**Санкт-Петербургский государственный электротехнический университет**

**“ЛЭТИ” им. В. И. Ульянова (Ленина)**

**(СПбГЭТУ “ЛЭТИ”)**

|  |
| --- |
|  |

**Направление подготовки:** 09.03.01 “Информатика и вычислительная техника”

**Профиль:** “Вычислительные машины, комплексы, системы и сети”

**Факультет компьютерных технологий и информатики**

**Кафедра вычислительной техники**

*К защите допустить:*

**Заведующий кафедрой**д. т. н., профессор\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М. С.Куприянов

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**БАКАЛАВРА**

**Тема: “Мобильное приложение для знакомства с умным городом”**

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_ Д. А. Буракаев

Руководитель

к. т. н., доцент \_\_\_\_\_\_\_\_\_\_\_\_\_ И. И. Холод

Консультант по экономическому

обоснованию к. э. н., доцент \_\_\_\_\_\_\_\_\_\_\_\_\_ И. М. Антонова

Консультант от кафедры

к. т. н., доцент, с. н. с. \_\_\_\_\_\_\_\_\_\_\_\_\_ И. С. Зуев

Санкт-Петербург

2020 г.

**Санкт-Петербургский государственный электротехнический университет**

**“ЛЭТИ” им. В. И. Ульянова (Ленина)**

**(СПбГЭТУ “ЛЭТИ”)**

|  |  |
| --- | --- |
| Направление 09.03.01 Информатика и вычислительная техника  Профиль Вычислительные машины, комплексы, системы и сети  Факультет компьютерных технологий  и информатики  Кафедра вычислительной техники | **УТВЕРЖДАЮ**  Заведующий кафедрой ВТ  д. т. н., профессор  (М. С. Куприянов)  “\_\_\_” \_\_\_\_\_\_\_\_\_\_ 201\_\_г. |

**ЗАДАНИЕ**

**на выпускную квалификационную работу**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | Буракаев Данияр Азаматович |  | Группа № | **6305** |

|  |  |
| --- | --- |
| **1. Тема** | **Мобильное приложение для знакомства с умным** |
| **городом** | |
| *(утверждена приказом № \_\_\_\_\_ от \_\_\_\_\_\_\_\_\_\_\_\_\_)* | |

Место выполнения ВКР: СПбГЭТУ «ЛЭТИ», кафедра ВТ

**2. Объект и предмет исследования**

Архитектура приложения на ОС Android. Системы удаленной поддержки приложения на ОС Android.

**3. Цель**

Разработка мобильного приложения на операционной системе Android, предназначенного для прохождения викторин с использованием картографических систем, а также проектирование удаленных веб-сервисов, поддерживающих его работу.

**4. Исходные данные**

Документация к используемым в работе инструментам, официальное руководство для разработчиков в ОС Android, тематическая литература, статьи в сети Интернет.

**5. Содержание**

1. Исследование принципов проектирования приложений в ОС Android.
2. Изучение подходов к построению систем для сетевого обмена данными с мобильным устройством.
3. Разработка мобильного приложения с использованием геоинформационных систем.
4. Интеграция коммерческих сервисов в существующее мобильное приложение.
5. Разработка системы хранения и обмена пользовательскими данными.

**6. Технические требования**

Разрабатываемый программный продукт должен выполнять следующие функции:

* обладать интуитивно понятным пользовательским интерфейсом;
* функционировать на всех актуальных версиях ОС Android (начиная с 17);
* возможность размещения удаленных сервисов на сервере семейства Apache Tomcat;
* производить безопасную аутентификацию пользователей;
* производить обмен данными между приложением на устройстве пользователя и удаленными сервисами системы;
* редактировать данные пользователя на сервере в соответствии с логикой работы мобильного приложения.

**7. Дополнительные разделы**

Экономическое обоснование проекта разрабатываемого приложения.

**8. Результаты**

* пояснительная записка к дипломному проекту;
* анализ комплекса инструментов и технологий, использованных при проектировании системы;
* описание ключевых модулей системы;
* листинги программ;
* примеры работы системы.

|  |  |  |
| --- | --- | --- |
| Дата выдачи задания |  | Дата представления ВКР к защите |
| «\_18\_» февраля 2019 г. |  | «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_ 201\_\_ г. |

|  |  |  |
| --- | --- | --- |
| Руководитель  к. т. н., доцент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | И. И. Холод |
| Студент | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | Д. А. Буракаев |

**Календарный план выпускной квалификационной работы**

**Санкт-Петербургский государственный электротехнический университет**

**“ЛЭТИ” им. В. И. Ульянова (Ленина)**

**(СПбГЭТУ “ЛЭТИ”)**

|  |
| --- |
|  |

|  |  |
| --- | --- |
| Направление (специальность)  Профиль (программа, специализация)  Факультет компьютерных технологий  и информатики  Кафедра вычислительной техники | **УТВЕРЖДАЮ**  Заведующий кафедрой ВТ  д. т. н., профессор  (М. С. Куприянов)  “\_\_\_” \_\_\_\_\_\_\_\_\_\_ 201\_\_г. |

**КАЛЕНДАРНЫЙ ПЛАН**

**выполнения выпускной квалификационной работы**

|  |  |
| --- | --- |
| Тема | **Мобильное приложение для знакомства с умным городом** |
|  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | Буракаев Данияр Азаматович |  | Группа № | **6305** |

|  |  |  |
| --- | --- | --- |
| № этапа | Наименование работ | Срок  выполнения |
| 1 | Обзор литературы по теме работы | 12.02.2020  19.02.2020 |
| 2 | Разработка серверной части приложения | 3.03.2020  26.04.2020 |
| 3 | Разработка пользовательского мобильного приложения | 27.04.2020  15.05.2020 |
| 4 | Внедрение сторонних корпоративных систем в приложение | 16.05.2020  26.05.2020 |
| 5 | Оформление пояснительной записки | 26.05.2020  2.06.2020 |
| 6 | Предварительное рассмотрение работы | 2.06.2020  7.06.2020 |
| 7 | Представление работы к защите | 7.06.2020 |

|  |  |  |
| --- | --- | --- |
| Руководитель  к. т. н., доцент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | И. И. Холод |
| Студент | **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** | Д. А. Буракаев |

**РЕФЕРАТ**

Дипломный проект посвящен проектированию и программной реализации приложения для мобильной операционной системы *Android*, систем для сетевого взаимодействия с приложением. Тема работы: «Мобильное приложение для знакомства с умным городом».

В работе были рассмотрены основные понятия и проблемы, связанные с разработкой мобильных приложений, систем обмена данными. Приведено описание основных модулей спроектированных программных продуктов, принципа их работы и взаимодействия.

В ходе производства были задействованы следующие среды программирования и конфигурации: *Android Studio* (ЯП Java, Kotlin), *MySQL Workbench* (SQL), *IntelliJ IDEA* 2020 (Java), *PyCharm* 2019 (Python 3.8). В качестве сервера для расположения удаленных сервисов выступал контейнер сервлетов *Apache Tomcat* 9 версии. Для разработки автоматизированной системы опроса пользователей была использована программная оболочка *Aiogram*. Полученный продукт может быть использован в качестве основы для разработки и отладки комплексного мобильного приложения, более полно удовлетворяющего условиям поставленной задачи, расширяющего его функционал.

Пояснительная записка к выпускной квалификационной работе бакалавра изложена на 91 странице машинописного текста, включает в себя 3 основных раздела, 24 рисунка и 49 ссылок на использованные в работе источники.

**ABSTRACT**

The developed software system consists of:

1. Android mobile application which passing quiz associated with some selected place in the city (via Google Maps Service), processing user personal data and synchronize it with Tomcat server.
2. Apache Tomcat server used for data fetching from MySQL database using Java servlet components.
3. *Telegram Bot* Application for automated survey of mobile application users (feedback and adding new places suggestions).

This report provides an overview of the various technologies used for developing system in the stated scientific and technical field, description of their implementation in the app-server communication system. Resulting software system meets all the previously set tasks.

**СОДЕРЖАНИЕ**

[ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 10](#_Toc42584212)

[ВВЕДЕНИЕ 14](#_Toc42584213)

[1 ОБЗОР НЕОБХОДИМЫХ КОМПОНЕНТОВ ДЛЯ ПОДДЕРЖАНИЯ РАБОТЫ ПРИЛОЖЕНИЯ 16](#_Toc42584214)

[1.1 Активности приложения 16](#_Toc42584215)

[1.1.1 Пользовательский интерфейс приложения Android 18](#_Toc42584216)

[1.1.2 Работа с данными в Android-приложении 20](#_Toc42584217)

[1.1.3 Виртуальное устройство на ОС Android 21](#_Toc42584218)

[1.2 Базы данных MySQL 22](#_Toc42584219)

[1.2.1 Преимущества над другими SQL СУБД 23](#_Toc42584220)

[1.3 Обеспечение сетевого взаимодействия с приложением 24](#_Toc42584221)

[1.3.1 Контейнер Apache Tomcat 24](#_Toc42584222)

[1.3.2 Веб-приложения 26](#_Toc42584223)

[1.4 Использование сторонних сервисов в приложении Android 28](#_Toc42584224)

[1.4.1 Интеграция Google Firebase 28](#_Toc42584225)

[1.4.2 Интеграция активности с использованием Google Maps API 29](#_Toc42584226)

[1.4.3 Разработка автоматической системы для анкетирования с использованием Aiogram API 30](#_Toc42584227)

[2 РАЗРАБОТКА СТРУКТУРЫ И МОДУЛЕЙ ПРОГРАММНОЙ СИСТЕМЫ 32](#_Toc42584228)

[2.1 Описание структуры мобильного приложения Android 32](#_Toc42584229)

[2.2 Функциональные модули Android-приложения 35](#_Toc42584230)

[2.2.1 Реализация различных способов обработки данных пользователя 35](#_Toc42584231)

[2.2.2 Асинхронные алгоритмы получения данных 37](#_Toc42584232)

[2.3 Структура сервисов для удаленного взаимодействия с устройством пользователя 38](#_Toc42584233)

[2.3.1 Описание базы данных системы 39](#_Toc42584234)

[2.3.2 Сервлет для обработки пользовательских запросов 39](#_Toc42584235)

[2.4 Разработка и внедрение модулей с использованием сторонних сервисов 41](#_Toc42584236)

[2.4.1 Интерфейс картографии Google Maps 41](#_Toc42584237)

[2.4.2 Интеграция библиотек сервиса аутентификации Firebase 42](#_Toc42584238)

[2.4.3 Анкетирование с использованием приложения Telegram и программной оболочки Aiogram API 42](#_Toc42584239)

[3 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОГО ПРОЕКТА 45](#_Toc42584240)

[3.1 Составление детализированного плана выполнения работ 45](#_Toc42584241)

[3.2 Оценка продолжительности выполнения работ для каждого из участников команды 46](#_Toc42584242)

[3.3 Оценка размеров заработной платы и социальных отчислений 50](#_Toc42584243)

[3.4 Расчет материальных затрат на комплектующие, накладные расходы, обслуживание системы 52](#_Toc42584244)

[3.5 Обзор способов реализации продукта 53](#_Toc42584245)

[3.6 Обзор и оценка конкурентной среды 54](#_Toc42584246)

[3.7 Выводы об экономической целесообразности разрабатываемого продукта 55](#_Toc42584247)

[ЗАКЛЮЧЕНИЕ 57](#_Toc42584248)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 59](#_Toc42584249)

[ПРИЛОЖЕНИЕ А. Разметка активности MapActivity 63](#_Toc42584250)

[ПРИЛОЖЕНИЕ Б. Листинг метода для аутентификации пользователя 66](#_Toc42584251)

[ПРИЛОЖЕНИЕ В. Листинг метода для обработки результатов работы активностей 67](#_Toc42584252)

[ПРИЛОЖЕНИЕ Г. Листинг ключевых методов активности MapActivity 68](#_Toc42584253)

[ПРИЛОЖЕНИЕ Д. Листинг метода-слушателя объекта типа Button 72](#_Toc42584254)

[ПРИЛОЖЕНИЕ Е. Листинг ключевых методов викторины 73](#_Toc42584255)

[ПРИЛОЖЕНИЕ Ж. Листинг модели QuizData 74](#_Toc42584256)

[ПРИЛОЖЕНИЕ И. Листинг метода адаптера, генерирующего вид пользовательского интерфейса 76](#_Toc42584257)

[ПРИЛОЖЕНИЕ К. Листинг метода ReceiveDataFromMySQLTask 77](#_Toc42584258)

[ПРИЛОЖЕНИЕ Л. Листинг методов сервлета для работы с базой данных 79](#_Toc42584259)

[ПРИЛОЖЕНИЕ М. Листинг метода-перехватчика пользовательского запроса 81](#_Toc42584260)

[ПРИЛОЖЕНИЕ Н. Пример содержимого текстового файла для локализации 82](#_Toc42584261)

[ПРИЛОЖЕНИЕ П. Регистрация в системе 84](#_Toc42584262)

[ПРИЛОЖЕНИЕ Р. Прохождение викторины 88](#_Toc42584263)

**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

*Android* – мобильная операционная система, основанная на модифицированной версии ядра Linux и другого программного обеспечения с открытым исходным кодом.

*Android Studio* – официальная интегрированная среда разработки для мобильной операционной системы Android, созданная на основе программного обеспечения JetBrains IntelliJ IDEA.

*MySQL* – система управления реляционными базами данных с открытым исходным кодом, написанная на ЯП C и C ++.

*MySQL Workbench* – средство визуального проектирования баз данных, которое объединяет разработку, администрирование, проектирование, создание и обслуживание баз данных SQL в единую интегрированную среду разработки для системы баз данных MySQL.

*IntelliJ IDEA* – интегрированная среда разработки (IDE), написанная на ЯП Java для разработки компьютерного программного обеспечения.

*PyCharm* – интегрированная среда разработки (IDE), используемая в компьютерном программировании, в особенно на ЯП Python.

*Telegram* – облачная служба обмена мгновенными сообщениями и передачи голоса по протоколу IP. Клиентские приложения Telegram доступны для Android, iOS, Windows Phone, Windows, macOS и Linux.

*Unix* – семейство многозадачных компьютерных операционных систем, основанных на оригинальной AT&T Unix.

*Linux* – семейство Unix-подобных операционных систем с открытым исходным кодом, основанных на ядре Linux.

*Wi-Fi* – семейство технологий беспроводных сетей, основанных на семействе стандартов IEEE 802.11, которые обычно используются для локальных сетевых устройств и доступа в Интернет.

*Формы* (англ. *Windows Forms, WF*) – бесплатная библиотека графических (GUI) классов с открытым исходным кодом, входящая в состав Microsoft .NET Framework или Mono Framework, предоставляющая платформу для написания многофункциональных клиентских приложений для настольных компьютеров, ноутбуков и планшетных ПК.

*Бизнес-логика* – часть программы, определяющая, как данные могут создаваться, хранится и использоваться.

*Пользовательский интерфейс* (англ. *User Interface*, *UI*) – часть программы, описывающая модели взаимодействия между клиентом и машиной. Цель взаимодействия - обеспечить эффективную работу и управление машиной со стороны оператора.

*Наследование* (англ. *Inheritance*) – в объектно-ориентированном программировании –механизм построения объекта или класса на основе другого объекта, сохраняя аналогичную реализацию.

*Базовый класс, суперкласс* – наследуемый класс.

*Стек* (англ. *Stack*) – абстрактный тип данных, содержащий набор элементов с двумя основными операциями: push (добавление в коллекцию) и pop (удаление последнего добавленного элемента).

*Метод обратного вызова* (англ. *Callback*) – исполняемый код, который передается в качестве аргумента другому коду. Код может выполняться немедленно, как при синхронном обратном вызове, или может произойти параллельно, как при асинхронном обратном вызове.

*Виджет* (англ. *App Widget*) – миниатюрные представления пользовательского интерфейса, которые можно встраивать в приложения (например, на главный экран) и периодически их обновлять.

*Canvas* – элемент языка разметки HTML5, позволяющий выполнять динамический рендеринг двумерных фигур и растровых изображений с помощью низкоуровневой технологии сценариев.

*OpenGL* – кросс-платформенный интерфейс прикладного программирования (API) для рендеринга 2D и 3D векторной графики. API обычно используется для взаимодействия с графическим процессором (GPU), чтобы добиться аппаратно-ускоренного рендеринга.

*Поток данных* (англ. *Data Stream*) – набор извлеченной информации от поставщика данных, набор передаваемых данных.

*Диск* (англ. *Disk Storage*) – это общая категория механизмов хранения, где данные записываются с помощью различных электронных, магнитных, оптических или механических воздействий на поверхностный слой одного или нескольких вращающихся дисков. Часто используется для обозначения хранилища вообще.

*Флеш-память* (англ. *Flash Memory*) – твердотельный энергонезависимый носитель данных.

*Суперпользователь* (англ. *Superuser*) – специальная учетная запись пользователя, использующаяся для системного администрирования системы или хранилища данных.

*XML* (*Extensible Markup Language*) – язык разметки, который определяет набор правил для кодирования документов в формате, который удобно считывается как человеком, так и вычислительной машиной.

*SQLite* – система управления реляционными базами данных, содержащаяся в библиотеке ЯП C. В отличие от многих других систем управления базами данных, SQLite не является ядром клиент-серверной базы данных и предназначена для локального хранения данных.

*Облачные хранилища* (англ. *Cloud Storage*) – модель хранения компьютерных данных, в которой цифровые данные хранятся в логических пулах в то время как физически они располагаются на удаленных хостинговых компаниях.

*Chrome OS* – операционная система на основе ядра Linux, разработанная Google на основе свободно распространяющегося программного обеспечения Chromium OS и в качестве основного пользовательского интерфейса использующего веб-браузер Google Chrome.

*Android SDK* (*Android Software Development Kit*) – программный комплекс, включающий в себя полный набор инструментов разработки приложений для ОС Android.

*Эмулятор* (англ. *Emulator*) – аппаратное или программное обеспечение, которое позволяет одной компьютерной системе (называемой хостом) повторять деятельность другой системы (называемая гостевой).

*SQL* (*Structured Query Language*) – предметно-ориентированный язык, используемый в программировании и предназначенный для управления данными, хранящимися в системе управления реляционными базами данных.

*Транзакция БД* (англ. *Database transaction*) – единица работы, исполняемая в системе управления базой данных (или аналогичной системе) над данными.

*Репликация* – возобновление, повторение. Расширение серверного комплекса повторением существующих систем.

*Кластеризация* – объединение определенного количества серверов в группу, образующих единый ресурс.

*Параллелизм* – свойство некоторых систем, при котором несколько вычислений выполняются одновременно.

*Solaris* – проприетарная операционная система Unix, изначально разработанная Sun Microsystems.

*OS/2* – серия компьютерных операционных систем, изначально созданная Microsoft и IBM под руководством дизайнера программного обеспечения IBM Эда Якобуччи.

*Программная оболочка* (англ. *Application Programming Interface, API*) – программный интерфейс, который определяет взаимодействие между несколькими программными посредниками.

*Java Сервлет* (англ. *Java Servlet*) – программный компонент Java, который расширяет возможности сервера, размещая на нем веб-приложения.

*Виртуальная машина Java* (англ. *Java virtual machine, JVM*) – виртуальная машина, которая позволяет компьютеру запускать программы Java, а также программы, написанные на других языках, но которые также скомпилированы в байт-код Java.

*Унифицированный указатель ресурса* (англ. *Uniform Resource Locator, URL*) – ссылка на веб-ресурс, указывающая на его местоположение в компьютерной сети и механизм его получения. Является частным случаем идентификатора ресурса URI.

*Контекст сервлета* (англ. *Servlet Context*) – набор методов, которые сервлет использует для связи со своим контейнером сервлета, например, для получения MIME-типа файла, отправки запросов или записи в файл журнала.

*HTML (Hypertext Markup Language)* –стандартный язык разметки для документов, предназначенный прежде всего для их отображения в веб-браузере.

*HTTP (Hypertext Transfer Protocol)* – протокол для передачи произвольных данных в сети Интернет.

*Геолокация* -определение реального географического местоположения некоторого электронного устройства.

*Firebase Auth* – служба, предназначенная для безопасной аутентификации пользователей, используя только код на стороне клиента.

*Google Maps* – веб-картографический сервис, предоставляющий спутниковые снимки Земли, аэрофотоснимки, карты улиц, панорамные панорамные виды улиц, программную оболочку для его внедрения в приложение.

*Фреймворк* (англ. *Framework*) – совокупность программных комплексов, обеспечивающих общие функциональные возможности в некоторой технической области.

*Конечный автомат* (англ. *Finite-state machine*) – математическая модель, абстрактная машина, которая может находиться ровно в одном из конечного числа состояний в любой момент времени.

*Google Mobile Service* (GMS) – совокупность приложений и сервисов, разработанных компанией Google, которые часто предустановлены на устройствах Android.

*Контроллер нажатий* (англ. *Click Listener*) – объект, предназначенный для «перехвата» (исполнения некоторого алгоритма) при нажатии на элемент пользовательского интерфейса. Частный случай Event Listener (контроллера событий).

*Конструктор* (англ. *Object Constructor*) – специальный тип подпрограммы, вызываемой при создания объекта, экземпляра класса, контролирующий свойства создаваемого объекта.

*JSON* (*JavaScript Object Notation*) – открытый стандартный формат обмена данными, который удобно считывается как человеком, так и вычислительной машиной.

*Универсальный идентификатор ресурса* (англ. *Uniform Resource Identifier*, *URI*) – это строка символов, которая однозначно идентифицирует некоторый ресурс.

*Асинхронный обмен данными* (англ. *Asynchronous communication*) – передача данных, как правило, без использования внешнего контролирующего тактового сигнала, для непрерывной передачи в устойчивом потоке данных.

*Токен* (англ. *Token*) – уникальный ключ, используемый для получения доступа к некоторому ресурсу.

*GoogleAds* – рекламная онлайн-платформа, разработанная компанией Google, в которой рекламодатели платят держателю платформы за показ кратких рекламных объявлений, предложений услуг, видеоконтента для пользователей Интернета.

**ВВЕДЕНИЕ**

Операционная система Android является одной из многих так называемых *Unix-подобных* операционных систем, то есть систем, базирующихся на модифицированном *ядре* ОС *Linux*. Ее история начинается [1] в середине 2004 года, когда команда молодых разработчиков взялась за создание удобной, более персонализированной мобильной системы, способной составить серьезную конкуренцию уже устоявшимся игрокам на рынке персональных портативных устройств: *Symbian* и *Windows Mobile*.

После выкупа [2] компании-разработчика *Android Inc.* компанией *Google* (ныне *Alphabet*) в июле 2005 года платформа продолжает свое развитие, наращивая свой функционал и внедряя инструменты, упрощающие взаимодействие с операционной системой не только для пользователей, но и для разработчиков программного обеспечения. В текущее время ОС Android является наиболее популярной [3] системой для мобильных телефонов и планшетных компьютеров: на момент 2020 года ее доля рынка составляет более чем 70% активных устройств.

Сетевые технологии тоже не стоят на месте: ежегодно растет площадь покрытия мобильными сетями самых отдаленных мест мира, увеличивается [4] и потребление мобильного трафика по сравнению с уже давно привычными *Wi-Fi* и проводными сетями. Все это приводит к расширению возможностей использования мобильных устройств в повседневной жизни. Однако вместе с тем продолжается [5] увеличение числа людей, полностью или частично заменяющих живое общение и взаимодействие активностью в социальных сетях. Текущая работа призвана если не ликвидировать негативные эффекты от «ухода в виртуальную реальность», то хотя бы произвести попытку синтеза этих разных подходов к коммуникации.

Цель настоящей работы – программная реализация приложения, провоцирующего пользователя на определенные взаимодействия (в частности, социальные) в городской среде.

Объект исследования – операционная система Android, ее программная оболочка для разработки приложений, *web-технологии* для удаленного сообщения с мобильным устройством. Предмет разработки – приложение и совокупность сервисов, которые его обслуживают.

Разрабатываемая система «Аркантаун» представляет собой прототип, совмещающий в себе функции социальной сети, мобильной игры типа "Викторина" и городского гида. Пользователь совместно с другими пользователями или в одиночку отвечает на вопросы, связанные с городскими достопримечательностями, памятниками архитектуры, рядом с которыми тот находится на момент работы приложения. Из этого проистекает определение задач, для которых система предназначена:

1. Знакомство с городом проживания или временного пребывания. Для работы в приложении пользователь должен прибыть на место, отмеченное на карте для того, чтобы ответить на тематические вопросы и получить некоторую награду. Система поощрений должна подстегивать пользователю к активному исследованию достопримечательностей, городской инфраструктуры, города как такового.

2. Социальный подтекст. Система поощряет совместное времяпрепровождение пользователей: сформированные в команду пользователи получают больше очков, чем пользователи-одиночки. Становлению слаженного коллектива помогает также и командная рейтинговая система, позволяющая соревноваться в знании города с остальными командами.

По итогу работы будет спроектирована система, демонстрирующая основные функции конечного продукта: работа с картографическими системами, геолокацией, тестирование пользователей, аутентификация. Пояснительная записка к выполненной работе будет содержать следующие разделы:

1. Аналитический анализ. В результате его проведения должны быть рассмотрены разные технологии и инструменты для построения мобильного приложения и обслуживающей его системы.

2. Технический анализ. Разбор ключевых программных модулей системы, структур данных, задействованных в ее работе.

3. Экономическое обоснование. Раздел содержит подробный обзор экономической стороны разработки системы: заработной платы специалистов, затраченного на производство времени, закупки необходимого оборудования, методы монетизации конечного продукта и т.д.

**1 ОБЗОР НЕОБХОДИМЫХ КОМПОНЕНТОВ ДЛЯ ПОДДЕРЖАНИЯ РАБОТЫ ПРИЛОЖЕНИЯ**

В существующем разделе рассмотрены основные понятия и проблемы разработки мобильного приложения, работа которого завязана на взаимодействии с удаленными сервисами. Подробно разобраны составные элементы мобильного приложения, а также программных продуктов, отвечающих за поддержание его работы, методы хранения и передачи данных между ними. Кроме того, в нем приведена краткая справочная информация о сторонних сервисах, интегрированных в готовое приложения с целью расширения и улучшения его функционала.

* 1. **Активности приложения**

Ключевым элементом любого приложения на Android является т.н. *активность* (англ. *Activity*). Деятельность некоторой активности в каждом случае можно рассматривать [6] как аналог окна или диалога в настольном приложении или страницы в классическом веб-приложении, так как и в том, и в другом случае речь идет о совокупности *форм*, связанных между собой *бизнес-логикой* приложения.

Активность производит построение *пользовательского интерфейса* в момент своего запуска, его изменение с течением времени, организацию взаимодействия с его элементами со стороны пользователя, а также переключение приложения на следующую активность. В некоторых случаях она играет роль дискретной точки входа [7] в приложение (то есть, в частности, один из способов связываться с ним со стороны других приложений).

По умолчанию интерфейс активности занимает большую часть экрана устройства, оставляя место для некоторых системных индикаторов, таких как часы, текущий уровень мобильного сигнала, заряда аккумулятора и прочее. С другой стороны следует иметь в виду, что на некоторых устройствах пользователь имеет возможность одновременно работать с более чем одной активностью (например, с использованием режима разделения экрана на телефоне или планшетном компьютере), самостоятельно изменяя ее размер на экране.

Активности являются *публичными классами*, *унаследованными* от *базового класса* android.app.Activity [8] (или, возможно, от какого-то другого класса, который сам наследует этот *суперкласс*). Каждая активность в системе помещается в специальный *стек* активностей [9]. То есть, когда новая активность инициируется, она обычно помещается на вершину текущего стека и становится текущей. Предыдущая активность (которая не обязательно является ее "родителем") всегда остается под ней в том же стеке и вновь выйдет на первый план в случае ее завершения или приостановки.

Существующая активность в течение своего жизненного цикла может принимать одно из четырех описанных состояний:

1. Активное состояние. Активность расположена в самой верхней позиции текущего стека, активно выполняется или ожидает действий пользователя.
2. Активность без фокуса. Активность все еще отображается пользователю, но другая имеет более высокий приоритет и в данный момент исполняется. Такое может происходить в многооконном режиме или в случае, если приложение не может быть корректно сфокусировано в текущем оконном режиме. Такая активность полностью сохраняется в системе и остается прикрепленной к оконному менеджеру.
3. Полностью скрытая активность. Активность по-прежнему сохраняет всю информацию о своем содержимом и состоянии, однако больше не видна пользователю. В случае необходимости (например, нехватки памяти для новой активности) будет немедленно уничтожена системой.
4. Уничтоженная активность. Система может ликвидировать активность либо при помощи сигнала о завершении, либо принудительно завершив процесс, поддерживающий работу этой активности. При необходимости повторного отображения активности пользователю, следует предусмотреть ее перезапуск и восстановление необходимого состояния.

На рисунке 1 демонстрируется жизненный цикл любой активности в ОС Android. Квадратные прямоугольники представляют *методы обратного вызова*, предназначенные для выполнения операций в каждом конкретном из описанных состояний. Цветные овалы – основные состояния, в которых может существовать активность.

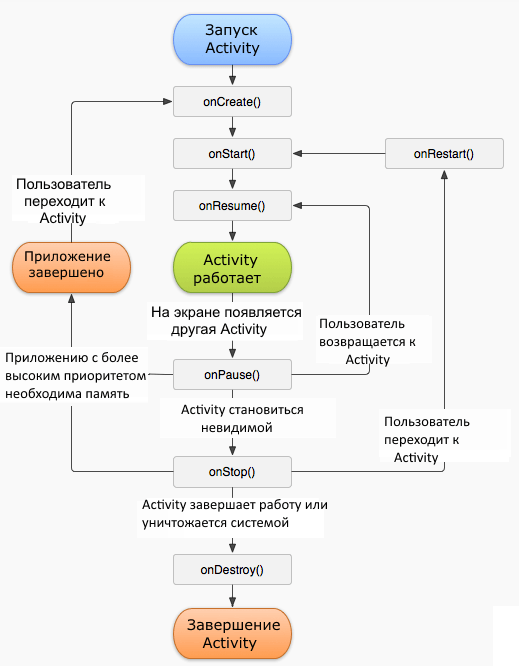


Рисунок 1 – Жизненный цикл активности Android-приложения

* + 1. **Пользовательский интерфейс приложения Android**

Большинство активностей ОС Android используют так называемый "каркас *виджетов*" для визуализации соответствующих им представлений пользовательского интерфейса, хотя существует также возможность задействовать специальные программные интерфейсы для двухмерной (*Canvas* API) и трехмерной графики (*OpenGL* API).

Пример процесса разметки элементов пользовательского интерфейса показан на рисунке 2.

В терминах Android виджет – это базовая единица пользовательского интерфейса [10]. Текстовые поля для ввода, кнопки разного вида, метки, списки и другие микроэлементы – все это относится к виджетам разметки. Таким образом, пользовательский интерфейс отдельно взятой активности может состоять из одного или нескольких виджетов, организованных особым образом.

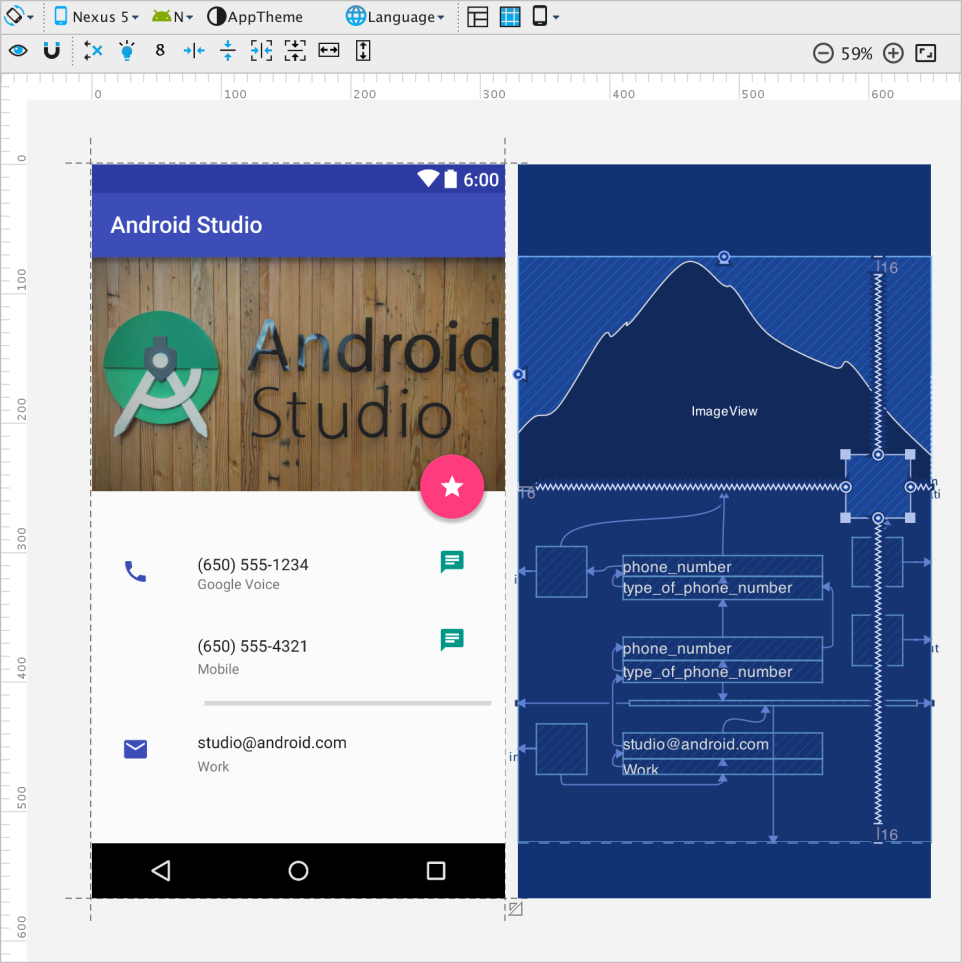


Рисунок 2 – Процесс разметки виджетов в Android Studio

Для организации их расположения, поведения при взаимодействии с пользователем и системой используются различные классы контейнеров, называемые *менеджерами компоновки*. Рассмотрим некоторые из них:

1. *GridLayoutManager* [11]. Элементы (виджеты или комплексные структуры), включенные в содержимое менеджера, располагаются в соответствии с настройками сетки *видов*.
2. *LinearLayoutManager* [12]. Элементы располагаются последовательно по вертикали или по горизонтали.
3. *RecyclerView.LayoutManager* [13]. Менеджер, определяющий не только расположение элементов, но и настройки политики добавления новых элементов, удаления уже существующих элементов.
4. *ConstraintLayout* [14]*.* Менеджер по умолчанию.

Пример файла разметки, содержащим виджеты, организованные с помощью последнего компоновщика, представлен в Приложении А.

При конструировании пользовательского интерфейса необходимо [15] также учитывать, что приложения в ОС Android могут работать на разных устройствах с разным разрешением экрана: телефонах, планшетах, телевизорах и т. д. Иногда полученный пользовательский интерфейс необходимо адаптировать для разных сред, чтобы обеспечить наилучший опыт взаимодействия с приложением для конечного пользователя.

* + 1. **Работа с данными в Android-приложении**

Android предлагает несколько [16] структурированных способов хранения данных, в частности:

1. Локальные или внешние файлы. Запись и чтение данных из файлов на устройстве, во внешнем хранилище.

В целом, Android использует стандартный [17] ввод-вывод файлов *Java* для локальных файлов. Для работы с файлами используются *File* и *InputStream* и *OutputWriter* и другие привычные классы для обработки данных. Отличительной чертой Android является то, что чтение и запись производится только в тех *каталогах*, к которым приложение имеет соответствующий уровень доступа. Этим обеспечивается *инкапсуляция* данных приложения, что дает возможность создания и взаимодействия с его приватными данными без возможности случайно или намеренно навредить работе остальным сервисам.

Под *внутренним хранилищем* понимается часть встроенного хранилища некоторого приложения. *Внешнее* же хранилище относится к пространству хранения, которое может быть смонтировано пользователем как *диск*. Впрочем, как правило, оно так же относится к встроенной *флэш-памяти* устройства, находящейся в отдельной от внутренней области хранилища. То есть ситуация поддержания работы с разными типами файлового хранилища усложняется тем, что пользователь будет воспринимать это пространство как «внутреннее хранилище» (например, в настройках файлового менеджера) в то время, как оно фактически доступно для чтения и записи любым другим приложениям.

Таким образом, файлы, записанные в собственном внутреннем хранилище приложения, не могут быть прочитаны или редактированы другими приложениями (не считая тех случаев [18], когда некоторому приложению предоставляются права *супер-пользователя*), но файлы во внешнем хранилище видны всем приложениям и пользователям, и могут быть отредактированы или удалены без ведома «управляющего» приложения.

1. *Общие настройки* (англ. *SharedPreferences*). Совместное или индивидуальное использование данных некоторого приложения.

Файлы – это быстрый и удобный способ хранения неструктурированных данных в Android. Но есть и другие более удобные способы хранения небольших фрагментов данных в виде пар «ключ-значение» [19] – технология SharedPreferences.

Программная оболочка инструмента SharedPreferences предоставляет средства для чтения и записи данных в файл, отформатированный как *XML-файл*. Такое хранилище особо полезно, когда существует необходимость хранить такие вещи, как общие настройки приложения, пользовательский прогресс, фильтры данных и так далее. Сформированный файл хранится в каталоге устройства /data/{имя\_приложения}. Как и обычные локальные файлы, данные, записанные в SharedPreferences, закреплены за конкретным приложением, могут быть не доступны для других приложений, и будут утеряны [20] после удаления приложения или сброса его настроек.

1. Локальные *базы данных* *SQLite*. Структурированный набор данных приложения, хранящийся на устройстве в виде отдельного файла.
2. *Облачные хранилища* данных. Получение и занесение данных с использованием удаленного веб-сервиса.
   * 1. **Виртуальное устройство на ОС Android**

На сегодняшний день используется более двух миллиардов [21] Android-устройств, представляющих тысячи различных моделей от десятков разных производителей. Сам Android эволюционировал с момента выпуска своей первой версии в 2008 году, но между разными типами устройств и разными версиями Android многие специалисты давно утвердили [22] термин «*фрагментация*», предполагая, что создание приложений, работающих во всех этих различных средах, невозможно.

Устройства Android бывают разных форм, размеров и цветов. Однако среди них существуют три доминирующих форм-фактора: телефон, планшетный компьютер, ноутбук (как правило, на базе *ОС Chrome*), и несколько менее распространённых: телевизор, носимые устройства (умные часы) и другие.

На самом деле большинство приложений может свободно работать на разных платформах при определенных условиях и должном уровне проработки. Тем не менее, важно понимать, что система Android не различает устройства по своему типу и предназначению: каждое конкретное устройство, работающее на этой операционной системе, идентифицируется по встроенным в нем возможностям. Таким образом, из устройства системными методами можно получить некоторую информацию, необходимую для работы приложения (или формирования отказа в запуске при отсутствии необходимого функционала): размер и разрешение экрана, телефония, наличие некоторых датчиков и сенсоров, и т.д.

При разработке приложения следует [23] исходить не из форм-фактора предполагаемого устройства пользователя, на котором оно будет в последствие установлено, а из того, какие возможности и функции являются достаточными для его полноценной работы, а без каких работа в нем считается невозможной. Такой подход рекомендуется использовать профессиональным разработчикам для облегчения адаптации приложения к устройствам разного типа. Однако даже в этом случае существует необходимость тестирования приложения если не на каждой актуальной разновидности устройств, то хотя бы на тех, для которых оно разрабатывается в первую очередь.

Чтобы заполнить промежутки между имеющимися у разработчика устройствами и возможными устройствами, набор инструментов разработчика *Android SDK* включает в себя *эмулятор* Android-устройств. С его помощью разработчик получает возможность отладки приложения, загружая его на виртуальное Android-устройство (англ. *Android Virtual Device*, *AVD*), располагающееся прямо на его вычислительной машине. Это устройство может иметь различные настройки и базироваться на любой из доступных версий Android. Пример выбора устройства для последующей эмуляции представлен на рисунке 3.

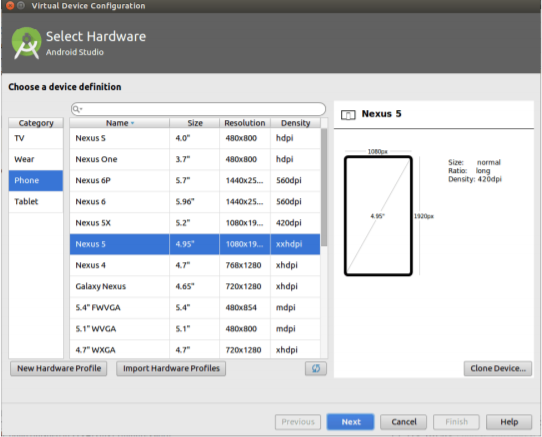


Рисунок 3 – Процесс настройки AVD

* 1. **Базы данных MySQL**

*MySQL* – это широко используемая [24] структурированная реляционная система управления базами данных с открытым исходным кодом. На текущем рынке информационных технологий тысячи веб-приложений используют MySQL, в том числе такие гиганты, как *Facebook*, *Twitter* и *Wikipedia*.

Данные в реляционной базе данных хранятся в организованном формате (как совокупность таблиц данных), так что информация может быть легко извлечена при помощи стандартизированных запросов языка *SQL*. Данные в ней хранятся в разных таблицах, состоящих из строк и столбцов. Кроме того, существует возможность построения *отношений* между различными таблицами, что позволяет эффективней извлекать необходимые данные.

* + 1. **Преимущества над другими SQL СУБД**

Система MySQL содержит в себе набор необходимых инструментов, которые делают ее более подходящей СУБД для профессиональной разработки в сравнении с остальными. Перечислим некоторые преимущества использования MySQL:

1. Безопасность. MySQL дает возможность контроля доступа к данным, что включает в себя: предоставление и отзыв требуемого доступа у некоторого пользователя, управления *ролями доступа*, что позволяет выстраивать сложную структуру прав доступа для множества разных по типу пользователей.

Все пароли пользователей хранятся в зашифрованном виде, что в действительности обеспечивает хороший уровень безопасности, позволяющий защитить конфиденциальную информацию от злоумышленников.

1. Масштабируемость. В виду постоянной необходимости [25] переработки существующих систем хранения информации, масштабируемые базы данных были бы предпочтительным решением, чтобы в любой момент удовлетворить самые неожиданные требования к масштабированию. СУБД MySQL – это проработанная система контроля баз данных, обеспечивающая поддержку при их масштабировании как горизонтальном (увеличение числа обрабатывающих сервисов), так и вертикально (расширение возможностей существующих сервисов техническими средствами) [26].
2. Открытый исходный код. СУБД MySQL – это система управления базами данных с открытым исходным кодом, которая позволяет быстро и легко отлаживать, обновлять и улучшать ее функциональность. Существует возможность также распространять расширенную версию MySQL, но для этого требуется специальная лицензия.
3. Высокая производительность. СУБД MySQL обеспечивает высокую скорость обработки *транзакций*. Она может кэшировать результаты обмена данными, что повышает производительность их чтения. *Репликация* и *кластеризация* обеспечивают лучший *параллелизм*, равномерное распределение рабочей нагрузки. *Индексы* базы данных также повышают производительность операторов запросов *SELECT* для больших объемов данных. Чтобы повысить производительность, MySQL версии 8 включила индексы в схему производительности для ускорения поиска данных.
4. Высокая надежность. Любой сбой или простой напрямую влияют на бизнес и доход от работы системы; следовательно, высокая надежность является фактором, который нельзя игнорировать. СУБД MySQL достаточно надежна [27] и предоставляет инструменты для постоянного мониторинга своей активности с использованием кластерных и репликационных конфигураций.

*Кластерные серверы* мгновенно обрабатывают сбои и управляют компонентом аварийного переключения, обеспечивая практически бесперебойную работу сервиса. Если один сервер выходит из строя, он перенаправляет запрос пользователя на другой узел и выполняет запрошенную операцию.

1. Кроссплатформенность. СУБД MySQL может работать на различных платформах, таких как *Windows*, *Linux*, *Solaris*, *OS 2* и другие. Кроме того, она имеет постоянную поддержку *программной оболочки* (англ. *application programming interface, API*) для всех известных языков программирования: PHP, C ++, Perl, Python, Java и прочие, что делает ее очень простой для интеграции в существующие решения разработки.
   1. **Обеспечение сетевого взаимодействия с приложением**

Хотя существуют мобильные приложения, разработанные исключительно для их использования без подключения к веб-сервисам, удаленного обмена пользовательской информацией, данными (например, встроенный калькулятор), зачастую необходимо кроме разработки собственно приложения располагать и настраивать сервисы, с которыми оно будет обмениваться данными, необходимыми для поддержания его работы.

* + 1. **Контейнер Apache Tomcat**

*Apache Tomcat* – это контейнер для веб-приложений и *сервлетов* на основе Java с открытым исходным кодом, который используется для размещения на сервере Java-приложений. Впервые система была разработана для компании Jakarta Tomcat [28], но из-за увеличения спроса она впоследствии была размещена в виде отдельного проекта под названием Apache Tomcat, которая до сих пор поддерживается организацией *The Apache Software Foundation*.

Apache Tomcat очень стабилен [29] и обладает всеми необходимыми функциями коммерческого контейнера веб-приложений, но при этом подпадает под лицензию *Open Source Apache* (т.е. подлежит свободному распространению). Tomcat также предоставляет дополнительные функциональные возможности, которые делают его отличным выбором для разработки полноценного решения для веб-приложений.

Помимо всего прочего, Tomcat по умолчанию включает в себя приложение *Tomcat Manager*. Оно устанавливается в контекстном пути /manager и предоставляет основные функции для управления веб-приложениями, запущенными на сервере Tomcat, из любого веб-браузера. Некоторые из предоставляемых функций включают возможность установки, запуска, остановки серверов, удаления и создания отчетов о работе конкретных веб-приложений.

Пример менеджера веб-приложений представлен на рисунке 4.

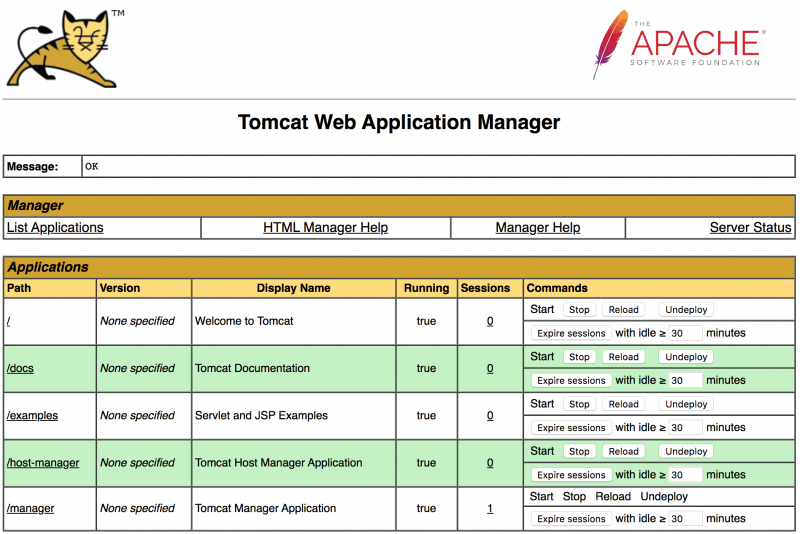


Рисунок 4 – Веб-интерфейс менеджера Tomcat

Экземпляр Tomcat, или сервер, является компонентом верхнего уровня в иерархии контейнеров Tomcat. То есть только один экземпляр Tomcat может располагаться на конкретной *виртуальной машине Java* (JVM). Этот подход обеспечивает безопасность для всех остальных приложений Java, работающих на той же физической машине, что и сервер Tomcat, в случае сбоя Tomcat и/или его JVM. При этом по-прежнему сохраняется возможность запуска нескольких экземпляров сервера в одном физическом блоке, но только в виде отдельных процессов Java, работающих на отдельных сетевых портах.

* + 1. **Веб-приложения**

Для любого веб-приложения база данных играет очень важную роль, поскольку является основным хранилищем информации, поддерживающим работу корпоративного приложения. Чтобы приложение работало корректно, необходима правильная конфигурация источника данных на уровне приложения.

Каждый раз, когда производится переход по *унифицированному указателю ресурса* (англ. *Uniform Resource Locator, URL*) некоторого веб-приложения, запрос отправляется на поддерживающий его веб-сервер. Как только веб-сервер примет запрос от браузера клиента, он производит его анализ, определяя предназначение этого обращения. Если требуется ответ в виде совокупности данных, он может перенаправить запрос на сервер базы данных (или обратиться к базе данных непосредственно, в зависимости от выбранной сетевой структуры). На основе этого запроса сервер базы данных извлечет содержимое и отправит ответ обратно на веб-сервер. Затем веб-сервер направляет ответ от сервера базы данных в клиентский браузер. Этот процесс может быть продемонстрирован схемой на рисунке 5.

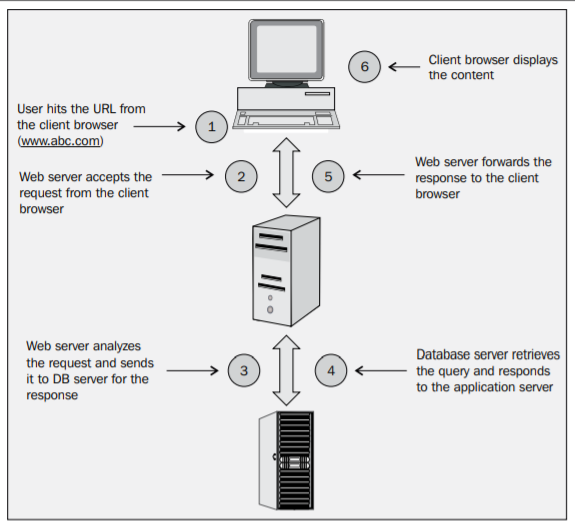


Рисунок 5 – Схема обмена информацией между сервером и браузером

Основная функция сервера Tomcat – выполнение роли контейнера для веб-приложений Java. Концепция веб-приложения была введена с выпуском спецификации Java-сервлета версии 2.2 [30]. Согласно этой спецификации, веб-приложение – это "набор сервлетов, HTML-страниц, классов и других ресурсов, которые можно объединять и запускать в нескольких контейнерах от нескольких поставщиков". На самом деле это означает, что веб-приложение представляет собой контейнер, который может содержать любую комбинацию из следующего списка объектов:

* сервлеты;
* *страницы Java-сервера;*
* служебные классы;
* статичные документы (файлы форматов *.html*, *.css*, изображения, внешние библиотеки);
* классы клиентов;
* конфигурационные данные веб-приложения.

Одной из основных характеристик веб-приложения является его связь с *контекстом сервлета* (англ. *ServletContext*). Каждое веб-приложение имеет один и только один контекст. Эти отношения контролируются контейнером сервлета Tomcat, который гарантирует, что никакие веб-приложения не будут конфликтовать друг с другом при доступе к объектам, расположенным в некотором контексте.

Страницы Java-сервера (англ. *Java Server Pages, JSP*) – это относительно простая в использовании [31], но мощная технология, которая чаще всего используется для генерации динамического *HTML-документа* на стороне сервера для выдачи его клиенту. Они являются прямым расширением Java-сервлетов, позволяющим разработчику встраивать логику языка программирования Java непосредственно внутрь генерируемой веб-страницы. При этом можно говорить, что страница *.jsp* отображается динамически, то есть при каждой перезагрузке страницы отображаемый результат может изменяться (если он привязан к времени, например).

Сервлет – это Java-программа, которая запускается в контейнере сервлетов и отвечает за прием запросов от клиентов и отправку ему ответов по *протоколу HTTP*. В стандартной программной оболочке *Java Servlet API* сервлеты определяются с использованием Java-интерфейса javax.servlet.Servlet, который должен реализовывать любой конкретный класс сервлета. Кроме этого существует возможность использования более простого в использовании суперкласса javax.servlet.http.HttpServlet, расширяемого конкретными реализациями сервлетов.

Для корректной работы сервлета необходимо расширить полученный класс методами doGet() и doPost() из класса HttpServlet. Переопределенные методы отвечают за обработку операций HTTP GET и HTTP POST соответственно. После производится компиляция полученного класса для его развертывания в Tomcat.

* 1. **Использование сторонних сервисов в приложении Android**

При разработке приложения в ОС Android часто возникает необходимость внедрение готовых программных инструментов для решения той или иной задачи. В рамках существующей работы были задействованы сервисы для безопасной аутентификации, а также комплексные фрагменты для *геолокации* и обеспечения работы с метками. Кроме того, была разработана форма обратной связи с пользователем приложения с использованием программной оболочки Telegram Bot API.

* + 1. **Интеграция Google Firebase**

*Firebase* – это готовый набор инструментов, которые позволяют реализовывать приложения для Android, работа которых завязана на хранении конфиденциальной информации безопасным и надежным способом. Firebase широко известен [32] своей простой в использовании программной оболочкой Firebase API, обширной документацией и поддержкой со стороны разработчиков.

Консоль Firebase – это место, где происходит настройка существующих приложений и их управление. Через нее осуществляется просмотр и редактирование зарегистрированных проектов, включенных сервисов, элементов данных. Пример консоли аутентификации пользователей в приложении представлен на рисунке 6.

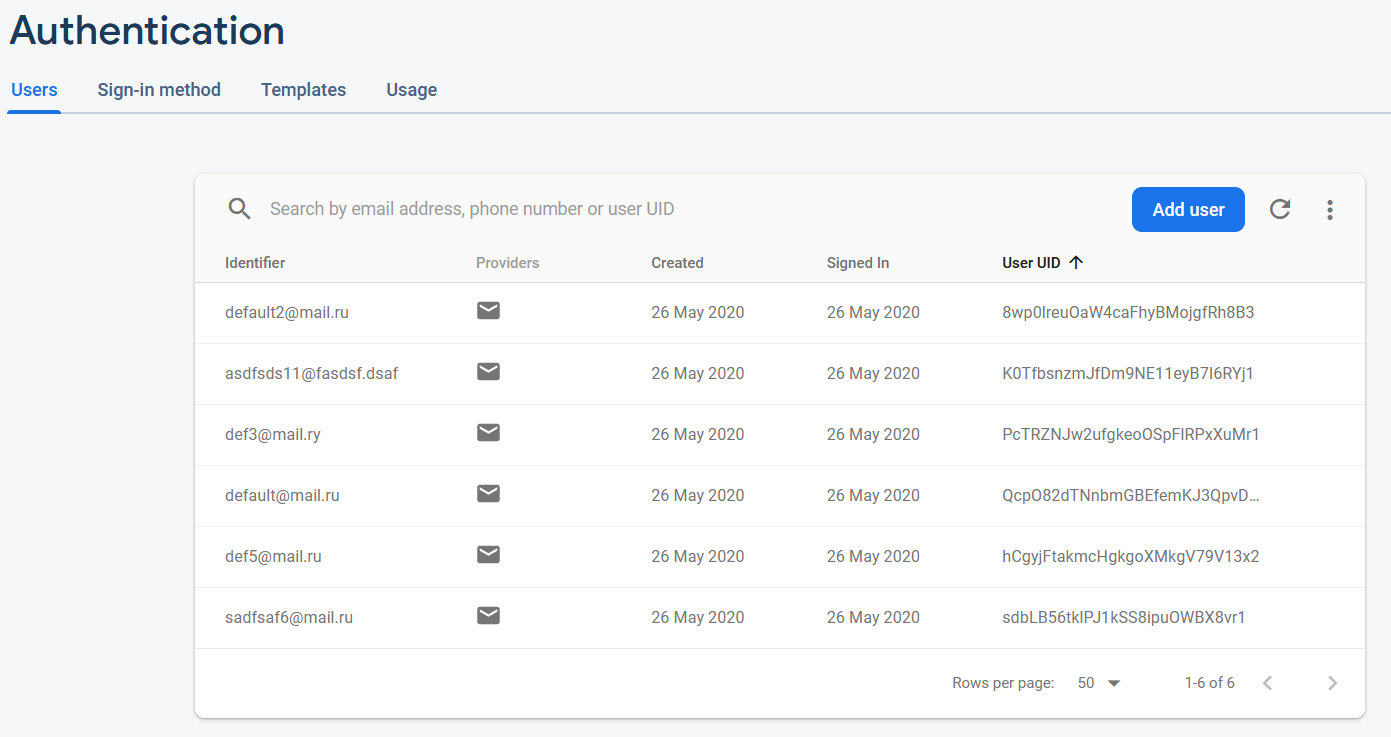


Рисунок 6 – Консоль учетных записей приложения

Большинству мобильных приложений, завязанных на использовании веб-сервисов, требуется идентификация пользователя, для настройки персональных действий и обеспечения безопасности пользовательских данных. Firebase поддерживает множество различных способов аутентификации пользователей. Инструмент *Firebase Auth* имеет встроенные функции для обработки попыток аутентификации со стороны сторонних сервисов, таких как Facebook, Twitter, *Github* или Google. Кроме того поддерживается возможность аутентификации пользователей по адресу электронной почты и паролю.

Для пользователя предоставляется возможность входа в систему двумя различными способами: используя собственный интерфейс или используя интерфейс с открытым исходным кодом Firebase, который также можно настраивать. Когда пользователь аутентифицируется, информация об этом пользователе возвращается на устройство посредством обратного вызовадля последующего ее использования при настройке взаимодействия с каждым конкретным пользователем.

Firebase также производит управление пользовательскими сеансами: пользователи могут оставаться в системе и после перезапуска приложения без необходимости повторно вводить свои персональные данные для аутентификации.

* + 1. **Интеграция активности с использованием Google Maps API**

Одним из наиболее интуитивных способов предоставления информации о физическом местоположении пользователя или какого-либо объекта, является интерактивная картографическая система *Google Maps*. При помощи инструмента GoogleMap из библиотеки *фрагментов* MapFragment можно создавать активности, включающие в себя интерактивную карту Google.

Карты Google поддерживают аннотации, привязанные к маркерам, фигуры и наложения изображений, которые можно закрепить за конкретными географическими точками [33]. Карты Google предлагают полный программный контроль над отображением карты, позволяя управлять углом обзора камеры, масштабированием, целевым местоположением и режимами отображения.

Доступ к программной оболочке Google API обеспечивается библиотекой карт служб Google Play, которую необходимо предварительно добавить в качестве зависимости к проекту разрабатываемого приложения.

Встроенный слой «Мое местоположение» предназначен для отображения текущего местоположения устройства на карте Google в виде синего маркера. При нажатии на кнопку «Мое местоположение» производится центрирование карты на последнем известном месте устройства. Для корректной работы фрагмента, приложению необходимо предоставить следующие разрешения, прописав их в манифесте приложения и, в последствии, проверив их наличие в самом приложении:

Manifest.permission.ACCESS\_FINE\_LOCATION

Позволяет программной оболочке определять местоположение с максимально возможной точностью от доступных поставщиков местоположения, включая *GPS*, а также данные *Wi-Fi* и *мобильной сотовой сети*.

Manifest.permission.ACCESS\_COARSE\_LOCATION

Позволяет программной оболочке использовать данные Wi-Fi и/или мобильной сотовой сети для определения местоположения устройства. API возвращает местоположение с точностью, приблизительно равной городской черте.

* + 1. **Разработка автоматической системы для анкетирования с  
       использованием Aiogram API**

В качестве платформы для разработки системы был выбран известный [34] *мессенджер* Telegram. Таким образом, появится возможность опроса максимального числа пользователей, независимо от платформы, которой каждый из них может отдавать предпочтение. Для обработки заявок от пользователей будет разработан сервис-бот, опрашивающий пользователя для внесения данных в хранилище для дальнейшей обработки модератором.

*Бот* – специальный аккаунт [35] в Telegram, созданный для того, чтобы автоматически обрабатывать и отправлять сообщения. Пользователи могут взаимодействовать с ботами при помощи сообщений и команд, отправляемых через обычные или групповые чаты. Логика бота контролируется при помощи HTTPS-запросов и дополнительного программного слоя для взаимодействия с ним. На рисунке 7 демонстрируется общая схема работы бота Telegram.

В качестве программного интерфейса для написания программы был выбран Telegram Bot API на языке программирования *Python версии 3.8*. Его главной особенностью является простота и удобство настройки подключений и обработки запросов от пользователей бота.

*Фреймворк* – полностью асинхронная оболочка *aiogram*. Его особенностью является возможность построения модели взаимодействия с пользователем с использованием технологий *конечного автомата*. То есть ситуационные ответы со стороны Telegram-бота могут контролироваться не только сообщениями пользователя и бизнес-логикой, обрабатывающей эти сообщения, но и логикой состояний абстрактного автомата, описывающего тот или иной этап опроса. Этот механизм нашел свое применение в модуле пошагового составления заявки на внесение новой записи базы данных, где существует необходимость предоставления пользователю выбора, когда сбор некоторого блока информации можно считать законченными.

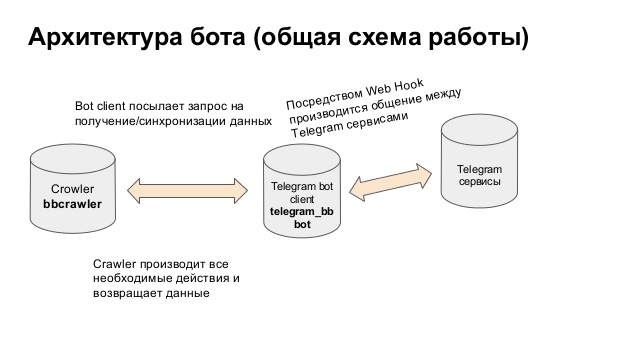


Рисунок 7 – Схема устройства взаимодействия с Telegram-ботом

**2 РАЗРАБОТКА СТРУКТУРЫ И МОДУЛЕЙ  
ПРОГРАММНОЙ СИСТЕМЫ**

В данном разделе рассмотрены вопросы реализации программного продукта: взаимодействия спроектированных активностей и отдельных модулей мобильного приложения, построения системы для удаленного обмена данными, аутентификации пользователей приложения, модели представления данных. Описаны также ключевые моменты, связанные с интеграцией сторонних корпоративных сервисов в работу существующего мобильного приложения.

* 1. **Описание структуры мобильного приложения Android**

Как и любое другое Android-приложение, разрабатываемое в рамках текущей работы состоит из модулей, объединяющих в себе алгоритмы для генерации и взаимодействия с пользовательским интерфейсом и внутренней бизнес-логикой приложения. Такие модули называют активностями (англ. Activity). На рисунке 8 представлена схема вызовов разных активностей приложения во время его работы.

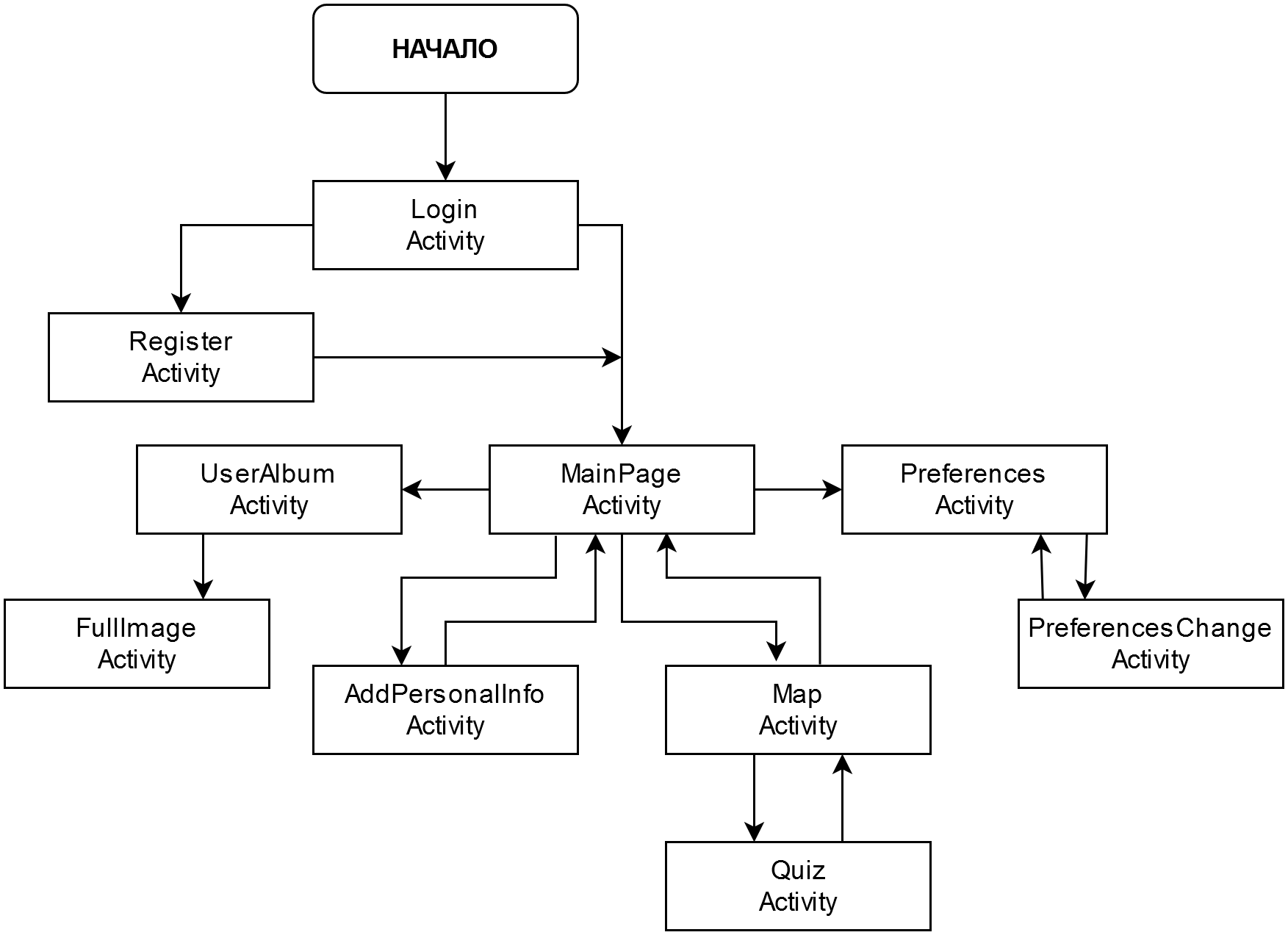


Рисунок 8 – Схема взаимодействия активностей приложения

Вызовы, обозначенные одинарной стрелкой – вызовы по умолчанию, предназначенные для последовательного выполнения активностей в соответствии с логикой работы приложения. Двусторонние стрелки – вызов активности для получения некоторого результата (например, число полученных за викторину баллов, сигналов о корректном или некорректном выполнении модуля). Как видно на приведенной схеме, большая часть таких отношений связана с активностью *MainPageActivity*. Приведем функциональное назначение этой и некоторых других активностей приложения, а также их ключевых методов:

1. LoginActivity. Активность предназначена для аутентификации пользователей приложения: обработки пары «электронная почта - пароль» на предмет их наличия на удаленном сервере, контроля входа и регистрации в системе по умолчанию, либо при помощи сторонней сети (аккаунта Google пользователя). Перенаправляет на активности:

* активность главного меню MainPageActivity в случае успешной аутентификации;
* активность регистрации в системе *RegisterActivity* при нажатии соответствующей кнопки пользователем;
* активность аутентификации в системе Google, по успешному завершению которой так же происходит переход к главной активности.

Метод defaultLogin() (листинг функции представлен в Приложении Б). Метод производит попытку входа в систему при помощи заданной пары «электронная почта - пароль». В зависимости от значения аргумента registerIfNotRegistered может произвести регистрацию пользователя в случае его отсутствия в системе (используется для регистрации пользователей с использованием персональной информации из сторонних сетей).

2. RegisterActivity. Активность предназначена для регистрации новых пользователей в системе. Контролирует ввод информации в соответствующие поля: «логин», «пароль», «электронная почта», производит отправку запроса на создание учетной записи новых пользователей приложения: обработки пары «электронная почта - пароль» на предмет их наличия на удаленном сервере, обработку регистрации в системе при помощи сторонней сети (аккаунта Google пользователя). Перенаправляет на активность MainPageActivity в случае успешной регистрации.

3. MainPageActivity. Основная активность приложения, «главное меню» пользователя. Отображает логин пользователя, его текущее местоположение (город на выбор с учетом страны пребывания) с возможностью смены, число полученных баллов и несколько функциональных клавиш:

* перехода к активности *AddPersonalInfoActivity* для получения дополнительной информации о пользователе;
* перехода к активности *MapActivity* для выбора места в указанном городе для прохождения тестирования;
* перехода к активности *UserAlbumActivity* для отображения тематических фотографий, хранящихся на устройстве пользователя.

Некоторые указанные переходы производятся по модели «переход для результата», то есть после завершения работы активности, производится обработка ее результатов в вызывающей активности (например, редактирование данных на удаленных сервисах, на устройстве пользователя). Листинг обработчиков результатов работы этих активностей представлен в Приложении В.

Приватные методы checkBonus() и hideInterest() предназначены для скрытия интерфейса, связанного с получением дополнительной информации от пользователя. В случае если пользователь уже указал сведения о себе, возможность перехода к этой активности повторно утрачивается.

4. PreferencesChangeActivity. Активность предназначена для просмотра и изменения настроек пользователя, таких как: пароль, имя пользователя, логин, адрес электронной почты, страна текущего пребывания. После выбора необходимого пункта производит попытку обновления информации на удаленном сервере, а после и на устройстве пользователя.

5. MapActivity. Активность для выбора места прохождения викторины, завязанная на использовании сервисов *Google Mobile Service* для отображения карты, меток, описаний и прочего. При выборе места для прохождения тестирования производит попытку получения информации о месте из удаленного сервиса, переход к активности для прохождения викторины *QuizActivity* по модели «переход для результата», ретранслируя результаты работы этой активности в неизменном виде обратно в главную активность MainPageActivity. Листинг описания класса представлен в Приложении Г.

Метод getCurrentDeviceLocation() получает текущее местоположение устройства (при наличии необходимых разрешений) и центрирует вид сформированной карты на нем.

6. QuizActivity. Активность для прохождения викторины, связанной с выбранным местом на карте. Отображает изображение достопримечательности, вопрос и список объектов типа *Button* в виде вариантов ответа. Переключение вопросов производится при помощи кнопок «Вперед» и «Назад». Листинг «*слушателя нажатий*» (англ. Click Listener) кнопки «Вперед» представлен в приложении Д.

Метод updateQuestion() обновляет текущий вопрос о достопримечательности: меняет текст вопроса на актуальный, перенастраивает адаптер, отвечающий за отображение списка кнопок с ответами. Листинг функции представлен в приложении Е.

* 1. **Функциональные модули Android-приложения**

Подраздел предназначен для описания некоторых дополнительных модулей приложения, которые задействованы в алгоритмах приведенных ранее активностей.

* + 1. **Реализация различных способов обработки данных  
       пользователя**

В существующей работе были использованы разные подходы к хранению и получению данных:

* Файлы в локальном хранилище приложения.
* Технология Shared Preferences.
* Данные на удаленном сервере.
* Графические изображения в сети Интернет.

Использование разных моделей обработки информации продиктовано функциональным назначением. Так, например, общие настройки используются для хранения информации о текущем аккаунте пользователя во внутренней памяти устройства. Доступ к этому типу данных осуществляется быстрее всего, что позволяет снизить количество обращений к удаленному серверу без крайней необходимости.

Данные на удаленном сервере могут иметь слишком большой объем для хранения на устройстве и могут подвергаться изменениям, требующим постоянного обновления базы данных на самом устройстве. Кроме того, база данных может оперировать конфиденциальной информацией пользователей приложения.

В качестве моделей, спроектированных для упрощения работы с данными приложения, выступают следующие классы:

1. *ArcantownAccount*. Содержит в себе информацию об аккаунте пользователя: идентификатор, адрес электронной почты пользователя, его имя и прочее. Содержит два *конструктора*: один для генерации экземпляра класса из объекта нотации *JSON*, полученной с удаленного сервера, а другой – для создания объекта используя локальное хранилище типа Shared Preferences. Используется для получения, изменения пользовательских данных, их синхронизации с удаленными сервисами.

2. *QuizData*. Используется в активности QuizActivity в качестве объекта, содержащего всю необходимую информацию для проведения тестирования: объект класса *Place*, содержащий всю необходимую информацию о выбранной пользователем достопримечательности, и список вопросов (объектов класса *Question*). Включает в себя методы getNext() и getPrevious(), предназначенные для получения следующего или предыдущего вопроса во время проведения викторины. Листинг описания класса представлен в Приложении Ж.

3. *Place*. Содержит информацию о месте, выбранном для тестирования: координаты на карте, *идентификатор ресурса* (англ. *Uniform Resource Identifier*, *URI*) изображения, название достопримечательности.

4. *Question*. Содержит информацию об отдельном вопросе викторины: идентификатор места, с которым он связан, текст вопроса, список ответов (объектов класса *Answer*).

5. *Answer*. Содержит в себе информацию об ответе на заданный вопрос викторины: идентификатор вопроса, текст ответа, маркер истинности.

Все вышеперечисленные классы содержат в себе конструкторы объектов, использующих для генерации своих экземпляров строки с объектами нотации JSON. Таким образом, отсутствует необходимость для каждого объекта разрабатывать сложные конструкторы, принимающие на вход большое количество аргументов для заполнения: класс более высокой иерархии (например, Question по отношению к Answer) получает из переданного ему объекта JSON необходимую ему информацию. Для создания и включения объектов более низкой иерархии используется оставшаяся информация из этого объекта.

Для представления некоторых данных используются специальные *объекты-адаптеры* (англ. *Adapters*), предназначенные для представления данных в качестве отдельных видов [36] в пользовательском интерфейсе:

1. *ButtonAdapter*. Используется для отображения кнопок с ответами в активности QuizActivity (во время проведения викторины). С его помощью осуществляется генерация объектов типа Button, их представление в объекте разметки *ListView* (список однотипных объектов), проверка на нажатие той или иной кнопки (хранение индекса последней нежатой кнопки). Листинг метода генерации вида адаптера представлен в Приложении И.

2. *CheckBoxAdapter*. Используется для отображения чекбоксов (или «флажков») в активности AddPersonalInfoActivity (добавление дополнительной информации о пользователе). С его помощью производится генерация объектов вида типа *CheckBox*, их представление в списке тем на выбор, вывод выбранных пользователем параметров.

3. *ImageAdapter*. Используется для отображения изображений в активности *UserAlbumActivity* (альбом с фотографиями пользователя). С его помощью производится генерация объектов вида типа *ImageView*, их представление в объекте разметки типа *GridView* (сетка однотипных объектов).

Использование адаптеров рекомендуется [37] для представления списка однотипных видов пользователю. Так, например, в случае с отображением возможных ответов на текущий вопрос существует возможность вместо использования списков объектов и соответствующих им адаптеров всегда отображать только лишь ограниченное количество кнопок с ответами. В таком случае разработка пользовательского интерфейса, отслеживание нажатий может быть организована гораздо проще, но тогда в системе возникнет ограничение, которое всегда нужно иметь в виду при разработке остальных функциональных модулей.

* + 1. **Асинхронные алгоритмы получения данных**

Во время разработки приложения, работа которого основана на коммуникации с удаленными сервисами, зачастую возникает необходимость в отображении тех или иных элементов пользовательского интерфейса после получения некоторой информации, при этом, не задерживая отображение остальной части пользовательского интерфейса. Для этого в операционной системе Android был разработан механизм *асинхронного выполнения* *AsyncTask*, с помощью которого некоторый процесс можно поместить в фоновый поток, не дожидаясь его полного исполнения для продолжения работы с приложением.

В существующей работе объекты асинхронного исполнения используются для получения информации с удаленного сервера: списка стран, городов в системе, информации об аккаунте пользователя для аутентификации или изменении данных и так далее. Для этой цели был реализован класс *ReceiveDataFromMySQLTask* (листинг класса представлен в Приложении К), наследующий ранее упомянутый интерфейс AsyncTask. В нем переопределен наследуемый метод doInBackground(), описывающий алгоритм, исполняющийся в отдельном потоке. Этот метод производит настройку соединения, подключение к серверу и передачу ему строк-меток, указанных в качестве аргументов (так, например, для получения списка стран вносится аргумент countries) для получения необходимых для работы приложения данных.

В свою очередь, для каждой отдельной задачи реализованы свои приватные классы, наследующие класс ReceiveDataFromMySQLTask, в которых переопределяются методы onPreExecute() и onPostExecute(), описывающие алгоритмы, производимые до запуска процесса в отдельном потоке и после соответственно. С их помощью можно, например, отобразить диалог ожидания получения информации с сервера до момента ее получения и анимировать обновленный объект после.

* 1. **Структура сервисов для удаленного взаимодействия с устройством пользователя**

Профиль разрабатываемого приложения предполагает его использование многими пользователями в системе, контролирующей проведение викторин, обрабатывающей и хранящей ее результаты. Таким образом, при помощи сервера происходит управление учетными записями пользователей, верификация их свойств (в частности, баллов, полученных за тестирование) и отправка информации на их устройства из единого источника. На рисунке 9 показана схема организации системы для удаленного сообщения с устройством каждого пользователя.

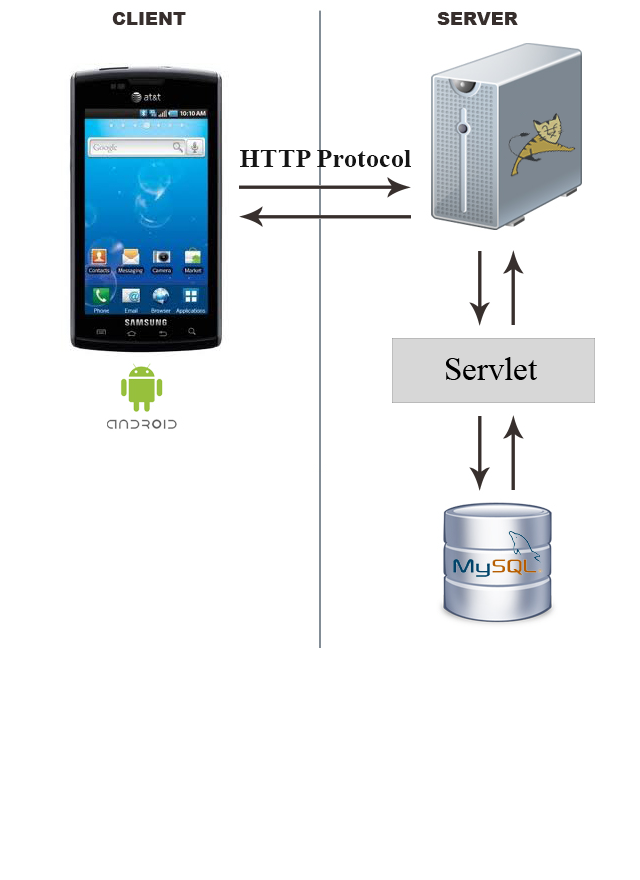


Рисунок 9 – Схема системы удаленных сервисов приложения

* + 1. **Описание базы данных системы**

В качестве системы управления базами данных была выбрана MySQL. Опишем назначение сконструированных таблиц данных:

* *accounts* – информация о персональных данных пользователя. Используется для получения данных после аутентификации, при регистрации нового пользователя, при изменении данных со стороны пользователя или приложения.
* *countries* – список стран, доступных в системе. Используется при выборе страны для прохождения викторины.
* *towns* – список городов, доступных в системе. Используется при выборе города для прохождения викторины.
* *places* – список мест, доступных для прохождения тестирования. Включает в себя название, ссылку на изображение, координаты места (пара «широта-долгота»), название страны и города расположения и прочую информацию. Используется для отображения меток на карте в мобильном приложении.
* *interests* – список возможных для выбора интересов пользователя. Используется при генерации интерфейса пользовательского приложения при получении дополнительной информации в активности AddPersonalInfoActivity.
* *questions* – таблица с вопросами. Содержит в себе идентификатор места, к которому относится заданный вопрос, текст вопроса, его тип (текстовый или графический), список тематик, которым вопрос удовлетворяет.
* *answers* – таблица с ответами, приложенным к некоторому вопросу. Содержит в себе идентификатор вопроса, к которому ответ имеет отношение, текст ответа, признак истинности (0 или 1), опциональную ссылку на изображение.

Подключение к базе данных из сервлета осуществляется при помощи технологии *MySQL Connection* локально (по IP-адресу 127.0.0.1) через порт 3306 с использованием пароля и логина администратора.

* + 1. **Сервлет для обработки пользовательских запросов**

Для обслуживания запросов клиента используется сервлет Java, наследующий интерфейс *HttpServlet*, который предназначен для обработки запросов типа GET. В зависимости от аргументов запроса (URL-меток, вписанных после адреса сервлета), сервлет при помощи объекта *PrintWriter* выводит [38] в выходной текстовый поток ту или иную информацию. Перечислим возможные запросы к серверу, поддающиеся обработке и генерированию ответа:

* «Сервер-null». В случае обращения непосредственно к веб-архиву производится вывод таблицы со списком команд, поддающихся обработке на сервлете.
* «Сервлет-null». В случае обращения к сервлету без указания меток выводится текст следующего содержания: "NULL\_ERROR". Используется при конфигурации сервлета: может использоваться в качестве эхо-сигнала со стороны клиента.
* want=specs. Список специальностей (интересов) на выбор. Генерирует объект JSON, содержащий массив с именем "specs" доступных в системе специальностей. В случае возникновения ошибки выводит текст следующего содержания: "false".
* want=countries. Список стран, доступных в системе. Генерирует объект JSON, содержащий массив с именем "countries" с названиями стран. В случае возникновения ошибки выводит текст следующего содержания: "false".
* want=towns&country=[название\_страны]. Список городов, находящихся в указанной стране. Генерирует объект JSON, содержащий массив с именем "towns" с названиями городов. В случае возникновения ошибки выводит текст следующего содержания: "false".
* want=places&country=[назв\_страны]&town=[назв\_города]. Список достопримечательностей, находящихся в указанном городе. Генерирует объект JSON, содержащий массив с именем "places" с названиями мест. В случае возникновения ошибки выводит текст следующего содержания: "false".
* want=account&email=[эл\_почта]. Информация об аккаунте с указанным адресом электронной почты. Генерирует объект JSON, содержащий объект с именем "account" с данными о пользователе. В случае возникновения ошибки выводит текст следующего содержания: "false". Листинг методов, задействованных в генерации ответа сервера, представлен в Приложении Л.
* want=put\_acc&auth\_type=[тип]&email=[эл\_почта]&login=[лог]. Внесение новой учетной записи в систему. В ответ генерирует объект JSON, содержащий объект с именем "account" с данными о новом пользователе. В случае возникновения ошибки выводит текст следующего содержания: "false".
* want=quiz&place\_id=[ид\_места]. Информация для проведения викторины. Генерирует объект JSON, содержащий объекты: "place" с данными о месте проведения, массив "questions" с объектами, содержащими вопросы к викторине. В случае возникновения ошибки выводит текст следующего содержания: "false".
* want=change&email=[эл\_почта]&field=[изменяемое\_поле]  
  &value=[новое\_значение]. Внесение изменений в учетную запись пользователя по указанному полю. В ответ генерирует объект JSON, содержащий объект с именем "account" с измененными данными пользователя. В случае возникновения ошибки выводит текст следующего содержания: "false".
* want=update\_quiz&email=[эл\_почта]&place\_id=[ид\_места]  
  &points=[число\_очков\_за\_викторину]. Обновление данных учетной записи (внесение идентификатора пройденной викторины и инкрементация очков, полученных за ее выполнение). В ответ генерирует объект JSON, содержащий объект с именем "account" с измененными данными пользователя. В случае возникновения ошибки выводит текст следующего содержания: "false".
  1. **Разработка и внедрение модулей с использованием  
     сторонних сервисов**

В данной работе используются несколько сервисов, предназначенных для расширения возможностей приложения. Так, например, сервис Firebase дает возможность реализовать безопасную аутентификацию пользователей с возможностью управления существующими учетными записями.

Учитывая профиль разрабатываемого приложения (интернациональность и ориентированность на использовании геоинформационных систем – картографических служб), возникает необходимость в использовании современных электронных карт с простой в использовании программного оболочкой, имеющих наибольшую область своего покрытия в мире. Сервис Google Maps удовлетворяет всем этим требованиям [39].

Для осуществления обратной связи с пользователем приложения была использована технология *Telegram Bots* [40], завязанная на работе со сторонним мобильным приложением *Telegram*. Для этого был разработана отдельная система, обслуживающая работу автоматизированной системы для обработки пользовательских запросов.

* + 1. **Интерфейс картографии Google Maps**

Интеграция сервиса в приложение включает в себя его регистрацию на единой платформе для разработчиков *Google Cloud*: создание проекта, генерация *токена* (уникального ключа, идентификатора приложения), настройка программной оболочки, сервисов.

Для работы с фрагментов карты в активности достаточно внести наследование интерфейса *OnMapReadyCallback* при объявлении активности. Для работы с картой после ее загрузки был переписан метод onMapReady(), который инициализирует модуль: вносит дополнительные элементы управления, иконки, определяет местоположение пользователя, вносит логику реагирования на действия пользователя. В случае если необходимые сервисы не установлены на устройство пользователя, либо не был получен ряд важных разрешений, приложение запросит их в отдельном окне.

Дополнительно было разработано окно поиска места *search\_box*. За обработку внесенной в текстовое поле информации отвечает метод geoLocate(), считывающий содержимое поля и генерирующий список результатов поиска: объектов типа *Address*. В случае нахождения указанного места он также производит центрирование на новой метке. Также для этого окна была переработана логика ввода текста: теперь после нажатия клавиши *Enter* на экранной клавиатуре устройства инициируется поиск (вместо перехода на новую строку), а сама клавиатура скрывается.

* + 1. **Интеграция библиотек сервиса аутентификации Firebase**

Как и в случае с сервисами Google Maps, для интеграции Firebase необходимо зарегистрировать приложение в системе *Firebase Console*.

После добавления всех необходимых библиотек в проектируемое приложение необходимо разработать логику аутентификации и добавления новых пользователей в систему. Для этого используется объект класса *FirebaseAuth* и его методы signInWithEmailAndPassword() и createUserWithEmailAndPassword() для попытки входа в систему и регистрации нового пользователя соответственно.

* + 1. **Анкетирование с использованием приложения Telegram и программной оболочки Aiogram API**

Для представления данных внутри приложения были разработаны два модуля: *UserRequest* и *Question*.

Модуль UserRequest содержит данные, полученные от пользователя в ходе опроса:

* идентификатор пользователя (целочисленная переменная);
* описание запроса (строка);
* описание места (строка);
* список вопросов (объектов типа Question).

Для создания экземпляра этого класса необходимо поместить в конструктор, помимо прочей информации, двумерный массив строк. Столбец двумерного массива с нулевым индексом будет содержать строки с вопросами, а оставшиеся столбцы – ответы, указанные пользователем (кроме последней, которая хранит индекс правильного ответа).

Класс Question содержит два элемента:

* текст вопроса (строка);
* ответы (массив строк).

Класс Question является служебным по отношению к UserRequest и для его создания необходимо передать в конструктор экземпляра соответствующий массив строк двумерного массива данных.

Модули *BotSingle* и *my\_utils* отвечают за бизнес-логику сервиса. *BotSingle* производит инициализацию приложения:

* настройку подключения;
* настройку прокси;
* создание сущности бота с использованием уникального *токена* Telegram-бота;
* выделение локальной памяти для хранения временной информации;
* создание обработчика сообщений.

Помимо этого, он содержит всю совокупность обработчиков сообщений от пользователя и группу состояний, с помощью которых производится выбор приоритетного обработчика сообщения. Листинг одного из обработчиков представлен в Приложении М.

*my\_utils* содержит следующие служебные функции:

* is\_moderator() (проверка идентификатора пользователя на наличие у последнего полномочий модератора, редактора запросов, в текущей системе);
* login\_moderator() (аутентификация модератора по паролю-логину, занесение идентификатора пользователя Telegram в файл с массивом текущих модераторов);
* add\_moderator() (добавление нового идентификатора в файл);
* get\_phrase\_from\_res() (метод для селекции строк с ответами системы пользователю на соответствующем ему языке).

За работу с данными, хранящимися на сервере держателя Telegram-бота, отвечает метод text\_storing() и методы аутентификации из модуля *my\_utils*. Первый производит запись запроса от пользователя в файл *new\_data.txt* в директории *data* в соответствии с логикой, описанной в файле *BotSingle* (то есть по окончании ввода пользователем данных командой ***/****cancel*). Остальные производят сверку данных о пользователе с содержимым файлов *moderators* (идентификаторы действующих модераторов) и *moderators\_auth* (ключи «логин-пароль» модераторов).

Для более наглядного обращения к ресурсам приложения, связанным с репликами бота, был реализован отдельный модуль *JsonTags*. Класс предназначен для хранения статических строк, содержащих ключи (нотации JSON) локализованных (для русского, английского и немецкого языков) данных. Пример некоторых полей:

* *WELCOME* – ключ строки приветствия пользователя;
* *WELCOME\_MODERATOR* – ключ строки приветствия модератора;
* *AUTH\_WELCOME* – ключ строки приветствия вновь авторизовавшегося модератора;
* *AUTH\_FAIL\_MODERATOR* – ключ строки, выводимой пользователю при неудавшейся попытке авторизации;
* *HELP\_USER* – ключ строки со справочной информацией для пользователя;

Ключи в этом классе (наряду с информацией о системном языке текущего пользователя) используются для выбора необходимой реплики в файле *inter\_lang* (содержимое файла представлено в Приложении Н), содержащим соответствующие строки для ответа пользователю.

**3 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОГО ПРОЕКТА**

Дипломный проект посвящен разработке и реализации мобильного приложения для знакомства с умным городом. В ходе работы над выпускной квалификационной работой были изучены технологии построения мобильного приложения в операционной системе Android, разработки и настройки удаленных сервисных систем для взаимодействия с клиентом приложения.

Экономическая целесообразность выпускной квалификационно работы бакалавра оценивалась по результатам выполнения следующих пунктов:

* составление детализированного плана выполнения работ;
* оценка продолжительности работ для каждого из участников команды;
* оценка размеров заработной платы и социальных отчислений;
* расчет материальных затрат: комплектующие, накладные расходы, обслуживание системы, себестоимость производства;
* обзор способов реализации разрабатываемого продукта;
* обзор и оценка конкурентной среды: потребность, рынки сбыта, сервисное обслуживание, цена и неценовые детерминанты;
* выводы об экономической целесообразности разрабатываемого продукта.
  1. **Составление детализированного плана выполнения работ**

Опишем совокупность работ, которые необходимо выполнить в рамках заданной работы. Главные этапы проектирования вместе с их подробным описанием приведены в Таблице 1.

План включает в себя следующие этапы: составление требований к разрабатываемому продукту, сбор команды разработчиков (которая может включать как технических специалистов, так и маркетологов и специалистов по связям с общественностью), разработка технического задания, сборка прототипа, исполняющего полностью или частично функции конечного продукта, разработка, тестирование системы, доработка. В самом конце результаты всего проделанного комплекса работ сводятся в обобщающий документ, пояснительную записку.

Таблица 1 – План выполнения работы

|  |  |
| --- | --- |
| Этапы проектирования | План работ |
| Составление требований | * Составление краткого описания приложения. * Выбор платформы и языка программирования для реализации. * Выделение в рамках темы работы первостепенных задач (модулей) для проектирования. * Выделение второстепенных задач проектирования. |
| Сбор команды разработчиков | * Нахождение технического разработчика. * Нахождение маркетолога. |
| Разработка ТЗ | * Общие положения. * Назначение и цели создания (развития) системы. * Характеристика создаваемого продукта. * Требования к системе. * Состав и содержание работ по созданию системы. * Порядок контроля и тестирования системы. * Требования к составу и содержанию работ по вводу системы в действие. * Требования к документированию. |
| Сборка прототипа | * Описание требований к прототипу создаваемого продукта. * Выделение из перечня первостепенных задач проектирования достаточного количества функций для демонстрации работы приложения. * Разработка прототипа. |
| Разработка и тестирование | * Разработка модуля для идентификации. * Разработка модуля для профориентации. * Разработка модуля главного меню. * Разработка модуля настроек. * Разработка модуля для работы с картами. * Разработка модуля для работы с маршрутом. * Разработка модуля для внесения данных со стороны разработчика. * Разработка модуля для тестирования. * Разработка модуля с рейтинговой таблицей. * Тестирование. |
| Оформление пояснительной записки | * Написание чернового варианта документа. * Согласование с руководителем, правки. * Составление пояснительной записки. |

* 1. **Оценка продолжительности выполнения работ для каждого из участников команды**

Продолжительность этапов «Составление требований» и «Сбор команды разработчиков» определяется по факту затраченного времени. Так, например, составление и обсуждение краткого описания приложения для разработчика суммарно заняло 5 часов, а выбор платформы и языка программирования – 0,5 часа. Сводная таблица для описанных этапов представлена в Таблице 2.

Таблица 2 – Составление требований и сбор команды разработчиков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Этапы проектирования | Описание работы | Время выполнения, чел.ч. | | |
| Разработчик | Маркетолог | Руководитель |
| Составление требований | Составление краткого описания приложения | 5,0 | 2,0 | 1,0 |
| Выбор платформы и языка программирования для реализации | 0,5 | - | 0,5 |
| Выделение в рамках темы работы первостепенных задач (модулей) для проектирования | 3,5 | 3.5 | 1,0 |
| Выделение второстепенных задач проектирования | 3,0 | 3,0 | 1,0 |
| Сбор команды разработчиков | Нахождение технического разработчика | 0.5 | 0.5 | 0.5 |
| Нахождение маркетолога | 0.5 | 0.5 | 0.5 |

Продолжительность выполнения остальных этапов описывается с помощью экспертных оценок по следующей формуле:

где:

* – ожидаемая длительность текущей работы.
* – наименьшая оценка длительности исполнения работы.
* – наибольшая оценка длительности исполнения работы.

Так, например, для описания общих положений технического задания по экспертной оценке потребуется от 0,5 до 1,5 часов. Следовательно, ожидаемая длительность этой работы будет равна

0,9 часов

Рассчитаем оценки по оставшимся этапам. Таблица, содержащая оцениваемую продолжительность выполнения последних четырех этапов проекта, представлена в Таблице 3.

После вычислим ожидаемую длительность выполнения каждой из описанных работ. Сводная таблица, содержащая продолжительность выполнения всех этапов разработки проекта представлена в Таблице 4.

Таблица 3 – Экспертная оценка необходимого для проектирования времени

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Этапы проектирования | Описание работы | Минимальная оценка времени исполнения, чел.ч. | | | Максимальная оценка времени исполнения, чел.ч. | | |
| Разработчик | Маркетолог | Руководитель | Разработчик | Маркетолог | Руководитель |
| Разработка ТЗ | Общие положения | 1,0 | 1,0 | 1,0 | 2,5 | 1,5 | 1,5 |
| Назначение и цели создания (развития) системы | 0.5 | 1,5 | - | 1,0 | 2,0 | - |
| Характеристика создаваемого продукта | 1,0 | 1,0 | 0,5 | 2,0 | 1,5 | 1,0 |
| Требования к системе | 0,5 | - | 0,5 | 1,5 | - | 1,0 |
| Состав и содержание работ по созданию системы | 1,0 | - | 1,0 | 2,5 | - | 2,0 |
| Порядок контроля и тестирования системы | 1,0 | - | 0,5 | 2,5 | - | 1,0 |
| Требования к составу и содержанию работ по вводу системы в действие | 1.5 | 0,5 | 0,5 | 2,5 | 1,0 | 1,0 |
| Требования к документированию | 1,0 | - | 1,0 | 2,0 | - | 1,5 |
| Сборка прототипа | Описание требований к прототипу создаваемого продукта | 0,5 | 0,5 | - | 1,5 | 1,0 | - |
| Выделение из перечня первостепенных задач проектирования достаточного количества функций для демонстрации работы приложения | 3,5 | 2,5 | 1,5 | 5,0 | 3,5 | 2,5 |
| Разработка прототипа | 14,0 | - | 1,0 | 21,0 | - | 2,5 |
| Разработка и тестирование | Разработка модуля для идентификации | 3,0 | - | - | 5,5 | - | - |
| Разработка модуля для профориентации | 1,5 | - | - | 3,5 | - | - |
| Разработка модуля главного меню | 2,0 | - | 0,5 | 5,5 | - | 2,0 |
| Разработка модуля настроек | 2,5 | - | - | 5,0 | - | - |
| Разработка модуля для работы с картами | 10,0 | - | 1,5 | 17,5 | - | 3,0 |
| Разработка модуля для работы с маршрутом | 2,0 | - | 1,0 | 4,0 | - | 2,0 |
| Разработка модуля для внесения данных со стороны разработчика | 2,5 | - | 1,0 | 4,5 | - | 2,0 |
| Разработка модуля для тестирования | 3,0 | - | 0,5 | 6,5 | - | 2,0 |
| Разработка модуля с рейтинговой таблицей | 2,5 | - | 0,5 | 4,0 | - | 1,0 |
| Тестирование | 2,0 | - | 0,5 | 3,5 | - | 1,5 |

Продолжение таблицы 3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Оформление пояснительной записки | Написание чернового варианта документа | 5,0 | 2,0 | 1,0 | 7,5 | 3,0 | 2,0 |
| Согласование с руководителем, правки | 3,5 | - | 1,0 | 6,5 | - | 2,5 |
| Составление пояснительной записки | 4,5 | - | 1,0 | 6,5 | - | 2,0 |

Таблица 4 – Трудоемкость выполнения проекта

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Этапы проектирования | Описание работы | Трудоемкость, чел.ч. | | |
| Разработчик | Маркетолог | Руководитель |
| Составление требований | Составление краткого описания приложения | 5,0 | 2,0 | 1,0 |
| Выбор платформы и языка программирования для реализации | 0.5 | - | 0,5 |
| Выделение в рамках темы работы первостепенных задач (модулей) для проектирования | 3,5 | 3,5 | 1,0 |
| Выделение второстепенных задач проектирования | 3,0 | 3,0 | 1,0 |
| Сбор команды разработчиков | Нахождение технического разработчика | 0.5 | 0,5 | 0,5 |
| Нахождение маркетолога | 0.5 | 0,5 | 0,5 |
| Разработка ТЗ | Общие положения | 1,6 | 1,2 | 1,2 |
| Назначение и цели создания (развития) системы | 0,7 | 1,7 | - |
| Характеристика создаваемого продукта | 1,4 | 1,2 | 0,7 |
| Требования к системе | 0,9 | - | 0,7 |
| Состав и содержание работ по созданию системы | 1,6 | - | 1,4 |
| Порядок контроля и тестирования системы | 1,6 | - | 0,7 |
| Требования к составу и содержанию работ по вводу системы в действие | 1,9 | 0,7 | 0,7 |
| Требования к документированию | 1,4 | - | 1,2 |
| Сборка прототипа | Описание требований к прототипу создаваемого продукта | 0,9 | 0,7 | - |
| Выделение из перечня первостепенных задач проектирования достаточного количества функций для демонстрации работы приложения | 4,1 | 2,9 | 1,9 |
| Разработка прототипа | 16,8 | - | 1,6 |

Продолжение таблицы 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Разработка и тестирование | Разработка модуля для идентификации | 4 | - | - |
| Разработка модуля для профориентации | 2,3 | - | - |
| Разработка модуля главного меню | 3,4 | - | 1,1 |
| Разработка модуля настроек | 3,5 | - | - |
| Разработка модуля для работы с картами | 13 | - | 2,1 |
| Разработка модуля для работы с маршрутом | 2,8 | - | 1,4 |
| Разработка модуля для внесения данных со стороны разработчика | 3,3 | - | 1,4 |
| Разработка модуля для тестирования | 4,4 | - | 1,1 |
| Разработка модуля с рейтинговой таблицей | 3,1 | - | 0,7 |
| Тестирование | 2,6 | - | 0,9 |
| Оформление документации | Написание чернового варианта документа | 6 | 2,4 | 1,4 |
| Согласование с руководителем, правки | 4,7 | - | 1,6 |
| Составление пояснительной записки | 5,3 | - | 1,4 |
| Сумма | | 104,3 | 20,3 | 27,7 |

* 1. **Оценка размеров заработной платы и социальных отчислений**

На основе полученных данных о трудоемкости выполняемых работ необходимо определить расходы на заработную плату разработчика. Определим оклад исходя из среднего ежемесячного оклада Android-разработчика в *S* = 53,247 тыс. рублей [41], числа рабочих дней в календарном месяце *d* = 21 и количества затрачиваемых часов *t* = 8:

317 рублей/чел.ч.

Аналогичный расчет проведем для остальных участников проекта:

Исходные данные:

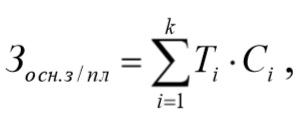
* *Sмар* = 59,000 тыс. рублей [42].
* *Sрук* = 74,000 тыс. рублей [43].
* *d* = 21 день.
* *t* = 8 часов.

Таким образом, ставка оставшихся участников проекта составляет:

351рубль/чел.ч.

440рублей/чел.ч.

Расходы на основную заработную плату вычисляются по следующей формуле:



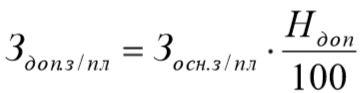
где:

* *Зосн.з/пл* - расходы на основную заработную плату исполнителей (руб.).
* *k* – количество исполнителей.
* *Ti*- время, затраченное i-м исполнителем на проведение исследования (дни или часы).
* *Ci*- ставка i-го исполнителя (руб./день или руб./час).

Следовательно, основная заработная плата технического специалиста равна:.

Заработная плата маркетолога: .

Заработная плата руководителя проекта: . Расчет дополнительной заработной платы производится по формуле:

**

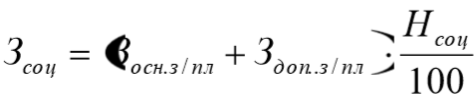
Расходы на дополнительную заработную плату, исходя из норматива заработной платы, равной 14%, становятся равны соответственно:

* *Здоп/раз* = .
* *Здоп/мар* = .
* *Здоп/рук* = .

Суммы расходов на оплату труда:

* .
* .
* .

Расходы на отчисления по страховым и социальным взносам вычисляются по следующей формуле:



Расходы на социальные отчисления с заработной платы, исходя из ставки обязательного страхования в 30%, становятся равны соответственно:

* *Зсоц/раз* = .
* *Зсоц/мар* = .
* *Зсоц/рук* = .

Сводная таблица, содержащая затраты на заработную плату участникам разрабатываемого проекта представлена в Таблице 5.

Таблица 5 – Затраты на оплату труда основных исполнителей

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Должность | Основная заработная плата, тыс. руб. | Дополнительная заработная плата, тыс. руб. | Социальные отчисления, тыс. руб. | Сумма, тыс. руб. |
| Разработчик |  | 4,629 | 11,308 | 49,000 |
| Маркетолог |  | 0,998 | 2,437 | 10,560 |
| Руководитель |  |  | 4,168 | 18,063 |
| Итоговая сумма по всем участникам | | | | 77,623 |

* 1. **Расчет материальных затрат на комплектующие, накладные расходы, обслуживание системы**

Стоимость покупных комплектующих с учетом усредненной стоимости доставки отдельного изделия, равной 500 рублей:

где:

* *Ni* – количество *i*-х комплектующих изделий, входящих в единицу продукции (шт.).
* *Цi* – цена приобретения *i*-го комплектующего (руб./шт.).

Рассчитаем стоимость покупных изделий для выполнения и поддержания работы проекта, предполагая именно покупку необходимого оборудования для реализации разработки:

1. Мобильный телефон с сенсорным дисплеем для тестирования работы приложения (стоимость = 11,770 тыс. рублей), 1 шт. [44].
2. Компьютер для разработки и занесения, расширения данных, необходимых для работы приложения (стоимость = 23,580 тыс. рублей), 1 шт. [45].

Таким образом, суммарная стоимость покупных комплектующих становится равна:

Рассчитаем величину затрат на эксплуатацию оборудования. Исходные данные:

* Планируемое время аренды серверного оборудования *t* = 80 суток.
* Стоимость ежемесячной аренды *V* = 6,900 тыс. рублей [46].

Таким образом, затраты на использования серверов течение трех месяцев становятся равны:

Рассчитаем величину затрат сторонних организаций. Исходные данные:

* Услуги связи и интернет: .
* Оплата дизайнера логотипа: .
* Реклама и маркетинг: .
* Расходы на проведение информационно-консультативных мероприятий: .

Таким образом, сумма затрат по статье равна:

Итоговая смета затрат на реализацию проекта представлена в Таблице 6.

Таблица 6 – Смета затрат проекта

|  |  |
| --- | --- |
| Наименование статьи | Сумма, тыс. руб. |
| Расходы на оплату труда | 59,710 |
| Отчисления на социальные нужды | 17,913 |
| Затраты на приобретение оборудования | 36,350 |
| Затраты на эксплуатацию | 20,700 |
| Затраты на оплату услуг сторонних организаций | 20,960 |
| Накладные расходы (20% от ФОТ) | 15,525 |
| Итого | 172, 158 |

* 1. **Обзор способов реализации продукта**

Реализация разрабатываемого продукта будет производиться несколькими различными способами:

1. Система выдачи персонализированной полноэкранной рекламы *GoogleAds: GoogleMobileAdsSDK*, встроенное в приложение по умолчанию;

Инструментарий Google Ads дает возможность встраивания рекламных объявлений в приложение: баннерная, полноэкранная, видео-реклама и пр.

1. Интегрированная реклама от торговых сетей и марок. Поскольку логика разрабатываемого приложения предполагает посещение пользователями городских достопримечательностей, отдельные торговые компании могут быть заинтересованы в сотрудничестве с целью привлечения покупателей. Такое сотрудничество может принимать различные формы: *таргетированная* (с учетом местоположения пользователя) реклама сети, акции по привлечению пользователей приложения с перечислением бонусных баллов и пр.
2. Реклама и поддержка от государственных учреждений. Существует возможность спонсирования, поддержки существующего приложения из федеральных и муниципальных фондов, заинтересованных в привлечении населения к изучению объектов национального наследия, объектов культуры, исторических памятников.
3. Способы монетизации внутри приложения. Покупка внутренней валюты с целью: разблокировки специальных возможностей, особых персональных статусов, оплаты премиальной подписки и т.д.
   1. **Обзор и оценка конкурентной среды**

Понимание удовлетворяемых потребностей пользователя разрабатываемого продукта проистекает из определения задач, для которых эта система и предназначена:

* Знакомство с городом/поселением. В последние годы из-за экономического кризиса наблюдается падение потребности населения в туристических услугах [47], связанных с посещением иностранных государств. В этой связи происходит рост спроса на внутренний, и даже локальный туризм (прежде всего речь идет о гостиничных и санитарно-оздоровительных услугах). Приложение позволяет систематизировать и оптимизировать опыт знакомства потенциального пользователя с достопримечательностями конкретного города или поселения.
* Социальная. Разрабатываемая система призвана поощрять пользователей к формированию команд для совместной работы в приложении. Образование новых социальных контактов, основанных на непосредственном (то есть не удаленным) взаимодействии, – еще одна потребность, удовлетворяемая существующим продуктом.

Исходя из обозначенных потребностей, определим перечень конкурентных услуг и выделим существенные преимущества над ними:

* + - 1. Посещение музеев, тематических мероприятий.

Недостатки:

1. привязка к конкретным датам и времени;
2. отталкивающий (прежде всего для нового пользователя) престиж «сухого повествования и вовлечения»;
3. посещение некоторых мероприятий происходит на платной основе, что отталкивает новичков.
   * + 1. «Квесты» по франшизе под известным брендом. Начало становления этого рынка приходится на период с 2014 по 2015 гг., однако на данный момент он претерпевает сжатие и рецессию.

Недостатки:

1. в существующей тематике нет образовательной направленности, связанной с развитием компетенций и опыта;
2. мероприятие носит исключительно развлекательный характер;
3. одноразовость, отсутствие уникального социального взаимодействия.
   * + 1. Сетевые и партнерские «квесты», настольные игры.

Недостатки:

1. не позволяют обретать новые знакомства, замыкание пользователя в локальном коллективе;
2. отсутствует аспект изучения городской среды, физической активности.
   * + 1. Мобильные викторины типа «Мозгобойня» и «Квиз».

Недостатки:

1. перенасыщение рынка мобильных викторин, падение интереса;
2. неудовлетворение потребность в социальном взаимодействии, обретении новых знакомств.

Разрабатываемое приложение относится к типу так называемых «условно бесплатным» продуктов, что будет способствовать более непринужденному привлечению пользователя. Единственное, что от него требуется установить приложение и авторизоваться.

Качество услуг будет обеспечиваться пополнением и обслуживанием актуальных баз данных, содержащих информацию по каждому из знаковых городских мест. Гибкость разработки, настройки и распространения, а также относительная простота расширяемости систем удовлетворяет условиям как сжимающегося, так и расширяющегося рынка.

* 1. **Выводы об экономической целесообразности  
     разрабатываемого продукта**

Прежде всего, выделим некоторые неценовые детерминанты спроса на существующий продукт:

Количество потребителей. Количество активных пользователей непосредственно влияет на опыт пользования приложением, т.к. обеспечивается большая персонализация алгоритма случайного подбора партнеров.

Потребности потребителей, мода на определенные модели социального взаимодействия. Точно спрогнозировать влияние этого фактора невозможно, но, учитывая тенденцию на все большую изоляцию каждого отдельного городского индивидуума в условиях существующих информационной и психологической перегрузок, потребность в разрабатываемом продукте не сможет сильно сместиться еще долгое время.

2. Факторы, влияющие на способность пользователя свободного перемещения в городской среде (пеший ход и транспорт): пандемии, чрезвычайные происшествия, запрет на посещение того или иного места и пр. Такие события часто хоть и носят временный характер, но все же оказывают негативное влияние на распространение приложения. Как правило, сопровождаются перманентным уменьшением аудитории, отталкиванием активных пользователей.

С учетом относительно небольших затрат на разработку и первичный запуск системы (более крупные вложения могут осуществляться при масштабировании проекта при первой необходимости), и относительно высокого спроса на продукт подобного типа, коммерческий риск его производства можно оценить как «минимальный». В качестве доказательства произведем прогноз финансовых результатов от реализации продукта. Исходные данные:

По статистике от независимых разработчиков стоимость одной тысячи показов баннерной рекламы составляет около 0,3 долларов США [48]. Демонстрация полноэкранной, видео или нативной (встраиваемой) рекламы тысячу раз – 0,5 долларов.

Среднее количество показов переключаемой баннерной рекламы за сессию (выбор места, построение маршрута, тестирование) – 10 шт.

Среднее количество показов переключаемой полноэкранной рекламы (видео) за сессию – 3 шт. Нативная реклама в списке доступных достопримечательностей – 2 шт.

Возьмем число активных пользователей в день n = 1000 чел. Тогда:

* Количество показов рекламы за 0,3 доллара:
* Количество показов рекламы за 0,5 доллара:
* Средняя ежемесячная выручка от персонализированной рекламы составит:

Полученной суммы уже хватает на поддержание работы серверного оборудования. Однако несколько больше денег приносит система внутренних покупок. По статистике от компании Google [49] выручка компаний-разработчиков более чем на 60% состоит из платного наполнения внутри приложений. Учитывая это, а также другие способы монетизации, описанные выше, можно установить момент выхода на нулевую прибыль (за вычетом стоимости содержания сервера и обслуживающего персонала) в 1,5–2 тысячи пользователей.

**ЗАКЛЮЧЕНИЕ**

Целью выпускной квалификационной работы была разработка мобильного приложения на ОС Android для ознакомления с городской средой, разработка программного обеспечения для системы, поддерживающей работу приложения.

В результате проделанной работы были разработаны модули приложения, отвечающие за следующие функции:

1. аутентификацию, регистрацию новых пользователей с использованием технологий сервиса Firebase и разработанного сервера Apache Tomcat, регистрацию новых пользователей в приложении;
2. хранение информации, получение данных их удаленных сервисов при помощи асинхронных методов;
3. обработку, преобразование данных внутри приложения;
4. интерактивное отображение информации пользователю устройства.

Примеры работы системы представлены в Приложении П и Приложении Р.

Кроме того, был разработан пользовательский интерфейс для каждой из активностей, с которой взаимодействует пользователь по ходу использования приложения. Для этого были задействованы самые разнообразные виджеты и организаторы разметки пользовательского интерфейса: LinearLayout, TextView, EditText, Button, ImageView, Spinner, Fragment, ConstraintLayout и многие другие.

Также были спроектированы сервисы:

* сервер для обработки пользовательских запросов, обмена данными с устройством пользователя с использованием СУБД MySQL;
* автоматизированная система опроса пользователей приложения, исполненная на платформе мессенджера Telegram.

Полученный программный продукт является всего лишь прототипом системы «Аркантаун». Полноценная система, кроме всего прочего, может поддерживать следующие возможности:

1. Безопасный обмен данными с сервером Apache Tomcat по защищенному протоколу HTTPS (HTTP over SSL) напрямую, без использования Firebase.
2. Подтверждение адреса электронной почты пользователя.
3. Интеграция системы с известными социальными сетями. Импорт персональных данных из социальных сетей.
4. Обмен изображениями через сервер.
5. Автоматизированная система для внесения новых данных о городских достопримечательностях через приложение.
6. Чат с возможностью отправки аудио- и видео-сообщений.
7. Отдельные модели выбора и проведения викторин для одиночных, спаренных пользователях и групп пользователей.
8. Рейтинговая таблица для отдельных участников и именованных групп.
9. «Зал славы» пользователей приложения. Модуль для отображения истории взаимодействия пользователя с приложением.
10. Генерация оповещений при приближении к достопримечательности с приглашением пройти тест.
11. Интернационализация приложения (расширение содержимого строковых ресурсов приложения, определение системного языка пользователя).
12. Профессиональный дизайн пользовательского интерфейса с использованием комплексных, многосоставных видов.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. PC World. Android founder: We aimed to make a camera OS = Основатель Android: Мы намеревались создать операционную систему для фотокамер. – <https://www.pcworld.com/article/2034723/android-founder-we-aimed-to-make-a-camera-os.html>
2. Business Week Online. Google Buys Android for Its Mobile Arsenal = Google выкупает Android в свой мобильный арсенал. – <https://web.archive.org/web/20051004084620/http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm>
3. Global Stats - Statcounter. Mobile Operating System Market Share Worldwide = Доли рынка мобильных операционных систем по миру. – <https://gs.statcounter.com/os-market-share/mobile/worldwide>
4. Broadband search. Mobile Vs. Desktop Internet Usage (Latest 2020 Data) = Сравнение используемого трафика на мобильных устройствах и на персональных компьютерах. – <https://www.broadbandsearch.net/blog/mobile-desktop-internet-usage-statistics>
5. American Psychological Association. Social media's growing impact on our lives = Происходит рост влияния социальных сетей на наши жизни. – <https://www.apa.org/members/content/social-media-research>
6. Сайт Александра Климова. Activity (Активность, Деятельность). – <http://developer.alexanderklimov.ru/android/theory/activity-theory.php>
7. Android Developer. Activity = Активность. – <https://developer.android.com/reference/android/app/Activity>
8. Android Developer. Activity Lifecycle = Жизненный цикл активности. – <https://developer.android.com/reference/android/app/Activity#ActivityLifecycle>
9. Сайт Александра Климова. Стек активностей и Задания (Stack and Task). – <http://developer.alexanderklimov.ru/android/stack.php>
10. Android Developer. Widget = Виджет. – <https://developer.android.com/reference/android/widget/package-summary>
11. Android Developer. GridLayoutManager = Менеджер разметки типа «сетка». – <https://developer.android.com/reference/androidx/recyclerview/widget/GridLayoutManager.DefaultSpanSizeLookup>
12. Android Developer. LinearLayoutManager = Менеджер линейной разметки. – <https://developer.android.com/reference/androidx/recyclerview/widget/LinearLayoutManager>
13. Android Developer. RecyclerView.LayoutManager = Менеджер разметки для динамического управления элементами. – <https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.LayoutManager>
14. Android Developer. ConstraintLayout = Менеджер ограничительной разметки. – <https://developer.android.com/reference/android/support/constraint/ConstraintLayout>
15. King C., Ortiz E., Ableson F., Sen R. - Android in Action – 2012 – 632 с.
16. Mark L. Murphy - The Busy Coder's Guide to Android Development – 2019 – 4236 с.
17. Android Developer. InputStream = Класс входного потока. – <https://developer.android.com/reference/java/io/InputStream>
18. How-To Geek. How to Root Your Android Phone with *SuperSU* and *TWRP* = Как предоставить права суперпользователя вашему телефону Android с использованием утилит SupersSU и TWRP. – <https://www.howtogeek.com/115297/how-to-root-your-android-why-you-might-want-to>
19. Tutorials Point. Android - Shared Preferences = Android – Общие настройки. – <https://www.tutorialspoint.com/android/android_shared_preferences.htm>
20. Stack Exchange. When an app is uninstalled, is its SharedPreferences database cleared? = Удаляется ли файл общих настроек при удалении приложения? – <https://android.stackexchange.com/questions/213576/when-an-app-is-uninstalled-is-its-sharedpreferences-database-cleared>
21. IXBT. 2,5 миллиарда. Google рассказала, сколько сейчас в мире устройств с ОС Android. – <https://www.ixbt.com/news/2019/05/08/2-5-google-android.html>
22. Android Authority. A unified kernel won’t necessarily solve Android fragmentation = Унифицированное ядро не всегда решает проблему фрагментации Android. – <https://www.androidauthority.com/android-fragmentation-linux-kernel-1057450/>
23. Android – Google Source. Philosophy and Goals = Философия и цели [разработки в ОС Android]. – <https://android.googlesource.com/platform/docs/source.android.com/+/jb-dev/src/about/philosophy.md>
24. Bailey J., Dominguez A., Djermanovic D. - Saving Data on Android (1st Edition) – 2019 – 289 с.
25. IEEE. The importance of extensible database systems for e-commerce = Важность фактора масштабируемости в электронной коммерции. – <https://ieeexplore.ieee.org/document/914814>
26. Хабр. Горизонтальное масштабирование. Что, зачем, когда и как? – <https://habr.com/ru/company/oleg-bunin/blog/319526/>
27. Dataconomy. The 5 Best Reasons to Choose MySQL – and its 5 Biggest Challenges = 5 лучших причин для выбора MySQL и 5 больших препятствий. – <https://dataconomy.com/2017/04/5-reasons-challenges-mysql>
28. Apache Tomcat 7 Essentials - T. Khare (Packt, 2012) BBS – 272 с.
29. Vukotic A., Goodwill J. - Apache Tomcat 7 (The Expert’s Voice® in Java) – 2011 – 277 с.
30. Java World. What's new in Java Servlet API 2.2? = Что нового в программной оболочке Java Servlet 2.2? – <https://www.javaworld.com/article/2076518/what-s-new-in-java-servlet-api-2-2-.html>
31. Java World. What is JSP? Introduction to JavaServer Pages = Что такое JSP? Введение в страницы Java-сервера. – <https://www.javaworld.com/article/3336161/what-is-jsp-introduction-to-javaserver-pages.html>
32. Bailey J., Dominguez A., Djermanovic D. - Saving Data on Android (1st Edition) – 2019 – 289 c.
33. Google Cloud. Maps = Карты. – <https://cloud.google.com/maps-platform/maps>
34. EXlibris. Исследование аудитории Telegram 2019. – <https://exlibris.ru/news/issledovanie-auditorii-telegram-2019>
35. Telegram Core. Telegram Bot API = Программная оболочка бота Telegram. – <https://core.telegram.org/bots/api>
36. Android Developer. Adapter = Адаптер, соединитель. – <https://developer.android.com/reference/android/widget/Adapter>
37. Envato Tuts+. Android From Scratch: Understanding Adapters and Adapter Views = Android с нуля: понимание принципа работы адаптера и вида адаптера. – <https://code.tutsplus.com/tutorials/android-from-scratch-understanding-adapters-and-adapter-views--cms-26646>
38. Oracle Docs. Class PrintWriter = Класс PrintWriter. – <https://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html>
39. Хабр. Яндекс.Карты, 2ГИС или всё же Google Maps? – <https://habr.com/ru/post/242015/>
40. Shark Develop. Telegram bots what it is and how it works = Telegram боты. Что это и как оно работает. – <http://sharkdevelop.com/boty-v-telegram/>
41. Apptractor. Зарплаты мобильных разработчиков 2017: деньги, платформы, стаж и регионы. – <https://apptractor.ru/info/articles/zarplata-mobilnyih-razrabotchikov-2017-dengi-platformyi-stazh-i-regionyi.html>
42. VC.RU. Сколько зарабатывают маркетологи и PR-специалисты в России и за рубежом. – <https://vc.ru/marketing/8187-marketing-salaries>
43. Trud.com. Статистика зарплат Руководитель проекта в Санкт-Петербурге. – <https://sankt-peterburg.trud.com/salary/865/67664.html>
44. Яндекс Маркет. Смартфон Xiaomi Redmi Note 8T 4/64GB. – <https://market.yandex.ru/product--smartfon-xiaomi-redmi-note-8t-4-64gb/630030594>
45. Яндекс Маркет. Ноутбук ASUS VivoBook 15 X512. – <https://market.yandex.ru/product--noutbuk-asus-vivobook-15-x512/518204228>
46. Beget. [Стоимость аренды сервера Supermicro SYS−5039MS−H12TRF]. – <https://beget.com/ru/dedicated-servers>
47. Ac.gov.ru. Бюллетень о текущих тенденциях российской экономики. – <https://ac.gov.ru/files/publication/a/11063.pdf>
48. Office Life. Сколько стоит реклама в мобильных версиях и приложениях в 2018 году. – <https://officelife.media/article/money/how-much-does-advertising-in-mobile-versions-and-apps-in-2018/>
49. Office Life. Сколько стоит реклама в мобильных версиях и приложениях в 2018 году. – <https://officelife.media/article/money/how-much-does-advertising-in-mobile-versions-and-apps-in-2018/>

**ПРИЛОЖЕНИЕ А. Разметка активности MapActivity**

**<!-- XML-файл, отвечающий за разметку пользовательского интерфейса активности MapActivity -->**

**<!-- Шапка, определяющая ограничивающий шаблон элементов пользовательского интерфейса -->**

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_height="match\_parent"

android:layout\_width="match\_parent"

xmlns:app="http://schemas.android.com/apk/res-auto">

**<!-- Фрагмент, отвечающий за отрисовку окна карты -->**

**<!-- ширина и высота match\_parent - в данном случае на весь экран -->**

<fragment xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/map"

android:name="com.google.android.gms.maps.SupportMapFragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

tools:context=".MapActivity" />

**<!-- Вложенный ограничивающий шаблон. Отрисовка поля поиска вверху интерфейса -->**

**<!-- Включает в себя EditText (поле ввода) и ImageView (иконка поиска) -->**

<androidx.constraintlayout.widget.ConstraintLayout

android:id="@+id/search\_box"

android:layout\_width="match\_parent"

android:layout\_height="50dp"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintTop\_toTopOf="parent"

android:layout\_marginTop="10dp"

android:layout\_marginStart="10dp"

android:layout\_marginEnd="10dp"

android:background="@drawable/white\_border">

<EditText

android:id="@+id/input\_search"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:layout\_marginLeft="50dp"

android:layout\_marginStart="50dp"

android:textSize="15sp"

android:textColor="#000"

android:background="@null"

android:hint="Enter Address, City or Zip Code"

android:imeOptions="actionSearch"

android:importantForAutofill="no"

tools:ignore="MissingConstraints" />

<ImageView

android:id="@+id/ic\_search"

android:layout\_width="30dp"

android:layout\_height="30dp"

app:layout\_constraintBottom\_toBottomOf="parent"

app:layout\_constraintTop\_toTopOf="parent"

app:layout\_constraintLeft\_toLeftOf="parent"

android:layout\_marginLeft="10dp"

android:src="@drawable/ic\_search\_black\_24dp"

android:layout\_marginStart="10dp" />

</androidx.constraintlayout.widget.ConstraintLayout>

**<!-- Иконка центрирования карты на местоположении устройства -->**

<ImageView

android:id="@+id/ic\_gps"

android:layout\_width="40dp"

android:layout\_height="40dp"

app:layout\_constraintTop\_toBottomOf="@id/search\_box"

app:layout\_constraintEnd\_toEndOf="parent"

android:layout\_marginEnd="15dp"

android:layout\_marginRight="15dp"

android:layout\_marginTop="15dp"

android:scaleType="centerCrop"

android:src="@drawable/ic\_gps"/>

**<!-- Иконка для открытия Telegram -->**

<ImageView

android:id="@+id/ic\_telegram"

android:layout\_width="40dp"

android:layout\_height="40dp"

app:layout\_constraintTop\_toBottomOf="@id/ic\_gps"

app:layout\_constraintEnd\_toEndOf="parent"

android:layout\_marginEnd="15dp"

android:layout\_marginRight="15dp"

android:layout\_marginTop="15dp"

android:scaleType="centerCrop"

android:src="@drawable/ic\_contact\_mail\_blue\_24dp"/>

</androidx.constraintlayout.widget.ConstraintLayout>

**ПРИЛОЖЕНИЕ Б. Листинг метода для аутентификации  
пользователя**

**// В качестве аргументов методу передается логин,**

**// пароль и булевое значение registerIfNotRegistered**

**// registerIfNotRegistered == true - в случае отсутствия**

**// пользователя в системе происходит его регистрация**

private fun defaultLogin(registerIfNotRegistered: Boolean, email: String, password: String) {

Log.d(TAG, "Default login: " + email + " : " + password)

mAuth!!.signInWithEmailAndPassword(email, password)

.addOnCompleteListener(this, object : OnCompleteListener<AuthResult?> {

override fun onComplete(task: Task<AuthResult?>) {

if (task.isSuccessful) {

**// Аутентификация прошла успешно**

Log.d(TAG, "Login complete! ($email, $password)")

Toast.makeText(this@LoginActivity, "Login complete!", Toast.LENGTH\_SHORT).show()

**// Формирование и выхов асинхронной задачи для получения**

**// персональных данных с сервера**

GetAccountInfoAndPassToMain(this@LoginActivity,  
 registerIfNotRegistered, password, email)

.execute("account&email=" + email)

} else {

**// Аутентификация прошла неудачно**

if (registerIfNotRegistered) {

**// Аутентификация провалилась, тогда нужно провести регистрацию**

**// Данные передаются в активность RegisterActivity,**

**// где пройдет автоматическая регистрация**

val intent = Intent(this@LoginActivity,  
 RegisterActivity::class.java)

.putExtra("auth\_type", password)

.putExtra("email", email)

**// Запуск активность регистрации**

startActivity(intent)

}

**// В результате неудачной аутентификации пользователь**

**// получит оповещение о неправильно введенных данных**

Log.w(TAG,"signInWithEmail: Failure ($email, $password)", task.exception)

Toast.makeText(this@LoginActivity, "Authentication failed", Toast.LENGTH\_LONG).show()

}

}

})

}

**ПРИЛОЖЕНИЕ В. Листинг метода для обработки результатов работы активностей**

**// Метод обрабатывает результат работы активности по признаку requestCode**

**// т.е. каждый конкретный код запроса отвечает за отдельную активность**

override fun onActivityResult(requestCode: Int, resultCode: Int,  
 data: Intent?) {

super.onActivityResult(requestCode, resultCode, data)

when (requestCode) {

SPECIALITY\_REQUEST -> {

**// Активность AddPersonalInfoActivity была вызвана и  
 // завершила свою работу**

if (resultCode == Activity.RESULT\_OK) {

**// Завершила работу успешно**

val newUserPoints = manager.pointsFromShared.toInt() + 3

**// Обновление данных на сервере и Shared Memory (локальные данные)**

manager.updateAccount("points", newUserPoints.toString());

manager.updateAccount("bonus", "1");

**// Обновление интерфейса, отображающего число очков**

textPoints.text = newUserPoints.toString()

**// Скрыть кнопку и приглашающий текст после прохождения опроса**

hideInterest()

}

else if (resultCode == Activity.RESULT\_CANCELED) {

**// Завершилась неудачно**

Log.d(TAG, "SPECIALITY\_REQUEST: Fail")

Toast.makeText(this, "Error...", Toast.LENGTH\_LONG).show()

}

}

MAP\_AND\_QUIZ\_REQUEST -> {

**// Активность MapActivity была вызвана и завершила свою работу**

**// (фактически обрабатывается результат активности QuizActivity,**

**// которая вызывается сразу после MapActivity)**

if (resultCode == Activity.RESULT\_OK) {

**// Завершила работу успешно**

if (data != null) {

**// Получение данных о пройденном тесте из data завершенной активности**

val addPoints = data.getIntExtra("points", 0)

val placeID = data.getIntExtra("placeid", 0)

**// Обновление интерфейса**

textPoints.text = (manager.pointsFromShared.toInt() + addPoints).toString()

**// Обновление данных на сервере и локальном хранилище**

manager.updateQuizInAccount(placeID.toString(), addPoints.toString());

}

} else

**// Завершилась неудачно**

Toast.makeText(this, "Error while passing and synchronising test", Toast.LENGTH\_LONG).show()

}}}

**ПРИЛОЖЕНИЕ Г. Листинг ключевых методов активности MapActivity**

**// Инициализация интерфейса карты после успешной загрузки**

private fun layoutInterfaceInit() {

**// Установка перехватчиков нажатий на иконки**

mGpsIcon = findViewById(R.id.ic\_gps)

mGpsIcon.setOnClickListener {

getCurrentDeviceLocation()

}

val mTelegramBot = findViewById<ImageView>(R.id.ic\_telegram)

mTelegramBot.setOnClickListener {

val intent = Intent(Intent.ACTION\_VIEW, Uri.parse("https://telegram.me/local\_places\_bot"))

startActivity(intent)

}

}

**// Получение текущего местоположения устройства и центрирование**

private fun getCurrentDeviceLocation() {

Log.d(TAG, "getCurrentDeviceLocation: Getting location")

**// Получение объекта клиента сервиса Google Maps**

mFusedLocationProviderClient =  
LocationServices.getFusedLocationProviderClient(this)

try {

**// Получение последнего доступного местоположения клиента**

val location = mFusedLocationProviderClient.lastLocation

location.addOnCompleteListener(OnCompleteListener { task ->

if (task.isSuccessful) {

**// Успешное получение данных**

Log.d(TAG, "onComplete: Found location")

val currentLocation = task.result

**// Перемещение камеры на указанные координаты**

moveCamera(LatLng(currentLocation!!.latitude,  
currentLocation!!.longitude), DEFAULT\_ZOOM)

}

else {

Log.d(TAG, "onComplete: Couldn't find location")

}

})

}

catch (e: SecurityException) {

Log.d(TAG, "getCurrentDeviceLocation: SecurityException: " + e.message)

}

}

**// Метод перемещения карты на указанные координаты и установки нового маркера**

private fun moveCamera(latLng: LatLng, zoom: Float, pinning: Boolean = false, markerTitle : String = "Null") {

Log.d(TAG, "moveCamera: Moving camera to" + "${latLng.latitude}:${latLng.longitude} (${zoom})")

**// Перемещение камеры по координатам с укаазнным приближением**

mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng, zoom))

**// Создание маркера**

if (pinning) {

val options = MarkerOptions()

.position(latLng)

.title(markerTitle)

mMap.addMarker(options)

}

}

**// Метод обработки результатов получения разрешений**

override fun onRequestPermissionsResult( requestCode: Int,  
permissions: Array<out String>, grantResults: IntArray) {

when(requestCode) {

LOCATION\_PERMISSIONS\_REQUEST\_CODE -> {

for (result in grantResults) {

**// Циклическая проверка всех необходимых разрешений**

if (result != PackageManager.PERMISSION\_GRANTED) {

**// Если хотя бы одно разрешение не было получено...**

mLocationPermissionsGranted = false

Log.d(TAG, "onRequestPermissionsResult: wasn't granted")

**// происходит завершение работы активности**

finish()

}

}

mLocationPermissionsGranted = true

Log.d(TAG, "onRequestPermissionsResult: was granted")

}

}

}

**// Метод обработки результатов викторины, которая пройдет**

**// после выбора места проведения в этой активности**

override fun onActivityResult(requestCode: Int, resultCode: Int,  
data: Intent?) {

super.onActivityResult(requestCode, resultCode, data)

when(requestCode) {

QUIZ\_REQUEST\_CODE -> when(resultCode) {

Activity.RESULT\_OK -> {

**// Если тестирование прошло успешно, то ее результаты передаются**

**// в главное меню без изменения**

this.setResult(Activity.RESULT\_OK, data)

finish()

}

}

}

}

**// Метод для алгоритма, вызываемого после подготовки карты**

override fun onMapReady(googleMap: GoogleMap) {

Toast.makeText(this, "BAM!", Toast.LENGTH\_SHORT).show()

mMap = googleMap

**// Проверка разрешений на геолокацию устройства**

mMap.uiSettings.isCompassEnabled = true

mMap.uiSettings.isZoomControlsEnabled = true

if (mLocationPermissionsGranted) {

getCurrentDeviceLocation()

mMap.isMyLocationEnabled = true

mMap.uiSettings.isMyLocationButtonEnabled = false

}

**// Установка перехватчика окна с вводом поискового запроса**

findViewById<EditText>(R.id.input\_search).setOnEditorActionListener { v, actionId, event ->

Log.d(TAG, "input\_search: Edit Action")

if (actionId == EditorInfo.IME\_ACTION\_SEARCH

|| actionId == EditorInfo.IME\_ACTION\_DONE

|| event.action == KeyEvent.ACTION\_DOWN

|| event.action == KeyEvent.KEYCODE\_ENTER) {

Log.d(TAG, "input\_search: Required action was triggered")

v.text = v.text.toString().replaceFirst("\n", "")

**// Если все разрешения выданы, произвести поиск**

geoLocate()

}

Log.d(TAG, "input\_search: Required action wasn't triggered")

false

}

**// Выставить все маркеры городских достопримечательностей,**

**// полученные из сервера**

for (p: Place in placesInTown) {

mMap.addMarker(MarkerOptions().position(p.location).title(p.title).snippet(p.description))

mMap.moveCamera(CameraUpdateFactory.newLatLng(p.location))

}

**// Установка перехватчика при нажатии на маркер**

**// (выбор места для проведения викторины)**

mMap.setOnInfoWindowClickListener(object:  
GoogleMap.OnInfoWindowClickListener {

override fun onInfoWindowClick(p0: Marker?) {

Log.d(TAG, "Marker info window clock: " + p0?.title)

var placeID = 0

**// Нахождение выбранного места в списке**

for (p: Place in placesInTown) {

placeID = p.id

if (p.title.equals(p0?.title))

break

}

**// Запуск асинхронного алгоритма, результатом которого  
 // будет переход к активности викторины**

GetQuizFromServer(applicationContext, this@MapActivity, QUIZ\_REQUEST\_CODE).execute("quiz&placeid=" + placeID)

}

})

}

**// GetQuizFromServer наследует класс ReceiveDataFromMySQLTask,  
// асинхронно отправляющий запрос на сервер**

**// и ожидающий ответа от него в виде всех данных о викторине**

private class GetQuizFromServer(context: Context, activity: Activity,  
requestCode: Int): ReceiveDataFromMySQLTask() {

private val mContext = context

private val mActivity = activity

private val mRequestCode = requestCode

override fun onPostExecute(result: String?) {

super.onPostExecute(result)

**// Запуск активности для прохождения викторины**

val intent = Intent(mContext, QuizActivity::class.java)

intent.putExtra("quiz", result)

mActivity.startActivityForResult(intent, mRequestCode)

}

}

**ПРИЛОЖЕНИЕ Д. Листинг метода-слушателя объекта типа Button**

**// Перехватчик, реагирующий на нажатие кнопки "Следующий вопрос"**

findViewById<Button>(R.id.next\_button).setOnClickListener(object: View.OnClickListener {

**// Переопределение метода нажатия на элемент**

override fun onClick(v: View?) {

Log.d(TAG, "Clicked: current=" + quiz.current + "correct="+correctAnswer)

if (listAdapter.popLastClickedPosition() == correctAnswer)

**// Если последняя нажатая кнопка ответа является правильным**

**// ответом на текущий вопрос,**

**// то в массив результатов заносятся полученные баллы**

results[quiz.current] = POINTS\_FOR\_ANSWER

else

**// Неудача**

results[quiz.current] = 0

**// Обновление пользовательского интерфейса:**

**// замена текста вопроса, создания нового списка кнопок с ответами**

updateQuestion(quiz.next)

}

})

**ПРИЛОЖЕНИЕ Е. Листинг ключевых методов викторины**

**// Метод обновления интерфейса викторины**

**// question - объект вопроса Question для установки**

private fun updateQuestion(question: Question?) {

if (question == null)

**// Список вопросов подошел к концу**

**// завершение викторины**

quizIsDone()

else {

**// Обновление текстового поля с вопросом**

questionText.text = question.question

correctAnswer = question.correctAnswerIndex

**// Очистка списка кнопок с доступными ответами**

buttonList.clear()

**// Циклическое добавление новых ответов в адаптер**

for (answer: Answer in question.answers)

buttonList.add(answer.answer)

Log.d(TAG, "NOB: ${buttonList.size}")

**// Обновление списка кнопок (а точнее его адаптера)**

listAdapter.notifyDataSetChanged()

}

}

**// Метод завершения викторины**

private fun quizIsDone() {

**// Суммирование результирующих баллов**

for (i: Int in results)

points += i

**// Занесение данных о полученных очках и идентификаторе места проведения**

intent = Intent()

.putExtra("points", points)

.putExtra("placeid", quiz.place.id)

Log.d(TAG, "Quiz is done: $points : " + results.toString())

Toast.makeText(this, "Done! [$points]", Toast.LENGTH\_SHORT).show()

**// Установка результата активности и завершение ее работы**

setResult(Activity.RESULT\_OK, intent)

finish()

}

**ПРИЛОЖЕНИЕ Ж. Листинг модели QuizData**

**// Класс предназначен для хранения и получения данных о викторине:**

**// место проведения, список вопросов с ответами и пр.**

public class QuizData {

private final static String TAG = "QuizData";

private int \_id;

private Place place;

private ArrayList<Question> questions;

public Place getPlace() {

return place;

}

public ArrayList<Question> getQuestions() {

return questions;

}

**// Конструктор объекта с использованием json-данных**

public QuizData(String jsonObject) {

\_id = -1;

questions = new ArrayList<>();

try {

**// Получение объекта "quiz" : {"place" : {...}, "questions" : [...] }**

JSONObject json = new JSONObject(jsonObject);

**// Генерация объекта из JSON "place" : {...}**

place = new Place(json.getJSONObject("place").toString());

Log.d("QuizData", "Place created: " + place.present());

**// Получение массива "questions" : [...]**

JSONArray arr = json.getJSONArray("questions");

**// Циклическая генерация объектов типа Question и**

**// занесение их в массив вопросов викторины**

for (int i = 0; i < arr.length(); i++)

questions.add(new Question(

arr.getJSONObject(i)));

Log.d(TAG, "Success!");

} catch (JSONException e) {

e.printStackTrace();

Log.d(TAG, "JSONException: ERROR while creating JSON object");

}

}

**// Возврат следующего вопроса в списке**

public Question getNext() {

if (questions.size() <= 0)

return null;

else if (\_id + 1 >= questions.size())

return null;

else if (\_id == -1)

\_id = 0;

else

\_id++;

return this.questions.get(\_id);

}

**// Возврат предыдущего вопроса в списке**

public Question getPrevious() {

if (questions.size() <= 0)

return null;

else if (\_id == -1)

\_id = 0;

else if (\_id - 1 < 0)

return null;

else

\_id--;

return this.questions.get(\_id);

}

public int getCurrent() {

return \_id;

}

}

**ПРИЛОЖЕНИЕ И. Листинг метода адаптера, генерирующего вид пользовательского интерфейса**

**// Метод адаптера для получения вида каждой конкретной кнопки**

**// из списка кнопок для ее отображения в группе**

@NonNull @Override

public View getView(final int position, @Nullable View convertView, @NonNull ViewGroup parent) {

ViewHolder mainViewHolder = null;

**// Вид не был сформирован**

if (convertView == null) {

**// Создание и сохранение модуля вида LayoutInflater**

LayoutInflater inflater = LayoutInflater.from(getContext());

convertView = inflater.inflate(layout, parent, false);

**// Заполнение данных о создаваемой кнопке:**

**// позиция, текст, цвет по умолчанию, перехватчик**

ViewHolder viewHolder = new ViewHolder();

viewHolder.button = (Button) convertView.findViewById(R.id.\_quiz\_button);

viewHolder.button.setText(getItem(position));

viewHolder.button.setTextColor(mContext.getResources()  
 .getColor(R.color.grey\_neutral));

viewHolder.button.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

lastClickedPosition = position;

if (lastClickedView != null)

**// При нажатии на кнопку цвет предыдущей меняется на цвет по умолчанию**

lastClickedView.setTextColor(mContext.getResources()

.getColor(R.color.grey\_neutral));

**// Цвет текста нажатой кнопки становится зеленым**

lastClickedView = (Button) v;

lastClickedView.setTextColor(mContext.getResources()  
 .getColor(R.color.green\_not\_alert));

}

});

**// Установка метки данных о кнопке**

**// связывание данных и представления**

convertView.setTag(viewHolder);

}

**// Вид был сформирован раньше, содержимое не изменилось**

else {

mainViewHolder = (ViewHolder) convertView.getTag();

mainViewHolder.button.setText(getItem(position));

}

return convertView;

}

**ПРИЛОЖЕНИЕ К. Листинг метода  
ReceiveDataFromMySQLTask**

**// Класс наследует интерфейс AsyncTask, в котором уже определен механизм**

**// исполнения асинхронных алгоритмов. Для его использования обычно**

**// переопределяют методы:**

**//1. doInBackground - действия, исполняемые в отдельном потоке (асинхронно)**

**//2. onPreExecute - действия, исполняемые до начала работы асинхр. алгоритма**

**//3. onPostExecute - действия, исполняемые по завершению асинхр. алгоритма**

**// Данный класс переопределяет только метод асинхронного выполнения –**

**// остальные переопределяются в активностях, использующих этот класс,**

**// т.к. результирующие действия в каждом конкретном случае разные**

public abstract class ReceiveDataFromMySQLTask extends AsyncTask<String, String, String> {

@Override

**// servletCommands - аргументы функции execute() класса**

protected String doInBackground(String... servletCommands) {

**// адрес сервлета сервера**

String SERVLET\_URL = "http://192.168.1.33:8088/DJWA\_war/t";

String CLASS\_TAG = "ReceiveDataFromMySQL";

HttpURLConnection connection = null;

BufferedReader reader = null;

**// Установка запроса при помощи URL-меток**

StringBuilder url = new StringBuilder(SERVLET\_URL + "?want=" +  
servletCommands[0]);

try {

**// Установка дополнительных аргументов HTTP-запроса**

for (int i = 1; i < servletCommands.length; i++)

url.append("&").append(servletCommands[i]);

**// Установка соединения с сервером**

**// вместе с отсыланием указанного запроса**

URL link = new URL(url.toString());

connection = (HttpURLConnection) link.openConnection();

connection.setConnectTimeout(5000);

connection.setReadTimeout(5000);

connection.connect();

**// Получение ответа от сервера**

InputStream stream = connection.getInputStream();

reader = new BufferedReader(new InputStreamReader(stream));

StringBuilder result = new StringBuilder();

String readerLine;

**// Считывание ответа сервера**

while((readerLine = reader.readLine()) != null) {

result.append(readerLine);

Log.d(CLASS\_TAG, "Line read from buffer: " + readerLine);

}

**// Возврат ответа**

**// Результат может быть использован в методе onPostExecute**

return result.toString();

} catch (MalformedURLException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

} finally {

\_close\_reader\_and\_disconnect(connection, reader);

}

return null;

}

**// Закрытие соединения и объекта-читателя**

private void \_close\_reader\_and\_disconnect(HttpURLConnection cn,  
BufferedReader rd) {

if (cn != null)

cn.disconnect();

if (rd != null) {

try {

rd.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

**ПРИЛОЖЕНИЕ Л. Листинг методов сервлета для работы с  
базой данных**

**// Функция для получения данных об учетной записи**

**// пользователя с электронной почтой email**

public static String getAccountInfoInSQL(String email) {

lastQuery = "SELECT \* FROM accounts WHERE email='" + email + "';";

return fetch(lastQuery, ACCOUNT\_MODE);

}

**// Функция для соединения с БД и получения из нее наборов данных**

**// в соответствии с запросов SQLQuery**

private static String fetch(String SQLQuery, int fetchingMode) {

Connection connection;

String result = "false";

try {

private static final String databaseSimplifiedURL = "jdbc:mysql://localhost:3306/arcantowndb?serverTimezone=UTC";

**// Подключение к базе данных от имени администратора**

connection = DriverManager.getConnection(databaseSimplifiedURL, "root", "qwerty123");

if (!connection.isClosed()) {

**// Подключение выполнено успешно**

System.out.println(DATABASE\_MANAGER\_TAG +  
 "Fetch: Connection established");

System.out.println(DATABASE\_MANAGER\_TAG + "Fetch: Query to execute: " + SQLQuery);

**// Получение набора данных с помощью запроса**

Statement statement = connection.createStatement();

ResultSet data = statement.executeQuery(SQLQuery);

**// Выбор обработчика набора данных в соответствии с выбранным режимом**

if (fetchingMode == SPECS\_MODE)

result = \_get\_one\_in\_json(data, "specs");

else if (fetchingMode == COUNTRY\_MODE)

result = \_get\_one\_in\_json(data, "countries");

else if (fetchingMode == TOWN\_MODE)

result = \_get\_one\_in\_json(data, "towns");

else if (fetchingMode == PLACES\_MODE)

result = \_get\_places\_in\_json(data);

else if (fetchingMode == ACCOUNT\_MODE)

**// Вычленение строки с информацией о пользователе**

result = \_get\_account\_in\_json(data);

else if (fetchingMode == SINGLE\_PLACE\_MODE)

result = \_get\_one\_place\_in\_json\_BALD(data);

else if (fetchingMode == QUESTIONS\_MODE)

result = \_get\_questions\_in\_json\_BALD(data);

else if (fetchingMode == ANSWERS\_MODE)

result = \_get\_answers\_json\_BALD(data);

connection.close();

System.out.println(DATABASE\_MANAGER\_TAG + "Fetch: Connection was closed as planned");

} else

System.out.println(DATABASE\_MANAGER\_TAG + "Fetch: Error: Connection was not established");

} catch (SQLException throwables) {

throwables.printStackTrace();

System.out.println(DATABASE\_MANAGER\_TAG + "Fetch: Error while fetching");

}

return result;

}

**// Метод для создания JSON-объекта из набора данных, полученного из БД**

public static String \_get\_account\_in\_json(ResultSet set) throws SQLException {

StringBuilder result = new StringBuilder();

**// Возврат "false" в случае пустого набора (вероятно, аккаунт не найден)**

if (!set.next())

return "false";

**// Генерация JSON-объекта "account" = {...}**

result.append("{").append("\"account\" : {");

result.append("\"id\": ").append(set.getInt(1)).append(",");

result.append("\"auth\_type\": \"").append(set.getString(2)).append("\",");

result.append("\"email\": \"").append(set.getString(3)).append("\",");

result.append("\"login\": \"").append(set.getString(4)).append("\",");

result.append("\"name\": \"").append(set.getString(5)).append("\",");

result.append("\"points\": ").append(set.getInt(6)).append(",");

result.append("\"country\": \"").append(set.getString(7)).append("\",");

result.append("\"town\": \"").append(set.getString(8)).append("\",");

result.append("\"completed\": \"").append(set.getString(9)).append("\",");

result.append("\"bonus\": ").append(set.getInt(10));

result.append("}}");

System.out.println(DATABASE\_MANAGER\_TAG + "Account formed: " + result);

return result.toString();

}

**ПРИЛОЖЕНИЕ М. Листинг метода-перехватчика пользовательского запроса**

**# Функция асинхронного перехвата пользовательского запроса (сообщения)**

**# Предназначена для получения персональных данных, представленных в**

**# соответствии с некоторым регулярным выражением**

**# Метка обработчика пользовательских запросов**

**# regexp - регулярное выражение**

**# Form.info - состояние, при котором перехватчик может работать**

**# Т.к. Form.info - инициирующее состояние автомата,**

**# этот перехватчик не может работать во время опроса пользователя**

**# (только в начале общения)**

@dp.message\_handler(regexp=REGEX\_AUTH\_CHECKER, state=Form.info)

async def auth(message: types.Message):

if not storage.data['is\_auth\_requested']:

**# Если аутентификация не требуется (т.е. переменная в хранилище,**

**# отвечающая за статус пользователя, указывает на то, что тот уже**

**# отослал персональные данные модератора), бот выводит сообщение**

**# об ошибке**

print("auth regex handler: not requested")

reply = get\_phrase\_from\_res(JsonTags.AUTH\_NOT\_REQUESTED,\

message.from\_user.language\_code)

else:

storage.data['is\_auth\_requested'] = False

print("auth regex handler: requested")

**# Проверка на подлинность введенных данных**

**# Функция login\_moderator проверяет сообщение на содержимое**

**# если в нем содержатся корректные данные, происходит аутентификация**

if login\_moderator(message.text, message.from\_user.id):

print("auth regex handler: login\_moderator is True")

**# Пользователь становится модератором**

storage.data['is\_moderator'] = True

reply = get\_phrase\_from\_res(JsonTags.AUTH\_SUCCESS\_MODERATOR,\  
 message.from\_user.language\_code)

else:

**# Ошибка. Неверные данные. Пользователь остается пользователем**

reply = get\_phrase\_from\_res(JsonTags.AUTH\_FAIL\_MODERATOR, \  
 message.from\_user.language\_code)

return SendMessage(message.chat.id, reply)

**ПРИЛОЖЕНИЕ Н. Пример содержимого текстового файла для локализации**

{

"mod\_commands\_available" : ["/start", "/info", "/help", "/logout", "/add", "/fetch\_one", " : ", "/cancel", "/commit\_answer", "/commit\_question"],

"user\_commands\_available" : ["/start", "/auth", "/info", "/help", "/add", " : ", "/cancel", "/commit\_answer", "/commit\_question"],

"phrases" : {

"info" : {

"ru" : "Бот предназначен для внесения заявок на добавление новых достопримечательностей и викторин в базу данных приложения \"Аркантаун\".\n.\nПринцип его работы заключается в приеме (в диалоговой форме) заявок на добавление новой записи в базу данных (расширяемый текстовый файл на сервере держателя) в виде опроса. Пользователь с особыми правами (модератор) может, в свою очередь, также удаленно получать эти записи для обработки и добавлением корректно оформленной записи в базу данных приложения.\n.\nПриложение является составной частью курсового проекта студента СПбГЭТУ \"ЛЭТИ\" Буракаева Д.А. (гр. 6305)",

"de" : "Der Bot ist so konzipiert, dass er eine Anfrage zum Hinzufügen neuer Orte und Tests zur Datenbank der mobilen Anwendung \"Arkantown\" bildet.\n.\nDas Prinzip seiner Arbeit besteht darin, (in einem Dialogformular) Anfragen zum Hinzufügen eines neuen Datensatzes zu akzeptieren die Datenbank (erweiterbare Textdatei auf dem Server des Inhabers) in Form einer Umfrage. Ein Benutzer mit besonderen Rechten (Moderator) kann diese Datensätze aus der Ferne empfangen, um sie zu verarbeiten und zur Anwendungsdatenbank hinzuzufügen.\n.\nDie Anwendung ist ein wesentlicher Bestandteil des Studentenprojekts von St. Staatliche Elektrotechnische Universität Petersburg \"LETI\" (Burakaev Daniiar, 6305)",

"en" : "The bot is designed to form request for adding new places and quizzes to the database of mobile application \"Arkantown\".\n.\nThe principle of its work is to accept (in a dialogue form) requests for adding a new record to the database (expandable text file on the holder's server) in the form of a survey. A user with special rights (moderator) may remotely receive these records for processing and adding them to the application database.\n.\nThe application is an integral part of the student project of St. Petersburg State Electrotechnical University \"LETI\" (Burakaev Daniiar, 6305)"

},

"user\_help" : {

"ru" : "Доступные пользователю команды:\n/help - вывод текущего окна\n/info - вывод текста с информацией об этом боте\n/start - начать работу с ботом\n/auth - провести аутентификацию как модератор\n/add - предложить запись в базу данных",

"de" : "Die Befehle, die dem Benutzer zur Verfügung stehen:\n/help - Dieses Fenster anzeigen\n/info - Text mit Informationen zu diesem Bot anzeigen\n/start - Mit dem Bot arbeiten\n/auth - Als Moderator authentifizieren\n/add - eine Anfrage anbieten",

"en" : "The commands available for the user:\n/help - display this window\n/info - display text with information about this bot\n/start - start working with the bot\n/auth - authenticate as a moderator\n/add - offer an request"

},

"mod\_help" : {

"ru" : "Доступные модератору команды:\n/help - вывод текущего окна\n/info - вывод текста с информацией об этом боте\n/start - начать работу с ботом\n/logout - прекратить работу с ботом от имени модератора\n/add - предложить запись в базу данных\n/fetch\_one - вывести одну запись из базы данных с заявками",

"de" : "Die Befehle, die dem Moderator zur Verfügung stehen:\n/help - Dieses Fenster anzeigen\n/info - Text mit Informationen zu diesem Bot anzeigen\n/start - Mit dem Bot arbeiten\n/logout - Mit dem Bot als Moderator nicht mehr arbeiten\n/add - eine Anfrage anbieten\n/fetch\_one - zeigt einen Eintrag aus der Datenbank an",

"en" : "The commands available for the moderator:\n/help - display this window\n/info - display text with information about this bot\n/start - start working with the bot\n/logout - stop working with the bot as a moderator\n/add - offer an request\n/fetch\_one - display one entry from the database"

}}

**ПРИЛОЖЕНИЕ П. Регистрация в системе**

На рисунке 10 представлено меню для аутентификации существующего пользователя.

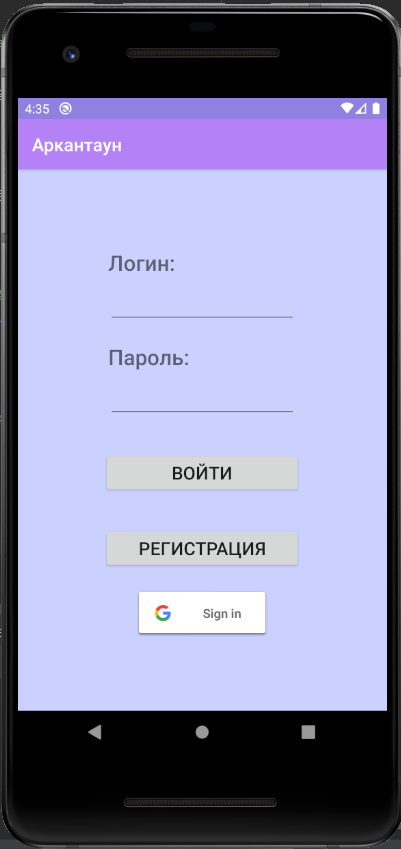


Рисунок 10 – Графический интерфейс активности LoginActivity

На рисунке 11 представлено меню для регистрации нового пользователя в системе.

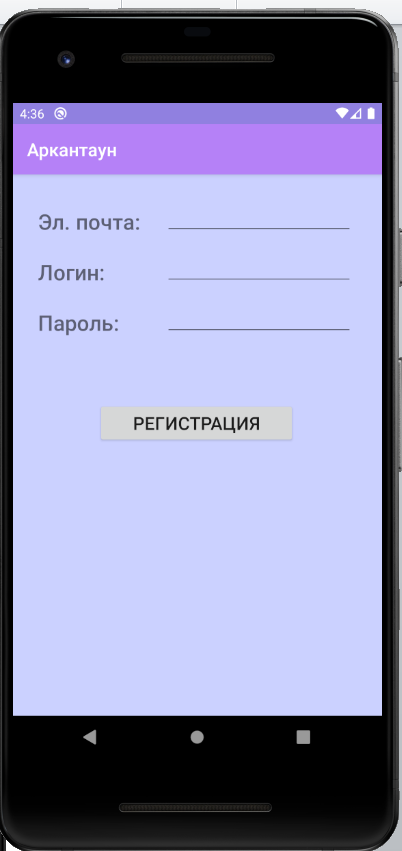


Рисунок 11 – Графический интерфейс активности RegisterActivity

На рисунке 12 представлена попытка ввода данных для создания нового аккаунта в системе.

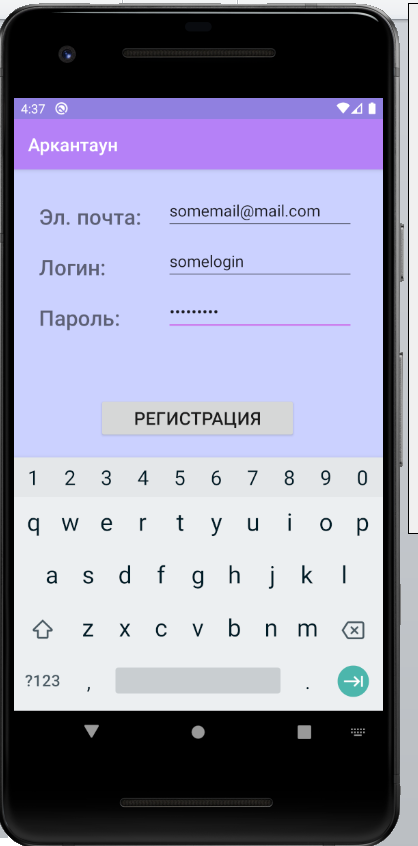


Рисунок 12 – Ввод персональных данных нового пользователя

На рисунке 13 представлено состояние базы данных пользователей до регистрации нового аккаунта.

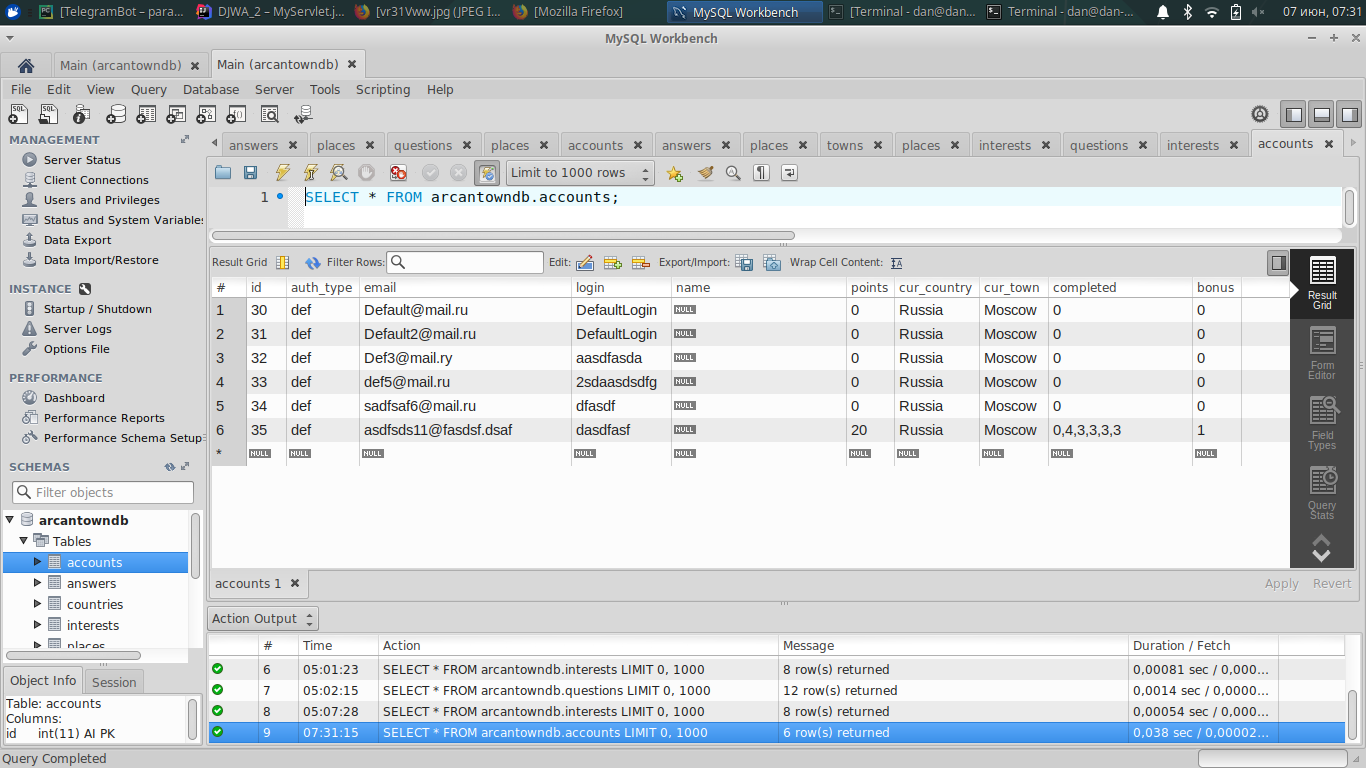


Рисунок 13 – Содержимое таблицы с данными пользователей системы

На рисунке 14 представлено состояние базы данных пользователей после регистрации нового аккаунта.

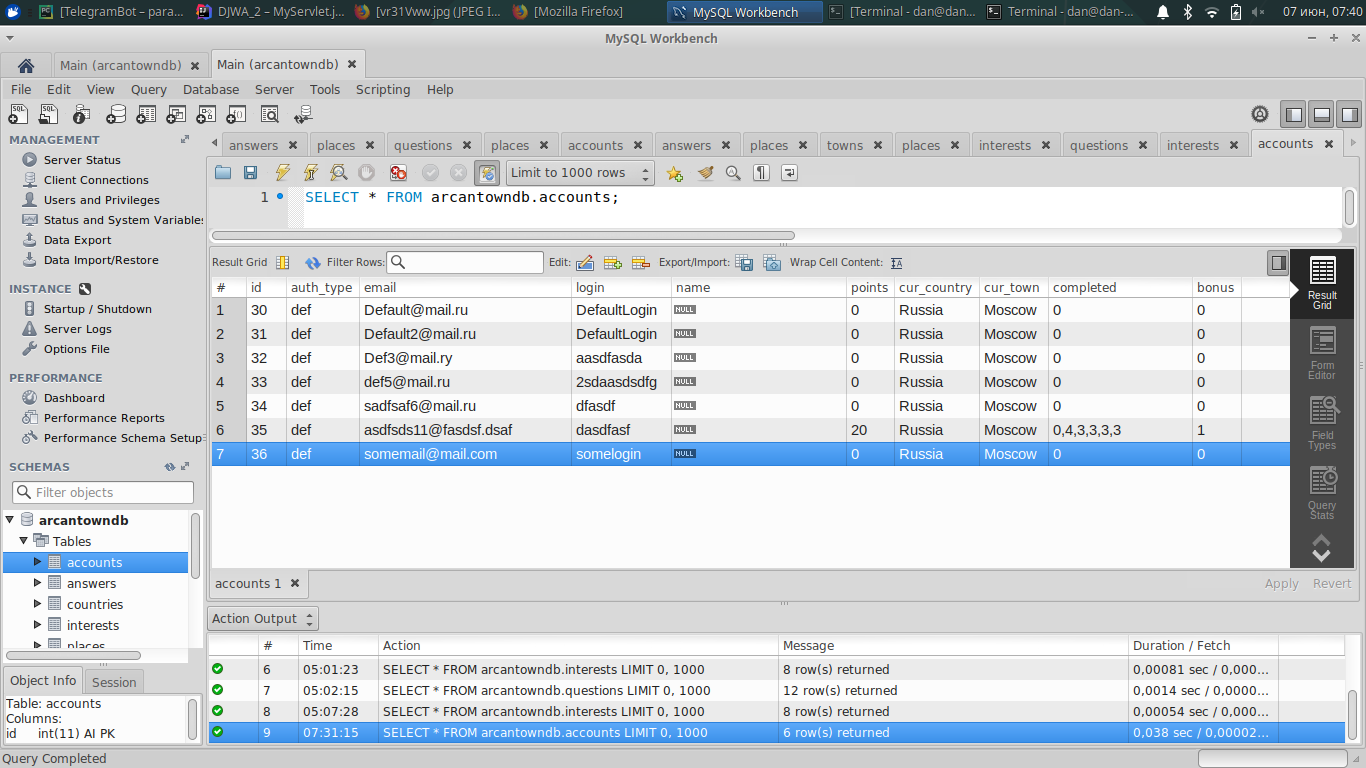


Рисунок 14 – Содержимое таблицы с данными пользователей системы после регистрации

На рисунке 15 представлено состояние менеджера пользователей в Firebase Concole после регистрации нового аккаунта.

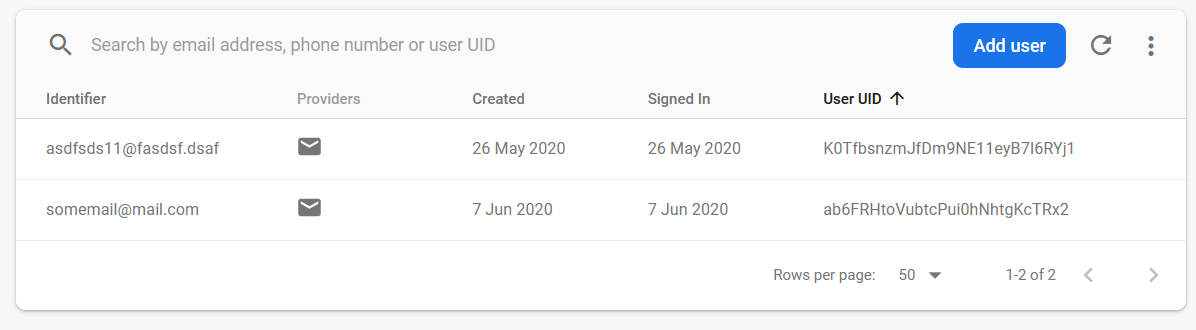


Рисунок 15 – Появление новой учетной записи в системе

На рисунке 16 представлена попытка входа в систему с использованием данных учетной записи.

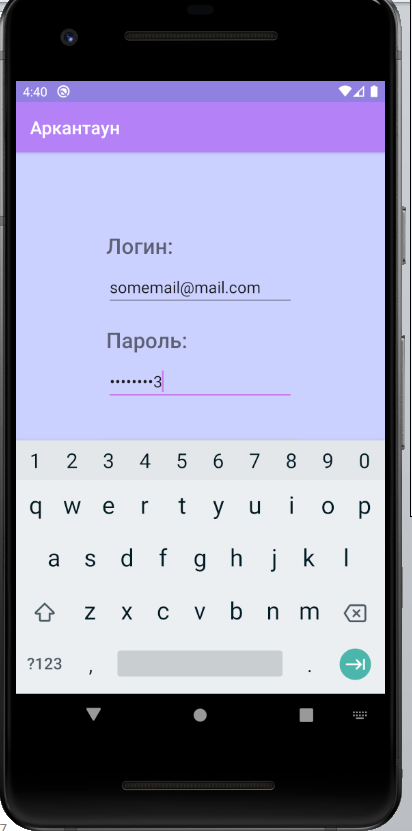


Рисунок 16 – Ввод персональных данных пользователя

На рисунке 17 представлен результат входа в систему.

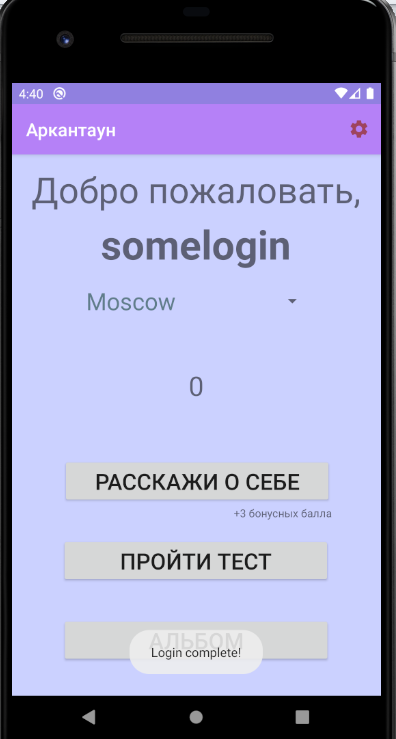


Рисунок 17 – Успешный вход в систему

Как видно из результатов взаимодействия, приложение успешно производит регистрацию новой учетной записи в системе, синхронизируя данные с удаленным сервером и сервисом Firebase.

**ПРИЛОЖЕНИЕ Р. Прохождение викторины**

На рисунке 18 представлено окно с выбором места прохождения викторины.

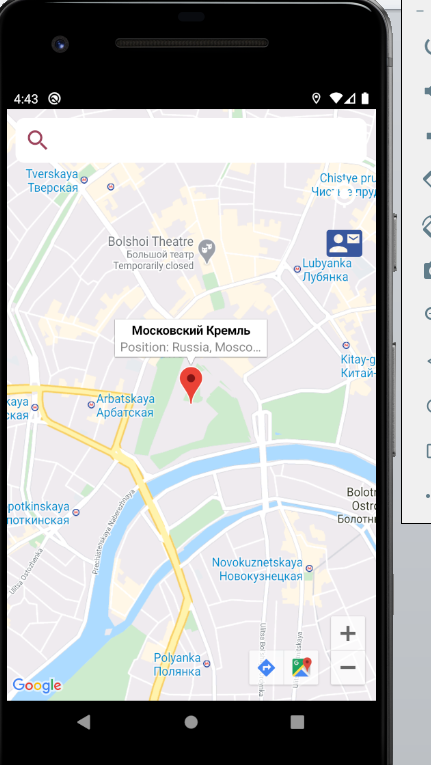


Рисунок 18 – Карта Google Maps с автоматически сформированной меткой

На рисунке 19 представлено окно с вопросом.

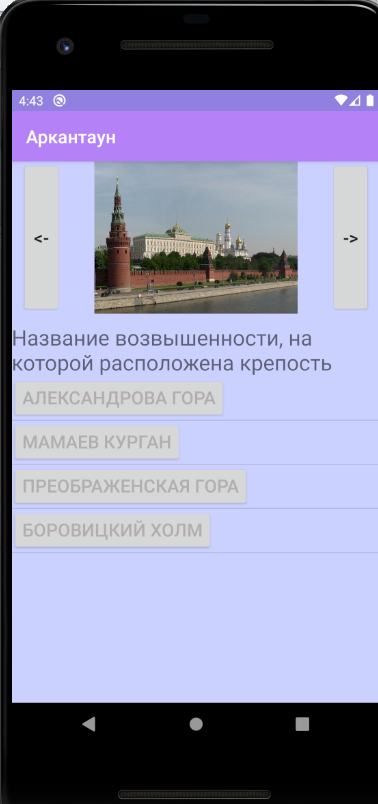


Рисунок 19 – Интерфейс викторины

На рисунке 20 продемонстрирован результат нажатия на одну из кнопок с ответом.

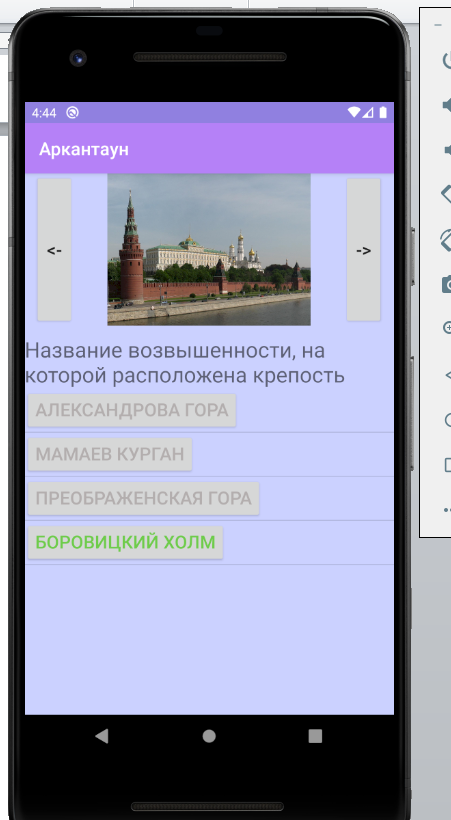


Рисунок 20 – Окраска выбранного варианта ответа

На рисунке 21 продемонстрирован результат прохождения всего теста с несколькими правильными ответами.

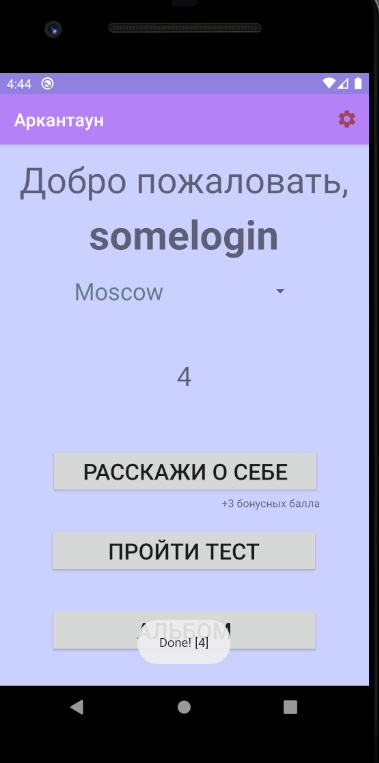


Рисунок 21 – Завершение теста и обновление баллов на главном экране

На рисунке 22 продемонстрирован результат прохождения всего теста с несколькими правильными ответами в базе данных пользователей системы.

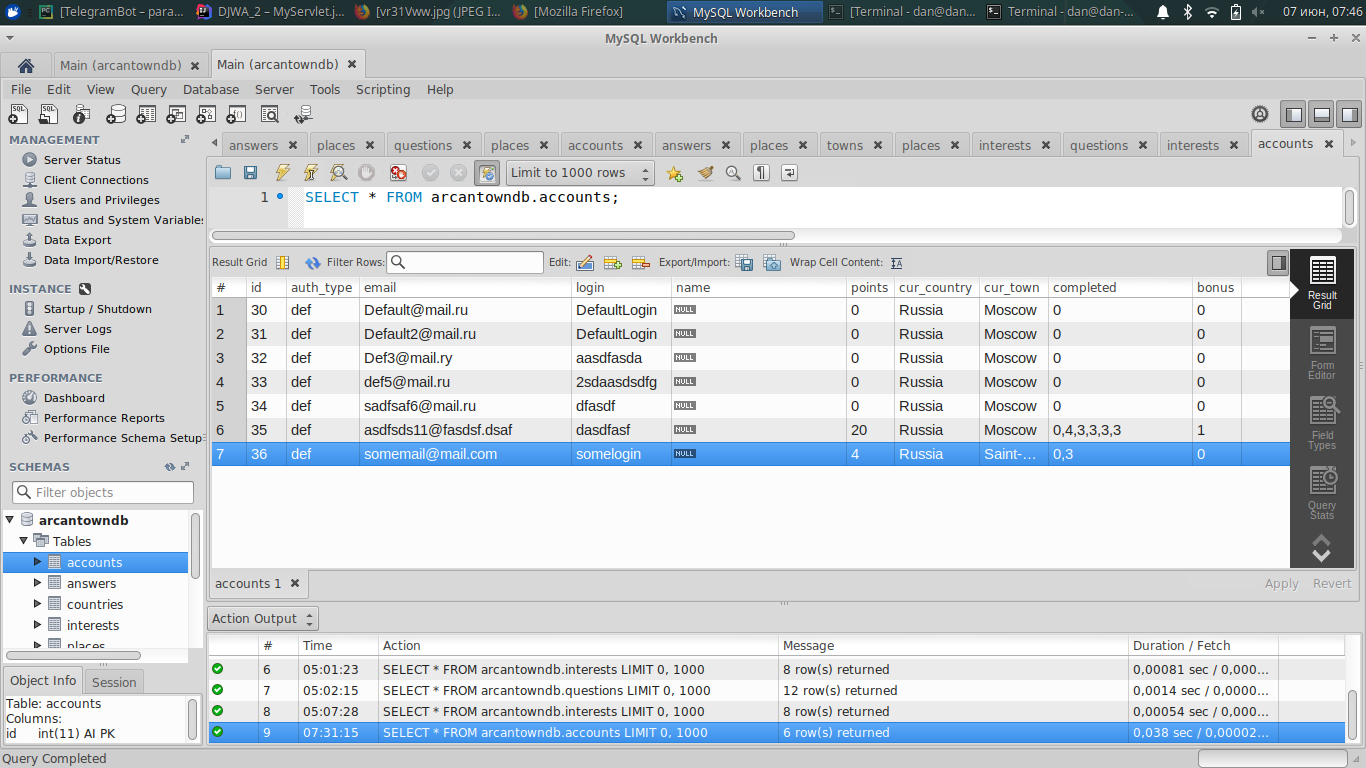


Рисунок 22 – Результат обновления данных пользователя после прохождения викторины

Как видно из скриншота, число баллов увеличилось с 0 до 4. На рисунке 23 продемонстрирована попытка смены города пребывания пользователя.

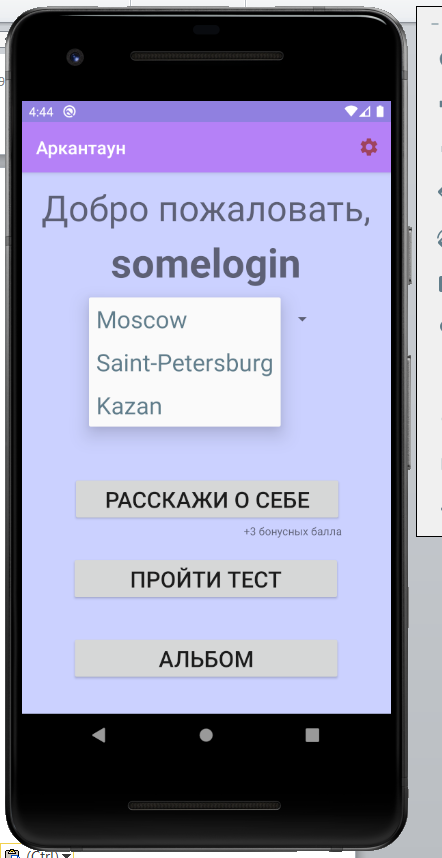


Рисунок 23 – Развернутый список городов текущей страны пребывания

На рисунке 24 продемонстрирована попытка прохождения викторины в новом городе.

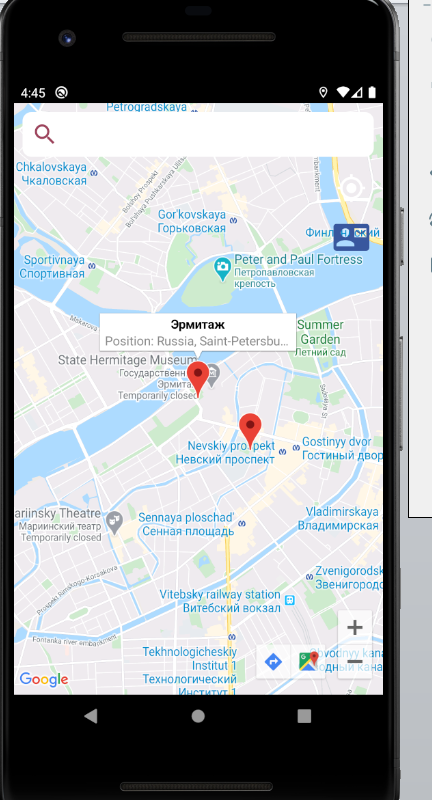


Рисунок 24 – Карта с новыми метками

Как видно из результатов взаимодействия, приложение успешно производит тестирование и смену города, синхронизируя данные с удаленным сервером и получая новые координаты меток.