Министерство науки и образования РФ

Федеральное государственное бюджетное учреждение

высшего образования

**«Тверской государственный технический университет»**

(ТвГТУ)

Кафедра программного обеспечения

**Отчет по преддипломной практике**

|  |
| --- |
| Выполнил:  студент группы  Б.ПИН.РИС - 21.06  Сорокин Е. А. |
| Проверила:  д.ф.-м.н. Калабин Александр Леонидович |

Тверь 2025

Оглавление

[Аннотация 4](#_Toc199354134)

[Введение 6](#_Toc199354135)

[Постановка цели и задач 7](#_Toc199354136)

[1. Аналитическая часть 9](#_Toc199354137)

[1.1. Предметная область 9](#_Toc199354138)

[1.2. Анализ существующих решений 10](#_Toc199354139)

[1.3. Новый необходимый функционал 11](#_Toc199354140)

[2. Средства разработки 13](#_Toc199354141)

[2.1. Клиентская часть (Android-приложение) 13](#_Toc199354142)

[2.2. Серверная часть 14](#_Toc199354143)

[2.3. База данных 15](#_Toc199354144)

[2.4. Среда разработки 16](#_Toc199354145)

[3. Проектная часть 18](#_Toc199354146)

[3.1. Описание приложения 18](#_Toc199354147)

[3.1. Клиентская часть (мобильное приложение) 19](#_Toc199354148)

[3.1.1. Экран авторизации (LoginActivity) 20](#_Toc199354149)

[3.1.2. Главный экран (MainActivity) — общее описание 22](#_Toc199354150)

[3.1.3 ViewModel и управление состоянием 30](#_Toc199354151)

[3.1.4 Сервис ActiveService и TCP-соединение 32](#_Toc199354152)

[3.1.5 Сервис BackgroundServiceLocation и отправка координат 34](#_Toc199354153)

[3.2 Серверная часть 36](#_Toc199354154)

[3.3 База данных 38](#_Toc199354155)

[3.3.1 Структура базы данных 38](#_Toc199354156)

[3.3.2 Связи и ограничения 39](#_Toc199354157)

[3.3.3 Расширение 40](#_Toc199354158)

[4. Документирование 41](#_Toc199354159)

[4.1 Установка и запуск 41](#_Toc199354160)

[4.1.1 Серверная часть 41](#_Toc199354161)

[4.1.2 Клиентская часть (Android-приложение) 41](#_Toc199354162)

[4.2 Работа с системой 42](#_Toc199354163)

[4.2.1 Авторизация 43](#_Toc199354164)

[4.2.2 Экран профиля 44](#_Toc199354165)

[4.2.3 Экран карты 45](#_Toc199354166)

[4.2.4 Сканирование 46](#_Toc199354167)

[4.3 Тестирование 47](#_Toc199354168)

[4.3.1 Методика тестирования 48](#_Toc199354169)

[4.3.2 Примеры сценариев 48](#_Toc199354170)

[4.3.3 Среда тестирования 51](#_Toc199354171)

[4.3.4 Общие результаты 52](#_Toc199354172)

[Заключение 53](#_Toc199354173)

[Литература 55](#_Toc199354174)

# Аннотация

Тема: Разработка мобильного приложения для сбора отзывов

***Автор дипломной работы:*** студент ТвГТУ группы Б.ПИН.РИС.21.06 Сорокин Евгений Александрович, специальность «Программная инженерия».

***Руководитель: Артемов Игорь Юрьевич***

Объём ВКР составляет 55 страницы, 13 рисунков, 3 таблицы.

***Примечание***: Данный проект направлен на создание мобильной платформы, позволяющей пользователям оставлять отзывы как о других пользователях, так и о различных общественных местах. Особое внимание уделяется разработке удобного и интуитивного интерфейса, обеспечивающего простоту взаимодействия между пользователями и поддержку различных сценариев использования — от оценки сервиса до персональных отзывов.

***Объект работы***: Андроид-приложение.

***Цель работы***: Разработка Android-приложения, обеспечивающего удобную систему оценки и сбора отзывов о пользователях и общественных объектах. Система должна быть масштабируемой, поддерживать хранение и обмен данными через серверную часть и обеспечивать пользователю простой и безопасный способ оставления отзывов.

В выпускную квалификационную работу входит введение, аналитическая часть, специальная часть, проектная часть, документирование, заключение и список литературы.

**Введение** содержит обзор темы, описание актуальности проекта, его практической значимости, цели и задач.

**Аналитическая часть** включает описание предметной области, обзор существующих решений, выявление требований к функционалу приложения.  
**Средства разработки** описывают выбор технологий и библиотек, используемых на всех этапах реализации.

**Проектная часть** представляет архитектуру и реализацию приложения, включая серверную и клиентскую стороны, а также ключевые алгоритмы.  
**Документирование** охватывает соответствие техническому заданию, тестирование, инструкцию по установке и руководство пользователя.  
В **заключении** подводятся итоги проделанной работы, даётся оценка достигнутых результатов.

В конце приводится **список использованной литературы**, включающий документацию, научные публикации и интернет-ресурсы.

# Введение

В условиях стремительного развития цифровых технологий и мобильных сервисов всё большую значимость приобретают платформы, ориентированные на пользовательский контент. Отзывы, оставленные другими людьми, играют важную роль при принятии решений — будь то выбор заведения, покупка товара или взаимодействие с другими пользователями. Несмотря на широкое распространение приложений для оценки организаций, существует очевидный дефицит решений, позволяющих оставлять отзывы не только на места, но и на конкретных людей. Между тем, такая возможность может оказаться крайне полезной в бытовых, учебных и профессиональных ситуациях — для формирования первого впечатления, а также для последующего анализа репутации человека.

Особую актуальность данное направление приобретает в локальных сообществах: студенты, жители жилых комплексов и простые пользователи, которым важно делиться личным опытом взаимодействия. Более того, наличие отзывов может повлиять, например, на решение работодателя о дальнейшем сотрудничестве, а значит, приложение приобретает не только социальную, но и практическую значимость.

Целью настоящей выпускной квалификационной работы является разработка Android-приложения, предназначенного для сбора, хранения и отображения отзывов о пользователях и общественных объектах. Предлагаемое решение должно обеспечить удобный интерфейс, простоту взаимодействия и расширенные функциональные возможности, включая: регистрацию и авторизацию пользователей, добавление отзывов к людям и местам, просмотр полученных и отправленных отзывов, а также загрузку фотографии в профиль.

В качестве технологической основы проекта выбраны: язык программирования Kotlin для клиентской части на платформе Android, TCP-сервер, реализованный на языке C++, база данных PostgreSQL, а также передача данных в формате JSON. Для отображения географических объектов интегрирован компонент Яндекс.Карт.

Практическая значимость разработки заключается в создании инструмента, способного повысить уровень осознанности при взаимодействии между пользователями, улучшить доверие в цифровом пространстве и расширить возможности для оценки и анализа опыта общения в повседневной жизни.

## Постановка цели и задач

**Целью** выпускной квалификационной работы является разработка мобильного Android-приложения, предназначенного для сбора, хранения и отображения отзывов о пользователях и общественных объектах с использованием клиент-серверной архитектуры.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1. Провести анализ существующих решений в области мобильных платформ для оценки пользователей и объектов, выявить их преимущества и недостатки.
2. Сформировать перечень функциональных требований к разрабатываемому приложению.
3. Выбрать и обосновать используемые инструменты и технологии разработки для мобильного клиента и серверной части системы.
4. Разработать архитектуру приложения, включающую:
   * мобильный клиент на языке Kotlin;
   * серверную часть на языке C++ с поддержкой TCP-соединений;
   * базу данных PostgreSQL для хранения информации о пользователях, отзывах и объектах.
5. Реализовать ключевые функции мобильного приложения:
   * регистрация и авторизация;
   * добавление и просмотр отзывов;
   * загрузка и отображение пользовательского изображения;
   * взаимодействие с сервером через протокол TCP и формат JSON.
6. Обеспечить корректное взаимодействие между клиентом и сервером, реализовать обмен данными в реальном времени.
7. Провести тестирование разработанного решения на различных устройствах и сценариях использования.
8. Подготовить документацию по установке, использованию и сопровождению приложения.
9. Задокументировать созданное решение.

# 1. Аналитическая часть

## 1.1. Предметная область

В современном цифровом обществе отзывы пользователей становятся важным инструментом оценки и формирования доверия. Наиболее активно такая практика применяется при выборе товаров, услуг, заведений и мест — пользователи делятся впечатлениями и влиянием на качество взаимодействия. Однако в повседневной жизни не меньшее значение имеют личные взаимодействия между людьми: соседи, арендодатели, попутчики, исполнители бытовых услуг и т.д. Возможность оставить отзыв о человеке на основе личного опыта может повысить прозрачность и надёжность этих взаимодействий.

В то же время большинство существующих решений не предусматривают открытого и доступного механизма оценки физических лиц. Это создаёт дефицит инструментов, позволяющих формировать персональную цифровую репутацию.

Особый интерес представляет опыт Китая, где внедрена система социального рейтинга граждан, агрегирующая данные о поведении и влияющая на доступ к различным услугам. Несмотря на государственную природу этой системы, её идея отражает растущую значимость оценки надёжности и репутации не только организаций, но и отдельных людей.

Мобильные устройства, особенно платформы на базе Android, предоставляют удобные средства для реализации подобных сервисов. Благодаря широкому распространению, возможностям геолокации, камере, сетевому обмену и доступу к системам авторизации, смартфоны являются оптимальной платформой для реализации системы отзывов между пользователями.

Таким образом, предметная область данной работы охватывает разработку Android-приложения, которое позволяет пользователям:

* оставлять и получать отзывы о других людях и объектах;
* формировать цифровой профиль с отзывами;
* взаимодействовать в рамках открытой, неформальной репутационной системы.

## 1.2. Анализ существующих решений

На сегодняшний день на рынке мобильных приложений широко представлены платформы, ориентированные на сбор отзывов и пользовательских оценок. Большинство из них фокусируются на товарах, услугах или общественных местах, и лишь немногие частично охватывают возможность оставления отзывов о конкретных людях. Ниже приведён обзор наиболее релевантных решений с указанием их функциональных возможностей и ограничений.

***Google Maps, Яндекс Карты, 2ГИС***

Эти платформы предоставляют возможность оставлять отзывы о местах (кафе, магазинах, учреждениях и т.д.). Пользователь может указать рейтинг, добавить комментарий и фотографии. Однако все отзывы привязаны исключительно к географическим объектам, и никакой функциональности по оценке конкретных сотрудников или пользователей не предусмотрено.

***Airbnb, BlaBlaCar***

Данные сервисы позволяют оставлять отзывы о людях — арендаторах, водителях или попутчиках. Однако отзывы возможны **только после подтверждённого взаимодействия через саму платформу**. Свободный обмен отзывами между пользователями, не связанными платформой напрямую, невозможен.

***Instagram, Telegram, VK***

В социальных сетях можно комментировать действия других пользователей, однако отсутствует структурированный механизм для формирования отзывов и рейтингов. Публикации могут носить оценочный характер, но они не создают накопительного цифрового профиля с отзывами.

***Trustpilot, IRecommend, Otzovik***

Эти площадки служат для публикации обзоров и отзывов о товарах, сервисах и компаниях. Тем не менее, возможность оценки физических лиц в них отсутствует.

***Китайская система социального рейтинга***

Интересным прецедентом является государственная система **социального рейтинга** в Китае, направленная на стимулирование "надёжного поведения" граждан. Система собирает разнородные данные (финансовые, правовые, поведенческие) и формирует на их основе рейтинг, который может влиять на доступ к услугам. Несмотря на радикальность подхода, он демонстрирует интерес общества и государства к инструментам цифровой репутации.

***Ограничения текущих решений***

На основании анализа можно выделить ряд общих недостатков:

* невозможность свободно оставлять отзывы на конкретного человека вне рамок платформы;
* отсутствие единого пользовательского рейтинга, агрегирующего мнения из разных источников;
* ограниченный круг сценариев: большинство решений ориентировано на бизнес, а не на повседневное социальное взаимодействие;
* ограниченные функции приватности, фильтрации и модерации пользовательского контента.

## 1.3. Новый необходимый функционал

Анализ существующих решений в сфере отзывов показывает, что большинство мобильных приложений предоставляют ограниченные возможности, ориентированные на оценку организаций или товаров. Отсутствует универсальный инструмент, позволяющий пользователям свободно обмениваться отзывами друг о друге или об объектах социальной инфраструктуры. На основе анализа предметной области и существующих решений была сформирована потребность в разработке мобильного сервиса, обладающего расширенным функционалом.

В рамках разрабатываемого Android-приложения предполагается внедрение следующих новых возможностей:

* **Оценка конкретных пользователей и объектов**, вне зависимости от того, происходило ли формальное взаимодействие внутри платформы. Это принципиально отличает приложение от таких решений, как Airbnb или BlaBlaCar, где оставить отзыв можно только после подтверждённого контакта [1].
* **Просмотр репутации пользователя по полученным отзывам**, что позволяет другим участникам формировать первичное впечатление на основе опыта предыдущих взаимодействий. Таким образом реализуется основа цифровой репутации.
* **Формирование открытого профиля с фотографией**, в котором отображаются отзывы, рейтинги, дата регистрации, а в перспективе — подтверждённые действия пользователя (например, участие в челленджах, общественная активность и т.д.).
* **Интерфейс для написания и редактирования отзывов**, содержащий как текстовые, так и числовые оценки.
* **Интеграция с картографическим сервисом** для привязки отзывов к конкретным местам, например, при оценке общественных локаций [2].
* **Серверное хранилище и синхронизация данных через TCP-протокол**, реализуемая с помощью C++-сервера и базы данных PostgreSQL, обеспечивающая масштабируемость и надёжность хранения отзывов.
* **Безопасность и приватность**, включающая защиту передаваемых данных, ограничение на редактирование чужих отзывов, а также систему блокировок и жалоб.

Добавление описанных функций делает разрабатываемое приложение гибким и универсальным решением, способным адаптироваться к различным пользовательским сценариям: от бытовых взаимодействий до неформальных оценок в студенческой среде или жилом комплексе.

# 2. Средства разработки

Разработка мобильного приложения велась с применением клиент-серверной архитектуры. Мобильный клиент реализован на платформе Android с использованием языка Kotlin. Серверная часть разработана на языке программирования C++ с поддержкой TCP-соединений и взаимодействием с реляционной базой данных PostgreSQL.

## 2.1. Клиентская часть (Android-приложение)

Для реализации мобильного приложения была выбрана платформа **Android** в связке с языком программирования **Kotlin** и фреймворком **Jetpack Compose**. Такой стек технологий обеспечивает современный подход к разработке пользовательского интерфейса, высокую производительность и гибкость архитектуры.

***Jetpack Compose***

**Jetpack Compose** — это официально поддерживаемый Google фреймворк для декларативной разработки интерфейсов в Android. Он позволяет создавать UI-элементы с помощью функций, отображающих текущее состояние приложения. Это упрощает управление состоянием и исключает необходимость в использовании XML-макетов [3].

***Activity и ViewModel***

Для управления жизненным циклом приложения и хранения бизнес-логики используются компоненты:

* **Activity** — служит точкой входа и контейнером для Compose UI.
* **ViewModel** (из библиотеки Android Jetpack) — обеспечивает устойчивое хранение данных при пересоздании интерфейса и упрощает организацию взаимодействия между слоями приложения [4].

***Сетевое взаимодействие***

Для установления связи с сервером используется класс Socket из стандартной библиотеки Java:

*import java.net.Socket*

С его помощью организуется низкоуровневое TCP-соединение, по которому происходит обмен сообщениями в формате **JSON**.

***Работа с JSON***

Для сериализации и десериализации данных применяется библиотека org.json:

*import org.json.JSONObject*

Этот инструмент используется для упаковки данных в формате ключ-значение, отправляемых серверу, и разбора ответов от него.

***Асинхронная обработка***

Для асинхронного выполнения сетевых операций применяется фреймворк **Kotlin Coroutines**:

*CoroutineScope(Dispatchers.IO).launch { ... }*

Асинхронность позволяет выполнять длительные операции (например, сетевые вызовы или чтение файлов) без блокировки основного потока, обеспечивая плавность и отзывчивость пользовательского интерфейса [5].

***Фоновые службы (Services)***

Для реализации непрерывной работы приложения в фоновом режиме используются две службы:

* **ActiveService** — служба, обеспечивающая активное взаимодействие с сервером и отслеживание фоновых процессов.
* **BackgroundServiceLocation** — отдельная служба, предназначенная для постоянного мониторинга геолокации пользователя. Это необходимо для привязки отзывов к месту событий и работы с картами.

Службы реализованы как наследники класса Service, что обеспечивает их работу независимо от активности пользовательского интерфейса.

## 2.2. Серверная часть

В качестве серверной части приложения реализован **отдельный TCP-сервер на языке C++**, обеспечивающий обработку клиентских соединений, приём и разбор JSON-команд, выполнение SQL-запросов и формирование ответов. Выбор C++ обусловлен следующими преимуществами:

* **Высокая производительность** и минимальные накладные расходы на обработку сетевых соединений.
* **Полный контроль над ресурсами**, включая работу с сокетами, буферами и потоками данных.
* **Гибкость при интеграции с PostgreSQL** благодаря использованию библиотеки libpq.

Для реализации сетевого взаимодействия в сервере используются системные вызовы socket, bind, listen, accept, обеспечивающие двустороннюю передачу данных по протоколу **TCP**. Обмен сообщениями между клиентом и сервером осуществляется в формате **JSON**, для чего используется современная библиотека **nlohmann/json** [6].

Преимущества выбранных инструментов:

* Простая интеграция JSON-структур с C++-типами;
* Поддержка всех необходимых операций: подключение к базе, выполнение команд, возврат результатов клиенту;
* Отсутствие зависимости от сторонних фреймворков.

Такая серверная архитектура позволяет обрабатывать авторизацию, регистрацию, работу с отзывами, а также взаимодействие с пользовательскими данными в режиме реального времени.

## 2.3. База данных

В качестве системы управления базами данных была выбрана **PostgreSQL 15** по следующим причинам [7]:

* **Надёжность и устойчивость к сбоям**. PostgreSQL обеспечивает сохранность данных за счёт использования **журнала транзакций (WAL)**, что позволяет восстановить информацию даже в случае аварийного завершения работы сервера.
* **Широкое распространение и открытость**. Являясь объектно-реляционной СУБД с открытым исходным кодом, PostgreSQL активно применяется в проектах различного масштаба.
* **Поддержка транзакций и ограничений целостности**, включая атомарность, согласованность, изолированность и долговечность (ACID).
* **Кроссплатформенность**. СУБД может использоваться на различных операционных системах, включая Linux и Windows.
* **Высокая производительность при работе с большими объёмами данных** и поддержка сложных SQL-запросов.

Для подключения к базе данных и выполнения SQL-операций на стороне сервера используется официальная клиентская библиотека libpq.

## 2.4. Среда разработки

Для разработки клиентской части приложения использовалась официальная интегрированная среда разработки **Android Studio Iguana**. Это одно из последних стабильных изданий Android Studio, предоставляющее широкие возможности для создания, тестирования и отладки Android-приложений.

Преимущества выбранной среды разработки:

* **Полная интеграция с Android SDK и Jetpack Compose**. Android Studio Iguana поддерживает последние версии Jetpack Compose, включая визуальный редактор компоновок и предпросмотр UI-компонентов в реальном времени.
* **Интеграция с Gradle**. Система сборки Gradle позволяет гибко управлять зависимостями, конфигурациями и модулями проекта.
* **Удобная отладка**. Встроенные инструменты отладки, включая логирование Logcat, инспекцию памяти, слежение за потоками и профилирование производительности.
* **Поддержка Kotlin**. Полная совместимость с Kotlin и его инструментами: рефакторинг, автодополнение, статический анализ кода, инспекции и генерация кода.
* **Симуляция разных устройств**. Android Emulator позволяет запускать и тестировать приложение на виртуальных устройствах с различными характеристиками (разрешение, версия ОС, объём памяти и т.д.).
* **Интеграция с Git**. Android Studio имеет встроенные средства для контроля версий и работы с Git-репозиториями.

В качестве дополнительных инструментов использовались:

* **ADB (Android Debug Bridge)** — для взаимодействия с физическими устройствами во время отладки;
* **Material Theme** — для визуального оформления интерфейса;
* **Яндекс MapKit SDK** — подключён через build.gradle для отображения карт и работы с координатами.

Android Studio Iguana предоставила весь необходимый инструментарий для реализации современного, масштабируемого и производительного Android-приложения.

# 3. Проектная часть

## 3.1. Описание приложения

Разрабатываемое программное обеспечение представляет собой **клиент-серверную систему**, предназначенную для сбора и отображения отзывов между пользователями. Приложение включает две основные компоненты: мобильный клиент, реализованный на платформе Android, и сервер, написанный на языке C++, обрабатывающий команды и обеспечивающий взаимодействие с базой данных PostgreSQL.

***Назначение системы***

Система предназначена для:

* хранения и отображения отзывов, оставленных пользователями друг о друге;
* реализации пользовательской авторизации;
* поддержки профиля с отображением имени, координат и времени последнего захода;
* сохранения оценок и комментариев;
* передачи и хранения аватаров;
* безопасной передачи данных через TCP-соединение с использованием формата JSON.

***Архитектура взаимодействия***

Система реализована по классической **клиент-серверной архитектуре**. Схема взаимодействия представлена ниже:

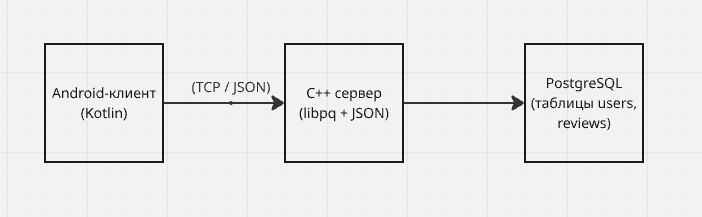


Рисунок 1. Схема взаимодействия.

Сервер принимает команды от клиента в формате JSON, обрабатывает их, взаимодействует с базой данных и возвращает клиенту ответ в том же формате. Все соединения устанавливаются по TCP-протоколу через заданный порт.

## 3.1. Клиентская часть (мобильное приложение)

Мобильный клиент разрабатывается на платформе **Android** с использованием **Kotlin** и фреймворка **Jetpack Compose** для построения пользовательского интерфейса. Приложение реализует прямое TCP-взаимодействие с серверной частью, выполняет авторизацию, позволяет пользователю просматривать и оставлять отзывы, управлять своим профилем и выполнять сопутствующие действия, такие как генерация QR-кодов, сканирование и отправка местоположения.

Клиентская архитектура построена по принципу разделения ответственности:

* интерфейс описывается декларативно в Jetpack Compose;
* бизнес-логика вынесена во ViewModel;
* фоновое взаимодействие (геолокация, сетевое соединение) реализовано через службы;
* обмен данными происходит в формате JSON через TCP-сокет.

***Структурная схема взаимодействия компонентов клиента***

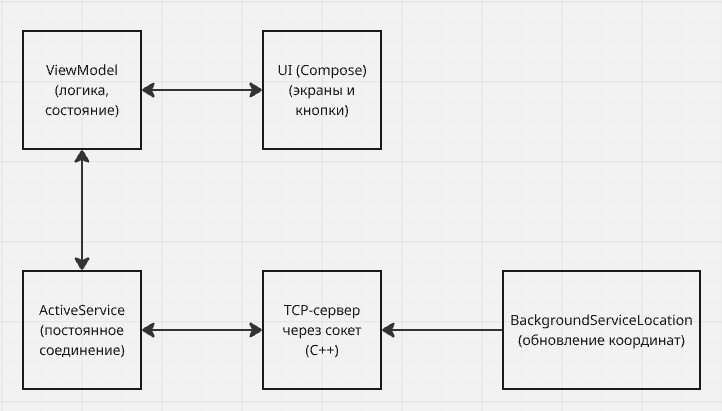


Рисунок 2. Структурная схема.

Каждый компонент работает независимо, но взаимодействует через ViewModel и общие состояния. Все сетевые вызовы происходят вне основного потока, что обеспечивает отзывчивость и стабильность работы интерфейса.

### 3.1.1. Экран авторизации (LoginActivity)

Одной из ключевых точек входа в приложение является экран авторизации, реализованный в классе LoginActivity.kt. Он обеспечивает проверку учётных данных пользователя и установку соединения с сервером. Данный экран выполняет роль первого шага к персонализированному взаимодействию: после успешного входа пользователю становится доступен основной функционал приложения.

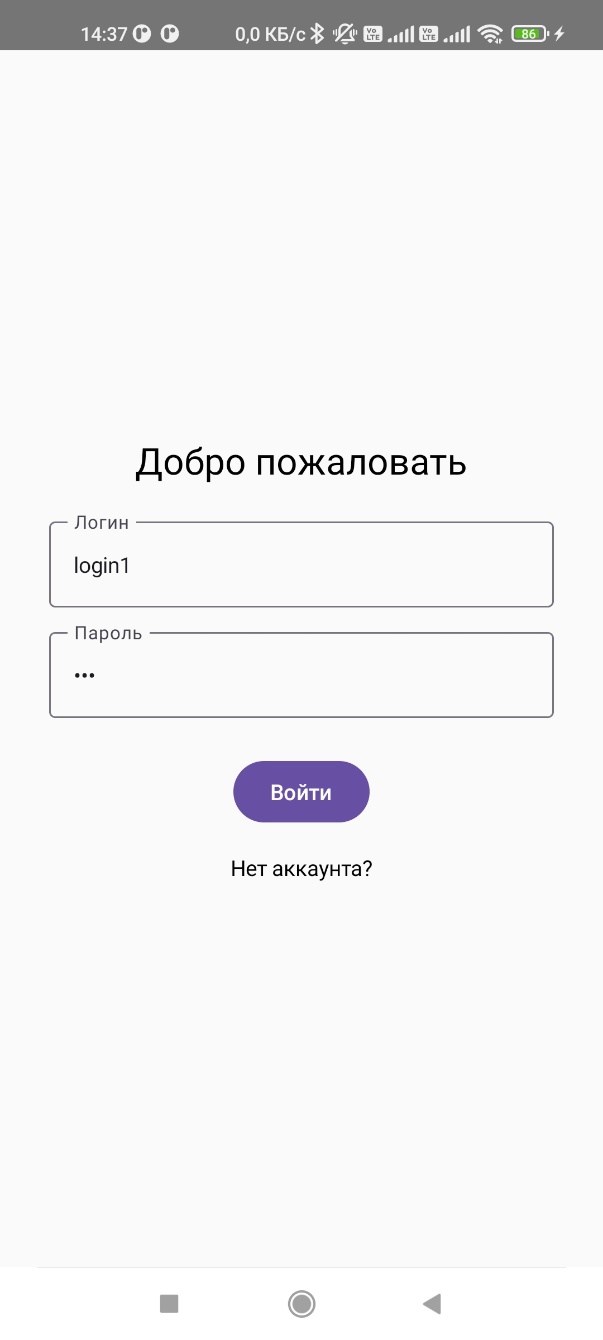


Рисунок 3. Экран авторизации.

***Назначение экрана***

Экран предназначен для:

* ввода логина и пароля;
* отправки аутентификационного запроса на сервер;
* обработки ответа (успешная авторизация или сообщение об ошибке);
* перехода к основному интерфейсу (MainActivity) при успешном входе.

***Запуск фоновых служб***

Сразу после запуска LoginActivity активируются службы:

* ActiveService — устанавливает TCP-соединение и обрабатывает команды от сервера (например, обновление профиля, приём отзывов);

***Структура интерфейса***

Интерфейс экрана построен с использованием компонентов **Jetpack Compose**:

* поля ввода (TextField) для логина и пароля;
* кнопка (Button) для отправки данных;
* текстовые уведомления (Text) для отображения ошибок и статуса.

***Логика авторизации***

После ввода данных пользователь нажимает кнопку "Войти", и выполняется следующая последовательность действий:

1. Формируется JSON-объект с типом команды sql\_login, полем typeSql (например, "login"), и строкой command, содержащей логин и пароль, разделённые пробелом.
2. JSON преобразуется в строку и передаётся серверу по TCP-соединению, созданному через Socket.
3. Сервер выполняет SQL-запрос к таблице myusers, проверяет соответствие пароля и возвращает JSON-ответ.
4. Ответ анализируется на клиенте:
   * если status == "success" — происходит переход к MainActivity;
   * если status == "error" — отображается сообщение об ошибке.

***Безопасность***

Пароль не хранится на клиенте и передаётся только один раз в момент входа. Сервер, в свою очередь, обрабатывает экранирование значений, чтобы предотвратить SQL-инъекции.

### 3.1.2. Главный экран (MainActivity) — общее описание

Главный экран (MainActivity) выполняет функцию **центрального диспетчера пользовательского интерфейса** после авторизации. Он отвечает за отображение основных разделов приложения, запуск и координацию фоновых служб, инициализацию карты и работу с NFC-модулем.

#### **Основные задачи активности**:

* привязка сервиса ActiveService, обеспечивающего TCP-соединение с сервером;
* инициализация и отображение интерфейса на базе **Jetpack Compose** (MyApp);
* по завершению запуск службы BackgroundServiceLocation, отслеживающей координаты пользователя;
* управление навигацией по трём вкладкам: профиль, карта, сканер.

Структура MainActivity наследуется от ComponentActivity, используется метод setContent {} для отображения всего пользовательского интерфейса через Composable-функции. При этом выполняется следующее:

* устанавливается соединение с ActiveService;
* через ViewModel (MyViewModel) передаётся управление состояниями экранов.

***Основные элементы***

* **MyApp()** — главная точка входа для визуальной части, использует Scaffold и BottomNavigationBar для отображения трёх вкладок:
  + Профиль (ProfileScreen)
  + Карта (MapScreen)
  + Сканер (Scanner)
* **Фоновые службы**:
  + ActiveService —приём/отправка JSON;
  + BackgroundServiceLocation — мониторинг координат в фоне по завершению приложения;

**3.1.2.1 Экран профиля**

Экран профиля (ProfileScreen.kt) является центральной частью пользовательского интерфейса, отображаемой после входа. Он позволяет пользователю управлять личными данными, просматривать среднюю оценку и отзывы, загружать аватар, а также делиться ссылкой на свой профиль.

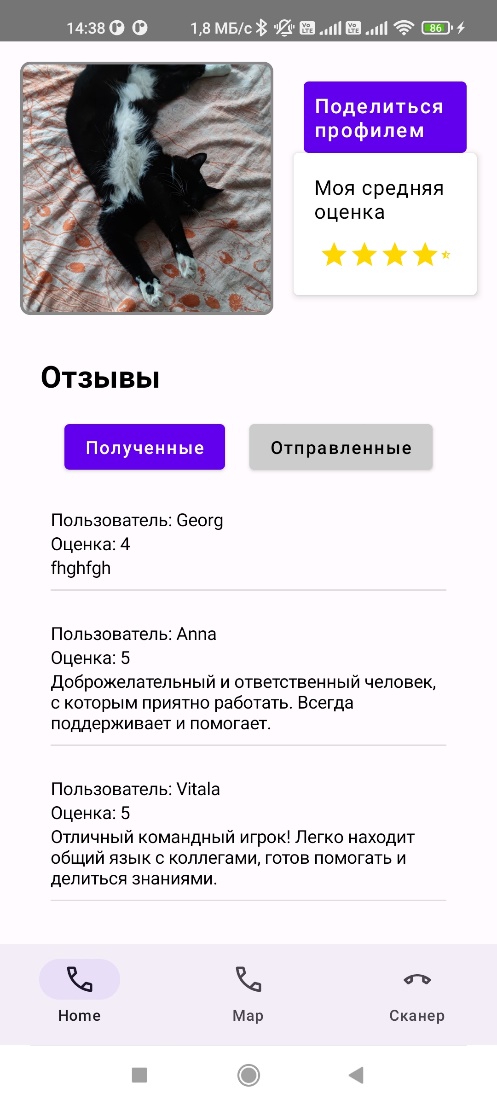


Рисунок 4. Экран профиля.

***Основные функции экрана:***

* Отображение аватара и возможность его загрузки/замены;
* Просмотр средней оценки в виде звёзд;
* Просмотр полученных и отправленных отзывов (по 5 последних);
* Просмотр всех отзывов в отдельном диалоге с фильтрацией;
* Кнопка "Поделиться профилем" с переходом на отдельный экран (ScreenRepost).

***1. Загрузка и отображение аватара***

Аватар пользователя выбирается через системный проводник файлов. При выборе изображения:

* выполняется загрузка байт изображения;
* изображение отправляется на сервер с помощью метода activeService.sendCommandFromActivity("", "send\_avatar", bytes);

Для отображения изображения используется AsyncImage из библиотеки **Coil**.

***2. Получение средней оценки***

При загрузке экрана отправляется SQL-команда на сервер:

SELECT AVG(rating) AS average\_rating FROM ratings WHERE myuser\_id = ...

После получения средней оценки (Float) на экране отображается соответствующее количество звёзд, с возможной половинчатой иконкой.

***3. Получение и отображение отзывов***

Отзывы делятся на два потока:

* **Полученные** — отзывы, оставленные другим пользователям о текущем;
* **Отправленные** — отзывы, оставленные текущим пользователем другим.

По 5 последних записей загружаются через SQL-запросы. Отзывы отображаются в табличном виде с переключением между вкладками. Каждая запись включает:

* имя автора,
* рейтинг (1–5),
* текст отзыва,
* дату.

***4. Полный просмотр всех отзывов***

При клике на звёзды открывается диалог ReviewsDialog, где загружаются все отзывы с возможностью сортировки:

* по дате (по убыванию/возрастанию);
* по рейтингу (по убыванию/возрастанию).

Отзывы сортируются с использованием перечисления SortType и функции sortReviews.

***5. Репост профиля***

При нажатии кнопки «Поделиться профилем» происходит переход на экран ScreenRepost, который позволяет выбор способа поделиться профилем: через QR-код, NFC метку.

**3.1.2.2 Вкладка «Карта» (MapScreen)**

Вкладка «Карта», реализованная в модуле MapScreen.kt, предназначена для визуального отображения текущего положения пользователя и других участников системы на карте, а также для взаимодействия с пользователями в географическом контексте. Карта реализована с использованием **Yandex MapKit**, что позволяет осуществлять плавную анимацию камеры, добавлять маркеры и обрабатывать события касания.

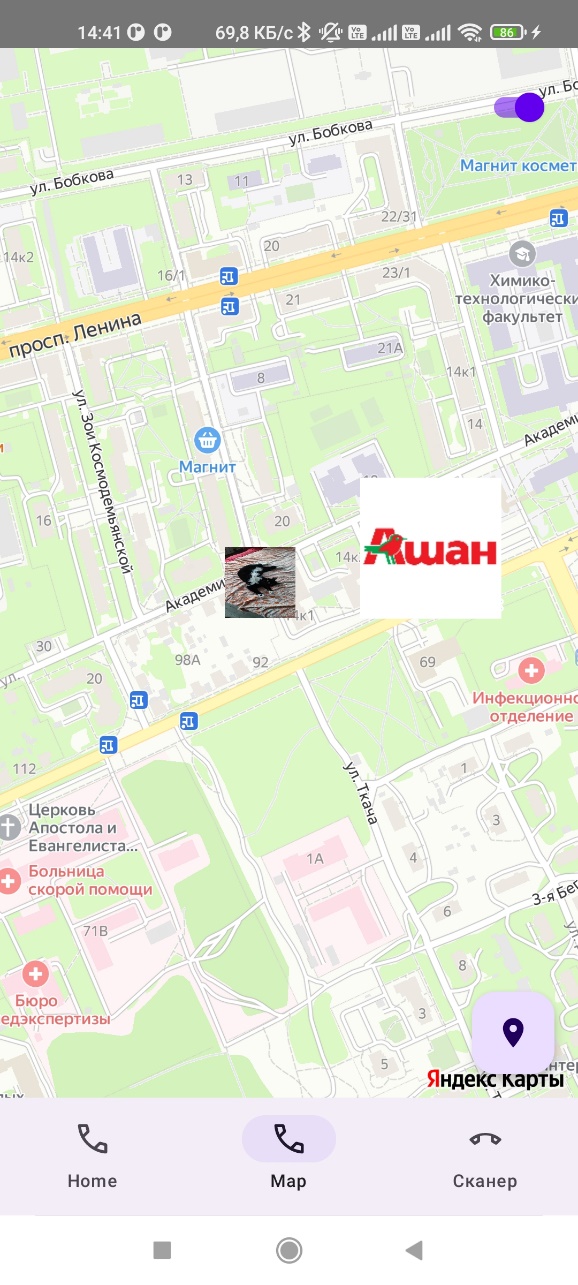


Рисунок 5. Вкладка «Карта».

***Основные функции экрана:***

* отображение местоположения текущего пользователя;
* отображение пользователей и объектов рядом с текущим пользователем на карте;
* возможность оставить отзыв пользователю или объекту;
* управление видимостью пользователя через ползунок;
* возвращение к текущему местоположению через специальную кнопку.

***Отображение карты***

При первом запуске выполняется инициализация MapView с перемещением камеры на текущие координаты пользователя. Если геолокация недоступна, используется значение из SharedPreferences по умолчанию.

val mapView = remember { MapView(context) }

Карта автоматически обновляется при получении координат от ViewModel и отображает:

* маркер текущего пользователя;
* маркеры других пользователей и объектов (с PlacemarkMapObject);
* иконки отображаются с помощью ImageProvider.fromResource.

***Видимость и ограничения***

Пользователь может управлять своей **видимостью** через переключатель Switch, находящийся в верхнем правом углу. Если пользователь выключает видимость, он получает предупреждение о том, что не сможет оставлять отзывы в течение 30 минут. Включение обратно активирует таймер с фиксацией времени в SharedPreferences.

***Взаимодействие с другими пользователями***

При нажатии на маркер другого пользователя открывается диалоговое окно с его информацией (UserInfoDialog). Диалог позволяет:

* просмотреть сведения о пользователе (полученные с сервера);
* оставить отзыв при соблюдении условий (30 минут с момента включения видимости);
* ввести текст и выбрать рейтинг от 1 до 5 звёзд;
* передать отзыв на сервер через ViewModel.

viewModel.sendingReview(reviewText, rating, userId)

***Дополнительные элементы интерфейса***

* **FloatingActionButton** — возвращает камеру к текущей геопозиции пользователя.
* **Обновление списка пользователей** на карте выполняется при запуске и при взаимодействии с интерфейсом.

***Логика блокировки отзывов***

Блокировка оставления отзывов на 30 минут реализована через хранение времени последнего включения видимости в SharedPreferences:

val elapsed = System.currentTimeMillis() - lastEnableTime

val canLeaveReview = elapsed >= halfHourMillis

Это позволяет контролировать злоупотребление функцией и увеличивает достоверность отзывов.

**3.1.2.3 Вкладка «Сканер» (QR / NFC)**



Рисунок 6. Вкладка «Сканер».

Вкладка «Сканер», реализованная в файле Scanner.kt, предоставляет пользователю два способа считывания информации о других пользователях — **через QR-код** и **с помощью NFC-меток**. Это расширяет варианты взаимодействия между пользователями, упрощая идентификацию и доступ к профилю другого участника.

***Назначение вкладки:***

* Сканирование QR-кодов с камеры устройства;
* Считывание информации с NFC-меток;
* Переход к профилю другого пользователя (ScannigProfile) после успешного сканирования если он находится рядом.

***Интерфейс вкладки***

Интерфейс состоит из двух кнопок, позволяющих выбрать способ сканирования:

* «Отсканировать через NFC»;
* «Отсканировать QR-код».

После выбора — отображается соответствующий модуль:

* **NFC** — выводит текст с метки (или сообщение об ошибке);
* **QR** — запускается камера и анализируется изображение с помощью библиотеки ML Kit Barcode.

***Работа с NFC***

При активации режима "ScanNFC" выполняется:

* инициализация NFC через класс NFCManager;
* считывание текста с метки, переданной через Intent;
* отображение результата на экране.

Если метка не распознана, выводится сообщение: «Не удалось прочитать текст с метки».

***Работа с QR-кодами***

При активации "scanQR":

1. Запрашивается разрешение на использование камеры (CAMERA);
2. После получения — запускается QRCodeScannerView;
3. Используется CameraX и ML Kit для захвата и анализа изображений в реальном времени;
4. Если штрихкод содержит текст:
   * вызывается handleBarcode();
   * извлекается текстовое значение;
   * происходит переход к экрану ScannigProfile, где отображаются данные сканируемого пользователя.

val intent = Intent(context, ScannigProfile::class.java).apply {

putExtra("text", text)

}

activityResultLauncher.launch(intent)

Флаг isScanningActive предотвращает повторный запуск перехода при многократном срабатывании.

***Безопасность и UX***

* Используются проверки на разрешения доступа к камере;
* Обработка ошибок вынесена в addOnCompleteListener;
* Интерфейс остаётся отзывчивым за счёт LaunchedEffect и управления состоянием через remember.

### 3.1.3 ViewModel и управление состоянием

В клиентской части приложения используется модель представления (MyViewModel), реализующая паттерн **MVVM** (Model–View–ViewModel). Компонент MyViewModel служит мостом между пользовательским интерфейсом и фоновыми сервисами, обрабатывает полученные от сервера данные и предоставляет их в реактивной форме интерфейсу Jetpack Compose.

***Основные задачи ViewModel:***

* централизованное управление состоянием пользовательского интерфейса;
* отправка SQL-команд на сервер через ActiveService;
* приём и декодирование JSON-ответов от сервера;
* обновление StateFlow, на которые подписаны элементы UI;
* управление состояниями загрузки и ошибок.

***Используемые состояния (StateFlow):***

1. \_reviewsFlow — список всех отзывов;
2. \_reviewsFlow5, \_reviewsFlow5From — последние 5 полученных и отправленных отзывов;
3. \_infoPeopleOnMap — список пользователей, отображаемых на карте;
4. \_avgRating — средняя оценка пользователя;
5. \_infoAboutUser — информация по выбранному пользователю (имя, время входа);
6. \_isLoading, \_isLoadingInfo — индикаторы фоновой загрузки.

Все эти состояния доступны в UI в виде StateFlow, что обеспечивает автоматическую реакцию интерфейса на изменения данных.

***Отправка SQL-команд***

Для отправки SQL-запросов реализован метод:

fun sendCommand(command: String, typeSQL: String)

Он перенаправляет запрос через activeService.sendCommandFromActivity(...), формируя JSON-команду, которую принимает сервер.

Примеры:

* Получение всех отзывов: fetchAllReviews();
* Получение пользователей на карте: getPeopleOnMap();
* Добавление отзыва: sendingReview().

***Обработка ответов сервера***

В методе subscribeToService() осуществляется подписка на поток responseFlow, формируемый ActiveService. Каждый ответ:

* парсится в JSONObject;
* анализируется по полю typeSQL;
* в зависимости от типа — десериализуется в список Review, InfoPeopleOnMap или InfoAboutUser;
* соответствующее StateFlow обновляется и отображается в интерфейсе.

val jsonObject = JSONObject(serverResponse)

val typeSQL = jsonObject.getString("typeSQL")

when (typeSQL) {

"allReviews" -> \_reviewsFlow.value = ...

"getPeopleOnMap" -> \_infoPeopleOnMap.value = ...

}

Для десериализации используется библиотека **Gson** и TypeToken.

***Управление ошибками и завершением загрузки***

Каждый блок обработки обёрнут в try-catch, что обеспечивает:

* защиту от сбоев при парсинге;
* очистку состояний и завершение анимации загрузки;
* вывод логов об ошибках через Log.e.

Индикаторы \_isLoading и \_isLoadingInfo используются для отображения прогресс-бара в пользовательском интерфейсе.

### 3.1.4 Сервис ActiveService и TCP-соединение

Фоновый сервис ActiveService играет ключевую роль в обеспечении взаимодействия клиента с сервером через **TCP-сокет**, реализуя двустороннюю асинхронную коммуникацию. Этот компонент также отвечает за **обновление координат** пользователя и отправку их на сервер, а также за **передачу бинарных данных**, например, аватара.

О**сновные задачи ActiveService:**

* установление TCP-соединения с сервером;
* формирование и отправка JSON-запросов;
* приём и парсинг ответов;
* передача координат пользователя;
* отправка изображений (аватаров) в бинарном формате;
* взаимодействие с ViewModel через SharedFlow.

***Установка соединения***

При старте сервиса (onCreate()) инициализируется подключение к серверу:

socket = Socket(serverIp, serverPort)

После установления соединения инициализируются потоки OutputStreamWriter, InputStreamReader, DataInputStream, DataOutputStream.

В случае ошибки подключения отправляется широковещательное сообщение с ошибкой (ACTION\_CONNECTION\_FAILED), что может быть обработано активностью.

***Отправка и приём данных***

Для отправки запроса из Activity используется метод:

fun sendCommandFromActivity(command: String, typeSql: String, avatar: ByteArray?)

Он вызывает sendToServer(...), который:

1. Проверяет сокет;
2. Формирует JSON-заголовок или объект RequestData;
3. Передаёт заголовок и/или тело (в случае с аватаром — бинарные байты);
4. Принимает и парсит ответ от сервера;
5. Эмитирует результат во SharedFlow<String> (responseFlow), на который подписана ViewModel.

***Пример обмена JSON:***

val jsonStr = Gson().toJson(RequestData("sql", typeSql, command))

dataOut!!.writeInt(jsonStr.length)

dataOut!!.write(jsonStr.toByteArray(Charsets.UTF\_8))

dataOut!!.flush()

Ответ считывается:

val respLen = dataIn!!.readInt()

val respBuf = ByteArray(respLen)

dataIn!!.readFully(respBuf)

***Отправка аватара***

Если тип запроса — send\_avatar, используется формат:

1. Заголовок JSON: { "type": "upload\_avatar", "user\_id": ..., "size": ... }
2. Отправка: [4 байта длины заголовка][JSON][бинарный файл]
3. Приём: [4 байта длины][JSON-ответ]

Этот подход позволяет серверу корректно интерпретировать как заголовок, так и необработанные данные изображения.

***Координаты и геолокация***

ActiveService использует FusedLocationProviderClient и LocationCallback для получения текущих координат пользователя. Координаты отправляются на сервер SQL-запросом:

UPDATE myusers SET latitude = ..., longitude = ..., last\_login = CURRENT\_TIMESTAMP WHERE user\_id = ...

Запрос отправляется с помощью метода sendDataSafely(...).

***Привязка и взаимодействие***

Сервис привязывается к активити через Binder, позволяя напрямую вызывать публичные методы:

val activeService = serviceBinder.getService()

activeService.sendCommandFromActivity(...)

ViewModel подписывается на responseFlow, получая все сообщения от сервера в режиме реального времени.

### 3.1.5 Сервис BackgroundServiceLocation и отправка координат

Служба BackgroundServiceLocation реализует работу в фоновом режиме и запускается даже при закрытом интерфейсе приложения. Её основная задача — **периодическая отправка текущего местоположения пользователя на сервер**, в том числе с использованием **Google Fused Location Provider** и резервных механизмов (GPS, сеть). Это позволяет обеспечить актуальность координат даже вне активной сессии.

***Основные функции службы:***

* запуск в foreground-режиме с уведомлением;
* получение местоположения с использованием FusedLocationClient;
* отправка координат на сервер каждые 60 секунд;
* использование нескольких источников геолокации (последнее известное, GPS, сеть);
* устойчивость к отказам соединения.

***Foreground-сервис***

Служба реализована как foreground-процесс с системным уведомлением, что гарантирует её работу даже при переходе приложения в фон. При старте создаётся NotificationChannel и отображается уведомление:

startForeground(1, notification)

Это предотвращает автоматическое завершение службы системой Android.

***Получение координат***

При каждой итерации фонового потока выполняется:

1. Попытка получить последнее местоположение через:

fusedLocationClient.lastLocation

1. Координаты отправляются на сервер в виде текста через TCP-соединение.

Функция getLocationSynchronously()использует HandlerThread для синхронного ожидания обновлений местоположения с таймаутом до 90 секунд.

***Отправка данных на сервер***

Подключение к серверу осуществляется через сокет:

val socket = Socket()

socket.connect(InetSocketAddress(serverIp, serverPort), 20000)

Затем отправляется последовательность строк:

* "offline send" — пометка типа соединения;
* текущее местоположение;
* координаты от GPS и от сети;
* завершающий символ .\n для обозначения конца пакета.

Передача выполняется через OutputStream, соединение закрывается корректно после завершения цикла.

***Обработка ошибок***

Если соединение не удаётся установить, информация логируется:

Log.e(TAG, "Error in backgroundService: ${e.message}")

и фоновый поток продолжает работу, пытаясь подключиться на следующей итерации.

***Использование в системе***

Служба запускается один раз при первом вызове и работает непрерывно в фоне. Это обеспечивает **автоматическое обновление координат** пользователя даже без активного взаимодействия с приложением, повышая надёжность и актуальность данных на карте.

## 3.2 Серверная часть

Серверная часть приложения реализована на языке программирования **C++** и предназначена для обработки всех поступающих запросов от мобильного клиента, выполнения SQL-команд и взаимодействия с базой данных PostgreSQL. Обмен с клиентом организован по протоколу **TCP** с сериализацией данных в формате **JSON**. Сервер функционирует в непрерывном режиме и обслуживает входящие соединения, выполняя роль промежуточного звена между приложением и базой данных.

***Основные функции сервера:***

* приём и разбор JSON-команд от клиента;
* выполнение SQL-запросов к базе данных;
* обработка авторизации пользователей;
* регистрация новых пользователей;
* приём и сохранение бинарных данных (аватаров);
* обновление координат пользователей;
* формирование и отправка структурированных JSON-ответов.

***Основные команды***

* **sql\_login** — авторизация: проверка логина и пароля в таблице myusers;
* **sql** — произвольные SQL-запросы (например, добавление отзыва, выборка данных);
* **upload\_avatar** — приём аватара в бинарном формате;
* **offline send** — обновление координат в фоне от BackgroundService.

Формат команды:

{

"type": "sql",

"typeSql": "allReviews",

"command": "SELECT ...;"

}

***Выполнение SQL***

Сервер подключается к базе данных PostgreSQL с использованием библиотеки libpq. Для каждого запроса:

1. Устанавливается соединение:

PGconn \*conn = PQconnectdb(...)

1. Выполняется SQL-команда:

PGresult \*res = PQexec(conn, query.c\_str());

1. Результаты сериализуются в JSON:

for (int i = 0; i < rows; ++i)

for (int j = 0; j < cols; ++j)

result[row][field] = PQgetvalue(res, i, j);

***Приём аватаров***

Для передачи изображения используется двухэтапная структура:

1. Заголовок JSON с типом upload\_avatar, user\_id и размером;
2. Бинарные данные (в байтах).

Сервер принимает данные по частям, сохраняет аватар в файл.

***Надёжность и устойчивость***

* Сервер работает в бесконечном цикле и автоматически принимает новые соединения;
* При ошибках логируются сообщения и соединение завершается;
* Бинарные данные читаются с учётом длины, указанной в заголовке.

## 3.3 База данных

База данных играет ключевую роль в функционировании системы, обеспечивая хранение информации о пользователях, их местоположении, а также оставленных и полученных отзывах. В качестве системы управления базами данных используется **PostgreSQL 15**, обладающая высокой производительностью, расширяемостью и поддержкой работы с бинарными данными.

### 3.3.1 Структура базы данных

Проект использует две основные таблицы: myusers и ratings. Они логически связаны между собой через внешние ключи, что позволяет отслеживать, кто оставил отзыв и кому он адресован.

***Таблица myusers***

Хранит сведения о зарегистрированных пользователях:

Таблица 1. Таблица myusers

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| user\_id | SERIAL PRIMARY KEY | Уникальный идентификатор пользователя |
| login | TEXT UNIQUE | Логин (уникальный) |
| pass | TEXT | Хеш пароля |
| username | TEXT | Имя пользователя |
| latitude | DOUBLE PRECISION | Географическая широта |
| longitude | DOUBLE PRECISION | Географическая долгота |
| last\_login | TIMESTAMP | Время последнего входа |
| visible | BOOLEAN | Флаг видимости пользователя |

***Таблица ratings***

Содержит отзывы между пользователями:

Таблица 2. Таблица ratings

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| id | SERIAL PRIMARY KEY | Уникальный идентификатор отзыва |
| sender\_id | INTEGER | ID пользователя, оставившего отзыв |
| receiver\_id | INTEGER | ID пользователя, получившего отзыв |
| rating | INTEGER | Оценка по 5-балльной шкале |
| review\_text | TEXT | Текст отзыва |
| created\_at | TIMESTAMP | Дата и время создания |

Оба поля sender\_id и receiver\_id являются внешними ключами, ссылающимися на myusers(user\_id).

### 3.3.2 Связи и ограничения

* **Уникальность логинов** обеспечивается ограничением UNIQUE(login);
* **Целостность данных** реализуется через каскадное удаление и внешние ключи;
* **Согласованность оценок** достигается типом INTEGER и проверкой допустимых значений (1–5) на стороне клиента;
* **Обновление координат** происходит при каждом входе и/или запуске фоновой службы.

### 3.3.3 Расширение

С учётом дальнейшего развития проекта структура БД позволяет:

* добавить таблицу avatars для хранения изображений (чтобы было не одно, а галереей);
* расширить таблицу отзывов полем emotion\_tag (например, «позитив», «нейтрально»);
* создать таблицу friendships для связи пользователей в виде социальных связей

# 4. Документирование

Документирование включает в себя описание порядка запуска, настройки и использования системы как со стороны разработчика, так и конечного пользователя. Также представлены инструкции по установке, рекомендации по тестированию и сведения о точности работы приложения.

## 4.1 Установка и запуск

### 4.1.1 Серверная часть

***Требования:***

* ОС: Linux (желательно Ubuntu/Debian)
* Компилятор: g++ с поддержкой C++11
* Библиотеки: libpq (PostgreSQL), nlohmann/json, libcurl (по необходимости)

***Сборка:***

g++ -std=c++11 -Wall -I/usr/local/include/pgsql -I/home/user/libs -o server server.cpp -lpq -lcurl

***Запуск сервера:***

./server

Сервер запускается на порту 44021 и ожидает подключения от клиента.

### 4.1.2 Клиентская часть (Android-приложение)

Мобильное приложение предназначено для установки на устройства с операционной системой **Android**. Реализовано на языке **Kotlin** с использованием библиотеки **Jetpack Compose** для построения интерфейса. Поддерживается как ручная установка, так и установка через Android Studio при необходимости.

***Совместимость***

* **Минимальная версия Android:** 7.0 (API 24)
* **Целевая версия:** Android 14 (API 34)
* **Поддержка архитектур:** ARMv7, ARM64
* **Требования к устройству:**
  + наличие доступа к интернету (Wi-Fi или мобильная сеть);
  + разрешение на доступ к геолокации;
  + (опционально) наличие NFC-модуля.

***Установка***

Для установки на устройство:

1. Скопировать файл .apk на Android-устройство (через USB, Bluetooth или облако);
2. Разрешить установку из неизвестных источников;
3. Открыть файл и подтвердить установку;
4. Приложение будет доступно в меню приложений под названием **"Diplom"**.

***Структура приложения:***

* **ID приложения:** com.example.diplom
* **Версия:** 1.0 (versionCode = 1, versionName = "1.0")
* **Сборка:** release без минификации;
* **Язык Java/Kotlin:** JVM 1.8;
* **UI:** Compose (kotlinCompilerExtensionVersion = 1.5.1)

***Дополнительные возможности***

* **Карты:** Используются карты Яндекс. Требуется активное подключение к интернету и включённая геолокация;
* **Фоновые службы:** Координаты пользователя продолжают передаваться даже при выключенном экране;
* **NFC:** Если устройство поддерживает NFC, можно обмениваться профилями через метки.

## 4.2 Работа с системой

Данный раздел описывает взаимодействие пользователя с приложением в реальных условиях эксплуатации. Рассматриваются все ключевые сценарии работы: авторизация, использование экрана профиля, карты, системы отзывов, а также механизмов сканирования и обмена данными.

### 4.2.1 Авторизация

При первом запуске приложения пользователь автоматически попадает на экран авторизации. На этом этапе приложение также запрашивает разрешения на доступ к данным о местоположении устройства. Всплывающее системное окно содержит три варианта:

* **«При использовании приложения»**,
* **«Только в этот раз»**,
* **«Запретить»**.

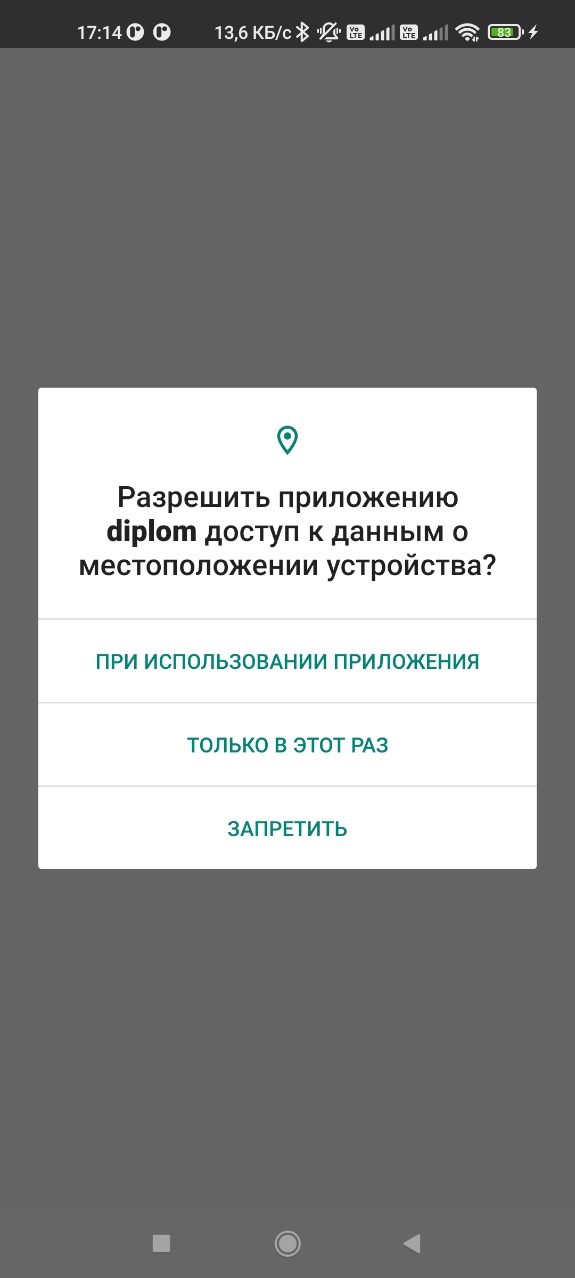


Рисунок 7. Разрешение к данным о местоположении.

Данное разрешение необходимо для корректной работы функционала, связанного с отображением карты, координат и фоновыми службами геолокации. При отказе от предоставления доступа часть функций будет недоступна, о чём пользователь будет проинформирован в интерфейсе.

После предоставления разрешения пользователь вводит логин и пароль. Ввод осуществляется через стандартные поля TextField, при этом ввод пароля скрыт для обеспечения конфиденциальности.

***Алгоритм авторизации:***

1. Ввод логина и пароля.
2. Отправка запроса на сервер с помощью TCP-соединения.
3. Сервер выполняет проверку данных в таблице myusers.
4. В зависимости от результата:
   * при успехе — переход на главный экран (MainActivity);
   * при неудаче — отображается сообщение об ошибке (например, "Неверный логин или пароль").

***Безопасность:***

* Пароль передаётся один раз в виде строки и не сохраняется на устройстве.
* При повторных входах пользователь должен заново авторизоваться.

### 4.2.2 Экран профиля

Экран профиля предоставляет пользователю доступ к его личной информации и истории взаимодействия с системой.

***Функции:***

* **Отображение аватара**:
  + Загружается из локального хранилища (SharedPreferences);
  + Пользователь может выбрать новое изображение из галереи;
  + Аватар передаётся на сервер в виде бинарного потока через JSON-заголовок upload\_avatar.
* **Средняя оценка**:
  + Рассчитывается на основе отзывов в таблице ratings;
  + Отображается в виде набора звёзд (от 0 до 5, с шагом 0.5).
* **Отзывы**:
  + Отображаются по две вкладки: «Полученные» и «Отправленные»;
  + Загружаются последние 5 записей по каждому направлению;
  + По клику на звёзды открывается окно со всеми отзывами, сортируемыми по дате и рейтингу.
* **Поделиться профилем**:
  + По нажатию открывается экран ScreenRepost;
  + Пользователь может сгенерировать ссылку, QR-код или отправить данные через NFC.

### 4.2.3 Экран карты

Экран отображает интерактивную карту, на которой показано местоположение пользователя и других участников.

***Функциональность:***

* **Местоположение пользователя**:
  + Определяется через Google Fused Location Provider;
  + Обновляется при входе в приложение и каждые 60 секунд фоновым сервисом;
  + Отображается меткой на карте с иконкой аватара.
* **Пользователи рядом**:
  + Координаты других участников запрашиваются с сервера;
  + Отображаются метками на карте;
  + При клике на метку — открывается диалог с именем, рейтингом и возможностью оставить отзыв.
* **Отправка отзывов**:
  + Разрешена, если пользователь включил режим видимости не менее 30 минут назад;
  + Открывается диалоговое окно с полем для текста и выбором рейтинга;
  + Отзыв сохраняется в базе данных через SQL-запрос.
* **Видимость**:
  + Управляется переключателем на экране;
  + При отключении пользователь становится невидимым и не может оставлять отзывы в течение 30 минут;
  + Таймер и состояние сохраняются в локальном хранилище устройства.

### 4.2.4 Сканирование

Раздел «Сканер» предоставляет два режима для идентификации других пользователей:

***NFC***

* Пользователь может приложить устройство к метке (или другому устройству);
* Из NFC-тега извлекается ID пользователя;
* При успешном считывании — происходит переход на экран ScannigProfile, где отображаются данные пользователя.

***QR-код***

При первом запуске нужно разрешить использование камеры для работы с QR-кодом.

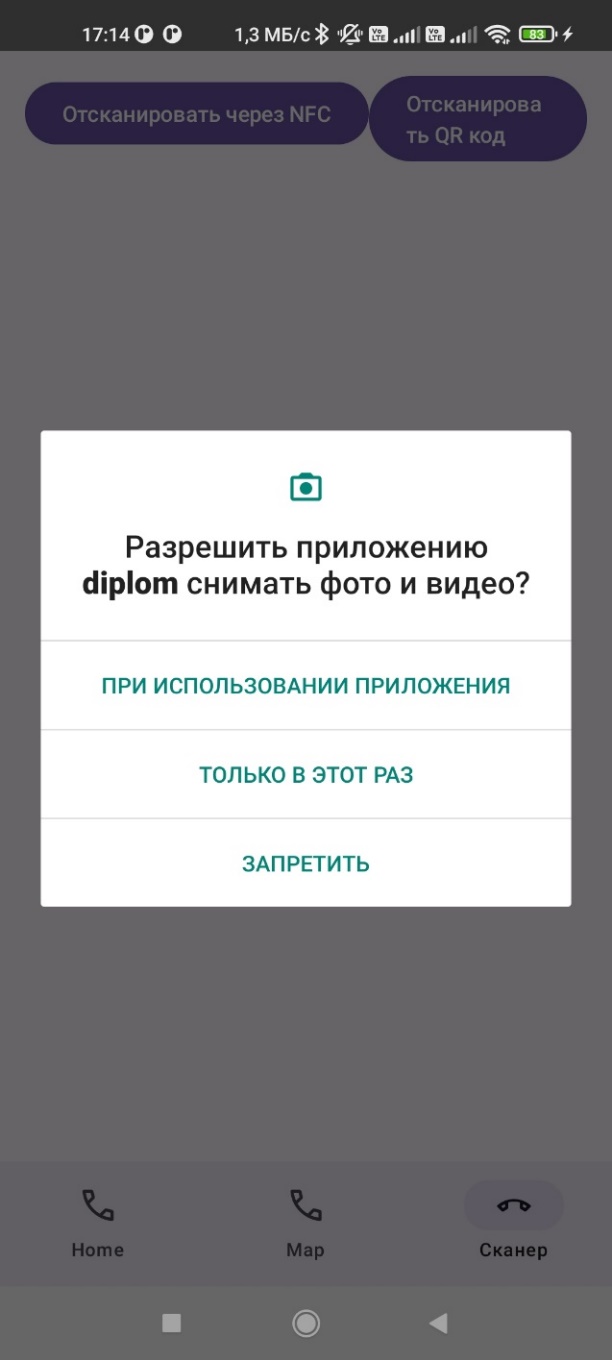


Рисунок 8. Разрешение на доступ к камере.

* Камера активируется для захвата изображения;
* С помощью библиотеки ML Kit распознаётся QR-код;
* Если QR-код содержит допустимую информацию (например, ID), выполняется переход к профилю пользователя;
* Повторное сканирование блокируется до закрытия экрана, предотвращая множественные переходы.

## 4.3 Тестирование

Система прошла ручное тестирование на реальных устройствах под управлением Android 11–14. Основное внимание уделялось стабильности подключения, корректности отображения информации, обработке ошибок и пользовательскому опыту. В процессе тестирования проверялись как успешные сценарии, так и реакции на ошибочные действия пользователя.

### 4.3.1 Методика тестирования

Тестирование проводилось по следующей схеме:

* Проверка доступности функций при различных состояниях соединения;
* Проверка валидации входных данных;
* Проверка откликов от сервера (обработка JSON-ответов);
* Тестирование фоновой передачи координат;
* Анализ отображения отзывов и профилей.

Каждый сценарий сопровождался регистрацией лога, визуальной проверкой интерфейса и сравнением с ожидаемым результатом.

### 4.3.2 Примеры сценариев

***Авторизация — неверный пароль***

Пользователь вводит корректный логин, но ошибается в пароле. Результат: появляется сообщение «Неверный пароль», вход не выполняется.

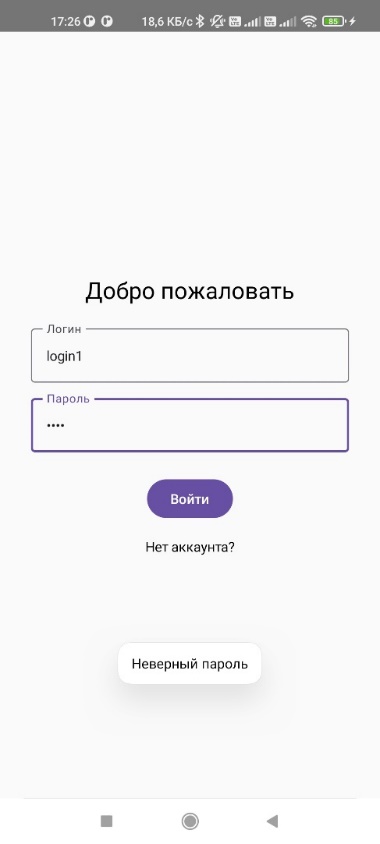


Рисунок 9. Проверка пароля.

***Регистрация — несовпадение паролей***

Пользователь указывает разные значения в поле «Пароль» и «Пароль ещё раз». Результат: отображается сообщение «Пароли не совпадают!», регистрация не выполняется.

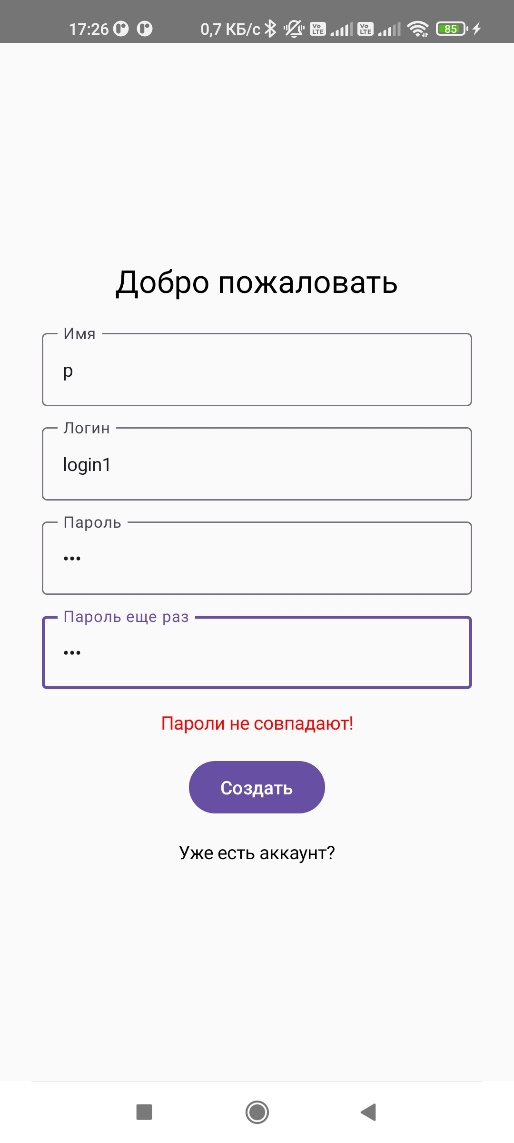


Рисунок 10. Проверка паролей на совпадение при регистрации.

***Регистрация — логин занят***

Пользователь вводит логин, уже занятый другим пользователем. Результат: появляется сообщение This login is already occupied.

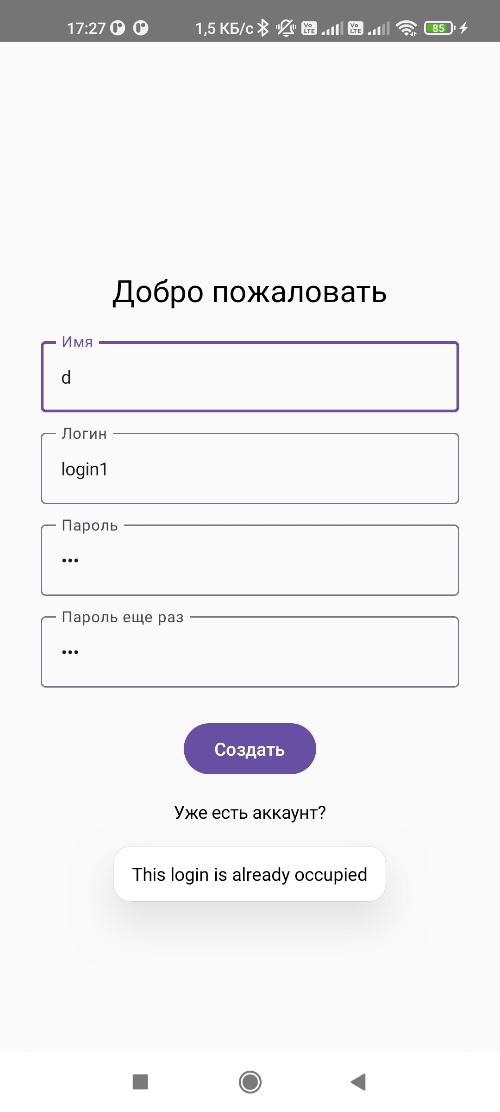


Рисунок 11. Проверка на занятость логина.

***Отзыв — неполный ввод***

Пользователь выбирает другого участника на карте, но не указывает оценку и/или текст. Результат: отображаются сообщения «Выберите рейтинг!» и «Отзыв не должен быть пустым!»

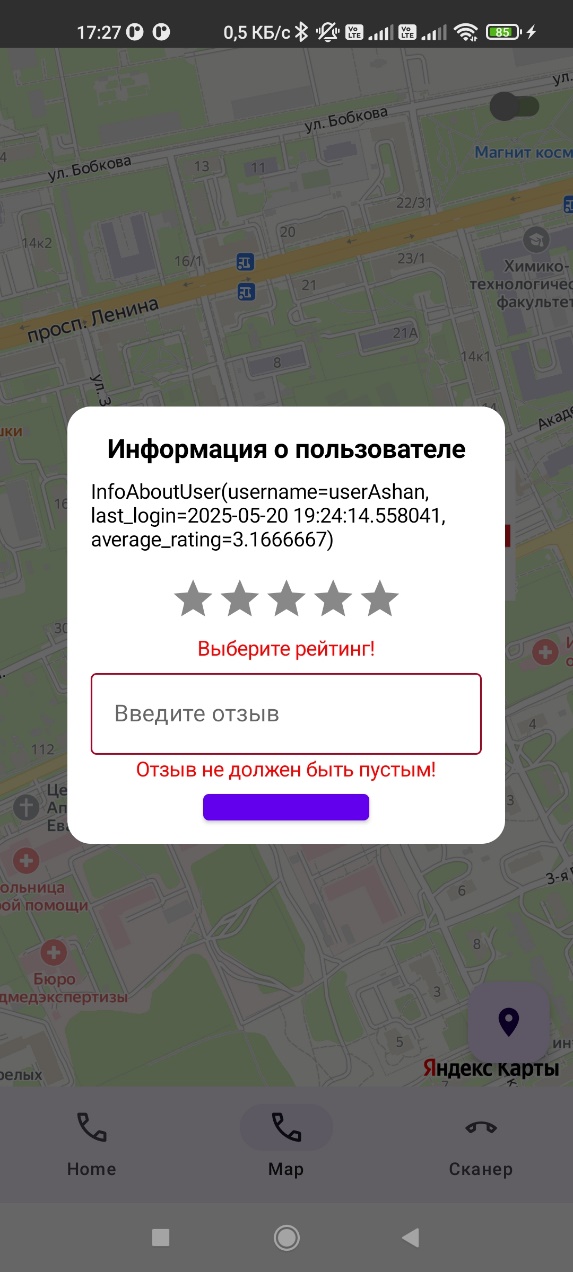


Рисунок 12. Проверка на заполненность полей при отправке отзыва.

***Отправка геолокации в фоне***

Даже когда приложение выключено, раз в промежуток времени обновляется позиция, чтобы другие пользователи могли поставить отзыв.



Рисунок 13. Прием данных на сервере.

### 4.3.3 Среда тестирования

* Устройства: смартфоны Xiaomi Mi 10 T Pro и Samsung A52;
* ОС: Android 10, 11;
* Сетевое подключение: Wi-Fi, 4G LTE;
* Сервер: Linux (PostgreSQL 15, TCP-сервер C++), подключение через статический IP.

### 4.3.4 Общие результаты

Таблица 3. Результаты по тестированию

|  |  |
| --- | --- |
| **Компонент** | **Результат** |
| Авторизация | Успешно, ошибки обрабатываются |
| Регистрация | Валидация работает корректно |
| Отображение карты и меток | Без сбоев |
| Отзывы и рейтинги | Отправка и получение стабильны |
| Работа NFC и QR | Сканирование корректно |
| Фоновые службы | Координаты обновляются каждые 60 сек |

# Заключение

В рамках выпускной квалификационной работы была выполнена разработка мобильного Android-приложения для сбора и отображения отзывов между пользователями. Система реализована в виде клиент-серверной архитектуры, включающей мобильный клиент на языке Kotlin с использованием Jetpack Compose и серверную часть, написанную на языке C++ с использованием библиотеки libpq для взаимодействия с базой данных PostgreSQL.

В процессе реализации были выполнены следующие задачи:

* проведён анализ существующих решений и обоснована актуальность проекта;
* спроектирована архитектура приложения, включающая основные модули: авторизация, профиль, карта, система отзывов, сканер QR/NFC;
* разработана серверная часть, обеспечивающая обработку TCP-запросов, выполнение SQL-команд и приём бинарных данных (аватаров);
* создана база данных, содержащая таблицы пользователей и отзывов, с обеспечением ограничений целостности;
* реализована клиентская часть с поддержкой фоновых служб, геолокации, загрузки изображений, отображения карты и отзывов;
* выполнено ручное тестирование на реальных устройствах, подтверждающее стабильность и корректность работы всех компонентов.

Разработанное приложение позволяет пользователям:

* оставлять и получать отзывы;
* оценивать других участников по 5-балльной шкале;
* управлять своей видимостью и профилем;
* обмениваться данными с помощью QR-кодов и NFC;
* отслеживать активность через карту с актуальными координатами.

Практическая значимость работы заключается в возможности применения системы в различных социально ориентированных сферах, таких как студенческие сообщества, аренда жилья, оказание частных услуг и локальные профессиональные сети.

Реализация проекта показала, что даже в условиях ограниченных ресурсов возможно создание надёжного, безопасного и удобного сервиса на базе открытых технологий, с полным циклом от проектирования до тестирования и внедрения.

# Литература

* 1. Отзывы // BlaBlaCar URL: https://support.blablacar.com/s/article/Отзывы-1729197048392?language=ru (дата обращения: 25.04.2025).
  2. Yandex MapKit и NaviKit SDK // Яндекс URL: https://yandex.ru/maps-api/docs/mapkit/index.html (дата обращения: 26.04.2025).
  3. Build better apps faster with Jetpack Compose // Developers URL: https://developer.android.com/compose (дата обращения: 26.04.2025).
  4. Обзор ViewModel // Developers URL: https://developer.android.com/topic/libraries/architecture/viewmodel?hl=ru (дата обращения: 26.04.2025).
  5. Coroutines // Kotlin URL: https://kotlinlang.org/docs/coroutines-overview.html#how-to-start (дата обращения: 26.04.2025).
  6. json // GitHub URL: https://github.com/nlohmann/json (дата обращения: 27.04.2025).
  7. СУБД PostgreSQL: принцип работы, преимущества и недостатки // ServerFlow URL: https://serverflow.ru/blog/stati/subd-postgresql-printsip-raboty-preimushchestva-i-nedostatki/ (дата обращения: 28.04.2025).