**СОДЕРЖАНИЕ**

1 Аналитическая часть 9

1.1 Описание предметной области 9

1.2 Сравнение с существующими аналогами 11

1.2.1 Интерфейс. 13

1.2.2 Организация и удобство. 14

1.2.3 Уведомления. 14

1.2.4 Интеграция. 14

1.3 Постановка задачи 14

1.4 Обоснование выбора методики, технологии и инструментальных средств проектирования и разработки 16

2 Проектирование и реализация ПО 22

2.1 Разработка модели БД 22

2.2 Разработка пользовательского интерфейса 24

2.3 Разработка основных функций программного обеспечения 35

2.4 Тестирование и отладка программного обеспечения 39

2.5 Защита программного продукта 47

3 Сопровождение 48

3.1 Руководство пользователя 48

3.1.1 Описание приложения. 48

3.1.2 Установка и настройка. 48

3.1.3 Решение проблем. 63

3.1.4 Вывод. 63

ЗАКЛЮЧЕНИЕ 64

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 65

ПРИЛОЖЕНИЕ А

ПРИЛОЖЕНИЕ Б. База данных

##### **ВВЕДЕНИЕ**

В современном мире люди сталкиваются с большим количеством препятствий, которые необходимо выполнить в определенные сроки. Для того, чтобы эффективно распределять свое время и приоритеты, многие люди используют различные инструменты планирования, такие как ежедневники, календари, напоминания, списки дел и т. д. Однако, не все из этих инструментов удобны, функциональны и доступны в любой ситуации. Например, бумажный ежедневник может быть потерян или испорчен, а электронный календарь может не синхронизироваться с другими устройствами или не иметь нужных опций.

Актуальность разработки мобильного планировщика обусловлена тем, что мобильные устройства являются наиболее популярными и доступными для большинства людей. По данным исследования компании Statista, в 2020 году количество пользователей мобильных устройств достигло 6,95 миллиарда человек, что составляет около 90% всего населения земли. Мобильное ПО позволяет использовать возможности смартфонов и планшетов для достижения определенных целей, включая планирование. Однако, не все существующие планировщики удовлетворяют потребностям и ожиданиям конечного потребителя. Некоторые из них имеют сложный интерфейс, низкую производительность, ограниченные функции, высокую стоимость или проблемы с безопасностью.

Предметом данной дипломной работы является разработка мобильного продукта планировщика, который бы обладал следующими преимуществами:

* Простота и удобство использования
* Высокий уровень кастомизации и персонализации (за счет применения динамических тем оформления Material You)
* Повышенная безопасность и конфиденциальность за счёт хранения только на локальном устройстве
* Полная отсутствие коммерциализации и открытый исходный код

Целью данной дипломной работы является создание мобильного приложения планировщика задач, которое бы помогало клиентам эффективно организовывать жизнь и работу, повышать продуктивность и достигать своих целей.

# Аналитическая часть

## Описание предметной области

Управление задачами и проектами — это процесс, который включает в себя планирование, исполнение и контроль деятельности, направленной на достижение определенных целей в ограниченные сроки и ресурсы. Сфера является неотъемлемой частью как профессиональной, так и личной жизни людей, которые хотят быть успешными, продуктивными и организованными.

Задача — это конкретное действие или набор действий, которые необходимо выполнить для реализации части проекта или более общей цели. Они могут быть разного типа, например, повседневные, периодические, краткосрочные, долгосрочные, личные, рабочие, индивидуальные, групповые и т. д. Элементы могут быть сформулированы в виде простых утверждений, например, "купить хлеб" или "написать отчет", или в виде более сложных и детализированных планов, например, "подготовиться к презентации: изучить материалы, составить тезисы, подобрать иллюстрации, отрепетировать выступление". Они могут иметь разные параметры, такие как срок выполнения, приоритет, статус, ответственный, комментарии, привязка к месту, времени, контексту и т. д.

Проект — это совокупность элементов, объединенных общей целью. Проекты также могут быть разного типа, например, личные, профессиональные, образовательные, творческие, социальные и т. д. Проекты могут быть сформулированы в виде конкретных и измеримых результатов, например, "построить дом", "запустить сайт", "получить диплом", или в виде более абстрактных и неопределенных целей, например, "улучшить здоровье", "развить навыки", "помочь сообществу".

Управление проектами помогает людям структурировать жизнь и работу, определять приоритеты, отслеживать прогресс, сотрудничать с другими людьми, анализировать результаты и вносить коррективы при необходимости.

Оно позволяет:

* увидеть полную картину своих целей и планов, а также их взаимосвязь и зависимость;
* разбить большие и сложные цели и проекты на более мелкие и управляемые части;
* распределить свое время, энергию и ресурсы оптимальным и эффективным образом;
* сфокусироваться на самых важных и срочных целях, избегая отвлечений и ненужных действий;
* следить за выполнением, контролировать сроки, качество и другие критерии;
* своевременно обнаруживать и решать проблемы, возникающие в ходе реализации;
* оценивать результаты, измерять их соответствие целям и ожиданиям, анализировать ошибки и успехи, делать выводы и уроки для будущих проектов.

Также система решает такие проблемы, как хаос, неопределенность, прокрастинация, недостаток мотивации, низкая эффективность, перегрузка информацией и стресс. Оно позволяет:

* привести в порядок свои мысли, идеи, желания и обязанности, выстроить их в логичную и последовательную систему;
* установить четкие и реалистичные цели, определить конкретные шаги и действия для их достижения, а также способы измерения и проверки их выполнения;
* преодолеть лень, страх, сомнения и другие психологические барьеры, которые мешают начать или продолжить работу;
* найти внутреннюю и внешнюю мотивацию для выполнения, видеть смысл и ценность своей деятельности, получать удовлетворение и признание от своих достижений;
* повысить производительность, качество и скорость работы, использовать свой потенциал на полную мощность, достигать лучших результатов за меньшее время и с меньшими затратами;
* снизить уровень информационного шума, отсеять ненужную и недостоверную информацию, сосредоточиться на той информации, которая необходима и полезна;
* уменьшить нагрузку на свой мозг, память и внимание, освободиться от лишних мыслей и забот, снять напряжение и давление, почувствовать спокойствие и уверенность.

Предметная область требует использования различных инструментов, методик и инструментов, которые могут облегчить и ускорить этот процесс. Существует множество подходов и техник управления, например метод SMART, матрица Эйзенхауэра, методологии Agile, Scrum, Kanban и т. д. Каждый из них имеет свои преимущества и недостатки, свои особенности и применимость в разных ситуациях. Однако, независимо от выбранного подхода или техники, необходимо иметь подходящий инструмент для его реализации.

## Сравнение с существующими аналогами

Сфера планирования является важным аспектом в жизни любого человека. Существует множество сервисов, которые предоставляют возможность управлять задачами и проектами. Ниже представлено сравнение существующих аналогов с функционалом uptask.

1) Trello - сервис от компании Atlassian, позволяющий взаимодействовать с элементами в форме досок канбан:

- Trello предоставляет интуитивный интерфейс и простую организацию в виде карточек, которые можно перемещать между списками.

- В Trello можно добавлять комментарии, прикреплять файлы и устанавливать сроки выполнения.

- Trello также поддерживает функцию уведомлений, что позволяет быть в курсе изменений.

- Одним из преимуществ Trello является возможность создания командных досок, где несколько человек могут работать над одним проектом.

- Trello также поддерживает Android и iOS.

- Существует возможность интеграции Trello с другими сервисами, такими как Slack, Google Drive и GitHub.

2) Asana - сервис, который предоставляет широкий спектр функционала:

- Программа позволяет создавать задачи, проекты и команды для совместной работы.

- Asana также поддерживает функцию установления сроков выполнения.

- В Asana можно создавать шаблоны для повторного использования.

- Продукт предоставляет возможность создания отчетов и диаграмм для анализа производительности.

3) Todoist - сервис, который предоставляет простой и интуитивный интерфейс:

- В Todoist можно создавать проекты и подпроекты для организации.

- ПО поддерживает функцию уведомлений и напоминаний.

- Todoist имеет сервисы для удобного доступа с мобильных устройств.

4) Microsoft To Do - сервис от компании Microsoft, который отличается интеграцией и простотой.

- To Do отличается своей не загруженностью, что делает его привлекательным для тех, которым нужен простой инструмент.

- Также он позволяет устанавливать сроки выполнения и прикреплять файлы.

- To Do поддерживает функцию уведомлений и напоминаний о сроках.

- To Do интегрируется с другими сервисами Microsoft, такими как Outlook и Teams.

- Позволяет создавать списки и делиться ими с другими, а также настраивать их внешний вид.

Из всего вышеперечисленного можно сделать вывод, что сервис надлежащего уровня должен сочетать в себе:

- Простой и интуитивный интерфейс для удобства использования.

- Функцию уведомлений и напоминаний.

- Мобильные приложения для быстрого доступа где угодно.

- Обширные возможности для интеграции с другими сервисами.

uptask, тем не менее, идет своим путем, используя уникальный подход.

### Интерфейс.

В отличие от конкурентов, uptask полностью полагается на дизайн-систему Material Design 3, что позволяет сохранить единый стиль с Android. Также, MD 3 предоставляет широкий набор готовых компонентов, что упрощает разработку и улучшает опыт, так как пользователи уже знакомы с этими компонентами.

### Организация и удобство.

uptask предлагает простую организацию дел - проекты (или же списки), в которых можно создавать задачи. Это позволяет легко группировать их по проектам и удобно ориентироваться в них. А также, каждый список можно дополнительно отсортировать или отфильтровать по желаемому критерию (например, по дате выполнения).

Для быстрой идентификации списка используется эмодзи, который с ним ассоциируется (на выбор).

### Уведомления.

uptask разрабатывается в соответствии с принципами других продуктов из той же категории, соответственно и уведомления будут работать также, как и в другом ПО. Пользователь получит уведомление о сроке выполнения, а также доступна возможность установить отдельное напоминание (за какое-то время или по локации), если это необходимо.

### Интеграция.

Сервис uptask не предполагает интеграции с другими сервисами, так как разрабатывается как самостоятельное ПО с упором на локальность. Однако, в будущем возможно добавление функционала для экспорта и импорта, чтобы пользователи могли обмениваться списками между собой, или же интеграции с календарем для синхронизации событий.

Далее приведена сравнительная таблица аналогов и разрабатываемого приложения.

Таблица 1 – Сравнение приложения с аналогами

|  |  |  |  |
| --- | --- | --- | --- |
| Критерий | Microsoft To Do | Google Tasks | Uptask |
| Автономность | Требуется постоянное подключение к интернету | Требуется постоянное подключение к интернету | Возможно полноценное использование в оффлайн-режиме |
| Стоимость | Полностью бесплатно | Полностью бесплатно | Полностью бесплатно |
| Открытость и настройки | Закрытый исходный код, настройка внешнего вида списков | Закрытый исходный код, отсутствие любых настроек | Открытый исходный код, бесконечная возможность настройки за счет тем и эмодзи |
| Функционал задач | Повторение, заметки, напоминания, флаги (приоритеты), вложения | Повторение, простые заметки, напоминания без дополнительных настроек | Напоминания с расширенными настройками, простые заметки, приоритеты, теги |
| Производительность | Средняя производительность и скорость добавления задач | Высокая производительность за счет минимализма, моментальное добавление | Низкая производительность, комплексная структура задач |
| Интуитивность и удобство | Построенный по стандартам Microsoft Fluent Design понятный и интуитивный интерфейс | Построенный по стандартам Material Design 2 простой и понятный интерфейс | Построенный по стандартам Material Design 3 адаптивный интерфейс с подсказками |

## Постановка задачи

Целью данной работы является разработка Uptask на платформе Android, которое позволит эффективно управлять своими делами. Система должна обладать следующими основными функциями:

- Создание и редактирование задач и проектов

- Назначение сроков, приоритетов и тегов

- Сортировка, фильтрация и поиск по различным критериям

- Просмотр статистики и прогресса

- Напоминания о сроках выполнения

Исходные данные состоят из следующих элементов:

- Задача — это конкретное действие или цель, которую человек хочет выполнить или достичь. Задача может иметь название, описание, срок, приоритет, теги и состояние (выполнена или нет).

- Проект — это группа дел, объединенных общей темой или целью. Проект может иметь название, описание, цвет и теги.

- Тег — это атрибут, который можно присвоить для более точной классификации или фильтрации. Тег представляет собой короткое слово или фразу, например #важно или #работа.

Выходные данные состоят из следующих элементов:

- Экраны — это интерфейсные элементы, которые отображают содержимое и позволяют клиенту взаимодействовать с ним. В Uptask предполагается наличие следующих экранов:

- Экран приветствия - отображает логотип и кнопку для начала работы

- Экран главный - отображает список всех проектов с возможностью переключаться между разными видами (сегодня, завтра, неделя, месяц, все), создавать новые или изменять существующие

- Карточка дела - отображает подробную информацию о выбранном элементе с возможностью редактировать ее атрибуты, отмечать, как выполненную или нет, а также удалять

- Экран проекта - отображает список всех элементов в выбранном проекте с возможностью создавать новые, сортировать, фильтровать и искать по разным критериям, а также редактировать или удалять проект

- Экран статистики - отображает текстовую информацию о прогрессе за определенный период времени (день, неделя, месяц, год).

- Данные — это информация, которая хранится в приложении и обрабатывается его логикой. В Uptask они представляют собой коллекцию задач, проектов и тегов, которые создаются, изменяются или удаляются. Информация хранится в локальной БД на устройстве.

Минимальные требования для Uptask определяются следующими параметрами:

- ОС - Android 5.0 (API 21) или выше

- Процессор - 1 ГГц или выше

- Оперативная память - 512 МБ или выше

- Внутренняя память - 70 МБ или выше

- Разрешение экрана - 320x480 пикселей или выше

- Подключение к интернету - не требуется

Для того, чтобы продукт мог работать с базой, предполагается использовать библиотеку JetBrains Exposed. В качестве фреймворка для интерфейса выбран Jetpack Compose, который поддерживается и развивается компанией Google как основная платформа для Android.

## Обоснование выбора методики, технологии и инструментальных средств проектирования и разработки

Для разработки под Android необходимо выбрать подходящие технологии и среды программирования, которые позволят реализовать требуемый функционал, обеспечить производительность, безопасность, удобство использования и масштабируемость. В данном разделе будут рассмотрены основные технологии и среды программирования, используемые для разработки, а также обоснован выбор каждой из них.

JetBrains Exposed - это легковесная библиотека для работы с таблицами на языке Kotlin. Она позволяет использовать SQL-выражения в виде типобезопасных и читаемых DSL (Domain Specific Language), а также предоставляет DAO (Data Access Object) слой для работы с объектами. Библиотека поддерживает различные СУБД, включая SQLite, MySQL, PostgreSQL и H2. JetBrains Exposed была выбрана в качестве инструмента для БД по следующим причинам:

- Она написана на Kotlin и хорошо интегрируется с другими библиотеками и средами программирования.

- Она имеет минимальный размер и не накладывает больших нагрузок на процессор и память устройства.

- Она позволяет работать с информацией как с объектами, так и с SQL-выражениями, что дает гибкость и удобство для разработчика.

- Она поддерживает H2 – движок таблиц, который был выбран для хранения локальной информации.

H2 - это открытый источник реляционной СУБД, написанный на Java. Он может работать как во встроенном режиме, так и в режиме сервера. H2 имеет высокую производительность, малый размер, поддерживает SQL-стандарт и шифрование. H2 был выбран в качестве движка по следующим причинам:

- легко интегрируется с JetBrains Exposed и позволяет использовать одинаковый синтаксис и API для работы с локальным и удаленным сервером.

- имеет малый размер (около 2 МБ) и не занимает много места на устройстве.

- поддерживает шифрование, что повышает безопасность и конфиденциальность информации.

- позволяет быстро и легко резервировать, и восстанавливать информацию, а также переносить их между различными устройствами и платформами.

Android - это операционная система для мобильных устройств, разработанная компанией Google. Она использует ядро Linux и предоставляет набор библиотек и инструментов для создания приложений, которые могут работать на различных устройствах с разными характеристиками и возможностями. Android был выбран в качестве платформы по следующим причинам:

- является самой популярной и распространенной операционной системой для мобильных устройств, которая охватывает более 70% рынка по данным на 2021 год.

- имеет открытый исходный код и поддерживается большим сообществом разработчиков, что обеспечивает доступность и актуальность документации, библиотек, инструментов и решений для различных проблем.

- предоставляет различные сервисы и ресурсы для улучшения качества и безопасности, такие как Google Play Services, Firebase, SafetyNet и т.д.

Material Design 3 - это система дизайна, разработанная компанией Google для создания единых и привлекательных интерфейсов для Android и других платформ. Она основана на принципах адаптивности, доступности, экспрессии и движения. Material Design 3 включает в себя набор гайдлайнов, компонентов, иконок, цветов, шрифтов и анимаций, которые помогают создавать красивые и функциональные интерфейсы. Material Design 3 был выбран в качестве дизайн-опоры по следующим причинам:

- обеспечивает единообразие и согласованность интерфейсов между различными устройствами и платформами, что повышает узнаваемость и доверие.

- поддерживает динамические темы, которые позволяют подстраивать цвета и стили интерфейса под предпочтения и условия потребителя, такие как темный режим, цветовая схема, освещение и т. д.

- имеет современный и эстетический вид, который привлекает внимание и интерес, а также отражает цели и характер.

- использует движение и анимацию, чтобы делать интерфейсы более живыми и интерактивными, а также подчеркивать связи и переходы между элементами.

Kotlin - это язык программирования, разработанный компанией JetBrains для создания систем на платформе JVM (Java Virtual Machine), Android и других. Он объединяет в себе преимущества функционального и объектно-ориентированного программирования, а также поддерживает многопарадигменность и мультиплатформенность. Kotlin был выбран в качестве единственного языка программирования для логики по следующим причинам:

- полностью совместим с Java и может использовать все существующие библиотеки и инструменты, разработанные для этого языка, а также легко переносить код между ними.

- имеет современный и лаконичный синтаксис, который упрощает чтение и написание кода, а также избавляет от лишних символов и шаблонов.

- имеет множество полезных функций и возможностей, таких как null-безопасность, расширения, лямбды, корутины, делегирование, дата-классы, смарт-касты и т. д., которые повышают продуктивность и качество кода.

- поддерживается и развивается компанией Google как официальный язык программирования для Android, что гарантирует его актуальность и поддержку.

Jetpack Compose - это фреймворк для создания интерфейсов на языке Kotlin для Android. Он использует декларативный подход, который позволяет описывать интерфейсы с помощью композабельных (составных) функций, а не с помощью XML-разметки. Jetpack Compose был выбран в качестве единственного инструмента для интерфейса по следующим причинам:

- упрощает процесс разработки интерфейсов, так как не требует отдельной разметки, синхронизации состояния, привязки данных и обработки жизненного цикла.

- позволяет создавать интерфейсы, которые автоматически реагируют на изменения событий и окружения, а также поддерживают различные темы, размеры экранов и ориентации.

- интегрируется с Material Design 3 и предоставляет набор готовых компонентов, которые соответствуют гайдлайнам и стандартам дизайна.

- поддерживается и развивается компанией Google как основная платформа для создания интерфейсов для Android, что гарантирует его актуальность и поддержку.

Android Studio — это официальная интегрированная среда разработки (IDE) для создания под Android, разработанная компанией Google. Она предлагает широкий спектр инструментов и функций для разработки, тестирования и отладки приложений, а также для управления проектами и ресурсами. Выбор Android Studio в качестве среды разработки обоснован следующими преимуществами:

- является официальной и рекомендуемой средой разработки для Android, что гарантирует ее актуальность, поддержку и совместимость с последними версиями платформы и библиотек.

- предлагает удобный и интуитивно понятный интерфейс, который позволяет быстро и легко создавать, редактировать и запускать ПО, а также настраивать их параметры и опции.

- включает в себя мощные инструменты для тестирования и отладки приложений, такие как эмуляторы, профилировщики, логгеры и дебаггеры, которые помогают обнаруживать и исправлять ошибки и проблемы в коде.

- поддерживает различные плагины и расширения, которые позволяют добавлять новые функции и возможности, а также интегрироваться с другими сервисами и инструментами, такими как VCS (Version Control System), облачные хранилища и CI/CD платформы.

- предоставляет доступ к обширной документации, примерам кода, учебным материалам и сообществу разработчиков, которые помогают учиться, решать проблемы и обмениваться опытом.

# Проектирование и реализация ПО

## Разработка модели БД

Разработка модели базы данных (БД) — это важный этап проектирования и реализации программного обеспечения, который определяет структуру, связи и ограничения данных, с которыми работает приложение. Модель БД представляет собой абстрактное описание того, как данные хранятся, обрабатываются и извлекаются из базы.

Существует множество типов баз данных, таких как иерархические, сетевые, объектно-ориентированные и документо-ориентированные, но наиболее распространенным является реляционная модель БД, которая использует табличный формат для представления данных. Реляционная БД состоит из одной или нескольких таблиц, каждая из которых имеет уникальное имя и содержит строки (записи) и столбцы (поля). Каждая запись представляет собой объект или сущность, а каждое поле представляет собой атрибут или свойство этого объекта.

В реляционной БД могут существовать связи между разными таблицами, которые позволяют соединять данные из разных источников и выполнять сложные запросы. Связи устанавливаются посредством ключей — специальных полей, которые идентифицируют каждую запись в таблице. Ключи могут быть первичными или внешними. Первичный ключ — это поле или набор полей, которые гарантируют уникальность каждой записи в таблице. Внешний ключ — это поле или набор полей, которые ссылаются на первичный ключ другой таблицы, создавая связь между ними.

Кроме того, в реляционной БД могут быть определены ограничения, которые задают правила и условия для данных, хранящихся в таблицах. Ограничения могут быть различными, например, проверка типа и диапазона значений, обязательность заполнения поля, уникальность значения и т. д. Ограничения помогают поддерживать целостность и надежность данных в БД.

В таблице 2 описывается информация о пользователях.

Таблица 2 – Описание логической структуры таблицы “Users”

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Ограничения |
| id | Int | Primary key, auto-increment |
| login | Varchar(50) | Unique, not null |
| password | Varchar(50) | Not null |

В таблице 3 представлена информация о задачах, c которыми связаны пользователи.

Таблица 3 – Описание логической структуры таблицы “UserTasks”

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Ограничения |
| id | Int | Primary key, auto-increment |
| userId | Int | Foreign key references Users(id), cascade on delete, not null |
| taskListId | Int | Foreign key references TaskLists(id), cascade on delete, not null |
| task | Varchar(50) | Not null |
| description | Varchar(450) | Nullable |
| dueDate | Datetime | Nullable |
| isDone | Bool | Not null |
| priority | Int | Nullable |
| attachment | Varchar(150) | Nullable |

В таблице 4 перечисляются теги, присвоенные делам.

Таблица 4 – Описание логической структуры таблицы “TaskTags”

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Ограничения |
| id | Int | Primary key, auto-increment |
| taskId | Int | Foreign key references UserTasks(id), cascade on delete, nullable |
| tag | Varchar(50) | Not null |

Таблица 5 представляет собой описание проектов, и пользователей, связанных с ними.

Таблица 5 – Описание логической структуры таблицы “TaskLists”

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Ограничения |
| id | Int | Primary key, auto-increment |
| userId | Int | Foreign key references Users(id>, cascade on delete, not null |
| name | Varchar(50) | Not null |
| emoji | Varchar(50) | Not null |

Листинг созданной схемы можно увидеть в Приложении Б.

Далее представлена схема связей между таблицами и их структура.

A diagram of a task

Description automatically generated

Рисунок 1 – Схема базы данных

## Разработка пользовательского интерфейса

Разработка UI/UX включает в себя несколько этапов:

- Обозначение основных экранов;

- Разработка дизайн-макета на платформе Figma;

- Реализация интерфейса на нужной платформе с использованием выбранных технологий.

Для системы были выбраны следующие экраны: регистрация, авторизация, проекты (списки), задачи из списка (выбирается пользователем), а также некоторые диалоговые окна, отвечающие за интерактивность (возможность добавить, изменить или удалить элемент).

Так как в качестве дизайн-системы был выбран Material Design 3, то все компоненты и экраны реализованы с помощью встроенных инструментов Android. Стоит отметить, что MD 3 хорошо встраивается в систему, а интерфейс пишется быстро и не отличается визуально от самой ОС. Такой подход не требует дополнительной разработки компонентов и логики для них, все собирается из конструктора. Так можно сконцентрироваться на идее ПО и основной функциональности.

Весь интерфейс приложения полностью локализован на Английский и Русский языки, обеспечивая максимальное покрытие целевой аудитории, при этом не забывая о иностранных клиентах.

Далее представлены скриншоты разработанного интерфейса.

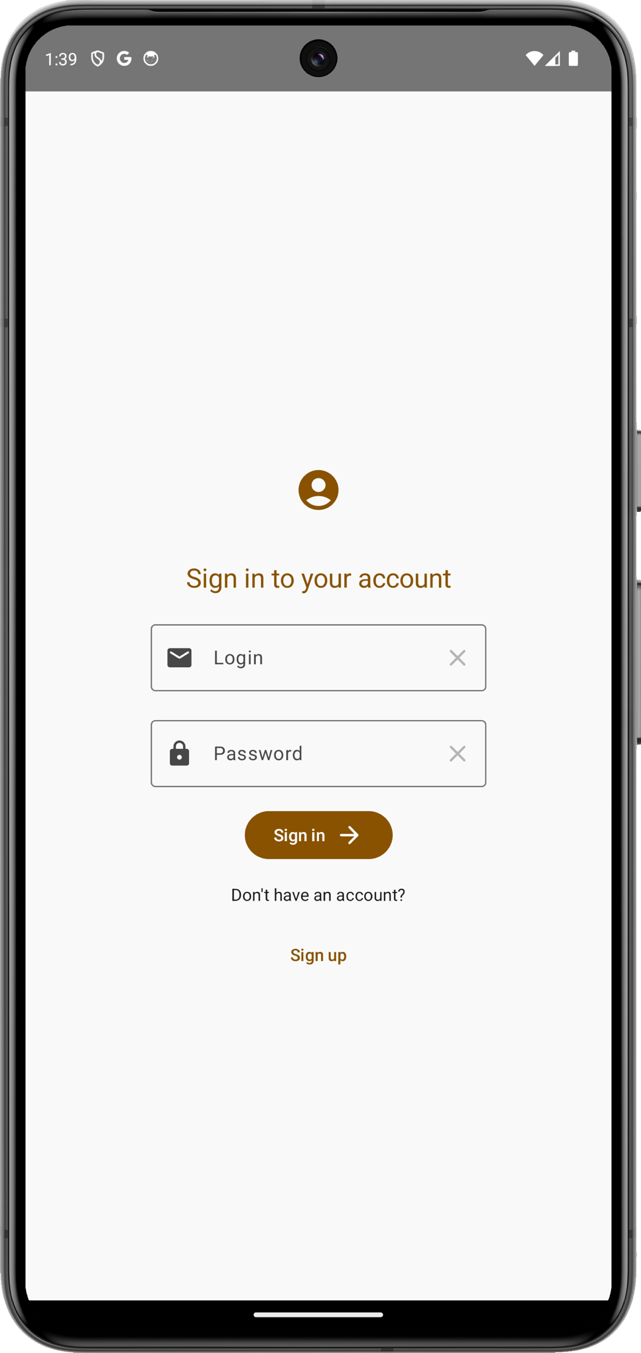


Рисунок 2 – Экран авторизации

Экран регистрации примерно совпадает с авторизацией, поэтому он здесь отдельно не показан. Экраны разработаны в соответствии с инструкциями от Google, не содержат лишнего шума, а также акцентируют внимание на важных действиях, например кнопка входа или выделенный цветом переход к экрану регистрации. Помимо текстового отображения подписи к текстовым полям, также есть иконки и возможность очистить поле, если введено слишком много символов.

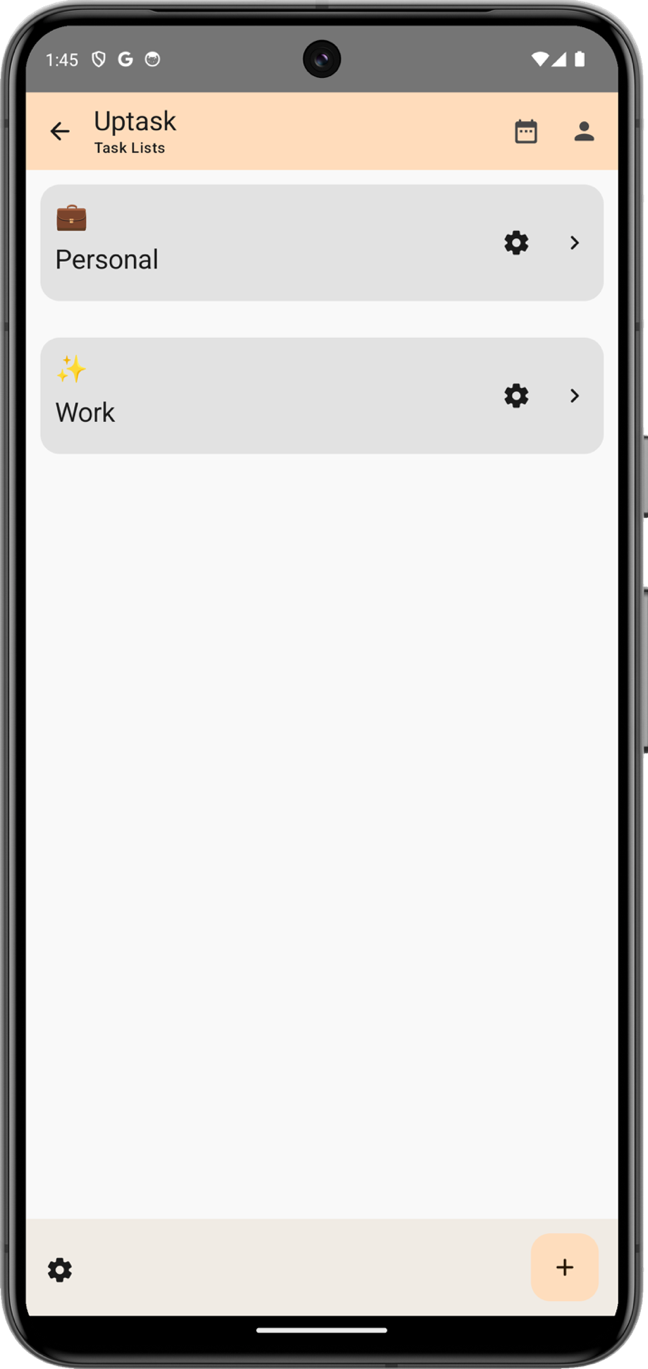


Рисунок 3 – Проекты

Проекты разработаны в соответствии с задуманным опытом пользователя, использование карточек отделяет каждый проект от остальных, а также помещает дополнительные элементы управления в один визуальный контейнер. Идея использования иконки или эмодзи для проекта также вдохновлена визуальной дифференциацией списков, обеспечивая определенный уровень персонализации.

Экран проектов и экран задач совпадает в нижней части, создавая универсальный опыт добавления новых элементов. Карточки также содержат иконку шестеренки, позволяя отредактировать параметры конкретного списка – и это работает также в делах.

Так как статистические данные и профиль не относятся к конкретному списку, они помещены в верхнюю панель действий вместе с заголовком текущего экрана и навигационными кнопками (например, для выхода из профиля или выхода из проекта).

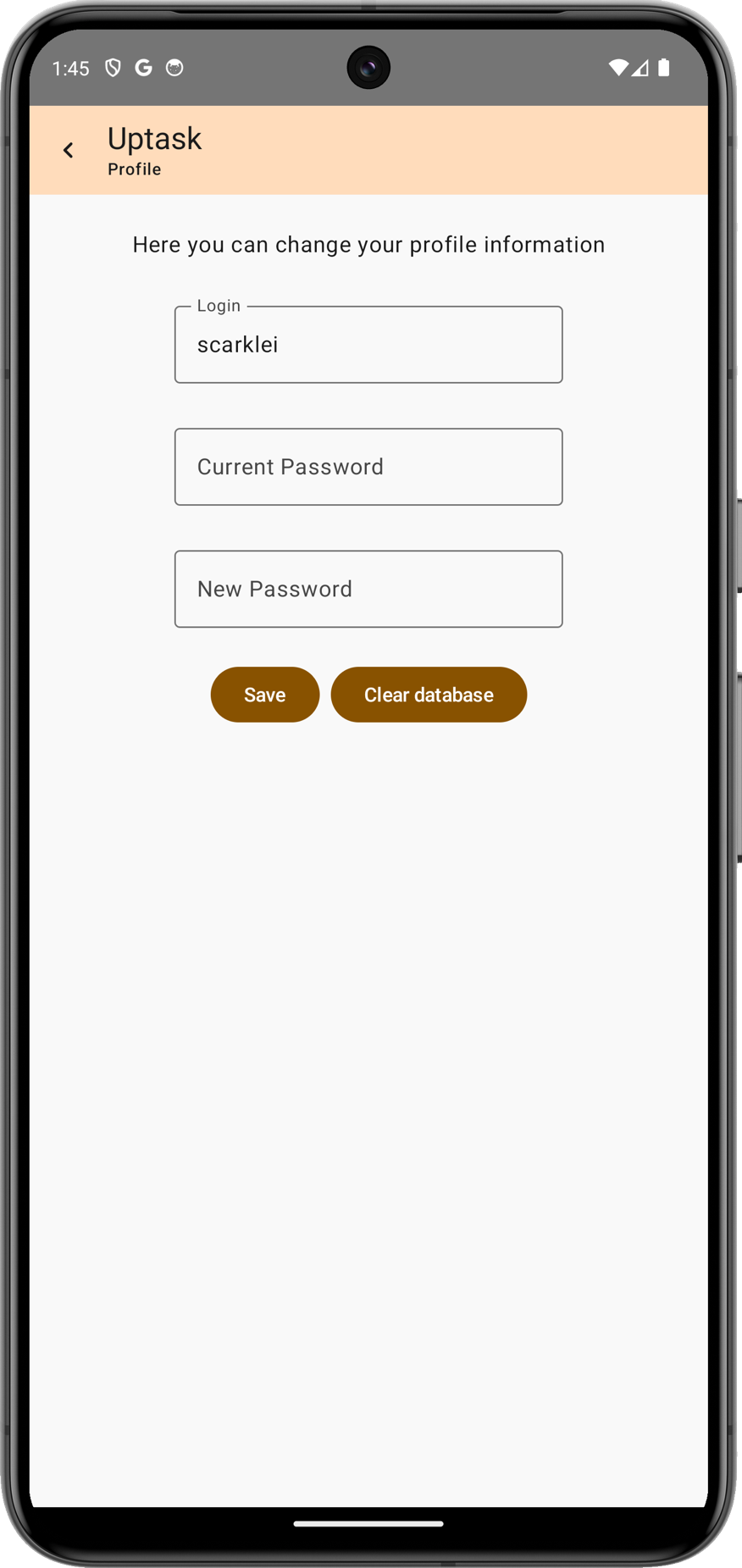


Рисунок 4 – Настройки профиля

Экран деталей профиля позволяет изменить логин или пароль, введя текущий (это делается для проверки, что за устройством настоящий пользователь, а не злоумышленник). Также здесь можно удалить все данные приложения – это удобно делать прямо из приложения и сразу заново регистрироваться, если что-то пошло не так. Экран сопровождается небольшим пояснением и навигацией назад.

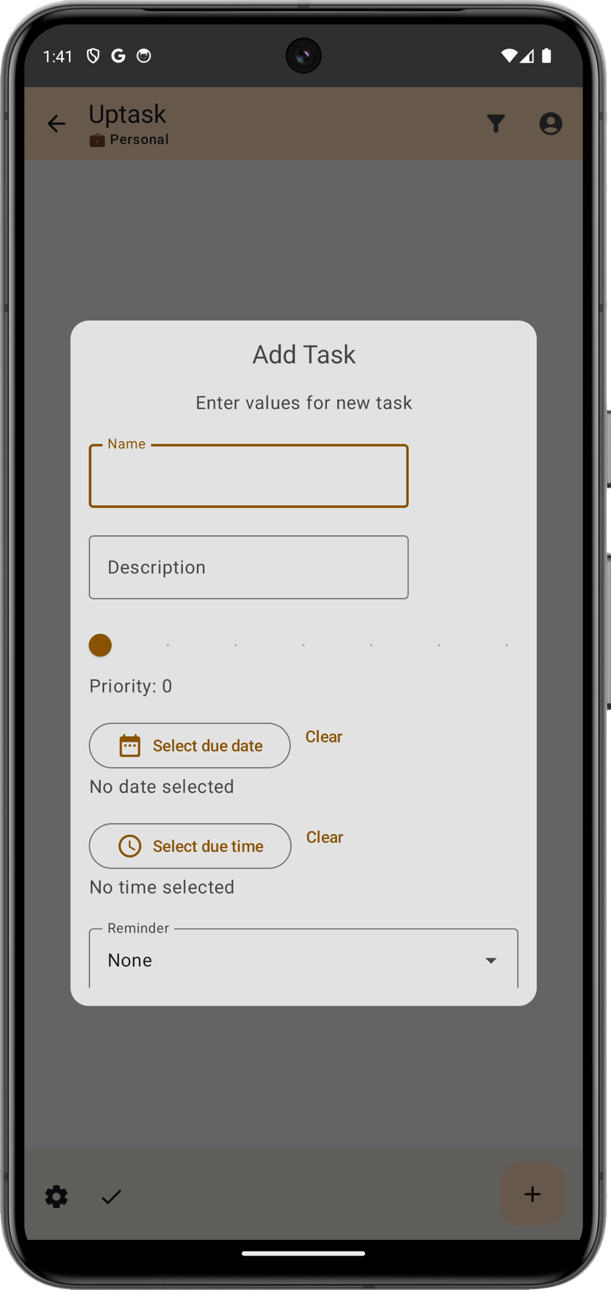


Рисунок 5 – Добавление задачи в проект

Диалоговое окно добавления задачи – самое важное в проекте, здесь происходит вся магия. Оно специально было сделано более высоким по иерархии чем основной экран, так как является полностью интерактивным и отменяемым. Каждое поле для ввода оснащено либо соответствующим элементом управления (например, слайдер приоритета или календарь для выбора даты), либо проверкой вводимых значений (имя, описание). Дату и выбранное время можно очистить и начать заново, а в поле напоминания предусмотрено значение отменяющее напоминание, а ввод в него значения осуществляется через выбор из нескольких вариантов во всплывающем меню. Приоритет используется в отображении карточки, от нуля до 5, где 0 – самый высокий, а 5 – самый низкий. В зависимости от выбранной даты карточка может поменять цвет на красный (если дело просрочено).

В этом же окне можно назначить задаче теги, они отображены в форме расширяющегося списка по мере заполнения новыми элементами, можно выбрать один из существующих в базе или добавить новый через иконку плюса внутри поля – это не захламляет экран и не сбивает пользователя. При этом, ввод значений также защищен: например, ввод одного и того же тега дважды невозможен (прикрепление двух к одной задаче), а если тег уже существует среди каких-либо задач, его нужно выбрать из списка, а не создавать заново.

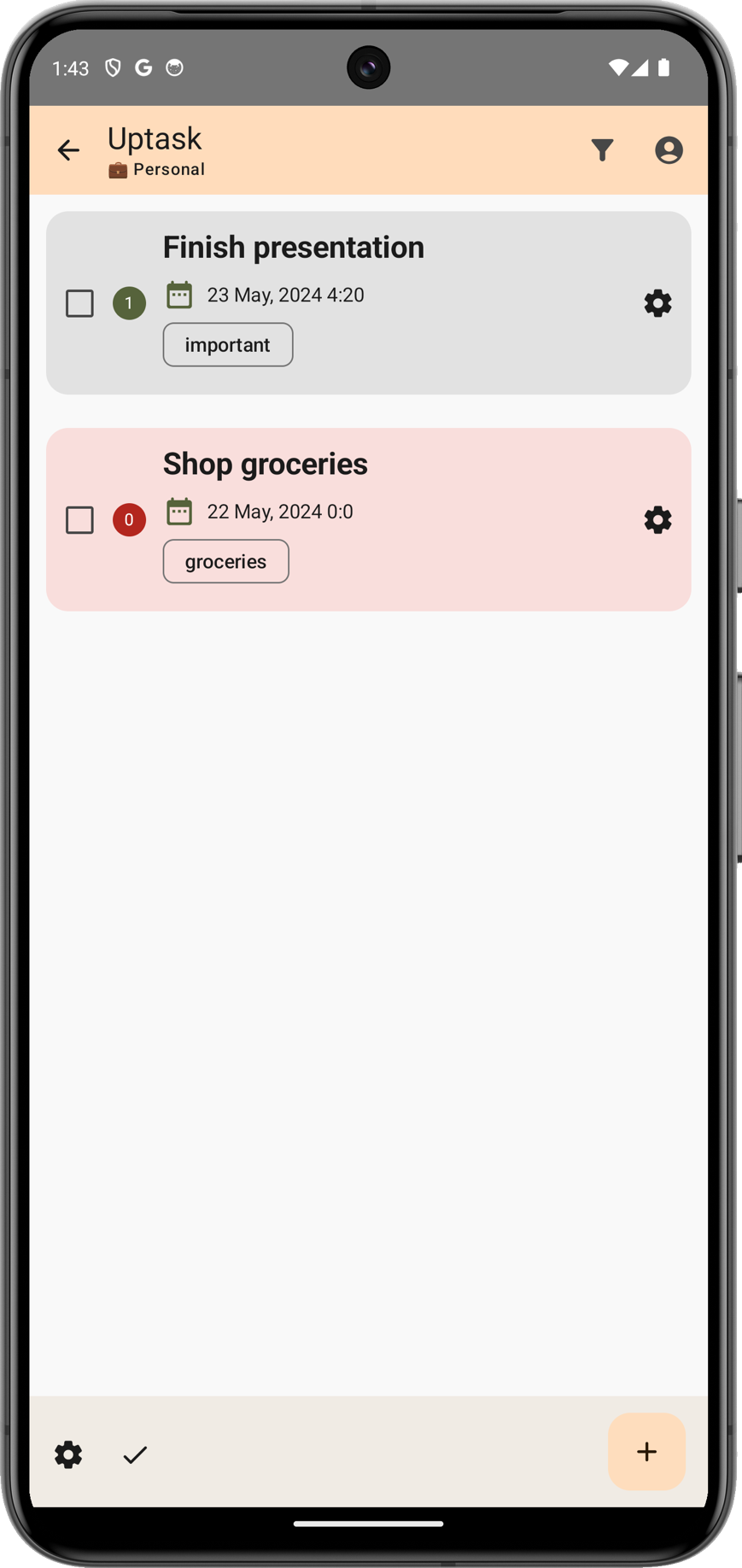


Рисунок 6 – Добавленные задачи

Экран задач обладает такой же универсальностью, как и предыдущие, отображая актуальные данные задачи, вне зависимости от того какие были указаны при создании. Круг приоритета меняет свой цвет от выбранного значения, а дата отображается в удобном для пользователя формате с расчетом на часовые пояса, слева располагается чекбокс изменения статуса задачи, которые после выполнения не исчезают, а перемещаются в раздел выполненных (иконка галочки на нижней панели), откуда их можно при желании удалить, или отменить выполнение (это спасает от ненамеренного нажатия и удаления важных данных).

На верхней панели расположены релевантные для списка действия, а также полезная информация о том, в каком проекте находится пользователь. Среди действий находится диалог фильтров и сортировки, который помогает ориентироваться по большому проекту или искать сразу по нескольким полям.

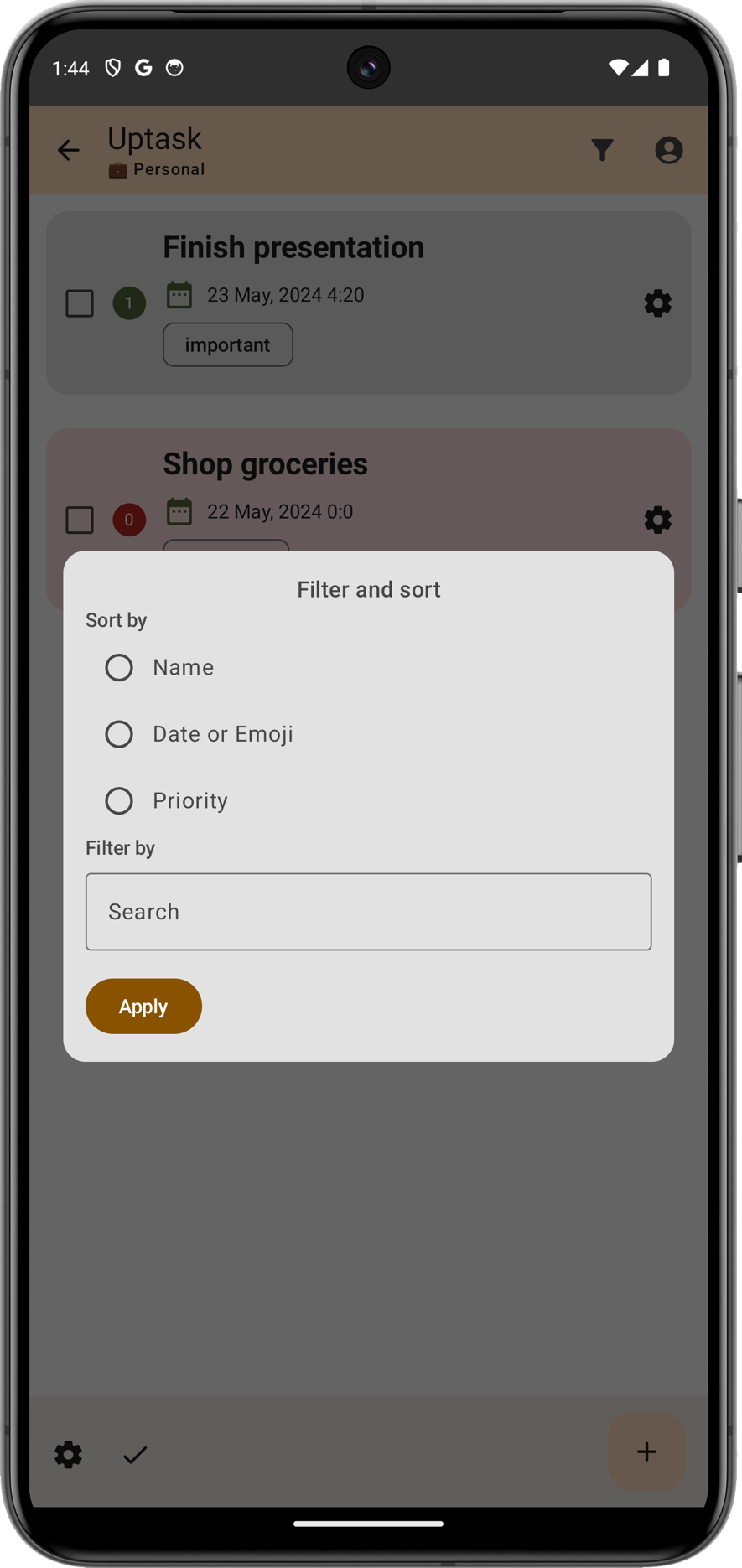


Рисунок 7 – Диалог фильтрации и сортировки

Для приложения были разработаны диаграмма компонентов и User Flow.

A diagram of a company

Description automatically generated

Рисунок 8 – User Flow

A screenshot of a computer

Description automatically generated

Рисунок 9 – Диаграмма компонентов

Диалог фильтров и сортировки является таким же опциональным и отменяемым, обладает небольшими надписями над производимыми действиями. Поле поиска является умным – оно проверяет содержание поискового значения внутри всех задач в списке, и для каждой задачи “смотрит” на несколько полей – название, описание и дату. Сортировка не позволяет выбрать направление самому, но выполняется в зависимости от выбранного значения, например имя сортируется по алфавиту, а приоритет отобразит самый высокий выше остальных, а самая ранняя дата (ближайшая задача) отобразится первее.

## Разработка основных функций программного обеспечения

Для базы данных были разработаны следующие классы и элементы:

Таблица 6 – Классы для работы с БД

|  |  |  |
| --- | --- | --- |
| Название класса | Назначение | Методы или особенности |
| UptaskDb.kt | Модель таблиц базы данных (сущностей), а также объекты-компаньоны для взаимодействия с ними. | connectToDb() сохраняет подключение к базе на протяжении сеанса |
| DatabaseState.kt | Класс, позволяющий в реакцию на изменения в базе обновлять интерфейс |  |

Для работы с логированием создан следующий класс:

Таблица 7 – Классы для работы с логированием

|  |  |  |
| --- | --- | --- |
| Название класса | Назначение | Методы или особенности |
| Logs.kt | Работает с логами, добавляет новые в общий список (с использованием SharedPreferences и простых строк) | statusChangeLog(  userId: Int,  task: UserTask  ) – запись изменения статуса задачи,  addTaskLog(  userId: User,  newTaskId: UserTask  ) – запись добавления задачи,  deleteTaskLog(  userId: Int,  task: UserTask  ) – запись удаления задачи |

Чтобы проверять вводимые значения был создан объект Checks.

Таблица 8 – Классы для проверок

|  |  |  |
| --- | --- | --- |
| Название класса | Назначение | Методы или особенности |
| Checks.kt | Проверка вводимых значений в поля ввода | emptyCheck(  name: String,  context: Context  ) – проверка пустых значений в имени, priorityCheck(priority: Int, context: Context) – проверка соответствия приоритета значению от 0 до 5,  tagCheck(tag: String, context: Context) – проверка пустого тега, tagExistsCheck(tag: String, context: Context) – проверка на то что тег уже существует в базе, tagAddedCheck(tags: MutableList<String>, tag: String, context: Context) – проверка на то что такой же тег уже добавлен к задаче, checkAuth(  login: String,  password: String,  activity: Activity,  context: Context  ) – авторизация, проверка пароля и логина |

Чтобы получать уведомления и создавать напоминания, был создан класс AlarmReceiver.

Таблица 9 – Класс для получения уведомлений

|  |  |  |
| --- | --- | --- |
| Название класса | Назначение | Методы или особенности |
| AlarmReceiver.kt | Получает уведомления, настраивает их и отправляет пользователю | onReceive(context: Context, intent: Intent) – обработка события получения уведомления |

Чтобы взаимодействовать с сущностями, были созданы следующие View model.

Таблица 10 – Классы для моделей

|  |  |  |
| --- | --- | --- |
| Название класса | Назначение | Методы или особенности |
| TagViewModel.kt | Взаимодействие с тегами | newTag(tagName: String, idTask: UserTask) – создание нового тега, removeTag(tagId: Int) – удаление тега |
| TaskViewModel.kt | Взаимодействие с задачами | newTask(  name: String,  desc: String? = "",  dueDate: LocalDateTime?,  priority: Int?,  taskList: TaskList,  userId: User  ) – добавление задачи  , removeTask(taskId: Int) – удаление задачи |
| UserViewModel.kt | Взаимодействие с пользователями | authUser(login: String, password: String, activity: Activity, context: Context) – вход в аккаунт, setCurrentUser(  sharedPref: SharedPreferences,  user: User  ) – запоминание пользователя |

Листинги перечисленных выше классов можно посмотреть в приложении А.

## Тестирование и отладка программного обеспечения

Тестовые случаи представлены в следующих таблицах.

Таблица 11 – Тестовый пример №1

|  |  |
| --- | --- |
| Тестовый пример | 1 |
| Приоритет тестирования | Высокий |
| Заголовок | Регистрация нового пользователя |
| Этапы теста | Запуск приложения и ввод данных в текстовые поля |
| Тестовые данные | Указываются корректные и некорректные данные: Пустые пароли и пароли из пробелов и почта без 'собаки' |
| Ожидаемый результат | Успешная регистрация и добавление в БД |
| Фактический результат | Пользователь зарегистрирован и добавлен в БД |
| Статус | Пройден |

Таблица 12 – Тестовый пример №2

|  |  |
| --- | --- |
| Тестовый пример | 2 |
| Приоритет тестирования | Высокий |
| Заголовок | Авторизация существующего пользователя |
| Этапы теста | Запуск приложения и ввод данных в текстовые поля |
| Тестовые данные | Вводятся те же данные что и в регистрации |
| Ожидаемый результат | Успешная авторизация |
| Фактический результат | Успешная авторизация |
| Статус | Пройден |

Таблица 13 – Тестовый пример №3

|  |  |
| --- | --- |
| Тестовый пример | 3 |
| Приоритет тестирования | Высокий |
| Заголовок | Создание нового проекта |
| Этапы теста | Вход под своим аккаунтом и нажатие на иконку плюса, ввод необходимых данных и подтверждение |
| Тестовые данные | Вводятся пустые и не пустые или слишком длинные данные (>48 символов) |
| Ожидаемый результат | Проект создан и отображается в списке проектов пользователя |
| Фактический результат | Проект создан и отображается в списке проектов пользователя |
| Статус | Пройден |

Таблица 14 – Тестовый пример №4

|  |  |
| --- | --- |
| Тестовый пример | 4 |
| Приоритет тестирования | Средний |
| Заголовок | Установка тегов для задачи |
| Этапы теста | Нажатие на иконку плюса и ввод необходимых тегов, подтверждение через плюс |
| Тестовые данные | Ввводятся пустые или состоящие из пробелов данные |
| Ожидаемый результат | Теги присвоены задаче |
| Фактический результат | Отображается только один тег, хотя добавлено несколько |
| Статус | Не пройден |

Таблица 15 – Тестовый пример №5

|  |  |
| --- | --- |
| Тестовый пример | 5 |
| Приоритет тестирования | Высокий |
| Заголовок | Удаление задачи из проекта |
| Этапы теста | Нажатие на иконку настроек задачи и нажатие на иконку корзины |
| Тестовые данные | Нет |
| Ожидаемый результат | Задача удалена и изменения отображены |
| Фактический результат | Задача удалена и изменения отображены |
| Статус | Пройден |

Таблица 16 – Тестовый пример №6

|  |  |
| --- | --- |
| Тестовый пример | 6 |
| Приоритет тестирования | Высокий |
| Заголовок | Добавление задачи в проект |
| Этапы теста | Зайти в свой аккаунт или создать новый, после чего добавить проект через меню, зайти в проект по нажатию на карточку и нажать на иконку плюса, ввести необходимые поля и подтвердить |
| Тестовые данные | Вводятся пустые и не пустые, или слишком длинные данные |
| Ожидаемый результат | Задача успешно добавлена в проект |
| Фактический результат | Задача успешно добавлена в проект |
| Статус | Пройден |

Таблица 17 – Тестовый пример №7

|  |  |
| --- | --- |
| Тестовый пример | 7 |
| Приоритет тестирования | Высокий |
| Заголовок | Редактирование задачи |
| Этапы теста | Нажатие на иконку настроек задачи, изменение необходимых полей и подтверждение |
| Тестовые данные | Вводятся те же данные, что и при добавлении задачи |
| Ожидаемый результат | Задача отредактирована и сразу отображена |
| Фактический результат | Изменения сохранены, но отображены только при перезапуске страницы |
| Статус | Не пройден |

Таблица 18 – Тестовый пример №8

|  |  |
| --- | --- |
| Тестовый пример | 8 |
| Приоритет тестирования | Высокий |
| Заголовок | Отображение списка проектов |
| Этапы теста | Зайти в свой аккаунт или создать новый, после чего добавить проект через меню |
| Тестовые данные | Нет |
| Ожидаемый результат | Все проекты пользователя отображены |
| Фактический результат | Все проекты пользователя отображены |
| Статус | Пройден |

Таблица 19 – Тестовый пример №9

|  |  |
| --- | --- |
| Тестовый пример | 9 |
| Приоритет тестирования | Высокий |
| Заголовок | Изменение статуса задачи |
| Этапы теста | Нажатие на чекбокс карточки |
| Тестовые данные | Нет |
| Ожидаемый результат | Задача меняет статус и исчезает |
| Фактический результат | Задача меняет статус и исчезает |
| Статус | Пройден |

Таблица 20 – Тестовый пример №10

|  |  |
| --- | --- |
| Тестовый пример | 10 |
| Приоритет тестирования | Средний |
| Заголовок | Проверка отображения статистики |
| Этапы теста | Запустить модуль Аналитика, выбрать временной период, нажать Применить |
| Тестовые данные | Период с 01.01.2024 по 31.03.2024 |
| Ожидаемый результат | Отображается статистика выполненных и невыполненных задач за январь 2024 года |
| Фактический результат | Отображается статистика выполненных и невыполненных задач за январь 2024 года |
| Статус | Пройден |

Таблица 21 – Тестовый пример №11

|  |  |
| --- | --- |
| Тестовый пример | 11 |
| Приоритет тестирования | Средний |
| Заголовок | Проверка изменения логина пользователя |
| Этапы теста | Войти в профиль, открыть настройки профиля, изменить логин, сохранить изменения |
| Тестовые данные | Новый логин: user1234 |
| Ожидаемый результат | Логин изменяется, доступ к аккаунту возможен с новым логином |
| Фактический результат | Логин изменяется, доступ к аккаунту возможен с новым логином |
| Статус | Пройден |

Таблица 22 – Тестовый пример №12

|  |  |
| --- | --- |
| Тестовый пример | 12 |
| Приоритет тестирования | Низкий |
| Заголовок | Проверка удаления базы данных |
| Этапы теста | Открыть настройки, выбрать опцию удаления базы данных, подтвердить действие |
| Тестовые данные | Нет |
| Ожидаемый результат | База данных удаляется, данные недоступны, а пользователь переходит на экран регистрации |
| Фактический результат | База данных удаляется, данные недоступны, а пользователь переходит на экран регистрации |
| Статус | Пройден |

Таблица 23 – Тестовый пример №13

|  |  |
| --- | --- |
| Тестовый пример | 13 |
| Приоритет тестирования | Низкий |
| Заголовок | Проверка отображения статистики за будущий период |
| Этапы теста | Запустить модуль Аналитика, выбрать будущий временной период, нажать Применить |
| Тестовые данные | Период с 01.01.2025 по 31.01.2025 |
| Ожидаемый результат | Отображается нулевая статистика |
| Фактический результат | Отображается нулевая статистика |
| Статус | Пройден |

## Защита программного продукта

Защита программного продукта обеспечивается с помощью следующих механизмов:

* Авторизация и регистрация пользователей. Для доступа к функционалу программы необходимо ввести логин и пароль, которые проверяются на соответствие базе данных. Если такого пользователя нет в базе, то он может зарегистрироваться, указав свой желаемый логин и пароль. Это позволяет защитить программу от несанкционированного использования.
* Защита от ввода неверных значений в поля. Программа проверяет корректность введенных данных и выводит сообщения об ошибках в случае нарушения правил ввода. Например, нельзя превышать допустимую длину строки или вводить пустые значения. Кроме того, программа проверяет согласованность данных между разными формами и таблицами, например, соответствие идентификаторов задач и тегов. Это позволяет избежать сбоев в работе программы и повысить ее удобство.
* Хранение всех данных только локально. Программа не использует интернет-соединение и не отправляет никаких данных на удаленные серверы. Все данные, включая базу пользователей, хранятся на смартфоне, на котором установлено ПО. Это позволяет обеспечить конфиденциальность данных и исключить возможность их утечки или похищения.
* Моментальный экспорт данных в читаемый формат. Если программа не подошла, но информация о задачах нужна, все дела можно экспортировать в файл .csv формата, cохраняющий описание, теги и прочую информацию, позволяя позже к ним вернуться.

# Сопровождение

## Руководство пользователя

Это руководство предназначено для пользователя, работающего с приложением планировщика задач uptask для Android. В нем описаны основные функции приложения, а также инструкции по установке, настройке и использованию.

### Описание приложения.

Приложение uptask предназначено для планирования и отслеживания задач. Оно позволяет создавать задачи, устанавливать им сроки выполнения, добавлять к ним описание, управлять тегами и просматривать статистику. Приложение также предоставляет возможность создания списков задач, фильтрации и сортировки задач по различным критериям.

### Установка и настройка.

Для установки и настройки приложения uptask на устройстве Android выполните следующие шаги:

1) Скачайте приложение с официального сайта (.apk файл);

2) Откройте файловый менеджер на устройстве и найдите скачанный файл;

3) Нажмите на файл для запуска установки приложения;

4) После установки откройте приложение и выполните первоначальную настройку параметров (создайте локальный аккаунт через раздел Регистрации), далее вы автоматически перейдете к основному разделу приложения.

A screenshot of a phone

Description automatically generated

Рисунок 10 – Начальный экран

На начальном экране можно увидеть, какие функции предоставляет приложение, а также его название и кнопку старта. Этот экран показывается только один раз, при следующих запусках приложение будет переходить на экран авторизации, или, если пользователь уже вошел в аккаунт – на главную страницу со списками задач.

A screenshot of a login form

Description automatically generated

Рисунок 11 – Экран авторизации

На экране авторизации можно перейти на регистрацию или наоборот, войти в существующий аккаунт. Любой неправильный ввод сопровождается подсказками по поводу их исправления, а интерфейс обращает внимание только на самые важные элементы и явно сообщает о текущем экране.

Так как приложение хранит все аккаунты локально, во время первого запуска нужно зарегистрироваться, введя новый логин и пароль, их важно запомнить или записать в менеджере паролей, без них не получить доступ к данным.

A screenshot of a login form

Description automatically generated

Рисунок 12 – Неправильный пароль

A screenshot of a login form

Description automatically generated

Рисунок 13 – Несуществующий логин

A screenshot of a phone

Description automatically generated

Рисунок 14 – Несколько списков задач на основном экране

Для того, чтобы выйти из аккаунта (сбросить запоминание), нужно нажать на навигационную иконку в левом верхнем углу. Также, верхняя панель содержит некоторые дополнительные действия, например статистика: по выбранному периоду дат будут отображаться статистические показатели выполненных задач среди всех созданных, а также количество задач которые были убраны из выполненных.

Переход на экран конкретного списка задач производится нажатием на любое место карточки. Добавление нового списка, как и с задачами, осуществляется через нижнюю панель и главную кнопку (FAB) в диалоговом окне.

A screenshot of a calendar

Description automatically generated

Рисунок 15 – Выбранный период и отображение статистики

A screenshot of a task list

Description automatically generated

Рисунок 16 – Редактирование списка задач

Чтобы отредактировать список, нужно нажать на иконку шестеренки на необходимой карточке. Далее существующие значения подставятся в поля ввода, и при их изменении нужно нажать на Modify или изменить, интерфейс обновится.

Если необходимо удалить какой-либо из списков, это делается в том же меню по иконке корзины. Обратите внимание, что это действие необратимо и вернуть задачи из удаленного списка не получится.

Далее, также на основной панели можно выбрать раздел профиля, где можно изменить данные (логин и пароль) на новые, а также очистить базу данных (это делается проще и быстрее без выхода из приложения) и сразу начать с чистого листа.

Для того, чтобы поменять пароль – необходимо ввести существующий, это обеспечивает уровень безопасности приложения. Значение же логина подставляется автоматически из текущей сессии, но при желании и логин можно изменить.

A screenshot of a login screen

Description automatically generated

Рисунок 17 – Экран профиля

A screenshot of a phone

Description automatically generated

Рисунок 18 – Экран задач

Экран задач же отображает конкретный список, в котором сгруппированы задачи. Каждая карточка – отдельная задача, отображается со всеми необходимыми полями, взаимодействие также происходит через иконку шестеренки, или по нажатию чекбокса (который переводит задачу в раздел выполненных).

A screenshot of a phone

Description automatically generated

Рисунок 19 – Диалог добавления задачи

A screenshot of a phone

Description automatically generated

Рисунок 20 – Диалог сортировки и фильтрации

### Решение проблем.

В случае возникновения проблем с приложением, попробуйте выполнить следующие действия:

1) Перезапустите приложение;

2) Перезагрузите устройство;

3) Обновите приложение до последней версии;

4) Очистите кеш приложения через встроенное меню подробностей приложения;

5) Переустановите приложение, если предыдущий пункт не помог.

Если проблема не устранена, обратитесь в официальному репозиторию и создайте Issue, далее ожидайте ответа от разработчика.

Также для решения проблем можно самостоятельно изучить исходный код приложения, который доступен в официальном репозитории на GitHub, и попытаться найти и исправить ошибку.

### Вывод.

Надеемся, что это руководство поможет вам в работе с приложением планировщика задач uptask для Android. Если у вас возникнут вопросы или проблемы, не стесняйтесь обращаться к разработчику или сообществу пользователей для получения помощи.

ЗАКЛЮЧЕНИЕ

По итогам данного дипломного проекта было разработано мобильное приложение для управления своими задачами и планами. Была спроектирована база данных, а также изучен фреймворк Exposed для взаимодействия с БД. Помимо этого, было проведено ручное тестирование основного функционала, неработающий функционал был отлажен.

Развить приложение в будущем можно следующими способами:

* Создать нативное приложение для платформы iOS и веб-версию для доступа с любого устройства;
* Локализовать на большее количество языков, выпустить в магазин приложений;
* Снабдить приложение полным шифрованием для того, чтобы данные пользователя ни за что не утекли с устройства;
* Интегрировать с системным календарем, позволяя соединить приложение с другими системами продуктивности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Material Design – актуальная дизайн-система Android-приложений, URL: https://m3.material.io (дата обращения 29.05.2024).

2. Jetpack Compose – современное решение для разработки графических интерфейсов в Android, URL: https://developer.android.com/jetpack/compose (дата обращения 29.05.2024).

3. Изучение альтернатив Google Tasks, URL: https://clickup.com/blog/google-tasks-alternatives/ (дата обращения 29.05.2024).

4. Документация по работе с Exposed Framework, URL: https://github.com/JetBrains/Exposed/wiki (дата обращения 29.05.2024).

5. Информация о СУБД H2, URL: http://www.h2database.com/html/main.html (дата обращения 29.05.2024).

6. Соколова, В. В.  Вычислительная техника и информационные технологии. Разработка мобильных приложений : учебное пособие для вузов / В. В. Соколова. — Москва : Издательство Юрайт, 2023. — 160 с. — (Высшее образование). — ISBN 978-5-534-16302-5. — Текст : электронный /\/ Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/530769 (дата обращения: 29.05.2024).

7. Навигация в приложении на Compose, URL: https://developer.android.com/jetpack/compose/navigation (дата обращения 29.05.2024).

8. Логика состояния приложения и объектов, URL: https://developer.android.com/jetpack/compose/state (дата обращения 29.05.2024).

9. Конкурент от Apple - Reminders, URL: https://support.apple.com/en-us/102484 (дата обращения 29.05.2024).

10. Различия между H2 и SQLite, URL: https://stackoverflow.com/questions/43481267/why-use-sqlite-over-h2-for-java-application (дата обращения 29.05.2024).

11. Документация по языку программирования Kotlin, URL: https://kotlinlang.org/docs/home.html (дата обращения 29.05.2024).

12. Задание напоминаний или запланированных уведомлений в Android, URL: https://developer.android.com/develop/background-work/services/alarms/schedule (дата обращения 29.05.2024).

13. Использование биометрической аутентификации в Android, URL: https://developer.android.com/training/sign-in/biometric-auth (дата обращения 29.05.2024).

ПРИЛОЖЕНИЕ А

AlarmReceiver.kt

package org.leftbrained.uptaskapp.classes

import android.app.NotificationManager

import android.content.BroadcastReceiver

import android.content.Context

import android.content.Intent

import androidx.core.app.NotificationCompat

import org.leftbrained.uptaskapp.R

class AlarmReceiver : BroadcastReceiver() {

override fun onReceive(context: Context, intent: Intent) {

val notificationManager =

context.getSystemService(Context.NOTIFICATION\_SERVICE) as NotificationManager

val taskName = intent.getStringExtra("taskName")

val taskDue = intent.getStringExtra("taskDueDate")

val taskDesc = intent.getStringExtra("taskDesc")

val notification = NotificationCompat.Builder(context, "reminders")

.setContentTitle("$taskName is due on $taskDue!")

.setContentText("The task description is: $taskDesc")

.setSmallIcon(R.drawable.icon)

.build()

val notificationId = System.currentTimeMillis().toInt()

notificationManager.notify(

notificationId,

notification

)

}

}

Checks.kt

package org.leftbrained.uptaskapp.classes

import android.app.Activity

import android.content.Context

import android.widget.Toast

import org.jetbrains.exposed.sql.transactions.transaction

import org.leftbrained.uptaskapp.R

import org.leftbrained.uptaskapp.db.Tag

import org.leftbrained.uptaskapp.db.UptaskDb

object Checks {

fun emptyCheck(

name: String,

context: Context

): Boolean {

if (name == "") {

Toast.makeText(

context,

context.getString(R.string.empty\_not\_allowed),

Toast.LENGTH\_SHORT

).show()

return false

}

return true

}

fun priorityCheck(priority: Int, context: Context): Boolean {

if (!priority.toString().matches(Regex("[0-5]"))) {

Toast.makeText(

context,

context.getString(R.string.priority\_between),

Toast.LENGTH\_SHORT

).show()

return false

}

return true

}

fun tagCheck(tag: String, context: Context): Boolean {

if (tag == "") {

Toast.makeText(

context,

context.getString(R.string.tag\_empty),

Toast.LENGTH\_SHORT

).show()

return false

}

return true

}

fun tagExistsCheck(tag: String, context: Context): Boolean {

if (transaction {

Tag.find {

UptaskDb.TaskTags.tag eq tag

}.toList().isNotEmpty()

}) {

Toast.makeText(

context,

context.getString(R.string.tag\_already\_exists),

Toast.LENGTH\_SHORT

).show()

return false

}

return true

}

fun tagAddedCheck(tags: MutableList<String>, tag: String, context: Context): Boolean {

if (tags.contains(tag)) {

Toast.makeText(

context,

context.getString(R.string.tag\_already\_added),

Toast.LENGTH\_SHORT

).show()

return false

}

return true

}

fun checkAuth(

login: String,

password: String,

activity: Activity,

context: Context

): Boolean {

if (login == "" || password == "") {

Toast.makeText(

activity,

context.getString(R.string.please\_fill\_all\_fields),

Toast.LENGTH\_SHORT

).show()

return false

}

return true

}

}

Exporter.kt

package org.leftbrained.uptaskapp.classes

import android.content.Context

import android.widget.Toast

import androidx.compose.runtime.Composable

import org.jetbrains.exposed.sql.SqlExpressionBuilder.eq

import org.jetbrains.exposed.sql.transactions.transaction

import org.leftbrained.uptaskapp.db.Tag

import org.leftbrained.uptaskapp.db.UptaskDb

import org.leftbrained.uptaskapp.db.User

import org.leftbrained.uptaskapp.db.UserTask

class Exporter {

fun dataToMarkdown(userLogin: String): String {

val user = transaction {

User.find {

UptaskDb.Users.login eq userLogin

}

}.toList().first()

val userTasks = transaction {

UserTask.find {

UptaskDb.UserTasks.userId eq user.id

}

}.toList()

val exported: StringBuilder = StringBuilder()

exported.append("TaskList, Task, Description, Tags, Priority, Due Date, Status\n")

userTasks.forEach {

val tags = transaction {

Tag.find {

UptaskDb.TaskTags.taskId eq it.id

}

}.toList()

exported.append(

"${it.taskListId.emoji} ${it.taskListId.name}, ${it.task}, ${it.description}, (${

tags.joinToString(

"; "

) { tag -> tag.tag }

}), ${it.priority}, ${it.dueDate}, ${it.isDone}\n"

)

}

return exported.toString()

}

}

Logs.kt

package org.leftbrained.uptaskapp.classes

import android.content.SharedPreferences

import kotlinx.datetime.Clock

import kotlinx.datetime.LocalDate

import kotlinx.datetime.TimeZone

import kotlinx.datetime.toLocalDateTime

import org.leftbrained.uptaskapp.db.User

import org.leftbrained.uptaskapp.db.UserTask

class Logs(var sharedPref: SharedPreferences) {

private var logsSet = sharedPref.getStringSet("logs", mutableSetOf())!!

fun statusChangeLog(

userId: Int,

task: UserTask

) {

with(sharedPref.edit()) {

val newLogs = logsSet.toMutableSet()

newLogs.add(

"${logsSet.size + 1} - ${

LocalDate.parse(

Clock.System.now()

.toLocalDateTime(TimeZone.currentSystemDefault()).date.toString()

)

} - $userId - ${if (task.isDone) "Done" else "Undone"} - ${task.id.value}"

)

putStringSet("logs", newLogs)

apply()

}

}

fun addTaskLog(

userId: User,

newTaskId: UserTask

) {

with(sharedPref.edit()) {

val newLogs = logsSet.toMutableSet()

newLogs.add(

"${logsSet.size + 1} - ${

LocalDate.parse(

Clock.System.now()

.toLocalDateTime(TimeZone.currentSystemDefault()).date.toString()

)

} - ${userId.id.value} - Add - ${newTaskId.id.value}"

)

putStringSet("logs", newLogs)

apply()

}

}

fun deleteTaskLog(

userId: Int,

task: UserTask

) {

with(sharedPref.edit()) {

val newLogs = logsSet.toMutableSet()

newLogs.add(

"${logsSet.size + 1} - ${

LocalDate.parse(

Clock.System.now()

.toLocalDateTime(TimeZone.currentSystemDefault()).date.toString()

)

} - $userId - Delete - ${task.id.value}"

)

putStringSet("logs", newLogs)

apply()

}

}

}

Stats.kt

package org.leftbrained.uptaskapp.classes

import android.content.SharedPreferences

import kotlinx.datetime.LocalDate

import kotlinx.datetime.LocalDateTime

class Stats(

var userId: Int = 0,

var totalTasks: Int = 0,

var completedTasks: Int = 0,

var undoneTasks: Int = 0

) {

fun checkStats(

sharedPref: SharedPreferences,

startDate: LocalDateTime,

endDate: LocalDateTime,

userId: Int

): Stats {

val logs = sharedPref.getStringSet("logs", mutableSetOf())

val totalTasks = logs?.count {

val date = LocalDate.parse(it.split(" - ")[1])

println("Date: $date")

val action = it.split(" - ")[3]

println("Action: $action")

println("User ID: $userId")

date in startDate.date..endDate.date && action == "Add" && userId == it.split(" - ")[2].toInt()

} ?: 0

println("Total tasks when counted: $totalTasks")

val completedTasks = logs?.count {

val date = LocalDate.parse(it.split(" - ")[1])

val action = it.split(" - ")[3]

date in startDate.date..endDate.date && action == "Done" && userId == it.split(" - ")[2].toInt()

} ?: 0

val undoneTasks = logs?.count {

val date = LocalDate.parse(it.split(" - ")[1])

val action = it.split(" - ")[3]

date in startDate.date..endDate.date && action == "Undone" && userId == it.split(" - ")[2].toInt()

} ?: 0

this.completedTasks = completedTasks

this.totalTasks = totalTasks

this.undoneTasks = undoneTasks

return Stats(userId, totalTasks, completedTasks, undoneTasks)

}

}

TagViewModel.kt

package org.leftbrained.uptaskapp.viewmodel

import androidx.lifecycle.ViewModel

import org.leftbrained.uptaskapp.db.Tag

import org.leftbrained.uptaskapp.db.UserTask

class TagViewModel: ViewModel() {

fun newTag(tagName: String, idTask: UserTask): Tag = Tag.new {

tag = tagName

this.taskId = idTask

}

fun removeTag(tagId: Int) = Tag.findById(tagId)?.delete()

}

TaskViewModel.kt

package org.leftbrained.uptaskapp.viewmodel

import androidx.lifecycle.ViewModel

import kotlinx.datetime.LocalDateTime

import org.jetbrains.exposed.sql.SqlExpressionBuilder.eq

import org.jetbrains.exposed.sql.transactions.transaction

import org.leftbrained.uptaskapp.db.TaskList

import org.leftbrained.uptaskapp.db.UptaskDb

import org.leftbrained.uptaskapp.db.User

import org.leftbrained.uptaskapp.db.UserTask

class TaskViewModel : ViewModel() {

fun getTaskById(taskId: Int): UserTask = transaction {

return@transaction UserTask.find(UptaskDb.UserTasks.id eq taskId).elementAt(0)

}

fun newTask(

name: String,

desc: String? = "",

dueDate: LocalDateTime?,

priority: Int?,

taskList: TaskList,

userId: User

): UserTask = transaction {

UserTask.new {

this.task = name

this.description = desc

this.dueDate = dueDate

this.priority = priority.toString().toInt()

this.taskListId = taskList

this.isDone = false

this.userId = userId

this.attachment = attachment

}

}

fun removeTask(taskId: Int) = transaction {

UserTask.find(UptaskDb.UserTasks.id eq taskId).elementAt(0).delete()

}

}

UserViewModel.kt

package org.leftbrained.uptaskapp.viewmodel

import android.app.Activity

import android.content.Context

import android.content.SharedPreferences

import android.widget.Toast

import androidx.lifecycle.ViewModel

import org.jetbrains.exposed.sql.SqlExpressionBuilder.eq

import org.jetbrains.exposed.sql.transactions.transaction

import org.leftbrained.uptaskapp.R

import org.leftbrained.uptaskapp.db.UptaskDb

import org.leftbrained.uptaskapp.db.User

class UserViewModel : ViewModel() {

fun authUser(login: String, password: String, activity: Activity, context: Context): User? {

val user = transaction { User.find { UptaskDb.Users.login eq login }.firstOrNull() }

if (user == null) {

Toast.makeText(

activity,

context.getString(R.string.user\_with\_this\_login),

Toast.LENGTH\_SHORT

).show()

return null

}

if (user.password != password) {

Toast.makeText(

activity,

context.getString(R.string.incorrect\_password),

Toast.LENGTH\_SHORT

).show()

return null

}

return user

}

fun setCurrentUser(

sharedPref: SharedPreferences,

user: User

) {

transaction {

with(sharedPref.edit()) {

putString("user", user.id.value.toString())

apply()

}

}

}

fun newUser(login: String, password: String): User = transaction {

User.new { this.login = login; this.password = password }

}

}

ПРИЛОЖЕНИЕ Б   
База данных

DatabaseState.kt

package org.leftbrained.uptaskapp.db

import androidx.compose.runtime.getValue

import androidx.compose.runtime.mutableIntStateOf

import androidx.compose.runtime.setValue

import androidx.lifecycle.ViewModel

class DatabaseStateViewmodel: ViewModel() {

var databaseState by mutableIntStateOf(1)

}

UptaskDb.kt

package org.leftbrained.uptaskapp.db

import org.jetbrains.exposed.dao.IntEntity

import org.jetbrains.exposed.dao.IntEntityClass

import org.jetbrains.exposed.dao.id.EntityID

import org.jetbrains.exposed.dao.id.IntIdTable

import org.jetbrains.exposed.sql.Database

import org.jetbrains.exposed.sql.ReferenceOption

import org.jetbrains.exposed.sql.kotlin.datetime.date

import org.jetbrains.exposed.sql.kotlin.datetime.datetime

class UptaskDb {

object Users: IntIdTable() {

val login = varchar("login", 50).uniqueIndex()

val password = varchar("password", 50)

}

object UserTasks: IntIdTable() {

val userId = reference("userId", Users, onDelete = ReferenceOption.CASCADE)

val taskListId = reference("taskListId", TaskLists, onDelete = ReferenceOption.CASCADE)

val task = varchar("task", 50)

val description = varchar("description", 450).nullable()

val dueDate = datetime("dueDate").nullable()

val isDone = bool("isDone")

val priority = integer("priority").nullable()

val attachment = varchar("attachment", 150).nullable()

}

object TaskTags: IntIdTable() {

val taskId = reference("taskId", UserTasks, onDelete = ReferenceOption.CASCADE).nullable()

val tag = varchar("tag", 50)

}

object TaskLists: IntIdTable() {

val userId = reference("userId", Users, onDelete = ReferenceOption.CASCADE)

val name = varchar("name", 50)

val emoji = varchar("emoji", 50)

}

object Logs: IntIdTable() {

val userId = reference("userId", Users)

val date = date("date")

val action = varchar("action", 50)

val taskId = reference("taskId", UserTasks, onDelete = ReferenceOption.CASCADE).nullable()

}

}

class User(id: EntityID<Int>) : IntEntity(id) {

companion object : IntEntityClass<User>(UptaskDb.Users)

var login by UptaskDb.Users.login

var password by UptaskDb.Users.password

}

class UserTask(id: EntityID<Int>) : IntEntity(id) {

companion object : IntEntityClass<UserTask>(UptaskDb.UserTasks)

var userId by User referencedOn UptaskDb.UserTasks.userId

var taskListId by TaskList referencedOn UptaskDb.UserTasks.taskListId

var task by UptaskDb.UserTasks.task

var description by UptaskDb.UserTasks.description

var dueDate by UptaskDb.UserTasks.dueDate

var isDone by UptaskDb.UserTasks.isDone

var priority by UptaskDb.UserTasks.priority

var attachment by UptaskDb.UserTasks.attachment

}

class Tag(id: EntityID<Int>) : IntEntity(id) {

companion object : IntEntityClass<Tag>(UptaskDb.TaskTags)

var taskId by UserTask optionalReferencedOn UptaskDb.TaskTags.taskId

var tag by UptaskDb.TaskTags.tag

}

class TaskList(id: EntityID<Int>) : IntEntity(id) {

companion object : IntEntityClass<TaskList>(UptaskDb.TaskLists)

var userId by User referencedOn UptaskDb.TaskLists.userId

var name by UptaskDb.TaskLists.name

var emoji by UptaskDb.TaskLists.emoji

}

class Log(id: EntityID<Int>) : IntEntity(id) {

companion object : IntEntityClass<Log>(UptaskDb.Logs)

var userId by User referencedOn UptaskDb.Logs.userId

var taskId by UserTask optionalReferencedOn UptaskDb.Logs.taskId

var date by UptaskDb.Logs.date

var action by UptaskDb.Logs.action