МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки / специальности)

Мобильные и сетевые технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка мобильного приложения для обмена геолокацией и мессенджинга с функцией управления пользователями»

Обучающийся У. С. Сижотхонов _____
(Инициалы Фамилия) (личная подпись)

Руководитель к.п.н., доцент, Е. А. Ерофеева
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Выпускная квалификационная работа выполняется на тему «Разработка мобильного приложения для обмена геолокацией и мессенджинга с функцией управления пользователями». Актуальность темы обоснована потребностью в надежных решениях для обмена местоположением в реальном времени и текстовыми сообщениями, а также необходимостью эффективного управления пользователями.

Задачей исследования является создание программного обеспечения для мобильных устройств, обеспечивающий передачу геолокаций, обмен сообщениями и управление пользователями. Реализация такого решения направлена на устранение существующих ограничений в организации взаимодействия данных.

В результате работы создано и протестировано приложение, позволяющее пользователям:

– обмениваться геолокацией в реальном времени,
– отправлять текстовые сообщения,
– управлять списком пользователей и групп.

Тестирование подтвердило работоспособность всех ключевых функций приложения. В завершение представлено готовое решение, соответствующее поставленным задачам.

Структура работы состоит из введения, трех глав, заключения и списка использованной литературы. Работа выполнена в объеме 48 страниц и включает в себя 35 иллюстраций и 1 таблицу.

Abstract

Graduation qualification work is performed on the topic “Development of mobile application for geolocation exchange and messaging with the function of user management”. The diploma work is executed in the volume of 48 pages and includes 35 illustrations and 1 table. The structure of the work consists of an introduction, three chapters, conclusion and list of used literature.

The relevance of the topic is justified by the need for reliable solutions for real-time location and text messaging, as well as the need for effective user management.

The task of the research is to create software for mobile devices that provides geolocation transfer, messaging and user management. The implementation of such a solution is aimed at eliminating the existing limitations in the organization of data interaction.

As a result of the work, an application that allows users to:

- exchange geolocation in real time,
- send text messages,
- manage the list of users and groups.

Testing confirmed the operability of all key features of the application. In the end, a ready-made solution that meets the set objectives is presented.

Оглавление

Введение

В современном мире мобильные приложения, совмещающие функции геолокации и мессенджинга, становятся все более востребованными. Они применяются в различных сферах: от логистики и управления персоналом до социальных сетей и образовательных проектов. Особую ценность такие решения представляют для учебных заведений, где важно оперативно отслеживать местоположение студентов, координировать их перемещение и обеспечивать быструю коммуникацию между участниками групп.

Существующие аналоги часто не предоставляют удобного интерфейса для управления группами пользователей, что усложняет процесс взаимодействия. Разработка специализированного приложения, объединяющего функции геолокации и мессенджинга, позволит решить эту проблему, повысив эффективность коммуникации и контроля в учебных и рабочих процессах.

Объектом исследования является процесс создания мобильного приложения, интегрирующего функционал обмена местоположения, текстовой коммуникации и управлением пользовательских групп.

Предметом исследования являются современные подходы и технологические решения, используемые при разработке мобильных приложений, в частности:

– визуализация геопозиции пользователей на интерактивной карте;

– механизмы организации группового взаимодействия;

– реализация системы обмена сообщениями в реальном времени;

– модули регистрации, аутентификации и управления учетными записями.

Цель работы - создание мобильного приложения с возможностью мониторинга текущего местоположения пользователей, обмена текстовыми сообщениями и управлением пользовательских групп.

Для достижения цели необходимо решить следующие задачи:

– Провести анализ существующих аналогов и сформулировать технические требования;

– Разработать архитектурное решение с продуманной системой навигации;

– Внедрить хранилище данных для информации о пользователях, группах и сообщениях;

– Осуществить тестирование функциональности и удобства использования приложения.

Структура работы включает аннотацию, введение, три главы, заключительную часть и перечень использованных источников.

Разработанное приложение предоставляет удобный инструмент для координации групп пользователей, сочетая в себе возможности геолокации и мессенджинга, что делает его актуальным для образовательных учреждений и малых предприятий. Кроме того, разрабатываемое приложение может быть адаптировано под различные сценарии использования, что расширяет его потенциальную аудиторию. Например, в образовательной среде преподаватели смогут быстро находить студентов на территории кампуса, организовывать встречи и рассылать важные уведомления. Для бизнеса такое решение упростит контроль за передвижением сотрудников, особенно в сферах логистики, доставки или сервисного обслуживания, где важно отслеживать местоположение работников в реальном времени.

Еще одним преимуществом приложения является его модульная архитектура, позволяющая легко масштабировать функционал в будущем. Например, можно добавить интеграцию с календарями, системой оповещений о важных событиях или даже внедрить аналитику перемещений для оптимизации рабочих процессов. Это делает приложение не только удобным, но и перспективным с точки зрения дальнейшего развития. Таким образом, сочетание геолокации, мессенджинга и гибких настроек управления группами обеспечивает универсальность решения, отвечающего потребностям разных сфер деятельности.

Глава 1. Сущность объекта исследования

1.1 Краткая характеристика компании

Выпускная квалификационная работа выполнялась на базе компании IT-компании "ВорлдИнтерТех РУС", которая занимается цифровизацией бизнеса, созданием облачных сервисов и программных решений, а также интернет-маркетингом.

Программное обеспечение, создаваемое компанией "ВорлдИнтерТех РУС", служит связующим звеном между компаниями и потребителями, предоставляя разнообразные сервисы, такие как услуги, аудит и маркетинг в сфере оптовых и розничных продаж.

Разработчик программного обеспечения занимает функциональную позицию в организации, отвечая за создание ПО, основываясь на техническом задания. Один из основных подразделений — это отдел разработки. Этот отдел занимается составлением технического задания на основе требований заказчика и разработкой программного обеспечения в соответствии с утвержденным ТЗ.

Организационная структура “ВорлдИнтерТех РУС” представлена на рисунке 1:

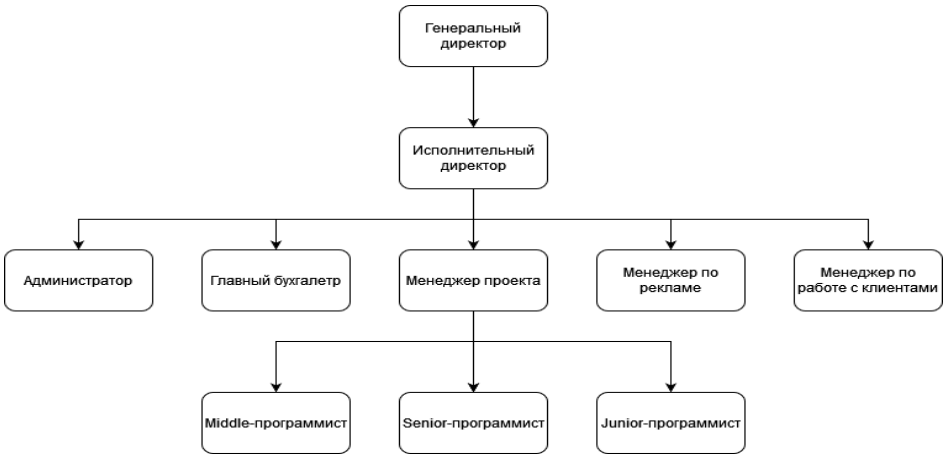

Рисунок 1 – Организационная структура компании

В подразделении отдела разработки входят сотрудники:

- Менеджер проекта
- Senior-программист
- Middle-программист
- Junior-программист

Функции сотрудников представлены на рисунке 2:

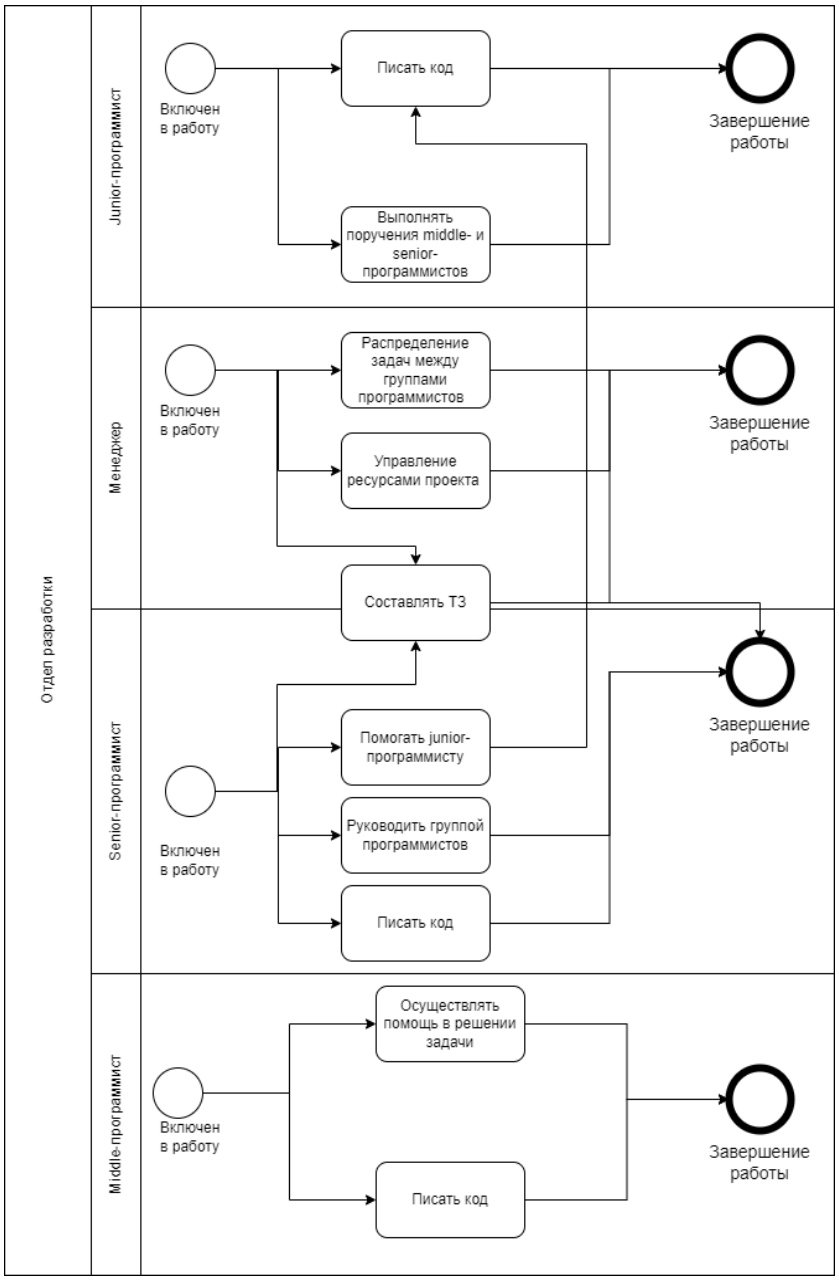

Рисунок 2 – функции, выполняемые сотрудниками отдела разработки

Рассмотрим роли и функции сотрудников отдела разработки, которые представлены на рисунке 2:

- Junior-программист. Его деятельность ограничена из-за недостатка опыта и знаний в области разработки программного обеспечения. Он занимается выполнением небольших задач по поддержке и улучшению уже существующих сервисов, а также написанием и выполнением тестов для ПО. Junior-программист может также брать на себя более сложные задания с целью повышения своей квалификации. Обычно он сильно зависит от более опытных коллег из-за своей неопытности.
- Middle-программист. Это самостоятельный специалист, обладающий достаточным опытом в разработке программного обеспечения, что позволяет ему справляться с более сложными задачами. Middle-программист также курирует младших коллег и оказывает им помощь в решении возникающих проблем
- Senior-программист. Это высококвалифицированный специалист с обширным опытом разработки, который хорошо разбирается в своей области. Он отвечает за исправление сложных ошибок, разработку новых сервисов и программных продуктов, а также проверку кода, написанного младшими программистами.
- Менеджер проекта: Этот специалист управляет ресурсами проекта, совместно с Senior-программистом устанавливает сроки выполнения задач и разрабатывает технические задания, взаимодействуя с заказчиком.

1.2 Определение требований к разрабатываемому приложению

Для реализации программы, были определены функциональные и нефункциональные требования. Задачей было создать приложение для отображения геолокации добавленных пользователей в реальном времени на цифровой географической карте, а также реализация мессенджера для

пользователей. Исходя из этого были выделены следующие функциональные требования:

- Возможность создавать и удалять аккаунт.
- Пользователь должен иметь возможность видеть текущее местоположение групп в реальном времени.
- Пользователь должен иметь возможность видеть свое текущее местоположение.
- Пользователи должны иметь возможность обмениваться текстовыми сообщениями.
- Пользователи должны иметь возможность создавать и редактировать группы
- Пользователи должны иметь возможность совершать телефонные вызовы.

Нефункциональные требования включают следующие пункты:

- Система должна иметь быстрый и отзывчивый интерфейс, чтобы обеспечивать комфортное взаимодействие с пользователями.
- Система должна быть доступна пользователям круглосуточно, за исключением плановых технических работ.
- Мобильная совместимость: Приложение должно быть совместимым с большинством версий мобильных телефонов
- Система должна быть стабильной и не подвержена частым сбоям.

Для реализации данных функций, необходимо:

- Определиться со средствами для разработки
- Определить язык программирования
- Создать базу данных для хранения данных пользователей
- Подключить необходимые библиотеки
- Получить API-ключи для доступа к функционалу карт

Выводы по главе 1.

В первой главе выпускной квалификационной работы был проведён анализ деятельности компании «ВорлдИнтерТех РУС», на базе которой

разрабатывается мобильное приложение. Компания специализируется на цифровизации бизнес-процессов и разработке программных решений, включая облачные сервисы и инструменты интернет-маркетинга. Особое внимание было уделено структуре и функционалу отдела разработки, в который входят сотрудники с различным уровнем квалификации: от Junior до Senior-программистов и менеджера проекта. Рассмотрение их ролей позволило выстроить обоснованную модель производственного процесса, включающего формирование технического задания, распределение задач и реализацию решений.

На основе анализа предметной области были сформулированы как функциональные, так и нефункциональные требования к разрабатываемому приложению. Ключевые функциональные требования включают отображение геолокации пользователей в реальном времени, обмен текстовыми сообщениями, создание и редактирование групп, а также возможность совершения телефонных вызовов. Нефункциональные требования касаются высокой стабильности, мобильной совместимости, скорости отклика интерфейса и круглосуточной доступности системы.

Также в главе обоснован выбор инструментов и технологий, необходимых для реализации проекта: подбор языка программирования, базы данных, библиотек и API для интеграции карт. Полученные результаты служат основой для проектирования архитектуры и последующей реализации приложения. Таким образом, проведённый анализ позволил чётко определить направления проектной деятельности и сформировал базу для эффективной реализации программного продукта, отвечающего требованиям целевой аудитории и особенностям бизнес-процессов.

Глава 2. Построение архитектуры и проектирование приложения

2.1 Выбор технологий для разработки

Современные подходы к созданию программных продуктов объединяют комплекс методик и инструментальных средств, направленных на разработку эффективных программных решений, соответствующих актуальным требованиям пользователей [1]. В контексте мобильной разработки существует широкий спектр технологических решений, каждый из которых обладает своими характеристиками. Ниже представлен анализ ключевых платформ и инструментов, применяемых в процессе создания мобильных приложений.

Разработка кроссплатформенных приложений дает возможность создавать единое решение сразу для нескольких операционных систем - Android, iOS и веб-платформ. Главное преимущество такого подхода в том, что можно использовать один и тот же код для всех платформ, что экономит время и ресурсы разработчиков. Однако у этого метода есть и недостатки: приложения могут работать немного медленнее, чем нативные, и не всегда могут в полной мере использовать все возможности устройства.

Веб-приложения – это еще одно решение. Они работают прямо в браузере и не требуют установки, что делает их доступными на любом устройстве. Но для работы обязательно нужно интернет-подключение, а доступ к функциям смартфона, таких как камера и GPS у них значительно меньше, чем у обычных приложений.

Нативная разработка – это подход создания отдельных версий приложения специально для каждой платформы (Android на Kotlin или Java, iOS на Swift или Objective-C). Такой подход дает максимальную производительность и полный доступ ко всем функциям устройства, но требует больше времени и ресурсов, так как нужно разрабатывать и поддерживать несколько версий приложения.

Нативная разработка обладает рядом существенных преимуществ перед кроссплатформенными и веб-решениями. Нативные приложения создаются с использованием специальных языков программирования и инструментов, разработанных конкретно для каждой платформы — например, Swift для iOS и Kotlin для Android [3]. Такой подход обеспечивает высокую эффективность работы приложений, поскольку они получают прямой доступ к оптимизированным системным функциям и библиотекам, созданным именно для этой операционной системы. В результате нативные приложения имеют меньшую задержку, более плавную анимацию и лучшее управление ресурсами, что особенно важно для графически насыщенных приложений, таких как игры или приложения с высокой нагрузкой на процессор [2]. Нативные приложения имеют полный доступ ко всем API и функциональным возможностям устройства. Это включает в себя использование камеры, GPS, сенсоров, хранилища и других аппаратных возможностей [2]. Нативные приложения используют элементы интерфейса, которые соответствуют стандартам дизайна на каждой платформе. Это означает, что пользователи будут получать более интуитивно понятный и привычный интерфейс, так как он будет следовать стандартам и паттернам, принятым на платформе. Нативная разработка позволяет создать интерфейс, который будет выглядеть и вести себя так, как ожидают пользователи [2]. У разработчиков есть доступ к большому количеству документации, примеров кода и форумов, что значительно упрощает процесс разработки и решение возникающих проблем [2]. Такие приложения не зависят от сторонних библиотек, поэтому снижается риск возникновения проблем совместимости [2].

На основании проведенного анализа различных подходов к разработке программного обеспечения, было принято решение о выборе нативного подхода создания приложения. Данный выбор обоснован необходимостью обеспечения доступа к API операционной системы, требованием к работе модуля геолокации, а также необходимостью достижения максимальной производительности.

Нативная разработка позволяет в полной мере реализовать заявленный функционал приложения.

Основным языком для разработки приложений под платформу Android является Kotlin, поэтому разработка велась на этом языке.

Сервис Appwrite был выбран в качестве бэкенда для приложения. Appwrite – это сервис с открытым исходным кодом, предоставляющий разработчикам основной набор функций реализации бэкенда [6]. От взаимодействия с базой данных до аутентификации, обновлений в реальном времени и многого другого [6]. Appwrite интегрируется с мобильными и веб-клиентами (JS, Flutter, Swift, Objective-C, Kotlin) и предоставляет нужную среду для бэкэнд-задач (Node, .NET, Python, PHP, Ruby, Deno, ограниченно на Go и Java) [7].

2.2 Выбор архитектуры для приложения

Выбор архитектурного решения представляет собой ключевой этап в процессе разработки программного обеспечения, поскольку он определяет принципы организации системы и способы реализации её функциональных требований. В программной инженерии значимым этапом стало появление паттернов проектирования в 1990-х годах, которые возникли как результат систематизации лучших практик объектно-ориентированного программирования [18]. Эти проверенные временем шаблоны предоставляют разработчикам унифицированные подходы к решению типовых задач, что способствует созданию более надежных и поддерживаемых приложений [8].

Современные архитектурные подходы, такие как MVC, MVP и MVVM, предлагают различные варианты структурирования компонентов приложения с четким распределением ответственности между ними. Каждая из этих архитектур определяет специфические принципы взаимодействия между элементами системы, их свойствами и взаимосвязями, что в конечном итоге влияет на такие характеристики продукта, как масштабируемость,

тестируемость и простота дальнейшего развития функциональности. Выбор конкретной архитектурной парадигмы осуществляется с учетом особенностей решаемой задачи и требований к конечному продукту.

Архитектурный паттерн MVC (Model-View-Controller) предлагает четкое разделение ответственности между тремя основными компонентами: модель (Model) отвечает за хранение данных и бизнес-логику приложения, представление (View) обеспечивает визуализацию информации для пользователя и передачу его действий контроллеру, контроллер (Controller) обрабатывает пользовательский ввод, вносит изменения в модель и обновляет представление соответствующим образом [9]. Такой подход обеспечивает эффективное разделение пользовательского интерфейса и прикладной логики, что значительно упрощает процессы тестирования, сопровождения и дальнейшего развития программного продукта [9].

Паттерн MVP (Model-View-Presenter) сохраняет базовое разделение компонентов также как MVC, но вносит существенные изменения в механизм взаимодействия между ними. В данной архитектуре модель (Model) сохраняет свою роль хранилища данных и бизнес-логики, в то время как представление (View) фокусируется исключительно на отображении информации, передавая все пользовательские действия представителю (Presenter). Представитель является посредником, содержащий всю логику представления и напрямую управляющего обновлением пользовательского интерфейса. Такой подход подходит для проектов, требующих строгого контроля за поведением интерфейса и повышенной тестируемости компонентов, поскольку обеспечивает более четкое разделение ответственности между слоями приложения.

MVVM (Model-View-ViewModel) представляет собой подход к организации кода в Android-приложениях, обеспечивающий четкое разделение между компонентами [10, с. 2]. В данной архитектуре представление отвечает исключительно за визуальное представление данных, ViewModel выступает в качестве посредника, обрабатывающего бизнес-

логику и подготавливающего данные для отображения, тогда как Model сосредоточена на работе с данными и основной логикой приложения [10, с. 2]. Такое разделение обязанностей значительно упрощает процесс разработки и тестирования, поскольку каждый компонент выполняет строго определенные функции, что способствует созданию более поддерживаемого и масштабируемого кода [10, с. 2].

Модель выполняет ключевую роль в управлении данными и бизнес-логикой приложения. Этот компонент взаимодействует с различными источниками данных, включая базы данных и веб-сервисы, полностью абстрагируясь от вопросов отображения информации и пользовательского интерфейса [11]. Такая изоляция позволяет сосредоточиться исключительно на обработке и предоставлении данных, не завися от особенностей их визуального представления.

Представление в MVVM отвечает за визуализацию и взаимодействие с пользователем. Получая подготовленные данные от ViewModel, этот слой обеспечивает их корректное отображение, фиксируя все пользовательские действия и передавая их для обработки [11]. Представление в этой архитектуре не содержит никакой бизнес-логики, что значительно упрощает его модификацию и тестирование.

ViewModel выступает в роли посредника между моделью и представлением, обеспечивая их слабую связанность. Этот компонент не только преобразует данные модели в удобный для отображения формат, но и реагирует на пользовательские действия, инициируя соответствующие изменения в модели [11]. Использование механизмов привязки данных позволяет автоматически синхронизировать представление при изменении состояния приложения, что делает MVVM особенно эффективным для разработки сложных и динамичных интерфейсов.

Схема работы MVVM представлена на рисунке 3.

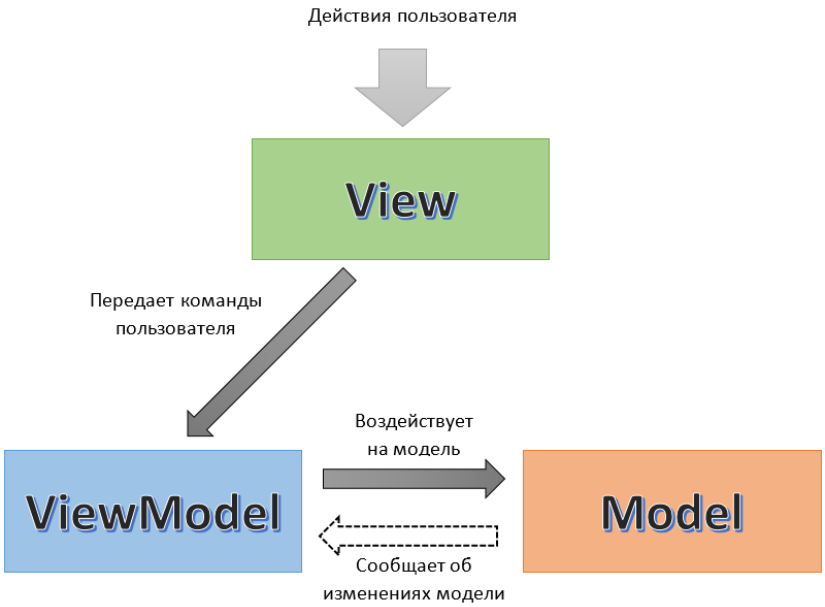

Рисунок 3 – Схема архитектуры MVVM

Когда пользователь выполняет какое-либо действие, представление фиксирует это событие и передает соответствующий запрос в ViewModel для обработки [9]. На этом этапе представление не содержит никакой бизнес-логики, а только выполняет роль посредника между пользователем и логикой приложения.

ViewModel, получив запрос от представления, обрабатывает его с использованием данных из модели. После выполнения необходимых вычислений и преобразований, ViewModel возвращает обновленные данные обратно в представление [10]. Такой подход обеспечивает согласованность данных во всех компонентах приложения и сохраняет разделение ответственности между ними.

Система классов приложения в архитектуре MVVM представлена на рисунке 4.

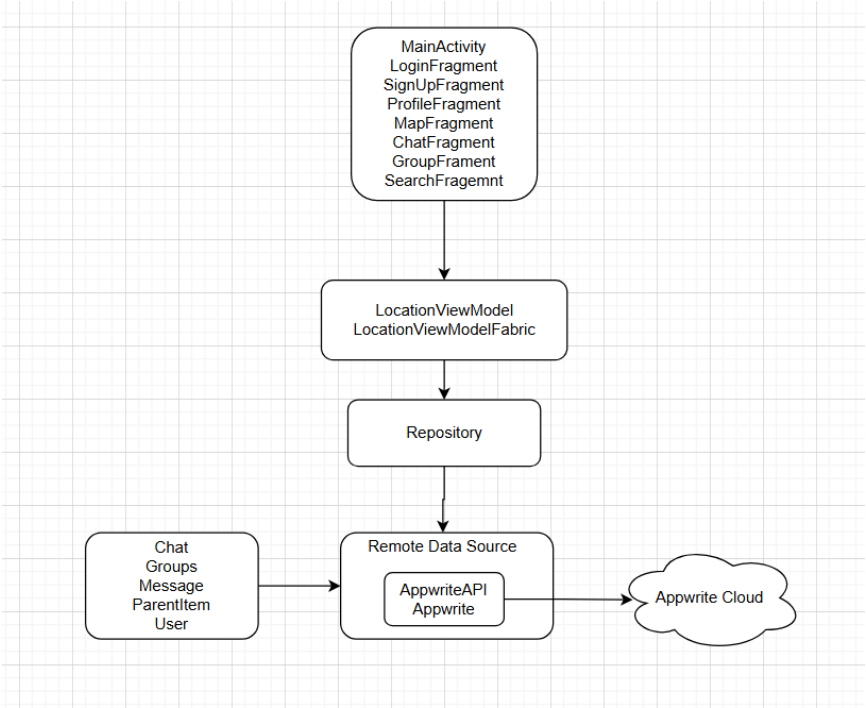

Рисунок 4 – Система классов приложения

После анализа существующих архитектурных подходов, была выбрана модель MVVM. Этот выбор обусловлен рядом ключевых преимуществ, которые особенно важны для разработки. Архитектура MVVM обеспечивает четкое разделение кода на логические компоненты, что соответствует принципам чистой архитектуры и облегчает поддержку проекта.

2.3 Моделирование модели данных

Концептуальная модель предметной области представляет собой фундаментальное описание структуры данных, отражающее потребности пользователей и особенности работы с информацией в конкретной сфере деятельности [12]. Эта модель формируется на основе анализа требований к обработке данных, которые выражаются в терминах, максимально приближенных к профессиональному языку специалистов данной области. На концептуальном уровне определяются ключевые сущности, их

характеристики и взаимосвязи, что создает целостное представление о предметной области без привязки к техническим деталям реализации.

Особенностью концептуального моделирования является его ориентация на естественное представление данных, соответствующее тому, как специалисты воспринимают свою предметную область [12]. Такой подход позволяет выявить и формализовать требования к данным, обеспечивая при этом взаимопонимание между всеми участниками проекта - от конечных пользователей до разработчиков. Полученная модель служит важным промежуточным звеном между бизнес-требованиями и технической реализацией системы.

В ходе проектирования информационной системы был использован метод ER-моделирования (Entity-Relationship), который представляет собой формализованный подход к описанию структуры данных. В основе данной методологии лежат три фундаментальные концепции:

- Сущность (Entity) - базовый строительный блок модели, отражающий значимый объект или понятие предметной области. Каждая сущность соответствует отдельному типу данных, подлежащих хранению в системе. Например, в контексте приложения для обмена сообщениями ключевыми сущностями могут быть "Пользователь", "Сообщение" и "Чат".
- Атрибут (Attribute) - характеристика, описывающая свойства и особенности сущности. Атрибуты определяют структуру хранимой информации, задавая типы данных и возможные значения для каждого параметра. Для сущности "Пользователь" типичными атрибутами могут быть "Имя", "Email" и "Дата регистрации".
- Связь (Relationship) - формальное описание взаимодействия между сущностями, выражающее бизнес-правила предметной области. Связи могут иметь различную кардинальность: один-к-одному (например, пользователь и его профиль), один-ко-многим (пользователь и его сообщения) или многие-ко-многим (пользователи и чаты).

– Ключ - специальный атрибут или набор атрибутов, однозначно идентифицирующий экземпляр сущности.

Данная методология наглядно представляет структуру данных и взаимосвязи между различными компонентами системы.

Процесс проектирования оптимальной структуры базы данных включает последовательное выполнение нескольких ключевых этапов [13]. Первоначально проводится тщательный анализ бизнес-требований и правил предметной области, что позволяет определить основные данные и их взаимосвязи. Затем выполняется детальное исследование отношений между сущностями, на основании которого данные организуются в таблицы с четкой структурой. Важным шагом является определение ограничений целостности для каждой таблицы, обеспечивающих корректность хранимой информации. Завершающим этапом выступает нормализация таблиц, направленная на устранение избыточности данных и достижение оптимальной структуры базы [13].

На рисунке 5 представлена концептуальная модель.

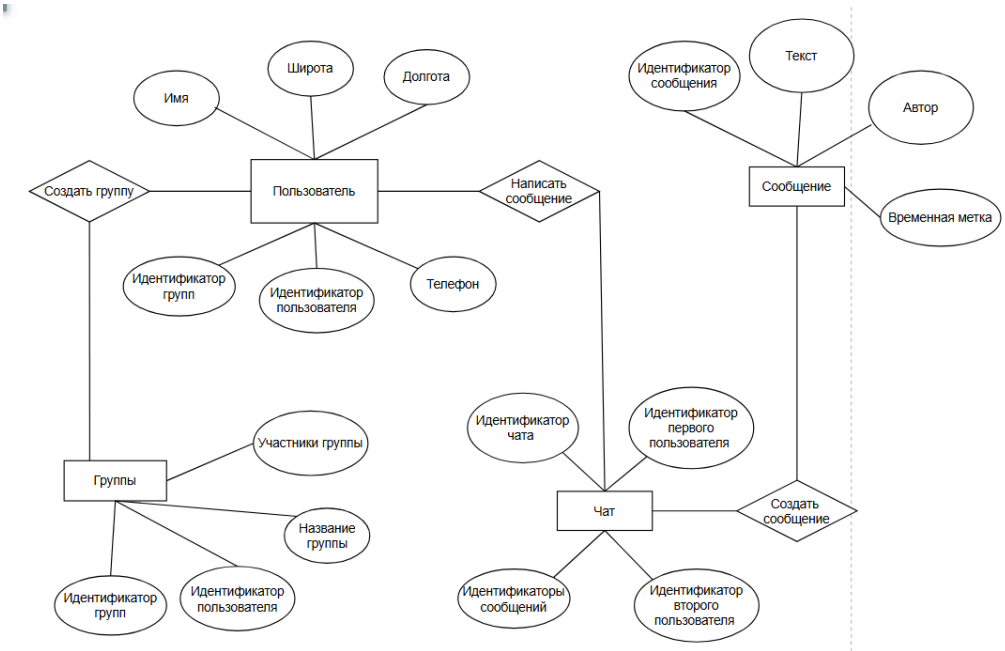

Рисунок 5 – Концептуальная модель

Логическая модель данных является следующим этапом концептуальной модели, где происходит ее детализация и техническая конкретизация [14, с. 27]. На данном уровне проектирования осуществляется углубленный анализ структуры информации: уточняются характеристики данных, форматы их представления и взаимосвязи между элементами. В логической модели формализуются универсальные принципы организации данных, которые должны поддерживаться любой системой управления базами данных соответствующего класса. Эта модель служит важным промежуточным звеном между концептуальным представлением предметной области и ее физической реализацией, обеспечивая технологическую независимость при сохранении семантической целостности данных [14, с. 27]. Эта модель представлена на рисунке 6.

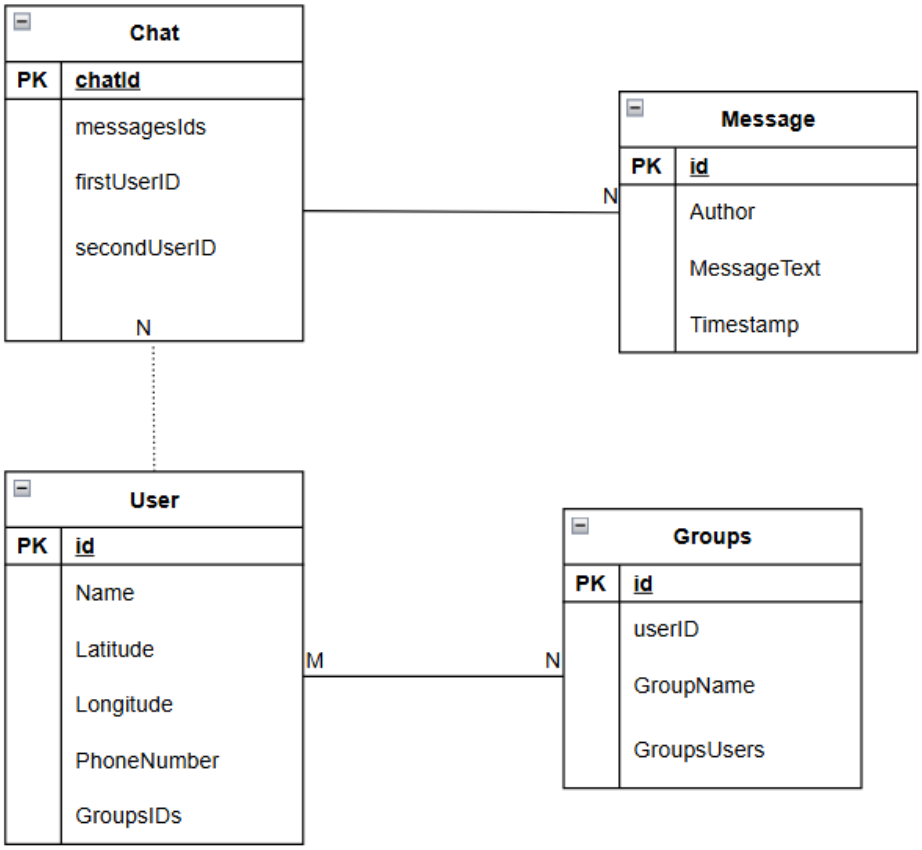

Рисунок 6– Логическая модель данных

Физическая модель базы данных разрабатывается на основе ранее созданной логической модели, адаптируя ее к особенностям конкретной системы управления базами данных [14, с. 81]. Этот этап проектирования предполагает техническую реализацию концептуальных сущностей и связей с учетом специфики выбранной платформы, включая определение типов данных, индексов, механизмов хранения и методов доступа. Ключевая задача физического проектирования заключается в эффективном преобразовании логической структуры данных в работоспособную схему базы данных, оптимизированную для конкретной СУБД, что обеспечивает максимальную производительность и надежность системы при работе с реальными данными [14, с. 81].

Физическая модель приведена на рисунке 7.

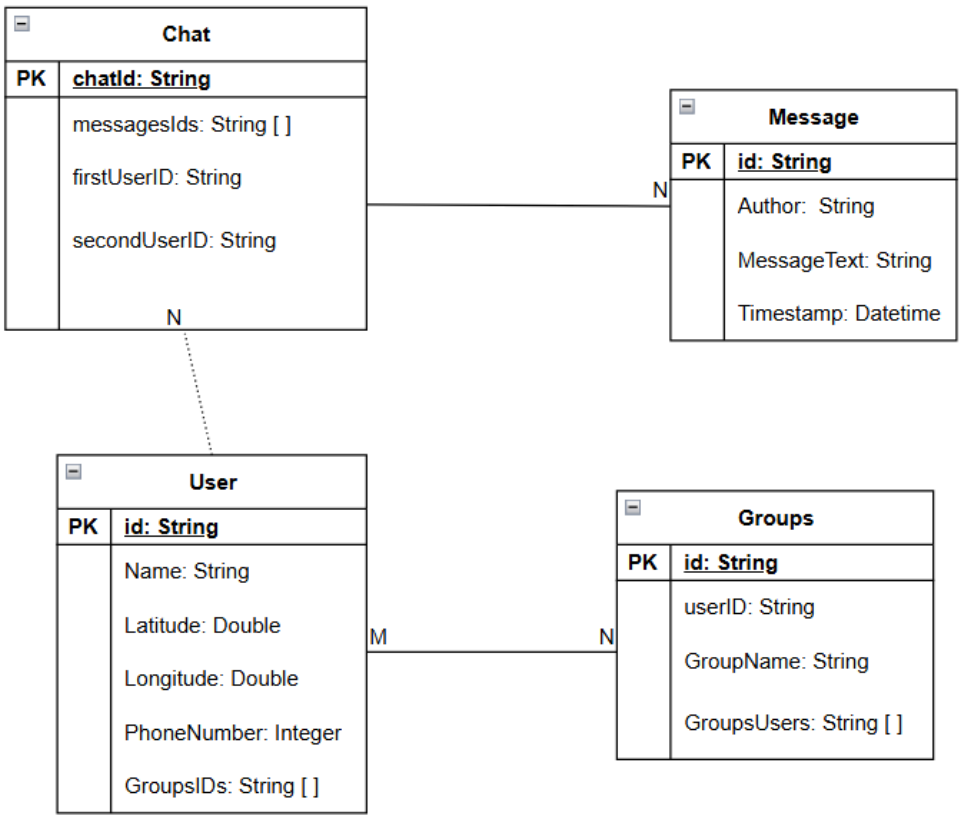

Рисунок 7 – Физическая модель данных

В таблице 1 представлено описание полей, приведенных в физической модели.

Таблица 1 – Структура таблиц

Название поля	Тип поля	Описание
User (Пользователь)		
Id	String	ID пользователя
Name	String	ФИО пользователя
Latitude	Double	Географическая широта местоположения
Longitude	Double	Географическая долгота местоположения
PhoneNumber	Integer	Номер телефона пользователя
GroupsIDs	String []	Массив идентификаторов групп, в которых состоит пользователь
Groups (Группы)		
id	String	ID группы
userID	String	Идентификатор создателя группы
GroupName	String	Название группы
GroupUsers	String []	Массив идентификаторов пользователей в группе
Message (Сообщение)		
id	String	ID сообщения
Author	String	ID автора сообщения
MessageText	String	Текст сообщения
Timestamp	Datetime	Дата и время отправки сообщения
Chat (Чат)		
chatId	String	ID чата
messageIds	String []	ID сообщений
firstUserID	String	ID первого пользователя в чате
secondUserID	Datetime	ID второго пользователя в чате

В таблице 1 представлена структура базы данных, используемой в разработанном мобильном приложении. Она охватывает четыре основные сущности: пользователь (User), группа (Groups), сообщение (Message) и чат (Chat), каждая из которых содержит определённый набор полей, необходимых для корректного хранения и обработки информации в приложении.

Сущность **User** отвечает за хранение информации о зарегистрированных пользователях. В неё входят поля: уникальный идентификатор, имя, координаты текущего местоположения (широта и долгота), номер телефона и список идентификаторов групп, к которым относится пользователь.

Сущность **Groups** содержит данные о созданных группах. Она включает идентификатор группы, идентификатор её создателя, название и список пользователей, входящих в состав группы. Эти поля обеспечивают возможность объединения пользователей в логические объединения и управления ими.

Сущность **Message** служит для хранения сообщений, отправляемых между пользователями. Поля этой таблицы включают идентификатор сообщения, идентификатор автора, текст сообщения и временную метку, фиксирующую дату и время отправки. Это позволяет отслеживать переписку в рамках различных чатов.

Сущность **Chat** реализует логику взаимодействия между двумя пользователями. Она содержит уникальный идентификатор чата, массив идентификаторов сообщений, а также идентификаторы участников переписки. Такое представление обеспечивает организацию диалога и связку сообщений с соответствующими пользователями.

Представленная структура данных отражает основные информационные потоки в приложении и обеспечивает необходимую функциональность для реализации обмена сообщениями, управления группами и отслеживания местоположения пользователей.

2.4 Разработка сценариев взаимодействия пользователей

Качество пользовательского интерфейса является одним из основных свойств программы. Качественно разработанный пользовательский интерфейс играет важную роль в обеспечении положительного опыта взаимодействия, напрямую влияя на степень удовлетворенности пользователей и их вовлеченность в работу с приложением [15]. Несоответствие интерфейса общепринятым стандартам проектирования и нормативным требованиям создает существенные трудности в процессе эксплуатации программного продукта [4]. В связи с этим, при разработке мобильного приложения

необходимо учитывать ряд фундаментальных требований к проектированию пользовательского интерфейса:

– Интерфейс приложения должен обладать интуитивно понятной структурой. Элементы дизайна необходимо выполнить в едином стилистическом решении. Интерактивные компоненты должны быть визуально выделены и расположены в зоне легкой досягаемости.
– Система обязана обеспечивать мгновенную и предсказуемую реакцию на любые пользовательские действия. Задержки при выполнении операций должны быть минимизированы, а переходы между экранами - плавными.
– Структура меню и организация переходов должны позволять пользователю быстро получать доступ к требуемым разделам. Ключевые функции приложения необходимо располагать в пределах 1-2 шагов от главного экрана, оптимизируя тем самым пользовательские маршруты.

  На рисунке 8 представлена диаграмма вариантов использования.

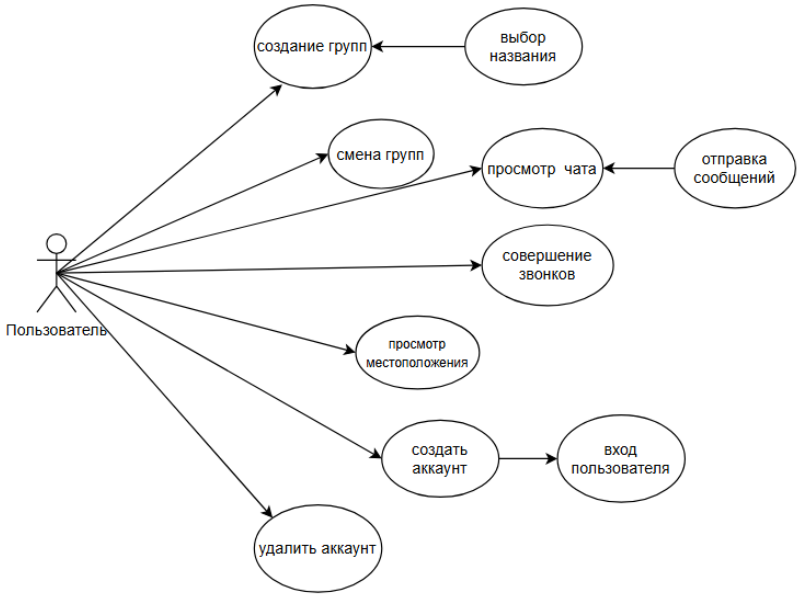

Рисунок 8 – Диаграмма вариантов использования

Диаграмма вариантов использования представляет систему на концептуальном уровне, демонстрируя её ключевые функциональные возможности и взаимодействия, что позволяет понять общую логику работы приложения "извне". В отличие от этого, диаграмма взаимодействия модулей (рисунок 9) раскрывает внутреннюю архитектуру системы, наглядно отображая структурные компоненты, распределение функциональности между ними и существующие взаимосвязи, что особенно ценно для анализа сложности системы и зависимостей между её частями при разработке и сопровождении программного обеспечения.

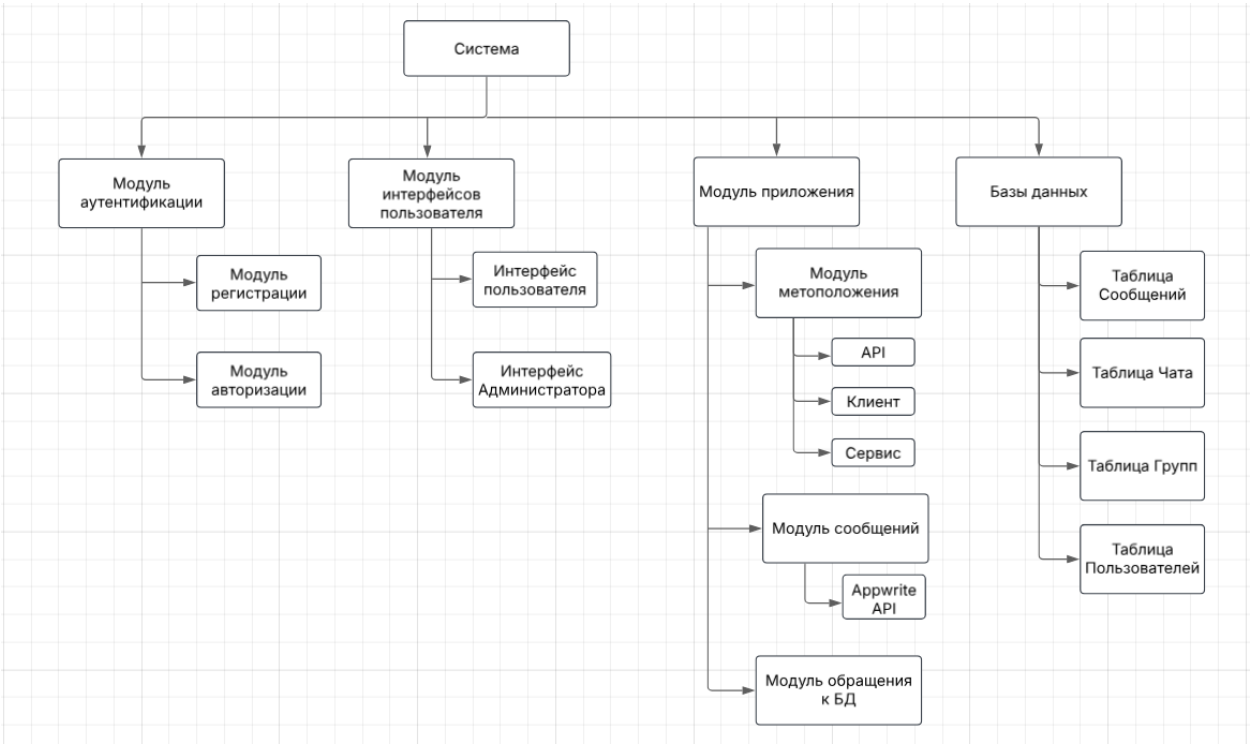

Рисунок 9 – Диаграмма взаимодействия модулей ПО

Модуль аутентификации и авторизации отвечает за вход пользователей в систему и управление сессиями. Работает с базой данных, чтобы подтверждать логины и пароли.

Модуль интерфейсов пользователя обеспечивает визуальное взаимодействие пользователя с приложением, включая экраны для работы с

картой, чатами и группами, получая данные от основного модуля приложения.

Управление хранением происходит в модуле базы данных. Этот модуль реализует запись, чтение и обновление данных в базе, взаимодействуя с другими модулями через запросы.

Модуль приложения координирует работу всех компонентов, обрабатывает запросы от интерфейса и управляет основной логикой приложения.

С помощью данных диаграмм, можно реализовать прототип пользовательского интерфейса. На рисунке 10 приведен прототип пользовательского интерфейса.

Рисунок 10 – Прототип пользовательского интерфейса

На рисунке показаны основные формы экранов. Основным элементом интерфейса для навигации будет нижняя панель которая в свою очередь будет переключать на остальные компоненты приложения,

Выводы по главе 2.

Во второй главе была проведена системная работа по проектированию архитектуры и технического фундамента мобильного приложения. На основе анализа современных подходов к разработке было принято решение в пользу нативной разработки на языке Kotlin, что позволило обеспечить высокую

производительность, стабильность работы и доступ ко всем возможностям мобильного устройства, включая модули геолокации и уведомлений. В качестве серверной платформы был выбран сервис Appwrite, обеспечивающий удобные инструменты для работы с базой данных, аутентификацией и хранением файлов.

Особое внимание было уделено выбору архитектурного паттерна. Изучив особенности моделей MVC, MVP и MVVM, был обоснован выбор архитектуры MVVM, обеспечивающей чёткое разделение между представлением, логикой и данными. Это решение позволило повысить читаемость и масштабируемость кода, упростить отладку и поддержку приложения, а также обеспечить удобную интеграцию с библиотеками Android Jetpack.

В рамках проектирования информационной системы была создана трёхуровневая модель данных: концептуальная, логическая и физическая. Эти модели охватывают ключевые сущности — пользователя, группы, сообщения и чаты — и обеспечивают эффективное хранение, обработку и взаимодействие данных. Физическая модель была адаптирована под возможности сервиса Appwrite, что обеспечило гибкость и надёжность хранения информации.

Кроме того, разработаны диаграммы вариантов использования и взаимодействия модулей, а также прототип интерфейса, отображающий основные пользовательские сценарии. Это обеспечило основу для логичной навигации в приложении и удобства использования конечным пользователем.

Таким образом, в главе 2 была реализована важнейшая проектная часть работы, заложившая прочную архитектурно-технологическую базу для последующей разработки и успешного функционирования мобильного приложения.

Глава 3. Разработка приложения и тестирование

3.1 Обзор разработанного программного решения

Разработанное решение представляет собой законченное мобильное приложение для обмена геолокацией и мессенджинга с функцией управления группами пользователей. Приложение обеспечивает пользователям удобный инструмент для координации действий в реальном времени.

Пользовательский интерфейс разрабатывался с использованием стандартных компонентов Android и языка разметки XML для создания адаптивных макетов [2]. Для реализации серверной части выбрана облачная платформа Appwrite, предоставляющая готовый SDK для интеграции с мобильным клиентом. Процесс разработки осуществлялся в официальной среде Android Studio, которая предлагает полный набор инструментов для написания, тестирования и отладки Android-приложений, что значительно упрощает создание программного продукта [2].

Приложение включает в себя следующий основной функционал:

- Авторизация и регистрация пользователей;
- Создание и управление группами;
- Добавление пользователей в группу;
- Поиск по пользователям;
- Обмен сообщениями в режиме реального времени;
- Отслеживание геолокации участников групп;
- Интуитивный интерфейс с нижней панелью навигации.

Решение соответствует стандартам UI/UX, обеспечивая простоту использования и высокую производительность. Архитектура MVVM и модель данных позволяют легко масштабировать приложение.

3.2 Создание проекта и интеграция зависимостей

Для начала создадим проект в Android Studio. После установки самой среды разработки переходим во окно создания проектов и выбираем “Empty Views Activity” (рисунок 11), далее заполняем данные нашего приложения и нажимаем “Finish” (рисунок 12). После этого создастся проект нашего приложения.

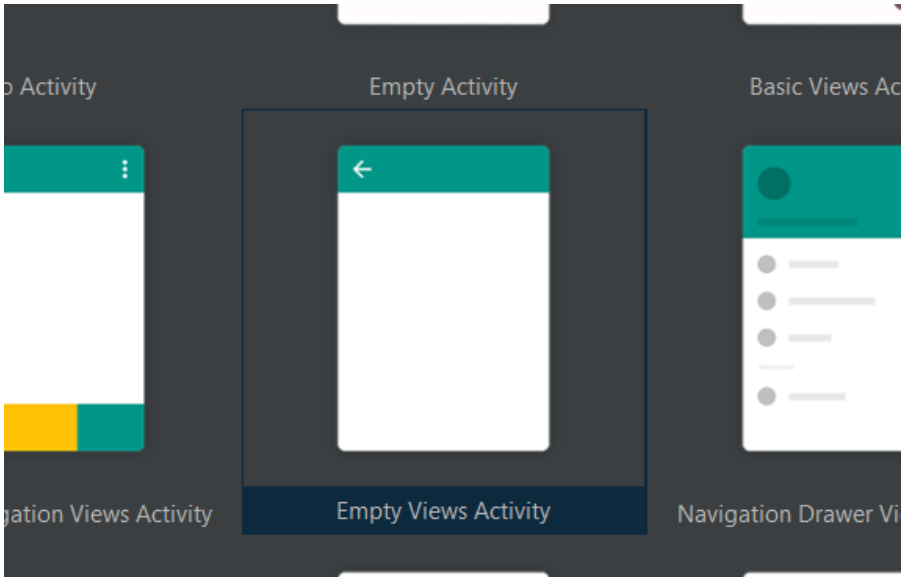

Рисунок 11– Выбор главного экрана приложения

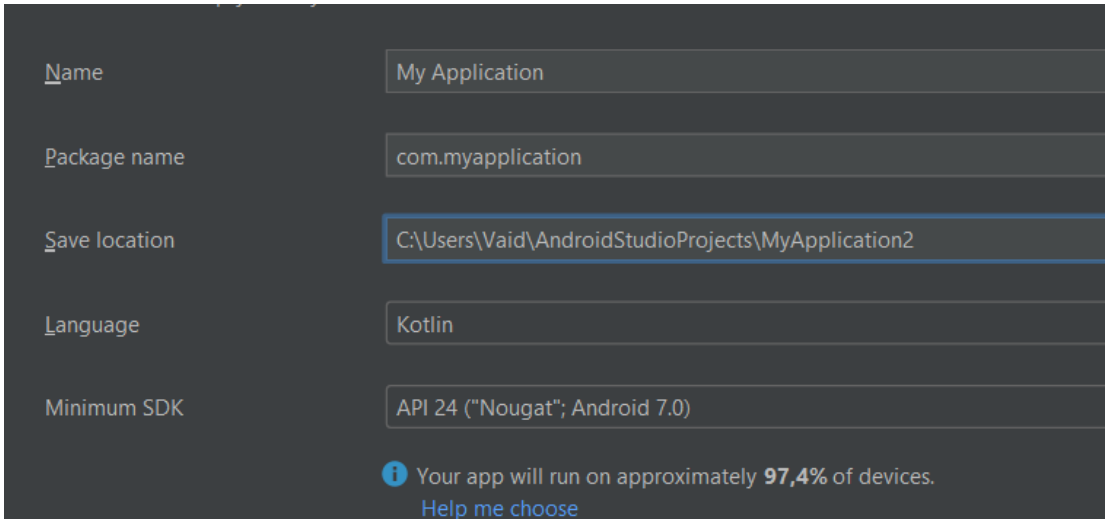

Рисунок 12 – Заполнение данных приложения

Добавим в зависимости проекта необходимые библиотеки для работы. Все зависимости находятся в файле “build.gradle.kts” – конфигурационный файл проектов на Kotlin, использующий систему сборки Gradle. Он позволяет определять зависимости, подключать плагины, настраивать параметры сборки, такие как версия компилятора и целевые платформы, создавать пользовательские задачи, указывать репозитории для загрузки зависимостей и задавать метаданные проекта (Рисунок 13).

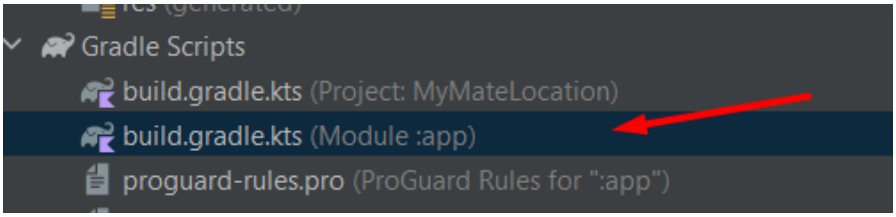

Рисунок 13 – Конфигурационный файл gradle

Главные зависимости в проекте составляют библиотеки для работы с нативными компонентами, навигацией, сопрограммами, картами, библиотеки Appwrite для получения доступа к его функциям. Зависимости приведены на рисунке 14.

```
// Navigation Components
implementation("androidx.navigation:navigation-fragment-ktx:2.7.5")
implementation("androidx.navigation:navigation-ui-ktx:2.7.5")

//Picasso
implementation("com.squareup.picasso:picasso:2.5.2")

//appwrite
implementation("io.appwrite:sdk-for-android:4.0.0")

//анимация
implementation("com.airbnb.android:lottie:6.4.0")

//карты
implementation("com.google.android.gms:play-services-location:18.0.0")
implementation("com.google.android.gms:play-services-maps:19.0.0")
```

Рисунок 14 – Зависимости приложения

В файле AndroidManifest.xml добавим разрешения для использования точного местоположения устройства. Разрешения приведены на рисунке 15.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

<uses-feature android:name="android.hardware.location.gps" />
<uses-feature android:name="android.hardware.location.network" />
```

Рисунок 15 – Разрешения приложения для доступа к геолокации.

На данный момент мы подключили все основные библиотеки и получили разрешения на использование модулей мобильного устройства.

3.3 Разработка функционала регистрации и аутентификации

Для реализации функционала для регистрации и аутентификации, зарегистрируемся в сервисе Appwrite, выбирая план биллинга как “Starter”. Это позволит создать один бесплатный проект. На рисунке 16 приведена форма создания организации.

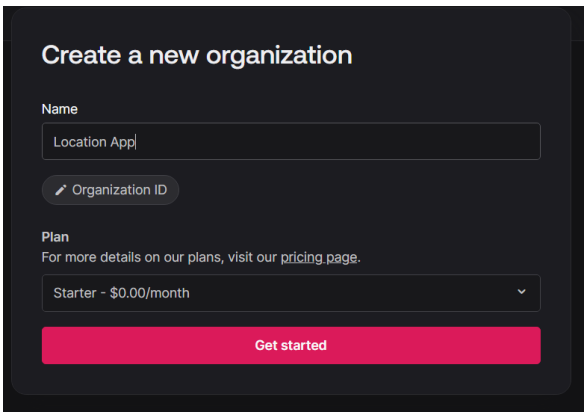

Рисунок 16 – Форма создания организации

После этого, на главной странице, отобразится консоль пользования, где приводится информация по количеству пользователей, таблиц, запросов к базе данных и хранилища файлов. Консоль отображена на рисунке 17.

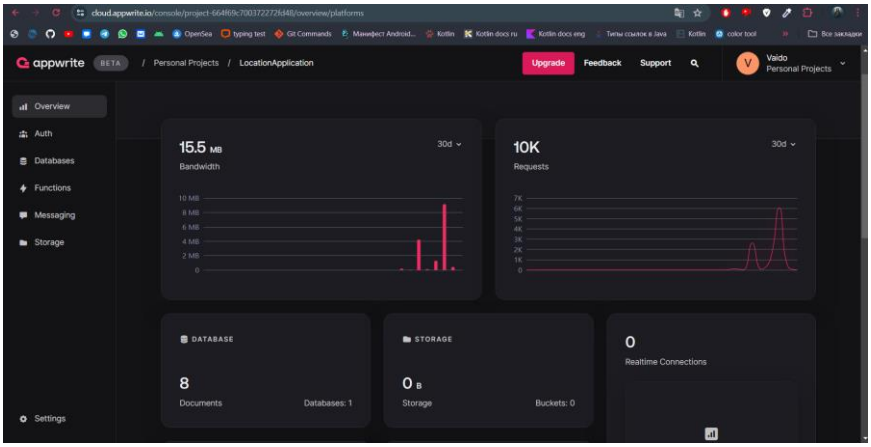

Рисунок 17 – Экран с информацией по проекту

Инициализируем Appwrite в созданном проекте. Инициализируем в классе Appwrite переменные для предоставления доступа к классам библиотек Account, Database, Client, Storage (рисунок 18).

```
class Appwrite private constructor() {

    companion object {
        @Volatile
        private var instance: Appwrite? = null

        fun getInstance(): Appwrite {
            return instance ?: synchronized( lock: this) {
                instance ?: Appwrite().also { instance = it }
            }
        }
    }

    private var client: Client? = null
    private var database: Databases? = null
    private var account: Account? = null
    private var storage: Storage? = null

    fun initialize(context: Context) {
        if (client == null) {
            client = Client(context)
                .setEndpoint(BASE_URL)
                .setProject(PROJECT_ID)

            database = Databases(client!!)
            account = Account(client!!)
            storage = Storage(client!!)
        }
    }
```

Рисунок 18 – Классы библиотек Appwrite

Согласно архитектуре MVVM, создаем слой абстракции для управления методами регистрации и авторизации путем использования интерфейса AppwriteAPI. Методы приведены на рисунке 19.

```
interface AppwriteAPI {

    suspend fun login(email:String, password: String): Session

    suspend fun signup(userID: String, name: String, email:String, password: String)
```

Рисунок 19 – Методы интерфейса авторизации

Наследуем интерфейс и реализуем их в классе Repository. На рисунке 20 приведена реализация данных методов.

```
class Repository : AppwriteAPI {

    private val appwriteDatabase = Appwrite.getInstance().getDatabaseInstance()
    private val appwriteAccount = Appwrite.getInstance().getAccountInstance()

    override suspend fun login(email: String, password: String): Session =
        appwriteAccount.createEmailSession(email, password)

    override suspend fun signup(userID: String, name: String, email: String, password: String){
        appwriteAccount.create(userID, email, password, name)
    }
```

Рисунок 20 – Реализация методов интерфейса

В классе инициализируются два объекта: “appwriteDatabase” для работы с базой данных Appwrite и “appwriteAccount” для управления учетными записями пользователей. Метод “login” принимает email и пароль, затем вызывает метод “createEmailSession” из Appwrite SDK, который создает сессию пользователя и возвращает объект “Session” при успешной аутентификации [20]. Метод “signup” принимает параметры пользователя и с

помощью метода “create” создает новую учетную запись в системе Appwrite. Оба метода являются suspend-функциями, что означает их возможность работать в сопрограммах для асинхронного выполнения без блокировки основного потока приложения. Таким образом, этот класс инкапсулирует логику взаимодействия с бэкенд-сервисом Appwrite, предоставляя простой интерфейс для регистрации и входа пользователей в приложение.

Опишем методы в классе ViewModel – промежуточным слоем между представлением и данными, обрабатывая логику приложения, сохраняя состояние интерфейса при изменениях конфигурации. Методы вызываем в сопрограммах для асинхронного выполнения. Фрагмент реализации класса приведен на рисунке 21.

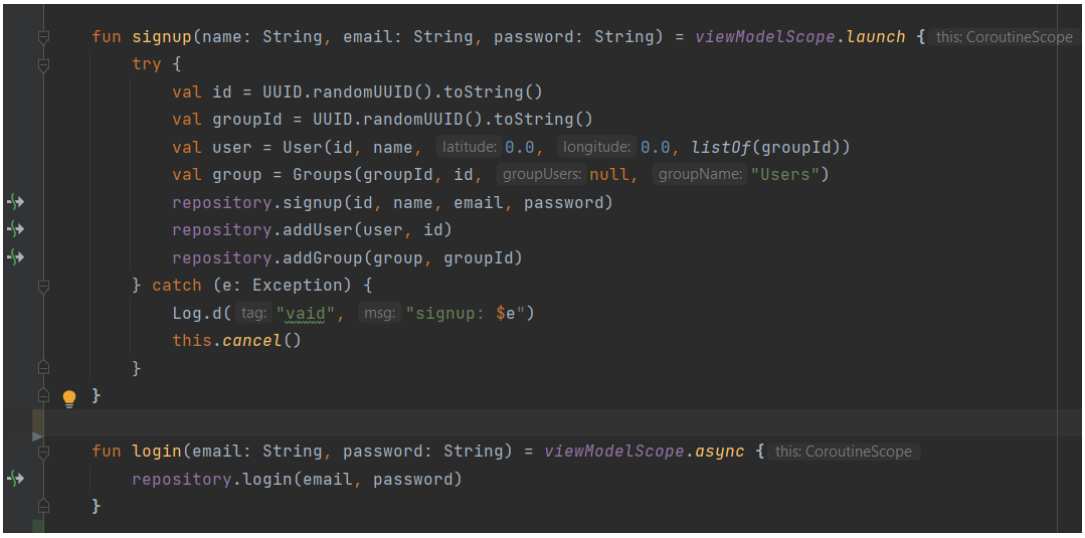

Рисунок 21 – Вызов методов в классе ViewModel

Корутины в Android-разработке представляют собой механизм для организации асинхронных операций, сочетающий высокую производительность с понятным синхронным стилем программирования [20]. В экосистеме Kotlin данная функциональность реализована в библиотеке Kotlin Coroutines, которая предоставляет разработчикам удобные инструменты для выполнения ресурсоемких задач в фоновом режиме,

включая сетевые запросы и взаимодействие с базами данных, без блокировки основного потока приложения. Этот подход значительно улучшает отзывчивость интерфейса и общую эффективность работы мобильного приложения, сохраняя при этом читаемость и поддерживаемость кода [20].

Для управления сопрограммами используются два основных метода:

– launch() - запускает новую корутину без возврата результата
– async() - выполняет асинхронную операцию с возвратом Deferred<T>

Блок viewModelScope – это встроенная область видимости для ViewModel, которая отменяет все выполняющиеся корутины при очистке ViewModel. Это предотвращает утечки памяти и выполнение операций после завершения жизненного цикла компонента [20].

Реализуем в классах LoginActivity и SignupActivity методы, отвечающие за пользовательский интерфейс. Реализации методов приведены на рисунке 22 и рисунке 23.

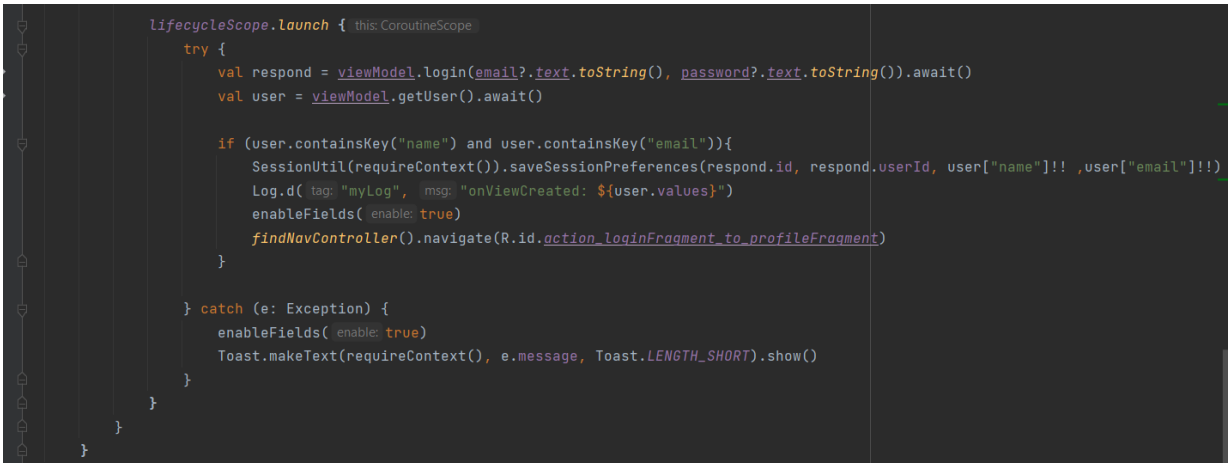

Рисунок 22 – Вызов метода в классе ViewModel

Рисунок 23 – Вызов метода в классе ViewModel

Интерфейс экранов было реализовано с помощью языка верстки XML. В реализации используется компонент FrameLayout как корневой контейнер, содержащий ProgressBar который отображается при загрузке, и вертикальный LinearLayout с заголовком, двумя полями ввода для почты и пароля, включая иконку переключения видимости пароля, и кнопкой "Войти" внизу экрана. Интерфейс представлен на рисунке 24.

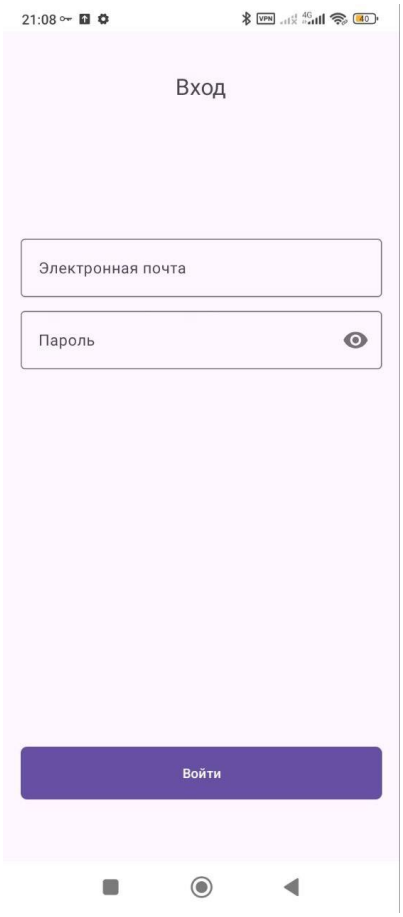

Рисунок 24 – Экран авторизации

На форме регистрации предусмотрено дополнительное поле для ввода имени пользователя, а также текстовая ссылка под основной кнопкой, позволяющая перейти к экрану авторизации. Визуальное представление и компоновка элементов интерфейса экрана регистрации отображены на рисунке 25.

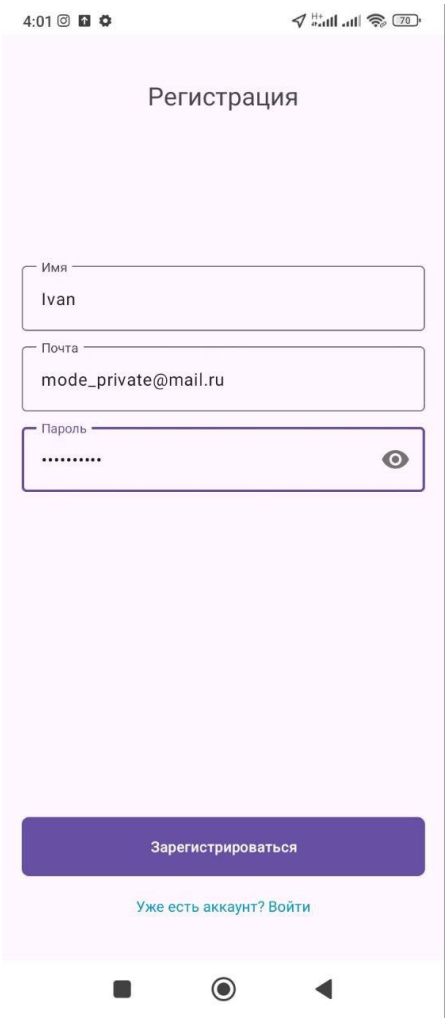

Рисунок 25 – Экран авторизации

В итоге функции входа и регистрации пользователя были реализованы в приложении.

3.4 Создание удаленной базы данных и модуля местоположения

Для хранения и обработки геоданных пользователей была разработана база данных с использованием платформы Appwrite. В разделе "Databases" создана база данных под названием "MainDatabase", содержащая четыре таблицы: "Messages" для хранения сообщений, "User" для учётных записей пользователей, "Groups" для информации о группах и "Chat" для данных чат-

комнат. По итогу получается база данных с четырьмя таблицами, представленными на рисунке 26.

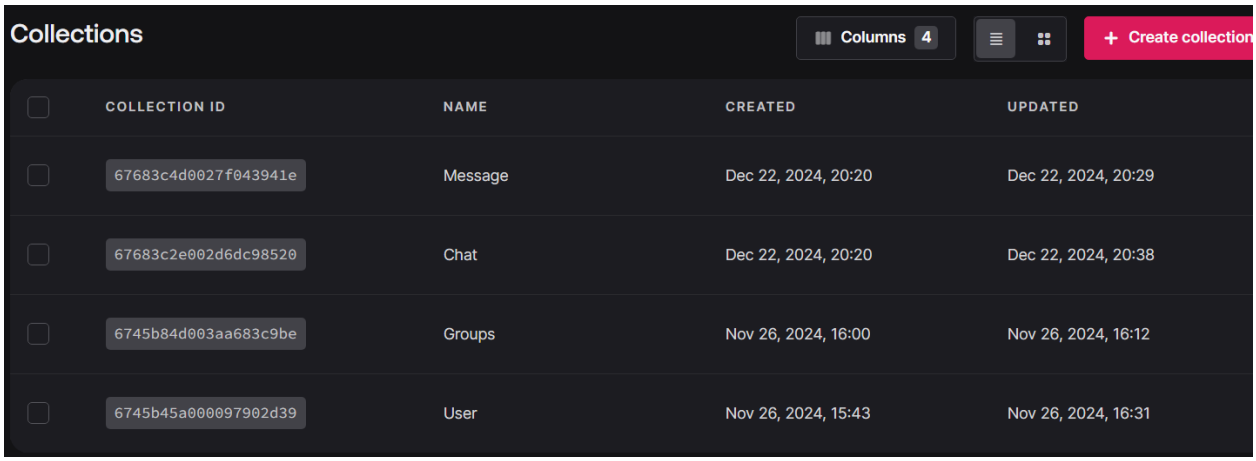

Рисунок 26 – Таблицы базы данных

Таблица “User” будет хранить информацию про местоположение пользователя (ширина и долгота), массив идентификаторов созданных групп и имя.

Таблица “Groups” хранит свой идентификатор, идентификатор пользователя, добавленных пользователей и имя группы.

Таблица “Chat” хранит два идентификатора, массив с элементами сообщений и свой идентификатор.

Таблица “Message” будет хранить id, автора сообщения, текст сообщения и временную метку для даты.

В проекте также определяем эти сущности в виде класса данных (рисунок 27).

```
data class User(
    val id: String? = null,
    val name: String?,
    val latitude: Double,
    val longitude: Double,
    val groupsIDs: List<String>?
): Serializable
```

Рисунок 27 - Класс данных пользователя

После этого сгенерируем API-ключ для доступа к Google картам. В консоли Google Cloud во вкладке “APIs & Services” нажимаем “Create Credentials” и генерируем ключ (рисунок 28). После этого в колонке “API Keys” появится наш сгенерированный ключ. Этот ключ мы прописываем в теге <meta-data> в AndroidManifes.xml.

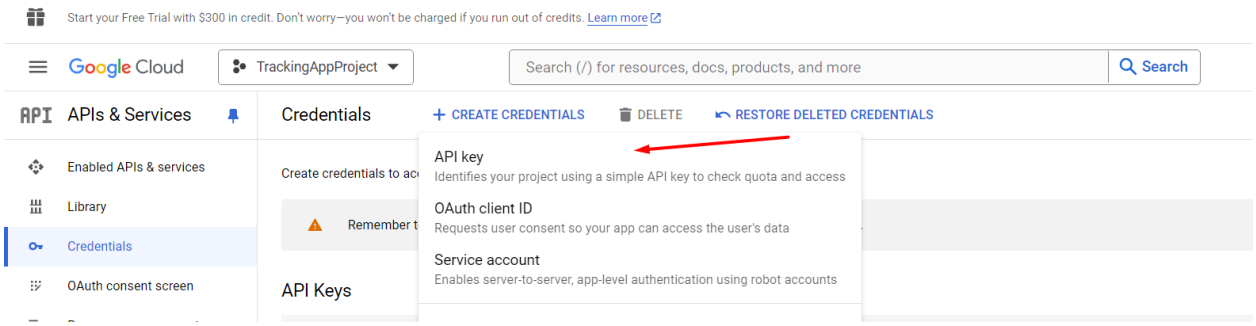

Рисунок 28– Генерация ключа

При реализации модуля местоположения создаем сервис для фонового отслеживания местоположения LocationService.

Сервис — это компонент приложения, предназначенный для выполнения длительных операций в фоновом режиме без пользовательского интерфейса. Сервис работает даже когда приложение свёрнуто или закрыто.

Сервис использует LocationClient для получения обновлений геопозиции через Fused Location Provider API из Google Play Services.

При создании сервиса инициализируется locationClient, а методы start() и stop() управляют его работой.

Метод start() запускает получение координат с интервалом в 3000 миллисекунд через Flow, обрабатывая успешные результаты и ошибки, используя сопрограммы в serviceScope чтобы избежать блокировки основного потока. Сервис поддерживает два действия через Intent, а при уничтожении отменяет все сопрограммы.

Метод stop() завершает работу через stopForeground(true) и stopSelf(), а данные локации передаются через Flow с обработкой состояний Result.Success и Result.Error. Особенности реализации включают изоляцию ошибок через SupervisorJob, логирование координат/ошибок, и остановку всех операций при завершении работы сервиса. Весь процесс построен на сопрограммах для асинхронности и безопасности работы с ресурсами. Реализация класса сервиса приведена на рисунке 29.

```
class LocationService : Service(){

    private val serviceScope = CoroutineScope( context: SupervisorJob() + Dispatchers.IO)
    private lateinit var locationClient: LocationClient


    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        when(intent?.action){
            ACTION_START -> start()
            ACTION_STOP -> stop()
        }

        return super.onStartCommand(intent, flags, startId)
    }

    private fun start(){
        locationClient
            .getLocationUpdates( interval: 3000L)
            .catch { e -> e.printStackTrace() }
            .onEach { location ->
                when(location) {
                    is Result.Success -> {
                        val lat = location.data.latitude
                        val lng = location.data.longitude
                        Log.d( tag: "myLog",  msg: "start: $lat, $lng")
                    }
                    is Result.Error -> Log.d( tag: "myLog",  msg: "start: ${location.exception.error}")
                }
            }
            .launchIn(serviceScope)
    }

    private fun stop(){
        stopForeground( removeNotification: true)
        stopSelf()
    }
```

Рисунок 29 – Реализация класса сервиса

Команды запуска сервиса для отслеживания локации вызываем во фрагменте MapFragment, которая отвечает за отображение географической карты и маркер позиции пользователя.

Таким образом были реализованы основные функции приложения.

3.5 Тестирование приложения

После того как во фрагментах был реализован весь заданный функционал, необходимо протестировать работу приложения. Запустим приложение на эмуляторе и реальном устройстве и протестируем весь его функционал. В первую очередь на главном экране должно отобразится местоположение пользователя и выбранной группы. Работа главного экрана отображена на рисунке 30.

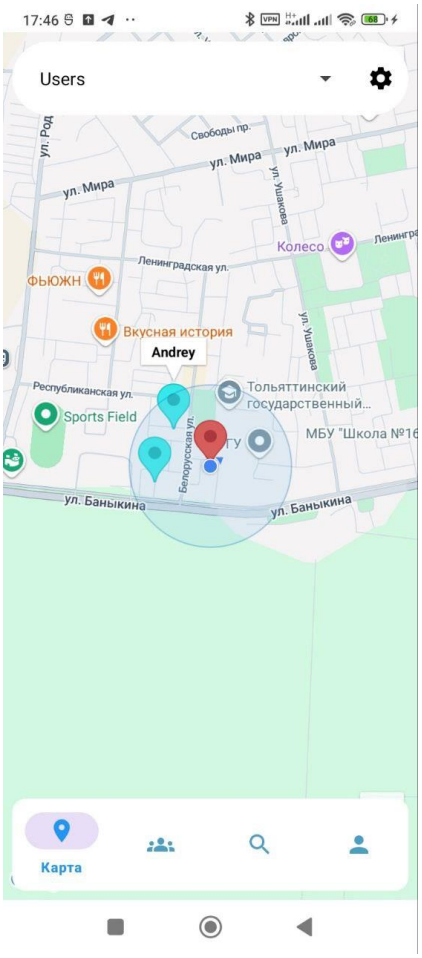

Рисунок 30 – главный компонент

В удаленной базе данных появились данные о местонахождении пользователя. Это приведено на рисунке 31. Таким образом модуль местоположения работает верно.

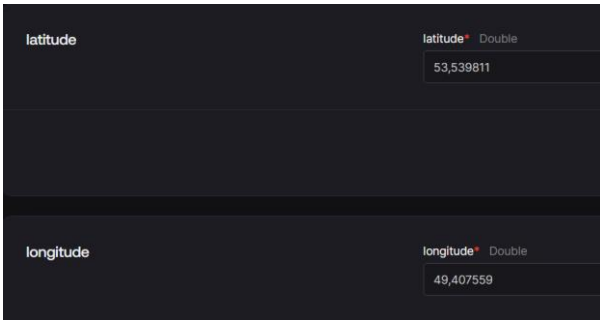

Рисунок 31– данные о местоположении в базе данных

Компонент мессенджера был протестирован на двух разных устройствах, с одного устройства отправлялись сообщения, а другое устройство принимало сообщения и также отправляла сообщения в чат. Экранные формы приведены на рисунках 32.

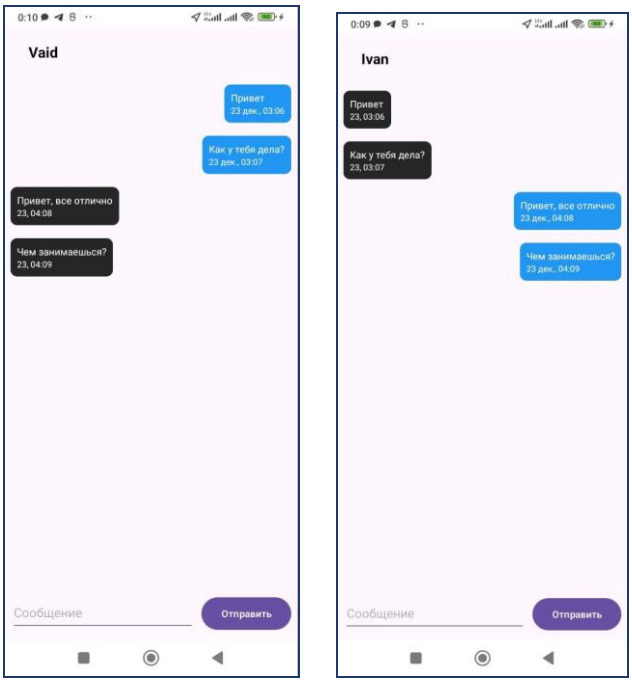

Рисунок 32 – Первое и второе устройства

По итогу, модуль отправки сообщений работает корректно и выполняет свои функции.

Фрагмент групп должен отображать количество групп и пользователей в них, при создании новой группы должно выводится диалоговое окно, в котором указывается названия группы. Экранная форма фрагмента представлена на рисунке 33.

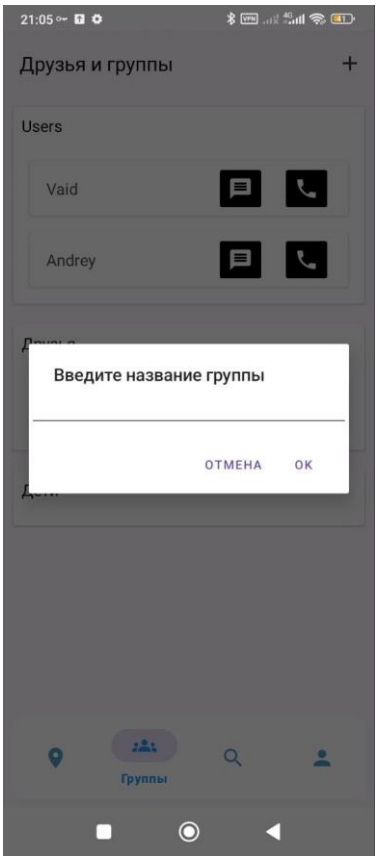

Рисунок 33 – Фрагмент группы

В рамках завершающего этапа разработки было выполнено тестирование работоспособности мобильного приложения на различных Android-устройствах. Экспериментальная проверка подтвердила стабильную работу всех функциональных модулей приложения, включая механизмы геолокации, обмена сообщениями и управления группами. Пользовательский интерфейс продемонстрировал корректное отображение и удобство

взаимодействия на всех тестируемых устройствах независимо от размера экрана и разрешения. Результаты тестирования подтверждают работоспособность всех заявленных функций.

Для публикации приложения в мобильных магазинах требуется подготовка APK-файла [17]. Данный файл представляет собой архив, содержащий весь необходимый для работы код, ресурсы и метаданные приложения в упакованном виде. Технически APK-файл представляет собой вариант архивного формата JAR, адаптированный специально для платформы Android, что обеспечивает его совместимость с магазинами приложений [20].

Для этого в Android Studio нужно перейти во вкладку “Build > Build Bundles / APKs > Build APK”. Это показано на рисунке 34.

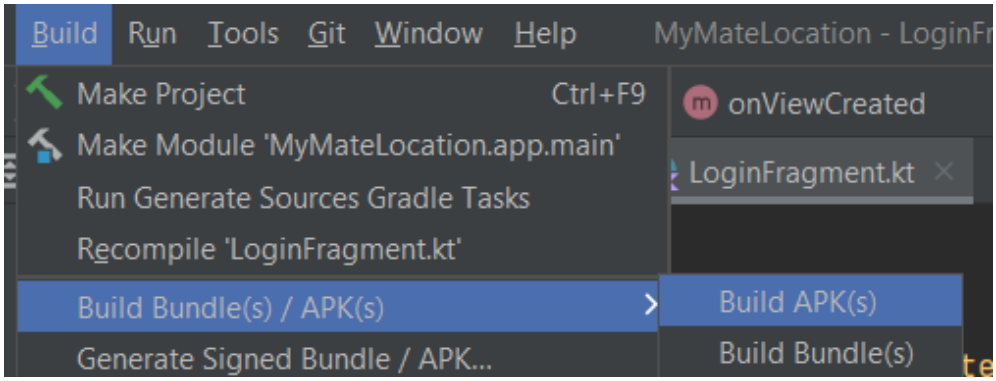

Рисунок 34 – Генерация APK

После этого, в директории проекта, создастся файл приложения, которое можно устанавливать на реальные устройства и опубликовывать в магазины приложений, такие как Google Play.

Выводы по главе 3

В третьей главе была реализована практическая часть проекта, включающая разработку и тестирование мобильного приложения, соответствующего ранее сформулированным требованиям. Разработка велась в среде Android Studio с использованием языка Kotlin и архитектурного паттерна MVVM, что обеспечило модульность, читаемость и надёжность

программного кода. В качестве серверного решения был применён сервис Appwrite, предоставляющий функциональность для хранения данных, регистрации и аутентификации пользователей, а также обмена сообщениями.

На начальном этапе реализован функционал регистрации и авторизации, включая взаимодействие с системой аккаунтов Appwrite и защиту пользовательских данных. Далее был создан модуль определения местоположения, построенный на основе сервиса Fused Location Provider, что позволило осуществлять фоновое отслеживание геолокации пользователей с последующим сохранением данных в удалённой базе. В рамках разработки был также реализован модуль обмена сообщениями, обеспечивающий двустороннюю передачу текстовой информации в режиме реального времени. Дополнительно реализован функционал создания и управления группами, включая отображение участников и работу с пользовательскими списками.

Интерфейс приложения был создан с соблюдением принципов UI/UX-дизайна, что обеспечило удобную навигацию, адаптивную верстку и минимизацию количества действий пользователя для выполнения ключевых операций. Проведённое тестирование подтвердило корректную работу всех модулей как на эмуляторах, так и на физических устройствах. Были проверены основные сценарии: вход в систему, определение местоположения, передача сообщений и управление группами.

В результате практической реализации создано полнофункциональное приложение, готовое к распространению. Подготовлен APK-файл, пригодный для установки на устройства и публикации в официальных магазинах приложений. Все этапы разработки выполнены в полном соответствии с поставленными задачами.

Заключение

В процессе выполнения выпускной квалификационной работы была достигнута основная цель исследования — разработано мобильное приложение, предназначенное для обмена геолокацией и сообщений с возможностью управления пользовательскими группами. Работа охватывала все этапы жизненного цикла программного обеспечения: от анализа предметной области до практической реализации и тестирования готового продукта.

На первом этапе был проведён анализ деятельности IT-компании «ВорлдИнтерТех РУС», определены ключевые функции сотрудников отдела разработки и специфика бизнес-процессов, что позволило сформулировать функциональные и нефункциональные требования к приложению. Это стало основой для выбора архитектурных и технологических решений.

Во второй главе был выполнен выбор инструментов и методов разработки. На основе сравнительного анализа современных подходов было принято решение использовать нативную разработку на языке Kotlin, архитектурный паттерн MVVM и облачную платформу Appwrite. Также была построена трёхуровневая модель данных и разработаны диаграммы взаимодействия компонентов, что обеспечило структурированность и масштабируемость создаваемой системы.

На завершающем этапе реализовано полноценное мобильное приложение с модулями авторизации, геолокации, обмена сообщениями и управления группами. Проведённое тестирование подтвердило работоспособность и стабильность всех компонентов, а также соответствие требованиям к пользовательскому интерфейсу.

Таким образом, поставленные задачи выполнены в полном объёме, а результаты работы могут быть использованы как основа для дальнейшего развития функциональности и внедрения аналогичных решений в других прикладных областях.

Список используемой литературы

1. Гусев, К. В. Технология разработки программных приложений: учебное пособие / К. В. Гусев, М. Б. Туманова, Е. А. Чернов. — Москва: РТУ МИРЭА, 2023. — ISBN 978-5-7339-1938-6. — Текст: электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/book/382706 (дата обращения: 06.06.2024). — Режим доступа: для авториз. пользователей. — С. 4.

2. Гриффитс Дэвид. Head First. Программирование для Android на Kotlin. 3-е изд. - Санкт-Петербург: Питер, 2023. - 912 с. - ISBN 978-5-4461-2016-1. - URL: https://ibooks.ru/bookshelf/391722/reading (дата обращения: 03.05.2025). - Текст: электронный.

3. Скин Джош, Гринхол Дэвид. Kotlin. Программирование для профессионалов. — СПб.: Питер, 2020. — 464 с.:ил.- ISBN 978-5-4461-1243-2.

4. Баланов, А. Н. Комплексное руководство по разработке: от мобильных приложений до веб-технологий: учебное пособие для вузов / А. Н. Баланов. — Санкт-Петербург: Лань, 2024. — ISBN 978-5-507-48841-4. — Текст: электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/394577 (дата обращения: 06.06.2024). — Режим доступа: для авториз. пользователей. — С. 97.

5. Фешина, Е. В. С. А. Куштанок, Т. А. Крамаренко, Р. Я. Скорбатюк Анализ технологий разработки мобильных приложений и информационных систем на базе операционной системы Android // Вестник Адыгейского государственного университета. Серия 4: Естественно-математические и технические науки. — 2022. — № 1. — С. 85-91. — ISSN 2410-3225. — Текст: электронный // Лань: электронно-библиотечная система. — URL: https://e.lanbook.com/journal/issue/331307 (дата обращения: 05.05.2025). — Режим доступа: для авториз. пользователей. — С. 6.

6. Хабр. Создание приложения для чата в реальном времени с помощью Angular и Appwrite [Электронный ресурс]. URL: https://habr.com/ru/companies/otus/articles/700016 (дата обращения: 06.05.2025)

7. Хабр. Appwrite, open-source бэкэнд-платформа [Электронный ресурс]. URL: https://habr.com/ru/companies/vdsina/articles/561482 (дата обращения: 06.05.2025)

8. Синицын, И. В. Разработка мобильных приложений : учебное пособие / И. В. Синицын, Е. А. Чернов, Ю. А. Воронцов. — Москва : РТУ МИРЭА, 2023 — Часть 1 — 2023. — ISBN 978-5-7339-1799-3. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/368735 (дата обращения: 04.06.2024). — Режим доступа: для авториз. пользователей. — С. 69-71.

9. Рогачев С. Обобщенный Model-View-Controller // URL: https://rsdn.org/article/patterns/generic-mvc.xml

10. Старушенкова Екатерина Евгеньевна, Палютин Руслан Раисович ПАТТЕРН ПРОЕКТИРОВАНИЯ MVVM, КАК ОДИН ИЗ СПОСОБОВ НАПИСАНИЯ «ЧИСТОГО» КОДА В ANDROID-ПРИЛОЖЕНИИ НА JETPACK COMPOSE // E-Scio. 2023. №4 (79). URL: https://cyberleninka.ru/article/n/pattern-proektirovaniya-mvvm-kak-odin-iz-sposobov-napisaniya-chistogo-koda-v-android-prilozhenii-na-jetpack-compose (дата обращения: 14.04.2025).

11. Вирт Н. Алгоритмы и структуры данных = Algoritms and data structure. — М.: Мир, 1989. — 360 с. — ISBN 5-03-001045-9.

12. . Практикум по информатике / Н. М. Андреева, Н. Н. Василюк, Н. И. Пак, Е. К. Хеннер. — 3-е изд., стер. — Санкт-Петербург : Лань, 2024. — 248 с. — ISBN 978-5-507-47299-4. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/359810 (дата обращения: 04.05.2025). — Режим доступа: для авториз. пользователей. — С. 106.

13. Гринченко, Н. Н. Проектирование моделей данных : учебное пособие / Н. Н. Гринченко, Н. И. Хизриева, С. Н. Баранова. — Рязань : РГРТУ, 2022. — 48 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/380387 (дата обращения: 04.06.2024). — Режим доступа: для авториз. пользователей. — С. 4.

14. Астапчук, В. А. Базы данных: проектирование и реализация : учебное пособие / В. А. Астапчук, Е. Н. Павенко, И. В. Эстрайх. — Новосибирск : НГТУ, 2023. — ISBN 978-5-7782-4917-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/404294 (дата обращения: 04.06.2024). — Режим доступа: для авториз. пользователей. — С. 27.

15. Шибанов, С.В. Автоматизированное проектирование пользовательских интерфейсов / С.В. Шибанов, А.А. Пашкин // Вестник Пензенского государственного университета. — 2016. — № 4. — С. 67-73. — ISSN 2410-2083. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/journal/issue/301580 — С. 1.

16. Гришанков, В. Что такое APK-файлы и зачем они нужны? // AndroiLime URL: https://androidlime.ru/apk-files

17. Burns B., Grant B., Oppenheimer D. Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. O'Reilly Media, 2018. – 166 с.

18. Chen P.P. The Entity-Relationship Model: Toward a Unified View of Data. ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976. – 68 с.

19. Appwrite. Appwrite Cloud Client API Reference [Электронный ресурс]. URL: https://appwrite.io/docs/references/cloud/client-web/account (дата обращения: 05.05.2025).

20. Filip Babić, Luka Kordić, Nishant Srivastava Kotlin Coroutines by Tutorials: Best Practices to Create Safe & Performant Asynchronous Code With Coroutines. - 3 edition изд. - NY: Razeware LLC, 2022. - 301 с.