|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/05 Современные интеллектуальные**

**программно-аппаратные комплексы.**

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОМУ ПРОЕКТУ***

***НА ТЕМУ:***

**Разработка клиент-приложения для NoSQL базы данных «Доставка»**

Студент ИУ6-21М\_\_\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Марчук И.С.**\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Фомин М.М.\_\_

(Подпись, дата) (И.О.Фамилия)

*2024 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_\_\_\_\_\_\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение курсового проекта**

по дисциплине «Распределенные базы данных»

Студент группы ИУ6-12М

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Марчук Иван сергеевич \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Тема курсового проекта

«Разработка клиент-приложения для NoSQL базы данных «Доставка»

Направленность КП (учебный, исследовательский, практический, производственный, др.)

исследовательский

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения проекта: 25% к 2 нед., 50% к 10 нед., 75% к 13 нед., 100% к 16 нед.

***Задание***

Разработать на платформе Android клиент-приложение для NoSQL базы данных. В качестве языка программирования взять Java. Архитектуру использовать MVVM. Запросы реализовать в асинхронном режиме. Предусмотреть регистрацию новых пользователей, создание маршрутов и добавление посылок.

***Оформление курсового проекта:***

Расчетно-пояснительная записка на \_66\_ листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.):

1. Название темы КП, задание.

2. Результаты разработки интерфейса.

3. Примеры кода, реализующего доступ к удаленной базе данных.

Дата выдачи задания « 10 » февраля 2024 г.

**Руководитель курсового проекта**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_Фомин М.М.\_\_\_

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Марчук И.С.**\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

**РЕФЕРАТ**

РПЗ 37 страниц, 26 рисунков, 1 таблица, 10 источников, 4 приложения

ANDROID, БАЗА ДАННЫХ, MONGODB,

Целью курсового проекта является закрепление и углубление знаний, приобретенных в процессе обучения по курсу “Распределенные Базы данных”, а также получение практических навыков разработки и модификации клиентов реальных баз данных и информационных систем

В ходе выполнения курсового проекта были выполнены следующие задачи:

* установлены необходимые программы: VirtualBox с Ubuntu, Elasticsearch, Neo4j, Hadoop и Spark;
* сгенерированы JSON файлы с необходимыми документами;
* проиндексированы документы, а также написан анализатор с маппингом и несколькими запросами;
* создана графовая базы данных и написан запрос к ней;
* сформированы csv-файлы, написан запрос к ним и с помощью мониторинга в Spark проанализирован процесс выполнения запроса.

**СОДЕРЖАНИЕ**

[**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 5**](#_Toc167586130)

[**ВВЕДЕНИЕ 6**](#_Toc167586131)

[**1. Обзор проекта 7**](#_Toc167586132)

[**1.1 Содействие одноранговой доставке посылок через связь с путешественниками 7**](#_Toc167586133)

[**1.2 План работы 7**](#_Toc167586134)

[**2. Проектирование системы 8**](#_Toc167586135)

[**2.1 Бизнес-процессы 8**](#_Toc167586136)

[**2.2 Диаграмма классов данных передаваемых между базой данных и приложением 11**](#_Toc167586137)

[**3. Android приложение 13**](#_Toc167586138)

[**3.1 Обзор технологий и инструментов 13**](#_Toc167586139)

[**3.1.1 Android SDK 13**](#_Toc167586140)

[**3.1.2 Язык программирования 13**](#_Toc167586141)

[**3.1.3 MongoDB и Atlas 13**](#_Toc167586142)

[**3.1.4 Связь с базой данных 14**](#_Toc167586143)

[**3.1.5 Material Design 3 14**](#_Toc167586144)

[**3.1.6 Асинхронная работа с данными 14**](#_Toc167586145)

[**3.2 Проектирование и разработка приложения 15**](#_Toc167586146)

[**3.2.1 Архитектура приложения 15**](#_Toc167586147)

[**3.2.2 Компоненты системы 15**](#_Toc167586148)

[**4.2.3 Модели данных 16**](#_Toc167586149)

[**3.2.4 Создание пользовательского интерфейса 19**](#_Toc167586150)

[**3.2.5 Навигация 23**](#_Toc167586151)

[**4.2.6 Работа с базой данных, подключение 25**](#_Toc167586152)

[**3.2.7 Работа с базой данных, Чтение и запись 26**](#_Toc167586153)

[**ЗАКЛЮЧЕНИЕ 29**](#_Toc167586154)

# ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

БД – база данных

MongoDB –

Realm –

СУБД – система управления базой данных

MVVM –

ViewModel –

Activity –

**Fragment** –

SQL (Structured Query Language) – структурированный язык запросов

XML –

NavigationGraph –

Android –

# ВВЕДЕНИЕ

Жизнь в другой стране часто сопровождается множеством трудностей, особенно для иностранных студентов, которые находятся вдали от своих семей и друзей. Привыкая к новой культуре и языковой среде, многие из нас испытывают тоску по дому и желание поделиться частичкой своей новой жизни с близкими. Однако расстояние и геополитические трудности, которые часто приводят к сбоям в работе международных служб доставки, усложняют процесс отправки и получения посылок.

Вдохновленный этим вызовом, я, совместно со своим другом, решил создать Android-приложение, которое поможет иностранным студентам, находящимся в России, найти людей, готовых взять на себя передачу посылок и открыток. Это приложение призвано стать мостом между студентами и их семьями, облегчая обмен вещами и поддерживая культурные связи.

Основная идея нашего проекта заключается в создании удобной платформы, на которой пользователи могут без проблем находить путешественников с пустым местом в багаже. Это не только решает практическую проблему доставки, но и способствует налаживанию новых знакомств и культурному обмену.

В этом отчете будет подробно рассмотрен процесс разработки Android-приложения, начиная с анализа требований и проектирования, и заканчивая реализацией. Мы обсудим используемые технологии и инструменты, такие как Android SDK 34, Java 8, архитектура MVVM, Material Design 3, библиотека Realm для работы с MongoDB, размещенной на сервере Atlas, и средства работы с ней в android приложении. Также будут описаны основные функциональные возможности приложения и способы их реализации.

# 

# 1. Обзор проекта

## 1.1 Содействие одноранговой доставке посылок через связь с путешественниками

Приложение упрощает два основных типа операций:

**Пост путешественника:**

Когда пользователь планирует отправиться в определенный пункт назначения, скажем, в город X, он может создать пост с указанием своих планов поездки.

Это пост служит объявлением для других пользователей, которым могут понадобиться товары, доставленные в город X.

Если есть посты от пользователей, которым требуется доставка посылок в город X, путешественник может связаться с ними, чтобы договориться о доставке посылки во время поездки.

**Запрос на доставку посылки:**

Если пользователю необходимо отправить посылку в город X, он может выполнить поиск по существующим постам от путешественников, планирующих отправиться туда.

В качестве альтернативы пользователь может создать новый пост, указав, что ему нужна доставка посылки в город X.

Путешественники, которые увидят эти посты, могут предложить доставить посылку во время поездки.

По сути, приложение выступает в качестве платформы для связи путешественников с людьми, которым требуется доставка товаров в определенные пункты назначения, что способствует созданию одноранговой сети доставки.

## 1.2 План работы

Учитывая масштабность нашего приложения, мы решили сотрудничать с моим одногрупником *А.Ш.Джабри*, чтобы оптимизировать наши усилия в рамках проекта. Вместе мы стремимся реализовать как можно больше основных функций в начальной версии, сохранив ее простой, но функциональной. По мере продвижения мы планируем расширять и совершенствовать приложение на основе ваших ценных отзывов и идей. Этот отчет, посвященный концептуализации и разработке клиентской части, закладывает основу для дальнейшего развития нашего проекта.

# 2. Проектирование системы

После изучения контекста нашего проекта и определения поставленных целей мы приступаем к анализу и проектированию нашей системы. Фазы анализа и проектирования в информационном проекте являются неотъемлемыми этапами, позволяющими прийти к практичному, согласованному и полному решению, соответствующему потребностям пользователей.

## 2.1 Бизнес-процессы

В этом разделе мы рассмотрим тонкости наших бизнес-процессов, чтобы охватить поток действий, взаимодействий и данных в нашей системе. Визуализируя эти процессы, мы можем определить области для оптимизации, упростить рабочие процессы и обеспечить эффективность нашего решения.

Благодаря тщательному анализу и продуманному дизайну мы заложим основу для разработки нашего приложения, гарантируя его полное соответствие требованиям и ожиданиям наших заинтересованных сторон. Теперь давайте перейдем к изучению бизнес-процессов с помощью подробных диаграмм.

Начиная со схемы бизнес-процесса регистрации:

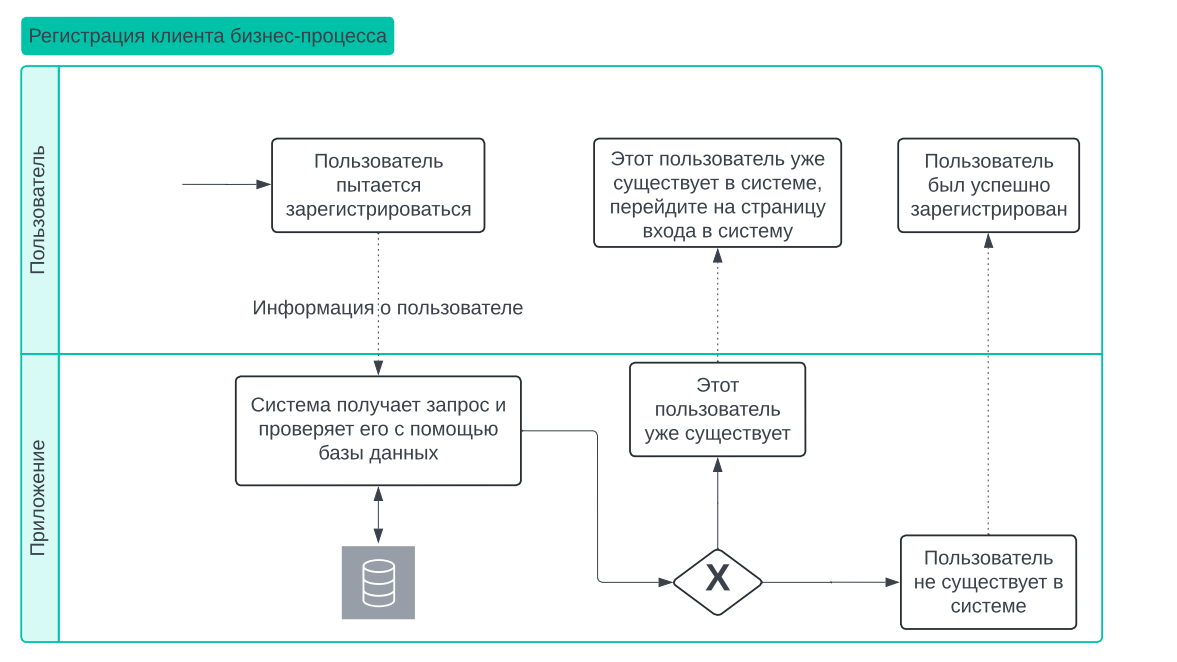


Рисунок 1 **–** Схема бизнес-процесса «Регистрация пользователя».

Вторая схема бизнес-процесса — это схема входа в систему, где каждый пользователь должен иметь учетную запись и входить в нее на случай, если он захочет что-то опубликовать.

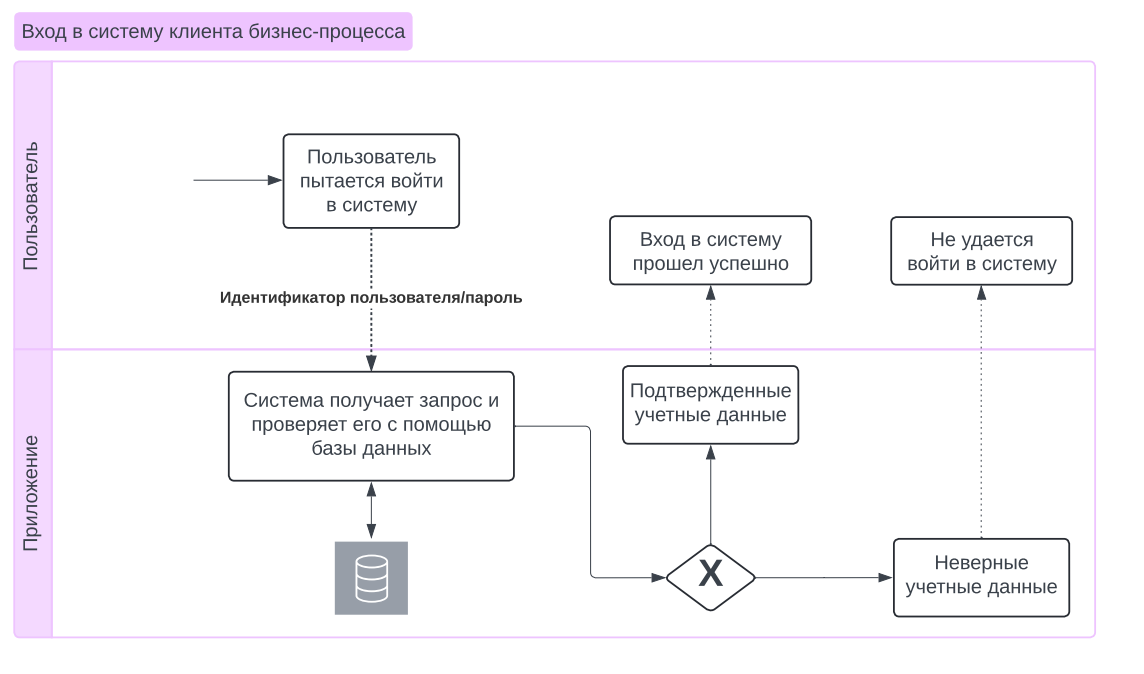


Рисунок 2 **–** Схема бизнес-процесса «Вход в систему».

Третья схема бизнес-процесса выглядит следующим образом: когда пользователь (путешественник) собирается куда-то в поездку и и у него есть место чтобы взять с собой посылку, или когда пользователю нужно что-то куда-то отправить и он ищет кого-то, кто туда направляется.

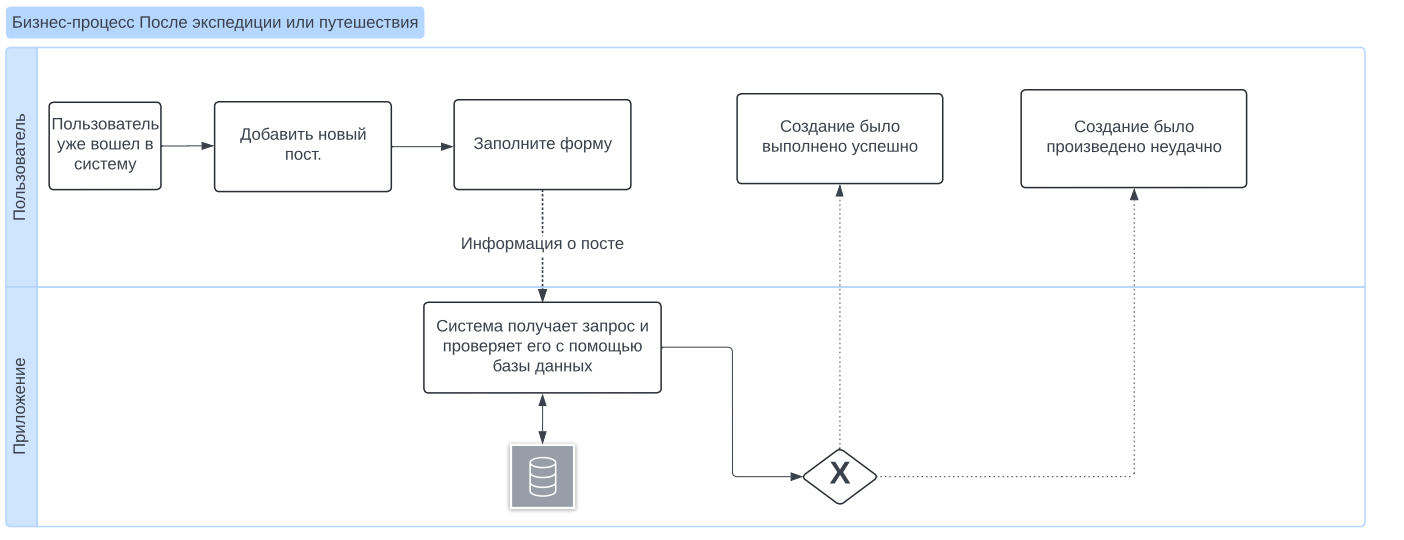
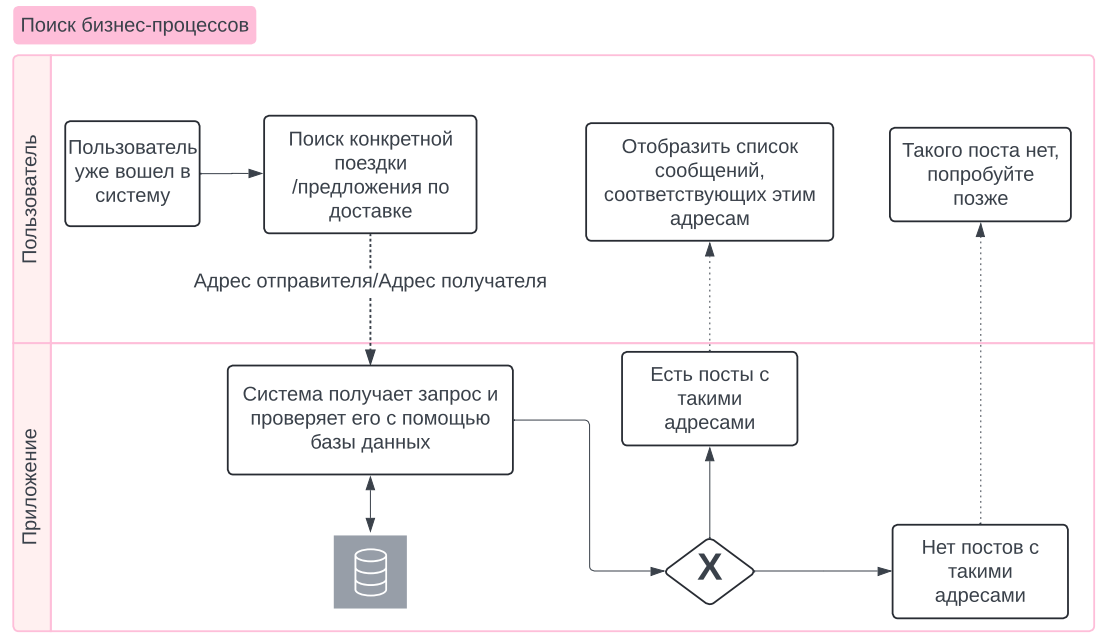


Рисунок 3 **–** Схема бизнес-процесса «новый маршрут».

Четвертая схема бизнес-процесса — это когда пользователь ищет конкретную запись, и в основном это происходит путем фильтрации адресов с места на место.



**Рисунок 4 –** Схема бизнес-процесса «Поиск».

Пятый и последний бизнес-процесс — это редактирование существующей записи.

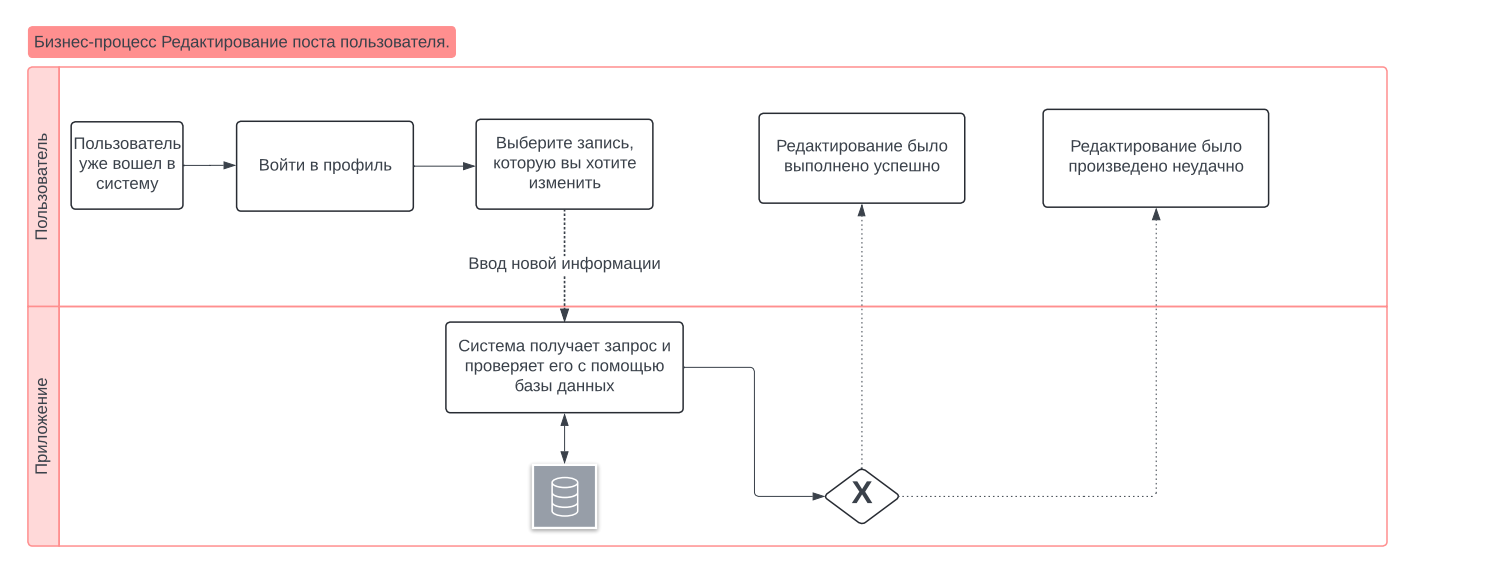


Рисунок 5 **–** Схема бизнес-процесса «Редактировать запись».

## 2.2 Диаграмма классов данных передаваемых между базой данных и приложением

Диаграмма классов — это схема, используемая для выражения статической структуры системы в термах классов и отношений между этими классами, класс характеризуется:

* Название класса
* Атрибут
* Метод

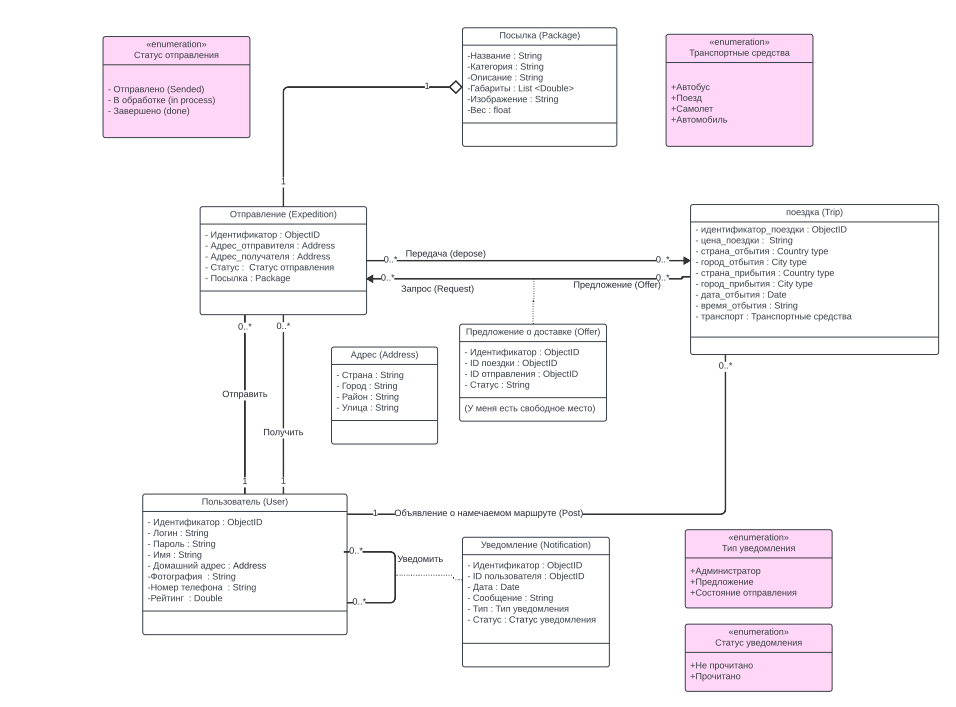


Рисунок 6 **–** Диаграмма классов базы данных.

# 3. Android приложение

## 3.1 Обзор технологий и инструментов

В этом разделе рассмотрим используемые технологии и инструменты, которые были задействованы при разработке Android приложения для взаимодействия с MongoDB, размещенной на сервере Atlas.

### **3.1.1 Android SDK**

Для разработки приложения была использована платформа Android SDK версии 34. Android SDK предоставляет набор инструментов и библиотек, необходимых для создания приложений под операционную систему Android. В SDK включены компиляторы, инструменты для отладки, эмуляторы и множество библиотек, упрощающих разработку.

### **3.1.2 Язык программирования**

Приложение было разработано с использованием языка программирования Java 8. Java является одним из наиболее распространенных языков для разработки под Android, предоставляя разработчикам мощные инструменты для создания надежных и масштабируемых приложений. И хотя сейчас большую популярность набирает язык Kotlin я выбрал именно язык Java, потому как мог быстро и каственно писать на нем код и обладал большим опытом по написанию программ на нем по сравнению с Kotlin.

### **3.1.3 MongoDB и Atlas**

MongoDB – это документоориентированная база данных NoSQL, которая предоставляет гибкую структуру данных, высокую производительность и масштабируемость. Atlas – это облачная платформа MongoDB, которая позволяет легко развертывать, управлять и масштабировать базы данных MongoDB без необходимости заботиться о инфраструктуре. Все данные в приложении хранятся в базе данных MongoDB на сервере Atlas, что обеспечивает доступность и безопасность данных.

### **3.1.4 Связь с базой данных**

Для взаимодействия с MongoDB была использована библиотека Realm. Realm – это мобильная база данных, которая предоставляет простой и интуитивный API для работы с данными. В проекте была использована библиотека io.realm версии 10.10.1. Realm позволяет выполнять все операции с базой данных в асинхронном режиме, что обеспечивает отзывчивость пользовательского интерфейса и предотвращает блокировку главного потока.

### **3.1.5 Material Design 3**

Для создания современного и интуитивно понятного пользовательского интерфейса была использована библиотека Material Design 3. Material Design – это язык дизайна, разработанный Google, который предоставляет набор инструментов и компонентов для создания привлекательных и удобных интерфейсов. Использование Material Design 3 позволяет приложению соответствовать современным стандартам дизайна и ожиданиям пользователей.

### **3.1.6** **Асинхронная работа с данными**

Все запросы к MongoDB выполнялись в асинхронном режиме, что обеспечивало своевременное обновление данных в приложении без необходимости хранить локальные копии базы данных на устройстве. Асинхронная работа позволяет поддерживать актуальное состояние данных, отражающее текущее состояние базы данных в реальном времени.

Использование асинхронных операций также улучшает производительность приложения, так как длительные сетевые запросы выполняются в фоновом режиме, не блокируя основной поток и не снижая отзывчивость пользовательского интерфейса.

Таким образом, использование указанных технологий и инструментов позволило создать эффективное, производительное и удобное в использовании Android приложение, обеспечивающее взаимодействие с базой данных MongoDB на сервере Atlas.

## 3.2 Проектирование и разработка приложения

### **3.2.1 Архитектура приложения**

В качестве архитектурного паттерна была выбрана MVVM + Single activity. Такая архитектура, помимо того, что считается самой актуальной и желательной при разработке приложений, позволяет:

* Сократить количество межмодульных вызовов, и использование памяти, так как для работы приложения используется только одна Activity. И при переключении экранов приложения просто подменяется разметка, а не перестраиваются большие куски приложения;
* Необходимо отслеживать жизненный цикл только одной Activity, и все инициализируются в ней;
* Создается класс ViewModel, который хранит все данные приложения и работает с моделью, более того обычно такие события как переворот экрана или смена цветовой темы вызывает перестройку всей Activity, однако ViewModel лишена этих проблем так как существует отдельно от жизненного цикла Activity;

### **3.2.2 Компоненты системы**

Приложение было разделено на следующие компоненты:

* MainActivity, главная страница интерфейса, служит для его отрисовки и навигации между фрагментами;
* MainViewModel, находится в памяти пока приложение работает.
* RegisterFragment, экран авторизации пользователя
* LoginFragment, экран регистрации пользователя
* UserFragment, экран профиля пользователя, с возможностью просмотра информации о пользователе и просмотра входящих уведомлений
* UserEditDialogFragment, диалог редактирования данных пользователя;
* ExpeditionsFragment, экран отображения отправленных пользователем посылок;
* ExpeditionEditDialogFragment, диалог создания и редактирования не отправленных посылок;
* TrajectoriesFragment, экран на котором можно посмотреть путешествия других пользователей, сделать заявку на отправку своей посылки и написать о своем планирующемся путешествии.

### **4.2.3 Модели данных**

Для хранения и обработки загруженных из Mongo DB данных я использовал Java классы по структуре схожие с таблицами в базе данных.

Листинг полей класса DBUser:

public class DBUser implements Serializable {

public static final String TABLE\_NAME = "User";

public static final String USER\_ID = "\_id";

public static final String USER\_PASSWORD = "Password";

public static final String USER\_ADDRESS = "Address";

public static final String USER\_EMAIL = "Email";

public static final String USER\_NAME = "Name";

public static final String USER\_PHONE\_NUMBER = "Phone number";

public static final String USER\_PICTURE = "Pictue";

public static final String USER\_RATING = "Rating";

private ObjectId \_id;

private String password;

private DBAddress address;

private String email;

private String name;

private String phoneNumber;

private String picture;

private Double rating;

…

}

Листинг полей класса DBTrip:

public class DBTrip {

public static final String TABLE\_NAME = "Trip";

public static final String TRIP\_ID = "\_id";

public static final String TRIP\_PRICE = "Price";

public static final String TRIP\_RECEIVING\_CITY = "Reciving city";

public static final String TRIP\_RECEIVING\_COUNTRY = "Reciving country";

public static final String TRIP\_SEND\_CITY = "Send city";

public static final String TRIP\_SEND\_COUNTRY = "Send country";

public static final String TRIP\_SEND\_DATE = "Sent date";

public static final String TRIP\_TRANSPORT = "Transport mean";

private final ObjectId \_id;

private final String price;

private final String receivingCity;

private final String receivingCountry;

private final String sendCity;

private final String sendCountry;

private final Date sentDate;

private final String transport;

…

}

Листинг полей класса DBPackage:

public class DBPackage {

public static final String TABLE\_NAME = "Pckage";

public static final String PACKAGE\_CATEGORY = "Category";

public static final String PACKAGE\_DESCRIPTION = "Description";

public static final String PACKAGE\_DIMENSIONS\_ARRAY = "Dimension";

public static final String PACKAGE\_WEIGHT = "Weight";

public static final String PACKAGE\_NAME = "Name";

public static final String PACKAGE\_PICTURE = "Pictures";

private final String category;

private final String description;

private final double[] dimensions;

private final double weight;

private final String name;

private final String picture;

…

}

Листинг полей класса DBNotification:

public class DBNotification {

public static final String TABLE\_NAME = "Notification";

public static final String NOTIFICATION\_ID = "\_id";

public static final String NOTIFICATION\_USER = "Id\_User";

public static final String NOTIFICATION\_DATE\_TIME = "Date";

public static final String NOTIFICATION\_MESSAGE = "Message";

public static final String NOTIFICATION\_TYPE = "Type";

public static final String NOTIFICATION\_STATUS = "Status";

public static final String NOTIFICATION\_TYPE\_VALUE\_ADMIN = "Admin";

public static final String NOTIFICATION\_TYPE\_VALUE\_SEND\_REQUEST = "Request";

public static final String NOTIFICATION\_TYPE\_VALUE\_EXPEDITION = "Expedition";

public static final String NOTIFICATION\_STATUS\_UNREAD = "Unread";

public static final String NOTIFICATION\_STATUS\_BEEN\_READ = "BeenRead";

private ObjectId \_id;

private ObjectId userId;

private Date date;

private String notificationMessage;

private String notificationType;

private String notificationStatus;

…

}

Листинг полей класса DBExpedition:

public class DBExpedition implements Serializable {

public static final String TABLE\_NAME = "Expedition";

public static final String EXPEDITION\_ID = "\_id";

public static final String EXPEDITION\_ADDRESS\_RECEIVER = "Address\_reciver";

public static final String EXPEDITION\_ADDRESS\_SENDER = "Address\_sender";

public static final String EXPEDITION\_STATUS = "Status";

public static final String EXPEDITION\_SENDER = "Sender";

public static final String EXPEDITION\_PACKAGE = "Package";

public static final String EXPEDITION\_STATUS\_VALUE\_WAIT\_SEND = "Wait";

public static final String EXPEDITION\_STATUS\_VALUE\_SENT = "Sent";

public static final String EXPEDITION\_STATUS\_VALUE\_DONE = "Done";

private ObjectId \_id;

private DBAddress dbAddressReceiver;

private DBAddress dbAddressSender;

private String status;

private ObjectId sender;

private DBPackage dbPackage;

…

}

Листинг полей класса DBAddress:

public class DBAddress {

public static final String[] ADDRESS\_LOCALS = {

"country",

"city",

"district",

"street"

};

private final String[] address;

…

}

Также у каждого из этих классов обязательно были прописаны методы для преобразования в обе стороны в класс Document. Так как данные полученные из Realm передаются в программу именно в виде этих классов. Для примера приведу методы преобразования класса DBUser.

Листинг методов класса DBUser:

// конструктор класса DBUser для создания локальной копии документа User

public DBUser(Document userDocument) {

this.\_id = userDocument.getObjectId(USER\_ID);

this.password = userDocument.getString(USER\_PASSWORD);

this.address = new DBAddress((Document) userDocument.get(USER\_ADDRESS));

this.email = userDocument.getString(USER\_EMAIL);

this.name = userDocument.getString(USER\_NAME);

this.phoneNumber = userDocument.getString(USER\_PHONE\_NUMBER);

this.picture = userDocument.getString(USER\_PICTURE);

this.rating = userDocument.getDouble(USER\_RATING);

}

// получение документа для отправки на сервер

public Document getDocument() {

Document result = new Document();

result.put(USER\_ID, this.\_id);

result.put(USER\_PASSWORD, this.password);

result.put(USER\_ADDRESS, address.getDocument());

result.put(USER\_EMAIL, this.email);

result.put(USER\_NAME, this.name);

result.put(USER\_PHONE\_NUMBER, this.phoneNumber);

result.put(USER\_PICTURE, this.picture);

result.put(USER\_RATING, this.rating);

return result;

}

### **3.2.4 Создание пользовательского интерфейса**

Прежде чем разрабатывать само приложение я сделал макет пользовательского интерфейса в редакторе Figma.

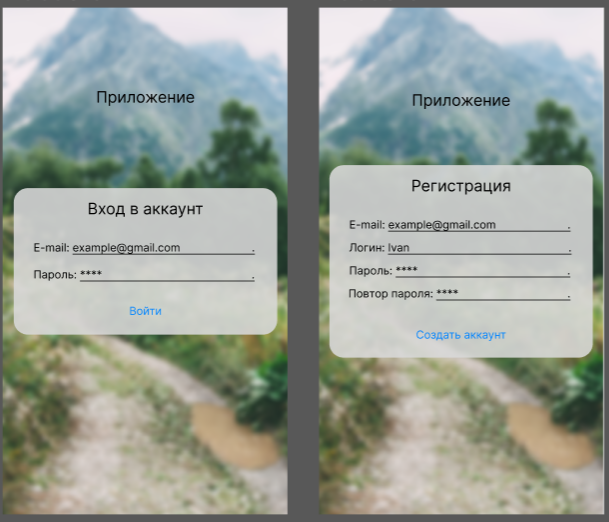


Рисунок 7 -Экраны входа пользователя в аккаунт и его регистрации, созданные в программе figma

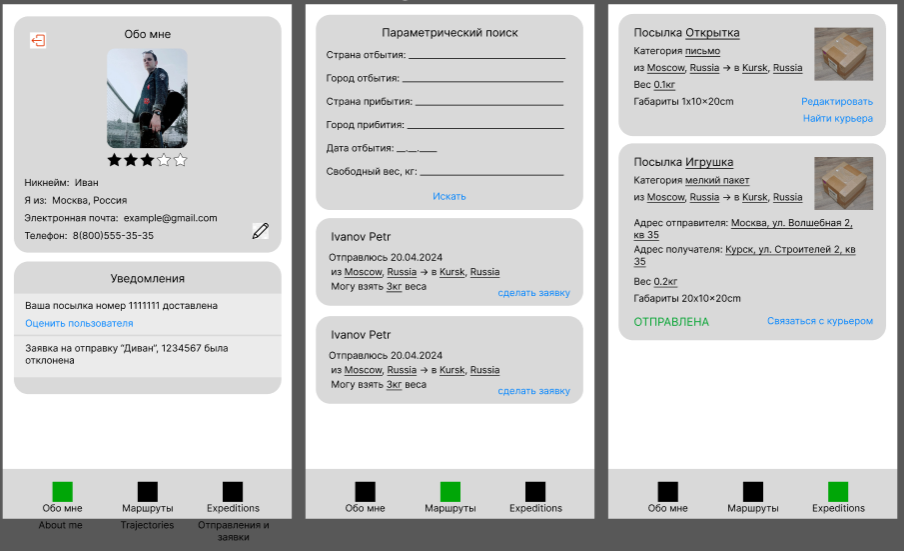


Рисунок 8 -Экран просмотра данных пользователя, экран просмотра маршрутов других пользователей и экран просмотра посылок пользователя созданные в программе figma

Подобное моделирование будущего интерфейса не только очень помогает правильно распределить элементы управления, но и помогает продумать и утвердить варианты использования пользователем приложения.

После проработки всех аспектов интерфейса я приступил к созданию разметки в Android Studio.

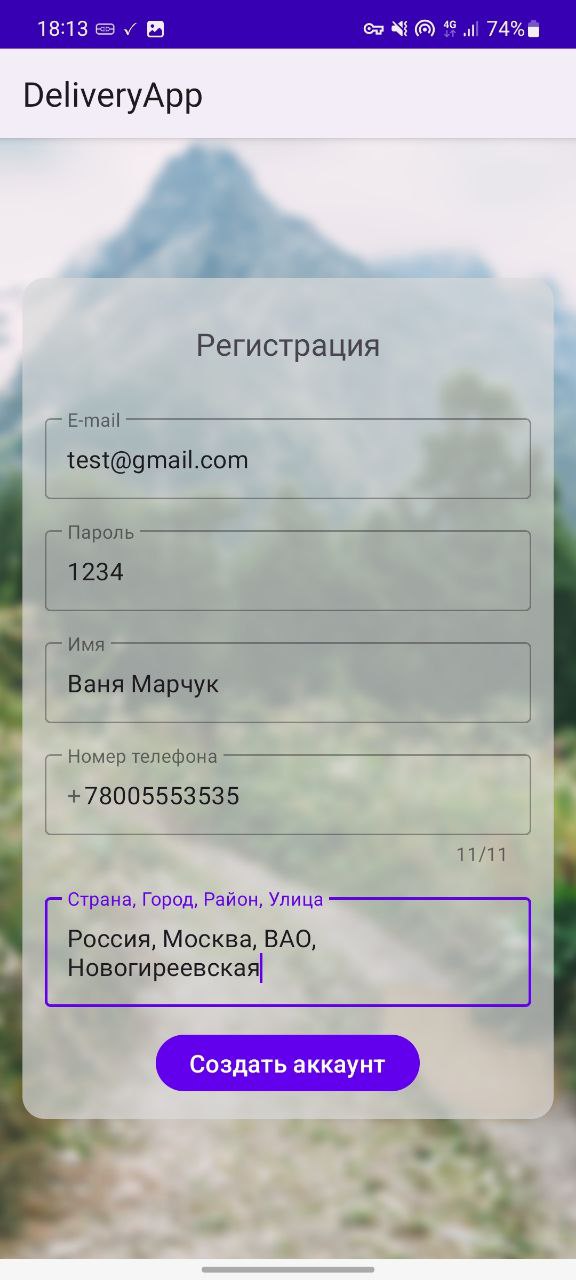
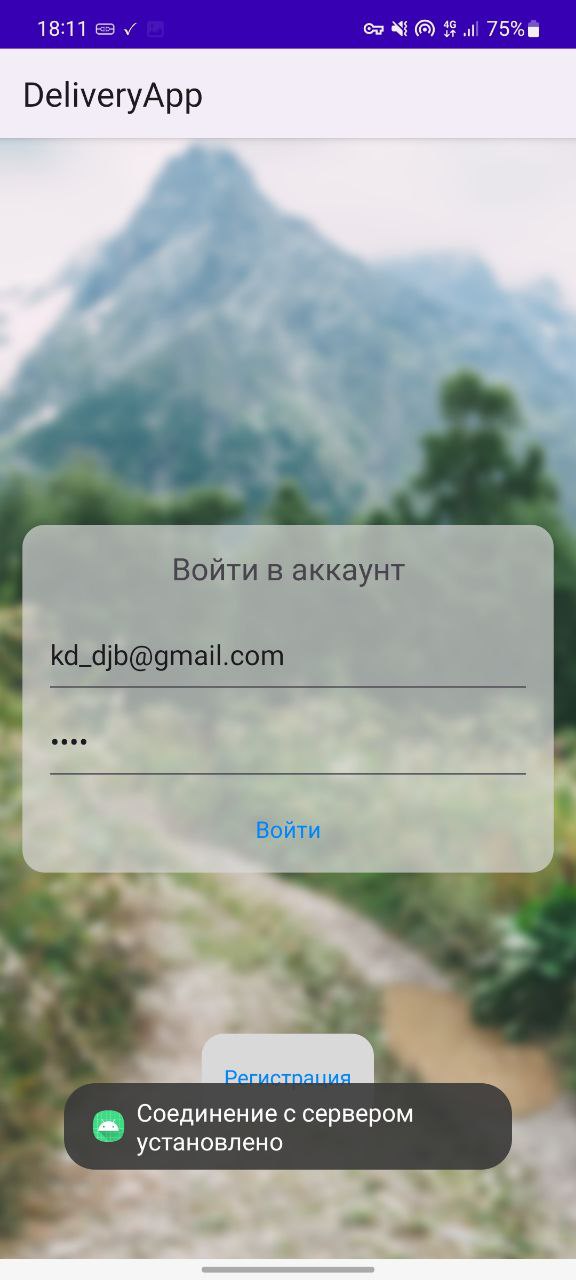


Рисунок 9 -Экраны авторизацци (слева) и регистрации (справа) пользоваетеля

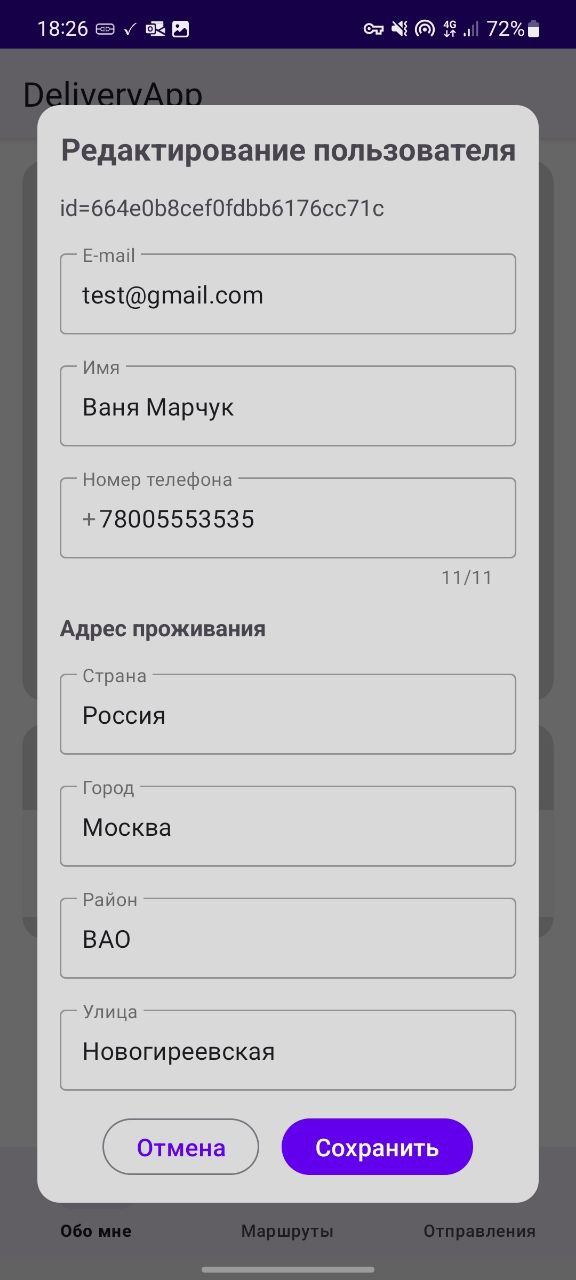
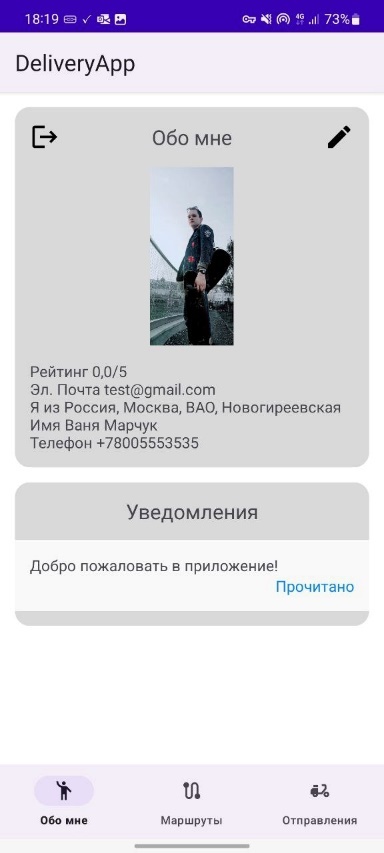


Рисунок 10 -Экран просмотра информации о пользователе и уведомлений (слева) и диалоговое окно редактирования пользователя (справа).

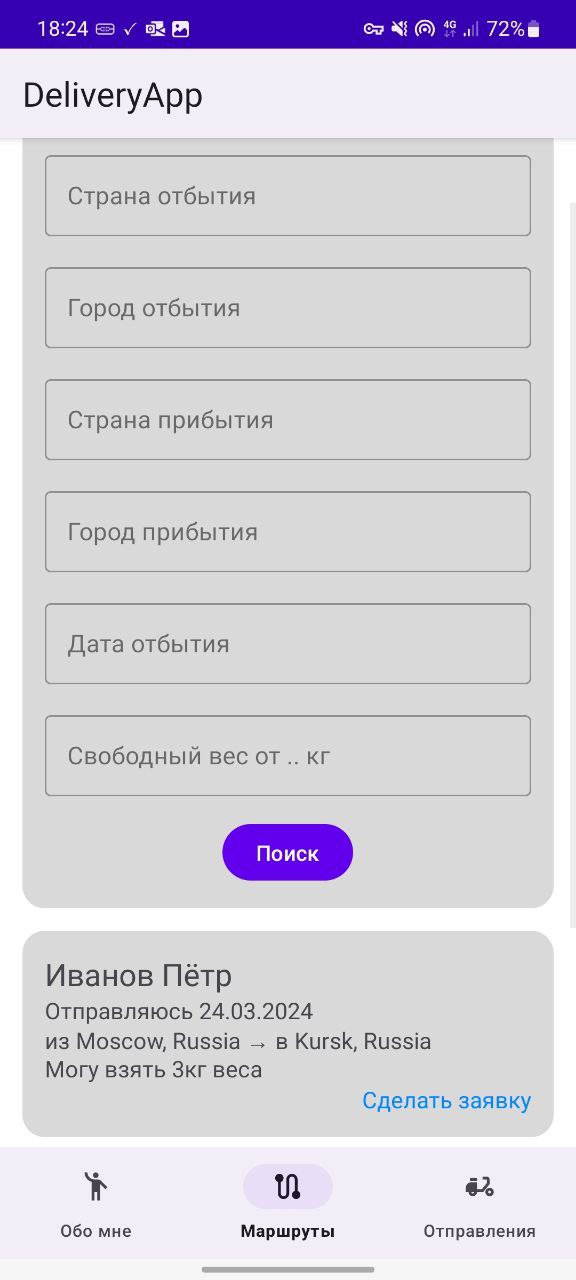


Рисунок 11 -Экран поиска и просмотра маршрутов других пользователей

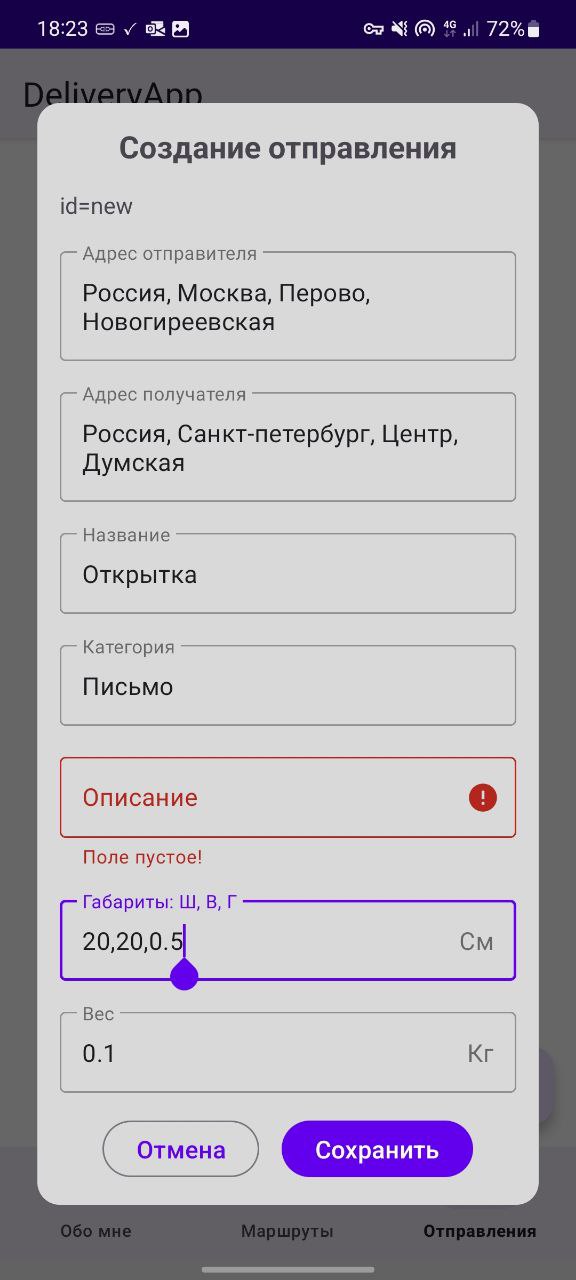
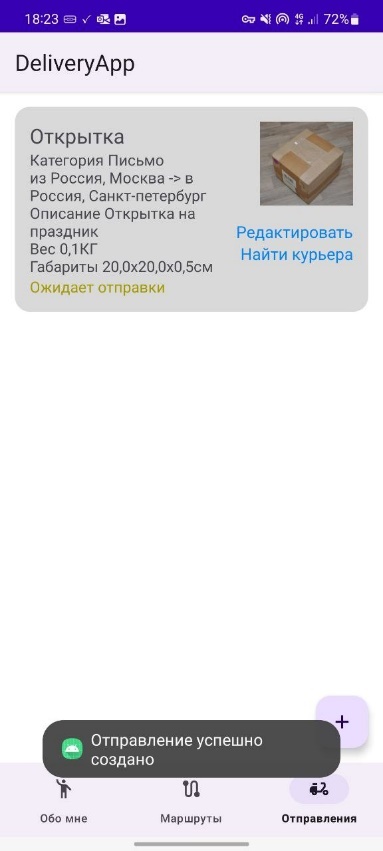


Рисунок 12 -Экранпросмотра отправлений пользователя (слева) и диалоговое окно редактирования отправления (справа)

### **3.2.5 Навигация**

Совместно с архитектурой MVVM в Android нередко используют схему навигации NavigationGraph. Это новейший подход проектирования связей и переходов между экранами приложения применимый только в SingleActivity приложениях.

Принцип его состоит в том, что программист с помощью языка разметки XML, описывает все переходы (action) между эканами (Fragments)

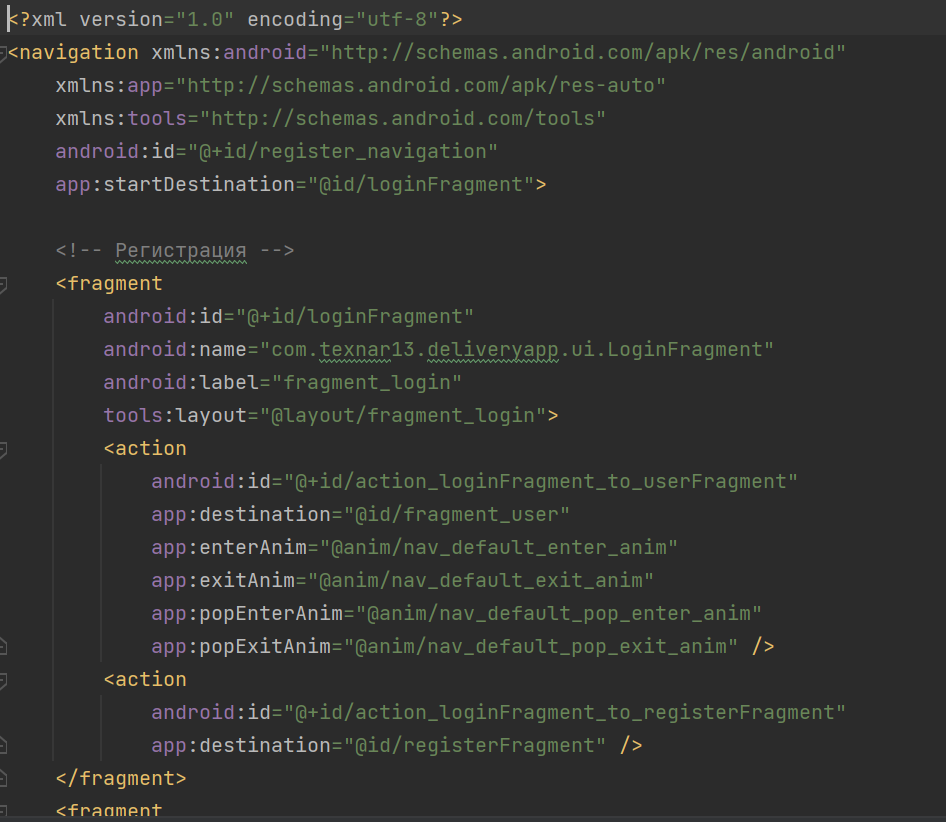


Рисунок 13 -Пример описания переходов (Action) для экрана (Fragment) аутентификации пользователя. Первый переход описывает переход на страницу пользователя после аутентификации, а второй описывает переход на страницу регистрации нового пользователя.

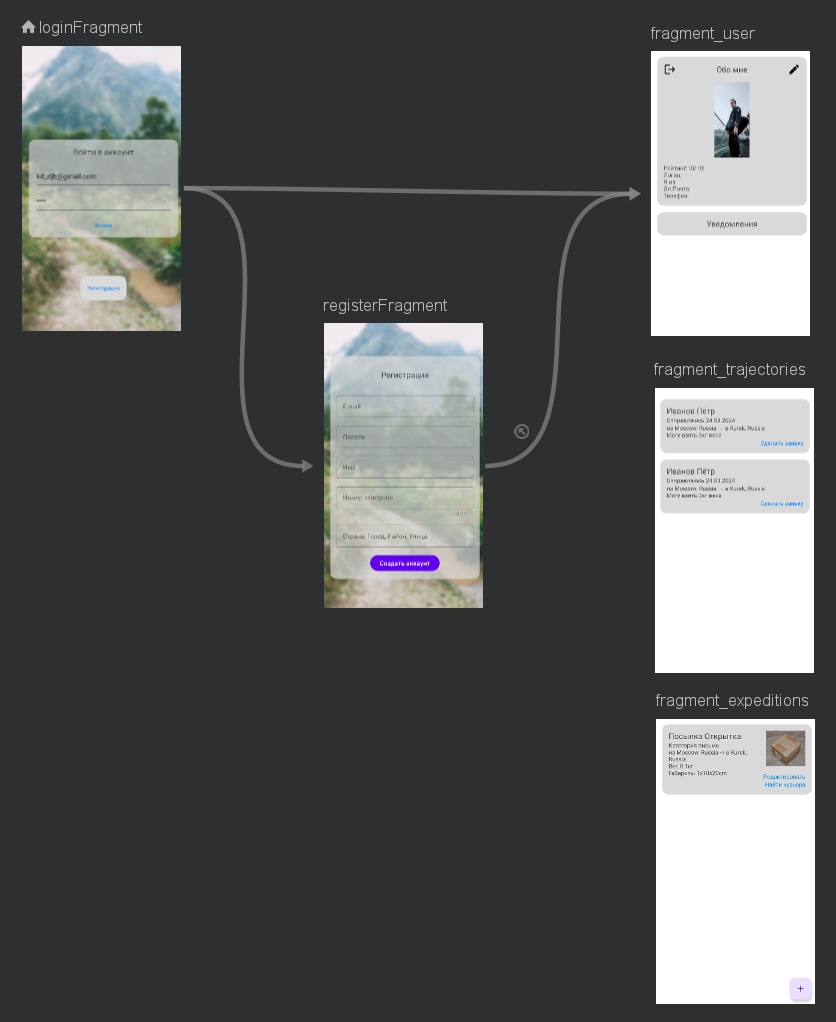


Рисунок 14 -Визуализация схемы переходов между экранами Navigation.

Однако как видно из рисунка выше, экраны пользователя, маршрутов и отправлений никак друг с другом не связаны. Сделано это по тому, что за навигацию между ними отвечает специальный компонент NavigationView, у которого есть готовая логика для реализации NavigationGraph.

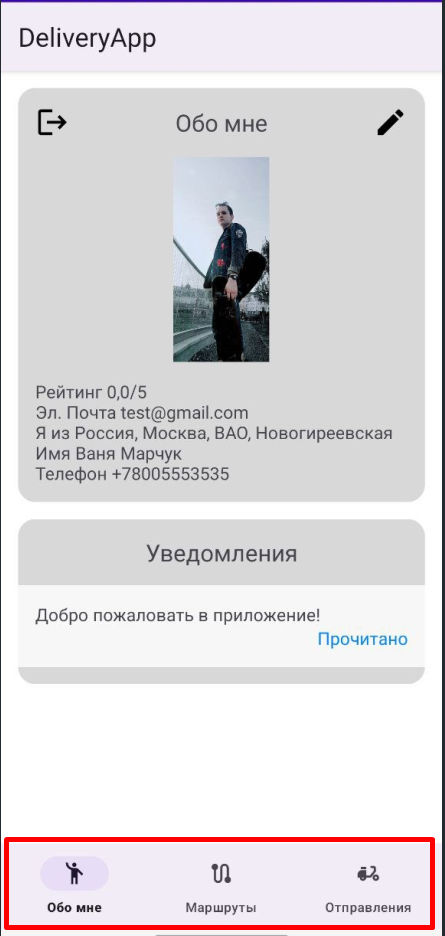


Рисунок 15 -Элемент NavigationView (обведен красным)

### **4.2.6 Работа с базой данных, подключение**

Вся логика работы с удаленной базой данных была помещена в MainViewModel, поскольку использование этого класса в архитектуре MVVM гарантирует, что ViewModel продолжит существовать в памяти приложения до тех пор, пока приложение не будет закрыто. То есть это отлично подходящее место в приложении для того чтобы организовать в нем подключение к удаленной БД.

Приведу фрагмент кода подключения к БД:

// подключение к онлайн бд

void connectDB() {

String appID = "application-0-epiwn";

App realmConnectionApp = new App(new AppConfiguration.Builder(appID).build());

// авторизация

Credentials apiKeyCredentials = Credentials.apiKey(apiKey);

AtomicReference<User> user = new AtomicReference<>();

realmConnectionApp.loginAsync(apiKeyCredentials, it -> {

if (it.isSuccess()) {

Log.e(MY\_LOG, "Successfully authenticated using an API Key.");

user.set(realmConnectionApp.currentUser());

MongoClient mongoClient = realmConnectionApp.currentUser().getMongoClient("mongodb-atlas");

// получаем ссылку на БД

mongoDatabase = mongoClient.getDatabase("DeliveryApplicationDB");

// уведомляем активность, что подключение установлено

connectionStatus.setValue(ConnectionStatusValue.STATUS\_CONNECTED);

} else {

Log.e(MY\_LOG, it.getError().toString());

// уведомляем активность, что подключение не произошло

connectionStatus.setValue(ConnectionStatusValue.STATUS\_ERROR);

// Повторная попытка

connectionStatus.setValue(ConnectionStatusValue.STATUS\_NONE);

// отложенный вызов

(new Handler(Looper.getMainLooper())).postDelayed(this::connectDB, 4000);

}

});

}

В этом коде происходит асинхронное подключение к удаленной базе данных и получение ссылки на пременную mongoDatabase, через которую в дальнейшем идет получение ссылок на коллекции для работы.

Подключение здесь происходит через пользователя user, но это не пользователь который пользуется приложением, а API клиент, созданный в MongoDB Atlas.

### **3.2.7 Работа с базой данных, Чтение и запись**

В качестве демонстрации работы чтения и записи приведу два метода из ViewModel: аутентификация пользователя и редактирование параметров пользователя.

Листинг методов аутентификации и редактирования пользователя:

// Аутентификация пользователя

void authUser(String email, String password) {

if (mongoDatabase == null) return;

// получаем таблицу пользователей

MongoCollection<Document> usersCollection = mongoDatabase.getCollection(DBUser.TABLE\_NAME);

// поиск пользователя в бд

Document query = new Document(DBUser.USER\_EMAIL, email);

usersCollection.find(query).first().getAsync(result -> {

if (result.isSuccess()) {

Document documentUser = result.get();

// если пользователь найден

if (documentUser != null) {

String pass = documentUser.getString(DBUser.USER\_PASSWORD);

if (hashPassword(password).equals(pass)) {

// вытягиваем данные из пользователя и сохраняем их

currentUser.setValue(new DBUser(documentUser));

sendToast("Авторизован");

// Получаем данные пользователя

loadUserNotifications();

loadUserExpeditions();

loadTrips();

} else {

sendToast("Пароль неподходит!");

}

} else

sendToast("Пользователь не найден");

} else {

sendToast("Ошибка подключения");

}

});

}

// редактирование пользователя

public void editUser(@NonNull DBUser editedUserData) {

Document editedDocument = editedUserData.getDocument();

// получаем таблицу пользователей

MongoCollection<Document> usersCollection = mongoDatabase.getCollection(DBUser.TABLE\_NAME);

// Получаем идентификатор пользователя из измененных данных

ObjectId userId = editedDocument.getObjectId(DBUser.USER\_ID);

// Обновляем документ пользователя новыми данными

usersCollection.findOneAndReplace(new Document(DBUser.USER\_ID, userId), editedDocument).getAsync(result -> {

if (result.isSuccess()) {

Document updatedUser = result.get();

if (updatedUser != null) {

// если данные сохранены успешно, обновляем глобальную копию переменной и интерфейс

currentUser.setValue(editedUserData);

sendToast("Данные пользователя успешно сохранены");

} else {

sendToast("Ошибка, пользователь не найден");

}

} else {

sendToast("Ошибка при сохранении данных пользователя");

}

});

}

Как видно из кода выше, для работы с документами в коллекции, необходимо получить из базы данных ссылку на саму коллекцию MongoCollection<Document> usersCollection. В качестве параметра коллекции передается документ, так как загрузка и выгрузка происходит именно через переменные этого типа. Именно по этому как я и описывал выше необходимы методы перевода документов в data классы и наоборот.

Все операции к базе данных (и чтение и запись) происходят через вызов метода getAsync(). И все они являются асинхронными, то есть нельзя отработать действия пользователя в интерфейсе и в том же вызове вернуть ему ответ от базы данных, потому что все происходит асинхронно и программе необходимо дождаться результатов. Это создает достаточно большое количество проблем при возвращении результата в пользовательский интерфейс, так как пришлось бы к каждому запросу еще прикладывать идентификатор поля к которому этот запрос был сделан.

Однако в архитектуре MVVM есть решение этой проблемы, и это LiveData. Это класс позволяющий элементам активности, например элементам интерфейса «подписаться» на изменения переменной находящейся во ViewModel. Допустим в примере выше я меняю переменную currentUser, её объявление выглядит так:

public MutableLiveData<DBUser> currentUser = new MutableLiveData<>();

То есть внутри себя она хранит обычный Data объект DBUser, однако еще такая переменная позволяет реализовать механизм подписки следующим образом:

mainViewModel.currentUser.observe(this, dbUser -> {

// вывод строки состояния пользователя

if (dbUser != null) {

StringBuilder address = new StringBuilder();

for (int i = 0; i < dbUser.getAddress().getArray().length; i++) {

address.append(dbUser.getAddress().getArray()[i]);

if (i != dbUser.getAddress().getArray().length - 1) address.append(", ");

}

userDescription.setText(String.format(Locale.getDefault(),

"Рейтинг %.1f/5\nЭл. Почта %s\nЯ из %s\nИмя %s\nТелефон %s",

dbUser.getRating(),

dbUser.getEmail(),

address,

dbUser.getName(),

dbUser.getPhoneNumber()

));

}

});

С помощью метода observe() можно прямо в разметке отслеживать изменения данных во ViewModel и выводить их когда данные загрузятся с сервера. При этом никакой предачи компонентов для отрисовки во ViewModel не происходит, компоненты сами подписываются на изменения данных. Получается разделение задач, ViewModel занимается работой с базой данных, Activity занимается отрисовкой элементов и «подписана» на ViewModel.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано мобильное Android-приложение, предназначенное для помощи иностранцам в поиске людей, готовых передать посылки или открытки на родину. Приложение взаимодействует с распределенной базой данных MongoDB, размещенной на сервере Atlas, что обеспечивает надежное хранение и доступность данных.

**Достигнутые цели**

Основная цель работы заключалась в создании Android-приложения, позволяющего пользователям находить путешественников, готовых взять их посылки. Для достижения этой цели были выполнены следующие задачи:

- Разработан интуитивно понятный пользовательский интерфейс с использованием библиотеки Material Design 3.

- Реализована асинхронная работа с базой данных с использованием библиотеки Realm, что обеспечило оперативное обновление данных и их сохранность.

**Основные результаты**

Разработанное приложение продемонстрировало высокую эффективность и стабильность в работе с MongoDB, размещенной на сервере Atlas. Были успешно реализованы ключевые функциональные возможности, включая регистрацию пользователей, поиск маршрутов, добавление посылок. Благодаря использованию асинхронных запросов, приложение обеспечивает актуальность данных в реальном времени без необходимости хранения локальных копий базы данных на устройстве пользователя.

#### Личный опыт и приобретенные знания

Работа над проектом позволила углубить знания в области разработки мобильных приложений для платформы Android, получение знаний осовременных архитектурах (MVVM) и методах работы с ними, а также изучить особенности работы с распределенными базами данных. Использование современных библиотек, таких как Realm, Material Design 3 и NavigationGraph значительно расширило понимание лучших практик и подходов к созданию производительных и удобных приложений. Кроме того, практический опыт асинхронной работы с данными позволил более глубоко освоить методы повышения производительности и отзывчивости приложений.

**Перспективы развития**

Разработанное приложение имеет значительный потенциал для дальнейшего развития. Возможные направления улучшений включают:

- Добавление новых функций, таких как система отзывов для пользователей;

- Система обмена сообщениями и фотографиями между пользователями;

- Расширение возможностей фильтрации и сортировки результатов поиска;

- Улучшение механизмов безопасности и защиты данных;

- Разработка версий приложения для других платформ, таких как iOS.

В заключение, выполненная работа продемонстрировала успешное применение современных технологий и подходов для решения практической задачи, а также открыла новые возможности для дальнейших исследований и разработок в области мобильных приложений и распределенных баз данных.