

МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”  
(ФГБОУ ВО «ВГУ»)

Факультет компьютерных наук

Кафедра программирования и информационных технологий

Мобильное приложение для отслеживания и формирования полезных  
привычек

ВКР Бакалаврская работа

09.03.04 Программная инженерия

Профиль «Информационные системы и сетевые технологии»

Допущено к защите в ГЭК

Зав. кафедрой  С.Д. Махортов, д.ф.-м.н., доцент \_\_.\_\_.20\_\_

Обучающийся  М.О.З. Тавфик, 4 курс, д/о

Руководитель  А.А. Вахтин, к.ф.-м.н., доцент

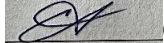
Воронеж 2025

МИНОБРНАУКИ РОССИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»  
(ФГБОУ ВО «ВГУ»)

Факультет компьютерных наук

Кафедра программирования и информационных технологий

УТВЕРЖДАЮ  
заведующий кафедрой  
программирования и  
информационных технологий

 С. Д. Махортов  
\_\_\_\_.\_\_\_\_.2025

**ЗАДАНИЕ**  
**НА ВЫПОЛНЕНИЕ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ**  
**ОБУЧАЮЩЕГОСЯ** Тавфика Мартена Осамы Захьяна

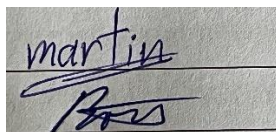
1. Тема работы «Мобильное приложение для отслеживания и формирования полезных привычек», утверждена решением ученого совета факультета компьютерных наук от 30.11.2024
2. Направление подготовки: 09.03.04 Программная инженерия
3. Срок сдачи законченной работы: 09.06.2025
4. Календарный план (строится в соответствии со структурой ВКР):

№	Структура ВКР	Сроки выполнения	Примечание
1	ВВЕДЕНИЕ	05.12.2024 – 10.12.2024	
2	1. Постановка задачи	12.12.2024 – 25.12.2024	
3	1.1. Требования к разрабатываемой системе	26.12.2024 – 29.12.2024	
4	1.1.1. Функциональные требования	09.01.2025 – 15.01.2025	
5	1.1.1.1. Для пользователей	15.01.2025 – 16.01.2025	
6	1.1.1.2. Для администратора	17.01.2025 – 18.01.2025	
7	1.1.2. Технические требования	19.01.2025 – 20.01.2025	
8	1.1.3. Нефункциональные требования	21.01.2025 – 22.01.2025	
9	1.2. Требования к интерфейсу	23.01.2025 – 24.01.2025	
10	1.3. Задачи, решаемые в процессе разработки	25.01.2025 – 26.01.2025	
11	2. Анализ предметной области	27.01.2025 – 28.01.2025	
12	2.1. Терминология предметной области	29.01.2025 – 30.01.2025	
13	2.2. Обзор аналогов	31.01.2025 – 01.02.2025	
14	2.2.1. Habitica	02.02.2025 – 03.02.2025	

15	2.2.2. Streaks	04.02.2025 – 05.02.2025	
16	2.2.3. Fabulous	06.02.2025 – 07.02.2025	
17	2.2.4. Loop Habit Tracker	08.02.2025 – 09.02.2025	
18	2.3. Моделирование системы	10.02.2025 – 11.02.2025	
19	2.3.1. Диаграмма вариантов использования (Use Case Diagram)	12.02.2025 – 13.02.2025	
20	2.3.2. Диаграмма деятельности (Диаграмма активности)	14.02.2025 – 15.02.2025	
21	2.3.3. Диаграмма классов (Class Diagram)	16.02.2025 – 17.02.2025	
22	2.3.4. Диаграмма ER (Entity-Relationship Diagram)	18.02.2025 – 19.02.2025	
23	2.3.5. Диаграмма последовательности (Sequence Diagram)	20.02.2025 – 21.02.2025	
24	3. Реализация	22.02.2025 – 23.02.2025	
25	3.1. Средства реализации	24.02.2025 – 25.02.2025	
26	3.2. Этапы реализации и интеграции	26.02.2025 – 27.02.2025	
27	3.3. Реализация интерфейса	28.02.2025 – 01.03.2025	
28	3.3.1. Страница входа (Sign In Page)	02.03.2025 – 03.03.2025	
29	3.3.2. Страница регистрации (Sign Up Page)	04.03.2025 – 05.03.2025	
30	3.3.3. Главная страница (Main Page)	06.03.2025 – 07.03.2025	
31	3.3.4. Страница профиля пользователя (User Profile Page)	08.03.2025 – 09.03.2025	
32	3.3.5. Страница статистики (Statistics Page)	10.03.2025 – 11.03.2025	
33	3.3.6. Плавающее окно для добавления новой привычки (Floating Window for Adding a New Habit)	12.03.2025 – 13.03.2025	
34	3.3.7. Добавление привычки (Habit Added)	14.03.2025 – 15.03.2025	
35	3.3.8. Редактирование или удаление привычки (Edit or Delete Habit)	16.03.2025 – 17.03.2025	
36	3.3.9. Редактирование привычки (Edit Habit)	18.03.2025 – 19.03.2025	
37	4. Тестирование	20.03.2025 – 21.03.2025	
38	4.1. Тестирование серверной части	22.03.2025 – 23.03.2025	
39	4.1.1. Пример тестирования: Логин	24.03.2025 – 25.03.2025	
40	4.2. Тестирование пользовательского интерфейса	26.03.2025 – 27.03.2025	
41	4.2.1. Пример тестирования: Главная страница	28.03.2025 – 29.03.2025	
42	ЗАКЛЮЧЕНИЕ	30.03.2025 – 07.04.2025	

Обучающийся

Руководитель



Тавфик М.О.З.

Вахтин А.А.

## РЕФЕРАТ

Бакалаврская работа 84 с., 18 рис., 16 использованных источников.

MOBILE APPLICATION, KOTLIN, HABIT TRACKING, ANDROID, PERSONAL DEVELOPMENT, JETPACK COMPOSE, MATERIAL DESIGN, FIREBASE AUTHENTICATION, RETROFIT, POSTGRESQL, APPWRITE STORAGE.

Объект исследования – мобильное приложение для отслеживания привычек для платформы Android, предназначенное для создания, редактирования, удаления и анализа привычек с помощью визуализации прогресса.

Цель исследования – разработать Android-приложение для отслеживания привычек с возможностью их создания, отметки выполнения, установки напоминаний и просмотра статистики.

Результаты работы: разработано клиент-серверное мобильное приложение с использованием Kotlin, Jetpack Compose, Firebase Authentication, Retrofit, PostgreSQL, Appwrite Storage; созданы UML-диаграммы (варианты использования, классов, последовательности, деятельности, ER); проведено тестирование серверной части и пользовательского интерфейса.

Область применения результатов: приложение может быть использовано широким кругом пользователей для повышения личной продуктивности и формирования полезных привычек.

## Содержание

<b>ВВЕДЕНИЕ .....</b>	<b>7</b>
<b>1. Постановка задачи.....</b>	<b>9</b>
<b>1.1. Требования к разрабатываемой системе.....</b>	<b>11</b>
<b>1.1.1. Функциональные требования .....</b>	<b>11</b>
<b>1.1.1.1. Для пользователей.....</b>	<b>11</b>
<b>1.1.1.2. Для администратора .....</b>	<b>12</b>
<b>1.1.2. Технические требования .....</b>	<b>13</b>
<b>1.1.3. Нефункциональные требования .....</b>	<b>15</b>
<b>1.2. Требования к интерфейсу.....</b>	<b>18</b>
<b>1.3. Задачи, решаемые в процессе разработки .....</b>	<b>21</b>
<b>2. Анализ предметной области .....</b>	<b>24</b>
<b>2.1. Терминология (гlossарий предметной области).....</b>	<b>24</b>
<b>2.2. Обзор аналогов.....</b>	<b>26</b>
<b>2.2.1. Habitica.....</b>	<b>27</b>
<b>2.2.2. Streaks .....</b>	<b>28</b>
<b>2.2.3. Fabulous.....</b>	<b>29</b>
<b>2.2.4. Loop Habit Tracker.....</b>	<b>31</b>
<b>2.3. Моделирование системы .....</b>	<b>33</b>
<b>2.3.1. Диаграмма вариантов использования (Use Case Diagram).....</b>	<b>33</b>
<b>2.3.2. Диаграмма деятельности (Диаграмма активности) .....</b>	<b>36</b>
<b>2.3.3. Диаграмма классов (Class Diagram) .....</b>	<b>40</b>
<b>2.3.4. Диаграмма ER (Entity-Relationship Diagram).....</b>	<b>44</b>
<b>2.3.5. Диаграмма последовательности (Sequence Diagram).....</b>	<b>47</b>
<b>3. Реализация .....</b>	<b>50</b>
<b>3.1. Средства реализации .....</b>	<b>50</b>
<b>3.2. Этапы реализации и интеграции .....</b>	<b>51</b>
<b>3.3. Реализация интерфейса .....</b>	<b>54</b>

3.3.1. Страница входа (Sign In Page) .....	56
3.3.2. Страница регистрации (Sign Up Page) .....	59
3.3.3. Главная страница (Main Page).....	62
3.3.4. Страница профиля пользователя (User Profile Page).....	64
3.3.5. Страница статистики (Statistics Page).....	67
3.3.6. Плавающее окно для добавления новой привычки (Floating Window for Adding a New Habit) .....	70
3.3.7. Добавление привычки (Habit Added).....	73
3.3.8. Редактирование или удаление привычки (Edit or Delete Habit)	76
3.3.9. Редактирование привычки (Edit Habit).....	79
4. Тестирование .....	82
4.1. Тестирование серверной части .....	82
4.1.1. Пример тестирования: Логин .....	82
4.2. Тестирование пользовательского интерфейса .....	83
4.2.1. Пример тестирования: Главная страница.....	84
ЗАКЛЮЧЕНИЕ.....	86
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ .....	87
ПРИЛОЖЕНИЕ А .....	90
ПРИЛОЖЕНИЕ Б.....	94

## ВВЕДЕНИЕ

Бакалаврская работа посвящена разработке мобильного приложения для отслеживания привычек на платформе Android. Актуальность темы обусловлена возрастающим интересом к саморазвитию и продуктивности в современном обществе. Многие люди стремятся формировать полезные привычки — такие как регулярные физические упражнения, чтение книг, медитация, соблюдение режима сна и здорового питания. Для эффективного контроля за выполнением подобных задач всё чаще используются специализированные мобильные приложения.

Разработка программного обеспечения такого типа позволяет не только помочь пользователям достигать личных целей, но и предоставляет возможность гибко адаптироваться под индивидуальные потребности. Предлагаемое приложение реализовано с использованием современных технологий: языка программирования Kotlin, Jetpack Compose для создания пользовательского интерфейса, Firebase для аутентификации и хранения данных, а также Retrofit и Appwrite для взаимодействия с внешними API.

Целью данной работы является проектирование и разработка удобного и функционального Android-приложения для отслеживания привычек, позволяющего пользователям создавать, редактировать, удалять привычки, отмечать их выполнение, просматривать статистику и получать напоминания о предстоящих действиях.

Для достижения указанной цели были поставлены следующие задачи:

- Провести анализ существующих решений в области отслеживания привычек.

- Спроектировать архитектуру приложения и определить используемые технологии.
- Разработать пользовательский интерфейс, соответствующий принципам Material Design.
- Реализовать основные функциональные модули приложения: регистрация и авторизация, управление привычками, система уведомлений, отображение статистики.
- Протестировать работоспособность и надёжность всех компонентов приложения.
- Оформить результаты разработки в виде курсовой работы.

Объектом исследования являются процессы управления привычками и методы их автоматизации с помощью мобильных приложений. Предметом исследования выступает программный продукт – Android-приложение для отслеживания привычек.

Научная новизна заключается в комплексном подходе к разработке приложения, сочетающего современные средства клиент-серверного взаимодействия, облачное хранение данных и адаптивный пользовательский интерфейс.

Практическая значимость работы состоит в том, что разработанное приложение может быть использовано широким кругом пользователей для повышения личной продуктивности и формирования полезных привычек.



## 1. Постановка задачи

Разработка мобильного приложения для отслеживания привычек направлена на создание программного продукта, который поможет пользователям формировать полезные привычки и отказываться от вредных через регулярное выполнение определённых действий. Приложение должно предоставлять удобный интерфейс и функциональность, позволяющую эффективно управлять привычками.

Цель данной работы — разработать Android-приложение, предназначенное для отслеживания личных привычек с возможностью их создания, редактирования, удаления, установки напоминаний и анализа прогресса. Основной задачей является проектирование и реализация клиент-серверного приложения, обеспечивающего работу с данными пользователя и сохранение информации в облачном хранилище.

Объектом разработки выступает система управления личными привычками, ориентированная на широкого пользователя, заинтересованного в саморазвитии и повышении личной продуктивности. Предметом исследования являются методы и технологии разработки мобильных приложений, а также способы взаимодействия клиента с сервером с использованием современных инструментов и библиотек.

Для достижения поставленной цели были сформулированы следующие задачи:

- Провести анализ существующих решений в области отслеживания привычек и определить их основные достоинства и недостатки.

- Спроектировать структуру базы данных и архитектуру приложения.
- Разработать пользовательский интерфейс, соответствующий принципам Material Design и обеспечивающий высокую степень юзабилити.
- Реализовать модуль регистрации и авторизации пользователей с использованием Firebase Authentication и Google Sign-In.
- Создать функционал добавления, редактирования и удаления привычек.
- Интегрировать систему уведомлений и напоминаний о предстоящих действиях.
- Реализовать возможность просмотра статистики выполнения привычек с помощью визуализации (например, диаграммы и графики).
- Обеспечить хранение данных пользователя в облачной системе (Firebase Firestore и Appwrite Storage).
- Протестировать работоспособность приложения, провести анализ его функционирования и выявить возможные ошибки.
- Подготовить документацию к проекту, включая описание архитектуры, используемых технологий и процесса тестирования.

Поставленные задачи позволят создать полноценное мобильное приложение, которое будет отвечать современным требованиям к программным продуктам данного типа и сможет быть использовано конечными пользователями для эффективного управления своими привычками.

## **1.1. Требования к разрабатываемой системе**

Разрабатываемая система представляет собой мобильное приложение для отслеживания привычек, ориентированное на платформу Android. Приложение предназначено для помощи пользователям в формировании полезных привычек и отказе от вредных через регулярное выполнение определённых действий. Для обеспечения высокого уровня функциональности и удобства использования были сформулированы следующие требования к системе.

### **1.1.1. Функциональные требования**

Функциональные требования определяют основные задачи, которые должна выполнять система, и разделены на требования, относящиеся к обычным пользователям и администратору (при необходимости).

#### **1.1.1.1. Для пользователей**

Пользователь — это физическое лицо, зарегистрировавшееся в системе и использующее приложение для управления своими привычками. Основные функциональные возможности для пользователя включают:

- **Регистрация и авторизация**

Пользователь должен иметь возможность зарегистрироваться в системе и войти в учетную запись. Авторизация реализуется через email/пароль и интеграцию с Google Sign-In.

- **Создание, редактирование и удаление привычек**

Пользователь может создавать новые привычки, задавать им параметры (название, описание, частота выполнения, время напоминания и др.), а также редактировать или удалять уже созданные привычки.

- Отметка выполнения привычки

Пользователь может ежедневно отмечать выполнение привычки, что позволяет формировать статистику прогресса.

- Установка напоминаний

Система предоставляет возможность настройки напоминаний о предстоящем выполнении привычки с использованием системного AlarmManager и NotificationManager.

- Визуализация статистики

Реализована возможность просмотра статистики выполнения привычек в виде графиков и диаграмм (в частности, используется библиотека MPAndroidChart для построения круговых диаграмм).

- Персонализация профиля

Пользователь может просматривать и редактировать свой профиль, включая имя, аватар, дату рождения и другие данные.

- Навигация между экранами

Предоставлена навигационная панель с доступом ко всем основным экранам: главная страница, статистика, профиль.

#### **1.1.1.2. Для администратора**

В рамках разрабатываемого приложения роль администратора не была реализована, поскольку приложение ориентировано на индивидуальное использование каждым пользователем. Все операции производятся в рамках личного аккаунта, и нет необходимости в централизованном управлении данными всех пользователей.

Однако, в случае масштабирования системы и перевода её в корпоративную или общественную модель использования, возможно добавление роли администратора с возможностью:

- Просмотра обобщённой статистики,
- Модерации пользовательского контента,
- Управления тарифными планами (если будет платная подписка),
- Мониторинга активности пользователей.

Таким образом, архитектура приложения спроектирована таким образом, чтобы в дальнейшем можно было легко добавить функционал администрирования.

### **1.1.2. Технические требования**

Приложение "Habit Tracker" представляет собой мобильное приложение для отслеживания привычек, разработанное с использованием современных технологий и подходов к программной инженерии. Оно предназначено для помощи пользователям в формировании полезных привычек и отказе от вредных через регулярное выполнение определённых действий.

Технические требования определяют параметры работы программной системы, включая используемые технологии, платформенные ограничения и программные зависимости:

- Платформа: Android (API уровня 24 и выше)
- Язык программирования: Kotlin
- Фреймворк пользовательского интерфейса: Jetpack Compose
- Дизайн-система: Material Design 3 — для обеспечения современного и согласованного внешнего вида приложения

- Серверная часть: реализована на базе Ktor (Kotlin backend framework), взаимодействующего с базой данных PostgreSQL через ORM Exposed
- Система управления базами данных: PostgreSQL версии 15
- Контейнеризация: Docker — для удобства запуска и масштабирования серверной части
- Взаимодействие между клиентом и сервером: REST API с обменом данными в формате JSON
- Авторизация и аутентификация: Firebase Authentication — для безопасного входа пользователей через email/пароль и Google Sign-In
- Работа с сетью: Retrofit — для выполнения HTTP-запросов к собственному серверу
- Хранение файлов: Appwrite Storage — для загрузки и хранения изображений, таких как аватары пользователей
- Локальное хранение данных: DataStore — для сохранения настроек и временных данных на устройстве пользователя
- Библиотеки и SDK:
  - Coil — для асинхронной загрузки и кэширования изображений
  - MPAndroidChart — для визуализации статистики привычек
  - Hilt — для внедрения зависимостей на Android
  - Coroutines и Flow — для асинхронной работы и реактивного программирования
  - Lottie — для анимаций в интерфейсе
  - ThreetenABP — для работы с датами и временем
- Интеграция с системными API: AlarmManager и NotificationManager — для реализации напоминаний о выполнении привычек

- Поддержка тем оформления: Light и Dark Mode — для улучшения пользовательского опыта в разных условиях освещения
- Локализация: минимальная поддержка английского и русского языков — для расширения целевой аудитории

Эти технические требования обеспечивают надёжность, производительность и масштабируемость приложения, а также соответствуют современным стандартам разработки мобильных приложений под Android.

### **1.1.3. Нефункциональные требования**

Нефункциональные требования описывают характеристики качества программного обеспечения, которые не связаны напрямую с его функциональностью, но оказывают существенное влияние на удобство использования, надёжность и дальнейшее развитие системы. Данные требования были учтены при разработке приложения "Habit Tracker".

- Производительность

Приложение должно быть отзывчивым и обеспечивать минимальную задержку между действиями пользователя и реакцией интерфейса. Все операции, связанные с сетью (запросы к серверу через Retrofit), а также локальные вычисления (например, отрисовка графиков в MPAndroidChart) выполняются асинхронно с использованием Kotlin Coroutines и Flow для предотвращения блокировки основного потока.

Все данные кэшируются локально с помощью DataStore, что позволяет ускорить загрузку информации при повторном запуске приложения. Использование библиотеки Coil для загрузки изображений также способствует

повышению производительности за счёт кэширования и оптимизации работы с медиафайлами.

- Надежность

Для обеспечения стабильной работы приложения реализованы механизмы обработки ошибок на всех уровнях: от получения данных с сервера до взаимодействия с локальным хранилищем. Все сетевые запросы оборачиваются в try-catch блоки, используются безопасные вызовы suspend-функций, а также применяется паттерн Result для унифицированного представления успешных и неуспешных результатов.

Система уведомлений реализована через AlarmManager и NotificationManager, что гарантирует своевременное выполнение задач даже после перезагрузки устройства. Приложение корректно сохраняет состояние экранов и активностей, что позволяет избежать потери данных при изменении ориентации или фоновых процессах.

- Безопасность

Безопасность данных является одним из ключевых аспектов. Для авторизации пользователей используется Firebase Authentication, поддерживающий вход через email/пароль и Google Sign-In. Все передаваемые данные шифруются по протоколу HTTPS, что обеспечивается настройкой networkSecurityConfig в Android-приложении.

Локальное хранение токенов и других конфиденциальных данных осуществляется с помощью безопасных методов, таких как EncryptedSharedPreferences. Пользовательские изображения загружаются



через Appwrite Storage с ограниченным доступом, что исключает возможность просмотра данных другими пользователями.

- Юзабилити (удобство использования)

Интерфейс приложения разработан в соответствии с принципами Material Design 3, что обеспечивает современный и интуитивно понятный внешний вид. Все элементы управления расположены логично, с учетом стандартов Android UX. Используются понятные иконки, последовательная навигация и четкие текстовые подсказки.

Для улучшения восприятия статистики выполнения привычек применяется визуализация с использованием библиотеки MPAndroidChart, где пользователь может наглядно оценить свои успехи и слабые места. Интерфейс адаптирован под разные размеры экранов и ориентации, а также поддерживает Light и Dark режимы отображения.

- Поддерживаемость

Архитектура приложения спроектирована по принципам MVVM (Model-View-ViewModel) с чётким разделением ответственности между слоями. Логика UI отделена от бизнес-логики, что облегчает модификацию и тестирование компонентов.

Взаимодействие с сервером вынесено в Repository-слой, что позволяет легко менять источник данных без изменения пользовательского интерфейса. Также используется внедрение зависимостей через Hilt, что делает код более гибким и тестируемым.

- Совместимость

Приложение разработано с поддержкой Android API 24 и выше, что охватывает большинство современных устройств. Вёрстка экранов выполнена с использованием Jetpack Compose, обеспечивающего гибкое и адаптивное отображение контента.

Использование ConstraintLayout, Column, Row и Box в Jetpack Compose позволяет эффективно использовать пространство на экранах различной диагонали и плотности пикселей. Все цвета, размеры и ресурсы вынесены в отдельные файлы ресурсов (colors.xml, dimens.xml), что упрощает их переопределение для разных конфигураций.

- **Расширяемость**

Проект разрабатывался с учётом возможности масштабирования и добавления новых функций. Например, в будущем возможно:

- Добавление роли администратора,
- Интеграция с облачным сервером Ktor на базе PostgreSQL,
- Реализация платформенной подписки,
- Синхронизация с другими сервисами (Google Calendar, Apple Health и др.).

Также предусмотрена возможность добавления локализации на другие языки с помощью строковых ресурсов, уже подготовленных в проекте.

## **1.2. Требования к интерфейсу**

Интерфейс приложения "Habit Tracker" разрабатывался с учетом принципов Material Design 3, что обеспечивает современный и визуально привлекательный внешний вид, а также интуитивно понятное взаимодействие

с пользователем. При проектировании пользовательского интерфейса были учтены следующие ключевые требования:

- Простота и интуитивность

Интерфейс должен быть простым и понятным даже для новых пользователей. Все элементы управления расположены логично, без излишней перегруженности. Навигация между экранами реализована через нижнюю панель навигации (Bottom Navigation), обеспечивающую быстрое перемещение между основными разделами: главная страница, статистика, профиль.

- Адаптивность

Приложение должно корректно отображаться на устройствах с различными размерами экранов и плотностью пикселей. Для этого используется Jetpack Compose, позволяющий гибко управлять расположением элементов и масштабированием контента. Интерфейс поддерживает как портретную, так и альбомную ориентацию экрана.

- Контрастность и читаемость

Все текстовые элементы имеют достаточный размер шрифта и высокую контрастность по отношению к фону. Цветовая схема соответствует рекомендациям Material Design 3 и поддерживает темный и светлый режимы отображения. Это улучшает восприятие информации и снижает нагрузку на зрение пользователя.

- Поддержка обратной связи

Каждое действие пользователя сопровождается визуальной или звуковой обратной связью (например, анимации при нажатии кнопок, всплывающие сообщения Toast, Snackbar). Это помогает пользователю понимать текущее состояние приложения и результат выполненного действия.

- **Согласованность**

Элементы интерфейса имеют единый стиль оформления, что повышает удобство использования. Иконки, кнопки, диалоговые окна и формы используют однотипные стили и анимации. Для создания согласованного UX-опыта применяются стандартные компоненты Material 3, такие как Button, Card, TextField, Dialog и др.

- **Доступность**

Интерфейс спроектирован с учетом требований доступности (Accessibility). Все элементы имеют описания для TalkBack (Android-считыватель с экрана), а цвета проверены на соответствие стандартам WCAG. Это позволяет использовать приложение людям с ограниченными возможностями зрения.

- **Эффективное использование экрана**

Для экономии пространства и упрощения восприятия используется карточный интерфейс. Информация о привычках отображается в виде плиток, которые можно легко отметить как выполненные или редактировать. Графики и диаграммы (реализованные с помощью библиотеки MPAndroidChart) предоставляют наглядное представление прогресса за день, неделю или месяц.

- **Уведомления и напоминания**

Система уведомлений реализована через AlarmManager и NotificationManager. Пользователь получает уведомления о предстоящих привычках в указанное время. Уведомления содержат четкие заголовки и действия (например, «Отметить как выполненное» или «Отложить»).

- Локализация

Приложение частично поддерживает локализацию на нескольких языках. В проекте подготовлены строковые ресурсы для английского и русского языков, что позволяет расширять целевую аудиторию и адаптировать приложение под разные регионы.

- Анимации и переходы

Интерфейс содержит плавные анимации при переходах между экранами, изменении состояния привычки, показе диалоговых окон. Анимации реализованы с использованием стандартных механизмов Jetpack Compose (AnimatedVisibility, animateFloatAsState, slideInHorizontally, fadeIn и т.п.), что делает взаимодействие более приятным и естественным.

### **1.3. Задачи, решаемые в процессе разработки**

В ходе реализации проекта «Habit Tracker» были сформулированы и последовательно решены следующие ключевые задачи, направленные на создание функционального и устойчивого мобильного приложения для отслеживания привычек.

- Исследование и анализ предметной области

Первым этапом стало изучение существующих решений в области отслеживания привычек и управления личными целями. Были проанализированы популярные приложения, такие как Habitica , Streaks ,

Fabulous и другие, с целью выявления их основных возможностей, недостатков и особенностей пользовательского интерфейса. Это позволило определить уникальные черты и преимущества проектируемого приложения.

- Определение требований к системе

На основе проведённого анализа были сформулированы функциональные и нефункциональные требования к приложению. Учитывались как потребности конечного пользователя (создание, редактирование и удаление привычек, напоминания, статистика), так и технические аспекты (использование Jetpack Compose, Firebase Authentication, Retrofit, Appwrite Storage и т.д.).

- Проектирование архитектуры приложения

Для обеспечения масштабируемости и поддерживаемости была выбрана архитектура MVVM (Model-View-ViewModel). В рамках этой архитектуры:

- Model : работа с данными через Firebase Firestore и Retrofit.
- ViewModel : управление состоянием экранов и обработка бизнес-логики.
- View : реализация UI с помощью Jetpack Compose и Material Design 3.

Также был спроектирован Repository-слой, обеспечивающий абстракцию между источниками данных и UI.

- Разработка пользовательского интерфейса

Разработка UI проводилась с использованием Jetpack Compose , что позволило создать современный и отзывчивый интерфейс. Для повышения юзабилити применялись:

- Анимации и переходы (AnimatedVisibility, animateFloatAsState)
- Темная и светлая темы оформления

- Поддержка различных размеров экранов
- Использование библиотеки MPAndroidChart для визуализации статистики
- Реализация авторизации и регистрации

Для входа в приложение была интегрирована система Firebase Authentication , поддерживающая:

- Авторизацию через email/пароль
- Вход через Google Sign-In

Это позволило обеспечить безопасное хранение учетных данных и предоставить удобный способ аутентификации.

- Интеграция с серверной частью

Для работы с данными на удаленном сервере было выполнено подключение к REST API с использованием Retrofit . Также были реализованы:

- Логика получения, отправки и обновления данных о привычках
- Обработка ошибок и сетевых исключений
- Кэширование данных через DataStore для работы в офлайне
- Хранение файлов и медиа

Для загрузки и хранения аватаров пользователей была интегрирована платформа Appwrite Storage . Это позволило организовать безопасное хранение изображений и ограничить доступ к ним только для владельцев.

- Реализация системы уведомлений

Для своевременного напоминания о необходимости выполнить привычку была реализована система уведомлений с использованием:

- AlarmManager — для планирования напоминаний
- NotificationManager — для отображения уведомлений

Эта система позволяет пользователю получать напоминания даже после перезагрузки устройства.

- **Тестирование и отладка**

Было проведено тестирование всех модулей приложения:

- Юнит-тестирование логики работы с привычками
- Тестирование сетевых запросов
- Проверка корректности отображения интерфейса
- Проверка работоспособности уведомлений

Использовались механизмы асинхронного программирования (Coroutines, Flow) для обеспечения надёжности и предсказуемости работы приложения.

- **Документирование и подготовка отчета**

Все этапы разработки были задокументированы. Создано описание архитектуры, используемых технологий, диаграмм UML и примеров кода. Подготовлены материалы для защиты выпускной квалификационной работы.

## **2. Анализ предметной области**

Анализ предметной области является важным этапом разработки программного обеспечения, поскольку позволяет понять суть задачи, определить ключевые сущности и их взаимодействие, а также сформулировать основные термины и концепции, используемые в рамках проекта.

### **2.1. Терминология (гlossарий предметной области)**

Для лучшего понимания структуры и функциональности приложения «Habit Tracker» приведем основные термины и понятия, используемые в данной предметной области:



- Привычка (Habit) — повторяющееся действие, которое пользователь стремится выполнять регулярно для достижения личных целей или формирования положительного поведенческого паттерна.
- Выполнение привычки (Completion) — отметка пользователем факта выполнения заданной привычки за определённый день. Может быть положительной (выполнено) или отрицательной (не выполнено).
- Напоминание (Reminder) — автоматическое уведомление, отправляемое пользователю в указанное время для напоминания о необходимости выполнить привычку.
- Статистика (Statistics) — информация, отражающая эффективность выполнения привычек за определённый период времени. Может отображаться в виде графиков, диаграмм и процентного соотношения выполненных/невыполненных действий.
- Профиль пользователя (User Profile) — данные, характеризующие пользователя: имя, фамилия, дата рождения, пол, аватар, настройки приложения.
- Firebase Authentication — сервис Google, предоставляющий инструменты аутентификации пользователей через email/пароль, Google Sign-In и другие способы входа.
- Retrofit — библиотека для Android, позволяющая реализовать клиентский HTTP API для взаимодействия с REST-серверами.
- Appwrite Storage — облачная система хранения файлов, используемая для загрузки и управления медиафайлами, такими как изображения профиля.

- DataStore — современная система хранения данных в Android, предназначенная для замены SharedPreferences. Используется для сохранения локальных данных пользователя.
- Jetpack Compose — современный набор библиотек для создания пользовательского интерфейса в Android-приложениях. Позволяет строить UI декларативно, с использованием Kotlin.
- Material Design 3 (MD3) — последняя версия дизайн-системы от Google, предлагающая обновлённые компоненты, цветовые схемы и принципы построения интерфейсов.
- MVVM (Model-View-ViewModel) — шаблон архитектуры приложений, разделение слоёв которого способствует чистоте кода и удобству тестирования.
- Coroutines — механизм асинхронного программирования в Kotlin, позволяющий выполнять долгие операции без блокировки главного потока.
- Flow — реактивный тип данных в Kotlin, предназначенный для работы с потоками данных во ViewModel.

Указанные термины используются на протяжении всей работы для точного описания архитектуры, функциональных возможностей и технической реализации разработанного мобильного приложения.

## **2.2. Обзор аналогов**

В рамках анализа предметной области были изучены существующие аналоги — мобильные приложения, предназначенные для отслеживания привычек и управления личными целями. Целью данного обзора является выявление ключевых функций, особенностей интерфейса и недостатков

популярных решений, чтобы на основе полученных данных спроектировать и разработать более удобное и функциональное приложение.

### **2.2.1. Habitica**

Habitica — это игровое приложение для отслеживания привычек и задач. Оно представляет собой геймифицированную систему, в которой пользователь выступает как персонаж, выполняющий задания (привычки) и получая за это награды, опыт и монеты.

Основные особенности:

- Геймификация: выполнение задач превращается в игру
- Система опыта, уровня, классов и экипировки
- Возможность добавления повторяющихся привычек, ежедневных задач и долгосрочных целей
- Поддержка команд и групповых задач
- Интеграция с календарем и напоминаниями

Недостатки:

- Сложный интерфейс, который может быть неудобен новичкам
- Многие функции доступны только через платную подписку
- Зависимость от игровой механики, что может отвлекать от реальных целей

На рисунке ниже показан интерфейс приложения Habitica, демонстрирующий ключевые функциональные возможности (см. Рисунок 1).

## The Habitica App's Key Features

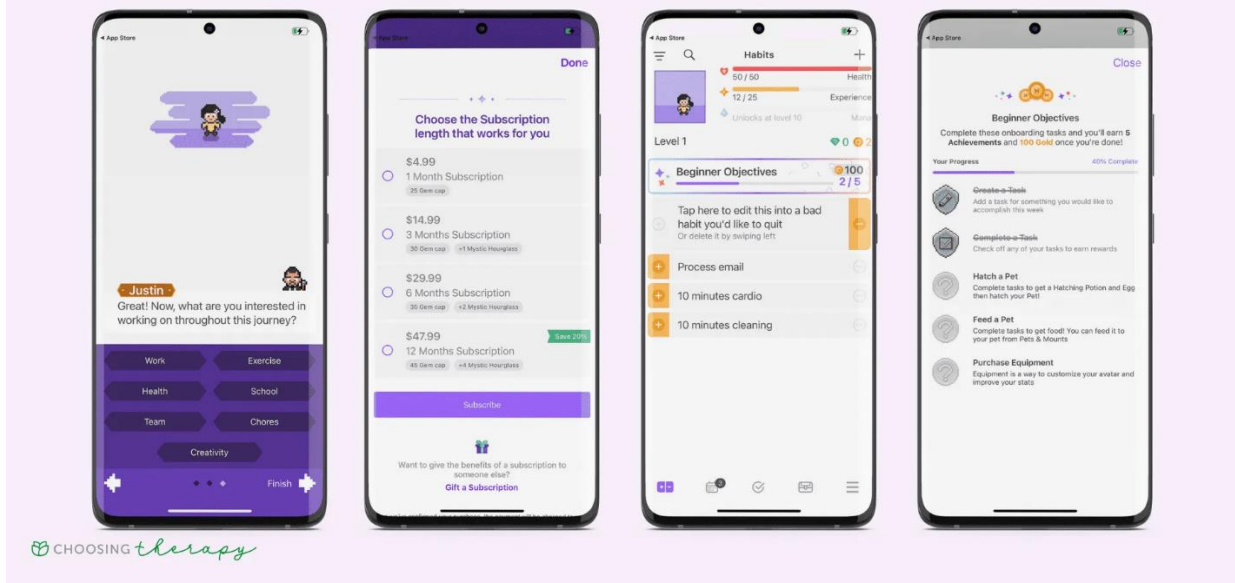


Рисунок 1 - Интерфейс приложения Habitica

### 2.2.2. Streaks

Streaks — минималистичное приложение для iOS, ориентированное на создание и поддержание полезных привычек. Принцип работы основан на поддержании "серии" (streak) — количества дней подряд, в которые пользователь выполнил определённую привычку.

Основные особенности:

- Минималистичный и интуитивно понятный интерфейс
- Визуализация прогресса в виде графика серий
- Поддержка уведомлений и напоминаний
- Возможность отслеживания до 12 привычек
- Синхронизация между устройствами через iCloud

Недостатки:

- Доступно только на устройствах Apple
- Ограниченное количество отслеживаемых привычек
- Отсутствие расширенной аналитики и фильтрации

На рисунке ниже показан интерфейс приложения Streaks, демонстрирующий ключевые функциональные возможности (см. Рисунок 2).



Рисунок 2 - Интерфейс приложения Streaks

### 2.2.3. Fabulous

Fabulous — это приложение, разработанное по принципам поведенческой психологии. Оно помогает пользователям формировать устойчивые привычки через структурированные рутинные практики.

Основные особенности:

- Фокус на научно обоснованных методах формирования привычек
- Персонализированные планы развития
- Ежедневные чек-листы и рекомендации
- Поддержка напоминаний и трекинга прогресса

Недостатки:

- Платная модель подписки
- Ограниченная гибкость в создании собственных привычек
- Сравнительно высокая стоимость

На рисунке ниже показан интерфейс приложения Fabulous, демонстрирующий ключевые функциональные возможности (см. Рисунок 3).

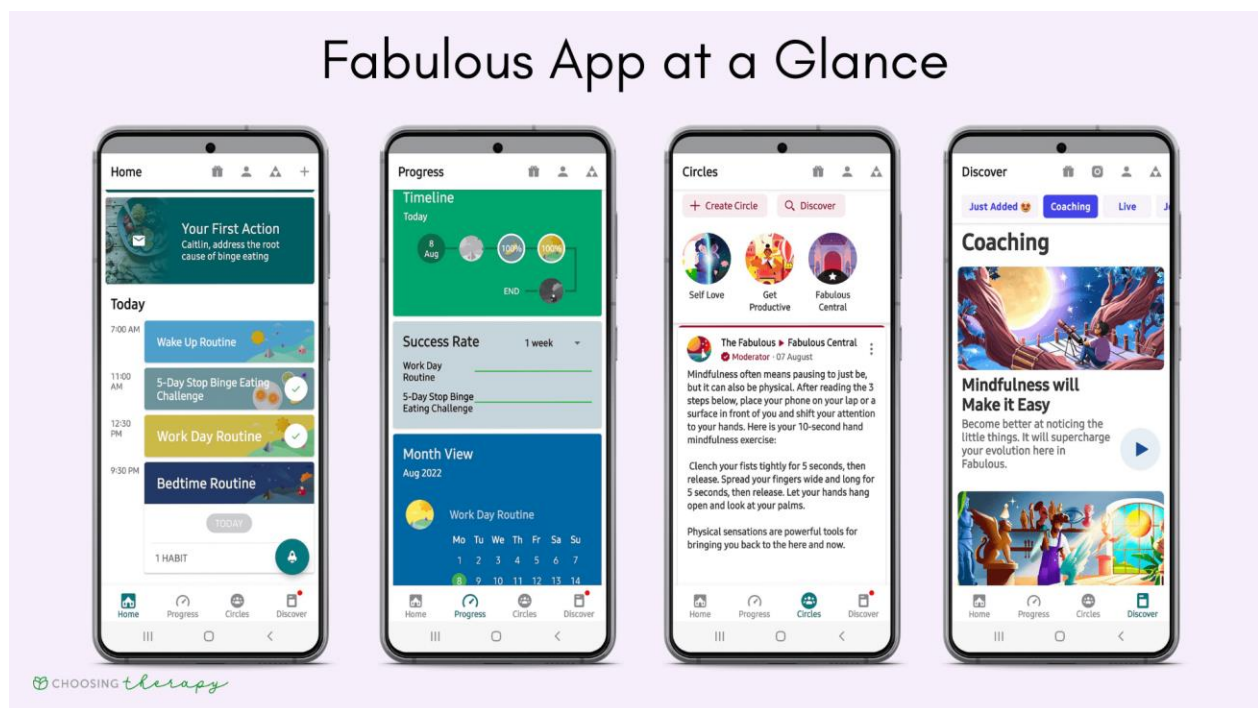


Рисунок 3 - Интерфейс приложения Fabulous

#### 2.2.4. Loop Habit Tracker

Loop Habit Tracker — это простое и бесплатное приложение с открытым исходным кодом для Android. Оно позволяет создавать и отслеживать привычки без лишних функций и отвлекающих элементов.

Основные особенности:

- Бесплатное и с открытым исходным кодом
- Поддержка разных типов привычек: ежедневные, еженедельные и т.д.
- Встроенный график эффективности
- Легкий и быстрый интерфейс

Недостатки:

- Ограниченный функционал (отсутствуют уведомления, облачная синхронизация)
- Неактивное развитие проекта
- Отсутствие продвинутой визуализации и аналитики

На рисунке ниже показан интерфейс приложения Loop Habit Tracker, демонстрирующий ключевые функциональные возможности (см. Рисунок 4).

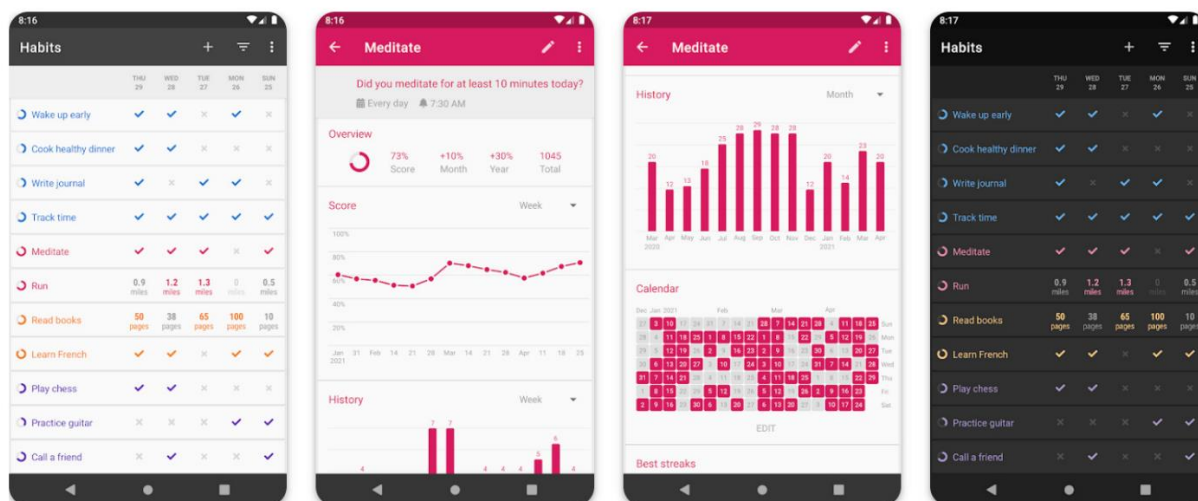


Рисунок 4 - Интерфейс приложения Loop Habit Tracker

## Анализ и выводы

На основании проведённого обзора можно сделать следующие выводы:

- Большинство существующих решений имеют ограниченную функциональность , либо требуют платной подписки для получения всех возможностей.
- Приложения, основанные на геймификации , могут быть менее практичны для пользователей, которым нужна минималистичная система .
- Мобильная доступность ограничена в некоторых случаях (например, Streaks — только для iOS).
- Открытые решения (например, Loop Habit Tracker) часто не имеют необходимого функционала для широкого круга пользователей.

Разрабатываемое приложение «Habit Tracker» направлено на объединение преимуществ рассмотренных аналогов, предоставляя при этом современный интерфейс, свободу в настройке привычек, систему уведомлений



и возможность синхронизации данных в облаке. Это делает его конкурентоспособным решением среди существующих продуктов на рынке.

### **2.3. Моделирование системы**

Для проектирования и разработки мобильного приложения "Habit Tracker" были созданы различные диаграммы UML, которые помогли визуализировать структуру системы, взаимодействие между её компонентами и потоки выполнения различных функций. В данном разделе представлены основные модели, используемые для анализа и спроектирования системы.

#### **2.3.1. Диаграмма вариантов использования (Use Case Diagram)**

Диаграмма вариантов использования (Use Case Diagram) представляет собой графическое представление основных функциональных возможностей приложения "Habit Tracker". На диаграмме (см. Рисунок 5) показано взаимодействие между пользователем и системой через различные варианты использования.

Описание диаграммы:

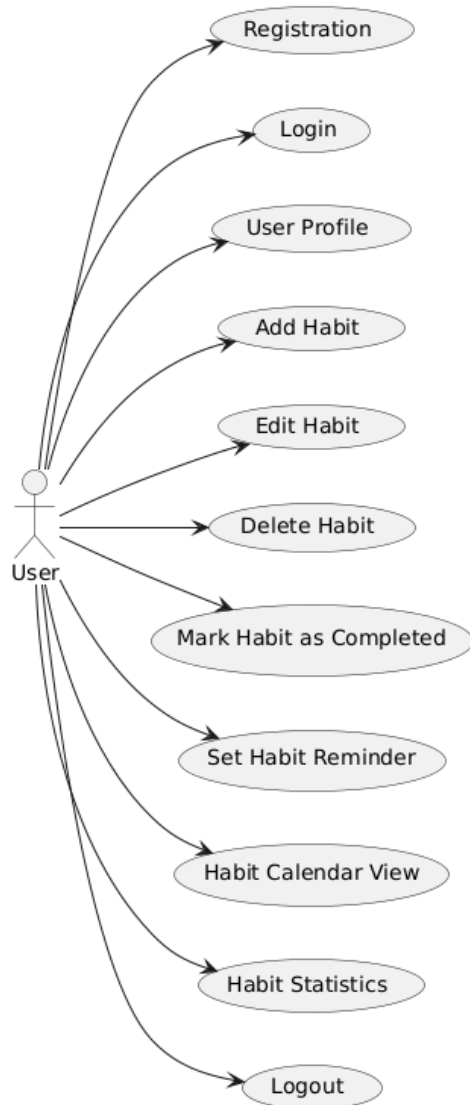


Рисунок 5 - Диаграмма вариантов использования

- Актор:
  - User — это главный актер, который взаимодействует с системой через различные варианты использования.
- Варианты использования (Use Cases):
  - Registration: Регистрация нового пользователя.
  - Login: Авторизация существующего пользователя.
  - User Profile: Просмотр и редактирование профиля пользователя.

- Add Habit: Создание новой привычки.
- Edit Habit: Редактирование существующей привычки.
- Delete Habit: Удаление привычки.
- Mark Habit as Completed: Отметка привычки как выполненной.
- Set Habit Reminder: Настройка напоминаний для привычки.
- Habit Calendar View: Просмотр календаря с расписанием привычек.
- Habit Statistics: Анализ статистики выполнения привычек.
- Logout: Выход из учетной записи.

Пояснение диаграммы:

На диаграмме видно, что пользователь может выполнять следующие действия:

- Регистрация и авторизация: Пользователь может зарегистрироваться или войти в систему.
- Управление профилем: После входа пользователь может просматривать и редактировать свой профиль.
- Управление привычками: Пользователь может создавать новые привычки, редактировать существующие, удалять их и отмечать выполненные.
- Настройка напоминаний: Для каждой привычки можно настроить время напоминания.
- Календарь привычек: Пользователь может просмотреть все свои привычки в виде календаря.

- Статистика выполнения: Система предоставляет возможность анализировать прогресс выполнения привычек за определённый период времени.
- Выход из системы: Пользователь может завершить сессию и выйти из приложения.

Значение диаграммы:

Диаграмма вариантов использования позволяет четко определить функционал, доступный пользователю, и выделить ключевые операции, которые система должна поддерживать. Это помогает в дальнейшем проектировании архитектуры и реализации логики работы приложения.

### **2.3.2. Диаграмма деятельности (Диаграмма активности)**

Диаграмма деятельности (Activity Diagram) представляет собой визуальное описание потока выполнения действий в системе, показывающее последовательность событий и переходов между различными состояниями. В данном разделе будет рассмотрена диаграмма, описывающая основной жизненный цикл работы пользователя с приложением "Habit Tracker". На диаграмме (см. Рисунок 6) представлены ключевые этапы взаимодействия пользователя с приложением, начиная с входа в систему и заканчивая выполнением действий с привычками.

Описание диаграммы:

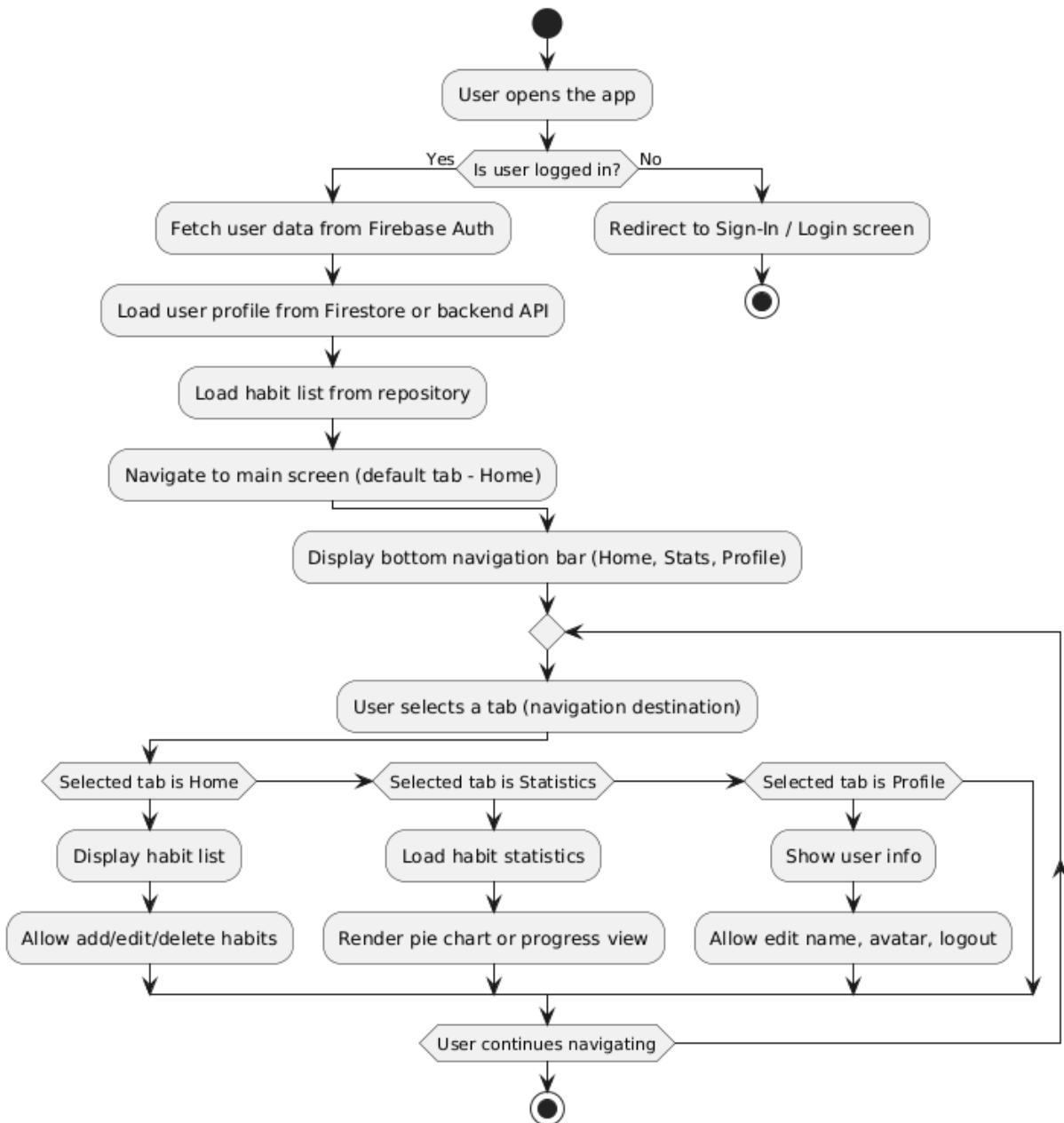


Рисунок 6 - Диаграмма деятельности изменения текущей директории

- Пользователь открывает приложение

Поток начинается с события "User opens the app" (Пользователь открывает приложение).

- Проверка аутентификации пользователя

Система проверяет, залогинен ли пользователь через Firebase Authentication:

- Если пользователь залогинен (Yes), то происходит загрузка данных профиля из Firestore или API.
- Если пользователь не залогинен (No), то его перенаправляют на экран входа/регистрации.
- Загрузка данных пользователя  
После успешной аутентификации система получает данные профиля пользователя из Firebase Firestore или другого backend-источника.
- Загрузка списка привычек  
Затем загружается список привычек пользователя из локального репозитория или сервера.
- Переход на главный экран  
После загрузки всех необходимых данных пользователь попадает на главный экран приложения (по умолчанию — вкладка "Home").
- Отображение нижней панели навигации  
На главном экране отображается нижняя панель навигации с тремя вкладками:
  - Home : Главная страница с списком привычек.
  - Stats : Страница статистики с графиками и аналитикой.
  - Profile : Страница профиля пользователя.
- Выбор пользователем вкладки  
Пользователь может выбирать одну из трёх вкладок:
  - Home : Отображается список привычек, где пользователь может добавлять, редактировать или удалять привычки.

- Stats : Загружается статистика выполнения привычек за выбранный период времени, отображается круговая диаграмма или прогресс-бар.
- Profile : Показывается информация о пользователе (имя, аватар, настройки). Здесь также доступны функции редактирования профиля и выхода из аккаунта.
- Продолжение навигации  
После выбора вкладки пользователь может продолжить работу с приложением, выполняя различные действия (например, отметку привычки как выполненной, изменение настроек или просмотр статистики).
- Конец потока  
Поток завершается, когда пользователь завершает работу с приложением.

#### Анализ диаграммы:

- Логическая структура: Диаграмма четко демонстрирует последовательность действий, начиная с запуска приложения и заканчивая взаимодействием с различными экранами.
- Условные переходы: Используется условие "Is user logged in?" для определения дальнейших действий (залогинен или нет).
- Многоступенчатый процесс: Каждый шаг детально описан, что помогает понять, как система реагирует на действия пользователя.

- Возврат к началу: После выбора вкладки пользователь может вернуться к предыдущим шагам, например, чтобы выбрать другую вкладку или повторно загрузить данные.

Значение диаграммы:

Диаграмма деятельности позволяет:

- Описать основной жизненный цикл работы с приложением.
- Увидеть взаимосвязь между различными экранами и модулями.
- Выявить возможные точки ошибок или проблемных мест (например, при загрузке данных из Firebase).
- Помочь в тестировании и отладке системы, так как каждый шаг явно описан и можно легко отслеживать поток выполнения.

### **2.3.3. Диаграмма классов (Class Diagram)**

Диаграмма классов представляет собой графическое представление структуры объектной модели системы, показывающее взаимосвязи между классами и их атрибуты/методы. Она помогает понять архитектуру приложения "Habit Tracker" с точки зрения объектно-ориентированного программирования. На рисунке (см. Рисунок 7) представлена диаграмма классов, которая демонстрирует основные компоненты системы, включая контроллеры, сервисы, репозитории, DTO (Data Transfer Objects), модели данных и сущности базы данных.

Описание диаграммы:



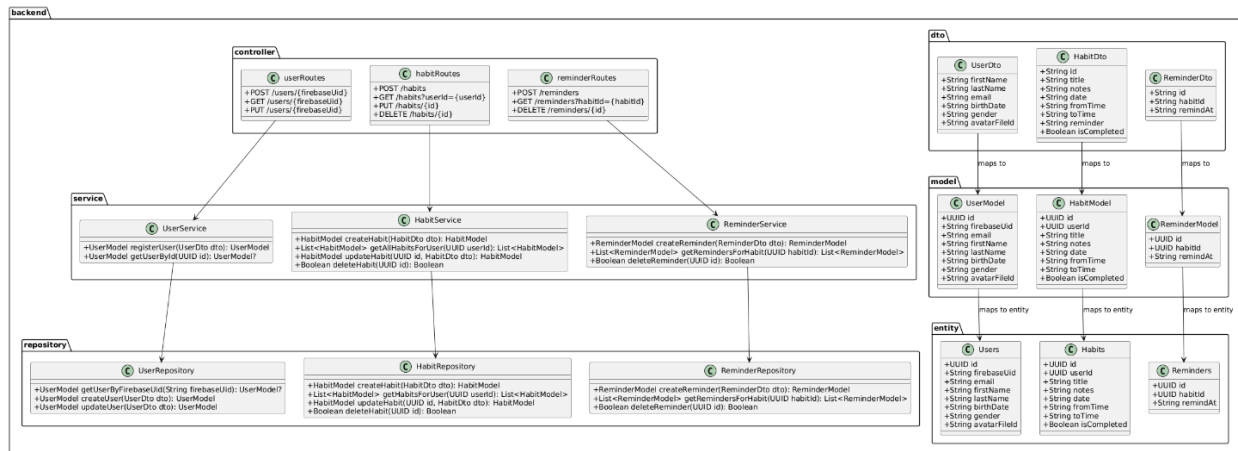


Рисунок 7 - Диаграмма классов

- Контроллеры (Controllers):
  - userRoutes: Обработывает запросы, связанные с пользователями (например, регистрация, получение информации о пользователе).
  - habitRoutes: Обработывает операции над привычками (добавление, обновление, удаление).
  - reminderRoutes: Обработывает напоминания для привычек.
- Сервисы (Services):
  - UserService: Определяет бизнес-логику работы с пользователями.
    - Методы:
      - registerUser(UserDto dto): UserModel
      - getUserByFirebaseUid(String firebaseUid): UserModel?
      - getUserById(UUID id): UserModel?
  - HabitService: Определяет бизнес-логику работы с привычками.
    - Методы:
      - createHabit(HabitDto dto): HabitModel
      - getAllHabitsForUser(UUID userId): List<HabitModel>
      - updateHabit(UUID id, HabitDto dto): HabitModel

- deleteHabit(UUID id): Boolean
- ReminderService: Определяет бизнес-логику работы с напоминаниями.
  - Методы:
    - createReminder(ReminderDto dto): ReminderModel
    - getRemindersForHabit(UUID habitId): List<ReminderModel>
    - deleteReminder(UUID id): Boolean
- Репозитории (Repositories):
  - UserRepository: Отвечает за взаимодействие с данными пользователей.
  - HabitRepository: Отвечает за работу с данными привычек.
  - ReminderRepository: Отвечает за работу с данными напоминаний.
- DTO (Data Transfer Objects):
  - UserDto: Представляет данные пользователя для передачи между слоями.
    - Атрибуты:
      - firstName, lastName, email, birthDate, gender, avatarField
  - HabitDto: Представляет данные привычки.
    - Атрибуты:
      - id, title, notes, date, fromTime, toTime, isCompleted
  - ReminderDto: Представляет данные напоминания.
    - Атрибуты:
      - id, habitId, remindAt
- Модели данных (Models):
  - UserModel: Представляет модель пользователя.

- Атрибуты:
  - id, firebaseUid, email, firstName, lastName, birthDate, gender, avatarField
- HabitModel: Представляет модель привычки.
  - Атрибуты:
    - id, firebaseUid, title, notes, date, fromTime, toTime, isCompleted
- ReminderModel: Представляет модель напоминания.
  - Атрибуты:
    - id, userId, habitId, remindAt
- Сущности базы данных (Entities):
  - Users: Таблица для хранения данных пользователей.
  - Habits: Таблица для хранения данных привычек.
  - Reminders: Таблица для хранения данных напоминаний.

#### Взаимосвязи:

- DTO → Models: DTO преобразуются в соответствующие модели данных (например, UserDto → UserModel).
- Models → Entities: Модели данных отображаются на сущности базы данных (например, UserModel → Users).
- Controllers → Services: Контроллеры вызывают методы сервисов для выполнения бизнес-логики.
- Services → Repositories: Сервисы используют репозитории для доступа к данным.

#### Значение диаграммы:

Диаграмма классов позволяет четко определить структуру системы, выделить ответственные за каждую функцию классы и их взаимодействия. Это помогает в дальнейшем проектировании и реализации логики работы приложения, а также упрощает поддержку и масштабирование системы.

#### **2.3.4. Диаграмма ER (Entity-Relationship Diagram)**

Диаграмма ER (Entity-Relationship Diagram) представляет собой графическое представление структуры базы данных, показывающее сущности (entities), атрибуты и отношения между ними. Она помогает визуализировать логическую модель данных приложения "Habit Tracker" и определить связи между различными таблицами. На рисунке (см. Рисунок 8) представлена диаграмма ER, которая демонстрирует основные сущности и их взаимосвязи.

Описание диаграммы:

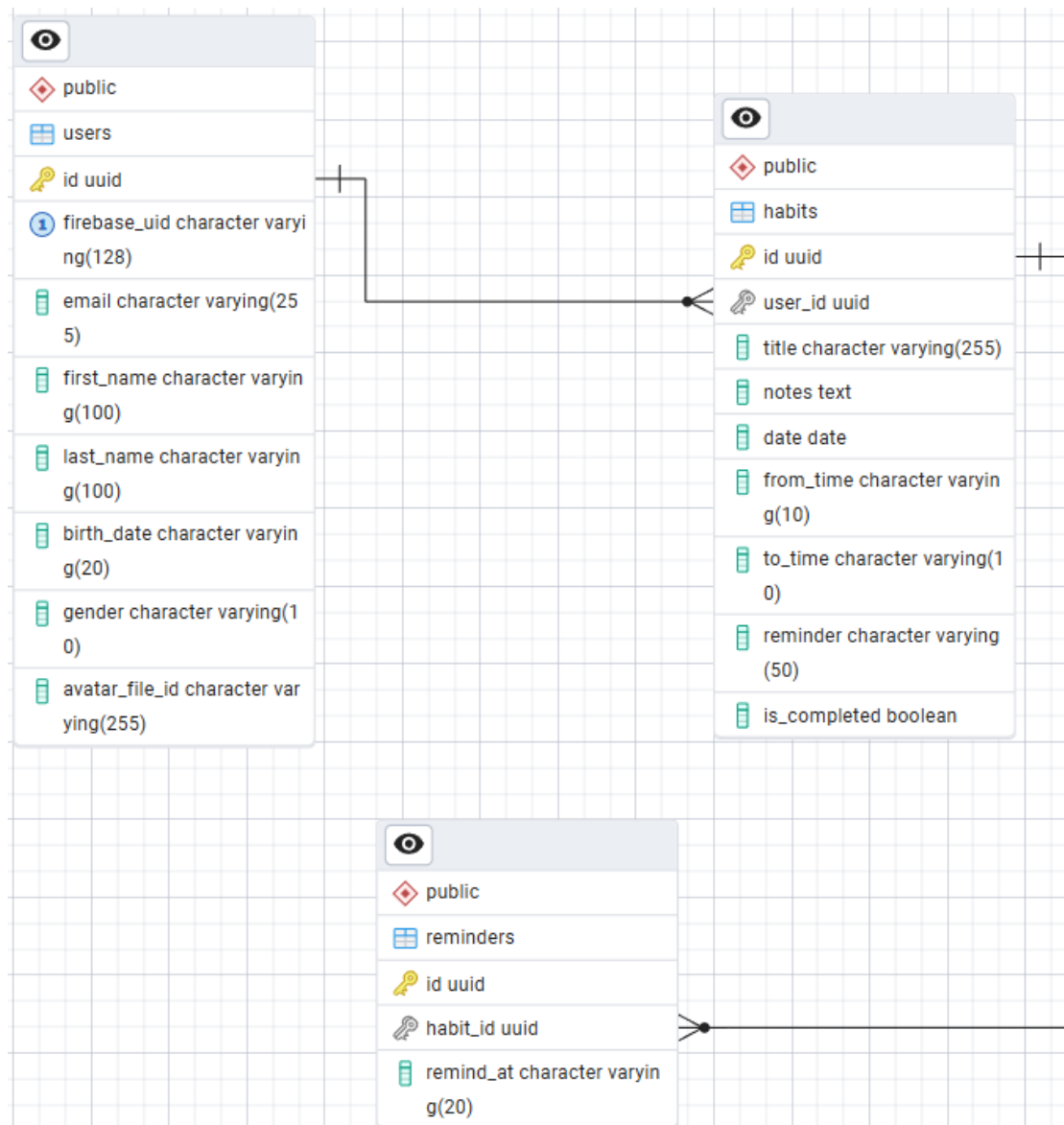


Рисунок 8 - Диаграмма ER

- Сущность users:
  - Представляет пользователей системы.
  - Атрибуты:
    - id (uuid): Уникальный идентификатор пользователя.
    - firebase\_uid (character varying(128)): Идентификатор пользователя из Firebase Authentication.

- email (character varying(255)): Электронная почта пользователя.
- first\_name (character varying(100)): Имя пользователя.
- last\_name (character varying(100)): Фамилия пользователя.
- birth\_date (character varying(20)): Дата рождения пользователя.
- gender (character varying(10)): Пол пользователя.
- avatar\_file\_id (character varying(255)): Идентификатор файла аватара пользователя.
- Сущность habits:
  - Представляет привычки, созданные пользователем.
  - Атрибуты:
    - id (uuid): Уникальный идентификатор привычки.
    - user\_id (uuid): Ссылка на пользователя, которому принадлежит привычка.
    - title (character varying(255)): Название привычки.
    - notes (text): Примечания к привычке.
    - date (date): Дата создания или выполнения привычки.
    - from\_time (character varying(10)): Время начала выполнения привычки.
    - to\_time (character varying(10)): Время окончания выполнения привычки.
    - reminder (character varying(50)): Текст напоминания для привычки.
    - is\_completed (boolean): Флаг, указывающий, выполнена ли привычка.
- Сущность reminders:

- Представляет напоминания, связанные с привычками.
- Атрибуты:
  - id (uuid): Уникальный идентификатор напоминания.
  - habit\_id (uuid): Ссылка на привычку, к которой относится напоминание.
  - remind\_at (character varying(20)): Время, когда должно быть отправлено напоминание.

Отношения между сущностями:

- users ↔ habits: Один-ко-многим (One-to-Many).
- Каждый пользователь может иметь несколько привычек, но каждая привычка принадлежит только одному пользователю.
- habits ↔ reminders: Один-ко-многим (One-to-Many).
- Каждая привычка может иметь несколько напоминаний, но каждое напоминание связано только с одной привычкой.

Значение диаграммы:

Диаграмма ER позволяет четко определить структуру базы данных, выявить ключевые сущности и их атрибуты, а также понять взаимосвязи между ними. Это помогает в проектировании базы данных и обеспечивает эффективное хранение и обработку данных в приложении.

### 2.3.5. Диаграмма последовательности (Sequence Diagram)

Диаграмма последовательности (Sequence Diagram) представляет собой графическое представление взаимодействия между различными компонентами системы в хронологическом порядке. Она показывает поток сообщений и вызовов между объектами, что помогает понять логику

выполнения определённых операций в приложении "Habit Tracker". На рисунке (см. Рисунок 9) представлена диаграмма последовательности, которая демонстрирует процесс создания новой привычки (Add Habit) с точки зрения взаимодействия между пользователем, клиентским приложением и серверной частью.

#### Описание диаграммы:

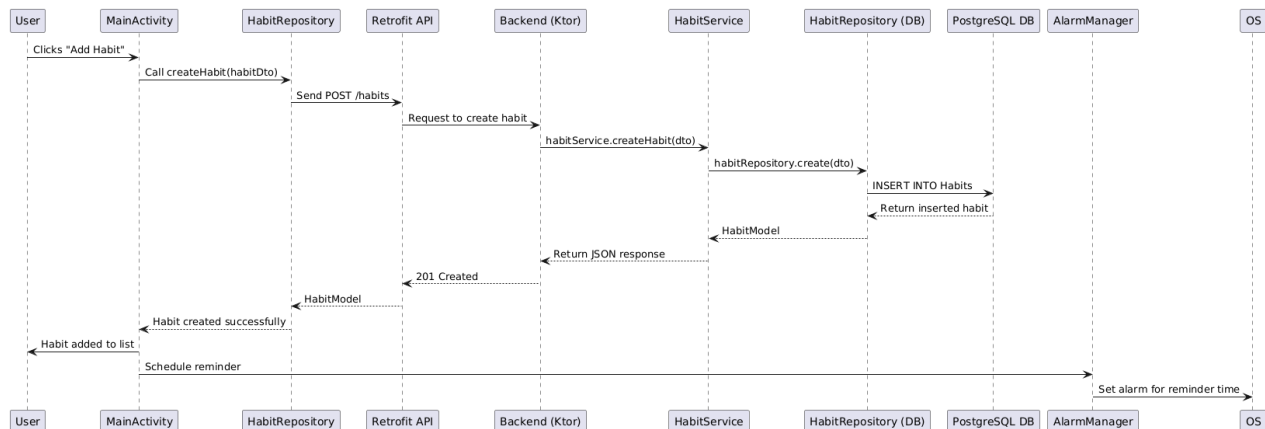


Рисунок 9 - Диаграмма последовательности

- Пользовательский интерфейс (MainActivity):
  - Пользователь нажимает кнопку "Добавить привычку" (Clicks "Add Habit").
  - Приложение отправляет запрос на создание привычки через HabitRepository.
- Клиентская часть (HabitRepository):
  - HabitRepository использует Retrofit для отправки HTTP-запроса на сервер (Send POST /habits).
- Серверная часть (Backend (Ktor)):
  - Серверный API получает запрос на создание привычки.



- Запрос передается в HabitService.
- Сервис (HabitService):
  - HabitService обрабатывает запрос и передаёт данные в HabitRepository (DB) для сохранения в базу данных.
- База данных (HabitRepository (DB)):
  - HabitRepository (DB) выполняет SQL-запрос INSERT INTO Habits, добавляя новую привычку в таблицу базы данных PostgreSQL.
- Возврат ответа:
  - После успешного добавления привычки сервер возвращает JSON-ответ с кодом состояния 201 Created.
  - Ответ преобразуется в модель HabitModel и возвращается обратно в клиентское приложение.
- Управление напоминаниями (AlarmManager):
  - После успешного создания привычки клиентское приложение устанавливает напоминание через AlarmManager для оповещения пользователя о времени выполнения привычки.

Значение диаграммы:

Диаграмма последовательности позволяет:

- Описать шаг за шагом процесс взаимодействия между компонентами системы.
- Показать поток данных от клиента к серверу и обратно.
- Выявить точки взаимодействия с внешними сервисами (например, AlarmManager).
- Понять, как различные части системы работают вместе для выполнения конкретной задачи.

### 3. Реализация

В данном разделе описывается процесс разработки мобильного приложения «Habit Tracker», включая используемые технологии, этапы реализации и особенности пользовательского интерфейса. Приложение было разработано с применением современных инструментов и библиотек, что позволило создать функциональный, отзывчивый и масштабируемый программный продукт.

#### 3.1. Средства реализации

Разработка мобильного приложения осуществлялась с использованием следующих технологий и инструментов:

Фронтенд (Android-приложение):

- Язык программирования: Kotlin — основной язык разработки Android-приложений.
- Библиотека UI: Jetpack Compose — декларативный фреймворк для построения пользовательского интерфейса.
- Дизайн-система: Material Design 3 — использовалась для соблюдения современного стиля и согласованности элементов интерфейса.
- Анимации и переходы: AnimatedVisibility, slideInHorizontally, fadeIn и другие анимационные API из Jetpack Compose.
- Работа с датами: Библиотека ThreetenABP — для работы с календарём и форматированием дат.
- Загрузка изображений: Coil (coil-compose) — для асинхронной загрузки и отображения аватаров пользователей.

- Визуализация данных: MPAndroidChart — для отображения графиков и диаграмм в разделе статистики.
- Сетевые запросы: Retrofit — для взаимодействия с REST API сервера.
- Управление зависимостями: Hilt — для внедрения зависимостей на уровне Android-приложения.
- Асинхронное программирование: Kotlin Coroutines и Flow — для выполнения асинхронных операций и реактивного программирования.
- Локальное хранение данных: DataStore — для хранения настроек пользователя и временных данных.
- Авторизация: Firebase Authentication — для безопасной авторизации через email/пароль и Google Sign-In.
- Облачное хранение файлов: Appwrite Storage — для загрузки и хранения медиафайлов, таких как аватары.
- Уведомления: AlarmManager и NotificationManager — для реализации напоминаний о привычках.

Эти технологии были выбраны с учетом их популярности, производительности, поддержки сообществом и возможности интеграции между собой.

### **3.2. Этапы реализации и интеграции**

Процесс разработки приложения был разделён на несколько ключевых этапов:

- Подготовительный этап:

- Изучение предметной области и анализ существующих решений.
- Постановка задач и формулирование требований.
- Выбор архитектуры приложения (MVVM).
- Определение списка используемых библиотек и фреймворков.
- Проектирование архитектуры:
  - Разработка диаграмм UML: вариантов использования, классов, ER-диаграммы, диаграммы последовательности и активностей.
  - Создание модели данных и определение связей между таблицами.
  - Проектирование клиент-серверного взаимодействия через REST API.
- Разработка серверной части:
  - Настройка и запуск PostgreSQL.
  - Реализация моделей данных и взаимодействие с базой через Exposed.
  - Разработка REST API с использованием Ktor.
  - Контейнеризация сервера с помощью Docker.
- Реализация клиентской части:
  - Создание экранов приложения: главный экран, экран статистики, профиль пользователя.
  - Интеграция Jetpack Compose и Material Design 3 для построения интерфейса.
  - Реализация логики регистрации, входа и управления профилем пользователя.

- Создание модулей добавления, редактирования и удаления привычек.
- Интеграция уведомлений через AlarmManager и NotificationManager.
- Подключение системы визуализации статистики с использованием MPAndroidChart.
- Интеграция с внешними сервисами:
  - Подключение Firebase Authentication для авторизации и регистрации.
  - Интеграция Retrofit для работы с REST API.
  - Использование Appwrite Storage для загрузки и хранения изображений.
- Тестирование и отладка:
  - Юнит-тестирование бизнес-логики.
  - Тестирование сетевых запросов и обработки ошибок.
  - Проверка корректности отображения интерфейса.
  - Тестирование уведомлений и работы в фоновом режиме.
- Оформление документации:
  - Написание технической документации.
  - Описание архитектуры, используемых технологий и примеров кода.
  - Подготовка материалов для защиты выпускной квалификационной работы.

### 3.3. Реализация интерфейса

Пользовательский интерфейс приложения «Habit Tracker» был реализован с использованием Jetpack Compose , что позволило создать современный, отзывчивый и легко поддерживаемый UI.

Основные компоненты интерфейса:

- MainActivity.kt — главная точка входа в приложение, содержащая навигацию между экранами.
- MainPage.kt — экран со списком привычек, календарём и кнопкой добавления новых привычек.
- HabitCalendar.kt — компонент календаря, позволяющий просматривать привычки за определённую дату.
- StatisticsPage.kt — экран, отображающий статистику выполнения привычек в виде графиков и диаграмм.
- ProfilePage.kt — страница профиля пользователя, где можно просмотреть и изменить данные аккаунта.
- AddHabitDialog.kt — диалоговое окно добавления новой привычки.
- EditHabitDialog.kt — диалоговое окно редактирования привычки.

Примеры реализации UI:

- Главная страница:
  - Отображается список привычек.
  - Каждая привычка имеет состояние: выполнена / не выполнена.
  - Возможность отметить привычку как выполненную через простое нажатие.

- Добавление привычки:
  - Форма содержит поля: название, описание, время выполнения, напоминание.
  - После сохранения привычка добавляется в список и отправляется на сервер через Retrofit.
- Статистика:
  - Используется библиотека MPAndroidChart.
  - Отображаются круговые диаграммы с процентным соотношением выполненных и невыполненных привычек.
- Профиль пользователя:
  - Отображается имя, аватар, дата рождения.
  - Предоставлена возможность редактирования данных и выхода из аккаунта.
- Нижняя панель навигации:
  - Включает три вкладки: Главная, Статистика, Профиль.
  - Реализована с анимацией переключения между вкладками.
- Анимации и переходы:
  - Анимированное переключение между вкладками.
  - Плавные появления и исчезновения диалоговых окон.
  - Анимации отметки выполненной привычки (scaleIn, fadeIn).
- Темы оформления:
  - Поддержка Light/Dark Mode с возможностью переключения через настройки.
  - Цветовая палитра была создана на основе Material Design 3.
- Локализация:

- Поддержка английского и русского языков.
- Все строковые ресурсы вынесены в res/values/strings.xml.

### **3.3.1. Страница входа (Sign In Page)**

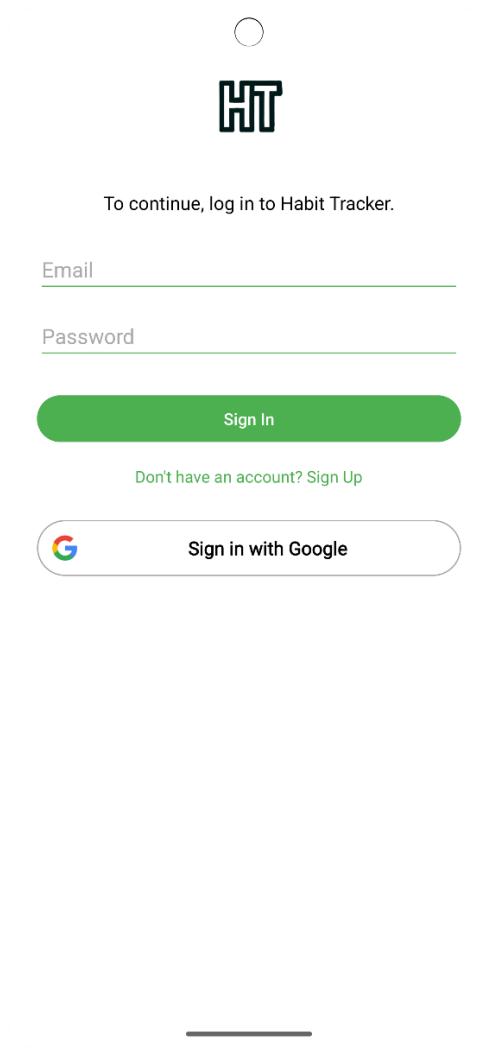
Страница входа является первым экраном, который пользователь видит при запуске приложения "Habit Tracker". Она предназначена для аутентификации пользователя и предоставления доступа к основным функциям приложения. На данной странице реализованы следующие возможности:

- Ввод электронной почты и пароля:  
Пользователь может ввести свои учетные данные (email и пароль) для авторизации через Firebase Authentication.
- Авторизация через Google:  
Для удобства пользователей реализована возможность входа через аккаунт Google. Это позволяет быстро войти в систему без необходимости ввода логина и пароля.
- Регистрация нового аккаунта:  
Если у пользователя нет учетной записи, он может перейти на страницу регистрации, нажав ссылку "Don't have an account? Sign Up".

Интерфейс страницы входа:

На рисунке (см. Рисунок 10) показано оформление страницы входа:





The image shows a login screen for an application called 'Habit Tracker'. At the top, there is a small circle icon and a logo consisting of the letters 'HT' in a stylized, blocky font. Below the logo, the text 'To continue, log in to Habit Tracker.' is displayed. There are two input fields: 'Email' and 'Password', each with a green underline. Below these fields is a green rounded button labeled 'Sign In'. Underneath the button, there is a link that says 'Don't have an account? Sign Up'. At the bottom, there is a rounded button with the Google logo and the text 'Sign in with Google'.

Рисунок 10 - Страница входа

Функциональность страницы входа:

- Обработка ввода данных:
  - Пользователь вводит email и пароль в соответствующие поля.
  - При нажатии кнопки "Sign In", данные отправляются на сервер Firebase Authentication для проверки.
- Авторизация через Google:

- Нажатие кнопки "Sign in with Google" открывает диалоговое окно Google для выбора аккаунта.
- После успешного выбора аккаунта, данные пользователя передаются в приложение через Firebase Authentication.
- Переход на страницу регистрации:
  - Если пользователь не имеет учетной записи, он может перейти на страницу регистрации, нажав ссылку "Sign Up".

Техническая реализация:

- Firebase Authentication: Используется для аутентификации пользователей через email/пароль и Google Sign-In.
- Jetpack Compose: Интерфейс страницы создан с использованием Jetpack Compose, что обеспечивает гибкость и простоту в разработке UI.
- Material Design 3: Стили и компоненты соответствуют Material Design 3, что гарантирует современный и согласованный внешний вид интерфейса.
- Анимации: Кнопки и поля ввода поддерживают плавные анимации при взаимодействии пользователя.

Значение страницы входа:

Страница входа играет ключевую роль в обеспечении безопасности и удобства использования приложения. Она позволяет:

- Защитить данные пользователей с помощью аутентификации.
- Предоставить различные способы входа (email/пароль, Google Sign-In).

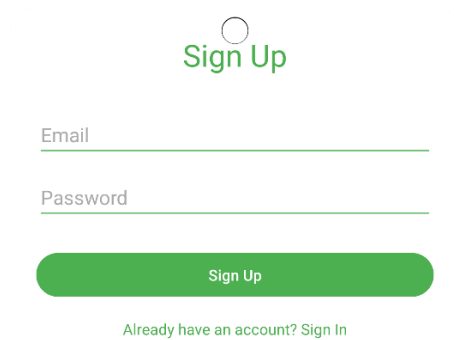
- Упростить процесс регистрации новых пользователей через ссылку "Sign Up".

### **3.3.2. Страница регистрации (Sign Up Page)**

Страница регистрации предназначена для создания новых учетных записей в приложении "Habit Tracker". Она позволяет пользователям зарегистрироваться, используя электронную почту и пароль. После успешной регистрации пользователь может войти в систему и начать использовать функционал приложения.

Интерфейс страницы регистрации:

На рисунке (см. Рисунок 11) показано оформление страницы регистрации:

A registration form titled "Sign Up" with a green header. It contains two input fields: "Email" and "Password", both with green borders. Below the fields is a green "Sign Up" button. At the bottom, there is a link that says "Already have an account? Sign In".

Sign Up

Email

Password

Sign Up

Already have an account? Sign In

---

Рисунок 11 - Страница регистрации

Функциональность страницы регистрации:

- Ввод данных:
  - Пользователь вводит свои данные (email и пароль) в соответствующие поля.
- Обработка ввода данных:
  - При нажатии кнопки "Sign Up", данные отправляются на сервер Firebase Authentication для проверки и создания новой учетной записи.

- Переход на страницу входа:
  - Если у пользователя уже есть учетная запись, он может перейти на страницу входа, нажав ссылку "Already have an account? Sign In".

Техническая реализация:

- Firebase Authentication: Используется для регистрации пользователей через email/пароль.
- Jetpack Compose: Интерфейс страницы создан с использованием Jetpack Compose, что обеспечивает гибкость и простоту в разработке UI.
- Material Design 3: Стили и компоненты соответствуют Material Design 3, что гарантирует современный и согласованный внешний вид интерфейса.
- Анимации: Кнопки и поля ввода поддерживают плавные анимации при взаимодействии пользователя.

Значение страницы регистрации:

Страница регистрации играет ключевую роль в обеспечении возможности создания новых учетных записей. Она позволяет:

- Защитить данные пользователей с помощью аутентификации.
- Предоставить удобный способ создания аккаунта через email/пароль.
- Упростить процесс перехода к основным функциям приложения после регистрации.

### 3.3.3. Главная страница (Main Page)

Главная страница является центральным экраном приложения "Habit Tracker", где пользователь может видеть список своих привычек, отмечать их выполнение и просматривать статистику. Она предоставляет наглядное представление о прогрессе пользователя в формировании полезных привычек.

Интерфейс главной страницы:

На рисунке (см. Рисунок 12) показано оформление главной страницы:

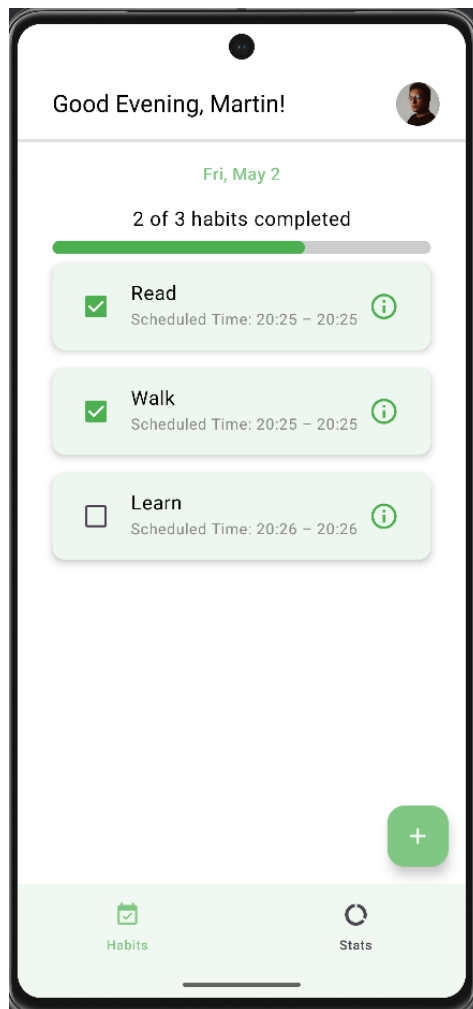


Рисунок 12 - Главная страница

Функциональность главной страницы:

- Отображение списка привычек:
  - Пользователь видит все свои привычки с указанием времени выполнения.
  - Для каждой привычки доступна возможность отметить её как выполненную или просмотреть дополнительную информацию.
- Отметка выполнения привычки:
  - Пользователь может отметить привычку как выполненную, нажав соответствующий флажок.
  - После отметки состояние привычки обновляется в базе данных через Retrofit API.
- Добавление новой привычки:
  - Нажатие кнопки "+" открывает диалоговое окно для создания новой привычки.
  - Пользователь может ввести название, описание, время выполнения и другие параметры привычки.
- Переход к статистике:
  - Нажатие на вкладку "Stats" перенаправляет пользователя на страницу анализа прогресса выполнения привычек.

Техническая реализация:

- Jetpack Compose: Интерфейс создан с использованием Jetpack Compose, что обеспечивает гибкость и простоту в разработке UI.
- Material Design 3: Стили и компоненты соответствуют Material Design 3, гарантируя современный и согласованный внешний вид интерфейса.

- Анимации: Используются анимации для плавного отображения состояния привычек и переходов между экранами.
- Уведомления: Если пользователь не выполнил привычку в указанное время, система отправляет уведомление через AlarmManager.

Значение главной страницы:

Главная страница играет ключевую роль в приложении "Habit Tracker", так как она:

- Предоставляет пользователю основной инструмент управления своими привычками.
- Отображает актуальный прогресс выполнения задач.
- Обеспечивает удобный доступ к функциям добавления, редактирования и удаления привычек.

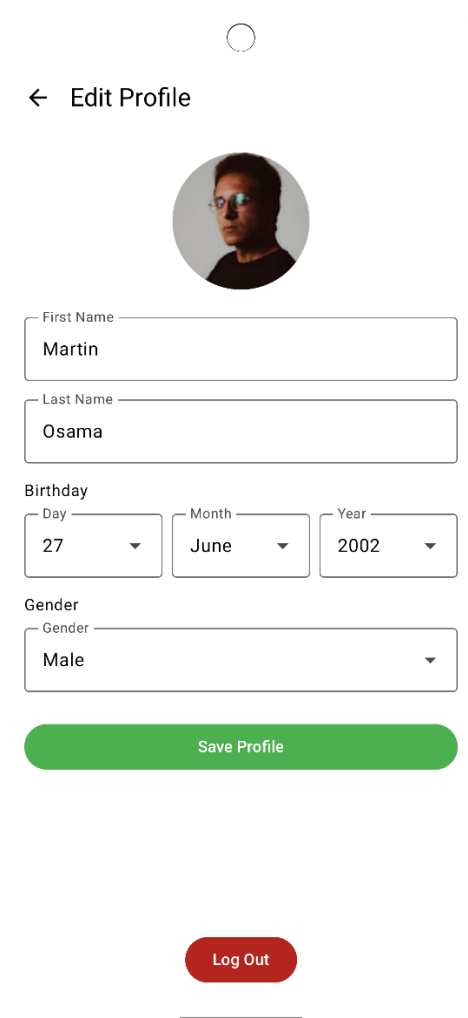
#### **3.3.4. Страница профиля пользователя (User Profile Page)**

Страница профиля пользователя ("Edit Profile") предоставляет возможность редактирования личных данных, таких как имя, фамилия, дата рождения и пол. Она также позволяет пользователю сохранять изменения и выходить из аккаунта.

Интерфейс страницы профиля:

На рисунке (см. Рисунок 13) показано оформление страницы профиля:





The image shows a user profile editing interface. At the top, there is a circular placeholder for a profile picture. Below it is a back arrow and the text "Edit Profile". The form contains several input fields: "First Name" with the value "Martin", "Last Name" with the value "Osama", and a "Birthday" section with three dropdowns for "Day" (27), "Month" (June), and "Year" (2002). There is also a "Gender" dropdown menu currently set to "Male". A green "Save Profile" button is positioned below the form fields. At the bottom of the page, there is a red "Log Out" button.

← Edit Profile

First Name  
Martin

Last Name  
Osama

Birthday

Day 27 ▼ Month June ▼ Year 2002 ▼

Gender  
Gender  
Male ▼

Save Profile

Log Out

Рисунок 13 - Страница профиля пользователя

Функциональность страницы профиля:

- Редактирование данных профиля:
  - Пользователь может изменить своё имя, фамилию, дату рождения и пол.
  - Изменения сохраняются после нажатия кнопки "Save Profile".
- Изменение аватара:

- Круглое изображение аватара является кликабельным элементом, позволяющим пользователю выбрать новое фото из галереи устройства или сделать снимок камеры.
- Сохранение изменений:
  - После внесения изменений данные отправляются на сервер через Retrofit API для обновления информации о пользователе.
- Выход из аккаунта:
  - Нажатие кнопки "Log Out" завершает сессию пользователя и очищает его данные из Firebase Authentication.

#### Техническая реализация:

- Firebase Authentication: Используется для аутентификации пользователя и получения его уникального идентификатора (uid).
- Jetpack Compose: Интерфейс создан с использованием Jetpack Compose, что обеспечивает гибкость и простоту в разработке UI.
- Material Design 3: Стили и компоненты соответствуют Material Design 3, гарантируя современный и согласованный внешний вид интерфейса.
- Выбор даты: Для выбора даты используется комбинация выпадающих списков для дня, месяца и года.
- Загрузка аватара: При изменении аватара изображение передается в Appwrite Storage для хранения, а URL аватара обновляется в базе данных.

#### Значение страницы профиля:

Страница профиля играет важную роль в приложении "Habit Tracker", так как она:

- Позволяет пользователям управлять своими личными данными.
- Обеспечивает удобство редактирования информации, такой как имя, фамилия, дата рождения и пол.
- Предоставляет возможность выхода из аккаунта, что важно для безопасности и управления доступом.

### **3.3.5. Страница статистики (Statistics Page)**

Страница статистики предоставляет пользователю возможность просматривать и анализировать прогресс выполнения привычек за определённый период времени. Она визуализирует данные в виде круговой диаграммы и списка выполненных/не выполненных привычек, что помогает пользователю понять свои достижения и области для улучшения.

Интерфейс страницы статистики:

На рисунке (см. Рисунок 14) показано оформление страницы статистики:

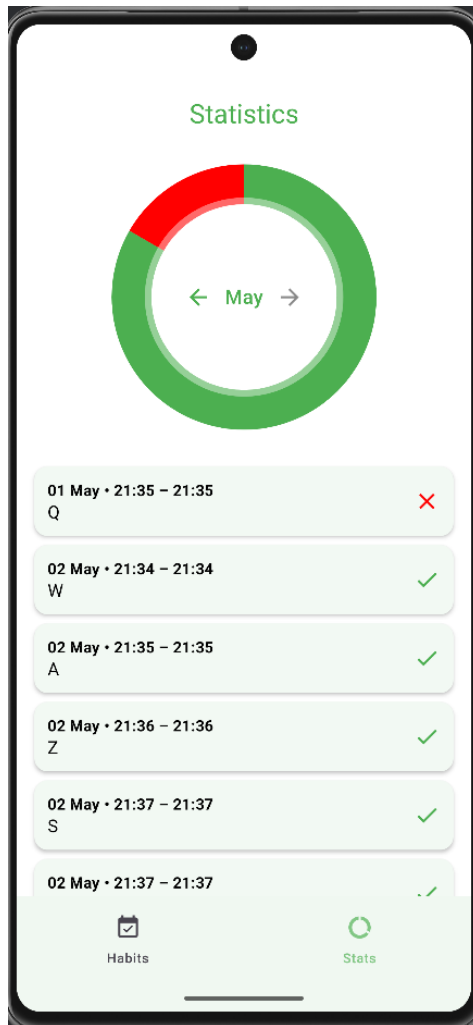


Рисунок 14 - Страница статистики

Функциональность страницы статистики:

- Визуализация прогресса:
  - Круговая диаграмма отображает процентное соотношение выполненных (✓) и не выполненных (✗) привычек за выбранный месяц.
- Переключение месяцев:
  - Пользователь может выбрать другой месяц для просмотра статистики, используя кнопки "<" и ">".

- Список привычек:
  - Показывает дату и время выполнения каждой привычки.
  - Отметки выполнения (✓ или X) для каждой привычки.
  - Возможность удалить запись о привычке через кнопку "X".
- Управление статистикой:
  - Пользователь может легко просматривать историю выполнения привычек за различные месяцы.

Техническая реализация:

- MPAndroidChart: Используется для создания круговой диаграммы и визуализации данных.
- Jetpack Compose: Интерфейс создан с использованием Jetpack Compose, что обеспечивает гибкость и простоту в разработке UI.
- Material Design 3: Стили и компоненты соответствуют Material Design 3, гарантируя современный и согласованный внешний вид интерфейса.
- Анимации: Кнопки и элементы управления поддерживают плавные анимации при взаимодействии пользователя.

Значение страницы статистики:

Страница статистики играет важную роль в приложении "Habit Tracker", так как она:

- Позволяет пользователям анализировать свой прогресс в формировании привычек.
- Предоставляет наглядное представление выполненных и не выполненных задач.

- Облегчает процесс мониторинга и планирования будущих действий.

### **3.3.6. Плавающее окно для добавления новой привычки (Floating Window for Adding a New Habit)**

Плавающее окно для добавления новой привычки ("New Habit") позволяет пользователю создать и настроить новые привычки прямо из главного экрана приложения. Окно открывается после нажатия кнопки "+" в нижнем правом углу главной страницы.

Интерфейс плавающего окна:

На рисунке (см. Рисунок 15) показано оформление плавающего окна:

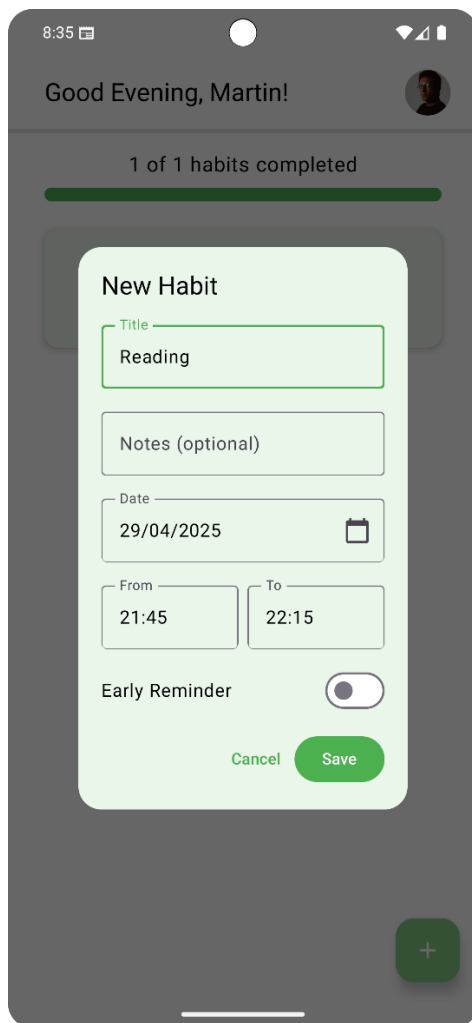


Рисунок 15 - Плавающее окно для добавления новой привычки

Функциональность плавающего окна:

- Добавление новой привычки:
  - Пользователь заполняет необходимые поля:
    - Название привычки.
    - Дополнительные примечания (по желанию).
    - Дату и время выполнения.
    - Флаг напоминания (если требуется).
- Сохранение данных:

- После нажатия кнопки "Save", данные отправляются на сервер через Retrofit API для сохранения новой привычки в базу данных.
- Если флажок напоминания активен, система устанавливает соответствующее напоминание через AlarmManager.
- Отмена добавления:
  - Нажатие кнопки "Cancel" закрывает диалоговое окно без сохранения изменений.

Техническая реализация:

- Jetpack Compose: Интерфейс создан с использованием Jetpack Compose, что обеспечивает гибкость и простоту в разработке UI.
- Material Design 3: Стили и компоненты соответствуют Material Design 3, гарантируя современный и согласованный внешний вид интерфейса.
- Анимации: Диалоговое окно появляется с анимацией появления из нижнего правого угла экрана.
- Календарь и выбор времени: Используются стандартные компоненты Android для выбора даты и времени.

Значение плавающего окна:

Плавающее окно для добавления новой привычки играет важную роль в приложении "Habit Tracker", так как оно:

- Предоставляет удобный способ создания новых привычек.
- Обеспечивает наглядное и интуитивно понятное управление параметрами привычки.



- Упрощает процесс настройки напоминаний и времени выполнения.

### **3.3.7. Добавление привычки (Habit Added)**

После успешного добавления новой привычки через плавающее окно, изменения отображаются на главной странице приложения. На этой странице пользователь может видеть список всех своих привычек, включая недавно добавленную.

Интерфейс после добавления привычки:

На рисунке (см. Рисунок 16) показано состояние главной страницы после добавления новой привычки:

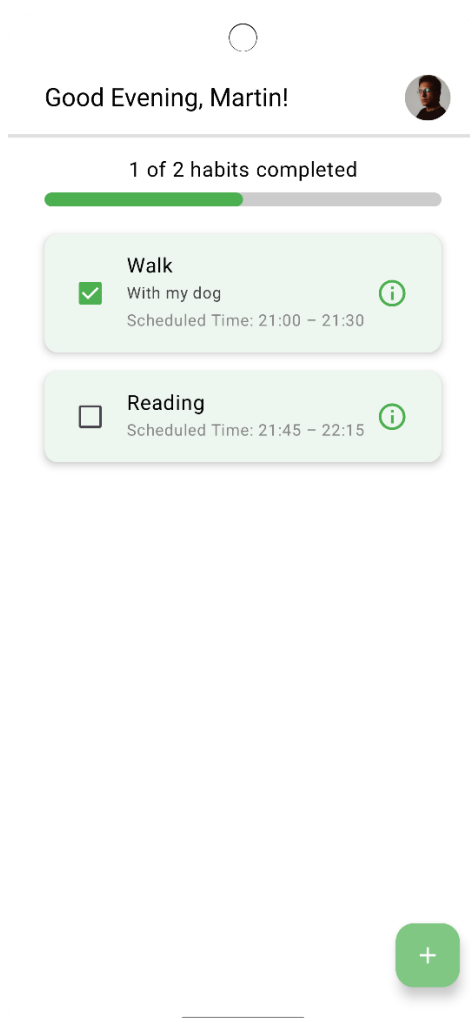


Рисунок 16 - Добавление привычки

Функциональность после добавления привычки:

- Отображение списка привычек:
  - После добавления новой привычки она появляется в списке на главной странице.
  - Для каждой привычки доступна возможность отметить её как выполненную или просмотреть дополнительную информацию.
- Обновление прогресса:

- Прогресс выполнения привычек обновляется автоматически после добавления новой привычки.
- Если привычка была выполнена, соответствующий флажок становится активным.
- Управление привычками:
  - Пользователь может редактировать или удалять существующие привычки через контекстное меню или дополнительные действия.

Техническая реализация:

- Jetpack Compose: Интерфейс создан с использованием Jetpack Compose, что обеспечивает гибкость и простоту в разработке UI.
- Material Design 3: Стили и компоненты соответствуют Material Design 3, гарантируя современный и согласованный внешний вид интерфейса.
- Анимации: Используются анимации для плавного отображения состояния привычек и переходов между экранами.
- Уведомления: Если пользователь не выполнил привычку в указанное время, система отправляет уведомление через AlarmManager.

Значение добавления привычки:

Добавление привычки играет важную роль в приложении "Habit Tracker", так как оно:

- Предоставляет пользователю возможность расширять список своих целей и задач.

- Обеспечивает наглядное отображение новых привычек в общем списке.
- Упрощает процесс управления и мониторинга выполнения задач.

### **3.3.8. Редактирование или удаление привычки (Edit or Delete Habit)**

Пользователь может редактировать или удалять существующие привычки через контекстное меню, которое открывается после нажатия на кнопку информации (i) рядом с выбранной привычкой. Это позволяет пользователю легко изменить параметры привычки или полностью удалить её из списка.

Интерфейс редактирования/удаления привычки:

На рисунке (см. Рисунок 17) показано оформление диалогового окна для выбора действия:

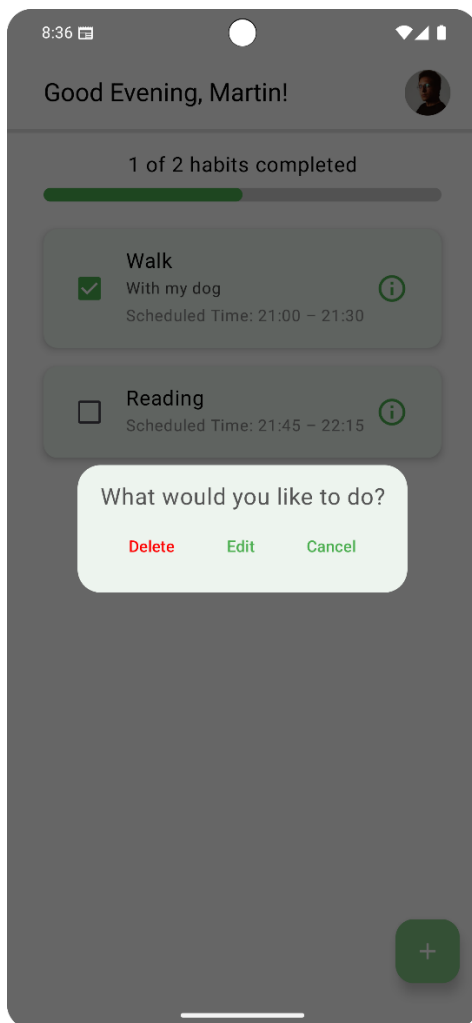


Рисунок 17 - Редактирование или удаление привычки

Функциональность редактирования/удаления привычки:

- Открытие диалогового окна:
  - После нажатия на кнопку информации (i) рядом с привычкой открывается диалоговое окно с вариантами действий.
- Удаление привычки:
  - Если пользователь выбирает опцию "Delete", система отправляет запрос на сервер для удаления привычки через Retrofit API.

- После успешного удаления привычка исчезает из списка на главной странице.
- Редактирование привычки:
  - Если пользователь выбирает опцию "Edit", открывается плавающее окно для редактирования параметров привычки (название, примечания, дата и время).
  - После внесения изменений пользователь может сохранить обновленные данные, которые будут отправлены на сервер через Retrofit API.
- Отмена действия:
  - Нажатие кнопки "Cancel" закрывает диалоговое окно без выполнения каких-либо действий.

#### Техническая реализация:

- Jetpack Compose: Интерфейс создан с использованием Jetpack Compose, что обеспечивает гибкость и простоту в разработке UI.
- Material Design 3: Стили и компоненты соответствуют Material Design 3, гарантируя современный и согласованный внешний вид интерфейса.
- Анимации: Диалоговое окно появляется с анимацией появления из центра экрана.
- Уведомления: При удалении привычки система автоматически удаляет связанные напоминания через AlarmManager.

#### Значение редактирования/удаления привычки:

Процесс редактирования или удаления привычки играет важную роль в приложении "Habit Tracker", так как он:

- Предоставляет пользователям возможность корректировать свои привычки в соответствии с текущими потребностями.
- Обеспечивает удобство управления списком привычек.
- Упрощает процесс очистки системы от ненужных или устаревших задач.

### **3.3.9. Редактирование привычки (Edit Habit)**

Пользователь может редактировать существующую привычку через плавающее окно "Edit Habit". Это позволяет изменить название, примечания и другие параметры выбранной привычки.

Интерфейс окна редактирования привычки:

На рисунке (см. Рисунок 18) показано оформление окна редактирования:

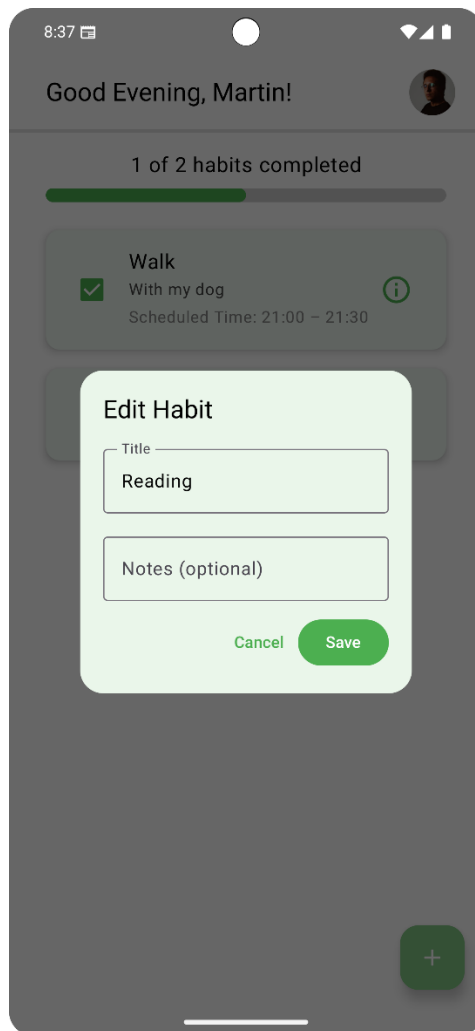


Рисунок 18 - Редактирование привычки

Функциональность окна редактирования:

- Изменение параметров привычки:
  - Пользователь может ввести новое название привычки в поле Title.
  - Дополнительные примечания можно добавить или изменить в поле Notes.
- Сохранение изменений:



- После нажатия кнопки "Save", данные отправляются на сервер через Retrofit API для обновления информации о привычке.
- Если изменения успешно сохранены, обновлённая информация отображается на главной странице.
- Отмена действий:
  - Нажатие кнопки "Cancel" закрывает диалоговое окно без сохранения изменений.

Техническая реализация:

- Jetpack Compose: Интерфейс создан с использованием Jetpack Compose, что обеспечивает гибкость и простоту в разработке UI.
- Material Design 3: Стили и компоненты соответствуют Material Design 3, гарантируя современный и согласованный внешний вид интерфейса.
- Анимации: Диалоговое окно появляется с анимацией появления из центра экрана.
- Уведомления: При успешном сохранении изменений система автоматически обновляет напоминания через AlarmManager.

Значение редактирования привычки:

Процесс редактирования привычки играет важную роль в приложении "Habit Tracker", так как он:

- Предоставляет пользователям возможность корректировать свои привычки в соответствии с текущими потребностями.
- Обеспечивает удобство управления списком привычек.
- Упрощает процесс изменения деталей уже существующих задач.

## 4. Тестирование

Тестирование является важным этапом разработки программного обеспечения, позволяющим убедиться в корректной работе всех модулей и функциональных возможностей приложения. В рамках данного проекта были проведены тесты серверной части, а также проверка пользовательского интерфейса на соответствие ожидаемому поведению.

### 4.1. Тестирование серверной части

Серверная часть приложения "Habit Tracker" реализована с использованием Ktor , PostgreSQL и Exposed ORM , что позволило создать надёжный и масштабируемый backend для обработки запросов от мобильного клиента.

Все операции взаимодействия между клиентом и сервером происходят через REST API, где данные передаются в формате JSON. Для тестирования серверной части были использованы такие инструменты, как Postman и JUnit-тесты (для unit-тестирования логики сервисов), а также логирование ошибок в Ktor.

#### 4.1.1. Пример тестирования: Логин

Одной из ключевых операций в системе является процесс авторизации пользователя. Ниже приведён пример тестирования этой функции на уровне сервера.

Цель тестирования:

Проверить корректность работы эндпоинта `/user/{firebaseUid}`, который возвращает данные пользователя по его Firebase UID.

Отправка GET-запроса:

GET <http://localhost:8080/user/tR9E0DtmGqalHTjewIheO5nm2Yq2>

Ожидаемый результат:

```
{  
  
  "id": " 1fb34c9b-7bd5-4b0b-8f37-38e31f274b16",  
  
  "firebase_uid": " tR9E0DtmGqalHTjewIheO5nm2Yq2",  
  
  "email": " martin.osama72@gmail.com",  
  
  "first_name": "Martin",  
  
  "last_name": "Osama",  
  
  "birth_date": "27/06/2002",  
  
  "gender": "Male",  
  
  "avatar_file_id":  
  "https://lh3.googleusercontent.com/a/ACg8ocLMAJ2actIayjJBlhQKufzKKmi-  
NSgzk7LdIOBAzCsE87WCZsx5=s96-c"  
  
}
```

#### **4.2. Тестирование пользовательского интерфейса**

Тестирование пользовательского интерфейса (UI-тестирование) направлено на проверку корректности работы графического интерфейса приложения, его реакции на действия пользователя и правильного отображения данных. В рамках проекта "Habit Tracker" были проведены тесты

на главной странице, экране профиля, экране добавления привычки и других ключевых экранах.

#### **4.2.1. Пример тестирования: Главная страница**

Цель тестирования:

Проверить корректность отображения и функциональности главной страницы приложения "Habit Tracker". В частности, убедиться, что:

- Пользователь видит список своих привычек после входа.
- Отметка выполненной привычки работает корректно.
- Навигация между экранами (например, к статистике или профилю) осуществляется без ошибок.

Шаги тестирования:

- Запуск приложения.
- Открытие главной страницы.
- Отметка привычки как выполненной.
- Добавление новой привычки.
- Редактирование привычки
- Удаление привычки
- Переход к статистике
- Переход к профилю

Ожидаемый результат:

- Главная страница загружается без ошибок.
- Список привычек отображается корректно.

- Отмечка выполнения, добавление, редактирование и удаление привычек работают правильно.
- Все элементы интерфейса отвечают требованиям UX/UI.
- Переходы между экранами происходят плавно и без перезагрузки.

## ЗАКЛЮЧЕНИЕ

В ходе выполнения бакалаврской работы была разработана система "Habit Tracker" — мобильное приложение для Android, позволяющее пользователям отслеживать и управлять своими привычками. Приложение предоставляет удобный интерфейс для добавления, редактирования, удаления и отметки выполненных привычек, а также визуализирует статистику их прогресса. Разработка была проведена с использованием современных технологий программной инженерии, таких как Kotlin , Jetpack Compose , Firebase Authentication , Retrofit , Appwrite Storage , MPAndroidChart и других.

Была реализована клиент-серверная архитектура, где серверная часть выполнена на базе Ktor и PostgreSQL , а клиентская — поддерживает сложную логику управления состояниями, обработки данных и взаимодействия с пользователем. Также были спроектированы диаграммы UML: вариантов использования, деятельности, классов, последовательности и ER-диаграмма, что позволило наглядно представить структуру системы и её компоненты.

Проведённый анализ аналогов показал, что существующие решения имеют ограничения в функциональности, стоимости или доступности. Разрабатываемое приложение "Habit Tracker" предлагает более гибкий и доступный подход к формированию полезных привычек, сочетая простоту использования с возможностью персонализации.

Тестирование всех модулей системы, включая авторизацию, работу с привычками и корректность отображения статистики, подтвердило работоспособность и надёжность программы. Пользовательский интерфейс соответствует принципам Material Design 3, обеспечивает высокую степень юзабилити и поддерживает темные и светлые режимы отображения.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Официальная документация Jetpack Compose

<https://developer.android.com/jetpack/compose>

Документация по Jetpack Compose — современному инструменту для создания пользовательского интерфейса на Android с декларативным подходом. Использовалась для реализации UI приложения.

2. Kotlin Coroutines

<https://kotlinlang.org/docs/coroutines-overview.html>

Официальная документация Kotlin Coroutines. Была использована для реализации асинхронной работы и управления потоками данных в приложении.

3. Retrofit – REST Client for Android and Java

<https://square.github.io/retrofit/>

Использовалась для выполнения HTTP-запросов к серверной части (Ktor API). Позволяет легко создавать клиентские запросы и обрабатывать ответы в формате JSON.

4. Firebase Authentication

<https://firebase.google.com/docs/auth>

Документация по Firebase Auth. Была использована для реализации системы авторизации через email/пароль и Google Sign-In.

5. Firebase Cloud Firestore

<https://firebase.google.com/docs/firestore>

Документация по Firebase Firestore. Использовалась для хранения и синхронизации данных пользователя между клиентским приложением и сервером.

6. MPAndroidChart – Android Chart Library

<https://github.com/PhilJay/MPAndroidChart>

Библиотека для визуализации статистики выполнения привычек в виде круговых диаграмм и графиков.

7. Appwrite – Open Source Backend Server

<https://appwrite.io>

Открытый backend-сервер, использованный для хранения изображений (аватаров пользователей).

8. Ktor – Kotlin Multiplatform Framework

<https://ktor.io>

Фреймворк, использованный для реализации REST API серверной части приложения. Позволил быстро создать серверную логику и взаимодействовать с базой данных PostgreSQL.

9. Exposed – Kotlin SQL Library

<https://github.com/JetBrains/Exposed>

ORM-библиотека Kotlin, использованная для работы с реляционной базой данных PostgreSQL.

10. Coil – Image Loading Library for Android

<https://coil-kt.github.io/coil/>

Библиотека для загрузки и отображения изображений в Jetpack Compose. Использовалась для отображения аватаров пользователей.

11. Material Design 3 (MD3)

<https://m3.material.io>

Руководство по Material Design 3. Использовалось для проектирования и оформления интерфейса мобильного приложения.

12. AlarmManager & NotificationManager – Android System Services

<https://developer.android.com/guide/components/alarm-manager>



<https://developer.android.com/guide/topics/ui/notifiers/notifications>

Системные службы Android, использованные для реализации уведомлений о времени выполнения привычек.

### 13. PostgreSQL – Relational Database Management System

<https://www.postgresql.org/docs/current/index.html>

Реляционная система управления базами данных, использованная как основное хранилище данных для серверной части приложения.

### 14. Docker – Containerization Platform

<https://www.docker.com/resources/what-container>

Инструмент контейнеризации, использованный для запуска серверной части приложения в изолированной среде.

### 15. Gson – JSON Parser for Java/Kotlin

<https://github.com/google/gson>

Библиотека для сериализации и десериализации объектов в JSON. Использовалась для обмена данными между клиентом и сервером.

### 16. Kotlin – Official Language for Android Development

<https://kotlinlang.org>

Язык программирования, использованный для разработки как клиентской, так и серверной частей приложения.

## ПРИЛОЖЕНИЕ А

```
package com.martinosama.habittracker

import android.content.Intent
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import android.widget.Toast
import androidx.activity.result.contract.ActivityResultContracts
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.LifecycleScope
import com.google.android.gms.auth.api.signin.GoogleSignIn
import com.google.android.gms.auth.api.signin.GoogleSignInAccount
import com.google.android.gms.auth.api.signin.GoogleSignInClient
import com.google.android.gms.auth.api.signin.GoogleSignInOptions
import com.google.android.gms.tasks.Task
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.GoogleAuthProvider
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.SetOptions
import com.martinosama.habittracker.backend.RetrofitInstance
import com.martinosama.habittracker.backend.UserRepository
import com.martinosama.habittracker.model.UserDto
import kotlinx.coroutines.launch

class SignInActivity : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth
    private lateinit var db: FirebaseFirestore
    private lateinit var googleSignInClient: GoogleSignInClient
    private val userRepository = UserRepository(RetrofitInstance.userApi)

    companion object {
        private const val RC_SIGN_IN = 100
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_sign_in)

        auth = FirebaseAuth.getInstance()
        db = FirebaseFirestore.getInstance()

        val gso = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken(getString(R.string.default_web_client_id))
            .requestEmail()
            .build()
        googleSignInClient = GoogleSignIn.getClient(this, gso)

        val emailInput = findViewById<EditText>(R.id.emailInput)
        val passwordInput = findViewById<EditText>(R.id.passwordInput)
        val signInButton = findViewById<Button>(R.id.signInButton)
        val signUpText = findViewById<TextView>(R.id.signUpText)
        val customGoogleSignInBtn = findViewById<View>(R.id.customGoogleSignInButton)

        signInButton.setOnClickListener {
            val email = emailInput.text.toString().trim()
            val password = passwordInput.text.toString().trim()

            if (email.isNotEmpty() && password.isNotEmpty()) {
                auth.signInWithEmailAndPassword(email, password)
                    .addOnCompleteListener { task ->
                        if (task.isSuccessful) {
                            val user = auth.currentUser
                            if (user != null) {
                                db.collection("users").document(user.uid)
                                    .get()
                                    .addOnSuccessListener { doc ->
                                        val profile = doc.toObject(UserProfile::class.java)
                                        val firstName = profile?.firstName?.takeIf { it.isNotBlank() }
                                            ?: "User"

                                        val intent = Intent(this, MainActivity::class.java)
                                        intent.putExtra("EXTRA_USER_NAME", firstName)
                                    }
                                }
                            }
                        }
                    }
            }
        }
    }
}
```

```

        startActivity(intent)
        finish()

        }.addOnFailureListener {
            val intent = Intent(this, MainActivity::class.java)
            intent.putExtra("EXTRA_USER_NAME", "User")
            startActivity(intent)
            finish()
        }
    } else {
        val intent = Intent(this, MainActivity::class.java)
        intent.putExtra("EXTRA_USER_NAME", "User")
        startActivity(intent)
        finish()
    }
} else {
    Toast.makeText(
        this,
        "Sign-in failed: ${task.exception?.message}",
        Toast.LENGTH_SHORT
    ).show()
}
} else {
    Toast.makeText(this, "Enter both email and password.", Toast.LENGTH_SHORT).show()
}
}

customGoogleSignInBtn.setOnClickListener {
    val signInIntent = googleSignInClient.signInIntent
    signInLauncher.launch(signInIntent)
}

signUpText.setOnClickListener {
    startActivity(Intent(this, SignUpActivity::class.java))
}
}

private val signInLauncher = registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
result ->
    if (result.resultCode == RESULT_OK) {
        val task = GoogleSignIn.getSignedInAccountFromIntent(result.data)
        if (task.isSuccessful) {
            firebaseAuthWithGoogle(task.result)
        } else {
            Toast.makeText(this, "Google Sign-In Failed", Toast.LENGTH_SHORT).show()
        }
    }
}

@Deprecated("Deprecated API, use ActivityResultLauncher instead")
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    if (requestCode == RC_SIGN_IN) {
        val task: Task<GoogleSignInAccount> = GoogleSignIn.getSignedInAccountFromIntent(data)
        if (task.isSuccessful) {
            val account = task.result
            firebaseAuthWithGoogle(account)
        } else {
            Toast.makeText(
                this,
                "Google Sign-In Failed: ${task.exception?.message}",
                Toast.LENGTH_SHORT
            ).show()
        }
    }
}

private fun firebaseAuthWithGoogle(account: GoogleSignInAccount?) {
    val credential = GoogleAuthProvider.getCredential(account?.idToken, null)
    auth.signInWithCredential(credential)
        .addOnCompleteListener { task ->
            if (task.isSuccessful) {
                val user = auth.currentUser
                if (user != null) {
                    val displayName = user.displayName ?: "User"
                    val photoUrl = user.photoUrl
                    val nameParts = displayName.split(" ")
                    val firstName = nameParts.getOrElse(0) { "" }

```



```

import com.google.firebase.auth.FirebaseAuth
import com.martinosama.habittracker.backend.RetrofitInstance
import com.martinosama.habittracker.backend.UserRepository
import com.martinosama.habittracker.model.UserDto
import kotlinx.coroutines.launch

class SignUpActivity : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_sign_up)

        auth = FirebaseAuth.getInstance()

        val emailInput = findViewById<EditText>(R.id.emailInput)
        val passwordInput = findViewById<EditText>(R.id.passwordInput)
        val signUpButton = findViewById<Button>(R.id.signUpButton)
        val signInText = findViewById<TextView>(R.id.signInText)

        signUpButton.setOnClickListener {
            val email = emailInput.text.toString().trim()
            val password = passwordInput.text.toString().trim()

            if (email.isNotEmpty() && password.isNotEmpty()) {
                auth.createUserWithEmailAndPassword(email, password).addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        Toast.makeText(this, "Account created successfully!", Toast.LENGTH_SHORT).show()
                        val user = auth.currentUser
                        val userDto = UserDto(
                            firstName = "",
                            lastName = "",
                            birthday = "",
                            gender = "",
                            profileImageUrl = null
                        )

                        val userRepository = UserRepository(RetrofitInstance.userApi)
                        lifecycleScope.launch {
                            try {
                                userRepository.createUser(user?.uid ?: "", user?.email ?: "", userDto)
                            } catch (e: Exception) {
                                e.printStackTrace()
                                Toast.makeText(this@SignUpActivity, "Failed to sync with backend",
                                    Toast.LENGTH_SHORT).show()
                            }
                        }
                        startActivity(Intent(this, MainActivity::class.java))
                        finish()
                    } else {
                        Toast.makeText(this, "Sign-up failed: ${task.exception?.message}",
                            Toast.LENGTH_SHORT).show()
                    }
                }
            } else {
                Toast.makeText(this, "Enter both email and password.", Toast.LENGTH_SHORT).show()
            }
        }

        signInText.setOnClickListener {
            val intent = Intent(this, SignInActivity::class.java)
            startActivity(intent)
            finish()
        }
    }
}

```

## ПРИЛОЖЕНИЕ Б

```
package com.martinosama.habittracker.backend.controller

import com.martinosama.habittracker.backend.dto.HabitDto
import com.martinosama.habittracker.backend.mapper.toDto
import com.martinosama.habittracker.backend.respondException
import com.martinosama.habittracker.backend.service.HabitService
import io.ktor.http.*
import io.ktor.server.application.*
import io.ktor.server.request.*
import io.ktor.server.response.*
import io.ktor.server.routing.*
import java.util.UUID

fun Route.habitRoutes(habitService: HabitService) {

    route("/habits") {

        get("/{firebaseUid}") {
            val uid = call.parameters["firebaseUid"]
            if (uid == null) {
                call.respond(HttpStatusCode.BadRequest, "Missing UID")
                return@get
            }

            try {
                val habits = habitService.getHabitsByUser(uid)
                call.respond(habits.map { it.toDto() })
            } catch (e: Exception) {
                call.respondException(e)
            }
        }

        get("/one/{habitId}") {
            val habitIdParam = call.parameters["habitId"]
            if (habitIdParam == null) {
                call.respond(HttpStatusCode.BadRequest, "Missing habit ID")
                return@get
            }

            val habitId = try {
                UUID.fromString(habitIdParam)
            } catch (e: IllegalArgumentException) {
                call.respond(HttpStatusCode.BadRequest, "Invalid UUID format")
                return@get
            }

            val habit = habitService.getHabitById(habitId)
            if (habit != null) {
                call.respond(habit.toDto())
            } else {
                call.respond(HttpStatusCode.NotFound, "Habit not found")
            }
        }

        post("/{firebaseUid}") {
            val uid = call.parameters["firebaseUid"]
            if (uid == null) {
                call.respond(HttpStatusCode.BadRequest, "Missing UID")
                return@post
            }

            val dto = call.receive<HabitDto>().copy(id = UUID.randomUUID().toString())
            val createdHabit = habitService.createHabit(uid, dto)
            call.respond(HttpStatusCode.Created, createdHabit)
        }

        put("/{habitId}") {
            val habitIdParam = call.parameters["habitId"]
            if (habitIdParam == null) {
                call.respond(HttpStatusCode.BadRequest, "Missing habit ID")
                return@put
            }

            val habitId = try {
                UUID.fromString(habitIdParam)
            }
```

```

    } catch (e: IllegalArgumentException) {
        call.respond(HttpStatusCode.BadRequest, "Invalid UUID format")
        return@put
    }

    val dto = call.receive<HabitDto>()
    val updated = habitService.updateHabit(habitId, dto)
    if (updated) {
        call.respond(HttpStatusCode.OK, "Habit updated")
    } else {
        call.respond(HttpStatusCode.NotFound, "Habit not found")
    }
}

delete("/{habitId}") {
    val habitIdParam = call.parameters["habitId"]
    if (habitIdParam == null) {
        call.respond(HttpStatusCode.BadRequest, "Missing habit ID")
        return@delete
    }

    val habitId = try {
        UUID.fromString(habitIdParam)
    } catch (e: IllegalArgumentException) {
        call.respond(HttpStatusCode.BadRequest, "Invalid UUID format")
        return@delete
    }

    val deleted = habitService.deleteHabit(habitId)
    if (deleted) {
        call.respond(HttpStatusCode.OK, "Habit deleted")
    } else {
        call.respond(HttpStatusCode.NotFound, "Habit not found")
    }
}
}

}

package com.martinosama.habittracker.backend.mapper

import com.martinosama.habittracker.backend.dto.HabitDto
import com.martinosama.habittracker.backend.entity.Habits
import com.martinosama.habittracker.backend.model.HabitModel
import org.jetbrains.exposed.sql.ResultRow
import java.time.format.DateTimeFormatter

fun ResultRow.toHabitModel() = HabitModel(
    id = this[Habits.id].value,
    userId = this[Habits.userId],
    title = this[Habits.title],
    notes = this[Habits.notes],
    date = this[Habits.date],
    fromTime = this[Habits.fromTime],
    toTime = this[Habits.toTime],
    reminder = this[Habits.reminder],
    isCompleted = this[Habits.isCompleted]
)

fun HabitModel.toDto(): HabitDto {
    val dateFormatter = DateTimeFormatter.ofPattern("dd/MM/yyyy")
    val timeFormatter = DateTimeFormatter.ofPattern("HH:mm")
    return HabitDto(
        id = this.id.toString(),
        title = this.title,
        notes = this.notes,
        date = this.date.format(dateFormatter),
        fromTime = this.fromTime.format(timeFormatter),
        toTime = this.toTime.format(timeFormatter),
        reminder = this.reminder,
        isCompleted = this.isCompleted
    )
}

package com.martinosama.habittracker.backend.repository

import com.martinosama.habittracker.backend.dto.HabitDto
import com.martinosama.habittracker.backend.entity.Habits
import com.martinosama.habittracker.backend.mapper.toHabitModel
import com.martinosama.habittracker.backend.model.HabitModel
import org.jetbrains.exposed.sql.*
import org.jetbrains.exposed.sql.SqlExpressionBuilder.eq

```

```

import org.jetbrains.exposed.sql.transactions.transaction
import org.slf4j.LoggerFactory
import java.time.LocalDate
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter
import java.util.*

class HabitRepository {
    private val logger = LoggerFactory.getLogger(HabitRepository::class.java)

    fun getHabitsForUser(userId: String): List<HabitModel> = transaction {
        logger.info("Querying habits for user: $userId")
        try {
            val result = Habits.select { Habits.userId eq userId }
                .orderBy(Habits.date to SortOrder.ASC, Habits.fromTime to SortOrder.ASC) // Correct syntax
                .map { it.toHabitModel() }
            logger.info("Query result for user $userId: $result")
            result
        } catch (e: Exception) {
            logger.error("Error querying habits for user $userId: ${e.message}", e)
            throw e
        }
    }

    fun getHabitById(id: UUID): HabitModel? = transaction {
        logger.info("Querying habit with ID: $id")
        try {
            val result = Habits.select { Habits.id eq id }
                .map { it.toHabitModel() }
                .singleOrNull()
            logger.info("Query result for habit $id: $result")
            result
        } catch (e: Exception) {
            logger.error("Error querying habit with ID $id: ${e.message}", e)
            throw e
        }
    }

    fun createHabit(userId: String, dto: HabitDto): HabitModel = transaction {
        logger.info("Inserting habit for user: $userId, dto: $dto")
        try {
            val dateFormatter = DateTimeFormatter.ofPattern("dd/MM/yyyy")
            val timeFormatter = DateTimeFormatter.ofPattern("HH:mm")
            val insert = Habits.insert {
                it[Habits.userId] = userId
                it[title] = dto.title
                it[notes] = dto.notes
                it[date] = LocalDate.parse(dto.date, dateFormatter)
                it[fromTime] = LocalTime.parse(dto.fromTime, timeFormatter)
                it[toTime] = LocalTime.parse(dto.toTime, timeFormatter)
                it[reminder] = dto.reminder
                it[isCompleted] = dto.isCompleted
            }
            val result = insert.resultedValues?.firstOrNull()?.toHabitModel()!!
            logger.info("Inserted habit: $result")
            result
        } catch (e: Exception) {
            logger.error("Error inserting habit for user $userId: ${e.message}", e)
            throw e
        }
    }

    fun updateHabit(id: UUID, dto: HabitDto): Boolean = transaction {
        logger.info("Updating habit with ID: $id, dto: $dto")
        try {
            val dateFormatter = DateTimeFormatter.ofPattern("dd/MM/yyyy")
            val timeFormatter = DateTimeFormatter.ofPattern("HH:mm")
            val updated = Habits.update({ Habits.id eq id }) {
                it[title] = dto.title
                it[notes] = dto.notes
                it[date] = LocalDate.parse(dto.date, dateFormatter)
                it[fromTime] = LocalTime.parse(dto.fromTime, timeFormatter)
                it[toTime] = LocalTime.parse(dto.toTime, timeFormatter)
                it[reminder] = dto.reminder
                it[isCompleted] = dto.isCompleted
            }
            logger.info("Update result for habit $id: $updated")
            updated > 0
        } catch (e: Exception) {
            logger.error("Error updating habit with ID $id: ${e.message}", e)
        }
    }
}

```



```

        throw e
    }
}

fun deleteHabit(id: UUID): Boolean = transaction {
    logger.info("Deleting habit with ID: $id")
    try {
        val deleted = Habits.deleteWhere { Habits.id eq id }
        logger.info("Delete result for habit $id: $deleted")
        deleted > 0
    } catch (e: Exception) {
        logger.error("Error deleting habit with ID $id: ${e.message}", e)
        throw e
    }
}
}

package com.martinosama.habittracker.backend.service

import com.martinosama.habittracker.backend.dto.HabitDto
import com.martinosama.habittracker.backend.model.HabitModel
import com.martinosama.habittracker.backend.repository.HabitRepository
import java.util.*
import org.slf4j.LoggerFactory

class HabitService(private val habitRepository: HabitRepository) {
    private val logger = LoggerFactory.getLogger(HabitService::class.java)

    fun getHabitById(id: UUID): HabitModel? {
        logger.info("Fetching habit with ID: $id")
        return habitRepository.getHabitById(id).also {
            logger.info("Habit fetched: $it")
        }
    }

    fun getHabitsByUser(userId: String): List<HabitModel> {
        logger.info("Fetching habits for user: $userId")
        return habitRepository.getHabitsForUser(userId).also {
            logger.info("Habits fetched for user $userId: $it")
        }
    }

    fun createHabit(userId: String, dto: HabitDto): HabitModel {
        logger.info("Creating habit for user: $userId, dto: $dto")
        return habitRepository.createHabit(userId, dto).also {
            logger.info("Habit created: $it")
        }
    }

    fun updateHabit(id: UUID, dto: HabitDto): Boolean {
        logger.info("Updating habit with ID: $id, dto: $dto")
        return habitRepository.updateHabit(id, dto).also {
            logger.info("Habit update result: $it")
        }
    }

    fun deleteHabit(id: UUID): Boolean {
        logger.info("Deleting habit with ID: $id")
        return habitRepository.deleteHabit(id).also {
            logger.info("Habit deletion result: $id")
        }
    }
}

```