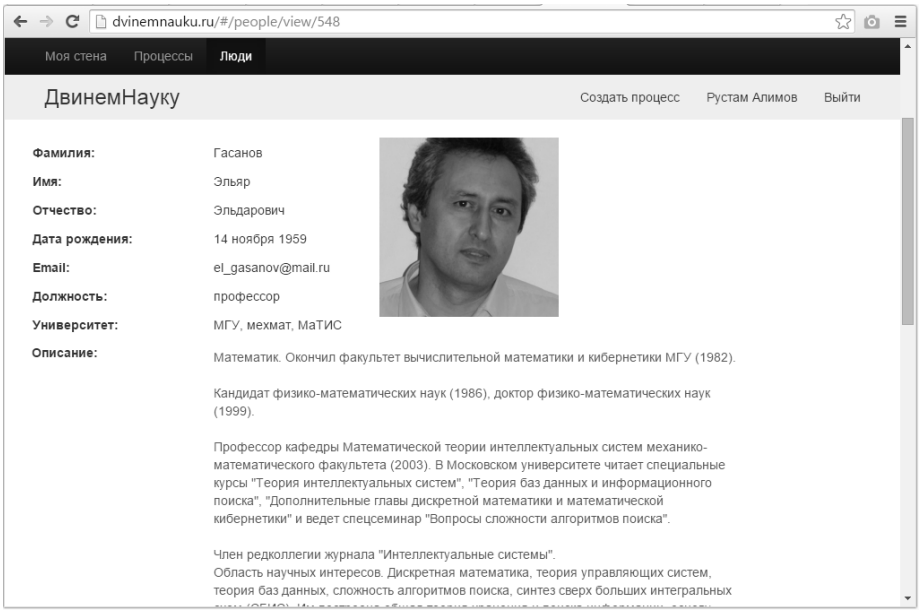

Рис. 4: Пользователь

индивидуально относиться к процессу обучения, находить время на каждого студента. Подробное описание системы ДвинемНауку не является предметом данной книги. О ней можно подробнее прочитать, например, в [1]. Весь разобранный в данной книге материал ведется в системе в виде различных процессов.

На рисунке 1 представлен скриншот процесса семинаров ректора МГУ им. М.В.Ломоносова академика РАН В.А. Садовничего «Спектральная теория дифференциальных операторов» и «Время, хаос и математические проблемы». На рисунке 2 представлен скриншот, отражающий некоторый список процессов студентов в рамках практикума. На рисунке 3 представлен скриншот с некоторыми пользователями системы, а на рисунке 4 – скриншот страницы пользователя системы.

1. Базовый веб-сайт

В данной главе мы рассматриваем реализацию простейшего аналога Википедии. Система должна позволять регистрироваться, авторизоваться в ней, давать возможность авторизованному пользователю создавать страницы, редактировать созданные им страницы, удалять их, а также просматривать страницы других пользователей и список всех страниц в системе.

Для реализации используем язык разметки HTML, каскадные таблицы стилей CSS, язык веб-программирования PHP, систему управления базами данных (СУБД) MySQL и язык запросов SQL, а также веб-сервер Apache. Для написания кода используем редактор Notepad++ (Windows) или Gedit (Linux). Для копирования файлов на сервер и редактирования файлов на сервере используем программу WinSCP (Windows) и Nautilus (Linux). О всех этих языках, программах и технологиях подробно рассказывается далее.

В разделе «Язык разметки HTML» мы рассматриваем реализацию простейших HTML страниц.

В разделе «Каскадные таблицы стилей CSS, верстка шаблона страницы» рассматриваем оформление HTML страниц с помощью каскадных таблиц стилей CSS и описываем процесс верстки шаблона нашего аналога Википедии.

В разделе «Простейший многостраничный сайт» создаем несколько страниц и связываем их ссылками, получаем наш первый сайт, доступный только на нашем компьютере.

В разделе «Веб-сервер Apache» мы показываем, как сделать наш сайт доступным пользователям на других компьютерах, установив и настроив на нем веб-сервер Apache. В частности, мы рассматриваем путь запроса от браузера пользователя до сервера и обратно, показываем, каким образом сервер обрабатывает запрос пользователя.

В разделе «Язык программирования PHP» мы замечаем, что страницы нашего сайта отличаются друг от друга только содержащимися в них данными, а внешний вид у них одинаковый. Исходя из этого мы понимаем, что внешний вид хотелось бы хранить отдельно от данных, чтобы не дублировать код, а брать внешний вид из одного файла на всех страницах нашего сайта. Для этой цели языка разметки HTML уже недостаточно, и возникает необходимость использовать какой-либо язык программирования. Поэтому мы устанавливаем на нашем

сервере язык веб-программирования PHP и переделываем наш сайт из HTML в PHP, путем выделения единого шаблона и хранения данных в отдельных PHP файлах.

В разделе «Система управления базами данных MySQL, язык запросов SQL» мы узнаем, что стандартом является хранение данных не в переменных языков программирования и не в текстовых файлах, а в базах данных. Например, в системе управления базами данных (СУБД) MySQL. Соответственно, мы устанавливаем на сервере СУБД MySQL и переписываем наш сайт так, чтобы данные хранились в ней и только в ней. В частности, мы узнаем, каким образом можно получать данные из СУБД MySQL, используя язык SQL и его оператор SELECT.

В разделе «Операции CRUD (create, read, update, delete) с сущностями системы» мы понимаем, что в стандартном веб-приложении, кроме просмотра страниц сущностей, вообще говоря, выполняются операции добавления, изменения и удаления сущностей, а также просмотра их списка. У набора операций добавления (create), просмотра (read), редактирования (update) и удаления (delete) существует акроним CRUD. В создаваемой нами системе имеются только две сущности — это страницы и пользователи. Легко понять, что должна быть возможность выполнять перечисленные операции с обеими этими сущностями. Для примера, в этом разделе для страниц мы реализуем функционал добавления, просмотра, редактирования, удаления и просмотра списка всех страниц, а для пользователей реализуем только функционал добавления (в случае сущности «Пользователь» принято функционал добавления называть регистрацией) и просмотра. Реализацию функционала редактирования, удаления и просмотра списка всех пользователей оставляем читателю.

В разделе «Авторизация пользователей» мы замечаем, что, в отличие от сущности страницы, для страницы «Пользователь» должен быть доступен функционал авторизации в системе. После того, как пользователь вводит логин и пароль, система должна помнить пользователя при переходе с одной страницы на другую и забывать пользователя при нажатии кнопки «Выйти» или при долгом бездействии пользователя. В данном разделе мы добавляем функционал авторизации и запоминания пользователя в нашей системе.

В разделе «Связь между сущностями системы» мы понимаем, что сущности системы не существуют отдельно, а связаны друг с дру-

гом. Так, при добавлении новой страницы в базе данных в записи этой страницы должен быть прописан пользователь, создавший эту страницу; на странице пользователя разумно выводить список страниц, созданных данным пользователем, а на странице вывода списка страниц выводить пользователей, создавших страницы. В данном разделе мы соответствующим образом обновляем наше веб-приложение. Кроме того, мы корректируем функционал редактирования и удаления страниц таким образом, чтобы только создатель страниц мог их редактировать и удалять.

В конце главы приводим заключение по проделанной работе.

1.1. Язык разметки HTML

Начинаем с создания нашего первого веб-сайта, состоящего из одной страницы. Прежде всего, создаем текстовый документ с расширением html на нашем компьютере, например, page.html. Для редактирования html-документов можно использовать редакторы Notepad++ [8] (Windows) и Gedit [9] (Linux) или любой другой удобный читателю редактор. Открываем созданный файл в редакторе и пишем «Hello world», сохраняем. Теперь открываем наш файл в веб-браузере, например, Chrome, и видим слова «Hello world».

Понятно, что кроме простого текста мы хотим видеть на нашей странице такие элементы, как заголовки, ссылки, списки, таблицы и прочее. В коде страницы эти и другие элементы задаются с помощью так называемых тегов. Так, главный заголовок страницы задается с помощью тега «h1», ссылка — при помощи тега «a», картинка — при помощи тега «img». Далее в книге мы вкратце описываем теги, которые используем, более полное их описание можно найти на сайте htmlbook.ru [10].

Создаем рисунок с названием msu.png и размещаем его в той же папке, где находится файл page.html. Добавляем следующие строчки в файл page.html.

```
<h1>Главный заголовок</h1>
<a href="http://yandex.ru">Текст ссылки</a>
<img src="msu.png" />
```

На рисунке 5 представлен результат открытия страницы в браузере. На данном примере можно увидеть, что существуют теги, состоящие из двух частей – открывающей и закрывающей, внутрь кото-

рых помещается содержимое (теги «h1» и «a» в примере), и теги, состоящие из одной части (тег «img» в примере). Также на примере тегов «img» и «a» можно увидеть, что у тегов могут быть так называемые атрибуты, например «src» у «img» и «href» у «a». Атрибут «src» задает адрес картинки, а «href» задает адрес страницы, на которую будет перенаправлен пользователь при нажатии на ссылку. Также результат вывода можно увидеть по ссылке http://book1.dvinemnauku.ru/page_1.html [20].

Рис. 5: Вывод заголовка «h1», ссылки «a» и картинки «img»

Описанные нами теги называются html-тегами в честь языка разметки HTML, который и является языком написания веб-страниц. Именно поэтому мы дали файлу page.html расширение html. Нужно понимать, что браузер превращает HTML-страницу в то отображение, что мы видим на экране, так как он является интерпретатором языка разметки HTML. Стоит заметить, что HTML-тегов очень много, но большая часть из них устарела и не рекомендуется к использованию. Так, сразу обратим ваше внимание на то, что не стоит использовать

специальные теги, такие как «b», «i», «table» и другие для оформления внешнего вида HTML-страницы. Ранее тег «table» активно использовался для верстки внешнего вида страницы, что на данный момент крайне не рекомендуется. Вместо этого необходимо использовать CSS — каскадные таблицы стилей, речь о которых пойдет в соответствующем разделе.

Теперь переходим к созданию страницы с большим числом различных тегов.

```
<h1>Главный заголовок страницы</h1>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit
    . Integer egestas <img src="smile-1.png" />, magna et
    accumsan dapibus, nisl enim sodales arcu, vel rhoncus
    ipsum urna a justo.</p>

<table>
    <tr>
        <td>Ячейка 1.1</td><td>Ячейка 1.2</td>
    </tr>
    <tr>
        <td>Ячейка 2.1</td><td>Ячейка 2.2</td>
    </tr>
</table>

<h2>Заголовок второго уровня 1</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit
    . Nam ante tellus, fermentum et leo quis, venenatis <
    img src="smile-2.png" /> interdum felis.</p>

<ol>
    <li>Элемент один</li>
    <li>Элемент два</li>
    <li>Элемент три</li>
</ol>

<h3>Заголовок третьего уровня 1</h3>
<p>Nam ante tellus, fermentum et leo quis, venenatis
    interdum felis. <a href="http://yandex.ru">Ссылка на Я
    ндекс</a>.</p>

<h3>Заголовок третьего уровня 2</h3>
<p>Nunc at rhoncus augue. Sed vel efficitur tellus. Morbi
    ac magna eros. Aenean ultrices nisi vitae diam euismod
    , eget interdum leo aliquam.</p>
```

На рисунке 6 приведен результат.

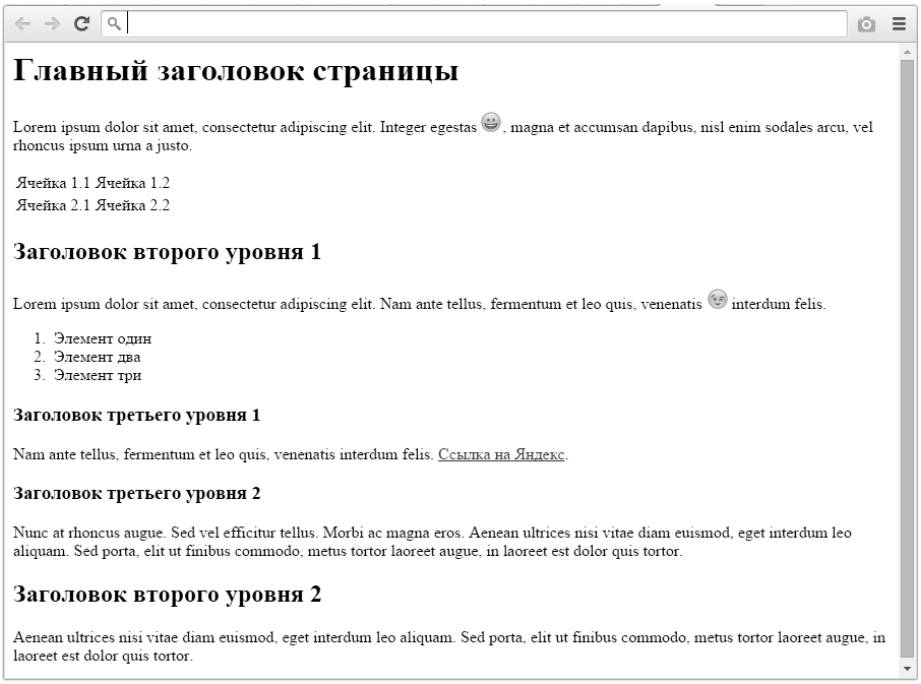

Рис. 6: HTML-страница с большим числом HTML-тегов

Тег «table» позволяет создать таблицу и используется с тегом «tr», который задает строку, и с тегом «td», который задает ячейку. Тег «ol» позволяет создать нумерованный список и используется вместе с тегом «li», который задает элемент списка. Теги «h1»-«h5» используются для формирования заголовков. Тег «p» используется для формирования параграфов. Результат вывода можно увидеть по ссылке http://book1.dvinemnauku.ru/page_2.html [21].

Перечисленные нами теги управляют элементами, которые мы хотим отобразить на экране. Существуют и другие теги. Например, тег «title» позволяет задать заголовок страницы, который выводится в браузерах в качестве подписи ко вкладке страницы. Также существуют теги, задача которых состоит только в том, чтобы задать корректную структуру веб-страницы. Это теги «html», «head» и «body». Так, весь HTML-код должен содержаться в теге «html», упомянутый тег «title» — внутри тега «head», а все остальные упомянутые теги — внутри «body».

Кроме задания заголовка с помощью тега «title», внутри тега «head» очень часто подключают каскадные таблицы стилей CSS, задают кодировку страницы для того, чтобы браузер знал, в какой кодировке ее нужно отображать (для этого используется тег «meta» и его атрибут «charset»), подключают скрипты на языке JavaScript. Обо всем этом речь пойдет далее.

Также стоит выделить теги «div» и «span», которые всегда располагаются внутри тега «body» и о которых речь также пойдет далее.

Из сказанного следует, что для того, чтобы HTML-код созданной нами страницы page.html был более корректным, необходимо добавить в page.html следующий код (здесь многоточие необходимо заменить на написанный нами ранее код).

```
<html>
    <head>
        <title>Заголовок, выводимый в качестве подписи ко
            вкладке браузера</title>
        <meta charset="utf8" />
    </head>
    <body>
        ...
    </body>
</html>
```

Результат вывода можно увидеть по ссылке http://book1.dvinemnauku.ru/page_3.html [22]. Обратите внимание на подпись вкладки браузера.

1.2. Каскадные таблицы стилей CSS, верстка шаблона страницы

На этом мы заканчиваем с созданием базовой HTML-страницы и переходим к ее оформлению. Как было сказано ранее, для оформления страницы следует использовать каскадные таблицы стилей CSS. Например, давайте сделаем, чтобы ссылка из приведенного выше примера отображалась жирным начертанием и была красного цвета.

Для этого заменяем код

```
<a href="http://yandex.ru">Ссылка на Яндекс</a>
```

на

```
<a href="http://yandex.ru" style="color: red; text-
    decoration: bold;">Ссылка на Яндекс</a>
```

Здесь мы применяем к тегу «a» HTML-атрибут «style». Значением этого HTML-атрибута является строка из двух CSS-атрибутов: «color» и «text-decoration», которые принимают значения «red» и «bold» соответственно. Заметьте, если HTML-атрибуты мы разделяем пробелами, а значения атрибутов следуют после знака «=», то CSS-атрибуты разделяются знаком «;» и значения следуют после названий не через «=», а через «:».

Как и в случае с HTML-тегами и атрибутами, мы будем вкратце описывать используемые нами CSS-атрибуты. Более полное описание читатель может найти на сайте htmlbook.ru [11].

На самом деле, не всегда удобно писать CSS-атрибуты внутри HTML-тега «style». Например, если бы мы хотели задать оформление не только одной ссылки, а всех ссылок «a» нашей страницы, то было бы удобно сохранить описание такого оформления в едином месте. Это можно сделать следующим образом внутри тега «head».

```
<html>
    <head>
        <title>Заголовок, выводимый в качестве подписи ко
            вкладке браузера</title>
        <meta charset="utf8" />
        <style>
            a {
                color: red;
                text-decoration: bold;
            }
        </style>
    </head>
```

Альтернативно бывает удобно создать файл с расширением css, например, style.css, вставить в него то же самое, что мы вставили внутрь тега «style», и заменить тег «style» и его внутренности на следующую строку.

```
<link rel="stylesheet" href="style.css" />
```

Данная строка подключает файл style.css к нашему документу. Как уже было сказано, все ссылки документа станут при этом красными и жирными.

Бывает необходимость придать оформление только части элемен-

тов HTML-страницы. Например, мы хотим, чтобы особо важные ссылки были выделены еще курсивом. Тогда соответствующим ссылкам присваиваем неуникальный идентификатор следующим образом.

```
<a class="important">Экземпляр особо важной ссылки</a>
```

Здесь всем важным ссылкам мы приписываем HTML-атрибут «class» со значением «important». Атрибут «class» не имеет никакого отношения к объектно-ориентированному программированию, он просто используется, когда мы хотим некоторым образом выделить/назвать некоторую группу (класс) тегов. Значение «important» – просто произвольное слово. Тогда таким ссылкам можно придать дополнительное оформление путем добавления следующих строк в CSS-файл или в тег «style».

```
.important {
    font-style: italic;
}
```

Как видно, в CSS до названия класса нужно поставить точку.

Если мы хотим выделить уникальный элемент HTML-страницы, то вместо HTML-атрибута «class» целесообразно использовать HTML-атрибут «id».

```
<a id="unique-link">Уникальная ссылка</a>
```

Тогда этой ссылке можно придать дополнительное оформление путем добавления следующих строк в CSS-файл или в тег «style».

```
#unique-link {
    font-style: italic;
}
```

Как видно, в CSS до названия уникального идентификатора нужно поставить знак решетки.

Как мы уже упоминали, существует два важных тега, которые используются внутри тега body — это тег «span» и, в особенности, тег «div». Включение содержимого в эти теги, вообще говоря, ничего особенного не делает, но дает возможность задавать оформление и применять функции языка JavaScript (о нем речь пойдет во второй главе) целиком к этому содержимому. Например, если мы хотим оформить некоторым образом часть слов нашего текста, мы можем заключить эти слова в тег «span», что само по себе никак не поменяет отображение этих слов. Но если мы припишем к тегу «span» CSS-атрибуты,

как мы делали ранее, слова или даже целые предложения приобретут необходимый нам вид.

В отличие от тега «span», если включить некоторое содержимое HTML-страницы в тег «div», оно перенесется на новую строку. HTML-теги, ведущие себя подобно тегу «div», называются блочными, а аналогичные тегу «span» называются строчными. В CSS строчность/блочность элемента задается при помощи CSS-атрибута «display». Так, значение «inline» этого атрибута делает соответствующий элемент строчным, а значение «block» — блочным. Из сказанного выше следует, что у тега «span» (как и у тегов «a», «img») значение этого атрибута по умолчанию выставлено как «inline», а у тега «div» (как и у тегов «h1», «p», «ul») значение этого атрибута выставлено как «block». Далее мы увидим, что бывают случаи, когда целесообразно менять значение данного атрибута, например, когда мы хотим использовать ненумерованный список «ul» для формирования горизонтального меню.

На базе тега «div» удобно рассмотреть некоторые CSS-атрибуты, о которых мы еще не упоминали. Это атрибуты «border», «padding» и «margin». Атрибут «border» задает оформление границы div'a, атрибут «padding» — отступ от границы div'a до содержимого div'a, а атрибут «margin» — отступ от границы div'a до окружения div'a. Далее приводим код, иллюстрирующий данные атрибуты.

```
<html>
    <head>
        <title>Заголовок страницы</title>
        <style>
            body {
                background-color: lightgrey;
                margin: 0;
                padding: 0;
            }
            #div1 {
                background-color: white;
                padding: 20px;
                margin: 10px;
                border: 5px solid black;
            }
        </style>
    </head>
```

```
    <body>
        <div id="div1">
            Nunc at rhoncus augue. Sed vel efficitur
                tellus. Morbi ac magna eros. Aenean
                ultrices nisi vitae diam euismod, eget
                interdum leo aliquam. Sed porta, elit ut
                finibus commodo, metus tortor laoreet
                augue, in laoreet est dolor quis tortor.
        </div>
    </body>
</html>
```

Заметим, что мы обнулили значения атрибутов «padding» и «margin» у тега «body», поскольку по умолчанию в большинстве браузеров они ненулевые. На рисунке 7 показана страница с описанным выше исходным кодом. Мы видим, что отступ от черной границы div'а до границ страницы составляет 10 пикселей, отступ от черной границы div'а до текста внутри div'а составляет 20 пикселей, а ширина самой границы — 5 пикселей, она отображена непрерывной линией («solid») черного цвета («black»). Результат вывода можно увидеть по ссылке http://book1.dvinemnauku.ru/page_4.html [23].

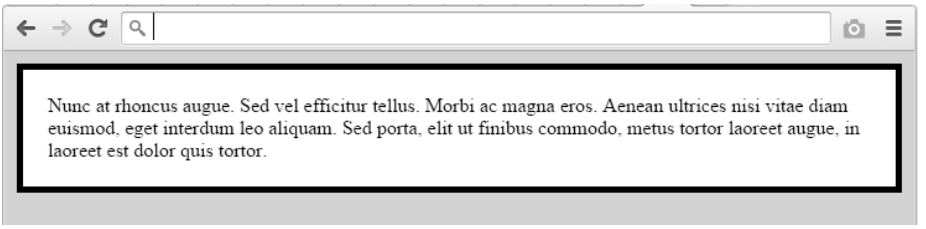

Рис. 7: CSS-атрибуты «padding», «margin» и «border»

Как и в случае HTML-тегов и HTML-атрибутов, существует множество различных CSS-атрибутов, описание которых можно найти в любом справочнике по CSS, например htmlbook.ru [11]. Из важных часто используемых CSS-атрибутов, о которых мы еще не упоминали, стоит выделить атрибуты «float» и «position», которые постоянно используются для размещения различных блоков веб-страницы относительно друг друга, например, для формирования колонок, хедера и футера страницы (верхней и нижней частей) и так далее. Об их использовании будет рассказано далее.

Приступаем к верстке макета нашей будущей Википедии. На изоб-

ражении 8 приведено расположение элементов на странице.

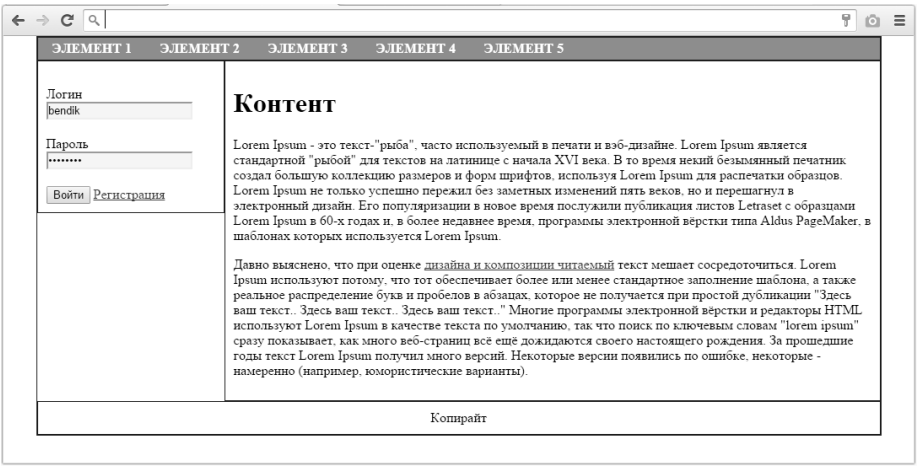

Рис. 8: Расположение элементов на странице

Как видно, нам нужно расположить меню сайта сверху, затем идут два блока, расположенные рядом друг с другом, которые содержат форму авторизации и основной контент. В самом низу располагаем немного информации о сайте, его правообладателях и прочее.

Для начала очистим наши файлы page.html и style.css. В начале файла page.html поместим начало HTML-страницы и тег «header», в котором подключим наш файл со стилями style.css.

```
<html>
    <head>
        <title>Заголовок страницы</title>
        <meta charset="utf-8" />
        <link rel="stylesheet" href="style.css" />
    </head>
```

Затем, в файле style.css уберем у тега «body» все отступы и добавим границу всем тегам «div», чтобы было удобнее верстать.

```
body {
    margin: 0;
    padding: 0;
}
div {
    border: 1px solid blue;
}
```

Итак, все подготовительные работы сделаны. Приступаем к непосредственной верстке страницы. Все содержимое страницы располагаем в div#container (элемент «div» с идентификатором «container») с точно заданной шириной в 1000px и с расположением по центру страницы.

```
<body>
    <div id="container">
```

Задаем стили div#container.

```
#container {
    width: 1000px;
    margin: auto;
}
```

Здесь CSS-атрибуту «margin» выставляем значение «auto», поскольку в таком случае браузер автоматически высчитывает равные отступы слева и справа от нашего div#container и располагает его по центру страницы.

Далее идет блок div#header включающий в себя меню ul#menu. Меню принято реализовывать в виде списка «ul» (ненумерованный список), содержащего ссылки. Фон меню делаем голубого цвета, а текст меню — белого. Каждый элемент меню выглядит как кнопка шириной в 120 пикселей и при наведении меняет цвет на темно-синий. Значения атрибутов «href» у ссылок «a» в нашем меню читатель может заменить на любые ссылки, которые захочет. Здесь мы оставляем их пустыми.

```
<div id="header">
    <ul id="menu">
        <li><a href="">Элемент 1</a></li>
        <li><a href="">Элемент 2</a></li>
        <li><a href="">Элемент 3</a></li>
        <li><a href="">Элемент 4</a></li>
        <li><a href="">Элемент 5</a></li>
    </ul>
</div>
```

Определяем стили элемента div#header.

```
#header {
    display: block;
    background-color: #00BFFF;
}
```

Здесь значение цвета фона div#header указано в шестнадцатеричном формате, подробнее читайте на сайте htmlbook.ru [12].

Так как меню у нас горизонтальное, то, соответственно, нам необходимо расположить элементы меню «li» в горизонтальном порядке. Для этого используем свойство «float», которое означает, с какой стороны элемент обтекается другими элементами, текстом и так далее. Если у элемента «li» установить свойство «float» в значение «left», то последующий элемент «li» начинает обтекать его справа и, следовательно, располагается справа от него. Это выполняется для каждого элемента «li», в результате чего получаем горизонтальное меню.

```
#menu li {
    float: left;
}
```

Кроме того, у элемента «ul» убираем маркировку, убираем отступы и выставляем свойство «overflow» в значение «hidden».

```
#menu {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
}
```

Далее настраиваем ссылки меню, устанавливаем их размеры, отступы и задаем изменение цвета при наведении на них.

```
#menu li a {
    display: block;
    width: 120px;
    font-weight: bold;
    color: #FFFFFF;
    text-align: center;
    padding: 4px;
    text-decoration: none;
    text-transform: uppercase;
}
#menu li a:hover {
    background-color: #339ABD;
}
```

Здесь запись «a:hover» означает, что стили применяются только тогда, когда на элемент наведен курсор.

На изображении 9 приведен результат. Также результат вывода страницы можно увидеть по ссылке

http://book1.dvinemnauku.ru/page_5.html [24].

Рис. 9: Главное меню

Далее необходимо расположить два блока «div» на одной линии. В левом будет форма авторизации, а в правом будет содержимое страницы.

```
<div id="left">
    <form>
        <label>Логин</label>
        <input type="text" name="login" />
        <label>Пароль</label>
        <input type="password" name="password" />
        <label></label>
        <button type="submit">Enter</button> <a href="/
            signup.php">Войти</a>
    </form>
</div>
<div id="right">
    <h1>Заголовок страницы</h1>
    <p>Содержимое страницы.</p>
</div>
```

Использованные здесь теги «form», «label», «input», «button» будут объяснены в разделе «Операции CRUD (create, read, update, delete) с сущностями системы».

Чтобы расположить два элемента «div» на одной линии, снова устанавливаем CSS-атрибут «float» в значение «left». Для левого div#left устанавливаем отступ в 10 пикселей и ширину в 200 пикселей.

```
#left {
    float: left;
    width: 200px;
    padding: 10px;
}
```

Для правого элемента «div» устанавливаем отступ в 10 пикселей и ширину в 750 пикселей.

```
#right {
    float: left;
    width: 750px;
    padding: 10px;
}
```

Далее располагаем нижнюю часть нашей страницы div#footer с копирайтом.

```
            <div id="footer">Copyright</div>
        </div>
    </body>
</html>
```

Так как в предыдущих элементах мы указывали обтекание, в данном элементе необходимо его отменить, чтобы он расположился под ними. Для этого есть CSS-атрибут «clear». Устанавливаем его в значение «both».

```
#footer {
    clear: both;
    text-align: center;
    padding: 10px;
}
```

Результат приведен на изображении 8. Результат вывода можно увидеть по ссылке http://book1.dvinemnauku.ru/page_6.html [25].

1.3. Простейший многостраничный сайт

Итак, на данный момент мы создали одну страницу нашей будущей Википедии. Чтобы создать простейший многостраничный сайт, нам достаточно продублировать эту страницу и поменять в созданных файлах текст и рисунки. Так мы можем получить сайт из любого количества страниц. Уже сейчас читателю должно быть понятно, что это не самый разумный подход, поскольку, копируя файлы, мы копируем не только данные, но и структуру HTML-документа. А ведь структура не меняется от страницы к странице, и было бы целесообразно хранить ее в едином месте для всех страниц сайта. В последующих разделах мы узнаем, как решить этот вопрос, а пока рассмотрим, как сделать созданный нами сайт доступным пользователям на других компьютерах.

1.4. Веб-сервер Apache

Понятно, что веб-сайты создаются для того, чтобы на них могли заходить люди из разных точек мира. И чтобы они могли просмотреть созданный нами сайт, необходимо провести некоторые дополнительные действия. А именно, нужно разместить наш веб-сайт на ПК, доступном из любой точки мира, и установить на этом ПК ПО, которое возвращало бы страницы нашего сайта при обращении к ним при помощи некоторых URL.

В упрощенном виде после ввода в адресную строку браузера имени некоторого сайта, например http://dvinemnauku.ru, браузер пользователя выполняет следующие действия. Первым делом он смотрит на слово, которое идет до символов «://». Это слово определяет протокол, по которому пользователь запрашивает данные. В нашем случае это протокол «http». Если бы пользователь ввел «https», это был бы «https», если пользователь ввел бы «ftp», то это был бы «ftp». В зависимости от протокола браузер пользователя обращается на сервер к тому или иному порту. Так, в случае «http» браузер пользователя по умолчанию обращается к 80 порту на сервере, в случае «https» обращается к 443 порту, а в случае «ftp» — к 21 порту.

Определившись с протоколом, браузер должен понять, к какому именно серверу в интернете ему обращаться. Для этого он смотрит вторую часть URL. В нашем случае это «dvinemnauku.ru». Эта часть также называется доменным именем. Браузер должен каким-то образом преобразовать это слово в IP-адрес, по которому уже можно найти соответствующий сервер в интернете. Для перевода доменных имен в IP-адреса в интернете существуют специальные сервера, называемые DNS-серверами.

После того, как браузер выяснил протокол, по которому необходимо обратиться к серверу, и его адрес, он обращается к этому серверу по этому адресу, используя данный протокол. Допустим, наш ПК и есть тот сервер, на который пришел запрос. Для того, чтобы наш сервер понял запрос от браузера пользователя и переслал в ответ сформированную HTML-станицу, на сервере должна быть программа, которая отвечает за это. Такие программы принято называть веб-серверами. Самым известным веб-сервером является Apache [13], обеспечивающий функционирование большинства сайтов в интернете. Если на нашем сервере (не надо путать сервер-ПК и веб-сервер-

программу) установлен и корректно настроен Apache, то он примет запрос пользователя, прочитает доменное имя из запроса пользователя, найдет в своих настройках, какая папка на сервере содержит файлы сайта, соответствующего данному доменному имени, и вернет пользователю файл из этой папки, прописанный по умолчанию в настройках Apache. По умолчанию это чаще всего файл с названием index.html.

Приведем пример. Предположим, что доменному имени dvinemnauku.ru в настройках Apache прописана папка /project/dvinemnauku.ru/public. Тогда, по умолчанию, при наборе пользователем адреса http://dvinemnauku.ru или http://dvinemnauku.ru/index.html сервер вернет файл /project/dvinemnauku.ru/public/index.html. Если же мы разместим на сервере в папке /project/dvinemnauku.ru/public файл myfile.html, то пользователь может открыть данную страницу, набрав в браузере http://dvinemnauku.ru/myfile.html. Вложенные папки создаются и открываются очевидным образом.

Важно понимать, что dvinemnauku.ru и test.dvinemnauku.ru — это совершенно разные домены. В настройках Apache первому домену можно указать одну папку, а второму — совершенно другую. Домены вида dvinemnauku.ru называются доменными именами второго уровня, test.dvinemnauku.ru — третьего уровня и так далее.

Из сказанного следует, что для разработки веб-сайтов на локальном компьютере читателю необходимо установить на него вебсервер Apache. В интернете легко найти инструкцию по установке вебсервера Apache для Вашей операционной системы. После первичной установки Apache на Вашем ПК создается папка, в которой располагается HTML-сайт, доступный по умолчанию. Скопируйте в эту папку созданные нами ранее файлы page.html и style.css. Тогда в браузере Вы можете открыть Ваш сайт, набрав http://localhost/page.html, либо http://127.0.0.1/page.html, либо, если 192.168.1.100 – Ваш IP-адрес в локальной сети, то и набрав http://192.168.1.100/page.html на любом компьютере в локальной сети, включая Ваш компьютер. IP-адрес 127.0.0.1 — это всегда адрес локальной машины, а localhost — его буквенный аналог, адреса http://localhost/ и http://127.0.0.1/ всегда доступны только на локальной машине. Если же у Вашего компьютера настроен внешний IP-адрес, например, 5.178.86.91, то Ваш сайт сразу становится доступным любому пользователю интернета по ад-

ресу http://5.178.86.91/page.html. В этом случае Вы можете купить у любого регистратора (например, R01 [14]) доменное имя, например, dvinemnauku.ru, и настроить его на Ваш IP-адрес, после чего Ваш сайт станет доступным по адресу http://dvinemnauku.ru/page.html.

1.5. Язык программирования PHP

Очевидно, что описанный ранее метод создания сайтов путем дублирования страниц можно применять только при очень малом количестве страниц. Но наша Википедия будет содержать много контента, и применять данный метод в нашем случае крайне неудобно. Например, мы решили немного поменять дизайн. Тогда нам придется менять все наши файлы. Данный недостаток можно было бы решить, если мы могли бы заменить конкретные данные на переменные некоторого языка, и в зависимости от загружаемой пользователем страницы определять эти переменные тем или иным образом. Понятно, что для решения этой задачи недостаточно языка разметки HTML. Именно в связи с этим был придуман язык программирования PHP [15] (изначально Personal Home Page).

Вообще говоря, чтобы из HTML-страницы сделать PHP-файл, достаточно сменить его расширение с html на php, предварительно настроив веб-сервер Apache, чтобы он воспринимал PHP-файлы. Так, обратившись к серверу при помощи URL http://dvinemnauku.ru/mypage.php, веб-сервер apache найдет файл mypage.php в папке сайта на сервере и вместо того, чтобы отдать его сразу браузеру (как он делал в случае HTML-страниц), он отправит данный файл интерпретатору языка PHP, установленному на сервере. Интерпретатор выполнит весь PHP-код на странице, в частности, определит все переменные в описанном выше шаблоне и на выходе сгенерирует необходимый текстовый файл. Именно этот файл (в нашем случае с HTML-кодом) веб-сервер и вернет браузеру пользователя.

Мы не будем останавливаться на описании базовых функций языка PHP и на его синтаксисе, поскольку это должно быть понятно из приводимых далее примеров всем, кто знаком с каким либо C-подобным языком программирования. Мы по возможности будем объяснять используемые функции по ходу повествования. Описание любой функции удобно смотреть на официальном сайте PHP [16].

Перед тем, как мы перейдем к написанию PHP-кода, Вам необходимо установить на Ваш ПК интерпретатор языка PHP и настроить Apache, чтобы при запросе PHP-файлов он использовал данный PHP-интерпретатор для их интерпретации. Как и в случае Apache, читатель легко найдет в интернете инструкции по установке и настройке PHP. Далее мы считаем, что у читателя установлены и настроены Apache и PHP.

Создаем в папке нашего сайта файлы wiki_page_1.php и wiki_page_2.php и помещаем в них по две переменные — $title для хранения заголовка и $content для хранения содержимого страницы.

```
<?php
    $title = "Заголовок страницы";
    $content = "Содержимое страницы.";
?>
```

Заметим, что любой PHP-код вообще говоря принято начинать на «<?php» и заканчивать на «?>». Любая переменная языка PHP начинается с символа «$».

Создаем файл wiki_template.php, в который копируем созданный нами ранее шаблон, но вместо заголовка и содержимого страницы вводим переменные $title и $content.

```
<html>
    <head>
        <title><?=$title?></title>
        <meta charset="utf-8" />
        <link rel="stylesheet" href="style.css" />
    </head>
    <body>
        <div id="container">
            <div id="header">
                <ul id="menu">
                    <li><a href="">Элемент 1</a></li>
                    <li><a href="">Элемент 2</a></li>
                    <li><a href="">Элемент 3</a></li>
                    <li><a href="">Элемент 4</a></li>
                    <li><a href="">Элемент 5</a></li>
                </ul>
            </div>
            <div id="left">
                <form>
                    <label>Логин</label>
                    <input type="text" name="login" />
```

```
                    <label>Пароль</label>
                    <input type="password" name="password"
                        />
                    <label></label>
                    <button type="submit">Войти</button> <
                        a href="">Регистрация</a>
                </form>
            </div>
            <div id="right">
                <h1><?=$title?></h1>
                <?=$content;?>
            </div>
            <div id="footer">Копирайт</div>
        </div>
    </body>
</html>
```

Заметим, что мы здесь использовали запись вида <?=$title?>, что является короткой записью для <?php echo $title;?>, а «echo» – это команда языка PHP, выводящая строку.

Далее поместим в наши файлы wiki_page_1.php и wiki_page_2.php включение файла wiki_template.php.

```
require("wiki_template.php");
```

Мы продолжим знакомство с языком PHP в последующих разделах.

1.6. Система управления базами данных MySQL, язык запросов SQL

Введением единого шаблона и хранением данных в различных PHP файлах мы не решили главную проблему — наличие множества файлов, которые в целом имеют единую структуру, но различные данные. Данную проблему мы можем решить хранением данных в СУБД MySQL.

Для простоты пока можно понимать базу данных (БД) как набор таблиц, какими Вы привыкли их видеть в Excel. Система управления базами данных (СУБД) — это программа, устанавливаемая на сервере, которая позволяет хранить базы данных и работать с таблицами этих БД и данными в них при помощи языка запросов SQL. Так, например, в нашей Википедии мы планируем хранить данные о

страницах (например, названия страниц и их содержимое) и информацию о пользователях (например, ФИО, логины, пароли). Понятно, что в нашем случае нам понадобится одна БД в СУБД MySQL с двумя таблицами — таблицей страниц и таблицей пользователей. Более подробно о СУБД MySQL можно посмотреть в Википедии [17].

Для дальнейшей работы нам требуется установить СУБД MySQL на сервере. Как и в случае с Apache и PHP, читатель легко найдет информацию об установке СУБД MySQL, ее настройке и настройке PHP для работы с ней в интернете.

В соответствии со сказанным выше, создаем в нашей СУБД MySQL базу данных wiki. Для того, чтобы это сделать нам нужна некоторая программа, предоставляющая удобный интерфейс для работы с СУБД MySQL. Одной из таких удобных программ является программа phpMyAdmin [18]. Читатель с легкостью найдет инструкцию по установке программы PhpMyAdmin в интернете. Используя phpMyAdmin, создаем в БД wiki таблицу wiki_pages со следующими полями: id — уникальный идентификатор страницы, title — заголовок страницы, content — содержимое страницы.

Теперь вместо хранения данных во множестве файлов мы создадим единый PHP-файл, которому будет передаваться уникальный идентификатор страницы. По данному идентификатору мы будем извлекать данные из БД и подключать файл шаблона, как и прежде.

Напомним, что у нас на данный момент имеются файлы wiki_page_1.php, wiki_page_2.php и wiki_template.php. Для открытия этих файлов в браузере мы используем URL http://dvinemnauku.ru/wiki_page_1.php и http://dvinemnauku.ru/wiki_page_2.php. Вместо этого мы хотим иметь один файл wiki_page.php, которому будем передавать идентификатор страницы, которую требуется отобразить. Тогда URL будут выглядеть следующим образом: http://dvinemnauku.ru/wiki_page.php?id=1 и http://dvinemnauku.ru/wiki_page.php?id=2. Передача переменных таким образом называется передачей переменных методом GET. В коде страницы wiki_page.php мы всегда можем получить значение переданной ей методом GET переменной путем чтения значения массива $_GET. Так, в нашем случае в коде страницы wiki_page.php нам доступна переменная $_GET['id'], значение которой для первого URL равно 1, а для второго — 2. В общем случае, передача переменных методом GET имеет следующий вид:

http://dvinemnauku.ru?var1=val1&var2=val2&var3=val3...

Переходим к созданию файла wiki_page.php и его описанию. В начале файла подключаемся к СУБД и выбираем необходимую нам БД.

```
<?php
$connect = mysqli_connect("127.0.0.1", "username", "
    userpassword");
imysqli_select_db($connect, "wiki");
mysqli_set_charset($connect, "utf8");
```

Используемая функция mysql_connect получает на вход адрес СУБД, в качестве которого указан IP-адрес 127.0.0.1, так как СУБД установлена на том же сервере, что и PHP-интерпретатор, далее передается имя пользователя БД и его пароль. Следующей функцией мы подключаемся к конкретной базе данных с названием wiki. Для верного отображения данных устанавливаем кодировку в значение utf8, .

Далее, используя PHP-функцию isset, проверяем, был ли передан параметр id методом GET в наш PHP-файл. Если он не был передан, выводим сообщение об ошибке и прекращаем работу PHP с помощью команды exit.

```
if (!isset($_GET["id"])) {
    echo "Укажите id страницы.";
    exit;
}
```

Далее пробуем получить данные о странице из БД с помощью передачи переменной с подключением к СУБД и SQL запроса в функцию mysqli_query.

```
$result = mysqli_query($connect, "SELECT * FROM wiki_pages
    WHERE id=".$_GET["id"]);
```

В переменную $result запишется результат обращения к БД. Если он равен false или число возвращенных записей равно 0, выводим сообщение о том, что страница не найдена и прекращаем работу.

```
if (!$result || mysqli_num_rows($result) == 0) {
    echo "В БД не существует страницы с таким параметром
        id.";
    exit;
}
```

С помощью вызова функции mysqli_fetch_assoc получаем acco-

циативный массив, содержащий данные о странице, и определяем переменные $title и $content, после чего можно подключить файл wiki_template.php, содержащий шаблон вывода.

```
$page = mysqli_fetch_assoc($result);
$title = $page["title"];
$content = $page["content"];
require("wiki_template.php");
```

Если читателю из контекста не очевидно значение каких-либо функцию, еще раз советуем посмотреть их определение на сайте php.net [16].

1.7. Операции CRUD (create, read, update, delete) с сущностями системы

На данный момент мы реализовали хранение данных страниц в СУБД MySQL и вывод страниц через единый файл с единым шаблоном. Теперь возникает вопрос о добавлении, редактировании и удалении страниц нашей Википедии. Кроме того, полезной будет страница со списком всех страниц нашей википедии. Рассмотрим сначала вывод всех страниц Википедии.

Для вывода всех страниц википедии воспользуемся ранее созданным шаблоном. Создаем файл all_pages.php со следующим содержимым.

Как и ранее, подключаемся к СУБД и выбираем необходимую нам БД.

```
<?php
$connect = mysqli_connect("127.0.0.1", "username", "
    userpassword");
imysqli_select_db($connect, "wiki");
mysqli_set_charset($connect, "utf8");
```

Устанавливаем заголовок страницы и инициализируем переменную $content.

```
$title = "Все страницы";
$content = "";
```

С помощью запроса к БД извлекаем все страницы и проверяем результат на наличие записей. Если записей нет, то выводим сообщение о том, что ни одной страницы в БД нет.

```
$result = mysqli_query($connect, "SELECT * FROM wiki_pages
     WHERE 1");
if (!$result || mysqli_num_rows($result) == 0) {
    $content = "В системе нет страниц.";
```

Иначе, если список страниц не пустой, то в цикле «while» дописываем к переменной $content очередной элемент «li» списка. И, как обычно, подключаем файл с шаблоном.

```
} else {
    $content = "<ul>";
    while ($page = mysqli_fetch_assoc($result)) {
        $content .= "<li><a href=\"/wiki_page.php?id=".
            $page["id"]."\">".$page["title"]."</a></li>";
    }
    $content .= "</ul>";
}
require("wiki_template.php");
```

Как можно было заметить, код подключения к БД повторяется два раза: один раз в файле wiki_page.php и второй раз в файле all_pages.php. Лучше, конечно, подключаться к БД из файла, который в дальнейшем будет включаться в другие PHP-файлы, в которых необходимо что-либо получать из БД. Для этого создадим файл connect_db.php и запишем в него подключение к БД.

```
<?php
$connect = mysqli_connect("127.0.0.1", "username", "
    userpassword");
imysqli_select_db($connect, "wiki");
mysqli_set_charset($connect, "utf8");
```

Удалим данный код из файлов wiki_page.php и all_pages.php и подключим в них наш новый файл connect_db.php.

```
require("connect_db.php");
```

На данный момент наше главное меню представляет из себя список ничем не полезных ссылок, которые были созданы при верстке шаблона. Но теперь у нас есть экран со списком всех страниц нашей википедии и необходимо разместить ссылку на него в нашем меню.

```
<ul id="menu">
    <li><a href="all_pages.php">Страницы</a></li>
</ul>
```

Ранее возникал вопрос о создании, редактировании и удале-

нии страниц нашей википедии. Существует общепринятый термин-аббревиатура CRUD от английских слов create, read, update, delete, что означает создание, чтение, обновление и удаление соответственно. Как мы упоминали ранее, в системах данный функционал реализуется в полном или в частичном виде для каждой из сущностей. На данный момент в нашей Википедии реализовано только чтение для сущности-страницы. Остальные пункты необходимо реализовать.

Понятно, что в базовом виде страница для создания и редактирования сущностей должна состоять из полей для ввода данных пользователем и из кнопки для сохранения этих данных, примерно как показано на рисунке 10.

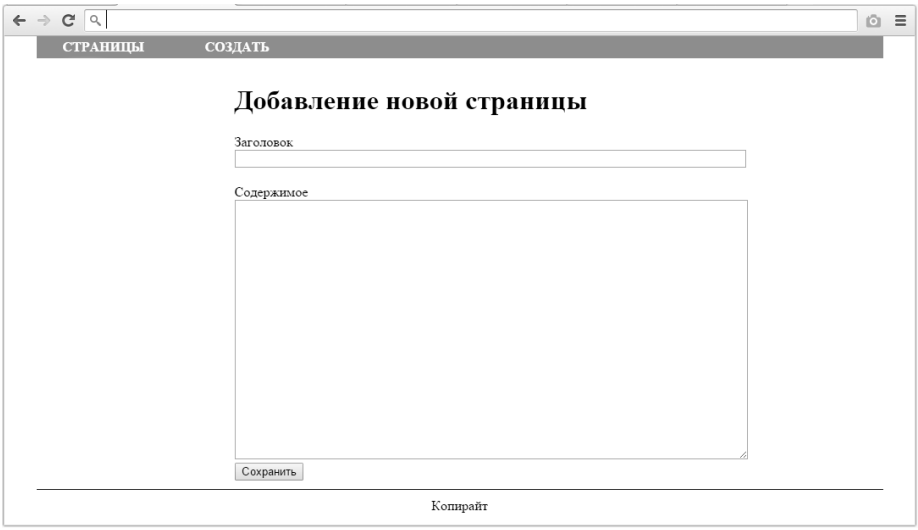

Рис. 10: Форма создания новых и редактирования существующих страниц

Для реализации подобных страниц в HTML используется понятие формы. Сразу приведем конечный код формы для создания новых и редактирования существующих страниц.

```
<h1>Добавление новой страницы</h1>
<form method="POST" action="create_update.php">
    <div>
        <label>Заголовок</label>
```

```
        <input type="text" name="title" value="" id="input
            -title">
    </div>
    <div>
        <label>Содержимое</label>
        <textarea name="content" id="input-content"></
            textarea>
    </div>
    <div>
        <button type="submit">Сохранить</button>
    </div>
</form>
```

Здесь тег «input» служит для создания поля для ввода. Если атрибуту «type» выставлено значение «text» или вообще не выставлено, то браузер отображает это как поле для ввода текста в одну строчку. Если атрибуту «type» выставлено значение «password», то символы вводимого текста отображаются звездочками. Если атрибуту «type» выставлено значение «checkbox», то вместо поля ввода текста будет отображено поле для выставления галочки, а если значение «radiobutton», то вместо поля ввода текста будет отображен кружок выбора одного из вариантов. Подробнее о теге «input» и значениях атрибута «type» можно прочитать на сайте htmlbook.ru [19].

Тег «textarea» служит для ввода многострочного текста, а тег «button» служит для создания кнопки.

Еще один важный тег из приведенного выше кода — это тег «form». Он используется для того, чтобы браузер понимал, какие именно введенные пользователем данные необходимо отправлять на сервер. Так, на одной веб-странице может быть одна группа полей для ввода данных для авторизации пользователя, а вторая — для создания новой страницы, как в нашем примере. Тогда первую группу полей необходимо заключить в первый тег «form», а вторую — во второй.

Важными атрибутами тега «form» являются атрибуты «method» и «action». Атрибут «action» задает адрес страницы, которая будет обрабатывать данные после нажатия на кнопку «Сохранить». По умолчанию атрибут «method» выставлен в значение GET. В примере выше мы выставили его в значение POST. Атрибут «method» определяет, каким образом введенные данные формы будут переданы на сервер для обработки — методом GET или методом POST. В слу-

чае выбора метода GET все введенные данные запишутся в URL, т.е. при нажатии на кнопку «Сохранить» произойдет переход на страницу http://dvinemnauku.ru/create_update.php?title=titl&content=cont. Здесь мы считаем, что пользователь ввел в поле Заголовка текст «titl», а в поле Содержимого текст «cont». В случае выбора метода POST также при нажатии на кнопку «Сохранить» браузер перейдет на страницу http://dvinemnauku.ru/create_update.php, но уже без перечисления переменных после знака вопроса. Переданные переменные все же будут доступны в create_update.php, но для их получения придется воспользоваться не массивом $_GET, как ранее, а массивом $_POST, как будет показано далее.

Важно также обратить внимание на атрибут «type» кнопки «button». Ему выставлено значение «submin». Это сделано для того, чтобы при нажатии на кнопку браузер перешел на страницу, указанную в атрибуте «action» тега «form» и передал ей все введенные пользователем данные.

Чаще всего при отправке данных заполненной формы Вы будете использовать метод POST — он позволяет отправлять данные большого размера, к тому же скрывает данные, которые не должны отображаться в URL (такие, как пароль). Тем не менее, иногда при отправке формы используется и метод GET. Например, именно он используется, когда Вы вводите запрос в поисковой системе. Связано это с тем, что при использовании метода GET есть возможность скопировать URL результата запроса и использовать его для статистики или для отправки другому пользователю системы.

Вернемся к нашей Википедии и к написанию функционала создания новой страницы и редактирования существующей. Для этого необходимо создать страницу с формой для ввода заголовка и содержимого страницы. Так как формы при создании и редактировании идентичны, реализуем данные две функции в одном PHP-файле create_update.php. Если данному скрипту передать идентификатор страницы, то он будет работать в режиме редактирования страницы. Если не передать идентификатор какой-либо страницы, файл будет работать в режиме создания новой страницы. Рассмотрим подробнее скрипт create_update.php. В самом начале подключаем файл с подключением к БД.

```
require("connect_db.php");
```

Далее проверяем, были ли присланы данные с формы редактирования.

```
if (!empty($_POST)) {
```

Если данные есть, создаем переменную $id и приравниваем ее значению null. Далее эта переменная понадобится нам при переходе на новую созданную или отредактированную существующую страницу википедии.

```
$id = null;
```

Проверяем, не был ли передан идентификатор какой-либо страницы методом GET.

```
if (!isset($_GET["id"])) {
```

Если идентификатора страницы нет, то необходимо создать новую страницу в БД. Для этого выполняется запрос INSERT.

```
mysqli_query($connect, "INSERT INTO wiki_pages (title,
    content) VALUES (\"".mysqli_real_escape_string(
    $connect, $_POST['title'])."\", \"".
    mysqli_real_escape_string($connect, $_POST['content'])
    ."\")");
```

Далее с помощью функции mysqli_insert_id получаем идентификатор новой страницы и сохраняем его в переменную $id.

```
$id = mysqli_insert_id($connect);
```

Следующим рассмотрим случай, когда был передан идентификатор страницы методом GET. Чтобы обновить страницу, необходимо выполнить запрос UPDATE.

```
} else {
    mysqli_query($connect, "UPDATE wiki_pages SET title
        =\"".mysqli_real_escape_string($connect, $_POST["
        title"])."\", content=\"".
        mysqli_real_escape_string($connect, $_POST["
        content"])."\" WHERE id=".$_GET["id"]);
```

Далее записываем идентификатор страницы в переменную $id.

```
    $id = $_GET["id"];
}
```

После того, как была создана новая страница или была обновлена существующая страница, переводим пользователя на данную страницу.

```
    header("Location: /wiki_page.php?id=".$id);
    exit;
}
```

На данный момент был рассмотрен случай, когда скрипту были переданы данные для сохранения или редактирования. Теперь рассмотрим случай, когда таких данные передано не было и необходимо вывести пустую форму, если не был передан идентификатор страницы и вывести заполненную форму, если был передан идентификатор страницы. Как было сказано ранее, форма для обоих действий идентична, поэтому нет смысла генерировать ее два раза. Для этого мы создадим переменные $titleValue и $contentValue и выведем их в соответствующих полях. Так как на данный момент мы не знаем, какое из двух действий выполняется, приравняем значение переменной $title пустой строке.

```
$title =""';
$titleValue = "";
$contentValue = "";
```

Далее проверяем, был ли передан идентификатор страницы для редактирования.

```
if (isset($_GET["id"])) {
```

Если был передан идентификатор страницы, необходимо получить из БД содержимое страницы и записать его в переменные $titleValue$contentValue. Для этого делаем запрос к БД и проверяем результат запроса.

```
$result = mysqli_query($connect, "SELECT * FROM wiki_pages
     WHERE id=".$_GET['id']);
if (!$result || mysqli_num_rows($result) == 0) {
    echo "Такой страницы не существует.";
    exit;
}
```

Если по запросу получили данные страницы, записываем их в переменные $titleValue и $contentValue. Также запишем в переменную $title информацию о том, что производится редактирование страницы.

```
    $page = mysqli_fetch_assoc($result);
    $titleValue = $page["title"];
    $contentValue = $page["content"];
    $title = "Редактирование страницы";
} else {
```

Рассмотрим случай, когда не был передан идентификатор страницы. В данном случае все проще — нужно просто оставить форму пустой. Так как мы уже установили значения переменных $titleValue$contentValue, нам остается только установить значение заголовка страницы.

```
    $title = "Создание новой страницы";
}
```

Генерируем форму.

```
$content = "
    <form method=\"POST\">
        <div>
            <label>Заголовок</label>
            <input type=\"text\" name=\"title\" value=\"".
                $titleValue."\" id=\"input-title\">
        </div>
        <div>
            <label>Содержимое</label>
            <textarea name=\"content\" id=\"input-content
                \">".$contentValue."</textarea>
        </div>
        <div>
            <button type=\"submit\">Сохранить</button>
        </div>
    </form>
";
```

Здесь мы используем атрибут «value» для установки значений инпутов по умолчанию.

Как обычно, подключаем файл с шаблоном.

```
require("wiki_template_new.php");
```

Как и в случае со страницей вывода списка всех страниц, необходимо добавить ссылку на страницу создания новой страницы в основном меню.

```
<ul id="menu">
    <li><a href="all_pages.php">Страницы</a></li>
    <li><a href="create_update.php">Создать</a></li>
</ul>
```

Теперь нам нужно добавить в наше меню ссылку на редактирование текущей страницы. Для этого в файле wiki_page.php создаем переменную $pageId и приравниваем ее идентификатору текущей страницы.

```
$page = mysqli_fetch_assoc($result);
$title = $page["title"];
$content = $page["content"];
$pageId = $page["id"];
```

Далее в основном шаблоне производим проверку на существование переменной $pageId и, если она существует, выводим ссылку на страницу редактирования текущей страницы.

```
<ul id="menu">
    <li><a href="all_pages.php">Страницы</a></li>
    <li><a href="create_update.php">Создать</a></li>
    <?php if (isset($pageId)):?>
        <li><a href="create_update.php?id=<?=$pageId?>">Ре
            дактировать</a></li>
    <?php endif;?>
</ul>
```

Кроме этого, необходимо скорректировать вывод всех страниц и добавить к ссылкам на страницы еще и ссылки на редактирование страниц.

```
$content = "<ul>";
while ($page = mysqli_fetch_assoc($result)) {
    $content .= "<li><a href=\"/wiki_page.php?id=".$page["
        id"]."\">".$page["title"]."</a> | <a href=\"/
        create_update.php?id=".$page["id"]."\">Редактирова
        ть</a></li>";
}
$content .= "</ul>";
```

Из CRUD остался не реализованным только функционал удаления страницы. Для реализации этого функционала создадим файл delete_page.php и первым делом подключим к нему файл с подключением к БД.

```
require("connect_db.php");
```

Потом проверяем, был ли передан идентификатор страницы для удаления и, если он отсутствует, выводим соответствующее сообщение пользователю.

```
if (!isset($_GET["id"])) {
    echo "Не указан идентификатор удаляемой страницы.";
    exit;
}
```

Производим запрос к БД на удаление записи о странице.

```
mysqli_query($connect, "DELETE FROM wiki_pages WHERE id=".
    $_GET["id"]);
```

И переводим пользователя на экран вывода списка всех страниц.

```
header("Location: /all_pages.php");
```

Для того, чтобы пользователь удобным образом мог удалять страницы, нужно снова добавить ссылку на удаление текущей страницы.

```
<ul id="menu">
    <li><a href="all_pages.php">Страницы</a></li>
    <li><a href="create_update.php">Создать</a></li>
    <?php if (isset($pageId)):?>
        <li><a href="create_update.php?id=<?=$pageId?>">Ре
            дактировать</a></li>
        <li><a href="delete_page.php?id=<?=$pageId?>">Удал
            ить</a></li>
    <?php endif;?>
</ul>
```

На данный момент реализован CRUD и вывод списка страниц нашей википедии. Далее необходимо реализовать функционал регистрации пользователей, которые также являются создателями страниц.

Для начала необходимо создать таблицу users в нашей БД. В данной таблице будут следующие поля: id — уникальный идентификатор пользователя, name — имя пользователя, login — логин и password — md5 хэш от пароля. Для создания таблицы снова используем phpMyAdmin.

Далее создадим файл signup.php и включим в него файл подключения к БД.

```
require("connect_db.php");
```

Если методом POST были переданы данные, добавляем нового пользователя в БД и переводим его на страницу всех страниц.

```
if (!empty($_POST)) {
    mysqli_query($connect, "INSERT INTO users (name, login
        , password) VALUES (\"".$_POST["name"]."\", \"".
        $_POST["login"]."\", \"".md5($_POST["password"])
        ."\")");
    $id = mysqli_insert_id($connect);
    header("Location: /all_pages.php");
}
```

Иначе выводим форму для регистрации.

```
$title = "Регистрация";
$content = "
    <form method=\"POST\">
        <label>ФИО</label>
        <input type=\"text\" name=\"name\">
        <label>Логин</label>
        <input type=\"text\" name=\"login\">
        <label>Пароль</label>
        <input type=\"password\" name=\"password\">
        <label></label>
        <button type=\"submit\">Регистрация</submit>
    </form>
';
require("wiki_template_new.php");
```

На рисунке 11 приведен внешний вид формы регистрации нового пользователя.

1.8. Авторизация пользователей

Добавим функционал авторизации пользователя. Создадим файл auth.php и в нем в первую очередь подключим файл с подключением к БД.

```
require("connect_db.php");
```

Далее, предполагая, что методом POST были переданы логин и пароль, пытаемся получить из БД соответствующего пользователя.

```
$result = mysqli_query($connect, "SELECT * FROM users
    WHERE login=\"".$_POST['login']."\" AND password=\"".
    md5($_POST["password"])."\"');
```

Если нет такого пользователя, выводим соответствующее сообщение.

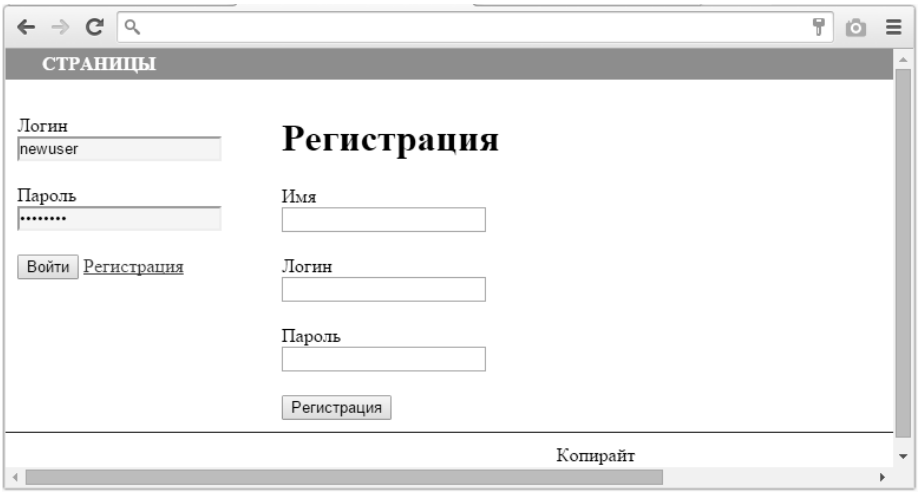

Рис. 11: Форма регистрации пользователей

```
if (!$result || mysqli_num_rows($result) == 0) {
    echo "Такого пользователя не существует.";
    exit;
}
```

Далее начинаем сессию с помощью функции session_start. После запуска сессии записываем в нее найденного пользователя.

```
session_start();
$_SESSION["user"] = mysqli_fetch_assoc($result);
```

После этого переводим пользователя на страницу вывода списка всех страниц википедии.

```
header("Location: /all_pages.php");
```

После того, как реализована авторизация, необходимо начинать сессию при каждом обращении пользователя на сервер и получать текущего пользователя из массива $_SESSION с индексом user. Для этого создадим файл session.php. В нем вызовем функцию session_start и проверим на существование индекса user в массиве $_SESSION. Если индекс user существует, то записываем $_SESSION[‘start’] в переменную $sessionUser. Если индекс user не существует, записываем в переменную $sessionUser значение false.

```
session_start();
$sessionUser = (isset($_SESSION["user"])) ? $_SESSION["
    user"] : false;
```

Далее, включаем файл session.php во все наши остальные файлы, как мы подключали файл connect_db.php. Для того, чтобы связать таблицу пользователей users и таблицу страниц википедии wiki_pages, необходимо в таблицу wiki_pages добавить поле user_id, в которое при создании страницы будет вписываться идентификатор текущего пользователя. Произведем соответствующие изменения в файле create_update.php.

```
if (!isset($_GET['id'])) {
    mysqli_query($connect, "INSERT INTO wiki_pages (title,
         content, user_id) VALUES (\"".
        mysqli_real_escape_string($connect, $_POST['title
        ']."\", \"".mysqli_real_escape_string($connect,
        $_POST["content"])."\",'".$sessionUser["id"]."')");
    $id = mysqli_insert_id($connect);
```

1.9. Связь между сущностями системы

После того, как была реализована авторизация, мы можем приступить к реализации правил доступа к страницам. Перечислим их: создавать новые страницы могут только авторизованные пользователи, редактировать и удалять существующие страницы могут только их авторы. Перечисленные правила доступа также влияют на пользовательский интерфейс следующим образом - выводить ссылку добавления новой страницы, только если пользователь авторизован; в списке страниц нашей википедии выводить ссылки на редактирование и удаление только у тех страниц, автором которых является текущий пользователь; выводить авторов на странице всех страниц; выводить ссылки редактирования и удаления текущей страницы, только если текущий пользователь является автором текущей страницы.

Можно заметить, что во многих случаях возникает задача проверки пользователя на авторизованность. Для проверки пользователя на авторизованность создадим файл check_auth.php, который будем включать в каждый файл, который требует авторизованности пользователя.

```
if (!$sessionUser) {
    echo "Необходимо авторизоваться.";
    exit;
}
```

Далее необходимо реализовать экран вывода данных о любом пользователе. Для этого создадим файл user.php.

```
require("connect_db.php");
require("session.php");
require("check_auth.php");
```

Потом проверяем, был ли передан идентификатор пользователя.

```
if (!isset($_GET["id"])) {
    echo "Укажите идентификатор пользователя.";
    exit;
}
```

Отправляем запрос в БД на получение пользователя и производим проверку.

```
$result = mysqli_query($connect, "SELECT * FROM users
    WHERE id=".$_GET["id"]);
if (!$result || !mysqli_num_rows($result)) {
    echo "Такого пользователя не существует.";
    exit;
}
```

Получаем пользователя и запрашиваем из БД все его страницы.

```
$user = mysqli_fetch_assoc($result);
$pages = array();
$pagesResult = mysqli_query($connect, "SELECT * FROM
    wiki_pages WHERE user_id=".$user["id"]);
```

Если список страниц не пустой, заполняем им массив $pages.

```
if ($pagesResult) {
    while ($page = mysqli_fetch_assoc($pagesResult)) {
        $pages[] = $page;
    }
}
```

Генерируем вывод.

```
$title = "Страница пользователя";
$content = "<p>".$user["name"]." (".$user["login"].")</p
    >";
$content .= "<h2>Страницы пользователя</h2>";
if (count($pages)) {
    $content .= "<ul>";
    foreach ($pages as $page) {
        $content .= "<li><a href=\"/wiki_page.php?id=".
            $page["id"]."\">".$page["title"]."</a></li>";
    }
    $content .= "</ul>";
} else {
    $content .= "<p>У данного пользователя еще нет страниц
        .</p>";
}

require("wiki_template_new.php");
```

1.10. Заключение

В данной главе мы познакомились с основами создания веб-сайтов — именно с языком разметки HTML, каскадными таблицами стилей CSS, языком программирования PHP, методами передачи данных веб-страницам GET и POST, веб-формами, хранением данных в СУБД MySQL, языком запросов SQL и одним из методов авторизации и запоминания пользователей сайта с помощью PHP. С примером реализации нашего аналога Википедии можно ознакомиться по ссылке http://book1.dvinemnauku.ru/all_pages.php [26].

2. Одностраничные приложения. Основы реализации географических онлайн-карт

В последнее десятилетие, в силу популярности и доступности интернета, актуальным направлением стало создание онлай-карт, доступных любому пользователю через браузер. В данном разделе рассматривается реализация онлайн-карты Москвы с расположенными на ней объектами (магазинами, кафе, ресторанами). Пользователь имеет возможность передвигаться по карте, переходить между тремя уровнями зума, переходить к таблице со списком объектов и с ее помощью находить любой объект на карте.

Цель данного раздела — дать описание основных приемов, которые можно использовать при создании онлайн-карт. Среди наиболее популярных картографических онлайн-систем можно выделить сервисы Google Maps [27] и Яндекс карты [28]. Стоит заметить, что кроме данных сервисов существуют также картографические JavaScript библиотеки, такие, как Open Layers [29] и Leaflet JS [30], на основе которых можно реализовывать интерактивные карты. Несмотря на существование вышеописанных сервисов и библиотек, в рамках корпоративных систем может быть полезным написание картографических онлайн модулей с нуля. Например, когда требуется узконаправленный функционал или нет возможности использовать внешние сервисы, а существующие библиотеки слишком избыточны для решения конкретной задачи.

В данной главе последовательно решается задача реализации онлайн-карты с возможностью перемещения по карте сдвигом курсора мыши, масштабирования карты в месте, где находится курсор, размещения объектов на карте путем редактирования списка объектов, а также поиска объектов и перехода из списка к карте с центрированием соответствующего объекта на карте.

При реализации картографических систем, карту не рассматривают как единое целое, а разбивают на изображения небольшого размера, называемые тайлами. При сдвиге карты, с сервера загружаются только те тайлы, которые попадают в видимую область. Связано это с тем, что цельное изображение карты может быть достаточно большого размера, тогда как на экран пользователя может поместиться только какая-то ее часть. Соответственно, одной из основных задач является

корректный расчет того, какие именно тайлы и в каком месте экрана необходимо отрисовать в каждый момент времени. Описание решения данной задачи дано в разделе **«Отрисовка тайлов»**.

Также предстоит решить, как именно перемещать карту. Этот вопрос рассматривается в разделе **«Событие движения дива map»**.

Другой важной задачей является корректная реализация переходов между уровнями зума. При увеличении или уменьшении уровня зума точка, на которую нажал пользователь, должна оказаться под курсором мыши после изменения размера. Описание реализации дается в разделе **«Изменение размера карты»**.

После решения вышеупомянутых задач необходимо решить проблему расстановки точек на карте. Каждая точка определяет местонахождение того или иного объекта на карте. В базе данных хранятся географические координаты объектов в виде долготы и широты, и задача состоит в том, чтобы по ним верно расставить объекты. Об этом рассказывается в разделе **«Отрисовка объектов на карте»**.

Также к подзадачам, связанным с работой с картой, стоит отнести центрирование на карте выбранной пользователем в таблице объектов точки. Эта проблема решается в разделе **«Центрирование точки»**.

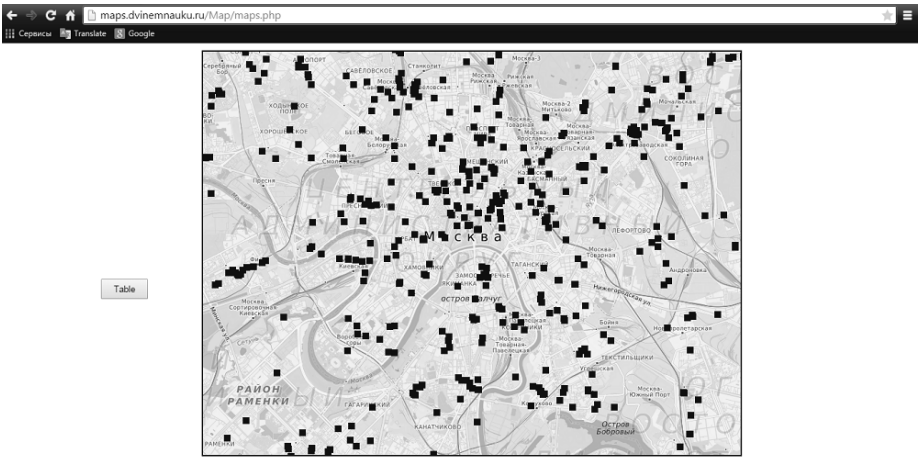

Рис. 12: Карта Москвы.

Работа со списком объектов подробно описана в разделе **«Список объектов»**. В подразделе **«Отображение таблицы»** описывается вы-

вод таблицы и поисковой формы на экран. В остальных подразделах описана серверная часть. В разделе **«Заполнение таблицы»** показано использование ajax запроса для заполнения строк данными из базы. В разделе **«Изменение данных в таблице»** описан механизм редактирования данных в таблице и сохранения их в базу. Последняя подзадача, а именно поиск данных в таблице базы данных, описана в разделе **«Поисковая форма»**.

На рисунке 12 приведен окончательный результат работы — карта Москвы с обозначенными на ней объектами.

2.1. База данных

Как и любое полноценное современное веб-приложение, онлайн-карты используют для работы данные, а именно список объектов, расположенных на ней. Список таких объектов хранится в базе данных с именем «maps». Для работы требуется две таблицы. В первой, с названием «moscow», хранятся уникальные id объектов (столбец id), их названия (столбец name), адреса (address), долготы (longitude) и широты (latitude), во второй, с названием «dots», хранятся экранные координаты (в пикселях) каждого объекта. Для этого используются столбцы X13, Y13, X15, Y15, X17, Y17 (по два столбца для каждого уровня зума). Эта таблица создана для удобства отображения точек на карте. Данные из «dots» позволяют не переводить географические координаты в экранные каждый раз, когда требуются эти координаты. На рисунке 13 приводится скриншот данной таблицы.

id	name	address	X13	Y13	X15	Y15	X17	Y17
2	Арбор Мунди	Москва, Лермонтовский проспект, 2К1	1432	1141	5730	4565	22923	18263
3	Dreamwear	Москва, улица Сущёвский Вал, 5С1	694	671	2777	2687	11109	10748
4	Apple	Москва, Русаковская улица, 1	918	736	3675	2945	14702	11783
5	Волшебный мир компьютеров (Ф-Центр)	Москва, Сухонская улица, 7А	867	249	3470	996	13882	3987
6	Гвоздь-2	Москва, проспект Андропова, 36	897	1286	3591	5147	14367	20588

Рис. 13: Таблица dots.

2.2. Основной функционал карт

Кратко опишем содержимое php-файла, в котором реализуется основной функционал карт. Сперва идет php-скрипт, где происходит подключение к базе данных. Далее следует HTML-заголовок, стандартный для большинства веб-страниц, в нем подключаются css-документ и библиотеки для работы с jQuery. Перед началом основного скрипта, описываемого далее, на страницу помещаются два основных дива и одна кнопка.

```
<div class = "mainbox">
    <div class = "map"></div>
</div>
<button class = "table_button">Table</button>
```

Внешний див «mainbox» представляет собой неподвижный прямоугольник с размерами 800 на 600, в котором помещается вся карта. Внутренний див «map» — это контейнер для тайлов, он не имеет размеров, и именно к нему привязывается событие «drag&drop», с помощью которого происходит движение карты внутри «mainbox». Кнопка «table_button» предназначена для перехода от карты к таблице со списком объектов.

Далее следует сам скрипт. Функции будут описаны в порядке их следования в файле.

2.2.1. Отрисовка тайлов

Функция Draw [31] отрисовывает тайлики внутри дива «mainbox». Функция принимает два параметра — левый и верхний сдвиг дива «map». Сначала считаем, с какого тайла начать отрисовку карты, другими словами, какой тайл окажется в верхнем левом углу дива «mainbox». Для этого берем отдельно левый и верхний сдвиги относительно «mainbox» и делим их на размер тайлов (curTileSize). От частного берем целую часть.

$$tileY = \left[\frac{-topOffset + offsetTopWindow}{curTileSize}\right]$$

$$tileX = \left[\frac{-leftOffset + offsetLeftWindow}{curTileSize}\right]$$

Затем в двойном (вложенном) цикле с помощью метода append добавляем к диву «map» новый див, свойству «background-image» которого присваиваем изображение нужного тайлика.

```
for (i = tileY - 1; i < countY + tileY + 1; i++){
    for (j = tileX - 1; j < countX + tileX + 1; j++){
        n = i+1;
        m = j+1;
        imgSrc = imgFile+'/'+i+'-'+j+'.png';
        $map.append('<div id = "'+n+'_'+m+'" class = "tile
            "> </div>');
        curTile = $('#'+n+'_'+m+'');
        curTile.css({
            'background-image': 'url('+imgSrc+')',
            'position': 'absolute',
            'top': (curTileSize) * i + 'px',
            'left': (curTileSize) * j  + 'px',
            'width': curTileSize  + 'px',
            'height': curTileSize  + 'px',
            'background-size' : curTileSize + 'px'
        });
    }
}
```

2.2.2. Событие движения дива map

Движение дива «map» — самое часто возникающее событие. Оно происходит при перетаскивании карты внутри дива «mainbox». При этом генерируются три новых события: «start», «drag», «stop» — начало движения, сам момент движения и прекращение движения соответственно. Мы используем только событие «stop», которое возникает в момент прекращения движения «map». Обработчик события в качестве параметра получает объект «ui», который имеет поле «ui.offset» — позиция перемещаемого элемента относительно начала документа. Внутри обработчика вызываются методы «Draw» и «DrawPoints».

```
$map.draggable({
    stop: function(event, ui){
        Draw(ui.offset.left, ui.offset.top);
        DrawPoints((-ui.offset.top + offsetTopWindow),(-ui
            .offset.left +
        offsetLeftWindow), zoom);
    }
});
```

2.2.3. Изменение размера карты

Функция Zoom [34] реализует изменение размеров карты, то есть увеличивает или уменьшает зум. Она принимает три аргумента — координаты мышки (mouseX, mouseY) относительно «mainbox» (то есть в левом верхнем углу дива координаты равны (0, 0)) и тип зума (zoomtype), который принимает значения 1 или -1 — увеличение и уменьшение зума соответственно. Процесс увеличения зума осуществляется следующим образом: размеры тайлов постепенно увеличиваются в 1.25, 1.5, 1,75 раз, а потом заменяются на тайлы из директории для следующего зума. Плавность увеличения обеспечивается функцией setTimeout, которая делает паузу перед выполнением кода, находящегося в нем, длина паузы устанавливается произвольно:

```
setTimeout(function(){
    ...
},30);
```

Здесь код, который находится внутри фигурных скобок вместо многоточия, выполнится с задержкой в 30 миллисекунд. Например, переход с зума уровня 13 на зум уровня 15 происходит следующим образом: сначала размер тайлов, который равен 256x256, умножается на 1.25, затем через 30 миллисекунд — на 1.5, еще через 60 миллисекунд умножается на 1.75 и, наконец, через 90 миллисекунд размер возвращается в исходный (256x256), и тайлы заменяются на необходимые для зума уровня 15. Очевидно, что если просто заменить тайлы таким образом, то область, которая отображалась на экране, сдвинется влево вниз, и точка, которую пользователь хотел увеличить, уйдет из его поля зрения, поэтому параллельно с изменением размера тайлов нужно менять положение карты. В представленном ниже коде описано увеличение тайлов на 1.25 и сдвиг карты таким образом, чтобы точка, на которую нажал пользователь, оказалась в том же месте, где и была изначально.

```
left_position = $map.position().left;
top_position = $map.position().top;
setTimeout(function(){
    l = mouseX - 1.25*(mouseX - left_position);
    t = mouseY - 1.25*(mouseY - top_position);
```

```
    $map.css
    ({
        'top': t + 'px',
        'left': l  + 'px',
    });
    curTileSize = startTileSize*1.25;
    Draw(l, t);
},0);
```

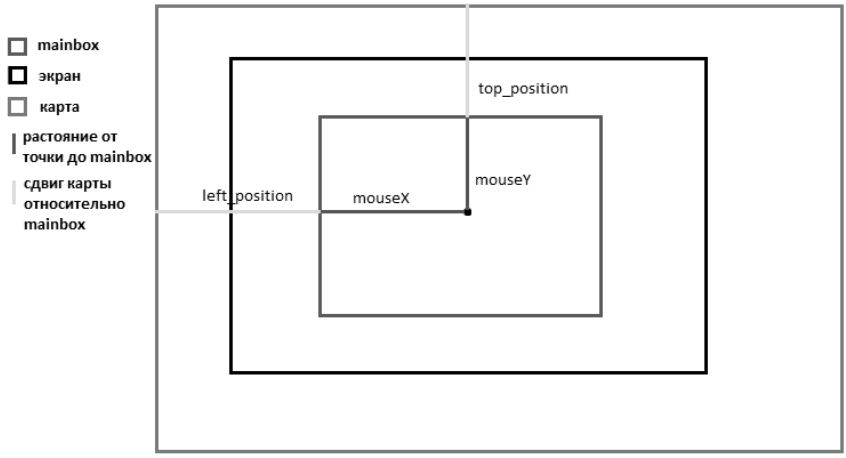

Рис. 14: Расчет новой позиции карты после изменения зума.

Разберем код построчно. Сначала получаем позицию карты относительно родительского элемента — дива «mainbox». Left_position — сдвиг карты влево, top_position — сдвиг карты вверх, средства jQuery позволяют получить эти данные с помощью метода position() элемента. Далее идет расчет новой позиции карты, на которую она должна встать после увеличения, рассмотрим его подробнее. Очевидно, если увеличить все тайлы на произвольное число (в нашем случае на 1.25), то расстояние от точки нажатия до левого верхнего угла карты увеличится на это же число, значит, чтобы вернуть точку в нужную позицию, необходимо отнять от текущей позиции разность увеличенного и исходного размеров. Сделаем это для левого сдвига, для верхнего все аналогично. Чтобы найти расстояние от точки нажатия до левой границы карты, необходимо к позиции курсора слева $mouseX$ прибавить левый сдвиг карты. Так как сдвиг — величина

отрицательная, то расстояние равно $mouseX - left\_position$, значит после увеличения оно будет равно $1.25 * (mouseX - left\_position)$. Нам необходимо сделать следующее: $left\_position - (1.25 * (mouseX - left\_position) - (mouseX - left\_position))$. Раскрыв скобки и произведя простые алгебраические действия, получим выражение: $mouseX - 1.25 * (mouseX - left\_position)$.

Далее, через css свойства объекта устанавливаем его в нужную позицию, увеличиваем размер тайликов и вызываем метод Draw. Таким же образом вызываем функцию setTimeout с задержками 30, 60 и 90. В последнем вызове заменяем директорию с тайлами на ту, где хранятся тайлы для следующего уровня зума. Например, для перехода от зума 13 к зуму 15 последний вызов setTimeout выглядит так:

```
setTimeout(function(){
    l = mouseX - 4*(mouseX - left_position);
    t = mouseY - 4*(mouseY - top_position);
    $map.css
    ({
        'top': t + 'px',
        'left': l  + 'px',
    });
    curTileSize = startTileSize ;
    imgFile = "../maptiles/z15";
    Draw(l, t);
    DrawPoints(l,t, zoom);
},90);
```

Как видно, размер тайлов возвращается в оригинальный (256x256), но расстояние до точки, на которую нажал пользователь, увеличилось в 4 раза. Это происходит потому, что один тайлик для зума уровня 13 содержит область, которую содержат 16 тайликов для зума уровня 15. Такое же соотношение выполняется при переходе от зума уровня 15 к уровню 17.

При двойном нажатии на карту левой кнопкой мыши должно произойти увеличение на один зум. В обработчике события запоминаются координаты курсора в момент нажатия и передаются как параметры в функцию Zoom, которая вызывается здесь же:

```
$map.on('dblclick', function(e){
    mouseX = event.x - offsetLeftWindow;
    mouseY = event.y - offsetTopWindow;
```

```
    Zoom(mouseX, mouseY, 1);
});
```

Обработчик события получает объект event, в котором содержатся поля event.x и event.y — координаты курсора мыши относительно экрана, но удобнее использовать координаты относительно mainbox. Для этого от event.x отнимаем расстояние между mainBox и левой границей экрана (offsetLeftWindow), аналогично поступаем с event.y. Вызываем функцию Zoom с параметром ZoomType, равным 1, так как по двойному нажатию левой кнопкой мыши мы увеличиваем зум.

При двойном нажатии на карту правой кнопкой мыши должно произойти уменьшение на один зум. В JavaScript нет события, отвечающего за двойное нажатие правой кнопкой мыши, но есть событие одинарного нажатия, при котором вызывается контекстное меню. Для того, чтобы искусственно создать событие двойного нажатия, поступим следующим образом. При первом нажатии будем запоминать время нажатия, а при втором вычислять разницу между текущим временем и временем предыдущего нажатия. Если разница не превышает одной секунды, то считаем, что пользователь совершил двойной клик правой кнопкой мыши и пожелал уменьшить уровень зума.

```
$map.on('contextmenu', function(e){
    if(cur_sec == -1){
        cur_time = new Date();
        cur_sec = cur_time.getSeconds();
    }
    else{
        times = new Date();
        tmp_sec = times.getSeconds() - cur_sec;
        if(tmp_sec<=1){
            mouseX = event.x - offsetLeftWindow;
            mouseY = event.y - offsetTopWindow;
            Zoom(mouseX, mouseY, -1);
        }
        cur_sec = -1;
    }
});
```

2.2.4. Отрисовка объектов на карте

Функция DrawPoints [32] с помощью AJAX запроса получает объекты, расположенные на карте в области, которая в данный момент

отображается на экране, и рисует их в виде точек. Запрос отправляется в файл «getPoints.php», который возвращает результат в json-формате. Затем в цикле точки (дивы небольшого размера) методом append добавляются к диву «map». Если при этом пользователь захотел выделить какой-либо объект, то есть перешел на карту из таблицы нажатием на какую-либо строку, то именно в этой функции нужный объект выделится цветом, отличным от остальных точек. В «getPoints.php» передаются координаты левого верхнего и правого нижнего углов экрана и текущий размер зума. С помощью этих данных выполняется SQL запрос к таблице «dots», где хранятся экранные координаты (в пикселях) каждого объекта для каждого размера зума.

2.2.5. Центрирование точки

Функция CountPosition [33] решает задачу перевода географических координат (долготы и широты) в экранные (пиксели). Она вызывается, если пользователь перешел на карту из таблицы, желая выделить на карте какой-либо объект. При этом с помощью GET запроса в файл передаются долгота и широта нужного объекта, которые являются параметрами функции. Перевод координат производится по формулам, которые были получены из следующих соотношений. С помощью Google Maps были определены точные координаты левого верхнего и нижнего правого углов карты. Далее, зная размеры карты для каждого из зумов, можно посчитать, сколько пикселей приходится на один градус широты и долготы. Также здесь меняется позиция дива «map» таким образом, чтобы выделенный объект оказался в центре видимой области карты.

2.3. Список объектов на карте

В текущем разделе мы опишем функционал, реализованный для работы с объектами на карте.

2.3.1. Отображение таблицы

При описании файла maps.php мы упомянули о кнопке, расположенной по левую сторону от карты, см. рис. 12. При нажатии на нее пользователь переходит на страницу с таблицей, содержащей список

всех объектов. Таблица не растянута на всю страницу, а, как и карта, расположена внутри дива размеров 800x600, будем называть его «Box». Также на странице расположена поисковая форма для поиска обьектов по имени или адресу. В начале файла идет php-скрипт, в котором вычисляются два важных значения — общее количество строк в таблице и количество строк, которые помещаются в Box.

```
<?php
    header('Content-Type: text/html; charset='cp-1251');
    $connect = mysqli_connect('localhost', 'maps', '
        maps12345')
    or die("MySQL connection error");
    mysqli_select_db($connect, 'maps') or die("Select
        database error");
    mysqli_set_charset($connect,'cp-1251');
    $q = mysqli_query($connect, "SELECT COUNT(*) FROM `
        moscow`");
    $numRows = mysqli_fetch_array($q)[0];
    $row_on_screen = (int)((600/20));
?>
```

Как видно, общее количество строк сохраняется в переменную $numRows, а количество строк, помещающихся в Box — в $row_on_screen. Число 600 в расчете $row_on_screen — высота Box, а 20 — высота строки.

Далее переместимся к тегу body. В нем расположен код, создающий саму таблицу. Так как количество строк достаточно велико, было принято решение создавать изначально количество строк, равное 2*$row_on_screen. То есть при обновлении страницы таблица имеет удвоенное количество строк, помещающихся в Box (в нашем случае это 60 строк). При скролле таблицы вниз к ней добавляются строки, а данные в них загружаются с помощью ajax запроса, иначе при большом количестве строк нагрузка на страницу была бы очень велика. Далее приведен код тега body, в котором создаются все элементы формы.

```
<body>
<div class ='find'>
<div>Name</div>
<input type="text" size="25" class='name'><br><br>
<div>Address</div>
<input type="text" size="25" class='address'><br><br>
```

```
<button class='find_button'>Find</button><br><br>
<button class='cancel_find'>Cancel</button>
</div>
<div class='Box'>
  <table class='table' border="1">
    <?php for($i=1;$i<=2*$row_on_screen;$i++) : ?>
      <tr class='Tr' id='<?=$i?>'>
        <td column_name= 'id' class='Td' ></td>
        <td column_name= 'name' class='Td' ></td>
        <td column_name= 'address' class='Td' ></td>
        <td column_name= 'longitude' class='Td'></td>
        <td column_name= 'breadth' class='Td' ></td>
      </tr>
    <?php endfor ?>
  </table>
</div>
</body>
```

2.3.2. Заполнение таблицы

Функция заполнения таблицы [35] вызывается при скролле таблицы. Она заполняет ее строки данными из базы. Функция принимает два параметра, begin — номер строки, которая при текущем уровне прокрутки находится вверху таблицы и cnt — количество строк, которое необходимо заполнить. Сначала формируется GET запрос к файлу rows.php, в котором выполняется соответствующий SQL запрос.

```
var request = "n=" + begin + "&m=" + cnt;
$.ajax({
    url : "rows.php",
    data : request,
    ...
})
```

Данные, полученные в результате запроса, возвращаются из файла rows.php в json формате.

```
$.ajax({
    url : "rows.php",
    data : request,
    dataType : "json",
    success: function(data){
        var str = JSON.stringify(data);
        var tmp = JSON.parse(str);
        var i;
```

```
        for (i = 0, j = begin; i < cnt; i++, j++){
            $tr = $('#'+j);
            $tr.children("td").eq(0).text(tmp[i].id);
            $tr.children("td").eq(1).text(tmp[i].name);
            $tr.children("td").eq(2).text(tmp[i].address);
            $tr.children("td").eq(3).text(tmp[i].longitude
                );
            $tr.children("td").eq(4).text(tmp[i].breadth);
        }
    }
});
```

Функция $.ajax(. . .) содержит секцию success, код внутри которой выполняется только в случае успешного выполнения запроса. Если rows.php успешно вернул данные, то в цикле осуществляется заполнение строк таблицы.

2.3.3. Событие скролла таблицы

Обработчик события работает следующим образом. При прокрутке считается количество прокрученных строк, и именно это количество строк добавляется в конец таблицы. Для них вызывается функция FillTable.

2.3.4. Изменение данных в таблице

При нажатии на любую ячейку таблицы, кроме ячейки id, она переходит в режим редактирования данных. То есть у пользователя есть возможность менять название или адрес любого объекта. Сохранение новых данных происходит в момент, когда с редактируемой ячейки снимается фокус, другими словами, когда пользователь нажимает в другое произвольное место таблицы. Сохранением данных в базу занимается функция UpdateTable, в качестве параметров она принимает id строки, название колонки, ячейка которой редактировалась — column_name и новые данные — nw_data. Эти данные с помощью ajax запроса отправляются в файл updateRows.php, в котором формируется update запрос к базе данных. Код функции небольшой, поэтому приведем его целиком:

```
    function UpdateTable(id, column_name, nw_data){
        var request = "id=" + id + "&column_name=" +
            column_name +
            "&nw_data="+nw_data;

     $.ajax({
         url : "updateRows.php",
         data : request,
     });
}
```

2.3.5. Поисковая форма

Поисковая форма позволяет пользователю искать объекты в таблице по имени или адресу. Пользователь вводит данные в поля Name и/или Address, и при нажатии на кнопку Find в таблице остаются только строки, удовлетворяющие запросу пользователя. Вышесказанное реализовано в обработчике события нажатия на кнопку Find посредством ajax запроса в файл find_rows.php.

```
$(".find_button").on('click', function(){
    var name = $('.name').val();
    var address = $('.address').val();
    var request = "name=" + name + "&address=" + address;
    $.ajax({
        url : "find_rows.php",
        data : request,
        dataType : "json",
        \\...
    });
});
```

Полученные данные, так же как и в функции FillTable, записываются в строки таблицы. Кнопка Cancel предназначена для возврата таблицы в прежнее состояние.

2.3.6. Фотография точки

Функционал загрузки фотографии точки реализован в файле postImage.php. В начале указывается тип ответа в результате работы приложения и подключается файл с проверкой авторизованности пользователя.

```
$responseType = 'json';
require('auth.php');
```

Затем производится проверка на существование и формат загруженного файла.

```
if (!isset($_FILES['image']) || $_FILES['image']['
    error'] != UPLOAD_ERR_OK) {
    response($responseType, '400', 'Can not upload
        file.', $res = 0);
}

if (exif_imagetype($_FILES['image']['tmp_name']) !=
    IMAGETYPE_JPEG) {
    response($responseType, '400', 'Can not upload not
         JPEG file.', $res = 0);
}
```

После всех проверок файл перемещается в необходимую директорию под названием <идентификатор точки>.jpg и в ответ выводится JSON со значением res = 1;

```
move_uploaded_file($_FILES['image']['tmp_name'],
    dirname(__FILE__).'/photo/'.$_GET['point_id'].'.
    jpg');
responseAnyJSON(200, array(), 1);
```

2.3.7. Товары точки

Список товаров точки можно получить с помощью функции getPointGoods, расположенной в файле functions.php. Функция принимает два параметра: $pageId - идентификатор страницы и $connection - подключение к БД.

```
function getPointGoods($pointId, $connect)
{
```

Производится запрос к БД.

```
    $q = 'SELECT * FROM points_goods pg JOIN goods g
        ON pg.good_id = g.id WHERE point_id='.$pointId
        .'';
    $res = mysqli_query($connect, $q);
```

Если результат запроса не пустой, заполняется массив $goods и возвращается как результат работы функции.

```
    $goods = array();
    if ($res) {
        while ($mas = mysqli_fetch_assoc($res)) {
            $goods[] = array(
                'id' => intval($mas['id']),
                'good_id' => intval($mas['good_id']),
                'point_id' => intval($mas['point_id'])
                    ,
                'count' => intval($mas['count']),
                'name' => $mas['name']
            );
        }
    }
    return $goods;
}
```

2.3.8. Авторизация пользователей

За функционал авторизации пользователей отвечают три файла. Файл login.php содержит функционал авторизации пользователя, файл auth.php отвечает за получение информации о текущем пользователе либо из массива $_SESSION, либо из БД по переданному GET параметру auth_key и файл logout.php удаляет запись о текущем пользователе из массива $_SESSION. Опишем каждый файл подробнее.

Файл login.php начинается с начала пользовательской сессии и подключения необходимых внешних файлов.

```
session_start();

require('connectDB.php');
require('response.php');
```

Если передан GET параметр, равный значению json, необходимо возвратить ответ в виде JSON. В противном случае ответ будет возвращен в формате html. Данное разделение сделано для того, чтобы мобильное приложение могло тоже авторизовать пользователя.

```
$responseType = (isset($_GET['type']) && $_GET['type']
    == 'json') ? 'json' : 'html';
```

Далее определяются некоторые переменные и проверяется массив $_POST на наличие индексов login и pwd, отвечающие за логин и пароль соответственно.

```
$message = '';
$loggedIn = false;
if (isset($_POST['login']) && isset($_POST['pwd'])) {
```

Далее делается запрос к БД для попытки загрузки пользователя с таким логином и паролем.

```
    $userDB = mysqli_query($connect, 'SELECT * FROM
        users WHERE login="'.mysqli_real_escape_string
        ($connect, $_POST['login']).'" AND pwd="'.md5(
        $_POST['pwd']).'"');
    $user = mysqli_fetch_assoc($userDB);
```

Если не удалость найти пользователя с заданными параметрами, необходимо вывести соответствующее сообщение.

```
    if (!$user) {
        if ($responseType == 'html') {
            $message = 'Wrong username or password.';
        } else {
            response($responseType, 400, 'Wrong
                username or password.', 0);
        }
    } else {
```

В другом случае, если удалось найти соответствующего пользователя, его данные записываются в массив $_SESSION.

```
        unset($user['pwd']);
        $_SESSION['user'] = $user;
        $loggedIn = true;
    }
}
```

Далее выводится ответ на запрос пользователя в зависимости от того, получилось ли залогинить пользователя.

```
    if ($responseType == 'json' && $loggedIn) {
        responseAnyJSON(200, $_SESSION['user'], 1);
    } else if ($responseType == 'json' && !$loggedIn) {
        response($responseType, 400, 'Wrong username of
            password.', 0);
    } else if ($responseType == 'html' && $loggedIn) {
        header('Location: '.currentURL().'/Map/maps.php');
        exit;
    }
?>

<!doctype html>
<html>
    <head>
    </head>
    <body>
        <form method="POST">
            <b><?=$message?></b>
            <input type="text" name="login">
            <input type="password" name="pwd">
            <button type="submit" name="">Enter</button>
        </form>
    </body>
</html>
```

Как было сказано ранее, файл auth.php отвечает за проверку авторизованности пользователя. В начале начинается пользовательская сессия и подключаются необходимые внешние файлы.

```
session_start();

require('connectDB.php');
require('response.php');
```

Далее, если есть GET параметр auth_key, то извлекаем пользователя с таким ключом авторизации и записываем его в массив $_SESSION.

```
    if (isset($_GET['auth_key'])) {
        $userDB = mysqli_query($connect, 'SELECT * FROM
            users WHERE auth_key="'.
            mysqli_real_escape_string($connect, $_GET['
            auth_key']).'"');
```

```
    if (!$userDB) {
        response($responseType, 401, 'Wrong
            authorization key.', 0);
    }
    $user = mysqli_fetch_assoc($userDB);
    unset($user['pwd']);
    $_SESSION['user'] = $user;
}
```

Далее, в массиве $_SESSION проверяется на существование индекс user. Если такого индекса нет, выводится сообщение об ошибке.

```
if (!isset($_SESSION['user'])) {
    responseOrRedirect($responseType, '/Map/login.php
        ', 401, 'Must authorize.', 0);
}
```

Если все прошло успешно и пользователь авторизован, данные о пользователе записываются в переменную $sessionUser.

```
$sessionUser = $_SESSION['user'];
```

В файле logout.php содержится удаление индекса user из массива $_SESSION, и пользователь перенаправляется на страницу с картой.

```
session_start();
require('response.php');

unset($_SESSION['user']);
header('Location: '.currentURL().'/Map/maps.php');
exit;
```

3. Поисковые системы

В данном разделе будут рассмотрены поисковые системы. Их задачей является осуществление полнотекстового поиска среди веб-страниц (далее просто документы). Необходимость в них возникла сравнительно недавно, с появлением и очень стремительным развитием интернета в 90-х годах XX века.

При построении поисковых систем разработчики сталкиваются с очень большим числом сложностей, возникающих вследствие большого числа факторов. В первую очередь это большое количество документов в интернете. Разные источники приводят разные цифры, но все сходятся в том, что их число оценивается миллиардами. Необходимо учесть, что многие сайты показывают сгенерированные документы, которые меняются со временем. При этом отсутствует какая-либо структурированность содержимого документов. Доступность добавления информации в интернет приводит к тому, что качество документов может быть как высоким, так и низким, так как контент создается не профессиональными редакторами, а кем угодно. Вследствие этого возникает задача построения поисковых систем, способных за малое время находить документы, которые будут потенциально полезны пользователю.

В разделе «Структура базы данных, подключение к ней и стоп-слова» будет рассмотрена структура базы данных и некоторые подготовительные работы для реализации поисковой системы. Большинство поисковых систем включают в себя кроулер, индексер и поисковый интерфейс.

Кроулер представляет из себя сбор документов из интернета с целью дальнейшей работы с ними и сохранения связей, которые состоят из ссылок между ними. Кроулер подробнее будет рассмотрен в разделе «Кроулер».

Индексер — это подсистема для индексации документов с целью получения информации о том, какие слова содержатся в каких документах. Индексер будет рассмотрен подробнее в разделе «Индексер».

Поисковый интерфейс — веб-интерфейс поисковой системы, позволяющий вводить поисковые запросы и получать в ответ подходящие страницы (поисковую выдачу). Поисковый интерфейс будет разработан в разделе «Поисковый интерфейс».

В дальнейшем мы рассмотрим каждую подсистему подробнее и

реализуем их. Не все документы, содержащие слова из поискового запроса, являются одинаково хорошими источниками информации для пользователей. Сергей Брин, сооснователь компании Google, сказал: «Мы пришли к выводу, что не все веб-страницы созданы равными. Люди — да, но не веб-страницы». Возникает задача сортировки документов, входящих в поисковую выдачу. Эта задача называется ранжированием (раздел «Ранжирование»). Далее будет рассмотрено и реализовано два типа ранжирования. Ссылочное ранжирование — это ранжирование документов на основе анализа ссылок между ними. Мы рассмотрим два алгоритма ссылочного ранжирования: HITS (раздел «Алгоритм ранжирования HITS») и PageRank (раздел «Алгоритм ранжирования PageRank»). Далее рассмотрим алгоритмы ранжирования на основе мер TF и IDF. В разделе «Подсчет мер TF и IDF» будет реализован алгоритм подсчета мер TF и IDF. Будут рассмотрены два алгоритма ранжирования документов по поисковым запросам: на основе расчета косинуса угла между векторами (раздел «Алгоритм ранжирования на основе подсчета косинусов углов между векторами») и Okapi BM25 (раздел «Алгоритм ранжирования Okapi BM25»).

Далее в разделе «Применение ранжирования в поисковой выдаче» будет рассмотрено применение алгоритмов ранжирования в генерации поисковой выдачи.

В итоге будет разработана поисковая система, которую будет возможно протестировать на реальных данных. На рисунках 15 и 16 приведены результаты поиска слова «plane» без ранжирования [40] и с ранжированием [41] соответственно.

3.1. Структура базы данных, подключение к ней и стоп-слова

Для реализации поисковой системы мы используем систему управления базами данных MySQL. Вначале необходимо разработать структуру базы данных. Назовем базу данных search_engine. Перечислим таблицы и их поля.

Таблица page служит для хранения документов, которые проиндексированы. Таблица состоит из полей id — уникальный идентификатор документа, url — URL документа, name — название файла на локальном диске, checked — был ли этот документ обработан кроулером, p_rank — PageRank документа, h_auth — оценка авторитетности

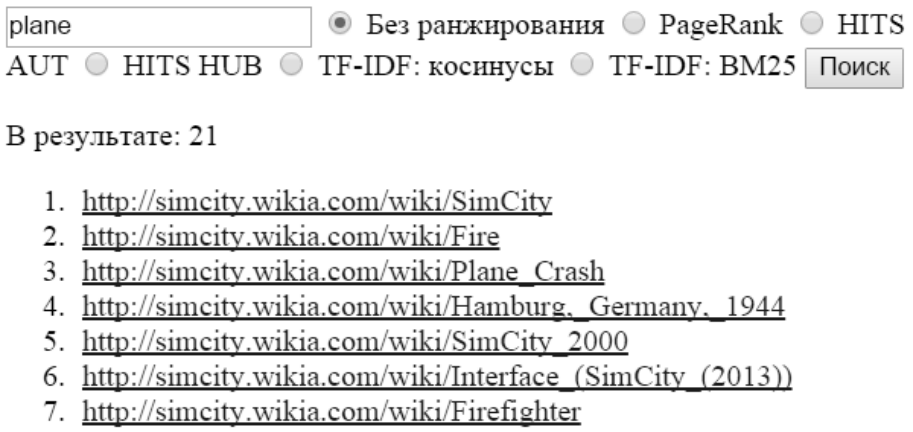

Рис. 15: Поиск слова «plane» без применения ранжирования.

документа по алгоритму HITS, h_hub — оценка хабности документа по алгоритму HITS.

Таблица link служит для хранения ссылок. В ней поле id — уникальный идентификатор ссылки, page_from — уникальный идентификатор документа, в котором находится ссылка, page_to — уникальный идентификатор документа, на который ведет ссылка.

Таблица term служит для хранения термов, полученных со всех проиндексированных документов. Она состоит из полей id — уникальный идентификатор терма, term — основа слова, полученная стеммингом (см. раздел «Индексер»), idf — мера IDF данного терма среди всех проиндексированных документов.

Таблица term_page служит для связки между таблицами page и term. В ней содержатся записи слов и документов. В ней id — идентификатор записи, term_id — уникальный идентификатор терма, page_id — уникальный идентификатор документа, num — сколько раз встречается терм в данном документе, tf — мера TF терма в данном документе, tf_idf — мера TF-IDF данного терма в данном документе.

Также в системе имеется таблица stop_word стоп-слов (см. описание далее в данном разделе), где id — уникальный идентификатор стоп-слова, term — основа стоп-слова.

Подключение к базе данных содержится в файле db.php [42]:

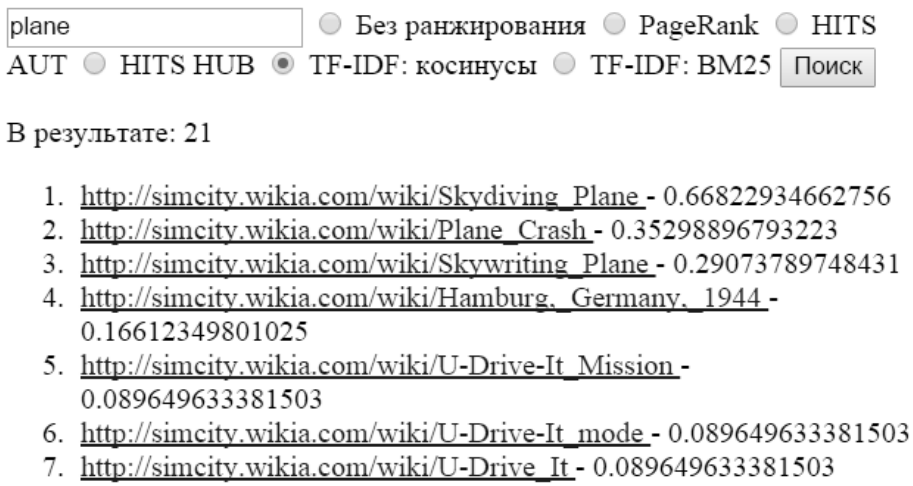

Рис. 16: Поиск слова «plane» с применением ранжирования.

```
define('DB_HOSTNAME', 'localhost');
define('DB_NAME', 'search_engine');
define('DB_USER', 'root');
define('DB_PASSWORD', '');

$dbLink = mysqli_connect(DB_HOSTNAME, DB_USER, DB_PASSWORD
    , DB_NAME) or die('Error: can\'t connect to the
    database.');
mysqli_set_charset($dbLink, 'utf8');
```

Далее необходимо заполнить таблицу стоп-слов. Список стоп-слов можно легко найти в интернете. Данная поисковая система работает с английским языком, и, соответственно, список стоп-слов берется для английского языка. Ранее было сказано, что работа будет вестись не с исходным видом слов, а с их основами, которые мы называем термами. Будем использовать реализацию алгоритма Porter Stemmer, разработанную Ричардом Хайсом. Стеммер размещен в файле stemmer.php. Создадим файл stopword.php [43] и добавим в него файлы подключения к базе данных и стеммера. Создадим массив стоп-слов, так как их не так много. Очистим таблицу стоп-слов и в цикле сохраним терм от каждого слова в базе данных.

```
include 'db.php';
include 'stemmer.php';

$stopWords = array("a", "able", "about", "across", "after
    ", "all", "almost", "also", "am", "among", "an", "and
    ", "any", "are", "as", "at", "be", "because", "been",
    "but", "by", "can", "cannot", "could", "dear", "did",
    "do", "does", "either", "else", "ever", "every", "for
    ", "from", "get", "got", "had", "has", "have", "he", "
    her", "hers", "him", "his", "how", "however", "i", "if
    ", "in", "into", "is", "it", "its", "just", "least", "
    let", "like", "likely", "may", "me", "might", "most",
    "must", "my", "neither", "no", "nor", "not", "of", "
    off", "often", "on", "only", "or", "other", "our", "
    own", "rather", "said", "say", "says", "she", "should
    ", "since", "so", "some", "than", "that", "the", "
    their", "them", "then", "there", "these", "they", "
    this", "tis", "to", "too", "twas", "us", "wants", "was
    ", "we", "were", "what", "when", "where", "which", "
    while", "who", "whom", "why", "will", "with", "would",
     "yet", "you", "your", "ain't", "aren't", "can't", "
    could've", "couldn't", "didn't", "doesn't", "don't", "
    hasn't", "he'd", "he'll", "he's", "how'd", "how'll", "
    how's", "i'd", "i'll", "i'm", "i've", "isn't", "it's",
     "might've", "mightn't", "must've", "mustn't", "shan't
    ", "she'd", "she'll", "she's", "should've", "shouldn't
    ", "that'll", "that's", "there's", "they'd", "they'll
    ", "they're", "they've", "wasn't", "we'd", "we'll", "
    we're", "weren't", "what'd", "what's", "when'd", "when
    'll", "when's", "where'd", "where'll", "where's", "who
    'd", "who'll", "who's", "why'd", "why'll", "why's", "
    won't", "would've", "wouldn't", "you'd", "you'll", "
    you're", "you've");

mysqli_query($dbLink, 'TRUNCATE TABLE stop_word');

foreach ($stopWords as $stopWord) {
    $term = PorterStemmer::Stem($stopWord);
    $q = "INSERT INTO stop_word(term) values('".$term."')
        ";
    mysqli_query($dbLink, $q);
}
```

Итак, на текущий момент разработана база данных для поисковой системы, создан файл с подключением к базе данных, заполнена таблица стоп-слов английского языка.

3.2. Кроулер

Задачей кроулера является сбор документов из интернета с целью дальнейшей работы с ними и сохранения связей между документами, задаваемыми ссылками между ними. Интернет можно представить в виде ориентированного графа, где вершины — это документы, а ребра — это ссылки. Документ, в котором находится ссылка — начальная вершина ребра, а документ, на который ведет ссылка — конечная вершина ребра. Данный граф называется веб-графом. Алгоритм работы кроулера похож на обход графа в ширину.

Для работы кроулера необходимо вести список документов, которые ему необходимо посетить и которые он уже посетил. Назовем список не посещенных документов границей (вследствие того, что работа кроулера похожа на обход графа в ширину). В начале работы граница задается нами вручную и список документов состоит только из одного заданного нами документа. Посещение документа состоит из следующих шагов. Сначала мы получаем документ из интернета. Далее выделяем текст из содержимого документа и сохраняем его на локальный диск. После этого собираем ссылки в документе и для каждой из них сохраняем документ, на который ведет ссылка, как граничный, если он отсутствует в списке документов. Сохраняем ссылку, если она отсутствовала в базе данных. Далее помечаем текущий документ как пройденный.

В реализации кроулера мы используем библиотеку PHP Simple HTML DOM Parser, которая позволяет легко работать с HTML страницами. Библиотека размещена в файл simple_html_dom.php.

До того, как начинать реализацию кроулера, мы создаем файл functions.php [44], в котором размещаются функции, используемые в различных подсистемах поисковой системы. Первой функцией в данном файле является функция getPath, используемая для получения пути к файлу на локальном диске по его названию. В данном случае файлы страниц находятся в директории crawl_data в корневой директории проекта.

```
function getPath($fileName){
    return dirname(__FILE__).'/crawl_data/'.$fileName;
}
```

Создадим файл crawler.php [45] и подключим к нему файл с подключением к базе данных, файл с общими функциями и библиотеку

PHP Simple HTML DOM Parser. В данном примере мы загружаем документы с сайта `http://simcity.wikia.com`. Это сайт с данными об игре Sim City.

```
include 'db.php';
include 'functions.php';
include 'simple_html_dom.php';
```

Уберем ограничение по времени выполнения скрипта:

```
set_time_limit(0);
```

Реализуем функцию getLinks, которая возвращает массив ссылок, находящихся в документе, сохраненном на локальном диске. Используя библиотеку PHP Simple HTML DOM Parser и ее функцию file_get_html, получаем объект для работы с HTML страницей. Если эта функция вернула значение false, возвращаем null. Извлекаем все ссылки из документа. В дальнейшем ссылки нам понадобятся в алгоритмах ранжирования результатов поиска.

```
function getLinks($filePath){
    if (!file_exists($filePath)) {
        return null;
    }

    $html = file_get_html($filePath);

    if ($html === false) {
        return null;
    }

    $retLinks = array();
    $links = $html->find('a');
    foreach ($links as $link) {
        $retLinks[] = $link->href;
    }

    $html->clear();
    unset($html);
    return !empty($retLinks) ? $retLinks : null;
}
```

Реализуем функцию findPageByUrl, которая находит документ в таблице page базы данных по его URL. Функция делает запрос к таблице page и возвращает его результат.

```
function findPageByUrl($url, $dbLink){
    $q = "SELECT * FROM page WHERE url='".$url."'";
    $res = mysqli_query($dbLink, $q);
    $page = mysqli_fetch_array($res);
    return $page;
}
```

Опишем функцию createNewPage, которая создает новый документ в базе данных. Вначале производим проверку документа с заданным URL в базе данных. Если документ с заданным URL уже существует в базе данных, возвращаем значение false. Иначе создаем документ в базе данных и возвращаем значение true.

```
function createNewPage($pageUrl, $dbLink){
    $q = "SELECT * FROM page WHERE url = '".$pageUrl."'";
    $res = mysqli_query($dbLink, $q);
    $page = mysqli_fetch_array($res);
    if ($page) {
        return false;
    }
    $q = "INSERT INTO page(url, checked) values('".
        $pageUrl."', '0')";
    mysqli_query($dbLink, $q);
    return true;
}
```

Создадим функцию generateRandomString для генерации случайной строки. Она понадобится при сохранении документа на локальном диске. На входе функция получает необходимую длину строки и возвращает строку необходимой длины, состоящую из случайных символов.

```
function generateRandomString($length = 10){
    $characters = '0123456789
        abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
        ';
    $charactersLength = strlen($characters);
    $randomString = '';
    for ($i = 0; $i < $length; $i++) {
        $randomString .= $characters[rand(0,
            $charactersLength - 1)];
    }
    return $randomString;
}
```

Опишем функцию checkUrl получения HTTP кода при обращении

по ссылке. Данная функция понадобится при загрузке документов на локальный диск. Если при обращении по данному URL ответ содержит код 200, то документ существует и его можно загрузить. Используя библиотеку curl, инициализируем HTTP запрос. Производим запрос, получаем заголовки и закрываем соединение. В конце возвращаем HTTP код.

```
function checkUrl($url) {
    $ch = curl_init();
    curl_setopt($ch, CURLOPT_URL, $url);
    curl_setopt($ch, CURLOPT_HEADER, 1);
    curl_setopt($ch , CURLOPT_RETURNTRANSFER, 1);
    $data = curl_exec($ch);
    $headers = curl_getinfo($ch);
    curl_close($ch);
    return $headers['http_code'];
}
```

Далее реализуем функцию loadPage, которая используется для сохранения документа на локальный диск по его URL. Если при обращении по данному URL возвращается статус, не равный 200, возвращаем значение null. Получаем объект библиотеки PHP Simple HTML DOM Parser для дальнейшей обработки документа. В документах с данными с сайта `http://simcity.wikia.com` нас интересует только содержимое элемента div с id, равным mw-content-text. Очищаем и удаляем объект PHP Simple HTML DOM Parser, чтобы освободить память. Удаляем скрипты JavaScript из документа. Удаляем все HTML теги, кроме тега a, из документа. Удаляем все внешние ссылки. Генерируем случайное название файла и сохраняем документ на локальный диск. Возвращаем название файла документа на локальном диске.

```
function loadPage($url){
    $check_url_status = checkUrl($url);
    if ($check_url_status != '200') {
        return null;
    }
    $html = file_get_html($url);
    if ($html === false) {
        return null;
    }
```

```
    foreach ($html->find('#mw-content-text') as $article)
        {
         $str = $article->innertext;
    }

    $html->clear();
    unset($html);

    $q = '/<script>.*<\/script>/';
    $str = preg_replace($q, '', $str);

    $q = '/<[^a\/].*?>/';
    $str = preg_replace($q, '', $str);

    $q = '/<\/[^a].*?>/';
    $str = preg_replace($q, '', $str);

    $q = '/<a[^>]+href[^>]+(https?|[?#:.]+)[^>]+>.+?<\/a
        >/';
    $str = preg_replace($q, '', $str);

    $fileName = generateRandomString().'.html';
    $pathToFile = getPath($fileName);

    $fp = fopen($pathToFile, 'w');

    if ($fp === false) {
        return null;
    }
    fwrite($fp, $str);
    fclose($fp);
    return $fileName;
}
```

Напишем функцию createNewLink, которая используется для создания новой ссылки в базе данных. Если такая ссылка уже существует в базе данных, возвращаем значение false. Сохраняем ссылку в базе данных и возвращаем значение true.

```
function createNewLink($from, $to, $dbLink){
    $q = "SELECT * FROM link WHERE page_from = ".$from."
        AND page_to = ".$to;
    $res = mysqli_query($dbLink, $q);
    $link = mysqli_fetch_array($res);
    if ($link) {
        return false;
    }
    $q = "INSERT INTO link(page_from, page_to) values(".
        $from.", ".$to.")";
    mysqli_query($dbLink, $q);
    return true;
}
```

Перейдем к описанию собственно работы кроулера. Перед запуском кроулера необходимо очистить таблицы page и link. Далее задаем начальную границу. В данном случае она состоит из одного документа с URL `http://simcity.wikia.com/wiki/SimCity`. В цикле прорабатываем все документы, пока еще остаются граничные документы. Получаем список ссылок в документе, сохраняем ссылки. Сохраняем в базе данных название файла на локальном диске и помечаем документ как пройденный. По результатам работы кроулера выводим на экран число загруженных документов.

```
mysqli_query($dbLink, 'TRUNCATE TABLE page');
mysqli_query($dbLink, 'TRUNCATE TABLE link');

$frontierUrls = array(
    'http://simcity.wikia.com/wiki/SimCity'
);
foreach ($frontierUrls as $url) {
    createNewPage($url, $dbLink);
}

$count = 0;
while (true) {
    $q = "SELECT * FROM page WHERE checked = 0 LIMIT 1";
    $res = mysqli_query($dbLink, $q);
    $page = mysqli_fetch_array($res);

    if (!$page) {
        break;
    }

    $fileName = loadPage($page['url']);
```

```
    if (!$fileName) {
        $q = "UPDATE page SET checked = 1 WHERE id = '".
            $page['id']."'";
        mysqli_query($dbLink, $q);
        continue;
    }
    $filePath = getPath($fileName);
    $urls = getLinks($filePath);
    if (is_array($urls)) {
        foreach ($urls as $url) {
            if (!preg_match('/^\//', $url)) {
                continue;
            }
            $url = 'http://simcity.wikia.com'.$url;
            createNewPage($url, $dbLink);
            $newPage = findPageByUrl($url, $dbLink);
            createNewLink($page['id'], $newPage['id'],
                $dbLink);
        }
    }

    $q = "UPDATE page SET name = '".$fileName."', checked
        = 1 WHERE id = '".$page['id']."'";
    mysqli_query($dbLink, $q);
    $count += 1;
}

echo 'Number of loaded pages: '.$count;
```

Если задать в качестве начальной границы документ `http://simcity.wikia.com/wiki/SimCity`, то в базе данных на момент написания книги сформировалось 335 документов и 333 из них загрузилось на локальный диск. Два документа, как оказалось, загружаются с HTTP статусом 404 (не найдено).

3.3. Индексер

Задачей индексера является индексация документов с целью получения информации о том, какие слова содержатся в каких документах. При этом в базе данных будем хранить только основы слов. Назовем их термами. Получение основы слова принято называть стеммингом. Также мы исключаем из рассмотрения предлоги, причастия, междометия, цифры, частицы и т. п., так как они встречаются в большей

части документов и, соответственно, мало помогают выделению отличий между документами. Список этих слов мы называем стоп-словами и храним в базе данных их основы.

В своей работе индексер использует данные, подготовленные другой подсистемой поисковой системы — кроулером.

Для каждого документа, сохраненного в базе данных, необходимо получить текст, сохраненный в файле на локальном диске, разбить текст по пробелам и получить список слов, находящихся в данном документе. Далее для каждого полученного слова необходимо получить основу слова при помощи стемминга. Если полученный терм является одним из стоп-слов, переходим к следующему слову. Если данного терма еще нет в базе данных, сохраняем его. Если нет записи о том, что данный терм находится в данном документе, создаем такую запись и увеличиваем счетчик числа вхождений данного терма в данный документ на единицу.

Добавим следующие функции в файл functions.php [44].

Первая функция — это findTerm. Данная функция находит терм в базе данных. Делается запрос к таблице term по ключевому терму $term и возвращается его результат.

```
function findTerm($term, $dbLink){
    $q = "SELECT * FROM term WHERE term = '".$term."'";
    $res = mysqli_query($dbLink, $q);
    $foundTerm = mysqli_fetch_array($res);
     return $foundTerm;
}
```

Следующая функция isItStopWord проверяет, является ли терм стоп-словом. Стоп-слова ищем по переменной $term, которая подается на вход функции, в таблице стоп-слов stop_word.

```
function isItStopWord($term, $dbLink){
    $q = "SELECT * FROM stop_word WHERE term = '".$term
        ."'";
    $res = mysqli_query($dbLink, $q);
    $foundTerm = mysqli_fetch_array($res);
    return (bool)$foundTerm;
}
```

Далее идет функция clearStr, которая оставляет в строке $str только буквы и пробелы.

```
function clearStr($str){
    $q = "/[^a-zA-Z ]/";
    $str = preg_replace($q, '', $str);
    return $str;
}
```

Теперь перейдем к описанию функций собственно индексера. Создаем файл indexer.php [46] и подключаем к нему файл с подключением к базе данных, функции из functions.php [44], библиотеку PHP Simple HTML DOM Parser и стеммер.

```
include 'db.php';
include 'functions.php';
include 'simple_html_dom.php';
include 'stemmer.php';
set_time_limit(0);
```

Реализуем функцию getArrayWordsFromFile, возвращающую массив слов, содержащихся в документе. По заданному пути $path проверяем существование файла. Извлекаем текст из документа. Вызываем ранее добавленную в файл functions.php [44] функцию clearStr. Возвращаем массив слов документа.

```
function getArrayWordsFromFile($path){
    if (!file_exists($path) || !is_file($path)) {
        return null;
    }

    $html = file_get_html($path);
    $plainText = $html->plaintext;

    $html->clear();
    unset($html);

    $str = clearStr($plainText);
    return explode(" ", $str);
}
```

Далее описываем функцию addTerms для добавления термов в базу данных. В цикле работаем с каждым словом. Переводим слово в нижний регистр и получаем терм стеммированием слова. Если терм является стоп-словом, переходим к следующему слову. Ищем терм в базе данных. Если нет терма в базе данных, добавляем его и записываем информацию о том, что данный терм содержится в данном документе. Иначе, если терм уже есть в базе данных, пытаемся загрузить

информацию о том, что данный терм находится в данном документе. Если терма нет в данном документе, создаем новую запись и указываем, что терм встречается в данном документе один раз на данный момент. Иначе, если уже есть запись о том, что данный терм есть в данном документе, увеличиваем число вхождений терма в данном документе на единицу.

```
function addTerms($page, $words, $dbLink){
    foreach ($words as $word) {
        if (strlen($word) == 0) {
            continue;
        }
        $word = strtolower($word);
        $wordStem = PorterStemmer::Stem($word);

        if (isItStopWord($wordStem, $dbLink)) {
            continue;
        }

        $term = findTerm($wordStem, $dbLink);
        if (!$term) {
            $q = "INSERT INTO term(term) values('".
                $wordStem."')";
            mysqli_query($dbLink, $q);

            $term = findTerm($wordStem, $dbLink);

            $q = "INSERT INTO term_page(term_id, page_id,
                num) value(".$term['id'].",".$page['id
                '].",1)";
            mysqli_query($dbLink, $q);

        } else {
            $q = "SELECT * FROM term_page WHERE term_id=".
                $term['id']." AND page_id=".$page['id'];
            $res = mysqli_query($dbLink, $q);
            $result = mysqli_fetch_array($res);

            if (!$result) {
                $q = "INSERT INTO term_page(term_id,
                    page_id, num) value(".$term['id'].",".
                    $page['id'].",1)";
                mysqli_query($dbLink, $q);
            } else {
```

```
                $num = $result['num'] + 1;
                $q = "UPDATE term_page SET num = ".$num."
                    WHERE term_id=".$term['id']." AND
                    page_id=".$page['id'];
                mysqli_query($dbLink, $q);
            }
        }
    }
}
```

Все необходимые функции перечислены. Теперь перейдем к реализации запуска алгоритма. В первую очередь очищаем таблицы term и term_page. Загружаем данные о всех документах из базы данных. Индексируем каждый документ в цикле. Если значение $page равно null, переходим в начало цикла.

```
mysqli_query($dbLink, 'TRUNCATE TABLE term');
mysqli_query($dbLink, 'TRUNCATE TABLE term_page');

$q = "SELECT * FROM page";
$pagesRes = mysqli_query($dbLink, $q);

while ($page = mysqli_fetch_array($pagesRes)) {
    if ($page === null) {
        continue;
    }
    $filePath = getPath($page['name']);
    $words = getArrayWordsFromFile($filePath);
    if (is_array($words)) {
        addTerms($page, $words, $dbLink);
    }
}
```

Получаем путь до файла документа и разбиваем его текст на слова. Если массив слов не пустой — добавляем термы и информацию об их содержании в данном документе в базу данных. После индексации всех документов, существующих на данный момент мы увидим, что число термов на наших страницах равно 4 389 (за исключением стоп-слов), также имеется 30 790 записей о том, что некоторый терм находится в некотором документе.

3.4. Поисковый интерфейс

Поисковый интерфейс — это веб-интерфейс поисковой системы, с которой непосредственно взаимодействуют ее пользователи. Задачей поискового интерфейса является получение поисковых запросов от пользователей и генерация поисковой выдачи.

Поисковый интерфейс состоит из двух экранов: экрана с пустой формой ввода поискового запроса, экрана с заполненной формой ранее введенным поисковым запросом и с отображением поисковой выдачи.

Форма ввода поискового запроса отправляет GET запрос в поисковую систему, в ответ на который она выдает сгенерированную поисковую выдачу. При отправке GET запроса с поисковым запросом выполняются следующие шаги. Полученный поисковый запрос разбивается по пустым символам на слова. Далее у каждого слова оставляем только основу с помощью стемминга (получаем список термов). Из полученного списка термов исключаем стоп-слова. Находим все документы, в которых встречаются все слова из запроса. Генерируем поисковую выдачу, которая состоит из ссылок на найденные документы.

Создаем файл search.php [47] и подключаем к нему файл с подключением к базе данных и стеммер.

```
<?php
include 'db.php';
include 'stemmer.php';
```

Реализуем функцию deleteStopWords для удаления стоп-слов из массива термов. В аргумент функции передаем массив $arrayString поискового запроса от пользователя. В цикле проходим по каждому слову запроса. Далее к этому слову применяем стеммер, который возвращает основу слова. Для основы слова проверяем, является ли она стоп-словом. Составялем запрос к таблице stop_word, который проверяет, является ли данное слово стоп-словом. Если оно не содержится в таблице stop_word, добавляем его в массив $searchWords. Таким образом, функция возвращает массив $searchWords, не содержащий стоп-слова.

```
function deleteStopWords($arrayStrings, $dbLink){
    $searchWords = array();
    foreach ($arrayStrings as $str) {
        $word = PorterStemmer::Stem($str);
        $query = "SELECT * FROM stop_word WHERE term = '".
            $word."'";
        $result = mysqli_query($dbLink, $query);
        $tmp = mysqli_fetch_array($result);
        if (!$tmp) {
            $searchWords[] = $word;
        }
    }
    return $searchWords;
}
```

Далее реализуем функцию findPageIdsByWords для нахождения уникальных идентификаторов документов по заданному массиву термов $searchWords. В начале пробуем найти документы, содержащие все термы из массива. Если таких документов нет, пробуем найти документы, содержащие хоть один терм из массива.

```
function findPageIdsByWords($searchWords, $dbLink){
    $ids = array();
    $pageIds = array();

    foreach($searchWords as $word) {
        $query = "SELECT * FROM term WHERE term = '".$word
            ."'";
        $result = mysqli_query($dbLink, $query);
        $tmp = mysqli_fetch_array($result);

        if ($tmp !== null) {
            $ids[] = $tmp['id'];
        }
    }

    if (empty($ids)) {
        return null;
    }

    $ids = implode(', ', $ids);
    $query = "SELECT * FROM term_page WHERE term_id IN (".
        $ids.")";
    $result = mysqli_query($dbLink, $query);
```

```
    while ($tmp = mysqli_fetch_array($result)) {
        $pageIds[] = $tmp['page_id'];
    }

    if (empty($pageIds)) {
        return null;
    }

    $countWords = count($searchWords);
    $ids = array_keys(array_filter(array_count_values(
        $pageIds), function ($value) use ($countWords) {
        return $value == $countWords;
    }));

    return !empty($ids) ? $ids : $pageIds;
}
```

Формируем данные для поисковой выдачи с помощью функции getResults. На вход функции подается массив $pagesIds, который содержит идентификаторы документов. По заданному массиву в таблице page находим данные документов. Создаем новый массив $resultLinks, который будет содержать найденные результаты. В цикле проходим по каждому элементу массива $pageIds и находим документ по его идентификатору. Добавляем в $resultLink найденный результат запроса. Возвращаем найденные документы, содержащиеся в массиве $resultLinks. Если после отправки пользователем поискового запроса в GET есть значение q, осуществляем поиск. Из строки запроса $_GET['q'] получаем массив $str, содержащий слова запроса. Удаляем стоп-слова из массива $str, используя функцию deleteStopWords, и записываем данные в массив $search_words. По заданному массиву слов $search_words находим индентификаторы документов, которые содержат эти слова, и используем функцию findPageIdsByWords, чтобы найти документы, содержащие данные слова. Если массив $pageIds, содержащий идентификаторы документов, не пустой, вызываем функцию getResults, которой передаем данный массив, и в дальнейшем выводим полученные данные на экран пользователя.

```
function getResults($pageIds, $dbLink){
    $resultLinks = array();
    for ($i = 0; $i < count($pageIds); $i++) {
        $query = "SELECT * FROM page WHERE id='".$pageIds[
            $i]."'";
```

```
        $res = mysqli_query($dbLink, $query);
        $mas = mysqli_fetch_array($res);
        $resultLinks['page'][] = $mas['url'];
    }
    return $resultLinks;
}

if (isset($_GET['q']) && !empty($_GET['q'])) {
    $str = explode(" ", $_GET['q']);
    $search_words = deleteStopWords($str, $dbLink);
    $pageIds = findPageIdsByWords($search_words,$dbLink);
    $result_links = array();
    if (!empty($pageIds)) {
        $result_links = getResults($pageIds, $dbLink);
    }
}
?>
```

Далее выводим поисковую форму. В этой форме пользователь вводит поисковый запрос. Формируем поисковую выдачу. Выводим результаты поиска на экран. Проверяем существование индекса q в массве $_GET. Если данный индекс есть, то был задан поисковый запрос. Проверяем массив $result_links на наличие результата поиска. Если результаты были получены, выводим на экран данные в виде ссылок на найденные документы. Иначе, если поиск не дал результатов, выводим пользователю соответствующее сообщение.

```
<html>
    <head>
        <title>Search</title>
    </head>
    <body>
        <form method="GET">
            <input type="text" name="q" value="<?=(isset(
                $_GET['q'])) ? $_GET['q'] : ''?>">
            <input type="submit" value="Search">
        </form>
        <?php if (isset($_GET['q'])) : ?>
        <?php if (isset($result_links) && !empty(
            $result_links)): ?>
            Results: <?=count($result_links['page'])?>
            <ol>
            <?php for ($i = 0;$i < count($result_links['
                page']);$i++): ?>
```

```
                <li>
                    <a href="<?= $result_links['page'][$i]
                        ?>"> <?= $result_links['page'][$i
                        ] ?>
                </li>
            <?php endfor; ?>
            </ol>
        <?php else: ?>
            <p>No results</p>
        <?php endif; ?>
        <?php endif; ?>
    </body>
</html>
```

На данный момент поисковая система выводит форму для введения поискового запроса, при заполнении которой осуществляется поиск по ранее проиндексированным документам. Результатом поиска является список ссылок на проиндексированные документы.

3.5. Ранжирование

На данный момент наша поисковая система может генерировать поисковую выдачу, в которой найденные документы располагаются в совершенно непредсказуемом порядке. Возникает задача сортировки документов поисковой выдачи в порядке убывания их значимости для пользователя, который ищет информацию. Сортировка поисковой выдачи называется ранжированием.

Реализуем два вида алгоритмов ранжирования: алгоритмы ссылочного ранжирования и алгоритмы ранжирования на основе мер TF и IDF.

Алгоритмы ссылочного ранжирования рассчитывают ранги документов на основе ориентированного графа, где вершины — это документы, а ребра — это ссылки. Начальной вершиной ребра является документ, в котором находится ссылка, а конечной вершиной является документ, на который указывается ссылка. Мы рассмотрим два алгоритма ссылочного ранжирования — HITS и PageRank. Алгоритмы ссылочного ранжирования рассчитывают одно или несколько чисел для каждого документа. Данные числа называются рангами и рассчитываются на основе ориентированного графа, где вершины — это документы, а ребра — это ссылки. Начальной вершиной ребра является документ, в котором находится ссылка, а конечной вершиной

является документ, на который указывается ссылка. Мы рассмотрим два алгоритма ссылочного ранжирования — HITS и PageRank.

Второй тип алгоритмов ранжирования использует меры TF и IDF для сравнения схожести документа и поискового запроса. TF — это мера важности терма для одного документа. IDF — это мера важности терма для всей коллекции документов. Для данного типа ранжирования рассмотрим два алгоритма. Первый алгоритм параметризует запросы и документы на основе термов, содержащихся в них, создавая векторы документов и запросов. Релевантность документа поисковому запросу определяется косинусом угла между вектором запроса и вектором документа — чем больше косинус угла, тем лучше документ подходит под данный поисковый запрос. Второй алгоритм — это Okapi BM25, представляющий собой формулу, содержащую меры TF и IDF.

3.6. Алгоритм ранжирования HITS

Рассмотрим алгоритм ранжирования HITS (англ. Hyperlink Induced Topic Search), который был предложен Джоном Клейнбергом в 1998 году. Идея алгоритма заключается в том, что документ имеет две роли: документ «автор» и документ «посредник».

Документ, на который ссылаются многие другие документы, должен быть хорошим «автором». В свою очередь документ, который указывает на многие другие, должен быть хорошим «посредником». При этом чем больше хороших «посредников» ссылаются на документ, тем лучшим «автором» он является, и наоборот, если документ содержит в себе ссылки на хороших «авторов», то он является хорошим «посредником». Основываясь на этом предположении, в алгоритме HITS для каждого документа итеративно рассчитываются две оценки: оценка авторитетности и посредническая оценка.

Оценку авторитетности будем далее называть авторитетностью документа, а посредническую — хабностью документа.

Рассмотрим два типа обновления: правило обновления авторитетности и хаб-обновление. Правило обновления авторитетности: для любого документа p имеем

$$auth(p) = \sum_{i=1}^{n} hub(i)$$

где n — общее количество документов, связанных с p, i — документ, связанный с p.

Следовательно, оценка авторитетности документа вычисляется как сумма значений оценок посреднических документов, которые указывают на этот документ.

Правило хаб-обновления: для любого документа p имеем

$$hub(p) = \sum_{i=1}^{n} auth(i)$$

где n — общее количество документов, на которые указывает p, i — документ, на который указывает p. Следовательно, хабность документа вычисляется как сумма значений оценок авторитетности документов, на которых он ссылается.

Рассмотрим, как вычисляются авторитетность и хабность документа. В начале ранжирования авторитетность и хабность каждого документа примем за 1. Затем выполняются правила обновления авторитетности и хаб-обновления. Далее происходит нормализация значений путем деления каждой хабности на квадратный корень из суммы квадратов всех хабностей, и деления каждой оценки авторитетности на квадратный корень из суммы квадратов всех оценок авторитетности. Если необходимо, повторяем шаги, начиная с правил обновления авторитетности и хаб-обновленя до тех пор, пока модуль разницы между предыдущим и текущим значением авторитетности и хабности каждого документа не будет меньше определенного числа, которое мы назовем $epsilon$. Чем меньше число $epsilon$, тем больше итераций понадобится для вычисления авторитетности и хабности.

Далее рассмотрим реализацию алгоритма HITS. Создадим файл hits.php [48] и подключим к нему файл с подключением к базе данных. Определим константу $epsilon$.

```
include "db.php";
set_time_limit(0);
const EPSILON = 0.001;
```

Реализуем функцию getCountPages для подсчета количества документов. Составляем запрос к базе данных, который подсчитывает количество документов и возвращает найденный результат.

```
function getCountPages($dbLink){
    $q = "SELECT COUNT(*) FROM page";
    $res = mysqli_query($dbLink, $q);
    $tmp = mysqli_fetch_array($res);
    return $tmp[0];
}
```

Далее реализуем функцию calculateAuth для подсчета авторитетности документов. В этой функции выполняется правило обновления авторитетности, которое описано выше. В качестве аргументов функции передаем ссылку на массив, содержащий авторитетность документов, в который будут записываться результаты подсчета правила обновления авторитетности. Кроме того, вторым аргументом функция получает текущую хабность документов. В цикле проходим по каждому документу и составляем запрос к таблице link, который возвращает документы, содержащие ссылки на текущий документ. Если данный запрос вернул не пустой результат, авторитетность документа вычисляется как сумма значений хабности документов, которые ссылаются на текущий документ. Иначе авторитетность документа приравниваем значению 0.

```
function calculateAuth(&$curAuth, $curHub, $dbLink){
    for ($i = 1; $i <= count($curAuth); $i++) {
        $pageIds = array();
        $sum = 0;

        $q = "SELECT * FROM link WHERE page_to = ".$i;
        $res = mysqli_query($dbLink, $q);
        $tmp = mysqli_fetch_array($res);

        if ($tmp !== null) {
            $pageIds[] = $tmp['page_from'];

            while ($tmp = mysqli_fetch_array($res)) {
                $pageIds[] = $tmp['page_from'];
            }

            for ($j = 0; $j < count($pageIds); $j++) {
                $sum += $curHub[$pageIds[$j]];
            }
            $curAuth[$i] = $sum;
        } else {
```

```
            $curAuth[$i] = 0;
        }
    }
}
```

Реализуем функцию calculateHub для подсчета хабности документов. В этой функции подсчитываем правило хаб-обновления, которое описано выше. В качестве аргументов функции передаем ссылку на массив, содержащий хабность документов, в который будут записываться результаты подсчета правила хаб-обновления. Кроме того, вторым аргументом функция получает текущую авторитетность документов. В цикле пробегаем по каждому документу и составляем запрос к таблице link, который возвращает документы, на которые ссылается текущий документ. Если данный запрос вернул не пустой результат, хабность документа вычисляется как сумма значений авторитетности документов, на которые ссылается текущий документ. Иначе хабность документа приравниваем значению 0.

```
function calculateHub(&$curHub, $curAuth, $dbLink){
    for ($i = 1; $i <= count($curHub); $i++) {
        $pageIds = array();
        $sum = 0;

        $q = "SELECT * FROM link WHERE page_from = ".$i;
        $res = mysqli_query($dbLink, $q);
        $tmp = mysqli_fetch_array($res);

        if ($tmp !== null) {
            $pageIds[] = $tmp['page_to'];

            while ($tmp = mysqli_fetch_array($res)) {
                $pageIds[] = $tmp['page_to'];
            }

            for ($j = 0; $j < count($pageIds); $j++) {
                $sum += $curAuth[$pageIds[$j]];
            }
            $curHub[$i] = $sum;
        } else {
            $curHub[$i] = 0;
        }
    }
}
```

Реализуем функцию conditionOfExit для проверки условия оста-

новки алгоритма. В качестве аргументов функции передаем текущие и предыдущие значения авторитетности и хабности документов. В теле функции опишем цикл, который пробегает по текущему и предыдущему значению авторитетности каждого документа. Если хотя бы для одного документа оказалось, что модуль разницы между текущим и предыдущим значением авторитетности больше *epsilon*, итерации по нахождению значений авторитетности и хабности необходимо продолжать. В этом случае функция возвращает значение false. Далее цикл, который также пробегает по текущему и предыдущему значению хабности документа. Если хотя бы для одного документа оказалось, что модуль разницы между текущим и предыдущим значением хабности больше *epsilon*, итерации по нахождению значений авторитетности и хабности необходимо продолжать. В этом случае функция возвращает значение false. Если не оказалось документов, у которых предыдущее и текущее значение авторитетности или хабности отличается более, чем на *epsilon*, функция возвращает значение true.

```
function conditionOfExit($curAuth, $prevAuth, $curHub,
    $prevHub){
    for ($i = 1; $i <= count($curAuth); $i++) {
        if (abs($curAuth[$i] - $prevAuth[$i]) > EPSILON) {
            return false;
        }
    }
    for ($i = 1; $i <= count($curHub); $i++) {
        if (abs($curHub[$i] - $prevHub[$i]) > EPSILON) {
            return false;
        }
    }
    return true;
}
```

Функция normalize используется для нормализации вектора. В аргумент функции передается вектор, который необходимо нормализовать. Для нормализации вектора необходимо каждую его компоненту поделить на длину вектора. Длина вектора равна квадратному корню из суммы квадратов компонентов.

```
function normalize($arr){
    $sum = 0;
    for ($i = 1; $i <= count($arr); $i++) {
        $sum += ($arr[$i] * $arr[$i]);
```

```
    }
    $sum = sqrt($sum);
    for ($i = 1; $i <= count($arr); $i++) {
        $arr[$i] *= (1 / $sum);
    }
    return $arr;
}
```

Далее производим начальную подготовку для запуска алгоритма: получаем количество документов, используя функцию getCountPages($dblink), и устанавливаем начальную авторитетность и хабность каждого документа в значение 1. В переменной $iterations будем хранить число итераций алгоритма. Выполняем подсчет авторитетности. Выполняем подсчет хабности. Нормализуем вектора авторитетности и хабности. Если выполняется условие остановки, выходим из цикла. Если условие остановки алгоритма не выполнилось, то в цикле для каждого предыдущего значения авторитетности и хабности приравнивается текущее значение авторитетности и хабности соответственно. Иначе цикл прерывается. Увеличиваем количество итераций на единицу. Записываем для каждого документа результат расчета его авторитетности и хабности.

```
$countPages = getCountPages($dbLink);

for ($i = 1; $i <= $countPages; $i++){
    $curAuth[$i] = 1;
    $curHub[$i] = 1;
    $prevAuth[$i] = 1;
    $prevHub[$i] = 1;
}

$iterations = 1;
while (true) {
    calculateAuth($curAuth, $curHub, $dbLink);
    calculateHub($curHub, $curAuth, $dbLink);
    $curAuth = normalize($curAuth);
    $curHub = normalize($curHub);
    if (!conditionOfExit($curAuth, $prevAuth, $curHub,
        $prevHub)) {
        for ($i = 1; $i <= count($curAuth); $i++) {
            $prevAuth[$i] = $curAuth[$i];
            $prevHub[$i] = $curHub[$i];
        }
```

```
    } else {
        break;
    }
    $iterations += 1;
}

for ($i = 1; $i <= count($curAuth); $i++) {
    $q = "UPDATE page SET h_auth = ".$curAuth[$i]." WHERE
        id = ".$i;
    mysqli_query($dbLink, $q);

    $q = "UPDATE page SET h_hub = ".$curHub[$i]." WHERE id
         = ".$i;
    mysqli_query($dbLink, $q);
}

echo $iterations;
```

После остановки алгоритма можно увидеть, что для выбранного значения *epsilon* и для ранее проиндексированных документов производится шесть итераций обновления авторитетности и хабности.

3.7. Алгоритм ранжирования PageRank

PageRank — один из алгоритмов ссылочного ранжирования, разработанный Ларри Пейджем и Сергеем Брином в 1998 году. PageRank итеративно рассчитывает числовое значение, которое характеризует «важность» документов. Рассмотрим формулу подсчета PageRank на определенном шаге.

$$PR(A) = \frac{1-d}{N} - d(\sum_{i=1}^{n} \frac{PR(B_i)}{N(B_i)})$$

где $PR(A)$ — PageRank документа A, d — коэффициент затухания, который означает вероятность того, что пользователь, зашедший в документ, перейдет по одной из ссылок, содержащейся в этом документе, а не закроет браузер (обычно его принимают равным 0.85), n — количество документов, ссылающихся на документ , N — количество всех документов, B_i — i-документ, который ссылается на документ A, $N(B_i)$ — количество исходящих ссылок из документа B_i.

Как видно из формулы, каждый документ отдает равную долю своей важности документам, на которых он ссылается. Рассмотрим,

как итеративно рассчитывается PageRank. Вначале каждому документу приравнивается значение PageRank, равное $1/N$. Далее на каждом шаге для каждого документа производим вычисление приведенной выше формулы. Вычисления заканчиваются, как только для каждого документа модуль разницы между текущим и предыдущим значением PageRank становится меньше заранее определенного числа $epsilon$. Чем меньше число $epsilon$, тем точнее будут значения PageRank каждого документа.

Далее рассмотрим реализацию расчета значений PageRank для ранее проиндексированных документов. Создадим файл pagerank.php [49] и включим в него файл с подключением к базе данных. Установим коэффициент затухания и значение $epsilon$.

```
include 'db.php';
set_time_limit(0);
const DAMPING_FACTOR = 0.85;
const EPSILON = 0.001;
```

Реализуем функцию conditionOfExit для проверки условия остановки алгоритма. В качестве аргументов функции передаем текущие и предыдущие значения PageRank всех документов. В теле функции описываем цикл, который для каждого документа высчитывает модуль разницы между текущим и предыдущим значением PageRank. Если хотя бы для одного документа оказалось, что модуль разницы между предыдущим и текущим значением PageRank больше $epsilon$, итерации по расчету PageRank необходимо продолжать. В этом случае функция возвращает значение false. Если не оказалось документов, у которых предыдущее и текущее значение PageRank отличается более, чем на $epsilon$, функция возвращает значение true.

```
function conditionOfExit($cur, $prev){
    for ($i = 1; $i <= count($cur); $i++) {
        if (abs($cur[$i]-$prev[$i]) > EPSILON){
            return false;
        }
    }
    return true;
}
```

Далее реализуем функцию getCountPages для подсчета количества документов. Составляется запрос к базе данных, который подсчитывает количество документов. Полученное значение является результатом

работы функции.

```
function getCountPages($dbLink){
    $q = "SELECT COUNT(*) FROM page";
    $res = mysqli_query($dbLink, $q);
    $tmp = mysqli_fetch_array($res);
    return $tmp[0];
}
```

Реализуем функцию getCountLinkOnPage для подсчета числа исходящих ссылок для всех документов. Все одинаковые ссылки в документе считаются за одну. В этой функции составляется запрос к таблице link, который для каждого документа подсчитывает количество исходящих ссылок. Результат работы функции возвращаются в виде ассоциативного массива, в котором ключи — это номера документов, а значения - это количество исходящих ссылок.

```
function getCountLinkOnPage($countPages, $dbLink){
    $countLinkOnPage = array();
    for ($i = 1; $i <= $countPages; $i++) {
        $q = "SELECT COUNT(*) FROM link WHERE page_from =
            ".$i;
        $res = mysqli_query($dbLink, $q);
        $tmp = mysqli_fetch_array($res);
        $countLinkOnPage[$i] = $tmp[0];

        if ($countLinkOnPage[$i] == 0) {
            $countLinkOnPage[$i] = 1;
        }
    }
    return $countLinkOnPage;
}
```

Затем делаем начальные приготовления для запуска итеративного алгоритма по подсчету значений PageRank. Подсчитываем общее число документов и создаем переменную $iteration для подсчета числа итераций алгоритма. Подсчитываем число уникальных ссылок в документах. Задаем начальное значение PageRank для каждого документа. Создаем переменную $prob, в которой будем хранить значение $(1 - d)/N$ формулы подсчета PageRank. В цикле считаем PageRank, пока не выполнится условие остановки. Выше в запросе к базе данных получаем массив идентификаторов документов, ссылающихся на i-ый документ, и записываем полученные данные в массив $dbLinkToPage. Далее производится вычисление PageRank для каждого документа по

формуле, которая была описана ранее. Полученное значение записываем в массив $curRank, который хранит PageRank для каждого документа. Проверяем условие остановки алгоритма. Если условие остановки алгоритма не выполнилось, в цикле для каждого документа записываем текущее значение PageRank вместо предыдущего. Увеличиваем число итераций на единицу и продолжаем работу цикла. Иначе, если условие остановки алгоритма выполнено, прерываем цикл и заканчиваем подсчет значений PageRank проиндексированных документов. Записываем посчитанные значения PageRank для каждого документа в базу данных и выводим число произведенных итераций на экран пользователя.

```
$iteration = 1;
$countPages = getCountPages($dbLink);
$countLinkOnPage = getCountLinkOnPage($countPages, $dbLink
    );

$startWeight = 1 / $countPages;

for ($i = 1; $i <= $countPages; $i++) {
    $curRank[$i] = $startWeight;
    $prevRank[$i] = $startWeight;
}

$prob = (1 - DAMPING_FACTOR) / $countPages;

while (true){
    for ($i = 1; $i <= $countPages; $i++) {

        $q = "SELECT page_from FROM link WHERE page_to =
            ".$i;
        $res = mysqli_query($dbLink, $q);
        $k = 0;

        while ($tmp = mysqli_fetch_array($res)) {
            $dbLinkToPage[$k++] = $tmp['page_from'];
        }
        $sum = 0;
        for ($j = 0; $j < $k; $j++) {
            $sum += ($curRank[$dbLinkToPage[$j]] /
                $countLinkOnPage[$dbLinkToPage[$j]]);
        }
        $curRank[$i] = $sum;
```

```
            $curRank[$i] *= DAMPING_FACTOR;
            $curRank[$i] += $prob;
        }
        if (!conditionOfExit($curRank, $prevRank)) {
            for ($i = 1; $i <= count($curRank); $i++) {
                $prevRank[$i] = $curRank[$i];
            }
            ++$iteration;
        } else {
            break;
        }
}

for ($i = 1; $i <= count($curRank); $i++) {
    $q = "UPDATE page SET p_rank = ".$curRank[$i]." WHERE
        id = ".$i;
    mysqli_query($dbLink, $q);
}
echo $iteration;
```

После того, как алгоритм выполнился, можно увидеть, что для выбранных значений коэффициента затухания и $epsilon$ производится семь итераций.

3.8. Подсчет мер TF и IDF

TF (от англ. term frequency — частота слова) — это отношение числа вхождения терма к количеству всех термов документа. То есть чем чаще входит терм в документ, тем важнее он для документа.

$$tf = \frac{n_i}{\sum_k n_k}$$

где n_i — число вхождений терма в документе, n_k — общее число термов в документе.

IDF (от англ. inverse document frequency — обратная документная частота) — инверсия частоты, с которой терм встречается во всех документах. Отражает следующий факт: если терм встречается практически в каждом документе, то он менее важен для поиска.

$$idf = \log \frac{N}{df_i}$$

где N — количество всех документов, df_i — количество документов, в которых встречается терм f_i.

Мера TF-IDF является произведением этих двух частот. Т. е. важный терм (с большим весом TF-IDF) — такой, который часто встречается в данном документе и редко в остальных.

$$tfidf = tf \cdot idf$$

Рассмотрим реализацию расчета мер TF и IDF в нашей поисковой системе. Создадим файл tf-idf.php [50] файл с подключением к базе данных.

```
include 'db.php';

set_time_limit(0);
```

Реализуем функцию getCountPages для подсчета количества документов. Составляем запрос к базе данных, который подсчитывает количество документов и возвращает найденный результат.

```
function getCountPages($dbLink){
    $q = "SELECT COUNT(*) FROM page";
    $res = mysqli_query($dbLink, $q);
    $tmp = mysqli_fetch_array($res);
    return $tmp[0];
}
```

Затем идет реализация функции getCountTermPages, используемой для подсчета числа записей в таблице term_page. Составляем запрос к таблице и возвращаем количество записей в таблице.

```
function getCountTermPages($dbLink){
    $q = "SELECT COUNT(*) FROM term_page";
    $res = mysqli_query($dbLink, $q);
    $tmp = mysqli_fetch_array($res);
    return $tmp[0];
}
```

Функция getCountWordsOnPage используется для подсчета числа термов в документе. В качестве аргумента функции передается идентификатор документа, для которого в таблице term_page подсчитывается количество термов в документе и возвращается найденный результат.

```
function getCountWordsOnPage($pageId, $dbLink){
    $q = "SELECT SUM(num) FROM term_page WHERE page_id =
        ".$pageId;
    $res = mysqli_query($dbLink, $q);
    $mas = mysqli_fetch_array($res);
    return $mas[0];
}
```

Далее реализована функция getCountPagesOnWord для подсчета числа документов, в которые входит терм. В качестве аргумента функции передается идентификатор терма. По этому идентификатору в таблице term_page находятся все документы, которые содержат соответствующий терм. Возвращается количество найденных документов.

```
function getCountPagesOnWord($wordId, $dbLink){
    $q = "SELECT COUNT(*) FROM term_page WHERE term_id =
        ".$wordId;
    $res = mysqli_query($dbLink, $q);
    $mas = mysqli_fetch_array($res);
    return $mas[0];
}
```

Затем идет расчет мер TF и IDF в цикле. Получаем начальные данные — количество документов $countPages, используя функцию getCountPages($dbLink), и число записей $countTermPages, используя функцию getCountTermPages($dbLink). В цикле для каждой записи из таблицы term_page будем подсчитывать меру TF-IDF. Рассчитываем меру TF по описанной ранее формуле. Получаем число вхождений терма в документ. Получаем общее число термов в документе. Вычисляем значение меры TF. Рассчитываем меру IDF по описанной ранее формуле. Получаем количество документов, в которых встречается терм. Вычисляем значение меры IDF.

$$idf = \log \frac{countPage}{countPagesOnWord}$$

Мера TF-IDF терма в документе является произведением мер TF и IDF.

$$tf\_idf = tf \cdot idf$$

Записываем результат в базу данных. Полученные значения мер TF и TF-IDF записываем в таблицу term_page. Полученное значение меры IDF запишем в таблицу term.

```
$countPages = getCountPages($dbLink);
$countTermPages = getCountTermPages($dbLink);

$query = "SELECT * FROM term_page";
$res = mysqli_query($dbLink, $query);

while ($termPage = mysqli_fetch_array($res)) {
    $frequencyInCurPage = $termPage['num'];
    $countWordsInCurPage = getCountWordsOnPage($termPage['
        page_id'], $dbLink);

    $tf = $frequencyInCurPage / $countWordsInCurPage;

    $countPagesOnWord = getCountPagesOnWord($termPage['
        term_id'], $dbLink);

    $idf = log($countPages / $countPagesOnWord);

    $tf_idf = $tf * $idf;

    $q = "UPDATE term_page SET tf_idf = ".$tf_idf.", tf =
        ".$tf." WHERE term_id = ".$termPage['term_id']."
        AND page_id = ".$termPage['page_id'];
    mysqli_query($dbLink, $q);

    $q = "UPDATE term SET idf = '".$idf."' WHERE id = ".
        $termPage['term_id']." AND idf = 0";
    mysqli_query($dbLink, $q);
}
```

3.9. Алгоритм ранжирования на основе подсчета косинусов углов между векторами

Мера TF-IDF показывает, насколько важным является терм в содержащем его документе и во всей коллекции документов. С данной мерой удобно работать, так как она представляет собой вещественное число. Для того, чтобы использовать меру TF-IDF для ранжирования, необходимо произвести параметризацию документов и поисковых запросов на ее основе. Удобным для работы является метод параметризации документов и запросов, представляющий собой составление векторов из мер TF-IDF термов, содержащихся в документе.

Рассмотрим метод составления векторов документов и векторов запросов. Размерность каждого вектора будет равна количеству термов

в коллекции документов. Для начала рассмотрим составление вектора документа:

$$\vec{d_j} = (w_{1j}, w_{2j}, \ldots, w_{nj})$$

где $\vec{d_j}$ — векторное представление j-го документа, w_{ij} — вес i-го терма в j-м документе (если i-й терм не содержится в j-м документе, значение w_{ij} берем равным значению 0), n — общее количество различных термов во всех документах коллекции.

Далее рассматривается составление вектора поискового запроса:

$$\vec{q} = (r_1, r_2, \ldots, r_n)$$

где $\vec{q}$ — векторное представление запроса, r_i — булев вес i-го терма (равен 1, если он встречается в запросе, и 0 в противном случае), n — общее количество различных термов во всех документах коллекции.

Получив вектора запроса и документа, можно вычислить меру сходства между двумя векторами с помощью измерения косинуса угла между ними, который будет показывать, насколько запрос и документ похожи. Косинус угла между вектором документа $\vec{d_j}$ и вектором запроса $\vec{q}$ рассчитывается по формуле:

$$cos(\vec{d_j}, \vec{q}) = \frac{\vec{d_j} \cdot \vec{q}}{\| \vec{d_j} \| \cdot \| \vec{q} \|}$$

где $\| \vec{d_j} \|$ и $\| \vec{q} \|$ — нормы векторов $\vec{d_j}$ и $\vec{q}$ соответственно, $\vec{d_j} \cdot \vec{q}$ — скалярное произведение векторов $\vec{d_j}$ и $\vec{q}$.

Отсюда можно сделать вывод, что чем меньше угол между векторами документа и запроса, тем больше косинус угла, следовательно, поисковый запрос и документ более похожи.

Алгоритм будет реализован в разделе «Применение ранжирования в поисковой выдаче».

3.10. Алгоритм ранжирования Okapi BM25

Okapi BM25 — это алгоритм ранжирования на основе расчета одноименной формулы, в которой используются меры TF и IDF.

Рассмотрим формулу алгоритма ранжирования Okapi BM25:

$$score(D, Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgl})}$$

где Q — поисковый запрос,
D — документ из коллекции,
q_i — термы (слова), которые содержатся в запросе Q,
$f(q_i, D)$ — частота терма q_i, т. е. мера TF терма q_i в документе D,
$IDF(q_i)$ — обратная документная частота терма q_i, т. е. мера IDF терма q_i ,
$| D |$ — количество всех термов в документе D (или длина документа),
$avgl$ — средняя длина документа в коллекции,
k_1 и b — свободные коэффициенты, где $k_1 = 1.5$, $b = 0.75$.

Okapi BM25 — это одна формула из целого семейства формул. Варьируя значения k_1 и b, можно получить различные формулы из данного семейства, и их результаты ранжирования будут немного отличаться. Подробное рассмотрение этого семейства формул выходит за рамки данного раздела.

Алгоритм ранжирования Okapi BM25 будет реализован в разделе «Применение ранжирования в поисковой выдаче».

3.11. Применение ранжирования в поисковой выдаче

В разделе «Поисковый интерфейс» было рассмотрено подробное описание поискового интерфейса нашей поисковой системы. В этом разделе будут применены и реализованы все типы алгоритмов ранжирования и выдача их результатов пользователю в поисковой выдаче. К нашему поисковому интерфейсу добавится возможность выбора из числа алгоритмов ранжирования, которые были описаны ранее: ссылочные алгоритмы ранжирования HITS и PageRank, алгоритмы ранжирования на основе мер TF-IDF — алгоритм на основе расчета косинуса угла между векторами и алгоритм Okapi BM25.

Создадим файл searchrank.php [51], к которому подключим файл с подключением к базе данных и стеммер. Определяем параметры k_1 и b для алгоритма Okapi BM25.

```
<?php
include 'db.php';
include 'stemmer.php';
define('BM25_K1', 1.5);
define('BM25_B', 0.75);
```

Вначале опишем функцию length для подсчета длины вектора.

На входе функция получает вектор-массив, длину которого нужно вычислить. Длина вектора равна квадратному корню из суммы квадратов компонентов. Пусть дан вектора $\vec{a}$, тогда длина вектора: $\mid \vec{a} \mid = \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2}$

```
function length($a) {
    $sum = 0;
    for ($i = 1; $i <= count($a); $i++) {
        $sum += ($a[$i] * $a[$i]);
    }
    $sum = sqrt($sum);
    return $sum;
}
```

Далее опишем функцию product для подсчета скалярного произведения векторов. На входе функция получает два вектор-массива и возвращает скалярное произведение векторов. Для подсчета скалярного произведения векторов воспользуемся следующей формулой: $\vec{a} \cdot \vec{b} = a_1 \cdot b_1 + a_2 \cdot b_2 + \ldots + a_n \cdot b_n$, где $\vec{a}$ и $\vec{b}$ — векторы.

```
function product($a, $b) {
    $sum = 0;
    for ($i = 1; $i <= count($a); $i++) {
        $sum += ($a[$i] * $b[$i]);
    }
    return $sum;
}
```

Функция buildVectorWords используется для построения вектора поискового запроса. Как было сказано ранее, $\vec{q} = (r_1, r_2, \ldots, r_n)$ где $\vec{q}$ — векторное представление запроса, r_i — булев вес i-го терма (равен 1, если он встречается в запросе, и 0 в противном случае), n — общее количество различных термов во всех документах коллекции. Получаем количество всех термов в коллекции и сохраняем его в переменную $countWords. Инициализируем начальный вектор-массив запроса и в цикле приравниваем значение 0. В цикле для каждого поискового терма в массиве $searchWords строим вектор запроса $vectorWords. Значение компоненты вектора будет равно 1, если терм встречается в нашей коллекции термов. После всех действий функция возвращает вектор-массив запроса.

```
function buildVectorWords($searchWords, $dbLink){
    $query = "SELECT COUNT(*) FROM term";
    $res = mysqli_query($dbLink, $query);
    $tmp = mysqli_fetch_array($res);

    $countWords = $tmp[0];
    $vectorWords = array();

    for ($i = 1; $i <= $countWords; $i++) {
        $vectorWords[$i] = 0;
    }

    foreach ($searchWords as $word) {
        $query = "SELECT * FROM term WHERE term = '".$word
            ."'";
        $res = mysqli_query($dbLink, $query);
        $tmp = mysqli_fetch_array($res);

        $vectorWords[$tmp['id']] = 1;
    }
    return $vectorWords;
}
```

Подсчет числа термов в базе данных осуществляется с помощью функции getCountTerms. Составляем запрос к таблице term и возвращаем результат в виде количества найденных термов.

```
function getCountTerms($dbLink){
    $q = "SELECT COUNT(*) FROM term";
    $res = mysqli_query($dbLink, $q);
    $tmp = mysqli_fetch_array($res);
    return $tmp[0];
}
```

Построение векторов для страниц реализовано в функции buildVectorPages. Как было сказано ранее, $\vec{d_j} = (w_{1j}, w_{2j}, \ldots, w_{nj})$ где $\vec{d_j}$ — векторное представление j-го документа, w_{ij} — вес i-го терма в j-м документе (если i-й терм не содержится в j-м документе, значение w_{ij} берем равным значению 0), n — общее количество различных термов во всех документах коллекции. В качества аргумента функции передается массив $pageIds, содержащий идентификаторы документов, которые содержат термы из поискового запроса. В результате работы функции возвращается массив векторов документов. Инициализируем массив $vectorPages, в который далее запишем векторы

документов. Получаем число термов в коллекции и записываем в переменную $termsCount. В цикле пробегаем по каждому документу массива $pageIds. Заполняем вектор-массив $vectorPages значениями 0 и устанавливаем размерность вектора равным $termsCount. По идентификаторам страниц из массива $pageIds составляем запрос к таблице term_page, который возвращает нам необходимые данные (идентификаторы термов, идентификаторы страниц и их меры TF-IDF). В цикле для каждой записи формируем $vectorPages, который представляет собой двумерный массив идентификаторов документов и идентификаторов термов. Значением двумерного массива $vectorPages будут веса термов в документах. Возвращаем заполненный двумерный массив $vectorPages.

```
function buildVectorPages($pageIds, $dbLink){
    $vectorPages = array();
    $termsCount = getCountTerms($dbLink);
    foreach ($pageIds as $id) {
        $vectorPages[$id] = array_fill(1, $termsCount, 0);
    }

    $query = "SELECT * FROM term_page WHERE page_id IN
        (".implode(', ', $pageIds).")";
    $res = mysqli_query($dbLink, $query);

    while ($tmp = mysqli_fetch_array($res)) {
        $vectorPages[$tmp['page_id']][$tmp['term_id']] =
            $tmp['tf_idf'];
    }
    return $vectorPages;
}
```

Функция tf_idf содержит подсчет ранга документа по поисковому запросу на основе алгоритма ранжирования по косинусам углов между векторами документов и запросов. В качестве аргументов функции передаются массив $searchWords термов поискового запроса и массив $pageIds идентификаторов документов, которые содержат термы из поискового запроса.

```
function tf_idf($pageIds, $searchWords, $dbLink){
    $vectorWords = buildVectorWords($searchWords, $dbLink)
        ;
    $vectorPages = buildVectorPages($pageIds, $dbLink);
```

```
    $tf_idf = array();
    foreach ($vectorPages as $key => $vectorPage) {
        $tf_idf[$key] = (product($vectorPage, $vectorWords
            )) / (length($vectorPage) * length(
            $vectorWords));
    }

    foreach ($tf_idf as $key => $value) {
        $t_rank['rank'][] = $value;
        $t_rank['id'][] = $key;
    }

    for ($i = 1; $i < count($t_rank['rank']); $i++) {
        $key = $t_rank['rank'][$i];
        $key_2 = $t_rank['id'][$i];
        $j = $i - 1;
        while (($j >= 0) && ($t_rank['rank'][$j] < $key))
            {
            $t_rank['rank'][$j + 1] = $t_rank['rank'][$j];
            $t_rank['id'][$j + 1] = $t_rank['id'][$j];
            --$j;
        }
        $t_rank['rank'][$j + 1] = $key;
        $t_rank['id'][$j + 1] = $key_2;
    }
    return $t_rank;
}
```

Создаем вектор запроса и векторы документов с помощью вызова ранее определенных функций. Далее инициализируем массив $tf_idf, который будет хранить значение косинуса угла между векторами запроса и документов. В цикле считаем косинус угла для вектора запроса и векторов документов. Создаем массив для перевода результатов расчета в более удобный вид, с дальнейшей его сортировкой по убыванию ранга.

Расчет рангов документов по поисковому запросу на основе алгоритма ранжирования Okapi BM25 реализован в функции bm25. В качестве аргумента функции передаем массив $searchWords термов поискового запроса и массив $pageIds идентификаторов документов, которые содержат термы из поискового запроса. Подсчитываем количество всех документов в таблице page. Составляем запрос в базе данных и полученные результаты записываем в переменную $countPages. Подсчитываем количество всех записей в таблице term_page. Состав-

ляем запрос в базе данных и полученные результаты записываем в переменную $countTermPage. Вычисляем среднюю длину документа в коллекции. Далее из таблицы term производим выборку термов поискового запроса. По результатам данного запроса идентификаторы термов записываем в массив $termsIds, а в массив $termsIDFs записываем метрики IDF соответствующих термов. Составляем запрос к таблице term_page. После получения данных применяется формула алгоритма Okapi BM25, результат которой записываем в массив $pageRank, состоящий из ключа (идентификатора документа) и значения формулы Okapi BM25. Создаем массив для перевода результатов расчета в более удобный вид, с дальнейшей его сортировкой по убыванию ранга.

```
function bm25($pageIds, $searchWords, $dbLink){
    $q = "SELECT COUNT(*) FROM page";
    $res = mysqli_query($dbLink, $q);
    $tmp = mysqli_fetch_array($res);
    $countPages = $tmp[0];

    $q = "SELECT COUNT(*) FROM term_page";
    $res = mysqli_query($dbLink, $q);
    $tmp = mysqli_fetch_array($res);
    $countTermPage = $tmp[0];
    $avgdl = $countTermPage / $countPages;

    $termsIds = array();
    $termsIDFs = array();
    for ($i = 0; $i < count($searchWords); $i += 1) {
        $searchWords[$i] = "'".$searchWords[$i]."'";
    }
    $query = "SELECT * FROM term WHERE term IN  (".implode
        (', ', $searchWords).")";
    $res = mysqli_query($dbLink, $query);
    while ($tmp = mysqli_fetch_array($res)) {
        $termsIds[] = $tmp['id'];
        $termsIDFs[$tmp['id']] = $tmp['idf'];
    }
    $pageRank = array();
    $query = "SELECT * FROM term_page WHERE page_id IN (".
        implode(', ', $pageIds).") AND term_id IN (".
        implode(', ', $termsIds).")";
    $res = mysqli_query($dbLink, $query);
```

```
    while ($tmp = mysqli_fetch_array($res)) {
        if (!isset($pageRank[$tmp['page_id']])) {
            $pageRank[$tmp['page_id']] = 0;
        }
        $pageRank[$tmp['page_id']] += $termsIDFs[$tmp['
            term_id']] * ($tmp['tf'] * (BM25_K1 + 1)) / (
            $tmp['tf'] + BM25_K1 * (1 - BM25_B + BM25_B *
            $countPages / $avgdl));
    }

    $t_rank = array(
        'rank' => array(),
        'id' => array()
    );

    foreach ($pageRank as $id => $rank) {
        $t_rank['rank'][] = $rank;
        $t_rank['id'][] = $id;
    }

    for ($i = 1; $i < count($t_rank['rank']); $i++) {
        $key = $t_rank['rank'][$i];
        $key_2 = $t_rank['id'][$i];
        $j = $i - 1;
        while (($j >= 0) && ($t_rank['rank'][$j] < $key))
            {
            $t_rank['rank'][$j + 1] = $t_rank['rank'][$j];
            $t_rank['id'][$j + 1] = $t_rank['id'][$j];
            --$j;
        }
        $t_rank['rank'][$j + 1] = $key;
        $t_rank['id'][$j + 1] = $key_2;
    }
    return $t_rank;
}
```

Функция deleteStopWords используется для удаления стоп-слов из массива. В качестве аргумента функции передаем массив поискового запроса пользователя, из которого удаляются стоп-слова. Подробное описание функции было в разделе «Поисковый интерфейс».

```
function deleteStopWords($arrayStrings, $dbLink){
    $searchWords = array();
    foreach ($arrayStrings as $str) {
        $word = PorterStemmer::Stem($str);
```

```
        $query = "SELECT * FROM stop_word WHERE term = '".
            $word."'";
        $result = mysqli_query($dbLink, $query);
        $tmp = mysqli_fetch_array($result);

        if (!$tmp) {
            $searchWords[] = $word;
        }
    }
    return $searchWords;
}
```

findPageIdsByWords — это функция нахождения уникальных идентификаторов документов по заданному массиву термов. В начале пробуем найти документы, содержащие все термы из массива. Если таких документов нет, то пробуем найти документы, содержащие хоть один терм из массива.

```
function findPageIdsByWords($searchWords, $dbLink){
    $ids = array();
    $pageIds = array();

    foreach($searchWords as $word) {
        $query = "SELECT * FROM term WHERE term = '".$word
            ."'";
        $result = mysqli_query($dbLink, $query);
        $tmp = mysqli_fetch_array($result);
        if ($tmp !== null) {
            $ids[] = $tmp['id'];
        }
    }

    if (empty($ids)) {
        return null;
    }
    $ids = implode(', ', $ids);

    $query = "SELECT * FROM term_page WHERE term_id IN (".
        $ids.")";
    $result = mysqli_query($dbLink, $query);

    while ($tmp = mysqli_fetch_array($result)) {
        $pageIds[] = $tmp['page_id'];
    }
```

```
    if (empty($pageIds)) {
        return null;
    }
    $countWords = count($searchWords);
    $ids = array_keys(array_filter(array_count_values(
        $pageIds), function ($value) use ($countWords) {
        return $value == $countWords;
    }));
    return !empty($ids) ? $ids : $pageIds;
}
```

Далее описана функция getResultsByPageRank, генерирующая поисковую выдачу на основе алгоритма ранжирования PageRank. На вход функции подается массив идентификаторов документов $pageIds. Инициализируем массив $resultLinks и составляем запрос к таблице page, который по идентификаторам документов из массива $pageIds находит документы и производит сортировку результатов по весу PageRank. Найденные результаты записываем в массив $resultLinks и возвращаем в качестве результата вызова функции.

```
function getResultsByPageRank($pageIds, $dbLink){
    $resultLinks = array();
    $query = "SELECT * FROM page WHERE id IN(".implode
        (',', $pageIds).") ORDER BY p_rank DESC";
    $res = mysqli_query($dbLink, $query);

    while ($tmp = mysqli_fetch_array($res)) {
        $resultLinks['page'][] = $tmp['url'];
        $resultLinks['rank'][] = $tmp['p_rank'];
    }
    return $resultLinks;
}
```

Поисковая выдача на основе алгоритма ранжирования HITS AUTH реализована в функции getResultsByHITSaut. Инициализируем массив $resultLinks и составим запрос к таблице page, который по идентификаторам документов из массива $pageIds находит документы и производит сортировку результатов по авторитетности документов. Найденные результаты записываем в массив $resultLinks и возвращаем в качестве результата вызова функции.

```
function getResultsByHITSaut($pageIds, $dbLink){
    $resultLinks = array();
```

```
    $query = "SELECT * FROM page WHERE id IN(".implode
        (',', $pageIds).") ORDER BY h_auth DESC";
    $res = mysqli_query($dbLink, $query);

    while ($tmp = mysqli_fetch_array($res)) {
        $resultLinks['page'][] = $tmp['url'];
        $resultLinks['rank'][] = $tmp['h_auth'];
    }
    return $resultLinks;
}
```

Поисковая выдача на основе алгоритма ранжирования HITS HUB выполняется с помощью функции getResultsByHITShub. Инициализируем массив $resultLinks и составим запрос к таблице page, который по идентификаторам документов из массива $pageIds находит документы и производит сортировку результатов по хабности страниц. Найденные результаты записываем в массив $resultLinks и возвращаем в качестве результата вызова функции.

```
function getResultsByHITShub($pageIds, $dbLink){
    $resultLinks = array();
    $query = "SELECT * FROM page WHERE id IN(".implode
        (',', $pageIds).") ORDER BY h_hub DESC";
    $res = mysqli_query($dbLink, $query);

    while ($tmp = mysqli_fetch_array($res)) {
        $resultLinks['page'][] = $tmp['url'];
        $resultLinks['rank'][] = $tmp['h_hub'];
    }
    return $resultLinks;
}
```

Далее идет поисковая выдача на основе алгоритма ранжирования по косинусам углов между векторами запроса и документов, реализованная в функции getResultsByTFIDF. На вход функции подается массив идентификаторов документов $pageIds и массив поискового запроса $searchWords. Инициализируем массив $resultLinks и получаем данные с помощью функции tf_idf($pageIds, $searchWords, $dbLink), которая рассчитывает ранг документа по поисковому запросу на основе алгоритма ранжирования по косинусам углов между векторами запроса и документов. Результат перезаписываем в $pageIds, который содержит идентификаторы и ранги документов. Далее, для каждой записи из $pageIds в цикле составляем запрос к таблице page, который

возвращает данные о документах. Найденные результаты записываем в массив $resultLinks, который возвращаем в качестве результата работы функции.

```
function getResultsByTFIDF($pageIds, $searchWords, $dbLink
    ){
    $resultLinks = array();
    $pageIds = tf_idf($pageIds, $searchWords, $dbLink);

    for ($i = 0; $i < count($pageIds['id']); $i++) {
        $query = "SELECT * FROM page WHERE id = '".
            $pageIds['id'][$i]."'";
        $res = mysqli_query($dbLink, $query);
        $mas = mysqli_fetch_array($res);
        $resultLinks['page'][] = $mas['url'];
        $resultLinks['rank'][] = $pageIds['rank'][$i];
    }
    return $resultLinks;
}
```

Поисковая выдача на основе алгоритма ранжирования Okapi BM25 реализована в функции getResultsByBM25. На вход функции подается массив идентификаторов страниц $pageIds и массив поискового запроса $searchWords. Инициализируем массив $resultLinks и получаем данные с помощью функции bm25($pageIds, $searchWords, $dbLink), которая подсчитывает ранги документов по поисковому запросу на основе алгоритма ранжирования Okapi BM25. Результат перезаписываем в $pageIds, который содержит идентификаторы документов и ранг документа. Далее, для каждой записи из $pageIds в цикле составляется запрос к таблице page, который возвращает данные о документах. Найденные результаты записываем в массив $resultLinks, который возвращаем в качестве результата работы функции.

```
function getResultsByBM25($pageIds, $searchWords, $dbLink)
    {
    $resultLinks = array();
    $pageIds = bm25($pageIds, $searchWords, $dbLink);

    for ($i = 0; $i < count($pageIds['id']); $i++) {
        $query = "SELECT * FROM page WHERE id = '".
            $pageIds['id'][$i]."'";
        $res = mysqli_query($dbLink, $query);
```

```
        $mas = mysqli_fetch_array($res);
        $resultLinks['page'][] = $mas['url'];
        $resultLinks['rank'][] = $pageIds['rank'][$i];
    }
    return $resultLinks;
}
```

Поисковая выдача без алгоритма ранжирования производится функцией getResultsWithoutAlgorithm. На вход функции подается массив $pagesIds, который содержит идентификаторы документов. В цикле по каждому элементу массива $pageIds находим документы по идентификаторам и добавляем в новый массив $resultLinks, который возвращаем в качестве результата работы функции.

```
function getResultsWithoutAlgorithm($pageIds, $dbLink){
    $resultLinks = array();
    for ($i = 0; $i < count($pageIds); $i++) {
        $query = "SELECT * FROM page WHERE id='".$pageIds[
            $i]."'";
        $res = mysqli_query($dbLink, $query);
        $mas = mysqli_fetch_array($res);
        $resultLinks['page'][] = $mas['url'];
    }
    return $resultLinks;
}
```

Далее идет алгоритм страницы поисковика с ранжированием результатов.

Если после отправки пользователем поискового запроса в GET есть значение q, то осуществляем поиск. Из строки запроса $_GET['q'] получаем массив $str слов запроса. Производим удаление стоп-слов из массива $str, используя функцию deleteStopWords, и записываем данные в массив $search_words. По заданному массиву слов $search_words находим идентификаторы документов, которые содержат эти слова, и воспользуемся функцией findPageIdsByWords, чтобы найти документы, содержащие данные слова. Если существуют документы, содержащие данные слова и передана информация о том, какой необходимо использовать алгоритм ранжирования, продолжаем работу по поиску документов. На данном этапе задача состоит в том, чтобы ранжировать найденные документы по тому или иному алгоритму.

Ранжируем документы на основе алгоритма PageRank. Ранжируем

документы на основе алгоритма HITS по авторитетности документов. Ранжируем документы на основе алгоритма HITS по хабности документов. Ранжируем документы на основе алгоритма на основе подсчета косинусов углов между векторами поискового запроса и документов. Для сравнения результатов была оставлена возможность поиска документов без применения алгоритмов ранжирования.

```
if (isset($_GET['q']) && !empty($_GET['q'])) {
    $str = explode(" ", $_GET['q']);
    $search_words = deleteStopWords($str, $dbLink);
    $pageIds = findPageIdsByWords($search_words, $dbLink);
    $result_links = array();
    if (!empty($pageIds)) {
        if (isset($_GET['alg'])) {
            switch ($_GET['alg']) {
                case 'PageRank':
                    $result_links = getResultsByPageRank(
                        $pageIds, $dbLink);
                    break;
                case 'HITSaut':
                    $result_links = getResultsByHITSaut(
                        $pageIds, $dbLink);
                    break;
                case 'HITShub':
                    $result_links = getResultsByHITShub(
                        $pageIds, $dbLink);
                    break;
                case 'TFIDF':
                    $result_links = getResultsByTFIDF(
                        $pageIds, $search_words, $dbLink);
                    break;
                case 'BM25':
                    $result_links = getResultsByBM25(
                        $pageIds, $search_words, $dbLink);
                    break;
                default:
                    $result_links =
                        getResultsWithoutAlgorithm(
                        $pageIds, $dbLink);
                    break;
            }
        } else {
            $result_links = getResultsWithoutAlgorithm(
                $pageIds, $dbLink);
```

```
        }
    }
}
```

Выводим форму поискового запроса. Выводим радио-кнопки для выбора алгоритмов ранжирования на экране пользователя. Формируем поисковую выдачу. Здесь на экран пользователя выводим результаты поиска. Проверяем существование индекса q в массиве $_GET. Если данный индекс есть, то был задан поисковый запрос. Проверяем массив $result_links на наличие результата поиска. Если результаты были получены, выводим на экран пользователя данные в виде ссылок на найденные документы. Иначе, если поиск не дал результатов, выводим пользователю соответствующее сообщение.

```
<html>
<head>
    <title>Search</title>
        </head>
<body>
<form method="GET">
    <input type="text" name="q" value="<?=(isset($_GET['q
        ])) ? $_GET['q'] : ''?>">

    <input type="radio" name="alg" value="NoRank" <?=(!
        isset($_GET['alg']) || (isset($_GET['alg']) &&
        $_GET['alg'] == 'NoRank')) ? 'checked="checked"' :
         ''?>> Without ranking
    <input type="radio" name="alg" value="PageRank" <?=(
        isset($_GET['alg']) && $_GET['alg'] == 'PageRank')
         ? 'checked="checked"' : ''?>> PageRank
    <input type="radio" name="alg" value="HITSaut" <?=(
        isset($_GET['alg']) && $_GET['alg'] == 'HITSaut')
        ? 'checked="checked"' : ''?>> HITS AUT
    <input type="radio" name="alg" value="HITShub" <?=(
        isset($_GET['alg']) && $_GET['alg'] == 'HITShub')
        ? 'checked="checked"' : ''?>> HITS HUB
    <input type="radio" name="alg" value="TFIDF" <?=(isset
        ($_GET['alg']) && $_GET['alg'] == 'TFIDF') ? '
        checked="checked"' : ''?>> TF-IDF: cos
    <input type="radio" name="alg" value="BM25" <?=(isset(
        $_GET['alg']) && $_GET['alg'] == 'BM25') ? '
        checked="checked"' : ''?>> TF-IDF: BM25
    <input type="submit" value="Search">
</form>
```

```
<?php if (isset($_GET['q'])) : ?>
<?php if (isset($result_links) && !empty($result_links)):
    ?>
Results: <?=count($result_links['page'])?>
        <ol>
    <?php for ($i = 0;$i < count($result_links['page']);$i
        ++): ?>
    <li>
        <a href="<?= $result_links['page'][$i] ?>"> <?=
            $result_links['page'][$i] ?> </a><?php if (
            isset($result_links['rank'][$i])):?> - <?=
            $result_links['rank'][$i] ?><?php endif;?>
        </li>
    <?php endfor; ?>
</ol>
<?php else: ?>
<p>No results</p>
        <?php endif; ?>
<?php endif; ?>
</body>
</html>
```

На данный момент поисковая система может ранжировать результаты поиска с помощью различных алгоритмов. При сравнении поисковой выдачи с ранжированием и без него очевидно, что качество поиска с использованием ранжирования намного лучше.

4. Кластеризация точек на карте и новостей

В данном разделе будет рассмотрена кластеризация двух типов объектов — точек на карте и новостных статей.

Нестрого говоря, кластеризация — это процесс разбиения некоторого множества на группы, состоящие из сходных (близких) объектов. Для кластеризации любого множества необходимо, во-первых, определить способ оценки сходства (близости) его элементов — другими словами, ввести функцию расстояния на этом множестве. Далее необходимо применить методы кластерного анализа для получения кластеризации. Выбор метода играет очень большую роль, различные методы обладают как достоинствами, так и недостатками.

Кластерный анализ широко применяется в биологии, медицине, социологии, психологии и многих других дисциплинах.

Работа с объектами на карте (торговыми точками, ресторанами и т. п.) подробно рассмотрена в главе «Одностраничные приложения. Основы реализации географических онлай-карт». Важно отметить, что с развитием инфраструктуры городов увеличивается и детализация карт. Сейчас в одном городе могут быть тысячи торговых точек, сотни остановок, десятки площадей и многое другое. Предположим, мы захотим выделить такие объекты на карте, допустим, обозначив их точками. Тогда при уменьшении масштаба карты в некоторой области экрана необходимо отрисовать огромное количество этих точек. Это порождает две проблемы. Во-первых, точки могут находиться настолько близко, что начнут перекрывать друг друга. Во-вторых, отображение большого количества объектов нагружает страницу. Для того, чтобы избежать этого, будет рассмотрена задача кластеризации объектов на карте методом k-средних и методом расстояний.

Кластеризация новостей во многом пересекается с главой «Поисковые системы»: тут также понадобятся кроулер, индексер и параметризация текстов новостей на основе меры TF-IDF. Для кластеризации новостей применяется алгоритм иерархической кластеризации снизу-вверх, по итогам работы которого можно выделить списки похожих друг на друга новостей.

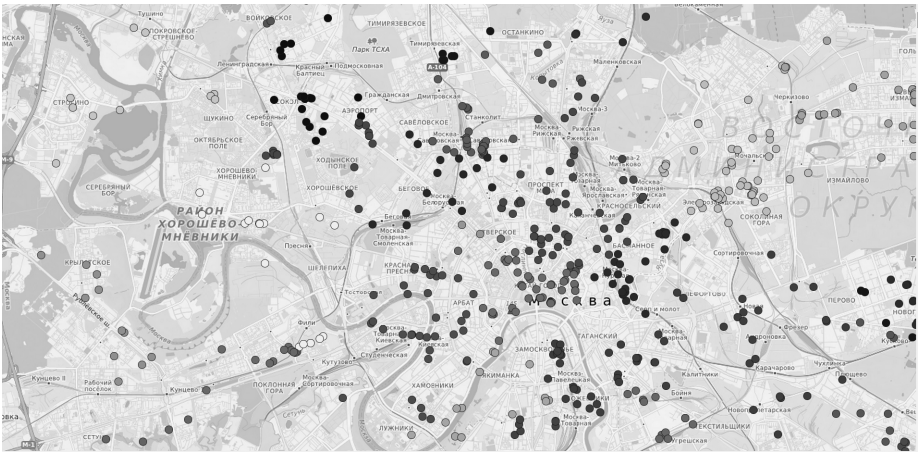

Рис. 17: Кластеризация методом расстояний.
Расстояние равно 3000 метров

4.1. Кластеризация объектов на карте

Рассмотрим задачу кластеризации выведенных на карту объектов (см. раздел 2). Будем использовать алгоритм «k-средних» и алгоритм кластеризации методом расстояний. Результатом работы каждого из алгоритмов является таблица в базе данных, в которой каждой точке сопоставлен соответствующий номер кластера. Алгоритм кластеризации методом расстояний описан в разделе **«Алгоритм кластеризации методом расстояний»**. Результат представлен на рисунке 17. Алгоритм кластеризации «k-средних» описан в разделе **«Алгоритм k-средних»**. Результат представлен на рисунке 18.

Будем использовать результаты, полученные в разделе 2 (базу данных, код обработчиков событий и прочее). Для решения задачи кластеризации нам потребуются дополнительные таблицы в базе данных. В таблице $listofpoints$ хранятся 1030 различных торговых точек Москвы. В таблице имеются следующие столбцы: идентификатор точки (id), название точки ($Name$), адрес каждой точки ($Address$), широта ($firstkoord$) и долгота ($secondkoord$). Таблица показана на рисунке 19.

В таблице $million$ 4 столбца. Первый и второй столбцы содержат идентификаторы точек из таблицы $listofpoints$, третий столбец задает расстояние между соответствующими точками в метрах. Чет-

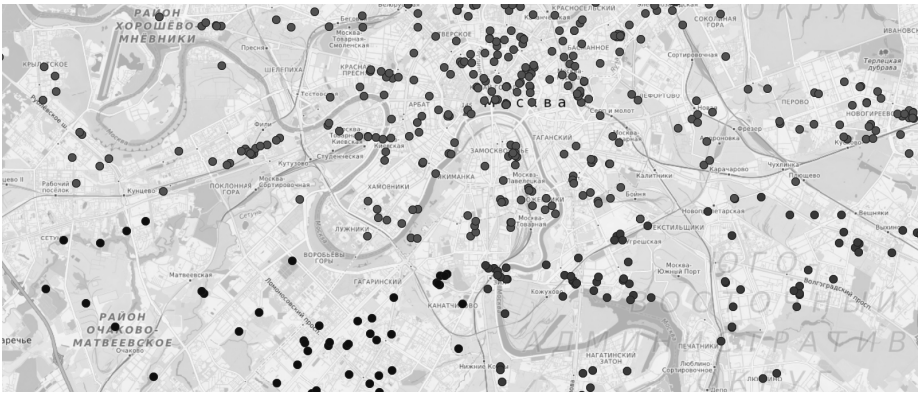

Рис. 18: Алгоритм k-средних. Параметр k=3

вертый столбец del является вспомогательным столбцом. Для вычисления расстояния между точками в таблице $million$ используем формулу

$$L = \arccos(\sin(\varphi_A) \cdot \sin(\varphi_B) + \cos(\varphi_A) \cdot \cos(\varphi_B) \cdot \cos(\lambda_A - \lambda_B)) \cdot R,$$

где φ_A и φ_B — широта, λ_A, λ_B — долгота данных пунктов, d — расстояние между пунктами, измеряемое в радианах длиной дуги большого круга земного шара. Расстояние между пунктами, измеряемое в километрах, определяется по формуле $L = d \cdot R$, где $R = 6371$ км — средний радиус земного шара.

Еще две таблицы — временные, они будут описаны в алгоритмах кластеризации.

4.1.1. Алгоритм кластеризации методом расстояний

В данном разделе мы описываем алгоритм решения задачи кластеризации объектов методом расстояний. Считаем, что алгоритм на входе получает некоторое число S и в каждом результирующем кластере любая точка находится на расстоянии не больше чем S от центра масс точек этого кластера. Разобьем задачу на следующие подзадачи.

- Записываем в таблицу $timemillion$ все строки таблицы $million$, значение расстояния которых меньше S.

id	Name	Adress	firstkoord	secondkoord	color
1	Доминатор2	Москва, Малая Семёновская улица, 28С13	55.7862	37.7165	1
2	Арбор Мунди2	Москва, Лермонтовский проспект, 2К1	55.7049	37.8354	1
3	Dreamwear2	Москва, улица Сущёвский Вал, 5С1	55.7847	37.5969	1
4	Apple2	Москва, Русаковская улица, 1	55.7922	37.6590	1
5	Волшебный мир 22комп	Москва, Сухонская улица, 7А	55.8564	37.6714	1
6	Гвоздь-2	Москва, проспект Андропова, 36	55.6458	37.6518	1
7	Атриум2	Москва, улица Земляной Вал, 33	55.7672	37.6571	1
8	Очаково2	Москва, Рябиновая улица, 44С3	55.6957	37.4372	1
9	Волшебный2 мир компь	Москва, улица Миклухо-Маклая, 55	55.6507	37.5545	1
10	Плеер.ру2	Москва, улица Мастеркова, 4	55.7390	37.6660	1
11	Доминатор	Москва, Малая Семёновская улица, 28С13	55.7852	37.7155	1
12	Арбор Мунди	Москва, Лермонтовский проспект, 2К1	55.7039	37.8454	1

Рис. 19: Таблица listofpoints

- Находим точку, в S-окрестности которой содержится максимальное число других точек.
- Для полученной точки находим все точки ее S-окрестности. Считаем эти точки рассмотренными и удаляем из таблицы $timemillion$ строки, содержащие эти точки.
- Запускаем рассмотренные шаги в цикле для сопоставления кластеров всем точкам таблицы $listofpoints$.
- Сохраняем итоговую кластеризацию в базе данных.

Далее рассматриваем каждый из описанных шагов более подробно.

Шаг 1. Записываем в таблицу $timemillion$ все строки таблицы $million$, значение расстояния которых меньше S

Напомним, что в таблице $million$ лежат все попарные расстояния между точками. Напишем запрос, который вынет из таблицы $million$ строки, соответствующие точкам, расстояние между которыми меньше S. Результат запроса представлен на рисунке 20.

```
SELECT id1, id2, distance FROM million WHERE distance <
    @S
```

id1	id2	distance
1	11	128
1	49	1721
1	62	1081
1	67	788
1	72	486

Рис. 20: Таблица timemillion

id1	amount
446	129
63	129
563	129
397	127
994	126

Рис. 21: Число элементов для центроидов

Шаг 2. Находим точку, в S-окрестности которой содержится максимальное число других точек

Результатом следующего запроса является таблица, каждая строка которой состоит из идентификатора точки и количества точек в ее S-окрестности, в порядке убывания количества. Результат представлен на рисунке 21.

```
SELECT id1, COUNT(*) as amount FROM timemillion
GROUP BY id1 ORDER BY amount DESC
```

Первая строка результата предыдущего запроса соответствует точке, имеющей наибольшее количество точек в ее S-окрестности. Будем называть ее центроидом. Записываем этот центроид в переменную $maximum$. Если $maximum$ равен 0, то каждая точка образует собственный кластер.

Шаг 3. Для полученной точки находим все точки ее S-окрестности, убираем найденные точки из рассмотрения

В следующих двух запросах удаляем из таблицы $timemillion$ центроид и все относящиеся к нему точки для того, чтобы далее они не рассматривались. Так как повторять запрос удаления для каждой точки нецелесообразно, сначала накапливаем id точек в строку str, а уже потом одним запросом удаляем.

```
DELETE FROM timemillion WHERE id1 = @maximum
DELETE FROM timemillion WHERE id1 IN(...)
```

Запускаем рассмотренные шаги в цикле для сопоставления кластеров всем точкам таблицы $listofpoints$

Применяем шаги с первого по третий. При помощи шага 1 записываем в таблицу $timemillion$ все строки таблицы $million$, значение расстояния которых меньше S. При помощи шага 2 находим точку, в S-окрестности которой содержится максимальное число других точек. При помощи шага 3 для полученной точки находим все точки ее S-окрестности. Считаем эти точки рассмотренными и удаляем из таблицы $timemillion$ строки, содержащие эти точки. На выходе получаем массив $finish$, где лежат id точки и номер кластера. Далее очищаем таблицу $timemillion$.

Сохраняем итоговую кластеризацию в базе данных

Обновляем таблицу $listofpoits$ и записываем значения массива $finish$ в соответствующие строки. Предположим, что функция $color$ сопоставляет заданному числу от 0 до 256^3 отдельный цвет. Предоставляем ее реализацию читателю. Применим эту функцию для раскраски всех точек. Результат показан на рисунке 22.

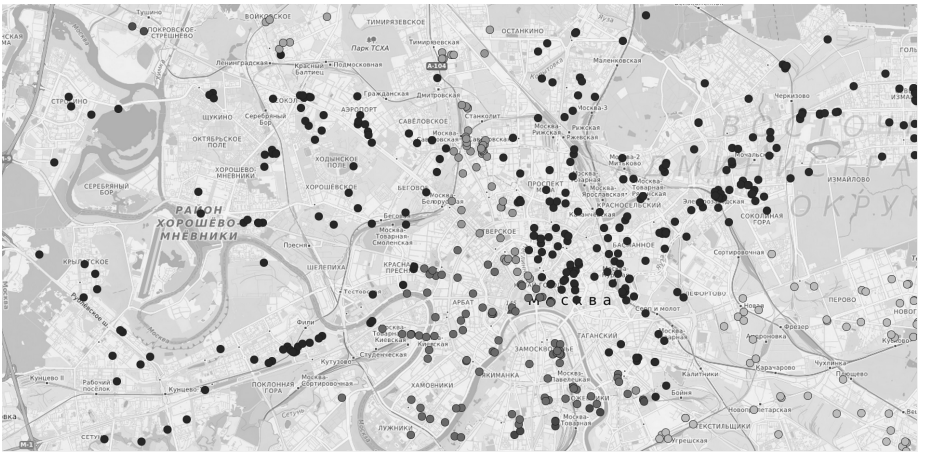

Рис. 22: Кластеризация методом расстояний. Расстояние равно 10000 метров

4.1.2. Алгоритм k-средних

В данном разделе мы описываем алгоритм кластеризации точек на карте методом k-средних. Реализация данного метода основывается на поиске центроидов и распределении точек по этим центроидам. Считаем, что алгоритм на входе получает некоторое число k — количество результирующих кластеров будет равняться k. Разобьем задачу на следующие подзадачи.

- Выбираем k первых точек и запоминаем их.
- Записываем во временную таблицу строки. Каждая стока имеет следующий вид: центроид, точка, относящаяся к центроиду, и расстояние между ними. Для каждого центроида перечисляются все относящиеся к нему точки.
- Находим минимальное расстояние от точек до центроидов.
- Для каждого кластера находим центр масс и ближайшую к нему точку по координатам. Переопределяем массив центроидов новыми данными.
- Проверяем на совпадения новые центроиды с центроидами из предыдущих итераций.
- Сохраняем итоговую кластеризацию в базе данных.

Шаг 1. Выбираем k первых точек и запоминаем их

Изначально данный алгоритм позволяет выбрать любые k точек из исходной таблицы с точками, для определенности мы выбираем первые из них. Выбранные точки мы назовем центроидами и удалим их из таблицы $million$. Пример удаления точек показан на рисунке 23. Удаленные точки не рассматриваются в следующих шагах алгоритма.

```
UPDATE million SET ‘del‘ = 0 WHERE id1 IN (...)
```

Шаг 2. Заполняем таблицу $timemillion$

С помощью следующих двух запросов мы выбираем точки, относящиеся к нашим центроидам, и записываем их во временную таблицу

id1	id2	distance	del
1	2	11722	1
1	3	7488	0
1	4	3660	1
1	5	8308	0
1	6	16147	1

Рис. 23
Пример удаления точек.

id1	id2	distance
11	1	128
11	2	11677
11	3	7424
11	4	3621
11	5	8392

Рис. 24
Таблица *timemillion*.

timemillion. Пример полученного результата можно видеть на рисунке 24.

```
SELECT * FROM million WHERE id1 IN (...) AND del=1
INSERT INTO timemillion (id1, id2, distance) VALUES (...)
```

Шаг 3. Находим минимальное расстояние от точек до центроидов

Результатом следующего SQL запроса является временная таблица *newtimemillion*, в которой лежат центроиды с соответствующими точками и расстояние между ними, таблица отсортирована по центроидам и по расстоянию в порядке возрастания. Все точки, относящиеся к одному центроиду, назовем кластером. Пример результата запроса можно видеть на рисунке 25.

```
INSERT INTO newtimemillion (id1,id2,mindist)
SELECT tm3.id1, tm2.id2, tm2.mindist FROM (SELECT tm1.id2,
min(tm1.distance) as mindist FROM timemillion tm1 GROUP BY
id2) as tm2 JOIN timemillion tm3 ON tm3.distance =
tm2.mindist AND tm3.id2=tm2.id2
```

Шаг 4. Для каждого кластера находим центр масс и ближайшую к нему точку по координатам. Переопределяем массив центроидов новыми данными

Данный запрос находит центр масс для каждого центроида путем сложения координат точек, относящихся к каждому из кластеров, и деления на их количество. Полученные координаты и будут являться центрами масс. Пример результата запроса представлен на рисунке 26.

id1	id2	mindist
11	1	128
11	4	3621
11	5	8392
11	7	4169
11	10	6005

Рис. 25
Таблица newtimemillion.

SredneeX	SredneeY
55.77062618	37.71102833
55.63937532	37.71179319
55.75962312	37.55219427

Рис. 26
Центры масс кластеров

```
SELECT AVG(firstkoord) as SredneeX,
AVG(secondkoord) as SredneeY FROM (SELECT * FROM
newtimemillion as tm1 JOIN listofpoints as tm2 ON
tm1.id2=tm2.id) as tm3 GROUP BY tm3.id1
```

Следующий SQL-запрос позволяет определить точку, которая лежит ближе всего к центру масс. Пример результата запроса представлен на рисунке 27.

id	min(ABS((firstkoord+secondkoord)-93.56114444))
1	0.00004444
2	0.00004444
3	0.00225556

Рис. 27: Поиск точек, ближайших к центрам масс своих кластеров

```
SELECT id, min(ABS(firstkoord+secondkoord-@summ))
FROM (SELECT * FROM newtimemillion as tm1
JOIN listofpoints as tm2 ON tm1.id2=tm2.id) as tm3
GROUP BY tm3.id1
```

Проверка на совпадение новых центроидов с любыми центроидами на предыдущих итерациях

Проверяем, были ли полученные центроиды зафиксированы на любой из предыдущих итерациях, если да, то завершаем алгоритм, если нет, добавляем полученные центроиды в массив со всеми остальными

центроидами, определенными на прошлых шагах, и повторяем шаги 2, 3 и 4.

Сохраняем итоговую кластеризацию в базе данных

Обновляем таблицу $listofpoits$ и записываем значения массива $finish$ в соответствующие строки. Предположим, что функция $color$ сопоставляет заданному числу от 0 до 256^3 отдельный цвет. Предоставляем ее реализацию читателю. Применим эту функцию для раскраски всех точек. Результат показан на рисунке 28.

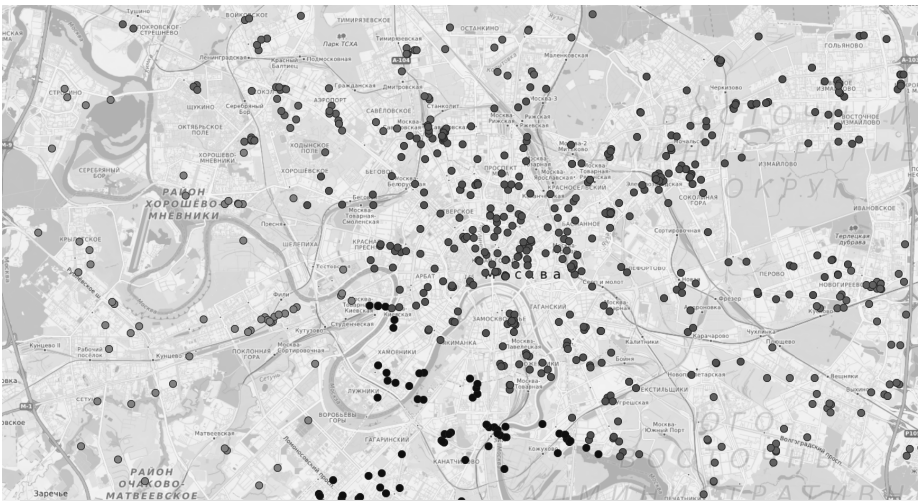

Рис. 28: Алгоритм k-средних. Параметр k=5

4.2. Кластеризация новостей

Задача кластеризации новостей во многом пересекается с задачей полнотекстового поиска по веб-страницам. В данной задаче, так же как и в задаче полнотекстового поиска, необходимо запустить кроулер для сбора статей, индексер для создания индекса и подсчет мер tf-idf с созданием векторов-параметров статей. Все это было описано в главе «Поисковые системы», и мы не будем подробно останавливаться на этом. На рисунке 29 приведен результат кластеризации новостей.

В главе «Поисковые системы» мы сравнивали векторы статей и поисковых запросов, но в данном случае мы будем сравнивать век-

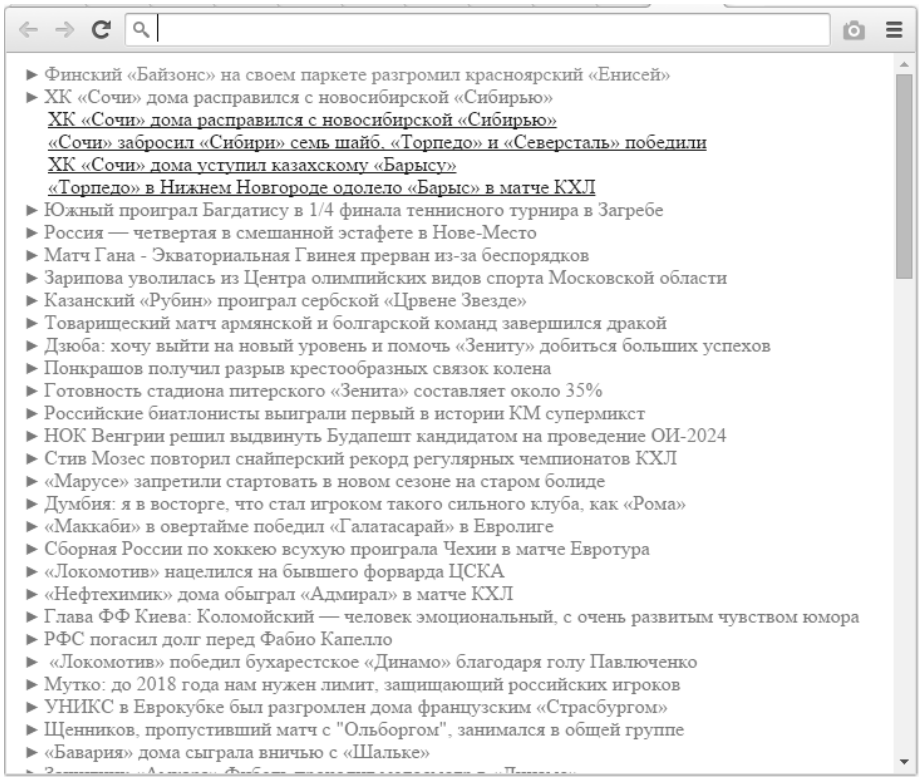

Рис. 29: Результат кластеризации новостей

торы статей между собой. В качестве расстояния между векторами возьмем косинус угла между ними, далее будем называть его косинусным расстоянием. Напомним, косинусное расстояние определяется по следующей формуле.

$$\rho(d, d') = 1 - \frac{\sum_{i=1}^{n} (d_i \cdot d'_i)}{\sqrt{\sum_{i=1}^{n} d_i^2} \cdot \sqrt{\sum_{i=1}^{n} d_i'^2}}$$

Реализуем данную формулу на языке программирования PHP. Создаем функцию getCosDistance, которая принимает в качестве параметров два вектора $a и $b.

```
function getCosDistance($a, $b) {
    $numerator = 0;
    $denominatorA = 0;
    $denominatorB = 0;
    for ($i = 0; $i < count($a); $i++) {
        $numerator += ($a[$i] * $b[$i]);
        $denominatorA += pow($a[$i], 2);
        $denominatorB += pow($b[$i], 2);
    }
    return 1 - ($numerator/(sqrt($denominatorA)*sqrt(
        $denominatorB)));
}
```

Реализуем несколько вспомогательных функций.

Функция getCountPages была реализована в главе 3, она подсчитывает общее число новостных страниц. Приведем ее код целиком.

```
function getCountPages($msql_link) {
    $q = "SELECT COUNT(*) FROM page";
    $res = mysqli_query($msql_link, $q);
    $tmp = mysqli_fetch_array($res);
    return $tmp[0] - 1;
}
```

Функция buildVectorPage используется для создания кластера, содержащего одну страницу. В качестве центроида (центра масс) кластера, очевидно, будет использоваться вектор данной страницы. Поэтому вся работа функции заключается в получении вектора страницы из базы данных, его парсинге и возвращении в качестве результата.

```
function buildVectorPage($pageIds, $link){
    $vectorPage = [];
    $query="SELECT * FROM page_vector WHERE page_id =" .
        $pageIds;
    $res = mysqli_query($link, $query);
    $tmp = mysqli_fetch_array($res);
    $arrayVector = explode(",", $tmp['vector']);

 for ($i = 1; $i <= count($arrayVector); $i++) {
     $vectorPage[$i] = (double)$arrayVector[$i-1];
 }

 return $vectorPage;
}
```

Функция buildVectorCluster объединяет два кластера. Вычисляет-

ся центроид нового кластера и возвращается в качестве результата.

```
function buildVectorCluster($vecI, $vecJ){
    $vectorCluster = [];

    for ($i = 0; $i < count($vecI); $i++) {
        $vectorCluster[$i] = ($vecI[$i]+$vecJ[$i])/2;
    }

    return $vectorCluster;
}
```

Функция saveCluster сохраняет центроид кластера в базу данных.

```
function saveCluster($vec, $msql_link){
    $vectorCluster = implode(',', $vec);

    $q = "INSERT INTO cluster(vector) value('" .
        $vectorCluster . "')";
    $res = mysqli_query($msql_link, $q);

    return mysqli_insert_id($msql_link);
}
```

Наконец, реализуем основную функцию hierarchical, которая и будет выполнять кластеризацию новостей иерархическим методом. Метод работает следующим образом. Вначале создается столько кластеров, сколько имеется новостей — по одной новости на кластер. Затем идет итеративный процесс объединения кластеров. На каждой итерации находится пара кластров с минимальным расстоянием между их центроидами. Если расстояние меньше некоторого порогового значения, объединяем два кластера в один. В противном случае завершаем итерации и берем в качестве результата полученную кластеризацию. В качестве порогового значения будем использовать 0.43.

В первую очередь объявим вспомогательные переменные и построим изначальные кластера — по одному кластеру на каждую новость. Для каждого кластера буем хранить его центроид и список составляющих его новостей (точнее, их индексов).

```
function hierarchical($msql_link){
    $countClusters = getCountPages($msql_link);
    $distance = [];
    $clusters = [];
    $check = true;

    $q = "SELECT * FROM page";
    $numOfPages = mysqli_query($msql_link, $q);

    for ($i = 1; $i <= $countClusters; $i++) {
        $j = mysqli_fetch_array($numOfPages)['id'];
        $clusters[$i] = [];
        $clusters[$i][0] = @buildVectorPage($j, $msql_link
            );
        $clusters[$i][1] = [];
        $clusters[$i][1][0] = intval($j);
    }
```

Вычислим попарные расстояния между кластерами.

```
for ($i = 1; $i <= $countClusters; $i++) {
    $distance[$i] = [];
    for ($j = $i; $j <= $countClusters; $j++) {
        $distance[$i][$j]=@getCosDistance($clusters[$i
            ][0], $clusters[$j][0]);
    }
}
```

Запустим итеративный процесс объединения кластеров. На каждой итерации ищем пару кластеров с минимальным расстоянием между центроидами.

```
while($check) {
    $check = false;
    $minDistance = 1;
    $minI = 0;
    $minJ = 0;
    for ($i = 1; $i <= $countClusters; $i++) {
        if (isset($clusters[$i])) {
            for ($j = $i+1; $j<=$countClusters; $j++){
                if (isset($clusters[$j])) {
                    if ($minDistance>$distance[$i][$j] and
                        $distance[$i][$j] > 0.0) {
```

```
                        $minDistance=$distance[$i][$j];
                        $minI = $i;
                        $minJ = $j;
                    }
                }
            }
        }
    }
```

Если найденное расстояние меньше порогового, объединяем кластеры. Для удобства новый кластер сохраняем на место одного из прежних кластеров. Пересчитываем новый центроид и объединяем составляющие новости.

```
if ($minDistance < 0.43 ) {
    $clusters[$minI][0]=@buildVectorCluster($clusters[
        $minI][0], $clusters[$minJ][0]);
    $c = count($clusters[$minJ][1]);
    for ($k = 0; $k < $c; $k++)
        array_push($clusters[$minI][1],$clusters[$minJ
            ][1][$k]);
    unset($clusters[$minJ]);
```

Далее необходимо вычислить расстояния между старыми кластерами и новыми. Вначале удалим старые расстояния, связанные с объединенными кластерами.

```
for ($j = $minJ + 1; $j <= $countClusters; $j++) {
    unset($distance[$minJ][$j]);
}
for ($j = 1; $j < $minJ; $j++) {
    unset($distance[$j][$minJ]);
}
```

Далее вычислим новые расстояния.

```
for ($j = $minI + 1; $j <= $countClusters; $j++) {
    if (isset($clusters[$j])) {
        $distance[$minI][$j]=@getCosDistance($clusters[
            $minI][0], $clusters[$j][0]);
    } else {
        unset($distance[$minI][$j]);
    }
```

```
}
for ($j = 1; $j < $minI; $j++) {
    if (isset($clusters[$j])) {
        $distance[$j][$minI]=@getCosDistance($clusters[$j
            ][0], $clusters[$minI][0]);
    } else {
        unset($distance[$j][$minI]);
    }
}
```

Выставляем флаг продолжения итераций в истину и переходим к следующей итерации.

```
$check = true;
```

После завершения итераций сохраняем полученные кластеры в базу данных.

```
$cCluster = 0;
$ccCluster = 0;
foreach($clusters as $cluster) {
    if (count($cluster[1]) > 0) {
```

Сохраняем информацию о центроиде.

```
$cId = saveCluster($cluster[0], $msql_link);
```

Затем сохраним данные о новостях, входящих в состав кластера.

```
$pages = "(".implode(', ', $cluster[1]).")";
$q="SELECT * FROM page LEFT JOIN page_vector ON
    page.id = page_vector.page_id WHERE page.id IN
    ".$pages;
$res = mysqli_query($msql_link, $q);
while ($page = mysqli_fetch_array($res)) {
    echo $cId." - ".$page['id']." - ".$page['title
        ']."\n";
    $distance = @getCosDistance($cluster[0], $page
        ['vector']);
    $q = "INSERT INTO cluster_page(cluster_id,
        page_id, distance) value('" . $cId . "',
        '" . $page['page_id'] . "', '" . $distance
         . "')";
```

```
            $resI = mysqli_query($msql_link, $q);
            $ccCluster++;
        }
        $cCluster++;
    }
}
```

Дальнейший вывод результатов кластеризации новостей оставим читателю.

5. Классификация изображений и музыки

Данный раздел посвящен разработке алгоритмов классификации музыкальных файлов и классификации изображений.

Задача классификации музыкальных файлов состоит в следующем. Дано множество музыкальных файлов. На вход подается файл, похожий на один из них (например, звукозапись, либо файл, полученный из исходного искажениями с помощью специализированных программ, и тому подобное). Требуется определить, с каким именно файлом из начального множества схож поданный на вход файл. Для этого нужно научиться парсить WAV-файлы, написать алгоритм получения из музыкального файла вектора для сравнения с другими файлами, выбрать метрику для определения близости векторов, а также написать WEB-приложение, позволяющее загружать музыкальные файлы и сравнивать загруженные файлы с исходными.

Кроме задачи классификации музыкальных файлов рассматривается задача классификации изображений. Дано множество из нескольких произвольных картинок. На вход подается картинка из этого множества, подверженная различным искажениям. Программа должна отнести картинку к своему оригиналу. Для этого нужно научиться считывать файлы изображений и представлять их в виде массива пикселей, написать алгоритм построения вектора по картинке, выбрать метрику для определения близости векторов, а также реализовать WEB-приложение, позволяющее загружать изображение и сравнивать его с эталонами для выявления оригинала.

Для простоты будет реализовано одно приложение, решающее обе задачи. Необходимо создать базу данных для хранения информации о файлах-образцах (как для классификации музыки, так и для классификации изображений) — это будет сделано в разделе «Структура базы данных». Также необходимо реализовать пользовательский интерфейс, который позволит загружать музыку и картинки и базу данных в качестве эталонов и сравнивать с ними тестовые файлы (раздел «Интерфейс»).

Для задачи классификации музыкальных файлов необходимо решить следующие подзадачи.

Первая подзадача состоит в парсинге WAV-файла. На вход подается музыкальный файл. Требуется реализовать класс для работы с WAV на языке программирования PHP [55]. На вход классу по-

дается файл, а на выходе получаем структуру, содержащую данные файла — характеристики файла (частота, битрейт и др.) и основной массив числовых данных, соответствующих звучанию музыкального файла во времени (раздел «Первичная обработка wav-файлов»).

Вторая подзадача состоит в создании алгоритма построения вектора для оценки различий между музыкальными файлами. Будем использовать так называемый кепстральный анализ. Необходимо реализовать алгоритм параметризации, основанный на MFCC-коэффициентах, на языке программирования PHP [56]. На вход алгоритма подается массив данных, полученный при парсинге WAV-файла, а на выходе получаем массив — вектор мелкепстральных коэффициентов (раздел «Составление вектора признаков по музыкальному файлу»).

Третья подзадача состоит в выборе метрики, позволяющей определять близость векторов, и реализации соответствующего алгоритма. Для этого используется алгоритм, основанный на методе динамического программирования (раздел «Вычисление расстояния между векторами mfcc-коэффициентов»).

Четвертая подзадача — создание функционала по загрузке, удалению и классификации музыкальных файлов, используя базу данных, пользовательский интерфейс и решения подзадач 1–3 (раздел «Классификация музыкальных файлов»).

Для задачи классификации изображений необходимо решить следующие подзадачи.

Первая подзадача состоит в считывании файла изображения для последующей работы с ним как с массивом пикселей. Для работы с изображениями будет реализован класс на языке программирования PHP, который читает файл изображения и возвращает массив пикселей [58] (раздел «Первичная обработка изображения»).

Вторая подзадача состоит в создании алгоритма построения вектора по картинке. Эти векторы мы будем использовать для сравнения картинок друг с другом. Для построения векторов будет использован алгоритм HOG [59] (раздел «Составление вектора признаков по изображению»).

Третья подзадача состояла в выборе метрики для работы с векторами и реализации соответствующего алгоритма. Будет использована метрика хи-квадрат (раздел «Вычисление расстояния между векторами признаков изображений»).

Четвертая подзадача — создание функционала по загрузке, удалению и классификации изображений, используя базу данных, пользовательский интерфейс и решения подзадач 1–3 (раздел «Классификация изображений»).

В итоге будет реализовано приложение, которое успешно решает поставленные задачи.

Для тестирования классификации музыки использовалось множество из 12 файлов-примеров и 35 файлов-тестов. Каждый файл (и среди примеров, и среди тестов) являлся звукозаписью известной мелодии, записанной с помощью программы Audacity 2.1.0 ([52]), имел продолжительность приблизительно 5–7 секунд, частоту дискретизации 8000 Гц и являлся моно-файлом. Таким образом, похожими считались аудиозаписи одной композиции. В результате из 35 тестовых файлов корректно были определены 30 файлов. На текущий момент на качество работы существенно влияет различие громкости — большинство из 5 неверно распознанных файлов как раз сильно отличалось этим от эталонных (как в сторону увеличения громкости, так и в сторону уменьшения). Архив музыкальных файлов можно скачать по ссылке [53].

Для тестирования классификации изображений использовалось множество из 9 файлов-примеров и 27 файлов-тестов (по 3 файла на каждый пример). Все тесты были получены из примеров с помощью программы Adobe Photoshop применением различных искажений. Из 27 файлов 24 были классифицированы верно. Текущий алгоритм распознавания довольно успешно справился с наложением различных фильтров, растягиванию и сжатию файлов. В то же время он показывает слабые результаты при распознавании таких искажений, как смена ориентации, повороты, смещение картинки относительно левого верхнего угла и т. п. Архив с изображениями можно скачать по ссылке [54].

Задачи, рассмотренные в данном разделе, имеют ясную практическую значимость. Не меньшую ценность представляют собой другие похожие задачи. Например, существует задача классификации музыкальных файлов по жанрам. Требуется отнести музыкальный файл не к файлу-оригиналу, а к целому классу файлов, относящихся к одному музыкальному жанру. Для решения этой и многих подобных задач требуются базовые знания такой дисциплины, как машинное обучение. Тем не менее те алгоритмы и подходы, которые будут рас-

смотрены далее, будут востребованы при их решении. Таким образом, решенные в этом разделе задачи могут быть усложнены и обобщены, и WEB-приложение, которое будет написано в процессе изучения раздела, может стать основой для решения более широкого круга задач.

5.1. Структура базы данных

В базе данных приложения будет храниться информация о файлах-эталонах (и музыкальных файлах, и изображениях). Создадим базу данных classification с одной таблицей samples, которая содержит следующие поля:

- id — id файла, первичный ключ, тип int(11).
- type — тип файла (музыка или изображение), тип varchar(10).
- name — наименование файла, тип varchar(64).
- vector — данные файла, тип longtext.
- file_path — путь к файлу, тип varchar(256).
- file_url — url файла, тип varchar(256).

Далее создадим файл connect_db.php для программного подключения к нашей базе данных. Зададим имя пользователя и пароль для подключения.

```
$connect=mysqli_connect('localhost','user','password');
if (!$connect) {
    echo 'Can't connect to database.';
    exit;
}
if (!mysqli_select_db($connect, 'classification')) {
    echo 'Can't connect to database.';
    exit;
}
mysqli_set_charset($connect, 'utf8');
```

5.2. Интерфейс

Теперь необходимо реализовать пользовательский интерфейс.

Создадим файлы interface.php, create_sample.php, test_file.php и delete_sample.php. Файл interface.php будет содержать пользовательский интерфейс и будет реализован сейчас, остальные файлы будут содержать различные функции работы с файлами (создание файла-эталона, нахождение файла-эталона и удаление файла-эталона) и будут реализованы позднее.

Подключаем к интерфейсу файл работы с базой данных.

```
require('connect_db.php');
```

Загружаем все классы изображений и музыкальных файлов из БД.

```
$samplesRes = mysqli_query($connect, 'SELECT * FROM
    samples WHERE 1');
$samples = array();
while ($sample = mysqli_fetch_assoc($samplesRes)) {
    $samples[] = $sample;
}
```

Создадим начало HTML разметки.

```
<!doctype html>
<html>
    <head>
        <meta charset="utf8" />
        <title>Classifying images and music</title>
        <style>
            #img-table {
                float: left;
                width: 600px;
                text-align: left;
                margin: 2px 10px;
                display: inline;
                border: 1px solid #000;
            }

            #music-table {
                float: left;
                text-align: left;
                margin: 2px 10px;
```

```
            display: inline;
            width: 400px;
            border: 1px solid #000;
        }

        #sample-load {
            clear: both;
        }
    </style>
    <script src="https://ajax.googleapis.com/ajax/libs
        /jquery/2.1.4/jquery.min.js"></script>
```

Далее напишем JS-функцию загрузки файла на сервер для классификации и показа результата на странице.

```
<script>
    $(function () {
        $('#test_file').on('submit', function (e) {
            e.preventDefault();
            var $this = $(this);
            if ($this.find('input[name=file]').val() ==
                '') {
                alert('Choose the file.');
                return;
            }

            var formData = new FormData(this);
            $.ajax({
                type: 'POST',
                url: $this.attr('action'),
                data: formData,
                cache: false,
                contentType: false,
                processData: false,
                dataType: 'json',
                success: function(data) {
                    var insertInto = $('#sample_'+data['
                        min']['sample']['id']);
                    switch ($this.find('input[name=
                        sample_type]:checked').val()) {
                        case 'img':
                            insertInto.append('<img src
                                ="'+data['fileUrl']+'"
                                />');
                            break;
```

```
                        case 'music':
                            insertInto.append('<audio src
                                ="'+data['fileUrl']+'"
                                controls=""></audio>'+data
                                ['additionalData']['
                                filename']+ ' | ');
                            break;
                    }
                },
                error: function(data) {
                    alert('Error.');
                }
            });
        });
    });
</script>
```

Реализуем вывод формы загрузки файла для классификации.

```
    </head>
    <body>
        <div>
            <h3>Test file</h3>
            <form method="POST" id="test_file" enctype="
                multipart/form-data" action="test_file.php">
                <label><input type="radio" name="sample_type"
                    value="img" checked="checked" /> Image</
                    label>
                <label><input type="radio" name="sample_type"
                    value="music" /> Music</label>
                <input type="file" name="file" />
                <button type="submit">Test</button>
            </form>
        </div>
```

Далее выводим список классов изображений и музыки.

```
    <div id="img-table">
        <h3>Images</h3>
        <?php foreach ($samples as $sample):?>
            <?php if ($sample['type'] != 'img') continue;?>
            <div id="sample_<?=$sample['id']?>">
                <img src="<?=$sample['file_url']?>" />
                <a href="/delete_sample.php?id=<?=$sample['id
                    ']?>">Delete class</a>
```

```
        </div>
    <?php endforeach;?>
</div>
<div id="music-table">
    <h3>Music</h3>
    <?php foreach ($samples as $sample):?>
        <?php if ($sample['type'] != 'music') continue;?>
        <div id="sample_<?=$sample['id']?>">
            <?=$sample['name']?>
            <audio src="<?=$sample['file_url']?>" controls
                =""></audio>
            <a href="/delete_sample.php?id=<?=$sample['id
                ']?>">Delete class</a>
        </div>
    <?php endforeach;?>
</div>
```

Наконец, выводим форму загрузки файла для создания нового класса.

```
          <div id="sample-load">
              <h3>Create class</h3>
              <form method="POST" enctype="multipart/form-
                  data" action="create_sample.php">
                  <label><input type="radio" name="
                      sample_type" value="img" checked="
                      checked" /> Image</label>
                  <label><input type="radio" name="
                      sample_type" value="music" /> Music</
                      label>
                  <input type="text" name="name" />
                  <input type="file" name="file" />
                  <button type="submit">Create</button>
              </form>
          </div>
      </body>
  </html>
```

Реализация пользовательского интерфейса завершена. Файлы create_sample.php, test_file.php и delete_sample.php, реализующие основной функционал приложения, будут заполнены позднее.

5.3. Первичная обработка wav-файлов

Итак, рассмотрим самый обычный WAV файл (Windows PCM). Он состоит из двух частей. Одна из них — заголовок файла, другая —

область данных. В заголовке файла хранится информация о размере файла, количестве каналов, частоте дискретизации, количестве бит в сэмпле (эту величину еще называют глубиной звучания) и другие характеристики.

Для большего понимания смысла описанных величин необходимы базовые знания об оцифровке звука. Звук состоит из колебаний, которые при оцифровке приобретают ступенчатый вид. Это обусловлено тем, что компьютер воспроизводит звук дискретно, т. е. в течение коротких промежутков времени производится колебание определенной фиксированной амплитуды. Продолжительность таких промежутков определяется частотой дискретизации — количеством промежутков в одной секунде. Например, если частота дискретизации равна 44.1 kHz, это значит, что продолжительность промежутка есть 1/44100 секунды. Современные звуковые карты поддерживают частоту дискретизации до 192 kHz.

Амплитуда звука в каждом промежутке времени выражается числом, занимающим в памяти (файле) 8, 16, 24 или 32 бита. Чем больше число занимает места в памяти (файле), тем больше диапазон значений для этого числа, а значит и для амплитуды.

Для получения стереозвука файл может состоять из нескольких независимых звуковых дорожек — каналов. Если в файле один канал (моно-вариант), значения амплитуд расположены в файле последовательно. Если имеется два или более каналов (стерео-вариант), используется более сложный способ хранения. Последовательно хранятся данные по всем промежуткам времени. Для каждого промежутка хранятся значения амплитуды для всех каналов, причем порядок следования каналов одинаковый для всего файла. Например, если имеется два канала, сначала идет значение амплитуды для левого канала, затем для правого, после снова для левого и так далее.

Совокупность амплитуды и короткого промежутка времени носит название сэмпл.

На рисунке 30 показана структура хранения WAV-файла в памяти.

Создадим на PHP класс WavParse, который будет считывать файл формата WAV и получать из него массив значений амплитуд и остальные характеристики [55].

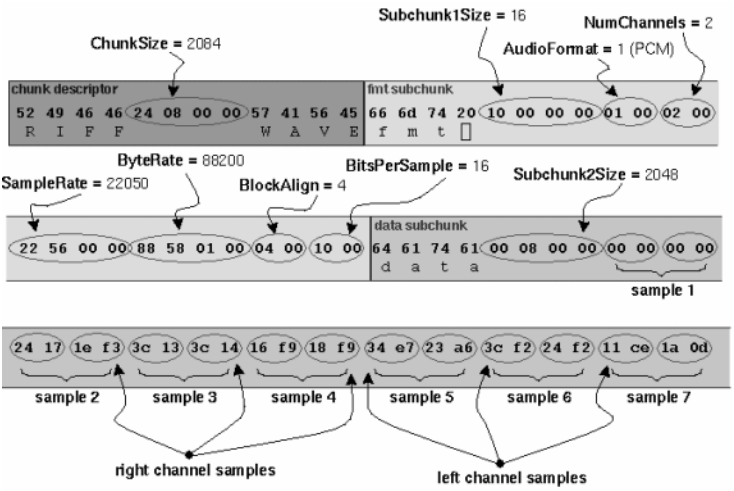

Рис. 30: Структура WAV-файла

```
class WavParse {
    public $wav;
    private static $HEADER_LENGTH = 44;
    ...
}
```

Создадим вспомогательные методы класса для считывания бинарных данных. Реализуем методы преобразования бинарных данных в строку — readString, в числа — readLong, и в слова — readWord. Кроме того, реализуем метод распаковки бинарных данных — readUnpacked.

```
public function readString($handle, $length) {
    return self::readUnpacked($handle, 'a*', $length);
}

public function readLong($handle) {
    return self::readUnpacked($handle, 'V', 4);
}

public function readWord($handle) {
    return self::readUnpacked($handle, 'v', 2);
}
```

```
private static function readUnpacked($handle, $type,
    $length){
    $r = unpack($type, fread($handle, $length));
    return array_pop($r);
}
```

Создадим конструктор **__construct** класса. В нем будет считывание wav-файла с проверкой на читаемость и корректность. Размер файла не может быть меньше размера заголовков.

```
public function __construct($filename) {
    if (is_readable($filename)) {
        $filesize = filesize($filename);
         if ($filesize < self::$HEADER_LENGTH)
            return false;
```

Реализуем считывание данных из заголовков файла.

```
        $handle = fopen ($filename, "r");
        $this->wav = array(
            'header' => array(
                'chunkid' => self::readString($handle,4),
                'chunksize' => self::readLong($handle),
                'format' => self::readString($handle, 4)
            ),
            'subchunk1' => array(
                'id' => self::readString($handle, 4),
                'size' => self::readLong($handle),
                'audioformat' => self::readWord($handle),
                'numchannels' => self::readWord($handle),
                'samplerate' => self::readLong($handle),
                'byterate' => self::readLong($handle),
                'blockalign' => self::readWord($handle),
                'bitspersample' => self::readWord($handle)
            ),
            'subchunk2' => array(
                'id' => fread ($handle, 4),
                'size' => self::readLong($handle),
                'data' => null
            )
        );
```

Приступим к считыванию непосредственно массива амплитуд. Для оптимизации памяти будем пропускать каждую вторую амплитуду. Если файл многоканальный, в качестве значения амплитуды берется среднее арифметическое значений амплитуд для всех каналов. Будем считать, что размер сэмплов составляет 1 или 2 байта, обработку

варианта с большим числом байтов предоставим читателю.

```
$data = [];
$peek = $this->wav["subchunk1"]["bitspersample"];
$byte = $peek / 8;

// Number pf bytes for skiping - depends on
    bitspersample and numchannels..
$skeepingBytesCount = $byte * $this->wav["
    subchunk1"] ["numchannels"];

while(!feof($handle)){
    //get number of bytes depending on bitrate
    $val = 0;
   for($j = 0; $j < $this->wav["subchunk1"]["
       numchannels"]; $j++){
         $bytes = array();
         for ($i = 0; $i < $byte; $i++){
             $bytes[$i] = fgetc($handle);
         }
         if(!isset($bytes[1]))
            $bytes[1]=null;
         switch($byte){
             //get value for 8-bit wav
             case 1:
                 $val += $this->findValue($bytes
                     [0],$bytes[1]);
                  break;
             //get value for 16-bit wav
             case 2:
                 if(ord($bytes[1]) & 128){
                     $temp = 0;
                 } else {
                     $temp = 128;
                 }
                 $temp = chr((ord($bytes[1]) & 127)
                      + $temp);
                 $val += $this->findValue($bytes
                     [0], $temp) / 256;
                 break;
         }
    }
    $data[] = floor($val / $this->wav["subchunk1
        "]["numchannels"]);
    //skip bytes for memory optimization
```

```
            fread ($handle, $skeepingBytesCount);
        }
        fclose ($handle);
        $this->wav["subchunk2"]["data"]=$data;
    }
}
```

Таким образом, после вызова конструктора объекта класса WavParse в свойстве wav будут храниться все данные wav-файла.

5.4. Составление вектора признаков по музыкальному файлу

Итак, мы научились получать из wav-файла массив амплитуд. Однако его непосредственное использование при оценке различий между музыкальными файлами достаточно затруднительно. Размер данного массива может быть достаточно большим, что потребует больших затрат машинного времени и памяти при вычислениях. Таким образом, достаточно важно было бы преобразовать массив так, чтобы сохранить информацию, достаточную для оценки различий между файлами, и при этом существенно сократить размер сравниваемых данных.

Одним из наиболее известных методов решения этой задачи являются мел-частотные кепстральные коэффициенты (МЧКК, MFCC). Они широко применяются в задачах распознавания речи. Основаны на двух ключевых понятиях: мел-шкала и кепстр.

Мел-шкала моделирует частотную чувствительность человеческого слуха. Специалистами по психоакустике было установлено, что изменение частоты в два раза в диапазоне низких и высоких частот человек воспринимает по-разному. В частотной полосе до 1000 Гц субъективное восприятие удвоения частоты совпадает с реальным увеличением частоты в два раза, поэтому до 1000 Гц мел-шкала близка к линейной. Для частот выше 1000 Гц мел-шкала является логарифмической.

Кепстр (cepstrum) — это результат дискретного косинусного преобразования от логарифма амплитудного спектра сигнала. Формально:

$$c(n) = DCT\{\log(|F\{f(t)\}|^2)\}$$

Создадим класс MFCC (mel frequency cepstral coefficients), который по массиву амплитуд будет строить вектор мел-кепстральных коэффициентов [56]. Объявим свойства класса и реализуем конструктор,

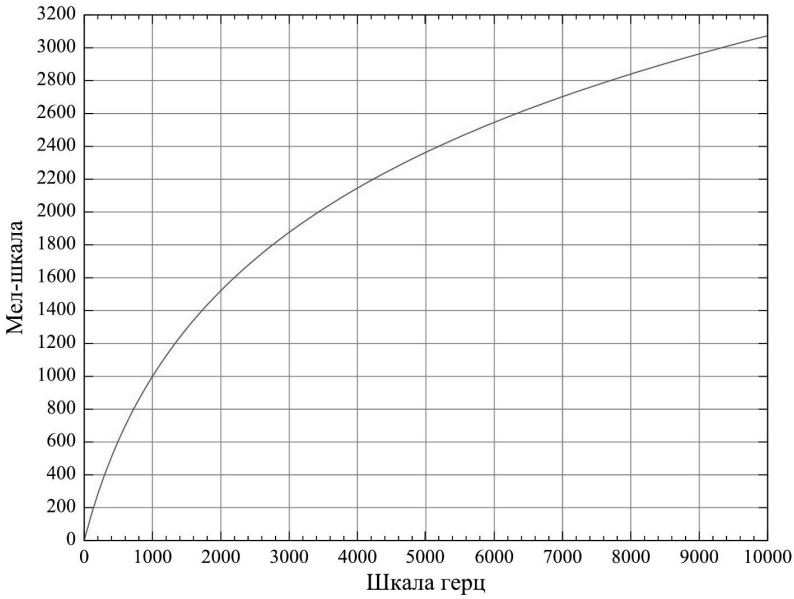

Рис. 31: Мел-шкала.

получающий в качестве параметров массив амплитуд и некоторые дополнительные характеристики музыкального файла.

```
class MFCC{
    public static $framesize = 220;
    public static $filterCount = 12;
    public static $flStart = 300;
    public static $flEnd = 8000;
    public static $coefCount = 10;
    public $fileSize;
    public $sampleRate;
    public $file=[];
    public $Re = [];
    public $Im = [];
    public $Pn = [];
    public $scaleCenter = [];
    public $Fsmp = [];
    public $Xn = [];
    public $Cn = [];
    public $_Hik = [];
```

```
    public function __construct($fileArray,
        $fileSampleRate) {
        $this->file = $fileArray;
        $this->fileSize = count($fileArray);
        $this->sampleRate = $fileSampleRate;
    }
    ...
}
```

Сами MFCC-коэффициенты вычисляются следующим способом. Вначале массив амплитуд разбивается на фреймы — группы по 220 амплитуд (значение 220 подобрано опытным путем и соответствует свойству \$framesize в коде). При этом используется метод скользящего окна, т. е. начало каждого последующего фрейма приходится на середину предыдущего. Для каждого фрейма вычисляются мел-кепстральные коэффициенты, в нашем случае будем вычислять 10 коэффициентов (значение 10 опять-таки получено опытным путем и соответствует свойству \$coefCount в коде).

Реализуем главную функцию **getMfcc**, которая и будет находить мел-кепстральные коэффициенты для массива амплитуд.

Вначале последовательно вызываем функции **setCenters**, **setSmp** и **setHik** (все указанные функции будут реализованы далее). Далее в цикле будут вычисляться MFCC для каждого фрейма.

Вначале делается дискретное преобразование Фурье (ДПФ) для фрейма (функцией **Fourier**).

$$ReP_n[k] = \frac{2}{N}\sum_{i=0}^{N-1} S_n[i]\cos(\frac{2\pi ki}{N}), k = 0, 1, \ldots, N-1$$

$$ImP_n[k] = -\frac{2}{N}\sum_{i=0}^{N-1} S_n[i]\sin(\frac{2\pi ki}{N}), k = 0, 1, \ldots, N-1$$

Здесь N — число сэмплов в фрейме, $P_n[k]$ — значение дискретного преобразования Фурье, $S_n[i]$ — значение амплитуды сэмпла (n — номер фрейма, i — индекс сэмпла во фрейме).

Далее вызываем функцию **setXn**, которая вычисляет энергию текущего фрейма Xn. Последнее — вызов функции **getCepstral**, которая собственно и вычисляет MFCC-коэффициенты.

```
public function getMfcc() {
    $frameiD = 0;
    $this->setCenters();
    $this->setSmp();
    $this->setHik();
    $this->Cn = [];
    for ($i = 0; $i < $this->fileSize; $i = $i + (int)(
        self::$framesize / 2);) {
        if ($i + self::$framesize > $this->fileSize)
            break;
        // Creating frame
        $frame = [];
        for ($j = 0; $j < self::$framesize; $j++) {
            $frame[$j] = $this->file[$i+$j];
        }

        // Discret Fourier Transformation
        $this->Pn = [];
        for ($k = 0; $k < self::$framesize; $k++) {
            $this->Pn[] = $this->Fourier($frame, $k);
        }

        $this->setXn($this->Pn);
        $this->getCepstral($this->Xn, $frameiD);

        $frameiD++;
    }
    // Clearing memory
    $this->file = [];
    $this->Pn = [];
    $this->scaleCenter = [];
    $this->Fsmp = [];
    $this->Xn = [];
    $this->_Hik = [];
}
```

Реализуем функцию **Fourier**, вычисляющую дискретное преобразование Фурье. Обнуляются переменные и осуществляется проход по циклу, высчитывая сумму. Аргументами функции являются input — текущий фрейм, k — индекс элемента фрейма. Вычисляется действительная и мнимая часть преобразования Фурье. Возвращается сумма квадратов действительной и мнимой части.

```
public function Fourier($input, $k) {
    $re = 0;
```

```
    $im = 0;
    for ($i = 0; $i < self::$framesize; $i++) {
        $re += $input[$i] * cos(2*M_PI*$k*($i) / self::
            $framesize);
        $im += $input[$i] * sin(2*M_PI*$k*($i) / self::
            $framesize);
    }
    $re = (2 / self::$framesize) * $re;
    $im = -(2 / self::$framesize) * $im;
    return $re*$re + $im*$im;
}
```

После получения спектра нужно расположить его на мел-шкале. Для этого мы используем фильтры, равномерно расположенные на мел-оси. Необходима функция, которая равномерно расположит эти фильтры. Поскольку вычисления одинаковы для всех фреймов, они будут вынесены за пределы цикла (функция **setCenters**).

Нужно задать количество фильтров coefCount, которое должно равняться количеству MFCC-коэффициентов (т. е. в нашем случае 10), а также начальную частоту flStart и конечную частоту flEnd, которые соответствуют нижней и верхней границе звуковых частот, воспринимаемых человеческим ухом (300 и 8000 Гц соответственно). Далее понадобится перевести частоты в мелы.

Реализуем функцию **herz2mel** перевода из Герц в Мелы. Аргументом функции является величина в Герцах. Перевод происходит по следующей формуле:

$$\hat{f}_{mel}(f_{hz}) = 1127ln(1 + \frac{f_{hz}}{700})$$

Реализуем функцию **mel2herz** перевода из Мелов в Герцы. Аргументом функции является величина в Мелах. Перевод величин происходит по следующей формуле:

$$\hat{f}_{hz}(f_{mel}) = 700(e^{\frac{f_{mel}}{1127}} - 1)$$

Реализуем функцию **setCenters**, находящей центральные частоты треугольных фильтров. Создается массив, хранящий центральные частоты. Начало и конец выбранного диапазона частот переводятся в мелы. Далее происходит задание центров частот в мел-шкале. Напоминаем, в нашем случае начало flStart = 300, конец flEnd =

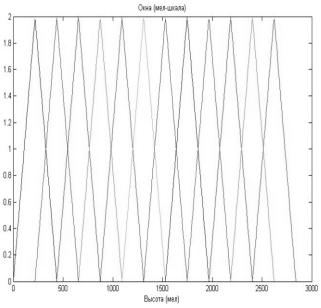

Рис. 32: Мел-шкала.

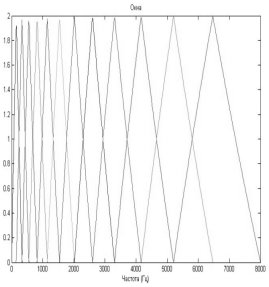

Рис. 33: Частота (Герц).

8000, coefCount = 10. Для удобства использует вспомогательная переменная filterCount = 12 (coefCount + 2). На мел-шкале отрезок $[flStart, flEnd]$ разбивается на $filterCount - 1$ равных непересекающихся подотрезков $[f_j, f_{j+1}], j = 1, 2, \ldots, filterCount - 1$ длины $len = \frac{flEnd - flStart}{filterCount - 1}$. Далее находятся их центры

$$C[i] = flStart + i \cdot len, i = 1, 2, \ldots, filterCount$$

и переводятся в шкалу Герц. Затем необходимо вычислить центральные частоты треугольных фильтров

$$C[i] = \hat{f}_{hz}(C[i]), i = 1, 2, \ldots, filterCount$$

Если перевести этот график (рис. 32) в частотную шкалу, можно увидеть такую картину (рис. 33).

На этом графике заметно, что фильтры «собираются» в области низких частот, обеспечивая более высокое «разрешение» там, где оно необходимо для распознавания.

```
public function setCenters() {
    $this->scaleCenter = [];
    $fl_l = $this->herz2mel(self::$flStart);
    $fl_h = $this->herz2mel(self::$flEnd);
    $len = ($fl_h-$fl_l) / (self::$filterCount-1);
    for ($i = 0; $i < self::$filterCount; $i++) {
        $this->scaleCenter[] = $this->mel2herz($fl_l+$len*
            $i);
    }
}
```

Для корректной работы центры треугольных фильтров переводятся из Гц в индексы массива значений дискретного преобразования Фурье. Для этого необходимо реализовать еще две функции.

Реализуем функцию **setSmp**, где scaleCentre — массив центров, полученный функцией setCenters, sampleRate — исходная частота дискретизации.

$$f_{smp}[i] = \frac{M}{F_s} C[i], i = 1, 2, \ldots, filterCount$$

F_s — частота дискретизации исходного сигнала.

Перемножением векторов спектра сигнала и оконной функции найдем энергию сигнала, которая попадает в каждый из фильтров анализа.

```
public function setSmp() {
    $this->Fsmp = [];
    for($i = 0; $i < self::$filterCount; $i++) {
        $this->Fsmp[] = floor(((self::$framesize + 1) *
            $this->scaleCenter[$i]) / $this->sampleRate);
    }
}
```

Теперь, используя полученные функцией **setSmp** коэффициенты, нужно получить значения, на которые будет домножаться результат дискретного преобразования Фурье для фрейма. Для этого реализуем функцию **setHik**. Смысл функции — «пропускание» числа через треугольный фильтр, полученный на предыдущем шаге. Это соответствует следующей формуле (рис. 34), где f_{smp} — массив, полученный функцией **setSmp**.

$$H_i[k] = \begin{cases} 0 & , \quad k < f_{smp}[i-1] \\ \frac{(k - f_{smp}[i-1])}{f_{smp}[i] - f_{smp}[i-1]} & , \quad f_{smp}[i-1] \le k \le f_{smp}[i] \\ \frac{(f_{smp}[i+1] - k)}{f_{smp}[i+1] - f_{smp}[i]} & , \quad f_{smp}[i] \le k \le f_{smp}[i+1] \\ 0 & , \quad k > f_{smp}[i+1] \end{cases}$$

Рис. 34

```
public function setHik() {
    $this->_Hik = [];
    $val = 0;
    for ($i = 1; $i <= self::$coefCount; $i++) {
        for ($k = 0; $k < self::$framesize; $k++) {
            if ($k < $this->Fsmp[$i-1])
                $val = 0;
            if ($k >= $this->Fsmp[$i-1] && $k <= $this->
                Fsmp[$i])
                $val = ($k - $this->Fsmp[$i-1]) / ($this->
                    Fsmp[$i] - $this->Fsmp[$i-1]);
            if ($k <= $this->Fsmp[$i+1] && $k >= $this->
                Fsmp[$i])
                $val = ($this->Fsmp[$i+1] - $k) / ($this->
                    Fsmp[$i+1] - $this->Fsmp[$i]);
            if ($k > $this->Fsmp[$i+1])
                $val = 0;
            $this->_Hik[$i - 1][] = $val;
        }
    }
}
```

Теперь реализуем функцию **setXn** получения фильтров отсчета спектральной плотности мощности сигнала, которая умножает его на специальный фильтр _Hik, полученный функцией **setHik**, после чего происходит взятие логарифма от результата.

$$X_n[i] = \sum_{k=1}^{coefCount} |P_n[k]|^2 H_i[k], i = 1, 2, \ldots, coefCount$$

$$X_n[i] = ln(X_n[i]), i = 1, 2, \ldots, coefCount$$

Зануляются переменные суммы и массив фильтров отсчета. Проходим в цикле по количеству фильтров и по числу элементов во фрейме. Значение $P_n[k]$ умножаем на соответствующий фильтр $H_i[k]$. От полученной суммы берется логарифм.

```
public function setXn($Pn) {
    $summ = 0.0;
    $this->Xn = [];
    for ($i = 1; $i <= self::$coefCount; $i++) {
        $summ = 0.0;
        for ($k = 0; $k < self::$framesize; $k++) {
            $summ += $Pn[$k] * $this->_Hik[$i - 1][$k];
        }
        $this->Xn[] = log($summ);
    }
}
```

Реализуем функцию **getCepstral** (получение кепстральных коэффициентов). После высчитывания коэффициентов $X_n[k]$ функцией **setXn** берется дискретное косинусное преобразование.

$$C_n[j] = \sum_{k=1}^{coefCount} X_n[k] cos(j(k - \frac{1}{2})\frac{\pi}{coefCount}),$$

$i = 1, 2, \ldots, coefCount$, $j = 1, 2, \ldots, coefCount$, $C_n[j]$ — массив кепстральных коэффициентов.

Необходимо пройти в цикле по количеству коэффициентов coefCount. Далее высчитывается дискретное косинусное преобразование Фурье.

```
public function getCepstral($Xn, $frameiD) {
    $summ = 0;
    $this->Cn[$frameiD] = [];
    for ($j = 0; $j < self::$coefCount; $j++) {
        $summ = 0;
        for ($k = 0; $k < self::$coefCount; $k++) {
            $summ += $Xn[$k] * cos($j * (2 * $k + 1) *
                M_PI / (self::$coefCount * 2));
        }
        $this->Cn[$frameiD][] = $summ;
    }
}
```

Таким образом, после вызова функции **getMfcc** значения MFCC будут сохранены в массив $Cn объекта класса MFCC.

5.5. Вычисление расстояния между векторами mfcc-коэффициентов

Итак, в предыдущем разделе были получены MFCC-коэффициенты музыкального файла. Теперь необходимо научиться оценивать различие между массивами MFCC-коэффициентов двух файлов, т. е. реализовать алгоритм, по смыслу вычисляющий «расстояние» между музыкальными файлами.

Напомним, что MFCC-коэффициенты музыкального файла представляют собой массив, каждый элемент которого в свою очередь является массивом — набором MFCC-коэффициентов соответствующего фрейма. Таким образом, для начала нужно выбрать способ вычисления расстояния между MFCC-коэффициентами двух фреймов. Поскольку размерности векторов MFCC-коэффициентов фреймов одинаковы (и равны 10 - см. предыдущий раздел), то можно воспользоваться любым известным способом вычисления расстояния между векторами. Будем использовать классическое Евклидово расстояние.

Для удобства не будем создавать новый класс, а добавим новые функции в класс MFCC, написанный в предыдущем разделе.

Реализуем функцию **getDis**, вычисляющую расстояние между двумя векторами. Vec1 и vec2 — вектора, offset1 и offset2 — смещения нужных данных относительно начал vec1 и vec2, len — длины сравниваемых векторов.

```
public function getDis($vec1, $offset1, $vec2, $offset2,
    $len){
    $dis = 0;
    for($i = 0; $i < $len; $i++){
        $dis += ($vec1[$offset1 + $i] - $vec2[$offset2 +
            $i]) * ($vec1[$offset1 + $i] - $vec2[$offset2
            + $i]);
    }
    return sqrt($dis);
}
```

Осталось определить расстояние между массивами MFCC-коэффициентов для музыкальных файлов в целом. Самый простой способ — найти сумму расстояний между соответствующими фреймами двух файлов. Однако, этот способ обладает рядом недостатков. Во-первых, количество фреймов у двух файлов может быть различным. А во-вторых, если один файл соответствует другому, но со смещением

по времени, то вычисленное расстояние не будет отражать различия между файлами — сравниваться будут фреймы, соответствующие разным частям музыкального файла.

Поэтому будем использовать другой метод. По сути перед нами стоит задача сравнения последовательностей разной длины (имеются в виду последовательности MFCC-коэффициентов фреймов). Для решения этой задачи существуют разные алгоритмы. Мы будем использовать алгоритм, основанный на методе динамического программирования.

Для удобства скопируем массив MFCC-коэффициентов всех фреймов в одномерный массив. Реализуем функцию **getVector**, которая будет это выполнять.

```
public function getVector() {
    $vector = [];
    foreach ($this->Cn as $vec) {
        foreach($vec as $elem) {
            $vector[] = $elem;
        }
    }
    return $vector;
}
```

Вычислим матрицу расстояний между фреймами двух музыкальных файлов. Далее будем искать путь из левого верхнего элемента матрицы в правый нижний с наименьшей суммой элементов. Из каждого элемента матрицы путь может идти вправо, вниз или вправо и вниз одновременно. Это — классическая задача динамического программирования. Реализуем соответствующий алгоритм. Для экономии памяти не будем хранить матрицу целиком, вместо этого будем вычислять ее элементы в процессе работы алгоритма.

Реализуем функцию **chisqr**.

```
public function chisqr($subtractArr = []) {
    $vec = $this->getVector();
    if (count($vec) < 1 || count($subtractArr) < 1)
        throw new Exception("Array can not be empty");
    $dist = 0;
```

```
    // dynamic programming algorithm
    $dim1 = count($vec) / self::$coefCount;
    $dim2 = count($subtractArr) / self::$coefCount;
    $dp = [];
    for($j = 0; $j < $dim2; $j++){
        $dp[0][$j] = $this->getDis($vec, 0, $subtractArr,
            $j * self::$coefCount, self::$coefCount);
    }
    for($i = 1; $i < $dim1; $i++){
        for($j = 0; $j < $dim2; $j++){
            $dis = $this->getDis($vec, $i * self::
                $coefCount, $subtractArr, $j * self::
                $coefCount, self::$coefCount);
            $dp[1][$j] = $dp[0][$j];
            if ($j > 0){
                if ($dp[0][$j - 1] < $dp[1][$j]){
                    $dp[1][$j] = $dp[0][$j - 1];
                }
                if ($dp[1][$j - 1] < $dp[1][$j]){
                    $dp[1][$j] = $dp[1][$j - 1];
                }
            }
            $dp[1][$j] += $dis;
        }
        for($j = 0; $j < $dim2; $j++){
            $dp[0][$j] = $dp[1][$j];
        }
    }
    $dis = $dp[0][$dim2 - 1];
    return $dis;
}
```

Таким образом, мы умеем количественно оценивать различия между музыкальными файлами. Осталось реализовать пользовательский интерфейс и работу с базой данных, где будет храниться информация о музыкальных файлах из начального множества. Это будет рассмотрено позднее, в разделе «Интерфейс», одновременно с пользовательским интерфейсом для следующей задачи — классификация изображений.

5.6. Классификация музыкальных файлов

Итак, осталось «привязать» реализованные в подзадачах 1–3 классы к пользовательскому интерфейсу и базе данных. Ранее были со-

зданы 3 файла — create_sample.php, delete_sample.php и test_file.php. Допишем в них недостающий функционал.

Предварительно необходимо создать несколько вспомогательных функций, которые понадобятся в различных случаях. Для этого создадим файл functions.php.

Реализуем функцию генерации случайной строки длины $length. Она понадобится в дальнейшем для генерации названия загружаемых файлов для сохранения на жесткий диск.

```
function generateRandomString($length = 10) {
    $characters = '0123456789abcdefghijklmnopqrstuvwxyz
        ABCDEFGHIJKLMNOPQRSTUVWXYZ';
    $charactersLength = strlen($characters);
    $randomString = '';
    for ($i = 0; $i < $length; $i++) {
        $randomString .= $characters[rand(0,
            $charactersLength - 1)];
    }
    return $randomString;
}
```

Реализуем функцию currentUrl, которая возвращает протокол и доменное имя сервера.

```
function currentURL() {
    $pageURL = 'http';
    ($_SERVER["SERVER_PORT"]===443) ? $pageURL.="s" : '';
    $pageURL .= "://";
    if ($_SERVER["SERVER_PORT"] != "80") {
        $pageURL .= $_SERVER["SERVER_NAME"].":".$_SERVER["
            SERVER_PORT"];
    } else {
        $pageURL .= $_SERVER["SERVER_NAME"];
    }
    return $pageURL;
}
```

Перейдем к созданию музыкальных файлов-образцов. Будем писать код в файле create_sample.php.

В самом начале увеличиваем количество памяти, которое может быть выделено для приложения, и подключаем внешние файлы с подключением к БД, с дополнительными функциями, с классом для парсинга WAV файлов и с классом MFCC.

```
ini_set('memory_limit', '2048M');
require('connect_db.php');
require('functions.php');
require('WavParse.php');
require('MFCC.php');
```

Далее производится сохранение загруженного файла на жесткий диск. Сначала проверяется файл на наличие, затем определяются такие параметры, как директория для загрузки, название и расширение файла. После этого файла сохраняется на жесткий диск.

```
if (!isset($_FILES['file'])) {
    echo 'No file.';
    exit;
}

$dir = dirname(__FILE__).'/files/';
$ext = pathinfo($_FILES['file']['name'],
   PATHINFO_EXTENSION);
$filename = generateRandomString().'.'.$ext;
$filePath = $dir.$filename;
move_uploaded_file($_FILES['file']['tmp_name'],$filePath);
```

Далее создается массив, который будет содержать результат извлечения вектора из изображения или музыкального файла.

```
$result = array(
    'type' => $_POST['sample_type'],
    'name' => $_POST['name'],
    'vector' => '',
    'file_path' => $filePath,
    'file_url' => '/files/'.$filename
);
```

При загрузке файла пользователь указывает тип файла. В зависимости от того, является ли тип файла изображением или музыкальным файлом, извлекаем вектор файла одним или другим способом. В данный момент реализуем работу только с музыкальными файлами.

```
switch ($_POST['sample_type']) {
    case 'img':
        // TODO
```

```
    case 'music':
        $sampleArr = new WavParse($filePath);
        $mfcc = new MFCC($sampleArr->wav["subchunk2"]["
            data"], $sampleArr->wav["subchunk1"]["
            samplerate"]);
        $mfcc->getMfcc();
        $result['vector'] = serialize($mfcc->getVector());
        break;
}
```

Далее сохраняем результат в БД и перенаправляем обратно на страницу интерфейса работы с приложением.

```
mysqli_query($connect,
    'INSERT INTO samples (`type`, `name`, `vector`, `
        file_path`, `file_url`)
    VALUES (
        "'.mysqli_real_escape_string($connect, $result['
            type']).'",
        "'.mysqli_real_escape_string($connect, $result['
            name']).'",
        "'.mysqli_real_escape_string($connect, $result['
            vector']).'",
        "'.mysqli_real_escape_string($connect, $result['
            file_path']).'",
        "'.mysqli_real_escape_string($connect, $result['
            file_url']).'"
    )');

header('Location: '.currentURL().'/interface.php');
exit;
```

Перейдем к файлу delete_sample.php, который будет использоваться для удаления класса из БД. Вначале подключим файлы connect_db.php и functions.php. Затем делается запрос к БД на удаление класса и производится перенаправление экрана работы с приложением.

```
require('connect_db.php');
require('functions.php');

mysqli_query($connect, 'DELETE FROM samples WHERE id='.
    mysqli_real_escape_string($connect, $_GET['id']));
header('Location: '.currentURL().'/interface.php');
exit;
```

Наконец, займемся ключевой частью — проверкой файла на сход-

ство с классами. Будем работать в файле test_file.php. Как и в случае с файлом create_sample.php, увеличиваем возможное количество памяти и подключаем необходимые файлы.

```
ini_set('memory_limit', '2048M');
require('connect_db.php');
require('functions.php');
require('WavParse.php');
require('MFCC.php');
```

Загружаем из БД векторы классифицированных музыкальных файлов или изображений в зависимости от того, какое значение указал пользователь при загрузке файла на проверку.

```
$vectorsRes = mysqli_query($connect, 'SELECT * FROM
    samples WHERE `type`="'.mysqli_real_escape_string(
    $connect, $_POST['sample_type']).'"');
if (!$vectorsRes) {
    echo 'Can not load vectors.';
    exit;
}
```

Далее сохраняем файл на жесткий диск.

```
$dir = dirname(__FILE__).'/files/';
$ext = pathinfo($_FILES['file']['name'],
    PATHINFO_EXTENSION);
$filename = generateRandomString().'.'.$ext;
$filePath = $dir.$filename;
move_uploaded_file($_FILES['file']['tmp_name'],$filePath);
```

Далее, используя реализованные функции для подсчета расстояния между векторами, находим класс с ближайшим вектором.

```
$vectors = array();
while ($vector = mysqli_fetch_assoc($vectorsRes)) {
    $vectors[] = $vector;
}

$result = array();
$additionalData = array();
```

```
switch ($_POST['sample_type']) {
    case 'img':
       // TODO
    case 'music':
        $additionalData['filename'] = $_FILES['file']['
            name'];
        $sampleArr = new WavParse($filePath);
        $mfcc = new MFCC($sampleArr->wav["subchunk2"]["
            data"], $sampleArr->wav["subchunk1"]["
            samplerate"]);
        $mfcc->getMfcc();
        $min = 1000000000;
        $class = null;
        foreach($vectors as $vector) {
            $res = $mfcc->chisqr(unserialize($vector['
                vector']));
            unset($vector['vector']);
            unset($vector['file_path']);
            $result[] = array(
                'sample' => $vector,
                'distance' => $res
            );
            if($res < $min) {
                $min = $res;
                $class = $vector;
            }
        }
        break;
}
```

Выводим результат сравнения загруженного файла со всем классами, самый ближайший класс и расстояние до него, название загруженного музыкального файла и ссылку на загруженный файл.

```
   echo json_encode(array(
       'result' => $result,
       'min' => array(
           'sample' => $class,
           'distance' => $min
       ),
       'additionalData' => $additionalData,
       'fileUrl' => '/files/'.$filename
   ));
   exit;
```

Таким образом, задача классификации музыкальных файлов полностью решена.

5.7. Первичная обработка изображения

Переходим к задаче классификации изображений. Для начала необходимо научиться извлекать из файла изображения массив составляющих его пикселей.

RGB (аббревиатура английских слов Red, Green, Blue — красный, зеленый, синий) — аддитивная цветовая модель, как правило, описывающая способ синтеза цвета для цветовоспроизведения. Далее мы будем рассматривать только RGB-файлы, в которых на каждый пиксель отводится 3 байта памяти (по одному байту на красный, зеленый и синий цвета).

Создадим вспомогательный класс Pixel, у которого будут три свойства — числа, соответствующие красному, зеленому и синему цветам.

```
class Pixel {
    public $R;
    public $G;
    public $B;

    public function __construct($r, $g, $b) {
        $this->R = $r;
        $this->G = $g;
        $this->B = $b;
    }
}
```

Создадим класс ImageObj, который позволит работать с изображением как с массивом пикселей [58].

Реализуем конструктор **__construct**, который будет считывать изображения и определять их размеры. Функция **imagecreatefromjpeg** возвращает идентификатор изображения, полученного из файла с заданным именем. Устанавливается количество столбцов width и строк height в массиве, Затем вызывается функция **setPixArray**, которая возвращает изображение в виде массива пикселей.

```
class ImageObj {
    public $url;
    public $ext;
    public $image;
    public $width;
```

```
    public $height;
    public $bmp = [];
    public $gs = [];

    function __construct($str) {
        $this->url = $str;
        $this->ext = pathinfo($this->url,
            PATHINFO_EXTENSION);
        switch ($this->ext) {
            case 'jpg':
            case 'jpeg':
                $this->image = imagecreatefromjpeg($this->
                    url);
                break;
            case 'png':
                $this->image = imagecreatefrompng($this->
                    url);
                break;
            default:
                throw new Exception("Incorrect file format
                    .");
        }
        $this->width = imagesx($this->image);
        $this->height = imagesy($this->image);
        self::setPixArray();
    }

    private function setPixArray() {
        $r; $g; $b;
        for ($y = 0; $y < $this->height; $y++) {
            for ($x = 0; $x < $this->width; $x++) {
                $rgb = imagecolorat($this->image,$x,$y);
                $r = ($rgb >> 16) & 0xFF;
                $g = ($rgb >> 8) & 0xFF;
                $b = $rgb & 0xFF;
                $pixel = new Pixel($r, $g, $b);
                $this->bmp[$y][$x] = $pixel;
            }
        }
    }
    ...
}
```

Работа с пикселями как с тройкой чисел достаточно неудобна. Поэтому преобразуем изображение в оттенки серого (grayscale). Такой подход приведет к существенному ускорению работы программы без

значительного снижения качества классификации. Яркость пикселя grayscale-изображения рассчитывается по следующей формуле:

$Y = 0.299R + 0.587G + 0.114B$

Реализуем функцию **getGreyScale** получения массива GrayScale (оттенков серого). Используя заполненный функцией **setPixArray** массив пикселей bmp, вычислим оттенки серого по формуле, указанной выше.

```
public function getGreyScale() {
    $arr = [];
    for ($y = 0; $y < $this->height; $y++) {
        for ($x = 0; $x < $this->width; $x++) {
            $arr[$y][$x] = (int)(0.299 * $this->bmp[$y][$x
                ]->R + 0.587 * $this->bmp[$y][$x]->G +
                0.114 * $this->bmp[$y][$x]->B);
            if ($arr[$y][$x] > 255){
                $arr[$y][$x] = 255;
            }
        }
    }
    return $arr;
}
```

Для наглядности реализуем функцию **getGrayImage** получения черно-белой картинки из RGB-изображения. Создаем новую картинку с теми же размерами, и в качестве RGB координат запишем «серость». Функция imagecolorallocate(resource $image,int $red,int $green,int $blue) возвращает идентификатор цвета в соответствии с заданными RGB-координатами.

```
public function getGrayImage() {
    $image = imagecreatetruecolor($this->width, $this->
        height);
    $arr = $this->getGreyScale();
    echo $this->width."::".$this->height;
    for ($i = 0; $i < $this->width; $i++) {
        for ($j = 0; $j<$this->height; $j++) {
           $val=imagecolorallocate($image, $arr[$j][$i],
               $arr[$j][$i], $arr[$j][$i]);
           imagesetpixel($image, $i, $j, $val);
        }
    }
    imagejpeg($image, Yii::getAlias('@image').
        DIRECTORY_SEPARATOR.'simpletext.jpg');
}
```

5.8. Составление вектора признаков по изображению

Существует много подходов составления вектора признаков для изображения. Мы будем использовать алгоритм HOG (Histogram of Oriented Gradients), который зарекомендовал себя как достаточно простой и в то же время достаточно эффективный для задач классификации изображений. Основной идеей алгоритма является допущение, что внешний вид и форма объекта на участке изображения могут быть описаны распределением градиентов интенсивности или направлением краев. Реализация этих дескрипторов может быть произведена путем разделения изображения на маленькие связные области, именуемые ячейками, и расчетом для каждой ячейки гистограммы направлений градиентов или направлений краев для пикселей, находящихся внутри ячейки. Комбинация этих гистограмм и является дескриптором. Для увеличения точности локальные гистограммы подвергаются нормализации. С этой целью вычисляется мера интенсивности на большем фрагменте изображения, который называется блоком, и полученное значение используется для нормализации. Нормализованные дескрипторы обладают лучшей инвариантностью по отношению к освещению.

Алгоритм HOG имеет несколько преимуществ над другими дескрипторами. Поскольку HOG работает локально, метод поддерживает инвариантность геометрических и фотометрических преобразований, за исключением ориентации объекта. Подобные изменения появятся только в больших фрагментах изображения.

Создадим класс Hog [59]. Конструктор класса будет получать параметр $img типа ImageObj и заполнять по нему нужные свойства.

```
class Hog {
    const HistogramCountOfChannel = 20; // Number of
        columns in histogram
    const cellSize = 8;
    public  $vector = [];
    public $gs = [];
    private $width;
    private $height;
    public $str="";
    public function __construct(ImageObj $img) {
        $this->gs = $img->getGreyScale();
        $this->height = count($this->gs);
```

```
        $this->width = count($this->gs[0]);
    }
    ...
}
```

В процессе работы алгоритма изображение разбивается на блоки размера 8x8 пикселей, для каждого блока вычисляется гистограмма ориентированных градиентов (массив размера 20), полученный массив размера $(height/8) \cdot (width/8) \cdot 20$ будет выполнять роль вектора признаков. Размер блока (8) объявлен в константе cellSize, размер гистограммы (20) — в константе HistogramCountOfChannel (см. листинг выше).

Рассмотрим подробно алгоритм получения гистограммы ориентированных градиентов для блока пикселей.

Вначале для каждого пикселя вычисляются два числовых параметра — так называемые *свертки с фильтрами Собеля*. Одна из сверток соответствует *горизонтальному оператору Собеля*, вторая — *вертикальному*. Обе свертки вычисляются по формулам, оперирующим со значениями в соседних пикселях к текущему пикселю (соседних по стороне или углу). Горизонтальный оператор Собеля вычисляет величину, по смыслу соответствующую производной яркости изображения в горизонтальном направлении. Аналогично, вертикальный оператор Собеля вычисляет величину, соответствующую производной яркости изображения в вертикальном направлении.

Реализуем три функции — получение соседних пикселей к текущему, применение горизонтального оператора Собеля и вертикального оператора Собеля.

Получение соседних клеток — несложная процедура. Единственный нюанс — нужно аккуратно рассмотреть граничные пиксели изображения, чтобы не получить ошибку.

```
 private function getCell($x, $y) {
     $cpyAr=[];
     $a=false;
     for($i = -1; $i <= 1; $i++){
         for($j = -1; $j <= 1; $j++){
             $indy = $y + $i;
             if ($indy < 0){
                 $indy = 0;
```

```
            } elseif ($indy >= $this->height){
                $indy = $this->height - 1;
            }
            $indx = $x + $j;
            if ($indx < 0){
                $indx = 0;
            } elseif ($indx >= $this->width){
                $indx = $this->width - 1;
            }
            $cpyAr[$i + 1][$j + 1] = $this->gs[$indy][
                $indx];
        }
    }
    return $cpyAr;
}
```

Работа горизонтального оператора Собеля проиллюстрирована на рисунке 35.

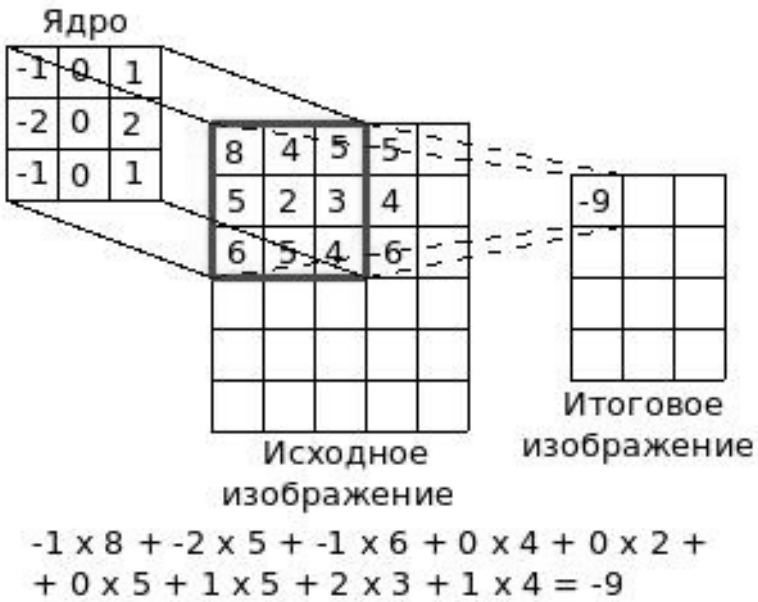

Рис. 35: Пример работы оператора Собеля.

Реализуем функцию **getGx**, соответствующую горизонтальному оператору Собеля.

```
private function getGx($x, $y) {
    $summ = 0;
    $operatorSobelH = [[-1,0,1],[-2,0,2],[-1,0,1]];
    $cpyAr = [];
    $cpyAr = $this->getCell($x, $y);
    for($i = 0; $i < count($operatorSobelH); $i++) {
        for($j = 0; $j < count($operatorSobelH); $j++) {
            $summ += $cpyAr[$i][$j] * $operatorSobelH[$i][
                $j];
        }
    }
    return $summ;
}
```

Аналогично реализуем функцию **getGy**, соответствующую вертикальному оператору Собеля.

```
private function getGy($x, $y) {
    $summ = 0;
    $operatorSobelV = [[-1,-2,-1],[0,0,0],[1,2,1]];
    $cpyAr = [];
    $cpyAr = $this->getCell($x, $y);
    for($i = 0; $i < count($operatorSobelV); $i++) {
        for($j = 0; $j < count($operatorSobelV); $j++) {
            $summ += $cpyAr[$i][$j] * $operatorSobelV[$i][
                $j];
        }
    }
    return $summ;
}
```

На рисунках 36–41 показаны результаты применения операторов Собеля к изображению (в качестве «серости» пикселей записывались нормированные значения операторов Собеля).

Вектор из двух числовых значений, полученных операторами Собеля, по смыслу является градиентом яркости изображения. Вектор-градиент обладает своим абсолютным значением и аргументом (углом).

После вычисления всех (8x8=64) градиентов пикселей будем собственно получать гистограмму ориентированных градиентов. Возьмем отрезок $[0, 2\pi]$ и разделим его на несколько равных частей (обычно используют от 8 до 32 частей, мы будем использовать 20 — значение константы HistogramCountOfChannel). Далее для каждого градиента найдем часть отрезка, содержащую его аргумент, и увеличим ее «вес»

Рис. 36
Картинка в первичном виде.

Рис. 37
Картинка после обработки.

Рис. 38
Картинка в первичном виде.

Рис. 39: Применение горизонтального оператора Собеля.

Рис. 40: Применение вертикального оператора Собеля.

Рис. 41: Применение оператора Собеля в обоих направлениях.

на абсолютное значение аргумента. Таким образом будет получена гистограмма ориентированных градиентов для блока пикселей. После этого для уменьшения чувствительности к изменению контрастности изображения выполним нормирование полученной гистограммы.

Реализуем три функции: нахождение аргумента вектора, вычисление абсолютного значения вектора и нормирование вектора. Для удобства будем оперировать с величинами углов в градусах, а не в радианах.

```
private function getDirection($Gx, $Gy) {
    $val = atan2($Gy, $Gx);
    if ($val < 0){
        $val += 2 * M_PI;
    }
    return $val * (180 / M_PI);
}

private function getAbsValuegOfGradient($Gx, $Gy) {
    $a = sqrt($Gx * $Gx + $Gy * $Gy);
    return $a;
}

public function normalize(&$arr) {
    $summ = 0;
    $norm = 0;
    for ($i = 0; $i < count($arr); $i++) {
        $summ += $arr[$i] * $arr[$i];
    }
    $norm = sqrt($summ);
    if ($norm > 1e-12) {
        for ($i = 0; $i < count($arr); $i++) {
            $arr[$i] = $arr[$i] / $norm;
        }
    }
}
```

Комбинируя результаты 6 функций, написанных ранее, реализуем функцию построения гистограммы **getLocalHistogramm**.

```
public function getLocalHistogramm($x, $y) {
    $period = 360 / self::HistogramCountOfChannel;
    $localHis = [];
    $Gx = 0;
    $Gy = 0;
```

```
    for($i = 0; $i < self::HistogramCountOfChannel; $i++)
        $localHis[$i] = 0;
    for($k = 0; $k < self::cellSize; $k++) {
        for($s = 0; $s < self::cellSize; $s++) {
            $Gx = $this->getGx($x+$s, $y+$k);
            $Gy = $this->getGy($x+$s, $y+$k);
            $localHis[(int)$this->getDirection($Gx,$Gy) /
                $period]+= $this->getAbsValuegOfGradient(
                $Gx,$Gy);
      }
    }
    $this->normalize($localHis);
    return $localHis;
}
```

Реализуем теперь основную функцию **getVector**, которая, последовательно вызывая функцию **getLocalHistogramm**, находит для каждого блока пикселей гистограмму и конкатенирует (соединяет) все гистограммы в один массив.

```
public function getVector() {
    for ($y = 0; $y < $this->height-self::cellSize; $y +=
        self::cellSize) {
        for ($x = 0; $x < $this->width-self::cellSize; $x
            += self::cellSize) {
            $localHis = $this->getLocalHistogramm($x,$y);
            for ($s = 0; $s < count($localHis); $s++) {
                $this->vector[] = $localHis[$s];
            }
        }
    }
    return $this->vector;
}
```

5.9. Вычисление расстояния между векторами признаков изображений

Теперь, как и в случае музыкальных файлов, для изображений необходимо выбрать способ вычисления расстояний между векторами признаков. Существует множество различных подходов к решению этой задачи. Мы будем использовать самый простой подход — стандартное покомпонентное вычисление расстояния между векторами с использованием метрики хи-квадрат.

В файл functions.php, содержащий различные вспомогательные функции, допишем функцию нахождения расстояния между векторами по метрике хи-квадрат.

```
function chisqr($vec, $subtractArr) {
    if (isset($vec['vector'])) {
        $vec = unserialize($vec['vector']);
    }

    if (empty($vec) || empty($subtractArr)) {
        throw new Exception("Empty array.");
    }

    $dist = 0;
    for ($i = 0; $i < min(count($vec), count($subtractArr)
        ); $i++) {
        if (abs($vec[$i] + $subtractArr[$i]) >
            0.000000001) {
            $dist += (($vec[$i] - $subtractArr[$i]) * (
                $vec[$i] - $subtractArr[$i])) / ($vec[$i]
                + $subtractArr[$i]);
        }
    }
    return $dist;
}
```

Как и в случае музыки, здесь возникает несколько проблем. Основная из них — возможная несогласованность размеров картинок, которая может привести к тому, что при покомпонентном вычислении расстояния между векторами могут сравниваться не те элементы, которые следует сравнивать. Например, при покомпонентном сравнении изображения 3x2 с изображением 2x3 третий элемент первой картинки будет соответствовать середине левой части изображения, а третий элемент второго изображения - правому верхнему углу. Очевидно, что такое сравнение дает мало информации о реальном различии между изображениями.

Выход из положения — приведение всех изображений к одинаковым размерам путем растяжения или сжатия (и изображений — эталонов, и изображений — тестов). Подробно это будет описано в следующем разделе.

5.10. Классификация изображений

В предыдущем разделе было сказано, что для использования самого простого способа оценки расстояния между векторами признаков изображений их необходимо приводить к одинаковым размерам. Реализуем для этого вспомогательную функцию **resize_image** в файле functions.php. Для удобства реализуем два варианта изменения размеров — приведение к фиксированному размеру и растяжение (сжатие) с сохранением пропорций.

```
function resize_image($file, $w, $h, $crop = false)  {
    list($width, $height) = getimagesize($file);
    $r = $width / $height;
    if ($crop) {
        $newwidth = $w;
        $newheight = $h;
    } else {
        if ($w/$h > $r) {
            $newwidth = $h*$r;
            $newheight = $h;
        } else {
            $newheight = $w/$r;
            $newwidth = $w;
        }
    }
    $src = imagecreatefromjpeg($file);
    $dst = imagecreatetruecolor($newwidth, $newheight);
    imagecopyresized($dst, $src, 0, 0, 0, 0, $newwidth,
        $newheight, $width, $height);
    return $dst;
}
```

Теперь, как и в случае музыкальных файлов, необходимо реализовать функционал по созданию файлов-образцов, удалению файлов-образцов и тестированию приложения на файлах-тестах.

Механизм создания файлов-образцов был реализован в файле create_sample.php. Подключим файлы с классами ImageObj и Hog, затем допишем участок кода, отвечающий за обработку файлов изображений (где мы ранее поставили комментарий // TODO). Приведем код файла целиком.

```
ini_set('memory_limit', '2048M');
require('connect_db.php');
require('functions.php');
require('WavParse.php');
require('MFCC.php');
require('ImageObj.php');
require('Hog.php');

if (!isset($_FILES['file'])) {
    echo 'File wasn't uploaded.';
    exit;
}
$dir = dirname(__FILE__).'/files/';
$ext = pathinfo($_FILES['file']['name'],
   PATHINFO_EXTENSION);
$filename = generateRandomString().'.'.$ext;
$filePath = $dir.$filename;
move_uploaded_file($_FILES['file']['tmp_name'],$filePath);
$result = array(
    'type' => $_POST['sample_type'],
    'name' => $_POST['name'],
    'vector' => '',
    'file_path' => $filePath,
    'file_url' => '/files/'.$filename
);
switch ($_POST['sample_type']) {
        case 'img':
```

Применим функцию **resize_image** к загруженному изображению (получив изображение стандартного размера 200x200), затем с помощью классов ImageObj и Hog получим вектор признаков изображения.

```
        $resizedImg = resize_image($filePath, 200, 200,
            true);
        switch ($ext) {
            case 'jpg':
            case 'jpeg':
                imagejpeg($resizedImg, $filePath);
                break;
            case 'png':
                imagepng($resizedImg, $filePath);
                break;
        }
        $obj = new ImageObj($filePath);
        $hog = new Hog($obj);
```

```
        $result['vector'] = serialize($hog->getVector());
        break;
    case 'music':
        $sampleArr = new WavParse($filePath);
        $mfcc = new MFCC($sampleArr->wav["subchunk2"]["
            data"], $sampleArr->wav["subchunk1"]["
            samplerate"]);
        $mfcc->getMfcc();
        $result['vector']=serialize($mfcc->getVector());
        break;
}
mysqli_query($connect,
    'INSERT INTO samples (`type`, `name`, `vector`, `
        file_path`, `file_url`)
    VALUES (
        "'.mysqli_real_escape_string($connect, $result['
            type']).'",
        "'.mysqli_real_escape_string($connect, $result['
            name']).'",
        "'.mysqli_real_escape_string($connect, $result['
            vector']).'",
        "'.mysqli_real_escape_string($connect, $result['
            file_path']).'",
        "'.mysqli_real_escape_string($connect, $result['
            file_url']).'"
    )');
header('Location: '.currentURL().'/interface.php');
exit;
```

Механизм удаления файлов-образцов, реализованный в файле delete_sample.php, не требует изменений — он одинаково работает с изображениями и музыкальными файлами.

Изменения в файле test_file.php будут аналогичны изменениям в файле create_sample.php. Подключаем классы для работы с изображениями и дописываем код вместо комментария // TODO. Приведем код файла целиком.

```
ini_set('memory_limit', '2048M');
require('connect_db.php');
require('functions.php');
require('WavParse.php');
require('MFCC.php');
require('ImageObj.php');
require('Hog.php');
```

```
$vectorsRes = mysqli_query($connect, 'SELECT * FROM
    samples WHERE `type`="'.mysqli_real_escape_string(
    $connect, $_POST['sample_type']).'"');
if (!$vectorsRes) {
    echo 'Can't get vector from database.';
    exit;
}
$dir = dirname(__FILE__).'/files/';
$ext = pathinfo($_FILES['file']['name'],
    PATHINFO_EXTENSION);
$filename = generateRandomString().'.'.$ext;
$filePath = $dir.$filename;
move_uploaded_file($_FILES['file']['tmp_name'],$filePath);
$vectors = array();
while ($vector = mysqli_fetch_assoc($vectorsRes)) {
    $vectors[] = $vector;
}
$result = array();
$additionalData = array();
switch ($_POST['sample_type']) {
    case 'img':
        $resizedImg = resize_image($filePath, 200, 200,
            true);
        switch ($ext) {
            case 'jpg':
            case 'jpeg':
                imagejpeg($resizedImg, $filePath);
                break;
            case 'png':
                imagepng($resizedImg, $filePath);
                break;
        }
        $obj = new ImageObj($filePath);
        $hog = new Hog($obj);
        $array = $hog->getVector();
        $min = 1000000;
        $class = 0;
        $classes = array();
        foreach($vectors as $vector) {
            $res = chisqr($vector, $array);
            unset($vector['vector']);
            unset($vector['file_path']);
```

```
            $result[] = array(
                'sample' => $vector,
                'distance' => $res);
            if($res < $min) {
                $min = $res;
                $class = $vector;
            }
        }
        break;
    case 'music':
        $additionalData['filename'] = $_FILES['file']['
            name'];
        $sampleArr = new WavParse($filePath);
        $mfcc = new MFCC($sampleArr->wav["subchunk2"]["
            data"], $sampleArr->wav["subchunk1"]["
            samplerate"]);
        $mfcc->getMfcc();
        $min = 1000000000;
        $class = null;
        foreach($vectors as $vector) {
            $res = $mfcc->chisqr(unserialize($vector['
                vector']));
            unset($vector['vector']);
            unset($vector['file_path']);
            $result[] = array(
                'sample' => $vector,
                'distance' => $res);
            if($res < $min) {
                $min = $res;
                $class = $vector;
            }
        }
        break;
}
echo json_encode(array(
    'result' => $result,
    'min' => array(
        'sample' => $class,
        'distance' => $min),
    'additionalData' => $additionalData,
    'fileUrl' => '/files/'.$filename
));
exit;
```

Задача по классификации изображений полностью решена. На рисунке 42 изображен интерфейс полученного приложения.

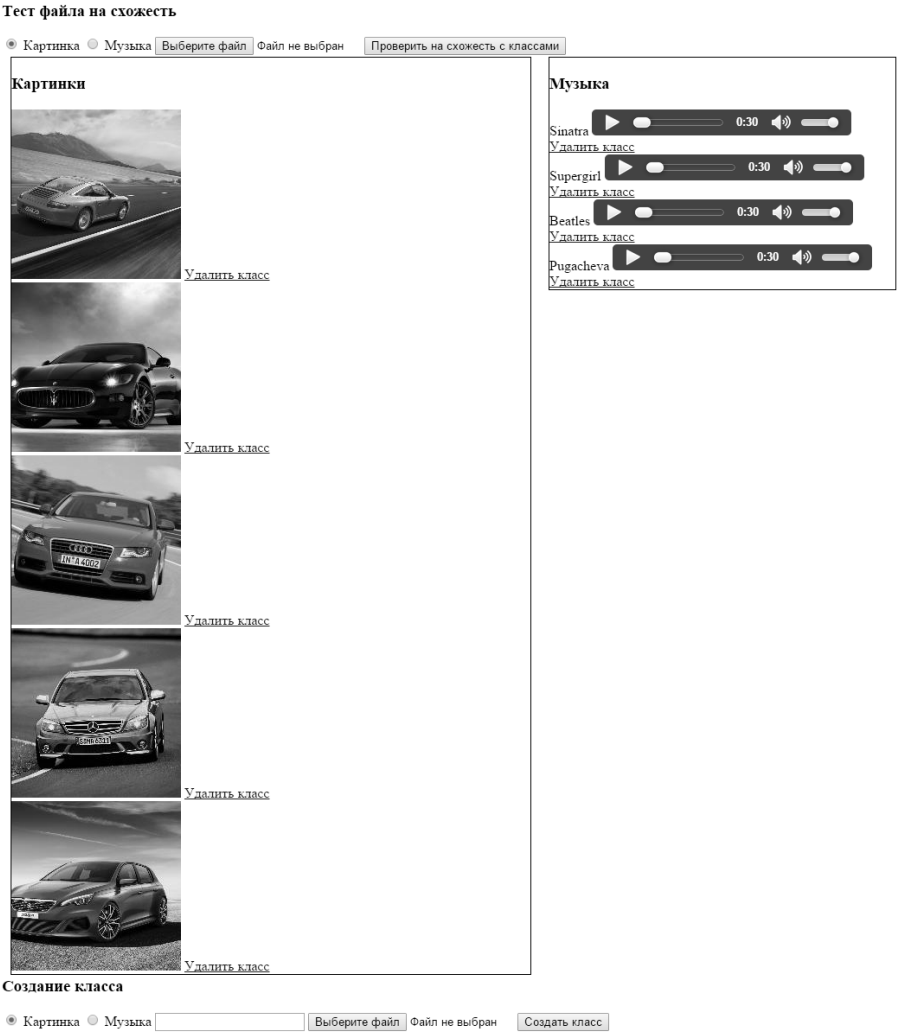

Рис. 42: Интерфейс веб-приложения
классификации изображений и музыки

6. iOS

В данной главе мы напишем мобильное приложение под платформу iOS, которое позволит мерчендайзерам производить посещения торговых точек, фиксировать выкладки, производить аудит цен, а также фиксировать наличие товара на полке и/или на складе.

Для реализации описанного функционала нам понадобится реализовать авторизацию пользователей в приложении, построить пользовательский интерфейс для отображения списка торговых точек и других фрагментов с целью реализации пользовательского взаимодействия с приложением. Далее мы разберем работу с картой, выставление маркеров и обработку действий над ними. Рассмотрим работу с камерой и обработку полученного с нее изображения. В результате мы получим работающее мобильное приложение для платформы iOS с описанным выше функционалом.

6.1. Установка и настройка среды разработки

Подразумевается что Mac OS уже установлена, имеется Macbook, iMac или эмулятор с установленной системой.

Скачиваем и устанавливаем последнюю версию Xcode. Доступна только на Mac.

6.2. Создание проекта

Запускаем Xcode. Слева выбираем «Create a new Xcode project». Далее выбираем «Single View Application». Нажимаем Next. Product name: Merch. Organization name: Dvinem Nauku. Company Idetifier: ru.dvinemnauku. При этом автоматически сформировался Bundle Identifier — ru.dvinemnauku.Merch. Language — Swift. Devices — iPhone. Отмечаем флажок Use Core Data. Нажимаем Next, затем Create. Проект создан.

6.3. Окно авторизации

Начиная с iOS версии 5 и выше в распоряжении разработчиков под iOS появился новый механизм разработки интерфейса приложения — Storyboard. Xcode автоматически создал файл Main.storyboard, в котором мы будем разрабатывать интерфейс нашего приложения и логику

переходов между экранами. Слева, в списке файлов проекта, найдем файл Main.storyboard и откроем его. Справа найдем флажок Use Sizes Classes и снимем его. В появившемся диалоге нажмем Disable Size Classes. Далее, находясь в редакторе Main.storyboard, выберем наш пока единственный View Controller. В меню Editor выбираем Embed in → Navigation Controller. Navigation Controller свяжется с нашей вьюшкой. Нажмем на наш View Controller, перейдем к Attributes Inspector и Title проставим «Профиль».

Перетащим на View Controller 4 кнопки и изменим их Title на «Загрузить данные», «Выгрузить данные», «Торговые точки», «Разлогиниться». Изменим размер всех кнопок так, чтобы текст на кнопке полностью помещался.

Далее создадим еще один View Controller. В библиотеке объектов выбираем View Controller и перетаскиваем на рабочую область.

Нажимаем на кнопку «Разлогиниться» и с зажатым Ctrl тянем ее на только что созданный View Controller. В появившемся диалоге выбираем modal.

Выбираем второй View Controller, открываем Attributes Inspector, задаем Title — «Авторизация».

Слева, в дереве элементов, нажимаем на каталог Merch правой кнопкой мыши, выбираем New File..., затем iOS → Source → Swift File. Далее нажимаем Next. Save as выставляем как AuthViewController, нажимаем Create. В созданный файл добавляем следующий код.

```
import UIKit

class AuthViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
}
```

Переходим к Main.storyboard. Выбираем View Controller «Авторизация». Открываем Identity Inspector, в выпадающем списке Class выбираем AuthViewController, в качестве Module выбираем Merch,

Storyboard ID выставляем как «Authorization».

Далее на View Controller «Авторизация» помещаем Label, два Text Field и Button. Выделяем Label, в Attributes Inspector задаем Text — «Авторизация», у первого текстового поля в Attributes Inspector задаем Placeholder — «Логин», у второго текстового поля задаем Placeholder — «Пароль». У кнопки задаем Title — «Войти».

Нажимаем кнопку Show the Assistant Editor, выбираем кнопку «Войти», зажимаем Ctrl, тянем ее в окно с кодом, отпускаем после метода viewDidLoad(). В появившемся окошке выставляем Connection — Action, Name — «loginBtn», Type — UIButton. Нажимаем Connect.

Выбираем View Controller «Профиль». Переходим к коду (в Assistant Editor или отдельно). Добавляем следующий код после vievDidLoad().

```
@IBAction func done(sender: UIStoryboardSegue) {
}
```

Переходим к Main.storyboard. Выделяем View Controller «Авторизация». Вверху View Controller есть заголовок с тремя иконками, зажимаем Ctrl, нажимаем на первую иконку, тянем курсор мыши к иконке Exit. В появившемся диалоге выбираем done. Открываем Document Outline слева от редактора Storyboard, находим в составе сцены «Авторизации» элемент «Unwind segue to Scene Exit», выделяем его. Справа в Attributes Inspector в поле Identifier пишем «done», ниже в поле Action уже будет выставлено значение «done:».

Теперь выделяем текстовое поле «Логин», зажимаем Ctrl, тянем курсор мыши к Assistant Editor, отпускаем курсор над методом viewDidLoad(), в появившемся окошке в поле Name пишем «loginField». Повторяем то же самое для текстового поля «Пароль», в поле Name пишем «passwordField».

Переходим к файлу AuthViewController.swift. В методе loginBtn() пишем код для проверки логина и пароля на пустоту и вызова метода для проверки их корректности на стороне сервера checkLoginAndPassword(), который мы опишем далее.

```
if loginField.text == "" || passwordField.text == "" {
    let alert:UIAlertView=UIAlertView(title: "Ошибка",
        message: "Логин и пароль не могут быть пустыми",
        delegate: self, cancelButtonTitle: "OK")
    alert.show()
}
else {
    checkLoginAndPassword(loginField.text, password:
        passwordField.text)
}
```

Добавим в класс поле prefs для хранения преференций приложения.

```
var prefs = NSUserDefaults()
```

Переопределим два метода родительского класса viewWillAppear() и viewDidAppear(). В методе viewWillAppear() проинициализируем поле для хранения переменных. А в методе viewDidAppear() проинициализируем значение текстового поля с логином, далее, если пользователь уже залогинен, откроем View Controller профиля.

```
override func viewWillAppear(animated: Bool) {
    self.prefs = NSUserDefaults.standardUserDefaults()
}

override func viewDidAppear(animated: Bool) {
    loginField.text = prefs.valueForKey("USERNAME") as?
        String
    if (prefs.valueForKey("ISLOGGEDIN") != nil) {
        if prefs.integerForKey("ISLOGGEDIN") == 1 {
            performSegueWithIdentifier("done", sender: nil
                )
        }
    }
}
```

Ниже добавим метод checkLoginAndPassword(), в котором проинициализируем класс для отправки запроса на сервер. Выставим заголовки POST запроса и отправим запрос на сервер.

```
func checkLoginAndPassword(login: String, password: String
    ) {
    var post:NSString = "login=\(login)&pwd=\(password)"

    var url = NSURL(string: "http://maps.dvinemnauku.ru/
        Map/login.php?type=json")
```

```
    var postData = post.dataUsingEncoding(
        NSUTF8StringEncoding)
    var request:NSMutableURLRequest = NSMutableURLRequest(
        URL: url!)
    request.HTTPMethod = "POST"
    request.HTTPBody = postData
    request.setValue("application/x-www-form-urlencoded",
        forHTTPHeaderField: "Content-Type")
    request.setValue("application/json",
        forHTTPHeaderField: "Accept")

    var reponseError: NSError?
    var response: NSURLResponse?

    var urlData: NSData? = NSURLConnection.
        sendSynchronousRequest(request, returningResponse
        :&response, error:&reponseError)
}
```

Далее добавим в этот метод код для проверки результата запроса и, в случае ошибки, выведем соответствующее сообщение.

```
  if ( urlData != nil ) {
  // Проверка ответа сервера
  } else {
    var alertView:UIAlertView = UIAlertView(title: "Ошибка а
        вторизации", message: "Невозможно соединиться с серв
        ером", delegate: self, cancelButtonTitle: "OK")
    alertView.show()
  }
```

В блоке if напишем код для проверки результата авторизации. Сначала проверим HTTP статус запроса, затем распарсим полученный JSON и прочитаем из него два параметра: res, который принимает значение 1 в случае успешной авторизации и 0 в случае неверного логина или пароля, и auth_key — ключ авторизации. Далее, в случае успешной авторизации сохраняем в преференции статус залогиненности, логин, ключ авторизации и переходим к главному экрану. В случае неудачи оповещаем пользователя о том, что логин или пароль неверны.

```
  let res = response as NSHTTPURLResponse!;
  if (res.statusCode >= 200 && res.statusCode < 300)
  {
```

```
    var responseData:NSString  = NSString(data:urlData!,
        encoding:NSUTF8StringEncoding)!
    var error: NSError?
    let jsonData:NSDictionary = NSJSONSerialization.
        JSONObjectWithData(urlData!, options:
        NSJSONReadingOptions.MutableContainers , error: &
        error) as NSDictionary

    var result: Int=jsonData.valueForKey("res") as Int
    var auth_key: String=jsonData.valueForKey("auth_key")
        as String

    if (result != 0) {
        prefs.setInteger(1, forKey: "ISLOGGEDIN")
        prefs.setObject(login, forKey: "USERNAME")
        prefs.setValue(auth_key, forKey: "AUTHKEY")
        prefs.synchronize()
        performSegueWithIdentifier("done", sender: nil)
    }
} else {
    var alertView:UIAlertView = UIAlertView()
    alertView.title = "Authorization error"
    alertView.message = "Invalid login or password"
    alertView.delegate = self
    alertView.addButtonWithTitle("OK")
    alertView.show()
}
```

Переходим к Main.storyboard, в Document Outline в сцене «Профиль» выбираем «Modal segue to Авторизация». В Attributes Inspector в поле Identifier пишем «auth».

Переходим к файлу ViewController.swift. В метод viewDidLoad() добавляем код для запуска View Controller «Авторизации».

```
performSegueWithIdentifier("auth", sender: nil)
```

Добавляем в класс поле для хранения преференций.

```
var prefs = NSUserDefaults()
```

Далее переопределяем родительский метод класса viewWillAppear(), в котором инициализируем поле prefs.

```
override func viewWillAppear(animated: Bool) {
    self.prefs = NSUserDefaults.standardUserDefaults()
}
```

В метод logoutBtn добавим следующий код.

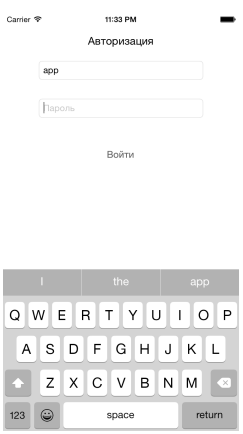

Рис. 43: Авторизация

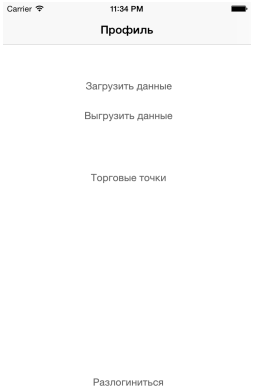

Рис. 44: Главное меню

```
prefs.setInteger(0, forKey: "ISLOGGEDIN")
prefs.setValue("", forKey: "AUTHKEY")
performSegueWithIdentifier("auth", sender: nil)
```

6.4. Загрузка списка торговых точек

Создадим класс для работы с сетью. Слева, в дереве файлов и каталогов, нажмем правой кнопкой на каталог Merch, выберем New File..., далее выберем iOS → Source → Swift File, имя зададим «ConnectionManager». Класс будет реализовывать протокол NSURLConnectionDataDelegate. В файл добавим следующий код:

```
import Foundation
import UIKit
import CoreData

class ConnectionManager : NSObject,
    NSURLConnectionDataDelegate {

}
```

Добавим поля authkey и URL в класс.

```
var authkey:String!
var URL : NSURL;
```

Также добавим конструктор для инициализации этих полей, в качестве параметра конструктор будет принимать authkey.

```
init(authkey: String) {
    self.authkey = authkey
    self.URL = NSURL(string: "http://maps.dvinemnauku.ru/
        Map/rows.php?n=0&m=1000&auth_key=\(authkey)")!;
}
```

Над классом ConnectionManager опишем протокол, который должен реализовывать класс, использующий ConnectionManager.

```
protocol ConnectionManagerDelegate {
    func downloadComplete()
    func error(message:String)
}
```

Добавим поле delegate, которое будет инициализироваться внешним классом.

```
var delegate:ConnectionManagerDelegate!
```

Далее реализуем метод downloadData() для создания и запуска в очередь асинхронного запроса к сети.

```
func downloadData() {
    var request = NSMutableURLRequest(URL: URL)
    request.HTTPMethod="POST"
    request.setValue("application/x-www-form-urlencoded",
        forHTTPHeaderField: "Content-Type")
    request.setValue("application/json",
        forHTTPHeaderField: "Accept")

    let queue:NSOperationQueue = NSOperationQueue()

    NSURLConnection.sendAsynchronousRequest(request, queue
        : queue, completionHandler: downloadComplete)
}
```

Теперь реализуем метод downloadComplete, который будет вызван, когда запрос вернет результаты. В случае ошибки вызовем метод error() класса, который реализует протокол ConnectionManagerDelegate, иначе распарсим полученные данные и вызовем метод downloadComplete() класса, который реализует протокол ConnectionManagerDelegate.

```
func downloadComplete(response: NSURLResponse!, data:
    NSData!, error: NSError!) {
    if error != nil {
        delegate.error(error.description);
    }
    else {
        var RecievedData : NSArray = NSJSONSerialization.
            JSONObjectWithData(data, options:
            NSJSONReadingOptions.MutableContainers, error:
             nil) as NSArray
        delegate.downloadComplete()
    }
}
```

Теперь нужно сохранить полученные данные в хранилище Core Data. Core Data — это фреймворк для работы с данными, хранимыми на устройстве. Слева, в дереве файлов и каталогов, откроем файл Merch.xcdatamodeld. Внизу нажмем кнопку Add Entity. Два раза кликнем по названию Entity и переименуем его в Points. Справа появится список атрибутов Attributes, пока еще пустой. Добавим атрибуты «id» (тип Integer 64), «items» (тип Binary Data), «latitude» (тип Double), «longitude» (тип Double), «name» (тип String), «photo» (тип String). Справа откроем Data Model inspector, в поле класс вместо Point напишем «Merch.Points».

Далее в списке ENTITIES выберем Points, в меню Editor выберем Create NSManagedObject Subclass..., дальше нажмем Next, установим флажок рядом с Points, нажмем Next, выберем Language — Swift, затем Create.

Создадим новый класс для работы с данными аналогично классу ConnectionManager. Слева, в дереве файлов и каталогов, нажмем правой кнопкой мыши на каталог Merch, выберем New File..., далее выберем iOS → Source → Swift File, имя зададим «DataManager». Добавим в файл следующий код.

```
import Foundation
import UIKit
import CoreData

class DataManager : NSObject {

}
```

Добавим в класс DataManager следующие поля.

```
var appDel:AppDelegate!
var context:NSManagedObjectContext!
var request:NSFetchRequest!
```

Добавим в класс конструктор, в котором проинициализируем поля класса, которые мы только что добавили.

```
override init() {
    appDel = UIApplication.sharedApplication().delegate as
        AppDelegate
    context = appDel.managedObjectContext
    request = NSFetchRequest(entityName: "Points")
}
```

Над классом DataManager опишем протокол, который должен реализовывать класс, использующий DataManager.

```
@objc protocol DataManagerDelegate : class {
    optional func reloadData(Points: AnyObject?)
}
```

Добавим поле delegate, которое будет инициализироваться внешним классом.

```
var delegate:DataManagerDelegate!
```

Далее добавим метод для получения данных, в котором выполняется запрос данных и передача их в метод reloadData(), реализуемый внешним классом.

```
func getData() {
    var error:NSErrorPointer = NSErrorPointer()
    var Data  = context.executeFetchRequest(request, error
        : error)
    delegate.reloadData!(Data?)
}
```

Далее добавим метод для очистки хранилища с данными.

```
func removeAllUserData() {
    var Data = context.executeFetchRequest(request, error:
        nil)
    var tmp: AnyObject
    for tmp in Data! {
        context.deleteObject(tmp as NSManagedObject)
    }
    context.save(nil)
}
```

Теперь вернемся к классу ConnectionManager. В метод

downloadComplete, перед вызовом метода delegate.downloadComplete(), добавим код для разбора пропарсенных данных и сохранения их в хранилище Core Data.

```
var A : DataManager = DataManager()
A.removeAllUserData()

var temp:NSDictionary!

let appDel = UIApplication.sharedApplication().delegate as
     AppDelegate
let context = appDel.managedObjectContext as
    NSManagedObjectContext?
let ent = NSEntityDescription.entityForName("Points",
    inManagedObjectContext: context!)

for temp in RecievedData {
    var newEntry = Points(entity: ent!,
        insertIntoManagedObjectContext: context)
    newEntry.name = temp.objectForKey!("name") as String
    newEntry.id = temp.valueForKey("id") as Int
    newEntry.latitude = temp.valueForKey!("latitude") as
        Double
    newEntry.longitude = temp.valueForKey!("longitude") as
         Double

   var itemData = NSJSONSerialization.dataWithJSONObject(
       temp.valueForKey("items") as NSArray, options:
       NSJSONWritingOptions.allZeros, error: nil) as
       NSData!
    newEntry.items = itemData
    context?.save(nil)
}
```

Перейдем к Main.storyboard, выделим View Controller «Профиль». Далее выделим кнопку «Загрузить данные», зажмем Ctrl и, протянув курсор мыши к Assistant Editor, отпустим курсор мыши над методом logoutBtn. Connection зададим Action, Name зададим «downloadBtn», Type зададим UIButton и нажмем Connect.

Откроем файл ViewController.swift, после родительского класса в заголовке класса допишем протокол ConnectionManagerDelegate.

```
class ViewController: UIViewController,
    ConnectionManagerDelegate {
```

И реализуем методы этого протокола.

```
func downloadComplete() {
 var alertView:UIAlertView = UIAlertView()
 alertView.title = "Загрузка"
 alertView.message = "Данные успешно загружены"
 alertView.delegate = self
 alertView.addButtonWithTitle("OK")
 alertView.show()
}

func error(message: String) {
 var alertView:UIAlertView = UIAlertView()
 alertView.title = "Загрузка"
 alertView.message = "Ошибка загрузки: \(message)"
 alertView.delegate = self
 alertView.addButtonWithTitle("OK")
 alertView.show()
}
```

Добавим поле для хранения ссылки на ConnectionManager в класс ViewController.

```
var DManager: ConnectionManager!
```

В метод viewWillAppear() добавим код для инициализации ссылки на ConnectionManager и укажем в качестве delegate текущий класс.

```
if (prefs.valueForKey("ISLOGGEDIN") != nil) {
    DManager = ConnectionManager(authkey: prefs.
        objectForKey("AUTHKEY") as String)
    DManager.delegate = self
}
```

В метод downloadBtn() добавим следующий код.

```
DManager.downloadData();
```

Перейдем к файлу Main.storyboard. В библиотеке объектов найдем Activity Indicator View, перетащим его на View Controller «Профиль». Затем зажмем Ctrl и перетащим курсор над методом viewDidLoad() в Assistant Editor. В появившемся окошке в поле Name напишем «activityView», затем нажмем Connect. Справа в Attributes Inspector установим флажки Animating и Hidden. Далее выделим кнопку «Загрузить данные», зажмем Ctrl и точно так же перетащим ее над методом viewDidLoad(). Name зададим «downloadButton». Нажмем Connect.

Перейдем к файлу ViewController.swift. В начало метода

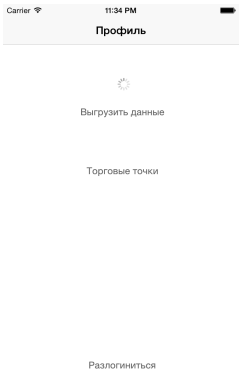

Рис. 45: Загрузка торговых точек

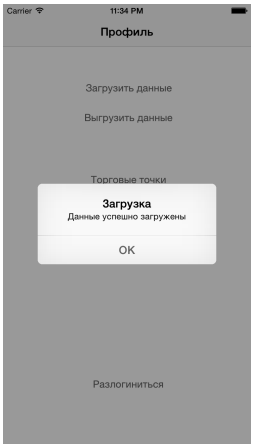

Рис. 46: Оповещение о статусе загрузки

downloadBtn добавим сокрытие кнопки и отображение индикатора активности.

```
downloadButton.hidden = true;
activityView.hidden = false;
```

Также в начало методов error() и downloadComplete() добавим код для сокрытия индикатора активности и отображения кнопки.

```
downloadButton.hidden = false;
activityView.hidden = true;
```

6.5. Список торговых точек

Теперь перейдем к Main.storyboard и добавим еще один View Controller. В библиотеке объектов выберем Table View Controller и перетащим на рабочую область. Далее выделим кнопку «Торговые точки», зажмем Ctrl, протянем курсор мыши к Table View Controller. Отпустим и в появившемся окошке выберем push.

Выберем только что созданный View Controller, откроем Attributes Inspector, зададим Title — «Торговые точки».

Слева, в дереве каталогов и файлов, нажмем правой кнопкой мыши на каталог Merch, выберем iOS → Source → Cocoa Touch Class. В поле Class напишем «Points», в поле Subclass of напишем «UITableViewController», Language выберем Swift. Нажмем Next,

а затем Create. Получим сгенерированный класс для нашего View Controller.

Откроем Main.storyboard, выберем View Controller «Торговые точки». В Identity Inspector в поле Custom Class → Class выберем созданный нами класс PointsTableViewController, в поле Identity → Storyboard ID пишем «points».

Откроем файл PointsTableViewController.swift, после родительского класса в заголовке класса допишем протокол DataManagerDelegate.

```
class PointsTableViewController: UITableViewController,
    DataManagerDelegate {
```

Добавим следующие поля в класс.

```
var array:NSArray = NSArray()
var Data:DataManager = DataManager();
let prefs = NSUserDefaults.standardUserDefaults()
```

В метод viewDidLoad добавим код для синхронизации преференций.

```
prefs.synchronize()
```

Теперь переопределим родительский метод viewDidAppear(), зададим delegate экземпляру класса DataManager и получим данные из хранилища Core Data.

```
override func viewDidAppear(animated: Bool) {
    Data.delegate = self
    Data.getData()
}
```

Реализуем метод reloadData() протокола DataManagerDelegate.

```
func reloadData(Points: AnyObject?) {
    array = Points as NSArray
    tableView.reloadData()
}
```

Теперь скорректируем метод tableView(tableView: UITableView, numberOfRowsInSection section: Int) таким образом, чтобы он возвращал корректное количество элементов списка.

```
return array.count
```

Также скорректируем метод numberOfSectionsInTableView(tableView: UITableView) так, чтобы он возвращал корректное количество секций в списке.

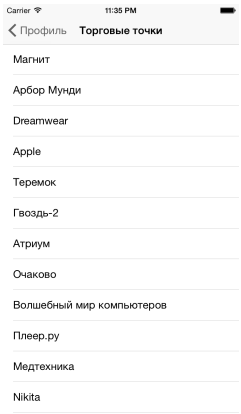

Рис. 47: Список торговых точек

Рис. 48: Карта

```
return 1
```

Раскомментируем и изменим метод tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath), который возвращает отображение строки списка.

```
let cell = UITableViewCell(style: UITableViewCellStyle.
    Default, reuseIdentifier: nil)
var object = array[indexPath.row] as NSManagedObject
var name = object.valueForKey("name") as? String

cell.textLabel?.text=name

return cell
```

6.6. Карта

Для работы с картой мы будем использовать Google Maps SDK для iOS. Первым делом нам нужно получить Google API Key. Подробно получение ключа описано здесь `https://developers.google.com/maps/documentation/ios/start#the_google_maps_api_key`. В результате мы получим ключ, что-то вроде AIzaSyBUD5SBJPAK2...

Затем скачаем SDK с сайта Google. Распакуем архив с Google Maps SDK под iOS, скопируем из архива GoogleMaps.framework в каталог Merch/Merch. Откроем Xcode IDE, слева в Project navigator

кликнем по проекту Merch, перейдем на вкладку Build Phases, развернем список Link Binary with Libraries и при помощи + добавим туда следующие фреймворки и библиотеки.

```
AVFoundation.framework
CoreData.framework
CoreLocation.framework
CoreText.framework
GLKit.framework
ImageIO.framework
libc++.dylib
libicucore.dylib
libz.dylib
OpenGLES.framework
QuartzCore.framework
SystemConfiguration.framework
```

Добавим также Google. Нажмем +, затем Add Other..., затем в подкаталоге Merch выберем GoogleMaps.framework и нажмем Open. После этого выделим в Project navigator добавленные фреймворки и библиотеки и переместим их в группу Supporting Files

Далее развернем список Copy Bundle Resources, нажмем +, затем Add Other..., затем в подкаталоге Merch/GoogleMaps.framework/Resources/ выберем GoogleMaps.bundle и нажмем Open и Finish.

Затем перейдем на вкладку Build Setting, наверху переключимся с Basic на All, найдем раздел Linking, далее найдем пункт Other Linker Flags, два раза кликнем справа от этого пункта, нажмем + и напишем «-ObjC», затем нажмем Enter.

Далее создадим Objective-C класс для того, чтобы Xcode создал файл Objective-C Bridging Header — это требуется, чтобы подключить Google карты к проекту. Нажмем File → New → File..., в появившемся диалоге выберем iOS → Source → Objective-C File, нажмем Next. Зададим любое имя файла, нажмем Next и Create. В появившемся диалоге нажмем Yes. Теперь удалим созданный нами файл, а в сгенерированный Merch-Bridging-Header.h добавим строку.

```
#import <GoogleMaps/GoogleMaps.h>
```

Откроем файл AppDelegate.swift. В метод application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) добавим строку.

```
GMSServices.provideAPIKey("YOUR_GOOGLE_MAPS_API_KEY")
```

Создадим View Controller класс для карты. Для этого нажмем File → New → File..., выберем iOS → Cocoa Touch Class, нажмем Next. В поле Class напишем Map, в Subclass of выберем UIViewController, Language выберем Swift. Нажмем Next и Create. Откроется сгенерированный файл MapViewController.swift.

Теперь откроем Main.storyboard и перетащим из библиотеки объектов в рабочую область View Controller. В Attributes Inspector зададим ему Title — Карта.

Откроем Identity inspector, в поле Custom Class → Class выберем MapViewController.

Дальше из библиотеки объектов перетащим в центр View Controller компонент View. Выделим View, в Identity Inspector в поле Custom CLass → Class выберем GMSMapView. Откроем Assistant Editor. Выделим View, зажмем Ctrl и перетащим курсор мыши над методом viewDidLoad, отпустим кнопку мыши, в появившемся окошке в поле Name напишем mapView и нажмем Connect.

Выберем View Controller «Торговые точки». Зажмем Ctrl, нажмем на первую иконку в заголовке View Controller и перетащим курсор мыши на View Controller карты.

В Document Outline в сцене Торговые точки выделим «Push segue to Карта», затем в Attributes Inspector в поле Identifier напишем «tomap».

Выделим Navigation Item в View Controller «Карта», в Attributes Inspector в поле Title напишем «Карта».

Откроем файл MapViewController.swift и добавим в класс следующие поля.

```
var locationManager:CLLocationManager!
var array:NSArray = NSArray()
var index:Int!
var data:DataManager = DataManager()
```

В заголовке файла после родительского класса UIViewController добавим протоколы GMSMapViewDelegate, UINavigationControllerDelegate, DataManagerDelegate, UIAlertViewDelegate, UIGestureRecognizerDelegate.

```
class MapViewController: UIViewController,
    GMSMapViewDelegate, UINavigationControllerDelegate,
    DataManagerDelegate, UIAlertViewDelegate,
    UIGestureRecognizerDelegate {
```

В методе viewDidLoad() зададим delegate экземпляру класса DataManager, запросим данные из хранилища Core Data, зададим delegate Navigation Controller и вызовем метод для инициализации карты, который будет описан дальше.

```
override func viewDidLoad() {
    super.viewDidLoad()

    data.delegate = self
    data.getData()

    self.navigationController?.
        interactivePopGestureRecognizer.delegate = self
    self.initMap()
}
```

Переопределим родительский метод viewDidAppear() для запроса данных из хранилища Core Data.

```
override func viewDidAppear(animated: Bool) {
    data.getData()
}
```

Создадим новый класс, который будет наследоваться от GMSMarker, чтобы можно было хранить в нем ID торговой точки. Нажмем File → New → File..., в появившемся диалоге выберем iOS → Source → Swift File. В поле Save as напишем «CustomMarker» и нажмем Create. В открывшийся файл добавим код.

```
class CustomMarker : GMSMarker {
    var id:Int!
}
```

Опишем метод для инициализации карты.

```
func initMap() {
    self.locationManager = CLLocationManager()
    locationManager.requestWhenInUseAuthorization()
    locationManager.startUpdatingLocation()


    var temp = array[index] as NSManagedObject

    var camera:GMSCameraPosition = GMSCameraPosition.
        cameraWithLatitude(temp.valueForKey("latitude") as
         Double, longitude: temp.valueForKey("longitude")
        as Double, zoom: 16)
    mapView.camera = camera
    mapView.settings.myLocationButton = true
    mapView.myLocationEnabled = true
    mapView.delegate = self
    mapView.mapType = kGMSTypeNormal
    self.view = mapView

    var t: NSManagedObject!

    for t in array {
        var marker = CustomMarker()
        marker.position = CLLocationCoordinate2DMake(t.
            valueForKey("latitude") as Double, t.
            valueForKey("longitude") as Double)
        marker.title = t.valueForKey("name") as String
        marker.id = t.valueForKey("id") as Int
        marker.map = mapView
    }
}
```

Переопределим родительский метод viewDidAppear() для запроса данных из хранилища Core Data.

```
override func viewDidAppear(animated: Bool) {
    data.getData()
}
```

Опишем метод протокола UIGestureRecognizerDelegate.

```
func gestureRecognizerShouldBegin(gestureRecognizer:
   UIGestureRecognizer) -> Bool {
    return false
}
```

Реализуем метод протокола DataManagerDelegate.

```
func reloadData(Points: AnyObject?) {
    array = Points as NSArray
}
```

Откроем файл PointsTableViewController.swift, добавим метод .

```
override func tableView(tableView: UITableView,
    didSelectRowAtIndexPath indexPath: NSIndexPath) {
    performSegueWithIdentifier("tomap", sender: indexPath.
        row)
}
```

Также раскомментируем и изменим метод prepareForSegue.

```
override func prepareForSegue(segue: UIStoryboardSegue,
    sender: AnyObject?) {
    if (segue.identifier=="tomap") {
        let VC = segue.destinationViewController as
            MapViewController
        VC.index = sender as? Int
        VC.array = self.array
    }

}
```

Теперь, чтобы отлавливать нажатие на информационное окошко над маркером, нам нужно реализовать метод mapView(mapView: GMSMapView!, didTapInfoWindowOfMarker marker: GMSMarker!) протокола GMSMapViewDelegate. Реализуем в нем диалог с запросом действия у пользователя.

```
func mapView(mapView: GMSMapView!,
    didTapInfoWindowOfMarker marker: GMSMarker!) {
    let alert:UIAlertView = UIAlertView(title: "", message
        : "", delegate: self, cancelButtonTitle: "Cancel",
         otherButtonTitles: "Сделать фото", "Выбрать фото
        ", "Список товаров")
    alert.show()
    mapView.selectedMarker = nil
}
```

6.7. Список товаров

Откроем файл Main.storyboard, из библиотеки объектов перетащим в рабочую область Table View Controller. Выбираем View Controller «Карта». Зажимаем Ctrl, нажимаем на первую иконку в

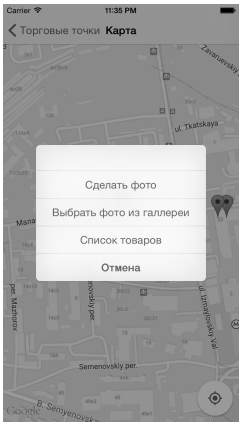

Рис. 49: Меню торговой точки

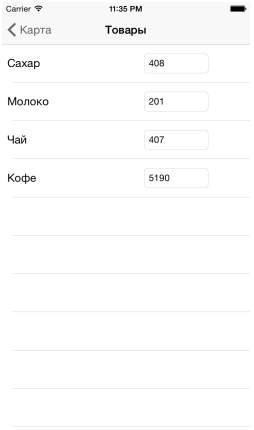

Рис. 50: Список товаров

заголовке View Controller и тащим курсор мыши на только что созданный Table View Controller. Отпускаем кнопку мыши, в появившемся диалоге выбираем push. Теперь выделяем Table View Controller, в Attributes Inspector в поле Title пишем «Товары».

Нажимаем на Navigation Item, в Attribute Inspector в поле Navigation Item → Title пишем «Товары». Нажимаем на ячейку под Prototype Cells внутри View Controller, увеличиваем размер области ячейки по высоте, помещаем на нее Label и Text Field из библиотеки объектов.

В Document Outline в сцене Карта нажимаем на пункт «Push segue to Товары» и в Attributes Inspector в поле Identifier пишем «goods».

Теперь создадим класс для View Controller «Товары». Нажмем File → New → File..., в появившемся диалоге выберем iOS → Source → Cocoa Touch Class. Нажмем Next. В поле класс напишем «Goods», в поле Subclass of выберем UITableViewController, в поле Language выберем Swift. Нажмем Next и Create.

В класс добавим два поля для хранения списка товаров и для хранения идентификатора торговой точки.

```
var items:NSMutableArray!
var point_id:Int!
```

Переопределим родительский метод viewWillAppear() и вызовем там метод для загрузки данных в таблицу.

```
override func viewWillAppear(animated: Bool) {
    tableView.reloadData()
}
```

Также создадим класс для строки таблицы. Нажмем File → New → File..., в появившемся диалоге выберем iOS → Source → Cocoa Touch Class. Нажмем Next. В поле класс напишем «Goods», в поле Subclass of Выберем UITableViewCell, в поле Language выберем Swift. Нажмем Next и Create.

В заголовке класса после родительского класса допишем протокол UITextFieldDelegate.

```
class GoodsTableViewCell: UITableViewCell ,
    UITextFieldDelegate{
```

Над классом опишем протокол, который будет использоваться внешним классом для обработки изменений в текстовых полях списка.

```
protocol GoodsTableViewCellDelegate {
    func valueChanged(sender: GoodsTableViewCell)
}
```

Также добавим в класс поле delegate, которое будет проинициализировано внешним классом.

```
var delegate:GoodsTableViewCellDelegate!
```

Откроем Main.storyboard. Выделим View Controller «Товары», на вкладке Identity Inspector в поле Custom Class → Class выберем GoodsTableViewController.

Далее нажмем на элемент под надписью Prototype Cells внутри View Controller, на вкладке Identity Inspector в поле Custom Class → Class выберем GoodsTableViewCell. С зажатым Ctrl перетащим в Assistant Editor компоненты Label и текстовое поле под поле delegate класса. Первый компонент назовем «nameLabel», второй «countField».

Затем в Attributes Inspector в поле Table View Cell → Identifier пишем «Cell».

Вернемся к файлу GoodsTableViewCell.swift и переопределим родительский метод класса.

```
override func awakeFromNib() {
    super.awakeFromNib()
    countField.delegate = self
}
```

Также реализуем метод textFieldDidEndEditing(textField: UITextField) протокола UITextFieldDelegate, в котором передадим классу, реализующему протокол GoodsTableViewCellDelegate, информацию об измененной строке.

```
func textFieldDidEndEditing(textField: UITextField) {
    delegate.valueChanged(self)
}
```

Теперь перейдем к классу GoodsTableViewController.swift. В заголовке класса после родительского класса добавим протокол GoodsTableViewCellDelegate.

```
class GoodsTableViewController: UITableViewController,
    GoodsTableViewCellDelegate {
```

Раскомментируем и изменим метод tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath).

```
override func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) ->
    UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier
        ("Cell") as GoodsTableViewCell
    cell.nameLabel.text = items[indexPath.row].
        objectForKey("name") as? String
    cell.delegate = self
    var count = items[indexPath.row].objectForKey("count")
        as Int

    cell.countField.text = "\(count)"

    return cell
}
```

Также переопределим метод tableView(tableView: UITableView, shouldHighlightRowAtIndexPath indexPath: NSIndexPath), чтобы не подсвечивать строку списка при нажатии и метод touchesBegan(touches: NSSet, withEvent event: UIEvent), чтобы при нажатии фокус получало текстовое поле.

```
override func tableView(tableView: UITableView,
    shouldHighlightRowAtIndexPath indexPath: NSIndexPath)
    -> Bool {
```

```
    return false
}

override func touchesBegan(touches: NSSet, withEvent event
    : UIEvent) {
    self.resignFirstResponder()
}
```

И скорректируем два метода для корректного отображения списка товаров.

```
override func tableView(tableView: UITableView,
    numberOfRowsInSection section: Int) -> Int {
    return array.count
}

override func numberOfSectionsInTableView(tableView:
    UITableView) -> Int {
    return 1
}
```

Далее реализуем метод valueChanged() протокола GoodsTableViewCellDelegate.

```
func valueChanged(sender: ItemTableViewCell) {

    var row = tableView.indexPathForCell(sender)?.row
    var count = (sender.countField.text as NSString).
        integerValue
    var temp : NSMutableDictionary = NSMutableDictionary(
        dictionary: items[row!] as NSDictionary)

    temp.setValue(count, forKey: "count")
    items[row!] = NSDictionary(dictionary: temp)
    var DM = DataManager()
    DM.updateItems(items, point_id: point_id)

    tableView.reloadData()
}
```

Откроем файл DataManager.swift и добавим метод updateItems() для обновления данных.

```
func updateItems(Items: NSArray, point_id: Int) {
    var req = NSBatchUpdateRequest(entityName: "Points")
    req.predicate = NSPredicate(format: "id == %d",
        point_id)
```

```
    var itemData : NSData! = NSJSONSerialization.
        dataWithJSONObject(Items, options:
        NSJSONWritingOptions.allZeros, error: nil) as
        NSData!

    req.propertiesToUpdate = ["items" : itemData]

    req.resultType = NSBatchUpdateRequestResultType.
        UpdatedObjectIDsResultType

    var res = context.executeRequest(req, error: nil) as
        NSBatchUpdateResult

    for point_id in res.result as NSArray {
        let object = context.existingObjectWithID(point_id
            as NSManagedObjectID, error: nil)
        context.refreshObject(object!, mergeChanges: true)
    }
    context.save(nil)
}
```

Перейдем к классу MapViewController.swift. Для обработки нажатия на кнопки диалога реализуем метод alertView(alertView: UIAlertView, clickedButtonAtIndex buttonIndex: Int) протокола UIAlertViewDelegate.

```
func alertView(alertView: UIAlertView,
    clickedButtonAtIndex buttonIndex: Int) {
    // Goods list
    if buttonIndex == 3 {
        var p: NSManagedObject =  array[index] as
            NSManagedObject
        performSegueWithIdentifier("goods", sender: p)
    }
}
```

В этом же классе раскомментируем метод prepareForSegue() и добавим в него следующий код.

```
override func prepareForSegue(segue: UIStoryboardSegue,
    sender: AnyObject?) {
    if segue.identifier == "goods" {
        let vc = segue.destinationViewController as
            GoodsTableViewController
```

```
        var it : NSArray = NSJSONSerialization.
            JSONObjectWithData(sender?.valueForKey("items
            ") as NSData, options: NSJSONReadingOptions.
            allZeros, error: nil) as NSArray

        vc.items = NSMutableArray(array: it)
        vc.point_id = sender?.valueForKey("id") as Int
    }
}
```

6.8. Выгрузка товаров на сервер

Откроем Main.storyboard. Найдем View Controller «Профиль». Выделим кнопку «Выгрузить данные», зажмем Ctrl, перетащим курсор мыши в Assistant Editor под метод downloadBtn(), отпустим кнопку мыши. В поле Connection укажем Action, в поле Name — «uploadBtn», в поле Type укажем UIButton и нажмем Connect.

Перетащим из библиотеки объектов на View Controller в центр кнопки «Выгрузить данные» Activity Indicator View, зажмем Ctrl, перетащим курсор мыши в Assistant Editor над методом viewDidLoad(). В появившемся окошке в поле Name напишем «activityUploadView» и нажмем Connect. То же самое сделаем для кнопки «Выгрузить данные», только в поле Name напишем «uploadButton». Выделим только что добавленный Activity Indicator View и в Attributes Inspector установим флажки Animating и Hidden.

Создадим новый класс для выгрузки данных аналогично классу DataManager. Нажмем File → New → File..., далее выберем iOS → Source → Swift File, имя зададим «UploadManager». Добавим в файл следующий код.

```
import Foundation
import UIKit
import CoreData

class UploadManager : NSObject {

}
```

Добавим в класс UploadManager следующие поля.

```
var appDel:AppDelegate!
var context:NSManagedObjectContext!
```

```
var request:NSFetchRequest!
var data:NSData!
var authkey:String!
```

Добавим в класс конструктор, в котором проинициализируем поля класса, которые мы только что добавили.

```
init(authkey:String) {
    self.authkey = authkey
    appDel = UIApplication.sharedApplication().delegate as
        AppDelegate
    context = appDel.managedObjectContext
    request = NSFetchRequest(entityName: "Points")
}
```

Над классом UploadManager опишем протокол, который будет реализован внешним классом.

```
protocol UploadManagerDelegate {
    func uploadComplete()
    func uploadError(message:String)
}
```

Добавим поле delegate, которое будет инициализироваться внешним классом.

```
var delegate:UploadManagerDelegate!
```

Далее добавим метод для формирования JSON по данным из хранилища Core Data и отправки запросов на сервер для обновления данных.

```
func sendJSONData() {
    var error:NSErrorPointer = NSErrorPointer()
    var Data  = context.executeFetchRequest(request, error
        : error)
    var arrayOfJSON = NSMutableArray()

    let priority = DISPATCH_QUEUE_PRIORITY_DEFAULT

    dispatch_async(dispatch_get_global_queue(priority, 0))
        {
    var tmp: NSManagedObject
    for tmp in Data! {
        var dict = NSMutableDictionary()
```

```
var itemData = NSJSONSerialization.
    JSONObjectWithData(tmp.valueForKey("items") as
     NSData, options: NSJSONReadingOptions.
    allZeros, error: nil) as NSArray

for it in itemData {
    dict.setValue(it.objectForKey("id") as Int,
        forKey: "id")
    dict.setValue(it.objectForKey("name") as
        String, forKey: "name")
    dict.setValue(it.objectForKey("count") as Int,
         forKey: "count")
    dict.setValue(it.objectForKey("good_id") as
        Int, forKey: "good_id")
    dict.setValue(it.objectForKey("point_id") as
        Int, forKey: "point_id")
    var data = NSJSONSerialization.
        dataWithJSONObject(dict, options:
        NSJSONWritingOptions.allZeros, error: nil)

    var point_id:Int = it.objectForKey("point_id")
         as Int!
    var good_id:Int = it.objectForKey("good_id")
        as Int!
    var count:Int = it.objectForKey("count") as
        Int!
    var url = NSURL(string: "http://maps.
        dvinemnauku.ru/Map/updateGood.php?point_id
        =\(point_id)&good_id=\(good_id)&count=\(
        count)&auth_key=\(self.authkey)")
    var request:NSMutableURLRequest =
        NSMutableURLRequest(URL: url!)
    request.HTTPMethod = "POST"
    request.HTTPBody = data
    request.setValue("application/x-www-form-
        urlencoded", forHTTPHeaderField: "Content-
        Type")
    request.setValue("application/json",
        forHTTPHeaderField: "Accept")

    var reponseError: NSError?
    var response: NSURLResponse?

    var urlData: NSData? = NSURLConnection.
        sendSynchronousRequest(request,
        returningResponse:&response, error:&
        reponseError)
```

```
            if ( error != nil ) {
                dispatch_async(dispatch_get_main_queue())
                    {
                    self.delegate.uploadError(error.
                        debugDescription)
                }
            } else {
                let res = response as NSHTTPURLResponse!;
                if (res.statusCode != 200)
                {
                    dispatch_async(dispatch_get_main_queue
                        ()) {
                        self.delegate.uploadError("Ошибка
                            выгрузки")
                    }
                    break
                }
            }
        }

        dispatch_async(dispatch_get_main_queue()) {
            self.delegate.uploadComplete()
        }
    }
    }
}
```

Откроем файл ViewController.swift, добавим поле для хранения ссылки на UploadManager.

```
var UManager: UploadManager!
```

В метод viewWillAppear() внутри блока if добавим код для инициализации экземпляра класса UploadManager и зададим ему delegate.

```
UManager = UploadManager(authkey: prefs.objectForKey("
    AUTHKEY") as String)
UManager.delegate = self
```

Добавим в заголовок класса ViewController после протокола ConnectionManagerDelegate протокол UploadManagerDelegate.

```
class ViewController: UIViewController,
    ConnectionManagerDelegate, UploadManagerDelegate {
```

Реализуем методы uploadComplete() и uploadError() протокола UploadManagerDelegate.

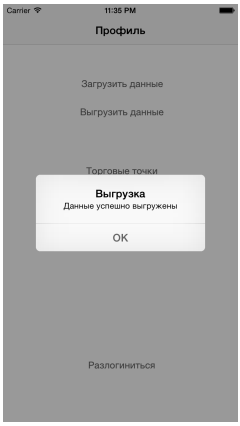

Рис. 51: Выгрузка данных

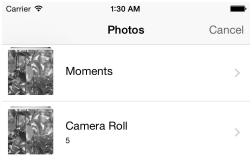

Рис. 52: Диалог выбора фотографии

```
func uploadComplete() {
    self.uploadButton.hidden = false
    self.activityUploadView.hidden = true

    var alertView:UIAlertView = UIAlertView()
    alertView.title = "Выгрузка"
    alertView.message = "Данные успешно выгружены"
    alertView.delegate = self
    alertView.addButtonWithTitle("OK")
    alertView.show()
}
func uploadError(message: String) {
    self.uploadButton.hidden = false
    self.activityUploadView.hidden = true

    var alertView:UIAlertView = UIAlertView()
    alertView.title = "Выгрузка"
    alertView.message = "Ошибка выгрузки: \(message)"
    alertView.delegate = self
    alertView.addButtonWithTitle("OK")
    alertView.show()
}
```

Найдем метод uploadBtn() и добавим код для показа индикатора активности, сокрытия кнопки «Выгрузить данные» и вызова метода выгрузки данных на сервер.

```
    self.uploadButton.hidden = true
    self.activityUploadView.hidden = false
    UManager.sendJSONData()
```

6.9. Работа с фото

Откроем класс DataManager.swift и добавим в него метод для сохранения фотографии для торговой точки в хранилище Core Data.

```
    func addMainPhoto(image:UIImage! , id: Int) {
        var req = NSBatchUpdateRequest(entityName: "Points")
        req.predicate = NSPredicate(format: "id == %d", id)

     if image != nil {
         var imageData = UIImageJPEGRepresentation(image,
             0.3)
         req.propertiesToUpdate = ["photo" : imageData]
     }
     else {
         req.propertiesToUpdate = ["photo" : ""]
     }

     req.resultType = NSBatchUpdateRequestResultType.
         UpdatedObjectIDsResultType
     var res = context.executeRequest(req, error: nil) as
         NSBatchUpdateResult

     for id in res.result as NSArray {
         let object = context.existingObjectWithID(id as
             NSManagedObjectID, error: nil)
         context.refreshObject(object!, mergeChanges: true)
     }
     context.save(nil)
}
```

Перейдем к классу MapViewController.swift. В заголовке класса добавим протокол UIImagePickerControllerDelegate. Реализуем метод этого протокола.

```
func imagePickerController(picker: UIImagePickerController
    !, didFinishPickingImage image: UIImage!, editingInfo:
     [NSObject : AnyObject]!) {
     var p : NSManagedObject = array[index] as
         NSManagedObject
     data.addMainPhoto(image, id: p.valueForKey("id") as
         Int)
     data.getData()
     self.dismissViewControllerAnimated(true, completion:
         nil)

}
```

В метод alertView(alertView: UIAlertView, clickedButtonAtIndex buttonIndex: Int) добавим код для получения фото из галереи и с камеры.

```
if buttonIndex == 2 {
     var imagePicker = UIImagePickerController()
     imagePicker.sourceType =
         UIImagePickerControllerSourceType.PhotoLibrary
     imagePicker.modalPresentationStyle =
         UIModalPresentationStyle.CurrentContext
     imagePicker.delegate = self
     self.presentViewController(imagePicker, animated: true
         , completion: nil)
}

if buttonIndex == 1 {
     var imagePicker = UIImagePickerController()
     imagePicker.sourceType =
         UIImagePickerControllerSourceType.Camera
     imagePicker.modalPresentationStyle =
         UIModalPresentationStyle.CurrentContext
     imagePicker.delegate = self
     self.presentViewController(imagePicker, animated: true
         , completion: nil)
}
```

6.10. Отправка фото на сервер

Откроем файл UploadManager.swift, найдем метод sendJSONData() и перед циклом «for it in itemData» добавим следующий код.

```
var imageData : NSData! = tmp.valueForKey("photo") as?
    NSData
var point_id:Int = tmp.objectForKey("id") as Int!
if imageData != nil {

}
```

Далее в блоке if проинициализируем класс для работы с сетью и выставим заголовки запроса.

```
var url = NSURL(string: "http://maps.dvinemnauku.ru/Map/
    postImage.php?point_id=\(point_id)&auth_key=\(self.
    authkey)")
var request:NSMutableURLRequest = NSMutableURLRequest(URL:
     url!)
request.HTTPMethod = "POST"
let boundary = "3676416B-9AD6-440C-B3C8-FC66DDC7DB45"
request.setValue("multipart/form-data; boundary=\(boundary
    )", forHTTPHeaderField: "Content-Type")
```

Затем сформируем тело запроса.

```
var body = NSMutableData();
let filename = "photo.jpg"
let mimetype = "image/jpg"

body.appendData("--\(boundary)\r\n".dataUsingEncoding(
    NSUTF8StringEncoding, allowLossyConversion: true)!)
body.appendData("Content-Disposition: form-data; name=\"
    image\"; filename=\"\(filename)\"\r\n".
    dataUsingEncoding(NSUTF8StringEncoding,
    allowLossyConversion: true)!)
body.appendData("\r\n".dataUsingEncoding(
    NSUTF8StringEncoding, allowLossyConversion: true)!)
body.appendData(imageData)
body.appendData("\r\n".dataUsingEncoding(
    NSUTF8StringEncoding, allowLossyConversion: true)!)
body.appendData("--\(boundary)--\r\n".dataUsingEncoding(
    NSUTF8StringEncoding, allowLossyConversion: true)!)
request.HTTPBody = body
```

И отправим запрос на сервер.

```
var reponseError: NSError?
var response: NSURLResponse?
var urlData: NSData? = NSURLConnection.
    sendSynchronousRequest(request, returningResponse:&
    response, error:&reponseError)
```

После этого проверим результат запроса от сервера, в случае ошибки оповестим пользователя и прекратим работу.

```
if ( error != nil ) {
    dispatch_async(dispatch_get_main_queue()) {
        self.delegate.uploadError(error.debugDescription)
    }
    return
} else {
    let res = response as NSHTTPURLResponse!;
    if (res.statusCode != 200)
    {
        dispatch_async(dispatch_get_main_queue()) {
            self.delegate.uploadError("Ошибка выгрузки фот
                o")
        }
        return
    }
}
```

7. Android

В данной главе мы, аналогично шестой главе, напишем мобильное приложение под платформу Android, которое позволит мерчендайзерам производить посещения торговых точек, фиксировать выкладки, производить аудит цен, а также фиксировать наличие товара на полке и/или на складе. В конце посещения — синхронизировать собранные данные с серверной частью.

Для реализации описанного функционала нам понадобится реализовать авторизацию пользователей в приложении, построить пользовательский интерфейс для отображения списка торговых точек и других фрагментов с целью реализации пользовательского взаимодействия с приложением. Далее мы разберем работу с картой, выставление маркеров и обработку действий над ними. Рассмотрим работу с камерой и обработку полученного с нее изображения. Узнаем принципы взаимодействия с базами данных. И, наконец, освоим синхронизацию данных с серверной частью. В результате мы получим работающее мобильное приложение для платформы Android с описанным выше функционалом.

7.1. Установка и настройка среды разработки

Скачиваем и устанавливаем последнюю версию Android Studio с сайта developer.android.com. Доступна на Mac, Windows и Linux.

Скачиваем и устанавливаем JDK 1.7.x, запросит Android Studio во время установки.

7.2. Создание проекта

Создаем новый проект в Android Studio. Application name зададим «Merch». Company Domain — «dvinemnauku.ru». При этом автоматически сформировался пакет ru.dvinemnauku.merch. Minimum SDK оставляем API 15: Android 4.0.3. Выбираем шаблон приложения — Add No Activity.

Слева нажимаем на вкладку Project. В появившемся дереве раскрываем элемент java, нажимаем правой кнопкой мыши на пакет ru.dvinemnauku.merch, выбираем в меню пункт New → Activity → Blank Activity. В появившемся диалоге выставляем галочку Launcher

Activity. Это значит, что эта активити главная и будет запущена при старте приложения. Создаем Activity.

Создалась активити. Для нее создался класс MainActivity.java в пакете ru.dvinemnauku.merch, в котором описывается логика работы Activity и activity_main.xml в каталоге res/layout, в котором описывается внешний вид Activity. После создания Activity в Android Studio откроется визуальный редактор внешнего вида Activity.

Создано минимальное приложение под Android, которое теперь мы будем развивать.

7.3. Окно авторизации

Создаем новую активити аналогично тому, как мы это делали в первый раз. На этот раз не выставляем галочку Launcher Activity. Activity Name задаем LoginActivity, остальные параметры оставляем как есть. Создаем Activity.

Откроется визуальный редактор файла activity_login.xml. Переходим на вкладку Text внизу редактора. Удаляем элемент TextView и добавляем поля для ввода логина и пароля и кнопку для авторизации внутри элемента RelativeLayout.

```
<EditText
    android:id="@+id/login"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    />

<EditText
    android:id="@+id/password"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/login"
    android:inputType="textPassword"
    />

<Button
    android:id="@+id/login_btn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/password"
    android:text="Login"
    />
```

Для кнопки прописываем обработчик нажатия на кнопку после свойства android:text:

```
android:onClick="startLogin"
```

В классе LoginActivity.java описываем метод startLogin, который будет вызываться при нажатии на кнопку авторизации:

```
public void startLogin(View v) {
    // Здесь будет код старта авторизации
}
```

В классе LoginActivity создадим два поля для хранения ссылок на поля ввода логина и пароля, а также переменные, в которые будут заноситься значения из этих полей перед процессом авторизации.

```
private EditText loginEdit;
private EditText passwordEdit;
String login, password;
```

В методе onCreate после вызова метода setContentView находим эти поля по их id и присваиваем ссылки на них созданным полям класса:

```
loginEdit = (EditText) findViewById(R.id.login);
passwordEdit = (EditText) findViewById(R.id.password);
```

Для авторизации нам необходимо сделать запрос к серверной части. Все потенциально долгие операции, в частности работу с сетью, необходимо выносить в отдельный поток. Android не позволяет выполнить обращение к сети в главном (UI) потоке. Это сделано для того, чтобы долгие операции не влияли на время отклика пользовательского интерфейса. Для выполнения обращения к сети в отдельном потоке мы напишем класс, наследованный от класса AsyncTask. Этот класс будет вложенным в класс LoginActivity.java.

```
public class AuthTask extends AsyncTask<Void, Void,
    Boolean> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }
```

```
    @Override
    protected Boolean doInBackground(Void... params) {
        return null;
    }

    @Override
    protected void onPostExecute(Boolean success) {
        super.onPostExecute(success);
    }
}
```

Метод onPreExecute() выполняется перед тем, как запустить долгую операцию, в главном (UI) потоке. Действия, которые нужно выполнить в отдельном потоке, описываются в методе doInBackground. Он выполняется в отдельном потоке (не в UI). Значение, которое возвращает метод doInBackground, получает метод onPostExecute() в качестве параметра. Метод onPostExecute() выполняется в главном (UI) потоке.

Код авторизации пользователя на сервере выглядит следующим образом. Сначала в класс AuthTask мы добавляем поле для хранения ключа авторизации, чтобы позже, при успешной авторизации, присвоить ему значение.

```
private String authKey;
```

Далее, в методе doInBackground мы заводим локальную переменную res для хранения информации о статусе авторизации.

```
int res = 0;
```

Далее мы инициализируем класс для обращения к сети, где указываем URL и метод, которым мы отправляем запрос (POST):

```
URL url = new URL("http://maps.dvinemnauku.ru/Map/login.
    php?type=json");
HttpURLConnection c = (HttpURLConnection)url.
    openConnection();
c.setRequestMethod("POST");
c.setDoInput(true);
c.setDoOutput(true);
```

Затем выводим параметры POST запроса:

```
OutputStream os = c.getOutputStream();
BufferedWriter writer = new BufferedWriter(
    new OutputStreamWriter(os, "UTF-8"));
```

```
writer.write("login="+login+"&pwd="+password);
writer.close();
os.close();
```

После чего отправляем запрос на сервер и получаем статус HTTP запроса:

```
c.connect();
int status = c.getResponseCode();
```

Если статус HTTP запроса 200, то авторизация прошла успешно. Мы читаем JSON, который прислал нам сервер, а именно значение параметра res, который содержит информацию о том, авторизовались мы или нет, и auth_key, который содержит ключ для обращения к другим сервисам серверной части.

```
if (status == 200) {
    JsonReader reader = new JsonReader(new
        InputStreamReader(c.getInputStream(), "UTF-8"));
    reader.beginObject();
    while (reader.hasNext()) {
        String name = reader.nextName();

        if (name.equals("res")) {
            res = reader.nextInt();
        } else if (name.equals("auth_key")) {
            authKey = reader.nextString();
        } else {
            reader.skipValue();
        }
    }
    reader.endObject();
}
```

Операции с сетью могут вызывать исключительные ситуации, например, если прервется соединение с сетью. Поэтому оборачиваем весь код обращения к сети в конструкцию try-catch, чтобы наше приложение не упало при возникновении таких ситуаций. В случае возникновения исключительной ситуации выводим в лог системы информацию об исключении.

```
try {
   // Код для работы с сетью
} catch (Exception e) {
    Log.e("AuthTask", "Network error", e);
}
```

В конце возвращаем статус авторизации для метода onPostExecute:

```
return (res == 1);
```

В методе onPostExecute мы проверяем, успешно ли прошла авторизация. В случае, если пользователь авторизован, мы возвращаем в главную активити ключ авторизации и завершаем текущую активити методом finish(). В противном случае уведомляем пользователя о том, что логин или пароль были введены неверно, и передаем фокус полю для ввода пароля.

```
if (success) {
    Intent result = new Intent();
    result.putExtra("auth_key", authKey);
    setResult(RESULT_OK, result);
    finish();
} else {
    passwordEdit.setError("Неверный логин или пароль!");
    passwordEdit.requestFocus();
}
```

В методе startLogin, который мы создавали ранее, заносим логин и пароль в соответствующие переменные для использования их во время авторизации и запускаем процесс авторизации, реализованный в классе AuthTask.

```
public void startLogin(View v) {
    login = loginEdit.getText().toString();
    password = passwordEdit.getText().toString();
    new AuthTask().execute();
}
```

Любое приложение под Android включает в себя манифест приложения (файл AndroidManifest.xml). Этот файл содержит информацию о разрешениях, которые нужны приложению, о компонентах приложения и другую информацию, которая используется системой Android. Для того, чтобы система Android предоставила нашему приложению доступ к сети, добавим в манифест приложения перед элементом application соответствующее разрешение:

```
<uses-permission android:name="android.permission.INTERNET
    " />
```

Окно авторизации должно запускаться, если пользователь не авторизован. Удобнее всего открывать окно авторизации в методе onCreate класса MainActivity.java после вызова метода setContentView, который

связывает логику работы Activity с ее внешним видом.

Запуск Activity авторизации производится следующим кодом:

```
startActivityForResult(new Intent(this, LoginActivity.
    class), LOGIN_ACTIVITY);
```

LOGIN_ACTIVITY — это константа, описанная в классе MainActivity и определяющая некий идентификатор, по которому можно определить, какая именно активити вернула ответ в главную активити:

```
private static final int LOGIN_ACTIVITY = 1;
```

Далее нам необходимо обработать ответ, который вернула LoginActivity после авторизации. Для этого необходимо в классе MainActivity реализовать метод onActivityResult, в котором мы проверяем, от какой активити пришел ответ и каков был статус запуска активити. Кроме того, в класс MainActivity добавим поле authKey для хранения ключа авторизации, а в методе onActivityResult присвоим этому полю значение.

```
private String authKey = null;

@Override
protected void onActivityResult(int requestCode, int
    resultCode, Intent data) {
     super.onActivityResult(requestCode, resultCode, data);

     if (requestCode == LOGIN_ACTIVITY) {
         if (resultCode == RESULT_OK) {
             // Пользователь успешно авторизован
             authKey = data.getStringExtra("auth_key");
         } else {
             finish();
         }
     }
}
```

7.4. Загрузка списка торговых точек

В каталоге res/layout находим файл activity_main.xml. Переходим на вкладку Text внизу визуального редактора. Удаляем элемент TextView. Меняем корневой элемент с RelativeLayout на LinearLayout. После свойства tools:context добавляем свойство

Рис. 53: Авторизация

Рис. 54: Главное меню

android:orientation="vertical". Внутрь элемента LinearLayout добавляем элементы типа Button, которые будут составлять меню нашего приложения. В каждую кнопку добавляем обработчики нажатия на кнопку аналогично тому, как мы делали это в предыдущем разделе.

```
<Button
    android:id="@+id/download"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Загрузить данные"
    android:onClick="onDownload"
    />

<Button
    android:id="@+id/upload"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Выгрузить данные"
    android:onClick="onUpload"
    />

<Button
    android:id="@+id/list"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
    android:text="Торговые точки"
    android:onClick="onList"
    />

<Button
    android:id="@+id/logout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Разлогиниться"
    android:onClick="onLogout"
    />
```

В классе MainActivity.java описываем все эти обработчики:

```
public void onDownload(View v) {
    // Загрузка данных
}

public void onUpload(View v) {
    // Выгрузка данных
}

public void onList(View v) {
    // Список торговых точек
}

public void onLogout(View v) {
    // Разлогиниться
}
```

Для загрузки списка торговых точек напишем класс DownloadTask, наследованный от класса AsyncTask, аналогично тому, как мы делали это для авторизации. Этот класс будет вложенным в класс MainActivity.java.

```
public class DownloadTask extends AsyncTask<Void, Void,
    Boolean> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }

    @Override
    protected Boolean doInBackground(Void... params) {
        return null;
    }
```

```
    @Override
    protected void onPostExecute(Boolean success) {
        super.onPostExecute(success);
    }
}
```

Во время загрузки данных из сети было бы хорошо показывать прогрессбар, чтобы пользователь понимал, что идет загрузка. Для этого откроем файл activity_main.xml и добавим под кнопкой с идентификатором @+id/download контрол ProgressBar с атрибутом android:visibility="gone"для того, чтобы он не отображался по умолчанию.

```
<ProgressBar
    android:id="@+id/progress"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:visibility="gone"
    />
```

В класс MainActivity.java добавим поле для хранения ссылки на этот ProgressBar.

```
private ProgressBar progressBar;
```

В метод onCreate класса MainActivity присвоим полю ссылку на ProgressBar.

```
progressBar = (ProgressBar) findViewById(R.id.progress);
```

В метод onPreExecute класса DownloadTask добавим код для того, чтобы сделать прогрессбар видимым. Для этого используется метод setVisibility(), на вход которого подается одна из констант класса View: VISIBLE (объект видим), INVISIBLE (объект невидим, но занимает пространство), GONE (объект невидим и не занимает пространства).

```
progressBar.setVisibility(View.VISIBLE);
```

В метод onPostExecute этого же класса добавим код для сокрытия прогрессбара.

```
progressBar.setVisibility(View.GONE);
```

Далее в методе doInBackground проинициализируем переменные,

в которые будем читать данные, получаемые от серверной части.

```
int id = 0;
String name = null;
double latitude = 0.0;
double longitude = 0.0;
boolean photo = false;
String address = null;
```

Затем проинициализируем класс для обращения к сети, отправим на сервер GET запрос и получим статус HTTP запроса.

```
URL url = new URL("http://maps.dvinemnauku.ru/Map/rows.php
    ?n=0&m=1000&auth_key="+authKey);
HttpURLConnection c = (HttpURLConnection)url.
    openConnection();
c.setRequestMethod("GET");
c.connect();
int status = c.getResponseCode();
```

После чего проверим статус HTTP запроса и, если он равен 200, то распарсим массив торговых точек, полученных от серверной части.

```
if (status == 200) {
    JsonReader reader = new JsonReader(new
        InputStreamReader(c.getInputStream(), "UTF-8"));
    reader.beginArray();
    while (reader.hasNext()) {
        reader.beginObject();
        while (reader.hasNext()) {
            String key = reader.nextName();

            if (key.equals("id")) {
                id = reader.nextInt();
            } else if (key.equals("name")) {
                name = reader.nextString();
            } else if (key.equals("address")) {
                address = reader.nextString();
            } else if (key.equals("latitude")) {
                latitude = reader.nextDouble();
            } else if (key.equals("longitude")) {
                longitude = reader.nextDouble();
            } else if (key.equals("photo")) {
                photo = reader.nextBoolean();
```

```
                } else {
                    reader.skipValue();
                }
            }
            reader.endObject();
        }
        reader.endArray();
    }
```

И если до этого момента ничего не случилось, значит все прошло успешно, возвращаем true и оборачиваем весь код для работы с сетью в конструкцию try-catch:

```
try {
     return true;
} catch (Exception e) {
     Log.e("DownloadTask", "Network error", e);
}
```

В конце метода возвращаем false, т. к. если не произошел выход из этого метода в блоке try-catch, то что-то пошло не так.

```
return false;
```

Данные, полученные от серверной части, следует хранить в БД. Для работы с БД нам нужно создать класс, который наследует SQLiteOpenHelper, и реализовать методы для создания и апгрейда базы данных. Для этого слева нажмем правой кнопкой мыши по пакету нашего приложения. Далее выберем New → Java Class и зададим имя классу Name: PointsDatabaseHelper. После имени только что созданного класса допишем extends SQLiteOpenHelper. Затем установим курсор на названии класса и нажмем Alt+Enter. В появившемся диалоге выберем Implement methods. После этого IDE добавит в класс следующие методы.

```
public class PointsDatabaseHelper extends SQLiteOpenHelper
     {

    @Override
    public void onCreate(SQLiteDatabase db) {

    }
```

```
    @Override
    public void onUpgrade(SQLiteDatabase db, int
        oldVersion, int newVersion) {

    }
}
```

Сразу добавим в начало класса две константы: первая — версия БД, вторая — имя файла БД.

```
public static final int DATABASE_VERSION = 1;
public static final String DATABASE_NAME = "Points.db";
```

После добавленных нами констант реализуем конструктор класса.

```
public PointsDatabaseHelper(Context context) {
    super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
```

Далее, над конструктором добавим еще две константы, которые содержат SQL код для создания и удаления таблицы с торговыми точками.

```
private static final String SQL_CREATE_ENTRIES =
    "CREATE TABLE POINTS (" +
        "_id INTEGER PRIMARY KEY," +
        "name TEXT," +
        "address TEXT," +
        "latitude TEXT," +
        "longitude TEXT," +
        "photo INTEGER," +
        "image TEXT," +
        "updated INTEGER DEFAULT 0" +
        " )";

private static final String SQL_DELETE_ENTRIES =
    "DROP TABLE IF EXISTS POINTS";
```

В метод onCreate добавим код для создания таблицы.

```
db.execSQL(SQL_CREATE_ENTRIES);
```

А в метод onUpgrade добавим код для удаления таблицы, а затем добавим вызов метода onCreate для создания таблицы. Таким образом апгрейд БД будет заключаться в удалении текущей таблицы и создании ее заново. Это понадобится при обновлении приложения на новую версию, без необходимости удаления старой версии приложения.

```
db.execSQL(SQL_DELETE_ENTRIES);
onCreate(db);
```

Теперь вернемся к классу DownloadTask. В самом начале конструкции try в методе doInBackground проинициализируем наш класс для работы с БД, откроем БД на запись и очистим таблицу торговых точек.

```
PointsDatabaseHelper mDbHelper = new PointsDatabaseHelper(
    MainActivity.this);
SQLiteDatabase db = mDbHelper.getWritableDatabase();
db.delete("POINTS", null, null);
```

Далее, после вызова метода reader.endObject(); добавим код для записи прочитанного объекта в БД.

```
ContentValues values = new ContentValues();
values.put("_id", id);
values.put("name", name);
values.put("address", address);
values.put("latitude", latitude);
values.put("longitude", longitude);
values.put("photo", photo);
db.insertWithOnConflict("Points", null, values,
    SQLiteDatabase.CONFLICT_REPLACE);
```

Последняя запись обеспечивает разрешение конфликтов при записи в таблицу точек с одинаковыми ID. Они просто будут заменяться (обновляться) в БД.

Перед тем, как завершить метод значением true, закроем БД.

```
        mDbHelper.close();
```

В метод onPostExecute добавим оповещение об успешности загрузки данных. Для этого используется класс Toast.

```
if (success) {
    Toast.makeText(MainActivity.this, "Данные успешно загр
        ужены", Toast.LENGTH_SHORT).show();
} else {
    Toast.makeText(MainActivity.this, "Ошибка загрузки дан
        ных", Toast.LENGTH_SHORT).show();
}
```

В метод onDownload класса MainActivity.java добавляем код для запуска процесса загрузки данных.

```
new DownloadTask().execute();
```

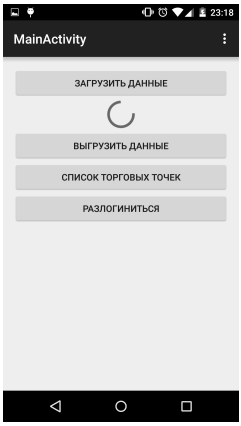

Рис. 55: Загрузка данных

Рис. 56: Оповещение о статусе загрузки

7.5. Список торговых точек

Окно со списком торговых точек будет содержать ViewPager, позволяющий переходить по страницам свайпом вправо или влево. На одной из страниц будет список точек, а на второй — карта. Кроме того, над ViewPager'ом будут табы для перехода по страницам. Создаем новую активити аналогично тому как мы делали это для создания активити авторизации. Activity Name задаем как PointsActivity, остальные параметры оставляем без изменений.

Откроется редактор activity_points.xml. Удаляем элемент TextView из корневого элемента RelativeActivity. Вместо него добавляем компонент ViewPager.

```
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</android.support.v4.view.ViewPager>
```

Внутрь только что добавленного компонента ViewPager добавим компонент PagerTitleStrip, он нужен для того, чтобы над VievPager'ом отобразить табы.

```
<android.support.v4.view.PagerTitleStrip
    android:id="@+id/pager_title_strip"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="top"
    android:background="#333333"
    android:textColor="#fff"
    android:paddingTop="10dp"
    android:paddingBottom="10dp" />
```

Кроме того у корневого элемента RelativeLayout удаляем свойства android:paddingLeft, android:paddingRight, android:paddingTop, android:paddingBottom, чтобы убрать отступы у экрана с торговыми точками.

Для функционирования ViewPager'а необходимо реализовать адаптер. Адаптер представляет собой класс, наследованный от абстрактного класса FragmentPagerAdapter. Класс будет вложенным в класс PointsActivity.java, который был создан, когда мы создавали активити.

```
public class PointsPagerAdapter extends
    FragmentPagerAdapter {
    public PointsPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    @Override
    public Fragment getItem(int position) {
        return null;
    }

    @Override
    public int getCount() {
        return 0;
    }
}
```

Метод getCount() должен возвращать количество страниц. У нас их будет две. Соответственно вместо 0 напишем 2. Кроме того, переопределим метод getPageTitle() родительского класса для того, чтобы задать заголовки страниц ViewPager'а. У нас это «Список точек» и «Карта».

```
@Override
public CharSequence getPageTitle(int position) {
    if (position == 0) {
        return "Список точек";
    } else {
        return "Карта";
    }
}
```

В класс PointsActivity добавим 2 поля — одно для хранения ссылки на ViewPager, второе для хранения экземпляра класса PointsPagerAdapter.

```
PointsPagerAdapter mPointsPagerAdapter;
ViewPager mViewPager;
```

В конец метода onCreate класса PointsActivity добавим код для инициализации адаптера и присваивания ссылки на компонент ViewPager. Также укажем ViewPager'у, что нужно использовать созданный нами адаптер.

```
mPointsPagerAdapter = new PointsPagerAdapter(
    getSupportFragmentManager());
mViewPager = (ViewPager) findViewById(R.id.pager);
mViewPager.setAdapter(mPointsPagerAdapter);
```

Далее нам необходимо реализовать 2 фрагмента, которые будут представлять собой страницы ViewPager'а. Первым создадим фрагмент для отображения списка торговых точек. Для этого слева в окне Project нажмем правой кнопкой мыши на имя пакета и выберем New → Fragment → Fragment (Blank). Fragment Name зададим как PointsFragment, остальные параметры оставим без изменений. В результате сгенерируется класс PointsFragment.java и fragment_points.xml. Перейдем к классу PointsFragment.java. Найдем метод newInstance(), удалим за ненадобностью параметры метода и передачу их фрагменту. В результате получим такой код метода.

```
public static PointsFragment newInstance() {
    PointsFragment fragment = new PointsFragment();
    return fragment;
}
```

Также удалим из класса PointsFragment константы ARG_PARAM1, ARG_PARAM2 и поля mParam1, mParam2. В методе onCreate удалим весь блок if, где идет присваивание зна-

чений удаленным полям. Удаляем метод onCreateView со всем его содержимым.

Далее, после названия класса изменим родительский класс с Fragment на ListFragment.

```
public class PointsFragment extends ListFragment {
...
}
```

Далее, для отображения данных списком в ListView следует создать класс Adapter, который отвечал бы за представление данных в списке. В данном случае источником данных является БД, и удобнее всего использовать CursorAdapter в качестве родительского класса. Для этого нажмем правой кнопкой мыши на пакет приложения слева в окне проекта и выберем New → Java Class. Name выставим как PointsCursorAdapter. В созданном классе PointsCursorAdapter.java после имени класса допишем extends CursorAdapter, затем нажмем Alt+Enter и в появившемся диалоге выберем Implement methods. В результате IDE сгенерирует следующий код.

```
public class PointsCursorAdapter extends CursorAdapter {
    @Override
    public View newView(Context context, Cursor cursor,
        ViewGroup parent) {
        return null;
    }

    @Override
    public void bindView(View view, Context context,
        Cursor cursor) {

    }
}
```

Добавим в класс конструктор.

```
public PointsCursorAdapter(Context context, Cursor c, int
    flags) {
    super(context, c, flags);
}
```

Теперь создадим лэйаут, отвечающий за внешний вид одной строки в списке торговых точек. Для этого нажмем правой кнопкой мыши на подкаталог res/layout и выберем New → Layout resource file. File name зададим как item_point. Остальные параметры оставим без изменений.

Нажмем OK. В визуальном редакторе перейдем на вкладку Text. В корневой элемент LinearLayout добавим два контрола TextView.

```
<TextView
    android:id="@+id/name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/
        textAppearanceLarge" />
<TextView
    android:id="@+id/address"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/
        textAppearanceMedium" />
```

Вернемся к классу PointsCursorAdapter. В метод newView вместо return null добавим следующий код.

```
return LayoutInflater.from(context).inflate(R.layout.
    item_point, parent, false);
```

В метод bindView добавим код для задания значений компонентам вьюшки.

```
TextView nameView = (TextView) view.findViewById(R.id.name
    );
TextView addressView = (TextView) view.findViewById(R.id.
    address);

String name = cursor.getString(cursor.
    getColumnIndexOrThrow("name"));
String address = cursor.getString(cursor.
    getColumnIndexOrThrow("address"));

nameView.setText(name);
addressView.setText(address);
```

Перейдем к классу PointsFragment. Переопределим родительский метод onActivityCreated. В нем мы проинициализируем вспомогательный класс для работы с БД, курсор для работы с БД, а также адаптер для отображения данных из БД в списке. Зададим созданный нами адаптер списку.

```
@Override
public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    PointsDatabaseHelper helper = new PointsDatabaseHelper
        (getActivity());
    SQLiteDatabase db = helper.getReadableDatabase();
    Cursor pointsCursor = db.rawQuery("SELECT * FROM
        POINTS", null);
    PointsCursorAdapter pointsAdapter = new
        PointsCursorAdapter(getActivity(), pointsCursor,
        0);
    setListAdapter(pointsAdapter);
}
```

Перейдем к вложенному в PointsActivity классу PointsPagerAdapter. В метод getItem() добавим следующий код.

```
return PointsFragment.newInstance();
```

Перейдем к названию класса PointsActivity, допишем после названия implements PointsFragment.OnFragmentInteractionListener. Нажмем Alt+Enter и выберем Implement methods. Нажмем OK. Этот обработчик нам пригодится далее, когда мы будем обрабатывать выбор торговой точки для отображения ее на карте.

Перейдем к классу MainActivity. В метод onList() добавим код для запуска активити с торговыми точками.

```
startActivity(new Intent(this, PointsActivity.class));
```

7.6. Карта

Первым делом нам нужно получить Google API Key. Подробно получение ключа описано здесь `https://developers.google.com/maps/documentation/android/signup`. В результате мы получим ключ, что-то вроде AIzaSyBUD5SBJPAK2...

Откроем манифест приложения (файл AndroidManifest.xml) и добавим внутрь элемента <application> следующую строку.

```
<meta-data android:name="com.google.android.geo.API_KEY"
    android:value="GOOGLE API KEY"/>
```

Вместо «GOOGLE API KEY» впишем полученный нами ключ. Далее, найдем в окне Project слева каталог Gradle Scripts, среди его элементов находим и открываем файл build.gradle (Module: app). По-

Рис. 57: Список торговых точек

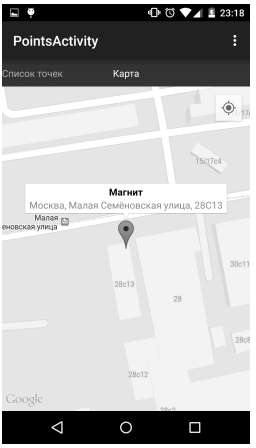

Рис. 58: Карта

сле строки compile 'com.android.support:appcompat-v7:22.2.0' добавим следующие строки.

```
compile 'com.google.android.gms:play-services-maps:7.5.0'
compile 'com.google.android.gms:play-services-location
    :7.5.0'
```

Эти библиотеки понадобятся нам для работы с картой. Наверху появится предложение синхронизировать gradle файл, нажмем «Sync Now».

Перейдем к вложенному в PointsActivity классу PointsPagerAdapter. Изменим метод getItem() следующим образом.

```
switch (position) {
    case 0:
        return PointsFragment.newInstance();
    case 1:
    default:
        SupportMapFragment maps = SupportMapFragment.
            newInstance();
        maps.getMapAsync(PointsActivity.this);
        return maps;
}
```

Нажмем на PointsActivity.this, затем нажмем на Alt+Enter и выберем Make 'PointsActivity' implement

'com.google.android.gms.maps.OnMapReadyCallback', затем нажмем OK. IDE сгенерирует метод onMapReady() в классе PointsActivity.java.

В класс PointsActivity добавим поле для хранения ссылки на объект карты.

```
private GoogleMap googleMap;
```

В метод onMapReady() сохраним ссылку на объект карты.

```
this.googleMap = googleMap;
```

Теперь напишем экземпляр класса AsyncTask аналогично тому, как мы это уже делали для получения списка торговых точек, для отображения их на карте. Класс будет вложенным в PointsActivity. Назовем его GetPointsTask.

```
public class GetPointsTask extends AsyncTask<Void, Void,
    Boolean> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }

    @Override
    protected Boolean doInBackground(Void... params) {
        return true;
    }

    @Override
    protected void onPostExecute(Boolean success) {
        super.onPostExecute(success);
    }
}
```

Создадим класс Point, представляющий собой модель торговой точки. Для этого слева в окне проекта нажмем правой кнопкой мыши на пакет приложения, выберем New → Java Class. Name зададим Point. Нажмем OK. Добавим в класс поля id, name, address, latitude, longitude, updated и геттеры/сеттеры для них. Должен получиться такой код.

```
public class Point {
    private int id;
    private String name;
    private String address;
    private double latitude;
    private double longitude;
    private boolean photo;
    private String image;
    private int updated;

    public Point() {
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public double getLatitude() {
        return latitude;
    }

    public void setLatitude(double latitude) {
        this.latitude = latitude;
    }

    public double getLongitude() {
        return longitude;
    }
```

```
    public void setLongitude(double longitude) {
        this.longitude = longitude;
    }

    public boolean isPhoto() {
        return photo;
    }

    public void setPhoto(boolean photo) {
        this.photo = photo;
    }

    public String getImage() {
        return image;
    }

    public void setImage(String image) {
        this.image = image;
    }

    public int getUpdated() {
        return updated;
    }

    public void setUpdated(int updated) {
        this.updated = updated;
    }
}
```

Переопределим методы equals() и hashCode() класса Object, от которого наследуются все классы в Java. Корректная реализация этих методов достаточно хорошо освещена в сети. Это пригодится нам далее для сравнения объектов.

```
@Override
public boolean equals(Object obj) {
    if (obj == this) {
        return true;
    }
    if (obj == null || obj.getClass() != this.getClass())
        {
        return false;
    }
    Point p = (Point) obj;
```

```
        return id == p.id
            && latitude == p.latitude
            && longitude == p.longitude
            && (name == p.name
            || (name != null && name.equals(p.getName())))
            && (address == p.address
            || (address != null && address.equals(p.getAddress
                ())));
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result
            + ((name == null) ? 0 : name.hashCode());
        result = prime * result + id;
        result = prime * result
            + ((address == null) ? 0 : address.hashCode());
        return result;
    }
```

В классе PointsActivity добавляем поле для хранения списка торговых точек и поле для хранения выбранной точки.

```
private List<Point> points;
private Point currentPoint = null;
```

В метод doInBackground вложенного класса GetPointsTask добавим код для чтения списка торговых точек из БД.

```
@Override
protected Boolean doInBackground(Void... params) {
    PointsDatabaseHelper mDbHelper = new
        PointsDatabaseHelper(PointsActivity.this);
    SQLiteDatabase db = mDbHelper.getReadableDatabase();
    points = new ArrayList<>();
    allMarkersMap.clear();

    Cursor cursor = db.rawQuery("SELECT * FROM POINTS",
        null);
    if (cursor.moveToFirst()) {
        do {
            Point p = new Point();
            p.setId(cursor.getInt(cursor.getColumnIndex("
                _id")));
```

```
                p.setName(cursor.getString(cursor.
                    getColumnIndex("name")));
                p.setAddress(cursor.getString(cursor.
                    getColumnIndex("address")));
                p.setLatitude(cursor.getDouble(cursor.
                    getColumnIndex("latitude")));
                p.setLongitude(cursor.getDouble(cursor.
                    getColumnIndex("longitude")));
                points.add(p);
            } while ((cursor.moveToNext()));
        }

        return true;
    }
```

В класс PointsActivity добавим еще одно поле для хранения соответствия точки ее маркеру на карте.

```
private Map<Point, Marker> allMarkersMap = new HashMap<
    Point, Marker>();
```

В метод onPostExecute добавим код для размещения торговых точек на карте, а также сохранения соответствия маркера точке.

```
for (Point p : points) {
    LatLng latLng = new LatLng(p.getLatitude(), p.
        getLongitude());
    Marker m = googleMap.addMarker(new MarkerOptions().
        position(latLng).title(p.getName())
            .snippet(p.getAddress()));
    allMarkersMap.put(p, m);
}
```

В метод onMapReady() добавим код для запуска процесса получения и размещения на карте точек.

```
new GetPointsTask().execute();
```

Туда же добавим отображение текущего местоположения на карте.

```
this.googleMap.setMyLocationEnabled(true);
```

Теперь перейдем к классу в PointsFragment. Внизу класса найдем интерфейс OnFragmentInteractionListener. Удалим сигнатуру метода public void onFragmentInteraction(Uri uri); и добавим вместо нее такую.

```
public void onPointSelected(Cursor c);
```

Также, удалим метод onButtonPressed() из класса PointsFragment.

В метод onActivityCreated добавим следующий код.

```
getListView().setOnItemClickListener(this);
```

Кликнем на this, затем нажмем Alt+Enter, выберем Make 'PointsFragment' implement 'android.widget.AdapterView.OnItemClickListener' и нажмем OK. IDE сгенерирует метод onItemClick() в классе PointsFragment.java.

В сгенерированный метод onItemClick() добавим следующий код.

```
if (mListener!=null) {
    mListener.onPointSelected((Cursor) getListAdapter().
        getItem(position));
}
```

Откроем класс Point.java. Добавим конструктор, который по данным из курсора заполняет поля объекта.

```
public Point(Cursor c) {
    this.id = c.getInt(c.getColumnIndex("_id"));
    this.name = c.getString(c.getColumnIndex("name"));
    this.address = c.getString(c.getColumnIndex("address")
        );
    this.latitude = c.getDouble(c.getColumnIndex("latitude
        "));
    this.longitude = c.getDouble(c.getColumnIndex("
        longitude"));
    this.photo = c.getInt(c.getColumnIndex("photo")) == 1;
    this.image = c.getString(c.getColumnIndex("image"));
    this.updated = c.getInt(c.getColumnIndex("updated"));
}
```

Перейдем к классу PointsActivity. Удалим метод onFragmentInteraction(). Добавим реализацию нового метода, который мы добавили в интерфейс OnFragmentInteractionListener. И добавим в него код для перехода к переданной точке на карте.

```
@Override
public void onPointSelected(Cursor c) {
    Point p = new Point(c);
```

```
        googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom
            (new LatLng(p.getLatitude(), p.getLongitude()),
            18));
        mViewPager.setCurrentItem(1);
        Marker m = allMarkersMap.get(p);
        if (m!=null) {
            m.showInfoWindow();
        }
    }
```

Теперь, чтобы отлавливать нажатие на информационное окошко над маркером, нам нужно реализовать интерфейс OnInfoWindowClickListener. Для этого в методе onMapReady() добавим строку this.googleMap.setOnInfoWindowClickListener(this);, кликнем на this, нажмем Alt+Enter, выберем Make 'PointsActivity' implement 'com.google.android.gms.maps.GoogleMap.OnInfoWindowClickListener' и нажмем OK. В сгенерированный IDE метод onInfoWindowClick() добавим следующий код для поиска торговой точки по маркеру, на который нажал пользователь.

```
    @Override
    public void onInfoWindowClick(Marker marker) {
        if (allMarkersMap.containsValue(marker)) {
            for (Map.Entry e : allMarkersMap.entrySet()) {
                if (e.getValue().equals(marker)) {
                    currentPoint = (Point) e.getKey();
                }
            }
        }

        if (currentPoint!=null) {
            Toast.makeText(this, "Выбрана торговая точка: " +
                p.getName(), Toast.LENGTH_SHORT).show();
        }
    }
```

7.7. Список товаров

Перейдем к классу PointsDatabaseHelper. Изменим версию БД на 2.

```
    public static final int DATABASE_VERSION = 2;
```

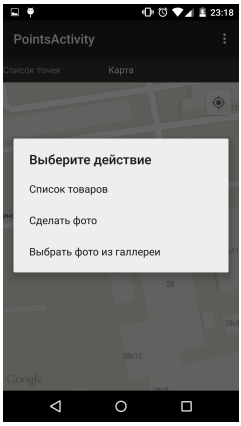

Рис. 59: Меню торговой точки

Рис. 60: Список товаров

Добавим в класс две константы с SQL для создания и удаления списка товаров.

```
private static final String SQL_CREATE_ITEMS =
    "CREATE TABLE ITEMS (" +
            "_id INTEGER PRIMARY KEY," +
            "point_id INTEGER," +
            "name TEXT," +
            "count INTEGER," +
            "updated INTEGER DEFAULT 0" +
            " )";

private static final String SQL_DELETE_ITEMS =
    "DROP TABLE IF EXISTS ITEMS";
```

Добавим в метод onCreate() строку.

```
db.execSQL(SQL_CREATE_ITEMS);
```

Добавим в метод onUpgrade() над строкой onCreate(db); строку.

```
db.execSQL(SQL_DELETE_ITEMS);
```

Перейдем в класс MainActivity.java. Найдем метод doInBackground вложенного класса DownloadTask. Найдем код, где парсится объект торговой точки. Перед последним блоком else добавляем код для чтения списка товаров, если они есть, и добавления их в таблицу БД.

```
} else if (key.equals("items")) {
    int itemId = 0;
    int itemPointId = 0;
    String itemName = null;
    int itemCount = 0;

    reader.beginArray();
    while (reader.hasNext()) {
        reader.beginObject();
        while (reader.hasNext()) {
            String key_ = reader.nextName();
            if (key_.equals("id")) {
                itemId = reader.nextInt();
            } else if (key_.equals("name")) {
                itemName = reader.nextString();
            } else if (key_.equals("point_id")) {
                itemPointId = reader.nextInt();
            } else if (key_.equals("count")) {
                itemCount = reader.nextInt();
            } else {
                reader.skipValue();
            }
        }
        reader.endObject();

        ContentValues values = new ContentValues();
        values.put("_id", itemId);
        values.put("name", itemName);
        values.put("point_id", itemPointId);
        values.put("count", itemCount);

        db.insertWithOnConflict("Items", null, values,
            SQLiteDatabase.CONFLICT_REPLACE);

    }
    reader.endArray();
} else {
    reader.skipValue();
}
```

Выше найдем строку, где мы очищали таблицу POINTS, и добавим очистку таблицы ITEMS.

```
db.delete("ITEMS", null, null);
```

Создадим новую Activity для отображения списка товаров торговой точки аналогично тому, как мы это уже делали. Activity Name

зададим как ItemsActivity, остальные параметры оставим без изменения. Откроем сгенерированный файл activity_items.xml в режиме текста и удалим его содержимое. Вместо того, что там было, добавим компонент ListView.

```
<ListView
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/listView"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:descendantFocusability="beforeDescendants"
/>
```

Откроем манифест приложения (файл AndroidManifest.xml), найдем элемент активити с именем ItemsActivity, после свойства android:label добавим свойство android:windowSoftInputMode="adjustPan". Элемент активити будет выглядеть так:

```
<activity
    android:name=".ItemsActivity"
    android:label="@string/title_activity_items"
    android:windowSoftInputMode="adjustPan">
</activity>
```

Откроем класс ItemsActivity.java и добавим в него поле для хранения ссылки на ListView.

```
private ListView list;
```

В методе onCreate() присвоим этому полю значение.

```
list = (ListView) findViewById(R.id.listView);
```

Перейдем к классу PointsActivity. В методе onInfoWindowClick вместо строки

```
Toast.makeText(this, "Selected trade point: " + p.getName
    (), Toast.LENGTH_SHORT).show();
```

добавим следующий код для запуска активити со списком товаров.

```
Intent itemsActivity = new Intent(this, ItemsActivity.
    class);
itemsActivity.putExtra("id", currentPoint.getId());
startActivity(itemsActivity);
```

Здесь мы передаем запускаемой активити идентификатор выбранной торговой точки.

Вернемся к классу ItemsActivity.java. Добавим в класс поле для хранения идентификатора выбранной торговой точки.

```
private int id;
```

А в метод onCreate() после вызова метода setContentView() добавим код для получения переданного ранее идентификатора выбранной торговой точки.

```
if (getIntent()!=null) {
    id = getIntent().getIntExtra("id", 0);
}
```

Создадим класс Item, представляющий собой модель товара некой торговой точки. Для этого слева в окне проекта нажмем правой кнопкой мыши на пакет приложения, выберем New → Java Class. Name зададим Item. Нажмем OK. Добавим в класс поля id, name, point_id, count, updated и геттеры/сеттеры для них. Должен получится такой код. Сразу добавим методы для сравнения объектов и конструктор, принимающий в качестве параметра курсор аналогично тому, как мы делали это для класса Point.java.

```
public class Item {
    private int id;
    private int pointId;
    private String name;
    private int count;
    private int updated;

    public Item() {
    }

    public Item(Cursor c) {
      this.id = c.getInt(c.getColumnIndex("_id"));
      this.name = c.getString(c.getColumnIndex("name"));
      this.pointId = c.getInt(c.getColumnIndex("point_id")
          );
      this.count = c.getInt(c.getColumnIndex("count"));
      this.updated = c.getInt(c.getColumnIndex("updated"))
          ;
    }
```

```
public int getId() {
  return id;
}

public void setId(int id) {
  this.id = id;
}

public String getName() {
  return name;
}

public void setName(String name) {
  this.name = name;
}

public int getPointId() {
  return pointId;
}

public void setPointId(int pointId) {
  this.pointId = pointId;
}

public int getCount() {
  return count;
}

public void setCount(int count) {
  this.count = count;
}

public int getUpdated() {
  return updated;
}

public void setUpdated(int updated) {
  this.updated = updated;
}

@Override
public boolean equals(Object obj) {
    if (obj == this) {
        return true;
    }
    if (obj == null || obj.getClass() != this.getClass
        ()) {
        return false;
    }
```

```
        Item p = (Item) obj;
        return id == p.id
                && pointId == p.pointId
                && (name == p.name
                || (name != null && name.equals(p.getName
                    ())));
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result
                + ((name == null) ? 0 : name.hashCode());
        result = prime * result + id;
        result = prime * result + pointId;
        return result;
    }

}
```

Вернемся к классу ItemsActivity. Добавим поле для хранения списка товаров торговой точки.

```
private List<Item> items = new ArrayList<>();
```

Создадим адаптер для списка товаров. Будем наследовать наш адаптер от класса ArrayAdapter. Нажмем правой кнопкой мыши на пакет приложения слева в окне проекта и выберем New → Java Class. Name выставим как ItemsArrayAdapter. После имени класса допишем extends ArrayAdapter<Item>, затем нажмем Alt+Enter и в появившемся диалоге выберем Create constructor matching super. В списке конструкторов выберем ArrayAdapter(context:Context, resource:int, objects:List<T>). В результате получим следующий код.

```
public class ItemsArrayAdapter extends ArrayAdapter<Item>
    {
    public ItemsArrayAdapter(Context context, int resource
        , List<Item> objects) {
        super(context, resource, objects);
    }
}
```

Теперь создадим файл, отвечающий за внешний вид одной строки в списке товаров. Для этого нажмем правой кнопкой мыши на подкаталог res/layout и выберем New → Layout resource file. File

name зададим как item_item. Остальные параметры оставим без изменений. Нажмем OK. В визуальном редакторе перейдем на вкладку Text. В корневом элементе LinearLayout изменим значение свойства android:orientation с "vertical"на "horizontal"и добавим два контрола: TextView и EditText.

```
<TextView
    android:id="@+id/item_name"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:text="New Text"
    android:layout_gravity="center_vertical"
    android:gravity="center_vertical"
    />

<EditText
    android:id="@+id/item_count"
    android:layout_width="100dp"
    android:layout_height="match_parent"
    android:layout_weight="0"
    android:inputType="number"
    android:ems="10"
    android:layout_gravity="center_vertical"
    android:gravity="center_vertical"
    />
```

Вернемся к классу ItemsArrayAdapter. Переопределим родительский метод getView() следующим образом.

```
@Override
public View getView(int position, View convertView,
    ViewGroup parent) {
    Item item = getItem(position);

    if (convertView == null) {
        convertView = LayoutInflater.from(getContext()).
            inflate(R.layout.item_item, parent, false);
    }
    TextView name = (TextView) convertView.findViewById(R.
        id.item_name);
    EditText count = (EditText) convertView.findViewById(R
        .id.item_count);
    name.setText(item.getName());
```

```
        count.setText(String.valueOf(item.getCount()));

        return convertView;
    }
```

Для того, чтобы отлавливать изменения в текстовом поле, создадим класс CountTextWatcher, реализующий интерфейс TextWatcher.

```
public class CountTextWatcher implements TextWatcher {

    private final EditText edit;

    public CountTextWatcher(EditText edit) {
        this.edit = edit;
    }

    @Override
    public void beforeTextChanged(CharSequence s, int
        start, int count, int after) {

    }

    @Override
    public void onTextChanged(CharSequence s, int start,
        int before, int count) {

    }

    @Override
    public void afterTextChanged(Editable s) {
        Item item = (Item) edit.getTag();
        try {
            item.setCount(Integer.parseInt(s.toString()));
        } catch (Exception e) {
            item.setCount(0);
        }
    }
}
```

В качестве параметра конструктор принимает ссылку на текстовое поле. В методе afterTextChanged() из свойства tag вынимаем экземпляр класса товара, получаем значение в текстовом поле и присваиваем его свойству count в классе товара.

В методе getView перед return convertView; добавляем следующий код.

```
count.setTag(item);
count.addTextChangedListener(new CountTextWatcher(count));
```

Вернемся к классу ItemsActivity. Добавим поле для хранения ссылки на адаптер.

```
private ItemsArrayAdapter itemsAdapter;
```

В методе onCreate после присваивания полю list ссылки на список добавим код для инициализации адаптера. Также зададим созданный нами адаптер списку.

```
itemsAdapter = new ItemsArrayAdapter(this, 0, items);
list.setAdapter(itemsAdapter);
```

Реализуем экземпляр класса AsyncTask аналогично тому, как мы это уже делали, для получения списка товаров заданной торговой точки. Класс будет вложенным в ItemsActivity. Назовем его GetItemsTask.

```
public class GetItemsTask extends AsyncTask<Void, Void,
    Boolean> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }

    @Override
    protected Boolean doInBackground(Void... params) {
        return true;
    }

    @Override
    protected void onPostExecute(Boolean success) {
        super.onPostExecute(success);
    }
}
```

В метод doInBackground вложенного класса GetItemsTask добавим код для чтения списка товаров торговой точки из БД.

```
@Override
protected Boolean doInBackground(Void... params) {
    PointsDatabaseHelper mDbHelper = new
        PointsDatabaseHelper(ItemsActivity.this);
```

```
    SQLiteDatabase db = mDbHelper.getReadableDatabase();
    items.clear();
    Cursor cursor = db.rawQuery("SELECT * FROM ITEMS WHERE
         point_id = ?", new String[]{String.valueOf(id)});
    if (cursor.moveToFirst()) {
        do {
            Item i = new Item(cursor);
            items.add(i);
        } while ((cursor.moveToNext()));
    }

    return true;
}
```

В метод onPostExecute добавим код для оповещения адаптера о том, что данные изменились.

```
@Override
protected void onPostExecute(Boolean success) {
    super.onPostExecute(success);
    itemsAdapter.notifyDataSetChanged();
}
```

В конец метода onCreate добавим код для запуска процесса получения списка товаров.

```
new GetItemsTask().execute();
```

Теперь перейдем к классу CountTextWatcher, вложенному в класс ItemsArrayAdapter. В методе afterTextChanged() добавим код для сохранения измененного значения в БД и изменим поле updated у товара и торговой точки на 1, что будет означать, что информация была обновлена.

```
PointsDatabaseHelper mDbHelper = new PointsDatabaseHelper(
    getContext());
SQLiteDatabase db = mDbHelper.getWritableDatabase();

ContentValues contentValues = new ContentValues();
contentValues.put("count", item.getCount());
contentValues.put("updated", 1);
db.update("ITEMS", contentValues, "_id = ?", new String[]{
    String.valueOf(item.getId())});
```

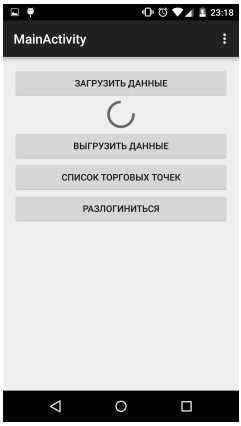

Рис. 61: Выгрузка товаров

Рис. 62: Оповещение о статусе выгрузки

```
contentValues = new ContentValues();
contentValues.put("updated", 1);
db.update("POINTS", contentValues, "_id = ?", new String
    []{String.valueOf(item.getPointId())});
db.close();
```

7.8. Выгрузка товаров на сервер

Перейдем к классу MainActivity.java. Напишем еще один экземпляр класса AsyncTask аналогично тому, как мы это уже делали. Класс будет вложенным в MainActivity. Назовем его UploadTask.

```
public class UploadTask extends AsyncTask<Void, Void,
    Boolean> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
    }

    @Override
    protected Boolean doInBackground(Void... params) {
        return true;
    }
```

```
    @Override
    protected void onPostExecute(Boolean success) {
        super.onPostExecute(success);
    }
}
```

В метод onPreExecute добавим отображение прогрессбара, а в метод onPostExecute — сокрытие прогрессбара и оповещение о статусе выгрузки.

```
@Override
protected void onPreExecute() {
    super.onPreExecute();
    progressBar.setVisibility(View.VISIBLE);
}

@Override
protected void onPostExecute(Boolean success) {
    progressBar.setVisibility(View.GONE);
    if (success) {
        Toast.makeText(MainActivity.this, "Данные успешно
            выгружены", Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(MainActivity.this, "Ошибка выгрузки
             данных", Toast.LENGTH_SHORT).show();
    }
}
```

Перейдем к методу doInBackground(). Добавим код, в котором пройдемся по всем измененным торговым точкам и по всем измененным товарам, после чего отправим их на сервер.

```
try {
    PointsDatabaseHelper mDbHelper = new
        PointsDatabaseHelper(MainActivity.this);
    SQLiteDatabase db = mDbHelper.getWritableDatabase();
    Cursor cursor = db.rawQuery("SELECT * FROM POINTS
        WHERE updated = 1", null);
    if (cursor.moveToFirst()) {
        do {
            Point p = new Point(cursor);
            Cursor cursorItems = db.rawQuery("SELECT *
                FROM ITEMS WHERE point_id = ? and updated
                = 1", new String[]{String.valueOf(p.getId
                ())});
```

```
            if (cursorItems.moveToFirst()) {
                do {
                    Item item = new Item(cursorItems);

                    URL url = new URL("http://maps.
                        dvinemnauku.ru/Map/updateGood.php?
                        point_id=" + item.getPointId() +
                        "&good_id=" + item.getId() + "&
                        count=" + item.getCount() + "&
                        auth_key="+authKey);
                    HttpURLConnection c = (
                        HttpURLConnection)url.
                        openConnection();
                    c.setRequestMethod("GET");
                    c.connect();
                    int status = c.getResponseCode();

                    if (status != 200) {
                        db.close();
                        return false;
                    }

                } while ((cursorItems.moveToNext()));
            }
        } while ((cursor.moveToNext()));
    }
    db.close();
} catch (Exception e) {
    e.printStackTrace();
}
return true;
```

Добавим в метод onUpload() запуск процесса выгрузки товаров.

```
public void onUpload(View v) {
    new UploadTask().execute();
}
```

7.9. Работа с фото

Добавим в манифест приложения разрешение на запись на карту памяти.

```
<uses-permission android:name="android.permission.
    WRITE_EXTERNAL_STORAGE" />
```

Перейдем к классу PointsActivity и добавим в него следующие константы.

```
private static final int TAKE_A_PICTURE = 1;
private static final int PICK_A_PICTURE = 2;
```

Также добавим поле для хранения пути к файлу.

```
private String filePath = null;
```

Добавим метод для создания файла, куда будет сохранена фотография.

```
private File createPhotoFile() {
    File image = null;
    try {
        File storageDir = Environment.
            getExternalStoragePublicDirectory(
                Environment.DIRECTORY_PICTURES);
        image = File.createTempFile(
                "photo_" + String.valueOf(currentPoint.
                    getId()),
                ".jpg",
                storageDir
        );
    } catch (IOException e) {
        Log.e("SavePhoto()", "I/O error", e);
    }
    return image;
}
```

Добавим в этот же класс метод для старта активити фотографирования.

```
public void takeAPicture() {
    Intent takeAPicture = new Intent(MediaStore.
        ACTION_IMAGE_CAPTURE);
    if (takeAPicture.resolveActivity(getPackageManager())
        != null) {
        File photo  = createPhotoFile();
        filePath = photo.getAbsolutePath();
        takeAPicture.putExtra(MediaStore.EXTRA_OUTPUT,
                Uri.fromFile(photo));
        startActivityForResult(takeAPicture,
            TAKE_A_PICTURE);
    }
}
```

А также добавим метод для выбора фото из галереи.

```
public void pickAPicture() {
    Intent takeAPicture = new Intent(Intent.ACTION_PICK,
            MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
    if (takeAPicture.resolveActivity(getPackageManager())
        != null) {
        startActivityForResult(takeAPicture,
            PICK_A_PICTURE);
    }
}
```

Здесь же переопределим метод родительского класса onActivityResult(). Здесь мы сначала проверим успешность получения фото. В случае, когда мы получаем фото с камеры, все просто. Путь мы уже задали в методе takeAPicture(), и теперь фото сохранено по указанному пути. В случае же с выбором фотографии из галереи нам предстоит выяснить местоположение выбранного файла. Добавим в onActivityResult следующий код.

```
if (resultCode == RESULT_OK) {
    if (requestCode == PICK_A_PICTURE) {
        Uri fileUri =  data.getData();
        String[] filePathColumn = { MediaStore.Images.
            Media.DATA };

        Cursor cursor = getContentResolver().query(fileUri
            , filePathColumn, null, null, null );
        cursor.moveToFirst();

        int columnIndex = cursor.getColumnIndex(
            filePathColumn[0] );
        filePath = cursor.getString( columnIndex );
        cursor.close();
    }
}
```

Теперь сохраним полученный путь в БД. В конце внешнего блока if добавим код для сохранения пути к файлу в БД. Этот код будет общим для обоих вариантов получения фото (как с камеры, так и из галереи). Заодно пометим, что данная торговая точка была обновлена.

```
PointsDatabaseHelper mDbHelper = new PointsDatabaseHelper(
    PointsActivity.this);
SQLiteDatabase db = mDbHelper.getWritableDatabase();
```

```
ContentValues contentValues = new ContentValues();
contentValues.put("image", filePath);
contentValues.put("updated", 1);
db.update("POINTS", contentValues, "_id = ?", new String
    []{String.valueOf(currentPoint.getId())});
db.close();
```

Теперь перейдем к методу onInfoWindowClick() и изменим код в блоке if (currentPoint!=null) ... так, чтобы вместо открытия списка товаров появлялся диалог выбора между просмотром списка товаров и получения фото с камеры или из галереи.

```
if (currentPoint!=null) {
    AlertDialog.Builder builder = new AlertDialog.Builder(
        this);
    builder.setTitle("Actions")
    .setItems(new String[]{"Список товаров", "Сделать фото
        ", "Выбрать фото из галереи"}, new DialogInterface
        .OnClickListener() {
        public void onClick(DialogInterface dialog, int
            which) {
            switch (which) {
                case 0:
                    Intent itemsActivity = new Intent(
                        PointsActivity.this, ItemsActivity
                        .class);
                    itemsActivity.putExtra("id",
                        currentPoint.getId());
                    startActivity(itemsActivity);
                    break;
                case 1:
                    takeAPicture();
                    break;
                case 2:
                    pickAPicture();
                    break;
            }
        }
    });
    Dialog d = builder.create();
    d.show();
}
```

7.10. Отправка фото на сервер

Перейдем к классу UploadTask, вложенному в MainActivity.java. Найдем метод doInBackground. После строки Point p = new Point(cursor); добавим код для отправки файла с фотографией на сервер (если делали фото). Файл мы будем отправлять так же, как мы бы отправляли файл с HTML формы, и если со стороны сервера получение файла таким образом — простая задача, то со стороны клиента действий будет определенно больше. Первым делом проверим, есть ли что отправлять.

```
if (!TextUtils.isEmpty(p.getImage())) {
    // File sending
}
```

Далее, внутри этого блока кода проинициализируем класс для обращения к сети и выставим заголовки POST запроса.

```
URL url = new URL("http://maps.dvinemnauku.ru/Map/
    postImage.php?point_id=" + p.getId() + "&auth_key="+
    authKey);
HttpURLConnection c = (HttpURLConnection)url.
    openConnection();
String boundary = "3676416B-9AD6-440C-B3C8-FC66DDC7DB45";
c.setRequestMethod("POST");
c.setRequestProperty("Connection", "Keep-Alive");
c.setRequestProperty("Cache-Control", "no-cache");
c.setRequestProperty("Content-Type", "multipart/form-data;
    boundary=" + boundary);
c.setUseCaches(false);
c.setDoOutput(true);
```

Здесь переменная boundary содержит последовательность символов, которая не должна встречаться в отправляемых данных. Подробнее об этом способе отправки данных на сервере вы можете ознакомиться здесь — (https://ru.wikipedia.org/wiki/Multipart/form-data).

Дальше нам нужно проинициализировать поток ввода, из которого мы будем читать содержимое файла для отправки на сервер.

```
Uri uri = Uri.parse(p.getImage());
InputStream is = null;
File f = new File(p.getImage());
```

```
if ( f.exists() ){
    try{
        is = new FileInputStream(f);
    } catch( Exception e){
        Log.e("UploadTask", "I/O error", e);
    }
}
```

Если файл существует и доступен для чтения, инициализируем поток вывода, отправляем содержимое файла на сервер и закрываем потоки ввода и вывода.

```
if (is != null) {
    DataOutputStream dos = new DataOutputStream(c.
        getOutputStream());
    dos.writeBytes("--" + boundary + "\r\n");
    dos.writeBytes("Content-Disposition: form-data; name
        =\"image\";filename=\""
            + "photo.jpg" + "\"" + "\r\n");
    dos.writeBytes("\r\n");

    int bytesAvailable = is.available();
    int bufferSize = Math.min(bytesAvailable, Integer.
        MAX_VALUE);
    byte[] buffer = new byte[bufferSize];
    while (is.read(buffer, 0, bufferSize) != -1) {
        dos.write(buffer, 0, bufferSize);
    }

    dos.writeBytes("\r\n");
    dos.writeBytes("--" + boundary + "--" + "\r\n");

    dos.flush();
    dos.close();
    is.close();
}
```

Далее проверяем статус запроса, и если что-то пошло не так, прерываем выгрузку данных.

```
c.connect();
int status = c.getResponseCode();
if (status != 200) {
    return false;
}
```

Список литературы

[1] В.В. Осокин, Р.Ф. Алимов. *Система онлайн сопровождения научно-образовательных процессов ДвинемНауку*. Интеллектуальные системы, том 18, выпуск 1, 2014, 76-102.

[2] В.В. Осокин, Р.А. Ахунов. *Сайт семинаров ректора МГУ*. Интеллектуальные системы, том 18, выпуск 1, 2014, 103-110.

[3] В.В. Осокин, В.А. Бендик. *Android как платформа для разработки корпоративных приложений на примере CRM*. Интеллектуальные системы, том 18, выпуск 2, 2014, 179-188.

[4] В.В. Осокин, М.В. Шегай. *Онлайн синхронизация данных в распределённой системе*. Интеллектуальные системы, том 18, выпуск 2, 2014, 189-196.

[5] В.В. Осокин, Т.Д. Аипов, З.А. Ниязова. *О классификации изображений и музыкальных файлов*. Интеллектуальные системы. Теория и приложения, том 19, выпуск 1, 2015, 49-70.

[6] В.В. Осокин, Р.Ф. Алимов, Р.Р. Хайдаров. *Основы реализации поисковой системы*. Интеллектуальные системы. Теория и приложения, том 19, выпуск 1, 2015, 71-98.

[7] В.В. Осокин, Н.Е. Горожанин, В.Р. Вавилов, Д.У. Камилов. *Основы реализации географических онлайн-карт*. Интеллектуальные системы. Теория и приложения, том 19, выпуск 2, 2015, 97-122.

[8] Редактор Notepad++ для Windows https://notepad-plus-plus.org/

[9] Редактор Gedit для Linux https://ru.wikipedia.org/wiki/Gedit

[10] Справочник по HTML-тегам http://htmlbook.ru/htm

[11] Справочник по CSS-атрибутам http://htmlbook.ru/css

[12] Форматы задания цветов в CSS
http://htmlbook.ru/css/value/color

[13] Веб-сервер Apache https://ru.wikipedia.org/wiki/Apache

[14] Регистратор доменных имен R01 http://r01.ru

[15] Язык программирования PHP https://ru.wikipedia.org/wiki/PHP

[16] Официальный сайт языка PHP https://secure.php.net

[17] Страница Википедии о СУБД MySQL
https://ru.wikipedia.org/wiki/MySQL

[18] phpMyAdmin – веб-интерфейся для работы с СУБД MySQL
https://ru.wikipedia.org/wiki/PhpMyAdmin

[19] Справка по тегу «input» на сайте htmlbook.ru
http://htmlbook.ru/html/input

[20] Вывод простейшей HTML-страницы
http://book1.dvinemnauku.ru/page_1.html

[21] HTML-страница с некоторыми тегами
http://book1.dvinemnauku.ru/page_2.html

[22] HTML-страница с верно оформленной структурой
http://book1.dvinemnauku.ru/page_3.html

[23] Пример работы CSS-атрибутов «margin», «padding» и «border»
http://book1.dvinemnauku.ru/page_4.html

[24] Главное меню нашего аналога Википедии
http://book1.dvinemnauku.ru/page_5.html

[25] Верстка нашего аналога Википедии
http://book1.dvinemnauku.ru/page_6.html

[26] Пример реализации аналога Википедии
http://book1.dvinemnauku.ru/all_pages.php

[27] Google maps https://maps.google.com.

[28] Яндекс карты https://maps.yandex.ru.

[29] Open Layers http://openlayers.org.

[30] Leaflet JS http://leafletjs.com/.

[31] Листинг: Отрисовка тайлов. Функция Draw
http://maps.dvinemnauku.ru/Map/Listing/listing.php#Draw

[32] Листинг: Отрисовка обьектов на карте. Функция DrawPoints http://maps.dvinemnauku.ru/Map/Listing/listing.php#DrawPoints

[33] Листинг: Центрирование точки. Функция CountPosition http://maps.dvinemnauku.ru/Map/Listing/listing.php#CountPosition

[34] Листинг: Изменение размера карты. Функция Zoom http://maps.dvinemnauku.ru/Map/Listing/listing.php#Zoom

[35] Листинг: Заполнение таблицы. Функция FillTable http://maps.dvinemnauku.ru/Map/Listing/listing.php#FillTable

[36] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze *An introduction to Information Retrieval* 2008: Cambridge University Press.

[37] Jon M. Kleinberg *Authoritative sources in a hyperlinked environment* 1999: Journal of the ACM.

[38] Sergey Brin, Lawrence Page *The Anatomy of a Large-Scale Hypertextual Web Search Engine* 1998: Computer Networks and ISDN Systems

[39] PHP Group *http://php.net*

[40] Поиск слова «plane» без применения ранжирования *http://search.dvinemnauku.ru/searchrank.php?q=plane&alg=NoRank*

[41] Поиск слова «plane» с применением ранжирования *http://search.dvinemnauku.ru/searchrank.php?q=plane&alg=TFIDF*

[42] Листинг файла db.php *http://search.dvinemnauku.ru/listing.html#connect-db*

[43] Листинг файла stopword.php *http://search.dvinemnauku.ru/listing.html#stopword*

[44] Листинг файла functions.php *http://search.dvinemnauku.ru/listing.html#functions*

[45] Листинг файла crawler.php *http://search.dvinemnauku.ru/listing.html#crawler*

[46] Листинг файла indexer.php *http://search.dvinemnauku.ru/listing.html#indexer*

[47] Листинг файла search.php *http://search.dvinemnauku.ru/listing.html#search*

[48] Листинг файла hits.php *http://search.dvinemnauku.ru/listing.html#algorithm-hits*

[49] Листинг файла pagerank.php *http://search.dvinemnauku.ru/listing.html#algorithm-pagerank*

[50] Листинг файла tf-idf.php *http://search.dvinemnauku.ru/listing.html#algorithm-tf-idf*

[51] Листинг файла searchrank.php *http://search.dvinemnauku.ru/listing.html#searchrank*

[52] Программа для работы с музыкальными файлами Audacity 2.1.0 *http://sourceforge.net/projects/audacity/*

[53] Архив музыкальных файлов — примеры и тесты *http://book5.dvinemnauku.ru/resources/5/Music.zip*

[54] Архив изображений — примеры и тесты *http://book5.dvinemnauku.ru/resources/5/Images.zip*

[55] Листинг — первичная обработка wav-файлов *http://book5.dvinemnauku.ru/listing5.php#wawParse*

[56] Листинг — составление вектора признаков по музыкальному файлу *http://book5.dvinemnauku.ru/listing5.php#mfcc*

[57] Классификация музыки и изображений — интерфейс приложения *http://book5.dvinemnauku.ru/interface.php*

[58] Листинг — первичная обработка изображения *http://book5.dvinemnauku.ru/listing5.php#imageParse*

[59] Листинг — составление вектора признаков по изображению *http://book5.dvinemnauku.ru/listing5.php#hog*

Подписано в печать: 10.08.2015
Тираж: 200 экз.