Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Воронежский государственный технический университет»

(ФГБОУ ВО «ВГТУ», ВГТУ)

Строительно-политехнический колледж

**Дипломный проект**

«Разработка программного продукта для повышения продуктивности и концентрации в любой деятельности»

«*Тема*»

09.02.07 Информационные системы и программирование

*(шифр, наименование профессии/специальности)*

Руководитель структурного подразделения

Н.А. Донцова

*(подпись, И.О. Фамилия)*

Руководитель дипломного проекта

В.В. Сокольников

*(подпись, И.О. Фамилия)*

Консультанты:

по разделу «Безопасность жизнедеятельности», «Охрана окружающей среды»

И.В. Михневич

*(подпись, И.О. Фамилия)*

по разделу «Экономическая часть»

Т.И. Смотрова

*(подпись, И.О. Фамилия)*

«Нормоконтроль»

Э.С. Чепкова

*(подпись, И.О. Фамилия)*

Обучающийся

В.А. Приходько

*(подпись, И.О. Фамилия)*

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Воронежский государственный технический университет»

(ФГБОУ ВО «ВГТУ», ВГТУ)

«УТВЕРЖДАЮ»

Сергеева С.И.

*(Ф.И.О., должность – заместитель директора)*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись)*

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_202\_\_ г.

**ЗАДАНИЕ**

**на выполнение дипломного проекта (дипломной работы)**

Обучающегося Приходько Владислава Анатольевича

*(фамилия, имя, отчество обучающегося)*

по основной профессиональной образовательной программе

09.02.07 Информационные системы и программирование

*(шифр, наименование профессии/специальности)*

Тематика дипломного проекта (работы): «Разработка программного продукта для повышения продуктивности и концентрации в любой деятельности», утверждена решением педагогического совета от 29.12. 2022г. протокол № 4

Тема дипломного проекта (работы): «Разработка программного продукта для повышения продуктивности и концентрации в любой деятельности», предложена преподавателем(руководителем)*.*

Срок сдачи законченной работы \_\_\_. \_\_\_. 202\_\_\_г.

Календарный план: (строится в соответствии со структурой дипломного проекта (работы))

|  |  |  |  |
| --- | --- | --- | --- |
| №пп | Структура дипломного проекта (работы) | Сроки выполнения | Примечание |
|  | Введение |  |  |
|  | Глава 1 Описание и анализ предметной области |  |  |
|  | 1.1. Анализ предметной области |  |  |
|  | 1.2. Анализ существующих программных решений |  |  |
|  | 1.3. Функциональные требования к мобильному приложению «…» |  |  |
|  | 1.4 Выбор средства разработки |  |  |
|  | Глава 2 Реализация программного решения |  |  |
|  | 2.1. Диаграмма классов |  |  |
|  | 2.2. Проектирование диаграммы Use-Case |  |  |
|  | 2.3. Проектирование ER-диаграммы |  |  |
|  | 2.4. Работа мобильного приложения «…» |  |  |
|  | Глава 3 «Безопасность жизнедеятельности», «Охрана окружающей среды» |  |  |
|  | 3.1. Общие требования при работе на мобильном устройстве |  |  |
|  | 3.2. Вредные и опасные факторы, воздействующие на человека при работе на мобильном устройстве |  |  |
|  | 3.3. Условия эксплуатации мобильного устройства |  |  |
|  | Глава 4 «Экономическая часть проекта» |  |  |
|  | 4.1 Обоснование экономической части |  |  |
|  | Заключение |  |  |
|  | Список литературы |  |  |
|  | Приложения |  |  |

5. Консультанты (с точным указанием относящихся к ним разделов выпускной квалификационной работы):

«Безопасность жизнедеятельности» И.В. Михневич

(*полное наименование раздела) (И.О. Фамилия)*

«Экономическая часть проекта» Т.И. Смотрова

(*полное наименование раздела) (И.О. Фамилия)*

Рассмотрено на заседании ПЦК 29.12.2022, протокол №4\_\_\_\_\_\_

Председатель ПЦК М.А. Попов

(*подпись, И.О. Фамилия)*

Руководитель В.В. Сокольников

(*подпись, И.О. Фамилия)*

Дата выдачи задания

Задание принял к исполнению

Обучающийся

(*подпись, И.О. Фамилия)*

Термины и определения

Список сокращений

ER -. Entity-Relationship model;

НДС - Налог на добавленную стоимость;

Содержание

[1 Описание и анализ предметной области 9](#_Toc199959081)

[1.1 Анализ существующих программных решений 10](#_Toc199959082)

[1.2 Функциональные требования к мобильному приложению 13](#_Toc199959083)

[1.3 Выбор средства разработки 15](#_Toc199959084)

[2 Реализация и проектирование мобильного приложения 18](#_Toc199959085)

[2.1 Диаграмма классов 19](#_Toc199959086)

[2.2 Проектирование диаграммы Use-Case 21](#_Toc199959087)

[2.3 Проектирование ER-диаграммы 24](#_Toc199959088)

[2.4 Руководство пользователя 27](#_Toc199959089)

[3 Безопасность жизнедеятельности. 46](#_Toc199959090)

[4 Экономическая часть проекта 49](#_Toc199959091)

[4.1 Определение трудоемкости разработки программного продукта 49](#_Toc199959092)

[4.2 Расчет расходов на оплату труда разработчика 51](#_Toc199959093)

[4.3 Расчет затрат на оплату машинного труда 52](#_Toc199959094)

[4.4 Определение накладных расходов и затрат на расходные материалы 55](#_Toc199959095)

[4.5 Составление сметы затрат на разработку программного продукта и структуры затрат 56](#_Toc199959096)

[4.6 Определение цены разработки ПО 56](#_Toc199959097)

[ЗАКЛЮЧЕНИЕ 58](#_Toc199959098)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 60](#_Toc199959099)

[Приложение A 62](#_Toc199959100)

ВВЕДЕНИЕ

В настоящее время, мобильные телефоны стали неотъемлемой часть нашей жизни. Приложения, нацеленные на использование с мобильных устройств, получают всё большее распространение и многие компании, разрабатывающие программные продукты, уделяют больше внимания именно мобильной разработке, в отличие от разработки приложений для персональных компьютеров .

Одним из популярных направлений в мобильной разработке стало создание приложений, позволяющих эффективно организовать свои дела, управлять временем и следить за своим настроением. Такие программные продукты значительно повышают уровень самоорганизации и самоконтроля.

Данная дипломная работа посвящена разработке программного продукта, предназначенного для повышения продуктивности и концентрации в любой деятельности. Проект включает в себя следующие ключевые компоненты: создание страниц с задачами, настраиваемый таймер по технике помодоро, создание отметок о настроении, просмотр статистики настроений, встроенный искусственный интеллект для генерации персонализированных советов по оптимизации дел и улучшению настроения, а также система достижений, мотивирующая к регулярному использованию приложения.

Приложение ориентировано на мобильную разработку с использованием таких технологий, как React Native, Firebase и интеграции искусственного интеллекта Meta AI. Особое внимание уделено интуитивно понятному и приятному дизайну, удобству пользования, а также возможности сохранения страниц с заметками и записей настроения в базе данных.

В процессе выполнения дипломного проекта будет произведен анализ существующих решений в данной области, сформулированы функциональные требования, спроектирована архитектура программного продукта, реализованы ключевые модули и компоненты. Кроме того, будет рассмотрено влияние мобильных технологий на здоровье пользователя, а также проведена экономическая оценка эффективности разработки.

Основными задачами, поставленными в дипломной работе, являются:

1. Анализ предметной области и существующих аналогов;
2. Формулировка требований и проектирование архитектуры мобильного приложения;
3. Реализация основных функций и пользовательского интерфейса;
4. Оценка безопасности и эргономики при работе с приложением;
5. Расчёт экономической эффективности и затрат на разработку программного продукта.

# 1 Описание и анализ предметной области

Мы живём в то время, когда главным навыком является адаптируемость и самообучаемость. От скорости и эффективности работы с информацией зависит множество факторов успешной работы. Многие специалисты и студенты, в попытках усвоить как можно больше тем, сталкиваются с большими нагрузками, которые значительно увеличивают уровень стресса. Чтобы помочь справиться с этим, появляется всё больше программных продуктов, нацеленных на планирование и контроль эмоционального состояния.

Несмотря на то, что в последнее время появляется всё больше приложений, позволяющих максимально эффективно распоряжаться своими ресурсами, в большей степени, все они выполняют только одну функцию. Из-за отсутствия комплексного подхода, пользователям приходится использовать несколько различных приложений, в результате чего уровень концентрации падает и на задачу затрачивается больше энергии.

Программный продукт для повышения продуктивности и концентрации в любой деятельности направлен на решение этой проблемы — он объединяет инструменты планирования задач, улучшения концентрации с помощью техники помодоро, отслеживания эмоционального состояния, а также мотивирует пользователя через систему достижений. Дополнительным преимуществом является использование искусственного интеллекта для анализа введенных данных и генерации персонализированных рекомендаций по оптимизации списка задач и улучшению настроения. Всё это делает приложение уникальным помощником в достижении баланса между продуктивностью и ментальным здоровьем.

Пользователи смогут:

* Создавать и группировать задачи по страницам,
* Отслеживать свое настроение и видеть статистику в виде графика,
* Использовать таймер по технике помодоро для улучшения концентрации,
* Получать советы от искусственного интеллекта в зависимости от задач и эмоционального состояния,
* Получать достижения за ежедневное использование приложения.

Таким образом, программный продукт для повышения продуктивности и концентрации в любой деятельности станет цифровым спутником осознанной и продуктивной жизни, повышающим эффективность ежедневных задач и поддерживающим эмоциональное здоровье.

# 1.1 Анализ существующих программных решений

Для анализа схожих мобильных приложений, направленных на повышение продуктивности и концентрацию в любой деятельности, были выбраны три программных продукта: Notion, Daylio и Focus To-Do. Каждый из этих продуктов решает определённую задачу: управление задачами, трекинг настроения или контроль фокуса на работе. Однако в отдельности они не предоставляют комплексного подхода, что и обусловило необходимость разработки приложения, объединяющего их сильные стороны.

1. Функциональность:

* Notion: универсальное приложение для заметок. Позволяет создавать вложенные страницы, таблицы, списки задач и шаблоны. Обладает гибкой структурой, возможностью коллаборации и интеграциями с другими сервисами. Однако не включает встроенных инструментов для анализа эмоционального состояния или поддержки продуктивности в реальном времени.
* Daylio: минималистичный трекер настроения, ориентированный на ежедневные записи с использованием смайликов, тегов и коротких заметок. Позволяет строить графики, анализировать тенденции, вести дневник. Отсутствуют функции задач, таймеров и поддержки фокусной работы.
* Focus To-Do: приложение, сочетающее список задач с таймером по методике помодоро. Пользователь может задавать длительность фокус-сессий, видеть статистику, использовать уведомления. Однако не предоставляет инструментов для анализа настроения или глубокого планирования.

1. Удобство использования:

* Notion: интерфейс интуитивно понятный, но требует времени на освоение. Многофункциональность оборачивается сложностью для новичков.
* Daylio: простой и интуитивно понятный интерфейс, позволяющий быстро фиксировать настроение в несколько кликов. Не требует обучения.
* Focus To-Do: четкая структура интерфейса. Пользователь легко может настроить таймер и список задач. Подходит для начинающих и опытных пользователей.

1. Эффективность:

* Notion: высокая, особенно при индивидуальной настройке и систематическом использовании. Но эффективность сильно зависит от мотивации пользователя.
* Daylio: обеспечивает регулярную фиксацию настроения, что может способствовать рефлексии и улучшению осознанности.
* Focus To-Do: высокая эффективность за счёт использования проверенной техники помодоро, особенно для выполнения задач в ограниченное время.

1. Надёжность и безопасность:

* Notion: стабильная работа, поддержка кроссплатформенности, облачная синхронизация, доступ через браузер и приложение.
* Daylio: надёжная работа, поддержка локального и облачного хранения.
* Focus To-Do: регулярные обновления, стабильная работа, поддержка синхронизации между устройствами.

Каждое из рассмотренных приложений решает одну задачу, но не охватывает всё необходимое для комплексной поддержки продуктивности и ментального здоровья. Программный продукт для повышения продуктивности и концентрации в любой деятельности объединяет лучшие элементы этих решений: гибкость и визуализацию как в Notion, трекинг настроения и статистику как в Daylio, таймер по технике помодоро как в Focus To-Do, а также дополняет их системой достижений и советами от искусственного интеллекта.

Для наглядного сравнения рассмотренных мобильных приложений, предназначенных для повышения продуктивности и концентрации, в таблице 1.1 приведен их сравнительный анализ по ключевым критериям: функциональность, удобство использования, эффективность, надежность и безопасность, поддержка ИИ, а также предполагаемая целевая аудитория.

Таблица 1.1 – Сравнительная таблица аналогов

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | Notion | Daylio | Focus To-Do |
| Функциональность | Универсальный редактор со списками и шаблонами. | Трекер настроения с тегами, заметками, статистикой. | Список задач с с помодоро-таймером и напоминаниями. |
| Удобство использования | Гибкий, но требует времени на изучение. | Очень простой, быстрый ввод настроения. | Четкая структура, простота настройки. |
| Эффективность | Высокая при правильной настройке и дисциплине. | Помогает вести рефлексию и осознанность. | Высокая за счет сессий фокусировки |
| Надежность и безопасность | Стабильная синхронизация, облачное хранение. | Безопасность, локальные и облачные копии. | Регулярные обновления. |
| Продолжение таблицы 1.1 | | | |
| Критерий | Notion | Daylio | Focus To-Do |
| Поддержка ИИ | Присутствует. | Отсутствует. | Отсутствует. |
| Целевая аудитория | Студенты, специалисты, любители гибкого планирования. | Пользователи, стремящиеся отслеживать настроение. | Люди, работающие по технике Pomodoro. |

# 1.2 Функциональные требования к мобильному приложению

В системе программного продукта для повышения продуктивности и концентрации в любой деятельности реализована одна основная роль — пользователь, который получает доступ ко всем ключевым функциям приложения. Разграничение прав доступа по типу администратора не предусмотрено, так как архитектура построена вокруг индивидуального использования и хранения персональных данных.

Пользователь — основная и единственная роль в системе. Она предназначена для студентов, специалистов, фрилансеров и всех, кто стремится повысить уровень своей организованности и эмоционального комфорта. Каждый пользователь может:

1. Создавать страницы — самостоятельные рабочие пространства (например: "Учёба", "Проект", "Личное").
2. Создавать и управлять задачами на каждой странице:

* добавление по нажатию Enter,
* автоматическое сохранение,
* редактирование, удаление,
* отметка выполнения задачи.

1. Отслеживать настроение — фиксировать ежедневное состояние, выбирая из пяти категорий (ужасное, грустное, нормальное, хорошее, отличное).
2. Просматривать статистику настроения в виде графиков по дням.
3. Использовать таймер по технике помодоро с тремя режимами: фокус, короткий и длинный перерыв.

* настройка длительности режимов;
* переход между режимами;
* уведомления об окончании сессии.

1. Получать персонализированные советы от ИИ:

* по улучшению настроения на основе введенных данных;
* по выполнению задач и планированию.

1. Получать достижения за активность, использование функций и прогресс:

* создание первой задачи;
* использование таймера;
* выполнение задач подряд несколько дней;;
* отметка настроения, включение тёмной темы и др.

1. Переключать тему оформления — светлая и тёмная, с сохранением предпочтений.

Приложение не предусматривает авторизации по ролям или централизованного управления. Такой подход делает его приватным, автономным и максимально удобным для персонального применения.

Функциональные требования отражают потребности в гибком, но простом инструменте для повседневного использования, направленном на повышение продуктивности и заботу о себе. Они также служат основой для построения интерфейса, выбора технологий и архитектурных решений, применяемых в проекте.

# 1.3 Выбор средства разработки

При разработке программного продукта для повышения продуктивности и концентрации основными критериями выбора инструментов и технологий стали: кроссплатформенность, простота интеграции, гибкость пользовательского интерфейса, поддержка современного стека JavaScript/TypeScript и возможность масштабирования. Особое внимание уделялось также удобству работы с базами данных, управлению состоянием и интеграции искусственного интеллекта.

В результате анализа были выбраны следующие средства разработки:

1. React Native — основной фреймворк для кроссплатформенной разработки мобильных приложений. Позволяет использовать один код для Android и iOS, снижая затраты времени и ресурсов. Обеспечивает поддержку функциональных компонентов, хуков и быструю сборку интерфейса.
2. TypeScript — надстройка над JavaScript, обеспечивающая строгую типизацию. Использование TypeScript повышает надёжность кода и снижает количество потенциальных ошибок на этапе разработки.
3. Expo — инструмент и платформа для ускоренной разработки на базе React Native. Упрощает работу с доступом к нативным API (камере, уведомлениям, хранилищу), а также обеспечивает быструю отладку и сборку приложения.
4. Firebase Firestore — облачная NoSQL база данных от Google. Используется для хранения всех пользовательских данных: задач, страниц, отметок настроения, достижений и т.п. Обеспечивает синхронизацию данных в реальном времени и оффлайн-доступ.
5. AsyncStorage — локальное хранилище, предназначенное для сохранения пользовательских настроек (например, тема оформления, уникальный идентификатор пользователя) и кэша.
6. Meta AI через OpenRouter — внешнее API, предоставляющее доступ к языковым моделям. Применяется для генерации персонализированных советов пользователю, как на основе настроения, так и в контексте задач.
7. PlantUML — инструмент для визуального моделирования программных систем с использованием простого текстового описания. Позволяет быстро создавать диаграммы прецедентов, классов, состояний, последовательностей и других UML-диаграмм.

Выбор данных инструментов обеспечил баланс между функциональностью, скоростью разработки и удобством поддержки проекта. Все технологии являются актуальными, активно поддерживаются сообществом и подходят как для учебных, так и для реальных коммерческих проектов.

# 2 Реализация и проектирование мобильного приложения

Проектирование программного продукта для повышения продуктивности и концентрации в любой деятельности, является важным этапом в создании любого мобильного приложения. Он позволяет проанализировать базовые потребности пользователей при использовании программного продукта.

Интуитивно понятный интерфейс, приятное визуальное оформление и система достижений привлекает пользователей и мотивирует чаще заходить в приложение. Персонализированные советы от искусственного интеллекта помогают пользователю чувствовать поддержку, что особенно важно при эмоциональном напряжении.

Повышение продуктивности достигается благодаря функции создания и планирования задач на отдельных страницах, персонализированным советам, позволяющим максимально эффективно решить поставленные задачи, а также интеграции таймера по технике помодоро с настройкой длительности сессий фокусировки и отдыха. Это позволяет пользователю лучше управлять временем, вниманием, избегать прокрастинации и сохранять продуктивность в течение дня.

Цифровая поддержка ментального состояния реализована через создание отметок настроения и просмотр графика статистики, что позволяет пользователю отслеживать свое настроение в разные дни. Такой функционал способствует развитию осознанности, а также помогает вовремя выявлять отклонения и регулировать уровень внутреннего комфорта с помощью персонализированных советов.

Таким образом, проектирование программного продукта для повышения продуктивности и концентрации в любой деятельности ориентировано на создание сбалансированного и доступного инструмента, сочетающего функции управления задачами, контроля настроения и концентрации. Подобный подход повышает ценность продукта с точки зрения конечного пользователя.

# 2.1 Диаграмма классов

Диаграмма классов это основной инструмент визуального моделирования в языке UML (Unified Modeling Language). Он применяется для описания логической структуры программных решений, включая ключевые сущности, их характеристики и взаимосвязи. В программном продукте для повышения продуктивности и концентрации в любой деятельности диаграмма классов позволяет наглядно представить архитектуру проекта и структуру данных, хранящихся в облачной базе Firebase Firestore.

На диаграмме отображены основные сущности, реализованные в виде коллекций Firestore: Page (страницы), Task (задачи), Mood (записи настроения), и Awards (достижения). Каждая из коллекций содержит документы с уникальными идентификаторами и соответствующими атрибутами, отражающими логику работы приложения.

Связи между сущностями реализуются через идентификаторы:

* каждая задача (Task) связана с конкретной страницей (Page) через поле page\_id;
* задачи (Task), страницы (Page), отметки настроений (Mood) и достижения (Award) связаны с пользователем через поле user\_id.
* Дополнительно в архитектуре присутствует внешний компонент AIService — это модуль, обращающийся к языковой модели через OpenRouter API. Он используется для генерации персонализированных советов на основе задач или отметок о настроении пользователя. AIService не является частью постоянного хранилища и не сохраняет данные в базе данных.
* Использование диаграммы классов в рамках проекта решает ряд задач:
* Визуализация архитектуры данных — демонстрирует структуру коллекций, поля и связи между ними;
* Выделение ключевых сущностей — таких как Page, Task, Mood,, Awards;
* Упрощение сопровождения и развития проекта — облегчает понимание логики хранения и обработки данных;
* Документирование проекта — диаграмма служит важным элементом технической документации.

На рисунке 2.1 представлена диаграмма классов, отражающая структуру программного продукта для повышения продуктивности и концентрации в любой деятельности. Главным элементом выступает связь между страницами (Page) и задачами (Task), а также вспомогательные модули: трекер настроения (Mood), система достижений (Awards) и внешний модуль AIService, взаимодействующий с основными компонентами.

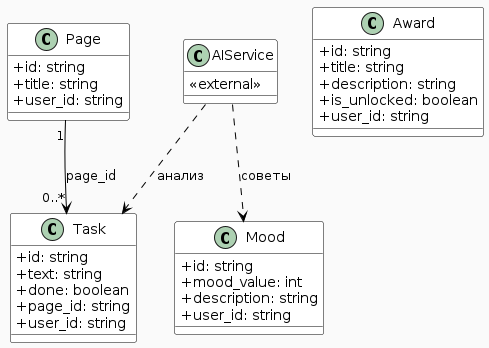


Рисунок 2.1 – Диаграмма классов UML мобильного приложения

# 2.2 Проектирование диаграммы Use-Case

Диаграмма вариантов использования (Use Case Diagram), используется для отображения того, какие действия может выполнять пользователь, и какие функции приложения при этом задействуются.

На диаграмме актор (пользователь) изображается в виде человека и представляет собой внешнего участника, взаимодействующего с приложением. Каждый прецедент (вариант использования) представлен в виде овала и описывает конкретное действие, доступное пользователю.

Поскольку программный продукт для повышения продуктивности и концентрации в любой области предназначен для индивидуального использования и не предусматривает разделения прав доступа, единственным актором является пользователь, обладающий доступом ко всем функциям системы. Он обладает следующими вариантами использования:

* Создание и управление задачами (добавление, редактирование, удаление, выполнение)
* Создание страниц для организации задач
* Создание отметок о настроении
* Просмотр статистики настроения
* Использование таймера по технике помодоро
* Получение советов от искусственного интеллекта
* Получение достижений за активность
* Переключение темы оформления

Данная диаграмма вариантов использования представлена на рисунке 2.2.

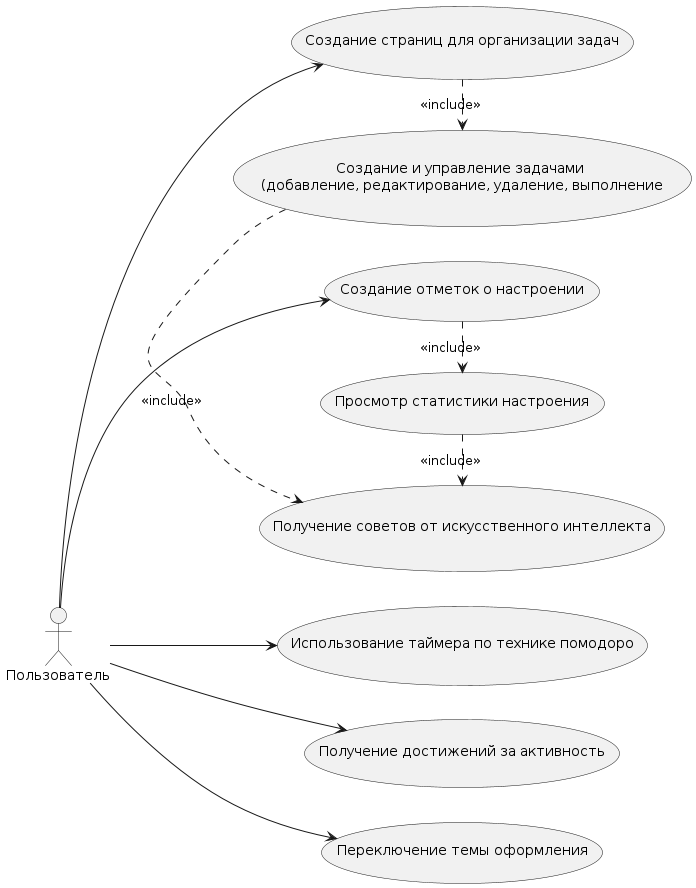


Рисунок 2.2 – Диаграмма use-case

# 2.3 Проектирование ER-диаграммы

На данном этапе проектирования сформирована логическая структура базы данных программного продукта для повышения продуктивности и концентрации в любой деятельности. Она основана на использовании Firebase Firestore — облачного NoSQL-хранилища, предоставляющего хранение данных в виде коллекций и документов. В отличие от реляционных баз данных, Firestore не использует строгие связи (foreign keys), а работает с гибкой структурой, где данные объединяются по идентификаторам и хранятся независимо.

Основные коллекции и их структура:

Pages (Страницы)

* id — уникальный идентификатор страницы
* title — название страницы
* user\_id — уникальный идентификатор пользователя

Tasks (Задачи)

* id — уникальный идентификатор задачи
* text— текст задачи
* done — статус выполнения
* page\_id — идентификатор страницы, к которой относится задача
* user\_id — уникальный идентификатор пользователя

Mood (Записи настроения)

* id — уникальный идентификатор записи
* mood\_value — уровень настроения
* description — комментарий
* user\_id — уникальный идентификатор пользователя

Awards (Достижения)

* user\_id — уникальный идентификатор пользователя
* title — название достижения
* is\_unlocked — статус получения (boolean)

Структура связей между сущностями:

* Page → Task — один ко многим: каждая страница содержит множество задач, связанные по page\_id.
* Остальные коллекции (Mood, Awards) автономны и не требуют связей между собой.
* Все данные сохраняются в контексте одного пользователя (user\_id), что упрощает структуру и позволяет легко фильтровать и разделять данные по пользователям.

Таким образом, база данных приложения построена вокруг четырех коллекций. Такая структура обеспечивает гибкость, масштабируемость и простоту реализации.

На рисунке 2.3 представлена логическая модель базы данных.

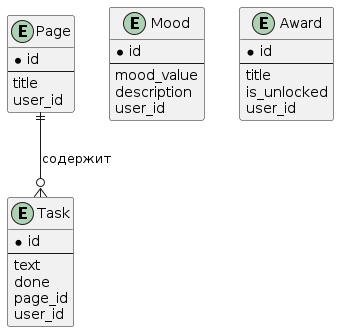


Рисунок 2.3 – Логическая модель базы данных

На рисунке 2.4 представлена физическая модель базы данных.

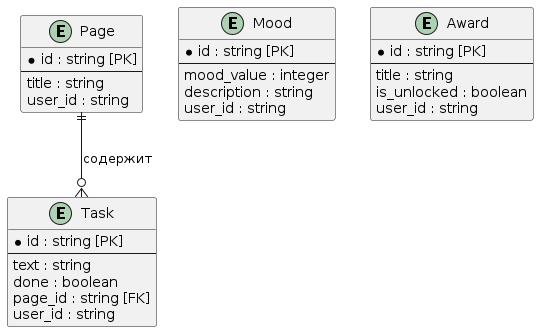


Рисунок 2.4 – Физическая модель базы данных

# 2.4 Руководство пользователя

При запуске приложения пользователя встречает окно загрузки с логотипом приложения. После загрузки он попадает на панель заметок. Здесь находятся созданные страницы.

Пользователь может набрать в поле ввода название страницы и нажать на “Enter” на клавиатуре. После чего появится новая страница.

Пользователь может один раз нажать на страницу и активируются 2 кнопки “Удалить страницу” и “Изменить страницу”. По нажатию на кнопку “Удалить страницу”, страница и все содержащиеся в ней заметки удалятся. По нажатию кнопки “Изменить страницу” в поле ввода появится название данной страницы, которое можно отредактировать и нажать “Enter”.

При долгом нажатии на страницу, пользователь переходит к заметкам данной страницы. Здесь таким же образом, как и при создании новой страницы он может создавать заметки.

После того, как пользователь создал заметки, активируется кнопка “Совет”. Нажав на неё, появится всплывающее окно с советом от нейросети по тому, как более эффективно решить задачи, указанные в заметках.

Нажав на заметку, пользователь может её выполнить. Визуально это отображается зачеркиванием текста заметки.

Долгим нажатием на заметку, пользователь может выделить её. После чего активируются 2 кнопки: “Удалить заметку” и “Изменить заметку”. Их функционал схож с кнопками на панели страниц.

Нажав на кнопку “Назад”, обозначенную стрелкой влево, пользователь вернется к панели страниц.

Вышеописанные действия представлены на рисунках 2.5 – 2.18.



Рисунок 2.5 – Окно загрузки

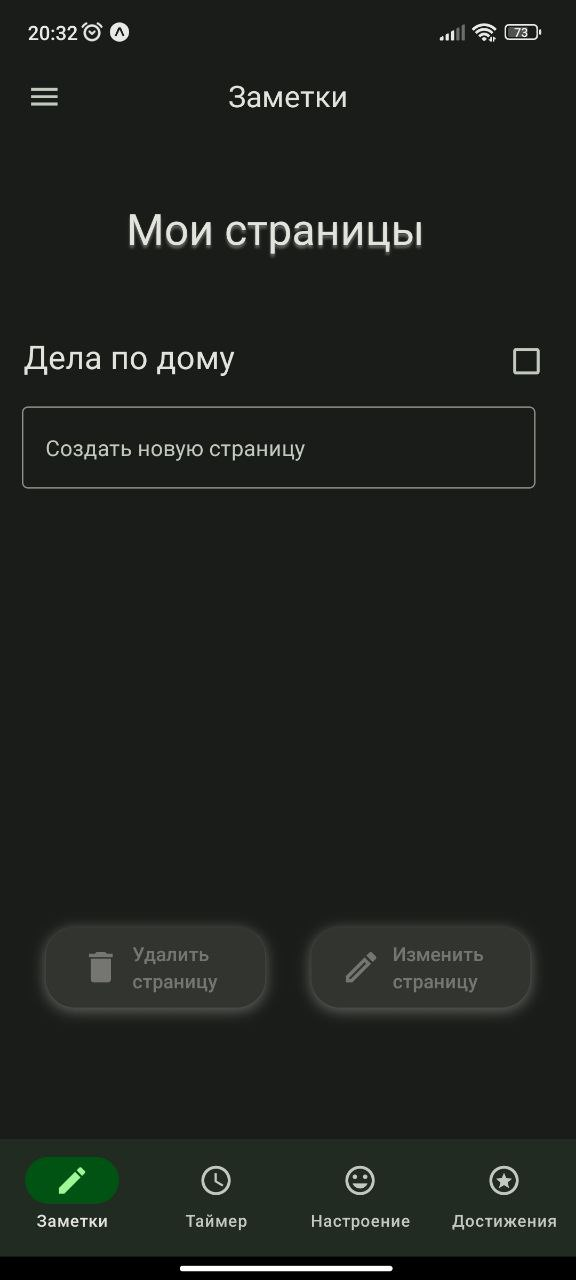


Рисунок 2.6 – Панель “Заметки”

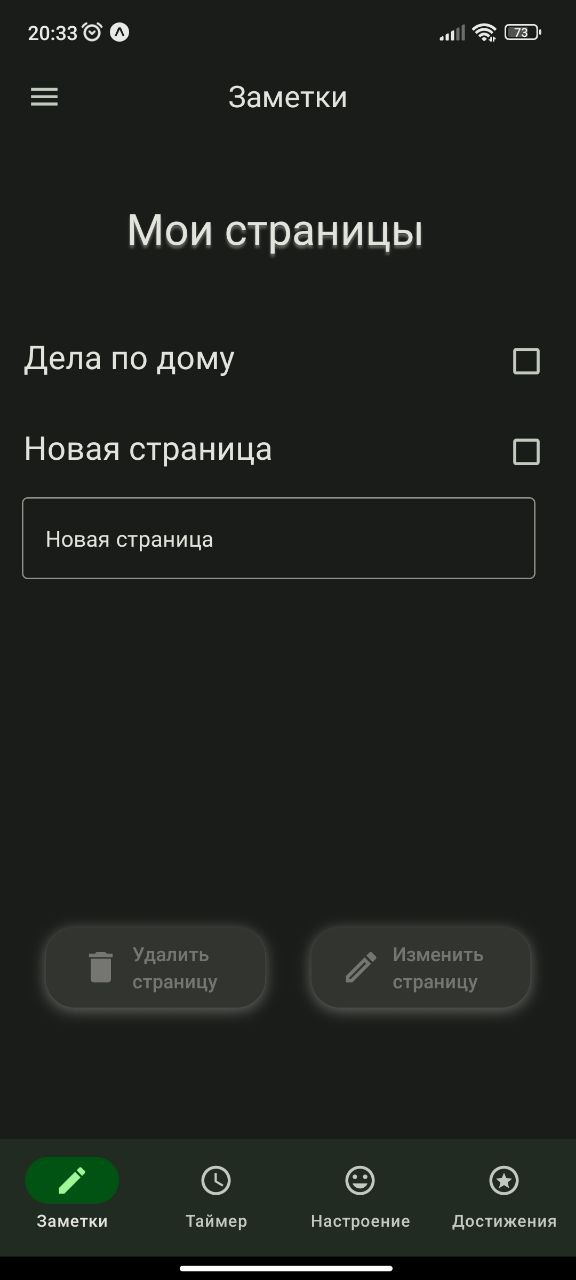


Рисунок 2.7 – Созданная страница

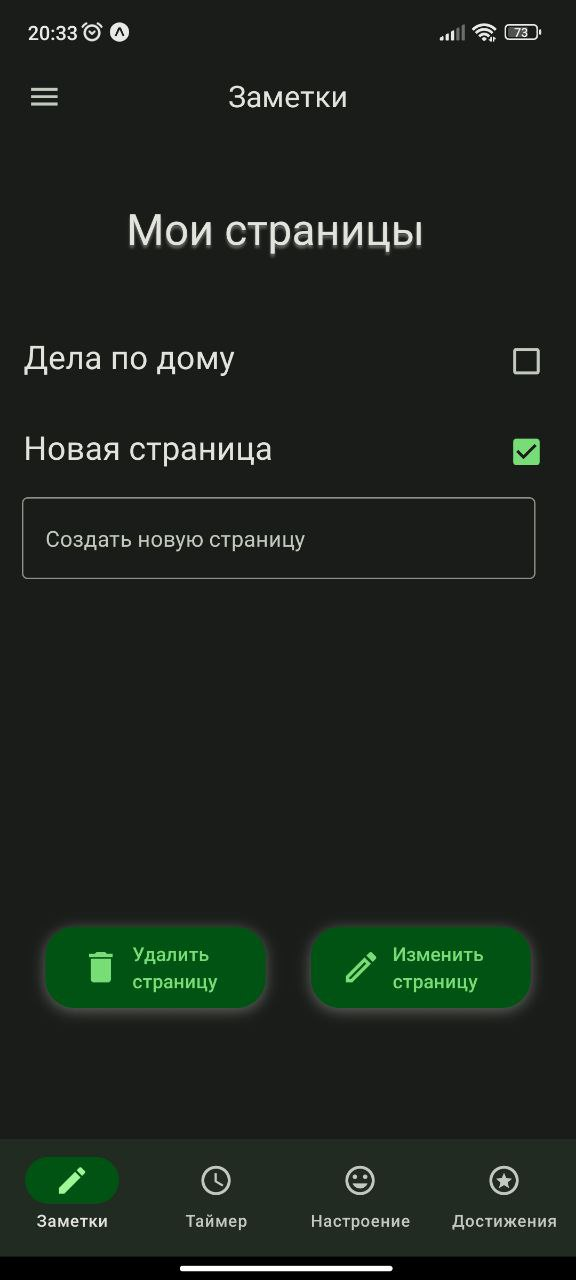


Рисунок 2.8 – Выбранная страница

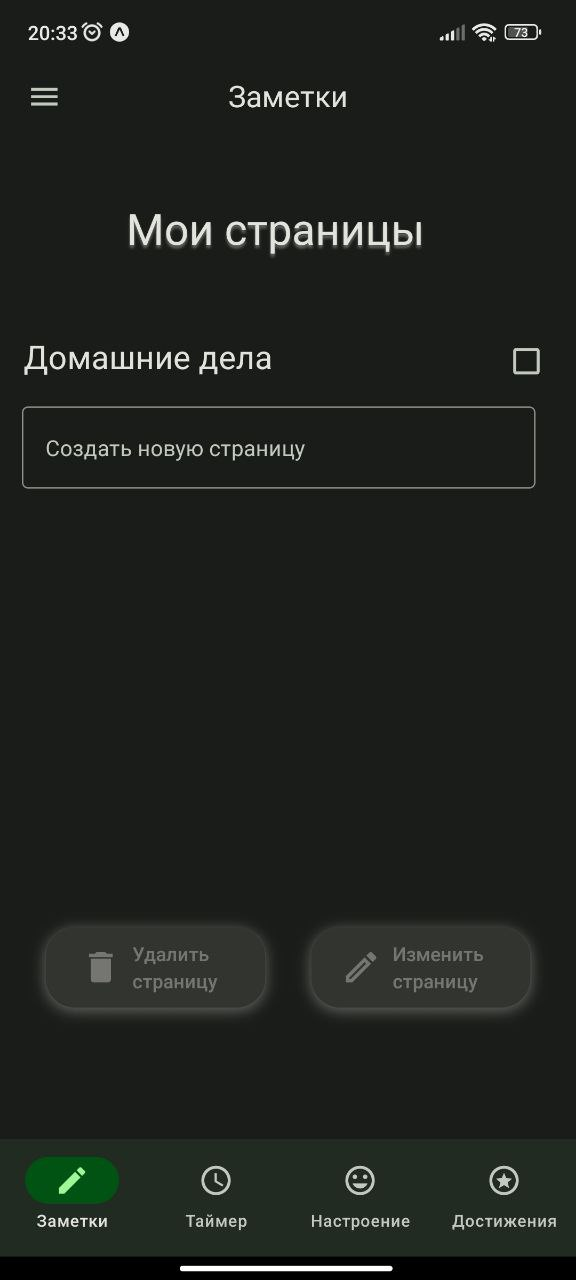


Рисунок 2.9 – Удалённая и измененная страница

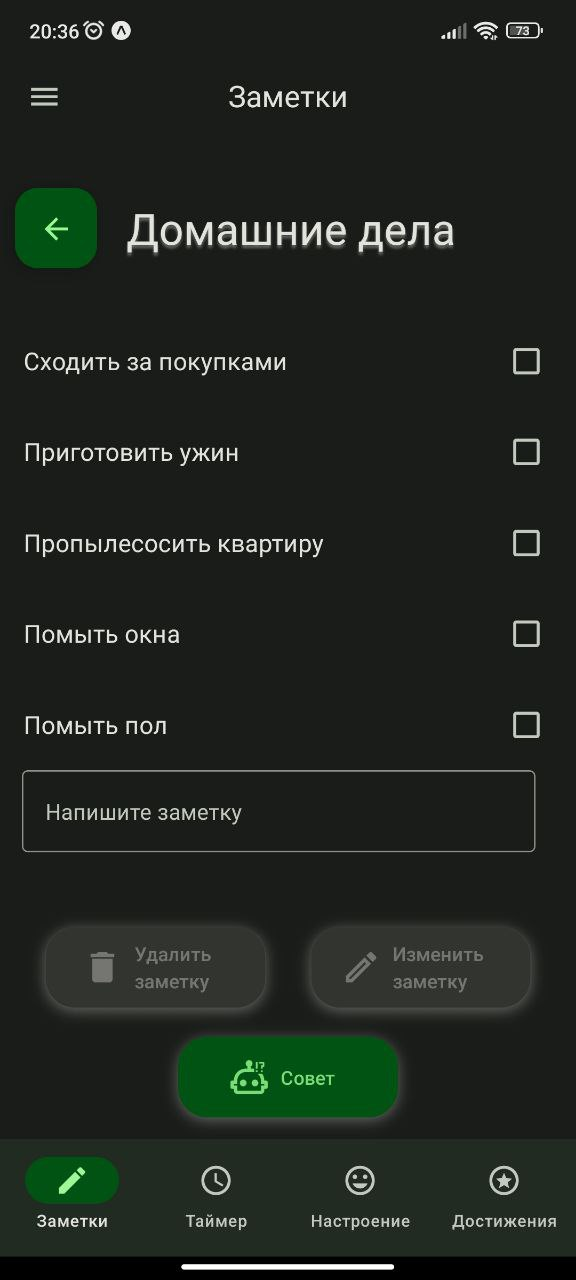


Рисунок 2.10 – Список заметок

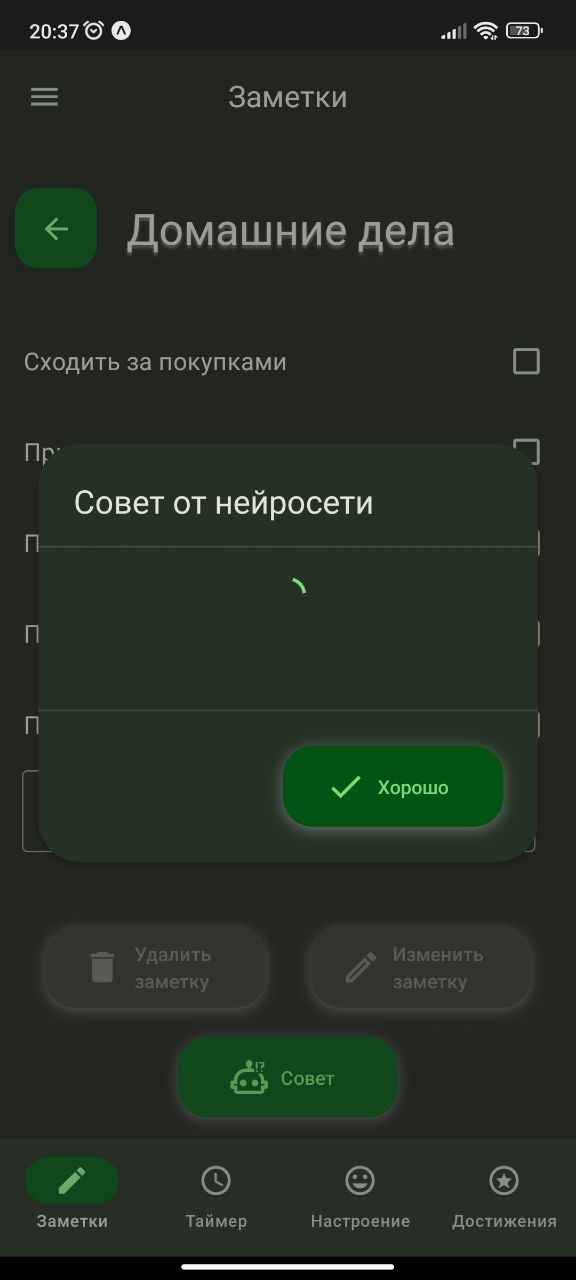


Рисунок 2.11 – Окно загрузки совета

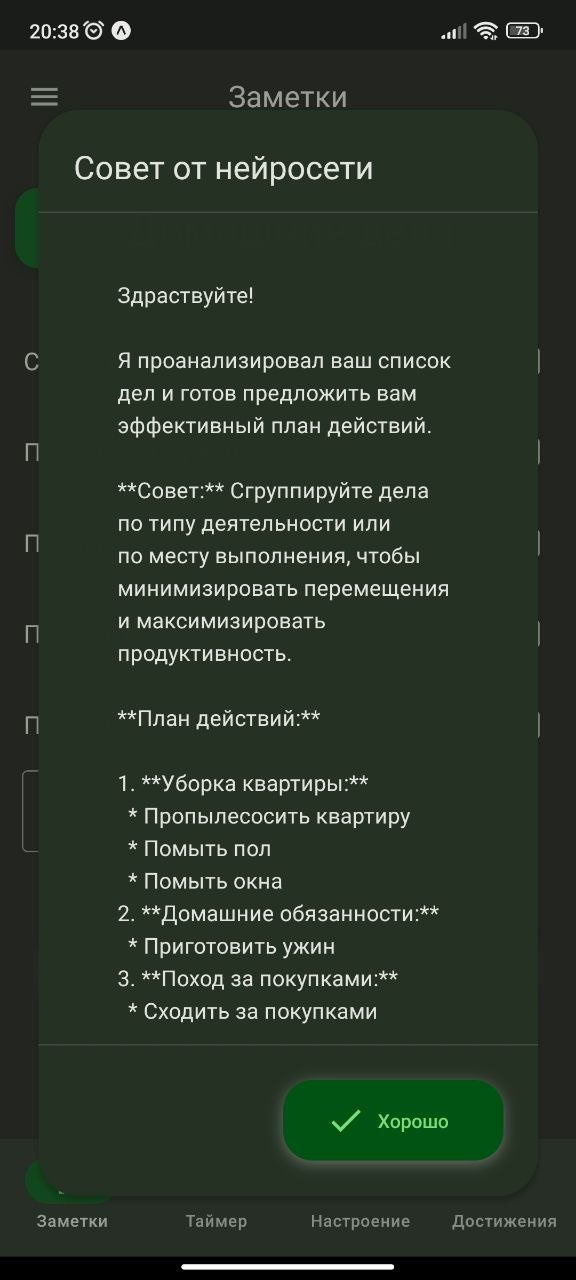


Рисунок 2.12 – Окно совета от нейросети

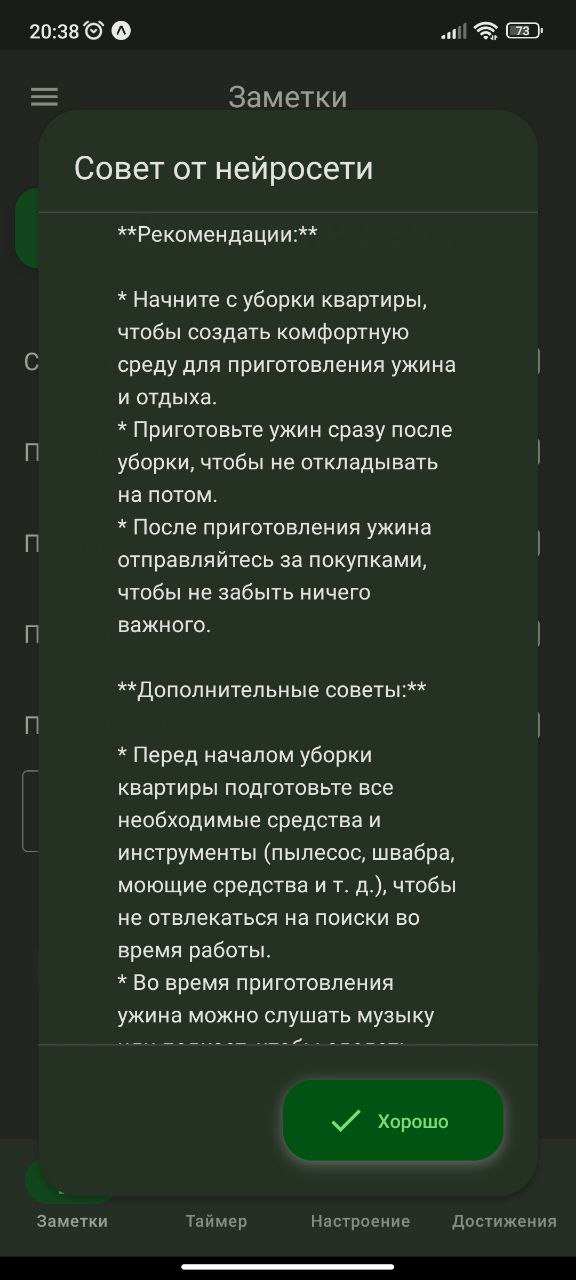


Рисунок 2.13 – Окно совета от нейросети

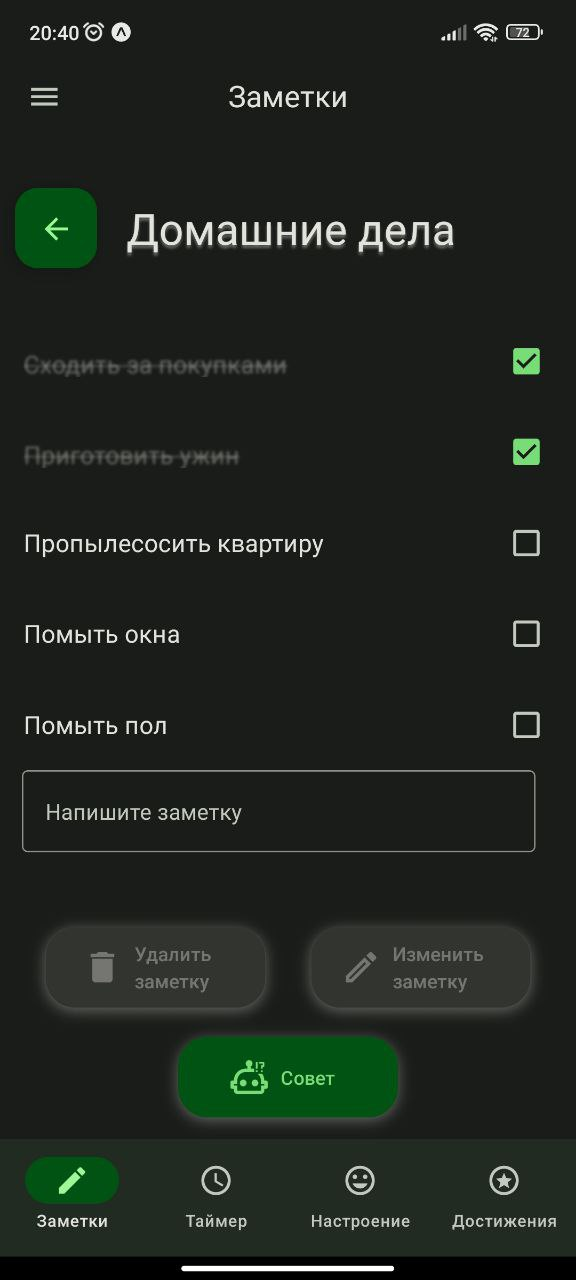


Рисунок 2.14 – Выполненные заметки

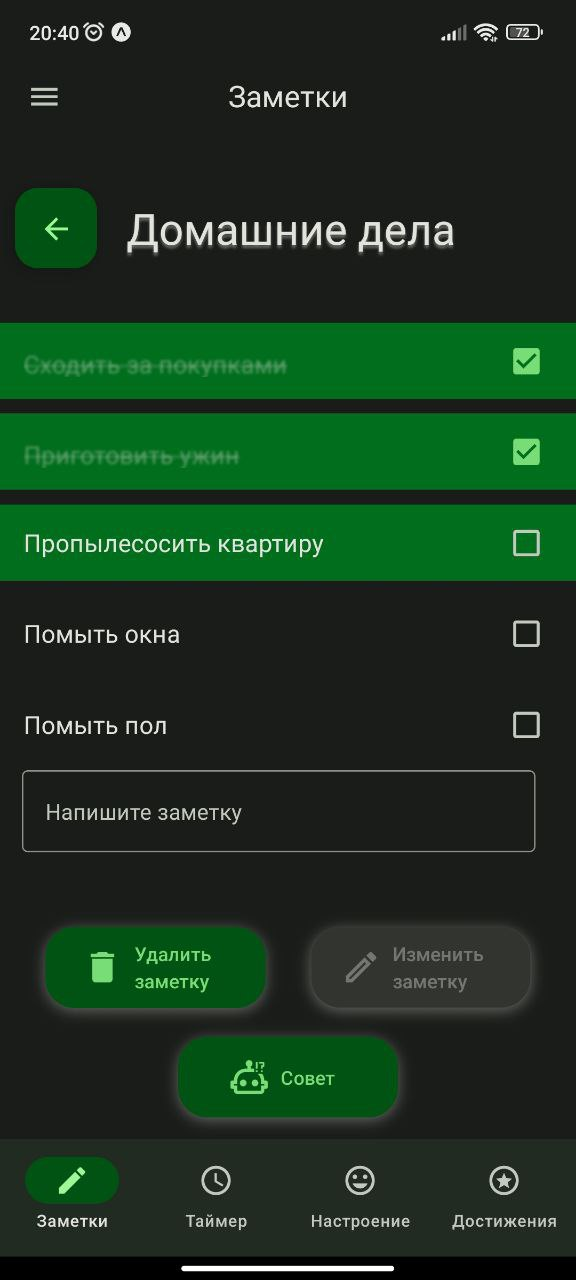


Рисунок 2.15 – Выделенные заметки

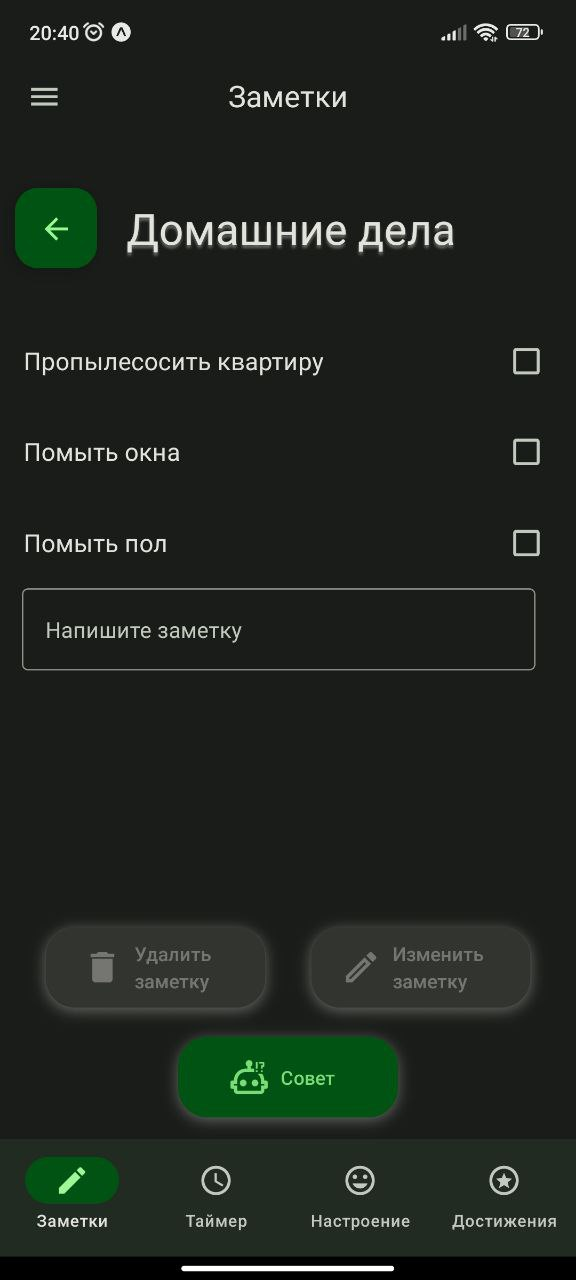


Рисунок 2.16 – Удаленные заметки

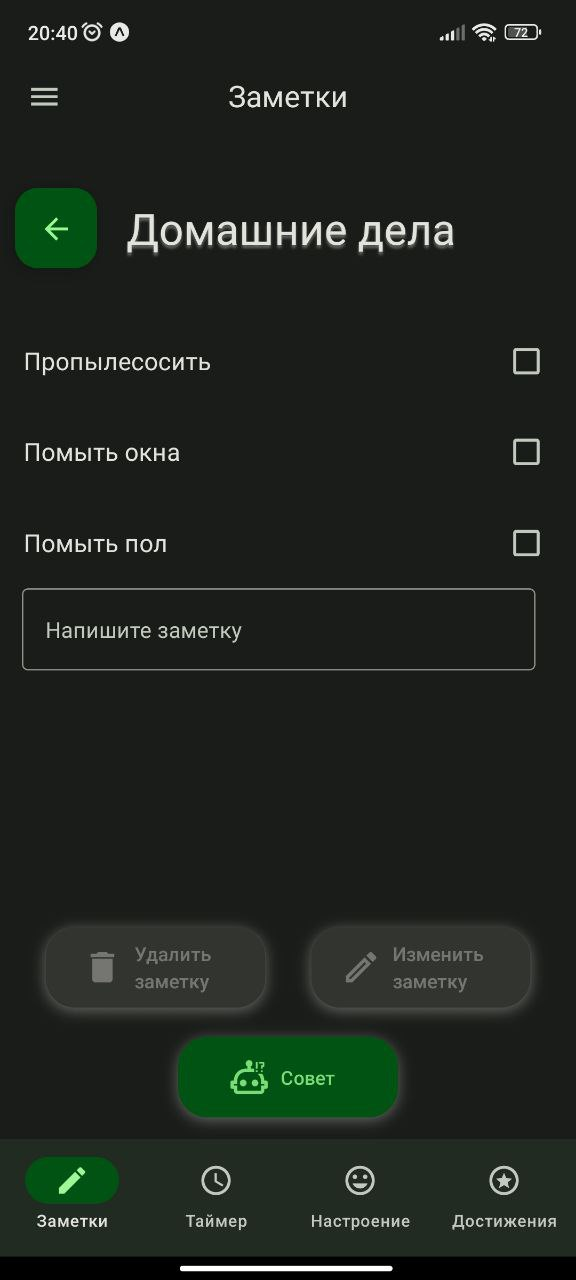


Рисунок 2.17 – Измененная заметка

При нажатии на “Таймер” в нижнем меню, пользователь переходит на страницу таймера.

Здесь пользователь может запустить таймер “помодоро”, который чередует периоды фокусировки и перерыва. После 25-ти минут фокусировки, идёт 5 минут перерыва. После 4-х кругов наступает большой перерыв, который длится 15 минут. Длительность периодов можно изменить в настройках

Пользователь может запустить таймер нажатием на кнопку “Пуск”, остановить его нажатием на кнопку “Пауза” и пропустить круг нажатием на кнопку “Дальше”

На таймере отображается название периода и оставшееся время. По истечению периода, пользователю приходит уведомление.

Вышеописанные действия представлены на рисунках 2.18 – 2.24.

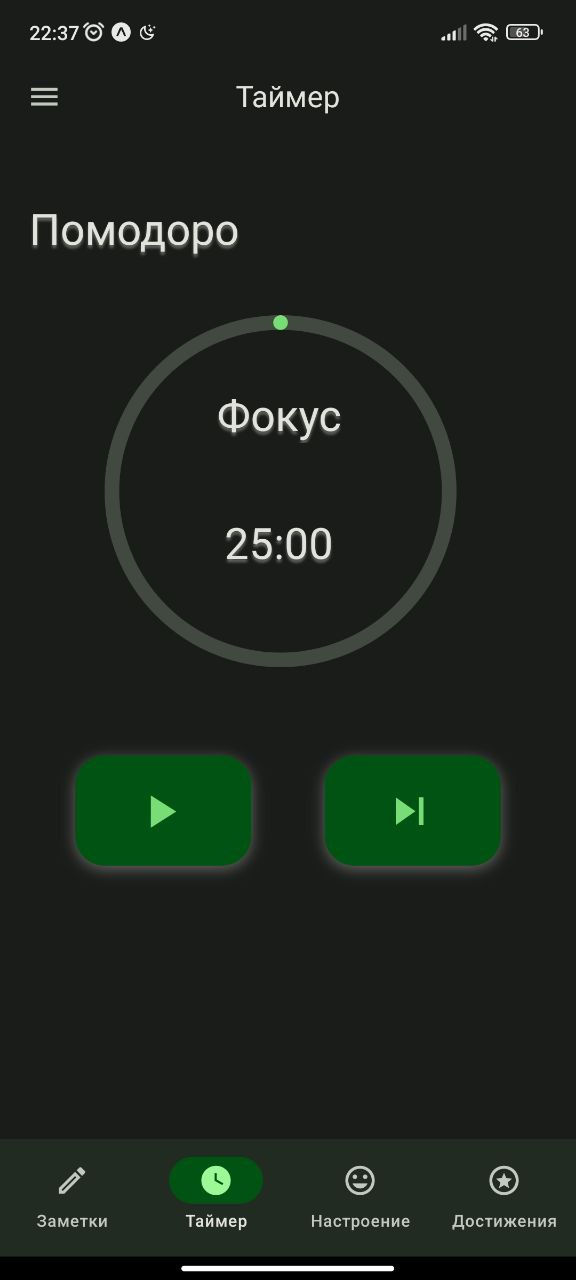


Рисунок 2.18 – Панель “Таймер”

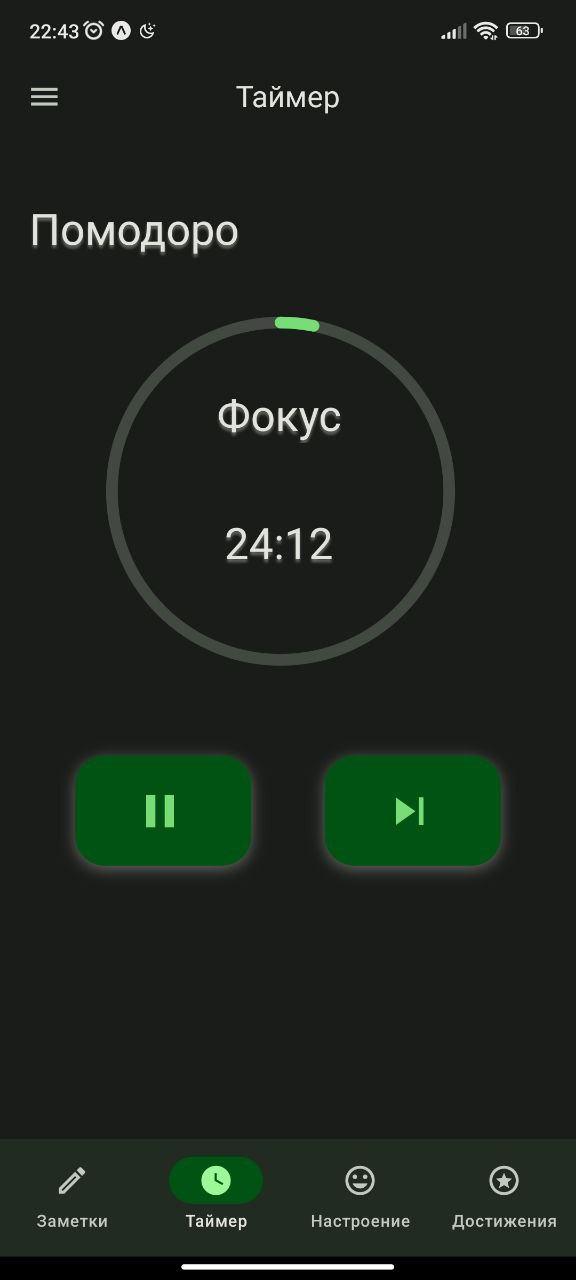


Рисунок 2.19 – Запуск таймера

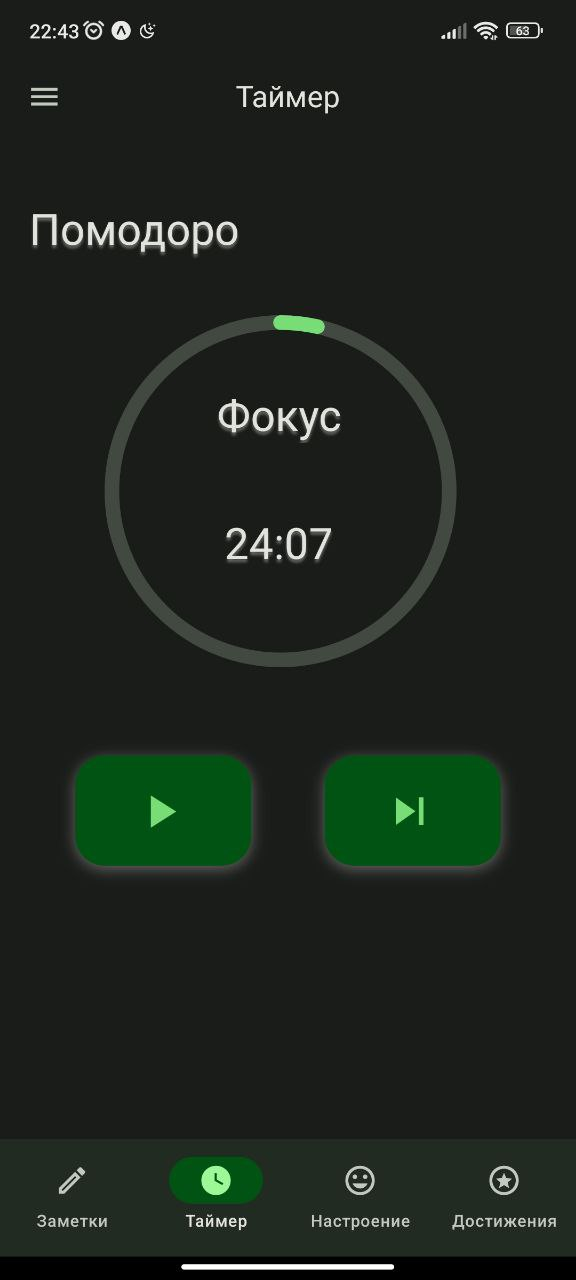


Рисунок 2.20 – Остановка таймера

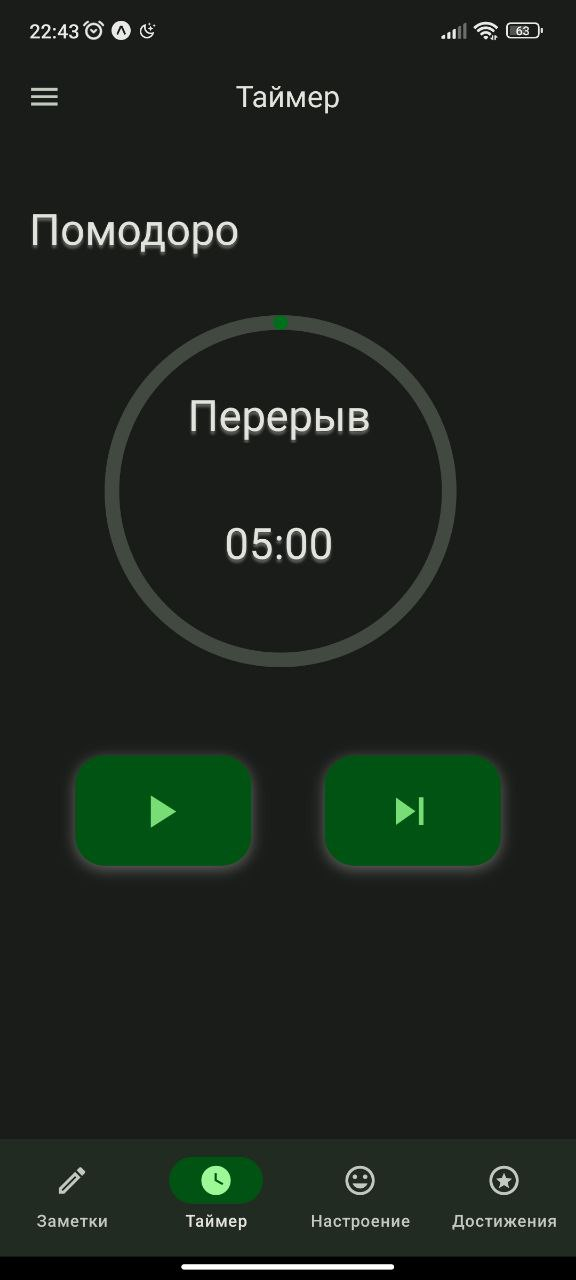


Рисунок 2.21 – Перерыв

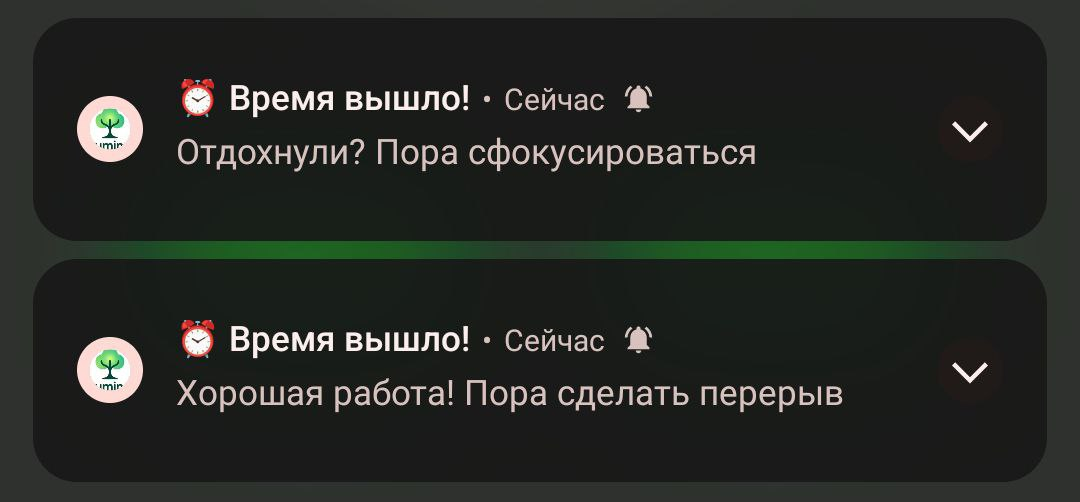


Рисунок 2.22 – Уведомление о перерыве

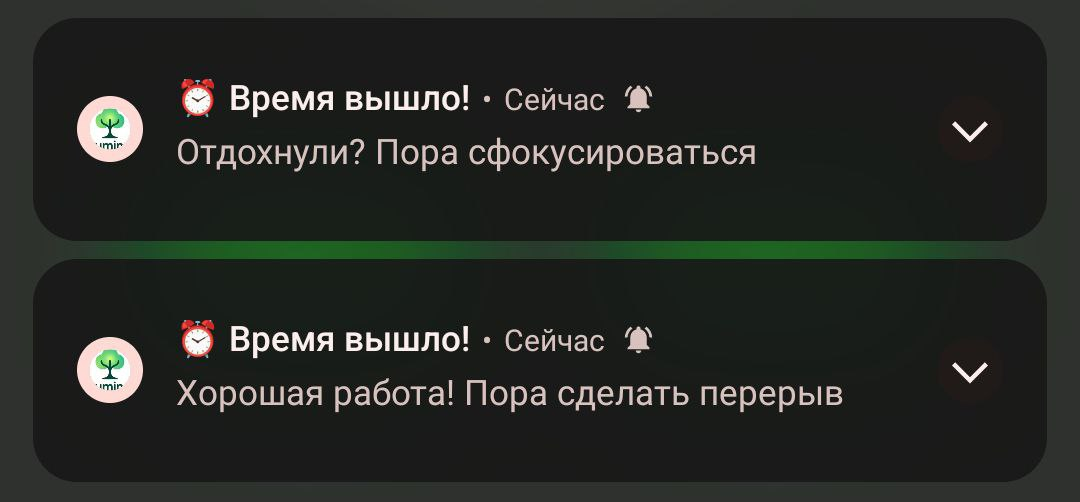


Рисунок 2.23 – Уведомление о фокусировке

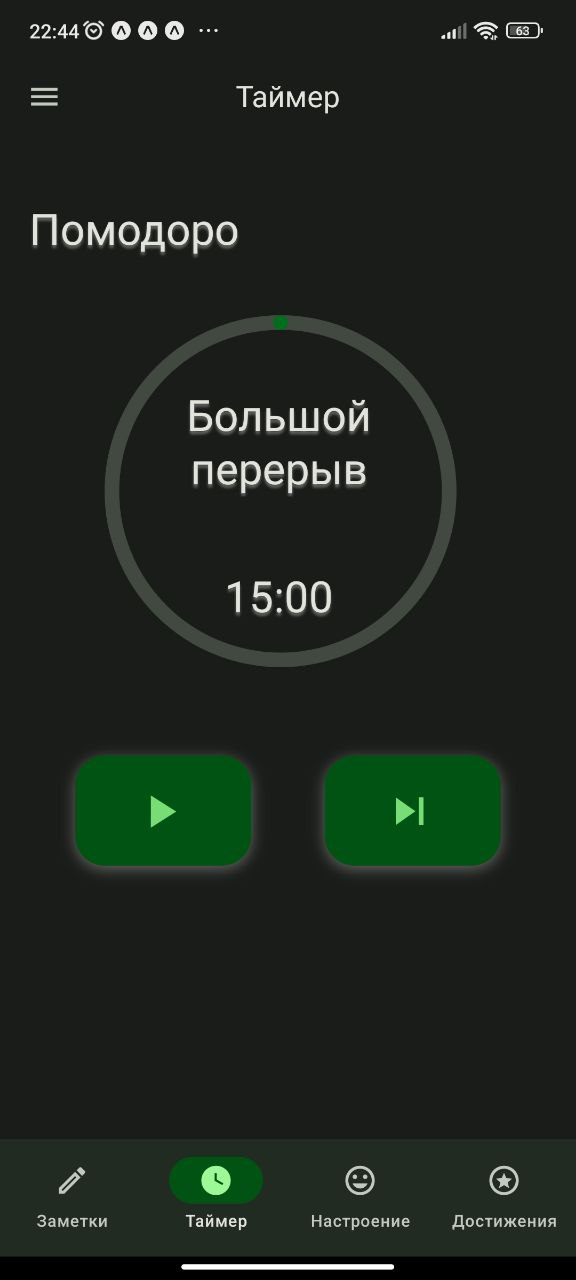


Рисунок 2.24 – Большой перерыв

При нажатии на “Настроение” в нижнем меню, пользователь переходит на панель трекера настроений.

Здесь пользователь может выбрать своё текущее настроение, нажав на один из “смайликов”. Затем, он может описать свое настроение, введя текст в поле “Опишите свое настроение”. По нажатии на кнопку с иконкой карандаша, новая запись о настроении появляется в списке.

Список отображает все записи, которые создавал пользователь, их дату, время и описание. Над списком, пользователь может увидеть график своего настроения, составленный из записей, и проследить в какой день какое настроение у него было.

После создания записи, активируется кнопка “Совет”, нажав на которую, пользователь может получить персонализированный совет от нейросети по тому, как улучшить текущее настроение.

Вышеописанные действия представлены на рисунках 2.25 - 2.26

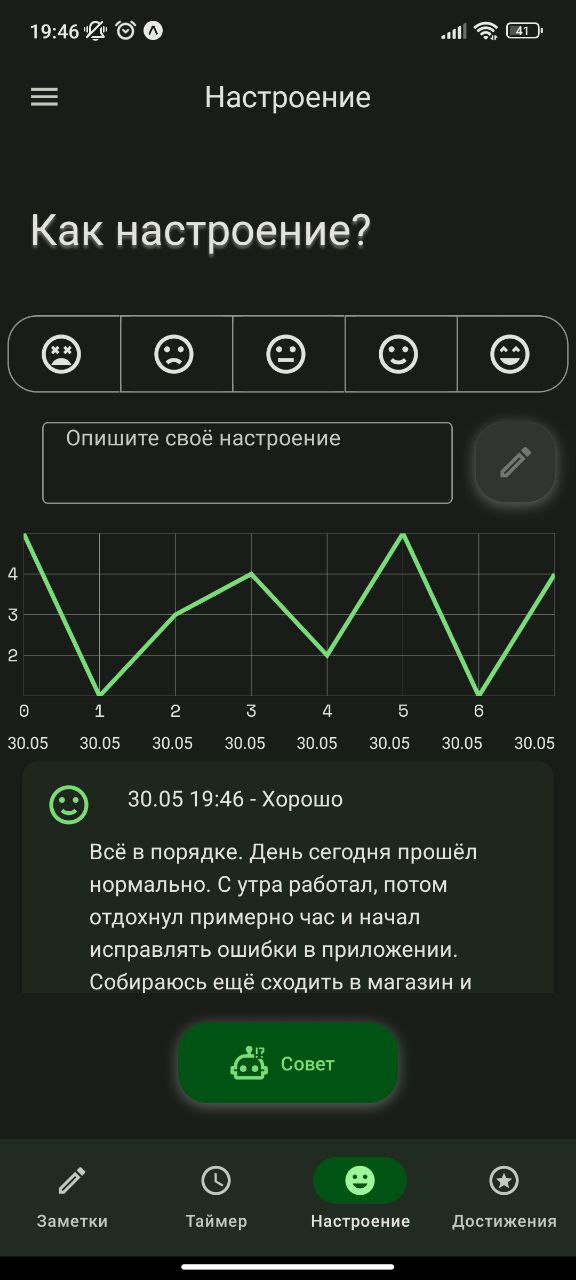


Рисунок 2.25 – Панель настроения

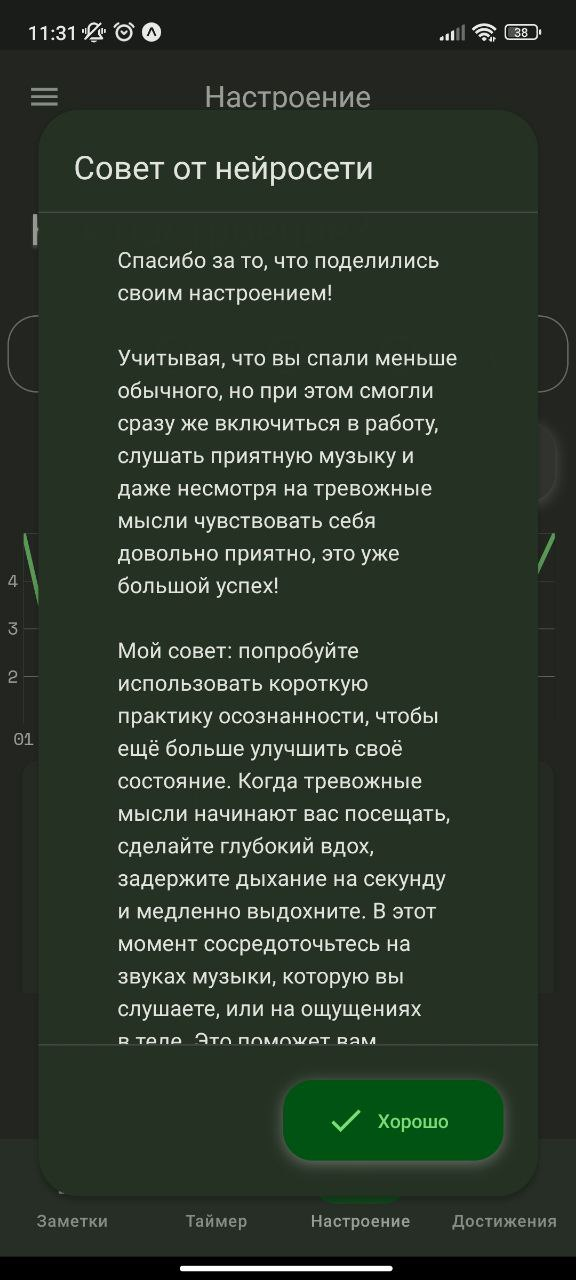


Рисунок 2.26 – Совет по настроению от нейросети

При нажатии на “Достижения” в нижнем меню, пользователь переходит на страницу своих достижений. Выполненные достижения выделены. Невыполненные прозрачны.

Вышеописанные действия представлены на рисунках 2.27-2.29

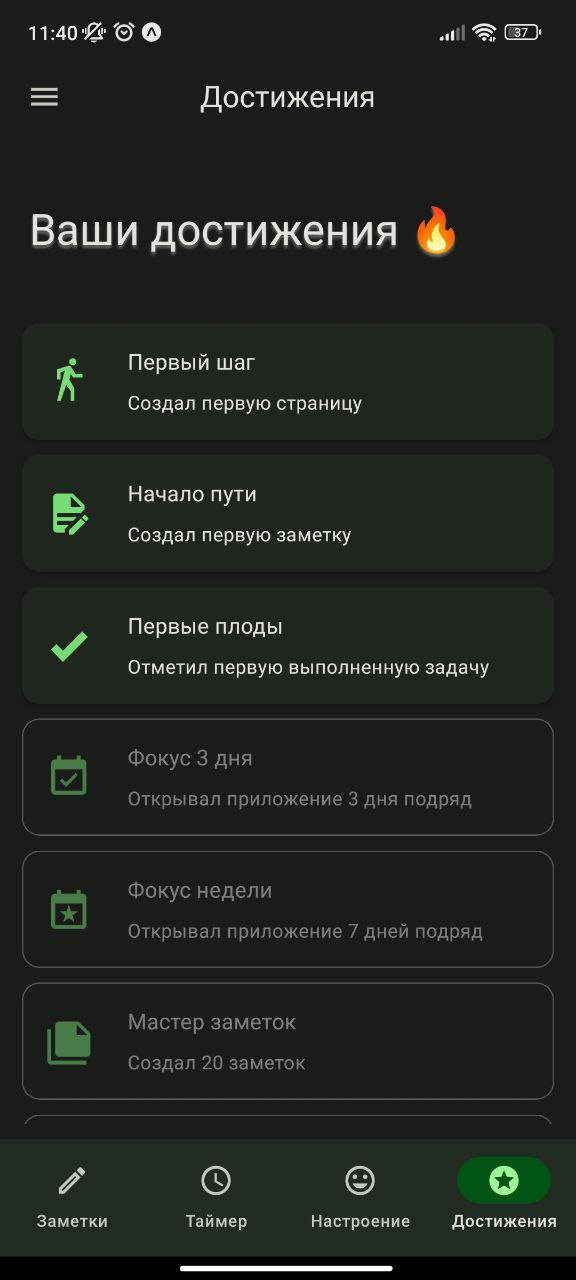


Рисунок 2.27 – Панель достижений

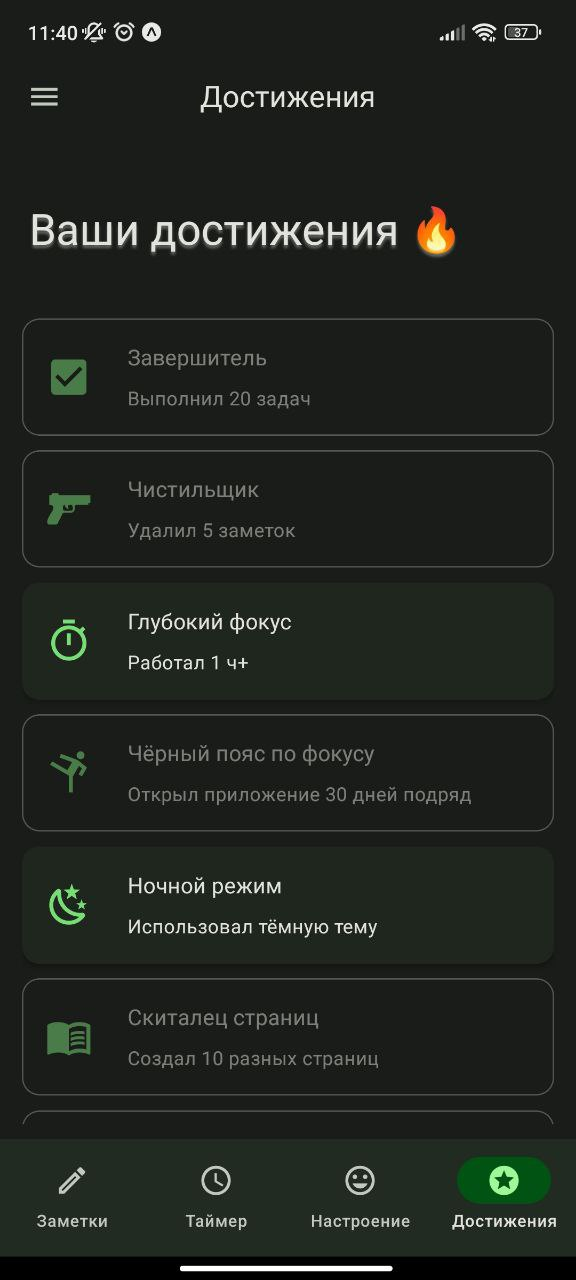


Рисунок 2.28 – Панель достижений

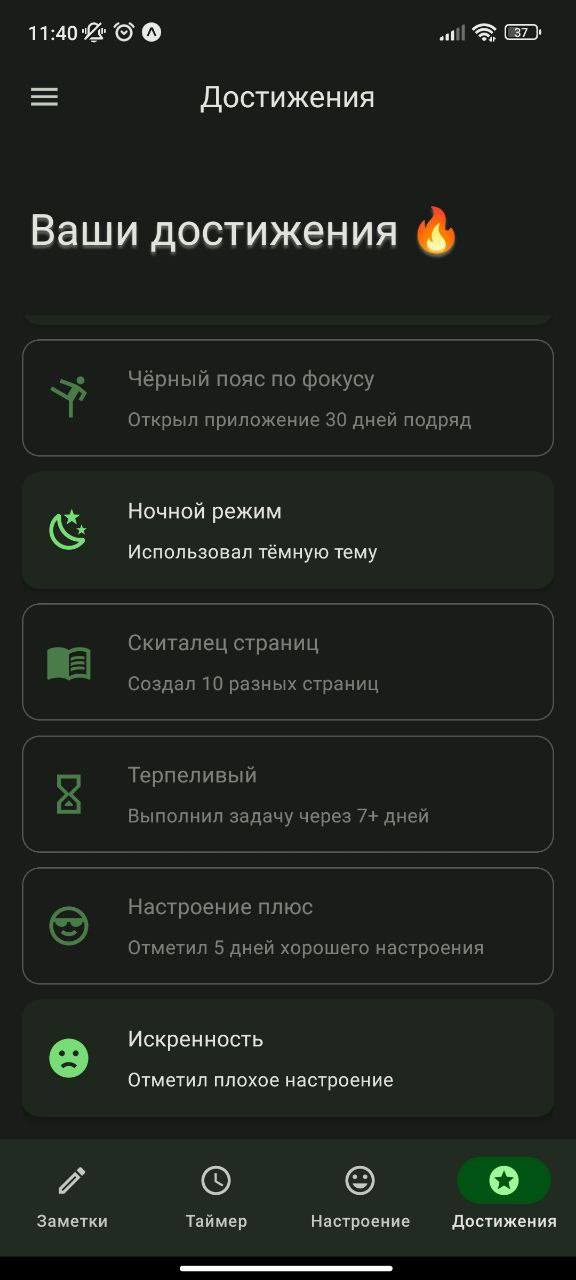


Рисунок 2.29 – Панель достижений

Из любой панели, пользователь может нажать на значок меню. После этого откроется боковая панель, из которой пользователь может сменить тему, либо перейти в настройки.

Вышеописанные действия представлены на рисунках 2.30 - 2.32

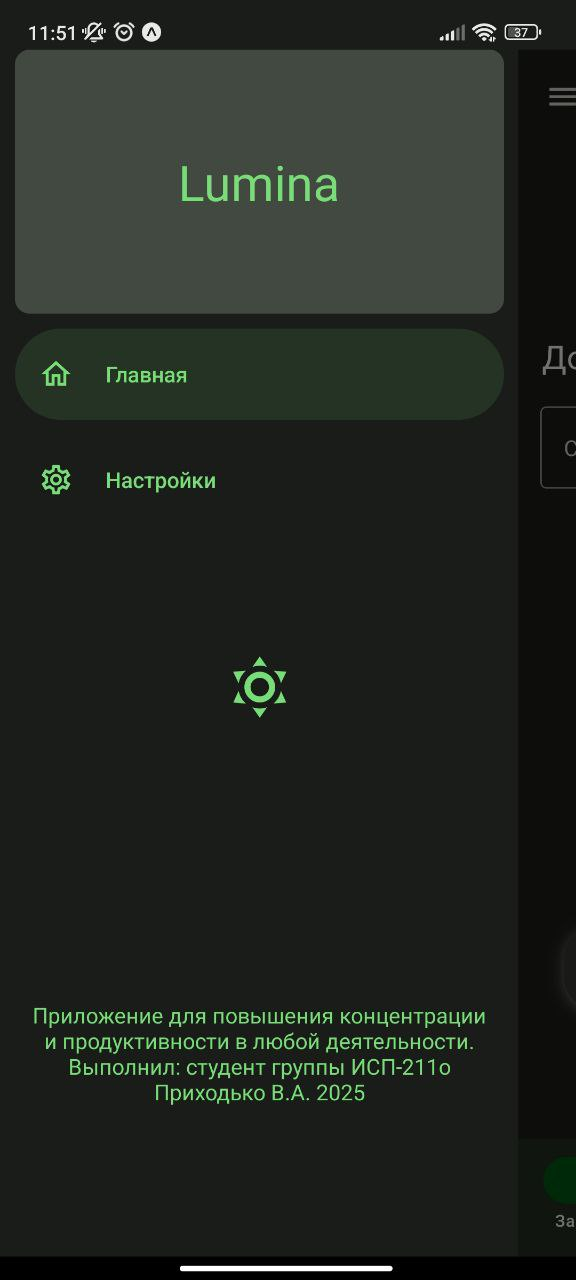


Рисунок 2.30 – Боковое меню

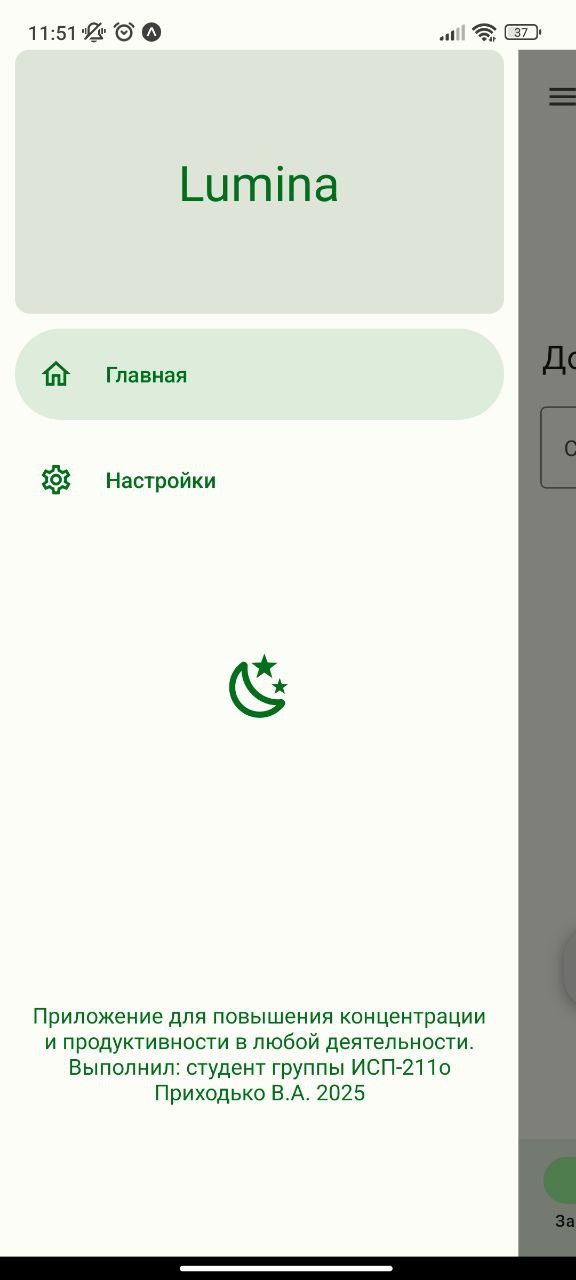


Рисунок 2.31 – Светлая тема

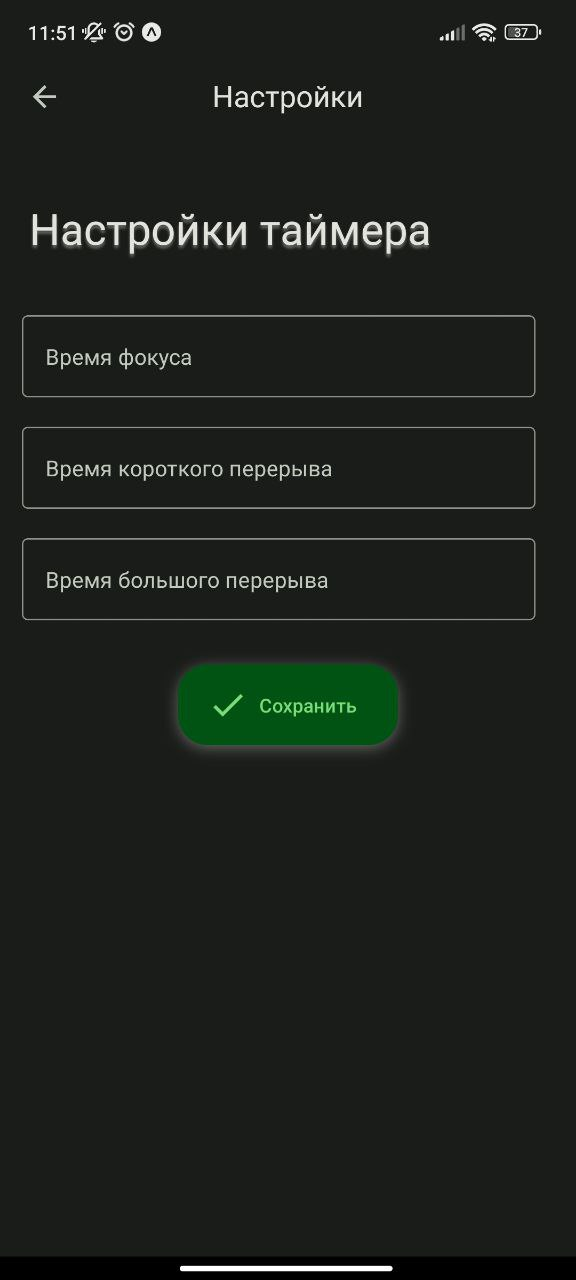


Рисунок 2.32 – Панель настроек

Перейдя в настройки, пользователь может изменить длительность периодов таймера и нажать на кнопку “Сохранить”. После этого изменения вступят в силу.

Вышеописанные действия представлены на рисунках 2.33 - 2.35

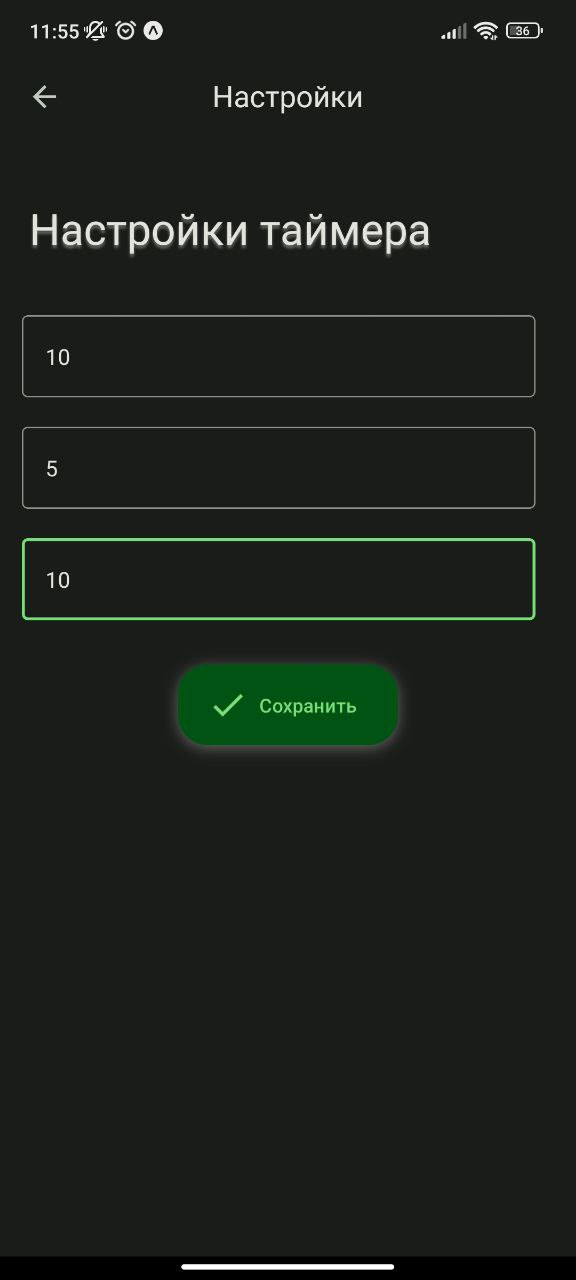


Рисунок 2.33 – Панель настроек

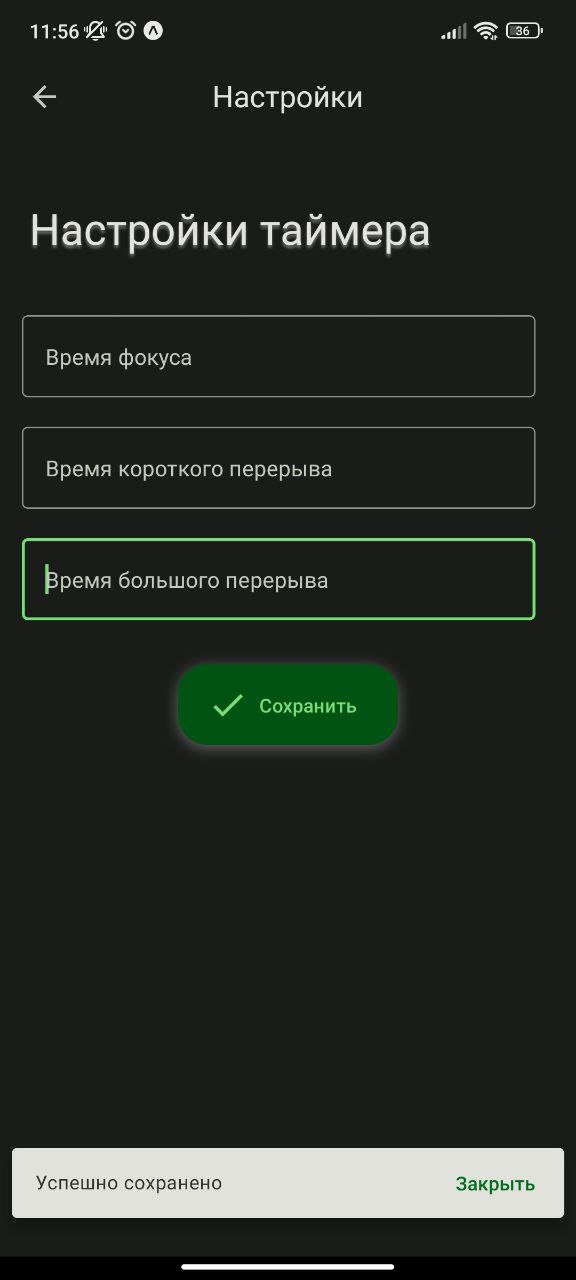


Рисунок 2.34 – Уведомление об успешном сохранении

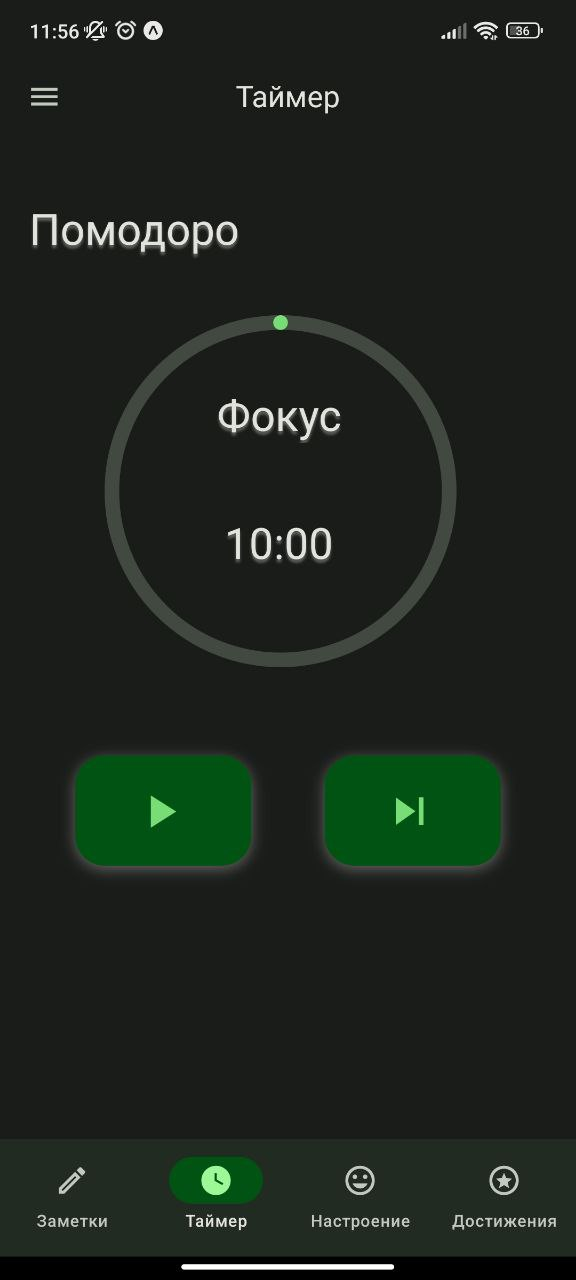


Рисунок 2.35 – Примененные изменения

# 3 Безопасность жизнедеятельности.

В процессе взаимодействия с мобильным приложением, установленным на смартфоне, крайне важно соблюдать ряд ключевых рекомендаций, направленных на защиту здоровья пользователя, сохранность личных данных и обеспечение стабильной работы устройства:

1. Использование антивирусного программного обеспечения и антишпионских инструментов помогает обезопасить конфиденциальные сведения от угроз со стороны вредоносных программ.
2. Применение сложных и уникальных паролей, а также отказ от их передачи третьим лицам, предотвращает несанкционированный доступ к системе.
3. Своевременное обновление как операционной системы устройства, так и самого приложения, позволяет устранить уязвимости и повысить устойчивость к потенциальным угрозам.
4. Загрузка программного обеспечения должна осуществляться исключительно из официальных источников (например, Google Play или App Store), чтобы избежать риска установки вредоносного контента.
5. Поддержание заряда аккумулятора на достаточном уровне помогает избежать сбоев в работе устройства и возможной потери данных в процессе тренировки.
6. Регулярное создание резервных копий пользовательской информации позволяет восстановить данные в случае технического сбоя.
7. Во избежание переутомления органов зрения при длительном использовании устройства рекомендуется применять фильтры синего света или специальные очки.
8. При работе с приложением в условиях повышенного шума следует использовать функции шумоподавления и ограничивать уровень звуковой нагрузки.
9. Приложение должно использоваться исключительно по назначению, в рамках цифровой этики.
10. Необходимо контролировать продолжительность непрерывного использования смартфона, чтобы снизить риск зрительного переутомления и напряжения в мышцах шеи и рук.

Потенциально вредные факторы, сопровождающие работу с мобильным устройством:

1. Электромагнитное излучение. Продолжительное воздействие высокочастотных волн может вызвать дискомфорт, нарушения сна и головные боли.
2. Усталость глаз. Долгое сосредоточение на небольшом экране может привести к снижению остроты зрения и головной боли.
3. Психоэмоциональные последствия. Чрезмерное использование гаджета способно провоцировать стресс и раздражительность.
4. Проблемы с опорно-двигательной системой. Неправильное или длительное удержание устройства вызывает нагрузку на суставы рук и пальцев.
5. Физические повреждения устройства. Падения и удары могут привести к поломке и утрате информации.

Рекомендации по снижению перечисленных рисков:

1. Использование чехлов и защитных экранов, способных снизить воздействие электромагнитного излучения.
2. Соблюдение режима труда и отдыха с обязательными перерывами и легкой разминкой.
3. Регулярное выполнение упражнений для пальцев и кистей рук.
4. Корректная настройка яркости экрана в зависимости от уровня освещённости в помещении.
5. Обеспечение качественной вентиляции и освещения на рабочем месте.
6. Удержание экрана смартфона на расстоянии не менее 30–40 см от глаз для снижения зрительной нагрузки.

Соблюдение вышеперечисленных рекомендаций способствует сохранению здоровья пользователя и повышает эффективность взаимодействия с мобильным приложением.

Особенности эксплуатации устройства для обеспечения технической надёжности:

1. Температурные условия. Работа устройства должна осуществляться в диапазоне от -10 до +40°C, что предотвращает его перегрев или переохлаждение.
2. Уровень влажности. Следует исключить попадание влаги, при необходимости — немедленно прекратить использование и обратиться в сервисный центр.
3. Физические воздействия. Рекомендуется избегать падений и ударов, которые могут привести к повреждению внутренних компонентов.
4. Электростатические разряды. Перед использованием устройства желательно снять статическое напряжение, особенно в условиях сухого воздуха.
5. Условия хранения. Хранить смартфон следует в сухом, затемнённом и прохладном месте вдали от прямых солнечных лучей.

Несоблюдение этих условий эксплуатации может привести к некорректной работе устройства и снижению его срока службы. В случае возникновения неисправностей необходимо обратиться за квалифицированной технической поддержкой.

# 4 Экономическая часть проекта

Экономическая часть дипломного проекта, выполняемая для системы программного продукта для повышения продуктивности должна содержать освещение следующих вопросов:

* Определение трудоемкости разработки программного продукта;
* Расчет расходов на оплату труда разработчика;
* Расчет затрат на оплату машинного времени;
* Определение накладных расходов, определение затрат на расходные материалы;
* Составление сметы затрат на разработку программного продукта и структуры затрат;
* Определение цены разработки ПО.

# 4.1 Определение трудоемкости разработки программного продукта

Трудоемкость разработки программного продукта была определена по методическим указаниям «Милованова Н.С., Демихова И.В. Экономика организации: методические указания к выполнению экономической части дипломного проекта для студентов *специальности 09.02.01 Компьютерные системы и комплексы, очной формы обучения*/ ФГБОУ ВО «Воронежский государственный технический университет» - Воронеж: Изд-во ВГТУ, 2023. - 20 с.»

Разработка программного продукта для повышения концентрации включает следующие основные этапы (таблица 4.1):

Таблица 4.1 – Этапы разработки

|  |  |
| --- | --- |
| Наименование этапа разработки | Трудоемкость, ч |
| Создание базы данных | 45 ч. |
| Разработка программного модуля | 180 ч. |
| Продолжение таблицы 4.1 | |
| Наименование этапа разработки | Трудоемкость, ч |
| Разработка тестовых заданий | 35 ч. |
| Создание документации | 50 ч. |
| Рефракторинг кода | 20 ч. |
| Руководство пользователя | 15 ч. |

Для расчета трудоемкости разработки (Тпо) используем следующую формулу:

|  |  |
| --- | --- |
| Тпо = tо + tал + tбл + tпр + tпод + tдок +tот, | (1) |

Где: tо– затраты труда на подготовку описания задачи,

tал – затраты труда на разработку алгоритма задачи,

tбл – затраты труда на разработку блоксхемы алгоритма задачи,

tпр – составление программы,

tпод – подготовка материалов в рукописи,

tдок – редактирование, печать и оформление документации,

tот – отладка программы.

Распределение трудоемкости по видам работ представлено в таблице 4.2.

Таблица 4.2 – Распределение трудоемкости по видам работ

|  |  |
| --- | --- |
| Вид работ | Трудоемкость, ч |
| Подготовка описания задачи (tо) | 25 |
| Разработка алгоритма (tал) | 100 |
| Разработка блок-схемы (tбл) | 75 |
| Программирование (tпр) | 180 |
| Подготовка материалов (tпод) | 40 |
| Оформление документации (tдок) | 40 |
| Отладка программы (tот) | 55 |

Трудоемкость разработки программного продукта (Тпо) составляет:

Тпо = 25 + 100 + 75 + 180 + 40 + 40 + 55 = 515 ч.

Структуру Тпо можно разделить на три этапа:

1. Подготовительный этап (Тпо1):

Тпо1 = tо + tал + tбл = 25 + 100 + 75 = 200 ч.

200 ч. = 38,83% · 515 ч.

2. Основной этап разработки (Тпо2):

Тпо2 = tпр + tпод +tот = 180 + 40 + 55 = 275 ч.

275 ч. = 53,39 % · 515 ч.

3. Завершающий этап (Тпо3):

Тпо3 = t док = 40 ч.

40 ч. = 7,76 % · 515 ч.

Общая трудоемкость разработки платформы составляет 515 часов, что соответствует нормативным значениям для проектов подобного вида. Наибольшие затраты приходятся на этап программирования (180 ч) и разработку блок-схем и алгоритма, что характерно для данного типа разработки.

# 4.2 Расчет расходов на оплату труда разработчика

Месячный оклад программиста (Зокл) = 62 000 руб./мес.

Месячный фонд рабочего времени (Fдс мес) = 176 ч. (при 40-часовой неделе).

Время разработки одного экземпляра программы (Тшт) = 515 ч.

Оплата труда, включаемая в затраты на данный продукт, определяется по следующей формуле:

(2)

где Зот – оплата труда за разработку программного продукта,

Fдс мес – месячный фонд рабочего времени инженера, ч.,

– время на разработку одного экземпляра программы, ч.

Исходя из формулы (2), получили следующее значение:

Зот = 62 000 \* = 181 420,45 руб.

Отчисления на социальные нужды составляют 30%: 181 420,45 \* 0,3 = 54 426,13 руб.

Итого общие расходы на оплату труда разработчика: Зот = 181 420,45 + 54 426,13 = 235 846,58 руб.

# 4.3 Расчет затрат на оплату машинного труда

Фактическое время использования компьютера при разработке программного продукта (tмаш) = 525 ч.

Балансовая стоимость ПК (Цбал) = 80 000 руб.

Норма амортизации компьютера (На) = 25%

Потребляемая мощность компьютера (Pw) = 280 Вт (системный блок + монитор)

Стоимость 1-го кВт-ч электроэнергии (Цэл) = 9 руб./кВт·ч.

Время использования компьютера за год (Fгод дс) = 2 100 ч.

Коэффициент интенсивного использования (A) = 0,98

Оклад техника согласно штатному расписанию (Зокл) = 35 000 руб.

Норма обслуживания для техников (Нобсл) = 20 ПК

Расчет затрат на оплату машинного труда определяется по следующей формуле:

Змаш = tмаш \* Sм-ч; (3)

где Змаш – затраты на оплату машинного времени, р.

tмаш – фактическое время использования компьютера при разработке программного продукта, ч.

Sм-ч – себестоимость часа работы компьютера, р.

Sм-ч = Згод/Fгод дс; (4)

где Згод - годовые затраты на эксплуатацию компьютера, р.

Fгод дс – время использования компьютера за год, ч.

Годовые затраты на эксплуатацию компьютера за год включают

следующие элементы затрат:

Згод = Ззп + Зам + Зэл + Звм +Зт.р.+ Зпр +Знакл; (5)

где Ззп – заработная плата техника по обслуживанию комп., р.,

Зам – годовые амортизационные отчисления, р.,

Зэл – стоимость электроэнергии, потребляемой компьютером за год, р,

Звм – стоимость вспомогательных материалов, необходимых для обеспечения нормальной работы комп., р.,

Зт.р.- затраты на ТО и ремонт компьютера, р.,

Зпр – прочие расходы, р.,

Знакл = накладные (общие) расходы.

Заработная плата техника определяется по нормам обслуживания:

Ззп = (Зокл\*12)/Нобсл, (6)

где Зокл –оклад техника согласно штатному расписанию,

Нобсл – норма обслуживания для техников ( 20 комп. в месяц).

Годовые амортизационные отчисления:

Зам = Цбал ∙ На, (7)

где Цбал – балансовая стоимость компьютера, р.,

На – норма амортизации компьютера (25 % в год).

Стоимость электроэнергии, потребляемой компьютером:

Зэл = (Рw\*Fгод дс∙Цэл∙А)/1000, (8)

где Рw – потребляемая мощность компьютера, Вт;

Цэл – стоимость 1-го кВт-ч электроэнергии, р./кВт-ч,

А – коэффициент интенсивного использования (0,98).

Потребляемая мощность компьютера определяется следующим образом:

Рw = Рw1 + Рw2, (9)

где Pw1 – потребляемая мощность системного блока, Вт,

Pw2 – мощность монитора, Вт.

По формуле (9) потребляемая мощность компьютера будет равна: Pw = 250+30 = 280 Вт.

По формуле (8) стоимость электроэнергии, потребляемой компьютером будет равна: Зэл = (280 \* 2100 \* 9 \* 0,98)/1000 = 5 186,16 руб./год.

По формуле (7) годовые амортизационные отчисления будут равны: Зам = 80 000 \* 0,25 = 20 000 руб./год.

По формуле (6) заработная плата техника по нормам обслуживания составила: Ззп = (35 000 \*12)/20 = 21 000 руб./год.

Затраты на вспомогательные материалы, необходимые для нормальной работы компьютера составляют: Звм = 800 руб./год (1% от стоимости ПК).

Затраты на ТО и ремонт компьютера (2% от стоимости ПК) составляют: Зт.р. = 80 000 \* 0,02 = 1 600 руб./год.

Прочие расходы (2 % от стоимости ПК) составляют: Зпр = 80 000 \* 0,02 = 1 600 руб./год.

Накладные (общие) расходы (30 % от зарплаты техника) составляют: Зпр = 21 000\* 0,3 = 6 300 руб./год.

По формуле (5) годовые затраты на эксплуатацию компьютера за год составили: Згод = 21 000 + 20 000 + 5 186,16 + 800 + 1 600 + 1 600 + 6 300 = 56 486,16 руб./год.

По формуле (4) себестоимость часа работы компьютера составит: Sм-ч = 56 486,16/2100 = 26,89 руб.

По формуле (3) затраты на оплату машинного времени составят: Змаш = 525 \* 26,89 = 14 117,25 руб.

# 4.4 Определение накладных расходов и затрат на расходные материалы

Накладные расходы (Знакл), включаемые в смету затрат на разработку программного продукта, принимаются равными 10 % от оплаты труда разработчика программы.

Знакл = Зот \* 0,1 = 181 420,45 \* 0,1 = 18 142,045 руб.

Расчет затрат на расходные материалы производится в таблице 4.3.

Таблица 4.3 – Затраты на расходные материалы

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование материалов | Цена, р. | Количество, шт. | Стоимость, р. |
| Флеш-карта (32 Гб) | 200 | 1 | 200 |
| Картридж для принтера | 2 000 | 3 | 6 000 |
| Бумага для принтера (500 листов) | 250 | 2 | 500 |
| Ручки | 30 | 10 | 300 |
| Итого затрат: | 7 000 | | |

# 4.5 Составление сметы затрат на разработку программного продукта и структуры затрат

Смета затрат приводится в таблице 4.

Таблица 4.4 – Смета затрат

|  |  |  |
| --- | --- | --- |
| Элементы затрат | Сумма, р. | Структура затрат, % |
| Оплата труда разработчика программы | 181 142,45 | 65,93 |
| Отчисления на социальные нужды | 54 426,13 | 19,8 |
| Оплата машинного времени | 14 117,25 | 5,13 |
| Накладные расходы | 18 142,045 | 6,6 |
| Затраты на расходные материалы | 7 000 | 2,54 |
| Итого (Зпо) | 274 827,875 | 100 |

Структура затрат представлена графически на рисунке 4.1.

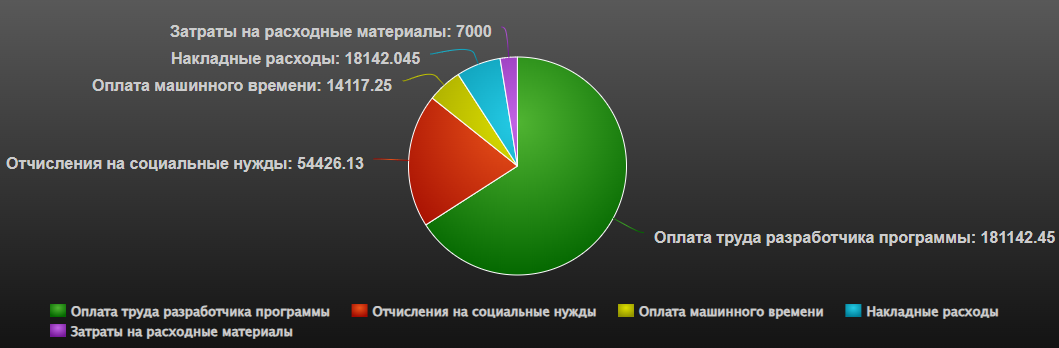


Рисунок 4.1 – Структура затрат

# 4.6 Определение цены разработки ПО

Часть стоимости разработки ПО, приходящаяся на одну копию:

Затраты на внедрение программы:

Затраты на внедрение ПО включают в себя затраты на оплату труда специалистов фирмы-разработчика, покупку необходимого оборудования для установки ПО, организацию рабочих мест и помещений, а также накладные расходы. Но, так как разработка происходит собственными специалистами на уже существующем оборудовании и рабочих местах заказчика, то затраты на покупку и обустройство отсутствуют (Звн = 0).

Общие затраты на разработку и внедрение ПО:

З = Δ КЗ + Звн = + 0 = руб.

Договорная цена с прибылью (процент прибыли, заложенный в цену (Пр) = 20%):

Цпо = (ΔКЗ + Звн) × (1 + Пр) = ( + 0) \* (1+ 0,2) 329 793,45 руб.

Учет НДС:

НДС = 0,2 \* Цпо = 0,2 \* 329 793,45 65 958,69 руб.

Окончательная цена ПО составляет:

Ц=Цпо + НДС = 329 793,45 + 65 958,69 = 395 752,14 руб.

Таким образом, разработка программного продукта для повышения концентрации экономически целесообразна. Основные затраты приходятся на оплату труда (65,93 %). Рыночная цена проекта составит 400 000 руб. с учетом НДС.

# ЗАКЛЮЧЕНИЕ

В рамках выполнения выпускной квалификационной работы был разработан и реализован программный продукт для повышения продуктивности и концентрации в любой деятельности.

В ходе выполнения работы были последовательно рассмотрены все ключевые этапы создания программного продукта: от анализа предметной области и формирования функциональных требований до проектирования архитектуры, оценки безопасности и проведения экономической оценки эффективности внедрения.

Созданный программный продукт обеспечивает пользователю простой и удобный интерфейс, систему организации задач по страницам, гибкий таймер по технике помодоро, визуальный трекер настроения с анализом эмоционального состояния и систему мотивации через достижения. Приложение ориентировано на индивидуальное использование, не требует регистрации, а данные сохраняются локально и в облачном хранилище Firebase Firestore.

Для визуализации архитектуры и логики работы приложения были построены диаграммы классов, диаграмма вариантов использования и ER-модель базы данных, что позволило обеспечить структурированность, масштабируемость и удобство сопровождения проекта. Приложение было протестировано на работоспособность и соответствие функциональным требованиям.

В работе также были учтены аспекты безопасности жизнедеятельности при работе с мобильными устройствами, включая эргономику, нагрузку на зрение и защиту пользовательских данных. Проведён расчёт экономической эффективности проекта, подтверждающий целесообразность его разработки и возможность дальнейшего применения.

Реализация программного продукта для повышения продуктивности и концентрации в любой деятельности продемонстрировала, что современные мобильные технологии могут служить эффективным инструментом не только для управления задачами, но и для поддержки ментального здоровья, формирования полезных привычек и повышения личной эффективности. Полученные в ходе работы знания и практический опыт могут быть использованы для дальнейшего развития проекта, а также при создании аналогичных решений в области цифровой продуктивности и саморазвития.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. React – официальная документация [Электронный ресурс] // reactjs.org: Документация по библиотеке React. — URL: https://reactjs.org/docs/getting-started.html (дата обращения: 28.04.2025).
2. React Native – официальная документация [Электронный ресурс] // reactnative.dev: Руководство по созданию мобильных приложений. — URL: https://reactnative.dev/docs/getting-started (дата обращения: 28.04.2025).
3. Expo – официальная документация [Электронный ресурс] // expo.dev: Среда разработки React Native-приложений. — URL: https://docs.expo.dev (дата обращения: 29.04.2025).
4. Firebase – документация по Firestore [Электронный ресурс] // firebase.google.com: Официальная документация Google Firebase. — URL: https://firebase.google.com/docs/firestore (дата обращения: 30.04.2025).
5. Firebase Authentication – руководство пользователя [Электронный ресурс] // firebase.google.com: Безопасность и авторизация. — URL: https://firebase.google.com/docs/auth (дата обращения: 30.04.2025).
6. JavaScript – справочник по языку [Электронный ресурс] // developer.mozilla.org: MDN Web Docs. — URL:https://developer.mozilla.org/ru/docs/Web/JavaScript (дата обращения: 01.05.2025).
7. TypeScript – официальная документация [Электронный ресурс] // typescriptlang.org: Документация по языку TypeScript. — URL: https://www.typescriptlang.org/docs (дата обращения: 01.05.2025).
8. OpenRouter – API-документация [Электронный ресурс] // openrouter.ai: Платформа доступа к языковым моделям. — URL: https://openrouter.ai/docs (дата обращения: 02.05.2025).
9. Material Design в React Native [Электронный ресурс] // callstack.github.io/react-native-paper: Компоненты и принципы дизайна. — URL: https://callstack.github.io/react-native-paper (дата обращения: 03.05.2025).
10. AsyncStorage в React Native [Электронный ресурс] // react-native-async-storage.github.io: Хранение данных локально. — URL: https://react-native-async-storage.github.io/async-storage/docs/usage (дата обращения: 04.05.2025).
11. PlantUML – создание UML-диаграмм [Электронный ресурс] // plantuml.com: Инструмент моделирования. — URL: https://plantuml.com/ru/class-diagram (дата обращения: 04.05.2025).
12. React Navigation – навигация в React Native [Электронный ресурс] // reactnavigation.org: Официальная документация. — URL: https://reactnavigation.org/docs/getting-started (дата обращения: 05.05.2025).

# Приложение A

Package.json

{

"name": "lumina",

"main": "expo-router/entry",

"version": "1.0.0",

"scripts": {

"start": "expo start",

"reset-project": "node ./scripts/reset-project.js",

"android": "expo start --android",

"ios": "expo start --ios",

"web": "expo start --web",

"test": "jest --watchAll",

"lint": "expo lint"

},

"jest": {

"preset": "jest-expo"

},

"dependencies": {

"@expo/vector-icons": "^14.0.2",

"@react-native-async-storage/async-storage": "1.23.1",

"@react-navigation/bottom-tabs": "^7.2.0",

"@react-navigation/drawer": "^7.3.9",

"@react-navigation/native": "^7.0.14",

"@shopify/react-native-skia": "1.5.0",

"expo": "~52.0.46",

"expo-blur": "~14.0.3",

"expo-constants": "~17.0.8",

"expo-font": "~13.0.4",

"expo-haptics": "~14.0.1",

"expo-linking": "~7.0.5",

"expo-notifications": "^0.29.14",

"expo-router": "~4.0.21",

"expo-splash-screen": "~0.29.24",

"expo-status-bar": "~2.0.1",

"expo-symbols": "~0.2.2",

"expo-system-ui": "~4.0.9",

"expo-web-browser": "~14.0.2",

"firebase": "^11.6.1",

"react": "18.3.1",

"react-dom": "18.3.1",

"react-native": "0.76.9",

"react-native-gesture-handler": "~2.20.2",

"react-native-get-random-values": "^1.11.0",

"react-native-paper": "^5.13.5",

"react-native-reanimated": "~3.16.1",

"react-native-safe-area-context": "^4.12.0",

"react-native-screens": "~4.4.0",

"react-native-skia": "^0.0.1",

"react-native-svg": "^15.8.0",

"react-native-web": "~0.19.13",

"react-native-webview": "13.12.5",

"uuid": "^11.1.0",

"victory-native": "^41.17.1"

},

"devDependencies": {

"@babel/core": "^7.25.2",

"@types/jest": "^29.5.12",

"@types/react": "~18.3.12",

"@types/react-native": "^0.72.8",

"@types/react-test-renderer": "^18.3.0",

"jest": "^29.2.1",

"jest-expo": "~52.0.6",

"react-test-renderer": "18.3.1",

"typescript": "^5.3.3"

},

"private": true

}

app.json

{

"expo": {

"name": "lumina",

"slug": "lumina",

"version": "1.0.0",

"orientation": "portrait",

"icon": "./assets/images/lumina.png",

"scheme": "lumina",

"userInterfaceStyle": "automatic",

"newArchEnabled": true,

"android": {

"package": "com.eyd8s.lumina",

"adaptiveIcon": {

"foregroundImage": "./assets/images/lumina.png",

"backgroundColor": "#ffffff"

}

},

"ios": {

"supportsTablet": true

},

"web": {

"bundler": "metro",

"output": "static",

"favicon": "./assets/images/lumina.png"

},

"assetBundlePatterns": ["\*\*/\*"],

"plugins": [

"expo-router",

[

"expo-splash-screen",

{

"image": "./assets/images/lumina.png",

"imageWidth": 200,

"resizeMode": "contain",

"backgroundColor": "#ffffff"

}

]

],

"experiments": {

"typedRoutes": true

},

"extra": {

"router": {

"origin": false

},

"eas": {

"projectId": "fdf1c5c7-2116-46cf-973d-c7f56ad93fa8"

}

}

}

}

\_layout.tsx

import { NavigationIndependentTree } from "@react-navigation/native";

import App from './index';

export default function Main() {

return (

<NavigationIndependentTree>

<App/>

</NavigationIndependentTree>

);

}

DrawerNavigator.tsx

import { createDrawerNavigator, DrawerItemList } from "@react-navigation/drawer";

import { View, StyleSheet, Pressable } from "react-native";

import { Text, Icon } from "react-native-paper";

import BottomTabs from "./BottomTabs";

import Settings from './tabs/Settings'

import { useThemeContext } from '../components/context/ThemeContext';

//создание навигатора

const Drawer = createDrawerNavigator();

export default function DrawerNavigator(){

//переменные для переключения темы и цветов

const { theme, isDarkTheme, toggleTheme } = useThemeContext();

//стили для компонентов меню

const styles = StyleSheet.create({

drawContainer:{

flex: 1,

marginHorizontal: 10,

gap: 10

},

textContainer:{

width: '100%',

height: 180,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: theme.colors.surfaceVariant ,

borderRadius: 10

},

title:{

color: theme.colors.primary,

fontSize: 36

},

textContainerBottom:{

width: '100%',

height: 180,

justifyContent: 'center',

alignItems: 'center',

marginTop: 50

},

description:{

color: theme.colors.primary,

fontSize: 16,

textAlign: 'center'

},

iconContainer:{

width: '100%',

height: 200,

justifyContent: 'center',

alignItems: 'center'

}

})

//сами коспоненты, навигатор, его контент и страницы

return(

<Drawer.Navigator screenOptions={{

//свойства навигатора, фон, как ведут себя элементы

headerShown: false,

drawerStyle:{

backgroundColor: theme.colors.background,

width: '90%',

},

drawerActiveTintColor: theme.colors.primary,

drawerLabelStyle:{

color: theme.colors.primary,

fontSize: 16,

marginHorizontal: 10,

marginVertical: 5

},

drawerType: 'slide'

}}

//контент, который также входит в меню (текст, иконки)

drawerContent={

(props) => {

return (

<View style={styles.drawContainer}>

<View style={styles.textContainer}>

<Text style={styles.title}

>Lumina</Text>

</View>

<DrawerItemList {...props} />

<View style={styles.iconContainer}>

<Pressable onPress={toggleTheme}>

<Icon source={isDarkTheme ? 'weather-sunny' : 'weather-night'} color={theme.colors.primary} size={50}/>

</Pressable>

</View>

<View style={styles.textContainerBottom}>

<Text style={styles.description}>

Приложение для повышения концентрации и продуктивности в любой деятельности. Выполнил: студент группы ИСП-211о Приходько В.А. 2025

</Text>

</View>

</View>

);

}

}>

{/\* экраны, т.е. кнопки по нажатию на которые можно перейти на страницу \*/}

<Drawer.Screen

name="Home"

component={BottomTabs}

options={{

drawerLabel: 'Главная',

title: 'Home',

drawerIcon: () => <Icon source='home-outline' size={24} color={theme.colors.primary}/>

}}

/>

<Drawer.Screen

name="Settings"

component={Settings}

options={{

drawerLabel: 'Настройки',

title: 'Settings',

drawerIcon: () => <Icon source='cog-outline' size={24} color={theme.colors.primary}/>

}}

/>

</Drawer.Navigator>

)

}

BottomTabs.tsx

import \* as React from 'react';

import { useState } from 'react';

import { BottomNavigation } from 'react-native-paper';

import Tasks from './tabs/Tasks';

import Focus from './tabs/Focus';

import Mood from './tabs/Mood';

import Awards from './tabs/Awards';

const BottomTabs = () => {

const [index, setIndex] = useState(0);

const [routes] = useState([

{ key: 'tasks', title: 'Заметки', focusedIcon: 'pencil', unfocusedIcon: 'pencil-outline'},

{ key: 'focus', title: 'Таймер', focusedIcon: 'clock-time-four', unfocusedIcon: 'clock-outline' },

{ key: 'mood', title: 'Настроение', focusedIcon: 'emoticon', unfocusedIcon: 'emoticon-outline' },

{ key: 'awards', title: 'Достижения', focusedIcon: 'star-circle', unfocusedIcon: 'star-circle-outline' },

]);

const renderScene = BottomNavigation.SceneMap({

tasks: Tasks,

focus: Focus,

mood: Mood,

awards: Awards,

});

return (

<BottomNavigation

navigationState={{ index, routes }}

onIndexChange={setIndex}

renderScene={renderScene}

/>

);

};

export default BottomTabs;

declaration.ts

declare module "\*.ttf" {

const content: number;

export default content;

}

Awards.tsx

import \* as React from 'react';

import { View} from 'react-native';

import { Appbar } from 'react-native-paper';

import { useNavigation } from "@react-navigation/native";

import { DrawerActions } from "@react-navigation/native";

import MainText from '../../components/elements/Typography/MainText';

import AwardsBox from '../../components/elements/Awards/AwardsBox';

export default function Awards() {

const navigation = useNavigation();

return (

<>

<Appbar.Header mode='center-aligned'>

<Appbar.Action icon='menu' onPress={() => navigation.dispatch(DrawerActions.openDrawer())} />

<Appbar.Content title="Достижения" />

</Appbar.Header>

<View style={{

width: '100%',

height: '90%'

}}>

<MainText>Ваши достижения 🔥</MainText>

<AwardsBox />

</View>

</>

);

}

Focus.tsx

import \* as React from 'react';

import { View} from 'react-native';

import { Appbar } from 'react-native-paper';

import { useNavigation } from "@react-navigation/native";

import { DrawerActions } from "@react-navigation/native";

import MainText from '../../components/elements/Typography/MainText';

import Timer from '../../components/elements/PomodoroTimer/Timer';

export default function Focus() {

const navigation = useNavigation();

return (

<>

<Appbar.Header mode='center-aligned'>

<Appbar.Action icon='menu' onPress={() => navigation.dispatch(DrawerActions.openDrawer())} />

<Appbar.Content title="Таймер" />

</Appbar.Header>

<View>

<MainText>Помодоро</MainText>

<Timer />

</View>

</>

);

}

Mood.tsx

import React, { useState, useEffect } from 'react';

import { View, ScrollView, Keyboard} from 'react-native';

import { Appbar, ActivityIndicator, Portal, Dialog } from 'react-native-paper';

import { useNavigation } from "@react-navigation/native";

import { DrawerActions } from "@react-navigation/native";

import { saveMood, getMoods, getMoodAdvice } from '@/components/data/moodService'

import MainText from '../../components/elements/Typography/MainText';

import ButtonEl from '../../components/elements/Buttons/ButtonEl';

import InputEl from '../../components/elements/Inputs/InputEl';

import MoodButton from '../../components/elements/Buttons/MoodButton';

import MoodList from '../../components/elements/Lists/MoodList';

import LabelText from "../../components/elements/Typography/LabelText";

import MoodChart from "../../components/elements/Charts/MoodChart";

//тип для создания заметки настроения

type MoodType = {

title: string,

moodValue: number,

description: string;

}

export default function Mood() {

//навигация для меню

const navigation = useNavigation();

//состояния выбранного настроения

const [selectedMood, setSelectedMood] = useState(0);

//состояния для поля ввода

const [inputValue, setInputValue] = useState('');

//состояния для видимости всплывающего окна

const [visibleDialog, setVisibleDialog] = useState(false);

//совет от нейросети

const [advise, setAdvise] = useState('');

//состояние загрузки

const [loading, setLoading] = useState(true)

//массив настроений

const [moods, setMoods] = useState<MoodType[]>([]);

//загрузка массива записей настроений из базы данных

useEffect(() => {

const loadMoods = async () => {

const storedMoods = await getMoods();

setMoods(storedMoods);

setLoading(false);

}

loadMoods();

}, []);

//переменная проверяющая возможность добавления записи

const canAddMood = selectedMood != 0;

//функция для форматирования даты и времени

const formatDate = (date: Date): string => {

const formatNumber = (num: number) => num.toString().padStart(2, '0');

const day = formatNumber(date.getDate());

const month = formatNumber(date.getMonth() + 1);

const hours = formatNumber(date.getHours());

const minutes = formatNumber(date.getMinutes());

return `${day}.${month} ${hours}:${minutes}`

}

//асинхронная функция создания записи настроения и добавления в базу

const handleAddMood = async () => {

if(selectedMood == 0) return

const newMood: MoodType = {

title: new Date().getTime().toString(),

moodValue: selectedMood,

description: inputValue

}

//сначала локально

setMoods([...moods, newMood]);

setInputValue('');

setSelectedMood(0)

Keyboard.dismiss();

//потом в бд

await saveMood(newMood)

//и совет от нейросети

const advise = await getMoodAdvice(inputValue);

setAdvise(advise);

}

return (

<>

{/\* Шапка и название страницы \*/}

<Appbar.Header mode='center-aligned'>

<Appbar.Action icon='menu' onPress={() => navigation.dispatch(DrawerActions.openDrawer())} />

<Appbar.Content title="Настроение" />

</Appbar.Header>

<View style={{width: '100%', height:'80%'}}>

{/\* Всплывающее окно с советом \*/}

<View>

<Portal>

<Dialog visible={visibleDialog} onDismiss={() => setVisibleDialog((prev) => !prev)} style={{maxHeight: '90%'}}>

<Dialog.Title>Совет от нейросети</Dialog.Title>

<Dialog.ScrollArea>

<ScrollView >

<Dialog.Content style={{paddingHorizontal: 0, marginVertical: 20}}>

<LabelText>{advise}</LabelText>

</Dialog.Content>

</ScrollView>

</Dialog.ScrollArea>

<Dialog.Actions>

<ButtonEl

typeButton='buttonLabel'

icon='check'

label='Хорошо'

onPress={() => setVisibleDialog((prev) => !prev)}

/>

</Dialog.Actions>

</Dialog>

</Portal>

</View>

{/\* Главный текст \*/}

<MainText>Как настроение?</MainText>

{/\* Панель выбора настроения \*/}

<View>

<MoodButton moodValue={selectedMood} setMoodValue={setSelectedMood} />

</View>

{/\* Поле ввода и кнопка \*/}

<View style={{

width: '100%',

flexDirection: 'row',

justifyContent: 'center',

marginVertical: 20

}}>

<InputEl

inputType='mood'

placeholder='Опишите своё настроение'

value={inputValue}

onChangeText={setInputValue}

/>

<ButtonEl

typeButton='buttonMoodEdit'

icon='pencil-outline'

label=''

onPress={handleAddMood}

disabled={!canAddMood}

/>

</View>

<View style={{flex: 1}}>

{/\* Индикатор загрузки \*/}

<ActivityIndicator animating={loading} style={{position: 'absolute', width: '100%'}}/>

<ScrollView>

{/\* Статистика \*/}

{moods.length > 0 && (

<MoodChart moods={moods} />

)}

{/\* Список заметок с настроением \*/}

{[...moods].reverse().map((mood) => (

<MoodList

key={mood.title}

title={

`${formatDate(new Date(+mood.title))} - ${

mood.moodValue == 1 ? 'Ужасно' :

mood.moodValue == 2 ? 'Грустно' :

mood.moodValue == 3 ? 'Нормально' :

mood.moodValue == 4 ? 'Хорошо' :

'Отлично'

}`

}

moodValue={mood.moodValue}

description={mood.description}

/>

))}

</ScrollView>

{/\* Кнопка спросить совет \*/}

<View style={{

width: '100%',

alignItems: 'center',

marginBottom: -60,

marginTop: 20

}}>

<ButtonEl

typeButton='buttonLabel'

icon='robot-confused-outline'

label='Совет'

onPress={() => setVisibleDialog((prev) => !prev)}

disabled={advise == ''}

/>

</View>

</View>

</View>

</>

);

}

Settings.tsx

import React, {useState} from 'react';

import { View} from 'react-native';

import { Appbar, Snackbar } from 'react-native-paper';

import { useNavigation } from "@react-navigation/native";

import { useThemeContext } from '../../components/context/ThemeContext';

import { useTimerSettings } from '../../components/context/TimerSettingsContext';

import MainText from '../../components/elements/Typography/MainText';

import ButtonEl from '../../components/elements/Buttons/ButtonEl';

import InputEl from '../../components/elements/Inputs/InputEl';

export default function Settings() {

//навигация для перехода назад

const navigation = useNavigation();

//цветовая тема

const { theme } = useThemeContext();

//состояния для отслеживания полей ввода

const [focusTimeValue, setFocusTimeValue] = useState('');

const [shortBreakTimeValue, setShortBreakTimeValue] = useState('');

const [longBreakTimeValue, setLongBreakTimeValue] = useState('');

//состояния снэкбара

const [visibleSnackbar, setVisibleSnackbar] = useState(false)

//функции из провайдера для изменения значений времени

const {

setFocusDuration,

setShortBreakDuration,

setLongBreakDuration

} = useTimerSettings()

//функция для сохранения изменений

const saveChanges = () => {

if(focusTimeValue != '') {

setFocusDuration(+focusTimeValue);

setVisibleSnackbar(true);

}

if(shortBreakTimeValue != '') {

setShortBreakDuration(+shortBreakTimeValue);

setVisibleSnackbar(true);

}

if(longBreakTimeValue != '') {

setLongBreakDuration(+longBreakTimeValue);

setVisibleSnackbar(true);

}

setFocusTimeValue('');

setShortBreakTimeValue('');

setLongBreakTimeValue('');

}

return (

<>

<Appbar.Header mode='center-aligned'>

<Appbar.Action icon='arrow-left' onPress={() => navigation.goBack()} />

<Appbar.Content title="Настройки" />

</Appbar.Header>

<View style={{

backgroundColor: theme.colors.background,

width: '100%',

height: '100%'

}}>

<View>

<MainText>Настройки таймера</MainText>

<View style={{

rowGap: 20

}}>

<InputEl

placeholder='Время фокуса'

inputType='settings'

value={focusTimeValue}

onChangeText={setFocusTimeValue}

/>

<InputEl

placeholder='Время короткого перерыва'

inputType='settings'

value={shortBreakTimeValue}

onChangeText={setShortBreakTimeValue}

/>

<InputEl

placeholder='Время большого перерыва'

inputType='settings'

value={longBreakTimeValue}

onChangeText={setLongBreakTimeValue}

/>

</View>

</View>

<View style={{

width: '100%',

alignItems: 'center',

marginTop: 30

}}>

<ButtonEl

typeButton='buttonLabel'

icon='check'

label='Сохранить'

onPress={saveChanges}

/>

</View>

<View style={{

position: 'absolute',

bottom: '10%',

width: '100%'

}}>

<Snackbar

visible={visibleSnackbar}

onDismiss={() => setVisibleSnackbar(false)}

action={{

label: 'Закрыть'

}}>

Успешно сохранено

</Snackbar>

</View>

</View>

</>

);

}

Tasks.tsx

import React, {useEffect} from 'react';

import { useNavigation } from "@react-navigation/native";

import { DrawerActions } from "@react-navigation/native";

import { View } from 'react-native';

import { Appbar } from 'react-native-paper';

import \* as Notifications from 'expo-notifications';

import TaskList from '../../components/elements/Lists/TaskList'

export default function Tasks() {

const navigation = useNavigation();

// Запрос разрешения на уведомления при старте

useEffect(() => {

const requestPermissions = async () => {

const { status } = await Notifications.requestPermissionsAsync();

if (status !== 'granted') {

console.warn('Уведомления не разрешены');

}

};

requestPermissions();

}, []);

return (

<>

<Appbar.Header mode='center-aligned'>

<Appbar.Action icon='menu' onPress={() => navigation.dispatch(DrawerActions.openDrawer())} />

<Appbar.Content title="Заметки" />

</Appbar.Header>

<View style={{width: '100%', height: '80%'}}>

<TaskList />

</View>

</>

);

}

Colors.ts

/\*\*

\* Below are the colors that are used in the app. The colors are defined in the light and dark mode.

\* There are many other ways to style your app. For example, [Nativewind](https://www.nativewind.dev/), [Tamagui](https://tamagui.dev/), [unistyles](https://reactnativeunistyles.vercel.app), etc.

\*/

export const Colors = {

light: {

"primary": "rgb(0, 110, 28)",

"onPrimary": "rgb(255, 255, 255)",

"primaryContainer": "rgb(148, 249, 144)",

"onPrimaryContainer": "rgb(0, 34, 4)",

"secondary": "rgb(16, 109, 32)",

"onSecondary": "rgb(255, 255, 255)",

"secondaryContainer": "rgb(157, 248, 152)",

"onSecondaryContainer": "rgb(0, 34, 4)",

"tertiary": "rgb(62, 106, 0)",

"onTertiary": "rgb(255, 255, 255)",

"tertiaryContainer": "rgb(185, 244, 116)",

"onTertiaryContainer": "rgb(15, 32, 0)",

"error": "rgb(186, 26, 26)",

"onError": "rgb(255, 255, 255)",

"errorContainer": "rgb(255, 218, 214)",

"onErrorContainer": "rgb(65, 0, 2)",

"background": "rgb(252, 253, 246)",

"onBackground": "rgb(26, 28, 25)",

"surface": "rgb(252, 253, 246)",

"onSurface": "rgb(26, 28, 25)",

"surfaceVariant": "rgb(222, 229, 216)",

"onSurfaceVariant": "rgb(66, 73, 64)",

"outline": "rgb(114, 121, 111)",

"outlineVariant": "rgb(194, 201, 189)",

"shadow": "rgb(0, 0, 0)",

"scrim": "rgb(0, 0, 0)",

"inverseSurface": "rgb(47, 49, 45)",

"inverseOnSurface": "rgb(240, 241, 235)",

"inversePrimary": "rgb(120, 220, 119)",

"elevation": {

"level0": "transparent",

"level1": "rgb(239, 246, 235)",

"level2": "rgb(232, 242, 229)",

"level3": "rgb(224, 237, 222)",

"level4": "rgb(222, 236, 220)",

"level5": "rgb(217, 233, 216)"

},

"surfaceDisabled": "rgba(26, 28, 25, 0.12)",

"onSurfaceDisabled": "rgba(26, 28, 25, 0.38)",

"backdrop": "rgba(44, 50, 42, 0.4)"

},

dark: {

"primary": "rgb(120, 220, 119)",

"onPrimary": "rgb(0, 57, 10)",

"primaryContainer": "rgb(0, 83, 19)",

"onPrimaryContainer": "rgb(148, 249, 144)",

"secondary": "rgb(130, 219, 126)",

"onSecondary": "rgb(0, 57, 10)",

"secondaryContainer": "rgb(0, 83, 18)",

"onSecondaryContainer": "rgb(157, 248, 152)",

"tertiary": "rgb(158, 215, 91)",

"onTertiary": "rgb(30, 55, 0)",

"tertiaryContainer": "rgb(46, 79, 0)",

"onTertiaryContainer": "rgb(185, 244, 116)",

"error": "rgb(255, 180, 171)",

"onError": "rgb(105, 0, 5)",

"errorContainer": "rgb(147, 0, 10)",

"onErrorContainer": "rgb(255, 180, 171)",

"background": "rgb(26, 28, 25)",

"onBackground": "rgb(226, 227, 221)",

"surface": "rgb(26, 28, 25)",

"onSurface": "rgb(226, 227, 221)",

"surfaceVariant": "rgb(66, 73, 64)",

"onSurfaceVariant": "rgb(194, 201, 189)",

"outline": "rgb(140, 147, 136)",

"outlineVariant": "rgb(66, 73, 64)",

"shadow": "rgb(0, 0, 0)",

"scrim": "rgb(0, 0, 0)",

"inverseSurface": "rgb(226, 227, 221)",

"inverseOnSurface": "rgb(47, 49, 45)",

"inversePrimary": "rgb(0, 110, 28)",

"elevation": {

"level0": "transparent",

"level1": "rgb(31, 38, 30)",

"level2": "rgb(34, 43, 33)",

"level3": "rgb(36, 49, 35)",

"level4": "rgb(37, 51, 36)",

"level5": "rgb(39, 55, 38)"

},

"surfaceDisabled": "rgba(226, 227, 221, 0.12)",

"onSurfaceDisabled": "rgba(226, 227, 221, 0.38)",

"backdrop": "rgba(44, 50, 42, 0.4)"

}

};

ThemeContext.tsx

import React, { createContext, useState, useContext, useEffect } from 'react';

import { MD3LightTheme, MD3DarkTheme } from 'react-native-paper';

import { useColorScheme } from 'react-native';

import AsyncStorage from "@react-native-async-storage/async-storage";

import { Colors } from '../constants/Colors';

import { unlockAward } from '../data/awardService';

const lightTheme = { ...MD3LightTheme, colors: Colors.light };

const darkTheme = { ...MD3DarkTheme, colors: Colors.dark }

type ThemeType = typeof lightTheme;

interface ThemeContextType {

theme: ThemeType;

toggleTheme: () => void;

isDarkTheme: boolean;

}

const ThemeContext = createContext<ThemeContextType>({

theme: lightTheme,

toggleTheme: () => {},

isDarkTheme: false,

});

const THEME\_KEY = "APP\_THEME";

export const ThemeProvider = ({ children }: { children: React.ReactNode }) => {

const colorScheme = useColorScheme();

const [isDarkTheme, setIsDarkTheme] = useState(colorScheme === "dark");

useEffect(() => {

const loadTheme = async () => {

try {

const storedTheme = await AsyncStorage.getItem(THEME\_KEY);

if (storedTheme !== null) {

setIsDarkTheme(storedTheme === "dark");

} else {

setIsDarkTheme(colorScheme === "dark");

}

} catch (error) {

console.error("Ошибка загрузки темы", error);

}

};

loadTheme();

}, []);

const toggleTheme = async () => {

try {

const newTheme = !isDarkTheme ? "dark" : "light";

if(newTheme === 'dark'){

await unlockAward("night\_mode");

}

setIsDarkTheme(!isDarkTheme);

await AsyncStorage.setItem(THEME\_KEY, newTheme);

} catch (error) {

console.error("Ошибка сохранения темы", error);

}

};

const theme = isDarkTheme ? darkTheme : lightTheme;

return (

<ThemeContext.Provider value={{ theme, toggleTheme, isDarkTheme }}>

{children}

</ThemeContext.Provider>

);

};

export const useThemeContext = () => useContext(ThemeContext);

TimerSettingsContext.tsx

import React, { createContext, useContext, useState, useEffect, ReactNode } from 'react';

import AsyncStorage from '@react-native-async-storage/async-storage';

type TimerSettingsContextType = {

focusDuration: number;

shortBreakDuration: number;

longBreakDuration: number;

setFocusDuration: (value: number) => void;

setShortBreakDuration: (value: number) => void;

setLongBreakDuration: (value: number) => void;

};

const TimerSettingsContext = createContext<TimerSettingsContextType | undefined>(undefined);

export const TimerSettingsProvider = ({ children }: { children: ReactNode }) => {

const [focusDuration, setFocusDurationState] = useState<number>(25);

const [shortBreakDuration, setShortBreakDurationState] = useState<number>(5);

const [longBreakDuration, setLongBreakDurationState] = useState<number>(15);

const STORAGE\_FOCUS\_KEY = 'FOCUS\_DURATION';

const STORAGE\_SHORT\_BREAK\_KEY = 'SHORT\_BREAK\_DURATION';

const STORAGE\_LONG\_BREAK\_KEY = 'LONG\_BREAK\_DURATION';

useEffect(() => {

const loadSettings = async () => {

try {

const savedFocus = await AsyncStorage.getItem(STORAGE\_FOCUS\_KEY);

const savedShortBreak = await AsyncStorage.getItem(STORAGE\_SHORT\_BREAK\_KEY);

const savedLongBreak = await AsyncStorage.getItem(STORAGE\_LONG\_BREAK\_KEY);

if (savedFocus !== null && savedShortBreak !== null && savedLongBreak !== null) {

setFocusDurationState(Number(savedFocus));

setShortBreakDurationState(Number(savedShortBreak));

setLongBreakDurationState(Number(savedLongBreak));

}

} catch (e) {

console.warn('Ошибка при загрузке таймера из AsyncStorage', e);

}

};

loadSettings();

}, []);

const setFocusDuration = async (value: number) => {

try {

await AsyncStorage.setItem(STORAGE\_FOCUS\_KEY, value.toString());

setFocusDurationState(value);

} catch (e) {

console.warn('Ошибка при сохранении таймера в AsyncStorage', e);

}

};

const setShortBreakDuration = async (value: number) => {

try {

await AsyncStorage.setItem(STORAGE\_SHORT\_BREAK\_KEY, value.toString());

setShortBreakDurationState(value);

} catch (e) {

console.warn('Ошибка при сохранении таймера в AsyncStorage', e);

}

};

const setLongBreakDuration = async (value: number) => {

try {

await AsyncStorage.setItem(STORAGE\_LONG\_BREAK\_KEY, value.toString());

setLongBreakDurationState(value);

} catch (e) {

console.warn('Ошибка при сохранении таймера в AsyncStorage', e);

}

};

return (

<TimerSettingsContext.Provider value={{

focusDuration, setFocusDuration,

shortBreakDuration, setShortBreakDuration,

longBreakDuration, setLongBreakDuration

}}>

{children}

</TimerSettingsContext.Provider>

);

};

export const useTimerSettings = () => {

const context = useContext(TimerSettingsContext);

if (!context) {

throw new Error('useTimerSettings must be used within a TimerSettingsProvider');

}

return context;

};

awardService.ts

import { doc, getDoc, setDoc, updateDoc } from "firebase/firestore";

import { db } from "./firebaseConfig";

import { getUserId } from "./userService";

// Тип для удобства

type AwardId =

| "first\_page" //

| "first\_note" //

| "first\_task\_done" //

| "focused\_3\_days" //

| "focused\_7\_days" //

| "20\_notes" //

| "20\_tasks\_done" //

| "deleted\_5\_notes" //

| "focus\_hour" //

| "karate" //

| "night\_mode" //

| "10\_pages" //

| "done\_after\_7\_days" //

| "5\_days\_good\_mood" //

| "mood\_sad" //

// Получить все награды

export const getUserAwards = async (): Promise<Record<string, boolean>> => {

const userId = await getUserId();

const docRef = doc(db, "awards", userId);

const docSnap = await getDoc(docRef);

return docSnap.exists() ? docSnap.data() as Record<string, boolean> : {};

};

//Разблокировать награду

export const unlockAward = async (awardId: AwardId): Promise<void> => {

const userId = await getUserId();

const docRef = doc(db, "awards", userId);

await updateDoc(docRef, {

[awardId]: true,

}).catch(async () => {

await setDoc(docRef, { [awardId]: true });

});

};

counterService.ts

import AsyncStorage from '@react-native-async-storage/async-storage';

//счётчик для достижений

export const getLocalCount = async (key: string): Promise<number> => {

const val = await AsyncStorage.getItem(key);

return val ? parseInt(val, 10) : 0;

};

export const saveLocalCount = async (key: string, value: number) => {

await AsyncStorage.setItem(key, value.toString());

};

firebaseConfig.ts

import { initializeApp } from "firebase/app";

import { getFirestore } from "firebase/firestore";

const firebaseConfig = {

apiKey: "AIzaSyAHN803\_xaNOYKYZrDjRfmkhcelGFg08rk",

authDomain: "lumina-a948b.firebaseapp.com",

projectId: "lumina-a948b",

storageBucket: "lumina-a948b.firebasestorage.app",

messagingSenderId: "711439690799",

appId: "1:711439690799:web:d5ae7b4fbaa0a26752aada"

};

const app = initializeApp(firebaseConfig);

const db = getFirestore(app);

export { db };

moodService.ts

import { collection, addDoc, getDocs, orderBy, query, where } from 'firebase/firestore'

import { db } from './firebaseConfig'

import { getUserId } from './userService';

import { unlockAward } from './awardService';

import { getLocalCount, saveLocalCount } from './counterService';

type MoodType = {

title: string;

moodValue: number;

description: string;

}

const MOODS\_COLLECTION = "moods";

//Сохранение новой записи

export async function saveMood(mood: {

title: string;

moodValue: number;

description: string;

})

{

try {

const userId = await getUserId();

await addDoc(collection(db, MOODS\_COLLECTION), {

...mood,

user\_id: userId,

});

//достижения

if(mood.moodValue === 1){

await unlockAward("mood\_sad");

}

let count = await getLocalCount("goodMoodCount") || 0;

count++;

await saveLocalCount("goodMoodCount", count);

if(count >= 5){

await unlockAward("5\_days\_good\_mood")

}

} catch(error){

console.error("Ошибка при добавлении настроения: ", error)

}

}

//Получение всех записей, отсортированных по дате

export async function getMoods(): Promise<MoodType[]>{

try{

const userId = await getUserId();

const snapshot = await getDocs(

query(collection(db, MOODS\_COLLECTION),

where("user\_id", "==", userId),

orderBy('title'))

);

const moods = snapshot.docs.map(doc => {

const data = doc.data();

return {

title: data.title,

moodValue: data.moodValue,

description: data.description

} as MoodType;

});

return moods;

} catch (error) {

console.error("Ошибка при получении настроений: ", error)

return [];

}

}

//запрос к нейронке по апи

const getAdviceFromOpenRouter = async (moodDescription: string) => {

const response = await fetch("https://openrouter.ai/api/v1/chat/completions", {

method: "POST",

headers: {

"Authorization": "Bearer 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",

"Content-Type": "application/json"

},

body: JSON.stringify({

model: "meta-llama/llama-4-scout:free",

messages: [

{

role: "user",

content: `

Пользователь описал своё настроение так: "${moodDescription}".

Дай короткий, но исчерпывающий персонализированный совет, на основе данного описания,

который поможет ему чувствовать себя лучше. В конце обязательно похвали пользователя.

Перед ответом хорошо подумай и дай совет, который действительно поможет пользователю.

`

}

]

})

});

if (!response.ok) {

const errorText = await response.text();

console.error("Ошибка OpenRouter:", errorText);

throw new Error(`Ошибка OpenRouter: ${response.status}`);

}

const result = await response.json();

return result.choices?.[0]?.message?.content ?? "Не удалось получить совет.";

};

export async function getMoodAdvice(description: string): Promise<string> {

try {

const advice = await getAdviceFromOpenRouter(description);

return advice;

} catch (error) {

console.error("Ошибка при получении совета: ", error);

return "Не удалось получить совет.";

}

}

streakService.ts

import AsyncStorage from '@react-native-async-storage/async-storage';

import { unlockAward } from './awardService';

const STREAK\_KEY = 'user\_streak';

const LAST\_OPEN\_KEY = 'last\_open\_date';

const getToday = () => {

const now = new Date();

return new Date(now.getFullYear(), now.getMonth(), now.getDate()); // обрезаем время

};

export const updateStreak = async () => {

const today = getToday();

const lastOpenStr = await AsyncStorage.getItem(LAST\_OPEN\_KEY);

let streak = 1;

if (lastOpenStr) {

const lastOpen = new Date(lastOpenStr);

const diff = (today.getTime() - lastOpen.getTime()) / (1000 \* 60 \* 60 \* 24);

if (diff === 1) {

const storedStreak = await AsyncStorage.getItem(STREAK\_KEY);

streak = storedStreak ? parseInt(storedStreak, 10) + 1 : 2;

} else if (diff === 0) {

return; // уже открывали сегодня

} else {

streak = 1; // streak сброшен

}

}

await AsyncStorage.setItem(STREAK\_KEY, streak.toString());

await AsyncStorage.setItem(LAST\_OPEN\_KEY, today.toISOString());

// выдача наград

if (streak === 3) await unlockAward("focused\_3\_days");

if (streak === 7) await unlockAward("focused\_7\_days");

if (streak === 30) await unlockAward("karate");

};

taskService.tsx

import { db } from './firebaseConfig';

import { getUserId } from './userService';

import {

collection,

getDocs,

setDoc,

deleteDoc,

doc,

updateDoc,

query,

where,

} from 'firebase/firestore';

import { unlockAward } from './awardService';

import PageType from '../types/PageType';

import NoteType from '../types/NoteType';

// ======= Страницы =======

export const getPages = async (): Promise<PageType[]> => {

const userId = await getUserId();

const q = query(collection(db, 'pages'), where('user\_id', '==', userId));

const snapshot = await getDocs(q);

return snapshot.docs.map(doc => doc.data() as PageType);

};

export const addPage = async (page: PageType) => {

const userId = await getUserId();

const docRef = doc(db, 'pages', page.id);

await setDoc(docRef, { ...page, user\_id: userId });

await unlockAward("first\_page");

};

export const deletePage = async (id: string) => {

try {

const pageRef = doc(db, 'pages', id);

// Удаляем связанные заметки

const notesQuery = query(

collection(db, 'notes'),

where('pageId', '==', id)

);

const notesSnapshot = await getDocs(notesQuery);

const deletePromises = notesSnapshot.docs.map(noteDoc =>

deleteDoc(noteDoc.ref)

);

await Promise.all(deletePromises);

// Удаляем саму страницу

await deleteDoc(pageRef);

} catch (error) {

console.error('Ошибка при удалении страницы:', error);

}

};

export const updatePage = async (

id: string,

data: Partial<Pick<PageType, 'title'>>

) => {

const docRef = doc(db, 'pages', id);

await updateDoc(docRef, data);

};

// ======= Заметки =======

export const getNotes = async (pageId: string): Promise<NoteType[]> => {

const userId = await getUserId();

const q = query(collection(db, 'notes'),

where('pageId', '==', pageId),

where('user\_id', '==', userId)

);

const snapshot = await getDocs(q);

return snapshot.docs.map(doc => doc.data() as NoteType);

};

export const addNote = async (note: NoteType) => {

const userId = await getUserId();

const docRef = doc(db, 'notes', note.id);

await setDoc(docRef, { ...note, user\_id: userId });

await unlockAward("first\_page");

};

export const deleteNote = async (id: string) => {

const docRef = doc(db, 'notes', id);

await deleteDoc(docRef);

};

export const updateNote = async (

id: string,

data: Partial<Pick<NoteType, 'text' | 'done'>>

) => {

const docRef = doc(db, 'notes', id);

await updateDoc(docRef, data);

};

//запрос к нейронке по апи

const getAdviceFromOpenRouter = async (tasksList: string[]) => {

const response = await fetch("https://openrouter.ai/api/v1/chat/completions", {

method: "POST",

headers: {

"Authorization": "Bearer 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",

"Content-Type": "application/json"

},

body: JSON.stringify({

model: "meta-llama/llama-4-scout:free",

messages: [

{

role: "user",

content: `

У пользователя есть список дел: "${tasksList}".

Дай короткий, но исчерпывающий персонализированный совет, на основе данного списка,

который поможет пользователю справиться с делами лучше и быстрее.

Можешь даже дать советы о том, как выполнить эти дела максимально эффективно.

Перед ответом хорошо подумай и дай совет, который действительно поможет пользователю.

`

}

]

})

});

if (!response.ok) {

const errorText = await response.text();

console.error("Ошибка OpenRouter:", errorText);

throw new Error(`Ошибка OpenRouter: ${response.status}`);

}

const result = await response.json();

return result.choices?.[0]?.message?.content ?? "Не удалось получить совет.";

};

export async function getTaskAdvice(tasksList: string[]): Promise<string> {

try {

const advice = await getAdviceFromOpenRouter(tasksList);

return advice;

} catch (error) {

console.error("Ошибка при получении совета: ", error);

return "Не удалось получить совет.";

}

}

userService.ts

import AsyncStorage from '@react-native-async-storage/async-storage';

import { v4 as uuidv4 } from 'uuid';

export const getUserId = async (): Promise<string> => {

let id = await AsyncStorage.getItem('user\_id');

if (!id) {

id = uuidv4();

await AsyncStorage.setItem('user\_id', id);

}

return id;

};

Award.tsx

import \* as React from 'react';

import { Card } from 'react-native-paper';

import MaterialCommunityIcons from '@expo/vector-icons/MaterialCommunityIcons';

import { useThemeContext } from "@/components/context/ThemeContext";

type CardProps = {

title: string,

subtitle: string,

icon: string,

done: boolean;

}

export default function Award({title, subtitle, icon, done} :CardProps){

const { theme } = useThemeContext();

return(

<Card style={{marginVertical: 5, opacity: done ? 1 : 0.5}} mode={done ? 'elevated' : 'outlined'}>

<Card.Title

title={title}

subtitle={subtitle}

style={{

height: 80

}}

left={

() =>

<MaterialCommunityIcons

name={icon as any}

size={32}

color={theme.colors.primary}

/>

}

/>

</Card>

)

}

AwardsBox.tsx

import { View, ScrollView } from 'react-native'

import { useCallback, useState } from 'react'

import AwardsEl from './AwardEl'

import { getUserAwards } from '@/components/data/awardService'

import { useFocusEffect } from '@react-navigation/native'

export default function AwardsBox(){

const [awards, setAwards] = useState<Record<string, boolean>>({});

useFocusEffect(

useCallback(() => {

const fetchAwards = async () => {

const data = await getUserAwards();

setAwards(data);

};

fetchAwards();

}, [])

);

return(

<View style={{marginHorizontal: 15, flex: 1}}>

<ScrollView>

<AwardsEl

title='Первый шаг'

subtitle='Создал первую страницу'

icon="walk"

done={awards["first\_page"] === true}

/>

<AwardsEl

title='Начало пути'

subtitle='Создал первую заметку'

icon="file-document-edit"

done={awards["first\_note"] === true}

/>

<AwardsEl

title='Первые плоды'

subtitle='Отметил первую выполненную задачу'

icon="check-bold"

done={awards["first\_task\_done"] === true}

/>

<AwardsEl

title='Фокус 3 дня'

subtitle='Открывал приложение 3 дня подряд'

icon="calendar-check"

done={awards["focused\_3\_days"] === true}

/>

<AwardsEl

title='Фокус недели'

subtitle='Открывал приложение 7 дней подряд'

icon="calendar-star"

done={awards["focused\_7\_days"] === true}

/>

<AwardsEl

title='Мастер заметок'

subtitle='Создал 20 заметок'

icon="note-multiple"

done={awards["20\_notes"] === true}

/>

<AwardsEl

title='Завершитель'

subtitle='Выполнил 20 задач'

icon="checkbox-marked"

done={awards["20\_tasks\_done"] === true}

/>

<AwardsEl

title='Чистильщик'

subtitle='Удалил 5 заметок'

icon="pistol"

done={awards["deleted\_5\_notes"] === true}

/>

<AwardsEl

title='Глубокий фокус'

subtitle='Работал 1 ч+'

icon="timer-outline"

done={awards["focus\_hour"] === true}

/>

<AwardsEl

title='Чёрный пояс по фокусу'

subtitle='Открыл приложение 30 дней подряд'

icon="karate"

done={awards["karate"] === true}

/>

<AwardsEl

title='Ночной режим'

subtitle='Использовал тёмную тему'

icon="weather-night"

done={awards["night\_mode"] === true}

/>

<AwardsEl

title='Скиталец страниц'

subtitle='Создал 10 разных страниц'

icon="book-open-variant"

done={awards["10\_pages"] === true}

/>

<AwardsEl

title='Терпеливый'

subtitle='Выполнил задачу через 7+ дней'

icon="timer-sand-complete"

done={awards["done\_after\_7\_days"] === true}

/>

<AwardsEl

title='Настроение плюс'

subtitle='Отметил 5 дней хорошего настроения'

icon='emoticon-cool-outline'

done={awards["5\_days\_good\_mood"] === true}

/>

<AwardsEl

title='Искренность'

subtitle='Отметил плохое настроение'

icon="emoticon-sad"

done={awards["mood\_sad"] === true}

/>

</ScrollView>

</View>

)

}

ButtonEl.tsx

import { View, StyleSheet } from "react-native"

import { FAB } from "react-native-paper"

import { useThemeContext } from '@/components/context/ThemeContext';

type ButtonProps = {

typeButton: string,

icon: string,

label: string,

onPress: () => void,

disabled?: boolean;

visible?: boolean;

}

export default function ButtonEl({typeButton, icon, label, onPress, disabled, visible}: ButtonProps){

const { theme } = useThemeContext();

const styles = StyleSheet.create({

buttonLabel:{

width: 150,

boxShadow: `0 2 10 0 ${theme.colors.onSurfaceDisabled}`,

borderRadius: 20

},

buttonFocus:{

width: 120,

paddingHorizontal: 20,

marginTop: 50,

boxShadow: `0 2 10 0 ${theme.colors.onSurfaceDisabled}`,

borderRadius: 20

},

buttonMoodEdit:{

width: 55,

paddingHorizontal: 0,

paddingVertical: 0,

boxShadow: `0 2 10 0 ${theme.colors.onSurfaceDisabled}`,

borderRadius: 20

}

})

return(

<View>

<FAB

style={

typeButton === 'buttonLabel' ? styles.buttonLabel :

typeButton === 'buttonFocus' ? styles.buttonFocus :

styles.buttonMoodEdit

}

icon={icon}

label={label}

onPress={onPress}

size="medium"

customSize={

typeButton === 'buttonFocus' ? 75 : 55

}

color={theme.colors.primary}

disabled={disabled}

visible={visible}

/>

</View>

)

}

MoodButton.tsx

import { View, StyleSheet } from "react-native";

import { SegmentedButtons } from "react-native-paper"

import MaterialCommunityIcons from '@expo/vector-icons/MaterialCommunityIcons';

type MoodButtonProps = {

moodValue: number,

setMoodValue: (e: any) => void;

}

export default function MoodButton({moodValue, setMoodValue}: MoodButtonProps){

return(

<View>

<SegmentedButtons

value={moodValue}

onValueChange={setMoodValue}

buttons={[

{

icon: ({ color }) =>

<MaterialCommunityIcons

name="emoticon-dead-outline" size={32} color={color} />,

value: 1

},

{

icon: ({ color }) =>

<MaterialCommunityIcons

name="emoticon-sad-outline" size={32} color={color} />,

value: 2

},

{

icon: ({ color }) =>

<MaterialCommunityIcons

name="emoticon-neutral-outline" size={32} color={color} />,

value: 3

},

{

icon: ({ color }) =>

<MaterialCommunityIcons

name="emoticon-happy-outline" size={32} color={color} />,

value: 4

},

{

icon: ({ color }) =>

<MaterialCommunityIcons

name="emoticon-excited-outline" size={32} color={color} />,

value: 5

},

]}

style={{

marginHorizontal: 5

}}

/>

</View>

)

}

MoodChart.tsx

import { useEffect, useState } from 'react';

import { View, Text } from 'react-native';

import { CartesianChart, Line } from 'victory-native';

import { useFont } from '@shopify/react-native-skia';

import spaceMono from '@/assets/fonts/SpaceMono-Regular.ttf';

import { useThemeContext } from "@/components/context/ThemeContext";

type MoodType = {

title: string; // метка времени в миллисекундах

moodValue: number;

description: string;

};

type MoodChartProps = {

moods: MoodType[];

};

export default function MoodChart({ moods }: MoodChartProps) {

//цветовая тема

const { theme } = useThemeContext();

//состояния времени, чтобы выводить график

const [data, setData] = useState<{ x: number; y: number; label: string }[]>([]);

//шрифт

const font = useFont(spaceMono, 12);

//посылаем асинхронный запрос к бд, получаем все отметки настроений и форматируем даты

useEffect(() => {

const limitedMoods = moods.slice(-8); // берём только последние 8 записей

const formattedData = [...limitedMoods]

.sort((a, b) => parseInt(a.title) - parseInt(b.title))

.map((mood, index) => {

const date = new Date(parseInt(mood.title));

const label = date.toLocaleDateString('ru-RU', {

day: '2-digit',

month: '2-digit'

});

return {

x: index,

y: mood.moodValue,

label,

};

});

setData(formattedData);

}, [moods]);// график обновляется при изменении moods

//возвращаем линейный график из настроений

return (

<View style={{ height: 150, width: '95%', marginHorizontal: 5 }}>

<CartesianChart

data={data}

xKey="x"

yKeys={['y']}

axisOptions={{

font,

labelColor: theme.colors.onSurface,

lineColor: theme.colors.outline

}}

>

{({ points }) => (

<Line points={points.y} color={theme.colors.primary} strokeWidth={3} />

)}

</CartesianChart>

{/\* Подписи под графиком \*/}

<View style={{ flexDirection: 'row', justifyContent: 'space-between', marginTop: 4 }}>

{data.map((point, index) => (

<Text key={index} style={{ fontSize: 12, color: theme.colors.onSurface }}>

{point.label}

</Text>

))}

</View>

</View>

);

}

InputEl.tsx

import { forwardRef } from 'react';

import { View } from "react-native";

import { TextInput } from "react-native-paper";

type InputProps = {

inputType: string;

placeholder: string;

value: string;

onChangeText: (text: string) => void;

onSubmitEditing?: (e: any) => void;

}

const InputEl = forwardRef<any, InputProps>(

({ inputType, placeholder, value, onChangeText, onSubmitEditing}, ref) => {

return (

<View style={{marginHorizontal: 15, width: inputType == 'mood' ? 280 : 350}}>

<TextInput

ref={ref}

mode="outlined"

placeholder={placeholder}

keyboardType={inputType == 'settings' ? 'number-pad' : 'default'}

value={value}

onChangeText={(text) => {

inputType == 'settings'

? onChangeText(text.replace(/[^0-9]/g, ''))

: onChangeText(text)

}}

onSubmitEditing={onSubmitEditing}

multiline={inputType == 'mood'}

/>

</View>

)

}

);

export default InputEl;

ListEl.tsx

import { View, StyleSheet } from 'react-native';

import { Checkbox } from 'react-native-paper';

import { useThemeContext } from '@/components/context/ThemeContext';

type ListProps ={

label: string,

listType: string,

checked?: boolean,

selected?: boolean,

onToggle: () => void;

onLongPress?: () => void;

}

export default function ListEl({

label, listType, checked, selected, onToggle, onLongPress

}: ListProps){

const { theme } = useThemeContext();

const styles = StyleSheet.create({

itemContainer:{

width: '100%',

marginVertical: 5,

backgroundColor: selected ? theme.colors.inversePrimary : 'transparent'

},

page:{

fontSize: 24

},

note:{

fontSize: 18

},

noteComplete:{

fontSize: 18,

color: 'transparent',

textShadowColor: theme.colors.onSurfaceDisabled,

textShadowOffset: {width : 0, height: 2},

textShadowRadius: 4,

textDecorationStyle: 'solid',

textDecorationLine: 'line-through'

},

noteSelected:{

fontSize: 18

},

item:{

width: '100%'

}

})

return(

<View style={styles.itemContainer}>

<Checkbox.Item

style={styles.item}

labelStyle={

checked && listType == 'note'

? styles.noteComplete

: selected && listType == 'note'

? styles.noteSelected

: listType == 'page'

? styles.page

: styles.note

}

label={label}

status={checked ? 'checked' : 'unchecked'}

onPress={onToggle}

onLongPress={onLongPress}

/>

</View>

)

}

MoodList.tsx

import { View } from "react-native"

import { Card } from 'react-native-paper';

import MaterialCommunityIcons from '@expo/vector-icons/MaterialCommunityIcons';

import { useThemeContext } from "@/components/context/ThemeContext";

import LabelText from "../Typography/LabelText";

type MoodListProps = {

title: string,

moodValue: number,

description: string;

}

export default function MoodList({title, moodValue, description}: MoodListProps){

const { theme } = useThemeContext();

return(

<View style={{marginHorizontal: 15, marginVertical: 5}}>

<Card>

<Card.Title style={{

minHeight: 40,

marginTop: 10

}}

title={title}

left={

() =>

<MaterialCommunityIcons

name={

moodValue == 1 ? 'emoticon-dead-outline' :

moodValue == 2 ? 'emoticon-sad-outline' :

moodValue == 3 ? 'emoticon-neutral-outline' :

moodValue == 4 ? 'emoticon-happy-outline' :

'emoticon-excited-outline'

}

size={32}

color={theme.colors.primary}/>

} />

<Card.Content style={{

paddingBottom: description == '' ? 0 : 16

}}>

<LabelText>

{description}

</LabelText>

</Card.Content>

</Card>

</View>

)

}

TaskList.tsx

import { useState, useRef, useEffect } from 'react';

import { View, Keyboard, ScrollView, TextInput } from 'react-native';

import { FAB, ActivityIndicator, Portal, Dialog } from 'react-native-paper';

import PageScreen from '../../Screens/PageScreen';

import NoteScreen from '../../Screens/NoteScreen';

import MainText from '../Typography/MainText';

import InputEl from '../Inputs/InputEl';

import ButtonEl from '../Buttons/ButtonEl';

import LabelText from '../Typography/LabelText';

import { useTaskLogic } from '../../hooks/useTaskLogic';

import { getPages, getNotes, getTaskAdvice } from '@/components/data/taskService';

//тип для страниц, каждая имеет айди и что в ней записано

type PageType = {

id: string;

title: string;

}

//тип для записей, каждая имеет айди, что в ней записано и состояние выполнения

type NoteType = {

id: string;

text: string;

done: boolean;

pageId: string;

}

export default function TaskList() {

//переменные для массива задач и страниц

const [notes, setNotes] = useState<NoteType[]>([]);

const [pages, setPages] = useState<PageType[]>([]);

//значение в поле ввода

const [inputValue, setInputValue] = useState('');

//выбранные задачи и страницы

const [selectedNotes, setSelectedNotes] = useState<string[]>([]);

const [selectedPages, setSelectedPages] = useState<string[]>([]);

//айди изменяемого элемент

const [editItemId, setEditItemId] = useState<string | null>(null)

//состояние выбранной страницы или списка страниц

const [currentPageId, setCurrentPageId] = useState<string | null>(null);

//отсылка к инпуту, чтобы можно было сделать фокус

const inputRef = useRef<TextInput>(null)

//переменные для состояния кнопок, можно ли удалять или изменять

const canDelete = selectedPages.length >= 1 || selectedNotes.length >= 1;

const canEdit = selectedPages.length === 1 || selectedNotes.length === 1;

//вывод названия текущей страницы

const currentPage = pages.find(page => page.id === currentPageId);

//состояния для видимости всплывающего окна

const [visibleDialog, setVisibleDialog] = useState(false);

//совет по делам от нейросети

const [advise, setAdvise] = useState('');

//состояние загрузки

const [loading, setLoading] = useState(true);

//импортируемые функции из хука

const {

handleAddOrEditItem,

toggleOnPress,

toggleOnLongPress,

handleDeleteItems,

handleEditItem,

} = useTaskLogic({

pages, setPages,

notes, setNotes,

selectedPages, setSelectedPages,

selectedNotes, setSelectedNotes,

inputValue, setInputValue,

currentPageId, setCurrentPageId,

editItemId, setEditItemId,

inputRef

});

//загрузка данных из бд при запуске

useEffect(() => {

setLoading(true);

const loadData = async () => {

const fetchedPages = await getPages();

setPages(fetchedPages);

};

setLoading(false);

loadData();

}, []);

//загрузка данных из бд при смене страницы

useEffect(() => {

setLoading(true);

const loadNotes = async () => {

if (currentPageId) {

const fetchedNotes = await getNotes(currentPageId);

setNotes(fetchedNotes);

} else {

setNotes([]);

}

setLoading(false);

};

loadNotes();

}, [currentPageId]);

//функция загрузки совета от нейросети

const showTaskAdvise = async () => {

//показ всплывающего окна

setVisibleDialog((prev) => !prev)

//совет от нейросети

const textFromNotes = notes.map(note => note.text);

const advise = await getTaskAdvice(textFromNotes);

setAdvise(advise);

}

//фукнция закрытия всплывающего окна

const closeTaskAdvise = () => {

setVisibleDialog((prev) => !prev);

setAdvise('');

}

return (

<View style={{ flex: 1, width: '100%', height: '90%' }}>

{/\* Всплывающее окно с советом \*/}

<View>

<Portal>

<Dialog visible={visibleDialog} onDismiss={closeTaskAdvise} style={{maxHeight: '90%'}}>

<Dialog.Title>Совет от нейросети</Dialog.Title>

<Dialog.ScrollArea>

<ScrollView >

<Dialog.Content style={{paddingHorizontal: 0, marginVertical: 20}}>

{/\* Индикатор загрузки \*/}

<ActivityIndicator animating={advise == '' ? true : false} style={{position: 'relative', width: '100%'}}/>

<LabelText>{advise}</LabelText>

</Dialog.Content>

</ScrollView>

</Dialog.ScrollArea>

<Dialog.Actions>

<ButtonEl

typeButton='buttonLabel'

icon='check'

label='Хорошо'

onPress={closeTaskAdvise}

/>

</Dialog.Actions>

</Dialog>

</Portal>

</View>

<View style={{

width: '100%',

flexDirection: 'row'

}}>

<FAB

style={{

height: 55,

marginTop: 30,

marginLeft: 10

}}

icon='arrow-left'

onPress={() => setCurrentPageId(null)}

visible={currentPageId !== null}

/>

<MainText>{currentPageId === null ? 'Мои страницы' : currentPage?.title}</MainText>

</View>

{/\* Вывод списка задач \*/}

<ScrollView>

{ currentPageId === null

// Страницы

? pages.map((page) => (

<PageScreen

key={page.id}

page={page}

selectedPages={selectedPages}

toggleOnPress={toggleOnPress}

toggleOnLongPress={toggleOnLongPress}

/>

))

// Заметки

: notes

.filter(note => note.pageId === currentPageId)

.map((note) => (

<NoteScreen

key={note.id}

note={note}

selectedNotes={selectedNotes}

toggleOnPress={toggleOnPress}

toggleOnLongPress={toggleOnLongPress}

/>

))

}

{/\* Вывод в конце всех задач пустое поле ввода для новой задачи \*/}

<InputEl

ref={inputRef}

inputType='list'

value={inputValue}

onChangeText={setInputValue}

onSubmitEditing={handleAddOrEditItem}

placeholder={currentPageId === null ? 'Создать новую страницу' : 'Напишите заметку'}

/>

</ScrollView>

{/\* Индикатор загрузки \*/}

<ActivityIndicator animating={loading} style={{position: 'relative', width: '100%'}}/>

{/\* Кнопки для управления задачами \*/}

<View style={{

width: '100%',

flexDirection: 'row',

flexWrap: 'wrap',

justifyContent: 'space-evenly',

rowGap: 20,

marginTop: 20,

marginBottom: -70

}}>

<ButtonEl

typeButton='buttonLabel'

icon='delete'

label={currentPageId === null ? 'Удалить страницу' : 'Удалить заметку'}

onPress={handleDeleteItems}

disabled={!canDelete}

/>

<ButtonEl

typeButton='buttonLabel'

icon='pencil-outline'

label={currentPageId === null ? 'Изменить страницу' : 'Изменить заметку'}

onPress={handleEditItem}

disabled={!canEdit}

/>

<ButtonEl

typeButton='buttonLabel'

icon='robot-confused-outline'

label='Совет'

onPress={showTaskAdvise}

disabled={notes.length < 1}

visible={currentPageId !== null}

/>

</View>

</View>

);

}

Timer.tsx

import { useState, useEffect } from 'react';

import { View, Vibration } from 'react-native';

import Animated, { useSharedValue, withTiming, withSequence, useAnimatedProps } from 'react-native-reanimated';

import { Svg, Circle } from 'react-native-svg';

import \* as Notifications from 'expo-notifications';

import { useThemeContext } from '@/components/context/ThemeContext';

import { useTimerSettings } from "@/components/context/TimerSettingsContext";

import MainText from '@/components/elements/Typography/MainText';

import ButtonEl from '../Buttons/ButtonEl';

import { unlockAward } from '@/components/data/awardService';

const AnimatedCircle = Animated.createAnimatedComponent(Circle) //сам таймер (основной круг)

const AnimatedCircleBack = Animated.createAnimatedComponent(Circle) // фоновый круг

//статусы таймера

type TimerMode = 'focus' | 'shortBreak' | 'longBreak'

export default function Timer(){

//берем время из провайдера

const {

focusDuration,

shortBreakDuration,

longBreakDuration

} = useTimerSettings()

//создаём переменные состояния

const [mode, setMode] = useState<TimerMode>('focus');

//вычисляем количество секунд из переменной провайдера

const [ secondsLeft, setsecondsLeft] = useState(focusDuration \* 60);

//активен ли таймер

const [isRunning, setIsRunning] = useState(false);

//счётчик пройдённых кругов

const [cycleCount, setCycleCount] = useState(0);

//следим за изменением состояния таймера

//в зависимости от состояния устанавливаем оставшееся время

useEffect(() => {

let duration = 0;

if(mode == 'focus') duration = focusDuration \* 60;

if(mode == 'shortBreak') duration = shortBreakDuration \* 60;

if(mode == 'longBreak') duration = longBreakDuration \* 60;

setsecondsLeft(duration);

}, [mode, focusDuration, shortBreakDuration, longBreakDuration])

//берем тему

const { theme } = useThemeContext();

//переменные для кругов

const radius = 120;

const strokeWidth = 10

const innerRadius = radius - strokeWidth / 2

const circumference = 2 \* Math.PI \* innerRadius

//переменные для анимации круга

const progress = useSharedValue<number>(0)

const animatedStrokeWidth = useSharedValue<number>(strokeWidth)

//стили, которые анимируем

const animatedPropsFill = useAnimatedProps(() => ({

strokeDasharray: [circumference \* progress.value, circumference],

strokeWidth: animatedStrokeWidth.value,

stroke: mode == 'focus' ? theme.colors.primary : theme.colors.inversePrimary

}))

const animatedPropsBack = useAnimatedProps(() => ({

strokeWidth: animatedStrokeWidth.value

}))

//Обработчик уведомлений

Notifications.setNotificationHandler({

handleNotification: async () => ({

shouldShowAlert: true,

shouldPlaySound: false,

shouldSetBadge: false,

}),

});

//фукнция, которая проверяет какой этап следующий

//устанавливает его плюс включает вибрацию и уведомление

const handleTimerEnd = async () => {

Vibration.vibrate();

await Notifications.scheduleNotificationAsync({

content: {

title: '⏰ Время вышло!',

body: mode === 'focus' ? 'Хорошая работа! Пора сделать перерыв'

: 'Отдохнули? Пора сфокусироваться'

},

trigger: null,

});

if(mode == 'focus'){

const nextCycle = cycleCount + 1;

setCycleCount(nextCycle);

if(nextCycle % 4 == 0){

setMode('longBreak');

}

else{

setMode('shortBreak');

}

//достижение

if(cycleCount \* focusDuration >= 60){

await unlockAward("focus\_hour");

}

}

else{

setMode('focus');

}

setIsRunning(false);

}

//отслеживание состояния активности

useEffect(() => {

if(!isRunning) return;

//интервал, который отсчитывает секунды (отнимает время) и запускает анимацию

const interval = setInterval(() => {

setsecondsLeft((prev) => {

if(prev <= 0){

progress.value = withTiming(progress.value - progress.value, {duration: 1000})

clearInterval(interval);

handleTimerEnd();

return 0;

}

progress.value = withTiming(progress.value + (1 / (focusDuration \* 60) ), {duration: 1000})

animatedStrokeWidth.value = withSequence(

withTiming(animatedStrokeWidth.value - 2, {duration: 500}),

withTiming(animatedStrokeWidth.value = 10, {duration: 500})

)

return prev - 1;

})

}, 1000);

return () => clearInterval(interval)

},[isRunning])

//преобразовываем секунды в формат мм:сс

const formatTime = (totalSeconds: number) => {

const minutes = Math.floor(totalSeconds / 60)

.toString()

.padStart(2, '0');

const seconds = (totalSeconds % 60)

.toString()

.padStart(2, '0');

return `${minutes}:${seconds}`

}

//статусы для вывода на экран

const modeLabel = {

focus: 'Фокус',

shortBreak: 'Перерыв',

longBreak: 'Большой перерыв'

}

// выводим элементы

return(

<View style={{ alignItems: 'center', justifyContent: 'center' }} >

<Svg style={{height: 250, width: 250}}>

<AnimatedCircleBack

cx={radius}

cy={radius}

r={innerRadius}

originX={radius}

originY={radius}

fill='transparent'

stroke={theme.colors.surfaceVariant}

strokeLinecap='round'

animatedProps={animatedPropsBack}

/>

<AnimatedCircle

cx={radius}

cy={radius}

r={innerRadius}

originX={radius}

originY={radius}

fill='transparent'

strokeLinecap='round'

rotation='-90'

animatedProps={animatedPropsFill}

/>

<View style={{

width: '95%'

}}>

<MainText style={{textAlign: 'center', paddingVertical: 50}}>{modeLabel[mode]}</MainText>

<MainText style={{textAlign: 'center', paddingVertical: 0}}>{formatTime(secondsLeft)}</MainText>

</View>

</Svg>

<View style={{flexDirection: 'row', columnGap: 50}}>

<ButtonEl

typeButton='buttonFocus'

icon={isRunning ? 'pause' : 'play'}

label=''

onPress={() => setIsRunning((prev) => !prev)}

/>

<ButtonEl

typeButton='buttonFocus'

icon={'skip-next'}

label=''

onPress={() => {

setIsRunning(false);

handleTimerEnd();

setsecondsLeft(

mode === 'focus'

? focusDuration \* 60

: mode === 'shortBreak'

? shortBreakDuration \* 60

: longBreakDuration \* 60

)

progress.value = withTiming(progress.value - progress.value, {duration: 1000})

}}

/>

</View>

</View>

)

}

LabelText.tsx

import { PropsWithChildren } from 'react';

import { View} from 'react-native';

import { Text } from 'react-native-paper';

export default function LabelText({children}: PropsWithChildren<{}>){

return(

<View style={{

width: '100%',

justifyContent: 'flex-start',

alignContent: 'flex-start',

}}>

<Text

variant='bodyLarge'

style={{

fontWeight: 'medium',

marginHorizontal: 30,

marginTop: 0,

paddingVertical: 0

}}

>{children}

</Text>

</View>

)

}

MainText.tsx

import React from 'react';

import { View, StyleProp, TextStyle } from 'react-native';

import { Text } from 'react-native-paper';

import { useThemeContext } from '@/components/context/ThemeContext';

type MainTextProps = {

children: React.ReactNode,

style?: StyleProp<TextStyle>

}

export default function MainText({children, style}: MainTextProps){

const { theme } = useThemeContext();

return(

<View style={{

width: '100%'

}}>

<Text

variant='headlineLarge'

style={[{

fontWeight: '500',

textShadowColor: theme.colors.onSurfaceDisabled,

textShadowOffset: {width : 0, height: 2},

textShadowRadius: 4,

marginHorizontal: 20,

paddingVertical: 40

}, style]}

>{children}

</Text>

</View>

)

}

SwitchText.tsx

import { PropsWithChildren } from 'react';

import { View} from 'react-native';

import { Text } from 'react-native-paper';

export default function SwitchText({children}: PropsWithChildren<{}>){

return(

<View style={{

width: '100%',

flex: 1,

justifyContent: 'flex-start',

alignContent: 'center',

}}>

<Text

variant='titleLarge'

style={{

fontWeight: 'regular',

marginHorizontal: 20,

paddingVertical: 40

}}

>{children}

</Text>

</View>

)

}

useTaskLogic.ts

import { Keyboard, TextInput } from 'react-native';

import {

addPage,

deletePage,

updatePage,

addNote,

deleteNote,

updateNote

} from '@/components/data/taskService'

import { unlockAward } from '@/components/data/awardService';

//подключение типов

import PageType from '../types/PageType';

import NoteType from '../types/NoteType';

import { getLocalCount, saveLocalCount } from '@/components/data/counterService';

//создание типов для хука

type UseTaskLogicParams = {

pages: PageType[];

setPages: React.Dispatch<React.SetStateAction<PageType[]>>;

notes: NoteType[];

setNotes: React.Dispatch<React.SetStateAction<NoteType[]>>;

selectedPages: string[];

setSelectedPages: React.Dispatch<React.SetStateAction<string[]>>;

selectedNotes: string[];

setSelectedNotes: React.Dispatch<React.SetStateAction<string[]>>;

inputValue: string;

setInputValue: React.Dispatch<React.SetStateAction<string>>;

currentPageId: string | null;

setCurrentPageId: React.Dispatch<React.SetStateAction<string | null>>;

editItemId: string | null;

setEditItemId: React.Dispatch<React.SetStateAction<string | null>>;

inputRef: React.RefObject<TextInput>;

};

//хуки :)

export const useTaskLogic = ({

pages, setPages,

notes, setNotes,

selectedPages, setSelectedPages,

selectedNotes, setSelectedNotes,

inputValue, setInputValue,

currentPageId, setCurrentPageId,

editItemId, setEditItemId,

inputRef

}: UseTaskLogicParams) => {

//добавление или изменение элемента в списке

const handleAddOrEditItem = async () => {

const trimmed = inputValue.trim();

if (trimmed === '') return;

//очистка поля ввода и скрытие клавиатуры

setInputValue('');

Keyboard.dismiss();

if (editItemId) {

if (currentPageId === null) {

// редактируем страницу

const updatedPages = pages.map(page =>

page.id === editItemId ? { ...page, title: trimmed } : page

)

setPages(updatedPages);

//в бд

await updatePage(editItemId, { title: trimmed });

setEditItemId(null);

setSelectedPages([]);

} else {

// редактируем заметку

const updatedNotes = notes.map(note =>

note.id === editItemId ? { ...note, text: trimmed } : note

)

setNotes(updatedNotes);

//в бд

await updateNote(editItemId, { text: trimmed });

setEditItemId(null);

setSelectedNotes([]);

}

}

else {

if (currentPageId === null) {

// создаём страницу

const newPage: PageType = {

id: Date.now().toString(),

title: trimmed,

};

setPages([...pages, newPage]);

//в бд

await addPage(newPage)

//достижение

let pageCount = await getLocalCount("pageCount") || 0;

pageCount++;

await saveLocalCount("pageCount", pageCount);

if(pageCount >= 10){

await unlockAward("10\_pages")

}

} else {

// создаём заметку

const newNote: NoteType = {

id: Date.now().toString(),

text: trimmed,

done: false,

pageId: currentPageId,

};

setNotes([...notes, newNote]);

//в бд

await addNote(newNote)

//достижение

await unlockAward('first\_note');

let noteCount = await getLocalCount("noteCount") || 0;

noteCount++;

await saveLocalCount("noteCount", noteCount);

if(noteCount >= 20){

await unlockAward("20\_notes")

}

}

}

};

//обработчик при нажатии на элемент списка

const toggleOnPress = async (id: string) => {

if (currentPageId === null) {

// работа со страницами (выбор страницы)

setSelectedPages(prev =>

prev.includes(id)

? prev.filter(i => i !== id)

: [...prev, id]

);

}

else {

// переключение выполнения задачи (выполнение задачи)

setNotes(prev =>

prev.map(note =>

note.id === id ? { ...note, done: !note.done } : note

)

);

//в бд

const targetNote = notes.find(note => note.id === id);

if (targetNote) {

await updateNote(id, { done: !targetNote.done });

//достижение

const daysAfterCreate = new Date(Math.abs(Date.now() - +targetNote.id)).getDate();

if(daysAfterCreate >= 7){

await unlockAward("done\_after\_7\_days")

}

}

//достижения

await unlockAward("first\_task\_done");

//

let doneCount = await getLocalCount("doneCount") || 0;

doneCount++;

await saveLocalCount("doneCount", doneCount);

if(doneCount >= 20){

await unlockAward("20\_tasks\_done")

}

}

};

//обработчик длительного нажатия на элемент

const toggleOnLongPress = (id: string) => {

if (currentPageId === null) {

// переход на страницу

setCurrentPageId(id);

} else {

// выделение заметки

setSelectedNotes(prev =>

prev.includes(id)

? prev.filter(i => i !== id)

: [...prev, id]

);

}

};

//удаление элементе

const handleDeleteItems = async () => {

if (currentPageId === null) {

//удаление страниц из бд

for(const id of selectedPages){

await deletePage(id);

}

setPages(prev => prev.filter(page => !selectedPages.includes(page.id)));

setSelectedPages([]);

} else {

//удаление заметок из бд

for(const id of selectedNotes){

await deleteNote(id);

}

setNotes(prev => prev.filter(note => !selectedNotes.includes(note.id)));

setSelectedNotes([]);

//счётчик для достижений

let deletionCount = await getLocalCount("deletionCount") || 0;

deletionCount++;

await saveLocalCount("deletionCount", deletionCount);

if(deletionCount >= 5){

await unlockAward("deleted\_5\_notes")

}

}

};

//установка изменяемого элемента

const handleEditItem = () => {

if (currentPageId === null) {

if (selectedPages.length !== 1) return;

const pageId = selectedPages[0];

const pageToEdit = pages.find(page => page.id === pageId);

if (!pageToEdit) return;

setInputValue(pageToEdit.title);

setEditItemId(pageId);

} else {

if (selectedNotes.length !== 1) return;

const noteId = selectedNotes[0];

const noteToEdit = notes.find(note => note.id === noteId);

if (!noteToEdit) return;

setInputValue(noteToEdit.text);

setEditItemId(noteId);

}

//фокус на поле ввода

inputRef.current?.focus();

};

//экспортируемые функции

return {

handleAddOrEditItem,

toggleOnPress,

toggleOnLongPress,

handleDeleteItems,

handleEditItem,

};

};

NoteScreen.tsx

import ListEl from "../elements/Lists/ListEl"

import NoteType from "../types/NoteType"

type NoteScreenProps = {

note: NoteType;

selectedNotes: string[];

toggleOnPress: (id:string) => void;

toggleOnLongPress: (id:string) => void;

}

export default function NoteScreen({note, selectedNotes, toggleOnPress, toggleOnLongPress}: NoteScreenProps){

return(

<ListEl

key={note.id}

label={note.text}

listType='note'

checked={note.done}

selected={selectedNotes.includes(note.id)}

onToggle={() => toggleOnPress(note.id)}

onLongPress={() => toggleOnLongPress(note.id)}

/>

)

}

PageScreen.tsx

import ListEl from "../elements/Lists/ListEl"

import PageType from "../types/PageType"

type PageScreenProps = {

page: PageType;

selectedPages: string[];

toggleOnPress: (id:string) => void;

toggleOnLongPress: (id:string) => void;

}

export default function PageScreen({page, selectedPages, toggleOnPress, toggleOnLongPress}: PageScreenProps){

return(

<ListEl

key={page.id}

label={page.title}

listType='page'

checked={selectedPages.includes(page.id)}

onToggle={() => toggleOnPress(page.id)}

onLongPress={() => toggleOnLongPress(page.id)}

/>

)

}

NoteType.ts

//тип для записей, каждая имеет айди, что в ней записано и состояние выполнения

type NoteType = {

id: string;

text: string;

done: boolean;

pageId: string;

}

export default NoteType;

PageType.ts

//тип для старниц, каждая имеет айди и что в ней записано

type PageType = {

id: string;

title: string;

} export default PageType;