**СОДЕРЖАНИЕ**

[Введение 4](#_Toc201218171)

[1 Теоретическая часть 6](#_Toc201218172)

[1.1 Инструменты разработки 6](#_Toc201218173)

[1.2 Основы прототипирования мобильных приложений в Figma 9](#_Toc201218174)

[1.3 Создание базы данных и API в Supabase 11](#_Toc201218175)

[1.4 Интерфейс пользователя 15](#_Toc201218176)

[1.5 Система контроля версий 17](#_Toc201218177)

[2 Системный анализ 20](#_Toc201218178)

[2.1 Системный анализ бизнес-процесса. 20](#_Toc201218179)

[2.2 Анализ рынка 20](#_Toc201218180)

[2.3 Диаграмма ERD 23](#_Toc201218181)

[2.4 Диаграмма Прецедентов 25](#_Toc201218182)

[2.5 Диаграмма деятельности 27](#_Toc201218183)

[2.6 Диаграмма последовательности 30](#_Toc201218184)

[3 Практическая часть 32](#_Toc201218185)

[3.1 Прототипирование приложения 32](#_Toc201218186)

[3.2 Разработка приложения. 35](#_Toc201218187)

[Заключение 48](#_Toc201218188)

[Список использованных источников 49](#_Toc201218189)

[Приложение А Основные конечные точки API 52](#_Toc201218190)

[Приложение Б Листинг кода. Класс SupabaseHelper. 57](#_Toc201218191)

[Приложение В Листинг кода. Объект TaskRepository. 77](#_Toc201218192)

Введение

В условиях современного мира, где время — самый ценный ресурс, человек стремится к максимальной организованности и эффективности в управлении своим временем, задачами и ресурсами.

Мобильные приложения становятся незаменимыми помощниками в повседневной жизни, предоставляя пользователям инструменты для структурирования планов, отслеживания прогресса в достижении целей и контроля над различными аспектами жизни, включая здоровье, финансы и саморазвитие. Однако, несмотря на разнообразие существующих приложений, многие из них не предлагают комплексного решения, охватывающего все важные сферы жизни пользователя. Поэтому создание многофункционального мобильного ежедневника, который объединит в себе множество полезных функций, является актуальной и востребованной задачей.

В рамках данного дипломного проекта поставлена следующая цель: создать мобильное приложение ежедневник, обладающий широким функционалом. Приложение будет включать в себя следующие модули:

* расписание дня с возможностью добавления задач, дневника благодарности и радости, заметок, отслеживания настроения и самочувствия;
* раздел для постановки целей и отслеживания привычек;
* блок для хранения идей и напоминаний о событиях;
* систему учета доходов и расходов;
* модуль контроля здоровья с подсчетом шагов, трекером питания и приема воды, трекером лекарств, а также измерениями веса и объемов тела;
* раздел для фиксации хобби;
* библиотеку статей с рекомендациями по различным темам; профиль пользователя с настройками интерфейса;
* аналитическую панель и напоминание о проверке необходимых вещей перед выходом из дома.

Для достижения цели проекта необходимо решить следующие задачи:

* провести анализ существующих мобильных приложений для управления временем и личными ресурсами, выявить их преимущества и недостатки;
* разработать требования к функциональности и пользовательскому интерфейсу приложения;
* спроектировать архитектуру приложения, включая схему данных и связи между ними;
* разработать прототип интерфейса приложения;
* реализовать основные функциональные модули приложения;
* провести тестирование приложения для обеспечения его стабильной работы и удобства использования;
* оценить эффективность разработанного решения и предложить возможные направления для дальнейшего развития.

1. Теоретическая часть

## Инструменты разработки

Для реализации дипломного проекта была выбрана платформа Android как наиболее распространённая мобильная операционная система с открытым исходным кодом, обладающая широкими возможностями кастомизации и глубокой интеграцией с современными технологиями.

1. Операционная система. Android — операционная система с открытым исходным кодом на базе ядра Linux, разработанная Google для смартфонов, планшетов и других устройств. Её архитектура состоит из нескольких ключевых слоёв:

* Ядро Linux — центральная часть операционной системы, взаимодействующая с базовым оборудованием.
* Собственные демоны и библиотеки — компоненты, взаимодействующие напрямую с ядром операционной системы.
* Уровень аппаратной абстракции (Hardware Abstraction Layer, HAL) — стандартный интерфейс для взаимодействия с аппаратными драйверами.
* Среда выполнения Android (Android Runtime, ART) — среда выполнения Java, преобразующая байт-код в машинный код.
* Системные службы — модульные компоненты для взаимодействия с оборудованием.
* Системные API — API, доступные только для партнеров и OEM-производителей. Эти API используются для разработки привилегированных приложений, которые должны быть предварительно установлены на устройство.
* Android API — это общедоступный интерфейс для разработки приложений с использованием стандартных инструментов.

Разработка под Android ведётся на Kotlin и Java, обеспечивая высокую производительность и интеграцию с операционной системой. Альтернативы, такие как Flutter и React Native, имеют ограничения в производительности и доступе к нативным API.

Жизненный цикл приложения управляется моделью Activity с состояниями от создания (onCreate) до уничтожения (onDestroy). Безопасность достигается через разрешения, sandbox и шифрование.

В перечень основных инструментов разработки входят интегрированная среда разработки Android Studio, платформа Firebase и утилита ADB.

Среди важных направлений развития Android можно отметить внедрение передовых технологий, таких как Jetpack Compose для декларативного дизайна пользовательского интерфейса, Kotlin Multiplatform для разработки приложений, работающих на разных платформах, и улучшение поддержки складных устройств. Хотя Android обладает преимуществами открытой экосистемы, платформа все еще сталкивается с проблемами, такими как разнообразие устройств и различия в версиях операционной системы.

1. Язык программирования. Kotlin — современный статически типизированный язык программирования, разработанный компанией JetBrains в 2011 году как альтернатива Java. В 2017 году Google объявила его приоритетным для разработки Android-приложений, способствуя его популярности среди мобильных разработчиков.

Основные преимущества Kotlin включают:

* Лаконичность — сокращение объёма кода и повышение его читаемости.
* Null-безопасность — система типов, которая минимизирует риски возникновения ошибки NullPointerException, улучшая стабильность приложений.
* Полная совместимость с Java — возможность работы с Java-библиотеками и фреймворками, позволяя плавно переходить на Kotlin без полной переработки кода.
* Современные языковые конструкции — поддержка extension-функций, data-классов, лямбда-выражений и других возможностей, которая делает код гибким и выразительным.
* Корутины — встроенная поддержка асинхронного программирования, упрощающая работу с фоновыми задачами и сетевыми запросами.
* Мультиплатформенная разработка — общий код для Android, iOS, веба и бэкенда (Kotlin Multiplatform Mobile, Ktor).

Kotlin применяется в мобильной разработке для Android, бэкенд-разработке (Ktor, Spring Boot), кроссплатформенных решениях (Kotlin Multiplatform Mobile), веб-разработке (Kotlin/JS) и Data Science.

Благодаря своим характеристикам, он стал востребованным языком программирования, предоставляя разработчикам мощные инструменты для создания качественного программного обеспечения.

1. Среда разработки. Android Studio — интегрированная среда разработки (Integrated Development Environment, IDE), созданная Google совместно с JetBrains. Она основана на IntelliJ IDEA Community Edition и предназначена для разработки приложений под Android. Впервые представленная в 2013 году, Android Studio заменила Eclipse ADT, интегрировав мощные возможности IntelliJ Idea в среду для Android-разработки.

Ключевые особенности Android Studio включают:

* Несколько вариантов запуска приложений: на реальном устройстве через USB-отладку, на эмуляторе, управляемом через AVD Manager, и в облаке с использованием Firebase Test Lab.
* Поддержка сервисов Google Cloud и Firebase.
* Инструменты для профилирования, позволяющие анализировать использование процессора, памяти, батареи и сетевого трафика.
* Готовые шаблоны проектов для быстрого старта разработки (например, Activity, Fragment).
* Гибкая система сборки с использованием Gradle.
* Интеграция с Git и другими системами контроля версий.
* Функция принятия изменений (Apply Changes) для мгновенного обновления кода без перестройки проекта.

Android Studio также поддерживает современные технологии, такие как Jetpack Compose для создания пользовательских интерфейсов, Android Jetpack для упрощения разработки и ML Kit для интеграции искусственного интеллекта.

Среда предоставляет обширную документацию, включающую руководства, API-справочники, практические уроки (Codelabs) и поддержку на Stack Overflow.

Таким образом, Android Studio является мощным и удобным инструментом для Android-разработки, сочетающим широкий функционал, современные технологии и инструменты для анализа производительности, что делает её отличным выбором для проектов любой сложности.

## Основы прототипирования мобильных приложений в Figma

Прототипирование — это процесс создания интерактивных макетов будущего приложения, имитирующих его функционал и пользовательский интерфейс.

Прототипирование позволяет визуализировать продукт, протестировать его удобство и согласовать концепцию с командой и заказчиком. Интерактивные прототипы помогают выявить ошибки на ранних этапах, сократить затраты времени и ресурсов.

Кроме того, качественный прототип улучшает пользовательский опыт (User eXperience, UX), обеспечивая интуитивную навигацию, логичное расположение элементов и адаптивный интерфейс.

Для создания прототипов используются различные инструменты, среди которых Figma занимает особое место благодаря своей гибкости, удобству совместной работы и мощным возможностям для дизайна интерфейсов. Ниже представлены основные виды прототипов, которые можно создавать в Figma:

* Статичные (Wireframes) – схематичные черно-белые макеты, отображающие структуру экранов и расположение ключевых элементов.
* Интерактивные (Clickable Prototypes) – связанные между собой экраны, имитирующие навигацию в приложении (например, переходы по кнопкам).
* Высокодетализированные (High-Fidelity) – прототипы, максимально приближенные к финальному дизайну, с анимациями, реальным контентом и продуманными интерфейсными решениями.

Figma предоставляет широкие возможности для дизайна и прототипирования, включая:

* Совместную работу в режиме реального времени.
* Создание компонентов и стилей для единообразия интерфейса.
* Интеграцию с популярными дизайн-системами, такими как Material Design.

Среди преимуществ Figma для мобильного дизайна можно выделить:

* Готовые шаблоны под iOS и Android.
* Тестирование прототипов на реальных устройствах через Figma Mirror.
* Автоматическую адаптацию под разные разрешения экранов с помощью Constraints и Auto Layout.

Прототипирование в Figma — это важный этап проектирования, который помогает создать удобный, функциональный и эстетически продуманный интерфейс.

## Создание базы данных и API в Supabase

1. Базы данных. База данных (БД) — это организованная система для хранения и управления информацией, обеспечивающая эффективное взаимодействие с большими объемами данных. Она поддерживает быстрое сохранение, извлечение, обновление и удаление данных.

В мобильных приложениях, работающих с обширными массивами информации, базы данных играют ключевую роль. В зависимости от организации данных, базы данных классифицируются на два основных типа:

* Реляционные (SQL) – данные хранятся в таблицах, связанных между собой (PostgreSQL, MySQL).
* Нереляционные (NoSQL) – данные организуются в виде документов, ключ-значение или графов (MongoDB, Redis).

В мобильных приложениях чаще всего используются реляционные базы данных, так как они обеспечивают целостность данных и удобны для сложных запросов.

Ключевые понятия реляционных баз данных включают:

* Таблица – структура для хранения данных (например, users, products).
* Строка (запись) – отдельный элемент данных в таблице.
* Столбец (поле) – атрибут записи (например, id, name, email).
* Первичный ключ (Primary Key, PK) – уникальный идентификатор записи.
* Внешний ключ (Foreign Key, FK) – связь между таблицами.
* Индексы – ускоряют поиск данных.

1. API. Программный интерфейс приложения (Application Programming Interface, API) – это программный интерфейс, обеспечивающий взаимодействие между различными программными компонентами. Он действует как универсальный "переводчик", позволяя обмениваться данными и функциональностью между приложениями и сервисами. В мобильной разработке API используется для получения актуальных данных с серверов, отправки пользовательской информации для обработки и интеграции с внешними сервисами, такими как платежные системы, аналитические платформы и социальные сети.

Наиболее распространенным типом API является REST API, основанный на клиент-серверной архитектуре и использующий стандартные HTTP-запросы. Его популярность объясняется простотой, масштабируемостью и следующими ключевыми принципами:

* Клиент-серверная архитектура — разделена на клиентскую (интерфейс) и серверную (обработка данных) части, что позволяет им развиваться независимо.
* Stateless (отсутствие состояния на сервере между запросами) — каждый запрос содержит всю необходимую информацию для обработки, а сервер не сохраняет данные о предыдущих запросах.
* Кеширование — благодаря HTTP-заголовкам ответы могут кешироваться, что ускоряет работу приложений.
* Единообразие интерфейса — стандартизированные URL-адреса, HTTP-методы (GET, POST, PUT, PATCH, DELETE) и форматы данных (JSON/XML).
* Слоистая система — поддержка промежуточных уровней (балансировщики нагрузки, прокси-серверы), повышающая гибкость и надежность системы.

REST API базируется на нескольких фундаментальных компонентах:

1. Ресурсы и их идентификация

Ресурсы представляют сущности системы (пользователи, товары, заказы и т.д.) и имеют уникальные идентификаторы, выраженные в URI (Uniform Resource Identifier). Например, https://api.example.com/users/123 указывает на конкретного пользователя с идентификатором 123.

1. Методы взаимодействия

Для работы с ресурсами используются стандартные HTTP-методы:

* GET — для получения информации о ресурсе;
* POST — для создания новых ресурсов;
* PUT — для полного обновления существующих ресурсов;
* PATCH — для частичного обновления существующих ресурсов;
* DELETE — для удаления ресурсов.

1. Форматы данных

Обмен данными между клиентом и сервером осуществляется в структурированных форматах данных. Наиболее распространен JSON благодаря своей простоте и удобочитаемости, хотя в некоторых случаях может использоваться XML.

1. Заголовки и метаданные

HTTP-заголовки играют важную роль в работе API, передавая дополнительную информацию:

* Content-Type — определяет формат передаваемых данных;
* Authorization — содержит учетные данные для аутентификации;
* Cache-Control — управляет кешированием ответов.

1. Коды состояния

Сервер использует стандартные HTTP-коды для информирования клиента о результате обработки запроса:

* 2xx — успешное выполнение (200 OK, 201 Created);
* 4xx — ошибки клиента (400 Bad Request, 404 Not Found);
* 5xx — серверные ошибки (500 Internal Server Error).

Использование REST API в мобильных приложениях обеспечивает возможность получения актуальных данных, отправки пользовательских данных и интеграции с внешними сервисами (платежи, аналитика, соцсети), что повышает функциональность и удобство использования приложений

1. Supabase. Supabase — это открытая платформа для разработки серверной части приложений, основанная на PostgreSQL. Она предоставляет широкий спектр инструментов и сервисов, которые позволяют разработчикам быстро создавать и масштабировать приложения без необходимости ручной настройки и обслуживания сервера.

Supabase включает в себя следующие компоненты:

* База данных PostgreSQL с поддержкой расширений.

PostgreSQL — мощная реляционная СУБД с поддержкой JSONB для работы с NoSQL-структурами.

Доступны расширения, такие как PostGIS для геоданных, pg\_cron для планирования задач и uuid-ossp для генерации уникальных идентификаторов. PostgreSQL также поддерживает Row-Level Security (RLS) для обеспечения безопасности данных на уровне строк.

* Автоматическое создание RESTful API.

Supabase использует PostgREST для генерации RESTful API на основе структуры базы данных. Это позволяет разработчикам создавать API без написания дополнительного кода, что ускоряет разработку.

* Система аутентификации.

Аутентификация реализована через микросервис GoTrue, поддерживающий email/пароль, интеграцию с OAuth-провайдерами (Google, GitHub, Apple и другие), использование Magic Links и генерацию JWT-токенов для безопасного доступа к API.

* Realtime API.

Поддержка подписок на изменения в базе данных осуществляется через WebSockets, обеспечивая актуальность данных в реальном времени.

* Хранилище файлов.

Supabase предоставляет S3-совместимое хранилище файлов с интеграцией сетей доставки контента (Content Delivery Network, CDN) для ускорения загрузки. Управление правами доступа осуществляется через RLS.

* Edge Functions.

Выполняются на базе Deno (TypeScript) и работают на edge-сетях, что позволяет размещать логику ближе к пользователям и минимизировать задержки.

* Клиентские наборы средств разработки.

Доступны наборы средств разработки (oftware Development Kit, SDK) для JavaScript/TypeScript, Flutter, Python, Swift и Kotlin, упрощая интеграцию Supabase в различные проекты.

* Панель управления.

Интуитивно понятный интерфейс для управления проектами, настройки базы данных, аутентификации и других компонентов.

Supabase сочетает функциональность PostgreSQL, простоту использования Firebase и преимущества открытого исходного кода, что делает её отличным выбором для стартапов, веб- и мобильных приложений, требующих масштабируемого, настраиваемого и готового к использованию бэкенда.

## Интерфейс пользователя

Создание пользовательского интерфейса (User Interface, UI) является ключевым этапом разработки Android-приложений. В экосистеме Android существует два принципиально различных подхода к построению UI:

* Традиционный XML-подход (View-based система).
* Современный декларативный подход (Jetpack Compose).

Расширяемый язык разметки (eXtensible Markup Language, XML) используется для декларативного описания пользовательских интерфейсов в Android. Этот подход основывается на следующих принципах:

* Разделение логики и представления: логика приложения реализуется на языках программирования Kotlin или Java, а структура интерфейса описывается в XML.
* Иерархия View и ViewGroup: для создания сложных интерфейсов применяется иерархия элементов View и ViewGroup.
* Поддержка конфигураций: XML позволяет создавать альтернативные ресурсы для различных конфигураций устройств.

Пример файла макета, такого как activity\_main.xml, представлен на рисунке 1.

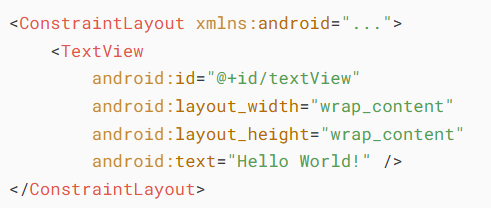


Рисунок 1 – Пример файла макета на xml

Преимущества XML-подхода включает:

* Совместимость: полная поддержка всех версий Android.
* Зрелость: богатая экосистема библиотек и инструментов.
* Визуальный редактор: удобный инструмент предварительного просмотра в Android Studio.
* Разделение ответственностей: четкое разделение кода и разметки.

Однако этот подход имеет и ограничения:

* Избыточность: необходимость большого объема шаблонного кода.
* Сложность динамических интерфейсов: трудности при программном изменении структуры.
* Производительность: потенциальные проблемы с глубокими иерархиями представлений.

Jetpack Compose — современный декларативный UI-фреймворк от Google, основанный на принципах реактивного программирования. Он полностью написан на Kotlin и предоставляет следующие возможности:

* Композиция: интерфейс строится из компонуемых функций.
* Модификаторы: гибкая система настройки внешнего вида.
* Реактивность: автоматическое обновление интерфейса при изменении состояния приложения.
* Material Design: встроенная поддержка современных стандартов

Пример интерфейса, реализованного с использованием Jetpack Compose (например, в файле MainActivity.kt), представлен на рисунке 2.



Рисунок 2 – Пример макета на Jetpack Compose

Ключевые преимущества Jetpack Compose включают:

* Производительность: оптимизированная система рендеринга.
* Гибкость: простое создание кастомных компонентов.
* Инструменты: горячая перезагрузка (Live Reload).
* Эффективность: значительное сокращение объема кода.

Среди текущих ограничений можно выделить:

* Совместимость: минимальная API 21 (Android 5.0+).
* Кривая обучения: необходимость освоения новой парадигмы.
* Экосистема: ограниченное количество сторонних библиотек.

## Система контроля версий

Система контроля версий (Version Control System, VCS) — ключевой инструмент в разработке программного обеспечения, обеспечивающий управление изменениями в исходном коде. VCS фиксируют все модификации, организуют совместную работу разработчиков, сохраняют историю изменений и поддерживают параллельные ветки разработки.

Существует три типа систем контроля версий:

* Локальные системы (например, RCS, SCCS) работают на одном компьютере, не поддерживают одновременный доступ нескольких пользователей и не обеспечивают централизованное хранение данных.
* Централизованные системы (например, CVS, Subversion) хранят историю изменений на едином сервере, упрощая управление доступом и обмен изменениями. Однако они уязвимы к потере данных при повреждении сервера.
* Распределенные системы (например, Git, Mercurial) предоставляют каждому разработчику полную копию репозитория, обеспечивая автономную работу и высокую отказоустойчивость. Эти системы поддерживают нелинейную разработку и гибкую модель ветвления, что делает их популярными в современных проектах.

Основные концепции, используемые в системах контроля версий, включают:

* репозиторий (хранилище всех версий проекта);
* коммит (фиксация изменений с комментариями, автором и временной меткой);
* ветвление (создание изолированных линий разработки);
* слияние (объединение изменений из разных веток).

Среди систем контроля версий уверенно лидирует Git. Его популярность обусловлена тремя важными факторами: высокой скоростью работы, простой и понятной системой ветвления, а также поддержкой сложных нелинейных процессов разработки.

Использование системы контроля версий предоставляет множество преимуществ:

* полное отслеживание изменений с возможностью отката к предыдущим версиям;
* эффективную организацию командной работы;
* автоматическое резервное копирование кода;
* удобное тестирование и развертывание;
* возможность документирования эволюции проекта через комментарии к коммитам.

Современные системы контроля версий стали неотъемлемой частью разработки программного обеспечения, улучшая командную работу и управление сложными проектами.

1. Системный анализ
2. Системный анализ бизнес-процесса.

В современной жизни от человека требуется высокая организованность, умение управлять временем и контролировать различные аспекты своей жизни. Это всё можно назвать грамотным планированием, которое имеет следующие плюсы:

* Повышение продуктивности. Структурирование задач помогает избежать хаоса и сосредоточиться на приоритетных целях.
* Снижение уровня стресса. Четкий план уменьшает тревожность, связанную с забытыми делами или нехваткой времени.
* Достижение долгосрочных целей. Систематическое отслеживание прогресса мотивирует и дисциплинирует, способствует личностному росту.
* Оптимизация личных ресурсов. Распределение времени между работой, отдыхом и саморазвитием ведет к балансу и предотвращает выгорание.
* Улучшение качества жизни. Учет здоровья, финансов и эмоционального состояния помогает поддерживать гармонию.

Для этого существуют многофункциональные ежедневники, которые объединяют в себе инструменты для планирования, самоанализа и контроля различных сфер жизни, что делает их востребованными среди пользователей, стремящихся к осознанному управлению своей деятельности.

1. Анализ рынка

На рынке представлено множество приложений, предлагающих функции планирования, однако лишь некоторые из них сочетают в себе все необходимые модули: расписание, трекинг здоровья, финансы, аналитику и напоминания.

1. Основные конкуренты. Для сравнительного анализа были выбраны следующие приложения:

* Ежедневник, список дел, планер от Cleaner + Antivirus + VPN company.
* Brite от Brite Technologies LLC.
* Сингулярити: список дел, todo от SingularityApp.

1. Сравнительный анализ функционала. Ниже в таблице 1 представлен сравнительный анализ приложений конкурентов.

Таблица – Сравнительный анализ функционала приложений конкурентов.

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | Ежедневник, список дел, планер | Brite | Сингулярити: список дел, todo |
| Планирование задач | Да | Да | Да |
| Автоперенос просроченных задач | Да | Нет | Нет |
| Дневник благодарности и радости | Нет | Нет | Нет |
| Заметки | Нет | Да | Нет |
| Трекер настроения | Нет | Да | Нет |
| Трекер самочувствия | Нет | Нет | Нет |
| Трекер целей | Нет | Да | Нет |
| Трекер привычек | Да | Да | Да |
| Учет доходов/расходов | Нет | Есть только расходы | Нет |
| Трекер здоровья (шаги, вода, лекарства) | Нет | Да | Нет |
| Трекер хобби | Нет | Нет | Нет |
| Библиотека статей | Нет | Да | Нет |
| Аналитика и статистика | Нет | Нет | Нет |
| Напоминания перед выходом | Нет | Нет | Нет |
| Темы интерфейса приложения | Да | Да | Да |
| Работа приложения | Нормально | Много багов, зависаний | Нормальная |

Проведенный анализ функционала приложений конкурентов позволил выявить следующие ключевые недостатки:

* + Ограниченность функционала.
  + Отсутствие автопереноса задач.
  + Проблемы с производительностью.

Исходя из анализа функционала приложений конкурентов были выделены следующие требования к продукту:

1. Требования к интерфейсу:
   * + минималистичный, интуитивно понятный дизайн;
     + единый стиль элементов (кнопки, шрифты, цвета);
     + адаптивность под разные устройства и ориентации экрана;
     + поддержка динамического изменения шрифта и др.
2. Требования к базе данных:
   * + использование Supabase (PostgreSQL) с Row-Level Security (RLS);
     + соблюдение третьей нормальной формы (3NF), индексация часто используемых полей;
     + шифрование конфиденциальных данных и кеширование тяжелых запросов.
3. Требования к API:
   * + реализация основных конечных точек для всех модулей (задачи, здоровье, финансы и т. д.). Основные конечные точки указаны в приложении А;
     + аутентификация через Supabase Auth (email, OAuth, JWT);
     + обновление данных в реальном времени (Supabase Realtime API).
4. Диаграмма ERD

В разрабатываемом мобильном приложении, чтобы не допустить избыточность данных были спроектированы сущности в соответствии с определенными входными данными.

Логическая модель системы создавалась с использованием MS Visio, представлена на рисунке 3.

****

Рисунок 3 – ERD – Диаграмма базы данных

1. Диаграмма Прецедентов

Диаграмма прецедентов представляет собой графическое изображение взаимодействий между актерами (участниками процесса) и прецедентами (действиями), которые они выполняют в системе. Для приложения многофункционального ежедневника сделано несколько диаграмм прецедентов, которые будут описывать основные функции и задачи, входящие в подсистемы и выполняемые пользователями.

В диаграммы будут входить только один актер – пользователь.

В приложении многофункционального ежедневника выполняются следующие функции по подсистемам:

1. Подсистема «Ежедневник»:

* просмотр цитаты дня;
* управление задачами;
* управление записями в дневнике благодарности и радости;
* управление заметками на день;
* управление отметками настроения;
* управление отметками самочувствия;
* управление целями;
* управление привычками;
* управление идеями;
* управление записями о доходах/ расходах;
* просмотр и удаление уведомлений;
* нажатие на кнопку «Подготовка к выходу из дома».

1. Подсистема «Здоровье и фитнес»:

* просмотр количества шагов на данный период времени;
* управление ежедневной физической активностью;
* управление приемами пищи;
* управление приемами воды;
* управление приемами лекарств;
* управление измерениями тела.

1. Подсистема «Настройки и поддержка»:

* настройка уведомлений;
* настройка показа цитат дня;
* настройка оформления;
* настройка содержимого списка важных вещей;
* настройка режима выхода из дома;
* написание и отправка сообщений в техническую поддержку.

1. Подсистема «Профиль»:

* просмотр информации о пользователе;
* редактирование информации о пользователе;
* изменение пароля;
* передача QR-кода/9- значного кода дружбы;
* просмотр профиля друга;
* выход из профиля.

1. Подсистема «Статьи»:

* выбор статьи по категориям;
* просмотр статей.

1. Подсистема «Аналитика»:

* просмотр графиков, формируемых по записям в ежедневник.

Например, на диаграмме прецедентов, которая представлена на рисунке (рисунок 4), показано взаимодействие актеров и прецеденты, связанные с работой с приложением многофункциональным ежедневником. Пользователь выполняет все вышеописанные функции и задачи.

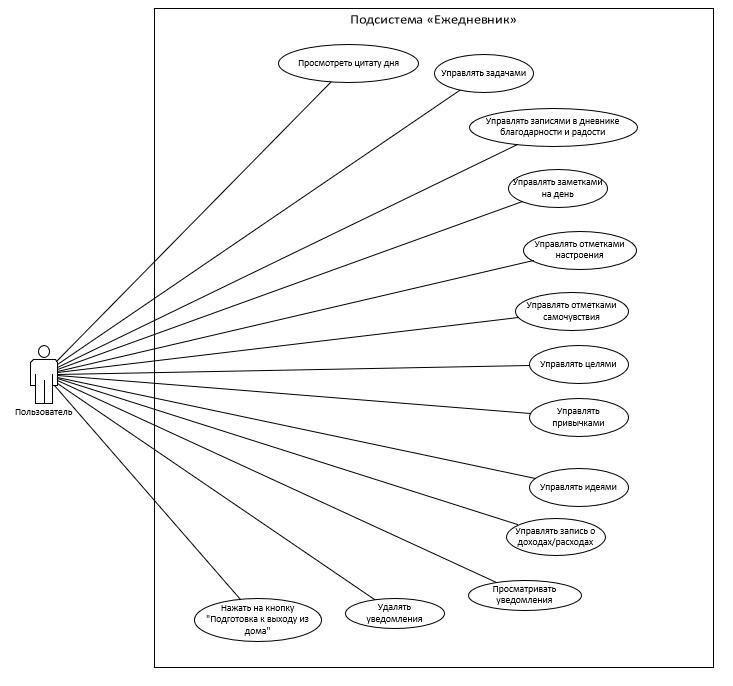


Рисунок 4– Диаграмма прецедентов подсистемы «Ежедневник»

1. Диаграмма деятельности

Диаграмма деятельности в нашем приложении многофункциональном ежедневнике используется для визуализации последовательности операций, связанных с созданием, редактированием, выполнением и отслеживанием записей в разделах «Ежедневник», «Здоровье и фитнес», в списке важных вещей и хобби, выбора и просмотра статей, добавления друзей, настройки оформления приложения, открытия цитаты дня, отметки в списке важных вещей, редактирования данных профиля и анализа личной продуктивности.

Пример диаграммы деятельности для процесса добавления записей в разделе «Ежедневник» показана на рисунке 5.

Начало процесса: процесс начинается с открытия пользователем приложения.

Проверка наличия интернета: после проверки наличия интернета в приложении осуществляются один из двух вариантов: при открытии приложения будет либо показ системой экрана ошибки интернета, либо переход на следующий шаг, при сохранении записи либо сохранение данных локально, либо сохранение в Supabase.

Проверка наличия авторизации: после проверки наличия авторизации в приложении осуществляются один из двух вариантов: переход системой на экран авторизации и завершение процесса, или переход на экран «Главная» раздел «Ежедневник».

Добавление записи: пользователь нажимает на кнопку «Добавить», выбирает, что ему нужно добавить, и система переносит его на экран добавления записи.

Сохранение записи: пользователь нажимает на кнопку «Сохранить», и система проводит проверку наличия интернета. При положительном исходе проверки отправляет запрос в базу данных для добавления записи. Запись сохраняется и идет ответ от базы данных. Потом идет проверка ответа от базы данных.

Проверка ответа от базы данных: после проверки ответа базы данных в приложении осуществляются один из двух вариантов: показа системой ошибки с базой данных или сообщения об успешном добавлении записи.

Завершение процесса: процесс завершается.

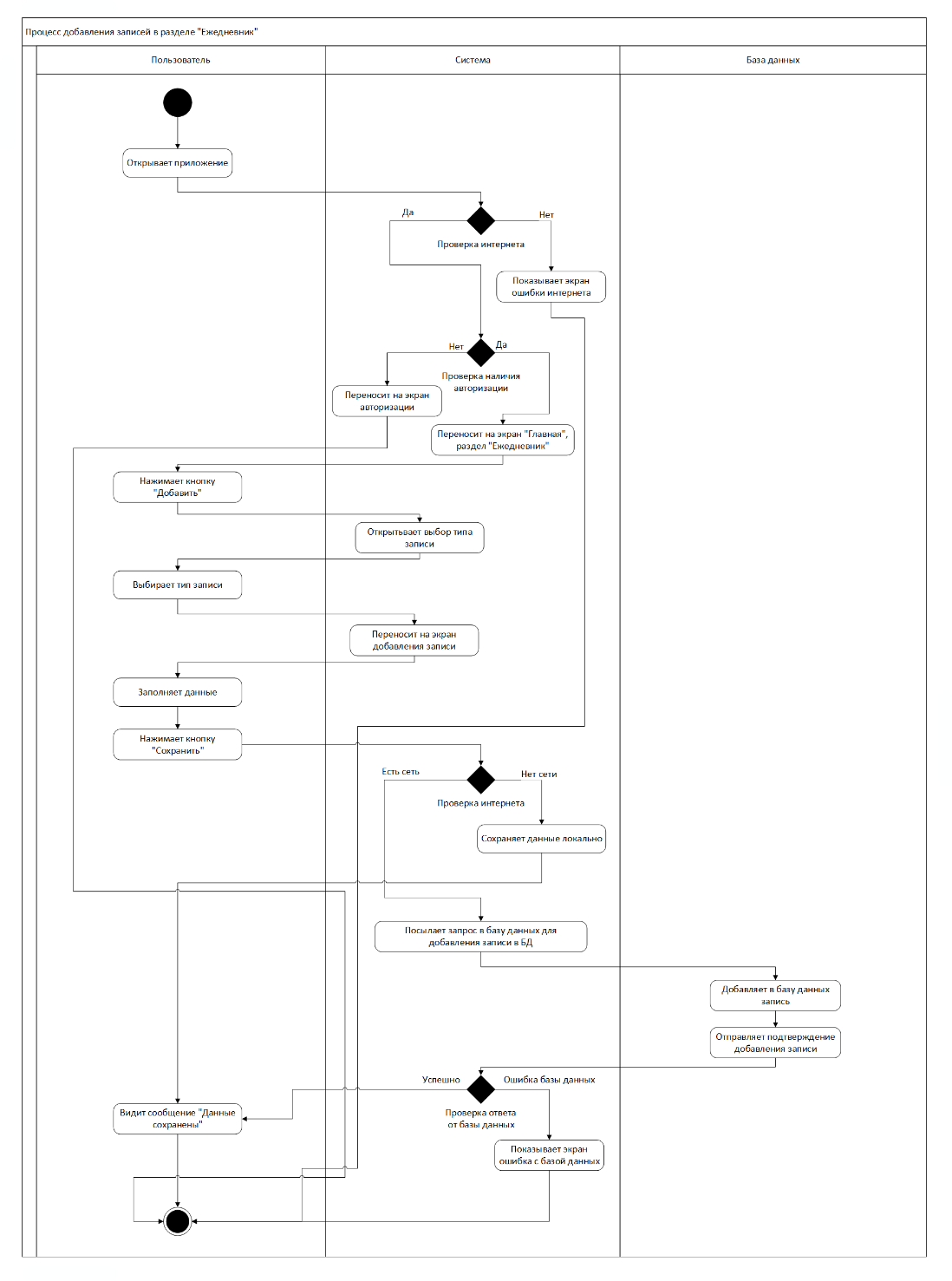


Рисунок 5 – Диаграмма деятельности процесса добавления записей в разделе «Ежедневник»

Диаграмма деятельности помогает наглядно представить последовательность шагов и условия, влияющие на ход процесса в приложении многофункциональном ежедневнике. Это упрощает понимание сложных процедур и позволяет улучшить организацию работы склада, где можно наглядно отследить какое действие в каком этапе проходит. Диаграмма деятельности позволяет увидеть весь жизненный путь процесса.

1. Диаграмма последовательности

Диаграмма последовательности — это инструмент для моделирования поведения системы во времени, который фокусируется на взаимодействии объектов и передаче сообщений между ними в рамках выполнения конкретного сценария.

В контексте системы управления разделами приложения (такими как «Ежедневник», «Здоровье и фитнес», «Хобби» и др.) диаграммы последовательности описывают ключевые процессы, такие как добавление, редактирование и просмотр записей, а также взаимодействие пользователя с интерфейсом и базой данных.

Элементы диаграммы последовательности

Актёры (Actor): участники процесса, инициирующие взаимодействие с системой. Основным актёром является Пользователь, который выполняет действия через интерфейс приложения (например, нажатие кнопок, выбор категорий, ввод данных).

Объекты (Object): Компоненты системы, участвующие в процессе, например:

* Интерфейсные экраны (добавления, редактирования, просмотра).
* База данных — хранилище информации, куда сохраняются или откуда извлекаются данные.

Линии жизни (Lifeline): представляют существование объекта во времени. Каждая линия жизни имеет свое начало и конец, соответствующие началу и концу существования объекта в рассматриваемом сценарии.

Сообщения (Message): линии, показывающие передачу сообщений между объектами. Сообщения могут быть синхронными (стрелка с заполненной головкой) или асинхронными (стрелка с пустой головкой).

Активации (Activation): прямоугольники на линиях жизни, обозначающие периоды активности объекта. Они показывают, когда объект выполняет определенные действия.

Пример диаграммы последовательности для процесса добавления записей в разделе «Ежедневник» показана на рисунке 6.

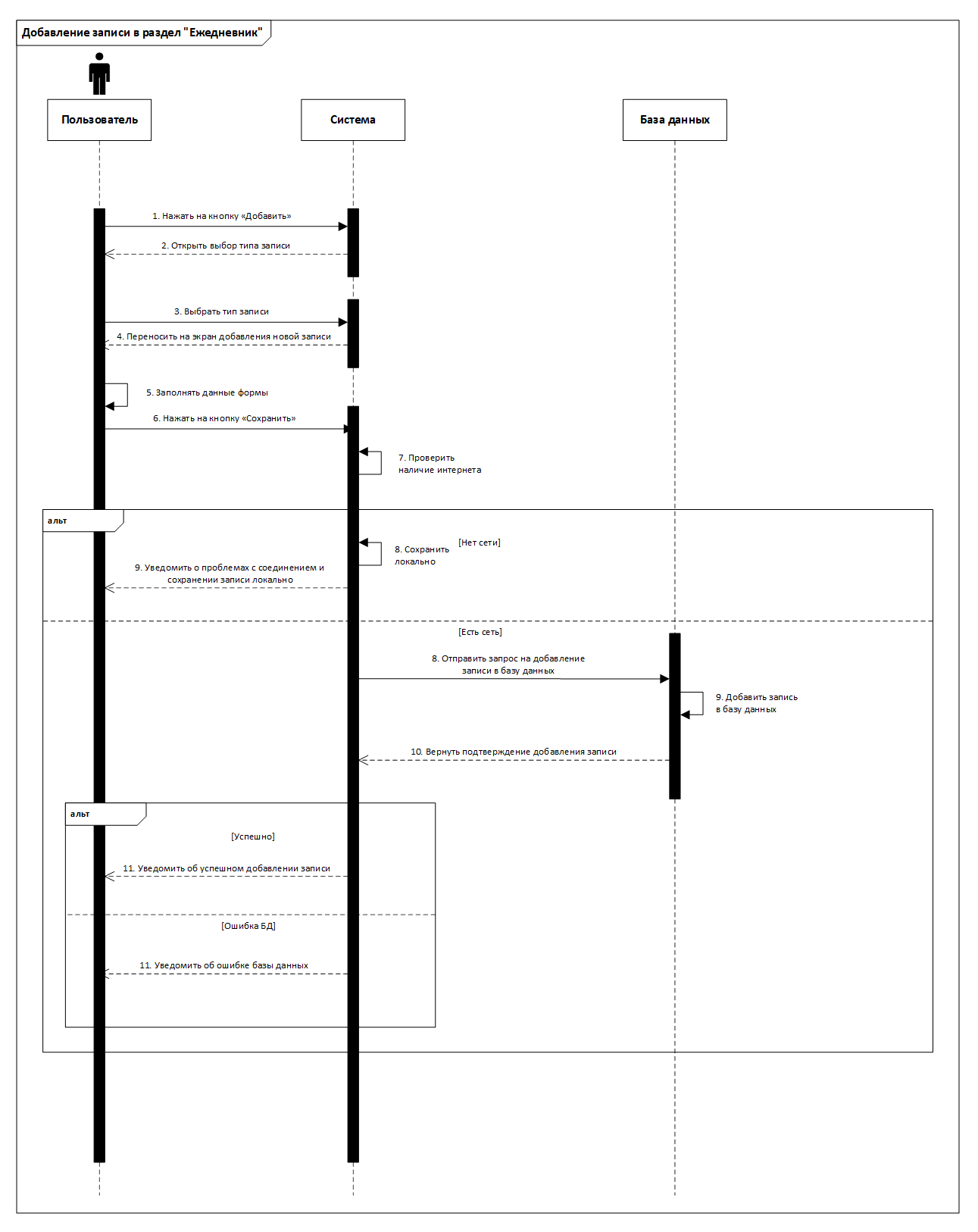


Рисунок 6 – Диаграмма последовательности процесса добавления новой записи в раздел «Ежедневник»

Анализ диаграммы последовательности позволяет обнаружить возможные задержки или неэффективности в процессе взаимодействия между объектами.

Документирование и обучение: диаграмма может служить полезным инструментом для документирования существующих процессов и обучения новых сотрудников.

1. Практическая часть
2. Прототипирование приложения

Прототипирование играет важную роль в процессе разработки, так как оно позволяет визуализировать идею продукта до его реализации, выявить потенциальные проблемы и улучшить пользовательский опыт. Прототипирование является инструментом для тестирования функциональности, навигации и дизайна, что экономит время и ресурсы, предотвращая дорогостоящие правки на поздних этапах. Кроме того, прототип помогает согласовать видение проекта между дизайнерами, разработчиками и заказчиками, обеспечивая единое понимание целей и задач.

Пользователь, назовем его Иван, впервые открывает приложение многофункциональный ежедневник. Заставка приложения представлена на рисунке 7.



Рисунок 7 – Заставка приложения

После заставки, система переносит пользователя на экран регистрации. Иван вводит email и заполняет свои данные, включая ник и дату рождения. На рисунке 8 представлены экраны регистрации и заполнения данных пользователя.

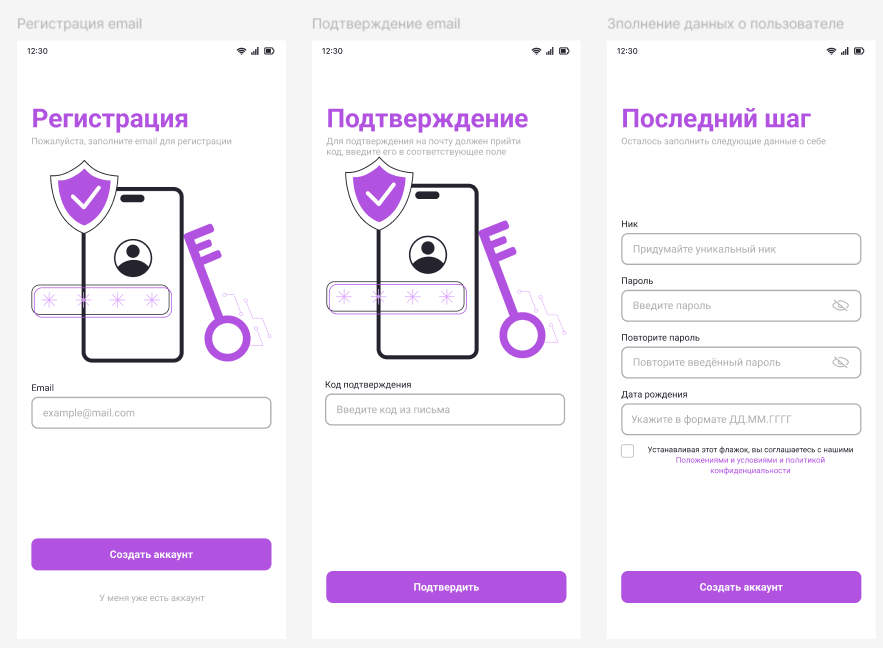


Рисунок 8 – Экраны регистрации, подтверждения и заполнения данных пользователя

После завершения регистрации Иван попадает на главный экран, где видит календарь, список задач, дневник благодарности и раздел для заметок. На главной странице Иван может добавлять задачи, например, «Сделать генеральную уборку», с указанием приоритета, даты выполнения и подзадач. На рисунке 9 представлен главный экран раздел «Ежедневник» и экран задачи.



Рисунок 9 – Главный экран раздел «Ежедневник» и экран задачи

В разделе «Здоровье и фитнес» Иван отслеживает свои шаги, активность и приемы пищи, что помогает контролировать физическое состояние. Раздел «Здоровье и фитнес» представлен на рисунке 10.

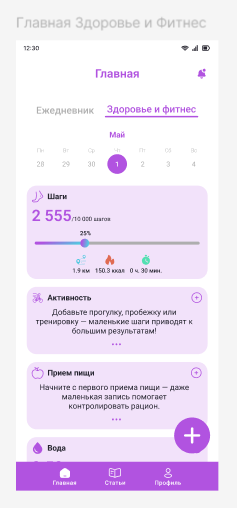


Рисунок 10 – Главный экран раздел «Здоровье и фитнес»

Прототип в Figma отражает этот сценарий, начиная с экранов приветствия и регистрации и заканчивая главным меню и функциональными разделами. Например, экран «Дневник благодарности и радости» включает поля для ввода благодарностей и радостей. Экраны для дневника благодарности и радости представлены на рисунке 11.

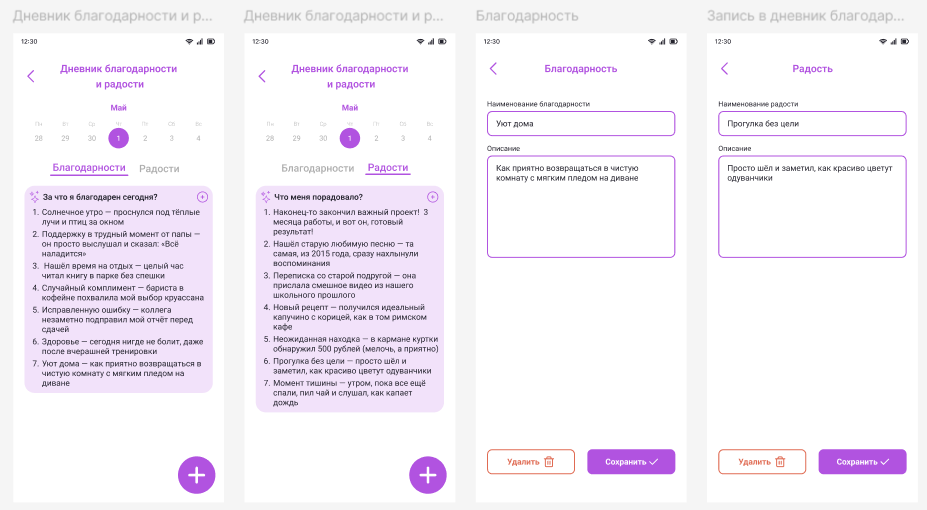


Рисунок 11 –Экран «Дневник благодарности и радости»

Важным аспектом прототипирования является тестирование удобства интерфейса. Например, на экран «Выход из дома» пользователь может отмечать собранные вещи, такие как телефон, ключи или зонт. Экран «Выход из дома» представлен на рисунке 12.

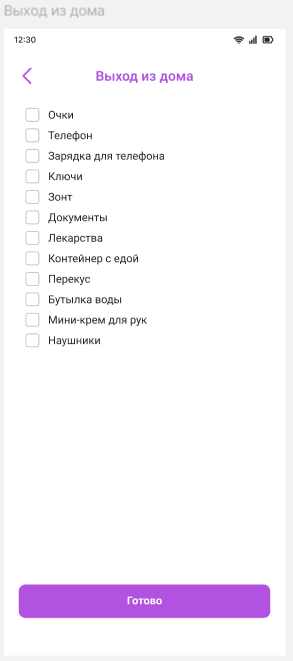


Рисунок 12 –Экран «Выход из дома»

Таким образом, прототипирование не только помогает детально проработать дизайн и функционал, но и обеспечивает удобство использования приложения. Оно позволяет учесть потребности пользователей на ранних этапах разработки, минимизировать ошибки и создать продукт, который действительно решает их задачи. В случае с приложением из документа, прототип в Figma стал основой для создания удобного и мотивирующего инструмента для планирования и саморазвития.

1. Разработка приложения

3.2.1 Проектирование интерфейса. Проектирование интерфейса является ключевым этапом разработки приложения, так как от его удобства и функциональности зависит пользовательский опыт. В данном проекте для создания интерфейса выбран Jetpack Compose — современный декларативный фреймворк от Google для разработки UI на Android. Его главное преимущество заключается в декларативном подходе, который позволяет описывать интерфейс в виде функций, автоматически обновляющих отображение при изменении данных. Это значительно упрощает разработку, делает код более читаемым и поддерживаемым.

Для адаптивной вёрстки используется Modifier, в котором задаюся размеры (fillMaxSize()) и отступы (padding()), а Column и Row организуют контент. Навигация реализована через NavHost и NavController, а WindowInsets адаптирует макет под системные элемент. Для единообразия дизайна применяются темы (YourDayTheme) с поддержкой светлого/темного режимов (isSystemInDarkTheme), а rememberSystemUiController динамически настраивает цвета статус бара. Пример использования Modifief.padding() и WindowInsets для адаптации под системные элементы представлен на рисунке 13. 

Рисунок 13 – Использования Modifief.padding() и WindowInsets для адаптации под системные элементы

Для проектирования интерфейса были созданы собственные компоненты: CustomDatePicker, CategoryChip и CustomToast.

CustomDatePicker предназначен для выбора даты в стандартном для пользователя формате. Компонент осуществляет отображение текущей даты в формате dd.MM.yyyy. При нажатии на компонент осуществлянтся отерыте диалогового окно для выбора даты с использованием стандартного компонента Material Design 3 DatePickerDialogдительский компонент.

Пример кода компонента представлен на рисунке 14.



Рисунок 14 – Код компонента CustomDatePicker

Пример применение компонента представлен на рисунке 15.

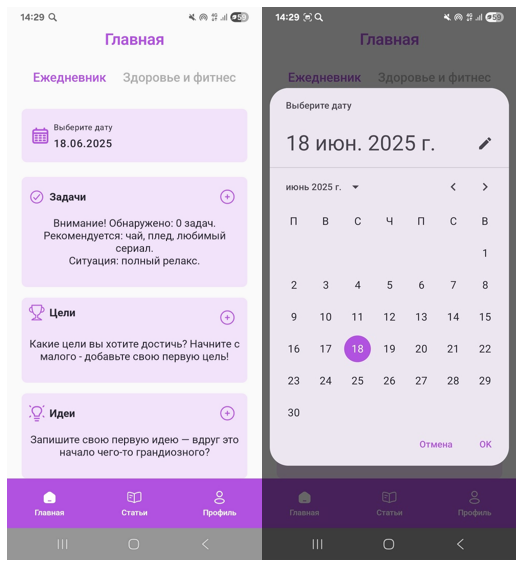


Рисунок 15 – Применение компонента CustomDatePicker

CategoryChip предназначен для выбора категорий статей. Компонент поддерживает состояния (выбран/не выбран) с изменением цвета и границы, через настройки внешнего вида через параметры isSelected и onClick. Пример кода компонента представлен на рисунке 16.



Рисунок 16 – Код компонента CategoryChip

Пример применение компонента представлен на рисунке 17.

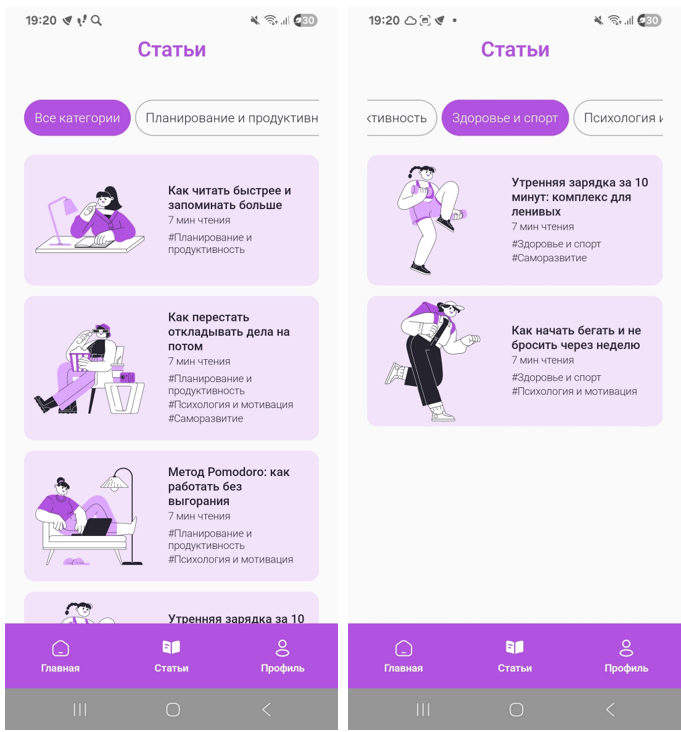


Рисунок 17 – Применение компонента CategoryChip

CustomToast предназначен для отображения уведомлений с поддержкой разных типов (успех, ошибка, информация). В компоненте реализована анимация появления и исчезновения и настройка длительности отображения. Пример кода компонента представлен на рисунке 18.



Рисунок 18 – Код компонента CustomToast

Пример применение компонента представлен на рисунке 19.

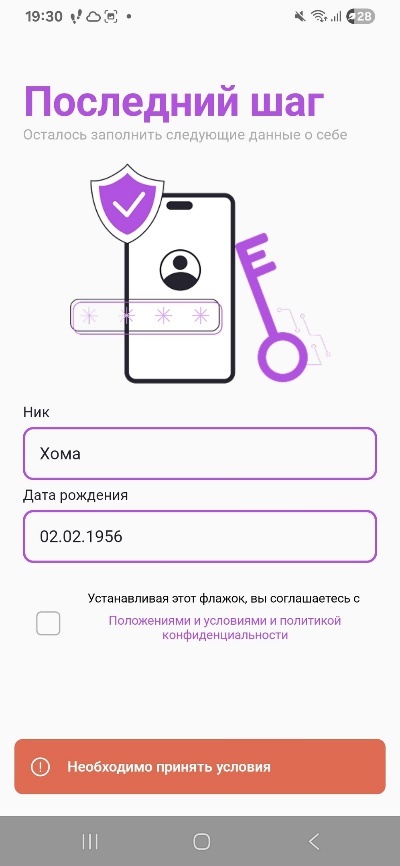


Рисунок 19 – Применение компонента CustomToast

Jetpack Compose успешно справился с задачей проектирования интерфейса, предоставив гибкие инструменты для создания адаптивных, модульных и интерактивных компонентов. Декларативный подход позволил сократить объем кода, упростить его поддержку и обеспечить высокую производительность. Использование кастомных компонентов (CustomDatePicker, CategoryChip, CustomToast) сделало интерфейс единообразным и удобным для пользователя. Таким образом, выбранная технология полностью оправдала ожидания и стала надежной основой для реализации UI приложения.

3.2.2 Разработка API на Supabase. Разработка API на Supabase с использованием Kotlin представляет собой процесс взаимодействия с облачной базой данных и другими сервисами Supabase через RESTful API или клиентские библиотеки. Supabase предоставляет готовые эндпоинты для работы с данными, аутентификации, хранения файлов и других функций, что значительно упрощает разработку бэкенда для мобильных приложений.

Для начала работы с Supabase необходимо создать проект на официальном сайте, после чего настроить базу данных. Экран создания базы данных представлен на рисунке 20.

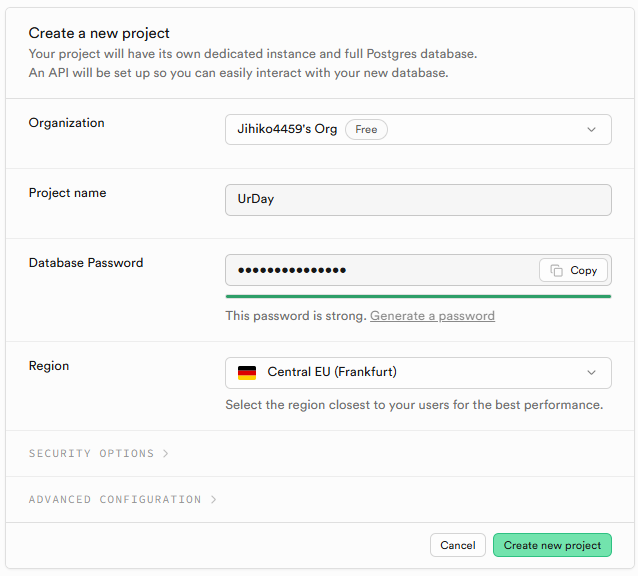


Рисунок 20 – Экран создания базы данных

В Supabase используется PostgreSQL, поэтому все таблицы и схемы создаются стандартными SQL-запросами. Например, для создания таблицы tasks в схеме tasks можно использовать запрос, представленный на рисунке 21.

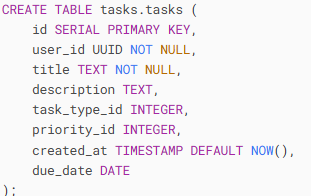


Рисунок 21 – Запрос для создания таблицы tasks в схеме tasks

Аналогично создаются другие таблицы, такие как user\_moods, motivational\_cards, notifications и т.д. Для удобства можно использовать SQL-скрипты, которые выполняются в Supabase SQL Editor.

Важным этапом было определение связей между таблицами и настройка правил доступа через Row Level Security (RLS), чтобы обеспечить безопасность данных. Например, пользователи могли получать доступ только к своим задачам, а администраторы — ко всем данным. Пример запроса на создания правил доступа расположена на рисунке 22.

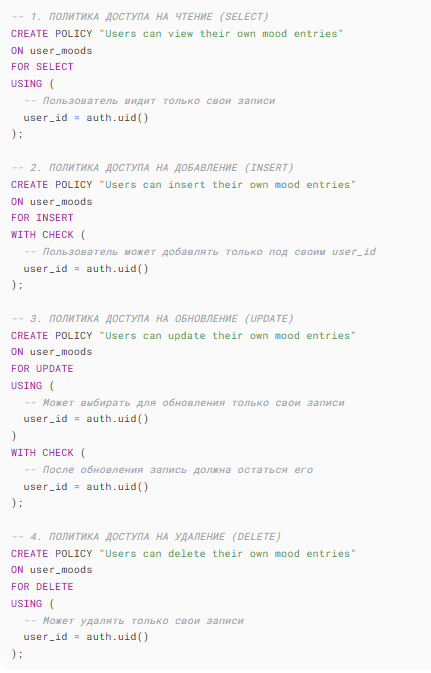


Рисунок 22 – Запрос создания правил доступа для таблицы настроений пользователя user\_moods

Процедуры и триггеры пишутся на PL/pgSQL. Например, процедура для ежедневного назначения мотивационных карточек пользователям представлена на рисунке 23.



Рисунок 23 – Запрос на создание процедуры для ежедневного назначения мотивационных карточек пользователям

Триггеры создаются для автоматического выполнения действий при изменении данных. Например, триггер для генерации задач при добавлении нового медикамента, расположеный на рисунке 24.



Рисунок 24 – Запрос создания правил доступа для таблицы настроений пользователя user\_moods

Хранение медиафайлов, таких как изображения для мотивационных карточек или аватары пользователей, было организовано через Supabase Storage. Этот сервис позволяет загружать, скачивать и управлять файлами, а также генерировать публичные URL для доступа к ним. Например, изображения для настроений хранились в бакете «illustration-of-articles», а их URL использовались в клиентской части приложения. Пример представлен на рисунке 25.

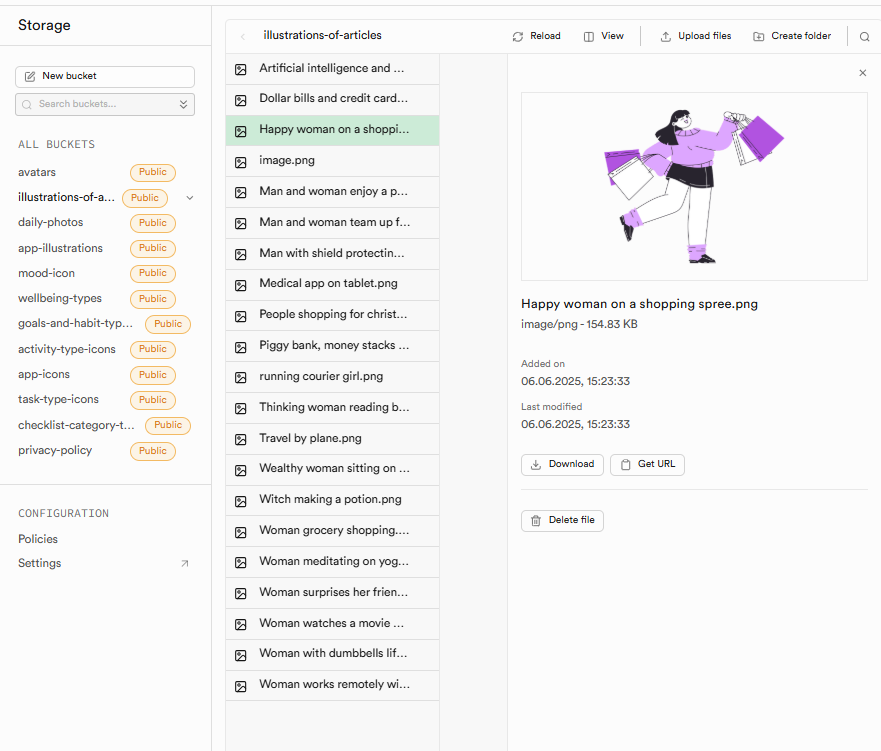


Рисунок 25 – Бакет «illustration-of-articles» для хранения изображений настроений

Использование Supabase значительно ускоряет разработку, так как платформа предоставляет готовые решения для типовых задач, таких как аутентификация, хранение данных и файлов. Это позволило сосредоточиться на реализации уникальной логики приложения, не тратя время на настройку инфраструктуры. В результате API получилось масштабируемым, безопасным и простым в поддержке, что соответствует требованиям современной веб-разработки.

3.2.3 Организация сетевого взаимодействия. Для организации сетевого взаимодействия был создан класс SupabaseHelper, который обеспечивает всю логику работы с серверной частью приложения. Этот класс обеспечивает выполнение операций CRUD (создание, чтение, обновление, удаление) через REST API Supabase, а также управление аутентификацией пользователей. Supabase был выбран в качестве backend-решения благодаря его готовым инструментам для аутентификации, хранения данных и файлов, что значительно ускорило разработку и снизило нагрузку на настройку инфраструктуры. Код класса разбит на логические блоки, представленные в приложении Б.

Аутентификация реализована через методы signUpWithEmail и signInWithEmail, которые используют встроенный модуль Supabase Auth. Для обработки ошибок применяется функция mapErrorToUserMessage, преобразующая технические сообщения в понятные пользователю формулировки. Например, ошибка «already registered» преобразуется в «Email уже зарегистрирован».

Для работы с данными используются методы getDailyTasks, getDailyGoals, getDailyIdeas и другие, которые выполняют запросы к Supabase PostgREST API. Эти методы обернуты в withAuth, обеспечивающий проверку авторизации и состояния сети. В случае ошибок соединения или отсутствия интернета возвращаются mock-данные, что гарантирует работу приложения даже в офлайн-режиме.

Методы saveTask, updateUserProfile и deleteTask обеспечивают создание, изменение и удаление данных. Они используют транзакционный подход и проверку прав доступа, чтобы исключить несанкционированные операции. Например, при удалении задачи проверяется, что она принадлежит текущему пользователю.

Supabase значительно упростил разработку, предоставив готовые решения для аутентификации, хранения данных и обработки запросов. Встроенная поддержка JWT-токенов и автоматическое обновление сессии повысили безопасность. PostgREST API позволил минимизировать ручное написание SQL-запросов, а встроенный механизм подписки на изменения данных упростил реализацию реактивного интерфейса. Благодаря этому сетевое взаимодействие стало надежным, масштабируемым и простым в поддержке.

Заключение

В ходе выполнения дипломного проекта было создано многофункциональное мобильное приложение, объединяющее инструменты для планирования, трекинга здоровья и саморазвития. Приложение разработано с использованием современных технологий, таких как Kotlin и Jetpack Compose, что обеспечило высокую производительность и удобство взаимодействия с пользователем.

Ключевые модули, включая управление задачами и раздел статей, были успешно реализованы и протестированы. Для хранения и обработки данных использовалась облачная платформа Supabase, предоставляющая надежное и масштабируемое решение для работы с базой данных и аутентификацией пользователей. Сетевое взаимодействие организовано через RESTful API, что позволило обеспечить быстрый обмен данными и их актуальность.

Интерфейс приложения был спроектирован с учетом принципов юзабилити и адаптивности, что делает его удобным для пользователей с разными устройствами и уровнями технической подготовки. Проведенные тесты подтвердили стабильность работы приложения и его готовность к публикации. В перспективе возможно добавление новых функций, таких как интеграция с умными устройствами и расширенная аналитика.

Список использованных источников

* + - 1. Разработка под Android [Электронный ресурс] https://habr.com/ru/hubs/android\_dev/articles (дата обращения 17.06.2025)
      2. Android Official Documentation [Электронный ресурс] https://www.android.com/intl/ (дата обращения 03.06.2025)
      3. Kotlin Official Documentation [Электронный ресурс] https://kotlinlang.org/ (дата обращения 13.06.2025)
      4. Google Developers. Официальная документация по Jetpack Compose. [Электронный ресурс] <https://developer.android.com/jetpack/compose> (дата обращения 20.05.2025)
      5. Supabase Documentation. [Электронный ресурс] <https://supabase.com/docs> (дата обращения 29.05.2025)
      6. Figma Official Guide. [Электронный ресурс] <https://www.figma.com/resources/learn-design/> (дата обращения 01.05.2025)
      7. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. — Питер, Санкт-Петербург, 2021. — 352 с.
      8. Фаулер М. UML. Основы. — Символ-Плюс, Санкт-Петербург, 2019. — 192 с.
      9. PostgreSQL Documentation. [Электронный ресурс] <https://www.postgresql.org/docs/> (дата обращения 29.05.2025)
      10. Material Design Guidelines. [Электронный ресурс] <https://material.io/design> (дата обращения 01.05.2025)
      11. Material Design Components for Android. [Электронный ресурс] <https://material.io/components?platform=android> (дата обращения 01.05.2025)
      12. Figma Community — Plugins & Templates. [Электронный ресурс] <https://www.figma.com/community/> (дата обращения 11.05.2025)
      13. Android Developers Blog. [Электронный ресурс] <https://android-developers.googleblog.com/> (дата обращения 30.05.2025)
      14. Ромео А. Прототипирование интерфейсов: Figma, Adobe XD, Sketch. — БХВ-Петербург, 2021. — 256 с.
      15. UML Distilled. Мартин Фаулер. — Вильямс, 2018. — 208 с.
      16. Роберт Мартин. Чистый код. — Питер, 2020. — 464 с.
      17. Дональд Норман. Дизайн привычных вещей. — Манн, Иванов и Фербер, 2021. — 368 с.
      18. Джефф Сазерленд. Scrum: революционный метод управления проектами. — Манн, Иванов и Фербер, 2020. — 288 с.
      19. Smashing Magazine — UI/UX Design. [Электронный ресурс] <https://www.smashingmagazine.com/category/design/> (дата обращения 10.05.2025)
      20. Руководство по Supabase. Часть 1. [Электронный ресурс] <https://habr.com/ru/companies/timeweb/articles/648761/> (дата обращения 23.05.2025)
      21. Руководство по Supabase. Часть 2. [Электронный ресурс] [https://habr.com/ru/companies/timeweb/articles/660183/](https://habr.com/ru/companies/timeweb/articles/648761/) (дата обращения 26.05.2025)
      22. Использование диаграммы классов UML при проектировании и документировании программного обеспечения. [Электронный ресурс] <https://habr.com/ru/articles/572234/> (дата обращения 03.03.2025)
      23. Использование диаграммы вариантов использования UML при проектировании программного обеспечения. [Электронный ресурс] <https://habr.com/ru/articles/566218/> (дата обращения 13.03.2025)
      24. Диаграмма последовательности (sequence-диаграмма). [Электронный ресурс] <https://habr.com/ru/articles/814769/> (дата обращения 16.03.2025)
      25. Освой Android играючи. Сайт Александра Климова. [Электронный ресурс] <https://developer.alexanderklimov.ru/android/theory/compose_theory.php> (дата обращения 30.05.2025)
      26. ANDROID KNOWLEDGE. [Электронный ресурс] <https://androidknowledge.com/> (дата обращения 17.06.2025)
      27. Руководство по языку Kotlin. [Электронный ресурс] <https://metanit.com/kotlin/tutorial/> (дата обращения 27.05.2025)
      28. Руководство созданию приложений под Android с помощью Kotlin и Jetpack Compose. [Электронный ресурс] <https://metanit.com/kotlin/jetpack/> (дата обращения 27.05.2025)
      29. Создание мобильных приложений на KOTLIN (Android). [Электронный ресурс] <https://neco-desarrollo.es/создание-мобильных-приложений-на-kotlin-android#google_vignette> (дата обращения 30.05.2025)

Приложение А  
Основные конечные точки API

Таблица А1 – Конечные точки API

|  |  |  |
| --- | --- | --- |
| Модуль/схема | Методы | Описание запроса |
| tasks | GET /tasks | Фильтрация по due\_date, is\_completed, is\_dependent, user\_id |
| tasks | POST /tasks | Добавление новой задачи |
| tasks | PATCH /tasks/{id} | Частичное обновление задачи |
| tasks | DELETE /tasks/{id} | Удаление задачи |
| tasks | POST /user\_task\_types | Добавление новой записи о типе задачи пользователя |
| tasks | GET /user\_task\_types | Получение записей о типах задач пользователя |
| tasks | PATCH /user\_task\_types/{id} | Частичное обновление записи о типе задачи пользователя |
| tasks | DELETE /user\_task\_types/{id} | Удаление записи о типе задачи пользователя |
| tasks | GET /task\_dependencies | Получение зависимостей задач |
| tasks | POST /task\_dependencies | Добавление новой зависимости задач |
| Tasks | DELETE /task\_dependencies/{id} | Удаление зависимости задач |

Продолжение таблицы А1

|  |  |  |
| --- | --- | --- |
| Модуль/схема | Методы | Описание запроса |
| daily | GET /ideas | Получение записей из идей пользователя |
| daily | POST /ideas | Добавление новой записи об идее пользователя |
| daily | PATCH /ideas/{id} | Частичное обновление записи об идее пользователя |
| daily | DELETE /ideas/{id} | Удаление записи об идее пользователя |
| daily | GET /hobbies | Получение записей о хобби пользователя |
| daily | POST /hobbies | Добавление новой записи о хобби пользователя |
| daily | PATCH /hobbies/{id} | Частичное обновление записи о хобби пользователя |
| daily | DELETE /hobbies/{id} | Удаление записи о хобби пользователя |
| goals\_and\_habits | GET /goals | Получение записей о целях пользователя |
| goals\_and\_habits | POST /goals | Добавление новой записи о цели пользователя |
| goals\_and\_habits | PATCH /goals/{id} | Частичное обновление записи о цели пользователя |
| goals\_and\_habits | DELETE /goals/{id} | Удаление записи о цели пользователя |
| goals\_and\_habits | GET /goal\_statuses | Получение записей о статусах цели с фильтрацией по user\_id |
| goals\_and\_habits | GET /goal\_types | Получение записей типов цели |

Продолжение таблицы А1

|  |  |  |
| --- | --- | --- |
| Модуль/схема | Методы | Описание запроса |
| goals\_and\_habits | GET /habits | Получение записей о привычках пользователя |
| goals\_and\_habits | POST /habits | Добавление новой записи о привычке пользователя |
| goals\_and\_habits | PATCH /habits/{id} | Добавление новой записи о привычке пользователя |
| goals\_and\_habits | DELETE /habits/{id} | Добавление новой записи о привычке пользователя |
| health\_and\_fitness | GET /medications | Получение записей о лекарствах пользователя |
| health\_and\_fitness | POST /medications | Добавление новой записи о лекарстве пользователя |
| health\_and\_fitness | PATCH /medications/{id} | Частичное обновление записи о лекарстве пользователя |
| health\_and\_fitness | DELETE /medications/{id} | Удаление записи о лекарстве пользователя |
| health\_and\_fitness | GET /steps | Получение записей о шагах пользователя |
| health\_and\_fitness | POST /steps | Добавление новой записи о шагах пользователя |
| health\_and\_fitness | PATCH /steps/{id} | Частичное обновление записи о шагах пользователя |
| health\_and\_fitness | DELETE/steps/{id} | Удаление записи о шагах пользователя |
| health\_and\_fitness | GET /user\_activity | Получение записей об активности пользователя |

Продолжение таблицы А1

|  |  |  |
| --- | --- | --- |
| Модуль/схема | Методы | Описание запроса |
| health\_and\_fitness | POST /user\_activity | Добавление новой записи об активности пользователя |
| health\_and\_fitness | PATCH /user\_activity/{id} | Частичное обновление записи об активности пользователя |
| health\_and\_fitness | DELETE /user\_activity/{id} | Удаление записи об активности пользователя |
| health\_and\_fitness | GET /activity\_types | Получение записей типах активности |
| content | GET /user\_article\_statuses | Получение записей об изменениях статусов статей для пользователя |
| content | GET /article\_categories | Получение записей о категориях статей |
| content | GET /articles\_in\_categories | Получение записей о статьях с фильтрацией по категориям |
| content | GET /articles | Получение записей о статьях |
| auth | GET /users/{id} | Получение записи о данных пользователя |
| auth | POST /users | Добавление нового пользователя |
| auth | PATCH /users/{id} | Частичное обновление данных пользователя |
| auth | DELETE /users/{id} | Удаление пользователя |
| auth | GET /profiles/{id} | Получение записи о данных профиля |
| auth | POST /profiles | Добавление нового профиля |

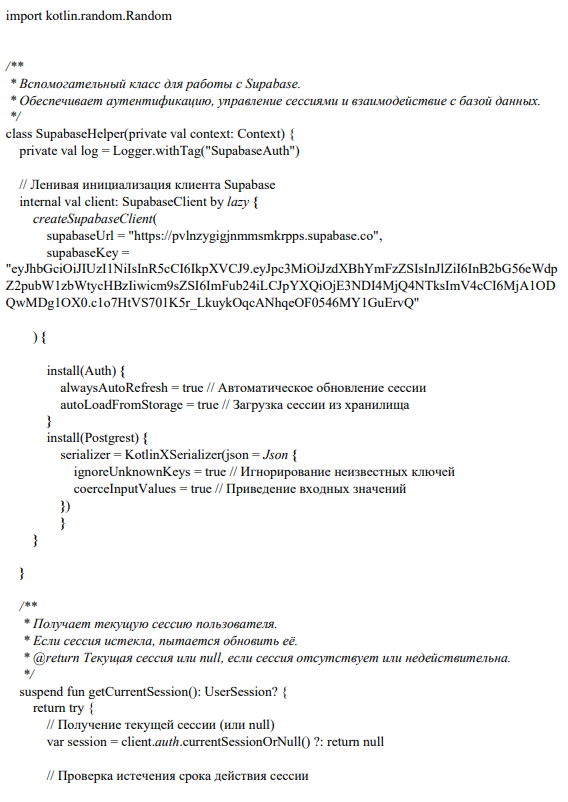
Продолжение таблицы А1

|  |  |  |
| --- | --- | --- |
| Модуль/схема | Методы | Описание запроса |
| auth | PATCH /profiles/{id} | Частичное обновление данных профиля |
| auth | DELETE /profiles/{id} | Удаление профиля |
| auth | GET /genders | Получение данный о поле |
| public | GET /days\_of\_the\_week | Получение дней недели |
| public | GET /units | Получение единиц измерения |

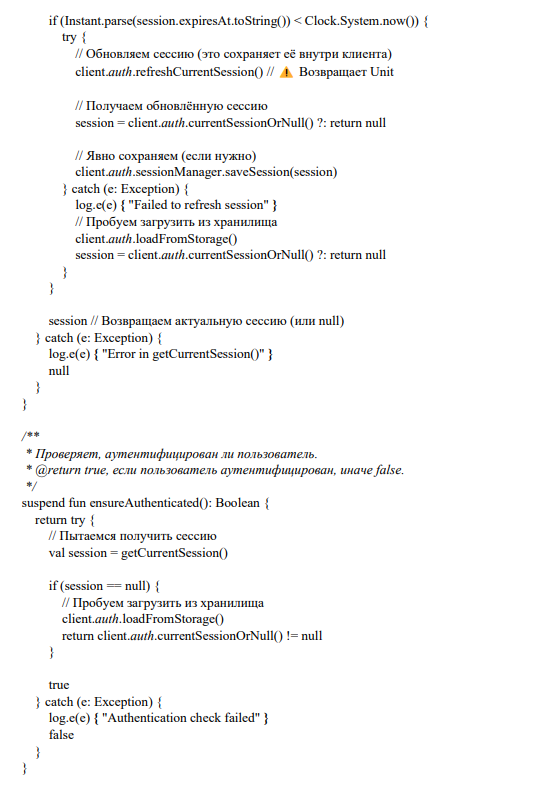
Приложение Б  
Листинг кода. Класс SupabaseHelper.



Рисунок Б1 – Класс SupabaseHelper



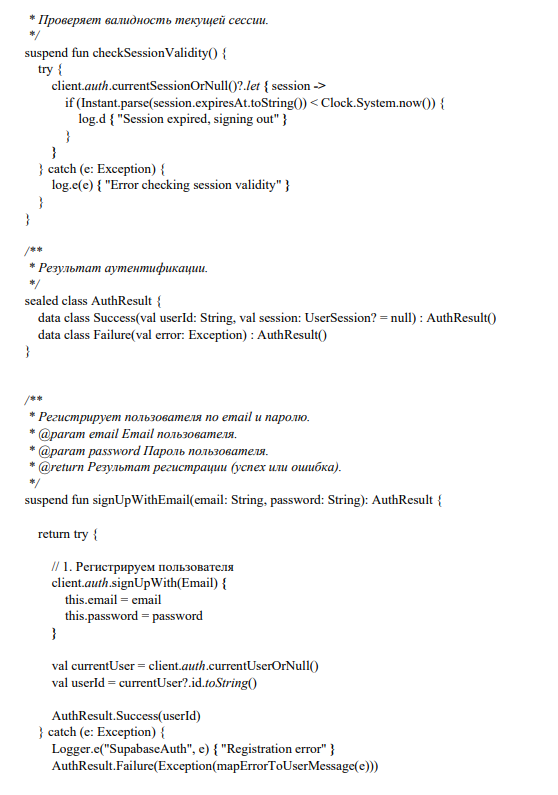
Продолжение рисунок Б1



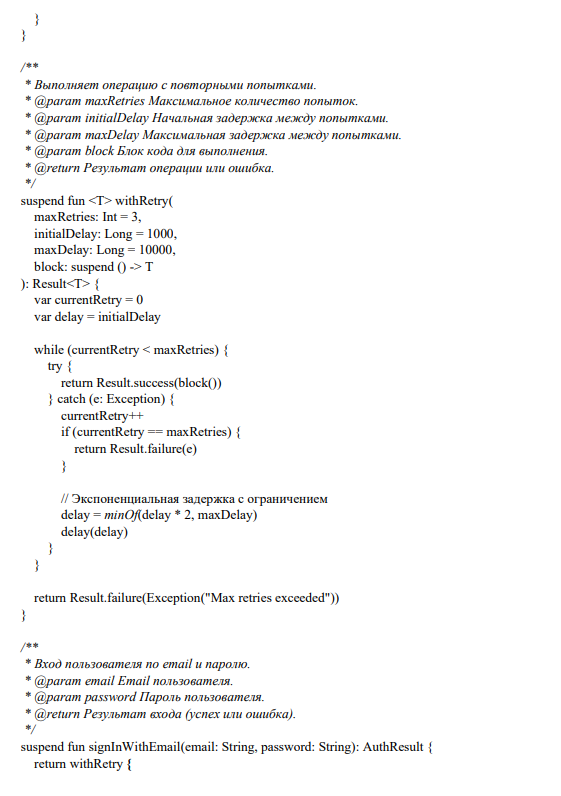
Продолжение рисунок Б1



Продолжение рисунок Б1



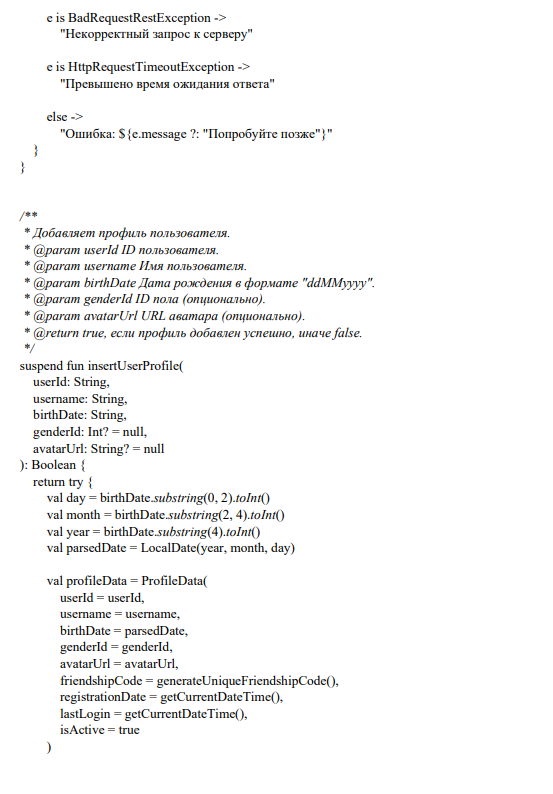
Продолжение рисунок Б1



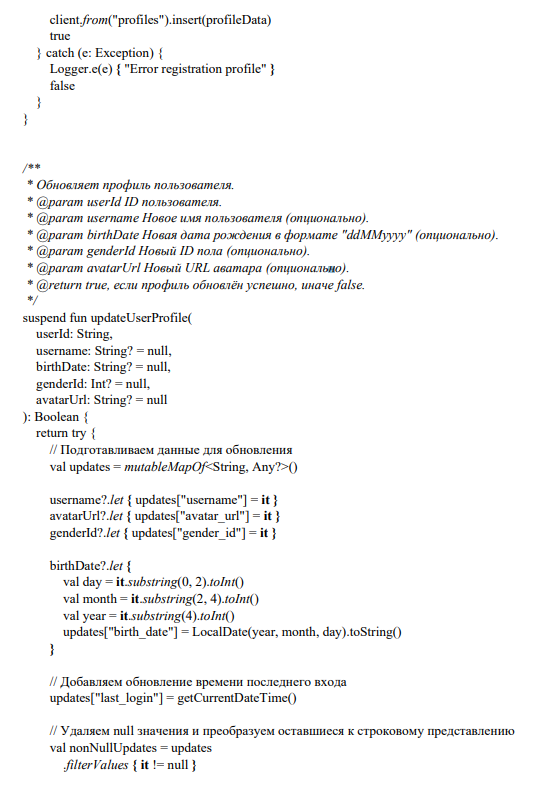
Продолжение рисунок Б1



Продолжение рисунок Б1



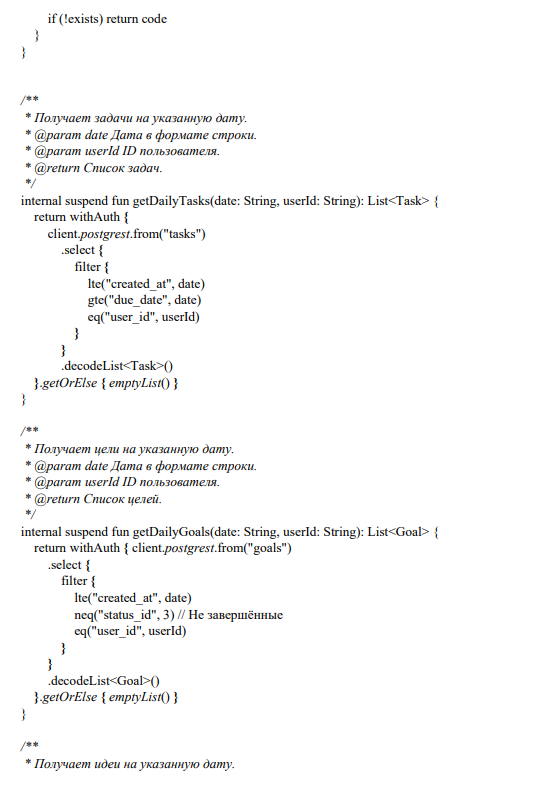
Продолжение рисунок Б1



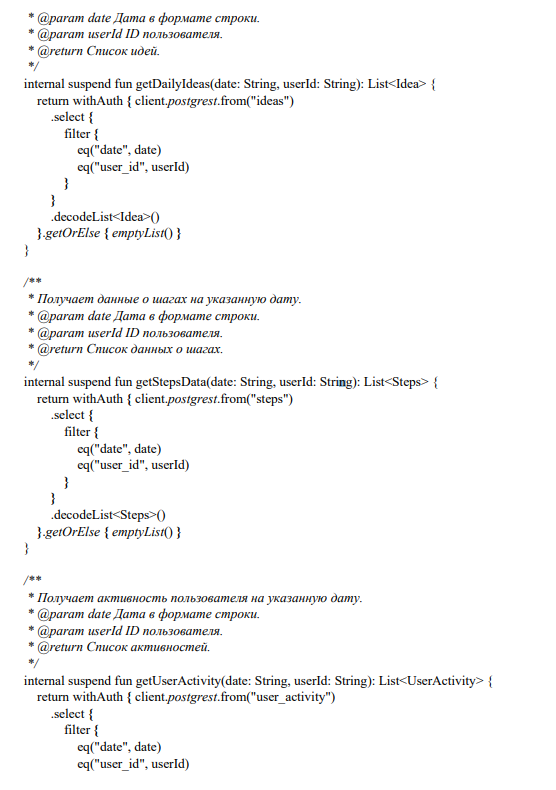
Продолжение рисунок Б1



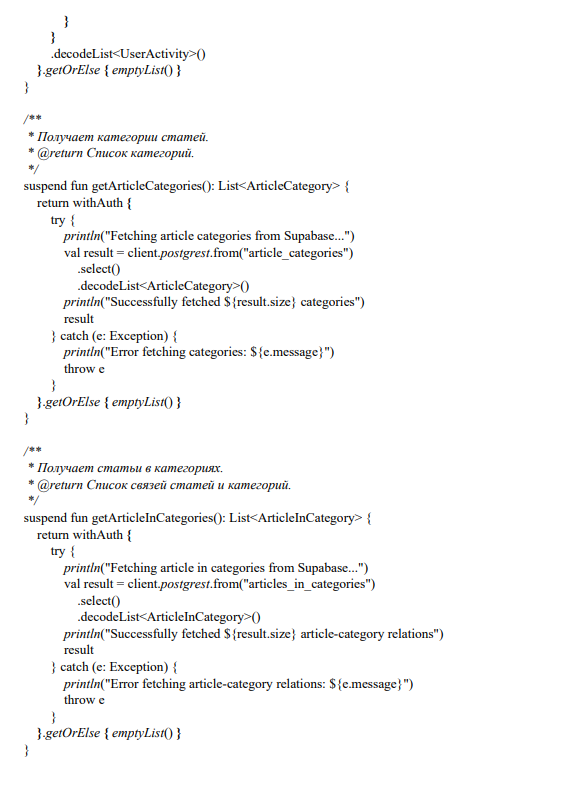
Продолжение рисунок Б1



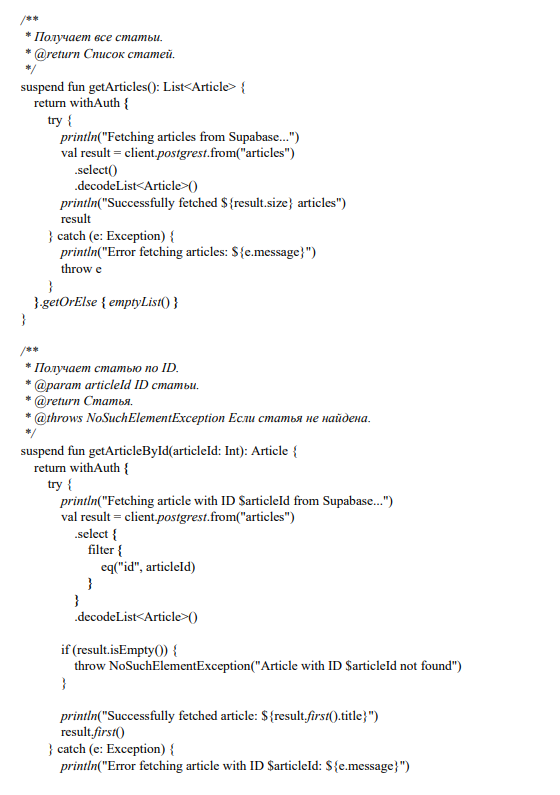
Продолжение рисунок Б1



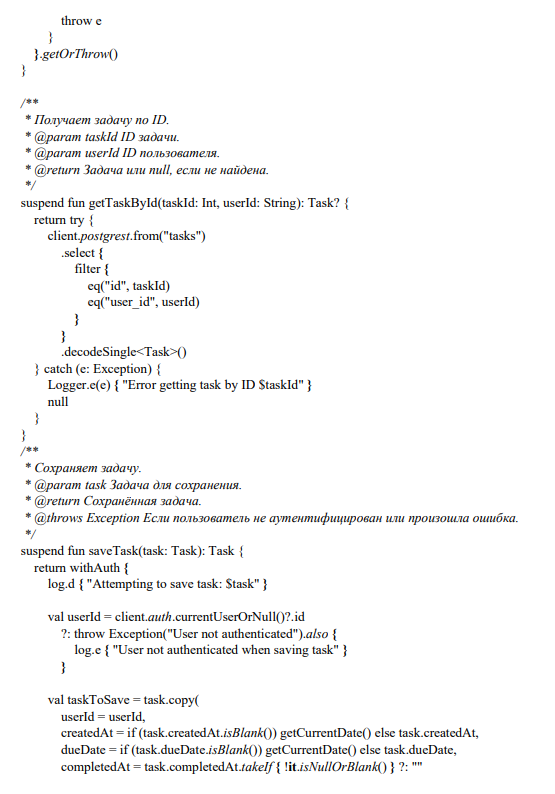
Продолжение рисунок Б1



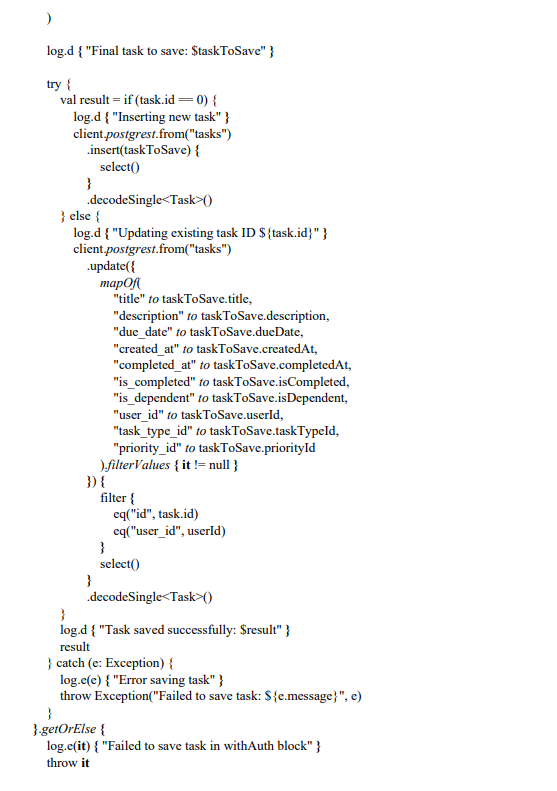
Продолжение рисунок Б1



Продолжение рисунок Б1



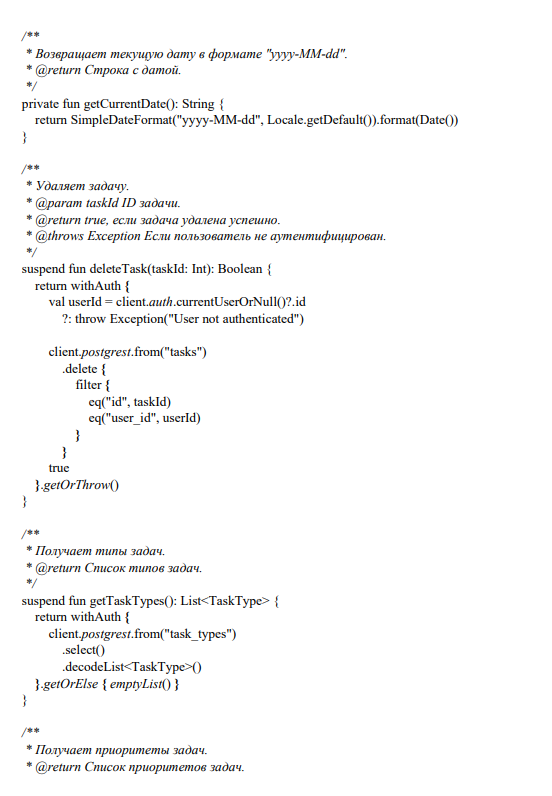
Продолжение рисунок Б1



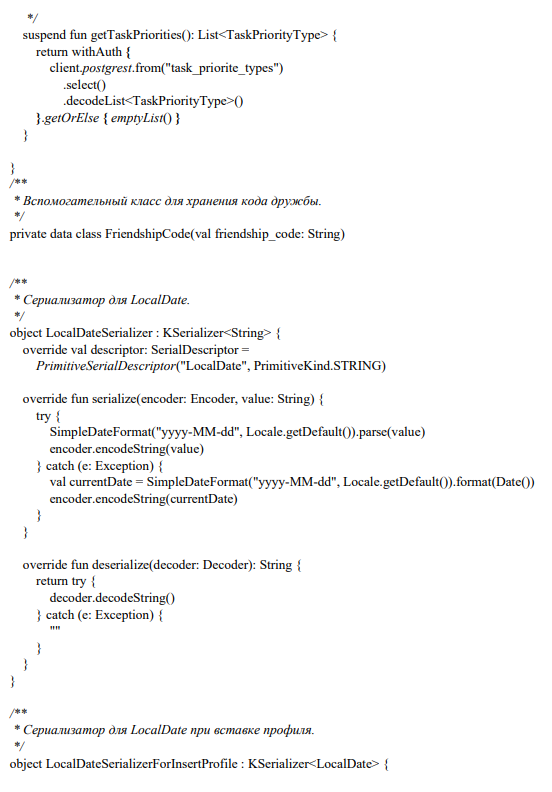
Продолжение рисунок Б1



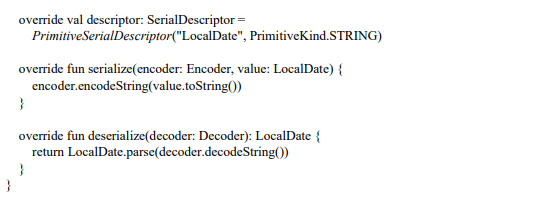
Продолжение рисунок Б1



Продолжение рисунок Б1



Продолжение рисунок Б1



Продолжение рисунок Б1

Приложение В  
Листинг кода. Объект TaskRepository.

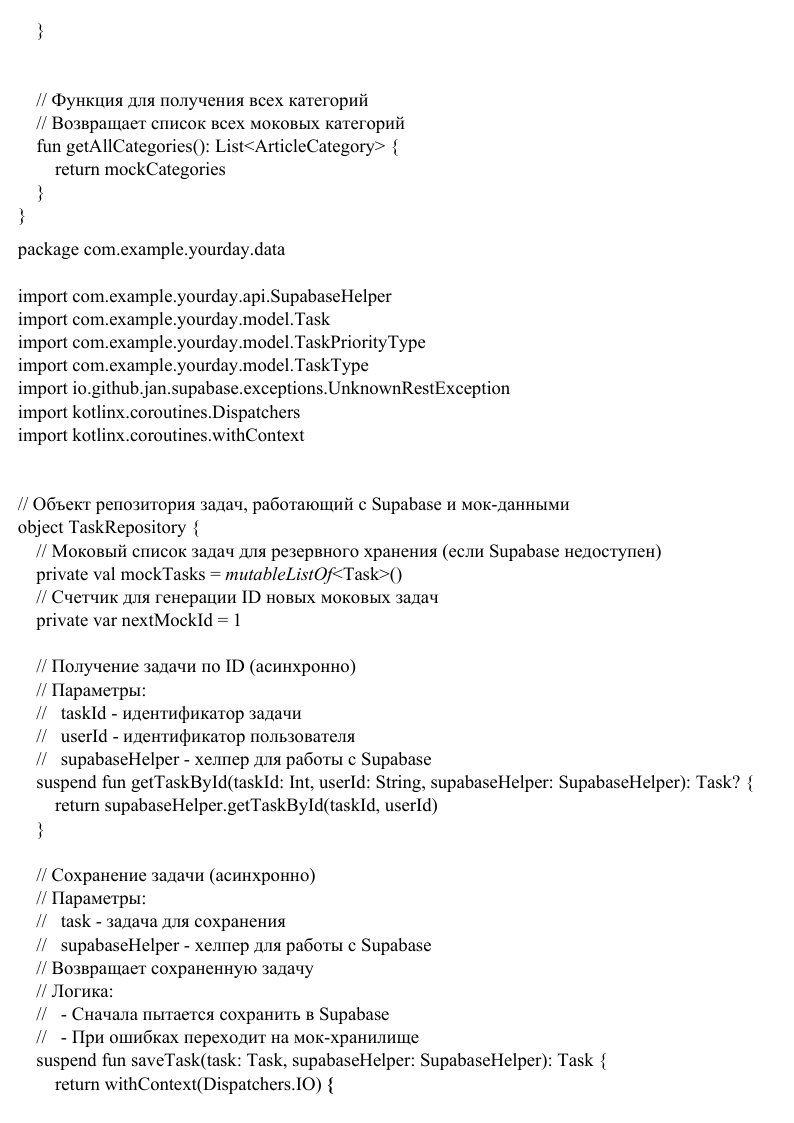
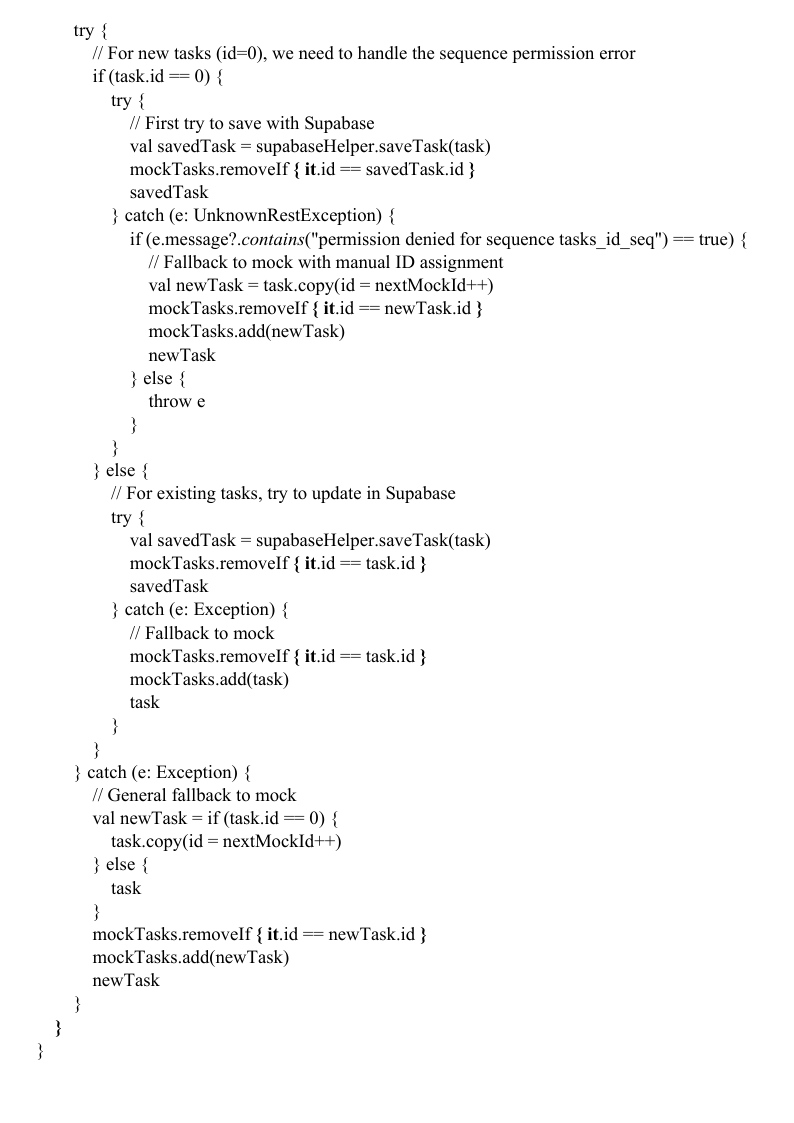
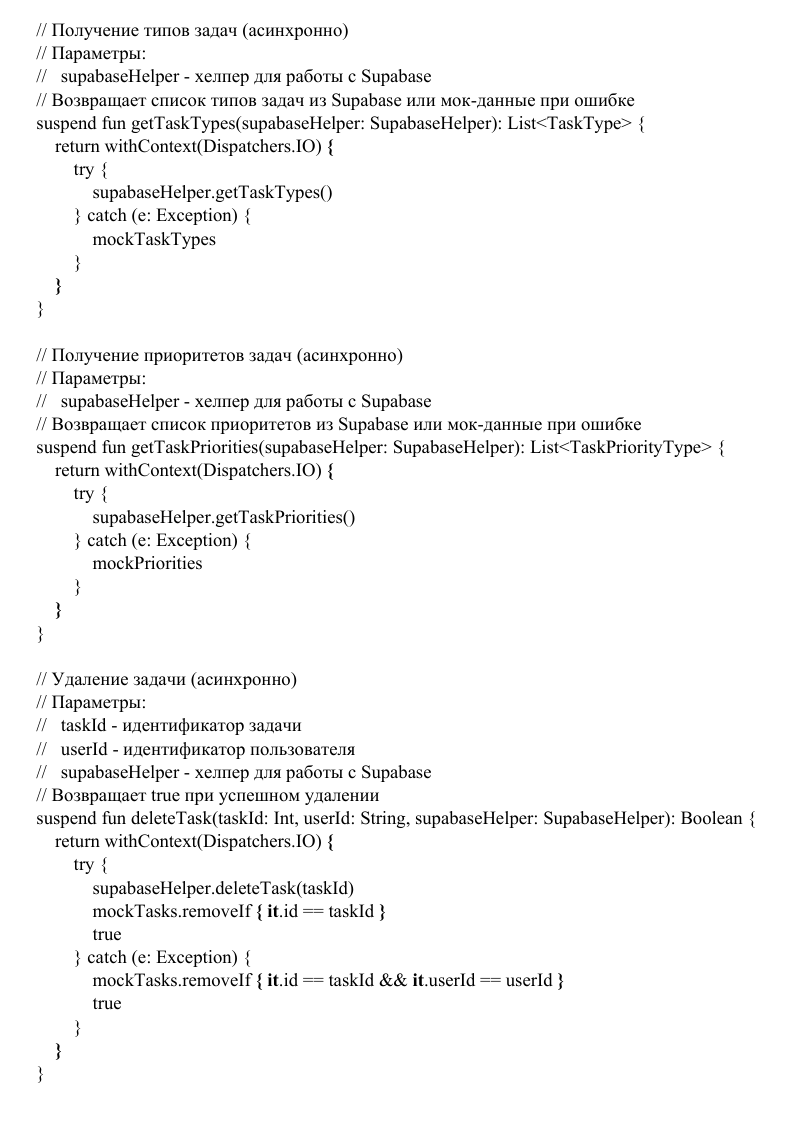


Рисунок B1 – Объект TaskRepository



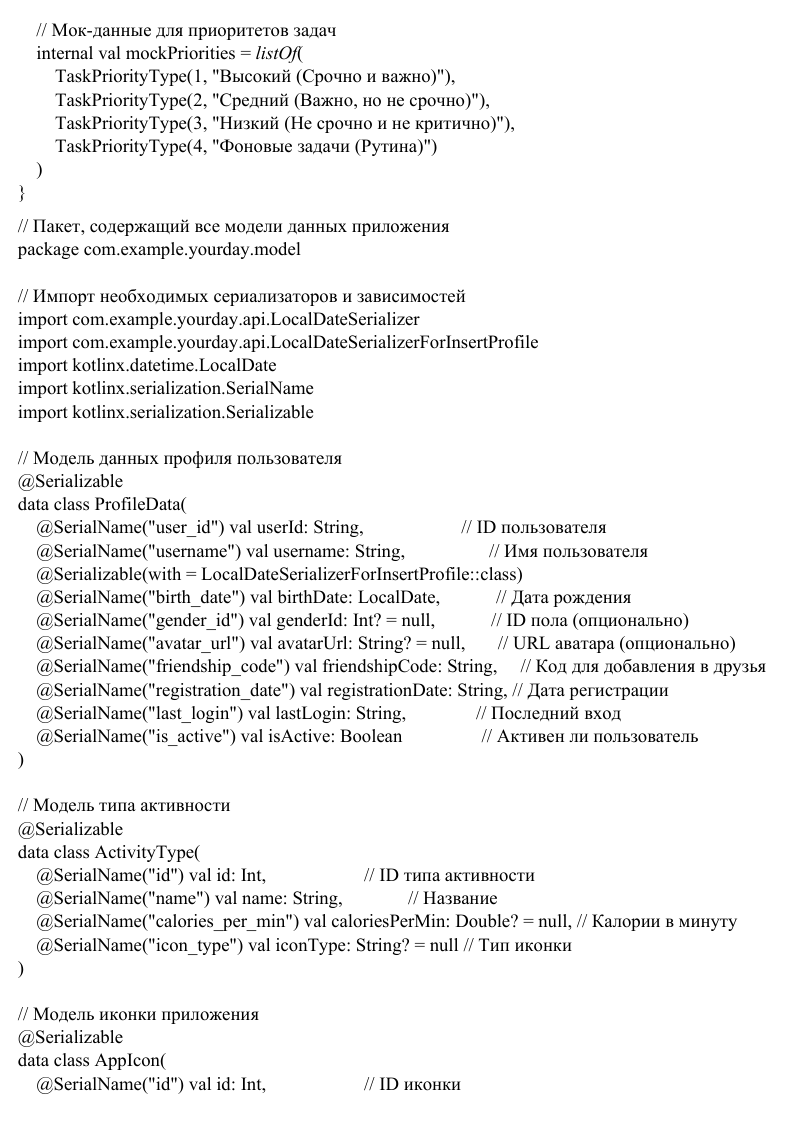
Продолжение рисунок В1



Продолжение рисунок В1



Продолжение рисунок В1



Продолжение рисунок В1