Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Языки программирования (ЯП)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

на тему

**«Менеджер Задач»**

БГУИР КР 1-40 01 01 215 ПЗ

Студент: 851002 Цыбулько К. Д.

Руководитель: асс. Шостак Е.В.

Минск 2019

Задание

# **СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 5](#_Toc27344043)

[**1.АНАЛИЗ ПРОТОТИПОВ, ЛИЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМУ ПРОГРАММНОМУ СРЕДСТВУ** 6](#_Toc27344044)

[1.1 Анализ программных средств планирования 6](#_Toc27344045)

[1.1.1 Менеджер задач Trello 6](#_Toc27344046)

[1.1.2 Менеджер задач LeaderTask 7](#_Toc27344047)

[1.1.3 Менеджер задач TickTick 8](#_Toc27344048)

[1.1.4 Менеджер задач Todoist 9](#_Toc27344049)

[1.2 Формирование требований к проектируемому программному средству 10](#_Toc27344050)

[**2. МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФЦНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ** 11](#_Toc27344051)

[2.1 Описание функциональных требований к программному средству 11](#_Toc27344052)

[2.2 Спецификация функциональных требований 11](#_Toc27344053)

[**3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА** 13](#_Toc27344054)

[3.1 Обобщённый алгоритм работы программного средства 13](#_Toc27344055)

[3.3 Основные алгоритмы работы с базой данных 14](#_Toc27344056)

[3.2 Проектирование структур данных 15](#_Toc27344057)

[3.4 Проектирование и разработка пользовательских элементов интерфейса управления 16](#_Toc27344058)

[**3.4.1 Проектирование пользовательского элемента «Календарь»** 17](#_Toc27344059)

[**3.4.2 Проектирование пользовательского элемента отображения сегодняшней даты** 17](#_Toc27344060)

[**3.4.3 Проектирование пользовательских элементов Задачи и Расписание** 18](#_Toc27344061)

[**4. СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА** 23](#_Toc27344062)

[4.1 Организация базы данных 23](#_Toc27344063)

[4.2 Описание работы базы данных 24](#_Toc27344064)

[4.3 Организация работы с интерфейсом 26](#_Toc27344065)

[**5. ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА** 27](#_Toc27344066)

[**6. РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ** 34](#_Toc27344067)

[**ЗАКЛЮЧЕНИЕ** 38](#_Toc27344068)

[**Список использованной литературы** 39](#_Toc27344069)

[**ПРИЛОЖЕНИE 1** 40](#_Toc27344070)

[**ПРИЛОЖЕНИE 2** 41](#_Toc27344071)

[**ПРИЛОЖЕНИE 3** 48](#_Toc27344072)

[**ПРИЛОЖЕНИE 4** 50](#_Toc27344073)

[**ПРИЛОЖЕНИE 5** 52](#_Toc27344074)

[**ПРИЛОЖЕНИE 6** 53](#_Toc27344075)

[**ПРИЛОЖЕНИE 7** 54](#_Toc27344076)

[**ПРИЛОЖЕНИE 8** 56](#_Toc27344077)

[**ПРИЛОЖЕНИE 9** 57](#_Toc27344078)

[**ПРИЛОЖЕНИE 10** 62](#_Toc27344079)

[**ПРИЛОЖЕНИE 10** 63](#_Toc27344080)

[**ПРИЛОЖЕНИE 11** 64](#_Toc27344081)

# **ВВЕДЕНИЕ**

Система управления задачами ⎯ это набор организационных и технологических методов и инструментов, которые поддерживают управление проектами в организации и помогают повысить эффективность их реализации. Часто термин система управления задачами трактуют более узко как автоматизированную или информационную систему управления задачами,т.е.программу.

Сегодня люди все больше прибегают к использованию электронных планировщиков задач, поэтому темой моего курсового проекта является разработка программного средства «Менеджер задач». Менеджер задач – это программное обеспечение, направленное на ведение списка задач и создание отметок об их выполнении.

Данное ПС должно позволить пользователям правильно распоряжаться своим временем с помощью функции расписания и планировать задачи с указанием дедлайна.

Признаком хорошего менеджера задач являются:

* Мультиплатформенность. Хороший планировщик имеет клиенты как под десктопные, так и под мобильные платформы – чтобы можно было записывать задачи и заметки на ходу с телефона, а обрабатывать их уже за компьютером. Поэтому при разработке курсового проекта был выбран фреймворк Qt, который является кроссплатформенным.
* Удобный интерфейс. Удобство интерфейса – понятие индивидуальное. Однако если интерфейс приложения кажется неудачным или перегруженным деталями, а добавить задачу невозможно за 2-3 секунды, то менеджер задач не приживется.
* Гибкость и адаптируемость. Поскольку не существует стандартной и подходящей для всех системы организации задач и проектов, хороший планировщик должен быть гибким и оставлять пользователю возможность для адаптации под себя. Кому-то нужны теги, а кому-то достаточно одних списков. Хороший менеджер задач позволяет пользователю самостоятельно определиться с тем, что ему нужно.

Данная пояснительная записка содержит следующие основные разделы. В первом разделе выполнен анализ прототипов, литературных источников. Во втором разделе сформированы функциональные требования к проектируемому программному средству. Третий раздел посвящен проектированию программного средства и содержит схемы работы модулей программного средства. Четвертый раздел описывает этапы создания программного средства. Пятый раздел содержит набор тестов и сценарии тестирования. В шестом разделе описано руководство по установке и использованию. Завершающий раздел содержит выводы о проделанной работе.

# **1.АНАЛИЗ ПРОТОТИПОВ, ЛИЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМУ ПРОГРАММНОМУ СРЕДСТВУ**

## 1.1 Анализ программных средств планирования

С потребностью выносить свои планы и списки дел из головы во внешний источник хранения информации люди сталкиваются давно, и по мере ускорения темпов жизни необходимость использования планировщиков и списков задач только растет.

Перечисленные ниже менеджеры задач предназначены для индивидуального планирования.

## Менеджер задач Trello

*Trello* [1] – программа для управления проектами небольших групп, разработанная Fog Creek Software.

Внешне **Trello** – это рабочий стол, на котором расположены **доски (Board)**, которым можно присвоить любые имена и названия,**списки (Lists)** и **карточки (Cards)**. Последние можно беспрепятственно перетягивать и перемещать. Trello наследует принципы работы японский системы канбан, выгодно отличаясь от других to-do – менеджеров за счёт возможности видеть весь объём работы на одной доске.

Плюсы:

- удобный и понятный интерфейс;

- многофункциональность в использовании (можно оставить доску заданий приватной, открывать отдельные карточки задач для других пользователей или сделать доску полностью открытой);

- использование с десктопа, данные синхронизируются между разными устройствами;

- есть возможность добавлять абсолютно любые документы в описание к задачам.

Минусы:

- нельзя фиксировать в программе время, затраченное на задачу;

- нет возможности выводить отчеты по задачам;

- для Windows существует только веб-версия.

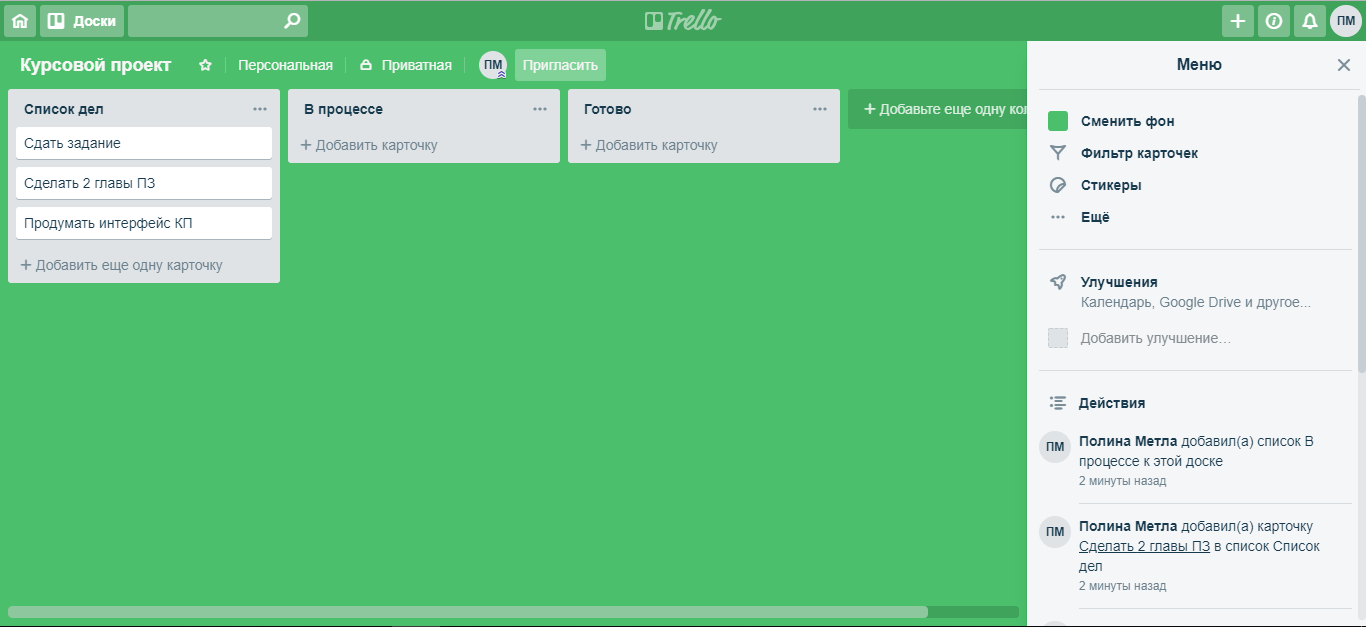


Рисунок 1.1 – Интерфейс приложения «Trello»

## Менеджер задач LeaderTask

*LeaderTask* [2] – то мультиплатформенный [органайзер](http://www.leadertask.ru/) с возможностью синхронизации данных между несколькими устройствами. Здесь для разделения задач применяется концепция проектов. Такой подход позволяет использовать сервис для любой задачи.

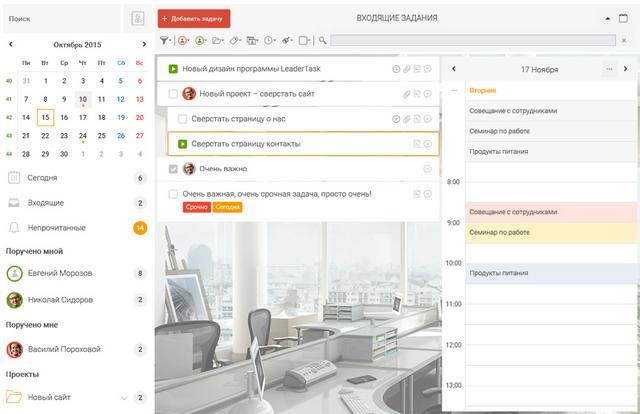


Рисунок 1.2 – Интерфейс приложения «LeaderTask»

Плюсы:

- удобный и простой интерфейс;

- многофункциональность и гибкость в применении (подойдет как для руководителя крупной компании, так и для простого студента);

- возможность создания подзадач и проектов;

- наличие функции зацикливания задач.

Минусы:

- отсутствие отчетов о выполнении задач;

- приложение является достаточно дорогим.

## 1.1.3 Менеджер задач TickTick

*TickTick* [3] –  todo менеджер и список дел на каждый день, органайзер gtd и планировщик mlo. Этот Менеджер синхронизирует задачи с веб-версией, приложениями для Android, iOS, расширениями для Chrome и Firefox.

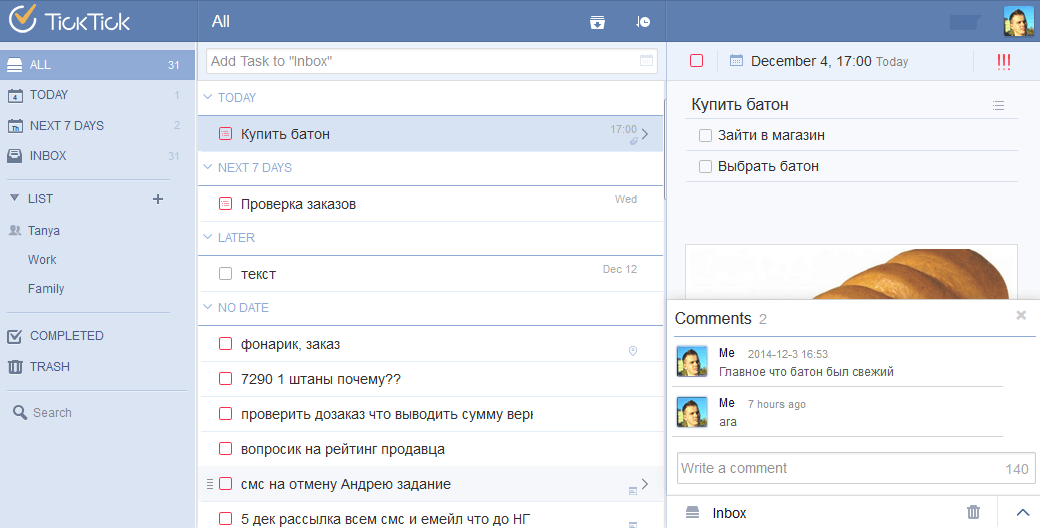


Рисунок 1.3 – Интерфейс приложения «TickTick»

Плюсы:

- напоминания по времени и местоположению;

- осуществление интеграции с календарем;

- классифицирование задачи метками;

- присутствуют различные параметры сортировки (по порядку, дате, имени, приоритету);

- добавление заметок или комментариев к задачам;

- быстрый поиск задач.

Минусы:

- отсутствие вложенности проектов;

- приложение является платным.

## 1.1.4 Менеджер задач Todoist

*Todoist* [4] – веб-сервис и набор программного обеспечения для управления задачами. Задачи могут также содержать заметки с файлами любого типа. Задачи можно помещать в проекты, сортировать по фильтрам, присваивать им метки, редактировать и экспортировать.

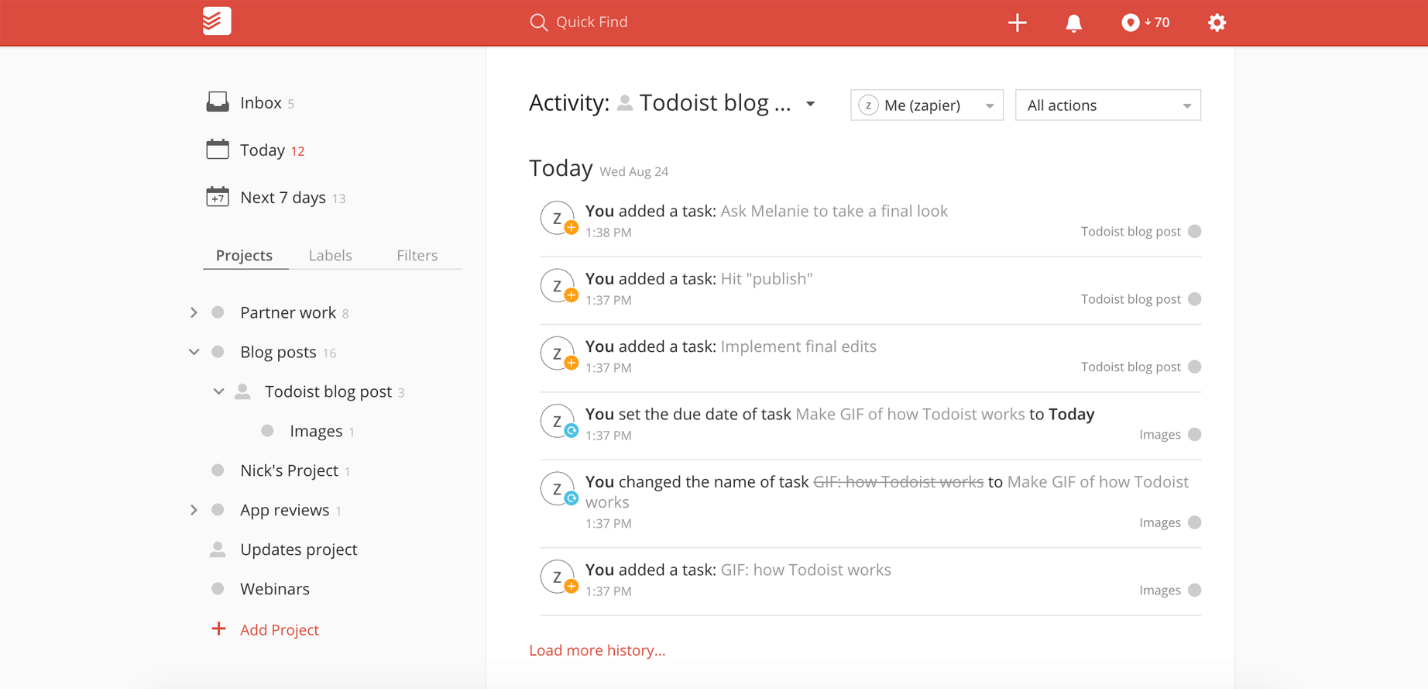


Рисунок 1.4 – Интерфейс приложения «Todoist»

Плюсы:

- задачи можно сортировать по категориям и приоритетности;

- можно открывать задачи и списки другим пользователям и вести их совместно;

- можно выводить отчеты по задачам.

- есть голосовые напоминания;

- можно отслеживать свою производительность и анализировать ее показатели.

Минусы:

- нельзя засекать прямо в программе время, затраченное на задачу в ручном или автоматическом режиме;

- нет статусов задач (типа «Делаю» / «Завершена» и т.д.), нет возможности вводить свои собственные статусы задач;

- многие функции (шаблоны проектов, резервная копия данных, уведомления на почту) доступны только в платной версии.

## 1.2 Формирование требований к проектируемому программному средству

Согласно заданию, на курсовой проект необходимо разработать программное средство менеджер задач. Основные функциональные возможности:

- иметь календарь с отображением месяца, года, числа и дня недели;

- свободное перемещение пользователя по календарю с отображением текущей (сегодняшней даты) и датой выбранной пользователем;

- отображение на календаре «помеченных дней» - дней, на которые запланированы какие-либо задачи;

- возможность составить расписание на день, состоящее из событий, где каждое событие имеет название, время начала, время конца и краткое описание;

- возможность редактирования временных рамок события в расписании;

- возможность добавления задач с указанием названия задачи, дедлайна ее выполнения (даты и времени) и кратким описанием;

- возможность редактирования дедлайна задачи и статуса ее выполнения;

- возможность удаления событий из расписания и задач;

- иметь удобный графический пользовательский интерфейс.

В качестве языка программирования был выбран С++ с использованием QML, для создания пользовательского интерфейса, и баз дынных SQL для хранения расписания и списка задач.

# **2. МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФЦНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ**

## 2.1 Описание функциональных требований к программному средству

Для функциональности ПС необходимо создание базы данных для хранения информации приложения. Она позволяет даже после закрытия приложения сохранить все изменения, внесенные пользователем. И при повторном открытии отображать прошлые изменения. Подробно о базе данных описано в главе 3.

Основными функциями ПС является добавление задач и составление расписания на день. Пользователь может просмотреть имеющиеся задачи, а также добавить личную задачу или событие на сегодня.

## 2.2 Спецификация функциональных требований

* Иметь календарь с отображением месяца, года, числа и дня недели.
* Свободное перемещение пользователя по календарю с отображением текущей (сегодняшней даты) и датой выбранной пользователем.

В программе имеется календарь, состоящий из элементов - дней, после нажатия, на которые работа с расписаниями и задачами начинает вестись именно с этими днями.

* Отображение на календаре «помеченных дней» - дней, на которые запланированы какие-либо задачи.

Если на день запланирована как минимум одна задача, то на угол элемента даты помещается красная пометка. При этом, если задача помечена как выполнена и не удалена из списка, то пометка отображена не будет.

* Возможность составить расписание на день, состоящее из событий, где каждое событие имеет название, время начала, время конца и краткое описание.

Так же присутствует обработка ошибок при вводе. Например, если пользователь вводит время окончания, которое раньше времени начала, то программное средство пометит поле ввода времени окончания события красным цветом.

* Возможность добавления задач с указанием названия задачи, дедлайна ее выполнения (даты и времени) и кратким описанием.
* Возможность редактирования временных рамок события в расписании.

При редактировании происходит сохранение данных в базе и пересортировка таблицы в интерфейсе, если требуется. Сама сортировка происходит по началу события.

* Возможность редактирования дедлайна задачи и статуса ее выполнения

При редактировании происходит сохранение данных в базе и пересортировка таблицы в интерфейсе, если требуется. Сама сортировка происходит сначала по дню, а потом на времени дедлайна.

* Возможность удаления событий из расписания и задач.
* Иметь удобный графический пользовательский интерфейс.

# **3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## 3.1 Обобщённый алгоритм работы программного средства

Так как для комфортного пользования необходим удобный и понятный пользовательский интерфейс, то для разработки клиентской части выбран фреймворк Qt, для построения дизайна и кроссплатформенной разметки будем использовать модуль фреймворка QtQuick.

Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML.

Для реализации поставленной задачи была создана локальная база данных. В программной реализации был создан объект базы данных db типа QSqlDatabase, с которым непосредственно и происходит работа. База данных состоит из 2 таблиц, отвечающих за список задач и событий расписания.

Схема алгоритма программы представлен в Приложении 1.

Для организации базы данных таблицы задач TABLE\_TASK были использованы поля, приведенные в Таблице 3.2.

Таблица 3.2

|  |  |  |
| --- | --- | --- |
| **Название** | **Назначение** | **Тип** |
| TABLE\_  TASK \_  START\_ID | Идентификатор задачи | int |
| TABLE\_TASK\_START\_DATE | Время начала выполнения задачи | VARCHAR(255) NOT NULL |
| TABLE\_TASK\_DEADLINE\_  TIME | Время дедлайна | VARCHAR(255) NOT NULL |
| TABLE\_TASK\_DEADLINE\_  DATE | День дедлайна | VARCHAR(255) NOT NULL |
| TABLE\_TASK\_TITLE | Название события | VARCHAR(255) NOT NULL |
| TABLE\_TASK\_DESC | Краткое описание задачи | VARCHAR(255) NOT NULL |

## 3.3 Основные алгоритмы работы с базой данных

Работа с обеими таблицами базы данных производится по одинаковым алгоритмам добавления и редактирования элементов, каждый из которых можно условно поделить на две части: внесение изменений в непосредственно саму базу данных и отображение изменений в пользовательском интерфейсе. Более подробные схемы внесения новых и редактирования уже имеющихся изображены на рисунках 3.1 и 3.2.

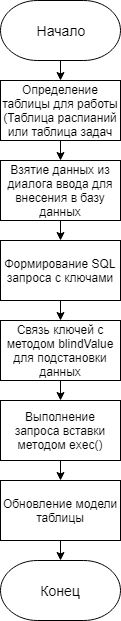


Рисунок 3.1 – Алгоритм добавления элемента в базу данных

## 3.2 Проектирование структур данных

Для организации базы данных таблицы расписания TABLE\_ SCHEDULE были использованы поля, приведенные в Таблице 3.1.

Таблица 3.1

|  |  |  |
| --- | --- | --- |
| **Название** | **Назначение** | **Тип** |
| TABLE\_  SCHEDULE\_  START\_ID | Идентификатор события | int |
| TABLE\_  SCHEDULE\_  START\_TIME | Время начала события | VARCHAR(255) NOT NULL |
| TABLE\_  SCHEDULE\_  END\_TIME | Время конца события | VARCHAR(255) NOT NULL |
| TABLE\_  SCHEDULE\_  WEEK | День события | VARCHAR(255) NOT NULL |
| TABLE\_  SCHEDULE\_  TITLE | Название события | VARCHAR(255) NOT NULL |
| TABLE\_  SCHEDULE\_  DESC | Краткое описание события | VARCHAR(255) NOT NULL |

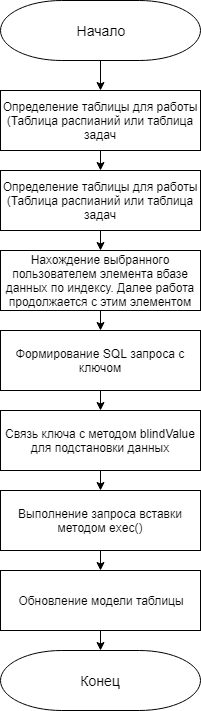


Рисунок 3.2 – Алгоритм редактирования элемента базы данных

## 3.4 Проектирование и разработка пользовательских элементов интерфейса управления

Для создания наиболее удобного пользовательского интерфеса в программном средстве использовались разработанные самостоятельно виджеты, которые позволили организовать взаимодействие между всеми элементами формы.

### **3.4.1 Проектирование пользовательского элемента «Календарь»**



Рисунок 4.1 – внешний вид пользовательского виджета «Календарь»

Данный элемент основан на базовом виджете Calendar, но при этом каждая его ячейка представляет собой элемент, который состоит из базовых компонентов Rectangle, Image и Label.

При выборе пользователем определенной даты, она выделяется цветом (при этом сегодняшний день выделяется более светлым цветом), и вся работа на форме происходит конкретно с этой датой. Пользователю предоставляется возможность добавлять задачи и расписание к этому дню.

Если на дату есть запланированные задачи, то она помечается красной меткой.

Код создания виджета представлен в файле main.qml в Приложении 2

### **3.4.2 Проектирование пользовательского элемента отображения сегодняшней даты**



Рисунок 4.2 – внешний вид отображения сегодняшнего дня

Элемент отображения даты состоит из базовых компонентов Rectangle и Label. Для отображения используется дата, выделенная в календаре, и меняется вместе с выделением новой даты.

Код создания виджета представлен в файле main.qml в Приложении 2

### **3.4.3 Проектирование пользовательских элементов Задачи и Расписание**

Обе таблицы строятся по одинаковому принципу: основываются на базовом элементе TableView, при этом вывод таблиц привязывается к базе данных и календарю.

Данные для таблиц выбираются из локальной базы данных в соответствии с условием (расписание – только на сегодняшнюю дату; задачи – на сегодняшнюю и все последующее) и сортируются по дате (для задач) и времени (для обеих таблиц).

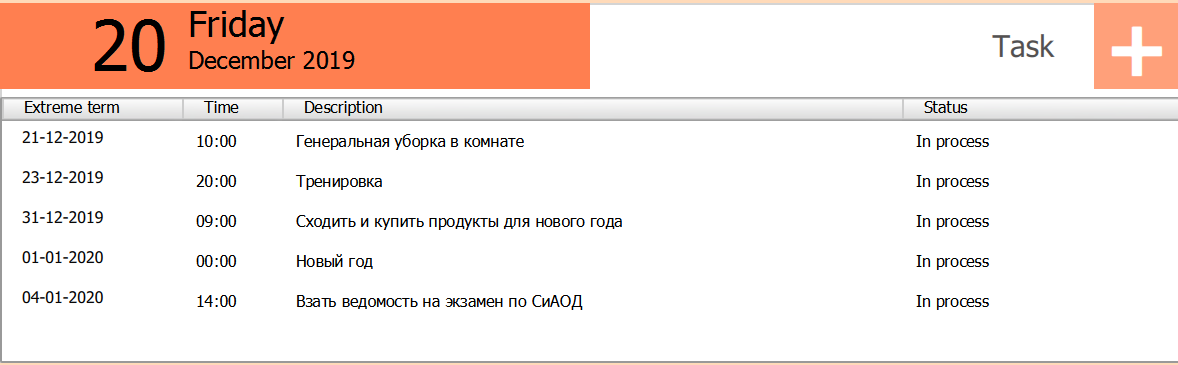


Рисунок 4.3а - внешний вид отображения таблицы задач

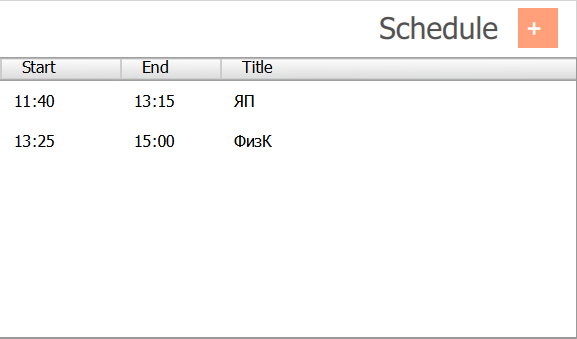


Рисунок 4.3б - внешний вид отображения расписания

При нажатии правой кнопки мыши на строку любой из таблиц, происходит вызов контекстного меню, с помощью которого можно просмотреть, изменить или удалить элемент.

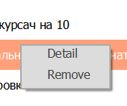


Рисунок 4.4 – внешний вид контекстного меню таблиц

При двойном клике на строку любой из таблиц, осуществляется переход в окно Detail (в которое так же можно попасть с помощью контекстного меню).

С помощью данного окна возможно редактирование данных: для расписания – редактирование начала и конца события, для задачи – дату, время дедлайна и статус задачи («В процессе» или «Выполнено»).

Оба диалоговых окна наследованы от главного и состоят из нескольких Label, Edit для редактирования полей, standardButton «OK» для сохранения изменений и CheckBox для изменения статуса (только для элементов таблицы задач).

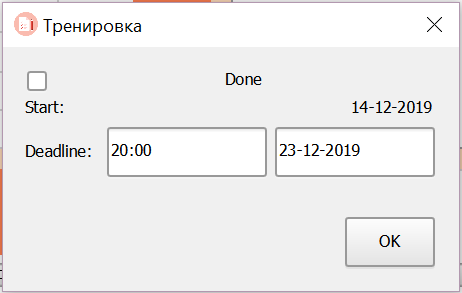


Рисунок 4.5а – внешний вид диалогового окна просмотра элемента таблицы задач

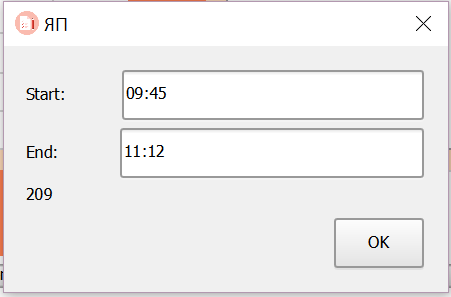


Рисунок 4.5б – внешний вид диалогового окна просмотра элемента таблицы расписания

Код создания диалоговых окон представлены в файлах DialogScheduleView.qml и DialogTaskView.qml в Приложениях 3 и 6

Диалоги удаления задачи или элемента расписания из таблиц абсолютно идентичные и состоят из одного Label и двух standardButtons «OK» и «Calcel» для отмены или подтверждения удаления элемента.

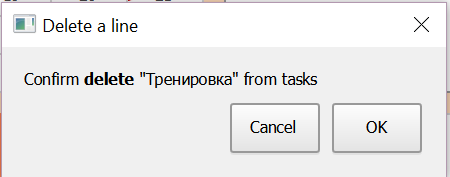


Рисунок 4.6 – внешний вид диалогового окна удаления элемента

Код создания диалоговых окон представлены в файлах DialogScheduleDelete.qml и DialogTaskDelete.qml в Приложениях 5 и 8

Кнопки для осуществления добавления элементов в таблицу состоят из стандартных компонентов Button и Label, при нажатии на которые открывается диалоговое окно добавления элемента в таблицу.

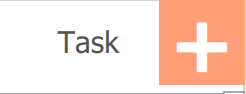


Рисунок 4.7а – внешний вид кнопки добавления элемента в таблицу задач



Рисунок 4.7б – внешний вид кнопки добавления элемента в таблицу расписания

Программная реализация кнопок представлена в файле main.qml в Приложении 2

Диалоговые окна добавления элемента содержат несколько Label и Edit, для внесения данных об элементе, который будет добавлен в базу данных. Так же диалоговые окна содержит стандартные кнопки для добавления элемента в базу данных или отмены добавления.

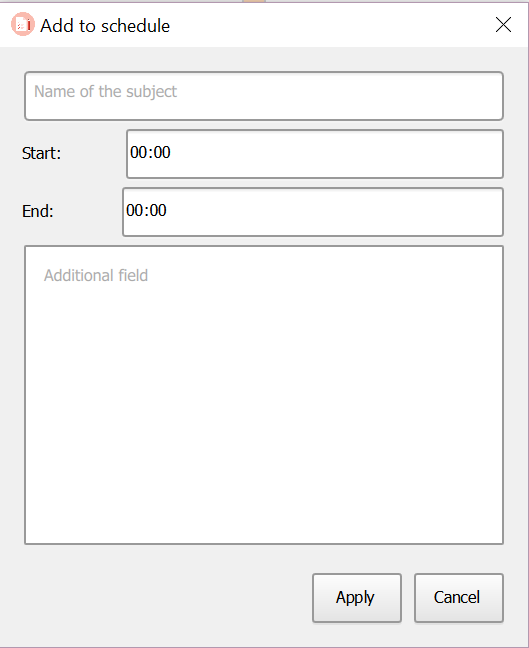


Рисунок 4.8а – внешний вид диалогового окна добавления элемента расписания

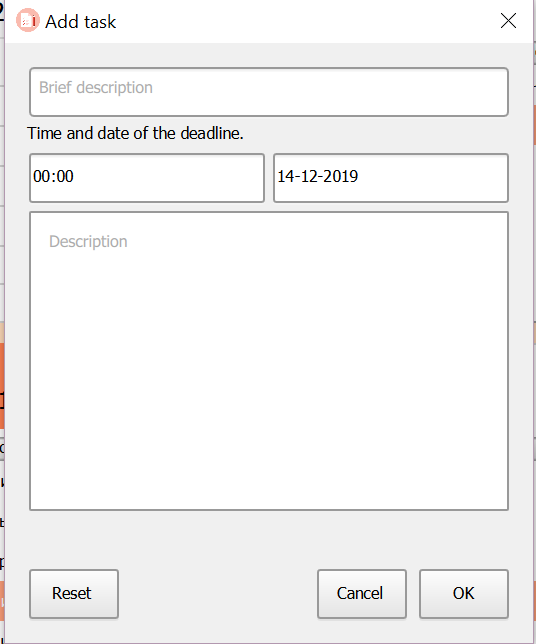


Рисунок 4.8б – внешний вид диалогового окна добавления задачи

Код создания диалоговых окон представлены в файлах DialogScheduleAdd.qml и DialogTaskAdd.qml в Приложениях 4 и 7

# **4. СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## 4.1 Организация базы данных

Разработка менеджера задач выполнялась на основе алгоритмов разработанных и представленных в разделе 3, а также на основе спецификации функциональных требований.

Полный листинг всех основных файлов программы приведен в Приложениях 2-11

В программном средстве используется одна база данных. Все основное взаимодействие происходит с ней.

В Таблице 4.1 представлены основные методы, вызываемые при начале и завершении работы с локальной базой данных.

Таблица 4.1. – Основные методы для внутренней работы с базой данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| openDataBase | Открытие базы данных | bool openDataBase(); | - | - |
| restoreDataBase | Восстановление базы данных | bool restoreDataBase(); | - | - |
| closeDataBase | Закрытие базы данных | void closeDataBase(); | - | - |
| createTable | Создание базы таблицы в базе данных | bool createTable(); | - | - |
| encodeDate | Преобразование даты в формат yyyyMMdd для базы данных | QString encodeDate(const QString &s); | s | Значение даты, которое нужно зашифровать для хранения в базу данных |

## 4.2 Описание работы базы данных

Таблица 4.2. – Основные методы добавления методов записей в таблицу базы данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| inserIntoTableTask | Вставка элемента в таблицу задач | bool inserIntoTableTask(const QVariantList &data); | data | Конвертируемый байт |
| inserIntoTableSchedule | Вставка элемента в таблицу расписания | bool inserIntoTableSchedule(const QVariantList &data); | - | - |
| inserIntoTable | Вставка элемента в базу данных, определение таблицы для вставки | bool inserIntoTable  (const QString &db, const QString &start, const QString &end,  const QString &date, const QString &title, const QString &desc, const QString &status); | db | Название таблицы базы данных |
| start | Время начала события (только для таблицы расписания) |
| end | Время конца события (для таблицы расписания) или время дедлайна (для таблицы задач) |
| date | Дата дедлайна или союытия |
| title | Название элемента |
| desc | Краткое описание элемента |
| status | Статус задачи (только для задачи) |
| removeRecord | Удаление записи из таблицы по её id | bool removeRecord(const QString &db, const QString &id); | db | Название таблицы базы данных |
| id | Индекс удаляемого элемента |

Продолжение таблицы 4.2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| searchTask | Проверка наличия записи в базе данных по дате | bool searchTask(const QString &date); | date | Дата записи |
| updatestatus | Обновление статуса задачи | void updatestatus(const QString &id, const QString &status); | id | Значение индекса элемента |
| status | Новое значение статуса задачи |
| updatedate | Обновление дня дедлайна задачи | void updatedate(const QString &id, const QString &deadlineday); | id | Значение индекса элемента |
| deadlineday | Новое значение даты дедлайна задачи |
| updatetime | Обновлении времени дедлайна задачи | void updatetime(const QString &id, const QString &deadlinetime); | id | Значение индекса элемента |
| deadlinetime | Новое значение времени дедлайна задачи |
| updatestart | Обновление времени начала события расписания | void updatestart(const QString &id, const QString &start); | id | Значение индекса элемента |
| start | Новое значение начала события расписания |
| updateend | Обновление времени конца события расписания | void updateend(const QString &id, const QString &end); | id | Значение индекса элемента |
| end | Новое значение конца события расписания |
| decodeDate | Преобразование даты в формат dd-MM-yyyy | QString decodeDate(const QString &s); | s | Значение даты, которое нужно расшифровать для вывода в таблицу |

## 4.3 Организация работы с интерфейсом

Таблица 4.3. – Основные методы работы с интерфейсными таблицами

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя подпрограммы** | **Описание** | **Заголовок подпрограммы** | **Имя параметра** | **Назначение параметра** |
| get | Получение значений по индексу из модели представления данных | QVariantMap get(int idx); | idx | Значение индекса элемента |
| updateModel | Метод обновления таблицы в модели представления данных | void updateModel(const QString &date); | date | Дата для которой нуждно вывести обновленную таблицу |
| data | Метод для получения данных из модели | QVariant data(const QModelIndex & index, int role) const; | index | Индекс модели представления данных |
| role | Номер роли |
| roleNames | Метод для получения имен ролей через хешированную таблицу | QHash<int, QByteArray> roleNames() cons | - | - |

# **5. ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА**

Тестирование - один из этапов разработки, поскольку при написании программного кода неизбежны. Тестирование – сравнение реального и ожидаемого поведения программы. Основная цель – выявление несоответствия функционирования программы спецификации и устранение их для повышения качества программного средства. Проводилось функциональное тестирование программы.

Функциональные тесты, проведенные над программой, представлены в таблице 5.1.

Таблица 5.1 – Результаты функционального тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Изменение даты | 1.Нажать на любую дату из календаря | Программа выделит ярким цветом выбранную дату | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 2 | Добавление задачи | 1. Нажать на кнопку добавления задачи  2. Ввести данные о задаче  3. Нажать на кнопку «ОК» | Программа обновит таблицу задач с добавлением только что введенной задачи | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 3 | Редактирование задачи | 1. Нажать на правую кнопку маши и из контекстного меню выбрать «Detail» или двойной клик по строке с задачей  2. Ввести новые данные дедлайна для задачи и изменить статус на «Done»  3. Нажать на кнопку «ОК» | Программа обновит таблицу задач с изменением данных о дедлайне данной задачи | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 4 | Удаление задачи | 1. Нажать на правую кнопку маши и из контекстного меню выбрать «Remove»  2. Нажать на кнопку «ОК» | Программа обновит таблицу задач без той задачи, которая была удалена | Тест пройден |
| 5 | Добавление события расписания | 1. Нажать на кнопку добавления события расписания  2. Ввести данные о событии  3. Нажать на кнопку «ОК» | Программа обновит расписание с добавлением только что введенного события | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 6 | Редактирование события расписания | 1. Нажать на правую кнопку маши и из контекстного меню выбрать «Detail» или двойной клик по строке с задачей  2. Ввести новые данные дедлайна для задачи  3. Нажать на кнопку «ОК» | Программа обновит таблицу задач с изменением данных о временных рамках события расписания | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 7 | Удаление события расписания | 1. Нажать на правую кнопку маши и из контекстного меню выбрать «Remove»  2. Нажать на кнопку «ОК» | Программа обновит расписание без того события, которое было удалено | Тест пройден |
| 8 | Сортировка расписания по времени начала события | 1. Нажать на кнопку добавления события расписания  2. Ввести данные о событии с более ранним временем начала, чем первое событие на сегодня  3. Нажать на кнопку «ОК» | Первым пунктом в расписании будет только что введенное событие | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Номер**  **теста** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 9 | Сортировка расписания по дате и времени дедлайна | 1. Нажать на кнопку добавления события расписания  2. Ввести данные о задаче с самым близким дедлайном  3. Нажать на кнопку «ОК» | Первой задачей в списке задач будет только что введенная | Тест пройден |
| 10 | Неправильный ввод времени окончания события в расписании | 1. Нажать на кнопку добавления события расписания  2. Ввести данные о событии с более ранним временем конца, чем время начала события  3. Нажать на кнопку «Apply» | Конец события будет подсвечен красным, событие не будет добавлено | Тест пройден |

Программа успешно прошла все тесты, что показывает корректность работы программы и соответствие функциональным требованиям.

# **6. РУКОВОДСТВО ПО ИСПОЛЬЗОВАНИЮ**

Для того, чтобы начать использовать программное средство, необходимо запустить файл A-Student.exe. После открытия программы появится окно, показанное на рисунке 6.1.

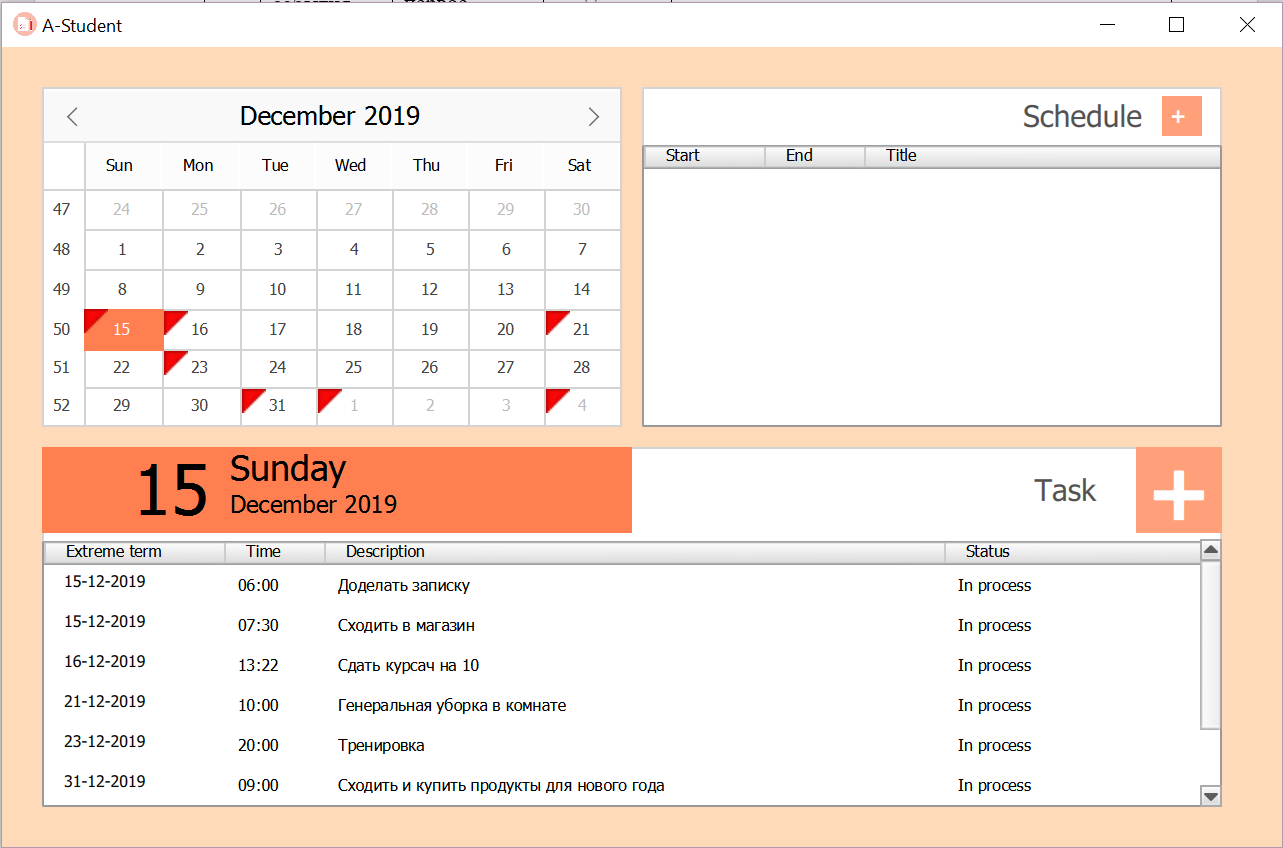


Рисунок 6.1 – Главная страница программы

Для изменения даты, с которой будет вестись работа, нужно кликнуть на любую дату на календаре – каждая ячейка представляет из себя объект, который связан со всеми остальными на форме.

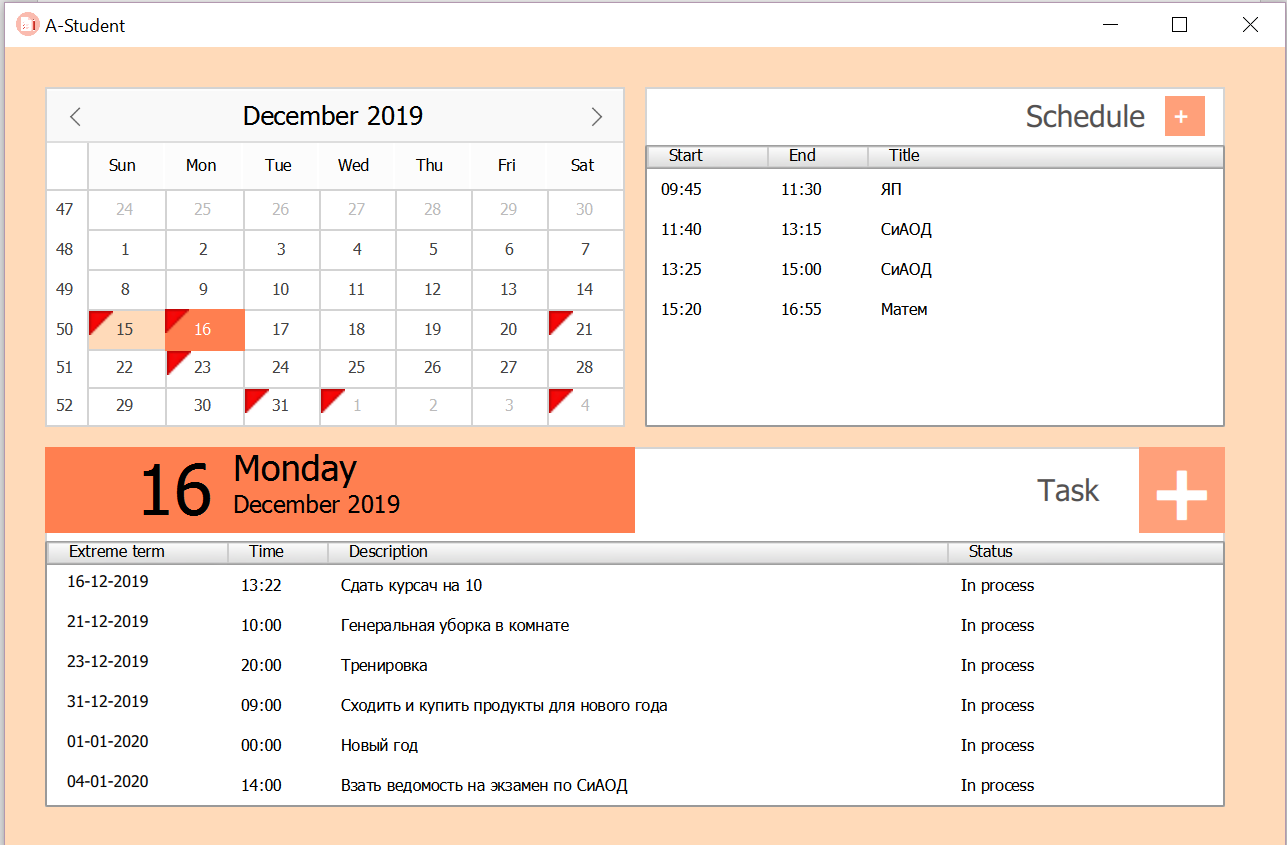


Рисунок 6.2 – Изменение даты

Для добавления задачи пользователю достаточно нажать на кнопку «+» рядом с надписью «Task», после этого появляется диалоговое окно ввода данных о задаче.

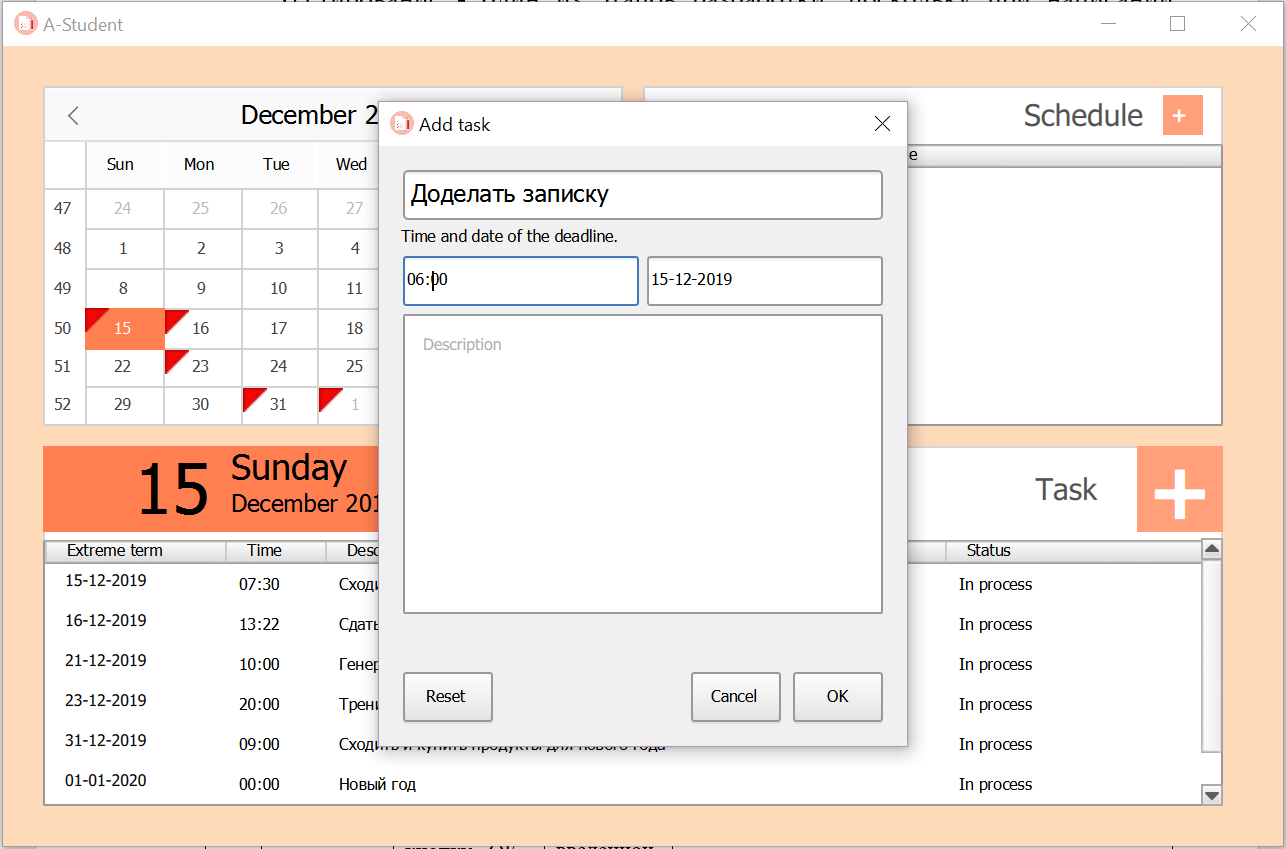


Рисунок 6.3 – Диалоговое окно добавления задачи

Для добавления события в расписание пользователю достаточно нажать на кнопку «+» рядом с надписью «Schedule», после этого появляется диалоговое окно ввода данных о событии.

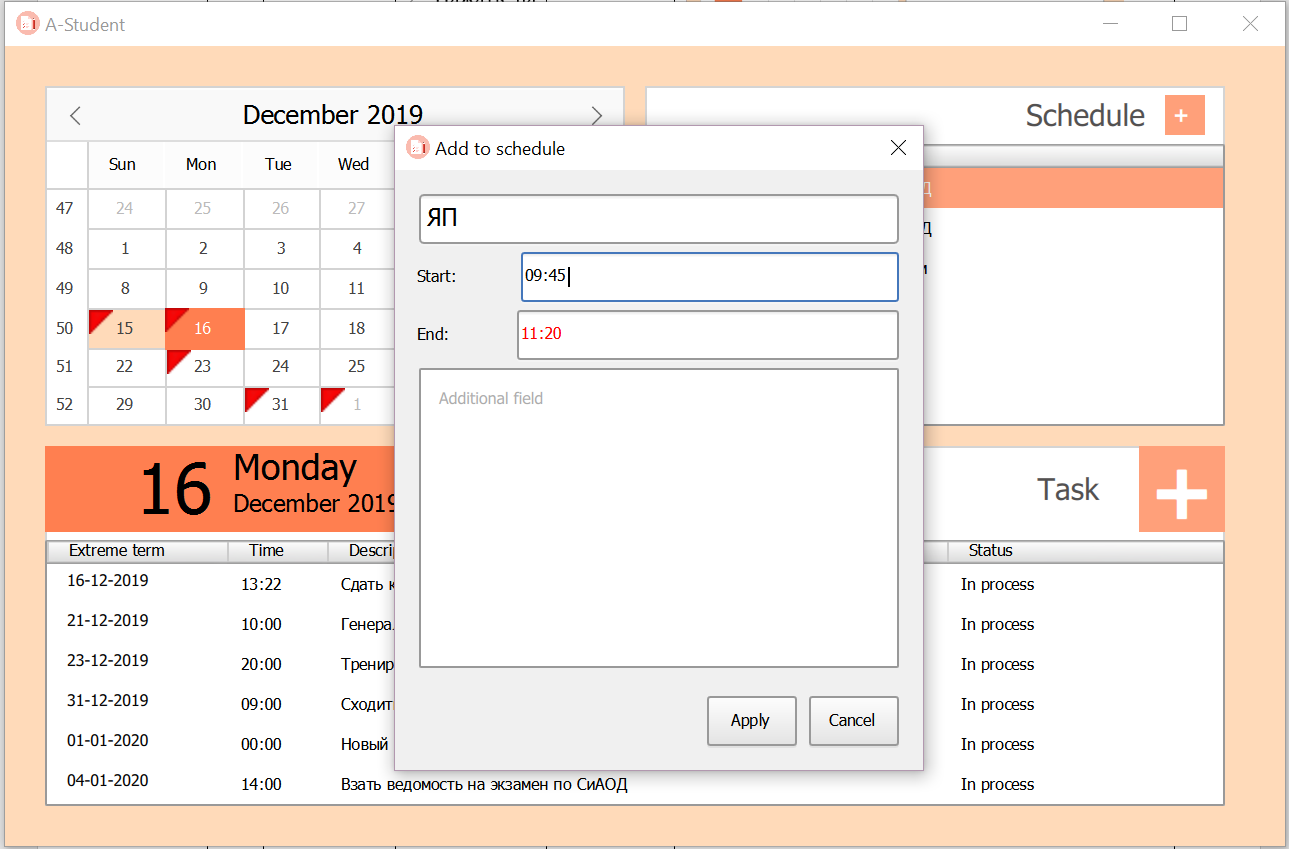


Рисунок 6.3 – Диалоговое окно добавления события в расписание

Для редактирования события расписания или задачи нужно перейти в контекстное меню (нажатие правой кнопки мыши по строке), и выбрать «Detail», осуществится переход в диалоговое окно просмотра/редактирования. Так же в это диалоговое окно можно попасть с помощью двойного клика по строке.

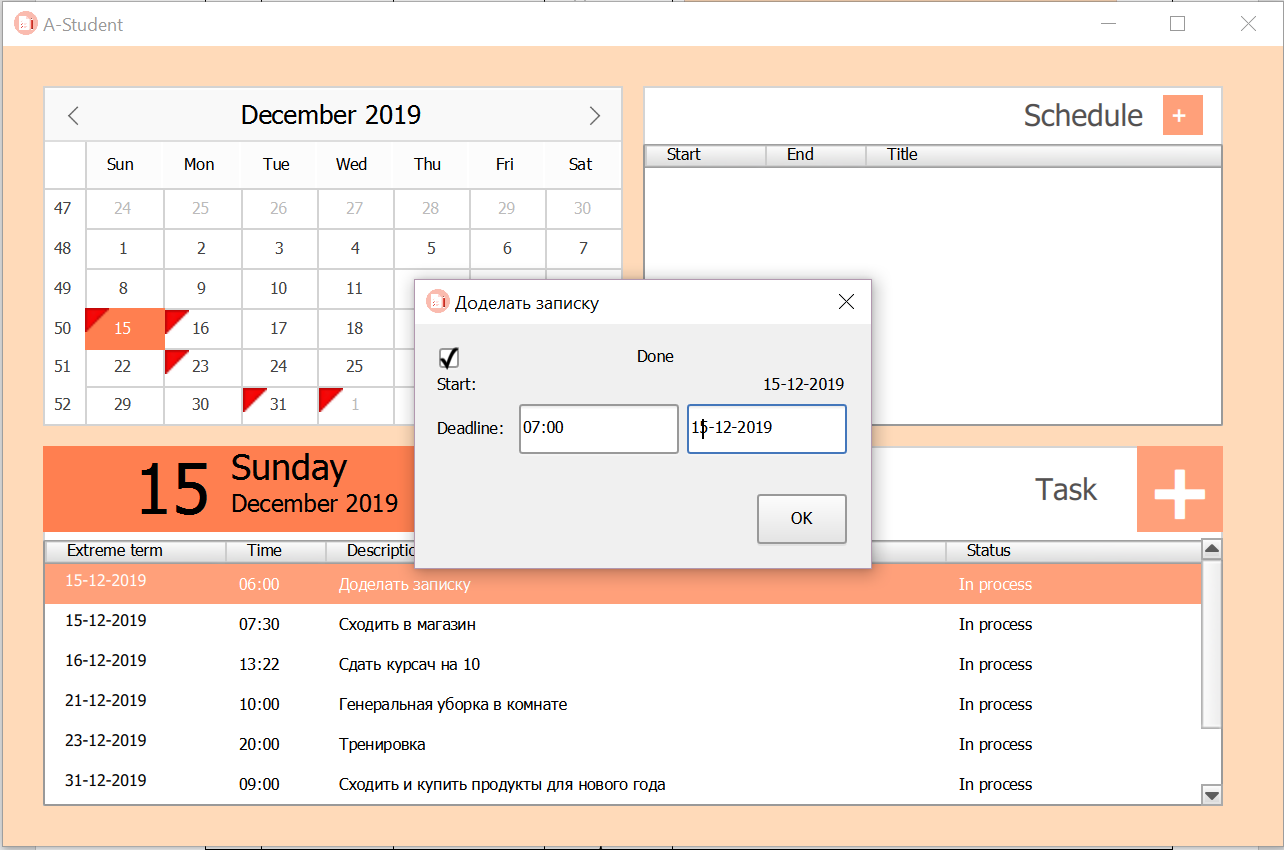


Рисунок 6.4а – Диалоговое окно редактирования задачи

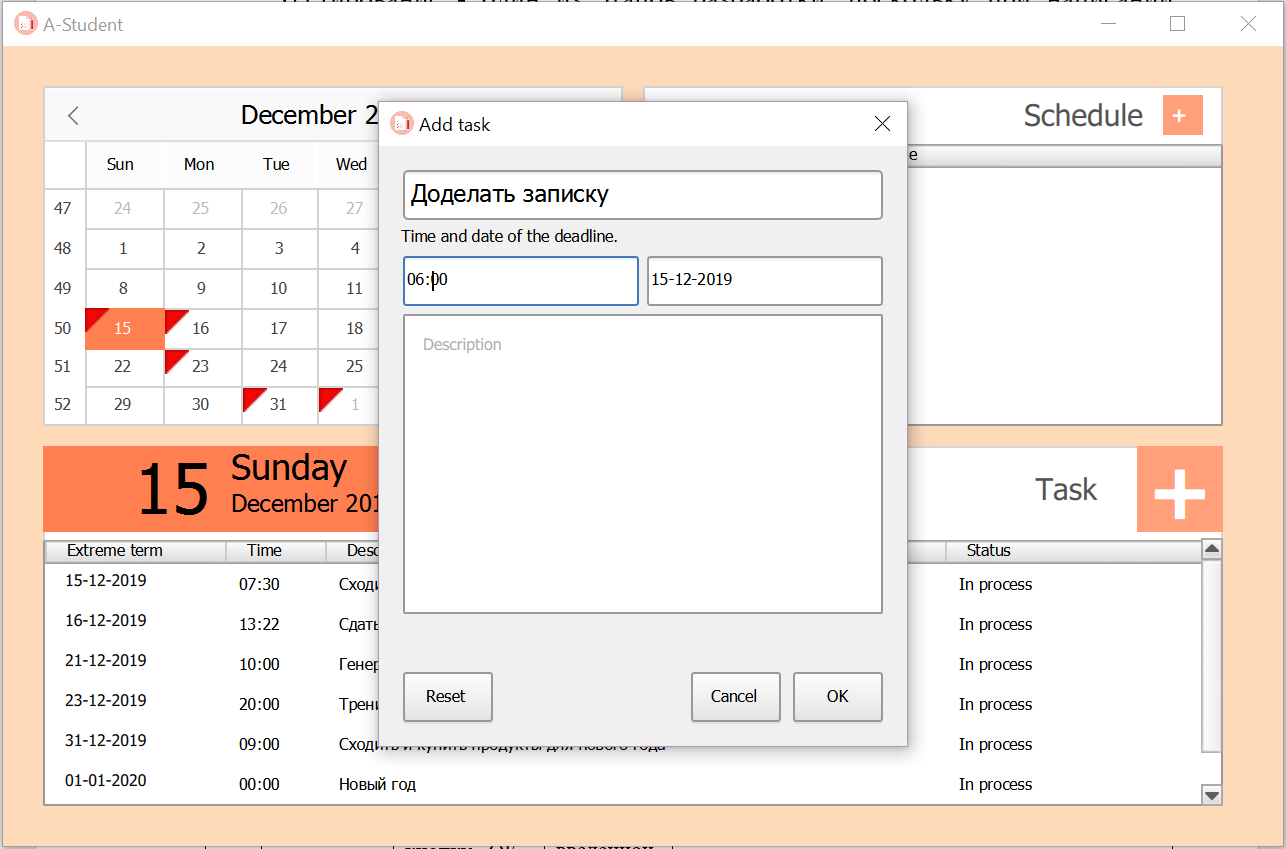


Рисунок 6.4б – Диалоговое окно редактирования события расписания

Для удаления события расписания или задачи нужно перейти в контекстное меню (нажатие правой кнопки мыши по строке), и выбрать «Remove». После этого появится диалоговое окна подтверждения удаления.

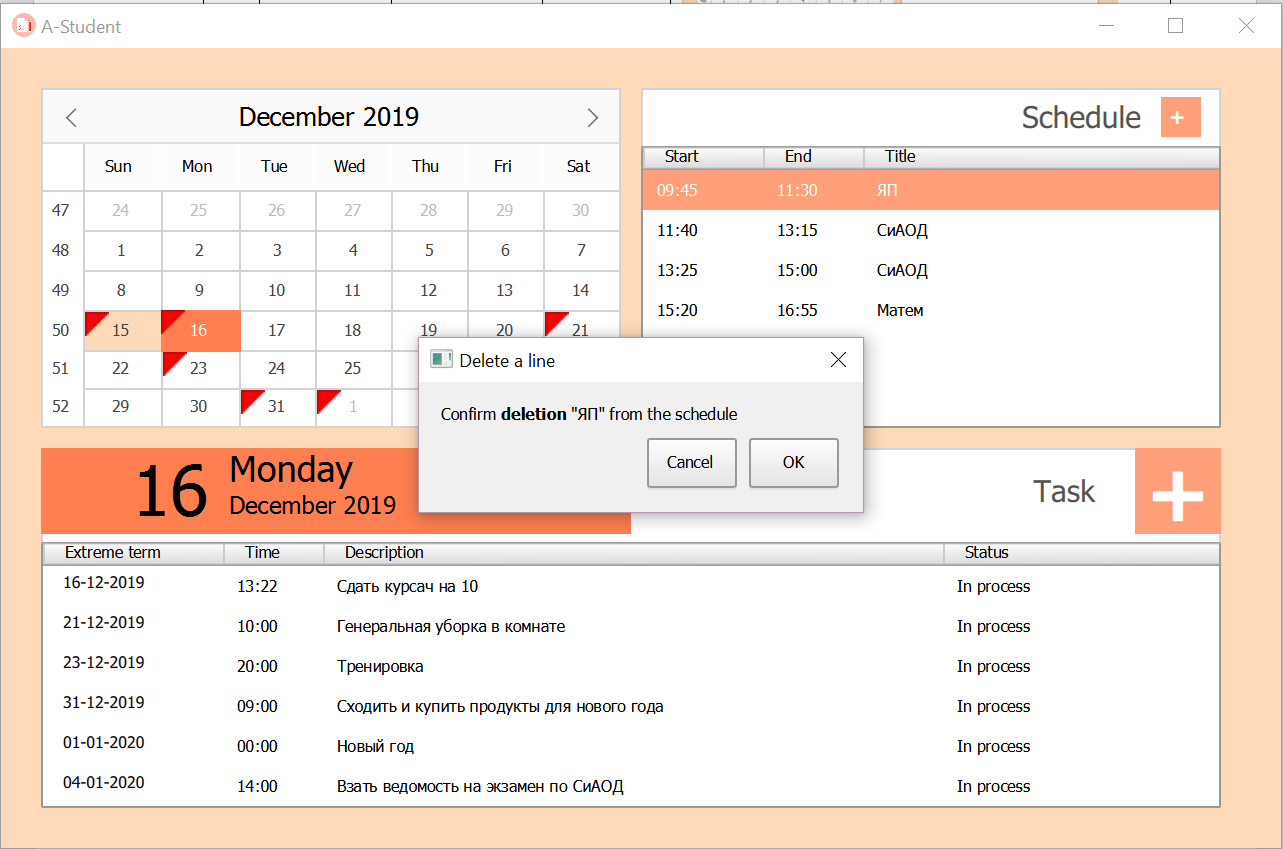


Рисунок 6.5а – Диалоговое окно подтверждения удаления события расписания

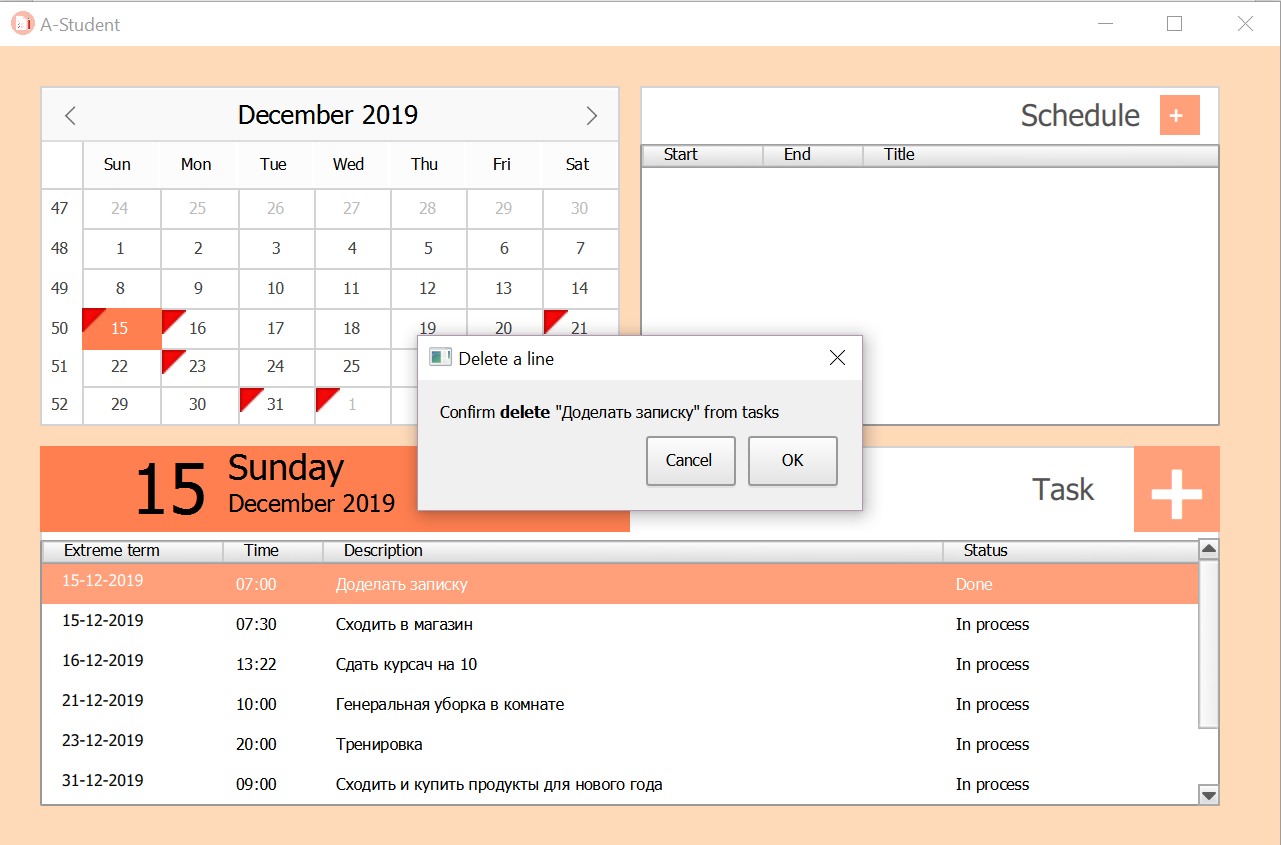


Рисунок 6.5б – Диалоговое окно подтверждения удаления задачи

# **ЗАКЛЮЧЕНИЕ**

В процессе выполнения курсового проекта была проанализирована предметная область, рассмотрены существующие аналоги и выявлены их преимущества и недостатки.

При разработке использовался фреймворк Qt, модули на языке программирования С++, базы данных SQL и модуль фреймворка QtQuick, для дизайна и кроссплатформенной разметки.

На этапе проектирования были разработаны: блок-схемы алгоритмов и графический интерфейс пользователя. С помощью создания собственных виджетов был спроектирован максимально комфортный и понятный пользовательский интерфейс.

В соответствии с поставленной целью было разработано приложение предоставляющее функционал менеджера задач, который позволяет составлять расписание на день, назначать дедлайны, отслеживать календарь запланированных дел. Разработанный менеджер отвечает трем основным критериям хорошего менеджера задач, приведенных в ведении, а именно: кроссплатформенность, удобный интерфейс, гибкость и адаптируемость. Данное программное средство будет удобно для студентов, так как разработанный менеджер имеет таблицу-расписание, которая может быть использовано как расписание занятий, таблицу с задачами, которую можно использовать для обозначения дедлайнов сдачи лабораторных или курсовых работ, и календарь, который позволит всегда иметь перед глазами ближайшие планы.

Проведено тестирование работоспособности разработанной программной части. Поставленная цель была выполнена в полном объеме, работоспособность подтверждена тестированием программного средства. Разработано руководство пользователя.

Разработанное программное средство имеет удобный, интуитивно понятный интерфейс и понятно даже для человека, который мало знаком с компьютером и почти его не использует в повседневной жизни, так как сам интерфейс максимально приближен к внешнему виду простого ежедневника.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

# **Список использованной литературы**

1. Менеджер задач Trello [Электронный ресурс] – Режим доступа : <https://trello.com/>
2. Менеджер задач LeaderTask [Электронный ресурс] – Режим доступа : <https://www.leadertask.ru/>
3. Менеджер задач TickTick [Электронный ресурс] – Режим доступа : <https://ticktick.com/>
4. Менеджер задач Todoist [Электронный ресурс] – Режим доступа : <https://todoist.com/>
5. Шлее М., Qt 5.10. Профессиональное программирование на С++. - СПб.: БХВ-Петербург, 2018. - 1072 с.: ил. - (В подлиннике)
6. Смелов В.В. Microsoft SQL Server 2008:основы Transact SQL / В.В. Смелов, Е.С. Мороз. - Мн.:БГТУ. - 2014.-440с.
7. Найт Б. Microsoft SQL Server 2008:руководство администратора для профессионалов / Б. Найт, К. Пэтел, В. Снайдер, Р.Лоффорт, С. Уорт - М.: Лори, 2004. - 560с.
8. Лафоре, Роберт Объектно-ориентированное программирование в C++ / Роберт Лафоре ; пер.: А. Кузнецов, М. Назаров, В. Шрага. - 4-е изд. - Санкт-Петербург [и др.] : Питер, 2017. - 923 с. : ил., схемы. - (Классика Computer science)
9. Summerﬁeld, Mark. Advanced Qt programming : creating great software with C++ and Qt 4 / Mark Summerﬁeld.
10. Седжвик Роберт, Фундаментальные алгоритмы на С++. Анализ/Структуры данных/ Сортировка/ Поиск Пер. с английского/Роберт Седжвик – К. : Издательство «ДиаСофт», 2001.-688 с.
11. Оффициальная документация по QT [Электронный ресурс] – Режим доступа : <https://doc.qt.io/qt-5/qtquick-index.html>

# **ПРИЛОЖЕНИE 1**

# **ПРИЛОЖЕНИE 2**

Файл main.qml

import QtQuick 2.6

import QtQuick.Window 2.2

import QtQuick.Controls 1.4

import QtQuick.Controls.Styles 1.1

import QtQuick.Layouts 1.1

import QtQuick.Dialogs 1.2

ApplicationWindow {

id: *applicationWindow*

visible: true

width: 640

height: 400

minimumWidth: 640

minimumHeight: 400

//color: "#f4f4f4"

color: "peachpuff"

property alias row: *row*

title: *qsTr*("A-Student ")

Flow {

id: *row*

anchors.fill: *parent*

clip: false

anchors.margins: 20

spacing: 10

layoutDirection: *Qt*.LeftToRight

// Слой с Календарем

Calendar {

id: *calendar*

// Язык в календаре

//locale: Qt.locale("en\_US")

width: (*parent*.width / 2) - 10

height: (*parent*.height / 2) - 10

frameVisible: true

weekNumbersVisible: true

focus: true

style: *calendarstyle*

}

// Стиль Item'ов в календаре

Component {

id: *calendarStyleItem*

Item {

property string curDate: *Qt*.formatDateTime(new *Date*(), "yyyyMMdd")

Rectangle {

anchors.fill: *parent*

color: if(styleData.date !== undefined && styleData.selected){

*modelSchedule*.updateModel(*Qt*.formatDate(*calendar*.selectedDate, "yyyyMMdd"))

*modelTask*.updateModel(*Qt*.formatDate(*calendar*.selectedDate, "yyyyMMdd"))

*tableSchedule*.selection.clear()

*tableSchedule*.currentRow =-1

*tableView*.selection.clear()

*tableView*.currentRow =-1

"#ff7f50"

}else if(styleData.date !== undefined &&

*Qt*.formatDate(styleData.date, "yyyyMMdd") == curDate) "peachpuff"

else "transparent"

anchors.margins: styleData.selected ? -1 : 0

Image {

visible: (*database*.searchTask(*Qt*.formatDate(styleData.date, "yyyyMMdd")) &&

*Qt*.formatDate(styleData.date, "yyyyMMdd") >= curDate)

anchors.margins: -1

width: 12

height: *width*

source: "qrc:/images/eventindicator.png"

}

}

Label {

text: styleData.date.getDate()

anchors.centerIn: *parent*

color: {

var *color* = '#dddddd';

if (styleData.valid) {

*color* = styleData.visibleMonth ? '#444' : '#bbb';

if (styleData.selected) *color* = 'white';

}

*color*;

}

}

}

}

// Стандартный стиль календаря

Component {

id: *calendarstyle*

CalendarStyle {

dayDelegate: *calendarStyleItem*

}

}

// Новый стиль календаря (нужно для обновления крайних сроков)

Component {

id: *newCalendarStyle*

CalendarStyle {

dayDelegate: *calendarStyleItem*

}

}

// Слой с Расписание

Rectangle {

width: (*parent*.width / 2) - 10

height: (*parent*.height / 2) - 10

border.color: *Qt*.darker(*color*, 1.2)

Text {

id: *scheduleLabel*

width: *parent*.width - *scheduleButton*.width - 20

y: 5

color: "#535151"

font.pixelSize: 15

horizontalAlignment: Text.AlignRight

verticalAlignment: Text.AlignVCenter

text: *qsTr*("Schedule")

}

Button {

id: *scheduleButton*

width: 20

height: *width*

anchors.top: *parent*.top

anchors.topMargin: (*scheduleLabel*.height / 2) - 5

anchors.right: *parent*.right

anchors.rightMargin: 10

style: ButtonStyle {

background: Rectangle {

border.width: control.activeFocus ? 2 : 1

border.color: control.pressed ? "#ff7f50" : "transparent"

color: '#ffa07a'

}

label: Text {

verticalAlignment: Text.AlignVCenter

font.bold: true

font.pointSize: 10

color: "white"

text: "+"

}

}

// Вызываем диалог для записи в базу данных

onClicked: {

*dialogAddShcedule*.defval()

*dialogAddShcedule*.open()

}

}

TableView {

id: *tableSchedule*

horizontalScrollBarPolicy: 1

anchors.topMargin: *scheduleLabel*.height + 10

anchors.fill: *parent*

model: *modelSchedule*

onWidthChanged:*scheduleTitle*.width = *Math*.max(100, *tableView*.width - *scheduleStart*.width + 10)

TableViewColumn {

id: *scheduleStart*

role: "schedulstart"

title: *qsTr*("Start")

width: 60

}

TableViewColumn {

id: *scheduleEND*

role: "schedulend"

title: *qsTr*("End")

width: 50

}

TableViewColumn {

id: *scheduleTitle*

role: "schedultitle"

title: *qsTr*("Title")

}

// Настройка строки в TableView для перехавата левого клика мыши

rowDelegate: Rectangle {

height: 20

color: styleData.selected ? "#ffa07a" : 'white';

MouseArea {

id: *mouseArea*

anchors.fill: *parent*

acceptedButtons: *Qt*.RightButton | *Qt*.LeftButton

onClicked: {

if(styleData.row >= 0) {

*tableSchedule*.selection.clear()

*tableSchedule*.selection.select(styleData.row)

*tableSchedule*.currentRow = styleData.row

}

*tableSchedule*.focus = true

/\* tableSchedule.selection.clear()

tableSchedule.selection.select(styleData.row)

tableSchedule.currentRow = styleData.row ? styleData.row : 0

tableSchedule.focus = true\*/

switch(*mouse*.button) {

case *Qt*.RightButton:

*contextSheduleMenu*.popup()

break

default:

break

}

}

onDoubleClicked: {

if (styleData.row >= 0)

*dialogScheduleView*.open()

}

}

}

}

}

// Слой с Задачами

Rectangle {

id: *rectangle*

width: *parent*.width - 10

height: *parent*.height / 2

border.color: *Qt*.darker(*color*, 1.2)

Text {

id: *taskLabel*

width: *parent*.width - *taskButton*.width - 20

height: *eventDay*.height

anchors.margins: 4

color: "#535151"

font.pixelSize: 15

horizontalAlignment: Text.AlignRight

verticalAlignment: Text.AlignVCenter

text: *qsTr*("Task")

}

// Вызываем диалог для записи в базу данных

Button {

id: *taskButton*

width: *eventDay*.height

height: *width*

anchors.top: *parent*.top

anchors.right: *parent*.right

style: ButtonStyle {

background: Rectangle {

border.width: control.activeFocus ? 2 : 1

border.color: control.pressed ? "#ff7f50" : "transparent"

color: "#ffa07a"

}

label: Text {

verticalAlignment: Text.AlignVCenter

horizontalAlignment: Text.AlignHCenter

font.bold: true

font.pointSize: 40

color: "white"

text: "+"

}

}

onClicked: {

*dialogAddTask*.defval()

*dialogAddTask*.open()

}

}

// Слой с выводом выбраной датой

Rectangle {

id: *rectangle1*

width: *calendar*.width + 5

height: *eventDay*.height

color: "#ff7f50"

Row {

id: *eventDate*

anchors.left: *parent*.left

anchors.leftMargin: *calendar*.width / 3 - 50

anchors.right: *parent*.right

anchors.rightMargin: *calendar*.width / 3

spacing: 10

Label {

id: *eventDay*

text: *calendar*.selectedDate.getDate()

font.pointSize: 35

}

Column {

height: *eventDay*.height

Label {

readonly property var options: { weekday: "long" }

text: *Qt*.locale("en\_US").standaloneDayName(*calendar*.selectedDate.getDay(), Locale.LongFormat)

font.pointSize: 18

}

Label {

text: *Qt*.locale("en\_US").standaloneMonthName(*calendar*.selectedDate.getMonth())

+ *calendar*.selectedDate.toLocaleDateString(*Qt*.locale("en\_US"), " yyyy")

font.pointSize: 12

}

}

}

}

TableView {

id: *tableView*

horizontalScrollBarPolicy: 1

anchors.topMargin: *eventDay*.height + 4

anchors.fill: *parent*

model: *modelTask*

onWidthChanged:*tasktitle*.width = *Math*.max(100, *tableView*.width - *endTime*.width -

*taskstatus*.width - *endDate*.width + 20)

TableViewColumn {

id: *endDate*

role: "enddate"

title: *qsTr*("Extreme term")

width: 90

delegate: Text {

x: 10

y: 3

text: *database*.decodeDate(styleData.value)

color: styleData.selected ? 'white' : 'black'

}

}

TableViewColumn {

id: *endTime*

role: "endtime"

title: *qsTr*("Time")

width: 50

}

TableViewColumn {

id: *tasktitle*

role: "tasktitle"

title: *qsTr*("Description")

}

TableViewColumn {

id: *taskstatus*

role: "taskstatus"

title: *qsTr*("Status")

}

// Настройка строки в TableView для перехавата левого клика мыши

rowDelegate: Rectangle {

height: 20

color:styleData.selected ? '#ffa07a' : 'white';

MouseArea {

anchors.fill: *parent*

acceptedButtons: *Qt*.RightButton | *Qt*.LeftButton

onClicked: {

if(styleData.row >= 0) {

*tableView*.selection.clear()

*tableView*.selection.select(styleData.row)

*tableView*.currentRow = styleData.row

}

*tableView*.focus = true

/\*tableView.selection.clear()

tableView.selection.select(styleData.row)

tableView.currentRow = styleData.row ? styleData.row : 0

tableView.focus = true\*/

switch(*mouse*.button) {

case *Qt*.RightButton:

if(styleData.row >= 0)

*contextTaskMenu*.popup() // Вызываем контексткное меню

break

default:

break

}

}

onDoubleClicked: {

if(styleData.row >= 0){

*dialogTaskView*.defval()

*dialogTaskView*.open()

}

}

}

}

}

// Контекстно меню предлагает удаление строки из базы данных

Menu {

id: *contextTaskMenu*

MenuItem {

text: *qsTr*("Detail")

onTriggered: {

/\* Вызываем диалоговое окно,

которое покажедетальную информацию о задании

\*/

*dialogTaskView*.defval()

*dialogTaskView*.open()

}

}

MenuItem {

text: *qsTr*("Remove")

onTriggered: {

*dialogTaskDelete*.open()

}

}

}

Menu {

id: *contextSheduleMenu*

MenuItem {

text: *qsTr*("Remove")

onTriggered: {

*dialogSchedueDelete*.open()

}

}

MenuItem {

text: *qsTr*("Detail")

onTriggered: {

//dialogScheduleView.defval()

*dialogScheduleView*.open()

}

}

}

// Определяем диалоговые окна

DialogTaskDelete {

id: *dialogTaskDelete*

}

DialogSchedueDelete {

id: *dialogSchedueDelete*

}

DialogTaskView {

id: *dialogTaskView*

}

DialogScheduleView {

id: *dialogScheduleView*

}

DialogTaskAdd {

id: *dialogAddTask*

}

DialogShceduleAdd {

id: *dialogAddShcedule*

}

}

}

}

# **ПРИЛОЖЕНИE 3**

Файл DialogTaskView.qml

import QtQuick 2.0

import QtQuick.Dialogs 1.2

import QtQuick.Controls 1.4

import QtQuick.Layouts 1.1

// Диалог подробного описания задания

Dialog {

title: *modelTask*.get(*tableView*.currentRow).tasktitle + ""

standardButtons: StandardButton.Ok

function *defval*() {

*doneTask*.checked = *modelTask*.get(*tableView*.currentRow).taskstatus === "Done" ? true : false

}

onAccepted: {

*database*.updatestatus(*modelTask*.get(*tableView*.currentRow).id, *doneTask*.checked ? "Done" : "In process")

*database*.updatedate(*modelTask*.get(*tableView*.currentRow).id, *dateTask*.text)

*database*.updatetime(*modelTask*.get(*tableView*.currentRow).id, *timeTask*.text)

*modelTask*.updateModel(*Qt*.formatDate(*calendar*.selectedDate, "yyyyMMdd"))

if(*calendar*.style === *newCalendarStyle*) *calendar*.style = *calendarstyle*

else *calendar*.style = *newCalendarStyle*

}

ColumnLayout {

width: *parent* ? *parent*.width : 100

CheckBox {

id: *doneTask*

Layout.fillWidth: true

text: *qsTr*("Done")

}

RowLayout {

Layout.alignment: *Qt*.AlignHCenter

Label {

text: *qsTr*("Start: ")

Layout.fillWidth: true

wrapMode: Text.WordWrap

}

Label {

Layout.alignment: *Qt*.AlignRight

text: *database*.decodeDate(*modelTask*.get(*tableView*.currentRow).startdate)

wrapMode: Text.WordWrap

}

}

RowLayout {

Layout.alignment: *Qt*.AlignHCenter

Label {

text: *qsTr*("Deadline: ")

Layout.fillWidth: true

wrapMode: Text.WordWrap

}

TextField{

id:*timeTask*

text: "<b>"+*modelTask*.get(*tableView*.currentRow).endtime+"</b> "

inputMask: "99:99"

inputMethodHints: *Qt*.ImhDigitsOnly

validator: RegExpValidator { regExp: /^([0-1]?[0-9]|2[0-3]):([0-5][0-9])$ / }

Layout.fillWidth: true

}

TextField{

id:*dateTask*

text: " <b>"+*database*.decodeDate(*modelTask*.get(*tableView*.currentRow).enddate)+"</b>"

inputMask: "99-99-9999"

inputMethodHints: *Qt*.ImhDigitsOnly

validator: RegExpValidator { regExp: /^([0-2]?[0-9]|3[0-1])-(0?[0-9]|1[0-2])-([2-9][0-9][0-9][0-9])$ / }

Layout.fillWidth: true

}

}

Label {

text: *modelTask*.get(*tableView*.currentRow).taskdesc + ""

Layout.fillWidth: true

Layout.fillHeight: true

y: 100

wrapMode: Text.WordWrap

}

}

}

# **ПРИЛОЖЕНИE 4**

Файл DialogTaskAdd.qml

import QtQuick 2.0

import QtQuick.Dialogs 1.2

import QtQuick.Controls 1.4

import QtQuick.Layouts 1.1

// Диалог добавления задач

Dialog {

height: 300

title: *qsTr*("Add task")

standardButtons: StandardButton.Ok | StandardButton.Cancel | StandardButton.Reset

function *defval*() {

*titleTask*.text = ""

*descriptionTask*.text = ""

*timeTask*.text = "00:00"

//dateTask.text = Qt.formatDate("2000-01-01 01:01:01.000000001 +1000", "dd-MM-yyyy")

*dateTask*.text = *Qt*.formatDate(calendar.selectedDate, "dd-MM-yyyy")

}

onAccepted: {

*database*.inserIntoTable("task", *Qt*.formatDateTime(new *Date*(), "dd-MM-yyyy"),

*timeTask*.text, *dateTask*.text, *titleTask*.text, *descriptionTask*.text, *qsTr*("In process"))

*modelTask*.updateModel(*Qt*.formatDate(calendar.selectedDate, "yyyyMMdd"))

if(calendar.style === newCalendarStyle) calendar.style = calendarstyle

else calendar.style = newCalendarStyle

}

onReset: {

*defval*()

}

ColumnLayout {

id: *column*

width: *parent* ? *parent*.width : 100

TextField {

Layout.fillWidth: true

id: *titleTask*

property string placeholderText: *qsTr*("Brief description")

font.pixelSize: 12

Text {

x: 5

y : 5

text: *titleTask*.placeholderText

color: "#aaa"

visible: !*titleTask*.text

}

}

Label {

text: *qsTr*("Time and date of the deadline.")

Layout.columnSpan: 2

Layout.fillWidth: true

wrapMode: Text.WordWrap

}

RowLayout {

Layout.alignment: *Qt*.AlignHCenter

TextField{

id:*timeTask*

text : "00:00"

inputMask: "99:99"

inputMethodHints: *Qt*.ImhDigitsOnly

validator: RegExpValidator { regExp: /^([0-1]?[0-9]|2[0-3]):([0-5][0-9])$ / }

Layout.fillWidth: true

}

TextField{

id:*dateTask*

text: *Qt*.formatDate(calendar.selectedDate, "yyyyMMdd")

inputMask: "99-99-9999"

inputMethodHints: *Qt*.ImhDigitsOnly

validator: RegExpValidator { regExp: /^([0-2]?[0-9]|3[0-1])-(0?[0-9]|1[0-2])-([2-9][0-9][0-9][0-9])$ / }

Layout.fillWidth: true

}

}

TextArea {

id: *descriptionTask*

Layout.fillWidth: true

Layout.fillHeight: true

wrapMode: Text.WrapAnywhere

property string placeholderText: *qsTr*("Description")

font.pixelSize: 12

Text {

y: 10

x: 10

text: *descriptionTask*.placeholderText

color: "#aaa"

visible: !*descriptionTask*.text

}

}

}

}

# **ПРИЛОЖЕНИE 5**

Файл DialogTaskDelete.qml

import QtQuick 2.0

import QtQuick.Dialogs 1.2

import QtQuick.Controls 1.4

import QtQuick.Layouts 1.1

// Диалог подтверждения удаления строки задачи из базы данных

Dialog {

title: *qsTr*("Delete a line")

Label {

text: *qsTr*("Confirm <b>delete</ b> \"" + *modelTask*.get (*tableView*.currentRow) .tasktitle

+ "\" from tasks")

Layout.columnSpan: 2

Layout.fillWidth: true

wrapMode: Text.WordWrap

}

standardButtons: StandardButton.Ok | StandardButton.Cancel

// При положительном ответе ...

onAccepted: {

*database*.removeRecord("task", *modelTask*.get(*tableView*.currentRow).id) // Удаляем строку

*modelTask*.updateModel(*Qt*.formatDate(*calendar*.selectedDate, "yyyyMMdd")); // Обновляем модель данных

if(*calendar*.style === *newCalendarStyle*) *calendar*.style = *calendarstyle*

else *calendar*.style = *newCalendarStyle* //Обновляем календарь

}

}

# **ПРИЛОЖЕНИE 6**

Файл DialogScheduleView.qml

import QtQuick 2.0

import QtQuick.Dialogs 1.2

import QtQuick.Controls 1.4

import QtQuick.Layouts 1.1

// Диалог подробного описания строки рассписания

Dialog {

title: *modelSchedule*.get(*tableSchedule*.currentRow).schedultitle + ""

standardButtons: StandardButton.Ok

onAccepted: {

*database*.updateend(*modelSchedule*.get(*tableSchedule*.currentRow).id, *schedulend*.text)

*database*.updatestart(*modelSchedule*.get(*tableSchedule*.currentRow).id, *schedulstart*.text)

*modelSchedule*.updateModel(*Qt*.formatDate(*calendar*.selectedDate, "yyyyMMdd"))

if(*calendar*.style === *newCalendarStyle*) *calendar*.style = *calendarstyle*

else *calendar*.style = *newCalendarStyle*

}

ColumnLayout {

width: *parent* ? *parent*.width : 100

RowLayout {

Layout.alignment: *Qt*.AlignHCenter

Label {

text: *qsTr*("Start: ")

Layout.fillWidth: true

wrapMode: Text.WordWrap

}

TextField {

id: *schedulstart*

text: "<b>"+*modelSchedule*.get(*tableSchedule*.currentRow).schedulstart +"</b> "

inputMask: "99:99"

inputMethodHints: *Qt*.ImhDigitsOnly

validator: RegExpValidator { regExp: /^([0-1]?[0-9]|2[0-3]):([0-5][0-9])$ / }

Layout.fillWidth: true

}

}

RowLayout {

Layout.alignment: *Qt*.AlignHCenter

Label {

text: *qsTr*("End: ")

Layout.fillWidth: true

wrapMode: Text.WordWrap

}

TextField{

id:*schedulend*

text: ""+*modelSchedule*.get(*tableSchedule*.currentRow).schedulend +""

inputMask: "99:99"

inputMethodHints: *Qt*.ImhDigitsOnly

validator: RegExpValidator { regExp: /^([0-1]?[0-9]|2[0-3]):([0-5][0-9])$ / }

Layout.fillWidth: true

}

}

Label {

text: *modelSchedule*.get(*tableSchedule*.currentRow).scheduldesc + ""

Layout.fillWidth: true

Layout.fillHeight: true

y: 100

wrapMode: Text.WordWrap

}

}

}

# **ПРИЛОЖЕНИE 7**

Файл DialogScheduleAdd.qml

import QtQuick 2.0

import QtQuick.Dialogs 1.2

import QtQuick.Controls 1.4

import QtQuick.Layouts 1.1

// Диалог добавления рассписания

Dialog {

height: 300

title: *qsTr*("Add to schedule")

standardButtons: StandardButton.Apply | StandardButton.Cancel

function *defval*() {

//

//dateShcedule.text = Qt.formatDate(calendar.selectedDate, "dd-MM-yyyy")

*titleShcedule*.text = ""

*descriptionShcedule*.text = ""

*startShcedule*.text = "00:00"

*endShcedule*.text = "00:00"

}

onApply: {

if(*startShcedule*.text < *endShcedule*.text) {

*database*.inserIntoTable("shcedule", *startShcedule*.text, *endShcedule*.text,*Qt*.formatDate(*calendar*.selectedDate, "yyyyMMdd"),

*titleShcedule*.text, *descriptionShcedule*.text, "")

//modelSchedule.updateModel(calendar.selectedDate.getDay())

*modelSchedule*.updateModel(*Qt*.formatDate(*calendar*.selectedDate, "yyyyMMdd"))

*dialogAddShcedule*.close()

} else {

*endShcedule*.textColor = 'red'

}

}

ColumnLayout {

width: *parent* ? *parent*.width : 100

TextField {

Layout.fillWidth: true

id: *titleShcedule*

property string placeholderText: *qsTr*("Name of the subject")

font.pixelSize: 12

Text {

x: 5

y : 5

text: *titleShcedule*.placeholderText

color: "#aaa"

visible: !*titleShcedule*.text

}

}

RowLayout {

Layout.alignment: *Qt*.AlignHCenter

Label {

text: *qsTr*("Start:")

Layout.fillWidth: true

}

TextField{

id: *startShcedule*

text : "00:00"

inputMask: "99:99"

inputMethodHints: *Qt*.ImhDigitsOnly

validator: RegExpValidator { regExp: /^([0-1]?[0-9]|2[0-3]):([0-5][0-9])$ / }

Layout.fillWidth: true

}

}

RowLayout {

Layout.alignment: *Qt*.AlignHCenter

Label {

text: *qsTr*("End: ")

Layout.fillWidth: true

}

TextField{

id: *endShcedule*

text : "00:00"

inputMask: "99:99"

inputMethodHints: *Qt*.ImhDigitsOnly

validator: RegExpValidator { regExp: /^([0-1]?[0-9]|2[0-3]):([0-5][0-9])$ / }

Layout.fillWidth: true

}

}

/\* RowLayout {

Layout.alignment: Qt.AlignHCenter

Label {

text: qsTr("Date : ")

Layout.fillWidth: true

}

TextField{

id: dateShcedule

text: Qt.formatDate(calendar.selectedDate, "ddMMyyyy")

inputMask: "99-99-9999"

inputMethodHints: Qt.ImhDigitsOnly

validator: RegExpValidator { regExp: /^([0-2]?[0-9]|3[0-1])-(0?[0-9]|1[0-2])-([2-9][0-9][0-9][0-9])$ / }

Layout.fillWidth: true

}

}\*/

TextArea {

id: *descriptionShcedule*

Layout.fillWidth: true

Layout.fillHeight: true

wrapMode: Text.WrapAnywhere

property string placeholderText: *qsTr*("Additional field")

font.pixelSize: 12

Text {

y: 10

x: 10

text: *descriptionShcedule*.placeholderText

color: "#aaa"

visible: !*descriptionShcedule*.text

}

}

}

}

# **ПРИЛОЖЕНИE 8**

Файл DialogScheduleDelete.qml

import QtQuick 2.0

import QtQuick.Dialogs 1.2

import QtQuick.Controls 1.4

import QtQuick.Layouts 1.1

// Диалог подтверждения удаления строки расписания из базы данных

Dialog {

//color: "peachpuff"

title: *qsTr*("Delete a line")

Label {

text: *qsTr*("Confirm <b>deletion</ b> \"" + *modelSchedule*.get(*tableSchedule*.currentRow).schedultitle

+ "\" from the schedule")

Layout.columnSpan: 2

Layout.fillWidth: true

wrapMode: Text.WordWrap

}

standardButtons: StandardButton.Ok | StandardButton.Cancel

onAccepted: {

*database*.removeRecord("schedule", *modelSchedule*.get(*tableSchedule*.currentRow).id)

// modelSchedule.updateModel(calendar.selectedDate.getDay());

*modelSchedule*.updateModel(*Qt*.formatDate(*calendar*.selectedDate, "yyyyMMdd"))

}

}

# **ПРИЛОЖЕНИE 9**

Файл sqldb.cpp

#include "sqldb.h"

#include "QCoreApplication"

SqlDB::SqlDB(QObject \*parent) : QObject(parent){ }

SqlDB::~*SqlDB*(){ }

/\* Методы для подключения к базе данных

\*/

void SqlDB::connectToDataBase() {

QString path = qApp->applicationDirPath() + "/" + DATABASE\_NAME;

/\* Перед подключением к базе данных производим проверку на её существование.

В зависимости от результата производим открытие базы данных или её создание

\*/

if(!QFile(path).exists()){

this->restoreDataBase();

} else {

this->openDataBase();

}

}

/\* Методы восстановления базы данных \*/

bool SqlDB::restoreDataBase() {

// Если база данных открылась ...

if(this->openDataBase()){

// Производим восстановление базы данных

return (this->createTable()) ? true : false;

} else {

qDebug() << "Не удалось восстановить базу данных";

return false;

}

return false;

}

/\* Метод для открытия базы данных \*/

bool SqlDB::openDataBase() {

/\* База данных открывается по заданному пути

и имени базы данных, если она существует \*/

QString path = qApp->applicationDirPath() + "/" + DATABASE\_NAME;

db = QSqlDatabase::addDatabase("QSQLITE");

db.setHostName(DATABASE\_HOSTNAME);

db.setDatabaseName(path);

if(db.open()){

return true;

} else {

return false;

}

}

// Методы закрытия базы данных

void SqlDB::closeDataBase() {

db.close();

}

// Метод для создания таблицы в базе данных

bool SqlDB::createTable() {

/\* В данном случае используется формирование сырого SQL-запроса

с последующим его выполнением.

\*/

QSqlQuery query;

if(!query.exec( "CREATE TABLE " TABLE\_TASK " ("

"id INTEGER PRIMARY KEY AUTOINCREMENT, "

TABLE\_TASK\_START\_DATE " VARCHAR(255) NOT NULL,"

TABLE\_TASK\_DEADLINE\_TIME " VARCHAR(255) NOT NULL,"

TABLE\_TASK\_DEADLINE\_DATE " VARCHAR(255) NOT NULL,"

TABLE\_TASK\_TITLE " VARCHAR(255) NOT NULL,"

TABLE\_TASK\_DESC " VARCHAR(255) NOT NULL,"

TABLE\_TASK\_STATUS " VARCHAR(255) NOT NULL)") ||

!query.exec( "CREATE TABLE " TABLE\_SCHEDULE " ("

"id INTEGER PRIMARY KEY AUTOINCREMENT, "

TABLE\_SCHEDULE\_START\_TIME " VARCHAR(255) NOT NULL,"

TABLE\_SCHEDULE\_END\_TIME " VARCHAR(255) NOT NULL,"

TABLE\_SCHEDULE\_WEEK " VARCHAR(255) NOT NULL,"

TABLE\_SCHEDULE\_TITLE " VARCHAR(255) NOT NULL,"

// TABLE\_SCHEDULE\_DATE " VARCHAR(255) NOT NULL,"

TABLE\_SCHEDULE\_DESC " VARCHAR(255) NOT NULL)")){

qDebug() << "DataBase: error of create " << TABLE\_TASK << " & " << TABLE\_SCHEDULE;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

// Метод для вставки записи в базу данных для заданий по готовому List

bool SqlDB::inserIntoTableTask(const QVariantList &data) {

/\* Запрос SQL формируется из QVariantList,

в который передаются данные для вставки в таблицу.

\*/

QSqlQuery query;

/\* В начале SQL запрос формируется с ключами,

которые потом связываются методом bindValue

для подстановки данных из QVariantList

\*/

query.prepare("INSERT INTO " TABLE\_TASK " ( "

TABLE\_TASK\_START\_DATE " ,"

TABLE\_TASK\_DEADLINE\_TIME ", "

TABLE\_TASK\_DEADLINE\_DATE ", "

TABLE\_TASK\_TITLE ", "

TABLE\_TASK\_DESC ", "

TABLE\_TASK\_STATUS " ) "

"VALUES (:StartDate, :DeadlineTime, :DeadlineDate, :TaskTitle, :TaskDescription, :TaskStatus)");

query.bindValue(":StartDate", data[0].toString());

query.bindValue(":DeadlineTime", data[1].toString());

query.bindValue(":DeadlineDate", data[2].toString());

query.bindValue(":TaskTitle", data[3].toString());

query.bindValue(":TaskDescription", data[4].toString());

query.bindValue(":TaskStatus", data[5].toString());

// После чего выполняется запросом методом exec()

if(!query.exec()){

qDebug() << "error insert into " << TABLE\_TASK;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

// Аналогично для рассписания

bool SqlDB::inserIntoTableSchedule(const QVariantList &data) {

QSqlQuery query;

query.prepare("INSERT INTO " TABLE\_SCHEDULE " ( "

TABLE\_SCHEDULE\_START\_TIME ", "

TABLE\_SCHEDULE\_END\_TIME ", "

TABLE\_SCHEDULE\_WEEK ", "

TABLE\_SCHEDULE\_TITLE ", "

// TABLE\_SCHEDULE\_DATE ", "

TABLE\_SCHEDULE\_DESC " ) "

"VALUES (:ScheduleStart, :ScheduleEnd, :ScheduleWeek, :ScheduleTitle, :ScheduleDescription)");

query.bindValue(":ScheduleStart", data[0].toString());

query.bindValue(":ScheduleEnd", data[1].toString());

query.bindValue(":ScheduleWeek", data[2].toString());

query.bindValue(":ScheduleTitle", data[3].toString());

query.bindValue(":ScheduleDescription", data[4].toString());

if(!query.exec()){

qDebug() << "error insert into " << TABLE\_SCHEDULE;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

// Метод формирования List для дальнейшей вставки в таблицу

bool SqlDB::inserIntoTable(const QString &db, const QString &start, const QString &end,

const QString &date, const QString &title, const QString &desc, const QString &status) {

QVariantList data;

data.append(db == "task" ? encodeDate(start) : start);

data.append(end);

data.append(db == "task" ? encodeDate(date) : date);

data.append(title);

data.append(desc);

if(db == "task")

data.append(status);

if(db == "task" ? inserIntoTableTask(data) : inserIntoTableSchedule(data))

return true;

else

return false;

}

// Метод для удаления записи из таблицы

bool SqlDB::removeRecord(const QString &db, const QString &id) {

// Удаление строки из базы данных будет производитсья с помощью SQL-запроса

QSqlQuery query;

QString b = db == "task" ? TABLE\_TASK : TABLE\_SCHEDULE;

// Удаление производим по id записи, который передается в качестве аргумента функции

query.prepare("DELETE FROM " + b + " WHERE id= :ID ;");

query.bindValue(":ID", id);

// Выполняем

if(!query.exec()){

qDebug() << "error delete row " << b;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

// Метод проверки наличия не выполненных задач в дате

bool SqlDB::searchTask(const QString &date) {

QSqlQuery query;

query.prepare("SELECT COUNT(\*) FROM " TABLE\_TASK " WHERE " TABLE\_TASK\_DEADLINE\_DATE " = '"+date+"' and "

TABLE\_TASK\_STATUS " = 'In process'");

if(query.exec())

{

while (query.next())

{

if(query.value(0).toInt() > 0)

return true;

else return false;

}

} else return false;

return false;

}

// Метод обновления статуса задачи по ее id

void SqlDB::updatestatus(const QString &id, const QString &status) {

QSqlQuery query;

query.prepare("UPDATE " TABLE\_TASK " SET "

TABLE\_TASK\_STATUS " = :STATUS WHERE ID = :ID;");

query.bindValue(":ID", id);

query.bindValue(":STATUS", status);

query.exec();

}

void SqlDB::updatedate(const QString &id, const QString &deadlineday) {

QSqlQuery query;

query.prepare("UPDATE " TABLE\_TASK " SET "

TABLE\_TASK\_DEADLINE\_DATE " = :DATE WHERE ID = :ID;");

query.bindValue(":ID", id);

query.bindValue(":DATE", encodeDate(deadlineday));

query.exec();

}

void SqlDB::updatetime(const QString &id, const QString &deadlinetime) {

QSqlQuery query;

query.prepare("UPDATE " TABLE\_TASK " SET "

TABLE\_TASK\_DEADLINE\_TIME " = :TIME WHERE ID = :ID;");

query.bindValue(":ID", id);

query.bindValue(":TIME",deadlinetime);

query.exec();

}

void SqlDB::updatestart(const QString &id, const QString &start) {

QSqlQuery query;

query.prepare("UPDATE " TABLE\_SCHEDULE " SET "

TABLE\_SCHEDULE\_START\_TIME " = :DATE WHERE ID = :ID;");

query.bindValue(":ID", id);

query.bindValue(":DATE", start);

query.exec();

}

void SqlDB::updateend(const QString &id, const QString &end) {

QSqlQuery query;

query.prepare("UPDATE " TABLE\_SCHEDULE " SET "

TABLE\_SCHEDULE\_END\_TIME " = :TIME WHERE ID = :ID;");

query.bindValue(":ID", id);

query.bindValue(":TIME",end);

query.exec();

}

QString SqlDB::encodeDate(const QString &s) {

QDateTime dt = QDateTime::fromString(s, "dd-MM-yyyy");

QString timeStr = dt.toString("yyyyMMdd");

return timeStr;

}

QString SqlDB::decodeDate(const QString &s) {

QDateTime dt = QDateTime::fromString(s, "yyyyMMdd");

QString timeStr = dt.toString("dd-MM-yyyy");

return timeStr;

}

# **ПРИЛОЖЕНИE 10**

Файл main.cpp

#include <QGuiApplication>

#include <QQmlApplicationEngine>

#include <QQmlContext>

#include <QIcon>

#include "sqldb.h"

#include "tasklistmodel.h"

#include "schedulelistmodel.h"

int main(int argc, char \*argv[])

{

QCoreApplication::setAttribute(Qt::AA\_EnableHighDpiScaling);

QGuiApplication app(argc, argv);

QQmlApplicationEngine engine;

// Добавляем иконку

app.setWindowIcon(QIcon("qrc:/images/checklis.ico"));

// Подключаемся к базе данных

SqlDB database;

database.connectToDataBase();

// Объявляем и инициализируем модели данных

TaskListModel \*modelTask = new TaskListModel();

ScheduleListModel \*modelSchedule = new ScheduleListModel();

// Обеспечиваем доступ к моделям и классам для работы с базой данных из QML

engine.rootContext()->setContextProperty("modelTask", modelTask);

engine.rootContext()->setContextProperty("modelSchedule", modelSchedule);

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

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

return app.exec();

}

# **ПРИЛОЖЕНИE 10**

Файл tasklistmodel.cpp

#include "tasklistmodel.h"

#include "sqldb.h"

TaskListModel::TaskListModel(QObject \*parent) :

QSqlQueryModel(parent) {

this->updateModel(QDate::currentDate().toString("yyyyMMdd"));

}

// Метод для получения данных из модели

QVariant TaskListModel::*data*(const QModelIndex & index, int role) const {

// Определяем номер колонки, адрес , по номеру роли

int columnId = role - Qt::UserRole - 1;

// Создаём индекс с помощью новоиспечённого ID колонки

QModelIndex modelIndex = this->*index*(index.row(), columnId);

/\* И с помощью уже метода data() базового класса

вытаскиваем данные для таблицы из модели \*/

return QSqlQueryModel::*data*(modelIndex, Qt::DisplayRole);

}

// Метод для получения имен ролей через хешированную таблицу.

QHash<int, QByteArray> TaskListModel::*roleNames*() const {

/\* То есть сохраняем в хеш-таблицу названия ролей

по их номеру

\*/

QHash<int, QByteArray> roles;

roles[IdRole] = "id";

roles[StartDate] = "startdate";

roles[DeadlineTime] = "endtime";

roles[DeadlineDate] = "enddate";

roles[TaskTitle] = "tasktitle";

roles[TaskDescription] = "taskdesc";

roles[TaskStatus] = "taskstatus";

return roles;

}

// Метод обновления таблицы в модели представления данных

void TaskListModel::updateModel(const QString &date) {

// Обновление производится SQL-запросом к базе данных

this->setQuery("SELECT id, "

TABLE\_TASK\_START\_DATE ", "

TABLE\_TASK\_DEADLINE\_TIME ", "

TABLE\_TASK\_DEADLINE\_DATE ", "

TABLE\_TASK\_TITLE ", "

TABLE\_TASK\_DESC ", "

TABLE\_TASK\_STATUS " FROM " TABLE\_TASK " WHERE " TABLE\_TASK\_DEADLINE\_DATE " >= '"+date+"' and "

TABLE\_TASK\_START\_DATE" <= '"+date+"' ORDER BY " TABLE\_TASK\_STATUS "," TABLE\_TASK\_DEADLINE\_DATE"," TABLE\_TASK\_DEADLINE\_TIME);

}

// Получение значений по строке из модели представления данных

QVariantMap TaskListModel::get(int idx) {

QVariantMap map;

foreach(int k, *roleNames*().keys()) {

map[*roleNames*().value(k)] = *data*(*index*(idx, 0), k);

}

return map;

}

# **ПРИЛОЖЕНИE 11**

Файл schedulelistmodel.cpp

#include "schedulelistmodel.h"

#include "sqldb.h"

ScheduleListModel::ScheduleListModel(QObject \*parent) :

QSqlQueryModel(parent){

this->updateModel(QDate::currentDate().toString("yyyyMMdd"));

}

QVariant ScheduleListModel::*data*(const QModelIndex & index, int role) const {

int columnId = role - Qt::UserRole - 1;

QModelIndex modelIndex = this->*index*(index.row(), columnId);

return QSqlQueryModel::*data*(modelIndex, Qt::DisplayRole);

}

QHash<int, QByteArray> ScheduleListModel::*roleNames*() const {

QHash<int, QByteArray> roles;

roles[IdRole] = "id";

roles[ScheduleStart] = "schedulstart";

roles[ScheduleEnd] = "schedulend";

roles[ScheduleWeek] = "schedulweek";

roles[ScheduleTitle] = "schedultitle";

roles[ScheduleDescription] = "scheduldesc";

return roles;

}

void ScheduleListModel::updateModel(const QString &week){

this->setQuery("SELECT id, "

TABLE\_SCHEDULE\_START\_TIME ", "

TABLE\_SCHEDULE\_END\_TIME ", "

TABLE\_SCHEDULE\_WEEK ", "

TABLE\_SCHEDULE\_TITLE ", "

// TABLE\_SCHEDULE\_DATE ", "

TABLE\_SCHEDULE\_DESC

" FROM " TABLE\_SCHEDULE " WHERE " TABLE\_SCHEDULE\_WEEK " = '"+week+"'" " ORDER BY " TABLE\_SCHEDULE\_START\_TIME

);

}

QVariantMap ScheduleListModel::get(int idx){

QVariantMap map;

foreach(int k, *roleNames*().keys()) {

map[*roleNames*().value(k)] = *data*(*index*(idx, 0), k);

}

return map;

}

ВЕДОМОСТЬ

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КР 1–40 01 01 215 ПЗ | | | | Пояснительная записка | | | | 65 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 851002 215 ПД | | | | "Менеджер задач"  А1, схема программы, чертеж | | | | **Формат А1** | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | **БГУИР КР 1-40 01 01 215 Д1** | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | "Менеджер задач"  Ведомость курсовой  работы |  | | | | Лист | Листов |
| Разраб. | | Цыбулько К.Д. |  | 16.12.19 | Т |  | |  | 65 | 65 |
| Пров. | | Шостк Е.В. |  | 16.12.19 | Кафедра ПОИТ  гр. 851002 | | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |