

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

Android-приложение для совместных беговых тренировок

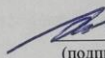
Студент

 29.05.26 Кузьмин Данила Владиславович
(подпись, дата)

Руководитель

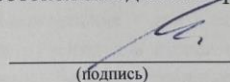
 29.05.26 Кузнецов А.В.
(подпись, дата)

Нормоконтролёр

 01.06.26 Бычков А.С.
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от 22.05.2026 г. № 19)

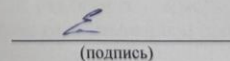
Заведующий кафедрой


(подпись)

Митрохин М.А.

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 11.06.2026 № 22)

Секретарь ГЭК


(подпись)

Гурин Е.И.

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«07» апреля 2026 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Кузьмин Данила Владиславович гр.22ВВП2 факультета вычислительной техники.

2. Тема работы: «Android-приложение для совместных беговых тренировок»

Тема утверждена приказом ПГУ № 377/О от «07» апреля 2026 г.

3. Руководитель бакалаврской работы доцент, каф. ВТ ПГУ Кузнецов Алексей Валерьевич.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать мобильное Android-приложение для совместных беговых тренировок, предоставляющее пользователям инструменты для отслеживания беговых тренировок, анализа статистики, организации совместных беговых тренировок с друзьями и виртуального соревнования с собственными прошлыми результатами.

5. Перечень подлежащих разработке вопросов

Требуется провести анализ предметной области: изучить существующие мобильные приложения для бега, выделить их плюсы и минусы, обосновать актуальность разработки приложения. Выбрать язык программирования для реализации серверной и клиентской части. Определить архитектуру, средство контейнеризации, развёртывания и протестировать приложение.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области	30	07.04.2026 – 11.04.2026	
2	Выбор архитектуры приложения и технологий, применяемых при проектировании и разработке	35	12.04.2026 – 16.04.2026	
3	Разработка клиентской части приложения	66	16.04.2026 – 30.04.2026	
4	Разработка серверной части приложения	65	01.05.2026 – 14.05.2026	
5	Тестирование приложения	30	14.05.2026 – 17.05.2026	
6	Оформление пояснительной записки	25	17.05.2026 – 21.05.2026	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы  07.04.26 Кузнецов А.В.
(подпись, дата)

Задание к исполнению принял студент  07.04.26 Кузьмин Д.В.
(подпись, дата)

Работу к защите допустить
Декан факультета  10.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 137 листов, 56 рисунков, 10 таблиц, 8 источников, 3 приложения.

KOTLIN, JETPACK COMPOSE, ANDROID, CLEAN ARCHITECTURE, MVVM, ROOM, RETROFIT, HILT, KTOR, POSTGRESQL, YANDEX MAPKIT, WEBSOCKET, DOCKER, GPS-ТРЕКИНГ, СОВМЕСТНЫЙ БЕГ

Цель работы – разработать мобильное приложение для платформы Android, предоставляющее пользователям инструменты для отслеживания беговых тренировок, анализа статистики, организации совместных беговых тренировок с друзьями.

Объектом разработки является мобильное Android-приложение для совместных беговых тренировок, предназначенное для отслеживания спортивной активности и совместного взаимодействия бегунов. Приложение позволяет пользователям: регистрироваться и авторизовываться, запускать тренировки с записью GPS-трека в реальном времени, сохранять историю забегов с детальной аналитикой темпа и времени, находить друзей и создавать приглашения на совместные беговые тренировки, а также соревноваться с самим собой, улучшая свои прошлые результаты.

Результатом проделанной работы является готовый программный продукт, включающий клиентскую часть под управлением ОС Android и серверную часть. Клиентское приложение реализовано с использованием языка программирования Kotlin, UI-фреймворка Jetpack Compose, архитектурного паттерна MVVM и принципов Clean Architecture. Для управления зависимостями был выбран фреймворк Hilt, локальное хранение данных организовано через библиотеку

					ПГУ 1.090301.39.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Android-приложение для совместных беговых тренировок Пояснительная записка	Лит.	Лист	Листов
Разраб.		Кузьмин Д.В.		29.05.24				
Провер.		Кузнецов А.В.		29.05.24			2	137
Н. Контр.		Бычков А.С.		01.06.24				
Утверд.		Митрохин М.А.		01.06.24		ФВТ, каф. ВТ гр. 22ВВП2		

Room, сетевое взаимодействие – Retrofit и KtorClient, работа с картами – через YandexMapKit. Серверная часть разработана на языке Kotlin с применением асинхронного фреймворка Ktor, в качестве СУБД используется PostgreSQL, доступ к данным реализован через ORM-библиотеку Exposed. Развертывание сервера осуществляется в контейнере Docker на облачной платформе Railway.

					ПГУ 1.090301.39.001 ПЗ	Лист
						3
Изм.	Лист	№ докум.	Подпись	Дата		

Содержание

Перечень сокращений и обозначений.....	6
Введение.....	7
1 Android-приложение для совместных беговых тренировок.....	8
1.1 Анализ предметной области	8
1.1.1 Анализ актуальности темы	8
1.1.2 Анализ аналогов.....	9
1.1.3 Постановка задачи	19
1.2 Выбор инструментов разработки	24
1.2.1 Выбор среды разработки.....	24
1.2.2 Выбор языка программирования.....	25
1.2.3 Выбор инструментов хранения данных и контейнеризации.....	26
1.2.4 Выбор архитектуры приложения	27
1.2.5 Выбор инструментов построения интерфейса	28
1.2.6 Выбор фреймворка для внедрения зависимостей	28
1.2.7 Выбор картографических сервисов	29
1.3 Реализация приложения	29
1.3.1 Базовая конфигурация приложения	29
1.3.1.1 Ресурсы значений и локализация	30
1.3.1.2 Векторная графика и иконки	30
1.3.2 Структура пакетов и модулей	30
1.3.3 Слой данных.....	31
1.3.4 Доменный слой	33
1.3.5 Слой представления.....	33
1.3.6 Сервисы, утилиты и внедрение зависимостей	35
1.3.7 Реализация серверной части.....	35
1.3.8 Структура приложения и топология системы.....	37
1.3.9 Структура баз данных	40
1.3.9.1 Локальная база данных	41
1.3.9.2 Серверная база данных	42

1.4 Тестирование	45
1.4.1 Регистрация и авторизация пользователя.....	46
1.4.2 Трекинг пробежки и работа с картами	51
1.4.3 Совместная беговая тренировка.....	54
1.4.4 Соревнование в режиме «Призрак».....	62
1.4.5 Управление списком друзей.....	64
1.4.6 Система достижений и статистика.....	67
2 Безопасность жизнедеятельности	70
2.1 Требования к организации рабочего места	70
2.2 Требования к освещению	71
2.3 Требования к режиму отдыха	71
Заключение	73
Список использованных источников	74
Приложение А – UML-диаграммы.....	76
Приложение Б – Листинг программы	81
Приложение В – Презентация.....	130

Перечень сокращений и обозначений

БД – база данных

ОС – операционная система

ПО – программное обеспечение

СУБД – система управления базами данных

API (Application Programming Interface) – программный интерфейс, набор готовых правил и команд для взаимодействия с приложением

GPS (Global Positioning System) – глобальная система позиционирования

IDE (Integrated Development Environment) – интегрированная среда разработки

JSON (JavaScript Object Notation) – текстовый формат для хранения и обмена данными

JWT (JSON Web Token) – это строка, которая используется для безопасной передачи данных между клиентом и сервером

MVVM (Model-View-ViewModel) – архитектурный шаблон, который разделяет пользовательский интерфейс и логику приложения

REST (Representational State Transfer) – это набор правил, который позволяет программам взаимодействовать с сервером через интернет

SQL (Structured Query Language) – язык структурированных запросов

UI (User Interface) – интерфейс, который видит пользователь

UML (Unified Modeling Language) – графический язык для проектирования

WSS (WebSocket Secure) – защищенная версия протокола WebSocket

XML (Extensible Markup Language) – расширяемый язык разметки

Введение

В современном мире поддержание здорового образа жизни и регулярная физическая активность являются важнейшей частью жизни каждого человека. Растущий интерес к занятиям спортом на открытом воздухе, в частности к бегу, создает высокий спрос на удобные и эффективные способы контроля прогресса и поддержания мотивации. Одним из наиболее доступных инструментов для этого являются специализированные мобильные приложения на платформе Android.

Основная цель данной выпускной работы – разработать приложение, которое будет закрывать необходимые потребности современного атлета в отслеживании собственных тренировок, позволит наглядно анализировать достигнутые результаты, а также поможет повысить мотивацию через элементы достижений и соревновательности. Данное приложение предоставляет возможности по точному учету пройденных маршрутов, анализу статистики забегов, а также уникальный функционал соревнования с другом во время совместной беговой тренировки и виртуальных соревнований против своих прошлых результатов.

Результатом данной работы будет создание Android-приложения, отвечающего современным запросам в сфере мобильной разработки, которое сочетает в себе интуитивно понятный интерфейс и необходимые функциональные возможности, что поможет существенно повысить эффективность тренировочного процесса и вовлеченность пользователя в занятия спортом.

1 Android-приложение для совместных беговых тренировок

1.1 Анализ предметной области

1.1.1 Анализ актуальности темы

В настоящее время контроль физической активности и поддержание здорового образа жизни становятся все более важными аспектами жизни каждого человека, и Android-приложения для мониторинга бега занимают в этом значимую роль. Они помогают пользователю вести учет своих тренировок, анализировать показатели активности, устанавливать личные цели и получать подробную статистику о своем прогрессе. Такие приложения становятся незаменимым инструментом для тех, кто стремится к регулярным занятиям спортом, пытается улучшить свои физические показатели и подходит более осознанно к организации тренировочного процесса.

При проведении анализа в данной области можно выделить несколько ключевых аспектов, которые являются значимыми при выборе приложения:

–функциональность: Android-приложение для мониторинга бега должно обладать набором необходимого функционала, такого как точный GPS-трекинг маршрута, расчет дистанции, времени и темпа, работа в фоновом режиме, система достижений, полный отчет тренировки, простая регистрация и авторизация, а также уникальные возможности, такие как функция совместной беговой тренировки для взаимодействия с другими атлетами и режим «Призрак» для соревнований с самим собой;

–удобство: при реализации приложения важным фактором является удобство его использования, так как это напрямую влияет на пользовательский опыт. Основными способами обеспечения комфортного взаимодействия являются интуитивно понятный интерфейс, обеспечивающий быструю навигацию, логичное расположение элементов управления и стабильная работа приложения даже при длительных нагрузках в фоновом режиме;

–дизайн: визуальная составляющая является важной частью при выборе приложения. Оно должно иметь современный дизайн с понятным расположением навигационных элементов, приятную цветовую палитру, не переутомляющую пользователя, а также узнаваемые иконки и кнопки, что облегчает взаимодействие с интерфейсом во время активного движения.

Таким образом, Android-приложения для совместных беговых тренировок – это важный и крайне востребованный продукт для пользователей, которые стремятся к активному образу жизни, контролю своих спортивных достижений и поиску дополнительной мотивации.

1.1.2 Анализ аналогов

На данный момент существует множество популярных мобильных приложений для трекинга спортивной активности и бега. Среди этих аналогов можно выделить наиболее известные и востребованные на рынке:

- «Strava»;
- «Nike Run Club»;
- «Runkeeper».

Рассмотрим приложение –Strava(рисунки 1– 6).



Рисунок 1 –Логотип приложения «Strava»

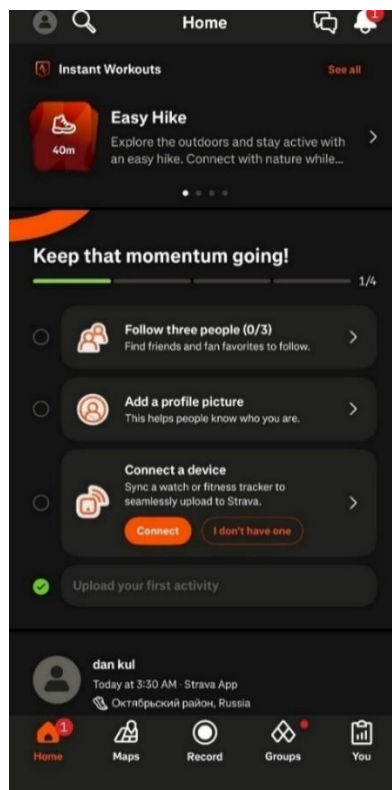


Рисунок 2—Домашнее меню

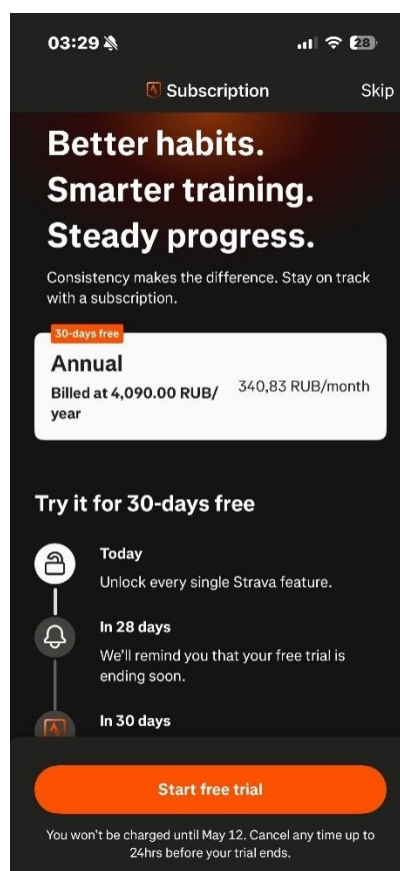


Рисунок 3–Надоедающая подписка

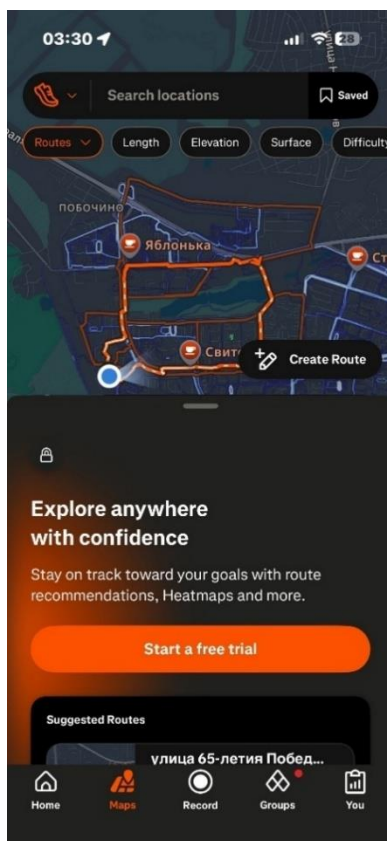


Рисунок 4–Вкладка карты

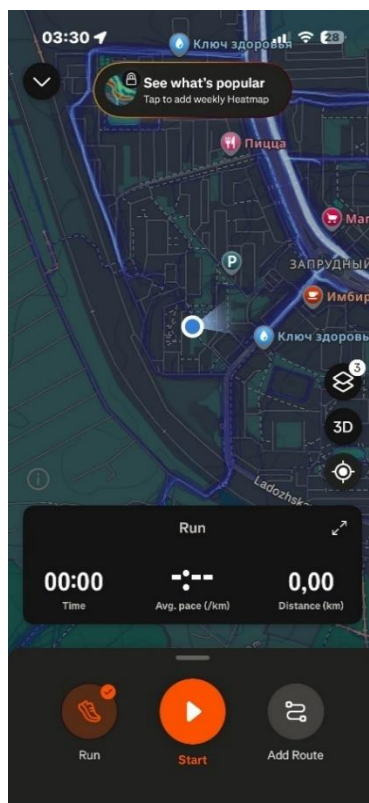


Рисунок 5—Интерфейс во время пробежки

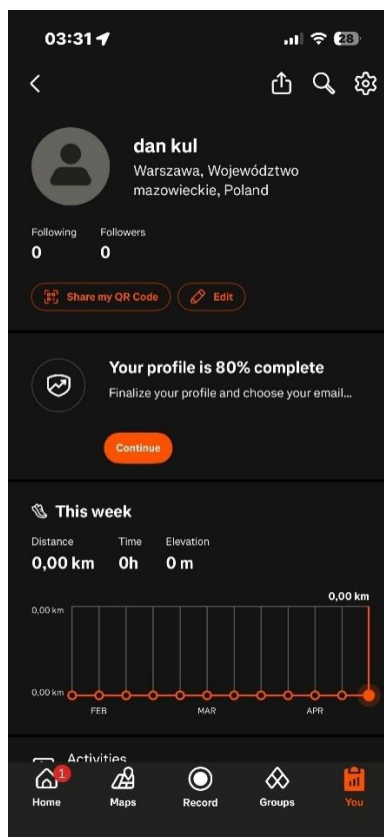


Рисунок 6—Статистика пользователя

Рассмотрим плюсы и минусы данного приложения.

Плюсы:

- огромная пользовательская база и развитая социальная сеть;
- высокая точность GPS—трекинга;
- подробный анализ преодоленных сегментов маршрута.

Минусы:

- перегруженный пользовательский интерфейс из—за огромного количества разных видов спорта;
- большинство продвинутых функций анализа скрыты за дорогостоящей платной подпиской;
- отсутствие функции прямого соревнования с собственными прошлыми забегами («призрак») в режиме реального времени на экране телефона;
- не работает в России без использования сторонних приложений.

Рассмотрим приложение –Nike Run Club(рисунки7– 11).



Рисунок 7–Логотип приложения «Nike Run Club»

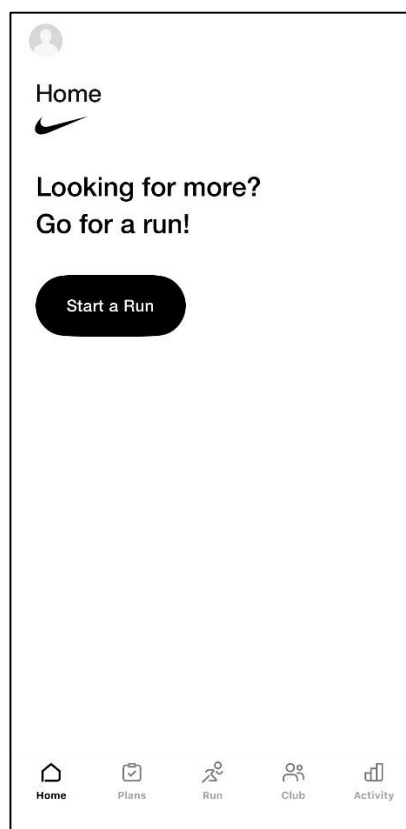


Рисунок 8– Интерфейс приложения

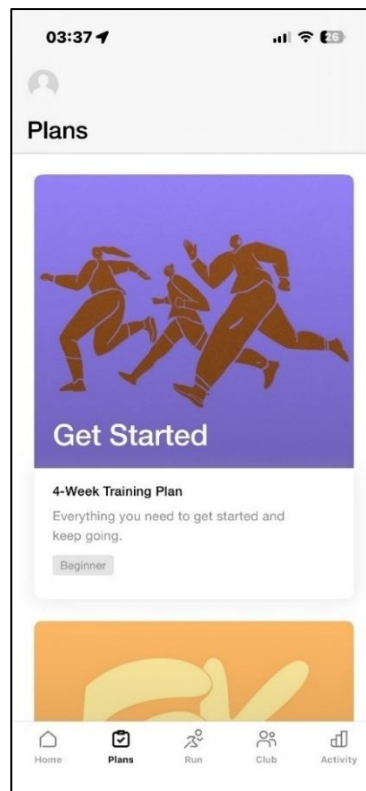


Рисунок 9—Планы для создания тренировок

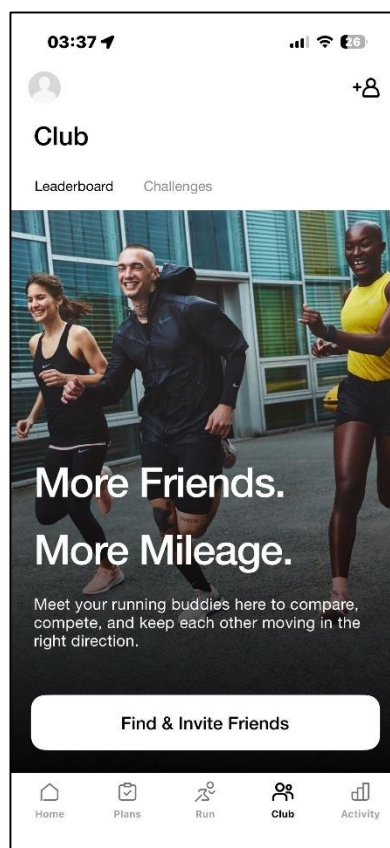


Рисунок 10—Вкладка клуб

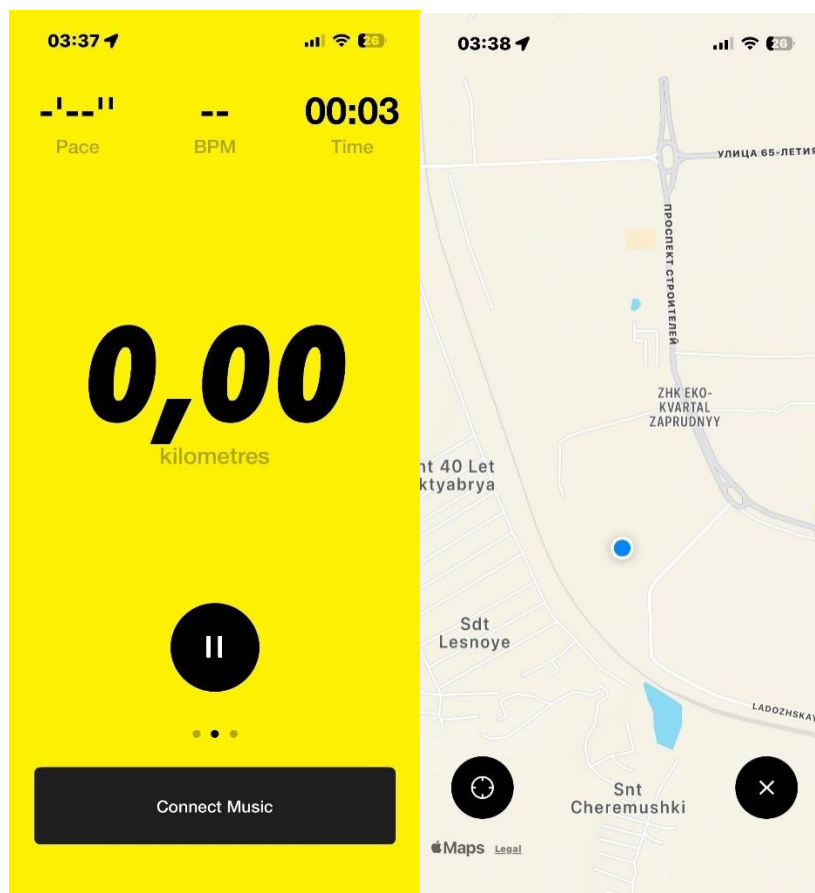


Рисунок 11–Интерфейс во время пробежки

Рассмотрим плюсы и минусы данного приложения.

Плюсы:

- современный, очень стильный и минималистичный интерфейс;
- отличная система мотивации (выдача наград за пробежки);

Минусы:

- закрытая экосистема, сильно привязанная к бренду (крайне сложно экспортировать свои данные тренировок);
- нестабильная работа на устройствах со слабым приёмником геолокации;
- фокус смещен на личные достижения без возможности интерактивных соревновательных "забегов–дуэлей" с друзьями на карте.

Рассмотрим приложение –Runkeeperот ASICS (рисунки 12–16).



Рисунок 12–Логотип приложения «Runkeeper»

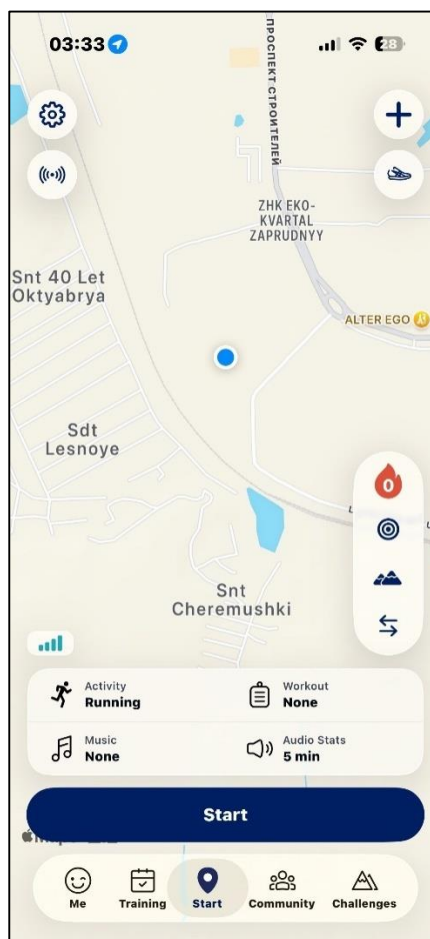


Рисунок 13–Интерфейс старта пробежки

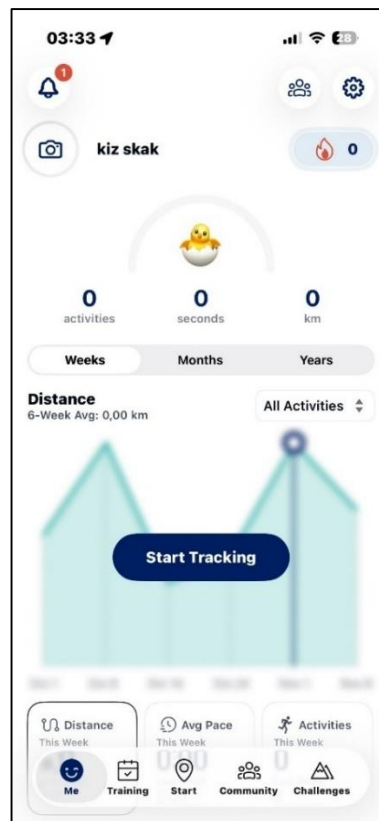


Рисунок 14–Меню статистики

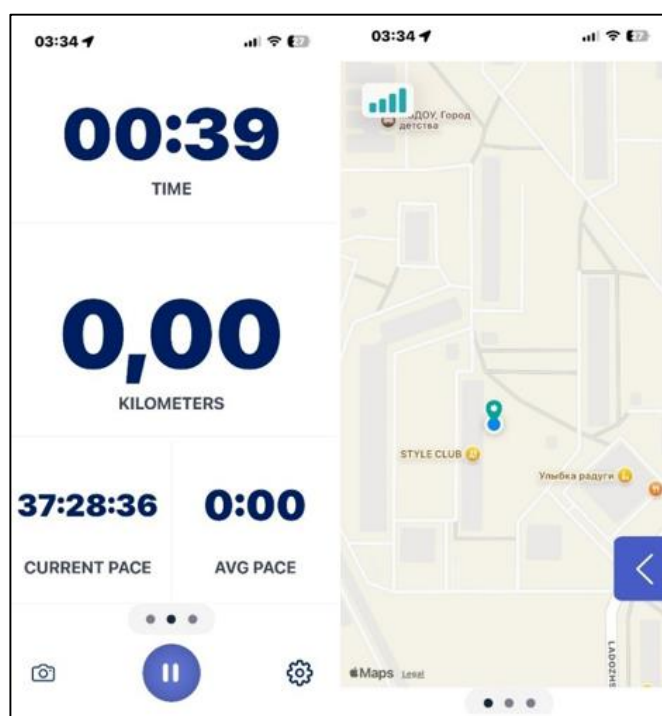


Рисунок 15–Интерфейс во время пробежки

Изм.	Лист	№докум.	Подпись	Дата

ПГУ 1.090301.39.001 ПЗ

Лист

17

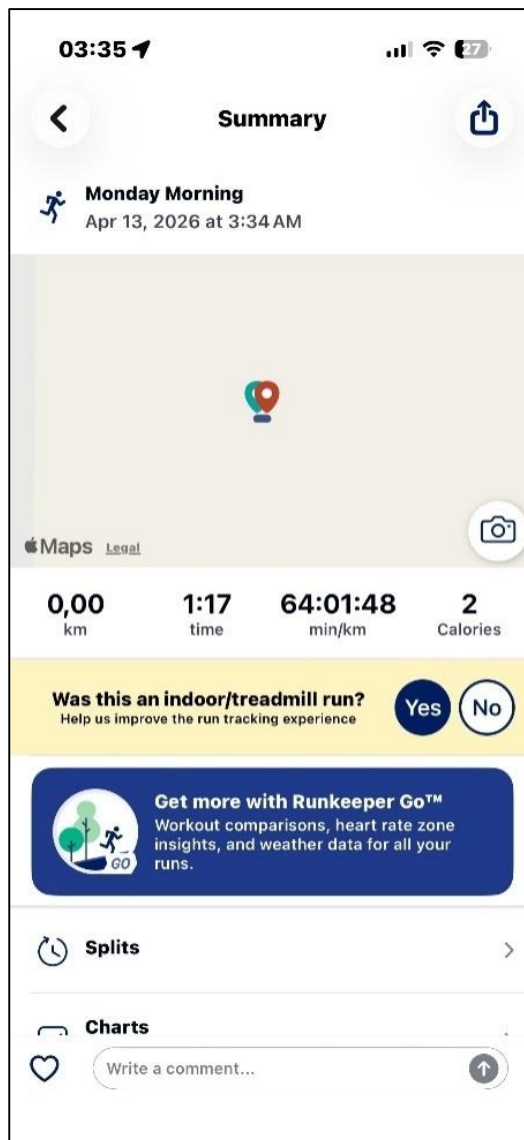


Рисунок 16–Статистика после тренировки

Рассмотрим плюсы и минусы данного приложения.

Плюсы:

- простая система запуска тренировки без лишних окон;
- малый вес приложения;
- базовое отслеживание работы полностью доступно офлайн.

Минусы:

- морально устаревший дизайн пользовательского интерфейса;
- слабая соревновательная механика между пользователями;
- навязчивая реклама премиум-версии;

Таблица 1 – Сравнение аналогов

П араметр	Strava	Nike Run Club	Runkeeper
Плюсы	Крупнейшее сообщество;	Стильный UI; Бесплатные аудиотренировки; Сильная геймификация.	Малый вес; Офлайн-доступ; Простой запуск.
Минусы	Перегруженный интерфейс; Дорогая подписка на продвинутую аналитику; Постоянно появляется реклама подписки; Не работает в России без использования сторонних приложений.	Привязка к бренду Nike; Проблемы с экспортом данных; Нет соревнований с друзьями на карте.	Устаревший дизайн; Навязчивая реклама подписки; Мало подключаемых устройств.

1.1.3 Постановка задачи

Цель – разработать Android-приложение для совместных беговых тренировок. Оно должно включать всебя GPS-трекинг пробежек, совместные беговые тренировки, режим виртуального соревнования, систему игровых достижений и инструменты для взаимодействия с другими пользователями.

Приложение должно соответствовать следующим критериям:

–понятный и простой интерфейс для комфортной навигации, а также современный и интуитивный дизайн;

–возможность записи, сохранения и просмотра истории забегов, а также управления профилем и списком друзей;

–реализация аналитических статистических отчетов для контроля спортивного прогресса пользователя;

–наличие уникальной функции «Призрак» длясоревнований с собственными рекордами в реальном времени;

–наличие уникальной возможности, если друг оказывается в радиусе 100 метров, обоим приходит push-уведомление с предложением совместной пробежки;

–приложение должно обладать необходимым базовым функционалом для полноценного мониторинга физической активности;

–интуитивно понятную авторизацию и регистрацию в приложении.

Для обеспечения корректной работы и стабильного функционирования всех модулей приложения, мобильное устройство пользователя должно соответствовать минимальным техническим требованиям. Учитывая использование современных инструментов разработки (JetpackCompose, Hilt) и необходимость высокоточного трекинга в фоновом режиме, были определены следующие требования:

– операционная система: Android 8.0 (API уровень 26) или выше. Данный лимит обусловлен использованием каналов уведомлений и ограничений на фоновые службы, введенных в этой версии ОС для оптимизации энергопотребления.

– аппаратные датчики: обязательное наличие модуляGPS. Рекомендуется наличие акселерометра и магнитометра для более плавной отрисовки положения пользователя на карте;

– параметры памяти: минимум 2 ГБ оперативной памяти для исключения принудительной остановки фонового сервиса геолокации системой Android.

– наличие сетевого соединения (Wi-Fi или 4G/LTE) для синхронизации данных с сервером Ktor, взаимодействия через WebSockets и загрузки сегментов карты Yandex MapKit.

Для правильной работы приложения пользователю необходимо предоставить приложению права на доступ к точному местоположению и доступ к местоположению в фоновом режиме («Всегда разрешать»), чтобы запись трекинга не прерывалась при сворачивании приложения или выключении экрана.

На рисунке 17 представлена UML-диаграмма прецедентов.

					ПГУ 1.090301.39.001 ПЗ	Лист
						21
Изм.	Лист	№докум.	Подпись	Дата		

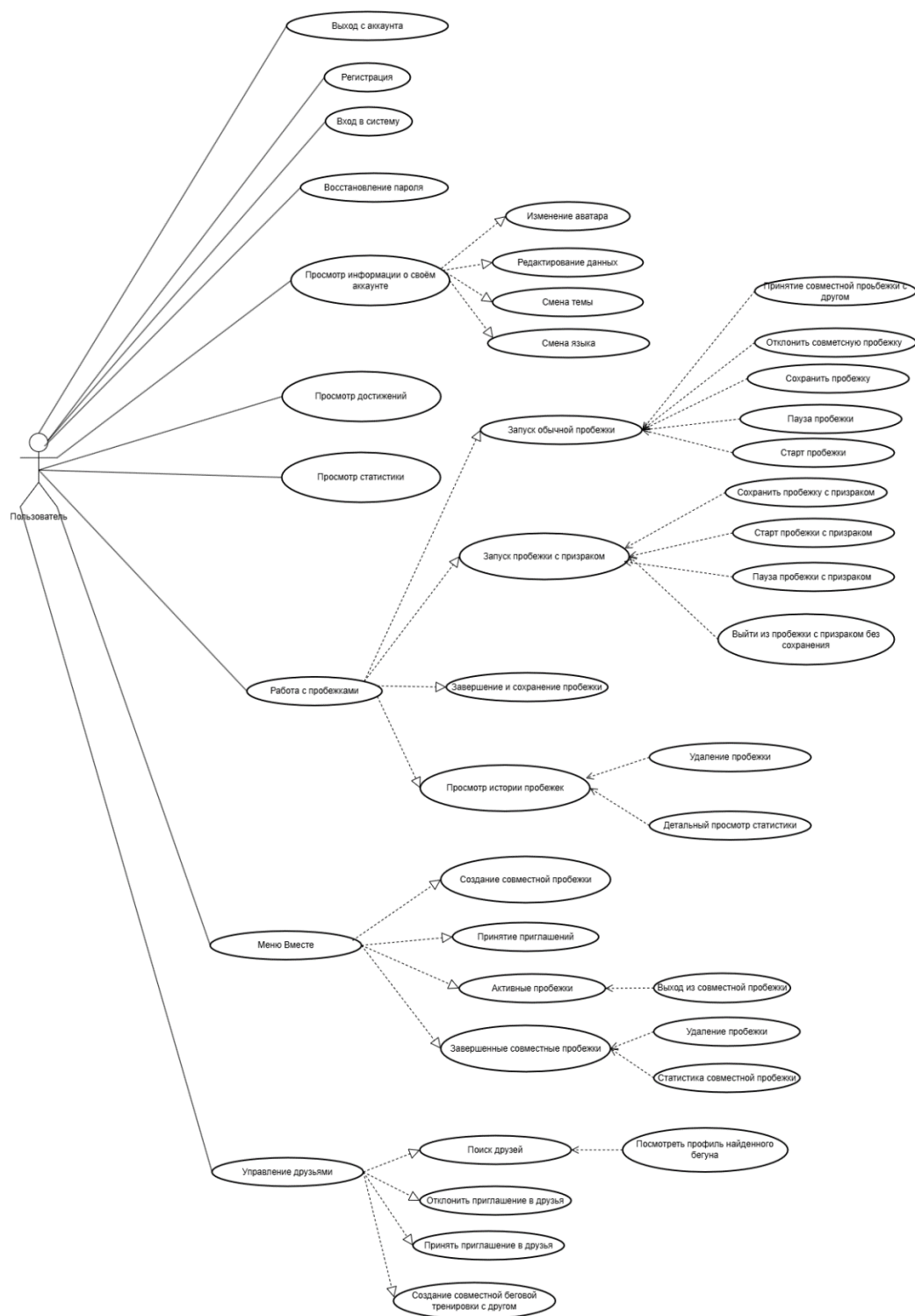


Рисунок 17 – UML-диаграмма прецедентов

На основе UML-диаграммы прецедентов можно выделить следующие основные сценарии взаимодействия:

– сценарии управления доступом и профилем:

а) регистрация и вход: новый пользователь создает учетную запись, данные которой сохраняются в удалённой БД PostgreSQL. Авторизованный пользователь получает доступ к приложению;

б) настройка аккаунта: пользователь может индивидуализировать интерфейс, изменяя аватар, выбирая цветовую тему приложения (светлая/темная) или язык интерфейса.

– сценарии мониторинга тренировок (работа с пробежками):

а) обычная пробежка: базовый сценарий, включающий старт, возможность постановки на паузу и последующее сохранение пробежки. Во время бега система отслеживает темп и дистанцию в реальном времени. В добавок если друг при запуске обычной пробежки попадает в радиус 100 метров, то обоим приходит уведомление о начале совместной пробежке;

б) пробежка с «призраком»: расширенный сценарий, в котором пользователь выбирает один из своих прошлых забегов. Система одновременно отрисовывает текущую позицию бегуна и позицию «виртуального соперника», позволяя визуально оценивать отставание или опережение графика;

в) управление историей: возможность детального просмотра карты и графиков скорости для любой из завершённых тренировок, а также удаление неактуальных записей.

– сценарии совместного взаимодействия и группового бега:

а) управление друзьями: поиск других пользователей по имени, отправка и принятие приглашений в друзья;

б) меню «Вместе»: организация совместных тренировок. Инициатор создает «встречу», а друзья получают уведомления. В процессе такого забега через WebSocket-соединение координаты всех участников отображаются на одной карте в реальном времени;

в) система достижений: автоматическая выдача цифровых наград при достижении определенных спортивных целей (например, первые 10 км или 5 дней бега подряд).

Данный набор прецедентов позволяет полностью реализовать концепцию бегового приложения для совместных тренировок, обеспечивая как техническую точность измерений, так и высокую вовлеченность пользователя.

1.2 Выбор инструментов разработки

Выбор правильного инструментария является фундаментом успешной реализации проекта. Для разработки приложения был проведен сравнительный анализ доступных сред разработки, языков программирования, систем управления базами данных и архитектурных паттернов.

1.2.1 Выбор среды разработки

Основным инструментом для создания мобильного приложения является интегрированная среда разработки. Рассмотрим три наиболее распространенных решения: Microsoft Visual Studio, IntelliJ IDEA и Android Studio.

Microsoft Visual Studio— мощная универсальная среда от Microsoft.

Плюсы:

- поддержка множества языков (C#, C++, Python);
- отличные инструменты для кроссплатформенной разработки.

Минусы:

- перегруженность интерфейса для специфических задач Android;
- высокие системные требования, сложность настройки нативного SDK.

IntelliJ IDEA—флагманская IDE от JetBrains.

Плюсы:

- мощный статический анализ кода;
- поддержка множества Java-фреймворков (Spring, Ktor);
- высокая стабильность.

Минусы:

- требует покупки платной версии (Ultimate) для доступа ко всем функциям;
- потребляет значительный объем оперативной памяти.

Android Studio– специализированная IDE от Google, построенная на базе IntelliJ IDEA Community Edition.

Плюсы:

- полная интеграция с Android SDK;
- встроенный визуальный редактор LayoutInspector;
- поддержкаGradleиJetpackCompose;
- бесплатность.

Выбор минимального уровня, поддерживаемого API, определяется данными об использовании различных версий ОС Android[1].

Минусом являетсятребовательность к ресурсам компьютера.

Выбор AndroidStudioобусловлен тем, что данная среда является официальной рекомендацией Googleдля платформы Android. Она обеспечивает полную поддержку языка Kotlin, а также интеграцию со встроенным эмулятором и средствами отладки, необходимым для тестирования взаимодействия с сервисами GooglePlay.

1.2.2 Выбор языка программирования

Для Android-разработки часто рассматриваются два языка: Java и Kotlin.

Java– проверенный временем объектно-ориентированный язык.

Плюсы:

- огромная база знаний;
- стабильность;
- строгая типизация.

Минусы:

- большое количество кода (boilerplate);
- отсутствие встроенной безопасности по null-ссылкам (Null Safety).

Kotlin– современный статически типизированный язык от JetBrains[2].

Плюсы:

- лаконичность;
- полная совместимость с Java;

- встроенная защита от NullPointerException;
- легкое использование многопоточности.

Минусом является более долгое время компиляции в некоторых случаях и необходимость изучения новых синтаксических конструкций.

Обоснование выбора: выбор сделан в пользу Kotlin. Его современные механизмы позволяют писать безопасный и читаемый код, а официальный статус от Google гарантирует получение обновлений и новых библиотек Jetpack в первую очередь на этом языке.

1.2.3 Выбор инструментов хранения данных и контейнеризации

Для реализации полнофункциональной системы, включающей мобильное приложение и серверную часть, необходимо было выбрать решения для локального и удаленного хранения данных, а также способ развертывания инфраструктуры.

Для локального хранения данных была выбрана библиотека Room, которая используется для кэширования данных на устройстве [3], что позволяет приложению работать в офлайн-режиме и быстро отображать историю забегов.

В качестве серверного хранилища данных была выбрана СУБД PostgreSQL. Такое решение связано с необходимостью надёжно хранить значительные объёмы информации о пользователях, их социальных связях и синхронизированных маршрутах, обеспечивая при этом высокий уровень целостности данных [4].

Для контейнеризации был выбран Docker, ниже рассмотрим плюсы и минусы.

Плюсы:

- изоляция зависимостей;
- гарантия идентичности окружения на этапе разработки и в продакшене;
- легкость развертывания через CI/CD.

Минусом является значительный объем занимаемого дискового пространства образами.

Выбор Использование Docker позволяет упаковать бэкенд на Ktor и базу данных PostgreSQL в легкие контейнеры. Это упрощает процесс развертывания на облачных платформах (например, Railway) и исключает ошибки, связанные с различием версий ПО в разных средах [5].

1.2.4 Выбор архитектуры приложения

Архитектура определяет масштабируемость и тестируемость проекта. Были рассмотрены паттерны MVC и MVVM[6].

Рассмотрим плюсы и минусы MVC архитектуры.

Плюсы:

- простота понимания для небольших проектов;
- скорость разработки.

Минусы:

- контроллер становится перегруженным, что затрудняет тестирование;
- трудности масштабирования.

Рассмотрим плюсы и минусы MVVM (Model-View-ViewModel) архитектуры.

Плюсы:

- четкое разделение бизнес-логики и UI;
- ViewModel сохраняет состояние при повороте экрана;
- идеальная совместимость с Jetpack Compose через StateFlow.

Минусы:

- высокий порог вхождения;
- необходимость использования механизмов наблюдения за данными.

После рассмотрения плюсов и минусов была выбрана архитектура MVVM совместно с принципами CleanArchitecture[7]. Это позволяет изолировать логику трекинга и соревнований от пользовательского интерфейса, обеспечивая высокую надежность и чистоту кода.

Схема паттерна архитектуры MVVM представлена на рисунке 18.

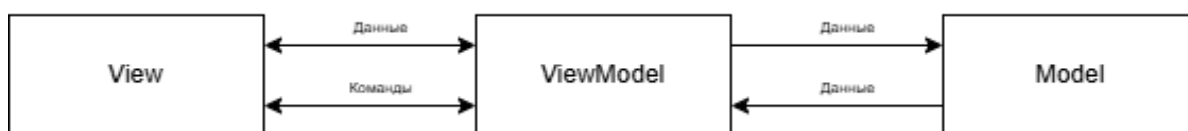


Рисунок 18 –Схема паттерна архитектуры MVVM

1.2.5 Выбор инструментов построения интерфейса

Традиционным подходом к построению UI в Android является использование XML-разметки. Однако в данном проекте был сделан выбор в пользу современной библиотеки Jetpack Compose.

Jetpack Compose – современный инструментарий для создания нативного интерфейса с использованием декларативного подхода.

Плюсы:

- написание UI полностью на Kotlin;
- автоматическое обновление интерфейса при изменении состояния;
- высокая скорость разработки сложных анимаций, интеграция с Material Design 3.

Минусы:

- необходимость пересмотра традиционных подходов к верстке;
- более высокие требования к версии Gradle и Kotlin.

Обоснование выбора: использование Jetpack Compose позволяет сократить объем кода и сделать интерфейс приложения максимально динамичным и отзывчивым, что крайне важно для отображения телеметрии бега в реальном времени.

1.2.6 Выбор фреймворка для внедрения зависимостей

В рамках проекта применяется Hilt – Android-библиотека, созданная на основе Dagger и нацеленная на упрощение механизма внедрения зависимостей.

Выбор в пользу данного инструмента обусловлен тем, что Hilt даёт возможность реализовать принцип инверсии управления, делая архитектуру приложения более гибкой и масштабируемой. Кроме того, фреймворк

обеспечивает удобную замену конкретных реализаций репозиториев на тестовые заглушки, что существенно упрощает модульное тестирование.

1.2.7 Выбор картографических сервисов

Для работы с картами был выбран Yandex MapKit SDK.

Плюсы:

- высокая детализация карт на территории РФ;
- поддержка офлайн–карт, плавный рендеринг треков;
- наличие подробной документации на русском языке.

Минусы:

- закрытый исходный код;
- необходимость получения API–ключа для использования.

Обоснование выбора: YandexMapkit обеспечивает лучшую производительность и точность отображения маршрутов.

1.3 Реализация приложения

1.3.1 Базовая конфигурация приложения

Главный файл настроек AndroidManifest.xml находится в корне папки с ресурсами. В нём прописана основная точка входа в программу (MainActivity), параметры запуска, а также запрошены нужные разрешения от системы, такие как доступ к точному местоположению и право выходить в интернет. В папке xml/ лежат дополнительные файлы с конфигурациями:

– backup_rules.xml – этот файл объясняет системе Android, что именно из данных приложения нужно копировать в облако Google Диска при создании резервной копии, а что копировать не следует;

– network_security_config.xml – здесь прописаны настройки безопасности для подключения к нашему серверу (Ktor), чтобы можно было настроить шифрование и доверенные сертификаты.

1.3.1.1 Ресурсы значений и локализация

В папках `values/` и аналогичных им хранятся строки, цвета и другие постоянные величины. Благодаря им приложение умеет переключать языки и внешний вид в зависимости от настроек телефона.

– `values/strings.xml` – хранит все тексты на русском языке (по умолчанию);

– `values-en/strings.xml` – точно такой же файл, но с переводом на английский. Система сама выберет его, если у пользователя в телефоне включён английский язык.

– `colors.xml` – перечень всех цветов, которые используются в приложении.

Это позволяет менять палитру в одном месте, а не по всему коду.

– `values/themes.xml` и `values-night/themes.xml` – отвечают за то, как приложение выглядит днём (светлая тема) и ночью (тёмная тема).

1.3.1.2 Векторная графика и иконки

Для того чтобы иконки и картинки выглядели чётко на любых экранах и при этом не занимали много места в памяти телефона, используется векторная графика в формате XML.

В папке `drawable/` лежат математические описания иконок бегунов, маркеров на карте для «призраков» и других декоративных элементов. Это гораздо экономичнее, чем хранить десятки разных PNG-картинок под разные размеры экранов.

В папке `miramar-anydpi-v26/` хранятся файлы `ic_launcher.xml` и `ic_launcher_round.xml`. Они нужны для того, чтобы система (начиная с Android 8) могла сама подгонять форму иконки приложения под стиль – например, делать её круглой или квадратной со скруглёнными углами.

1.3.2 Структура пакетов и модулей

Чтобы код приложения оставался понятным и его можно было легко тестировать, весь проект разбит на слои по принципу Clean Architecture. Всего таких слоёв три:

–слой данных (Data) – занимается получением информации отовсюду: из базы данных, из интернета, с датчиков телефона;

–доменный слой (Domain) –это ядро бизнес–логики приложения.В нём реализованы все ключевые алгоритмы:как считать темп бега, когда выдавать награды и так далее;

–слой представления (Presentation) – это то, что видит пользователь. Экран, кнопки, карта, графики.

Такое разделение позволяет менять дизайн или даже базу данных, не ломая при этом основные функции программы.

1.3.3 Слой данных

Слой данных отвечает за надёжную работу с информацией при нестабильном сетевом соединении или её нет совсем. Используется подход «Offline–first» (сначала работаем локально). Это значит, что пользователь сначала сохраняет данные в память телефона, а уже потом они в фоне отправляются в облако. Так приложение не зависает, если вы бежите в парке с плохим интернетом. На рисунке 19 представлено UML–диаграмма взаимодействия классов слоя данных.

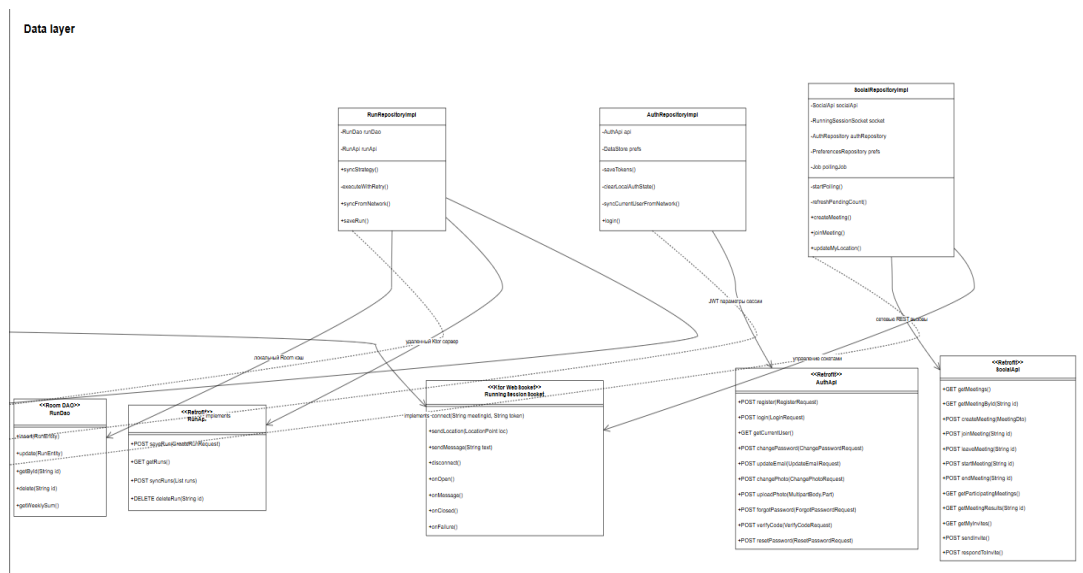


Рисунок 19 –UML–диаграмма классов слоя данных

Как устроено локальное хранилище (папка data.local):

–runsocialdatabase.kt– главный класс базы данных Room. Он управляет созданием таблиц и обновлением их структуры при выходе новых версий приложения. Чтобы хранить сложные данные о координатах маршрута (списки точек), встроен специальный конвертер, превращающий их в текст формата JSON.

Сущности (папка entity/):

–runentity.kt– хранит всю информацию о пробежке: когда начали, сколько пробежали, средний темп, калории и статус отправки на сервер;

–userentity.kt– копия данных о пользователе, чтобы лишний раз не загружать их из интернета;

–ghosttrackentity.kt– сохранённые маршруты для функции «Призрак».

Объекты доступа (папка dao/): здесь лежат интерфейсы RunDao, UserDao, GhostDao. В них описаны SQL-запросы. Библиотека Room сама превращает их в работающий код, что исключает ошибки ручного написания запросов.

Как устроена работа с интернетом (папка data.remote):

–сетевые интерфейсы (папка api/): это описание того, какие запросы мы можем отправить на сервер. Всё разложено по полочкам: AuthApi.kt для входа и регистрации, RunApi.kt для обмена данными о пробежках, SocialApi.kt для общения с друзьями, GhostTrack.kt для загрузки «призрачных» треков;

–объекты передачи (папка dto/): это классы-контейнеры для приёма и отправки данных в формате JSON. Благодаря им изменения на сервере не нарушат работу приложения;

–websocket-соединение: RunningSessionSocket.kt позволяет клиенту и серверу общаться в реальном времени, обмениваясь координатами во время совместного забега.

Репозитории (папка data.repository):

–runrepositoryImpl.kt– следит за тем, чтобы пробежка сначала сохранилась в телефоне, а потом гарантированно ушла на сервер (если не получилось с первого раза, система повторит попытку в фоне);

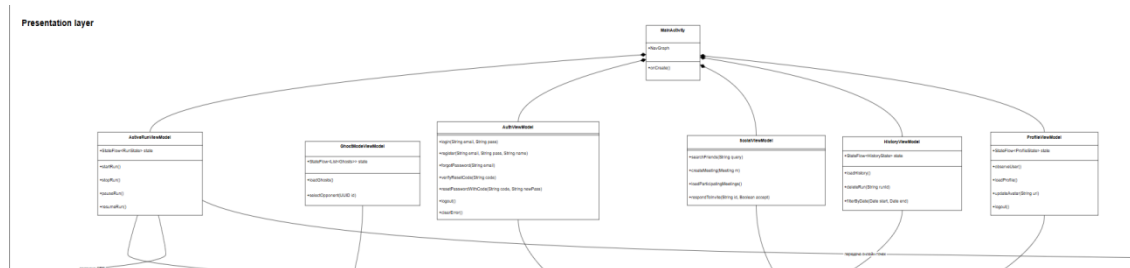


Рисунок 21 –UML–диаграмма классов слоя представления

Как управляется состояние экрана:

Экран не хранит данные у себя. управляют данными модели представления (HomeViewModel, ActiveRunViewModel). Они хранят текущее состояние и сообщают экрану, когда нужно перерисоваться (например, пришла новая точка GPS). Нажатия кнопок тоже уходят в эти модели, а не обрабатываются на экране.

За переходы между экранами отвечает файл NavGraph.kt. Приложение разбито на логические части:

- authgraph– экраны входа, регистрации и восстановления пароля.
- maingraph– главный экран с нижним меню (профиль, история, друзья).
- activerungraph– режим активной тренировки, где нельзя случайно нажать «Назад» и свернуть карту.

Основные экраны:

–activeRunScreen.kt– показывает карту Яндекс.Карт и накладывает на неё прозрачные панели со скоростью и расстоянием. Подписка на GPS работает постоянно, даже при выключенном экране;

–ghostmodescreen.kt– экран выбора виртуального соперника. Здесь можно сравнить свой темп с рекордным и посмотреть, есть ли шансы победить «призрака»;

–historyScreen.kt– красивый вывод статистики прошлых тренировок с графиками темпа и набором высоты;

–friendsScreen.kt– поиск других бегунов, просмотр их профилей и таблица лидеров.

1.3.6 Сервисы, утилиты и внедрение зависимостей

Чтобы приложение могло работать в фоне, не выключаясь при блокировке экрана, используются системные сервисы Android. На рисунке 22 представлено UML–диаграмма взаимодействие классов системных сервисов.

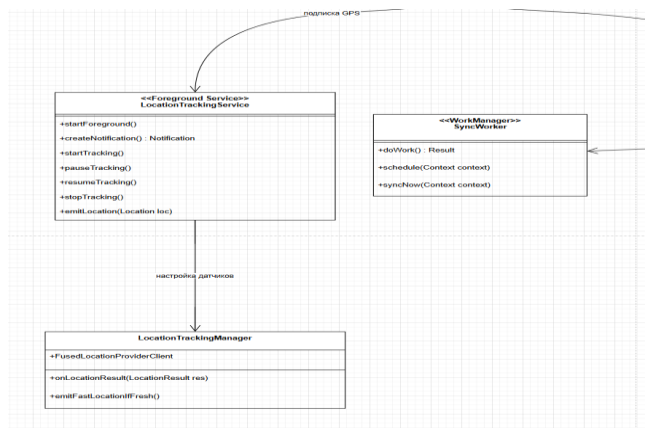


Рисунок 22 –UML–диаграмма классов системных сервисов

Системные сервисы:

–**LocationTrackingService.kt**– служба переднего плана. Она висит в шторке уведомлений, показывает текущий темп и гарантирует, что система не завершит процесс приложения принудительно во время записи трека;

–**LocationTrackingManager.kt**– класс–помощник для работы с GPS. Он настраивает частоту обновления координат и превращает системные вызовы в удобный поток данных.

Внедрение зависимостей (Hilt):

Чтобы исключить ручное создание и внедрение зависимостей используется библиотека Hilt. Она автоматически подставляет нужные компоненты туда, где они требуются. Это ускоряет разработку и упрощает замену компонентов. За создание сетевых клиентов отвечает **NetworkModule.kt**, за базу данных – **DatabaseModule.kt**.

1.3.7 Реализация серверной части

Помимо телефона, работает мощный сервер, написанный на том же языке Kotlin с использованием фреймворка Ktor. Такой подход позволил использовать

одни и те же классы (например, описание пробежки) и на телефоне, и на сервере, что исключило дублирование и возможные ошибки синхронизации.

Маршрутизация и защита:

Все запросы к серверу разделены по темам (AuthRoutes, RunRoutes и т.д.). Доступ к личным данным открыт только после проверки электронного ключа (JWT–токена). Пароли пользователей не хранятся в открытом виде, а превращаются в необратимый криптографический слепок с помощью алгоритма BCrypt.

База данных на сервере:

Используется база данных PostgreSQL, а для удобной и безопасной работы с ней из кода – библиотека Exposed [8]. Она позволяет писать запросы на Kotlin, а не на языке SQL, защищая от ошибок и взломов.

Совместный бег в реальном времени:

Для того чтобы вы могли видеть на карте перемещения друга, используется протокол WebSocket. Сервер держит постоянное подключение с каждым телефоном в группе. Как только один бегун делает шаг и его телефон отправляет координату, сервер пересылает её с минимальной задержкой (менее 30 миллисекунд) всем остальным участникам.

Для обеспечения безопасности передачи данных в реальном времени используется протокол WebSocketSecure (WSS). Клиентское приложение устанавливает зашифрованное соединение по адресу: `wss://scala-run-production.up.railway.app/ws/run-session/{meetingId}`.

Механизмы безопасности:

- шифрование транспортного уровня: все данные между клиентом и сервером шифруются с использованием TLS;
- аутентификация: каждое WebSocket–соединение требует валидации JWT–токена в заголовке Authorization;
- авторизация: доступ к WebSocket–эндпоинтам разрешен только аутентифицированным пользователям;

Клиентское устройство – вычислительный узел конечного пользователя, на котором функционирует мобильное приложение. Клиентская часть реализует пользовательский интерфейс, обработку действий пользователя, отображение данных и формирование сетевых запросов к серверной части системы. Приложение взаимодействует с локальной базой данных для хранения и получения данных без необходимости постоянного подключения к сети.

Локальная база данных – компонент хранения данных, размещённый на клиентском устройстве. Реализуется с использованием технологии Room, работающей поверх встроенной СУБД SQLite[3]. Локальная база данных используется для кэширования информации, хранения пользовательских данных, результатов тренировок и другой служебной информации. Наличие локального хранилища обеспечивает повышение производительности приложения и поддержку работы в офлайн-режиме.

Сервер приложения–серверная часть построена на фреймворкеKtor, задачами которого являются обработка клиентских запросов, реализация бизнес-логики, синхронизация данных между пользователями и взаимодействие с удалённой базой данных.

Удалённая база данных – централизованное хранилище данных системы, реализованное на базе системы управления базами данных PostgreSQL. Данный компонент предназначен для долговременного хранения информации всех пользователей, включая учетные записи, статистику активности, результаты тренировок и другие данные, необходимые для функционирования приложения.

Взаимодействие между узлами системы осуществляется через сетевые протоколы передачи данных. Клиентское приложение отправляет запросы на сервер приложений посредством REST API или WebSocket-соединений через сеть Интернет. Сервер, в свою очередь, выполняет обработку полученных данных и сохраняет их в удалённой базе данных PostgreSQL.

Для обмена данными в реальном времени во время совместных пробежек применяется протоколWSS. Данный протокол обеспечивает двустороннюю связь

с минимальной задержкой и шифрование передаваемых координат GPS и метаданных участников. WebSocket-соединения защищены JWT-аутентификацией и изолированы по уникальным идентификаторам заботов (meeting_Id).

Архитектура программного продукта реализует стратегию Offline-first, обеспечивающую устойчивую работу приложения при нестабильном сетевом соединении. Согласно данной стратегии, все изменения данных, инициированные пользователем, сначала сохраняются в локальной базе данных Room. После восстановления или наличия сетевого соединения данные асинхронно синхронизируются с сервером приложений, который обеспечивает их обработку и запись в централизованное хранилище PostgreSQL.

Применение данной архитектурной модели позволяет обеспечить высокую отказоустойчивость системы, уменьшить задержки при работе с данными, снизить нагрузку на серверную часть и повысить удобство использования приложения для конечных пользователей.

Диаграмма последовательностей представлена на рисунке 25.

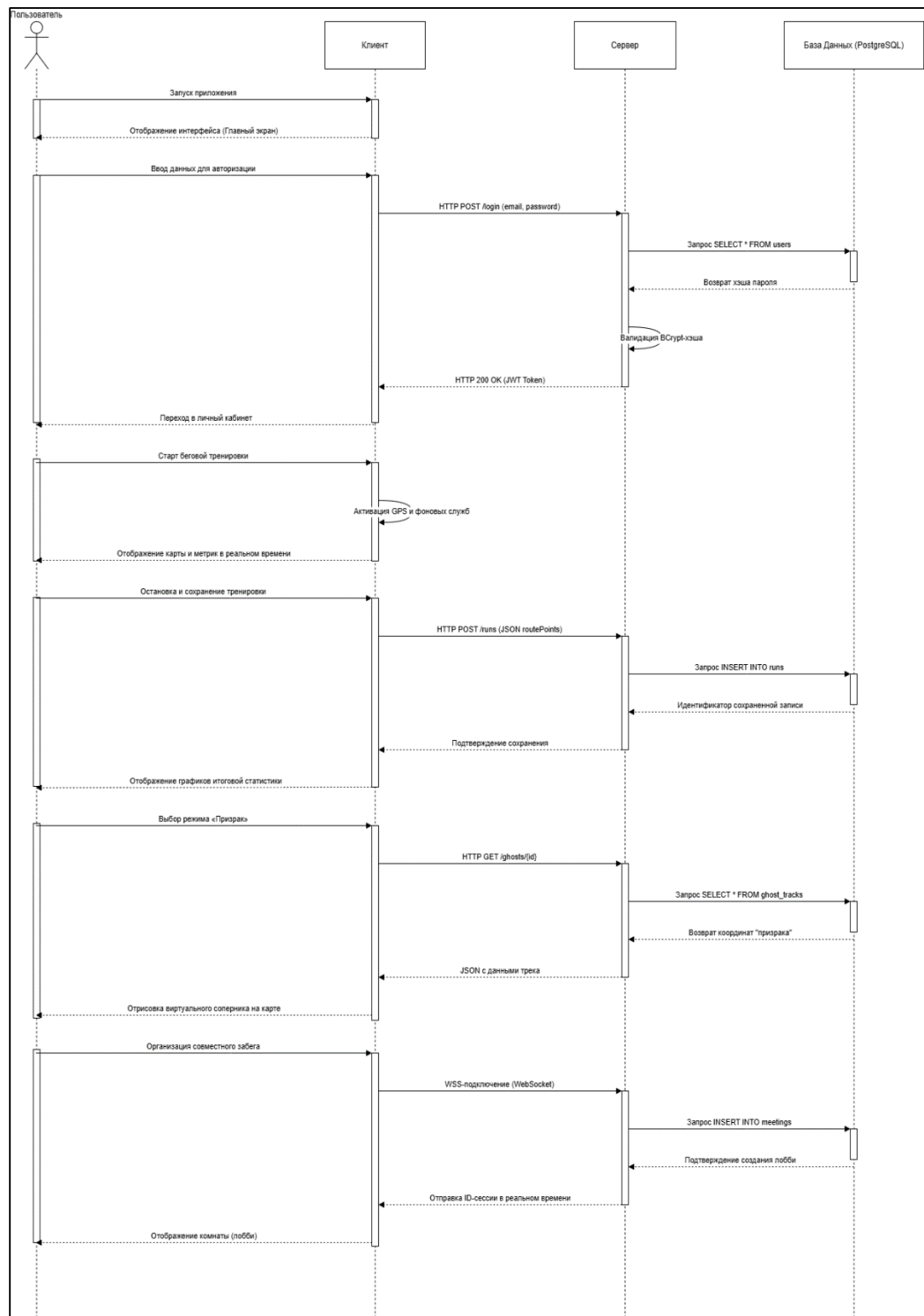


Рисунок 25–UML-диаграмма последовательностей

1.3.9 Структура баз данных

Для реализации полнофункциональной системы была спроектирована распределенная архитектура данных, состоящая из локального хранилища на мобильном устройстве и централизованной облачной базы данных. Это

обеспечивает возможность работы в автономном режиме с последующей синхронизацией.

1.3.9.1 Локальная база данных

Локальное хранилище используется для кэширования профиля пользователя и обеспечения быстрого доступа к истории личных достижений. Основные атрибуты сущностей Room представлены на таблицах 2 – 5:

Таблица 2– Атрибуты сущности UserEntity

Поле	Тип данных	Описание
id	String(UUID)	Уникальный идентификатор пользователя
email	Varchar	Электронная почта
displayName	Varchar	Имя для отображения
level	Varchar	Уровень (BEGINNER, PRO ит.д.)
weeklyDistanceMeters	Double	Дистанция за текущую неделю
totalDistanceMeters	Double	Суммарная дистанция за всё время

Таблица3–АтрибутысущностиRunEntity

Поле	Тип данных	Описание
id	String(UUID)	Уникальный идентификатор забега
userId	String(UUID)	ID владельца записи
distanceMeters	Double	Расстояние в метрах
durationMillis	Long	Длительность в миллисекундах
routePointsJson	Text (JSON)	Координаты маршрута для отрисовки
syncStatus	Varchar	Статус синхронизации с сервером

Таблица 4– Атрибуты сущности GhostTrackEntity

Поле	Тип данных	Описание
id	String(UUID)	ID записи «призрака»
totalTimeMillis	Long	Время эталонного забега
pointsJson	Text(JSON)	Точки телеметрии «призрака»

Поле	Тип данных	Описание
isPersonalBest	Boolean	Флаг лучшего результата

Таблица5–АтрибутысущностиAchievementEntity

Поле	Тип данных	Описание
id	String	Уникальный коддостижения
title	Varchar	Название награды
isUnlocked	Boolean	Статус получения

На рисунке 26 представлена схема связей всех таблиц в локальной базе данных ROOM.

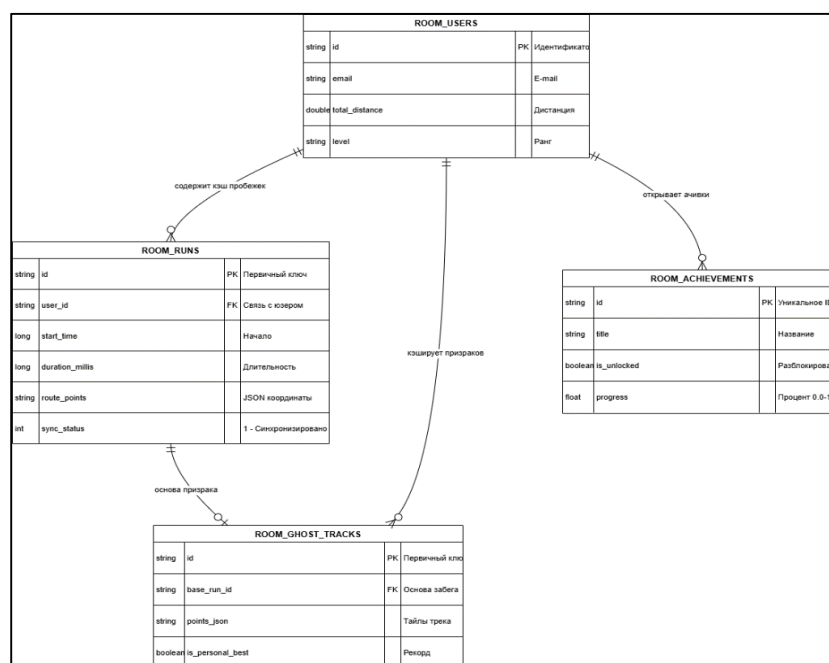


Рисунок 26 – Схема связей таблиц в базе данных ROOM

1.3.9.2 Серверная база данных

Серверная база данных служит единым источником достоверных данных для всех пользователейиобеспечивает механизмы совместного

взаимодействия. На таблицах 6 – 10 представлены атрибуты сущностей удалённой базы данных (PostgreSQL).

Таблица 6– Атрибуты сущности Users

Поле	Тип данных	Описание
id	UUID	Уникальный идентификатор пользователя
email	Varchar	Электронная почта (логин)
password_hash	Varchar	Хэш пароля для безопасной авторизации
display_name	Varchar	Имя для отображения в профиле и рейтингах
avatar_url	Varchar	Ссылка на изображение профиля (для медиа–конкурсов)
level	Varchar	Уровень активности
weekly_distance_meters	Double	Пройденная дистанция за текущую неделю
total_distance_meters	Double	Суммарная пройденная дистанция

Таблица 7– Атрибуты сущности Friendships

Поле	Тип данных	Описание
user_id	UUID	ID инициатора связи
friend_id	UUID	ID целевого пользователя
status	Varchar	Статус
created_at	DateTime	Дата отправки запроса

Таблица 8– Атрибуты сущности Meetings

Поле	Тип данных	Описание
id	UUID	Уникальный ID встречи
creator_id	UUID	Организатор тренировки
title	Varchar	Название мероприятия
scheduled_time	DateTime	Запланированное время старта

Поле	Тип данных	Описание
status	Varchar	Состояние

Таблица 9– Атрибуты сущности Participants

Поле	Тип данных	Описание
meeting_id	UUID	Идентификатор встречи
user_id	UUID	Идентификатор участника
joined_at	DateTime	Время присоединения к группе

Таблица 10– Атрибуты сущности MeetingInvitations

Поле	Тип данных	Описание
id	UUID	ID приглашения
meeting_id	UUID	Ссылка на мероприятие
invitee_id	UUID	Кому адресовано приглашение

На рисунке 27 представлена схема связей всех таблиц в удаленной базе данных PostgrgreSQL.

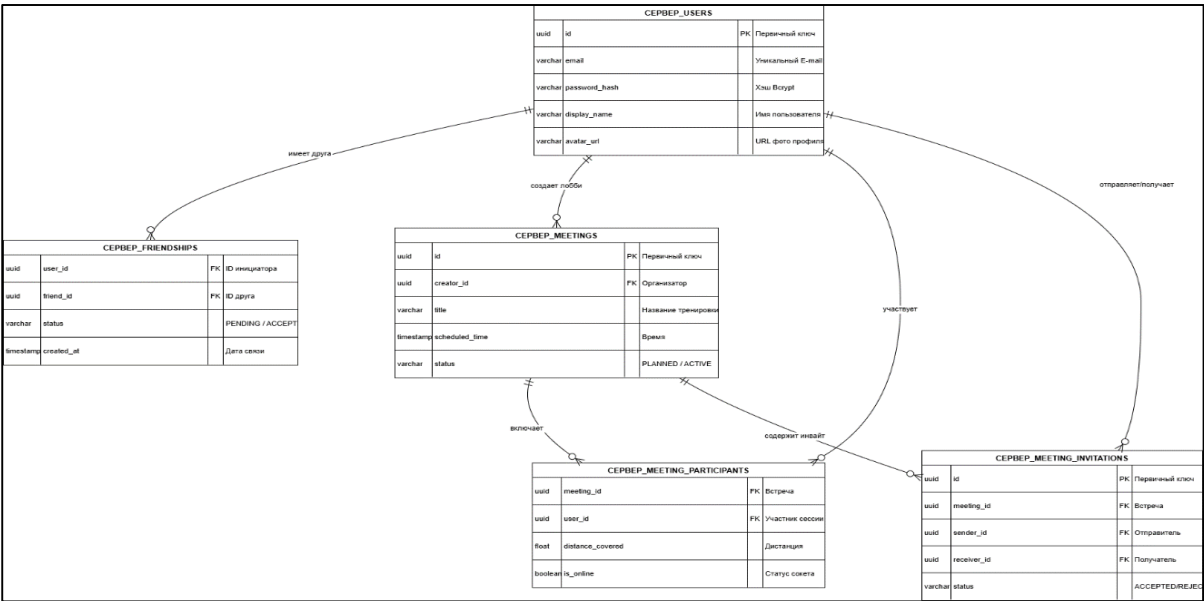


Рисунок 27 – Схема связей таблиц в базе данных PostgreSQL

1.4 Тестирование

В данном разделе описываются методы тестирования, примененные для проверки работоспособности всех модулей системы. Тестирование проводилось на реальном устройстве Xiaomi Redmi Note 9 (Android 11) в полевых условиях. На рисунке 28 представлена схема использования приложения.

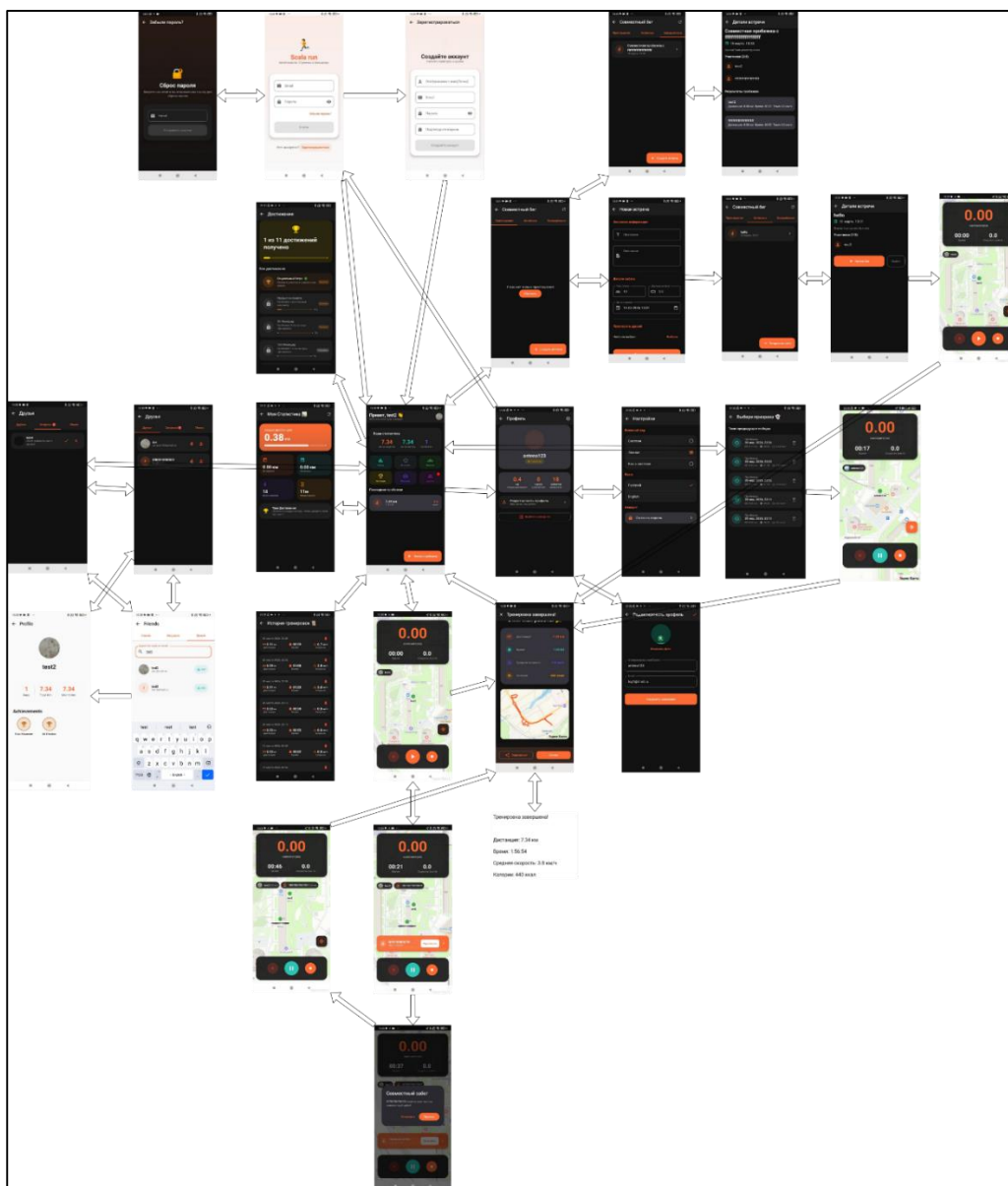


Рисунок 28 – Схема взаимодействия приложения

1.4.1 Регистрация и авторизация пользователя

Было проведено тестирование модуля авторизации: проверка корректности создания профиля, восстановления пароля через EmailService и валидации JWT–токенов.

На рисунке 29 представлен экран при входе в приложение. Интерфейс содержит поля для ввода электронной почты и пароля, кнопку «Войти», гиперссылку «Забыли пароль?» для восстановления доступа к аккаунту, а также кнопку «Зарегистрироваться», которая перенаправляет пользователя на форму создания учётной записи. В верхней области экрана расположены логотип приложения и его девиз «Бегай вместе. Стремись к большему». Рисунок 30 демонстрирует экранную форму для регистрации нового пользователя. Форма включает четыре обязательных поля: отображаемое имя (логин), электронная почта, пароль и подтверждение пароля. Реализована клиентская валидация: проверка формата email, минимальной длины пароля и совпадения введенных паролей. При успешной регистрации данные отправляются на сервер Ktor, где создается запись в таблице Users базы данных PostgreSQL.

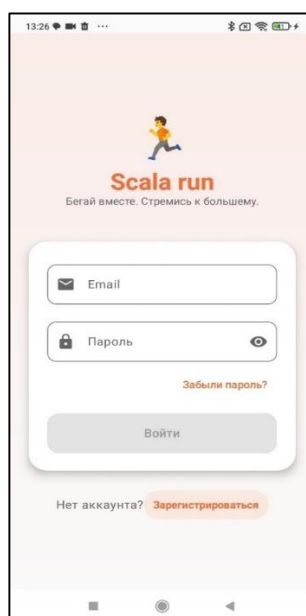


Рисунок 29– Экран авторизации приложения

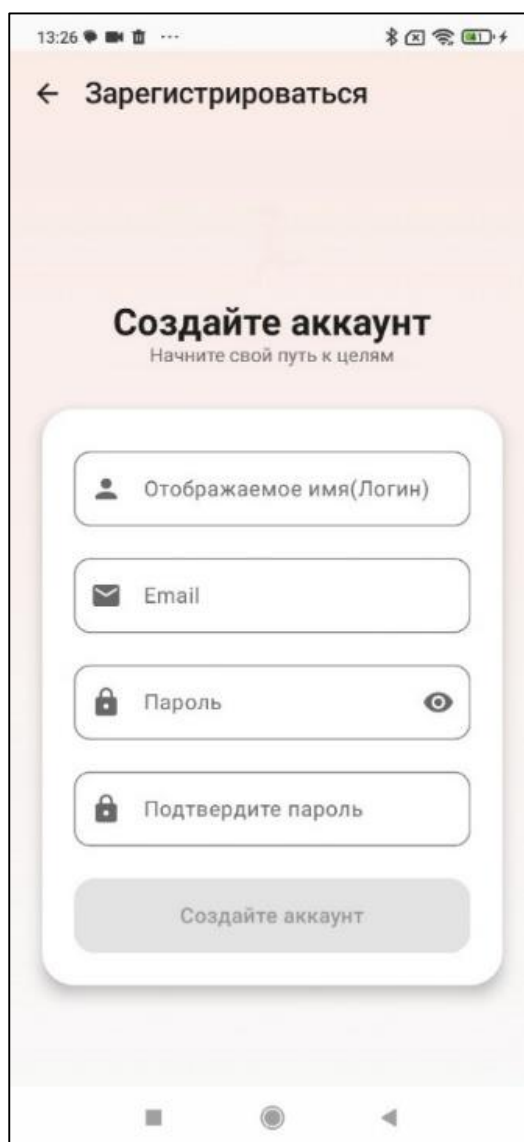


Рисунок 30– Экран регистрации нового пользователя

На рисунке 31 показан главный экран приложения после успешной авторизации под учетной записью «test2». Экран отображает сводную статистику пользователя (дистанция за неделю и месяц, количество пробежек), быстрый доступ к основным модулям (Статы, История, Вместе, Награды, Призрак, Друзья) и карточку последней завершенной пробежки с указанием дистанции (7.34 км) и средней скорости (3.8 км/ч). Оранжевая кнопка «Начать пробежку» инициирует запуск Foreground Service для GPS–трекинга.

					ПГУ 1.090301.39.001 ПЗ	Лист
						47
Изм.	Лист	№докум.	Подпись	Дата		

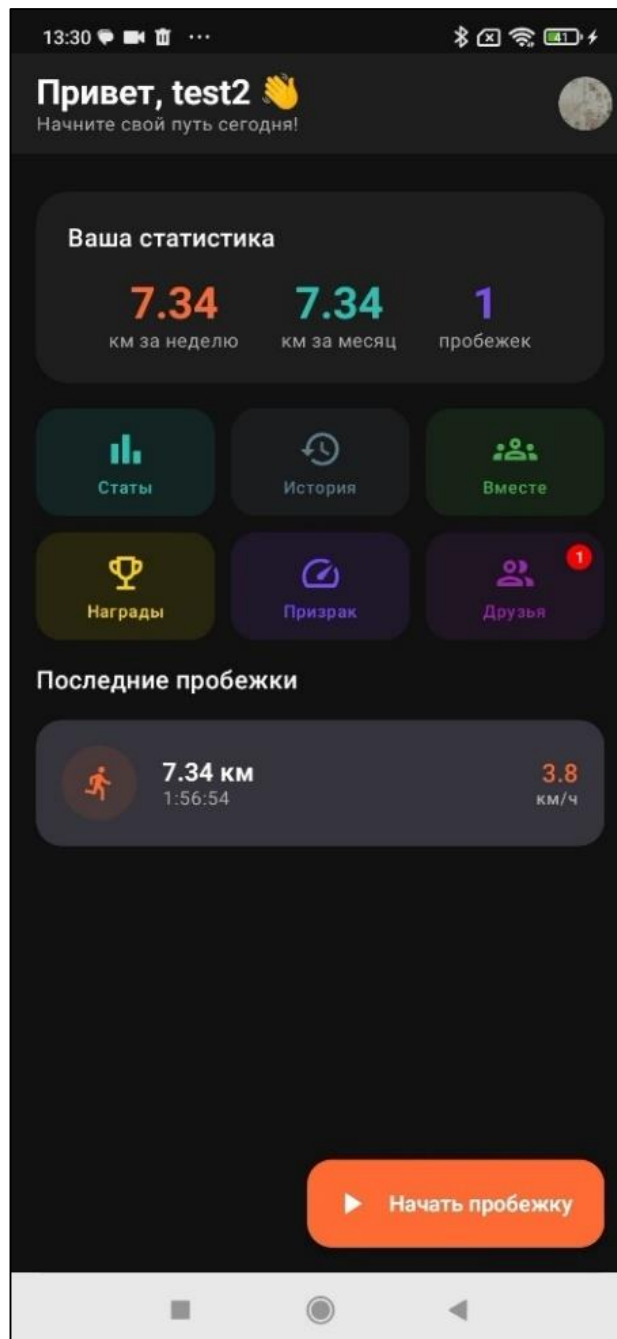


Рисунок 31– Главный экран приложения после авторизации

На рисунке 32 представлен экран профиля пользователя «antena123». Отображаются: аватар пользователя, уровень активности («Новичок»), общая дистанция (0.4 км), время в движении (0 часов) и количество завершенных забегов (18). Также доступны функции редактирования профиля (имя, фото, настройки) и кнопка выхода из аккаунта. Данные профиля синхронизируются с таблицей Users в облачной базе PostgreSQL.

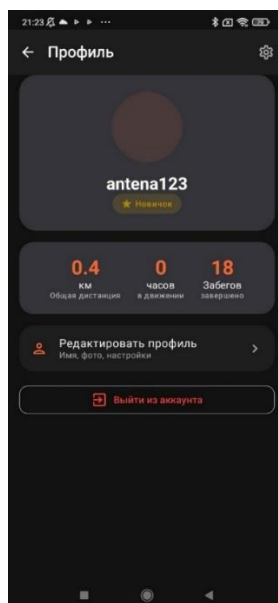
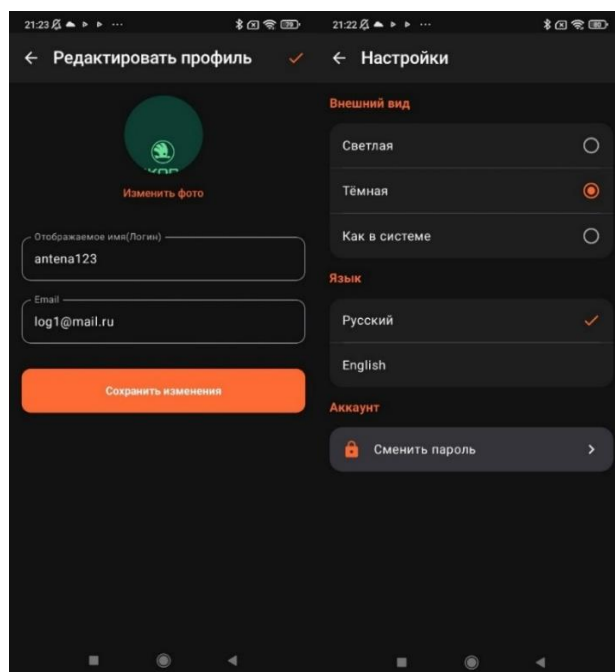


Рисунок 32– Экран профиля пользователя

На рисунках33 представлен экран редактирования профиля пользователя «antena123», можно сменить: аватар через изменить фото;отображаемое имя – логин;Email. Также при нажатии на шестерёнку можно изменить внешний вид приложения, выбрав светлую или тёмную тему; поменять язык интерфейса (English, Русский) и сменить пароль от аккаунта.



Рисунки33– Экран редактирования профиля и настройки приложения

На рисунке 34 представлен интерфейс после нажатия на кнопку «Забыли пароль?» для восстановления доступа через EmailService. Далее на рисунках 35 представлены получение кода для восстановления пароля отправленный на указанный Email. После ввода кода переходите на окно с новым паролем и подтверждением нового пароля, который представлен на рисунке 36.

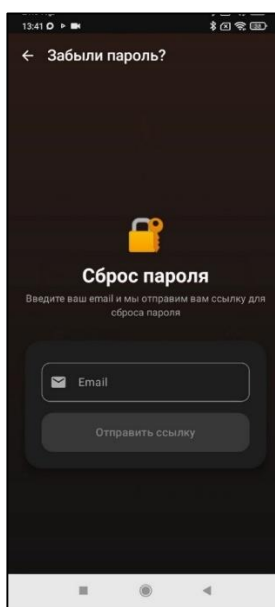
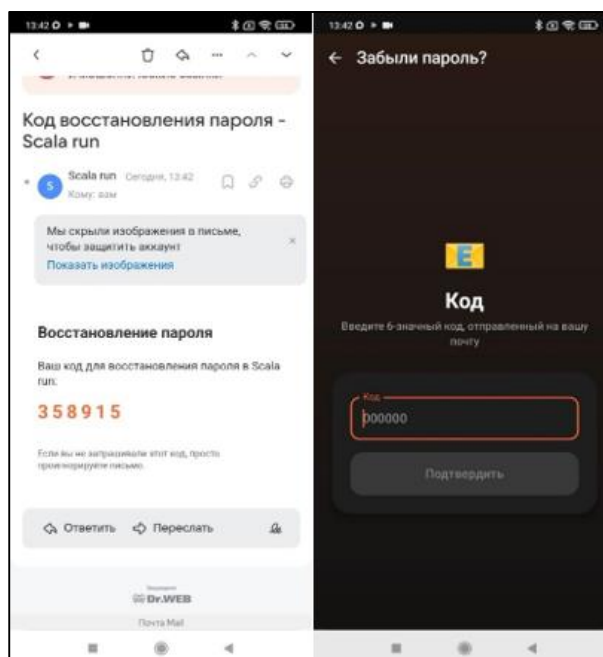


Рисунок 34 – Экран после нажатия на кнопку «Забыли пароль?»



Рисунки 35 – Получение кода на указанную почту и интерфейс ввода кода

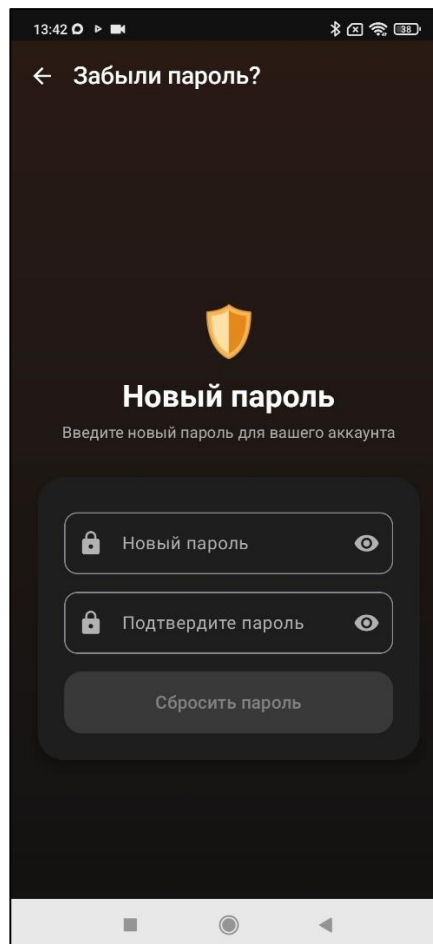


Рисунок 36 – Экран интерфейса создания нового пароля

1.4.2 Трекинг пробежки и работа с картами

Полевые испытания GPS–трекера проводились на различных дистанциях. Была осуществлена проверка стабильности записи маршрута при заблокированном экране с использованием Foreground Service. Для проверки логики расчета темпа и дистанции использовались юнит–тесты ActiveRunViewModel.

На рисунке 37 показан экран активной пробежки. В верхней части расположена информационная панель с текущей дистанцией (0.00 км), временем (00:00) и скоростью (0.0 км/ч). На карте отображается позиция пользователя «test2» в виде зеленого маркера. Чуть ниже слева расположен овал, в котором находится логин пользователя и его аватар. Перед нижней частью справа расположена кнопка центрирования для возвращения к местоположению (удобно

при отдалении карты и нахождения на ней друзей). В нижней части экрана расположены кнопки управления: отмена (красный крестик), пауза (оранжевая) и завершение (оранжевый квадрат справа). Трекинг осуществляется через FusedLocationProviderClient с интервалом обновления 1 секунда.

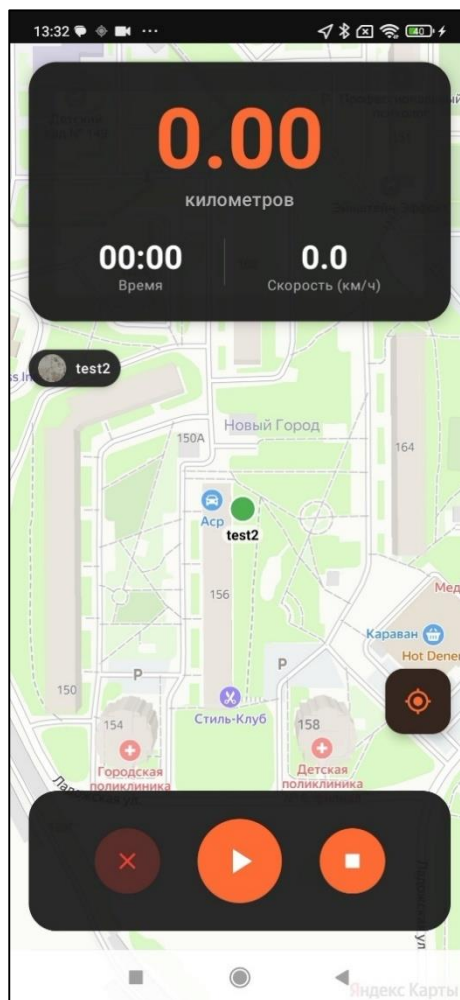
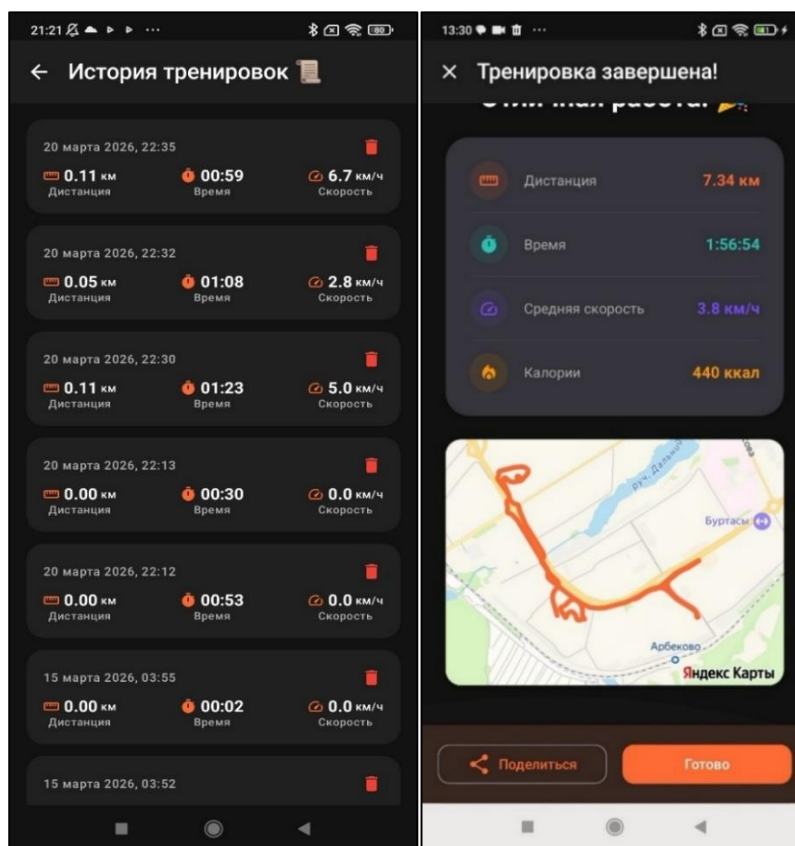


Рисунок 37– Экран активной пробежки с GPS–трекингом на карте

На рисунке 38 представлен интерфейс «История», то есть показаны все тренировки с краткой статистикой: дистанция в км, время, и средняя скорость. Также представлен экран завершения тренировки. Отображается итоговая сводка: дистанция (7.34 км), время (1:56:54), средняя скорость (3.8 км/ч) и расчетные калории (440 ккал). Ниже показана карта с нанесенным маршрутом забега. Пользователю доступны кнопки «Поделиться» и «Готово» для сохранения

результата. Данные сохраняются одновременно в локальную БД Room (таблица RunEntity) и синхронизируются с облачным сервером.



Рисунки 38 – Экран интерфейса история тренировок и открытая завершенная тренировка

На рисунке 39 представлена отправка завершенной тренировки по нажатию на кнопку «Поделиться».

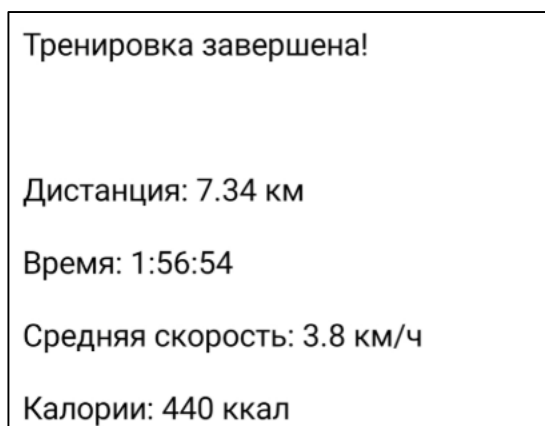


Рисунок 39 – Отправка завершенной тренировки

1.4.3 Совместная беговая тренировка

Было проведено комплексное тестирование модуля совместных забегов, включая создание встречи, отправку приглашений, обнаружение друзей в радиусе 100 метров и отслеживание участников на карте в реальном времени через WebSocket-соединение.

На рисунках40 показан экран активной совместной пробежки с уведомлением «Друг рядом!» и кнопкой «Пригласить». На карте одновременно отображаются два участника (на карте отображаются только те пользователи, которые добавлены в друзья): «test2» и «fffffffffffffffffffff» с зелеными маркерами. Верхняя панель содержит чипы с именами всех участников забега.

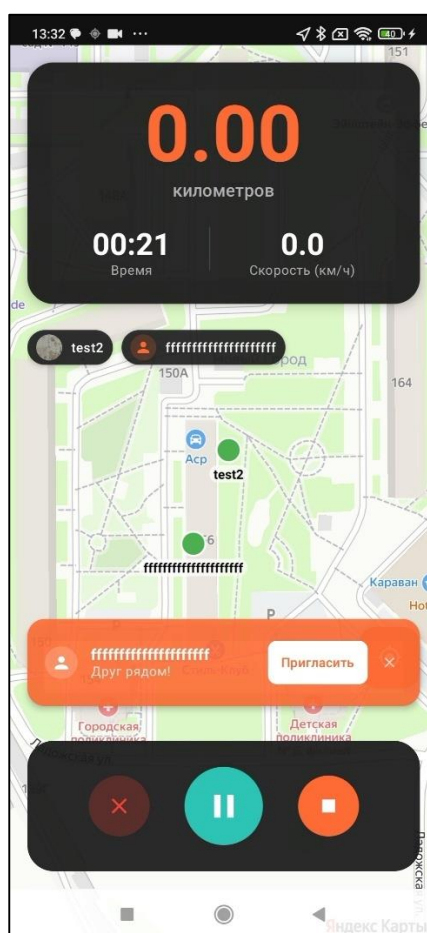
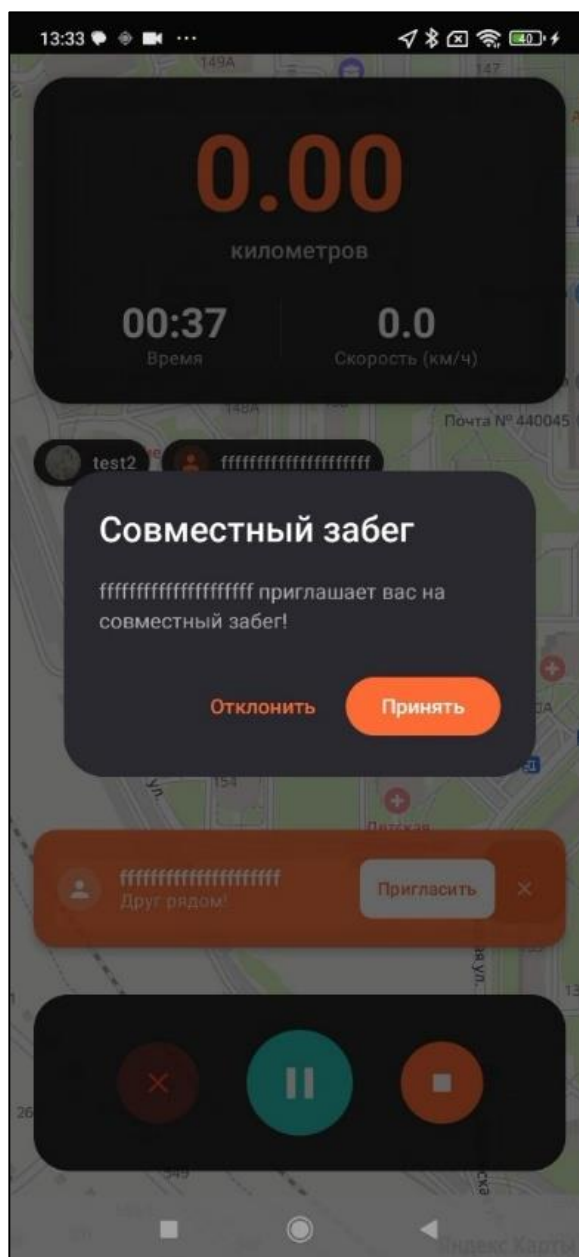


Рисунок 40 –Уведомление о близости друга во время пробежки

На рисунке 41 показано всплывающее диалоговое окно приглашения на совместный забег. Пользователь «fffffffffffffffffffff» приглашает на совместный

забег; доступны кнопки «Отклонить» и «Принять». Данный диалог реализован через WebSocket–канал, обеспечивающий мгновенную доставку уведомлений.



Рисунки41 – Диалог приглашения на совместный забег

На рисунке 42 представлен экран совместной пробежки после принятия приглашения. На карте отображаются оба участника («test2» и «ffffffffffffffffffff») с индикацией пройденной дистанции (0.0 км) рядом с каждым маркером. Верхняя панель содержит чипы обоих участников с указанием их текущей дистанции.

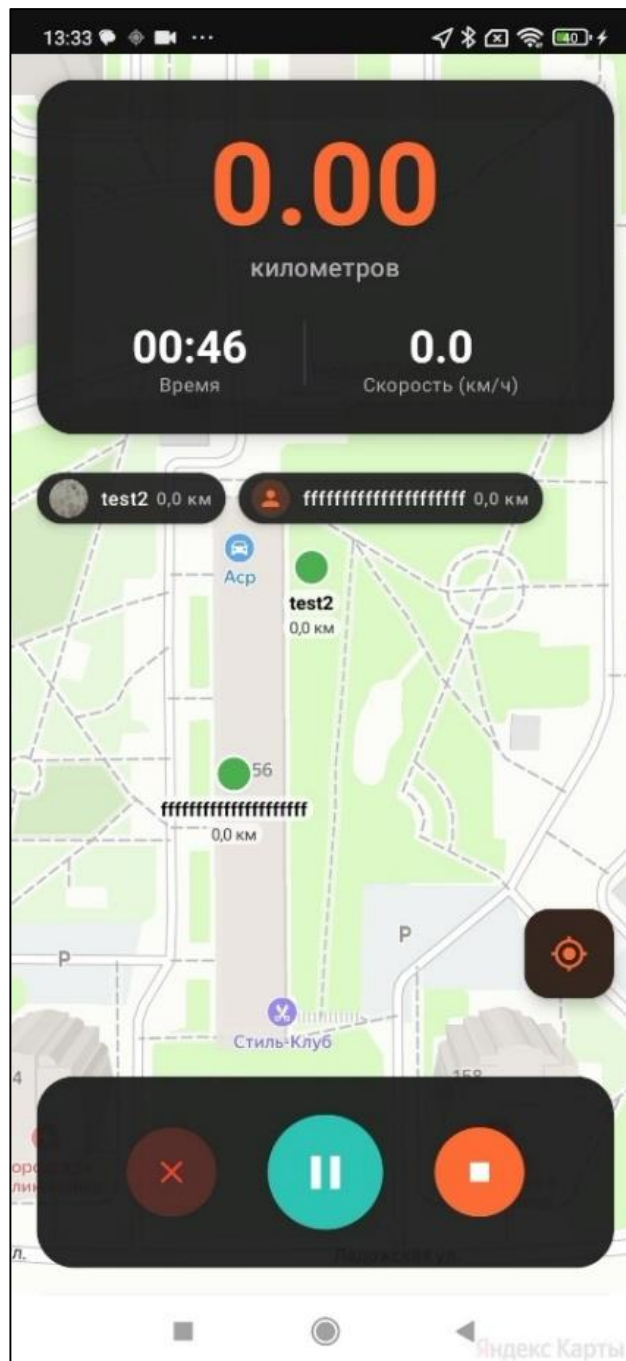


Рисунок 42– Совместный забег с двумя участниками на карте

На рисунке 43 показана вкладка «Завершённые» экрана совместного бега. В списке отображается завершённая совместная пробежка с пользователем «ffffffffffffffffffff» (18 марта, 13:33). Завершённые встречи сохраняются в истории для последующего просмотра результатов.

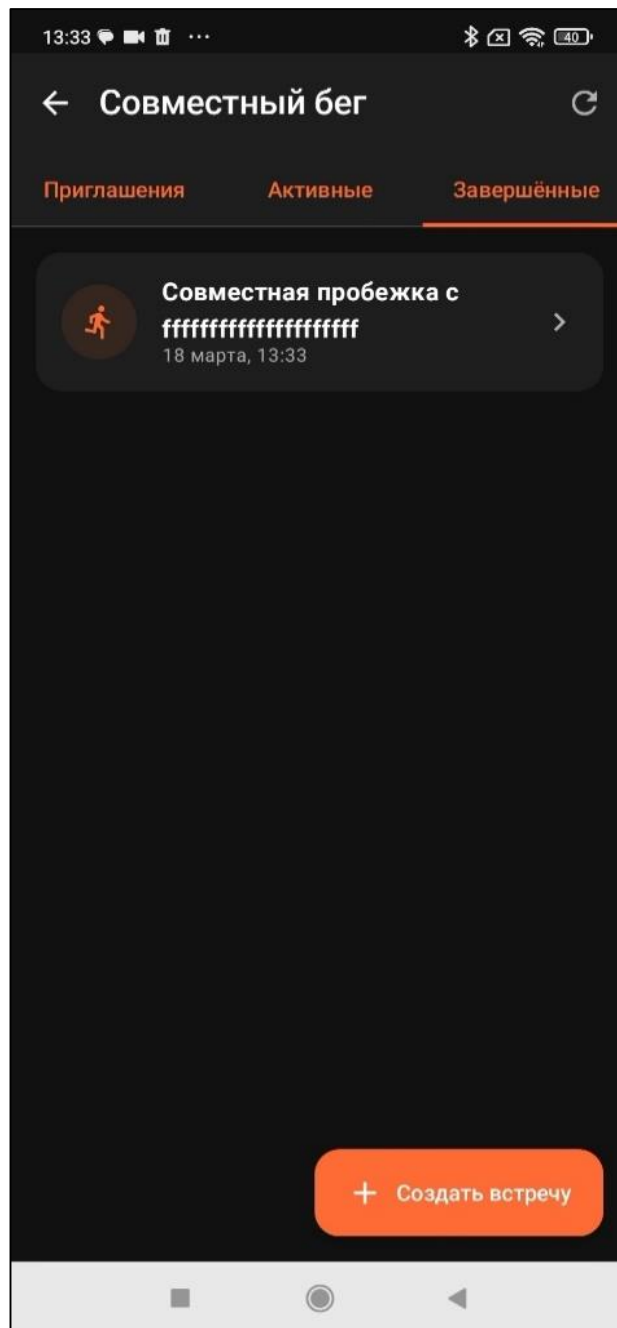
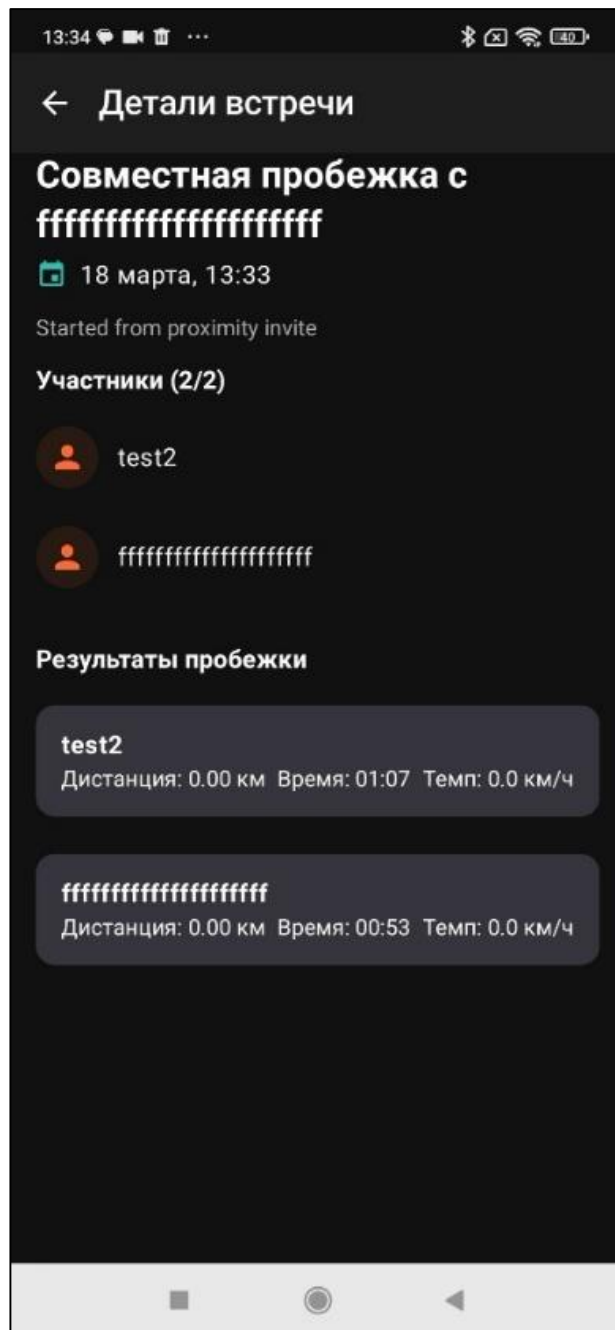


Рисунок 43 –Список завершённых совместных тренировок с другом

На рисунке 44 представлен экран деталей завершённой встречи. Отображается название совместной пробежки, дата проведения, список участников (2/2) и итоговые результаты каждого участника: «test2» (дистанция 0.00 км, время 01:07) и «ffffffffffffffffffff» (дистанция 0.00 км, время 00:53). Данные результатов сохраняются в таблице Participants серверной базы данных.



Рисунки44– Детали завершённой совместной пробежки с результатами участников

На рисунке 45 показан экран «Совместный бег» на вкладке «Приглашения». Интерфейс информирует пользователя об отсутствии входящих приглашений и предоставляет кнопку «Обновить» для повторного запроса к серверу. В нижней части расположена кнопка «Создать встречу» для инициации нового совместного забега.

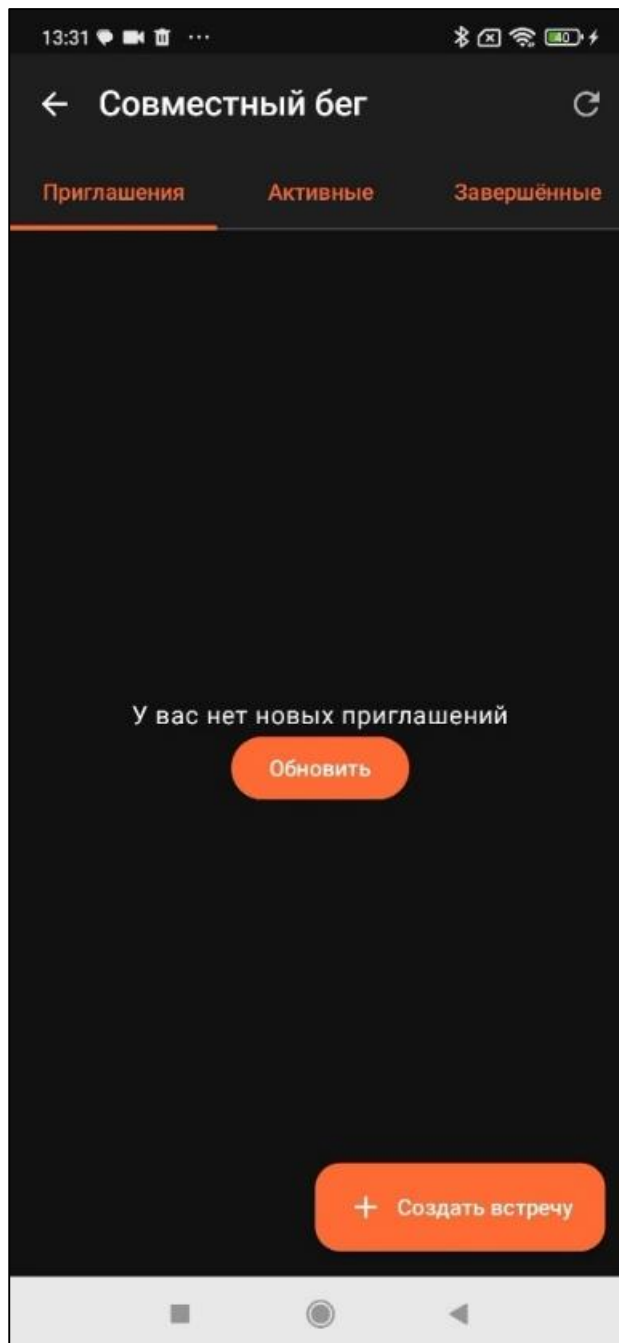


Рисунок 45 –Экран совместного бега: вкладка «Приглашения»

На рисунке 46 представлена форма создания новой встречи. Пользователь заполняет название и описание мероприятия, задает параметры: количество участников (10), планируемую дистанцию (5.0 км), дату и время старта (19.03.2026 13:31). Также доступна функция «Пригласить друзей» для выбора конкретных пользователей из списка друзей. После нажатия кнопки «Создать

встречу» данные отправляются на сервер и сохраняются в таблице Meetings базы данных PostgreSQL.

13:31

← Новая встреча

Основная информация

Т Название

Описание

Детали забега

Участники: 10

Дистанция (км) 5.0

Дата и время 19.03.2026 13:31

Пригласить друзей

Никто не выбран

Выбрать

Рисунки46– Форма создания новой встречи для совместного бега

На рисунке 47 показана вкладка «Активные» с созданной встречей «hello» (18 марта, 13:31). Встреча отображается в списке активных мероприятий и доступна для присоединения другим участникам.

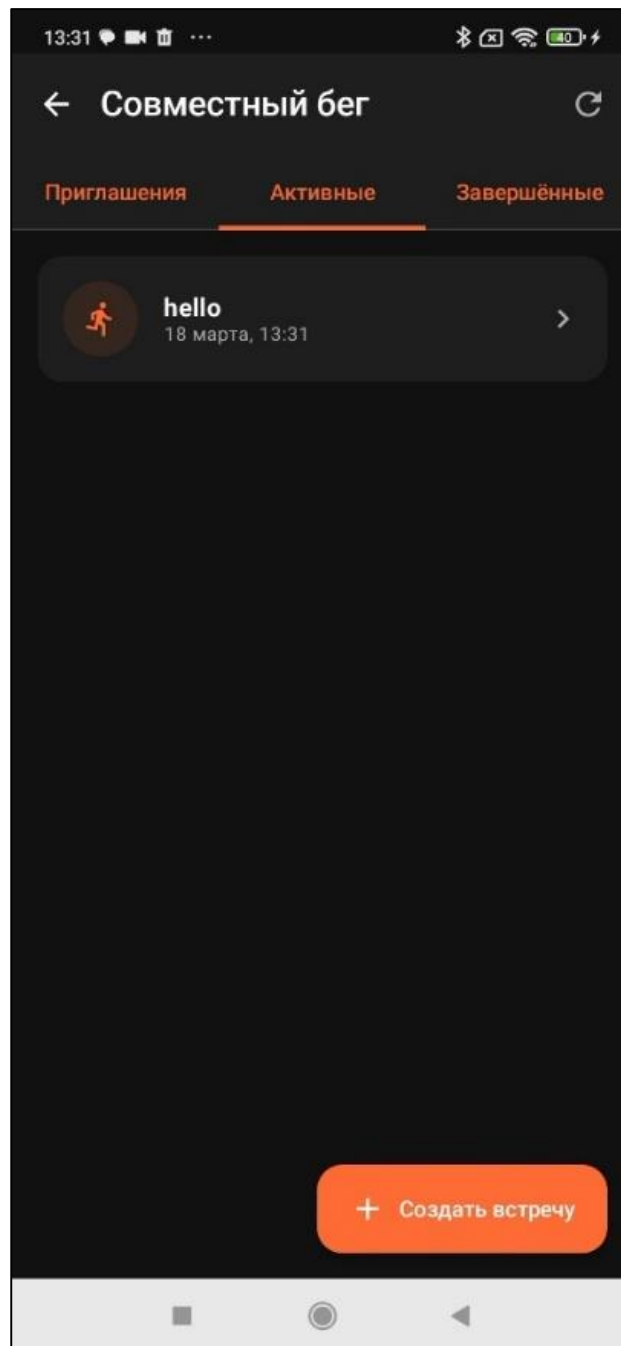
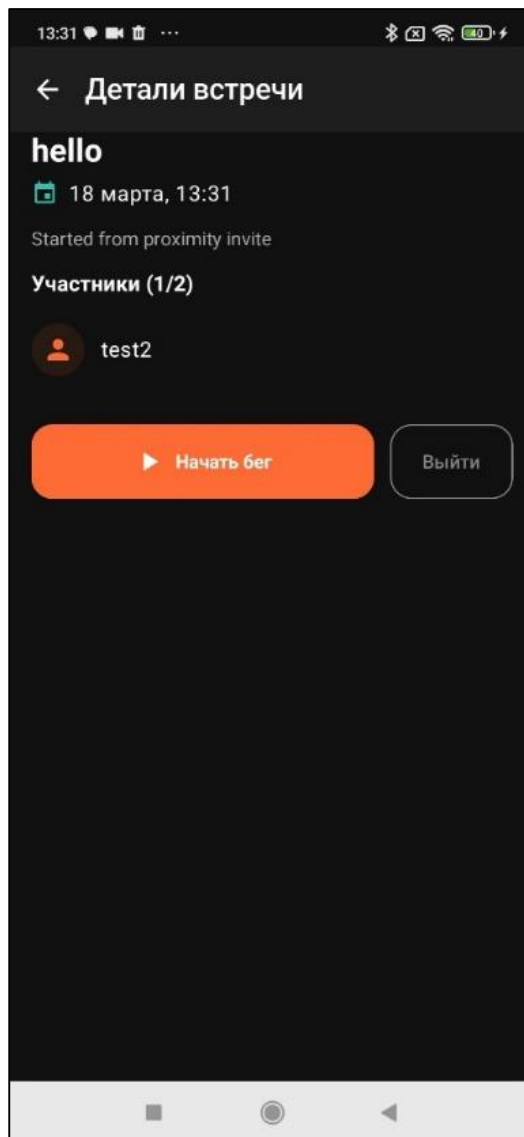


Рисунок 47 –Список активных встреч для совместного бега

На рисунке 48 представлен экран деталей встречи «hello». Отображается дата создания, пометка «Startedfromproximityinvite» (встреча инициирована при обнаружении друга в радиусе 100 метров), список участников (1/2) и кнопки «Начать бег» / «Выйти». Данный экран подтверждает корректную работу механизма приглашений по близости.



Рисунки 48 –Детали встречи с возможностью начать совместный бег

1.4.4 Соревнование в режиме «Призрак»

Произведена верификация корректности отображения «виртуального соперника» на карте во время бега и синхронизации его данных с сервером.

На рисунке 49 показан экран выбора «призрака» для соревнования. Отображается список предыдущих пробежек пользователя с указанием даты, дистанции, времени и средней скорости. Каждая запись может быть удалена (иконка корзины) или выбрана в качестве эталонного забега для режима Virtual Ghost. Данные загружаются из таблицы GhostTrackEntity локальной базы Room.

					ПГУ 1.090301.39.001 ПЗ	Лист
						62
Изм.	Лист	№докум.	Подпись	Дата		

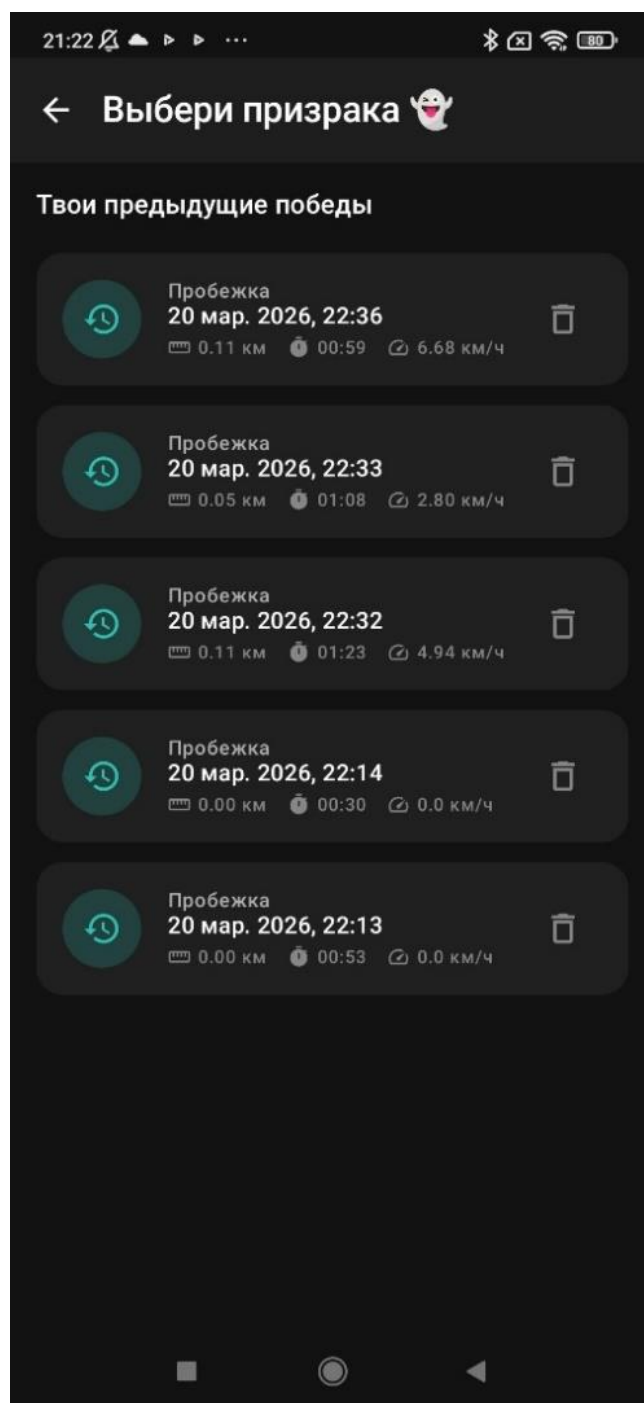
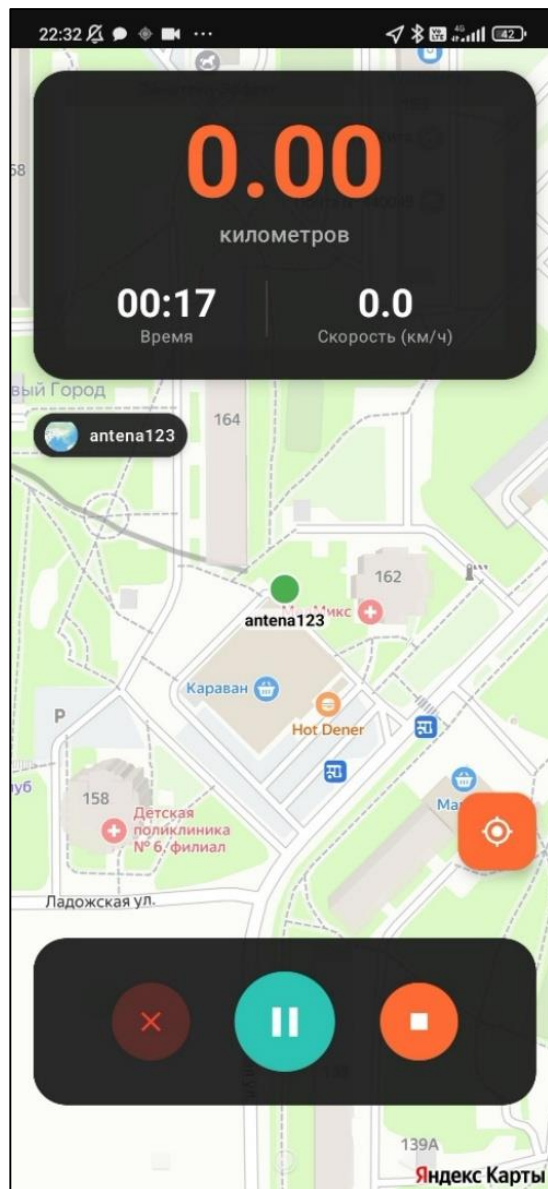


Рисунок 49 –Экран выбора эталонной пробежки для режима «Призрак»

На рисунке 50 представлен экран активной пробежки в режиме «Призрак». На карте отображается позиция пользователя «antena123» с текущей дистанцией (0.00 км), временем (00:17) и скоростью (0.0 км/ч). Позиция виртуального соперника отрисовывается параллельно на карте, позволяя атлету визуально оценивать опережение или отставание от графика.



Рисунки 50 – Активная пробежка в режиме призрак

1.4.5 Управление списком друзей

Осуществлена проверка отправки запросов в друзья, обновления статусов заявок и поиска пользователей по имени или email.

На рисунке 51 представлен список друзей пользователя (вкладка «Друзья»). Отображаются принятые контакты: «Ant» (antena1995@mail.ru) и «ffffffffffffffffffff» (fan@mail.ru). Для каждого друга доступны кнопки: иконка бегуна (пригласить на совместный забег) и иконка удаления (удалить из друзей). На вкладке «Запросы» виден индикатор с числом входящих заявок(1).

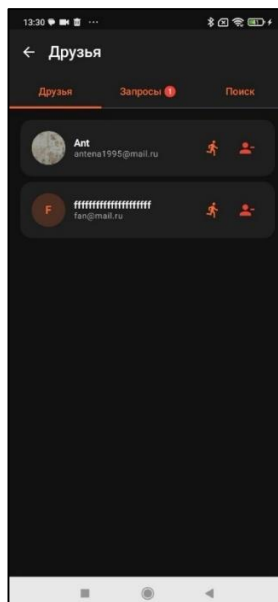
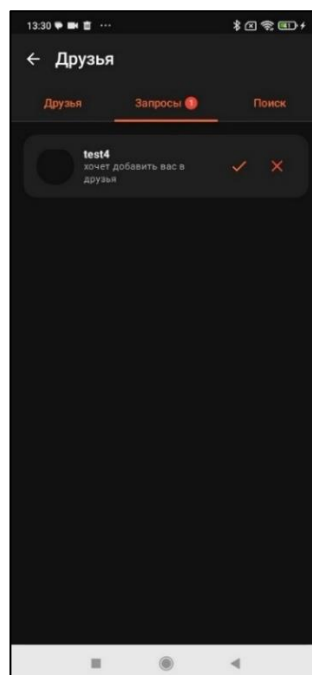


Рисунок 51 –Список друзей с возможностью приглашения на забег

На рисунке 52 показана вкладка «Запросы» с входящей заявкой в друзья от пользователя «test4». Отображается текст «хочет добавить вас в друзья» с кнопками принятия (галочка) и отклонения (крестик). Статус заявки обновляется в таблице Friendships серверной базы данных (поле status меняется с PENDING на ACCEPTED).



Рисунки 52 – Входящий запрос в друзья с кнопками принятия и отклонения

На рисунке 53 показан экран поиска пользователей (вкладка «Search»). Пользователь вводит запрос «test» в поисковую строку, и система отображает найденных пользователей: «test2» (test@mail.ru) и «test3» (test1@mail.ru). Рядом с каждым результатом расположена кнопка «Add» (на английском интерфейсе, на русском «Добавить») для отправки запроса в друзья. Поиск выполняется через REST API серверной части.

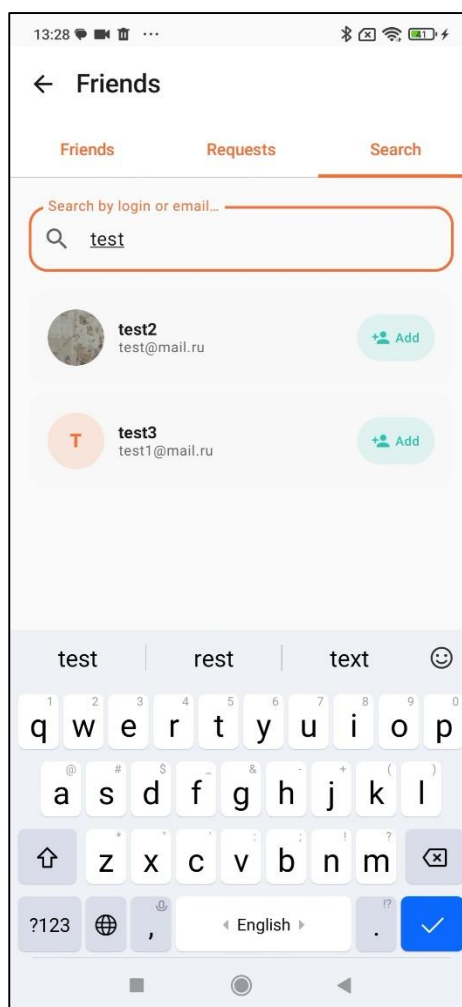
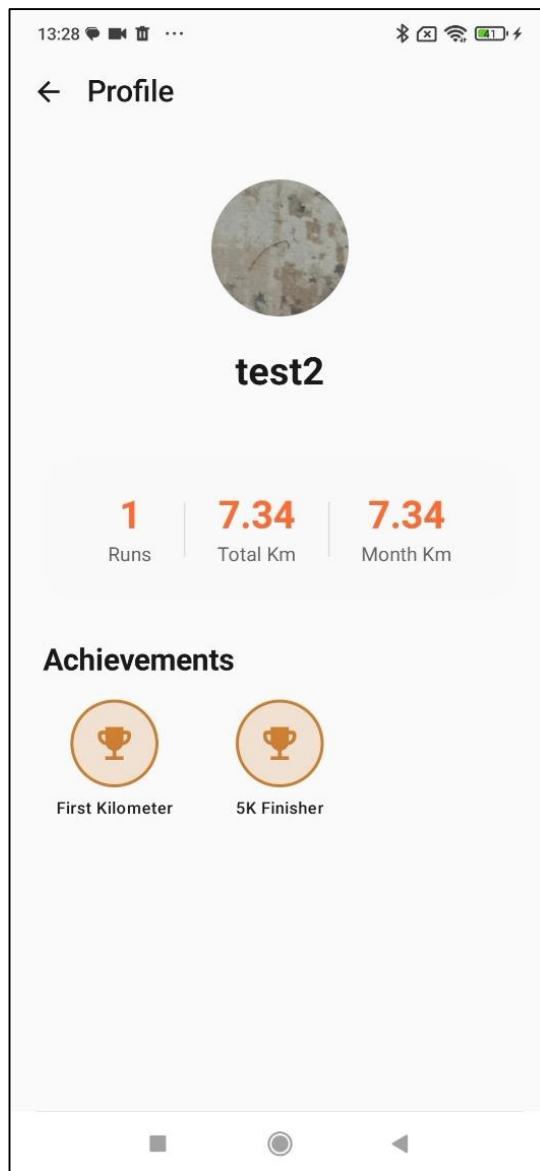


Рисунок 53 –Поиск пользователей для добавления в друзья

На рисунке 54 представлен экран профиля другого пользователя «test2». Отображаются: аватар, общая статистика (1 забег, 7.34 км суммарно, 7.34 км за месяц) и полученные достижения: «First Kilometer» и «5K Finisher». Данный экран подтверждает корректную работу модуля достижений и синхронизацию данных профиля между клиентом и сервером.



Рисунки 54 – Профиль пользователя с достижениями и статистикой

1.4.6 Система достижений и статистика

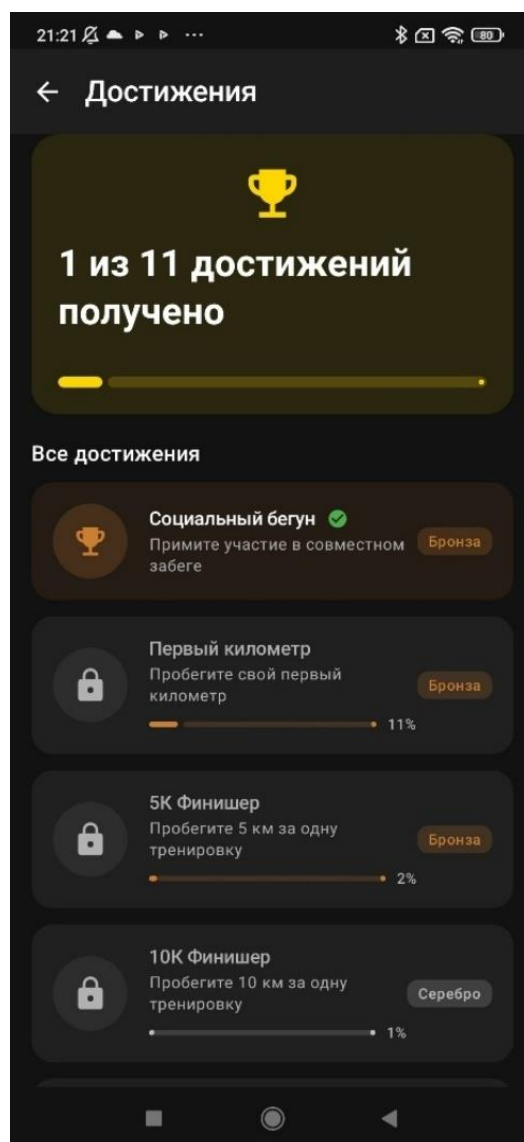
Протестирован механизм выдачи уникальных достижений, а также корректность отображения агрегированной статистики пользователя.

На рисунке 55 показан экран «Моя Статистика» с детальной аналитикой. Отображаются: общая дистанция (0.38 км) с прогресс-баром, дистанция за неделю и месяц (0.00 км), общее количество забегов (18), суммарное время (11 мин), а также баннер «Твои Достижения» с призывом заглянуть в раздел наград. Статистические данные агрегируются из локальной базы Room и синхронизируются с сервером.



Рисунок 55 –Детальная статистика пользователя

На рисунке 56 представлен экран «Достижения» с полным каталогом наград. В верхней части отображается общий прогресс: «1 из 11 достижений получено» с прогресс–баром. Ниже расположен список всех достижений с описанием условий получения и индикаторами прогресса: «Социальный бегун» (принять участие в совместном забеге – получено, бронза), «Первый километр» (пробежать первый километр – 11%), «5К Финишер» (пробежать 5 км за одну тренировку – 2%), «10К Финишер» (пробежать 10 км за одну тренировку – серебро, 1%). Заблокированные достижения отмечены иконкой замка.



Рисунки 56 – Каталог достижений с прогрессом выполнения

2 Безопасность жизнедеятельности

В процессе разработки программного обеспечения программист проводит значительное количество времени за персональным компьютером. Длительная работа со средствами вычислительной техники сопровождается воздействием ряда неблагоприятных факторов: повышенным зрительным напряжением, статической нагрузкой на опорно-двигательный аппарат и воздействием электромагнитных полей. В данном разделе рассматриваются требования к организации условий труда, обеспечивающих сохранение здоровья и высокую работоспособность.

2.1 Требования к организации рабочего места

Организация рабочего места программиста должна основываться на принципах эргономики для минимизации утомляемости. Основные требования включают:

- рабочий стол: высота рабочей поверхности стола должна составлять 725 мм. Поверхность стола должна быть матовой для исключения световых бликов. Ширина стола должна обеспечивать возможность размещения монитора, клавиатуры и документов без стеснения движений;

- рабочее кресло: для сидения рекомендуется применять кресло с поворотным и подъёмным механизмом, позволяющее настраивать высоту сиденья, а также углы наклона спинки и самого сиденья. Спинка должна обладать анатомической формой, обеспечивающей опору для поясничного изгиба позвоночника;

- размещение видеомонитора: экран монитора должен находиться на расстоянии 600-700 мм от глаз пользователя, но не ближе 500 мм. Направление взгляда должно быть перпендикулярно центру экрана или под небольшим углом вниз (10-20°);

– пространственные нормы: на одно рабочее место с ПК должно выделяться не менее 4,5 м² площади и 15 м³ объема помещения для обеспечения нормального воздухообмена.

2.2 Требования к освещению

Освещение в помещении для работы с вычислительной техникой должно быть смешанным (естественным и искусственным):

– естественное освещение: рабочие места должны располагаться так, чтобы естественный свет падал сбоку, желательно с левой стороны. Для защиты от прямых солнечных лучей и исключения бликов на экранах окна должны быть оборудованы регулируемыми жалюзи или плотными шторами.

– искусственное освещение должно осуществляться системой комбинированного освещения (общее + местное). Уровень освещенности на поверхности стола должен составлять от 300 до 500 люкс. В качестве источников света рекомендуется использовать светодиодные лампы с нейтральным белым спектром.

2.3 Требования к режиму отдыха

Для поддержания высокого уровня работоспособности и предотвращения профессиональных заболеваний необходимо соблюдение режима труда и отдыха:

– регламентированные перерывы: при интенсивной работе за компьютером необходимо делать перерывы по 10-15 минут через каждые 60 минут работы. Во время перерывов рекомендуется покидать рабочее место и выполнять производственную гимнастику (упражнения для снятия напряжения с мышц шеи, спины и рук).

– зрительная гимнастика: каждые 20-30 минут работы следует переводить взгляд на удаленные объекты (на расстояние более 5 метров) в течение 20 секунд для расслабления глазных мышц.

– микроклимат: в помещении следует поддерживать оптимальную температуру воздуха 22-24 °С и относительную влажность 40-60%.
Рекомендуется регулярное проветривание в течение всего рабочего дня.

– электробезопасность: все устройства должны быть заземлены, а кабели питания защищены от механических повреждений и не должны пересекать проходы.

					ПГУ 1.090301.39.001 ПЗ	Лист
						72
Изм.	Лист	№докум.	Подпись	Дата		

Заключение

В ходе выполнения данной выпускной квалификационной работы было разработано Android-приложение для совместных беговых тренировок. В ходе разработки были получены и закреплены практические навыки создания Android-приложения в современной интеграционной среде Android Studio с использованием языка программирования Kotlin, фреймворка Jetpack Compose, а также баз данных Room и PostgreSQL и использование контейнеров Docker для развертывания серверной части приложения.

Результат работы – это приложение, которое соблюдает современные архитектурные стандарты разработки (паттерн MVVM и Clean Architecture), позволяет пользователю записывать и анализировать свои пробежки, просматривать статистику, управлять списком друзей и интегрировано с картографическим сервисом Yandex MapKit. Главной отличительной чертой приложения является совместная беговая тренировка и режим «Призрак», позволяющий соревноваться с виртуальным соперником на основе прошлых результатов.

В дальнейшем возможно улучшение приложения, которое может включать в себя добавление функционала командных соревнований, интеграцию с носимыми фитнес-браслетами и умными часами, а также развитие игровой механики для повышения мотивации и комфортного использования приложения целевой аудиторией.

Список использованных источников

1. Gartner.com: «Отчёт о распространении на рынке мобильных устройств и их мировых продажах от компании Gartner». – [Электронный ресурс] / Режим доступа: <https://www.gartner.com/en/newsroom/press-releases/2020-03-03-gartner-says-global-smartphone-sales-fell-slightly-in/> (дата обращения 20.04.2026).

2. Metanit.com: «Руководство по языку Kotlin». – [Электронный ресурс] / Режим доступа: <https://metanit.com/kotlin/tutorial/>, свободный. (дата обращения: 17.04.2026).

3. Habr.com: «Библиотека “Room” для начинающего Android-разработчика». / [Электронный ресурс] – Режим доступа: <https://habr.com/ru/articles/713518/> (дата обращения 22.04.2026).

4. Б. Новиков, Е. Горшкова, Н. Графеева– Основы технологий баз данных: учебное пособие (дата обращения 01.05.2026).

5. И.В. Ананченко, Т.В. Зудилова, С.Е. Иванов –Контейнеризатор приложений docker– установка, настройка, основы управления приложениями в средах с поддержкой контейнеризации: Учебно–методическое пособие. – [Электронный ресурс] / Режим доступа: https://books.ifmo.ru/book/2652/konteynerizator_prilozheniy_docker_%E2%80%94ustanovka,_nastroyka,_osnovy_upravleniya_prilozheniyami_v_sredah_s_podderzhkoy_konteynerizacii:_uchebno-metodicheskoe_posobie..htm (дата обращения: 03.05.2026).

6. GitVerse.ru: «Что такое MVVM: шаблон, как работает, преимущества и примеры». – [Электронный ресурс] / Режим доступа: <https://gitverse.ru/blog/articles/development/116-что-такое-mvvm-shablon-kak-rabotaet-preimushhestva-i-primery-ispolzovaniya> (дата обращения: 15.04.2026).

7. Habr.com: «Коротко о главном: CleanArchitecture, RobertC. Martin». – [Электронный ресурс] / Режим доступа: <https://habr.com/ru/articles/464185/> (дата обращения: 03.05.2026).

8. Habr.com: «Руководство по работе с фреймворком KotlinExposed». – [Электронный ресурс] / Режим доступа: <https://habr.com/ru/companies/otus/articles/555134/> (дата обращения: 04.05.2026).

					ПГУ 1.090301.39.001 ПЗ	Лист
						75
Изм.	Лист	№докум.	Подпись	Дата		

Приложение А

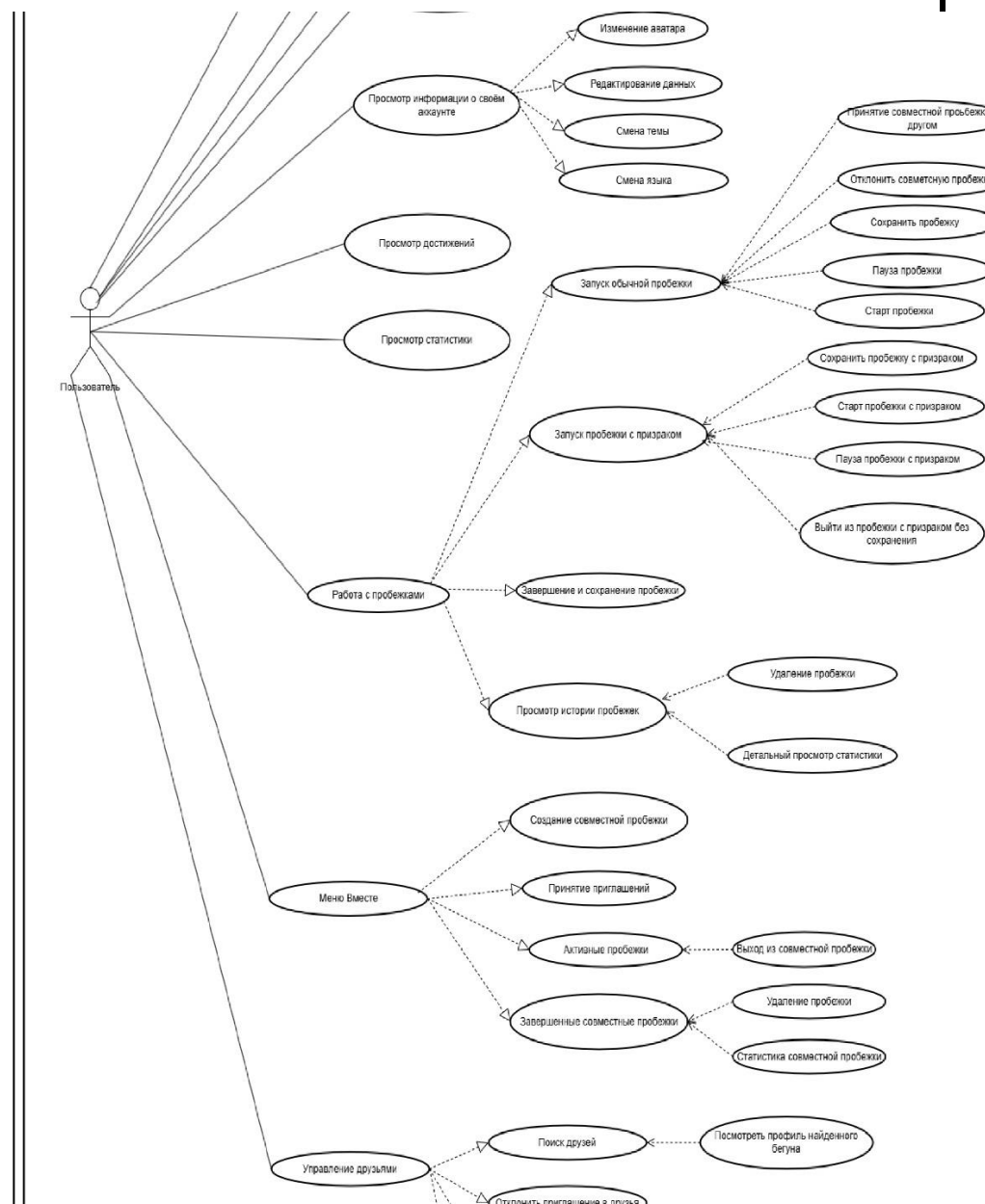
(обязательное)

Android–приложение для совместных беговых тренировок

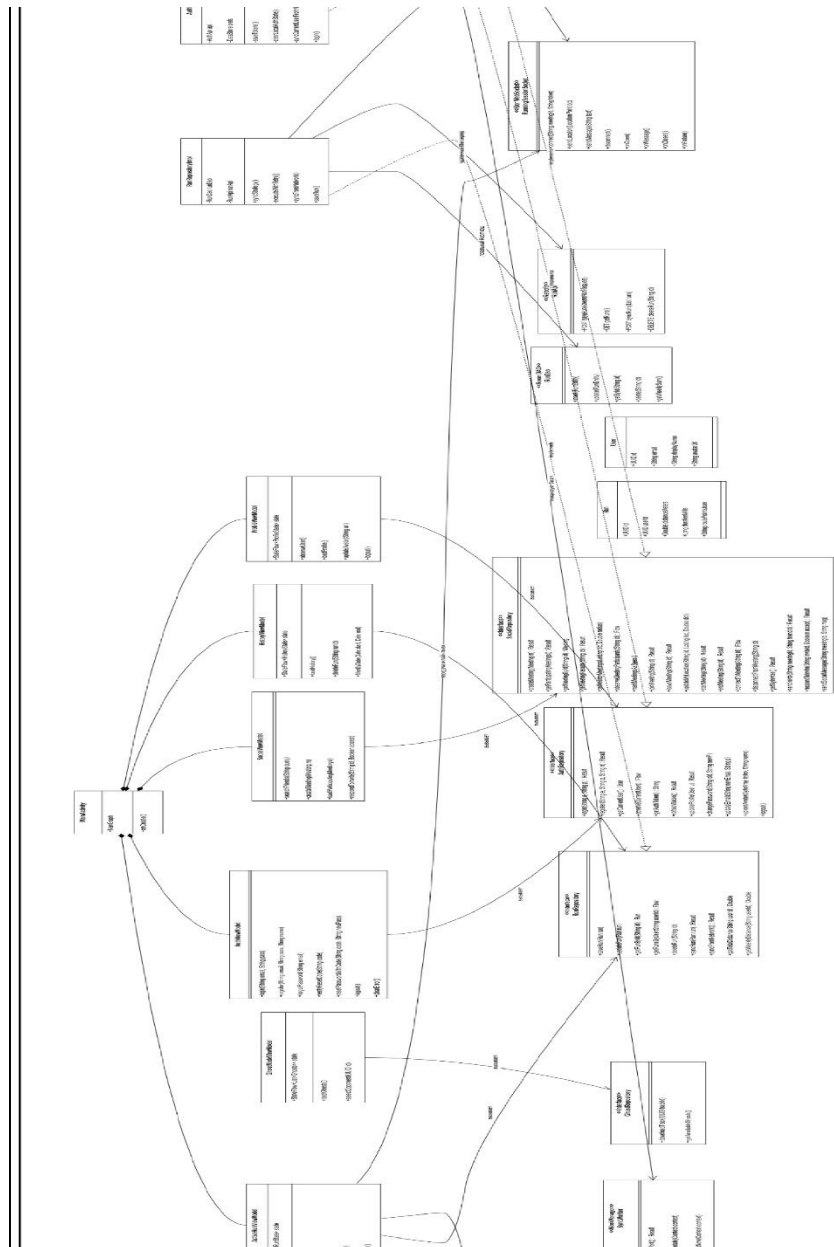
UML-диаграммы

					ПГУ 1.090301.39.001 ПЗ	Лист
						76
Изм.	Лист	№докум.	Подпись	Дата		

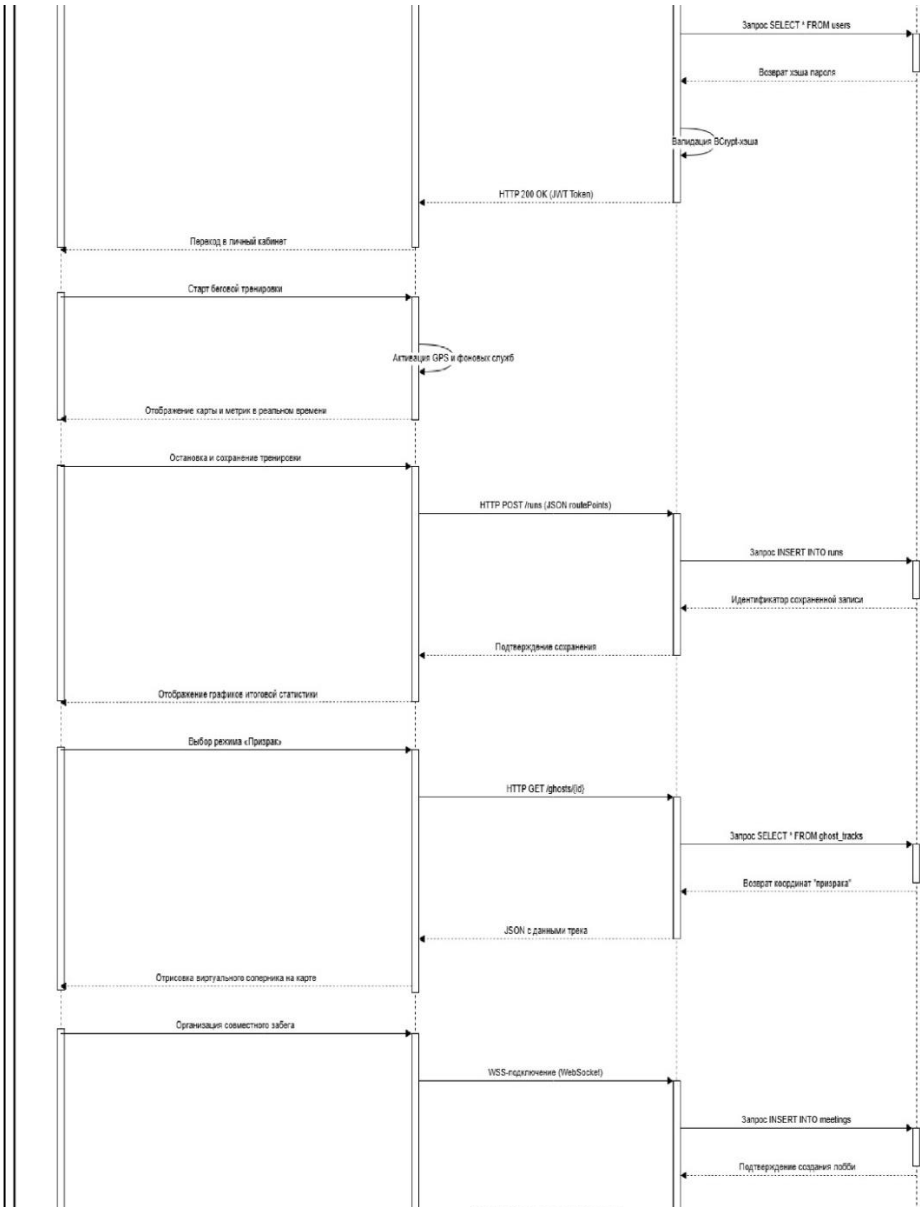
Приложение А.1 –UML-диаграмма прецедентов



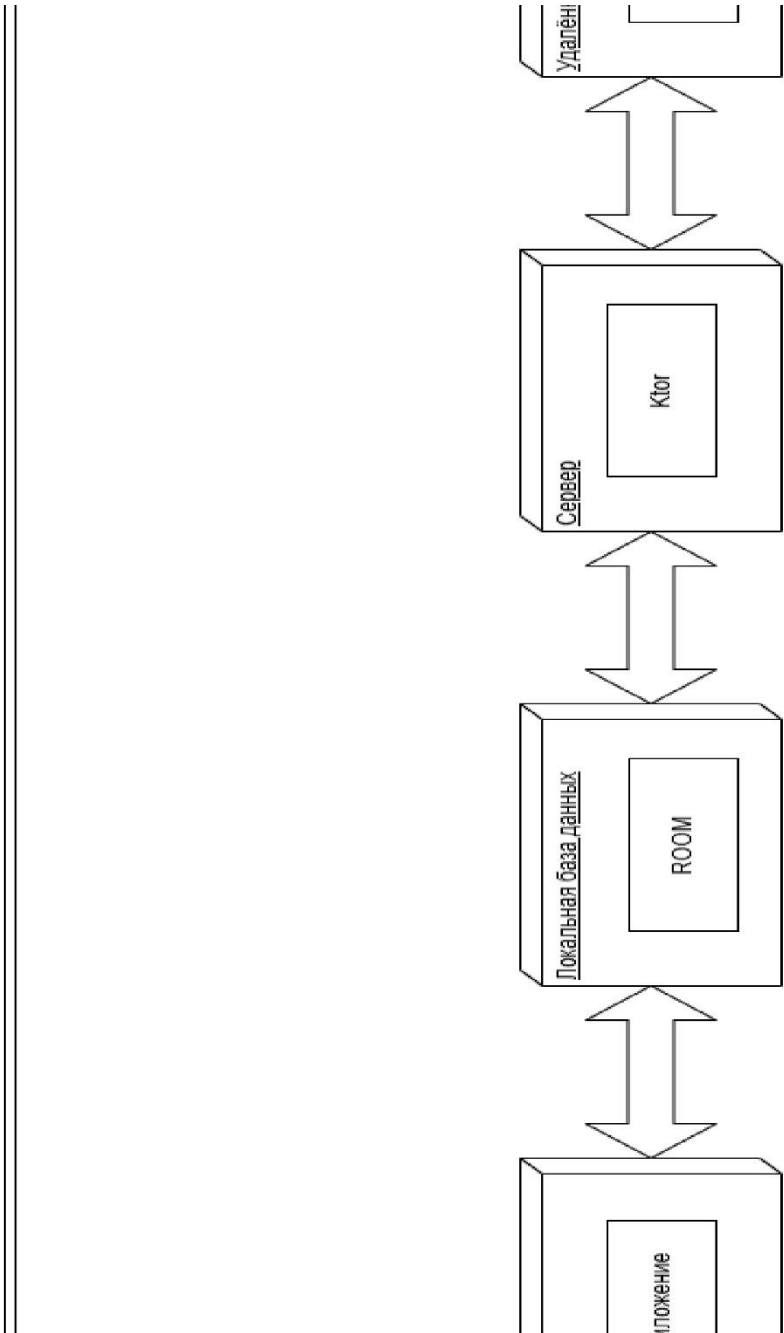
Приложение А.2 –UML-диаграмма всех классов



Приложение А.3 –UML-диаграмма последовательностей



Приложение А.4 –UML-диаграмма развертывания приложения



Приложение Б

(обязательное)

Android–приложение для совместных беговых тренировок

Листинг программы

					ПГУ 1.090301.39.001 ПЗ	Лист
						81
Изм.	Лист	№докум.	Подпись	Дата		

Клиентская часть

Файл FriendsScreen.kt

```
package com.runsocial.app.presentation.screens.social

@OptIn(ExperimentalMaterial3Api::class)
@Composable
    Text(title)
        if (index == 1
&&screenState.incomingRequests.isNotEmpty()) {
    Spacer(modifier = Modifier.width(4.dp))
        Badge {
Text("${screenState.incomingRequests.size}")
        }
    }
    }
    }
    )
    }
}

when (screenState.selectedTab) {
    0 ->FriendListSection(
        friends = screenState.friends,
isLoading = screenState.isLoading,
onRemoveFriend = { socialViewModel.removeFriend(it) },
onInviteToRun = onRunInvite,
onFriendClick = onProfileOpen
    )
    1 ->RequestListSection(
        requests = screenState.incomingRequests,
onAccept = { socialViewModel.acceptRequest(it) },
onDecline = { socialViewModel.declineRequest(it) },
onRequestClick = onProfileOpen
    )
}
```

					ПГУ 1.090301.39.001 ПЗ	Лист
						82
Изм.	Лист	№докум.	Подпись	Дата		


```

onClick = { onRequestClick(request.fromUserId) }

        )

    }

}

@Composable
private fun RequestItemRow(
    request: FriendRequestDto,
    onAccept: () -> Unit,
    onDecline: () -> Unit,
    onClick: () -> Unit
) {
    @Composable
    private fun UserSearchSection(
        searchQuery: String,
        searchResults: List<UserSearchDto>,
        isLoading: Boolean,
        onQueryChange: (String) -> Unit,
        onAddFriend: (String) -> Unit,
        onUserClick: (String) -> Unit
    ) {
        OutlinedTextField(
            value = searchQuery,
            onValueChange = onQueryChange,
            searchResults.isEmpty() &&searchQuery.length>= 2 -> {
                { user ->
                    FoundUserRow(
                        user = user,
                        onAddFriend = { onAddFriend(user.id) },
                        onClick = { onUserClick(user.id) }
                    )
                }
            }
        )
    }
}

```

```

        }
    }
}

}

}    maxLines = 1,
        overflow = TextOverflow.Ellipsis
    )
}
when (user.friendshipStatus) {
    "ACCEPTED" -> {
AssistChip(
onClick = {},

        label = {
Text(stringResource(R.string.already_friends), style =
MaterialTheme.typography.labelSmall) }
        )
    }
    "PENDING" -> {
AssistChip(
onClick = {},

        label = {
Text(stringResource(R.string.pending), style =
MaterialTheme.typography.labelSmall) }
        )
    }
    else -> {
FilledTonalButton(
onClick = onAddFriend,
contentPadding = PaddingValues(horizontal = 12.dp, vertical = 4.dp)
        )
    }
}
}
}

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						85
Изм.	Лист	№докум.	Подпись	Дата		

```

    }

}

@Composable
private fun UserAvatar(name: String, backgroundColor: Color =
Primary) {
    Surface(
        modifier = Modifier.size(48.dp),
        shape = CircleShape,
        color = backgroundColor.copy(alpha = 0.2f)
    ) {
        Box(contentAlignment = Alignment.Center) {
            Text(
                text = name.take(1).uppercase(),
                style = MaterialTheme.typography.titleMedium,
                fontWeight = FontWeight.Bold,
                color = backgroundColor
            )
        }
    }
}

```

Файл GhostModeScreen.kt

```

package com.runsocial.app.presentation.screens.run

private fun formatDuration(millis: Long): String {
    val duration = millis.milliseconds
    val minutes = duration.inWholeMinutes
    val seconds = duration.inWholeSeconds % 60
    return "%02d:%02d".format(minutes, seconds)
}

```

Файл ActiveRunScreen.kt

```

Box(
    modifier = Modifier
        .fillMaxSize()
        .background(MaterialTheme.colorScheme.background)
) {
    // Real Yandex MapKit View
    val context = androidx.compose.ui.platform.LocalContext.current
}

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						86
Изм.	Лист	№докум.	Подпись	Дата		

```

val mapView = remember { com.yandex.mapkit.mapview.MapView(context) }

DisposableEffect(mapView) {
    val map = mapView.mapWindow.map
    val cameraListener = CameraListener{ _, _, reason, _ ->
        if (reason == CameraUpdateReason.GESTURES) {
            followUserLocation = false
        }
    }
    map.addCameraListener(cameraListener)
    mapView.onStart()
    onDispose {
        map.removeCameraListener(cameraListener)
        mapView.onStop()
    }
}

androidx.compose.ui.viewinterop.AndroidView(
    factory = { mapView },
    modifier = Modifier.fillMaxSize(),
    update = { view ->
        val map = view.mapWindow.map
        if (followUserLocation && uiState.currentLatitude !=
            null && uiState.currentLongitude != null) {
            val targetLat = uiState.currentLatitude!!
            val targetLon = uiState.currentLongitude!!
            val distanceToTarget = FloatArray(1)
            Location.distanceBetween(
                map.cameraPosition.target.latitude,
                map.cameraPosition.target.longitude,
                targetLat,
                targetLon,
                distanceToTarget
            )
            val currentZoom = map.cameraPosition.zoom.takeIf { it > 0f } ?:
                FOLLOW_MIN_ZOOM
            val followZoom = if (currentZoom < FOLLOW_MIN_ZOOM) FOLLOW_MIN_ZOOM
                else currentZoom

            if (distanceToTarget[0] >
                CAMERA_RECENTER_DISTANCE_METERS || map.cameraPosition.zoom <
                FOLLOW_MIN_ZOOM) {
                map.move(
                    com.yandex.mapkit.map.CameraPosition(
                        com.yandex.mapkit.geometry.Point(targetLat, targetLon),
                        followZoom,
                        0f,
                        0f
                    ),
                    Animation(Animation.Type.SMOOTH, 0.35f),
                    null
                )
            }
        }
    }
}

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						87
Изм.	Лист	№докум.	Подпись	Дата		

```

        }
    }
    map.mapObjects.clear()
    uiState.ghostTrack?.let { ghost ->
    valghostPoints = ghost.points.map {
    com.yandex.mapkit.geometry.Point(it.latitude, it.longitude)
        }
    valghostPolyline = com.yandex.mapkit.geometry.Polyline(ghostPoints)
    map.mapObjects.addPolyline(ghostPolyline).apply {
    setStrokeColor(Color.Gray.copy(alpha = 0.5f).toArgb())
    setStrokeWidth(3f)
        }
    }
    if (uiState.routePoints.size>= 2) {
    val points = uiState.routePoints.map {
    com.yandex.mapkit.geometry.Point(it.latitude, it.longitude)
        }
    val polyline = com.yandex.mapkit.geometry.Polyline(points)
    map.mapObjects.addPolyline(polyline).apply {
    setStrokeColor(Primary.toArgb())
    setStrokeWidth(5f)
        }
    }
    participants.forEach { participant ->
    participant.currentLocation?.let { loc ->
    val point = com.yandex.mapkit.geometry.Point(loc.latitude,
    loc.longitude)
    WindowRecomposer crash
    valtagView = createParticipantView(
        context,
        participant.userName,
        participant.distanceCovered,
        uiState.isJointRun
    )
    valviewProvider = ViewProvider(tagView)
    map.mapObjects.addPlacemark(point, viewProvider)
        }
    }
    )
    if (uiState.currentLatitude != null
    &&uiState.currentLongitude != null) {
    FloatingActionButton(
    onClick = {
    val map = mapView.mapWindow.map
    valtargetPoint = com.yandex.mapkit.geometry.Point(
    uiState.currentLatitude!!,
    uiState.currentLongitude!!
    )
    map.move(

```

```

com.yandex.mapkit.map.CameraPosition(targetPoint,
RECENTER_BUTTON_ZOOM, 0f, 0f),
Animation(Animation.Type.SMOOTH, 0.4f),
            null
        )
followUserLocation = true
    },
containerColor = if (followUserLocation) Primary else
MaterialTheme.colorScheme.surface,
contentColor = if (followUserLocation) Color.White else Primary
    ) {
Icon(
imageVector = Icons.Default.MyLocation,
contentDescription = stringResource(R.string.center_on_me)
    )
    }
    }
    // Stats overlay at the top
Column(modifier = Modifier.align(Alignment.TopCenter)) {
StatsOverlay(
            distance = uiState.distanceKm,
            duration = uiState.formattedDuration,
            pace = uiState.formattedSpeed,
ghostComparison = uiState.ghostComparison
    )

        if (participants.isNotEmpty()) {
Spacer(modifier = Modifier.height(8.dp))
ParticipantsList(participants = participants, showDetails =
uiState.isJointRun)
        }
    }
ControlsPanel(
isRunning = uiState.isRunning,
isPaused = uiState.isPaused,
onPauseResume = {
            if (uiState.isPaused) viewModel.resumeRun()
            else viewModel.pauseRun()
        },
onStop = { viewModel.stopRun() },
onDiscard = {
viewModel.discardRun()
onDiscardRun()
        },
        modifier = Modifier.align(Alignment.BottomCenter
    // Nearby Friend Notification
AnimatedVisibility(
            visible = uiState.canInviteNearby,
            enter = slideInVertically{ it } + fadeIn(),
            exit = slideOutVertically{ it } + fadeOut(),
            modifier = Modifier

```

```

        .align(Alignment.BottomCenter)
        .padding(bottom = 120.dp)
        .padding(horizontal = 16.dp)
    ) {
        uiState.nearbyFriend?.let { friend ->
        NearbyFriendNotification(
            friendName = friend.userName,
            onInvite = viewModel::inviteNearbyFriend,
            onDismiss = { viewModel.dismissProximityInvite(friend.userId) }
        )
    }
    if (uiState.showInviteDialog) {
        AlertDialog(
            onDismissRequest = viewModel::declineInvite,
            title = {
                Text(stringResource(R.string.joint_run_invite)) },
            text = {
                Text(stringResource(R.string.friend_invites_you,
                    uiState.incomingInvite?.initiatorName ?:
                    stringResource(R.string.friend))) },
            confirmButton = {
                Button(onClick = viewModel::acceptInvite) {
                    Text(stringResource(R.string.accept))
                }
            },
            dismissButton = {
                TextButton(onClick = viewModel::declineInvite) {
                    Text(stringResource(R.string.decline))
                }
            }
        )
    }
    if (uiState.isJoiningMeeting) {
        Box(
            modifier = Modifier
                .fillMaxSize()
                .background(Color.Black.copy(alpha = 0.5f)),
        )
    }
}

LaunchedEffect(Unit) {
    viewModel.startRun()
}

private const val CAMERA_RECENTER_DISTANCE_METERS = 4f
private const val FOLLOW_MIN_ZOOM = 17.5f
private const val RECENTER_BUTTON_ZOOM = 18.5f
@Composable

```

```

private fun StatsOverlay(
    distance: Double,
    duration: String,
    pace: String,
    ghostComparison: com.runsocial.app.domain.model.GhostComparison =
        com.runsocial.app.domain.model.GhostComparison.NotAvailable,
    modifier: Modifier = Modifier
) {
    Card(
        modifier = modifier
        .fillMaxWidth()
        .padding(16.dp)
        .statusBarsPadding(),
    ) {
        // Ghost info row if available
        if (ghostComparison
            !is com.runsocial.app.domain.model.GhostComparison.NotAvailable) {
            GhostStatusRow(ghostComparison)
            HorizontalDivider(modifier = Modifier.padding(vertical = 12.dp))
        }
        Text(
            text = "%.2f".format(java.util.Locale.US, distance),
            style = MaterialTheme.typography.displayLarge.copy(
                fontSize = 72.sp,
                fontWeight = FontWeight.Bold
            ),
            color = Primary
        )
        // Divider
        Box(
            modifier = Modifier
                .height(40.dp)
                .width(1.dp)
                .background(MaterialTheme.colorScheme.outlineVariant)
        )

        StatColumn(
            value = pace,
            label = stringResource(R.string.speed_unit_kmh)
        )
    }
}

@Composable
private fun GhostStatusRow(comparison:
    com.runsocial.app.domain.model.GhostComparison) {
    val (text, color, icon) = when (comparison) {
        is com.runsocial.app.domain.model.GhostComparison.Ahead ->
            Triple(stringResource(R.string.ahead_by, comparison.distanceMeters),
                Success, Icons.Default.TrendingUp)
    }
}

```

```

        is com.runsocial.app.domain.model.GhostComparison.Behind ->
Triple(stringResource(R.string.behind_by,
comparison.distanceMeters), Warning, Icons.Default.TrendingDown)
        is com.runsocial.app.domain.model.GhostComparison.OnPace ->
        Triple(stringResource(R.string.on_pace_ghost), Primary,
Icons.Default.Equalizer) // Changed from Remove to Equalizer
        else ->Triple("", Primary, Icons.Default.TrendingUp)
    }
@Composable
private fun ControlsPanel(
    isRunning: Boolean,
    isPaused: Boolean,
    onPauseResume: () -> Unit,
    onStop: () -> Unit,
    onDiscard: () -> Unit,
    modifier: Modifier = Modifier
) {
    Icon(
        if (isPaused) Icons.Default.PlayArrow else
Icons.Default.Pause,
        contentDescription = if (isPaused) stringResource(R.string.resume)
else stringResource(R.string.pause),
        tint = Color.White,
        modifier = Modifier.size(36.dp)
    )
    IconButton(
        onClick = onStop,
        modifier = Modifier
        .size(56.dp)
        .clip(CircleShape)
        .background(Primary)
    ) {
        Icon(
            Icons.Default.Stop,
            contentDescription = stringResource(R.string.finish),
            tint = Color.White
        )
    }
}

@Composable
private fun NearbyFriendNotification(
    friendName: String,
    onInvite: () -> Unit,
    onDismiss: () -> Unit
) {
    Button(
        shape = RoundedCornerShape(8.dp)

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						92
Изм.	Лист	№докум.	Подпись	Дата		

```

        ) {
            Text(stringResource(R.string.invite_to_run_short),
fontSize = 12.sp)
        }

Spacer(modifier = Modifier.width(8.dp))

IconButton(
onClick = onDismiss,
        modifier = Modifier.size(24.dp)
    ) {
        Icon(Icons.Default.Close, null, tint = Color.White.copy(alpha =
0.7f), modifier = Modifier.size(16.dp))
    }
}

}

@Composable
private fun ActiveRunScreen_Stub() {} // Keep it here for structure
private fun createParticipantView(
    context: android.content.Context,
    name: String,
    distance: Double,
showDetails: Boolean
): android.view.View {
    val density = context.resources.displayMetrics.density

    // Root layout
    val root = LinearLayout(context).apply {
        layoutParams = android.view.ViewGroup.LayoutParams(
            android.view.ViewGroup.LayoutParams.WRAP_CONTENT,
            android.view.ViewGroup.LayoutParams.WRAP_CONTENT
        )
        orientation = LinearLayout.VERTICAL
        gravity = Gravity.CENTER_HORIZONTAL
    }

    // The Circle Marker
    val circle = android.widget.ImageView(context).apply {
        val size = (16 * density).toInt()
        layoutParams = LinearLayout.LayoutParams(size, size)
        val drawable = GradientDrawable().apply {
            shape = GradientDrawable.OVAL
            setColor(android.graphics.Color.parseColor("#4CAF50")) // Green for
friends
            setStroke((2 * density).toInt(), android.graphics.Color.WHITE)
        }
        background = drawable
    }
    root.addView(circle)
}

```

```

        // Name Label (Always visible)
valnameText = TextView(context).apply {
    text = name
    setTextColor(android.graphics.Color.BLACK)
    textSize = 12f
    setTypeface(null, android.graphics.Typeface.BOLD)
    setPadding((4 * density).toInt(), 0, (4 * density).toInt(), 0)
    gravity = Gravity.CENTER

    // Background for text to improve legibility on map
valtextBg = GradientDrawable().apply {
    shape = GradientDrawable.RECTANGLE
    cornerRadius = 8 * density
    setColor(android.graphics.Color.rgb(180, 255, 255, 255)) // White
    with alpha
    }
    background = textBg

layoutParams = LinearLayout.LayoutParams(
    LinearLayout.LayoutParams.WRAP_CONTENT,
    LinearLayout.LayoutParams.WRAP_CONTENT
).apply {
    topMargin = (2 * density).toInt()
    }
    }
root.addView(nameText)

    if (showDetails) {
        // Distance Label (Only in Joint Runs)
valdistText = TextView(context).apply {
    text = context.getString(R.string.distance_km_format,
distance)
    setTextColor(android.graphics.Color.parseColor("#333333"))
    textSize = 10f
    setPadding((4 * density).toInt(), 0, (4 * density).toInt(), 0)
    gravity = Gravity.CENTER

valdistBg = GradientDrawable().apply {
    shape = GradientDrawable.RECTANGLE
    cornerRadius = 6 * density
    setColor(android.graphics.Color.rgb(180, 255, 255, 255))
    }
    background = distBg

layoutParams = LinearLayout.LayoutParams(
    LinearLayout.LayoutParams.WRAP_CONTENT,
    LinearLayout.LayoutParams.WRAP_CONTENT
).apply {
    topMargin = (1 * density).toInt()
    }
    }

```

```

    }
    root.addView(distText)
    }

    return root
}

```

Файл AuthRepositoryImpl.kt

```

private val context.dataStore: DataStore<Preferences> by
preferencesDataStore(name = "auth_prefs")
@Singleton
class AuthRepositoryImpl @Inject constructor(
    @ApplicationContext private val context: Context,
    private val authApi: AuthApi,
    private val userDao: UserDao
) : AuthRepository {

    private object PrefsKeys {
        val ACCESS_TOKEN = stringPreferencesKey("access_token")
        val USER_ID = stringPreferencesKey("user_id")
        const val OFFLINE_TOKEN = "offline_token"
    }

    val response = executeWithRetry {
        authApi.login(LoginRequest(email, password))
    }
    saveTokens(authResponse.token, authResponse.userId)

    val user = User(
        id = authResponse.userId,
        email = email,
        displayName = authResponse.username,
        avatarUrl = authResponse.avatarUrl
    )
    userDao.deleteAll()
    userDao.insert(UserEntity.fromDomain(user))
    Result.success(user)
    } else {
        val errorMsg = response.errorBody()?.string() ?: "Login failed"
        Result.failure(Exception(errorMsg))
    }
    } catch (e: Exception) {
        Result.failure(e)
    }
}

    override suspend fun register(email: String, password: String,
        displayName: String): Result<User> {
        return try {
            val response = executeWithRetry {
                authApi.register(RegisterRequest(email, displayName, password))
            }
            saveTokens(authResponse.token, authResponse.userId)

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						95
Изм.	Лист	№докум.	Подпись	Дата		

```

val user = User(
    id = authResponse.userId,
    email = email,
    displayName = authResponse.username,
    avatarUrl = authResponse.avatarUrl
)
userDao.deleteAll()
userDao.insert(UserEntity.fromDomain(user))
Result.success(user)
    } else {
        val errorMsg = response.errorBody()?.string()?.takeIf {
            it.isNotBlank() }
            ?: "Registration failed"
        Result.failure(Exception(errorMsg))
    }
    } catch (e: Exception) {
        Result.failure(e)
    }
}

override suspend fun logout() {
    clearLocalAuthState()
}

override suspend fun getCurrentUser(): User? {
    // For now, return from local DB. Later maybe fetch from API
    /users/me
    return userDao.getCurrentUser()?.toDomain()
}

override fun observeCurrentUser(): Flow<User?> {
    return userDao.observeCurrentUser().map { it?.toDomain() }
}

override suspend fun isLoggedIn(): Boolean {
    val prefs = context.dataStore.data.first()
    val userId = prefs[PrefsKeys.USER_ID]
    val token = prefs[PrefsKeys.ACCESS_TOKEN]
    val isValidSession = !userId.isNullOrEmpty() &&
        !token.isNullOrEmpty() &&
        token != PrefsKeys.OFFLINE_TOKEN

    if (!isValidSession && (!userId.isNullOrEmpty() ||
        !token.isNullOrEmpty())) {
        clearLocalAuthState()
    }

    return isValidSession
}

```

```

        override suspend fun getAuthToken(): String? {
            return
context.dataStore.data.first()[PrefsKeys.ACCESS_TOKEN]
        }

        override suspend fun refreshToken(): Result<Unit> {
            // Not implemented on backend yet
            return Result.success(Unit)
        }
        override suspend fun forgotPassword(email: String): Result<Unit>
{
            return try {
val response =
authApi.forgotPassword(com.runsocial.app.data.remote.dto.ForgotPassw
ordRequest(email))
                if (response.isSuccessful) {
Result.success(Unit)
                } else {
val errorMsg = response.errorBody()?.string() ?: "Failed to send
reset code"
Result.failure(Exception(errorMsg))
                }
            } catch (e: Exception) {
Result.failure(e)
            }
        }
        override suspend fun verifyResetCode(email: String, code:
String): Result<Unit> {
            return try {
val response =
authApi.verifyCode(com.runsocial.app.data.remote.dto.VerifyCodeReque
st(email, code))
                if (response.isSuccessful) {
Result.success(Unit)
                } else {
val errorMsg = response.errorBody()?.string() ?: "Invalid code"
Result.failure(Exception(errorMsg))
                }
            } catch (e: Exception) {
Result.failure(e)
            }
        }
        override suspend fun resetPasswordWithCode(email: String, code:
String, new: String): Result<Unit> {
            return try {
val response =
authApi.resetPassword(com.runsocial.app.data.remote.dto.ResetPasswor
dRequest(email, code, new))
                if (response.isSuccessful) {
Result.success(Unit)
                } else {

```

```

val errorMsg = response.errorBody()?.string() ?: "Failed to reset
password"
Result.failure(Exception(errorMsg))
    }
    } catch (e: Exception) {
Result.failure(e)
    }
}
    override suspend fun updateProfile(user: User): Result<User> {
userDao.update(UserEntity.fromDomain(user))
    return Result.success(user)
    }
    override suspend fun changePassword(old: String, new: String):
Result<Unit> {
    return try {
val response = authApi.changePassword(ChangePasswordRequest(old,
new))
        if (response.isSuccessful) {
Result.success(Unit)
        } else {
val errorMsg = response.errorBody()?.string() ?: "Failed to change
password"
Result.failure(Exception(errorMsg))
        }
    } catch (e: Exception) {
Result.failure(e)
    }
}
    override suspend fun updateEmail(newEmail: String, password:
String): Result<Unit> {
    return try {
        // 1. Call API
val response =
authApi.updateEmail(com.runsocial.app.data.remote.dto.UpdateEmailReq
uest(newEmail, password))
        if (response.isSuccessful) {
            // 2. Update local DB
val currentUser = userDao.getCurrentUser()
            if (currentUser != null) {
val updatedUser = currentUser.copy(email = newEmail)
userDao.update(updatedUser)
            }
Result.success(Unit)
        } else {
val errorMsg = response.errorBody()?.string() ?: "Failed to update
email"
Result.failure(Exception(errorMsg))
        }
    } catch (e: Exception) {
Result.failure(e)
    }
}

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						98
Изм.	Лист	№докум.	Подпись	Дата		

```

    }
    override suspend fun updateAvatar(avatarUrl: String?):
Result<Unit> {
        return try {
val response = authApi.changePhoto(ChangePhotoRequest(avatarUrl))
            if (response.isSuccessful) {
val currentUser = userDao.getCurrentUser()
                if (currentUser != null) {
userDao.update(currentUser.copy(avatarUrl = avatarUrl))
                }
Result.success(Unit)
            } else {
val errorMsg = response.errorBody()?.string()?.takeIf {
it.isNotBlank() }
?: "Failed to update photo"
Result.failure(Exception(errorMsg))
            }
        } catch (e: Exception) {
Result.failure(e)
        }
    }

    override suspend fun uploadAvatar(imageBytes: ByteArray,
fileName: String): Result<String> {
        return try {
val extension = fileName.substringAfterLast('.', "").lowercase()
val mediaType = when (extension) {
            "png" -> "image/png"
            "webp" -> "image/webp"
            else -> "image/jpeg"
        }.toMediaTypeOrNull()
val requestBody = imageBytes.toRequestBody(mediaType)
val part = MultipartBody.Part.createFormData("photo", fileName,
requestBody)
val response = authApi.uploadPhoto(part)
val currentUser = userDao.getCurrentUser()
            if (currentUser != null) {
userDao.update(currentUser.copy(avatarUrl = avatarUrl))
            }
Result.success(avatarUrl)
        } else {
val errorMsg = response.errorBody()?.string()?.takeIf {
it.isNotBlank() }
?: "Failed to update photo"
Result.failure(Exception(errorMsg))
        }
    } catch (e: Exception) {
Result.failure(e)
    }
}

    private suspend fun saveTokens(accessToken: String, userId:
String) {

```

```

context.dataStore.edit { prefs ->
prefs[PrefsKeys.ACCESS_TOKEN] = accessToken
prefs[PrefsKeys.USER_ID] = userId
    }
}

private suspend fun clearLocalAuthState() {
context.dataStore.edit { it.clear() }
userDao.deleteAll()
}

private suspend fun <T>executeWithRetry(
maxAttempts: Int = 3,
initialDelayMs: Long = 1500,
    block: suspend () -> T
): T {
    var attempt = 1
    var delayMs = initialDelayMs

    while (true) {
        try {
            return block()
        } catch (e: Exception) {
            val shouldRetry = e is IOException
            if (!shouldRetry || attempt >= maxAttempts) {
                throw e
            }
            delay(delayMs)
            delayMs *= 2
            attempt++
        }
    }
}

```

Файл RunRepositoryImpl.kt

```

@Singleton
class RunRepositoryImpl @Inject constructor(
    private val runDao: RunDao,
    private val runApi: RunApi
) : RunRepository {
    private val converters = Converters()
    override suspend fun saveRun(run: Run) {
        val routePointsJson = converters.fromRoutePointList(run.routePoints)
        runDao.insert(RunEntity.fromDomain(run, routePointsJson))
    }
    override suspend fun updateRun(run: Run) {
        val routePointsJson = converters.fromRoutePointList(run.routePoints)
        runDao.update(RunEntity.fromDomain(run, routePointsJson))
    }
    override suspend fun getRunById(id: String): Run? {

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						100
Изм.	Лист	№докум.	Подпись	Дата		

```

        return runDao.getById(id)?.let { entity ->
val routePoints = converters.toRoutePointList(entity.routePointsJson)
entity.toDomain(routePoints)
        }
    }
    override fun getRunsByUser(userId: String): Flow<List<Run>> {
        return runDao.getAllByUser(userId).map { entities ->
entities.map{ entity ->
val routePoints = converters.toRoutePointList(entity.routePointsJson)
entity.toDomain(routePoints)
        }
    }
    }
    override fun getRecentRuns(userId: String, limit: Int):
Flow<List<Run>> {
        return runDao.getRecentByUser(userId, limit).map { entities
->
entities.map{ entity ->
val routePoints = converters.toRoutePointList(entity.routePointsJson)
entity.toDomain(routePoints)
        }
    }
    }
    override suspend fun deleteRun(id: String) {
runDao.softDeleteById(id)
    }
    override suspend fun syncRun(run: Run): Result<Unit> {
        return try {
val runDto = com.runsocial.app.data.remote.dto.CreateRunRequest(
            id = run.id,
startTime = run.startTime.format(DateTimeFormatter.ISO_DATE_TIME),
endTime = run.endTime?.format(DateTimeFormatter.ISO_DATE_TIME) ?:
"",
distanceMeters = run.distanceMeters,
durationMillis = run.durationMillis,
avgPaceMinPerKm = run.avgPaceSecondsPerKm / 60.0,
routePoints = run.routePoints.map{ point ->
NetworkLocationPoint(
                    latitude = point.latitude,
                    longitude = point.longitude,
                    timestamp = point.timestamp,
                    altitude = point.altitude,
distanceMeters = point.distanceFromStart
                )
            },
meetingId = run.meetingId
        )
val response = runApi.saveRun(runDto)
        if (response.isSuccessful) {
Result.success(Unit)
        } else {

```

```

Result.failure(Exception("Failed to sync run:
${response.message()}"))
    }
    } catch (e: Exception) {
Result.failure(e)
    }
}

override suspend fun syncFromNetwork(): Result<Unit> {
println("DEBUG: syncFromNetwork starting")
    return try {
val response = runApi.getRuns()
println("DEBUG: syncFromNetwork response received:
${response.code()}")
        if (response.isSuccessful) {
val runs = response.body() ?: emptyList()
println("DEBUG: syncFromNetwork fetched ${runs.size} runs")
runs.forEach { dto ->
val existing = runDao.getById(dto.id)
            if (existing == null) {
                try {
println("DEBUG: syncFromNetwork inserting new run: ${dto.id}")
val start = java.time.LocalDateTime.parse(dto.startTime)
val end = java.time.LocalDateTime.parse(dto.endTime)

val run = com.runsocial.app.domain.model.Run(
                    id = dto.id,

userId = dto.userId,
startTime = start,
endTime = end,
distanceMeters = dto.distanceMeters,
durationMillis = dto.durationMillis,
avgPaceSecondsPerKm = (dto.avgPaceMinPerKm * 60.0).toInt(),
caloriesBurned = (dto.distanceMeters / 1000.0 * 60.0).toInt(),
routePoints = dto.routePoints.map{ p ->
com.runsocial.app.domain.model.RoutePoint(
                                latitude = p.latitude,
                                longitude = p.longitude,
                                timestamp = p.timestamp,
                                altitude = p.altitude ?:

0.0,
distanceFromStart = p.distanceMeters ?: 0.0
                                )
                                },
                                status =

com.runsocial.app.domain.model.RunStatus.COMPLETED,
meetingId = dto.meetingId,
ghostTrackId = null
                                )

valroutePointsJson = converters.fromRoutePointList(run.routePoints)

```

```

runDao.insert(com.runsocial.app.data.local.entity.RunEntity.fromDomain(
run, routePointsJson).copy(syncStatus = SyncStatus.SYNCED))
        } catch (e: Exception) {
println("DEBUG: syncFromNetwork error parsing run ${dto.id}:
${e.message}")
        }
        } else {
println("DEBUG: syncFromNetwork run ${dto.id} already exists
locally")
        }
    }
Result.success(Unit)
    } else {
println("DEBUG: syncFromNetwork failed with code ${response.code()}:
${response.errorBody()?.string()}")
Result.failure(Exception("Failed to fetch runs from network"))
    }
    } catch (e: Exception) {
println("DEBUG: syncFromNetwork exception: ${e.message}")
e.printStackTrace()
Result.failure(e)
    }
}
override suspend fun getPendingSyncRuns(): List<Run> {
    return runDao.getPendingSync().map { entity ->
val routePoints = converters.toRoutePointList(entity.routePointsJson)
entity.toDomain(routePoints)
    }
}
override suspend fun updateSyncStatus(runId: String, status:
SyncStatus) {
runDao.updateSyncStatus(runId, status)
    override suspend fun getTotalRuns(userId: String): Int {
        return runDao.getTotalRuns(userId)
    }
    override suspend fun getWeeklyDistance(userId: String): Double {
val startOfWeek = LocalDateTime.now()
.with(TemporalAdjusters.previousOrSame(DayOfWeek.MONDAY))
.withHour(0).withMinute(0).withSecond(0)
        return runDao.getWeeklyDistance(userId,
startOfWeek.toString()) ?: 0.0
    }
    override suspend fun getMonthlyDistance(userId: String): Double
{
val startOfMonth = LocalDateTime.now()
.withDayOfMonth(1)
.withHour(0).withMinute(0).withSecond(0)
        return runDao.getMonthlyDistance(userId,
startOfMonth.toString()) ?: 0.0
    }
}

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						103
Изм.	Лист	№докум.	Подпись	Дата		

}

Файл SocialApi.kt

```
interface SocialApi
    @GET("api/meetings")
    suspend fun getMeetings(
        @Query("lat") latitude: Double,
        @Query("lon") longitude: Double,
        @Query("radius") radiusKm: Double = 10.0
    ): Response<List<MeetingDto>>
    @GET("api/meetings/{id}")
    suspend fun getMeetingById(@Path("id") id: String):
Response<MeetingDto>
    @POST("api/meetings")
    suspend fun createMeeting(@Body meeting: MeetingDto):
Response<MeetingDto>
    @POST("api/meetings/{id}/join")
    suspend fun joinMeeting(@Path("id") id: String): Response<Unit>
    @POST("api/meetings/{id}/leave")
    suspend fun leaveMeeting(@Path("id") id: String): Response<Unit>
    @POST("api/meetings/{id}/start")
    suspend fun startMeeting(@Path("id") id: String):
Response<MeetingDto>
    @POST("api/meetings/{id}/end")
    suspend fun endMeeting(@Path("id") id: String):
Response<MeetingDto>
    @GET("api/meetings/participating")
    suspend fun getParticipatingMeetings():
Response<List<MeetingDto>>
    @GET("api/meetings/{id}/results")
    suspend fun getMeetingResults(@Path("id") id: String):
Response<List<com.runsocial.app.data.remote.dto.NetworkRun>>
    @GET("api/meetings/invites")
    suspend fun getMyInvites(): Response<List<MeetingInvitationDto>>
    @POST("api/meetings/{id}/invite")
    suspend fun sendInvite(
        @Path("id") meetingId: String,
        @Body request: CreateInvitationRequest
    ): Response<Unit>
    @POST("api/meetings/invites/{id}/respond")
    suspend fun respondToInvite(
        @Path("id") inviteId: String,
        @Query("accept") accept: Boolean
    ): Response<Unit>
}
```

Файл GhostTrack.kt

```
val avgPaceSecondsPerKm: Int,
val points: List<GhostPoint>,
val isPersonalBest: Boolean = false,
val recordedAt: LocalDateTime = LocalDateTime.now()
) {
```

					ПГУ 1.090301.39.001 ПЗ	Лист
						104
Изм.	Лист	№докум.	Подпись	Дата		

```

val formattedPace: String get() {
    val minutes = avgPaceSecondsPerKm / 60
    val seconds = avgPaceSecondsPerKm % 60
    return "%d:%02d".format(minutes, seconds)
}

}

data class GhostPoint(
    val latitude: Double,
    val longitude: Double,
    val timestampMillis: Long, // Milliseconds from start
    val distanceFromStart: Double
)

sealed class GhostComparison {
    data class Ahead(val distanceMeters: Double) : GhostComparison()
    data class Behind(val distanceMeters: Double) : GhostComparison()
    data object OnPace : GhostComparison()
    data object NotAvailable : GhostComparison()
}

```

Файл RunApi.kt

```

package com.runsocial.app.data.remote.api
interface RunApi {
    @GET("api/runs")
    suspend fun getRuns(
        @Query("page") page: Int = 0,
        @Query("size") size: Int = 20
    ): Response<List<NetworkRun>
    @GET("api/runs/{id}")
    suspend fun getRunById(@Path("id") id: String):
Response<NetworkRun>
    @POST("api/runs")
    suspend fun saveRun(@Body run: CreateRunRequest):
Response<NetworkRun>
    @POST("api/runs/sync")
    suspend fun syncRuns(@Body runs: List<NetworkRun>):
}

```

Файл AuthApi.kt

```

package com.runsocial.app.data.remote.api
interface AuthApi {
    @POST("api/auth/register")
    suspend fun register(@Body request: RegisterRequest):
Response<AuthResponse>
    @POST("api/auth/login")
    suspend fun login(@Body request: LoginRequest):
Response<AuthResponse>
    @POST("api/auth/change-password")
    suspend fun changePassword(@Body request:
ChangePasswordRequest): Response<ResponseBody>
    @POST("api/auth/update-email")
    suspend fun updateEmail(@Body request: UpdateEmailRequest):
Response<ResponseBody>
}

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						105
Изм.	Лист	№докум.	Подпись	Дата		

```

        @POST("api/auth/change-photo")
        suspend fun changePhoto(@Body request: ChangePhotoRequest):
Response<ResponseBody>
        @Multipart
        @POST("api/auth/upload-photo")
        suspend fun uploadPhoto(@Part photo: MultipartBody.Part):
Response<AvatarUploadResponse>
        @POST("api/auth/forgot-password")
        suspend fun forgotPassword(@Body request:
ForgotPasswordRequest): Response<ResponseBody>
        @POST("api/auth/verify-code")
        suspend fun verifyCode(@Body request: VerifyCodeRequest):
Response<ResponseBody>
        @POST("api/auth/reset-password")
        suspend fun resetPassword(@Body request: ResetPasswordRequest):
Response<ResponseBody>
    }

```

Файл GhostTrackEntity.kt

```

package com.runsocial.app.data.local.entity
@Entity(tableName = "ghost_tracks")
@TypeConverters(Converters::class)
fun toDomain(points: List<GhostPoint>): GhostTrack {
    return GhostTrack(
        id = id,
        userId = userId,
        routeId = routeId,
        totalDistanceMeters = totalDistanceMeters,
        totalTimeMillis = totalTimeMillis,
        avgPaceSecondsPerKm = avgPaceSecondsPerKm,
        points = points,
        isPersonalBest = isPersonalBest,
        recordedAt = recordedAt
    )
}

```

Файл UserEntity.kt

```

package com.runsocial.app.data.local.entity
@Entity(tableName = "users")
@TypeConverters(Converters::class)
data class UserEntity(
    @PrimaryKey val id: String,
    val email: String,
    val displayName: String,
    val avatarUrl: String?,
    val level: UserLevel,
    val totalDistanceMeters: Double,
    val totalTimeMillis: Long,
    val totalRuns: Int,
    val weeklyDistanceMeters: Double,
    val monthlyDistanceMeters: Double,

```

```

val createdAt: LocalDateTime
) {
    fun toDomain(): User {
        return User(
            id = id,
            email = email,
            displayName = displayName,
            avatarUrl = avatarUrl,
            level = level,
            totalDistanceMeters = totalDistanceMeters,
            totalTimeMillis = totalTimeMillis,
            totalRuns = totalRuns,
            weeklyDistanceMeters = weeklyDistanceMeters,
            monthlyDistanceMeters = monthlyDistanceMeters,
            createdAt = createdAt
        )
    }
}

```

Файл RunEntity.kt

```

package com.runsocial.app.data.local.entity
import androidx.room.Entity
import androidx.room.PrimaryKey
import androidx.room.TypeConverters
import com.runsocial.app.data.local.db.Converters
import com.runsocial.app.domain.model.RoutePoint
import com.runsocial.app.domain.model.Run
import com.runsocial.app.domain.model.RunStatus
import com.runsocial.app.domain.model.SyncStatus
import java.time.LocalDateTime
@Entity(tableName = "runs")
@TypeConverters(Converters::class)
data class RunEntity(
    val avgPaceSecondsPerKm: Int,
    val maxPaceSecondsPerKm: Int,
    val caloriesBurned: Int,
    val routePointsJson: String, // JSON serialized route points
    val status: RunStatus,
    val syncStatus: SyncStatus,
    val meetingId: String?,
    val ghostTrackId: String?,
    val isDeleted: Boolean = false
) {
    fun toDomain(routePoints: List<RoutePoint>): Run {
        return Run(
            id = id,
            userId = userId,
            startTime = startTime,
            endTime = endTime,
            distanceMeters = distanceMeters,
            durationMillis = durationMillis,
            avgPaceSecondsPerKm = avgPaceSecondsPerKm,
            maxPaceSecondsPerKm = maxPaceSecondsPerKm,

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						107
Изм.	Лист	№докум.	Подпись	Дата		

```

caloriesBurned = caloriesBurned,
routePoints = routePoints,
        status = status,
syncStatus = syncStatus,
meetingId = meetingId,
ghostTrackId = ghostTrackId,
isDeleted = isDeleted
    )
}
companion object {
    fun fromDomain(run: Run, routePointsJson: String): RunEntity
{
        return RunEntity(
            id = run.id,
userId = run.userId,
startTime = run.startTime,
endTime = run.endTime,
distanceMeters = run.distanceMeters,
durationMillis = run.durationMillis,
avgPaceSecondsPerKm = run.avgPaceSecondsPerKm,
maxPaceSecondsPerKm = run.maxPaceSecondsPerKm,
caloriesBurned = run.caloriesBurned,
routePointsJson = routePointsJson,
        status = run.status,
syncStatus = run.syncStatus,
meetingId = run.meetingId,
ghostTrackId = run.ghostTrackId,
isDeleted = run.isDeleted
        )
    }
}

```

Файл NavGraph.kt

```

package com.runsocial.app.presentation.navigation
val base = "active_run"
val params = mutableListOf<String>()
    if (ghostId != null) params.add("ghostId=$ghostId")
    if (meetingId != null)
params.add("meetingId=$meetingId")
    return if (params.isEmpty()) base else
"$base?${params.joinToString("&")}"
}
}
data object RunSummary : Screen("run_summary/{runId}") {
    fun createRoute(runId: String) = "run_summary/$runId"
}
// Profile
data object Profile : Screen("profile")
data object EditProfile : Screen("edit_profile")
data object Settings : Screen("settings")
data object Achievements : Screen("achievements")

```

```

// Social
data object Social : Screen("social")
data object CreateMeeting : Screen("create_meeting")
data object MeetingDetail : Screen("meeting/{meetingId}") {
    fun createRoute(meetingId: String) = "meeting/$meetingId"
}
// Ghost
data object GhostMode : Screen("ghost_mode")
// Statistics & History
data object History : Screen("history")
data object Stats : Screen("stats")
data object Friends : Screen("friends")
data object FriendProfile : Screen("friend_profile/{userId}") {
    fun createRoute(userId: String) = "friend_profile/$userId"
}
}
@Composable
fun RunSocialNavGraph(
    navController: NavHostController,
    startDestination: String = Screen.Login.route
) {
    LoginScreen(
        onLoginSuccess = {
            navController.navigate(Screen.Home.route) {
                popUpTo(Screen.Login.route) { inclusive = true }
            }
        },
        onNavigateToRegister = {
            navController.navigate(Screen.Register.route)
        },
        onNavigateToForgotPassword = {
            navController.navigate(Screen.ForgotPassword.route)
        }
    )

    composable(Screen.Register.route) {
        RegisterScreen(
            onRegisterSuccess = {
                navController.navigate(Screen.Home.route) {
                    popUpTo(Screen.Login.route) { inclusive = true }
                }
            },
            onNavigateBack = {
                navController.popBackStack()
            }
        )

        composable(Screen.ForgotPassword.route) {
            ForgotPasswordScreen(
                onNavigateBack = {
                    navController.popBackStack()
                }
            )
        }
    }
}

```

```

        }
    )
}
// Main screens
composable(Screen.Home.route) {
    HomeScreen(
        onStartRun = {
            navController.navigate(Screen.ActiveRun.createRoute())
        },
        onNavigateToProfile = {
            navController.navigate(Screen.Profile.route)
        },
        onNavigateToSocial = {
            navController.navigate(Screen.Social.route)
        },
        onNavigateToAchievements = {
            navController.navigate(Screen.Achievements.route)
        },
        onNavigateToGhostMode = {
            navController.navigate(Screen.GhostMode.route)
        },
        onNavigateToHistory = {
            navController.navigate(Screen.History.route)
        },
        onNavigateToStats = {
            navController.navigate(Screen.Stats.route)
        },
        onNavigateToFriends = {
            navController.navigate(Screen.Friends.route)
        }
    )
}
composable(
    route = Screen.ActiveRun.route,
    arguments = listOf(
        navArgument("ghostId") {
            type = NavType.StringType
            nullable = true
        },
        navArgument("meetingId") {
            type = NavType.StringType
            nullable = true
        }
    )
) {
    ActiveRunScreen(
        onRunCompleted = { runId ->
            navController.navigate(Screen.RunSummary.createRoute(runId)) {
                popUpTo(Screen.Home.route)
            }
        }
    )
}

```

```

        },
        onDiscardRun = {
            navController.popBackStack()
        }
    )
}
composable(Screen.GhostMode.route) {
    GhostModeScreen(
        onNavigateBack = { navController.popBackStack() },
        onStartGhostRun = { ghostId ->
            navController.navigate(Screen.ActiveRun.createRoute(ghostId)) {
                popUpTo(Screen.GhostMode.route) { inclusive = true }
            }
        }
    )
}
RunSummaryScreen(
    runId = runId,
    onNavigateBack = {
        navController.navigate(Screen.Home.route) {
            popUpTo(Screen.Home.route) { inclusive = true }
        }
    }
)
// Profile
composable(Screen.Profile.route) {
    ProfileScreen(
        onNavigateBack = { navController.popBackStack() },
        onNavigateToEditProfile = {
            navController.navigate(Screen.EditProfile.route) },
        onNavigateToSettings = {
            navController.navigate(Screen.Settings.route) },
        onLogout = {
            navController.navigate(Screen.Login.route) {
                popUpTo(0) { inclusive = true }
            }
        }
    )
}
composable(Screen.EditProfile.route) {
    EditProfileScreen(
        onNavigateBack = { navController.popBackStack() }
    )
}
composable(Screen.Settings.route) {
    SettingsScreen(
        onNavigateBack = { navController.popBackStack() }
    )
}

```

```

composable(Screen.Achievements.route) {
    AchievementsScreen(
        onNavigateBack = { navController.popBackStack() }
    )
}
// Social
composable(Screen.Social.route) {
    SocialScreen(
        onNavigateBack = { navController.popBackStack() },
        onMeetingClick = { meetingId ->

            navController.navigate(Screen.MeetingDetail.createRoute(meetingId))

        },
        onCreateMeeting = {
            navController.navigate(Screen.CreateMeeting.route)
        }
    )
}
composable(Screen.CreateMeeting.route) {
    CreateMeetingScreen(
        onNavigateBack = { navController.popBackStack() },
        onMeetingCreated = {
            navController.popBackStack()
        }
    )
}
composable(
    route = Screen.MeetingDetail.route,
    arguments = listOf(navArgument("meetingId") { type =
        NavType.StringType })
) {
    MeetingDetailScreen(
        onNavigateBack = { navController.popBackStack() },
        onStartSocialRun = { meetingId ->
            navController.navigate(Screen.ActiveRun.createRoute(meetingId =
                meetingId))
        }
    )
}
// Statistics & History
composable(Screen.History.route) {
    HistoryScreen(
        onNavigateBack = { navController.popBackStack() },
        onRunClick = { runId ->
            navController.navigate(Screen.RunSummary.createRoute(runId))
        }
    )
}
composable(Screen.Stats.route) {
    StatsScreen(
        onNavigateBack = { navController.popBackStack() }
    )
}

```

```

        )
    }
    composable(Screen.Friends.route) {
        FriendsScreen(
            onNavigateBack = { navController.popBackStack() },
            onInviteToRun = { friendId ->
                navController.navigate(Screen.CreateMeeting.route)
            },
            onNavigateToFriendProfile = { friendId ->
                navController.navigate(Screen.FriendProfile.createRoute(friendId))
            }
        )
    }
}

```

Файл MainActivity.kt

```

package com.runsocial.app.presentation
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {
    private val requestPermissionLauncher =
        registerForActivityResult(
            androidx.activity.result.contract.ActivityResultContracts.RequestMultiplePermissions()
        ) { permissions ->
            val fineLocationGranted =
                permissions[android.Manifest.permission.ACCESS_FINE_LOCATION] ?: false
            val coarseLocationGranted =
                permissions[android.Manifest.permission.ACCESS_COARSE_LOCATION] ?: false

            if (fineLocationGranted || coarseLocationGranted) {
                startMapKit()
            }
        }

    private var isMapKitStarted = false
    val permissionsToRequest = mutableListOf(
        android.Manifest.permission.ACCESS_FINE_LOCATION,
        android.Manifest.permission.ACCESS_COARSE_LOCATION
    )

    requestPermissionLauncher.launch(permissionsToRequest.toTypedArray())
    setContent {
        val viewModel: MainViewModel =
            androidx.hilt.navigation.compose.hiltViewModel()
        val theme by viewModel.theme.collectAsState()
        val language by viewModel.language.collectAsState()
        val startDestination by viewModel.startDestination.collectAsState()
        LaunchedEffect(language) {

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						113
Изм.	Лист	№докум.	Подпись	Дата		

```

val appLocale: LocaleListCompat =
    LocaleListCompat.forLanguageTags(language)
    AppCompatActivity.setApplicationLocales(appLocale)
    }
val darkTheme = when (theme) {
    AppTheme.LIGHT -> false
    AppTheme.DARK -> true
    AppTheme.SYSTEM -> isSystemInDarkTheme()
    }
RunSocialTheme(darkTheme = darkTheme) {
    Surface(
        modifier = Modifier.fillMaxSize(),
        color = MaterialTheme.colorScheme.background
    ) {
        if (startDestination != null) {
            val navController = rememberNavController()
            RunSocialNavGraph(
                navController = navController,
                startDestination = startDestination!!
            )
        } else {
        }
    }
}
}
private fun checkPermissionsAndStartMapKit() {
    if (androidx.core.content.ContextCompat.checkSelfPermission(
        this,
    ) {
        startMapKit()
    }
}
private fun startMapKit() {
    if (!isMapKitStarted) {
        MapKitFactory.getInstance().onStart()
        isMapKitStarted = true
    }
}
}

```

Файл RepositoryModule.kt

```

package com.runsocial.app.di
import com.runsocial.app.domain.repository.*
import com.runsocial.app.data.repository.*
import dagger.Binds
import dagger.Module
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton
@Module
@InstallIn(SingletonComponent::class)
abstract class RepositoryModule {

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						114
Изм.	Лист	№докум.	Подпись	Дата		

```

        @Binds
        @Singleton
        abstract fun bindAuthRepository(impl: AuthRepositoryImpl):
AuthRepository
        @Binds
        @Singleton
        abstract fun bindRunRepository(impl: RunRepositoryImpl):
RunRepository
        @Binds
        @Singleton
        abstract fun bindSocialRepository(impl: SocialRepositoryImpl):
SocialRepository
        @Binds
        @Singleton
        abstract fun bindAchievementRepository(impl:
AchievementRepositoryImpl): AchievementRepository
        @Binds
        @Singleton
        abstract fun bindGhostRepository(impl: GhostRepositoryImpl):
GhostRepository
        @Binds
        @Singleton
        abstract fun bindPreferencesRepository(impl:
PreferencesRepositoryImpl): PreferencesRepository
    }

```

Файл LocationTrackingService.kt

```

package com.runsocial.app.service
@AndroidEntryPoint
class LocationTrackingService : Service() {
    @Inject
    lateinit var fusedLocationClient: FusedLocationProviderClient
    @Inject
    lateinit var locationTrackingManager:
com.runsocial.app.data.manager.LocationTrackingManager
    private val serviceScope = CoroutineScope(SupervisorJob() +
Dispatchers.IO)
    private val locationCallback = object : LocationCallback() {
        override fun onLocationResult(result: LocationResult) {
            result.lastLocation?.let { location ->
            serviceScope.launch {
                locationTrackingManager.emitLocation(location)
            }
        }
    }
    private val locationRequest = LocationRequest.Builder(
        Priority.PRIORITY_HIGH_ACCURACY,
        1500L
    ).apply {
        setMinUpdateIntervalMillis(1000L)
        setMinUpdateDistanceMeters(2f)
    }
}

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						115
Изм.	Лист	№докум.	Подпись	Дата		

```

setWaitForAccurateLocation(true)
}.build()
    override fun onCreate() {
super.onCreate()
startForeground()
    }
    override fun onStartCommand(intent: Intent?, flags: Int,
startId: Int): Int {
    when (intent?.action) {
        ACTION_START ->startTracking()
        ACTION_PAUSE ->pauseTracking()
        ACTION_RESUME ->resumeTracking()
        ACTION_STOP ->stopTracking()
    }
    return START_STICKY
}
    override fun onBind(intent: Intent?): IBinder? = null
    private fun startForeground() {
val notification = createNotification()

        if (Build.VERSION.SDK_INT>= Build.VERSION_CODES.Q) {
startForeground(
            NOTIFICATION_ID,
            notification,
            ServiceInfo.FOREGROUND_SERVICE_TYPE_LOCATION
        )
        } else {
startForeground(NOTIFICATION_ID, notification)
        }
    }
    private fun createNotification(): Notification {
valpendingIntent = PendingIntent.getActivity(
        this,
        0,
        Intent(this, MainActivity::class.java),
        PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.FLAG_IMMUTABLE
    )
    return NotificationCompat.Builder(this, CHANNEL_ID)
        .setContentTitle(getString(R.string.app_name))
        .setContentText(getString(R.string.tracking_notification_content_des
c))
        .setSmallIcon(android.R.drawable.ic_menu_mylocation)
        .setContentIntent(pendingIntent)
        .setOngoing(true)
        .setPriority(NotificationCompat.PRIORITY_LOW)
        .build()
    }
    private fun startTracking() {
        try {
fusedLocationClient.requestLocationUpdates(
locationRequest,

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						116
Изм.	Лист	№докум.	Подпись	Дата		

```

locationCallback,
Looper.getMainLooper()
    )
    } catch (e: SecurityException) {
        // Permission not granted
stopSelf()
    }
}
private fun pauseTracking() {
fusedLocationClient.removeLocationUpdates(locationCallback)
}
private fun resumeTracking() {
startTracking()
}
private fun stopTracking() {
fusedLocationClient.removeLocationUpdates(locationCallback)
stopForeground(STOP_FOREGROUND_REMOVE)
stopSelf()
}
override fun onDestroy() {
super.onDestroy()
fusedLocationClient.removeLocationUpdates(locationCallback)
serviceScope.cancel()
}
companion object {
    const val CHANNEL_ID = RunSocialApp.TRACKING_CHANNEL_ID
    const val NOTIFICATION_ID = 1001

    const val ACTION_START = "ACTION_START"
    const val ACTION_PAUSE = "ACTION_PAUSE"
    const val ACTION_RESUME = "ACTION_RESUME"
    const val ACTION_STOP = "ACTION_STOP"
}
}

```

ФайлHistoryScreen.kt

```

package com.runsocial.app.presentation.screens.history
@OptIn(ExperimentalMaterial3Api::class)
@Composable
private fun RunHistoryItem(
    run: Run,
    onClick: () -> Unit,
    onDelete: () -> Unit
) {
Card(
    modifier = Modifier
.fillMaxWidth()
.clickable(onClick = onClick),
    shape = RoundedCornerShape(16.dp),
    colors = CardDefaults.cardColors(
containerColor = MaterialTheme.colorScheme.surfaceVariant.copy(alpha
= 0.5f)

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						117
Изм.	Лист	№докум.	Подпись	Дата		

```

        )
    ) {
Column(
            modifier = Modifier.padding(16.dp)
        ) {
Row(
            modifier = Modifier.fillMaxWidth(),
horizontalArrangement = Arrangement.SpaceBetween,
verticalAlignment = Alignment.CenterVertically
        ) {
Text(
            text =
run.startTime.format(DateTimeFormatter.ofPattern("dd MMMM yyyy,
HH:mm")),
            style = MaterialTheme.typography.labelMedium,
            color =
MaterialTheme.colorScheme.onSurfaceVariant
        )
IconButton(onClick = onDelete, modifier = Modifier.size(24.dp)) {
Icon(
Icons.Default.Delete,
contentDescription = stringResource(R.string.delete),
            tint = MaterialTheme.colorScheme.error,
            modifier = Modifier.size(20.dp)
        )
    }
}
Spacer(modifier = Modifier.height(8.dp))
Row(
            modifier = Modifier.fillMaxWidth(),
horizontalArrangement = Arrangement.SpaceBetween
        ) {
StatItem(
            icon = Icons.Default.Straighten,
            value = "%.2f".format(java.util.Locale.US,
run.distanceKm),
            unit = stringResource(R.string.km),
            label = stringResource(R.string.distance_label)
        )
StatItem(
            icon = Icons.Default.Timer,
            value = run.formattedDuration,
            unit = "",
            label = stringResource(R.string.time)
        )
        // Convert pace (s/km) to speed (km/h)
val speedKmh = if (run.avgPaceSecondsPerKm > 0) 3600.0 /
run.avgPaceSecondsPerKm else 0.0
StatItem(
            icon = Icons.Default.Speed,

```

```

        value = "%.1f".format(java.util.Locale.US,
speedKmh),
        unit = stringResource(R.string.km_h),
        label = stringResource(R.string.speed_label)
    )
    }
}
}
@Composable
private fun StatItem(
    icon: ImageVector,
    value: String,
    unit: String,
    label: String
) {
    Column(horizontalAlignment = Alignment.CenterHorizontally) {
        Row(verticalAlignment = Alignment.CenterVertically) {
            Icon(icon, null, modifier = Modifier.size(16.dp), tint = Primary)
            Spacer(modifier = Modifier.width(4.dp))
            Text(
                text = value,
                style = MaterialTheme.typography.titleMedium,
                fontWeight = FontWeight.Bold
            )
        }
    }
}

```

Файл AchievementsScreen.kt

```

package com.runsocial.app.presentation.screens.achievements
import com.runsocial.app.presentation.theme.*
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun AchievementsScreen(
    onNavigateBack: () -> Unit,
    viewModel: AchievementsViewModel = hiltViewModel()
) {
    val allDbAchievements by viewModel.achievements.collectAsState()
    // Dynamic data - only consider truly unlocked ones
    val allCatalogAchievements = AchievementsCatalog.ALL_ACHIEVEMENTS
    val unlockedTypes = allDbAchievements.filter{ it.isUnlocked }.map {
        it.type }.toSet()
    val progressMap = allDbAchievements.associate{ it.type to it.progress
    }
    val achievements = remember(unlockedTypes, progressMap) {
        allCatalogAchievements.map{ achievement ->
            achievement.copy(
                isUnlocked = unlockedTypes.contains(achievement.type),
                progress = progressMap[achievement.type] ?: 0f
            )
        }.sortedWith(compareByDescending<Achievement> { it.isUnlocked
        }).thenBy { it.tier })
    }
    val unlockedCount = unlockedTypes.size
}

```

```

val totalCount = achievements.size
Scaffold(
    topBar = {
        TopAppBar(
            title = {
                Text(stringResource(R.string.achievements_title)) },
            navigationIcon = {
                @Composable
                private fun AchievementCard(achievement: Achievement) {
                    val tierColor = when (achievement.tier) {
                        AchievementTier.BRONZE -> AchievementBronze
                        AchievementTier.SILVER -> AchievementSilver
                        AchievementTier.GOLD -> AchievementGold
                        AchievementTier.PLATINUM -> AchievementPlatinum
                    }

                    // Tier badge
                    Surface(
                        shape = RoundedCornerShape(8.dp),
                        color = tierColor.copy(alpha = 0.2f)
                    ) {
                        Text(
                            text =
                                stringResource(achievement.tier.resourceId),
                            style = MaterialTheme.typography.labelSmall,
                            color = tierColor,
                            modifier = Modifier.padding(horizontal = 8.dp,
                                vertical = 4.dp)
                        )
                    }
                }
            }
        )
    }
)

```

Серверная часть

Файл Dockerfile

```

# Build stage
FROM eclipse-temurin:21-jdk AS build
WORKDIR /app
# Copy the entire project
COPY . .
# Build the fat JAR for the backend
RUN ./gradlew :backend:shadowJar --no-daemon
# Run stage
FROM eclipse-temurin:21-jre
WORKDIR /app
# Copy the JAR from the build stage
COPY --from=build /app/backend/build/libs/backend-all.jar app.jar
# Expose the port (Ktor default 8080)
EXPOSE 8080
# Run the app
ENTRYPOINT ["java", "-jar", "app.jar"]

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						120
Изм.	Лист	№докум.	Подпись	Дата		

ФайлRuns.kt

```
@Serializable
data class LocationPointDto(
    val latitude: Double,
    val longitude: Double,
    val timestamp: Long, // Epoch millis
    val altitude: Double? = null,
    val speed: Float? = null
)
object Runs : Table() {
    val id = uuid("id")
    val userId = uuid("user_id").references(Users.id)
    val startTime = datetime("start_time")
    val endTime = datetime("end_time")
    val distance = double("distance_meters")
    val durationMillis = long("duration_millis").default(0L)
    val avgPace = double("avg_pace")
    val routeJson = text("route_json") // Store points as JSON string
    val meetingId = uuid("meeting_id").references(Meetings.id).nullable()
    val isDeleted = bool("is_deleted").default(false)
    override val primaryKey = PrimaryKey(id)
}
```

ФайлDatabaseFactory.kt

```
package com.runsocial.backend.data
import com.runsocial.backend.data.models.Users
import com.runsocial.backend.data.models.Runs
import com.runsocial.backend.data.models.Friendships
import io.ktor.server.config.*
import kotlinx.coroutines.Dispatchers
import org.jetbrains.exposed.sql.*
import
org.jetbrains.exposed.sql.transactions.experimental.newSuspendedTransaction
import org.jetbrains.exposed.sql.transactions.transaction
object DatabaseFactory {
    fun init(config: ApplicationConfig) {
        val driverClassName =
            config.property("storage.driverClassName").getString()
        val jdbcURL = config.property("storage.jdbcURL").getString()
        val user = config.property("storage.user").getString()
        val password = config.property("storage.password").getString()
        val database = Database.connect(jdbcURL, driverClassName, user,
            password)
        transaction(database) {
            SchemaUtils.createMissingTablesAndColumns(
                Users,
                Runs,
                com.runsocial.backend.data.models.Meetings,
                com.runsocial.backend.data.models.Participants,
                com.runsocial.backend.data.models.PasswordResetTokens,
```

					ПГУ 1.090301.39.001 ПЗ	Лист
						121
Изм.	Лист	№докум.	Подпись	Дата		

```

        Friendships,
com.runsocial.backend.data.models.MeetingInvitations
    )
    }
}
suspend fun <T>dbQuery(block: suspend () -> T): T =
newSuspendedTransaction(Dispatchers.IO) { block() }
}

```

ФайлRouting.kt

```

package com.runsocial.backend.data.models
import kotlinx.serialization.Serializable
import org.jetbrains.exposed.sql.Table
import org.jetbrains.exposed.sql.javatime.datetime
import java.time.LocalDateTime
@Serializable
data class User(
val id: String,
val username: String,
val email: String,
valavatarUrl: String? = null
)
object Users : Table() {
val id = uuid("id")
val username = varchar("username", 50).uniqueIndex()
val email = varchar("email", 255).uniqueIndex()
valavatarUrl = varchar("avatar_url", 512).nullable()
valpasswordHash = varchar("password_hash", 255)
valcreatedAt = datetime("created_at").clientDefault {
LocalDateTime.now() }
    override valprimaryKey = PrimaryKey(id)
}

```

ФайлSecurity.kt

```

package com.runsocial.backend.plugins
import io.ktor.server.application.*
import io.ktor.server.auth.*
import io.ktor.server.auth.jwt.*
fun Application.configureSecurity() {
    // TODO: Configure JWT Authentication
    authentication {
jwt("auth-jwt") {
    realm = "RunSocial Backend"
verifier(
        com.auth0.jwt.JWT
.require(com.auth0.jwt.algorithms.Algorithm.HMAC256("secret-key-
that-should-be-changed-in-production"))
.withAudience("runsocial-users")
.withIssuer("https://runsocial.app/")
.build()
    )
    validate { credential ->

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						122
Изм.	Лист	№докум.	Подпись	Дата		

```

        if (credential.payload.getClaim("userId").asString()
!= "") {
JWTPrincipal(credential.payload)
        } else {
            null
        }
    }
}
}
}

```

ФайлSockets.kt

```

package com.runsocial.backend.plugins
import io.ktor.server.application.*
import io.ktor.server.websocket.*
import java.time.Duration
fun Application.configureSockets() {
install(WebSockets) {
pingPeriod = Duration.ofSeconds(15)
    timeout = Duration.ofSeconds(15)
maxFrameSize = Long.MAX_VALUE
    masking = false
}
}

```

ФайлAuthRepository.kt

```

package com.runsocial.backend.repository
class AuthRepository(private val userDao: UserDao) {
    private fun hashPassword(password: String): String {
        return
Base64.getEncoder().encodeToString(password.toByteArray(Charsets.UTF
_8))
    }
    private fun checkPassword(password: String, hash: String):
Boolean {
        return hashPassword(password) == hash
    }
    suspend fun register(request: RegisterRequest):
Result<AuthResponse> {
valnormalizedEmail = request.email.trim().lowercase()
valnormalizedUsername = request.username.trim()
        if (!validateEmail(normalizedEmail)) {
            return Result.failure(Exception("Invalid email format"))
        }
        if (normalizedUsername.isBlank()) {
            return Result.failure(Exception("Username is required"))
        }
        if (userDao.findUserByEmail(normalizedEmail) != null) {
            return Result.failure(Exception("Email already in use"))
        }
        if (userDao.findUserByUsername(normalizedUsername) != null)
{

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						123
Изм.	Лист	№докум.	Подпись	Дата		

```

        return Result.failure(Exception("Username already
taken"))
    }
    val user = try {
        userDao.createUser(
            email = normalizedEmail,
            username = normalizedUsername,
            passwordHash = hashPassword(request.password)
        ) ?: return Result.failure(Exception("Failed to create user"))
    } catch (e: Exception) {
        val message = e.message?.lowercase().orEmpty()
        val registrationError = when {
            message.contains("users_email") || (message.contains("email")
            &&message.contains("duplicate")) ->
            Exception("Email already in use")
            message.contains("users_username") || (message.contains("username")
            &&message.contains("duplicate")) ->
            Exception("Username already taken")
            message.contains("duplicate key") || message.contains("unique
constraint") ->
            Exception("User already exists")
            else ->Exception("Registration failed")
        }
        return Result.failure(registrationError)
    }
    val token = generateToken(user.id)
    return Result.success(
        AuthResponse(
            token = token,
            userId = user.id,
            username = user.username,
            avatarUrl = user.avatarUrl
        )
    )
}
suspend fun login(request: LoginRequest): Result<AuthResponse> {
    if (!validateEmail(request.email)) {
        return Result.failure(Exception("Invalid email format"))
    }
    val user = userDao.findUserByEmail(request.email)
    ?: return Result.failure(Exception("User not found"))
    val storedHash = userDao.getPasswordHash(request.email)
    ?: return Result.failure(Exception("Password hash missing"))
    if (!checkPassword(request.password, storedHash)) {
        return Result.failure(Exception("Wrong password"))
    }
    val token = generateToken(user.id)
    return Result.success(
        AuthResponse(
            token = token,
            userId = user.id,

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						124
Изм.	Лист	№докум.	Подпись	Дата		

```

        username = user.username,
        avatarUrl = user.avatarUrl
    )
}

suspend fun changePassword(userId: String, old: String, new:
String): Result<Unit> {
    val storedHash = userDao.getPasswordHashById(userId) ?: return
    Result.failure(Exception("User not found"))
    if (!checkPassword(old, storedHash)) {
        return Result.failure(Exception("Wrong old password"))
    }
    if (checkPassword(new, storedHash)) {
        return Result.failure(Exception("New password cannot be
the same as the old one"))
    }
    val updated = userDao.updatePassword(userId, hashPassword(new))
    return if (updated) Result.success(Unit) else
    Result.failure(Exception("Update failed"))
}

suspend fun changePhoto(userId: String, avatarUrl: String?):
Result<Unit> {
    val normalizedAvatarUrl = avatarUrl?.trim()?.takeIf { it.isNotEmpty()
    }
    if (normalizedAvatarUrl != null &&
    !validatePhotoUrl(normalizedAvatarUrl)) {
        return Result.failure(Exception("Invalid photo URL"))
    }
    val updated = userDao.updateAvatar(userId, normalizedAvatarUrl)
    return if (updated) {
        Result.success(Unit)
    } else {
        Result.failure(Exception("User not found"))
    }
}

private fun generateToken(userId: String): String {
    return JWT.create()
    .withAudience("runsocial-users")
    .withIssuer("https://runsocial.app/")
    .withClaim("userId", userId)
    .withExpiresAt(Date(System.currentTimeMillis() + 60000 * 60 * 24))
    .sign(Algorithm.HMAC256("secret-key-that-should-be-changed-in-
production"))
}

suspend fun createPasswordResetToken(email: String):
Result<String> {
    val normalizedEmail = email.trim()
    if (!validateEmail(normalizedEmail)) {
        return Result.failure(Exception("Invalid email format"))
    }
    val user = userDao.findUserByEmail(normalizedEmail)

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						125
Изм.	Лист	№докум.	Подпись	Дата		

```

?: return Result.failure(Exception("User not found"))
val code = (100000..999999).random().toString()
val expiresAt = LocalDateTime.now().plusHours(1)
dbQuery {
    PasswordResetTokens.insert {
        it[PasswordResetTokens.userId] = UUID.fromString(user.id)
        it[PasswordResetTokens.token] = code
        it[PasswordResetTokens.expiresAt] = expiresAt
    }
}
return Result.success(code)
}

suspend fun verifyPasswordResetCode(email: String, code:
String): Result<Unit> {
    val user = userDao.findUserByEmail(email)
    ?: return Result.failure(Exception("User not found"))
    return dbQuery {
        val resetToken = PasswordResetTokens.selectAll()
        .where {
            (PasswordResetTokens.userId eq
            UUID.fromString(user.id)) and
            (PasswordResetTokens.token eq code) and
            (PasswordResetTokens.used eq false) and
            (PasswordResetTokens.expiresAt greater
            LocalDateTime.now())
        }
        .orderBy(PasswordResetTokens.createdAt, SortOrder.DESC)
        .limit(1)
        .singleOrNull()
        if (resetToken != null) {
            Result.success(Unit)
        } else {
            Result.failure(Exception("Invalid or expired code"))
        }
    }
}

suspend fun resetPassword(email: String, code: String,
newPassword: String): Result<Unit> {
    val user = userDao.findUserByEmail(email)
    ?: return Result.failure(Exception("User not found"))
    val storedHash = userDao.getPasswordHash(email)
    if (storedHash != null && checkPassword(newPassword,
storedHash)) {
        return Result.failure(Exception("New password cannot be
the same as the old one"))
    }
    return dbQuery {
        val resetToken = PasswordResetTokens.selectAll()
        .where {
            (PasswordResetTokens.userId eq
            UUID.fromString(user.id)) and

```

```

(PasswordResetTokens.token eq code) and
(PasswordResetTokens.used eq false) and
(PasswordResetTokens.expiresAt greater
LocalDateTime.now())
    }
.orderBy(PasswordResetTokens.createdAt, SortOrder.DESC)
.limit(1)
.singleOrNull()
?:return@dbQueryResult.failure<Unit>(Exception("Invalid or expired
code"))
val updated = userDao.updatePassword(user.id,
hashPassword(newPassword))
    if (!updated) {
return@dbQueryResult.failure<Unit>(Exception("Failed to update
password"))
    }
PasswordResetTokens.update({
(PasswordResetTokens.userId eq
UUID.fromString(user.id)) and
(PasswordResetTokens.token eq code)
    }) {
    it[used] = true
    }
Result.success(Unit)
    }
suspend fun updateEmail(userId: String, newEmail: String,
password: String): Result<Unit> {
    if (!validateEmail(newEmail)) {
        return Result.failure(Exception("Invalid email format"))
    }
valstoredHash = userDao.getPasswordHashById(userId) ?: return
Result.failure(Exception("User not found"))
    if (!checkPassword(password, storedHash)) {
        return Result.failure(Exception("Wrong password"))
    }
val updated = userDao.updateEmail(userId, newEmail)
    return if (updated) Result.success(Unit) else
Result.failure(Exception("Email already taken or update failed"))
    }
private fun validateEmail(email: String): Boolean {
    return email.matches(Regex("[A-Za-z0-9+_.-]+@[A-Za-z0-9.-
]+\.\.[A-Za-z]{2,}$"))
    }
private fun validatePhotoUrl(url: String): Boolean {
    return try {
valuri = URI(url)
val scheme = uri.scheme?.lowercase()
        (scheme == "http" || scheme == "https") &&
!uri.host.isNullOrEmpty()
    } catch (_: Exception) {

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						127
Изм.	Лист	№докум.	Подпись	Дата		

```

        false
    }
}

```

Файл Application.kt

```

package com.runsocial.backend
import com.runsocial.backend.data.DatabaseFactory
import com.runsocial.backend.data.dao.UserDao
import com.runsocial.backend.data.dao.RunDao
import com.runsocial.backend.data.dao.FriendDao
import com.runsocial.backend.plugins.*
fun Application.module() {
    DatabaseFactory.init(environment.config)
    val userDao = UserDao()
    val runDao = RunDao()
    val friendDao = FriendDao()
    val authRepository = AuthRepository(userDao)
    val sessionManager = RunSessionManager(userDao)
    val emailService =
        com.runsocial.backend.service.EmailService(environment.config)
    configureSerialization()
    configureSecurity()
    configureSockets()
    configureRouting(authRepository, runDao, sessionManager,
        emailService, friendDao, userDao)
}

```

Файл docker-compose.yml

```

services:
  postgres:
    image: postgres:15-alpine
    container_name: runsocial-db
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
      POSTGRES_DB: runsocial
    ports:
      - "5433:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    restart: unless-stopped
  pgadmin:
    image: dpage/pgadmin4
    container_name: runsocial-pgadmin
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@runsocial.app
      PGADMIN_DEFAULT_PASSWORD: admin
    ports:
      - "5050:80"
  depends_on:
    - postgres

```

					ПГУ 1.090301.39.001 ПЗ	Лист
						128
Изм.	Лист	№докум.	Подпись	Дата		

```

    restart: unless-stopped
  backend:
    build:
      context: .
  dockerfile: backend/Dockerfile
  container_name: runsocial-backend
  ports:
    - "8080:8080"
  environment:
    - DB_URL=jdbc:postgresql://postgres:5432/runsocial
    - DB_USER=postgres
    - DB_PASSWORD=password
    - JWT_AUDIENCE=scala-run-users
    - JWT_REALM=Scala run Backend
    - EMAIL_ENABLED=true
  depends_on:
    - postgres
    restart: unless-stopped
  volumes:
  postgres_data:

```

Приложение В

(обязательное)

Android-приложение для совместных беговых тренировок

Презентация

					ПГУ 1.090301.39.001 ПЗ	Лист
						130
Изм.	Лист	№докум.	Подпись	Дата		

Тема ВКР: Android-приложение для совместных беговых тренировок

Автор работы:

студент гр.22ВВП2 Кузьмин Д.В.

Научный руководитель:

доцент, каф. ВТ ПГУ Кузнецов А.В.

Пенза 2026

Цель работы

Основной целью данной работы является разработать Android-приложение для совместных беговых тренировок, предоставляющее пользователям инструменты для отслеживания беговых тренировок, анализа статистики, организации совместных беговых тренировок с друзьями, соревнования со своими прошлыми результатами и редактирование профиля. Также приложение должно позволять осуществлять регистрацию и авторизацию пользователя.

Актуальность работы

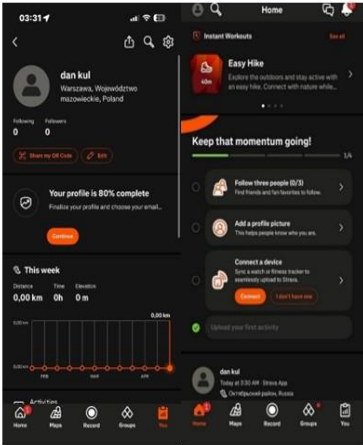
В условиях растущего интереса к здоровому образу жизни особую востребованность приобретают мобильные приложения для отслеживания физической активности. Android-приложения для бега позволяют пользователям отслеживать тренировки, анализировать результаты и контролировать личный прогресс. Благодаря сочетанию функциональности, удобства использования и современных технологий такие решения помогают более эффективно организовывать тренировочный процесс. Разработка Android-приложения для совместных беговых тренировок предоставляет дополнительные возможности взаимодействия между пользователями и способствует повышению мотивации к регулярным занятиям спортом.

Постановка задачи

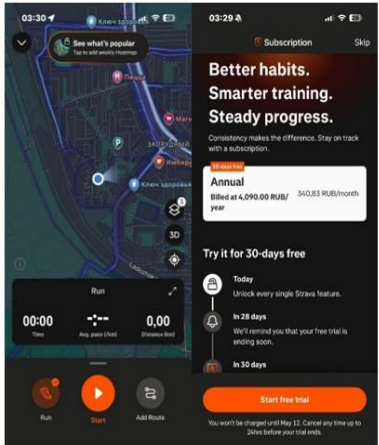
Приложение должно соответствовать следующим требованиям:

- обеспечивать регистрацию и авторизацию пользователей;
- выполнять GPS-трекинг маршрута с определением дистанции, времени тренировки и темпа бега;
- сохранять историю тренировок и предоставлять доступ к статистике спортивной активности;
- поддерживать систему достижений для повышения мотивации пользователей;
- предоставлять возможность добавления друзей и взаимодействия между участниками приложения;
- реализовывать функцию совместных беговых тренировок;
- реализовать функцию позволяющую соревноваться с собственными результатами в режиме реального времени;
- отправлять push-уведомления пользователям при обнаружении друга в радиусе 100 метров с предложением начать совместную тренировку;
- обладать современным, интуитивно понятным интерфейсом и обеспечивать удобную навигацию по приложению для подрастающего поколения.

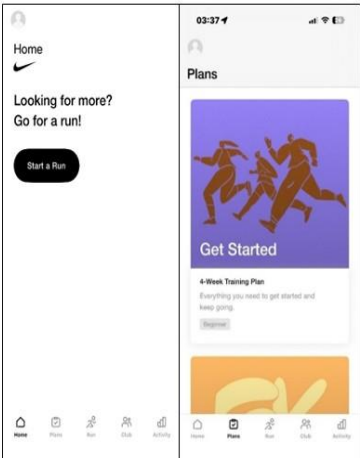
Аналоги



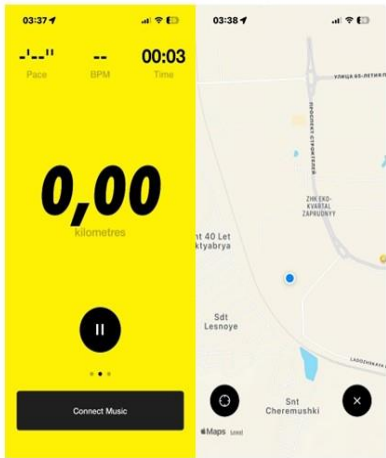
«Strava»



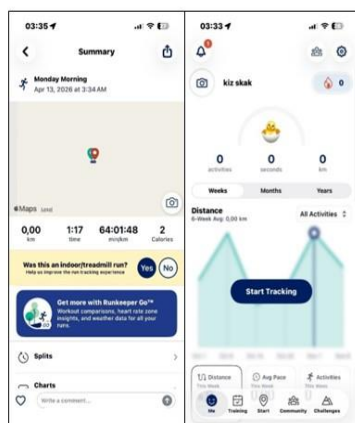
Аналоги



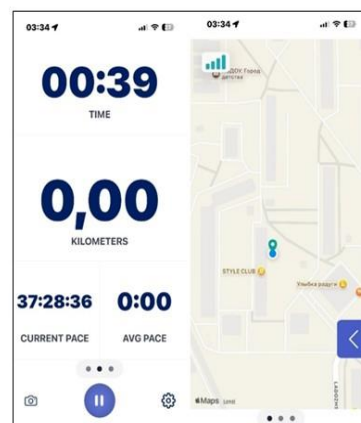
«Nike Run Club»



Аналоги



«Runkeeper»



Стек технологий



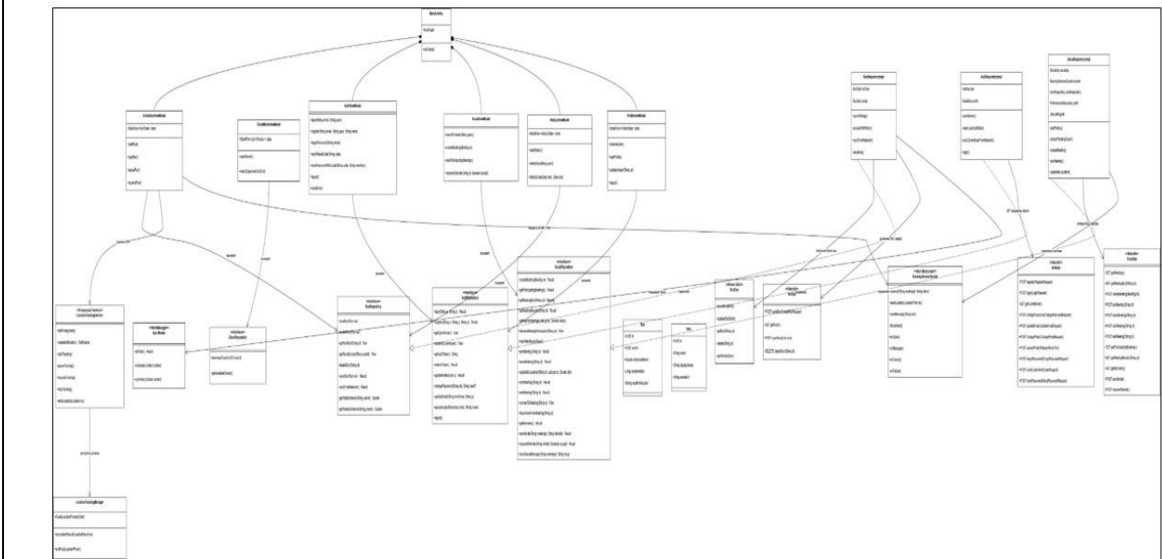
Изм.	Лист	№докум.	Подпись	Дата

ПГУ 1.090301.39.001 ПЗ

Лист

134

UML-диаграмма классов



Демонстрация работы приложения



Заключение

В ходе выполнения данной выпускной квалификационной работы было разработано Android-приложение для совместных беговых тренировок, предназначенное для отслеживания физической активности пользователей, анализа результатов тренировок и организации совместных пробежек.

В дальнейшем развитие приложения будет направлено на загрузку в Rustore, интеграцию с фитнес-браслетами и умными часами, добавление чатов для общения, добавление AI-ассистента для эффективных тренировок и улучшение игровых механик, что позволит повысить вовлеченность пользователей и эффективность тренировочного процесса.

СПАСИБО ЗА ВНИМАНИЕ