МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**МНОГОФУНКЦИОНАЛЬНЫЙ ПЕРСОНАЛЬНЫЙ МЕНЕДЖЕР ЗАДАЧ**

| *Факультет* | *Кафедра* |
|---|---|
| вычислительной техники | «Вычислительная техника» |

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

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

**БАКАЛАВРСКАЯ РАБОТА**

на тему

# МНОГОФУНКЦИОНАЛЬНЫЙ ПЕРСОНАЛЬНЫЙ МЕНЕДЖЕР ЗАДАЧ

**Студент** 09.06.2026 (подпись, дата) Сергунов Максим Романович (ФИО полностью)

**Руководитель** 09.06.2026 (подпись, дата) Токарев А. Н. (фамилия, инициалы)

**Нормоконтролёр** 09.06.2026 (подпись, дата) Бычков А.С. (фамилия, инициалы)

*Работа допущена к защите* (протокол зас. кафедры от 22.05. 2026 г. № 14)

**Заведующий кафедрой** _____ (подпись) М.А. Митрохин (фамилия, инициалы)

*Работа защищена с отметкой* хорошо (протокол заседания ГЭК от 15 июня 2026 № 35)

**Секретарь ГЭК** _____ (подпись) Е.И. Гурин (фамилия, инициалы)

Пенза 2026

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

*Факультет*
вычислительной техники

*Кафедра*
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«07» 04 2026 г.

**ЗАДАНИЕ**
**НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Сергунов Максим Романович гр. 22ВВП1 факультета вычислительной техники

2.Тема работы: «Многофункциональный персональный менеджер задач»

Тема утверждена приказом ПГУ № 34710 от "07" 04 2026 г.

3. Руководитель бакалаврской работы к. т. н., доцент кафедры ВТ Токарев А. Н.

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

Разработать настольное приложение для ведения и контроля задач. Функционал приложения включает в себя: создание, редактирование и удаление задач, распределение задач по проектам, установку приоритетов и сроков выполнения, работу с подзадачами и тегами, систему напоминаний, отображение задач в виде списка и Kanban-доски, а также генерацию QR-кода для задач с установленным сроком выполнения. Приложение должно иметь графический интерфейс и обеспечивать постоянное хранение данных с использованием встроенной базы данных SQLite.

5. Перечень подлежащих разработке вопросов

Требуется провести анализ предметной области: изучить существующие сервисы и приложения для управления задачами, выявить их достоинства и недостатки. Выбрать и обосновать средства разработки, подходящие для создания настольного приложения. Разработать архитектуру приложения, структуру базы данных и пользовательский интерфейс. Провести тестирование реализованного функционала и оформить пояснительную записку.

6. Календарный график выполнения работы

| № п/п | Наименование этапов работы | Объем работы | Срок выполнения | Подпись руководителя |
|---|---|---|---|---|
| 1 | Ознакомление с темой, разработка плана проведения работ | 50 | 07.04.2026 – 12.04.2026 | |
| 2 | Анализ требований | 10 | 12.04.2026 – 13.04.2026 | |
| 3 | Выбор технологий, применяемых при проектировании и разработке | 50 | 13.04.2026 – 20.04.2026 | |
| 4 | Описание алгоритмов реализации | 40 | 20.04.2026 – 23.04.2026 | |
| 5 | Разработка персонального менеджера задач | 150 | 23.04.2026 – 08.05.2026 | |
| 6 | Разработка графической части приложения | 24 | 08.05.2026– 21.05.2026 | |

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы _____ Токарев А. Н.
*(подпись, дата)*

Задание к исполнению принял студент _____ Сергунов М. Р.
*(подпись, дата)*

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

**Реферат**

Работа содержит 140 листов., 28 рисунков, 9 таблиц, 23 источников, и 3 приложения.

МНОГОФУНКЦИОНАЛЬНЫЙ ПЕРСОНАЛЬНЫЙ МЕНЕДЖЕР ЗАДАЧ, C++, QT, QML, SQLITE, ПРИЛОЖЕНИЕ, УПРАВЛЕНИЕ ЗАДАЧАМИ, KANBAN-ДОСКА

Цель работы – разработать многофункциональное настольное приложение для ведения и контроля задач с локальным хранением данных.

Разработанное приложение позволяет пользователю создавать, редактировать и удалять задачи, распределять их по проектам, задавать приоритеты и сроки выполнения, работать с подзадачами и тегами, устанавливать напоминания, просматривать задачи в виде списка и Kanban-доски, а также формировать QR-код для задач с установленным сроком выполнения.

Результатом работы является приложение, обеспечивающее удобное планирование и контроль выполнения задач без необходимости регистрации, подписки и постоянного подключения к сети Интернет. Проведённое тестирование подтвердило корректность работы основных функций приложения и соответствие разработанного программного продукта поставленным требованиям.

# Содержание

Перечень сокращений и обозначений ..... 5
Введение ..... 6
1 Многофункциональны персональный менеджер задач ..... 7
1.1 Анализ предметной области ..... 7
1.1.1 Описание потребности потребителя ..... 8
1.2 Анализ существующих аналогов ..... 9
1.2.1 Сервис “Todoist” ..... 9
1.2.2 Сервис “Google Tasks” ..... 11
1.2.3 Сервис “Trello” ..... 12
1.3 Постановка задачи ..... 14
1.4 Выбор инструментальных средств ..... 17
1.4.1 Выбор языка программирования ..... 17
1.4.2 Выбор фреймворка для разработки приложения ..... 18
1.4.3 Выбор СУБД ..... 19
1.4.4 Выбор среды обработки ..... 20
1.4.5 Выбор технологии создания пользовательского интерфейса ..... 21
1.5 Архитектура приложения ..... 21
1.6 Реализация ..... 24
1.6.1 Описание работы приложения ..... 25
1.6.2 Структуры базы данных ..... 26
1.6.3 Описание компонентов ..... 28
1.6.4 Основные алгоритмы работы приложения ..... 38
1.7 Тестирование ..... 42
1.7.1 Добавление задачи ..... 42
1.7.2 Изменение статуса задачи ..... 43
1.7.3 Просмотр и выбор задачи ..... 43
1.7.4 Изменение статуса приоритета задачи ..... 44
1.7.5 Создание проекта ..... 45
1.7.5 Установка дедлайна задачи ..... 46
1.7.6 Установка напоминания ..... 48
1.7.7 Работа с Kanban-доской ..... 49
1.8 Руководство пользователя ..... 50

1.8.1 Результат работы приложения ..... 50
2 Безопастность жизнедеятельности ..... 57
Заключение ..... 59
Список использованных источников ..... 60
Приложение А – UML-диаграммы ..... 62
Приложение Б – Листинг программы ..... 67
Приложение В – Презентация ..... 135

## Перечень сокращений и обозначений

В данной работе используются следующие сокращения обозначений

БД – база данных

ПК – персональный компьютер

ПО – программное обеспечение

IDE (Integrated Development Environment) – интегрированная среда разработки

UI (User Interface) – пользовательский интерфейс

UML (Unified Modeling Language) – унифицированный язык моделирования

QML (Qt Modeling Language) – декларативный язык разметки интерфейса, используемый в фреймворке Qt

SQL (Structured Query Language) – язык структурированных запросов для работы с реляционными базами данных

Qt – кроссплатформенный фреймворк для разработки приложений на языке C++

SQLite – встраиваемая реляционная система управления базами данных

QR-код (Quick Response code) – двумерный штрихкод для быстрого считывания информации

Kanban – метод визуального управления задачами с распределением по статусам выполнения

## Введение

Умение орагнизовывать собственную работу напрямую влияет на её результат. Когда задачи разбросаны по заметкам, мессенджерам и памяти, то неизбежно что-то упускается. Инструменты управления задачами решают данную проблему.

На рынке существует немало подобных решений, однако каждое из них имеет свои ограничения. Одни сервисы хорошо справляются с простыми списками, но не дают возможности структурировать работу по проектам. Другие предлагают богатый функционал, но часть возможностей доступна по подписке, а остальные полностью зависят от облака и становятся недоступны без подключения к сети.

Разработка собственного локального приложения представляет интерес – как способ получить инструмент без лишних ограничений

Объект исследования: процесс ведения и контроля задач пользователем.

Предмет исследования: настольное приложение для управления задачами с локальным хранением данных.

Цель исследования: разработка многофункционального персонального менеджера задач на основе C++ и Qt с использованием базы данных.

Задача исследования:

- анализ предметной области и существующих аналогов;
- выбор и обоснование инструментальных средств разработки;
- проектирование архитектуры приложения и структуры базы данных;
- реализация основного функционала приложения;
- тестирование разработанного приложения.

Актуальность исследования: востребованность локальных инструментов планирования, независимых от сетевого подключения и сторонних сервисов.

Практическая значимость: готовый инструмент повседневного планирования, не требующий регистрации, подписки и подключение к интернету.

# 1 Многофункциональны персональный менеджер задач

## 1.1 Анализ предметной области

Управление задачами представляет собой процесс фиксации, распределения и контроля выполнения различных действий, которые пользователь должен выполнить в определенный срок или в рамках конкретного проекта [1], [2], [3]. Такая деятельность встречается как в учебной, так и в рабочей и повседневной среде. Задачи могут отличаться по важности, срокам, объему, принадлежности к определенному проекту и текущему состоянию выполнения. Поэтому для их удобной обработки требуется не просто список записей, а система, позволяющая упорядочивать информацию и быстро получать доступ к нужным данным.

В процессе планирования пользователь обычно сталкивается с несколькими основными действиями: добавлением новой задачи, определением срока ее выполнения, установкой приоритета, отслеживанием статуса и контролем завершения. Если количество задач небольшое, эти действия можно выполнять вручную. Однако при увеличении нагрузки обычный список становится менее удобным, так как в нем сложно быстро определить, какие дела являются срочными, какие уже выполнены, а какие относятся к определенному направлению работы. Кроме того, для более точной организации могут использоваться теги, подзадачи и фильтры, позволяющие выделять нужную часть информации.

Важным элементом предметной области является контроль сроков. Многие задачи имеют конкретную дату или время выполнения, поэтому пользователь должен заранее видеть приближающиеся дедлайны. Если система не напоминает о важных сроках, то велик риск пропустить задачу, в связи с этим в приложениях для управления задачами часто используются напоминания, отображение просроченных задач.

Также при работе с задачами важно учитывать их текущее состояние. Одни задачи только запланированы, другие уже находятся в процессе выполнения, третьи завершены. Для наглядного отображения такого процесса может

использоваться канбан-подход, при котором задачи распределяются по колонкам в зависимости от статуса. Это позволяет пользователю оценивать не только отдельные записи, но и общий ход выполнения работы.

В итоге область управления задачами включает в себя хранение, структурирование, отображение и контроль выполнения пользовательских дел. Разработка приложения в данной области требует учета таких функций, как создание и редактирование задач, работа со сроками, приоритетами, проектами, тегами, подзадачами и статусами выполнения. Именно эти особенности должны быть положены в основу дальнейшей разработки программного продукта.

1.1.1 Описание потребности потребителя

В повседневной жизни человек регулярно сталкивается с необходимостью удерживать в голове множество дел одновременно: рабочие поручения, учебные задания, личные планы и текущие обязательства. При небольшом количестве задач достаточно простого списка, но, когда задач становится значительно больше, такой подход перестаёт справляться — дела теряются, сроки пропускаются, а приоритеты смешиваются.

Основными пользователями подобного приложения являются несколько категорий людей. Первая – это студенты и учащиеся, которым необходимо совмещать учёбу, личные дела и подработку. Для них важна возможность быстро зафиксировать задачу, указать срок и не забыть о ней в нужный момент. Вторая категория – специалисты, ведущие несколько направлений работы одновременно. Им требуется распределять задачи по проектам, отслеживать их статус и видеть общую картину выполнения. Третья категория – люди, которые просто хотят навести порядок в личных делах, не прибегая к сложным инструментам с избыточным функционалом.

Общей потребностью всех перечисленных пользователей является наличие простого, быстрого и надёжного инструмента, который работает локально, не требует регистрации и позволяет в любой момент просмотреть, добавить или изменить задачу. При этом приложение должно поддерживать как базовые

функции – создание и отметку выполнения, так и дополнительные возможности: разбивку на проекты, установку приоритетов, напоминания и наглядное отображение статусов.

## 1.2 Анализ существующих аналогов

До начала разработки приложения для управления задачами следует изучить существующие решения, которые уже применяются в данной сфере. Рассмотрение аналогов помогает понять, какие возможности являются наиболее востребованными, какие элементы интерфейса удобны для пользователя, а также какие недостатки стоит избежать при создании собственного приложения.

### 1.2.1 Сервис “Todoist”

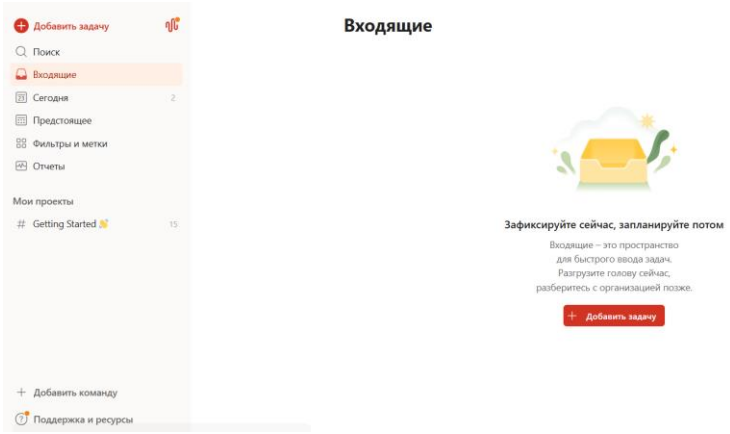



Рисунок 1 – Главная страница сервиса Todoist

На рисунке 2 представлена форма добавления задачи, где видны срок выполнения, приоритет, проект или метки.

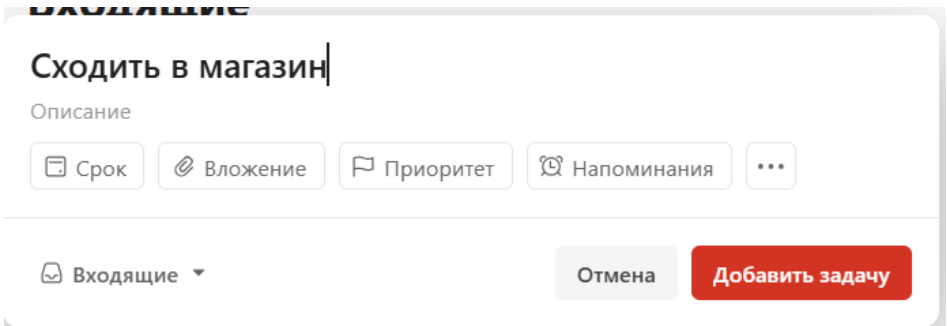



Рисунок 2 – Создание задачи в сервисе Todoist

На рисунке 3, показана детальное окно задачи, где можно изменять её свойства.

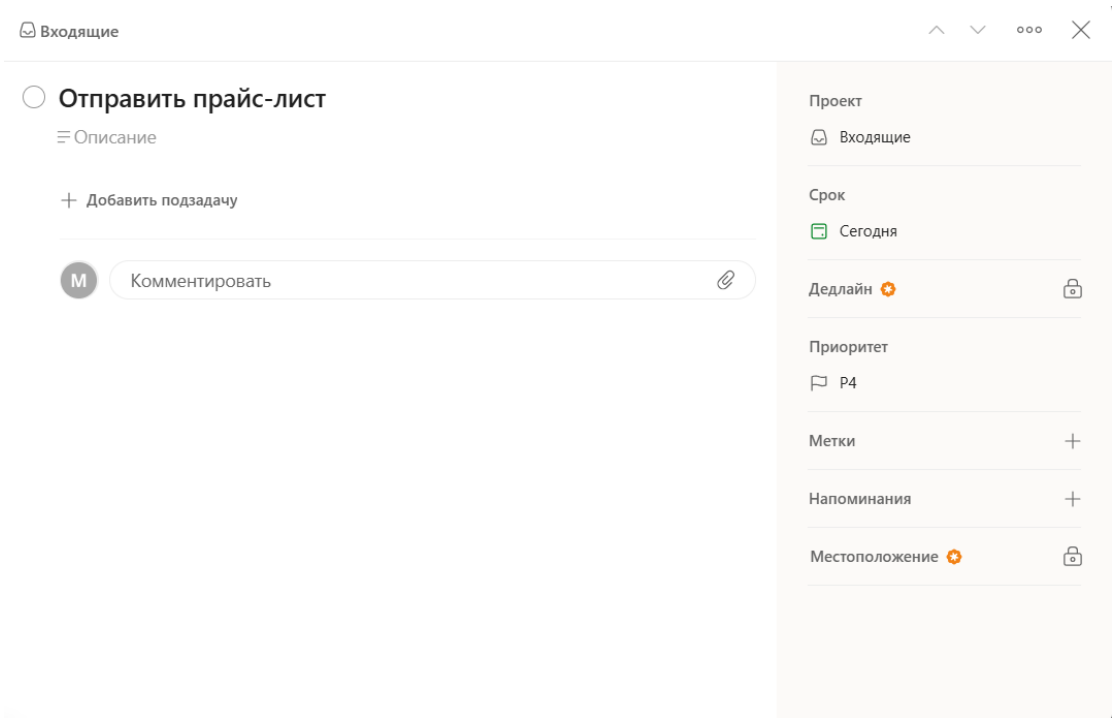



Рисунок 3 – Детальное окно задачи

Достоинства сервиса Todoist:

– современный и понятный пользовательский интерфейс;

– возможность распределять задачи по проектам;

– поддержка сроков выполнения, приоритетов, меток и фильтров;

– наличие синхронизации между разными устройствами;

– возможность использовать сервис как для личного, так и для рабочего планирования.

Недостатки сервиса Todoist:

– часть функций доступна только в платной версии;

– для полноценной работы требуется учетная запись;

– приложение ориентировано на облачную синхронизацию, что не всегда удобно при необходимости локального использования;

– большое количество функций может быть избыточным для пользователя, которому нужен простой настольный менеджер задач.

1.2.2 Сервис “Google Tasks”

Google Tasks представляет собой сервис для создания и ведения списков задач, входящий в экосистему Google. На рисунке 4 отображен главный экран Google Tasks.

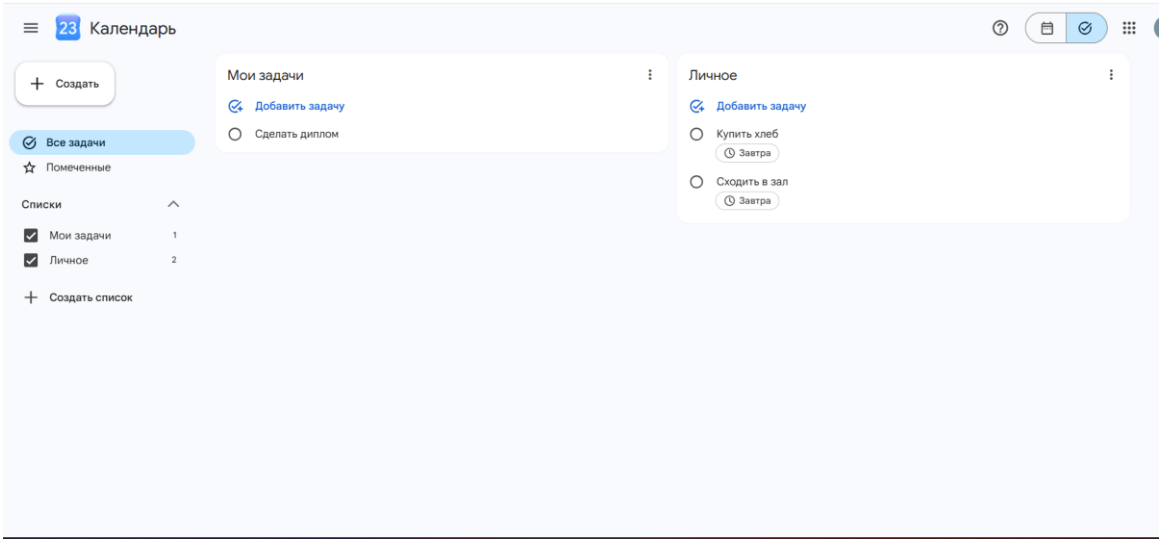



Рисунок 4 – Главный экран сервиса Google Tasks

На рисунке 5 представлено окно подробной информации о задаче в Google Tasks.

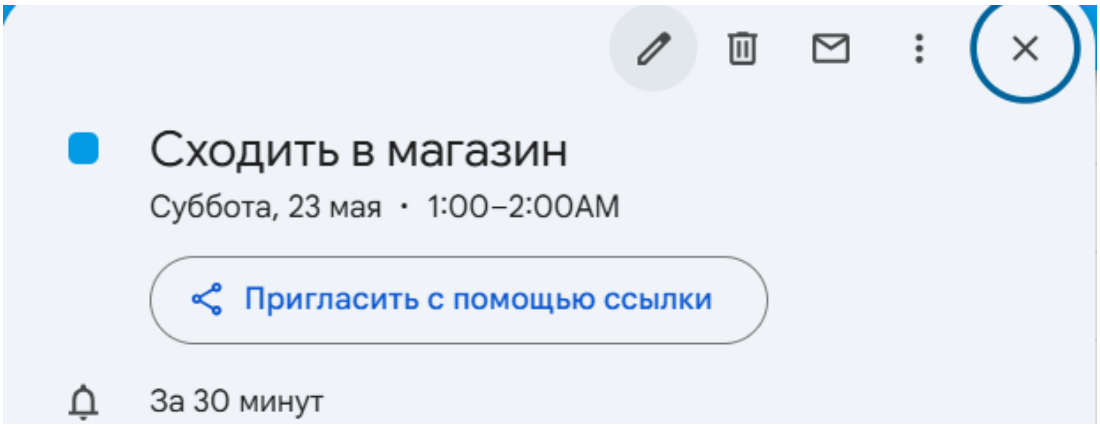



Рисунок 5 – Подробная информация о задаче в Google Tasks

На рисунке 6, показана интеграция с Google Calendar, где можно увидеть, задачи в виде календаря

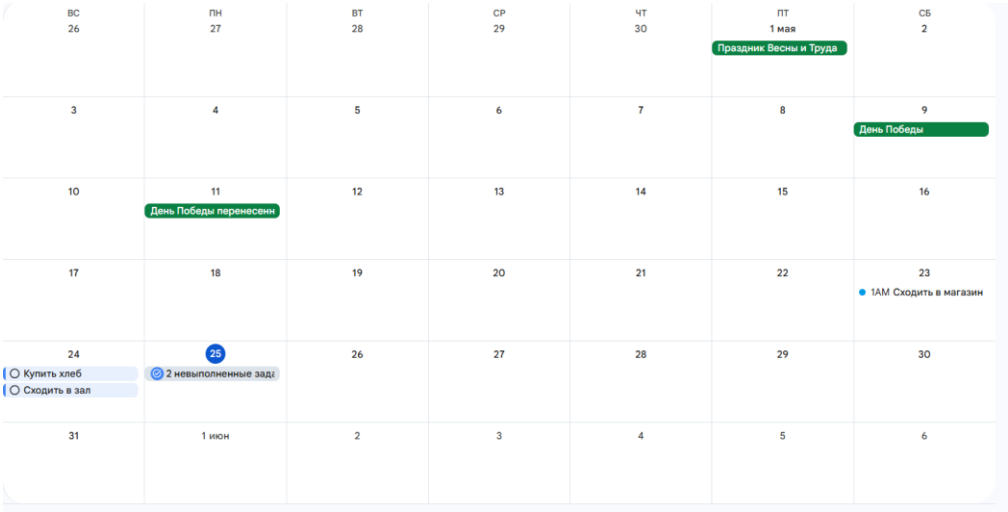



Рисунок 6 – Интеграция с Google Calendar

Достоинства сервиса Google Tasks:

– простой и понятный интерфейс;

– возможность создавать задачи и распределять их по спискам;

– поддержка сроков выполнения и подзадач;

– интеграция с сервисами Google, включая Gmail и Google Calendar;

– удобство для пользователей, которые уже используют учетную запись Google.

Недостатки сервиса Google Tasks:

– ограниченный набор функций по сравнению с более развитыми менеджерами задач;

– отсутствует полноценная канбан-доска;

– нет расширенной системы приоритетов и тегов;

– приложение больше подходит для простых списков дел, чем для работы с несколькими проектами;

– зависимость от экосистемы Google.

1.2.3 Сервис “Trello”

На рисунке 7 представлена канбан-доска в сервисе Trello.

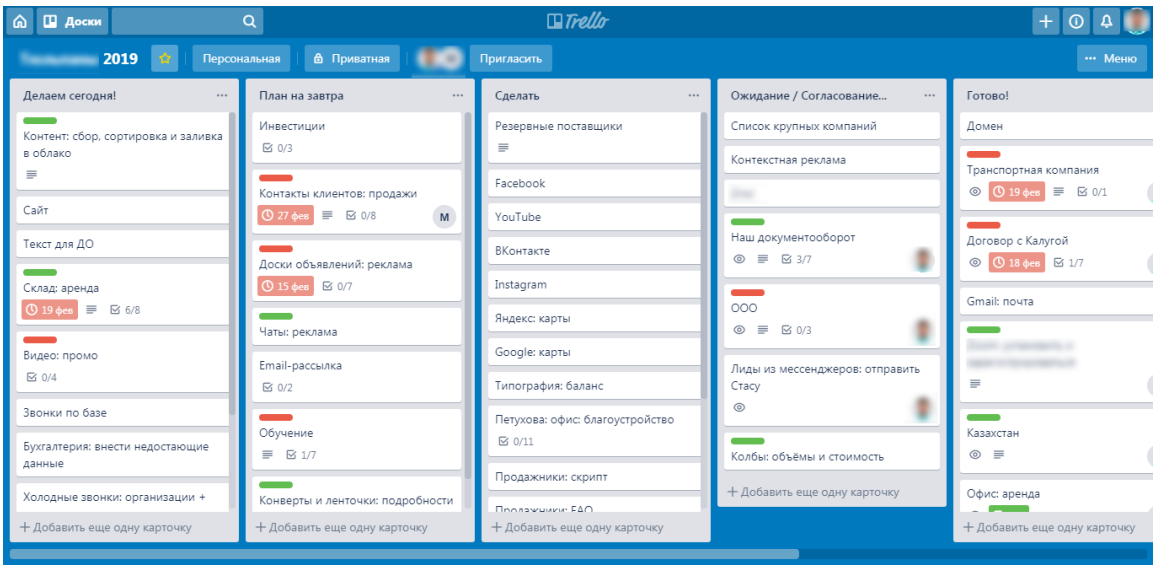



Рисунок 7 – Канбан-доска в сервисе Trello

На рисунке 8 представлена карточка задачи в сервисе Trello.

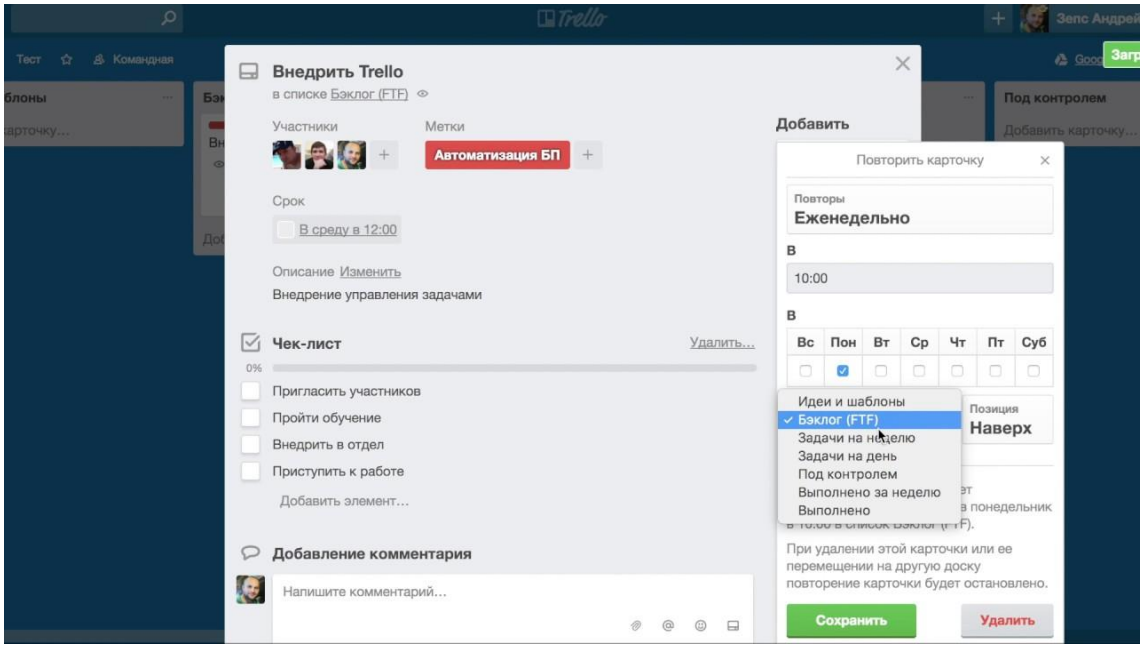



Рисунок 8 – Карточка задачи в сервисе Trello

Достоинства сервиса Trello:

– наглядное отображение задач с помощью канбан-доски;

– удобное перемещение карточек между разными статусами;

– возможность использовать сервис для командной работы;

– поддержка описаний, меток, сроков и чек-листов внутри карточек;

− простое визуальное представление хода выполнения проекта.

Недостатки сервиса Trello:

− при большом количестве задач доска может становиться перегруженной;

− сервис больше ориентирован на командную и проектную работу;

− для личного планирования интерфейс может быть не всегда удобен;

− некоторые расширенные возможности доступны только в платных тарифах.

### 1.3 Постановка задачи

Разрабатываемая программа представляет собой настольное приложение для ведения и контроля задач. Основная идея проекта заключается в создании удобного инструмента, с помощью которого пользователь сможет фиксировать свои дела, распределять их по отдельным направлениям и отслеживать выполнение в процессе работы. Приложение должно быть достаточно простым для повседневного использования, но при этом включать функции, необходимые для полноценного планирования. В ряде популярных аналогов часть полезных функций, например, расширенная работа с проектами, напоминания, дополнительные представления задач или синхронизация, может быть доступна только в платных версиях. Кроме того, некоторые сервисы зависят от постоянного подключения к интернету, требуют регистрации или могут быть недоступны в отдельных регионах. Поэтому разработка локального настольного приложения является актуальной задачей, так как позволяет создать инструмент, которым можно пользоваться без привязки к сторонним онлайн-сервисам.

Сначала в программе требуется реализовать добавление задач. Пользователь должен иметь возможность быстро создать новую запись, указав её название, а затем при необходимости дополнить её дополнительными параметрами. К таким параметрам относятся срок выполнения, уровень важности, принадлежность к определённому проекту, наличие напоминания, а также дополнительные элементы, позволяющие подробнее описать задачу. Благодаря этому задача в системе будет

представлять собой не просто строку в списке, а полноценный объект с набором свойств.

Отдельная часть разработки связана с отображением задач. В приложении необходимо предусмотреть несколько вариантов просмотра: обычный список и представление в виде Kanban-доски. Список удобен для быстрого просмотра всех задач и их основных параметров, а Kanban-доска позволяет видеть текущее состояние работы: какие задачи ещё не начаты, какие находятся в процессе выполнения, а какие уже завершены. Такой способ отображения делает контроль над выполнением более наглядным.

Немало важным элементом приложения является панель подробной информации о задаче. При выборе конкретной задачи пользователь должен видеть её основные свойства и иметь возможность изменить их без создания новой записи. В данной панели необходимо предусмотреть редактирование названия, проекта, приоритета и срока выполнения. Кроме того, следует добавить поддержку подзадач и тегов. Подзадачи позволяют разбить крупное дело на отдельные шаги, а теги помогают дополнительно обозначать задачи по смыслу или тематике. Пользователь также должен иметь возможность указать, когда программа должна напомнить о приближающемся сроке выполнения. Такая функция помогает не пропускать важные задачи и делает приложение более полезным при ежедневном планировании. В программе должна быть реализована такая функция, как генерация QR-кода, чтобы пользователь не был привязан к приложению и всегда мог отследить нужные ему задачи вне компьютера или ноутбука. Для создания QR-кода задача должна иметь установленный срок выполнения, так как именно он является важной частью события.

Отдельным требованием к приложению является надёжное сохранение данных между сеансами работы. Так как, пользователь может накапливать задачи постепенно – добавляя новые, редактируя существующие и распределяя их по проектам – вся эта информация не должна теряться после закрытия программы. Для этого в приложении используется встроенная реляционная база данных SQLite,

которая работает непосредственно в процессе самого приложения и не требует установки отдельного сервера или настройки подключения [4], [5]. Задачи, проекты, подзадачи и теги хранятся в структурированном виде по отдельным таблицам, что позволяет эффективно выполнять выборку, фильтрацию и изменение записей. При каждом запуске приложения данные автоматически загружаются из базы, а любое изменение немедленно отражается в хранилище, то есть, пользователь продолжает работу с актуальным состоянием задач вне зависимости от того, когда было закрыто приложение.

Диаграмма вариантов прецедентов представлена на рисунке 10 и в приложении А.1

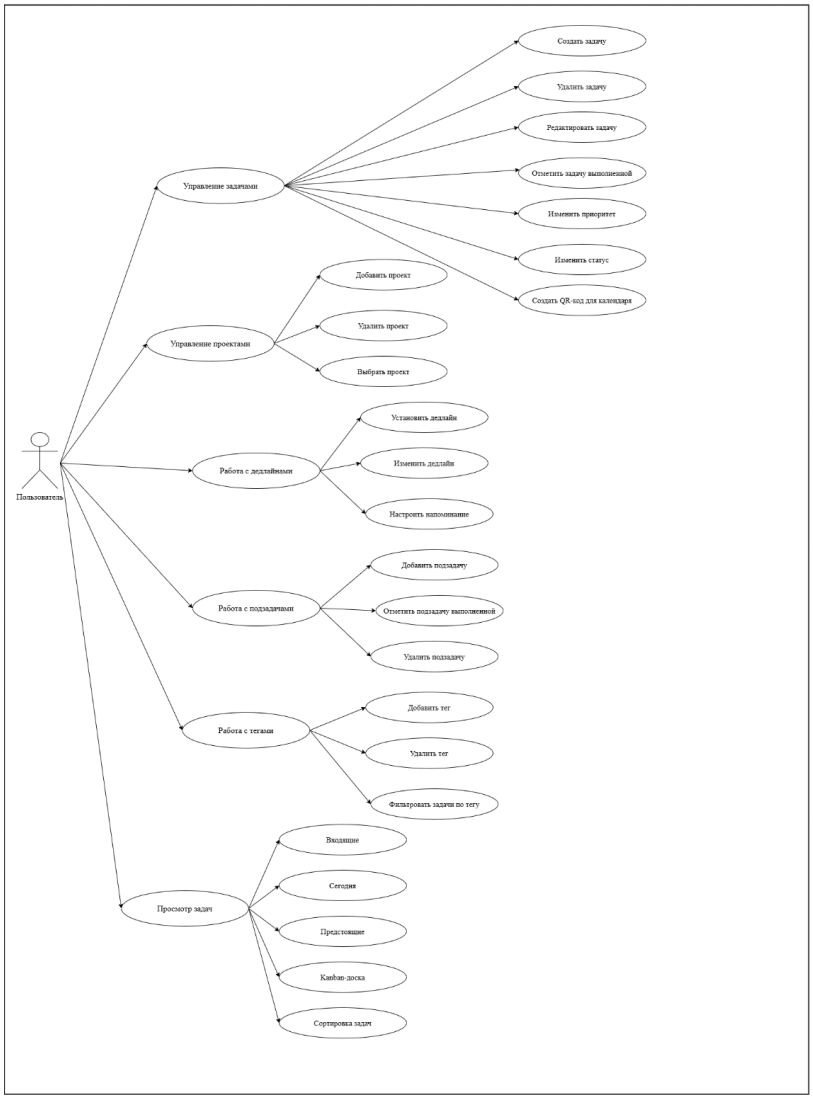



Рисунок 10 – UML- диаграмма вариантов прецедентов.

## 1.4 Выбор инструментальных средств

В этом разделе был проведен анализ выбора программных средств, которые применяются при создании приложения. Исходя из этого, были выбраны технологии, которые обеспечивают удобство реализации как визуальной части, так и внутреннюю обработку пользовательских данных.

1.4.1 Выбор языка программирования

Для написания логики приложения можно было рассмотреть несколько языков программирования: Python, C# и C++.

Python является распространённым языком программирования, который применяется в разных областях: от небольших пользовательских программ до серверных приложений и обработки данных. Для создания графических интерфейсов на Python есть множество библиотек. Однако для приложения, где интерфейс постоянно взаимодействует с моделью данных, выполняется фильтрация, сортировка и обновление задач, Python может быть не самым удачным выбором. Кроме того, программа на Python может требовать установки дополнительных зависимостей, что не всегда удобно для конечного пользователя.

C# тоже можно рассматривать как вариант для создания настольного приложения. Этот язык часто используется при разработке программ под Windows, поскольку хорошо работает вместе с платформой .NET. Однако C# оказался менее подходящим вариантом, так как приложение планируется строить на Qt и использовать QML для интерфейса, поэтому C# не является основным и наиболее естественным выбором [6], [7], [8].

Язык C++ подходит для разработки приложений, в которых требуется стабильная работа внутренней логики и тесное взаимодействие с графическим интерфейсом. Кроме того, в C++ позволяет удобно описывать данные в виде отдельных структур и классов [9], [10]. Также немаловажным преимуществом было то, что у C++ высокая производительность при обработке данных [10].

По итогу, в качестве языка программирования, был выбран C++, потому что он позволяет построить надёжную основу приложения, где данные

обрабатываются быстро, структура проекта остаётся понятной, а функционал можно постепенно дополнять без полной переработки программ.

1.4.2 Выбор фреймворка для разработки приложения

Так как основным языком разработки является C++, то рассмотрим варианты, которые хорошо подходят для создания приложений на данном языке. Такие варианты, как Qt, wxWidgets и GTKmm.

wxWidgets является известной библиотекой для разработки графических приложений на C++ [11]. Она позволяет создавать программы для разных операционных систем и использовать стандартные элементы интерфейса, характерные для конкретной платформы [11]. Это делает внешний вид приложения привычным для пользователя. Однако wxWidgets больше подходит для классических интерфейсов с обычными окнами, кнопками, списками и таблицами. Если потребуется я более гибкая визуальная настройка, современное оформление и удобная работа с нестандартными элементами интерфейса, разработчику может понадобиться больше ручной доработки.

GTKmm – это инструмент, который позволяет разрабатывать графические приложения на C++ с использованием возможностей GTK [12]. Этот инструмент также позволяет создавать настольные приложения с графическим интерфейсом и использовать готовые элементы управления [12]. GTKmm может быть удобен для программ, ориентированных на Linux-среду, так как GTK широко применяется именно в этой экосистеме. При разработке под Windows использование GTKmm может оказаться менее удобным, поскольку потребуется уделять больше внимания настройке окружения, сборке и внешнему виду приложения.

Qt является одним из наиболее распространённых фреймворков для разработки приложений на C++ [6], [13]. Он предоставляет средства для создания графического интерфейса, обработки событий, работы с файлами, моделями данных и другими частями программы. Кроме того, Qt поддерживает создание интерфейса с помощью QML, что позволяет сделать внешний вид приложения более современным и гибким [7], [8].

В итоге, проведенных вышеперечисленных вариантов был выбран Qt. Он лучше подходит для разрабатываемого приложения, так как позволяет объединить логику на C++ с современным интерфейсом.

1.4.3 Выбор СУБД

Для постоянного хранения данных приложения необходимо выбрать подходящую систему управления базами данных. Были рассмотрены три варианта: MySQL, PostgreSQL и SQLite.

PostgreSQL – объектно-реляционная СУБД, отличающаяся широкой поддержкой стандартов SQL и расширенными возможностями работы с данными [14]. Она поддерживает сложные типы данных, хранимые процедуры, полнотекстовый поиск и гибкую систему расширений [14]. PostgreSQL также работает по клиент-серверной модели и рассчитана на нагруженные системы, где требуются сложные запросы и высокая точность при работе с данными.

MySQL – одна из наиболее широко применяемых реляционных СУБД с клиент-серверной архитектурой [15]. Она поддерживает работу с большим объёмом данных, обеспечивает одновременный доступ множества пользователей и предоставляет развитые средства управления правами и соединениями [15]. MySQL устанавливается как отдельный серверный процесс, к которому приложение подключается через сетевой интерфейс, что делает её распространённым выбором для веб-приложений и многопользовательских систем [15].

SQLite представляет собой встраиваемую реляционную базу данных, которая не использует отдельный серверный процесс [5]. Вся база данных хранится в одном файле на диске, а взаимодействие с ней происходит через библиотеку, подключаемую непосредственно к приложению [5]. SQLite поддерживает стандартный синтаксис SQL, включая создание таблиц, выборку, изменение и удаление записей, а также внешние ключи для связи между таблицами. В составе Qt имеется готовый драйвер QSQLITE, входящий в модуль QtSql, что позволяет работать с такой базой данных без подключения сторонних библиотек [4].

По итогам рассмотрения для хранения данных была выбрана SQLite. Поскольку разрабатываемое приложение является однопользовательским и работает локально на компьютере пользователя, необходимости в серверной СУБД нет. MySQL и PostgreSQL потребовали бы установки и настройки отдельного серверного процесса, что создало бы дополнительные зависимости при развёртывании программы [15], [14]. SQLite лишена этого ограничения: она встраивается непосредственно в приложение, не требует никакой предварительной настройки со стороны пользователя и при этом предоставляет полноценный реляционный инструментарий, достаточный для хранения задач, проектов, подзадач и тегов.

1.4.4 Выбор среды обработки

Для разработки приложения можно использовать Qt Creator, CLion или Visual Studio.

Visual Studio представляет собой полноценную среду разработки. Она содержит встроенные инструменты для написания, проверки и отладки кода, поддерживает работу с C++ и другими языками программирования [16]. Однако при работе с Qt и QML часть возможностей может потребовать дополнительной настройки, поэтому такая среда больше подходит для общей C++-разработки, чем для проекта, полностью ориентированного на Qt [6], [7].

CLion – это среда разработки от JetBrains, предназначенная для C и C++ [17]. Она поддерживает современные инструменты разработки, работу с CMake, отладку и анализ кода [17]. CLion также может использоваться для Qt-проектов: в документации JetBrains отдельно описана работа с Qt, включая создание CMake-based Qt project [17].

Qt Creator предназначен для создания приложений с использованием технологий Qt [18]. В нём предусмотрены инструменты для работы с C++, QML, сборкой, запуском и отладкой проекта [18]. Также среда позволяет удобно работать как с программной логикой, так и с файлами интерфейса, что важно при создании приложения с разделением визуальной и внутренней части [18]. Разработчику не

нужно вручную настраивать, так как большое количество инструментов, так как основные возможности уже предусмотрены.

После рассмотренных перечисленных вариантов, была выбрана среда разработки Qt Creator, потому что он не требует сложной предварительной настройки и позволяет выполнять основные этапы разработки в одном рабочем пространстве.

1.4.5 Выбор технологии создания пользовательского интерфейса

В Qt можно использовать разные подходы к созданию интерфейса. Наиболее подходящими вариантами являются Qt Widgets и Qt Quick.

Qt Widgets – это классический способ разработки интерфейсов в Qt. Он хорошо подходит для традиционных программ, где используются стандартные окна, кнопки, таблицы, списки и формы [6], [13]. Такой вариант хорошо подходит для привычных оконных программ, но при разработке интерфейса с большим количеством визуальных состояний, анимаций и нестандартных элементов его использование может усложнить работу.

Qt Quick ориентирован на создание более гибких и визуально настраиваемых интерфейсов [7]. В QML удобно описывать элементы через их свойства, состояния и связи между компонентами [7]. Такой вариант подходит, для программ, где требуется более свободная настройка расположения, внешнего вида и поведения элементов интерфейса.

Для проекта был выбран Qt Quick. Он позволяет быстрее создавать визуальные компоненты и проще менять оформление приложения. При этом логика задач остаётся на C++, а QML отвечает за отображение данных и обработку действий пользователя [8].

## 1.5 Архитектура приложения

Архитектура приложения построена на разделении внутренней логики и пользовательского интерфейса. Такой подход позволяет не смешивать обработку данных с визуальным отображением, за счёт чего проект становится более

понятным, удобным для изменения и дальнейшего расширения. В Qt подобное разделение хорошо соответствует подходу Model/View, при котором данные находятся в модели, а представление получает их через определённый интерфейс и отображает пользователю [6], [19], [13].

В основе приложения находится класс TaskModel, реализованный на C++. Он отвечает за хранение и обработку задач, а также предоставляет интерфейсу доступ к необходимым данным. В модели описываются основные свойства задачи: название, статус выполнения, проект. Также в данной части программы выполняются основные операции: добавление задачи, удаление, изменение названия, установка дедлайна, изменение приоритета, переключение статуса выполнения и работа с проектами. В заголовочном файле модели перечислены роли задачи, включая TitleRole, DoneRole, StatusRole, PriorityRole, DeadlineRole, ProjectRole, SubTasksTextRole и TagsRole, что показывает, какие данные передаются в интерфейс [19]. Связь между C++-логикой и QML-интерфейсом выполняется при запуске приложения. В главном файле создаётся объект TaskModel, после чего он передаётся в QML как контекстное свойство taskModel. Благодаря этому интерфейс может обращаться к функциям модели и получать из неё актуальные данные [8], [19]. Такой способ позволяет оставить основную обработку данных на стороне C++, а в QML вынести отображение и обработку действий пользователя [7], [8].

Пользовательский интерфейс приложения реализован на QML и состоит из нескольких самостоятельных визуальных компонентов. Главное окно приложения задаёт общие размеры, цветовую схему, выбранный раздел, выбранную задачу и режим отображения. В нём подключаются основные части интерфейса: боковая панель, список задач, панель подробной информации и Kanban-доска. Такой подход позволяет разделить интерфейс на отдельные блоки, каждый из которых отвечает за свою область экрана и выполняет конкретную функцию.

Боковая панель используется для навигации по приложению. В ней отображаются основные разделы, такие как входящие задачи, задачи на сегодня,

предстоящие задачи, проекты и другие элементы. Через эту панель пользователь выбирает, какие задачи должны отображаться в центральной части окна. При выборе раздела интерфейс вызывает соответствующие методы модели, которые изменяют текущий фильтр. Таким образом, боковая панель не хранит задачи самостоятельно, а только управляет тем, какие данные необходимо показать[7], [19].

Центральная область приложения отвечает за отображение задач. В обычном режиме задачи выводятся в виде списка карточек. Каждая карточка показывает основные сведения: название, проект, срок выполнения, приоритет, статус и теги. При нажатии на карточку выбранная задача передаётся в главное окно, после чего её данные отображаются в правой панели. Такой принцип делает работу с задачами более удобной: пользователь видит общий список, а подробности открываются отдельно. [8], [19]

Правая панель предназначена для просмотра и редактирования выбранной задачи. В ней можно изменить все основные функции. При изменении данных панель не редактирует их напрямую внутри интерфейса, а вызывает методы модели. После этого модель обновляет задачу и сохраняет изменения. Такой вариант помогает избежать дублирования данных в разных частях программы [19],[20].

Отдельным режимом отображения является Kanban-доска. В этом режиме задачи распределяются по колонкам в зависимости от их статуса: к выполнению, в процессе и выполнено. Kanban-доска получает данные из той же модели, что и обычный список, но отображает их в другом виде. Это является важным преимуществом выбранной архитектуры: одни и те же данные могут быть представлены пользователю по-разному без создания отдельного хранилища для каждого режима.

Сохранение данных относится к внутренней логике приложения. При каждом изменении задачи модель незамедлительно фиксирует актуальное состояние в базе данных SQLite, а при запуске программы выполняет загрузку всех ранее

сохранённых записей [4], [5]. Задачи, проекты, подзадачи и теги хранятся в отдельных таблицах, связанных между собой через идентификаторы, что позволяет точечно изменять конкретные записи, не затрагивая остальные данные. Такой механизм делает приложение независимым от внешних сервисов и позволяет пользователю продолжить работу после закрытия программы без потери каких-либо данных [5].

Система напоминаний реализуется на стороне модели. При запуске приложения создаётся таймер, который проверяет наступление времени напоминаний. Если для задачи установлен срок выполнения и выбран момент уведомления, программа может показать системное сообщение пользователю [21].

Дополнительной частью архитектуры является генерация QR-кода. Пользователь выбирает задачу, после чего интерфейс обращается к модели для создания изображения QR-кода. Если задача не выбрана или у неё отсутствует срок выполнения, программа выводит сообщение об ошибке. Данная функция также встроена в общую структуру приложения: интерфейс отвечает за действие пользователя и отображение результата, а формирование данных выполняется во внутренней логике [7], [8].

Описанная структура приложения даёт возможность развивать отдельные его части независимо друг от друга: новый раздел интерфейса не затрагивает логику обработки задач, а доработка модели данных не требует переработки визуальных компонентов.

## 1.6 Реализация

В этом разделе рассматривается процесс реализации настольного приложения для управления задачами [6], [7]. Описывается общая логика работы программы, структура программных модулей, организация хранения данных, а также взаимодействие пользовательского интерфейса с внутренней моделью приложения [22].

1.6.1 Описание работы приложения

Расположение основных компонентов системы и их связь с физическими узлами показаны на рисунке 11 и в приложении А.2.

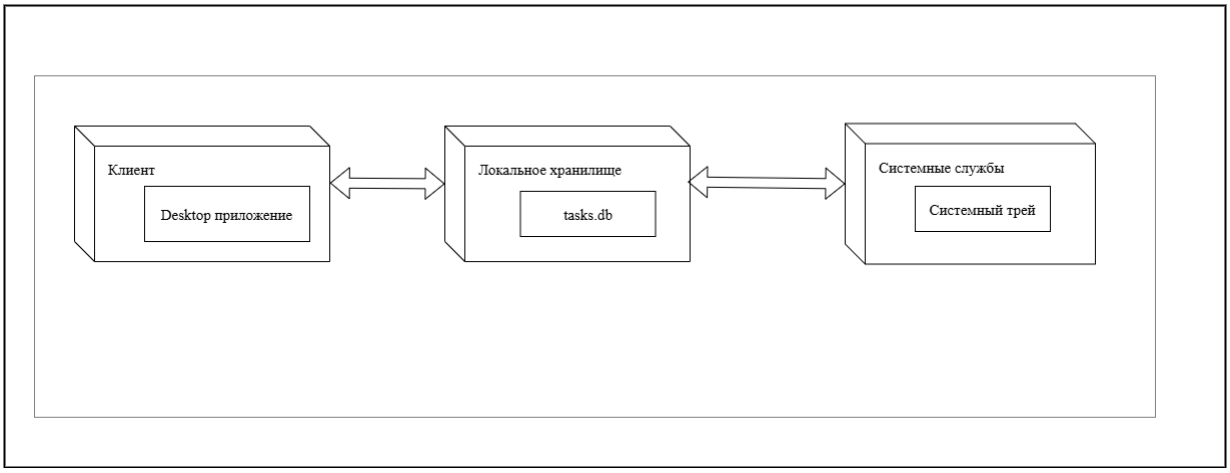



Рисунок 11 – UML-диаграмма развертывания системы

После запуска приложения пользователь взаимодействует с графическим интерфейсом, через который выполняет основные действия с задачами [6], [7]. При создании, изменении или удалении задачи данные передаются во внутреннюю модель приложения, где выполняется их обработка. Внутренняя модель выполняет необходимые операции с задачами: добавление, редактирование, удаление, фильтрацию и получение данных [19]. После успешного выполнения операции изменения сохраняются в локальном файле, а интерфейс приложения обновляется и отображает актуальное состояние задач.

Если действие не может быть выполнено, например, из-за отсутствия выбранной задачи или некорректных данных, состояние приложения не изменяется. Последовательность выполняемых действий при работе с задачей представлена на рисунке 12 и в приложении А.3.

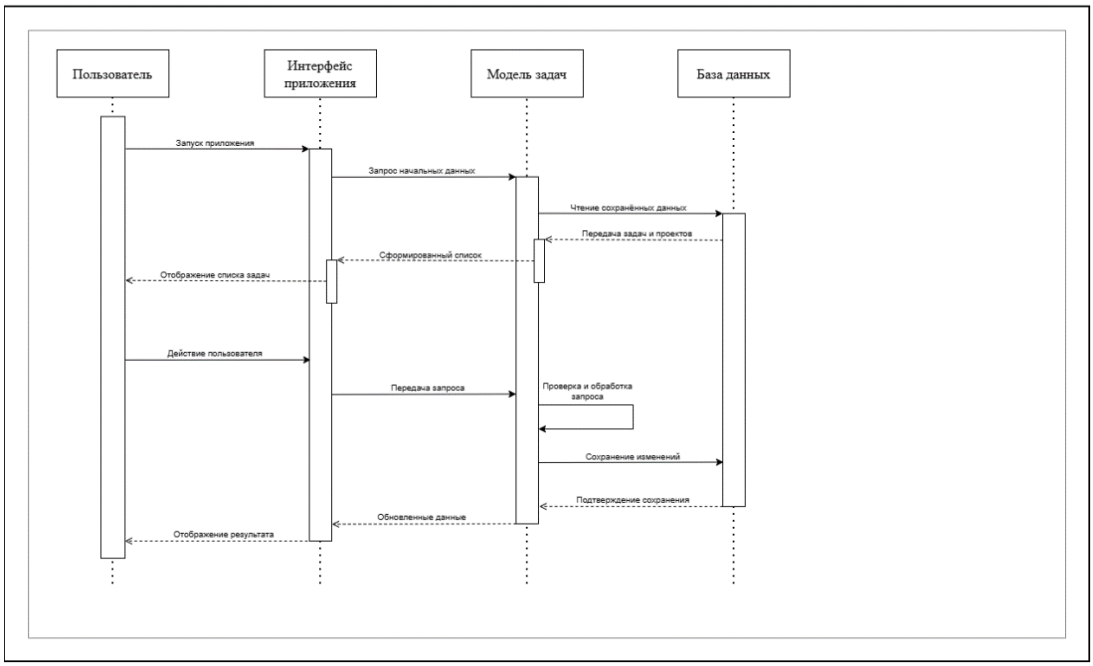



Рисунок 12 – UML-диаграмма последовательности функционирования приложения

1.6.2 Структуры базы данных

В разработанном приложении данные хранятся в реляционной базе данных SQLite [23], [4]. Поскольку структура базы данных определяет эффективность хранения и обработки информации, в данном разделе рассматриваются основные таблицы, их атрибуты и связи между ними.

Проведя анализ предметной области, было решено использовать следующие таблицы:

Таблица tasks. В данной таблице хранится основная информация о задачах.

Таблица 1 – tasks

| Поле | Тип данных | Описание | Ключ |
|---|---|---|---|
| id | INTEGER | Идентификатор задачи | PK |
| title | TEXT | Название задачи | |
| done | INTEGER | Признак выполнения | |
| status | TEXT | Статус задачи | |
| priority | INTEGER | Приоритет задачи | |

| Поле | Тип данных | Описание | Ключ |
|---|---|---|---|
| project | TEXT | Название проекта | |
| has_deadline | INTEGER | Наличие срока выполнения | |
| deadline | TEXT | Срок выполнения | |
| has_reminder | INTEGER | Наличие напоминания | |
| reminder_minutes | INTEGER | Время напоминания (в минутах) | |
| reminder_shown | INTEGER | Признак показа напоминания | |

Таблица subtasks. В данной таблице хранится информация о подзадачах, привязанных к конкретной задаче.

Таблица 2 – subtasks

| Поле | Тип данных | Описание | Ключ |
|---|---|---|---|
| id | INTEGER | Идентификатор подзадачи | PK |
| task_id | INTEGER | Идентификатор задачи | FK |
| title | TEXT | Название подзадачи | |
| done | INTEGER | Признак выполнения | |
| pos | INTEGER | Порядковый номер | |

Таблица tags. В данной таблице хранятся теги, привязанные к задачам.

Таблица 3 – tags

| Поле | Тип данных | Описание | Ключ |
|---|---|---|---|
| task_id | INTEGER | Индетификатор задачи | PK, FK |
| tag | TEXT | Значение тега | PK |

Таблица projects. В данной таблице хранится информация о проектах.

Таблица 4 – projects

| Поле | Тип данных | Описание | Ключ |
|---|---|---|---|
| name | TEXT | Название проекта | PK |

### 1.6.3 Описание компонентов

В основе приложения используется разделение на две основные части: внутреннюю логику и визуальное представление. Внутренняя часть реализована на C++, а пользовательский интерфейс разработан с использованием QML [7], [8]. Такой подход позволяет отделить обработку данных от отображения информации на экране. В результате изменения во внешнем виде приложения не требуют полной переработки логики, а доработка функций работы с задачами не затрагивает большую часть интерфейсных компонентов [8].

Пользовательский интерфейс приложения реализован с использованием QML-компонентов [7]. QML применяется для описания визуальных элементов приложения и их поведения, а основная логика обработки данных вынесена в C++-класс TaskModel. Такой подход соответствует рекомендуемой схеме интеграции Qt Quick и C++, при которой интерфейс реализуется на QML, а логика и модель данных – на C++ [8].

Модель данных отвечает за хранение задач в оперативной памяти, обработку действий пользователя и передачу данных в интерфейс. В приложении модель представлена классом TaskModel, который наследуется от QAbstractListModel [19]. При наследовании от QAbstractListModel необходимо реализовать методы rowCount() и data(), а для использования дополнительных ролей в QML – переопределить roleNames().

UML-диаграмма компонентов приложения представлена на рисунке 13 и в приложении А.4.

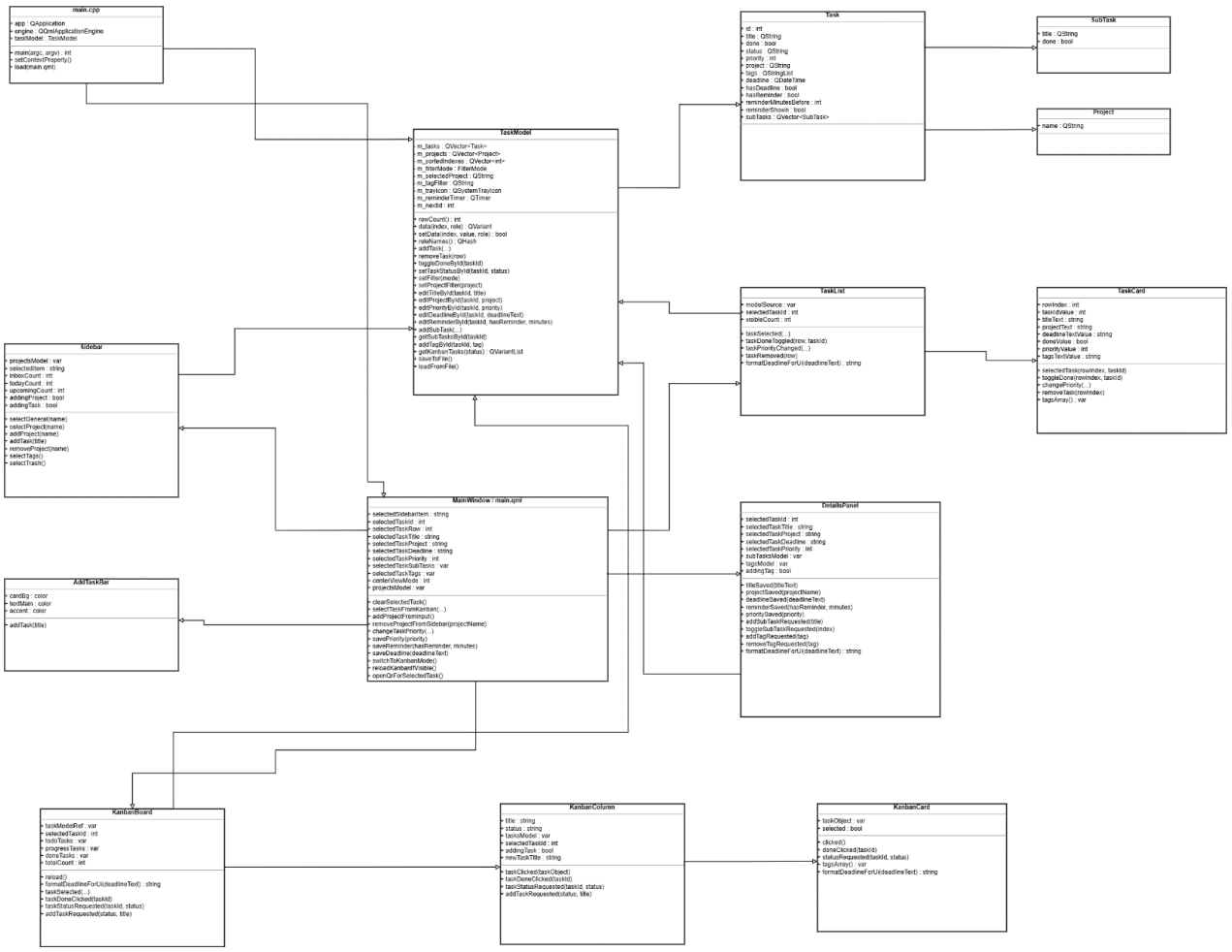



Рисунок 13 – UML-диаграмма классов приложения

В Qt Quick для отображения данных применяется подход model-view-delegate: модель хранит данные, представление отвечает за их размещение на экране, а делегат определяет внешний вид отдельного элемента.

Model:

− TaskModel.h – описание класса модели, структур задач, проектов и подзадач, а также ролей, через которые данные передаются в QML;

− TaskModel.cpp – реализация методов добавления, удаления, редактирования, фильтрации, сортировки, сохранения задач, обработки напоминаний и генерации QR-кода;

− main.cpp – запуск приложения, создание объекта модели и передача его в QML-интерфейс.

ViewModel:

Модель представления является связующим уровнем между внутренней логикой и визуальной частью приложения. В данном проекте она не выделена в отдельный файл, как это часто бывает в веб-приложениях, а реализуется через методы и свойства TaskModel, доступные из QML.

− передача taskModel из C++ в QML через контекст приложения;

− роли модели, через которые интерфейс получает свойства задач;

− методы изменения задачи, вызываемые из QML;

− сигналы обновления данных после изменений;

− функции фильтрации и получения задач для разных режимов отображения.

View:

− main.qml – главное окно приложения, объединяющее визуальные компоненты и связывающее их с моделью;

− Sidebar.qml – боковая панель навигации;

− TaskList.qml – список задач;

− TaskCard.qml – карточка отдельной задачи;

− DetailsPanel.qml – панель подробной информации;

− AddTaskBar.qml – элемент быстрого добавления задачи;

− KanbanBoard.qml – общий компонент Kanban-доски;

− KanbanColumn.qml – колонка Kanban-доски;

− KanbanCard.qml – карточка задачи внутри Kanban-доски.

Components:

− Sidebar.qml – навигация по разделам и проектам;

− TaskList.qml – вывод задач списком;

− TaskCard.qml – отображение отдельной задачи в списке;

− DetailsPanel.qml – просмотр и редактирование выбранной задачи;

− AddTaskBar.qml – быстрое добавление новой задачи;

− KanbanBoard.qml – отображение задач в формате Kanban;

− KanbanColumn.qml – колонка Kanban-доски;

− KanbanCard.qml – карточка задачи в Kanban-доске.

Рассмотрим подробнее реализацию основных алгоритмов приложения.

Точкой входа в программу является файл main.cpp, представлена на листинге 1. В нём создаётся объект приложения, запускается QML-движок, создаётся экземпляр модели задач и передаётся в QML под именем taskModel [8]. После этого загружается основной файл интерфейса main.qml.

Листинг 1 – Подключение модели задач к QML

```
int main(int argc, char *argv[])
{
    QQuickStyle::setStyle("Basic");

    QApplication app(argc, argv);

    QQmlApplicationEngine engine;

    TaskModel taskModel;
    engine.rootContext()->setContextProperty("taskModel",
&taskModel);

    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));

    if (engine.rootObjects().isEmpty())
        return -1;

    return app.exec();
}
```

В данном фрагменте ключевым является вызов setContextProperty. Он делает объект taskModel доступным в QML-файлах. После этого интерфейс может вызывать методы добавления задач, редактирования, фильтрации, сортировки, работы с проектами, подзадачами и другими функциями.

Основная структура данных описана в файле taskmodel.h, представлена на листинге 2. В проекте задача представлена структурой Task, которая содержит идентификатор, название, признак выполнения, статус, приоритет, проект, список тегов, дедлайн, параметры напоминания и список подзадач. Также отдельно описаны структуры Project и SubTask.

Листинг 2 – Структуры данных задачи, проекта и подзадачи

```
struct SubTask{
    QString title;
    bool done;
};
struct Project{
    QString name;
};
struct Task{
    int id;
    QString title;
    bool done;
    QString status = "todo";
    int priority;
    QString project;
    QStringList tags;

    QDateTime deadline;
    bool hasDeadline;
    bool hasReminder = false;
    int reminderMinutesBefore = 0;
    bool reminderShown = false;

    QVector<SubTask> subTasks;
};
```

Такая структура позволяет хранить не только название задачи, но и дополнительные параметры, которые нужны для полноценной работы приложения. Например, поле status используется для Kanban-доски, priority – для сортировки по важности, deadline – для сроков выполнения, а subTasks и tags – для более подробного описания задачи.

Для того чтобы QML мог получать отдельные свойства задачи, в модели используются роли, представлен на листинг 3. Каждая роль соответствует определённому полю задачи: названию, статусу, приоритету, сроку выполнения, проекту, тегам и другим данным. В файле taskmodel.cpp метод roleNames() задаёт имена ролей, которые затем используются в QML-компонентах.

Листинг 3 – Роли модели задач

```
QHash<int, QByteArray> TaskModel::roleNames() const{
    QHash<int, QByteArray> roles;
    roles[IdRole] = "taskId";
    roles[TitleRole] = "title";
```

```
    roles[DoneRole] = "done";
    roles[StatusRole] = "status";
    roles[PriorityRole] = "priority";
    roles[DeadlineRole] = "deadlineText";
    roles[HasDeadlineRole] = "hasDeadline";
    roles[HasReminderRole] = "hasReminder";
    roles[ReminderMinutesBeforeRole] = "reminderMinutesBefore";
    roles[ReminderShownRole] = "reminderShown";
    roles[OverdueRole] = "isOverdue";
    roles[ProjectRole] = "project";
    roles[SubTasksTextRole] = "subTasksText";
    roles[TagsRole] = "tagsText";
    return roles;
}
```

Благодаря этому QML-компоненты могут обращаться к данным задачи через понятные имена, например title, project, priority, deadlineText или tagsText. Это упрощает отображение задач в списке, карточках и Kanban-доске.

Создание новой задачи выполняется методом addTask, представлена на листинг 4. Перед добавлением проверяется, чтобы название задачи не было пустым. Затем создаётся объект Task, ему присваивается уникальный идентификатор, название, приоритет, проект и стандартный статус todo. Если пользователь указал дедлайн, строка с датой преобразуется в QDateTime. После добавления задача сохраняется в общий список, выполняется сортировка, обновляются счётчики боковой панели и данные записываются в файл.

Листинг 4 – Метод добавления задачи

```
void TaskModel::addTask(const QString &title,
                        int priority,
                        const QString &project,
                        const QString &deadlineText,
                        bool hasDeadLine)
{
    if (title.trimmed().isEmpty())
        return;

    beginResetModel();

    Task task;
    task.id = m_nextId++;
    task.title = title;
    task.done = false;
    task.status = "todo";
```

```
        task.priority = priority;
        task.project = project;

        QDateTime parsedDeadline =
            QDateTime::fromString(deadlineText, "yyyy-MM-dd
HH:mm");

        if (hasDeadLine && parsedDeadline.isValid()){
            task.deadline = parsedDeadline;
            task.hasDeadline = true;
        } else {
            task.deadline = QDateTime();
            task.hasDeadline = false;
        }

        m_tasks.append(task);
        sortTask();
        endResetModel();

        notifySidebarCountsChanged();
        saveTaskToDb(task);
    }
```

Данный метод показывает общий принцип работы модели: сначала изменяются данные, затем обновляется представление модели, после чего новое состояние сохраняется. Такой подход используется и в других действиях, например при удалении задачи, изменении приоритета или дедлайна.

Для изменения выполнения задачи используется метод toggleDoneById, рассмотрена на листинг 5. Он находит задачу по идентификатору и переключает её состояние. Если задача отмечена выполненной, ей присваивается статус done, иначе статус возвращается к todo. После этого список задач пересортировывается и сохраняется.

Листинг 5 – Переключение состояния выполнения задачи

```
void TaskModel::toggleDoneById(int taskId){
    int sourceRow = findTaskIndexById(taskId);
    if (sourceRow == -1)
        return;

    beginResetModel();

    m_tasks[sourceRow].done = !m_tasks[sourceRow].done;

    if (m_tasks[sourceRow].done){
```

```
            m_tasks[sourceRow].status = "done";
        } else {
            m_tasks[sourceRow].status = "todo";
        }

        sortTask();
        endResetModel();

        notifySidebarCountsChanged();
        saveTaskToDb(m_tasks[sourceRow]);
    }
```

Такой алгоритм нужен для синхронизации обычного списка задач и Kanban-доски. Если пользователь отмечает задачу выполненной, она должна измениться не только визуально, но и перейти в соответствующий статус.

Для отображения задач в виде Kanban-доски используется статус задачи, показана на листинге 6. Метод setTaskStatusById изменяет статус конкретной задачи. В программе предусмотрены три состояния: todo, progress и done. Если передан другой статус, метод не выполняет изменение.

Листинг 6 – Изменение статуса задачи для Kanban-доски

```
    void TaskModel::setTaskStatusById(int taskId, const QString
&status){
        if (status != "todo" && status != "progress" && status !=
"done")
            return;

        int sourceRow = findTaskIndexById(taskId);
        if(sourceRow == -1)
            return;

        beginResetModel();

        m_tasks[sourceRow].status = status;
        m_tasks[sourceRow].done = (status == "done");

        sortTask();
        endResetModel();

        saveTaskToDb(m_tasks[sourceRow]);
    }
```

Для получения задач конкретной колонки используется метод getKanbanTasks, представлена на листинг 7. Он проходит по отсортированному

списку задач, проверяет их статус и дополнительно учитывает текущие фильтры. В результате в QML передаётся список задач, который уже можно отобразить в нужной колонке Kanban-доски.

Листинг 7 – Получение задач для колонки Kanban

```
    QVariantList TaskModel::getKanbanTasks(const QString &status)
const{
        QVariantList result;

        if(status != "todo" && status != "progress" && status !=
"done")
            return result;

        for(int sortedIndex : m_sortedIndexes){
            if(sortedIndex < 0 || sortedIndex >= m_tasks.size())
                continue;
            const Task &task = m_tasks[sortedIndex];
            if(task.status != status)
                continue;

            if (!taskMatchesCurrentFilters(task))
                continue;
            QVariantMap taskMap;
            taskMap["taskId"] = task.id;
            taskMap["title"] = task.title;
            taskMap["done"] = task.done;
            taskMap["status"] = task.status;
            taskMap["priority"] = task.priority;
            taskMap["project"] = task.project;
            taskMap["hasDeadline"] = task.hasDeadline;
            taskMap["deadlineText"] =
                task.hasDeadline ? task.deadline.toString("yyyy-MM-
dd HH:mm") : "";

            result.append(taskMap);
        }

        return result;
    }
```

Проекты используются для группировки задач. Метод addProject проверяет введённое название: оно не должно быть пустым и не должно повторять уже существующий проект. После добавления проекта обновляются данные боковой панели и выполняется сохранение.

Листинг 8 – Добавление проекта

```
void TaskModel::addProject(const QString &name){
    QString trimmedName = name.trimmed();
    if (trimmedName.isEmpty())
        return;

    for (const Project &project : m_projects){
        if (project.name == trimmedName)
            return;
    }

    Project project;
    project.name = trimmedName;
    m_projects.append(project);

    notifySidebarCountsChanged();
    saveProjectToDb(project);
}
```

В интерфейсе предусмотрены два режима отображения задач: обычный список и Kanban-доска, представлен на листинге 9. В main.qml для этого используется StackLayout, который показывает нужный компонент в зависимости от значения centerViewMode. В обычном режиме отображается TaskList, а в режиме Kanban – KanbanBoard.

Листинг 9 – Переключение режимов отображения

```
StackLayout {
    id: centerStack

    Layout.fillWidth: true
    Layout.fillHeight: true

    currentIndex: mainWindow.centerViewMode

    TaskList {
        id: taskList
        modelSource: taskModel
        selectedTaskId: mainWindow.selectedTaskId
    }

    KanbanBoard {
        id: kanbanBoard
        taskModelRef: taskModel
        selectedTaskId: mainWindow.selectedTaskId
    }
}
```

1.6.4 Основные алгоритмы работы приложения

В данном разделе были рассмотрены алгоритмы работы приложения. Для каждого из них построена UML-диаграмма деятельности, отражающая порядок выполняемых шагов и возможные ответления в зависимости от условий.

Первым алгоритмом рассмотрим создание новой задачи. Когда пользователь вводит название и подтверждает добавление, программа убеждается, что поле не осталось пустым, если название указано, формируется объект задачи с уникальным индетификатором. Потом обрабатывается дата дедлайна: если она указана и корректна, то она привязывается к задаче, иначе поле остаётся незаполненным. После этого задача попадает в общий список, далее список пересортировывается, боковая панель обновляет свои счетчики, а запись уходит в базу данных.

Последовательность описанных действий представлена на рисунке 14.

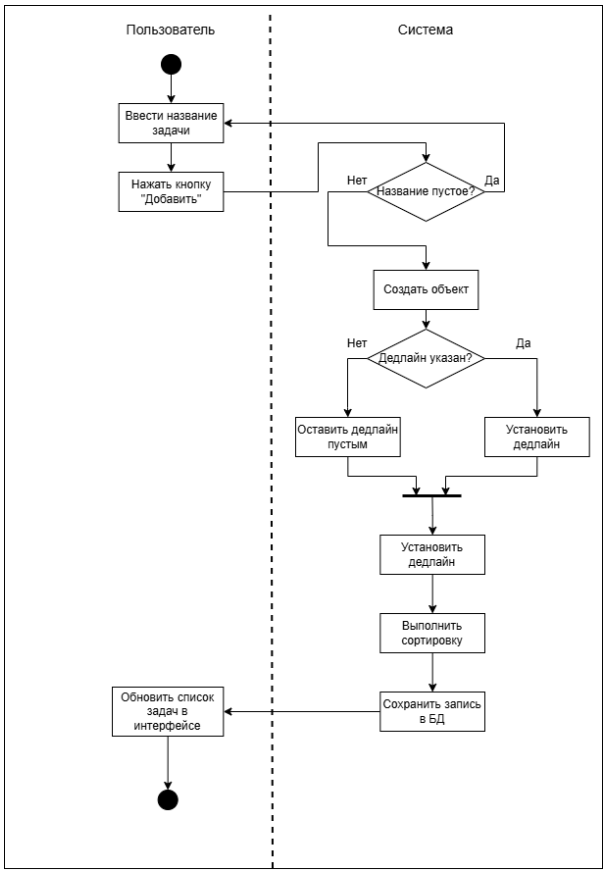



Рисунок 14 – UML-диаграмма деятельности последовательности добавления задачи

Диаграмма деятельности фильтрации задач по разделам представлена на рисунке 15.

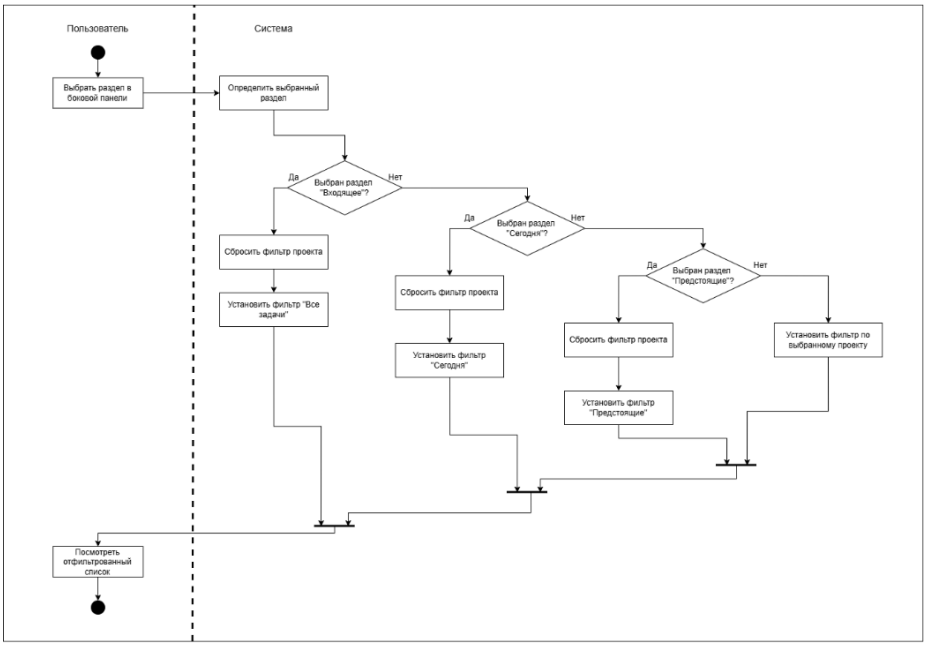



Рисунок 15 – UML-диаграмма деятельности фильтрации задач

Диаграмма деятельности изменения статуса задачи представлена на рисунке 16.

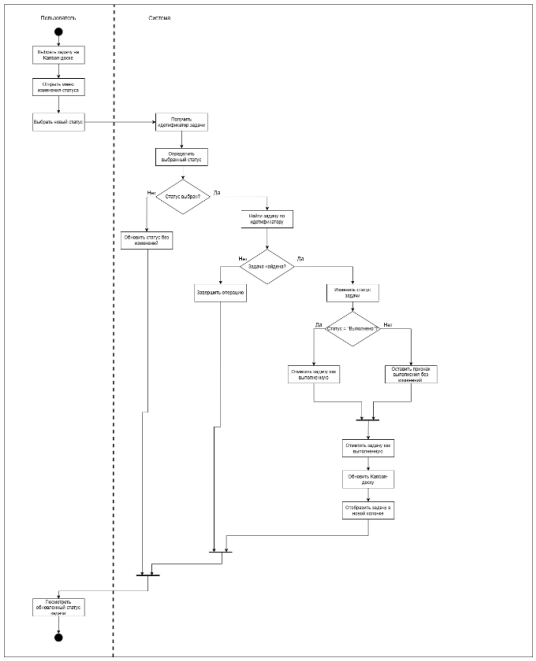


Рисунок 16 – UML-диаграмма деятельности изменения статуса задачи

Второй рассмотренный алгоритм – генерация QR-кода. Здесь логика строится на двух последовательных проверках, прежде чем дойти до генерации. Сначала приложение смотрит, выбрал-ли пользователь какую-либо задачу, если нет, то на экране появится подсказка выбрать задачу. Если задача выбрана, проверяется наличие дедлайна, поскольку именно он формирует содержимое QR-кода. Если дедлайн остутствует, то пользователь получает сообщение, что нужно добавить дедлайн к задаче. Когда обе проверки пройдены, генерируется изображение QR-кода, которое сохраняется и открывается в отдельном окне

Последовательность описанных действий представлена на рисунке 17.

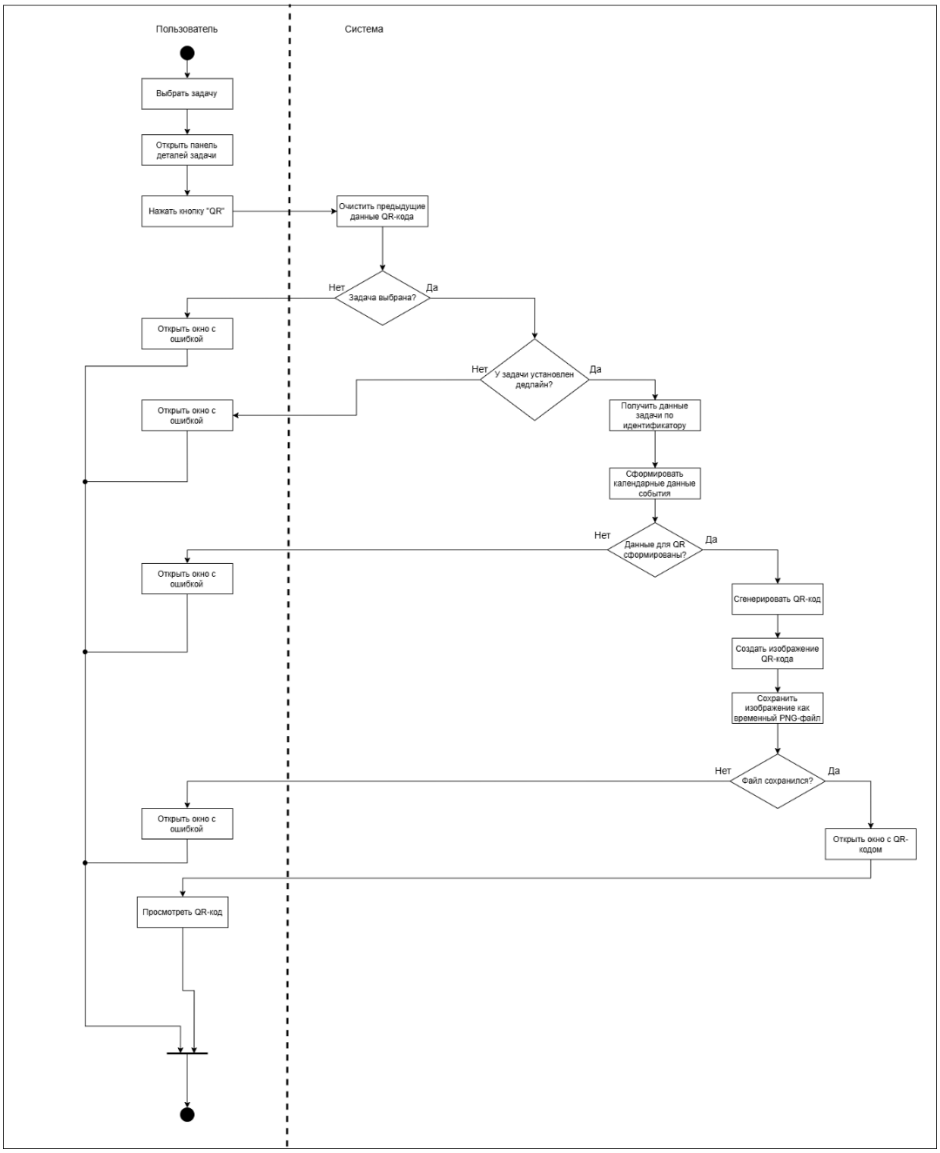



Рисунок 17 – UML-диаграмма деятельности генерации QR-кода.

Третий алгоритм – работа напоминаний. С момента запуска приложения в фоне работает таймер, который переодически обходит список задач. Для каждой задачи он проверяет 3 вещи: есть-ли деделайн, включено-ли напоминание и не показывалось ли оно уже. Если в условия сходятся и текущее время достигло расчетного момента напоминая, то в системном трее появляется уведомление с названием задачи. После этого задача помечается как уведомленная, чтобы одно и то же напоминание не всплывало повторно.

Последовательность описанных действий представлена на рисунке 18.

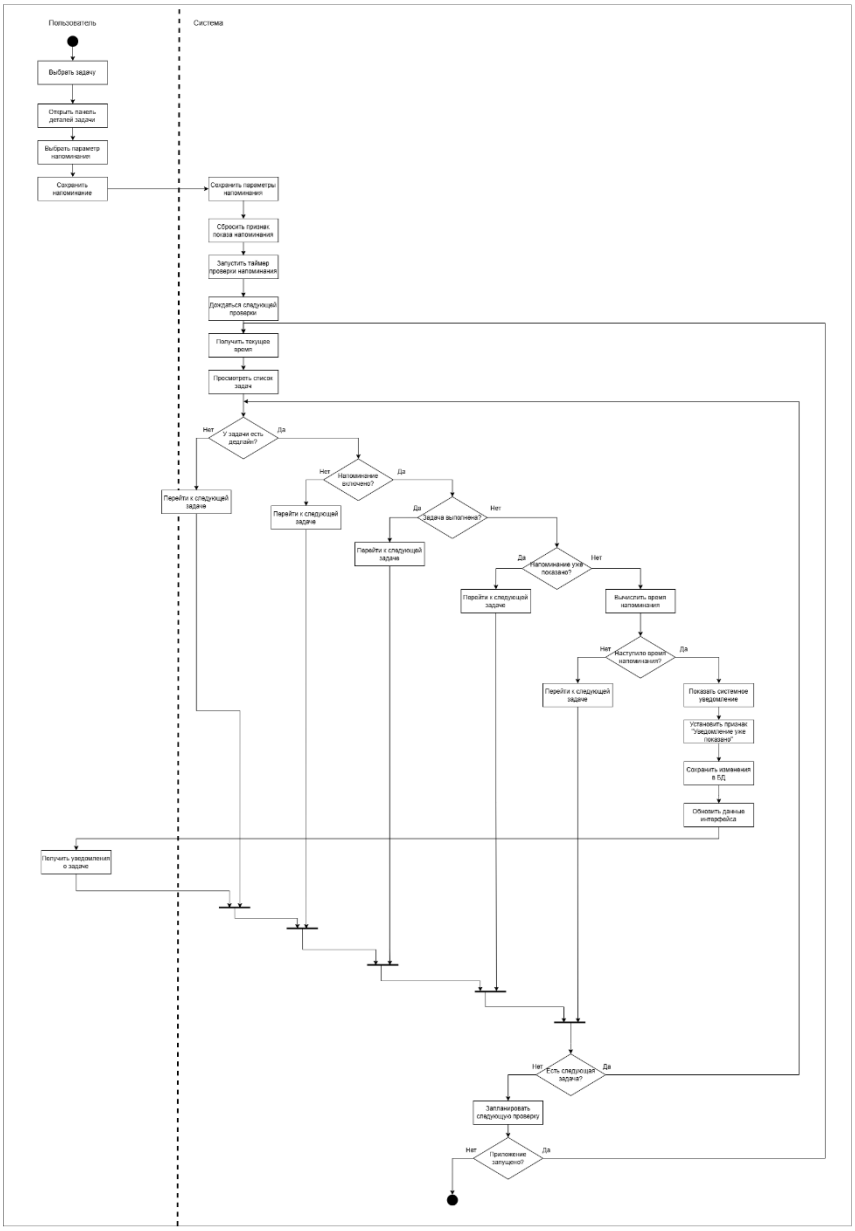



Рисунок 18 – UML-диаграмма деятельности системы напоминаний.

## 1.7 Тестирование

Для проверки работоспособности клиентского приложения «Task Manager» необходимо провести его тестирование. Для тестирования были составлены тест-кейсы.

Тест-кейсы составлены для следующих действий:

− добавление задачи;

− удаление задачи;

− изменение статуса выполнения задачи;

− изменение приоритета задачи;

− создание проекта;

− фильтрация задач по разделам;

− работа с Kanban-доской;

− установка дедлайна;

− установка напоминания.

1.7.1 Добавление задачи

В таблице 2 представлен тест-кейс добавления задачи

Таблица 2 – Тест-кейс добавление задачи

| Шаги | Данные | Ожидаемый результат | Фактический результат |
|---|---|---|---|
| Нажать на кнопку «Новая задача» в боковой панели | | Открывается поле ввода названия задачи | Совпадает с ожидаемым результатом |
| Ввести название задачи и нажать кнопку «Добавить» | Название: «Подготовить отчёт» | В списке задач появляется новая задача с указанным названием | Совпадает с ожидаемым результатом |
| Попытаться добавить задачу с пустым названием | | Задача не создаётся, кнопка добавления недоступна или действие не выполняется | Совпадает с ожидаемым результатом |

Данный тест-кейс представлен для проверки функции добавления новой задачи в приложении. Пользователь должен иметь возможность создать задачу

только в том случае, если поле названия заполнено. Если поле пустое, приложение не должно добавлять задачу в список, так как задача без названия не несёт полезной информации для пользователя.

1.7.2 Изменение статуса задачи

В таблице 3 представлен тест-кейс изменения статуса задачи

Таблица 3 – Тест-кейс изменения статуса задачи

| Шаги | Данные | Ожидаемый результат | Фактический результат |
|---|---|---|---|
| Выбрать задачу из списка | Задача: «Подготовить отчёт» | Открывается карточка задачи | Совпадает с ожидаемым результатом |
| Нажать на чекбокс выполнения | Статус: не выполнена | Задача помечается как выполненная | Совпадает с ожидаемым результатом |
| Повторно нажать на чекбокс | Статус: выполнена | Задача снова становится активной | Совпадает с ожидаемым результатом |
| Проверить отображение задачи в Kanban-доске | Статус: выполнена | Задача переносится в колонку «Готово» | Совпадает с ожидаемым результатом |

В приложении пользователь может отмечать задачи как выполненные и при необходимости возвращать их обратно в активное состояние. Это важная функция для планировщика задач, так как она позволяет отслеживать прогресс выполнения работы.

1.7.3 Просмотр и выбор задачи

В таблице 4 представлен тест-кейс просмотра и выбора задач.

Таблица 4 – Тест-кейс просмотра и выбора задач.

| Шаги | Данные | Ожидаемый результат | Фактический результат |
|---|---|---|---|
| Запустить приложение | | Открывается главное окно приложения со списком задач и боковой панелью | Совпадает с ожидаемым результатом |
| Нажать на задачу в списке | Задача: «Подготовить отчёт» | В правой части окна открылась панель с подробной информацией о выбранной задаче | Совпадает с ожидаемым результатом |
| Проверить отображение названия выбранной задачи | Название: «Подготовить отчёт» | В панели свойств отображается название выбранной задачи | Совпадает с ожидаемым результатом |
| Закрыть панель выбранной задачи | Нажать кнопку закрытия | Выбранная задача сбрасывается, панель возвращается в исходное состояние | Совпадает с ожидаемым результатом |

Этот тест-кейс проверяет работу механизма выбора задачи. После нажатия на задачу пользователь должен увидеть подробную информацию о ней в правой панели. Это позволяет просматривать и изменять параметры задачи: название, проект, дедлайн, приоритет, напоминание, подзадачи и теги.

1.7.4 Изменение статуса приоритета задачи

В таблице 5 представлен тест-кейс изменение статуса приоритета задачи.

Таблица 5 – Тест-кейс изменение статуса приоритета задачи.

| Шаги | Данные | Ожидаемый результат | Фактический результат |
|---|---|---|---|
| Выбрать задачу из списка | Задача: «Подготовить отчёт» | Открылась панель с информацией о выбранной задаче | Совпадает с ожидаемым результатом |
| Нажать на чекбокс выполнения задачи | Статус: не выполнена | Задача отмечена как выполненная, появился значок выполнения | Совпадает с ожидаемым результатом |
| Повторно нажать на чекбокс выполнения задачи | Статус: выполнена | Задача снова стала невыполненной | Совпадает с ожидаемым результатом |
| Проверить задачу в Kanban-доске | Статус: выполнена | Задача отображается в колонке завершённых задач | Совпадает с ожидаемым результатом |

Тест-кейс предназначен для проверки изменения состояния задачи. Пользователь должен иметь возможность отметить задачу как выполненную, когда работа по ней завершена. После этого задача должна визуально измениться: появляется отметка выполнения, а сама задача считается завершённой.

1.7.5 Создание проекта

В таблице 6 представлен тест-кейс изменение статуса приоритета задачи.

Таблица 6 – Тест-кейс создания проекта.

| Шаги | Данные | Ожидаемый результат | Фактический результат |
|---|---|---|---|
| Нажать кнопку добавления проекта в боковой панели | | На экране появилось поле для ввода названия проекта | Совпадает с ожидаемым результатом |
| Ввести название проекта и подтвердить создание | Название проекта: «Учёба» | Новый проект появился в списке проектов | Совпадает с ожидаемым результатом |
| Выбрать созданный проект | Проект: «Учёба» | В центральной части окна отображаются задачи выбранного проекта | Совпадает с ожидаемым результатом |
| Попытаться создать проект с пустым названием | Название проекта: пустое поле | Проект не создаётся | Совпадает с ожидаемым результатом |

Этот тест-кейс предназначен для проверки создания проектов. Проекты позволяют пользователю группировать задачи по определённым направлениям, например, «Учёба», «Работа» или «Личное». После создания новый проект должен появиться в боковой панели приложения.

1.7.5 Установка дедлайна задачи

В таблице 7 представлен тест-кейс изменение статуса приоритета задачи.

Таблица 7 – Тест-кейс установки дедлайна задачи.

| Шаги | Данные | Ожидаемый результат | Фактический результат |
|---|---|---|---|
| Выбрать задачу из списка | | Открылась панель свойств задачи | Совпадает с ожидаемым результатом |
| Нажать на поле «Дедлайн» | Название проекта: «Учёба» | Открылась панель свойств задачи | Совпадает с ожидаемым результатом |
| Выбрать дату и время дедлайна | Проект: «Учёба» | У задачи установлен выбранный дедлайн | Совпадает с ожидаемым результатом |
| Проверить отображение дедлайна в карточке задачи | Название проекта: пустое поле | В карточке задачи отображается выбранная дата и время | Совпадает с ожидаемым результатом |
| Очистить дедлайн | Нажать кнопку очистки | Дедлайн удалён, отображается значение «Не установлен» | Совпадает с ожидаемым результатом |

Тест-кейс проверяет функцию установки дедлайна. Дедлайн позволяет пользователю указать дату и время, к которым задача должна быть выполнена. После установки дедлайна эта информация должна отображаться в свойствах задачи и в её карточке.

1.7.6 Установка напоминания

В таблице 8 представлен тест-кейс изменение статуса приоритета задачи.

Таблица 8 – Тест-кейс установки напоминания

| Шаги | Данные | Ожидаемый результат | Фактический результат |
|---|---|---|---|
| Выбрать задачу с установленным дедлайном | Задача: «Подготовить отчёт» | Открылась панель свойств задачи | Совпадает с ожидаемым результатом |
| Нажать на поле «Напоминание» | | Открылся список вариантов напоминания | Совпадает с ожидаемым результатом |
| Выбрать вариант «В момент» | Напоминание: в момент дедлайна | У задачи установлено напоминание в момент дедлайна | Совпадает с ожидаемым результатом |
| Выбрать вариант «За 10 минут» | Напоминание: за 10 минут | У задачи установлено напоминание за 10 минут до дедлайна | Совпадает с ожидаемым результатом |
| Выбрать вариант «За 1 час» | Напоминание: за 1 час | У задачи установлено напоминание за 1 час до дедлайна | Совпадает с ожидаемым результатом |
| Отключить напоминание | Значение: «Нет» | Напоминание удалено | Совпадает с ожидаемым результатом |

Этот тест-кейс проверяет работу напоминаний. Напоминание помогает пользователю не забыть о задаче и заранее получить уведомление перед наступлением дедлайна. Для корректной работы напоминания у задачи должен быть установлен срок выполнения.

1.7.7 Работа с Kanban-доской

В таблице 9 представлен тест-кейс работы Kanban-доски

Таблица 9 – Тест-кейс работы с Kanban-доской

| Шаги | Данные | Ожидаемый результат | Фактический результат |
|---|---|---|---|
| Переключиться в режим Kanban-доски | | На экране отображаются колонки Kanban-доски | Совпадает с ожидаемым результатом |
| Проверить наличие колонки «К выполнению» | | Отображается колонка «К выполнению» | Совпадает с ожидаемым результатом |
| Проверить наличие колонки «В процессе» | | Отображается колонка «В процессе» | Совпадает с ожидаемым результатом |
| Проверить наличие колонки завершённых задач | | Отображается колонка для выполненных задач | Совпадает с ожидаемым результатом |
| Изменить статус задачи на «В процессе» | Задача: «Подготовить отчёт» | Отображается колонка для выполненных задач | Совпадает с ожидаемым результатом |
| Изменить статус задачи на «Готово» | Задача: «Подготовить отчёт» | Напоминание удалено | Совпадает с ожидаемым результатом |

Данный тест-кейс проверяет работу Kanban-доски. Kanban-доска нужна для визуального разделения задач по стадиям выполнения. Пользователь может видеть, какие задачи ещё нужно выполнить, какие находятся в процессе, а какие уже завершены.

## 1.8 Руководство пользователя

### 1.8.1 Результат работы приложения

Разработанное приложение «Task Manager» представляет собой настольную систему управления задачами. Приложение позволяет пользователю создавать задачи, распределять их по проектам, назначать сроки выполнения, устанавливать приоритеты, добавлять подзадачи и теги, а также контролировать выполнение задач с помощью списка и Kanban-доски.

Главное окно приложения состоит из нескольких логических областей. В левой части расположена боковая панель навигации, в центральной части отображается список задач или Kanban-доска, а в правой части находится панель подробной информации о выбранной задаче. В качестве основного окна используется QML-элемент ApplicationWindow, который может применяться как корневой элемент приложения и позволяет управлять внешним видом и компоновкой окна из QML-кода. Главное меню изображен на рисунке 19.

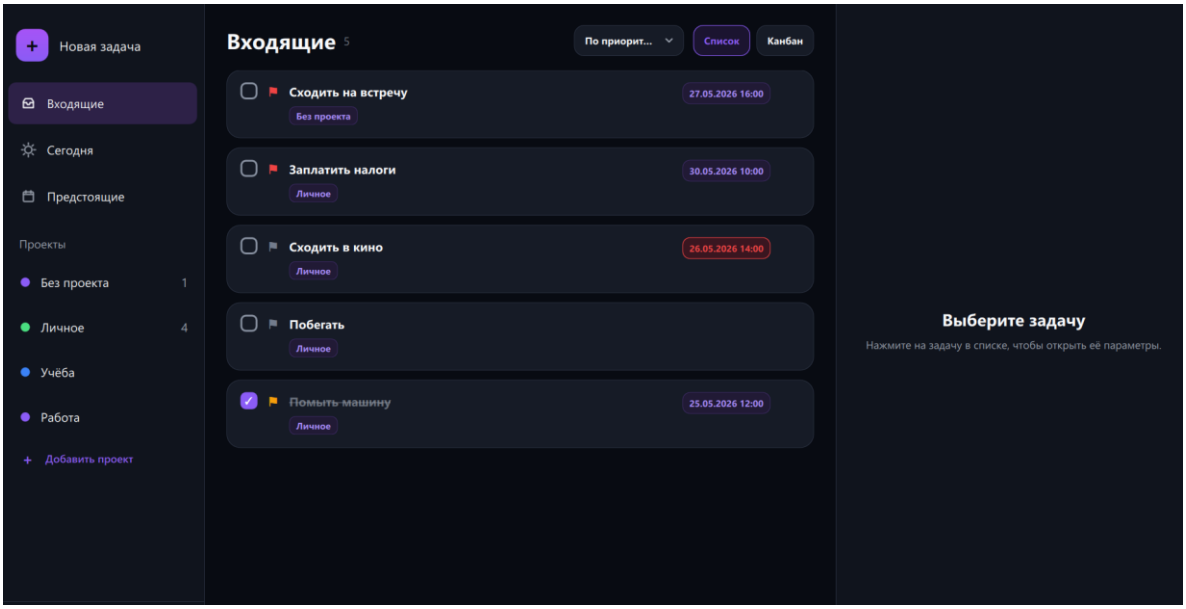



Рисунок 19 – Главная меню

В боковой панели располагаются основные разделы приложения. Пользователь может перейти к входящим задачам, задачам на сегодня, предстоящим задачам, тегам, корзине, а также выбрать один из созданных

проектов. Кроме того, через боковую панель можно создать новую задачу или добавить новый проект. Боковая панель изображена на рисунке 20.

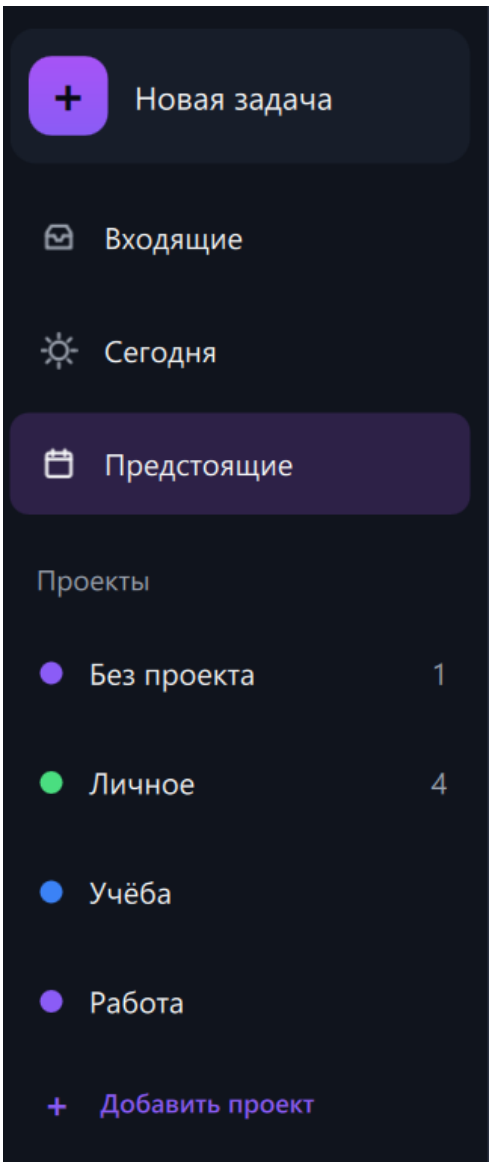



Рисунок 20 – Боковая панель

Боковая панель позволяет фильтровать задачи по разделам. При выборе раздела «Сегодня» в центральной области отображаются только те задачи, срок выполнения которых приходится на текущий день. Раздел «Предстоящие» показывает задачи с дедлайном в ближайшие дни. При выборе раздела «Просроченные» отображаются задачи, срок которых уже истёк. Фильтрация по разделам изображена на рисунке 21.

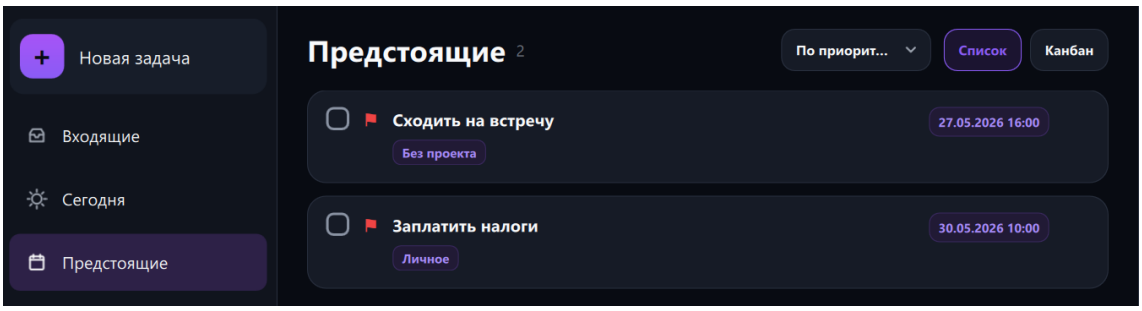



Рисунок 21 – Фильтрация задач по разделам

Для добавления новой задачи пользователь вводит название задачи в поле добавления и нажимает кнопку «Добавить» или клавишу Enter. После этого задача появляется в общем списке и становится доступной для дальнейшего редактирования. Добавление задач изображено на картинке 22.

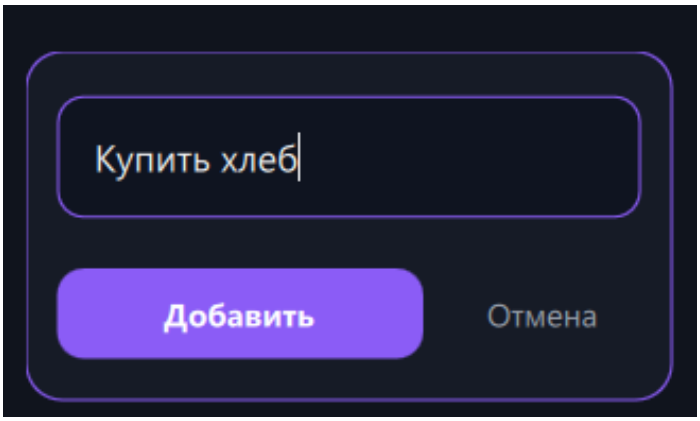



Рисунок 22 – Добавление новой задачи

Задачи в основном режиме отображаются в виде списка карточек. В карточке отображаются название задачи, проект, срок выполнения, приоритет, состояние выполнения и теги. Пользователь может выбрать задачу для просмотра подробной информации, отметить её как выполненную или изменить её параметры. Список задач изображен на рисунке 23.

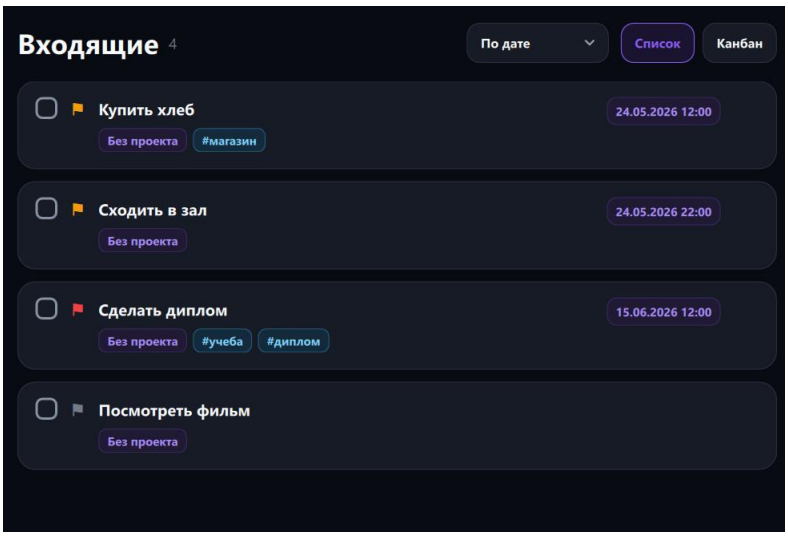



Рисунок 23 – Список задач

При выборе задачи в правой части окна открывается панель подробной информации. В ней пользователь может изменить название задачи, выбрать проект, установить срок выполнения, задать приоритет, включить напоминание, добавить подзадачи и теги. Правая панель изображена на рисунке 24.

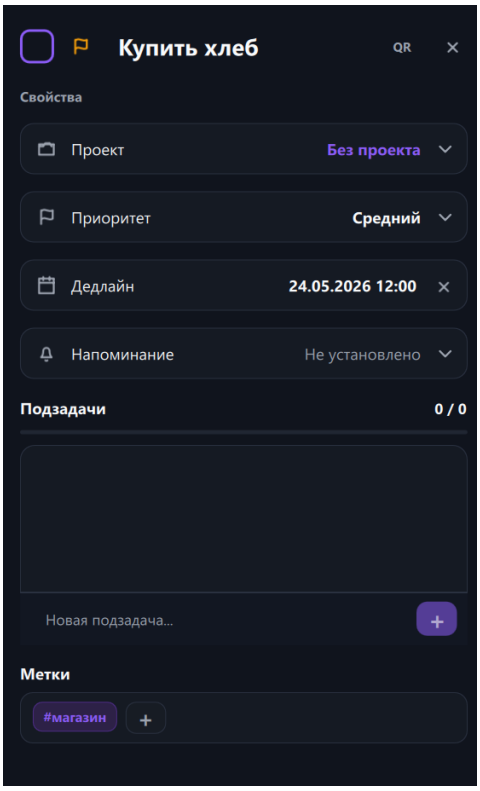



Рисунок 24 – Панель подробной информации о задаче

В приложении также реализована работа с подзадачами и тегами. Подзадачи позволяют разбить задачу на несколько небольших действий, а теги помогают дополнительно группировать задачи по смыслу. Подзадачи и теги изображены на рисунке 25.

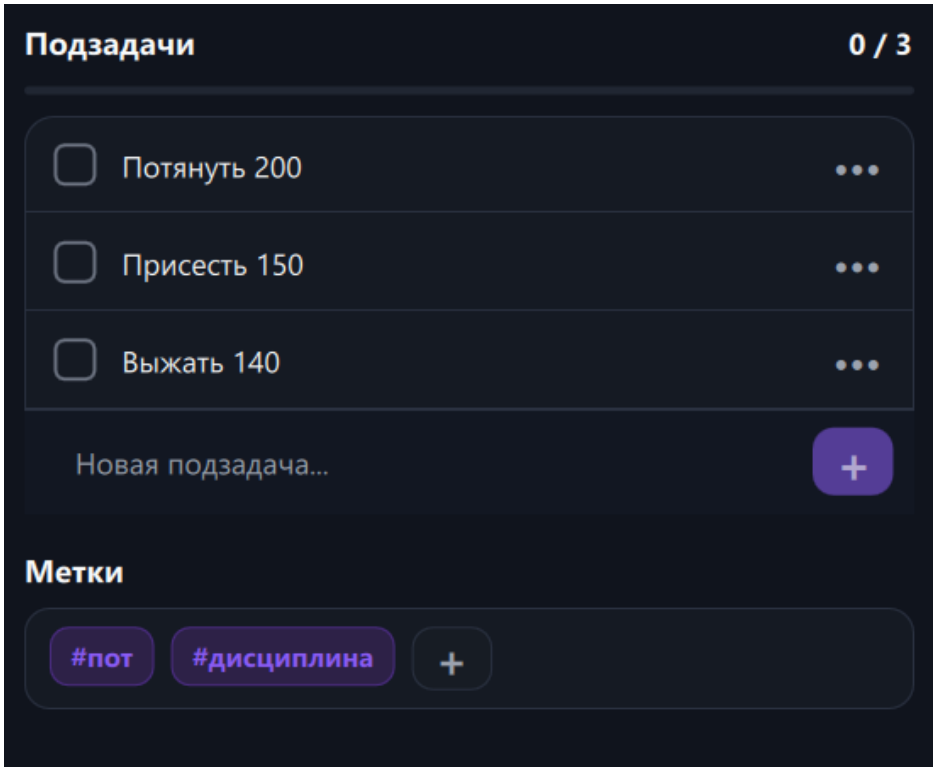



Рисунок 25 – Подзадачи и теги выбранной задачи

Для задач, у которых установлен срок выполнения, реализована функция создания QR-кода. Перед генерацией приложение проверяет, выбрана ли задача и задан ли для неё дедлайн. Если задача не выбрана, пользователю выводится сообщение «Сначала выберите задачу». Если дедлайн отсутствует, приложение сообщает о необходимости сначала установить срок выполнения. После успешной проверки вызывается функция генерации QR-кода, результат сохраняется в виде изображения и отображается пользователю в отдельном окне. QR-код может использоваться для быстрого добавления задачи в календарь или передачи информации о ней на другое устройство. На рисунке 26 показан QR-код задачи.

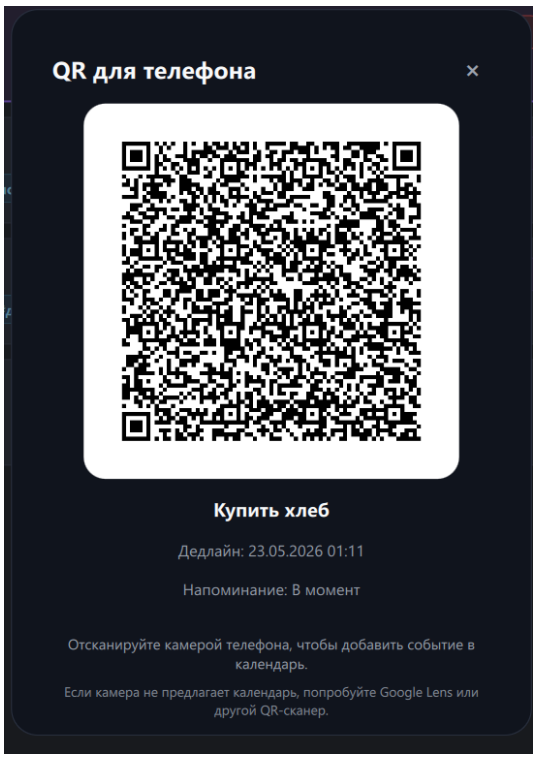



Рисунок 26 – QR-код задачи

При попытке сгенерировать QR-код без выбранной задачи приложение выводит сообщение «Сначала выберите задачу». Если задача выбрана, но дедлайн не установлен, отображается сообщение о необходимости сначала указать срок выполнения. Сообщения об ошибках изображены на рисунке 27.

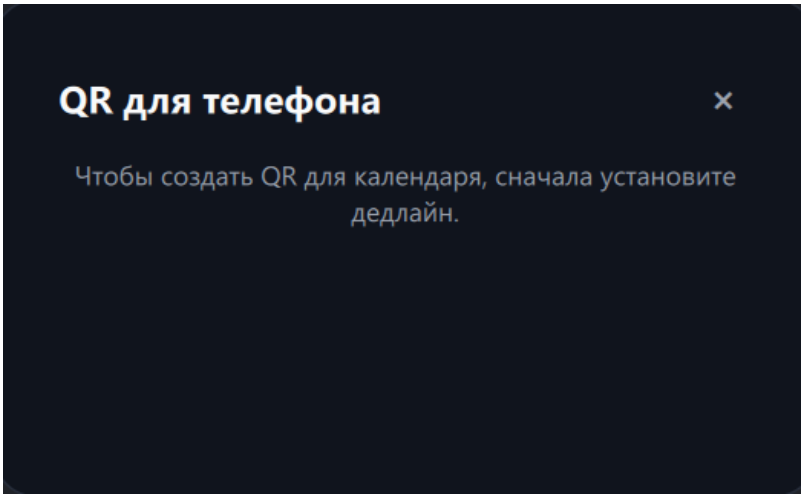



Рисунок 27 – Сообщение об ошибке при генерации QR-кода

В приложении, также реализован альтернативный режим отображения задач – Kanban-доска. Для переключения между режимами используется кнопка в верхней части центральной области окна. В режиме Kanban задачи распределяются по трём колонкам в зависимости от текущего статуса: «К выполнению», «В процессе» и «Готово». Пользователь может изменить статус задачи прямо из карточки на доске, после чего она автоматически перемещается в соответствующую колонку. Kanban-доска изображена на рисунке 28.

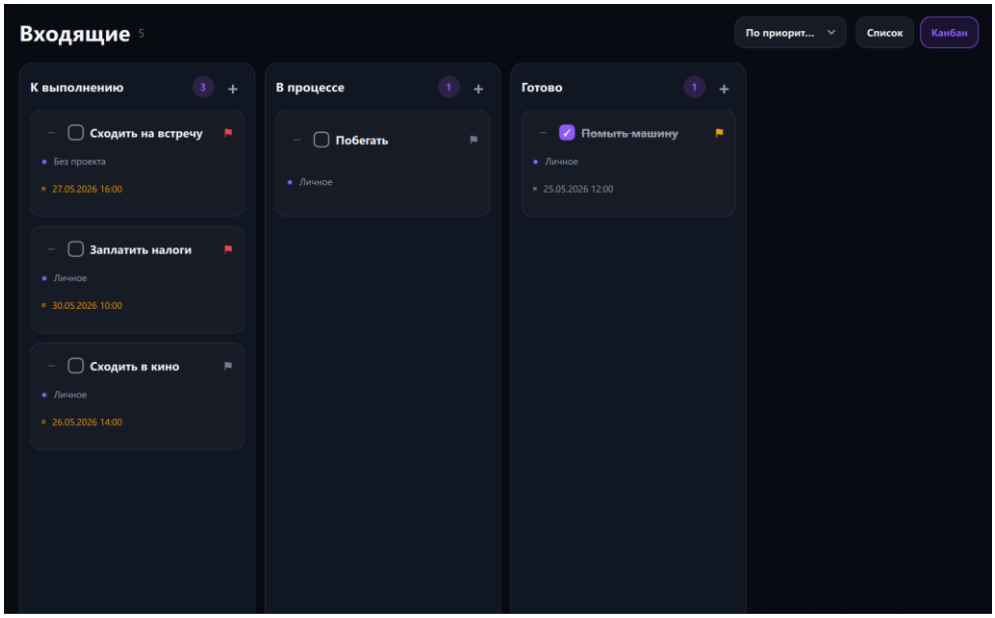



Рисунок 28 – Kanban-доска

## 2 Безопастность жизнедеятельности

Разработка настольного приложения предполагает длительную работу за персональным компьютером, поэтому соблюдение требований безопстности труда является неотъемлемой частью организации рабочего процесса. Несоблюдение этих требований может привести к ухудшению зрения, нарушению осанки, повышенной утомляемости и другим проблемам со здоровьем.

Освещённость рабочего места должна находится в диапозоне 300–500 лк. Недостаточное освещение вынуждает глаза работать в напряженном режиме, а избыточное создает блики на экране. Естественный свет должен падать преимущественно слева от рабочего стола. Источники света не должны распологаться непосредственно перед монитором или за спиной пользователя. Яркость бликов на экране не должна превышать 40 кд/м$^2$, яркость потолка – 200 кд/м$^2$. В качестве искуственного освещения предпочтительно использовать люминесцентные лампы с рассеивателями, обеспечивающие равномерное распределение света без прямого отражения на экране.

Монитор следует распологать на расстоянии 600–700 мм от глаз пользвателя, но не ближе 500 мм. Верхний край экрана должен находиться на уровне глаз или чуть ниже – это снижает нагрузку на шейный отдел позвоночника. Расстояние между боковыми поверхностями соседних мониторов должно быть не менее 1,2 м, а между рабочими столами – не менее 2 м.

Рабочее кресло должно быть регулируемым по высоте и углу наклона спинки. Правильная посадка предполагает, что бёдра расположены параллельно полу, а стопы опираются на него полностью. Сиденье и спинка стула должны быть покрыты не электризующимися материалами.

Температура воздуха в рабочем помещении должна поддрерживаться в диапозоне 19–23℃, относительная влажность – около 55%. Помещение необходимо регулярно проветривать. Непрерывная работа за компьютером не должна превышать одного часа, после чего рекомендуется делать перерыв

продолжительностью 1 –15 минут. При работе в ночное время продолжительность увеличивается на 30 %.

Все используемые устройства должны быть надлежащим образом подключены и заземлены. Правильное заземление исключает риск поражения электрическим током при косании корпуса оборудования. Не допускается использование поврежденных кабелей и розеток, а также размещение жидкостей вблизи работающей техники.

Сидячий образ работы является одним из факторов, способствующий развитию заболеваний позвоночника и суставов. При длительном нахождении в одной позе мышцы спины и шеи испытывают статическое напряжение, что со временем может привести к болевым ощущениям и нарушению осанки. Для снижения этого воздействия рекомендуется каждые 45–60 минут вставать с рабочего места и выполнять несложные урпражения: повороты и наклоны головы, круговые движения плечами и потягивания. Кроме того, важно следить за положением запястий при работе с клавиатурой и мышью, они не должны находится в согнутом или приподнятом положениии в течение длительного времени, так как это создаёт нагрузку на сухожилия и может привести к туннельному синдрому.

Соблюдение перечисленных требований существенно снизит негативное воздействие длительной работы за компьютером на здровье. Грамотная организация рабочего места, правильный режим труда и отдыха, контроль освещённсти и микроклимата в совокупности формируют условия, при которых продуктивная работа не наносит вреда здоровью пользователя. Эти меры особенно актуальны при разработке программного обеспечения, поскольку такая деятельност предполагает непрерывную работу за компьютером на протяжении нескольких часов подряд.

## Заключение

В ходе выполнения выпускной квалификационной работы было разработано проведено исследование предметной области управления задачами, в результате которого были проанализированы существующие решения, выявлены их достоинства и ограничения, а также определены требования к разрабатываемому приложению.

На основании проведённого анализа было разработано локальное приложение для ведения и контроля задач. В качестве языка программирования выбран C++, в качестве фреймворка Qt с использованием технологии QML для построения интерфейса. Для хранения данных применена встроенная реляционная база данных SQLite, что обеспечило надёжное сохранение информации между сеансами работы без зависимости от внешних сервисов.

Разработанное приложение реализует полный цикл работы с задачами от создания и редактирования до контроля выполнения через Kanban-доску и систему напоминаний. Дополнительно реализована функция генерации QR-кода, позволяющая отслеживать задачи вне рабочего места.

В ходе тестирования подтверждено соответсвие приложения требованиям, сформулированным в постановке задачи.

Задание на дипломное проектирование выполнено в полном объёме.

В дальнейшем приложение можеть быть доработано путём добавления экспорта задач в сторонние форматы, интеграции с календарными сервисами, а также реалзация многопользовательского режима для совместного введения проектов.

## Список использованных источников


1. Сервис управления задачами Todoist [Электронный ресурс]/Режим доступа: https://todoist.com, свободный. (Дата обращения 09.04.2026).
2. Сервис управления задачами Google Tasks [Электронный ресурс]/Режим доступа: https://tasks.google.com, свободный. (Дата обращения 09.04.2026).
3. Сервис управления задачами Trello [Электронный ресурс]/Режим доступа: https://trello.com, свободный. (Дата обращения 09.04.2026).
4. Документация QtSql и QSQLITE [Электронный ресурс]/Режим доступа: https://doc.qt.io/qt-6/qtsql-index.html, свободный. (Дата обращения 30.03.2026).
5. Документация SQLite [Электронный ресурс]/Режим доступа: https://www.sqlite.org/docs.html, свободный. (Дата обращения 01.04.2026).
6. Попов Е. Руководство по Qt и C++ [Электронный ресурс]/Режим доступа: https://metanit.com/cpp/qt, свободный. (Дата обращения 25.03.2026).
7. Попов Е. Введение в QML и Qt Quick [Электронный ресурс]/Режим доступа: https://metanit.com/cpp/qt/4.1.php, свободный. (Дата обращения 26.03.2026).
8. Попов Е. Интеграция QML с C++ [Электронный ресурс]/Режим доступа: https://metanit.com/cpp/qt/6.1.php, свободный. (Дата обращения 27.03.2026).
9. Попов Е. Руководство языку C++ [Электронный ресурс]/Режим доступа: https://metanit.com/cpp/tutorial/, свободный. (Дата обращения 25.03.2026).
10. Страуструп Б. Язык программирования C++. Специальное издание / Перевод С. Анисимов – Москва.: Издательский дом «Вильямс», 2011. – 1136 с.
11. Шлее М. Qt 5.10. Профессиональное программирование на C++. – СПб.: «БХВ-Петербург», 2018. – 1072 с.
12. Документация wxWidgets [Электронный ресурс]/Режим доступа: https://docs.wxwidgets.org/stable/, свободный. (Дата обращения 14.04.2026).
13. Документация GTKmm [Электронный ресурс]/Режим доступа: https://gtkmm.gnome.org/ru/documentation.html, свободный. (Дата обращения 14.04.2026).

14. Документация PostgreSQL [Электронный ресурс]/Режим доступа: https://www.postgresql.org/docs/, свободный. (Дата обращения 16.04.2026).

15. Документация MySQL [Электронный ресурс]/Режим доступа: https://dev.mysql.com/doc/, свободный. (Дата обращения 16.04.2026).

16. Документация Visual Studio [Электронный ресурс]/Режим доступа: https://learn.microsoft.com/ru-ru/cpp/build/vscpp-step-0-installation, свободный. (Дата обращения 15.04.2026).

17. Документация CLion [Электронный ресурс]/Режим доступа: https://www.jetbrains.com/help/clion/qt-tutorial.html, свободный. (Дата обращения 15.04.2026).

18. Документация Qt Creator [Электронный ресурс]/Режим доступа: https://metanit.com/cpp/qt/1.2.php, свободный. (Дата обращения 02.04.2026).

19. Документация QAbstractListModel [Электронный ресурс]/Режим доступа: https://doc.qt.io/qt-6/qabstractlistmodel.html, свободный. (Дата обращения 28.03.2026).

20. Метод Kanban: принципы и применение [Электронный ресурс]/Режим доступа: https://kanbanize.com/ru/kanban-resources/getting-started/what-is-kanban, свободный. (Дата обращения 20.03.2026).

21. Документация QSystemTrayIcon [Электронный ресурс]/Режим доступа: https://doc.qt.io/qt-6/qsystemtrayicon.html, свободный. (Дата обращения 29.03.2026).

22. Фаулер М. UML. Основы. Краткое руководство по стандартному языку объектного моделирования / Перевод А. Панин – СПб.: «Символ-Плюс», 2011. – 192 с.

23. Попов Е. Руководство по SQLite [Электронный ресурс]/Режим доступа: https://metanit.com/sql/sqlite/, свободный. (Дата обращения 28.03.2026).

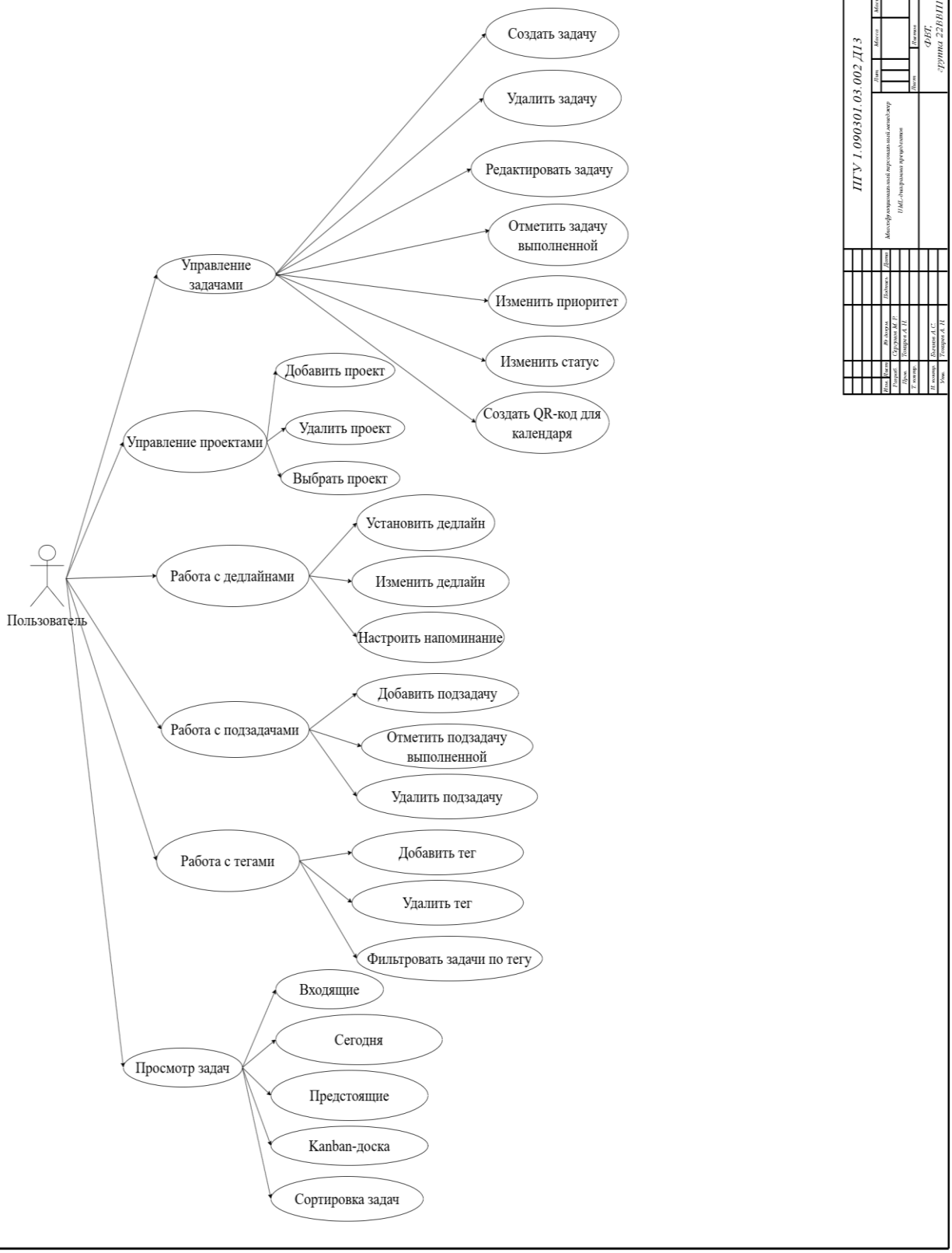
**Приложение А**

**(обязательное)**

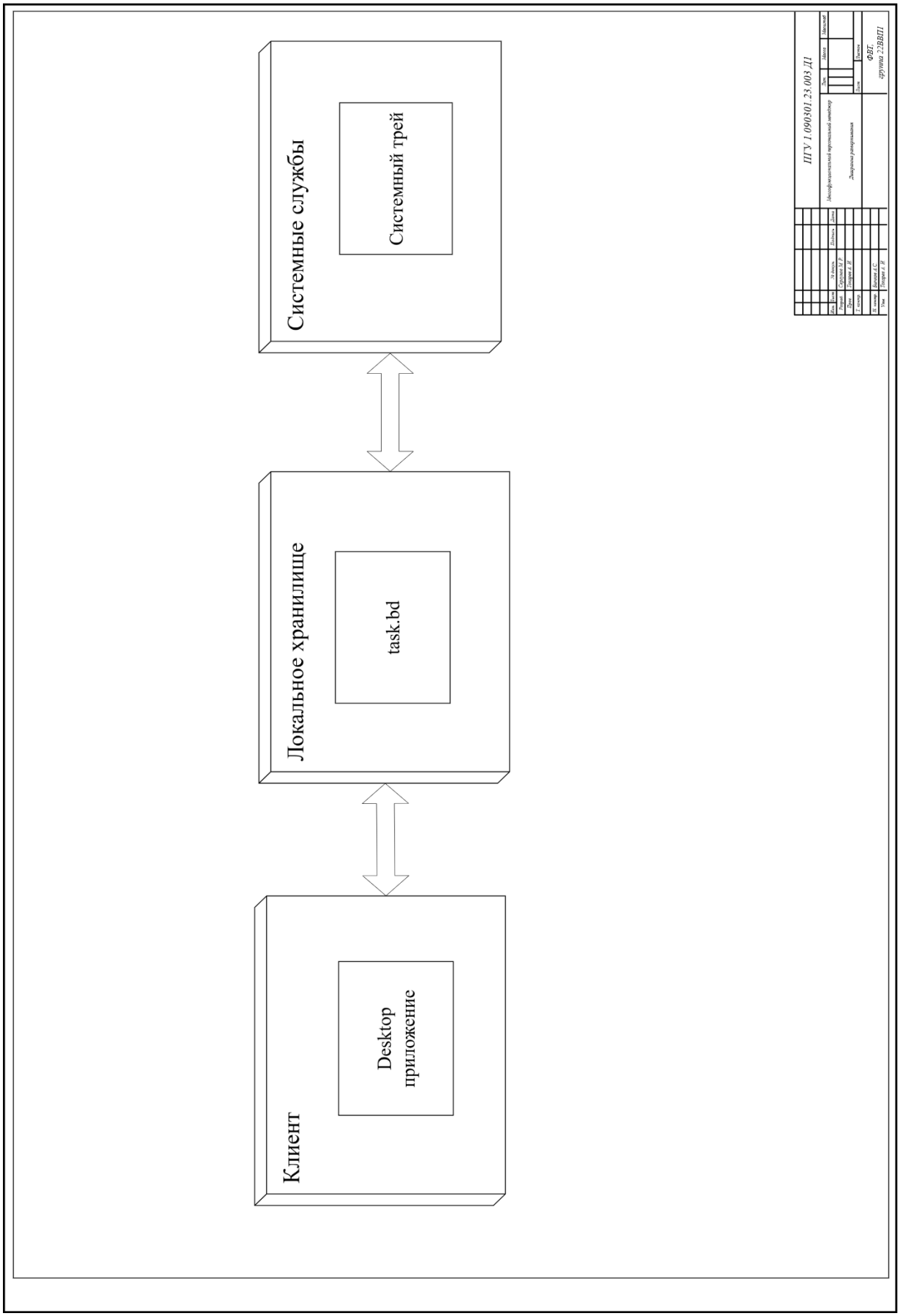
Приложение для ведения и контроля задач

UML-диаграммы

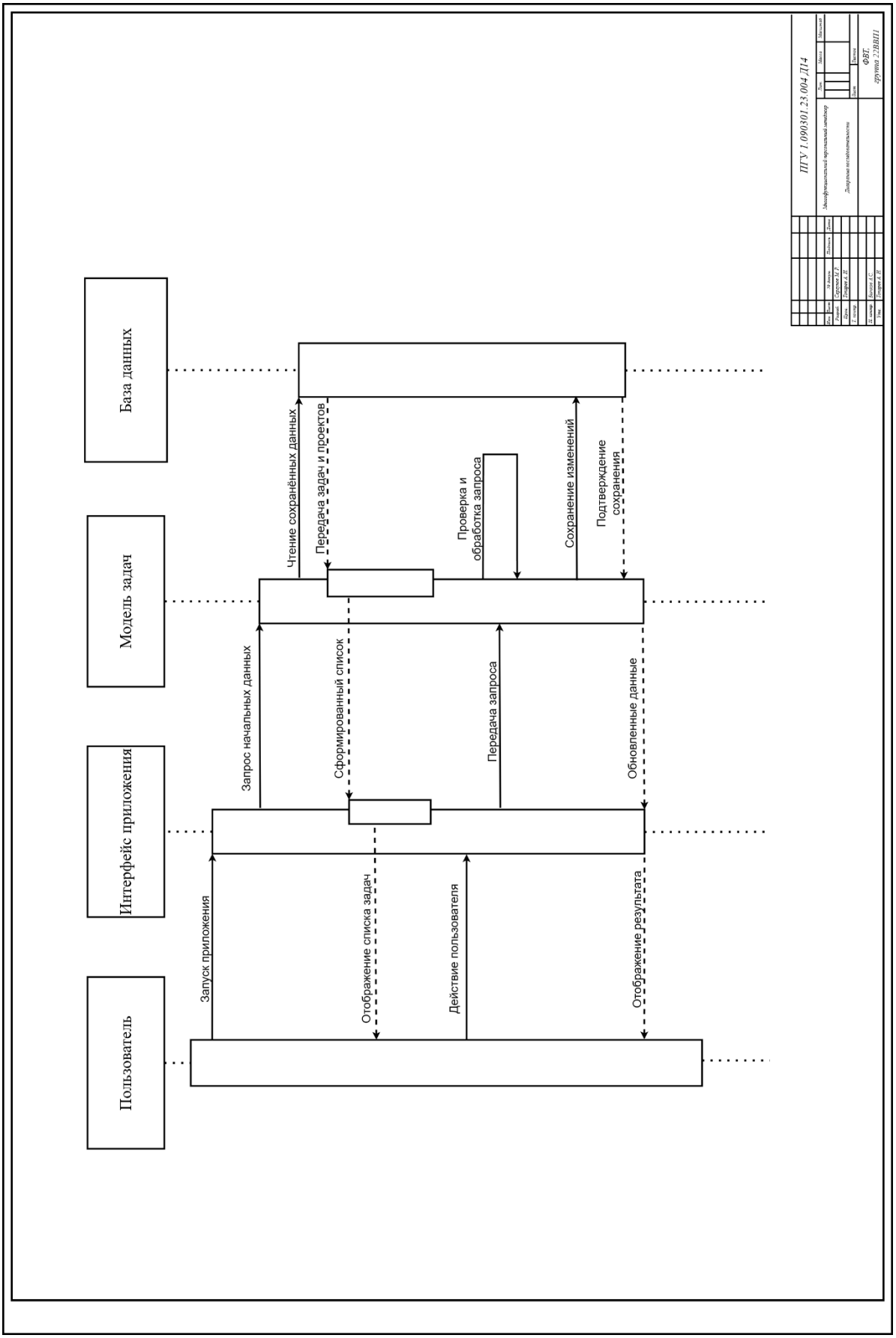
Приложение А.1 – Диаграмма прецедентов



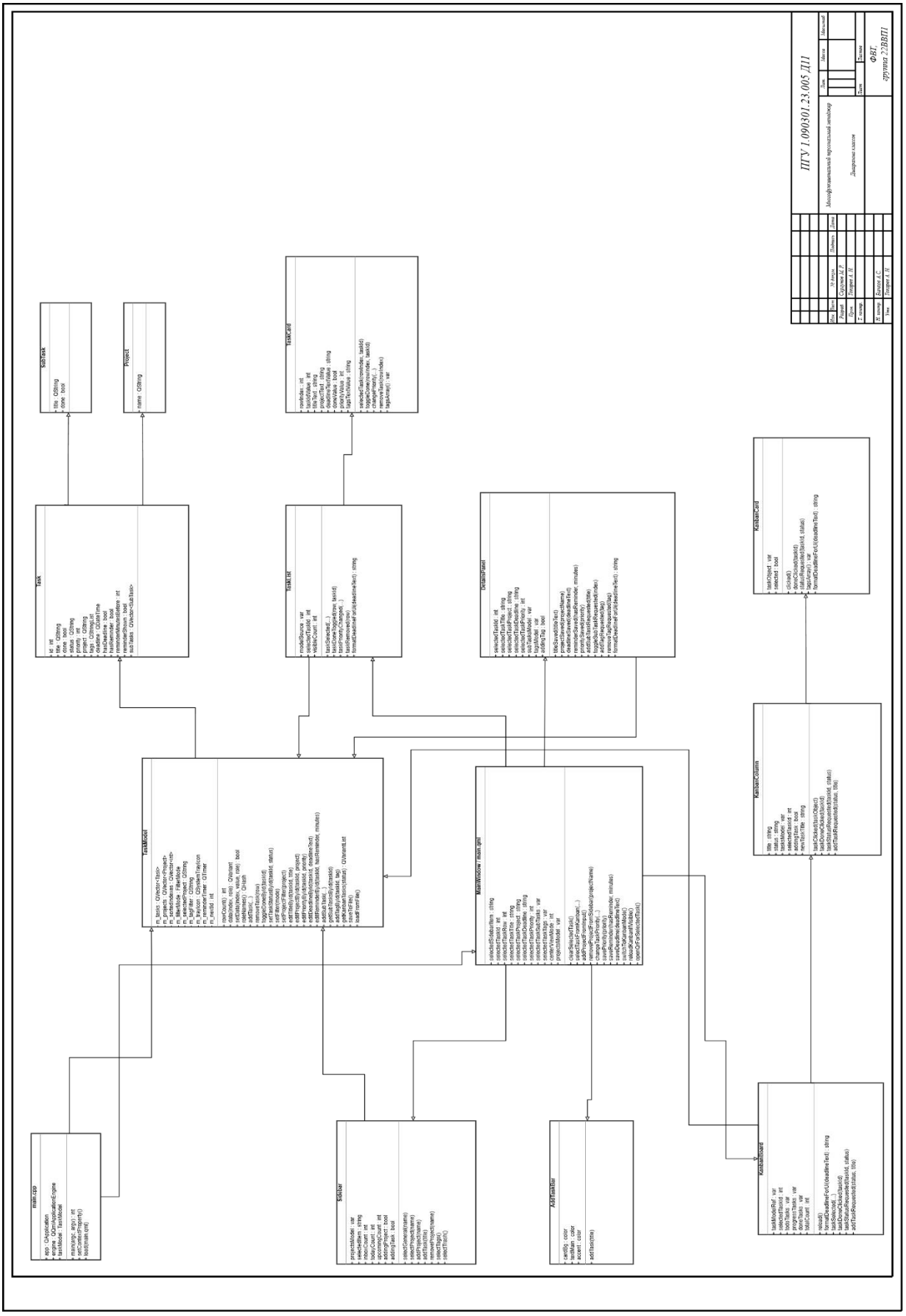
Приложение А.2 – Диаграмма развертывания



Приложение А.3 – Диаграмма последовательности



Приложение А.4 – Диаграмма классов



**Приложение Б**

**(обязательное)**

Приложение для ведения и контроля задач

Листинг программы

Файл main.cpp

```
#include <QApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>
#include <QQuickStyle>
#include "taskmodel.h"

int main(int argc, char *argv[])
{
    QQuickStyle::setStyle("Basic");

    QApplication app(argc, argv);

    QQmlApplicationEngine engine;

    TaskModel taskModel;
    engine.rootContext()    ->    setContextProperty("taskModel",
&taskModel);
    engine.load(QUrl(QStringLiteral("qrc:/main.qml")));

    if (engine.rootObjects().isEmpty())
        return -1;

    return app.exec();
}
```

Файл taskmodel.cpp

```
TaskModel::TaskModel(QObject *parent)
    : QAbstractListModel(parent)
{
    initDatabase();
    loadFromDatabase();

    if (m_projects.isEmpty()){
        addDefaultProjects();

    }

    m_trayIcon = new QSystemTrayIcon(this);
    m_trayIcon->setIcon(QApplication::style()-
>standardIcon(QStyle::SP_ComputerIcon));
    m_trayIcon -> show();

    connect(m_trayIcon,   &QSystemTrayIcon::messageClicked,this,
&TaskModel::notificationClicked);

    m_reminderTimer = new QTimer(this);
    m_reminderTimer->setTimerType(Qt::PreciseTimer);
    m_reminderTimer->setSingleShot(true);
```

```
        connect(m_reminderTimer, &QTimer::timeout, this, [this]() {
            checkReminders();
            scheduleNextReminderCheck();
        });

        checkReminders();

        scheduleNextReminderCheck();
    }
    void TaskModel::initDatabase(){
        m_db = QSqlDatabase::addDatabase("QSQLITE");
        m_db.setDatabaseName("task.db");

        if(!m_db.open()){
            qWarning()<<"Не удалось открыть базу данных:" <<
m_db.lastError().text();
            return;
        }

        QSqlQuery q(m_db);

        q.exec("PRAGMA foreign_keys = ON");
        q.exec(R"(CREATE TABLE IF NOT EXISTS projects ( name TEXT
PRIMARY KEY))");

        q.exec(R"(
            CREATE TABLE IF NOT EXISTS tasks (
                id INTEGER PRIMARY KEY,
                title TEXT NOT NULL,
                done INTEGER NOT NULL DEFAULT 0,
                status TEXT NOT NULL DEFAULT 'todo',
                priority INTEGER NOT NULL DEFAULT 0,
                project TEXT NOT NULL DEFAULT 'Без проекта',
                has_deadline INTEGER NOT NULL DEFAULT 0,
                deadline TEXT NOT NULL DEFAULT '',
                has_reminder INTEGER NOT NULL DEFAULT 0,
                reminder_minutes INTEGER NOT NULL DEFAULT 0,
                reminder_shown INTEGER NOT NULL DEFAULT 0
            )
        )");

        q.exec(R"(
            CREATE TABLE IF NOT EXISTS subtasks (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                task_id INTEGER NOT NULL,
                title TEXT NOT NULL,
                done INTEGER NOT NULL DEFAULT 0,
                pos INTEGER NOT NULL DEFAULT 0
            )
        )");

        q.exec(R"(
```

```
            CREATE TABLE IF NOT EXISTS tags(
                task_id INTEGER NOT NULL,
                tag TEXT NOT NULL,
                PRIMARY KEY (task_id, tag)
            )
        )");
    }

    void TaskModel::loadFromDatabase(){
        beginResetModel();

        m_projects.clear();
        m_tasks.clear();

        QSqlQuery q(m_db);

        q.exec("SELECT name FROM projects ORDER BY rowid");
        while (q.next()) {
            Project p;
            p.name = q.value(0).toString();
            m_projects.append(p);
        }

        q.exec("SELECT id, title, done, status, priority, project, "
               "has_deadline,        deadline,        has_reminder,
reminder_minutes, reminder_shown "
               "FROM tasks ORDER BY id");

        while(q.next()){
            Task task;
            task.id = q.value(0).toInt();
            task.title = q.value(1).toString();
            task.done = q.value(2).toBool();
            task.status = q.value(3).toString();
            task.priority = q.value(4).toInt();
            task.project = q.value(5).toString();
            task.hasDeadline = q.value(6).toBool();
            task.deadline                                          =
QDateTime::fromString(q.value(7).toString(), "yyyy-MM-dd HH:mm");
            task.hasReminder = q.value(8).toBool();
            task.reminderMinutesBefore = q.value(9).toInt();
            task.reminderShown = q.value(10).toBool();

            m_tasks.append(task);
        }

        q.exec("SELECT task_id, title, done FROM subtasks ORDER BY
task_id, pos, id");
        while(q.next()){
            int taskId = q.value(0).toInt();
            SubTask st;
            st.title = q.value(1).toString();
```

```
            st.done = q.value(2).toBool();

            for(Task &task : m_tasks){
                if (task.id == taskId){
                    task.subTasks.append(st);
                    break;
                }
            }
        }

        q.exec("SELECT task_id, tag FROM tags ORDER BY task_id");
        while(q.next()){
            int taskId = q.value(0).toInt();
            QString tag = q.value(1).toString();

            for(Task &task : m_tasks){
                if(task.id == taskId){
                    task.tags.append(tag);
                    break;
                }
            }
        }

        m_nextId = 1;
        for(const Task &task : m_tasks){
            if(task.id >= m_nextId)
                m_nextId = task.id + 1;
        }

        sortTask();
        endResetModel();

    }

    void TaskModel::saveTaskToDb(const Task &task){
        QSqlQuery q(m_db);

        q.prepare(R"(
            INSERT OR REPLACE INTO tasks
                (id, title, done, status, priority, project,
                 has_deadline,        deadline,        has_reminder,
reminder_minutes, reminder_shown)
            VALUES
                (:id, :title, :done, :status, :priority, :project,
                 :has_deadline,       :deadline,       :has_reminder,
:reminder_minutes, :reminder_shown)
        )");

        q.bindValue(":id", task.id);
        q.bindValue(":title", task.title);
        q.bindValue(":done", task.done ? 1 : 0);
        q.bindValue(":status", task.status);
```

```
        q.bindValue(":priority", task.priority);
        q.bindValue(":project", task.project);
        q.bindValue(":has_deadline", task.hasDeadline);
        q.bindValue(":deadline", task.hasDeadline
                                    ? task.deadline.toString("yyyy-
MM-dd HH:mm")
                                    : "");
        q.bindValue(":has_reminder", task.hasReminder ? 1 : 0);
        q.bindValue(":reminder_minutes",
task.reminderMinutesBefore);
        q.bindValue(":reminder_shown", task.reminderShown ? 1 : 0);

        if (!q.exec())
            qWarning()<<"SaveTaskToDb           error:"           <<
q.lastError().text();

        q.prepare("DELETE FROM subtasks WHERE task_id = :id");
        q.bindValue(":id",task.id);
        q.exec();

        for(int i = 0; i < task.subTasks.size(); ++i){
            q.prepare("INSERT INTO subtasks (task_id, title, done,
pos) "
                      "VALUES (:task_id, :title, :done, :pos)");
            q.bindValue(":task_id", task.id);
            q.bindValue(":title", task.subTasks[i].title);
            q.bindValue(":done",  task.subTasks[i].done ? 1 : 0);
            q.bindValue(":pos", i);
            q.exec();
        }

        q.prepare("DELETE FROM tags WHERE task_id = :id");
        q.bindValue(":id", task.id);
        q.exec();

        for(const QString &tag : task.tags){
            q.prepare("INSERT  INTO  tags  (task_id,  tag)  VALUES
(:task_id, :tag)");
            q.bindValue(":task_id", task.id);
            q.bindValue(":tag", tag);
            q.exec();
        }
    }

    void TaskModel::deleteTaskFromDb(int taskId){
        QSqlQuery q(m_db);

        q.prepare("DELETE FROM tasks WHERE id = :id");
        q.bindValue(":id", taskId);
        q.exec();

        q.prepare("DELETE FROM subtasks WHERE task_id = :id");
```

```
        q.bindValue(":id", taskId);
        q.exec();

        q.prepare("DELETE FROM tags WHERE task_id = :id");
        q.bindValue(":id", taskId);
        q.exec();
    }

    void TaskModel::saveProjectToDb(const Project &project){
        QSqlQuery q(m_db);

        q.prepare("INSERT OR IGNORE INTO projects (name) VALUES
(:name)");
        q.bindValue(":name", project.name);

        if(!q.exec())
            qWarning()<<"saveProjectToDb         error:"         <<
q.lastError().text();
    }

    void TaskModel::deleteProjectFromDb(const QString &name){
        QSqlQuery q(m_db);

        q.prepare("DELETE FROM projects WHERE name = :name");
        q.bindValue(":name", name);
        q.exec();
    }
    void TaskModel::addTask(const QString &title,
                            int priority,
                            const QString &project,
                            const   QString   &deadlineText,   bool
hasDeadLine){
        if (title.trimmed().isEmpty())
            return;
        beginResetModel();
        //          beginInsertRows(QModelIndex(),  m_tasks.size(),
m_tasks.size());
        Task task;
        task.id = m_nextId++;
        task.title = title;
        task.done = false;
        task.status = "todo";
        task.priority = priority;
        task.project = project;

        QDateTime                  parsedDeadline                  =
QDateTime::fromString(deadlineText, "yyyy-MM-dd HH:mm");

        if (hasDeadLine && parsedDeadline.isValid()){
            task.deadline = parsedDeadline;
            task.hasDeadline = true;
        } else{
```

```
            task.deadline = QDateTime();
            task.hasDeadline = false;
        }
        m_tasks.append(task);
        sortTask();
        //    endInsertRows();
        endResetModel();
        notifySidebarCountsChanged();
        saveTaskToDb(task);;
    }

    void TaskModel::removeTask(int row){
        int sourceRow = mapToSource(row);
        if (sourceRow == -1)
            return;
        int taskId = m_tasks[sourceRow].id;
        beginResetModel();
        //    beginRemoveRows(QModelIndex(), row, row);
        m_tasks.removeAt(sourceRow);
        sortTask();
        endResetModel();
        notifySidebarCountsChanged();

        deleteTaskFromDb(taskId);
    }
    void TaskModel::toggleDoneById(int taskId){
        int sourceRow = findTaskIndexById(taskId);
        if (sourceRow == -1)
            return;

        beginResetModel();
        m_tasks[sourceRow].done = !m_tasks[sourceRow].done;
        if (m_tasks[sourceRow].done){
            m_tasks[sourceRow].status = "done";
        } else{
            m_tasks[sourceRow].status = "todo";
        }

        sortTask();
        endResetModel();

        notifySidebarCountsChanged();
        saveTaskToDb(m_tasks[sourceRow]);
    }

    void  TaskModel::setTaskStatusById(int  taskId,  const  QString
&status){
        if  (status!= "todo" && status != "progress" && status !=
"done")
            return;

        int sourceRow = findTaskIndexById(taskId);
```

```
        if(sourceRow==-1)
            return;

        beginResetModel();

        m_tasks[sourceRow].status = status;
        m_tasks[sourceRow].done = (status == "done");

        sortTask();
        endResetModel();
        saveTaskToDb(m_tasks[sourceRow]);
    }

    QVariantList TaskModel::getKanbanTasks(const QString &status)
const{
        QVariantList result;

        if(status!= "todo" && status != "progress" && status !=
"done")
            return result;

        for(int sortedIndex : m_sortedIndexes){
            if(sortedIndex < 0 || sortedIndex >= m_tasks.size())
                continue;

            const Task &task = m_tasks[sortedIndex];

            if(task.status != status)
                continue;

            if (!taskMatchesCurrentFilters(task))
                continue;

            QVariantMap taskMap;

            taskMap["taskId"] = task.id;
            taskMap["title"] = task.title;
            taskMap["done"] = task.done;
            taskMap["status"] = task.status;
            taskMap["priority"] = task.priority;
            taskMap["project"] = task.project;
            taskMap["hasDeadline"] = task.hasDeadline;
            taskMap["deadlineText"]     =     task.hasDeadline     ?
task.deadline.toString("yyyy-MM-dd HH:mm") : "";
            taskMap["hasReminder"] = task.hasReminder;
            taskMap["reminderMinutesBefore"]                        =
task.reminderMinutesBefore;
            taskMap["tagsText"] = task.tags.isEmpty() ? "" : "#" +
task.tags.join(" #");

            result.append(taskMap);
        }
```

```
        return result;
    }

    void TaskModel::setFilter(int mode){
        m_filterMode = mode;
        beginResetModel();
        endResetModel();
    }
    void TaskModel::sortTask()
    {
        m_sortedIndexes.clear();

        for (int i = 0; i < m_tasks.size(); ++i) {
            m_sortedIndexes.append(i);
        }

        std::sort(m_sortedIndexes.begin(),    m_sortedIndexes.end(),
[this](int a, int b) {
            const Task &t1 = m_tasks[a];
            const Task &t2 = m_tasks[b];

            // Выполненные задачи вниз
            if (t1.done != t2.done)
                return !t1.done && t2.done;

            switch (m_sortMode) {
            case SortByPriority:
                if (t1.priority != t2.priority)
                    return t1.priority > t2.priority;

                if (t1.hasDeadline != t2.hasDeadline)
                    return t1.hasDeadline;

                if (t1.hasDeadline && t2.hasDeadline && t1.deadline
!= t2.deadline)
                    return t1.deadline < t2.deadline;

                return t1.id > t2.id;

            case SortByTitle: {
                int                          cmp                          =
QString::localeAwareCompare(t1.title.toLower(), t2.title.toLower());

                if (cmp != 0)
                    return cmp < 0;

                return t1.id > t2.id;
            }

            case SortByCreated:
                return t1.id > t2.id;
```

```
            case SortByDate:
            default:
                if (t1.hasDeadline != t2.hasDeadline)
                    return t1.hasDeadline;

                if (t1.hasDeadline && t2.hasDeadline && t1.deadline
!= t2.deadline)
                    return t1.deadline < t2.deadline;

                if (t1.priority != t2.priority)
                    return t1.priority > t2.priority;

                return t1.id > t2.id;
            }
        });
    }
    void TaskModel::checkReminders(){
        QDateTime realNow = QDateTime::currentDateTime();
        QDateTime now(realNow.date(),
                      QTime(realNow.time().hour(),
                            realNow.time().minute(),
                            0));

        bool changed = false;

        for (Task &task : m_tasks){
            if (!task.hasDeadline)
                continue;

            if (!task.hasReminder)
                continue;

            if (task.done)
                continue;

            if (task.reminderShown)
                continue;

            QDateTime   reminderTime   =   task.deadline.addSecs(-
task.reminderMinutesBefore *60);

            if (now >= reminderTime){
                if (m_trayIcon){
                    m_trayIcon     ->     showMessage("Напоминание",
task.title, QSystemTrayIcon::Information, 10000);
                }

                task.reminderShown = true;
                changed = true;
                saveTaskToDb(task);
            }
```

```
        }

        if (changed){
            if (rowCount() > 0){
                emit  dataChanged(index(0,0),  index(rowCount()  -1,
0));
            }
        }
    }
    void TaskModel::addProject(const QString &name){
        QString trimmedName = name.trimmed();
        if (trimmedName.isEmpty())
            return;

        for (const Project &project : m_projects){
            if (project.name == trimmedName)
                return;
        }

        Project project;
        project.name = trimmedName;
        m_projects.append(project);
        notifySidebarCountsChanged();

        saveProjectToDb(project);
    }

    void TaskModel::removeProject(const QString &name){
        if (name == "Без проекта")
            return;

        for (Task &task : m_tasks){
            if (task.project == name){
                task.project = "Без проекта";
                saveTaskToDb(task);
            }
        }

        for (int i = 0; i < m_projects.size(); ++i){
            if (m_projects[i].name == name){
                m_projects.removeAt(i);
                break;
            }
        }

        emit dataChanged(index(0, 0), index(rowCount() - 1,0));
        notifySidebarCountsChanged();
        deleteProjectFromDb(name);
    }
    QString TaskModel::generateTaskQrPngById(int taskId){
        QString payload = getTaskCalendarPayloadById(taskId);
```

```
        if(payload.isEmpty())
            return "";

        QByteArray payloadUtf8 = payload.toUtf8();

        qrcodegen::QrCode qr = qrcodegen::QrCode::encodeText(
            payloadUtf8.constData(),
            qrcodegen::QrCode::Ecc::MEDIUM);

        int qrSize = qr.getSize();
        int scale = 8;
        int border = 4;
        int imageSize = (qrSize + border * 2) * scale;

        QImage image(imageSize, imageSize, QImage::Format_RGB32);
        image.fill(Qt::white);

        QPainter painter(&image);
        painter.setPen(Qt::NoPen);
        painter.setBrush(Qt::black);

        for (int y = 0; y < qrSize; ++y){
            for (int x = 0; x < qrSize; ++x){
                if (qr.getModule(x,y)){
                    painter.drawRect((x + border) * scale,
                                     (y + border) * scale,
                                     scale,
                                     scale);
                }
            }
        }

        painter.end();

        QString fileName = QDir::tempPath()
            + "/task_qr_"
            + QString::number(taskId)
            + "_"
            + QString::number(QDateTime::currentMSecsSinceEpoch())
            + ".png";

        if (!image.save(fileName, "PNG"))
            return "";

        return QUrl::fromLocalFile(fileName).toString();
    }
    QString TaskModel::getTaskCalendarPayloadById(int taskId) const
{
        int sourceRow = findTaskIndexById(taskId);

        if(sourceRow==-1)
            return "";
```

```
        const Task &task = m_tasks[sourceRow];

        if(!task.hasDeadline)
            return "";

        QDateTime startDateTime = task.deadline;
        QDateTime endDateTime = task.deadline.addSecs(30*60);

        QString payload;

        payload += "BEGIN:VCALENDAR\r\n";
        payload += "VERSION:2.0\r\n";
        payload += "PRODID:-//TaskManager//Task QR//RU\r\n";
        payload += "BEGIN:VEVENT\r\n";
        payload   +=   "UID:task-"   +   QString::number(task.id)   +
"@taskmanager.local\r\n";
        payload             +=             "DTSTAMP:"             +
QDateTime::currentDateTimeUtc().toString("yyyyMMdd'T'HHmmSS'Z'")    +
"\r\n";
        payload += "SUMMARY:" + escapeIcsText(task.title) + "\r\n";
        payload  +=  "DTSTART:"  +  formatIcsDateTime(startDateTime)  +
"\r\n";
        payload  +=  "DTEND:"  +  formatIcsDateTime(endDateTime)  +
"\r\n";

        if(task.hasReminder){
            payload += "BEGIN:VALARM\r\n";
            payload += "ACTION:DISPLAY\r\n";
            payload            +=            "TRIGGER:"            +
reminderToIcsTrigger(task.reminderMinutesBefore) + "\r\n";
            payload += "DESCRIPTION:" + escapeIcsText(task.title) +
"\r\n";
            payload += "END:VALARM\r\n";
        }

        payload += "END:VEVENT\r\n";
        payload += "END:VCALENDAR\r\n";

        return payload;
    }
```

Файл Sidebar.qml

```
import QtQuick 2.15
import QtQuick.Controls 2.15
import QtQuick.Layouts 1.15

Rectangle {
    id: root

    property var projectsModel: []
```

```
property string selectedItem: "Входящие"

property color panelBg: "#10141D"
property color cardBg: "#151922"
property color cardHover: "#1D2433"
property color textMain: "#F9FAFB"
property color textMuted: "#A2AABD"
property color accent: "#8B5CF6"
property color accentSoft: "#332453"
property color borderColor: "#2A3140"
property color danger: "#EF4444"

property int inboxCount: 0
property int todayCount: 0
property int upcomingCount: 0
property int tagsCount: 0
property int trashCount: 0

property bool addingProject: false
property bool addingTask: false

signal selectGeneral(string name)
signal selectProject(string name)
signal addProject(string name)
signal addTask(string title)
signal removeProject(string name)
signal selectTags()
signal selectTrash()

Layout.preferredWidth: 264
Layout.fillHeight: true

color: panelBg
border.color: borderColor

ColumnLayout {
    anchors.fill: parent
    spacing: 0

    Flickable {
        Layout.fillWidth: true
        Layout.fillHeight: true
        clip: true
        boundsBehavior: Flickable.StopAtBounds
        contentHeight: contentColumn.implicitHeight + 24

        Column {
            id: contentColumn
            width: parent.width - 20
            x: 10
            y: 18
            spacing: 12
```

```
                    // Кнопка / поле создания новой задачи
                    Rectangle {
                        width: parent.width
                        height: root.addingTask ? 132 : 72
                        radius: 14
                        color:  root.addingTask  ?  root.cardBg  :
newTaskHover.hovered ? "#171D29" : "transparent"
                        border.color: root.addingTask ? root.accent
: "transparent"
                        border.width: root.addingTask ? 1 : 0
                        clip: true

                        Behavior  on  height  {  NumberAnimation  {
duration: 160; easing.type: Easing.OutCubic } }
                        Behavior  on  color  {  ColorAnimation  {
duration: 130 } }
                        Behavior on border.color { ColorAnimation {
duration: 130 } }

                        HoverHandler { id: newTaskHover }

                        Item {
                            anchors.fill: parent
                            visible: !root.addingTask

                            RowLayout {
                                anchors.fill: parent
                                anchors.leftMargin: 10
                                anchors.rightMargin: 10
                                spacing: 14

                                Rectangle {
                                    Layout.preferredWidth: 42
                                    Layout.preferredHeight: 42
                                    radius: 12
                                    gradient: Gradient {
                                        GradientStop { position: 0;
color: "#A855F7" }
                                        GradientStop { position: 1;
color: root.accent }
                                    }
                                    Text {
                                        anchors.centerIn: parent

anchors.verticalCenterOffset: -2
                                        text: "+"
                                        color: "#080B12"
                                        font.pixelSize: 28
                                        font.bold: true
                                    }
                                }
```

```
                                Text {
                                    text: "Новая задача"
                                    color: root.textMain
                                    font.pixelSize: 17
                                    font.weight: Font.Medium
                                    Layout.fillWidth: true
                                    elide: Text.ElideRight
                                }
                            }

                            MouseArea {
                                anchors.fill: parent
                                cursorShape: Qt.PointingHandCursor
                                onClicked: {
                                    root.addingTask = true
                                    Qt.callLater(function()        {
taskInput.forceActiveFocus() })
                                }
                            }
                        }

                        ColumnLayout {
                            anchors.fill: parent
                            anchors.margins: 12
                            spacing: 10
                            visible: root.addingTask

                            TextField {
                                id: taskInput
                                Layout.fillWidth: true
                                Layout.preferredHeight: 46
                                placeholderText: "Название задачи"
                                placeholderTextColor: root.textMuted
                                color: root.textMain
                                font.pixelSize: 14
                                verticalAlignment:
TextInput.AlignVCenter
                                selectionColor: root.accent
                                background: Rectangle {
                                    radius: 10
                                    color: "#0F1420"
                                    border.color:
taskInput.activeFocus ? root.accent : root.borderColor
                                    border.width: 1
                                }
                                leftPadding: 14
                                rightPadding: 14
                                onAccepted: addTaskNow()
                            }

                            RowLayout {
```

```
                                    Layout.fillWidth: true
                                    spacing: 8

                                    ButtonLike {
                                        Layout.fillWidth: true
                                        text: "Добавить"
                                        enabledState:
taskInput.text.trim() !== ""
                                        onClicked: addTaskNow()
                                    }

                                    ButtonLike {
                                        Layout.preferredWidth: 74
                                        text: "Отмена"
                                        enabledState: true
                                        muted: true
                                        onClicked: {
                                            taskInput.text = ""
                                            root.addingTask = false
                                        }
                                    }
                                }
                            }
                        }

                        Column {
                            width: parent.width
                            spacing: 6

                            NavItem {
                                width: parent.width
                                title: "Входящие"
                                iconName: "inbox"
                                count: root.inboxCount
                                selected:     root.selectedItem     ===
"Входящие"
                                onClicked:
root.selectGeneral("Входящие")
                            }

                            NavItem {
                                width: parent.width
                                title: "Сегодня"
                                iconName: "sun"
                                count: root.todayCount
                                selected:     root.selectedItem     ===
"Сегодня"
                                onClicked: root.selectGeneral("Сегодня")
                            }

                            NavItem {
                                width: parent.width
```

```
                                title: "Предстоящие"
                                iconName: "calendar"
                                count: root.upcomingCount
                                selected:     root.selectedItem     ===
"Предстоящие"
                                onClicked:
root.selectGeneral("Предстоящие")
                            }
                        }

                        Text {
                            width: parent.width
                            leftPadding: 14
                            topPadding: 12
                            bottomPadding: 2
                            text: "Проекты"
                            color: root.textMuted
                            font.pixelSize: 15
                            font.weight: Font.Normal
                        }

                        Column {
                            width: parent.width
                            spacing: 6

                            Repeater {
                                model: root.projectsModel
                                delegate: ProjectItem {
                                    width: parent.width
                                    title: modelData
                                    selected:   root.selectedItem   ===
modelData
                                    count:   (taskModel.sidebarRevision,
taskModel.projectCount(modelData))
                                    dotColor:        projectColor(index,
modelData)
                                    canDelete:   modelData   !==   "Без
проекта"
                                    onClicked:
root.selectProject(modelData)
                                    onDeleteClicked:
root.removeProject(modelData)
                                }
                            }

                            Rectangle {
                                width: parent.width
                                height: 42
                                radius: 10
                                color:     addProjectHover.hovered     ?
root.cardHover : "transparent"
                                visible: !root.addingProject
```

```
                HoverHandler { id: addProjectHover }

                RowLayout {
                    anchors.fill: parent
                    anchors.leftMargin: 16
                    anchors.rightMargin: 14
                    spacing: 14

                    Text {
                        text: "+"
                        color: root.accent
                        font.pixelSize: 20
                        font.bold: true
                        Layout.preferredWidth: 18
                        horizontalAlignment:
Text.AlignHCenter
                    }

                    Text {
                        text: "Добавить проект"
                        color: root.accent
                        font.pixelSize: 14
                        font.weight: Font.DemiBold
                        Layout.fillWidth: true
                    }
                }

                MouseArea {
                    anchors.fill: parent
                    cursorShape: Qt.PointingHandCursor
                    onClicked: {
                        root.addingProject = true
                        projectInput.forceActiveFocus()
                    }
                }
            }

            Rectangle {
                width: parent.width
                height: 88
                radius: 14
                color: root.cardBg
                border.color: root.accent
                visible: root.addingProject

                ColumnLayout {
                    anchors.fill: parent
                    anchors.margins: 10
                    spacing: 8

                    TextField {
```

```
                                    id: projectInput
                                    Layout.fillWidth: true
                                    Layout.preferredHeight: 32
                                    placeholderText:       "Название
проекта"
                                    placeholderTextColor:
root.textMuted
                                    color: root.textMain
                                    font.pixelSize: 13
                                    selectionColor: root.accent
                                    background:  Rectangle  {  color:
"transparent" }
                                    onAccepted: addProjectNow()
                                }

                                RowLayout {
                                    Layout.fillWidth: true
                                    spacing: 8

                                    ButtonLike {
                                        Layout.fillWidth: true
                                        text: "Добавить"
                                        enabledState:
projectInput.text.trim() !== ""
                                        onClicked: addProjectNow()
                                    }

                                    ButtonLike {
                                        Layout.preferredWidth: 70
                                        text: "Отмена"
                                        enabledState: true
                                        muted: true
                                        onClicked: {
                                            projectInput.text = ""
                                            root.addingProject     =
false
                                        }
                                    }
                                }
                            }
                        }
                    }

                }
            }

            Rectangle {
                Layout.fillWidth: true
                Layout.preferredHeight: 72
                color: "#0E131C"
                border.color: root.borderColor
```

```
                Row {
                    anchors.left: parent.left
                    anchors.leftMargin: 26
                    anchors.verticalCenter: parent.verticalCenter
                    spacing: 28

                    SmallIconButton { iconName: "gear" }
                    SmallIconButton { iconName: "moon" }
                }
            }
        }


        function addTaskNow() {
            var title = taskInput.text.trim()
            if (title === "") return
            root.addTask(title)
            taskInput.text = ""
            root.addingTask = false
        }

        function addProjectNow() {
            var name = projectInput.text.trim()
            if (name === "") return
            root.addProject(name)
            projectInput.text = ""
            root.addingProject = false
        }

        function projectColor(index, name) {
            if (name === "Без проекта") return root.accent
            var colors = ["#4ADE80", "#3B82F6", "#8B5CF6", "#F97316",
"#EC4899", "#22D3EE"]
            return colors[(index - 1 + colors.length) %
colors.length]
        }

        component NavItem: Rectangle {
            id: item
            property string title: ""
            property string iconName: "inbox"
            property int count: 0
            property bool selected: false
            property bool hovered: false
            signal clicked()

            height: 54
            radius: 10
            color: selected ? root.accentSoft : hovered ?
root.cardHover : "transparent"

            RowLayout {
```

```
                anchors.fill: parent
                anchors.leftMargin: 16
                anchors.rightMargin: 14
                spacing: 14

                LineIcon {
                    iconName: item.iconName
                    color:   item.selected   ?   root.textMain   :
root.textMuted
                    Layout.preferredWidth: 20
                    Layout.preferredHeight: 20
                    Layout.alignment: Qt.AlignVCenter
                }

                Text {
                    text: item.title
                    color:   item.selected   ?   root.textMain   :
root.textMain
                    font.pixelSize: 16
                    font.weight: Font.Normal
                    Layout.fillWidth: true
                    elide: Text.ElideRight
                }

                Text {
                    text: item.count > 0 ? String(item.count) : ""
                    color:   item.selected   ?   root.accent   :
root.textMuted
                    font.pixelSize: 16
                    font.weight: Font.Normal
                    horizontalAlignment: Text.AlignRight
                    Layout.preferredWidth: 26
                }
            }

            HoverHandler { onHoveredChanged: item.hovered = hovered
}
            MouseArea   {   anchors.fill:   parent;   cursorShape:
Qt.PointingHandCursor; onClicked: item.clicked() }
        }

        component ProjectItem: Rectangle {
            id: item
            property string title: ""
            property color dotColor: root.accent
            property int count: 0
            property bool selected: false
            property bool canDelete: true
            property bool hovered: false
            signal clicked()
            signal deleteClicked()
```

```
            height: 52
            radius: 10
            color:  selected  ?  root.accentSoft  :  hovered  ?
root.cardHover : "transparent"

            HoverHandler {
                id: projectHover
                onHoveredChanged: item.hovered = hovered
            }

            // Основная кликабельная зона проекта. Справа оставляем
место под крестик,
            //  чтобы  MouseArea  проекта  не  перекрывала  кнопку
удаления.
            MouseArea {
                anchors.left: parent.left
                anchors.top: parent.top
                anchors.bottom: parent.bottom
                anchors.right: parent.right

                // справа всегда оставляем место под кнопку удаления,
                // а не только при hover
                anchors.rightMargin: item.canDelete ? 44 : 0

                cursorShape: Qt.PointingHandCursor
                onClicked: item.clicked()
            }

            RowLayout {
                anchors.fill: parent
                anchors.leftMargin: 16
                anchors.rightMargin: 12
                spacing: 14

                Rectangle {
                    Layout.preferredWidth: 12
                    Layout.preferredHeight: 12
                    radius: 6
                    color: item.dotColor
                    Layout.alignment: Qt.AlignVCenter
                }

                Text {
                    text: item.title
                    color: root.textMain
                    font.pixelSize: 16
                    font.weight: Font.Normal
                    Layout.fillWidth: true
                    elide: Text.ElideRight
                }

                Text {
```

```
                    visible: !item.hovered || !item.canDelete
                    text: item.count > 0 ? String(item.count) : ""
                    color:    item.selected    ?    root.accent    :
root.textMuted
                    font.pixelSize: 16
                    Layout.preferredWidth: 28
                    horizontalAlignment: Text.AlignRight
                }

                Rectangle {
                    id: deleteButton
                    visible: item.hovered && item.canDelete
                    Layout.preferredWidth: 30
                    Layout.preferredHeight: 30
                    radius: 8
                    color:   deleteHover.hovered   ?   "#3A1620"   :
"#1D2433"
                    z: 10

                    Text {
                        anchors.centerIn: parent
                        text: "×"
                        color:  deleteHover.hovered  ?  "#FCA5A5"  :
root.textMuted
                        font.pixelSize: 20
                        font.bold: true
                    }

                    HoverHandler { id: deleteHover }

                    MouseArea {
                        anchors.fill: parent
                        z: 11
                        cursorShape: Qt.PointingHandCursor
                        onClicked: function(mouse) {
                            mouse.accepted = true
                            item.deleteClicked()
                        }
                    }
                }
            }
        }

        component ButtonLike: Rectangle {
            property string text: ""
            property bool enabledState: true
            property bool muted: false
            signal clicked()
            Layout.preferredHeight: 34
            radius: 10
            color: muted ? "transparent" : enabledState ? root.accent
: root.borderColor
```

```
            Text { anchors.centerIn: parent; text: parent.text;
color: muted ? root.textMuted : "white"; font.pixelSize: 12;
font.bold: !muted }
            MouseArea { anchors.fill: parent; enabled:
parent.enabledState; cursorShape: Qt.PointingHandCursor; onClicked:
parent.clicked() }
        }

        component SmallIconButton: Rectangle {
            property string iconName: "gear"
            width: 28
            height: 28
            radius: 8
            color: iconHover.hovered ? root.cardHover : "transparent"
            LineIcon { anchors.centerIn: parent; width: 22; height:
22; iconName: parent.iconName; color: root.textMuted }
            HoverHandler { id: iconHover }
        }

        component LineIcon: Canvas {
            id: canvas
            property string iconName: "inbox"
            property color color: root.textMuted
            onPaint: {
                var ctx = getContext("2d")
                ctx.reset()
                ctx.strokeStyle = color
                ctx.fillStyle = color
                ctx.lineWidth = 1.7
                ctx.lineCap = "round"
                ctx.lineJoin = "round"
                var w = width, h = height

                function line(x1,y1,x2,y2){ ctx.beginPath();
ctx.moveTo(x1*w,y1*h); ctx.lineTo(x2*w,y2*h); ctx.stroke() }
                function rect(x,y,rw,rh,r){ ctx.beginPath();
ctx.roundedRect(x*w,y*h,rw*w,rh*h,r*w,r*h); ctx.stroke() }
                function circle(x,y,r){ ctx.beginPath();
ctx.arc(x*w,y*h,r*w,0,Math.PI*2); ctx.stroke() }

                if (iconName === "inbox") {
                    ctx.beginPath(); ctx.moveTo(.18*w,.38*h);
ctx.lineTo(.3*w,.2*h); ctx.lineTo(.7*w,.2*h);
ctx.lineTo(.82*w,.38*h); ctx.lineTo(.82*w,.78*h);
ctx.lineTo(.18*w,.78*h); ctx.closePath(); ctx.stroke()
                    line(.18,.46,.38,.46); line(.38,.46,.44,.58);
line(.44,.58,.56,.58); line(.56,.58,.62,.46); line(.62,.46,.82,.46)
                } else if (iconName === "sun") {
                    circle(.5,.5,.23); for (var i=0;i<8;i++){ var
a=i*Math.PI/4;
line(.5+Math.cos(a)*.34,.5+Math.sin(a)*.34,.5+Math.cos(a)*.45,.5+Mat
h.sin(a)*.45) }
```

```
                } else if (iconName === "calendar") {
                    rect(.18,.22,.64,.62,.06);
line(.18,.38,.82,.38); line(.34,.14,.34,.28); line(.66,.14,.66,.28)
                } else if (iconName === "tag") {
                    ctx.beginPath();        ctx.moveTo(.22*w,.46*h);
ctx.lineTo(.46*w,.22*h);                    ctx.lineTo(.78*w,.22*h);
ctx.lineTo(.78*w,.54*h);  ctx.lineTo(.54*w,.78*h);  ctx.closePath();
ctx.stroke(); circle(.62,.38,.035)
                } else if (iconName === "trash") {
                    line(.26,.3,.74,.3);      line(.42,.18,.58,.18);
line(.36,.18,.64,.18);                    rect(.32,.3,.36,.52,.04);
line(.42,.42,.42,.72); line(.58,.42,.58,.72)
                } else if (iconName === "gear") {
                    circle(.5,.5,.18);  circle(.5,.5,.06);  for  (var
j=0;j<8;j++){                    var                    b=j*Math.PI/4;
line(.5+Math.cos(b)*.28,.5+Math.sin(b)*.28,.5+Math.cos(b)*.39,.5+Mat
h.sin(b)*.39) }
                } else if (iconName === "moon") {
                    ctx.beginPath();
ctx.arc(.56*w,.48*h,.32*w,Math.PI*.55,Math.PI*1.55,true);
ctx.arc(.43*w,.42*h,.28*w,Math.PI*1.55,Math.PI*.55,false);
ctx.stroke()
                }
            }
            onColorChanged: requestPaint()
            onIconNameChanged: requestPaint()
        }
    }
```

Файл DetailsPanel.qml

```
import QtQuick 2.15
import QtQuick.Controls 2.15
import QtQuick.Layouts 1.15

Rectangle {
    id: root

    property int selectedTaskId: -1
    property int selectedTaskRow: -1
    focus: true

    onSelectedTaskIdChanged: {
        console.log("DETAILS PANEL ID:", selectedTaskId)
    }

    property string selectedTaskTitle: ""
    property string selectedTaskProject: ""
    property string selectedTaskDeadline: ""
    property int selectedTaskPriority: 0
    property bool selectedTaskHasReminder: false
    property int selectedTaskReminderMinutesBefore: 0
```

```
property bool selectedTaskDone: false
signal doneToggled(bool doneValue)
signal closeRequested()
signal qrRequested()

property var projectsModel: []

property color appBg: "#080B12"
property color panelBg: "#10141D"
property color cardBg: "#151922"
property color cardHover: "#1D2433"
property color textMain: "#F9FAFB"
property color textMuted: "#9CA3AF"
property color accent: "#8B5CF6"
property color accentSoft: "#2D2147"
property color borderColor: "#2A3140"
property color danger: "#EF4444"
property color success: "#10B981"

property bool titleEditing: false

property date deadlineDraftDate: new Date()
property int deadlineDraftHour: 18
property int deadlineDraftMinute: 0
property date calendarMonth: new Date()

property var subTasksModel: []
property string newSubTaskText: ""
property var tagsModel: []
property string newTagText: ""
property bool addingTag: false



signal titleSaved(string titleText)
signal projectSaved(string projectName)
signal deadlineSaved(string deadlineText)
signal reminderSaved(bool hasReminder, int minutesBefore)
signal prioritySaved(int priorityValue)

signal addSubTaskRequested(string title)
signal toggleSubTaskRequested(int subIndex)
signal removeSubTaskRequested(int subIndex)
signal editSubTaskRequested(int subIndex, string title)
signal reloadSubTasksRequested()
signal addTagRequested(string tagName)
signal removeTagRequested(string tagName)

Layout.preferredWidth: 420
Layout.minimumWidth: 360
Layout.maximumWidth: 520
Layout.fillHeight: true
```

```
            color: panelBg
            border.color: borderColor

            function pad2(value) {
                return value < 10 ? "0" + value : "" + value
            }

            function deadlineDraftTimeText() {
                return     pad2(deadlineDraftHour)     +     ":"     +
pad2(deadlineDraftMinute)
            }

            function exitTextEditing() {
                root.forceActiveFocus()
            }

            function formatDeadlineForModel(date) {
                return date.getFullYear() + "-"
                        + pad2(date.getMonth() + 1) + "-"
                        + pad2(date.getDate()) + " "
                        + pad2(deadlineDraftHour) + ":"
                        + pad2(deadlineDraftMinute)
            }

            function formatDeadlineForUi(deadlineText) {
                if (deadlineText === "")
                    return "Нет"

                var parts = deadlineText.split(" ")
                var dateParts = parts[0].split("-")

                return dateParts[2] + "." +
                       dateParts[1] + "." +
                       dateParts[0] + " " +
                       parts[1]
            }

            function reminderLabel() {
                if (!selectedTaskHasReminder)
                    return "Нет"

                if (selectedTaskReminderMinutesBefore === 0)
                    return "В момент"

                if (selectedTaskReminderMinutesBefore === 10)
                    return "За 10 минут"

                if (selectedTaskReminderMinutesBefore === 60)
                    return "За 1 час"

                if (selectedTaskReminderMinutesBefore === 1440)
```

```
                return "За 1 день"

            return "За " + selectedTaskReminderMinutesBefore + " мин"
        }

        function monthName(month) {
            var names = ["Январь", "Февраль", "Март", "Апрель",
                         "Май",    "Июнь",    "Июль",    "Август",
"Сентябрь",
                         "Октябрь", "Ноябрь", "Декабрь"]

            return names[month]
        }

        function daysInMonth(dateValue) {
            return         new         Date(dateValue.getFullYear(),
dateValue.getMonth() + 1, 0).getDate()
        }

        function firstDayOfMonth(dateValue) {
            var    day    =    new    Date(dateValue.getFullYear(),
dateValue.getMonth(), 1).getDay()
            return day === 0 ? 6 : day - 1
        }

        function completedSubTasksCount() {
            var count = 0

            for (var i = 0; i < subTasksModel.length; i++) {
                if (subTasksModel[i].done)
                    count++
            }

            return count
        }

        function subTasksProgress() {
            if (!subTasksModel || subTasksModel.length === 0)
                return 0

            return completedSubTasksCount() / subTasksModel.length
        }

        function cleanTagText(value) {
            var clean = value.trim()

            if (clean.indexOf("#") === 0)
                clean = clean.substring(1).trim()

            return clean
        }
```

```
        function submitTag() {
            var clean = cleanTagText(root.newTagText)

            if (clean === "")
                return

            root.addTagRequested(clean)
            root.newTagText = ""
            root.addingTag = false
            root.exitTextEditing()
        }

        function cancelTagInput() {
            root.newTagText = ""
            root.addingTag = false
            root.exitTextEditing()
        }

        Item {
            id: emptyState
            visible: root.selectedTaskId < 0
            anchors.fill: parent

            Column {
                anchors.centerIn: parent
                width: parent.width - 40
                spacing: 8

                Text {
                    width: parent.width
                    text: "Выберите задачу"
                    color: textMain
                    font.pixelSize: 22
                    font.bold: true
                    horizontalAlignment: Text.AlignHCenter
                }

                Text {
                    width: parent.width
                    text: "Нажмите на задачу в списке, чтобы открыть
её параметры."
                    color: textMuted
                    font.pixelSize: 14
                    wrapMode: Text.Wrap
                    horizontalAlignment: Text.AlignHCenter
                }
            }
        }

        Flickable {
            visible: root.selectedTaskId >= 0
            anchors.fill: parent
```

```
            contentHeight: contentColumn.implicitHeight + 40
            clip: true

            TapHandler {
                acceptedButtons: Qt.LeftButton

                onTapped: {
                    root.exitTextEditing()
                }
            }

            ColumnLayout {
                id: contentColumn

                width: parent.width - 40
                x: 20
                y: 20
                spacing: 16

                RowLayout {
                    Layout.fillWidth: true
                    Layout.preferredHeight: 44
                    spacing: 14

                    Rectangle {
                        id: detailCheckBox

                        Layout.preferredWidth: 32
                        Layout.preferredHeight: 32
                        radius: 9

                        color:  root.selectedTaskDone  ?  accent  :
"transparent"
                        border.width: 2
                        border.color: root.selectedTaskDone ? accent
: accent

                        Text {
                            anchors.centerIn: parent
                            visible: root.selectedTaskDone
                            text: "✓"
                            color: "white"
                            font.pixelSize: 17
                            font.bold: true
                        }

                        MouseArea {
                            anchors.fill: parent
                            cursorShape: Qt.PointingHandCursor

                            onClicked: {
```

```
root.doneToggled(!root.selectedTaskDone)
                                }
                            }
                        }

                        Icon {
                            iconName: "flag"
                            iconColor: root.selectedTaskPriority === 2 ?
danger
                                     : root.selectedTaskPriority === 1 ?
"#F59E0B"
                                     : "#737C8C"
                            Layout.preferredWidth: 22
                            Layout.preferredHeight: 22
                        }

                        TextField {
                            id: headerTitleInput

                            Layout.fillWidth: true
                            text: root.selectedTaskTitle

                            background: null
                            color:  root.selectedTaskDone  ?  textMuted  :
textMain
                            font.pixelSize: 22
                            font.bold: true
                            selectByMouse: true
                            selectionColor: accent
                            selectedTextColor: "white"

                            opacity: root.selectedTaskDone ? 0.65 : 1

                            onAccepted: {
                                if (text.trim() !== "") {
                                    root.titleSaved(text.trim())
                                    focus = false
                                }
                            }

                            onEditingFinished: {
                                if (text.trim() !== "" && text.trim() !==
root.selectedTaskTitle) {
                                    root.titleSaved(text.trim())
                                }
                            }
                        }

                        Rectangle {
                            Layout.preferredWidth: 34
                            Layout.preferredHeight: 30
```

```
                        radius: 10
                        color: qrMouse.containsMouse ? "#242B38" :
"transparent"

                        Text {
                            anchors.centerIn: parent
                            text: "QR"
                            color: textMuted
                            font.pixelSize: 12
                            font.bold: true
                        }

                        MouseArea {
                            id: qrMouse
                            anchors.fill: parent
                            hoverEnabled: true
                            cursorShape: Qt.PointingHandCursor

                            onClicked: {
                                root.qrRequested()
                            }
                        }
                    }

                    Rectangle {
                        Layout.preferredWidth: 30
                        Layout.preferredHeight: 30
                        radius: 10
                        color: closeMouse.containsMouse ? "#242B38"
: "transparent"

                        Icon {
                            anchors.centerIn: parent
                            width: 20
                            height: 20
                            iconName: "x"
                            iconColor: textMuted
                        }

                        MouseArea {
                            id: closeMouse
                            anchors.fill: parent
                            hoverEnabled: true
                            cursorShape: Qt.PointingHandCursor

                            onClicked: root.closeRequested()
                        }
                    }
                }

                Text {
                    text: "Свойства"
```

```
                    color: textMuted
                    font.pixelSize: 13
                    font.bold: true
                }

                PropertyRow {
                    iconName: "project"
                    title: "Проект"
                    value: root.selectedTaskProject === "" ? "Без
проекта" : root.selectedTaskProject
                    accentValue: root.selectedTaskProject !== ""
                    showChevron: true

                    onClicked: projectPopup.open()

                    Popup {
                        id: projectPopup

                        y: parent.height + 6
                        width: parent.width
                        height:
Math.min(projectColumn.implicitHeight + 16, 260)
                        padding: 8

                        background: Rectangle {
                            radius: 16
                            color: cardBg
                            border.color: borderColor
                        }

                        Column {
                            id: projectColumn
                            width: parent.width
                            spacing: 6

                            Repeater {
                                model: {
                                    var list = root.projectsModel ?
root.projectsModel.slice() : []

                                    if (list.indexOf("Без проекта")
=== -1)
                                        list.unshift("Без проекта")

                                    return list
                                }

                                delegate: Rectangle {
                                    property bool hovered: false

                                    width: parent.width
                                    height: 34
```

```
                                    radius: 10

                                    color:  root.selectedTaskProject
=== modelData || hovered
                                           ? accentSoft
                                           : "transparent"

                                    Text {
                                        anchors.centerIn: parent
                                        text: modelData
                                        color:
root.selectedTaskProject === modelData ? accent : textMain
                                        font.pixelSize: 13
                                        font.bold:
root.selectedTaskProject === modelData
                                    }

                                    HoverHandler {
                                        onHoveredChanged:
parent.hovered = hovered
                                    }

                                    MouseArea {
                                        anchors.fill: parent
                                        cursorShape:
Qt.PointingHandCursor

                                        onClicked: {

root.projectSaved(modelData === "Без проекта" ? "" : modelData)
                                            projectPopup.close()
                                        }
                                    }
                                }
                            }
                        }
                    }
                }

                PropertyRow {
                    iconName: "flag"
                    title: "Приоритет"
                    value:   root.selectedTaskPriority   ===   2   ?
"Высокий"
                         :   root.selectedTaskPriority   ===   1   ?
"Средний"
                         : "Низкий"
                    accentValue: false
                    showChevron: true

                    onClicked: priorityPopup.open()
```

```
                    Popup {
                        id: priorityPopup

                        y: parent.height + 6
                        width: parent.width
                        height: 124
                        padding: 8

                        background: Rectangle {
                            radius: 16
                            color: cardBg
                            border.color: borderColor
                        }

                        Column {
                            anchors.fill: parent
                            spacing: 6

                            PriorityItem {
                                title: "⚑ Низкий"
                                selected:  root.selectedTaskPriority
=== 0
                                colorValue: "#9CA3AF"

                                onClicked: {
                                    root.prioritySaved(0)
                                    priorityPopup.close()
                                }
                            }

                            PriorityItem {
                                title: "⚑ Средний"
                                selected:  root.selectedTaskPriority
=== 1
                                colorValue: "#F59E0B"

                                onClicked: {
                                    root.prioritySaved(1)
                                    priorityPopup.close()
                                }
                            }

                            PriorityItem {
                                title: "⚑ Высокий"
                                selected:  root.selectedTaskPriority
=== 2
                                colorValue: danger

                                onClicked: {
                                    root.prioritySaved(2)
                                    priorityPopup.close()
```

```
                                }
                            }
                        }
                    }
                }

                PropertyRow {
                    iconName: "calendar"
                    title: "Дедлайн"
                    value: root.selectedTaskDeadline === "" ? "Не
установлен" : root.formatDeadlineForUi(root.selectedTaskDeadline)
                    mutedValue: root.selectedTaskDeadline === ""
                    showChevron: root.selectedTaskDeadline === ""
                    showClear: root.selectedTaskDeadline !== ""

                    onClicked: {
                        root.deadlineDraftDate = new Date()
                        root.calendarMonth = new Date()
                        root.deadlineDraftHour = 12
                        root.deadlineDraftMinute = 0
                        deadlinePopup.open()
                    }
                    onClearClicked: {
                        root.deadlineSaved("")
                    }

                    Popup {
                        id: deadlinePopup

                        y: parent.height + 8
                        width: Math.max(parent.width, 390)
                        height: 460
                        padding: 14
                        modal: false
                        focus: true
                        closePolicy:      Popup.CloseOnEscape      |
Popup.CloseOnPressOutside

                        background: Rectangle {
                            radius: 18
                            color: cardBg
                            border.color: borderColor
                        }

                        ColumnLayout {
                            anchors.fill: parent
                            spacing: 10

                            RowLayout {
                                Layout.fillWidth: true
                                Layout.preferredHeight: 32
```

```
                                    Text {
                                        Layout.fillWidth: true
                                        text:
root.monthName(root.calendarMonth.getMonth())     +     "     "     +
root.calendarMonth.getFullYear()
                                        color: textMain
                                        font.pixelSize: 15
                                        font.bold: true
                                    }

                                    SmallButton {
                                        textValue: "‹"
                                        onClicked: {
                                            root.calendarMonth   =   new
Date(root.calendarMonth.getFullYear(),

root.calendarMonth.getMonth() - 1,

1)
                                        }
                                    }

                                    SmallButton {
                                        textValue: "›"
                                        onClicked: {
                                            root.calendarMonth   =   new
Date(root.calendarMonth.getFullYear(),

root.calendarMonth.getMonth() + 1,

1)
                                        }
                                    }
                                }

                                GridLayout {
                                    Layout.fillWidth: true
                                    columns: 7
                                    rowSpacing: 4
                                    columnSpacing: 4

                                    Repeater {
                                        model: ["Пн", "Вт", "Ср", "Чт",
"Пт", "Сб", "Вс"]

                                        Text {
                                            Layout.preferredWidth: 40
                                            Layout.preferredHeight: 22
                                            text: modelData
                                            color: textMuted
                                            font.pixelSize: 11
```

```
                                        horizontalAlignment:
Text.AlignHCenter
                                        verticalAlignment:
Text.AlignVCenter
                                    }
                                }

                                Repeater {
                                    model: 42

                                    Rectangle {
                                        property   int   dayNumber:
index - root.firstDayOfMonth(root.calendarMonth) + 1
                                        property   bool   validDay:
dayNumber >= 1 && dayNumber <= root.daysInMonth(root.calendarMonth)
                                        property date thisDate: new
Date(root.calendarMonth.getFullYear(),

root.calendarMonth.getMonth(),

Math.max(1, dayNumber))
                                        property  bool  selectedDay:
validDay &&

root.deadlineDraftDate.getFullYear() === thisDate.getFullYear() &&

root.deadlineDraftDate.getMonth() === thisDate.getMonth() &&

root.deadlineDraftDate.getDate() === thisDate.getDate()
                                        property bool hovered: false

                                        Layout.preferredWidth: 40
                                        Layout.preferredHeight: 32
                                        radius: 10

                                        color: selectedDay ? accent
                                             : hovered && validDay ?
accentSoft
                                             : "transparent"

                                        Text {
                                            anchors.centerIn: parent
                                            text:  parent.validDay  ?
parent.dayNumber : ""
                                            color:
parent.selectedDay ? "white"
                                                 : parent.validDay ?
textMain
                                                 : "transparent"
                                            font.pixelSize: 13
                                            font.bold:
parent.selectedDay
```

```
                                        }

                                        HoverHandler {
                                            enabled: parent.validDay
                                            onHoveredChanged:
parent.hovered = hovered
                                        }

                                        MouseArea {
                                            anchors.fill: parent
                                            enabled: parent.validDay
                                            cursorShape:
Qt.PointingHandCursor

                                            onClicked: {

root.deadlineDraftDate = parent.thisDate
                                            }
                                        }
                                    }
                                }
                            }

                            Rectangle {
                                Layout.fillWidth: true
                                Layout.preferredHeight: 74
                                radius: 12
                                color: "#111722"
                                border.color: borderColor

                                RowLayout {
                                    anchors.fill: parent
                                    anchors.leftMargin: 12
                                    anchors.rightMargin: 12
                                    spacing: 10

                                    Text {
                                        text: "Время"
                                        color: textMuted
                                        font.pixelSize: 13
                                        Layout.preferredWidth: 54
                                    }

                                    Item {
                                        Layout.fillWidth: true
                                    }

                                    ColumnLayout {
                                        spacing: 4

                                        Text {
                                            text: "Часы"
```

```
                                        color: textMuted
                                        font.pixelSize: 11
                                        Layout.alignment:
Qt.AlignHCenter
                                    }

                                    RowLayout {
                                        spacing: 6

                                        SmallButton {
                                            textValue: "−"
                                            onClicked:
root.deadlineDraftHour = Math.max(0, root.deadlineDraftHour - 1)
                                        }

                                        Text {
                                            text:
root.pad2(root.deadlineDraftHour)
                                            color: textMain
                                            font.pixelSize: 16
                                            font.bold: true

Layout.preferredWidth: 28

horizontalAlignment: Text.AlignHCenter
                                        }

                                        SmallButton {
                                            textValue: "+"
                                            onClicked:
root.deadlineDraftHour = Math.min(23, root.deadlineDraftHour + 1)
                                        }
                                    }
                                }

                                Text {
                                    text: ":"
                                    color: textMain
                                    font.pixelSize: 18
                                    font.bold: true
                                    Layout.topMargin: 14
                                }

                                ColumnLayout {
                                    spacing: 4

                                    Text {
                                        text: "Мин"
                                        color: textMuted
                                        font.pixelSize: 11
                                        Layout.alignment:
Qt.AlignHCenter
```

```
                                        }

                                        RowLayout {
                                            spacing: 6

                                            SmallButton {
                                                textValue: "−"
                                                onClicked: {

root.deadlineDraftMinute -= 1
                                                    if
(root.deadlineDraftMinute < 0)

root.deadlineDraftMinute = 59
                                                }
                                            }

                                            Text {
                                                text:
root.pad2(root.deadlineDraftMinute)
                                                color: textMain
                                                font.pixelSize: 16
                                                font.bold: true

Layout.preferredWidth: 28

horizontalAlignment: Text.AlignHCenter
                                            }

                                            SmallButton {
                                                textValue: "+"
                                                onClicked: {

root.deadlineDraftMinute += 1
                                                    if
(root.deadlineDraftMinute > 59)

root.deadlineDraftMinute = 0
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                            RowLayout {
                                Layout.fillWidth: true
                                Layout.preferredHeight: 40
                                spacing: 8

                                Rectangle {
                                    Layout.fillWidth: true
                                    Layout.fillHeight: true
```

```
                                    radius: 12
                                    color: "#111722"
                                    border.color: borderColor

                                    Text {
                                        anchors.centerIn: parent
                                        text: "Очистить"
                                        color: textMuted
                                        font.pixelSize: 13
                                        font.bold: true
                                    }

                                    MouseArea {
                                        anchors.fill: parent
                                        cursorShape:
Qt.PointingHandCursor

                                        onClicked: {
                                            root.deadlineSaved("")
                                            deadlinePopup.close()
                                        }
                                    }
                                }

                                Rectangle {
                                    Layout.fillWidth: true
                                    Layout.fillHeight: true
                                    radius: 12
                                    color: accent
                                    border.color: accent

                                    Text {
                                        anchors.centerIn: parent
                                        text: "Сохранить"
                                        color: "white"
                                        font.pixelSize: 13
                                        font.bold: true
                                    }

                                    MouseArea {
                                        anchors.fill: parent
                                        cursorShape:
Qt.PointingHandCursor

                                        onClicked: {

root.deadlineSaved(root.formatDeadlineForModel(root.deadlineDraftDat
e))
                                            deadlinePopup.close()
                                        }
                                    }
                                }
```

```
                        }
                    }
                }
            }

            PropertyRow {
                iconName: "bell"
                title: "Напоминание"
                value:  root.reminderLabel()  ===  "Нет"  ?  "Не
установлено" : root.reminderLabel()
                mutedValue: !root.selectedTaskHasReminder
                showChevron: true

                onClicked: reminderPopup.open()

                Popup {
                    id: reminderPopup

                    y: parent.height + 6
                    width: parent.width
                    height: 188
                    padding: 8

                    background: Rectangle {
                        radius: 16
                        color: cardBg
                        border.color: borderColor
                    }

                    Column {
                        anchors.fill: parent
                        spacing: 6

                        ReminderItem {
                            title: "Нет"
                            selected:
!root.selectedTaskHasReminder

                            onClicked: {
                                root.reminderSaved(false, 0)
                                reminderPopup.close()
                            }
                        }

                        ReminderItem {
                            title: "В момент"
                            selected:
root.selectedTaskHasReminder                                    &&
root.selectedTaskReminderMinutesBefore === 0

                            onClicked: {
                                root.reminderSaved(true, 0)
```

```
                                    reminderPopup.close()
                                }
                            }

                            ReminderItem {
                                title: "За 10 минут"
                                selected:
root.selectedTaskHasReminder                                                    &&
root.selectedTaskReminderMinutesBefore === 10

                                onClicked: {
                                    root.reminderSaved(true, 10)
                                    reminderPopup.close()
                                }
                            }

                            ReminderItem {
                                title: "За 1 час"
                                selected:
root.selectedTaskHasReminder                                                    &&
root.selectedTaskReminderMinutesBefore === 60

                                onClicked: {
                                    root.reminderSaved(true, 60)
                                    reminderPopup.close()
                                }
                            }

                            ReminderItem {
                                title: "За 1 день"
                                selected:
root.selectedTaskHasReminder                                                    &&
root.selectedTaskReminderMinutesBefore === 1440

                                onClicked: {
                                    root.reminderSaved(true, 1440)
                                    reminderPopup.close()
                                }
                            }
                        }
                    }
                }

                ColumnLayout {
                    Layout.fillWidth: true
                    spacing: 10

                    RowLayout {
                        Layout.fillWidth: true
                        Layout.preferredHeight: 22

                        Text {
```

```
                    text: "Подзадачи"
                    color: textMain
                    font.pixelSize: 15
                    font.bold: true
                    Layout.fillWidth: true
                }

                Text {
                    text: completedSubTasksCount() + " / " +
subTasksModel.length
                    color: textMain
                    font.pixelSize: 14
                    font.bold: true
                }
            }

            Rectangle {
                Layout.fillWidth: true
                Layout.preferredHeight: 4
                radius: 2
                color: "#202632"

                Rectangle {
                    width:          parent.width          *
root.subTasksProgress()
                    height: parent.height
                    radius: parent.radius
                    color: accent

                    Behavior on width {
                        NumberAnimation {
                            duration: 180
                            easing.type: Easing.OutCubic
                        }
                    }
                }
            }

            Rectangle {
                Layout.fillWidth: true
                Layout.preferredHeight: 188
                radius: 14
                color: "#151A23"
                border.width: 1
                border.color: borderColor
                clip: true

                ColumnLayout {
                    anchors.fill: parent
                    spacing: 0

                    ListView {
```

```
                                    id: subtaskListView

                                    Layout.fillWidth: true
                                    Layout.preferredHeight: 138

                                    clip: true
                                    boundsBehavior:
Flickable.StopAtBounds
                                    interactive:
root.subTasksModel.length > 3

                                    model: root.subTasksModel

                                    ScrollBar.vertical: ScrollBar {
                                        width: 5
                                        policy:  subtaskListView.count >
3
                                                ? ScrollBar.AsNeeded
                                                : ScrollBar.AlwaysOff
                                    }

                                    delegate: Rectangle {
                                        id: subtaskRow

                                        property   bool   editCommitted:
false
                                        property bool hovered: false
                                        property string subTitle: {
                                            if   (typeof   modelData   ===
"undefined" || modelData === null)
                                                return ""

                                            if                    (typeof
modelData["title"] !== "undefined")
                                                return
modelData["title"]

                                            if  (typeof  modelData.title
!== "undefined")
                                                return modelData.title

                                            return ""
                                        }

                                        property bool subDone: {
                                            if   (typeof   modelData   ===
"undefined" || modelData === null)
                                                return false

                                            if (typeof modelData["done"]
!== "undefined")
                                                return modelData["done"]
```

```
                                        if   (typeof   modelData.done
!== "undefined")
                                            return modelData.done

                                        return false
                                    }

                                    width: subtaskListView.width
                                    height: 46
                                    color:  hovered  ?  "#1B222E"  :
"transparent"

                                    RowLayout {
                                        anchors.fill: parent
                                        anchors.leftMargin: 14
                                        anchors.rightMargin: 12
                                        spacing: 12

                                        Rectangle {
                                            Layout.preferredWidth:
20
                                            Layout.preferredHeight:
20
                                            radius: 5

                                            color:
subtaskRow.subDone ? accent : "transparent"
                                            border.width: 1.6
                                            border.color:
subtaskRow.subDone ? accent : "#6B7280"

                                            Text {
                                                anchors.centerIn:
parent
                                                visible:
subtaskRow.subDone
                                                text: "✓"
                                                color: "white"
                                                font.pixelSize: 13
                                                font.bold: true
                                            }

                                            MouseArea {
                                                anchors.fill: parent
                                                cursorShape:
Qt.PointingHandCursor

                                                onClicked: {

root.toggleSubTaskRequested(index)
                                                }
```

```
                                        }
                                    }

                                    TextField {
                                        id: subtaskTitleField

                                        Layout.fillWidth: true
                                        text:
subtaskRow.subTitle

                                        color:
subtaskRow.subDone ? textMuted : textMain
                                        font.pixelSize: 14
                                        font.strikeout:
subtaskRow.subDone
                                        opacity:
subtaskRow.subDone ? 0.72 : 1

                                        selectByMouse: true
                                        background: null
                                        padding: 0
                                        leftPadding: 0
                                        rightPadding: 0

                                        verticalAlignment:
TextInput.AlignVCenter

                                        onAccepted: {
                                            editCommitted = true

                                            if (text.trim() !==
"" && text.trim() !== subtaskRow.subTitle)

root.editSubTaskRequested(index, text.trim())


root.exitTextEditing()
                                        }

                                        onEditingFinished: {
                                            if (editCommitted) {
                                                editCommitted =
false

                                                return
                                            }

                                            if (text.trim() !==
"" && text.trim() !== subtaskRow.subTitle)

root.editSubTaskRequested(index, text.trim())
                                        }
```

```
                                        Keys.onEscapePressed: {
                                            editCommitted = true
                                            text                =
subtaskRow.subTitle

root.exitTextEditing()
                                        }
                                    }

                                    Text {
                                        text: "•••"
                                        color: textMuted
                                        font.pixelSize: 18
                                        font.bold: true
                                        horizontalAlignment:
Text.AlignHCenter
                                        Layout.preferredWidth:
30
                                    }

                                    Rectangle {
                                        visible:
subtaskRow.hovered
                                        Layout.preferredWidth:
24
                                        Layout.preferredHeight:
24
                                        radius: 8
                                        color:
removeMouse.containsMouse ? "#2A3140" : "transparent"

                                        Text {
                                            anchors.centerIn:
parent
                                            text: "×"
                                            color: danger
                                            font.pixelSize: 18
                                        }

                                        MouseArea {
                                            id: removeMouse
                                            anchors.fill: parent
                                            hoverEnabled: true
                                            cursorShape:
Qt.PointingHandCursor

                                            onClicked: {

root.removeSubTaskRequested(index)
                                            }
                                        }
                                    }
```

```
                                    }

                                    Rectangle {
                                        anchors.left: parent.left
                                        anchors.right: parent.right
                                        anchors.bottom:
parent.bottom
                                        height: 1
                                        color: borderColor
                                        opacity:      index      ===
root.subTasksModel.length - 1 ? 0 : 1
                                    }

                                    HoverHandler {
                                        onHoveredChanged:
subtaskRow.hovered = hovered
                                    }
                                }
                            }

                            Rectangle {
                                Layout.fillWidth: true
                                Layout.preferredHeight: 50
                                color: "#121722"

                                Rectangle {
                                    anchors.left: parent.left
                                    anchors.right: parent.right
                                    anchors.top: parent.top
                                    height: 1
                                    color: borderColor
                                }

                                RowLayout {
                                    anchors.fill: parent
                                    anchors.leftMargin: 14
                                    anchors.rightMargin: 10
                                    spacing: 10

                                    TextField {
                                        id: newSubtaskInput

                                        Layout.fillWidth: true
                                        text: root.newSubTaskText
                                        placeholderText:      "Новая
подзадача..."
                                        placeholderTextColor:
textMuted
                                        color: textMain
                                        font.pixelSize: 14
                                        selectByMouse: true
                                        background: null
```

```
                                        Keys.onEscapePressed: {
                                            text = ""
                                            root.newSubTaskText = ""
                                            root.exitTextEditing()
                                        }

                                        onTextChanged: {
                                            root.newSubTaskText    =
text
                                        }

                                        onAccepted: {
                                            if (text.trim() !== "")
{

root.addSubTaskRequested(text.trim())
                                                root.newSubTaskText
= ""

                                                text = ""
                                            }

                                            root.exitTextEditing()
                                        }
                                    }

                                    Rectangle {
                                        visible:
newSubtaskInput.activeFocus || newSubtaskInput.text.length > 0
                                        Layout.preferredWidth: 32
                                        Layout.preferredHeight: 32
                                        radius: 10
                                        color:
cancelSubtaskMouse.containsMouse ? "#242B38" : "transparent"

                                        Text {
                                            anchors.centerIn: parent
                                            text: "×"
                                            color: textMuted
                                            font.pixelSize: 20
                                        }

                                        MouseArea {
                                            id: cancelSubtaskMouse
                                            anchors.fill: parent
                                            hoverEnabled: true
                                            cursorShape:
Qt.PointingHandCursor

                                            onClicked: {

newSubtaskInput.text = ""
```

```
                                                root.newSubTaskText
= ""

newSubtaskInput.focus = false

root.forceActiveFocus()
                                                    }
                                                }
                                            }

                                            Rectangle {
                                                Layout.preferredWidth: 38
                                                Layout.preferredHeight: 32
                                                radius: 10

                                                color:
addSubtaskMouse.containsMouse ? "#9B6DFF" : accent
                                                opacity:
newSubtaskInput.text.trim() === "" ? 0.55 : 1

                                                Text {
                                                    anchors.centerIn: parent
                                                    text: "+"
                                                    color: "white"
                                                    font.pixelSize: 24
                                                    font.bold: true
                                                }

                                                MouseArea {
                                                    id: addSubtaskMouse

                                                    anchors.fill: parent
                                                    hoverEnabled: true
                                                    enabled:
newSubtaskInput.text.trim() !== ""
                                                    cursorShape:
Qt.PointingHandCursor

                                                    onClicked: {
                                                        if
(newSubtaskInput.text.trim() !== "") {

root.addSubTaskRequested(newSubtaskInput.text.trim())

root.newSubTaskText = ""

newSubtaskInput.text = ""
                                                    }


newSubtaskInput.focus = false
```

```
root.forceActiveFocus()
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }

                        ColumnLayout {
                            Layout.fillWidth: true
                            spacing: 8

                            Text {
                                text: "Метки"
                                color: textMain
                                font.pixelSize: 15
                                font.bold: true
                            }

                            Rectangle {
                                Layout.fillWidth: true
                                Layout.preferredHeight:
tagsFlow.implicitHeight + 18
                                Layout.minimumHeight: 48
                                radius: 12
                                color: "#151A23"
                                border.width: 1
                                border.color: borderColor
                                clip: true

                                Flow {
                                    id: tagsFlow

                                    anchors.left: parent.left
                                    anchors.right: parent.right
                                    anchors.top: parent.top
                                    anchors.leftMargin: 12
                                    anchors.rightMargin: 12
                                    anchors.topMargin: 9

                                    spacing: 8

                                    Repeater {
                                        model: root.tagsModel

                                        delegate: Rectangle {
                                            id: tagChip

                                            property bool hovered: false
```

```
                                        property     string     tagText:
modelData

                                        width:  tagLabel.implicitWidth +
(hovered ? 34 : 20)
                                        height: 28
                                        radius: 9

                                        color:  hovered  ?  "#33234F"  :
accentSoft
                                        border.width: 1
                                        border.color: hovered ? accent :
"#4C2D83"


                                        Behavior on width {
                                            NumberAnimation {
                                                duration: 120
                                                easing.type:
Easing.OutCubic
                                            }
                                        }

                                        RowLayout {
                                            anchors.fill: parent
                                            anchors.leftMargin: 10
                                            anchors.rightMargin: 8
                                            spacing: 6

                                            Text {
                                                id: tagLabel

                                                text:        "#"        +
tagChip.tagText
                                                color: accent
                                                font.pixelSize: 13
                                                font.bold: true
                                            }

                                            Text {
                                                visible: tagChip.hovered
                                                text: "×"
                                                color: textMuted
                                                font.pixelSize: 15
                                                font.bold: true

                                                MouseArea {
                                                    anchors.fill: parent
                                                    cursorShape:
Qt.PointingHandCursor

                                                    onClicked: {
```

```
root.removeTagRequested(tagChip.tagText)
                                                }
                                            }
                                        }
                                    }

                                    HoverHandler {
                                        onHoveredChanged:
tagChip.hovered = hovered
                                    }
                                }
                            }

                            Rectangle {
                                id: addTagChip

                                width: root.addingTag ? 190 : 38
                                height: 30
                                radius: 10

                                color: root.addingTag
                                       ? "#111722"
                                       : addTagMouse.containsMouse
                                         ? "#202738"
                                         : "#171C24"

                                border.width: 1
                                border.color: root.addingTag
                                              ? accent
                                              :
addTagMouse.containsMouse
                                                ? "#3A4254"
                                                : borderColor

                                Behavior on width {
                                    NumberAnimation {
                                        duration: 160
                                        easing.type: Easing.OutCubic
                                    }
                                }

                                Text {
                                    visible: !root.addingTag
                                    anchors.centerIn: parent
                                    text: "+"
                                    color: addTagMouse.containsMouse
? accent : textMuted

                                    font.pixelSize: 22
                                    font.bold: true
                                }
```

```
                                RowLayout {
                                    visible: root.addingTag
                                    anchors.fill: parent
                                    anchors.leftMargin: 10
                                    anchors.rightMargin: 6
                                    spacing: 6

                                    Text {
                                        text: "#"
                                        color: accent
                                        font.pixelSize: 13
                                        font.bold: true
                                    }

                                    TextField {
                                        id: tagInput

                                        Layout.fillWidth: true
                                        text: root.newTagText
                                        placeholderText: "метка"
                                        placeholderTextColor:
textMuted

                                        color: textMain
                                        font.pixelSize: 13
                                        selectByMouse: true
                                        background: null
                                        verticalAlignment:
TextInput.AlignVCenter

                                        onTextChanged: {
                                            root.newTagText = text
                                        }

                                        onAccepted: {
                                            root.submitTag()
                                        }

                                        Keys.onEscapePressed: {
                                            root.cancelTagInput()
                                        }

                                        onEditingFinished: {
                                            if   (root.addingTag   &&
text.trim() !== "")

                                                root.submitTag()
                                        }
                                    }

                                    Rectangle {
                                        Layout.preferredWidth: 24
                                        Layout.preferredHeight: 24
                                        radius: 8
```

```
                                        color:
saveTagMouse.containsMouse ? accentSoft : "transparent"

                                        Text {
                                            anchors.centerIn: parent
                                            text: "✓"
                                            color: accent
                                            font.pixelSize: 14
                                            font.bold: true
                                        }

                                        MouseArea {
                                            id: saveTagMouse
                                            anchors.fill: parent
                                            hoverEnabled: true
                                            cursorShape:
Qt.PointingHandCursor

                                            onClicked:
root.submitTag()
                                        }
                                    }

                                    Rectangle {
                                        Layout.preferredWidth: 24
                                        Layout.preferredHeight: 24
                                        radius: 8
                                        color:
cancelTagMouse.containsMouse ? "#242B38" : "transparent"

                                        Text {
                                            anchors.centerIn: parent
                                            text: "×"
                                            color: textMuted
                                            font.pixelSize: 16
                                            font.bold: true
                                        }

                                        MouseArea {
                                            id: cancelTagMouse
                                            anchors.fill: parent
                                            hoverEnabled: true
                                            cursorShape:
Qt.PointingHandCursor

                                            onClicked:
root.cancelTagInput()
                                        }
                                    }
                                }

                                MouseArea {
```

```
                                    id: addTagMouse

                                    anchors.fill: parent
                                    hoverEnabled: true
                                    cursorShape:
Qt.PointingHandCursor
                                    enabled: !root.addingTag

                                    onClicked: {
                                        root.addingTag = true
                                        root.newTagText = ""
                                        tagInput.forceActiveFocus()
                                    }
                                }
                            }
                        }
                    }
                }

            }
        }

        component PropertyRow: Rectangle {
            id: rowRoot

            property string iconName: ""
            property string title: ""
            property string value: ""
            property bool accentValue: false
            property bool mutedValue: false
            property bool showChevron: true
            property bool showClear: false
            property bool hovered: false

            signal clicked()
            signal clearClicked()

            Layout.fillWidth: true
            Layout.preferredHeight: 48
            radius: 13

            color: hovered ? "#1B222E" : "#151A23"
            border.width: 1
            border.color: hovered ? "#3A4254" : borderColor

            Behavior on color { ColorAnimation { duration: 120 } }
            Behavior on border.color { ColorAnimation { duration: 120
} }

            RowLayout {
                anchors.fill: parent
                anchors.leftMargin: 14
```

```
                anchors.rightMargin: 12
                spacing: 12

                Icon {
                    iconName: rowRoot.iconName
                    iconColor: textMuted
                    Layout.preferredWidth: 22
                    Layout.preferredHeight: 22
                }

                Text {
                    text: rowRoot.title
                    color: textMain
                    font.pixelSize: 15
                    Layout.preferredWidth: 116
                }

                Text {
                    Layout.fillWidth: true
                    text: rowRoot.value
                    color: rowRoot.mutedValue ? textMuted
                         : rowRoot.accentValue ? accent
                         : textMain
                    font.pixelSize: 15
                    font.bold: !rowRoot.mutedValue
                    elide: Text.ElideRight
                    horizontalAlignment: Text.AlignRight
                }

                Rectangle {
                    visible: rowRoot.showClear
                    Layout.preferredWidth: 24
                    Layout.preferredHeight: 24
                    radius: 8
                    color:  clearMouse.containsMouse  ?  "#242B38"  :
"transparent"

                    Icon {
                        anchors.centerIn: parent
                        width: 18
                        height: 18
                        iconName: "x"
                        iconColor: textMuted
                    }

                    MouseArea {
                        id: clearMouse
                        anchors.fill: parent
                        hoverEnabled: true
                        cursorShape: Qt.PointingHandCursor

                        onClicked: function(mouse) {
```

```
                            mouse.accepted = true
                            rowRoot.clearClicked()
                        }
                    }
                }

                Icon {
                    visible:          rowRoot.showChevron          &&
!rowRoot.showClear
                    iconName: "chevronDown"
                    iconColor: textMuted
                    Layout.preferredWidth: 20
                    Layout.preferredHeight: 20
                }
            }

            HoverHandler {
                onHoveredChanged: rowRoot.hovered = hovered
            }

            MouseArea {
                anchors.fill: parent
                z: -1
                cursorShape: Qt.PointingHandCursor
                onClicked: rowRoot.clicked()
            }
        }

        component Icon: Canvas {
            id: iconRoot

            property string iconName: ""
            property color iconColor: textMuted

            implicitWidth: 22
            implicitHeight: 22

            onIconNameChanged: requestPaint()
            onIconColorChanged: requestPaint()
            onPaint: {
                var ctx = getContext("2d")
                ctx.reset()
                ctx.clearRect(0, 0, width, height)

                ctx.strokeStyle = iconColor
                ctx.fillStyle = iconColor
                ctx.lineWidth = 1.8
                ctx.lineCap = "round"
                ctx.lineJoin = "round"

                var s = Math.min(width, height)
                var x = width / 2
```

```
        var y = height / 2

        if (iconName === "project") {
            ctx.beginPath()
            ctx.rect(4, 7, 14, 10)
            ctx.moveTo(4, 9)
            ctx.lineTo(8, 9)
            ctx.lineTo(10, 6)
            ctx.lineTo(14, 6)
            ctx.lineTo(16, 9)
            ctx.stroke()
        } else if (iconName === "flag") {
            ctx.beginPath()
            ctx.moveTo(6, 18)
            ctx.lineTo(6, 5)
            ctx.moveTo(7, 5)
            ctx.bezierCurveTo(10, 3, 13, 7, 17, 5)
            ctx.lineTo(17, 13)
            ctx.bezierCurveTo(13, 15, 10, 11, 7, 13)
            ctx.stroke()
        } else if (iconName === "calendar") {
            ctx.beginPath()
            ctx.rect(4, 5, 14, 14)
            ctx.moveTo(4, 9)
            ctx.lineTo(18, 9)
            ctx.moveTo(8, 3.5)
            ctx.lineTo(8, 6.5)
            ctx.moveTo(14, 3.5)
            ctx.lineTo(14, 6.5)
            ctx.stroke()
        } else if (iconName === "bell") {
            ctx.beginPath()
            ctx.moveTo(6, 15)
            ctx.quadraticCurveTo(8, 13, 8, 9)
            ctx.bezierCurveTo(8, 5, 14, 5, 14, 9)
            ctx.quadraticCurveTo(14, 13, 16, 15)
            ctx.lineTo(6, 15)
            ctx.stroke()

            ctx.beginPath()
            ctx.moveTo(10, 18)
            ctx.quadraticCurveTo(11, 19, 12, 18)
            ctx.stroke()
        } else if (iconName === "chevronDown") {
            ctx.beginPath()
            ctx.moveTo(6, 8)
            ctx.lineTo(11, 13)
            ctx.lineTo(16, 8)
            ctx.stroke()
        } else if (iconName === "x") {
            ctx.beginPath()
            ctx.moveTo(7, 7)
```

```
                ctx.lineTo(15, 15)
                ctx.moveTo(15, 7)
                ctx.lineTo(7, 15)
                ctx.stroke()
            } else if (iconName === "more") {
                ctx.beginPath()
                ctx.arc(11, 5.5, 1.3, 0, Math.PI * 2)
                ctx.arc(11, 11, 1.3, 0, Math.PI * 2)
                ctx.arc(11, 16.5, 1.3, 0, Math.PI * 2)
                ctx.fill()
            }
        }
    }

    component SmallButton: Rectangle {
        id: buttonRoot

        property string textValue: ""
        property bool hovered: false

        signal clicked()

        Layout.preferredWidth: 28
        Layout.preferredHeight: 28
        radius: 8
        color: hovered ? accentSoft : "transparent"

        Text {
            anchors.centerIn: parent
            text: buttonRoot.textValue
            color: textMain
            font.pixelSize: 16
            font.bold: true
        }

        HoverHandler {
            onHoveredChanged: buttonRoot.hovered = hovered
        }

        MouseArea {
            anchors.fill: parent
            cursorShape: Qt.PointingHandCursor
            onClicked: buttonRoot.clicked()
        }
    }

    component ReminderItem: Rectangle {
        id: reminderItem

        property string title: ""
        property bool selected: false
        property bool hovered: false
```

```
        signal clicked()

        width: parent.width
        height: 30
        radius: 8
        color: selected || hovered ? accentSoft : "transparent"

        Text {
            anchors.centerIn: parent
            text: reminderItem.title
            color: textMain
            font.pixelSize: 13
            font.bold: reminderItem.selected
        }

        HoverHandler {
            onHoveredChanged: reminderItem.hovered = hovered
        }

        MouseArea {
            anchors.fill: parent
            cursorShape: Qt.PointingHandCursor
            onClicked: reminderItem.clicked()
        }
    }

    component PriorityItem: Rectangle {
        id: priorityItem

        property string title: ""
        property bool selected: false
        property bool hovered: false
        property color colorValue: textMain

        signal clicked()

        width: parent.width
        height: 30
        radius: 8
        color: selected || hovered ? accentSoft : "transparent"

        Text {
            anchors.centerIn: parent
            text: priorityItem.title
            color: priorityItem.colorValue
            font.pixelSize: 13
            font.bold: priorityItem.selected
        }

        HoverHandler {
            onHoveredChanged: priorityItem.hovered = hovered
```

```
            }

            MouseArea {
                anchors.fill: parent
                cursorShape: Qt.PointingHandCursor
                onClicked: priorityItem.clicked()
            }
        }
    }
```

Файл TaskList.qml

```
    import QtQuick 2.15
    import QtQuick.Controls 2.15
    import QtQuick.Layouts 1.15

    Rectangle {
        id: root

        property var modelSource
        property int selectedTaskId: -1
        property alias visibleCount: listView.count

        property color appBg: "#080B12"
        property color cardBg: "#161B26"
        property color cardHover: "#1D2433"
        property color textMain: "#F9FAFB"
        property color textMuted: "#9CA3AF"
        property color accent: "#8B5CF6"
        property color accentSoft: "#2D2147"
        property color borderColor: "#2A3140"
        property color danger: "#EF4444"
        property color success: "#10B981"

        signal taskSelected(int rowIndex,
                            int taskIdValue,
                            string titleValue,
                            string projectValue,
                            string deadlineValue,
                            int priorityValue,
                            bool hasReminderValue,
                            int reminderMinutesBeforeValue,
                            bool doneValue)

        signal taskDoneToggled(int rowIndex, int taskIdValue)
        signal taskPriorityChanged(int rowIndex, int taskIdValue, int
priorityValue)
        signal taskRemoved(int rowIndex)

        Layout.fillWidth: true
        Layout.fillHeight: true
        color: "transparent"
```

```
        function formatDeadlineForUi(deadlineText) {
            if (deadlineText === "") return "Нет"
            var parts = deadlineText.split(" ")
            var dateParts = parts[0].split("-")
            return  dateParts[2]  +  "."  +  dateParts[1]  +  "."  +
dateParts[0] + " " + parts[1]
        }

        ListView {
            id: listView
            anchors.fill: parent
            spacing: 12
            clip: true
            model: root.modelSource

            delegate: TaskCard {
                width: listView.width

                property int modelTaskId: taskId
                property string modelTitle: title
                property string modelProject: project
                property string modelDeadlineText: deadlineText
                property bool modelHasDeadline: hasDeadline
                property bool modelDone: done
                property bool modelIsOverdue: isOverdue
                property int modelPriority: priority
                property bool modelHasReminder: hasReminder
                property       int       modelReminderMinutesBefore:
reminderMinutesBefore
                property string modelTagsText: typeof tagsText ===
"undefined" ? "" : tagsText

                rowIndex: index
                taskIdValue: modelTaskId
                selected: root.selectedTaskId === modelTaskId

                titleText: modelTitle
                projectText: modelProject
                deadlineTextValue:
root.formatDeadlineForUi(modelDeadlineText)
                hasDeadlineValue: modelHasDeadline
                overdue: modelIsOverdue
                doneValue: modelDone
                priorityValue: modelPriority
                tagsTextValue: modelTagsText

                appBg: root.appBg
                cardBg: root.cardBg
                cardHover: root.cardHover
                textMain: root.textMain
                textMuted: root.textMuted
```

```
                accent: root.accent
                accentSoft: root.accentSoft
                borderColor: root.borderColor
                danger: root.danger
                success: root.success

                onSelectedTask:                function(cardRowIndex,
cardTaskIdValue) {
                    console.log("SELECTED   TASK:",   cardRowIndex,
cardTaskIdValue, modelTitle)

                    root.taskSelected(
                        cardRowIndex,
                        cardTaskIdValue,
                        modelTitle,
                        modelProject,
                        modelDeadlineText,
                        modelPriority,
                        modelHasReminder,
                        modelReminderMinutesBefore,
                        modelDone
                    )
                }

                onToggleDone:                  function(cardRowIndex,
cardTaskIdValue) {
                    root.taskDoneToggled(cardRowIndex,
cardTaskIdValue)
                }

                onChangePriority:              function(cardRowIndex,
cardTaskIdValue, newPriorityValue) {
                    root.taskPriorityChanged(cardRowIndex,
cardTaskIdValue, newPriorityValue)
                }

                onRemoveTask: function(cardRowIndex) {
                    root.taskRemoved(cardRowIndex)
                }
            }
        }
    }
```

**Приложение В**

**(обязательное)**

Приложение для ведения и контроля задач

Презентация

# Многофункциональный персональный менеджер задач

Выполнил студент группы 22ВВП1
Сергунов Максим Романович
Научный руководитель
к.т.н. доцент кафедры ВТ Токарев А.Н.

Пенза 2026

## Цель работы

**Разработать локальное приложение для удобного ведения и контроля задач.**

- создание настольного приложения для ведения и контроля задач;
- локальное хранение данных без необходимости постоянного подключения к сети;
- работа с проектами, приоритетами, дедлайнами, тегами и подзадачами;
- наглядное отображение задач в виде списка и Kanban-доски.

## Актуальность

Тема актуальна, потому что пользователю нужен простой инструмент планирования, который работает локально и не перегружен лишними ограничениями.

### Проблема

- задачи часто хранятся в заметках, мессенджерах и памяти;
- из-за этого теряются сроки, приоритеты и общий контроль.

### Недостатки аналогов

- многие решения зависят от облака и интернета;
- часть полезных функций доступна только после регистрации или подписки.

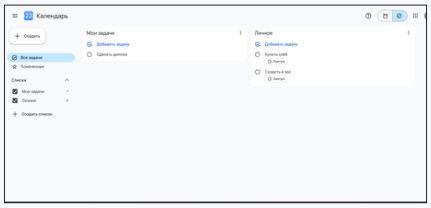
### Почему это важно?

- нужен простой локальный инструмент для ежедневного планирования;
- приложение должно быть доступным, наглядным и удобным в использовании.

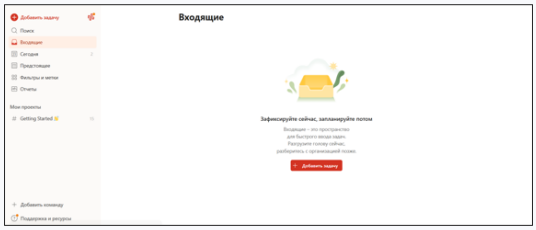
## Аналоги

Trello

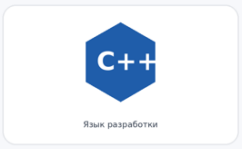
Google Tasks



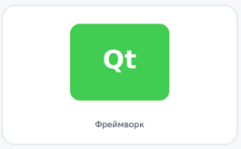
Todoist



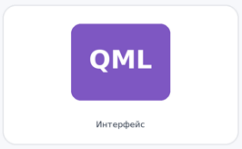

## Стек технологий



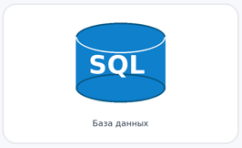

C++



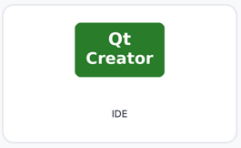

Qt




QML



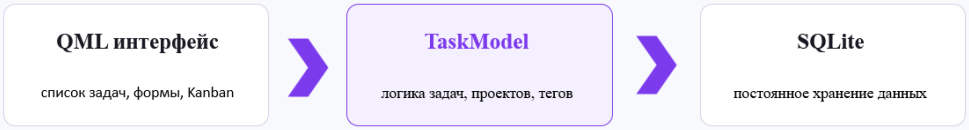

SQLite



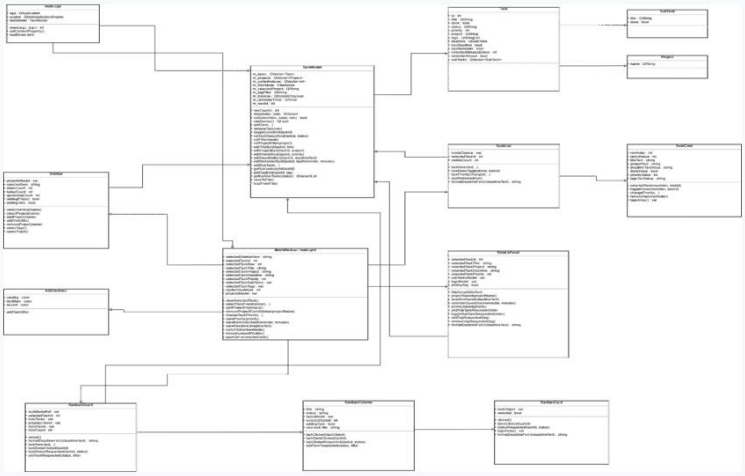

Qt Creator

## Диаграмма прецедентов

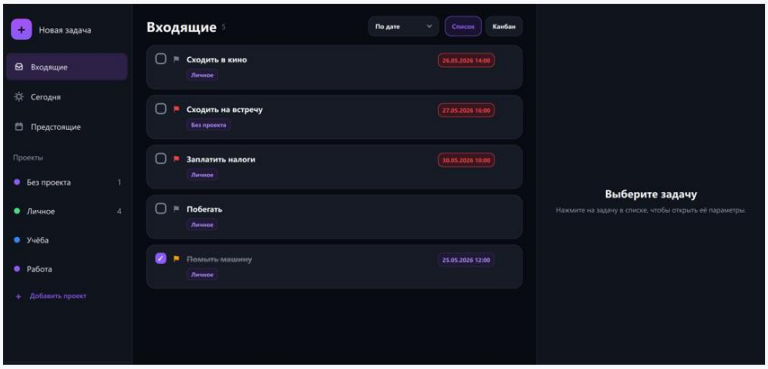
## Архитектура приложения



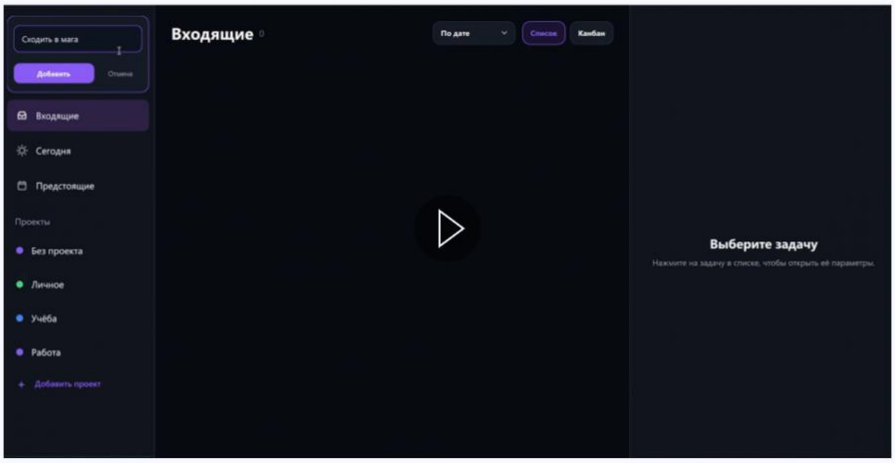

## Структура и модель данных



## Главный экран приложения

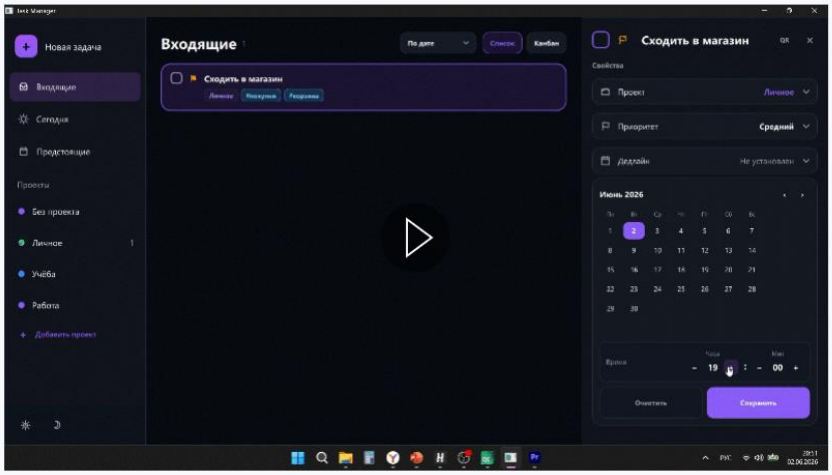


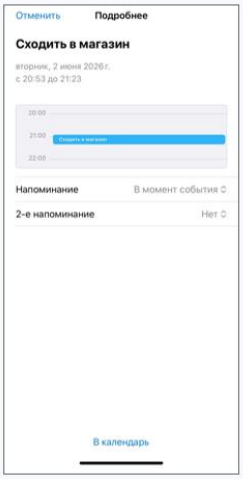
## Создание и редактирование задачи




9

## Дедлайн, напоминание и QR-код

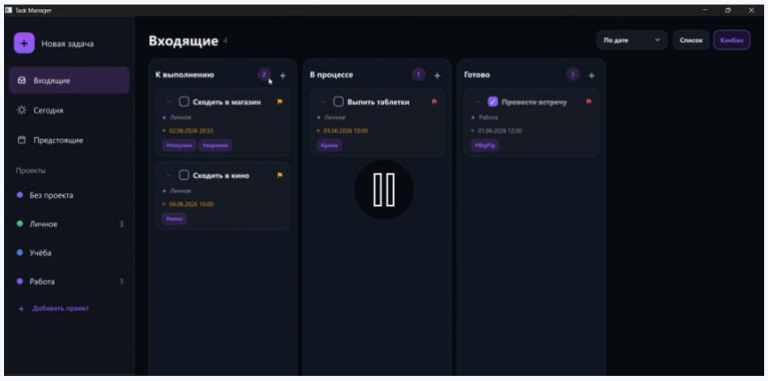



Результат сканирования QR-кода

10

## Kanban-доска




11

## Спасибо за внимание!

13