Содержание

[Введение 5](#_Toc117065290)

[1 Сбор, анализ и формирование требований к программному продукту 6](#_Toc117065291)

[1.1 Сбор требований 6](#_Toc117065292)

[1.2 Анализ и формирование требований 7](#_Toc117065293)

[2 Проектирование и разработка архитектуры программного продукта 9](#_Toc117065294)

[2.1 Построение диаграммы связей 9](#_Toc117065295)

[2.2 Разработка сценария использования 10](#_Toc117065296)

[2.3 Архитектура программного продукта 14](#_Toc117065297)

[2.4 Прототипирование и дизайн программного продукта 15](#_Toc117065298)

[3 Разработка программного продукта 21](#_Toc117065299)

[3.1 Инструментальные и программные средства разработки 21](#_Toc117065300)

[3.2 Календарный план разработки 22](#_Toc117065301)

[4 Тестирование программного продукта 24](#_Toc117065302)

[4.1 Выбор метода обеспечения качества 24](#_Toc117065303)

[4.2 Тестирование программного продукта 24](#_Toc117065304)

[5 Разработка документации на программный продукт 28](#_Toc117065305)

[Заключение 29](#_Toc117065306)

[Список литературы 30](#_Toc117065307)

Приложение А [30](#_Toc117065307)

Компонент [db\_view 31](#_Toc117065308)

Компонент [editor 36](#_Toc117065309)

Компонент [globals 40](#_Toc117065326)

Компонент [main 40](#_Toc117065326)

Компонент [palettes 40](#_Toc117065326)

Компонент [tabmanager 40](#_Toc117065326)

Компонент [toolbar 40](#_Toc117065326)

Компонент [tree 40](#_Toc117065326)

Компонент [utils 40](#_Toc117065326)

Компонент [window 40](#_Toc117065326)

Компонент [window 40](#_Toc117065326)

Компонент [window 40](#_Toc117065326)

# Введение

Целью курсового проекта является разработка редактора кода “Light”. Программу предполагается использовать для удобного редактирование кода, а так же базы данных SQLite. Приложение имеет графический интерфейс, написано на Python, и поддерживает подсветку синтаксиса для языков программирования.

Главной задачей была реализация стандартных функций редактора кода. Создание текстового файла, открытие, копирование, вставка, подсветка синтаксиса, смены темы оформления.

Моя роль в проекте – разработчик, задача которого заключалась в создании основного функционала приложения, дизайна окна, и функций backend’а.

# 1 Сбор, анализ и формирование требований к программному продукту

## 1.1 Сбор требований

На этапе сбора требований необходимо составить вопросы для заказчика и сформировать их в таблицу-опросник.

|  |  |
| --- | --- |
| Вопрос | Ответ |
| 1. Название компании | ИП Шатохин |
| 2. Род деятельности | Создание программных продуктов. |
| 3. С какой целью планируется использовать продукт? | С целью разработки программных решений, а так же редактирования текста. |
| 4. Что необходимо в плане функционала? | Наличие подсветки синтаксиса, сохранение файла, создание, возможность открыть несколько файлов срзу при помощи вкладок. |
| 5. Как вы представляете идеальный интерфейс? | Целостный, интуитивно понятный. |
| 6. Каким редактором кода вы пользуйтесь сечас? | Visual Studio Code. |
| 7. Важная ли поддержка редактирования базы данных? | Да. |
| 8. Есть ли системные требования к приложению? | Практически любой X86-совместимый процессор |
| 9. Важно ли улучшение функционала? | Да, расширяемость продукта - признак хорошего программиста. |
| 10. Какие качества продукты заставили бы вас перейти на новый? | Открытость, функциональность, простота. |
| 11. каковы сроки проекта? | 5 ноября 2022года. |
| 12. Что для вас важнее: количество фонкционала или качество? | Качество. Избыток бесполезных функций никому не нужен. |
| 13. Знаете ли вы своих конкурентов? | Да, это Vsual Studio Code, Atom, Sublime Text |
| 14. Как вы хотите продвигать продукт? | Планируется продвигать продукт по лицензии GPL |

## 1.2 Анализ и формирование требований

Целевой аудиторией продукта являются программисты, из чего будут составлены такие требования, которые смогут обеспечить удобство работы конкретно этой группы людей.

Целями создания проекта является:

* Подсветка синтаксиса кода
* Вкладки с именем и расширением для каждого открытого файла, а так же возможность переключения между ними.
* Смена цветовой схемы со светлой на темную, и наоборот.
* Возможность редактирования базы данных, с отображением столбцов и строк.
* Возможность открыть последние открытые пользователем файлы через отдельную кнопку в меню «Файл».

# 2 Проектирование и разработка архитектуры программного продукта

## 2.1 Построение диаграммы связей

Диаграмма связей программного продукта представляет собой графическую схему взаимодействия объектов проектируемого ПП. Схема взаимодействия программных модулей показывает структуру программы, её составные части (модули), отражает связь между ними. Схема отражает не только подчиненность модулей, но и порядок их вызова или функционирования программы. Схема дополняется расшифровкой функций, выполняемых модулями.

Графическое изображение схемы взаимодействия модулей приложения «Light» представлено на рисунке 1.

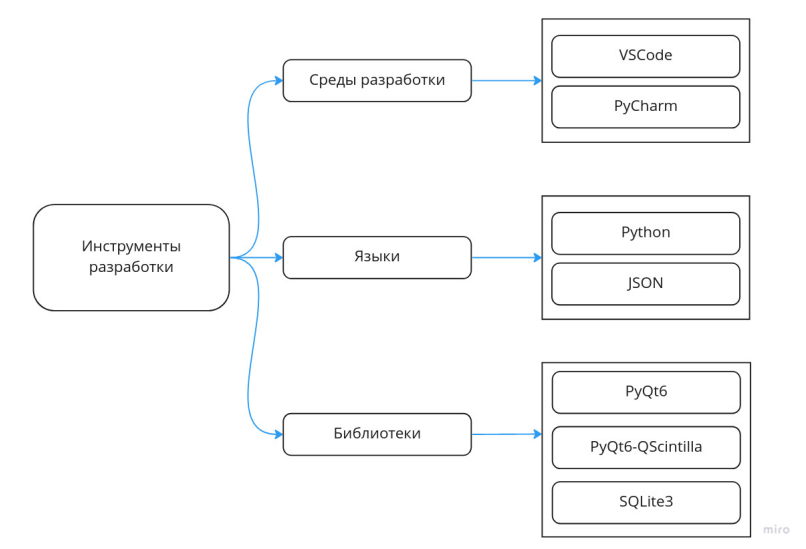


Рисунок 1 – Схема Mind Map

## 2.2 Разработка сценария использования

Сценарий использования, проектируемого ПП можно разработать с помощью диаграмм вариантов использования (Use-case) и деятельности UML. Диаграмма Use-case разработана с целью определения функционала приложения «Grafics Boom» для его пользователей, и представлена на рисунке 2.

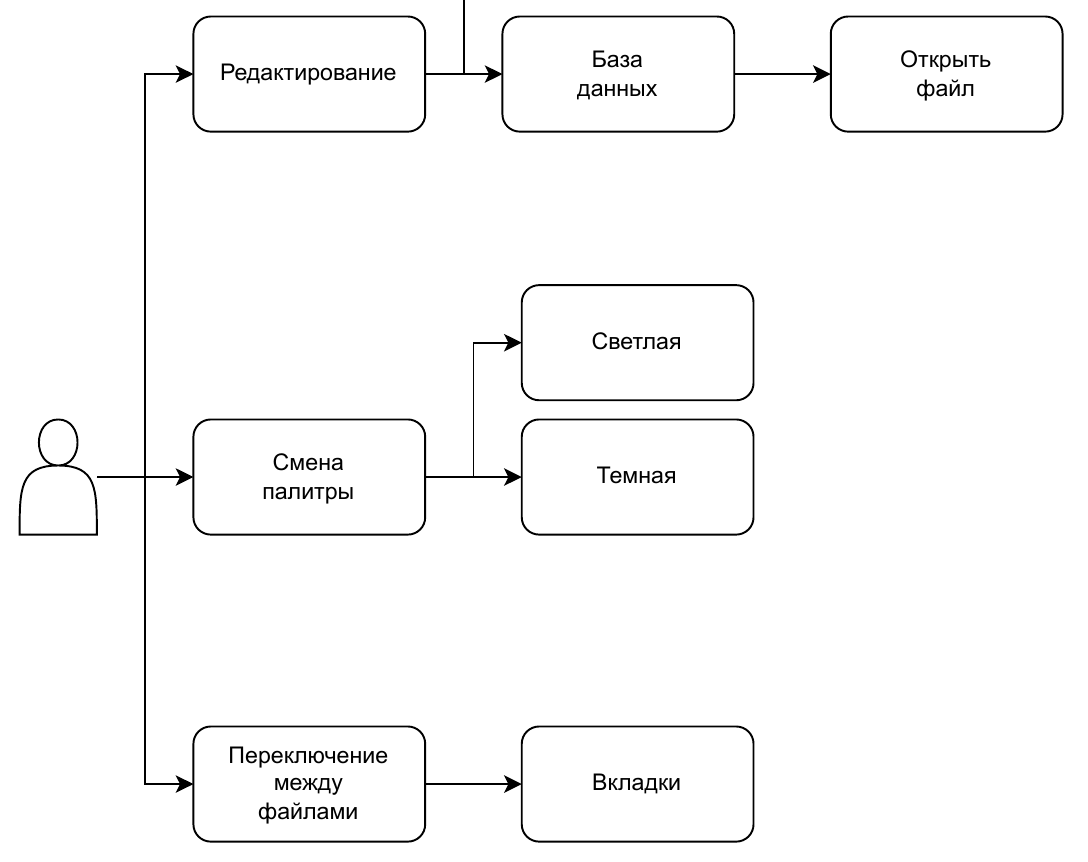


Рисунок 2 - Диаграмма использования со стороны пользователя

Диаграмма деятельности (активности) UML позволяет более детально визуализировать конкретный вариант использования. Эта диаграмма представляет собой блок-схему, которая наглядно показывает, как поток управления переходит от одной деятельности к другой. Диаграмма деятельности для вариантов использования всех функций представлена на рисунке 3.

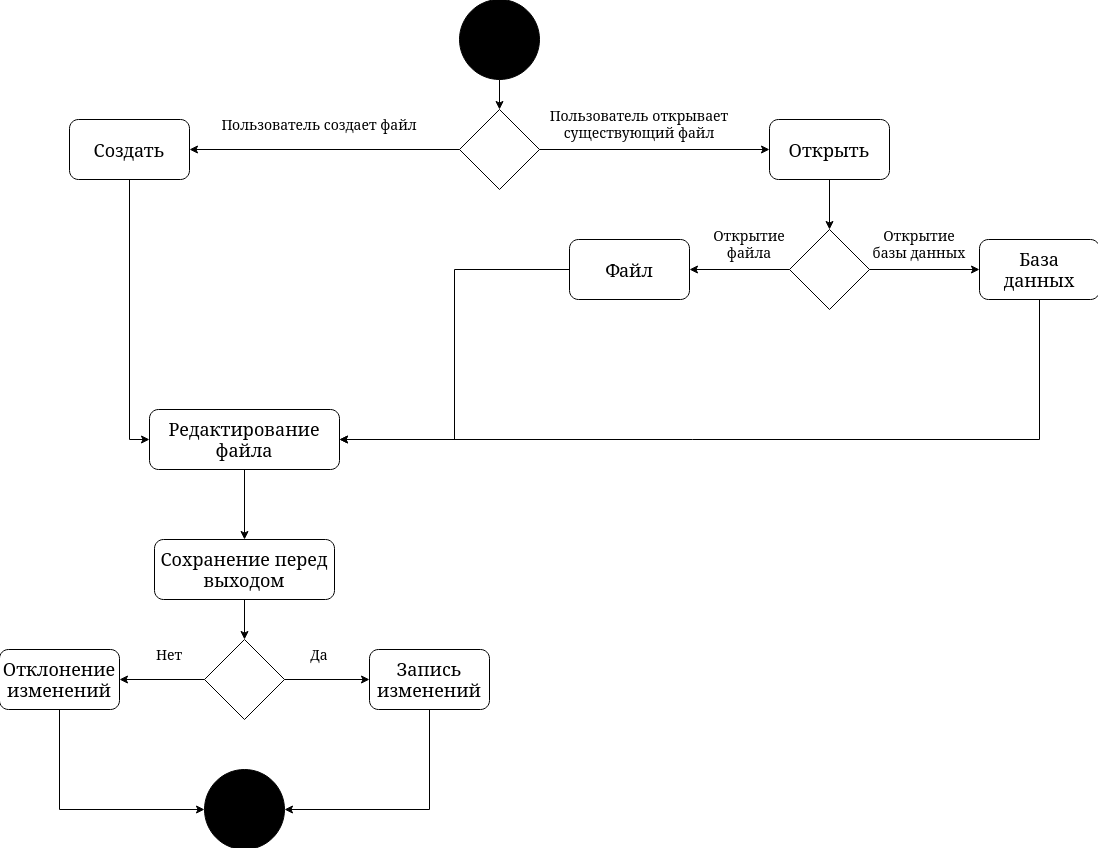


Рисунок 3 – Диаграмма деятельности для всех функций редактора

## 2.3 Архитектура программного продукта

При проектировании редактора кода «Light» была использована программа Visual Studio Code. Сама программа написана на языке программирования Python, приложение будет устанавливаться на твердотельный накопитель в компьютере пользователя.

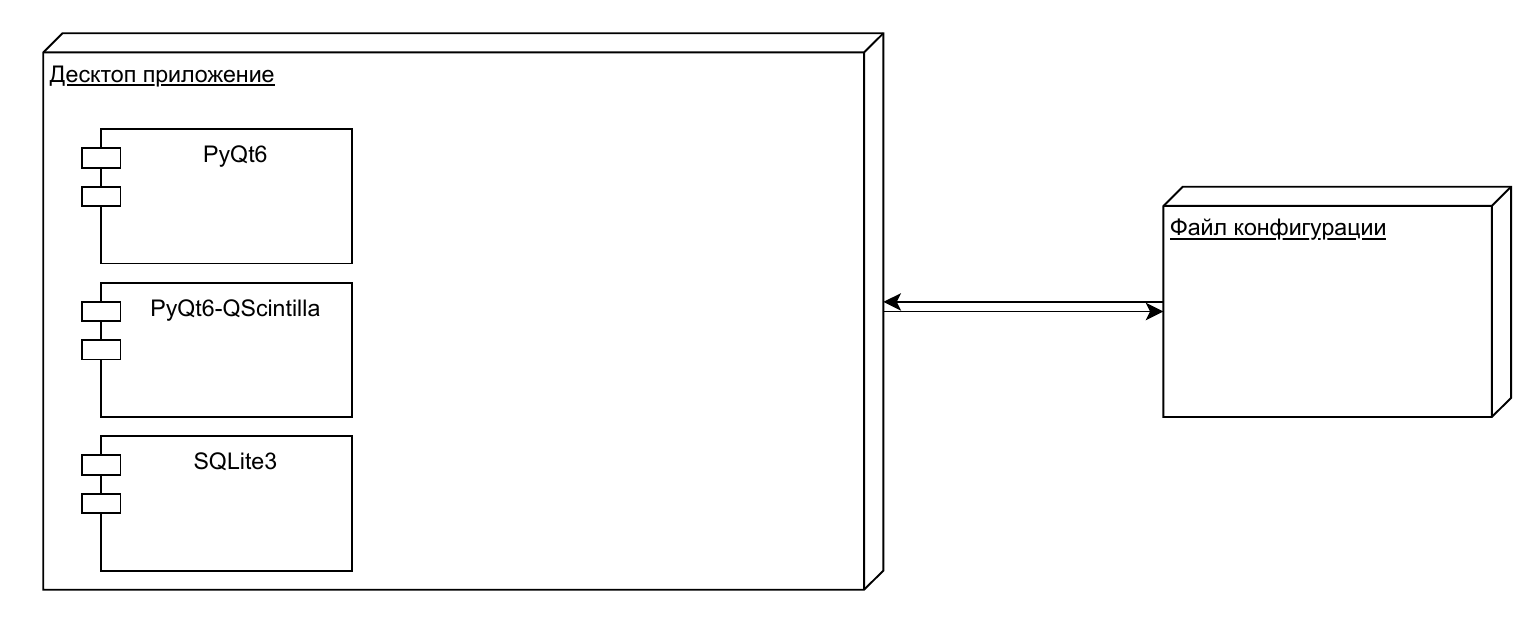


Рисунок 4 – Архитектура ПП

Почти всё современное программирование построено на принципах ООП и для проектирования внутренней структуры объектно-ориентированного приложения используют диаграмму классов (class diagram), предназначенную для представления программы в виде классов и связей между ними. Диаграмма классов приложения «Light» представлена на рисунке 5.

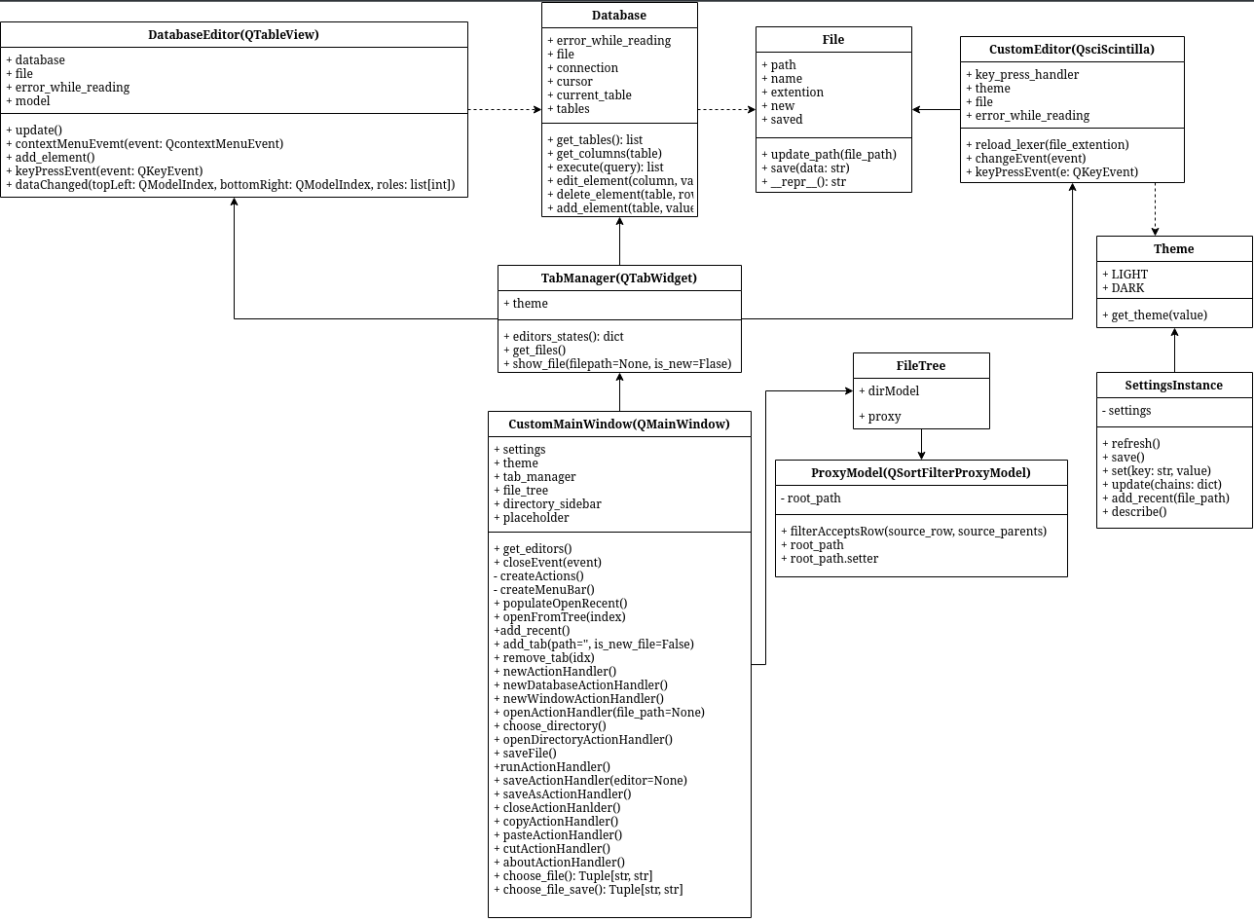


Рисунок 5 – Диаграмма классов проектируемого ПП

## 2.4 Прототипирование и дизайн программного продукта

Прототип — это набросок продукта, в котором заключены его внешний

вид, логика работы и основная функциональность. Для презентации прототипа показывают, к чему и куда приводит взаимодействие будущего пользователя с элементами интерфейса.

Прототипы главного окна и внутренних фукнций приложения «Light», разработанные с помощью сервиса Figma, представлены ниже на рисунках 6-8.

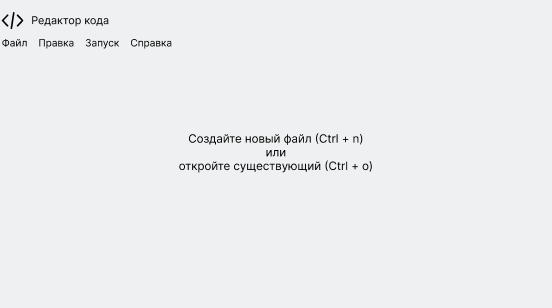


Рисунок 6 – Главное окно программы

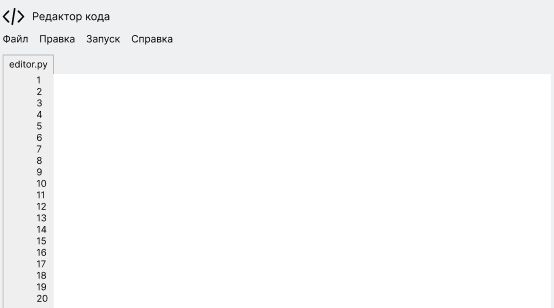


Рисунок 7 – Открытие файла

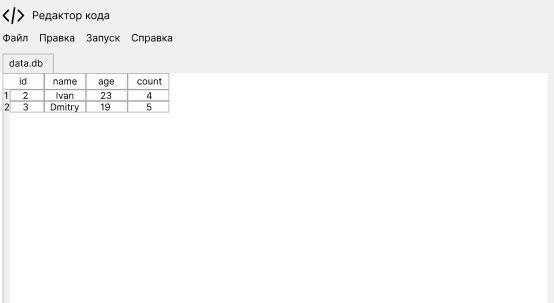


Рисунок 8 – Открытие базы данных SQLite

# 3 Разработка программного продукта

## 3.1 Инструментальные и программные средства разработки

Для разработки программного продукта были использованы следующие инструментальные и программные средства:

Программно-инструментальные средства — это программные продукты, предназначенные для разработки программного обеспечения. К ним относят системы программирования, которые включают систему команд процессора и периферийных устройств, трансляторы с различных языков программирования.

Для разработки проекта были использованы следующие программные средства разработки:

Python — высокоуровневый язык программирования общего назначения с динамической типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ.

GitHub - сервис онлайн-хостинга репозиториев, обладающий всеми функциями распределённого контроля версий и функциональностью управления исходным кодом — всё, что поддерживает Git и даже больше. Также GitHub может похвастаться контролем доступа, багтрекингом, управлением задачами и вики для каждого проекта.

Visual Studio Code — редактор кода, который поддерживает большое количество дополнений. Расширение «Python» предоставляет средства для анализа кода, инструмент для запуска юнит-тестов и поддерживает веб-разработку на Django. PyCharm разработана компанией Microsoft на фреймворке Electron.

Trello - платформа для совместной работы распределенных команд. Проще говоря, является канбан-доской.

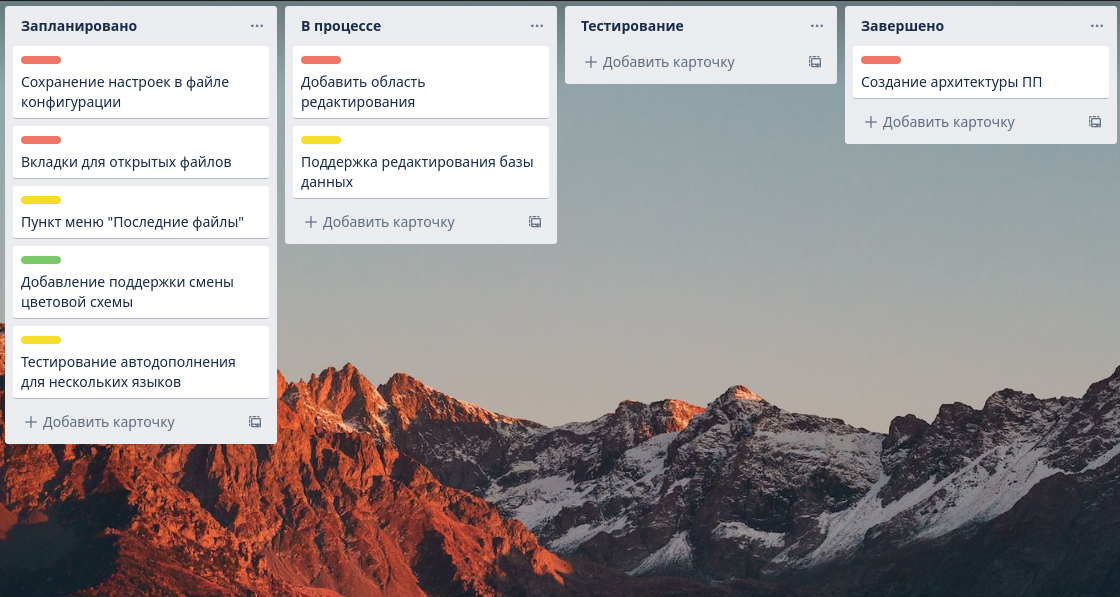
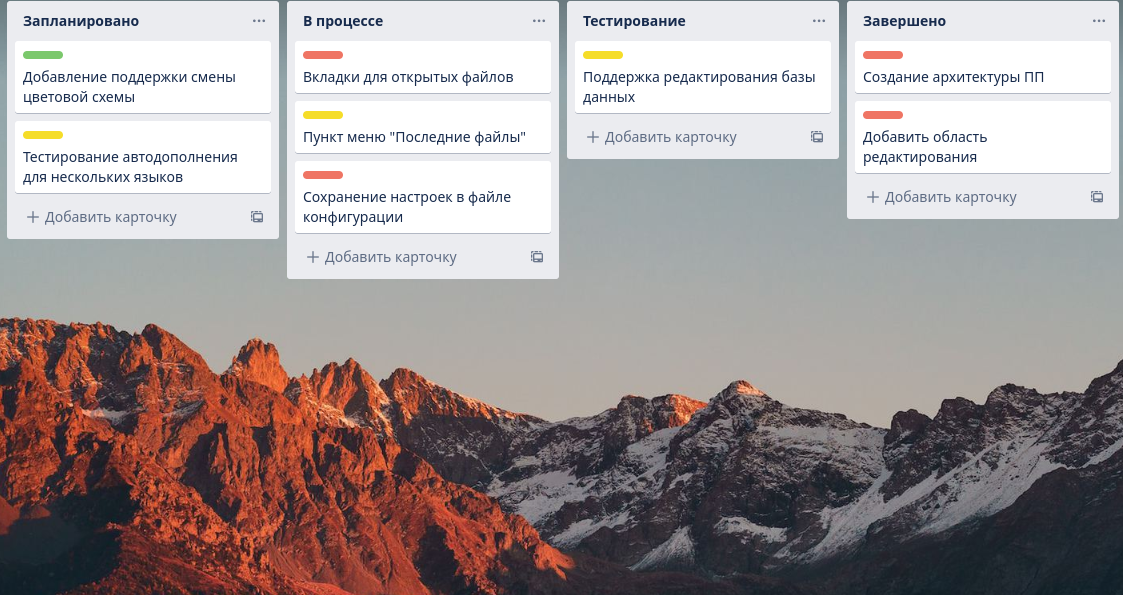
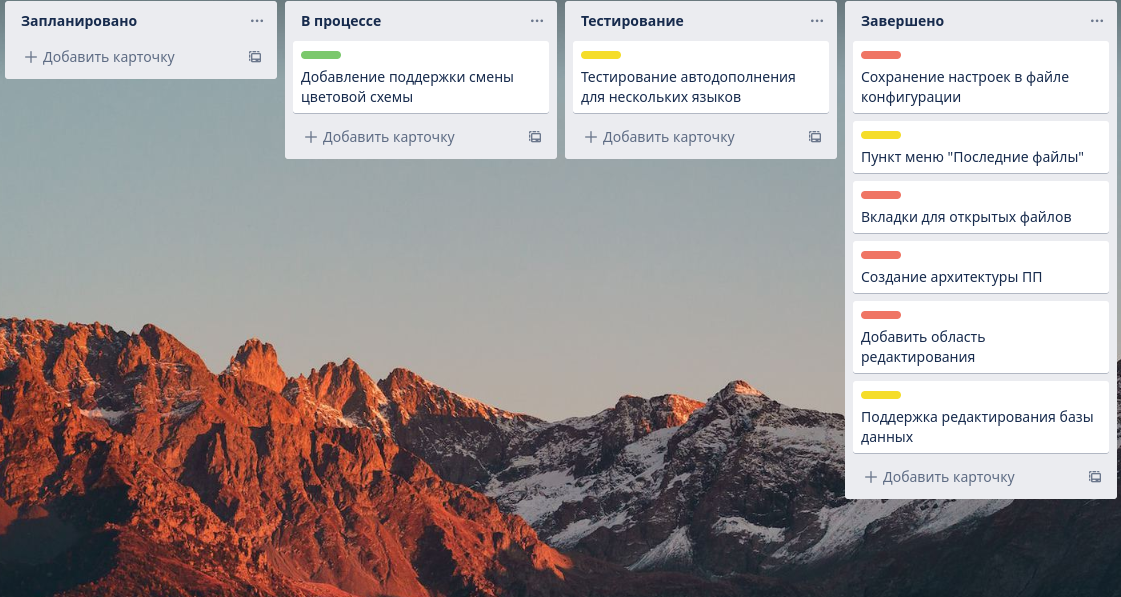


Рисунок 9 — доска Miro с начальными задачами

Рисунок 10 — доска Miro при выполнении задач

Рисунок 11 — доска Miro после выполнения 80% задач

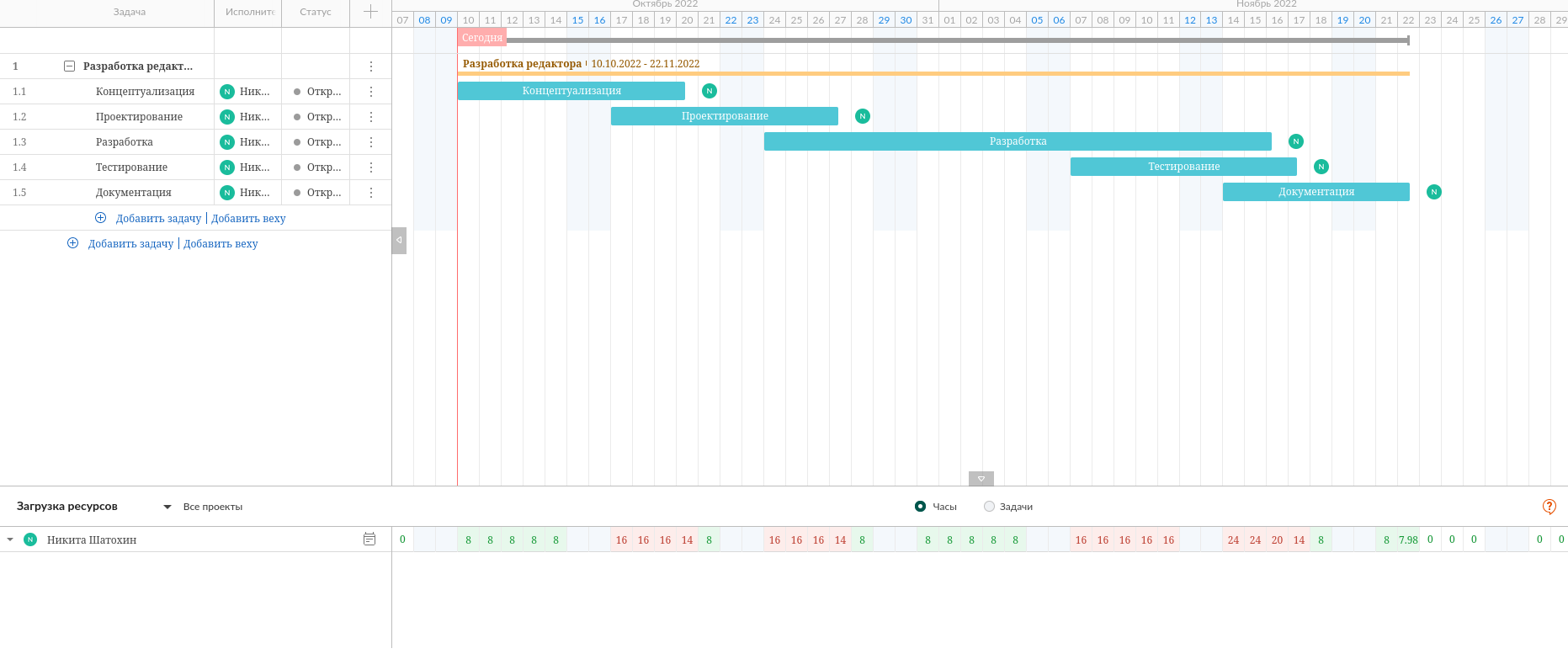
## 3.2 Календарный план разработки

План разработки удобно представлять с помощью диаграммы Ганта.

Диаграмма Ганта — это визуальный способ отображения запланированных задач.

Горизонтальные графики широко используются для планирования проектов любых размеров в разных отраслях и сферах. Это удобный способ показать, какая работа планируется к выполнению в определенный день и время.

Календарный план разработки игрового приложения в виде диаграммы Ганта представлен на рисунке 12.

Рисунок 12 – Календарный план работы

# 4 Тестирование программного продукта

## 4.1 Выбор метода обеспечения качества

Тестирование программного обеспечения - это оценка разрабатываемого программного обеспечения/продукта, чтобы проверить его возможности, способности и соответствие ожидаемым результатам. Существуют различные типы методов, используемые в области тестирования и обеспечения качества.

Тестирование – процесс исследования и контроль качества, который состоит из планирования, проектирования, собственно проверки и анализа ее результатов.

Таблица 1 – План тестирования

|  |  |  |
| --- | --- | --- |
| Тест кейс, № | Название | Описание |
| 1 | Тест сохранения файла | Тестирование файла на возможность его сохранения исключительно после уведомления пользователя |
| 2 | Тест открытия несуществующего файла | Тестирование программы на попытку открыть несуществующий файл |
| 3 | Тест подсветки синтаксиса кода | Тестирование области редактирования на предмет подсветки синтаксиса кода для каждого из поддерживаемых языков программирования |
| 4 | Тест настроек | Тестирование поведения программы при некорректном файле настроек |
| 5 | Тест работоспособности программы в среде GNU/Linux | Тестирование работоспособности программы в среде GNU/Linux с оболочкой KDE Plasma в сеансе Wayland |

## 4.2 Тестирование программного продукта

Для проведения тестирования программного продукта был составлен план проведения тестирования, который представлен в таблице 2.

Таблица 2 – Тест-кейс 1

|  |  |
| --- | --- |
| Идентификатор | 1 |
| Название | Тест сохранения файла |
| Приоритет | Высокий |
| Описание | Тестирование файла на возможность его сохранения исключительно после уведомления пользователя |
| Предусловие | Предусловие отсутствует |
| Шаги тестирования | 1. Запустить программу «Light» 2. Создать файл через меню «Файл» > «Создать новый файл» 3. Заполнить только что созданный файл текстовой информацией 4. Закрыть программу |
| Ожидаемый результат | Всплывающее окно с предложением о сохранении файла перед закрытием редактора |
| Постусловие | Постусловие отсутствует |
| Фактический результат | 1. Запускается программа «Light» 2. Создается файл через меню «Файл» > «Создать новый файл» 3. Только что созданный файл заполняется текстовой информацией 4. Программа закрывается 5. Появляется всплывающее окно с предложением о сохранении файла перед закрытием редактора |
| Статус | Пройден успешно |

Таблица 3 – Тест-кейс 2

|  |  |
| --- | --- |
| Идентификатор | 2 |
| Название | Тест открытия несуществующего файла |
| Приоритет | Высокий |
| Описание | Тестирование программы на попытку открыть несуществующий файл |
| Предусловие | Предусловие отсутствует |
| Шаги тестирования | 1. Запустить программу «Light» 2. Открыть несуществующий файл через меню «Файл» > «Открыть последние» 3. Выбрать из списка путь к файлу, который больше не существует. |
| Ожидаемый результат | Всплывающее окно с сообщением об ошибке, указывающей на отсутствие файла на диске |
| Постусловие | Постусловие отсутствует |
| Фактический результат | 1. Запускается программа «Light» 2. Открывается несуществующий файл через меню «Файл» > «Открыть последние» 3. Выбирается из списка путь к файлу, который больше не существует. 4. Появляется всплывающее окно с сообщением об ошибке, указывающей на отсутствие файла на диске |
| Статус | Пройден успешно |

Таблица 4 – Тест-кейс 3

|  |  |
| --- | --- |
| Идентификатор | 3 |
| Название | Тест подсветки синтаксиса кода |
| Приоритет | Высокий |
| Описание | Тестирование области редактирования на предмет подсветки синтаксиса кода для каждого из поддерживаемых языков программирования |
| Предусловие | Предусловие отсутствует |
| Шаги тестирования | 1. Запустить программу «Light» 2. Открыть файл с расширением поддерживаемого языка программирования через меню «Файл» > «Открыть файл...» |
| Ожидаемый результат | Всплывающее окно с сообщением об ошибке, указывающей на отсутсвие файла на диске |
| Постусловие | Постусловие отсутствует |
| Фактический результат | 1. Запускается программа «Light» 2. Открывается файл с расширением поддерживаемого языка программирования через меню «Файл» > «Открыть файл...» 3. В области редактирования текст каждой из лексем конкретного языка программирования из поддерживаемых, приобретает нужный цвет |
| Статус | Пройден успешно |

Таблица 5 – Тест-кейс 4

|  |  |
| --- | --- |
| Идентификатор | 4 |
| Название | Тест настроек |
| Приоритет | Высокий |
| Описание | Тестирование поведения программы при некорректном файле настроек |
| Предусловие | Предусловие отсутствует |
| Шаги тестирования | 1. Открыть файл «settings.json» 2. Ввести некорректные настройки 3. Запустить программу «Light» |
| Ожидаемый результат | 1. Запуск программы «Light» |
| Постусловие | Постусловие отсутствует |
| Фактический результат | 1. Открывается файл «settings.json» 2. Вводятся некорректные настройки 3. Запускается программа «Light» |
| Статус | Пройден успешно |

Таблица 6 – Тест-кейс 5

|  |  |
| --- | --- |
| Идентификатор | 5 |
| Название | Тест работоспособности программы в среде GNU/Linux |
| Приоритет | Высокий |
| Описание | Тестирование работоспособности программы в среде GNU/Linux с оболочкой KDE Plasma в сеансе Wayland |
| Предусловие | Предусловие отсутствует |
| Шаги тестирования | 1. Запустить программу «Light» |
| Ожидаемый результат | 1. Запуск программы «Light» 2. Использование ею протокола Wayland без прослойки совместимости с Xorg |
| Постусловие | Постусловие отсутствует |
| Фактический результат | 1. Запуск программы «Light» 2. Использоваение ею протокола Wayland без прослойки совместимости с Xorg 3. Отображение темы оформления, установленной пользователем в оболочке |
| Статус | Пройден успешно |

# 5 Разработка документации на программный продукт

К видам документации, разрабатываемой на разных этапах жизненного цикла программного продукта, относятся:

1. технические требования;
2. технические спецификации;
3. сведения о выпуске;
4. руководства (например, по установке ПП, пользователя, администратора, программиста, по технической поддержке и т.д.);
5. описание проекта;
6. планы;
7. задания исполнителям (задание, распределённое между конкретными людьми или группами, участвующими в реализации проекта);
8. отчёты о ходе работ - создаются менеджерами для контролирующих органов;
9. протоколы встреч и обсуждений;
10. отчёты о результатах активности;
11. журналы.

По своему назначению и ориентации на определенные задачи и группы пользователей, документацию ПП можно разбить на две группы:

1. программные документы (технологические документы управления разработкой ПП) - документы, которые предназначены, прежде всего, для самих разработчиков и их начальства;
2. эксплуатационные программные документы (документы, входящие в состав программных систем) - документы, предназначенные для конечных пользователей или же обслуживающего персонала, позволяющие им осваивать и квалифицированно применять эти системы для решения конкретных функциональных задач.

# Заключение

В данной курсовой работе мы познакомились с технологиями разработки ПО и получили навыки работы с различными инструментами, таких как: Trello, Miro, а также диаграммами Ганта.

Было разработано приложение «Light». Оно реализовано в установленные сроки и полностью соответствует потребностям пользователей.

Так же стоит отметить, что по итогу курсовой работы были проверены и закреплены знания в области разработки на языке программирования Python.

В дальнейшем планируется добавить поддержку пользовательских плагинов.

# Список литературы

* 1. Интернет-ресурс VScode - https://code.visualstudio.com/
  2. Интернет-ресурс Python - https://ru.wikipedia.org/wiki/Python
  3. Интернет-ресурс GitHub - https://github.com
  4. Интернет-ресурс Figma - https://www.figma.com/
  5. Интернет-ресурс Trello - https://trello.com
  6. Интернет-ресурс Gantpro - https://ganttpro.com/ru/
  7. Интернет ресурс Metanit по языку программирования Python - https://metanit.com/python/tutorial/
  8. Голицына О.Л, Партыка Т.Л, Попов И.И. Языки программирования — Форум, Инфра-М, 2017 - 677с.
  9. Мацяшек Л.А. Анализ требований и проектирование системы. Разработка информационных систем с использованием UML: Издательский дом «Вльямс» - 2017г. - 458с.
  10. Калабухова, Г.В. Компьютерный практикум по информатике: офисные технологии: учеб. пособие для студентов вузов, обуч. по напр. и спец. «Социальная работа» / Г.В. Калабухова, В.М. Титов. - М.: ФОРУМ: ИНФРА-М, 2013. - 336 с.
  11. Гагарина, Л.Г. Технология разработки программного обеспечения: учеб. пособие / Л.Г. Гагарина, Е.В. Кокорева, Б.Д. Виснадул; под ред. Л.Г. Гагариной. - М.: ИД «Форум»: ИНФРА-М, 2015. - 400 с
  12. Бахтизин, В.В. Технология разработки программного обеспечения: учеб. пособие / В.В. Бахтизин, Л.А. Глухова. - Минск: БГУИР, 2014. - 267 с.

# Приложение А

# Компонент db\_view

import logging

from globals import LOGGING\_LEVEL

from PyQt6.Qsci import \*

from PyQt6.QtCore import \*

from PyQt6.QtGui import \*

from PyQt6.QtWidgets import \*

from PyQt6.QtSql import \*

import sqlite3

from editor import File

logger = logging.getLogger(\_\_name\_\_)

logger.setLevel(LOGGING\_LEVEL)

logger.debug("Creating database connection")

class Database:

def \_\_init\_\_(self, file: File | str):

self.error\_while\_reading = False

self.file = file if isinstance(file, File) else File(file)

try:

self.connection = sqlite3.connect(self.file.path)

logger.debug(f"Подключение к базе данных {self.file.path} успешно")

except sqlite3.OperationalError as e:

logger.error(f"Не удалось открыть базу данных: {e}")

QMessageBox.critical(None, "Ошибка", f"Не удалось открыть базу данных: {e}")

self.error\_while\_reading = True

return

self.cursor = self.connection.cursor()

self.current\_table = self.get\_tables()[0][0]

# select current table

self.cursor.execute(f"SELECT \* FROM {self.current\_table}")

self.tables = self.get\_tables()

def get\_tables(self) -> list:

self.cursor.execute("SELECT name FROM sqlite\_master WHERE type='table';")

return self.cursor.fetchall()

def get\_columns(self, table):

self.cursor.execute(f"PRAGMA table\_info({table})")

return self.cursor.fetchall()

def execute(self, query) -> list:

self.cursor.execute(query)

return self.cursor.fetchall()

def edit\_element(self, column, value, table, identifier):

table = table if table else self.table

command = f"UPDATE {table} SET {column} = '{value}' WHERE id = {identifier}"

logger.debug(command)

self.cursor.execute(command)

self.connection.commit()

def delete\_element(self, table, row\_id):

self.cursor.execute(f"DELETE FROM {table} WHERE id = {row\_id}")

self.connection.commit()

def add\_element(self, table, values):

self.cursor.execute(f"INSERT INTO {table} VALUES ({values})")

self.connection.commit()

class DatabaseEditor(QTableView):

def \_\_init\_\_(self, database: Database, \*args, \*\*kwargs):

super(DatabaseEditor, self).\_\_init\_\_(\*args, \*\*kwargs)

self.database = database

self.file = database.file

self.error\_while\_reading = False

if database.error\_while\_reading:

self.error\_while\_reading = True

return

self.model = QStandardItemModel()

self.setModel(self.model)

self.setEditTriggers(QAbstractItemView.EditTrigger.AllEditTriggers)

self.update()

def update(self):

self.model.clear()

self.model.setHorizontalHeaderLabels([column[1] for column in self.database.get\_columns(self.database.tables[0][0])])

for row in self.database.execute(f"SELECT \* FROM {self.database.tables[0][0]}"):

items = [QStandardItem(str(item)) for item in row]

self.model.appendRow(items)

self.resizeColumnsToContents()

self.resizeRowsToContents()

def contextMenuEvent(self, event: QContextMenuEvent):

menu = QMenu(self)

add\_element\_action = QAction("Добавить элемент", self)

add\_element\_action.triggered.connect(self.add\_element)

menu.addAction(add\_element\_action)

menu.exec(event.globalPos())

super(DatabaseEditor, self).contextMenuEvent(event)

def add\_element(self):

self.model.appendRow([QStandardItem() for \_ in range(self.model.columnCount())])

self.database.add\_element(self.database.tables[0][0], ",".join(["NULL" for \_ in range(self.model.columnCount())]))

self.update()

def keyPressEvent(self, event: QKeyEvent):

if event.key() == Qt.Key.Key\_Delete:

self.database.delete\_element(self.model.item(self.currentIndex().row(), 0).text(), self.currentIndex().row())

self.update()

super(DatabaseEditor, self).keyPressEvent(event)

def dataChanged(self, topLeft: QModelIndex, bottomRight: QModelIndex, roles: list[int]):

element = self.model.item(topLeft.row(), topLeft.column()).text()

column = self.model.horizontalHeaderItem(topLeft.column()).text()

table = self.database.current\_table

identifier = topLeft.row() + 1

self.database.edit\_element(column, element, table, identifier)

# Компонент editor

# import ntpath

# import PyQt6.Qsci as lexers

# from PyQt6.Qsci import QsciScintilla

# from PyQt6.QtCore import Qt

# from PyQt6.QtGui import QColor, QFont, QKeyEvent

# from PyQt6.QtWidgets import QMessageBox

# from palettes import Theme

# class File:

# DATABASE\_EXTENTIONS = ('.db', '.sqlite', '.sqlite3', '.sqlitedb', '.sqlitedb3', '.sqlitedb3-shm', '.sqlitedb3-wal',)

# 

# def \_\_init\_\_(self, file\_path=None, saved=True):

# if file\_path:

# self.path = file\_path

# self.name = ntpath.basename(file\_path)

# self.extention = ntpath.splitext(file\_path)[1]

# self.new = False

# else:

# self.path = None

# self.name = None

# self.extention = None

# self.new = True

# 

# self.saved = saved

# 

# def update\_path(self, file\_path):

# """

# Метод `update\_path` обновляет путь к файлу а так же поля `name` и `extention`.

# """

# self.path = file\_path

# self.name = ntpath.basename(file\_path)

# self.extention = ntpath.splitext(file\_path)[1]

# 

# def save(self, data: str) -> None:

# """

# Метод `save` сохраняет изменения в файл.

# """

# with open(self.path, 'w', encoding='utf8') as f:

# f.write(data)

# 

# self.saved = True

# if self.new:

# self.new = False

# 

# def \_\_repr\_\_(self) -> str:

# return f"File({self.path}, {self.saved})"

# class CustomEditor(QsciScintilla):

# def \_\_init\_\_(self, theme, file\_path=None, file\_object=None, \*args, key\_press\_handler=None, \*\*kwargs):

# super(CustomEditor, self).\_\_init\_\_(\*args, \*\*kwargs)

# 

# self.key\_press\_handler = key\_press\_handler

# self.theme = theme

# 

# if file\_path or file\_object:

# try:

# self.file = file\_object if file\_object else File(file\_path)

# with open(self.file.path, "r", encoding='utf8') as f:

# text = f.read()

# self.setText(text)

# self.reload\_lexer(self.file.extention)

# except UnicodeDecodeError:

# msgBox = QMessageBox()

# msgBox.setIcon(QMessageBox.Icon.Critical)

# msgBox.setText(

# "Неподдерживаемый тип файла"

# )

# msgBox.setWindowTitle("Ошибка")

# msgBox.addButton('Продолжить', QMessageBox.ButtonRole.AcceptRole)

# msgBox.exec()

# 

# self.error\_while\_reading = True

# return

# else:

# self.file = File()

# 

# self.error\_while\_reading = False

# self.setUtf8(True)

# self.setCaretLineVisible(True)

# self.setCaretLineBackgroundColor(QColor('#e5e5e5'))

# self.setTabIndents(True)

# self.setAutoIndent(True)

# self.setIndentationGuides(False)

# self.setIndentationsUseTabs(True)

# self.setIndentationWidth(4)

# self.setMarginsBackgroundColor(QColor("#efefef"))

# self.setMarginLineNumbers(1, True)

# self.setMarginWidth(1, 50)

# self.setAutoCompletionSource(QsciScintilla.AutoCompletionSource.AcsAll)

# self.setAutoCompletionCaseSensitivity(True)

# self.setAutoCompletionReplaceWord(True)

# self.setAutoCompletionUseSingle(

# QsciScintilla.AutoCompletionUseSingle.AcusAlways)

# self.setAutoCompletionThreshold(0)

# self.setFolding(QsciScintilla.FoldStyle.BoxedTreeFoldStyle)

# self.setBraceMatching(QsciScintilla.BraceMatch.SloppyBraceMatch)

# self.setMatchedBraceBackgroundColor(QColor("white"))

# self.setUnmatchedBraceForegroundColor(QColor("red"))

# 

# if self.theme == Theme.DARK:

# self.setCaretLineBackgroundColor(QColor('#3b3b3b'))

# 

# editor\_bg = QColor("#353535")

# margins\_bg = QColor("#818181")

# margins\_text = QColor('white')

# 

# self.setFoldMarginColors(margins\_bg, margins\_bg)

# 

# self.setMarginsForegroundColor(margins\_text)

# self.setMarginsBackgroundColor(margins\_bg)

# self.setPaper(editor\_bg)

# 

# 

# def reload\_lexer(self, file\_extention):

# """

# Метод для перезагрузки лексера в зависимости от расширения файла.

# """

# match file\_extention[1:]:

# case 'json':

# lexer = lexers.QsciLexerJSON()

# case 'py':

# lexer = lexers.QsciLexerPython()

# case 'html':

# lexer = lexers.QsciLexerHTML()

# case 'cs':

# lexer = lexers.QsciLexerCSharp()

# case 'cpp':

# lexer = lexers.QsciLexerCPP()

# case 'java':

# lexer = lexers.QsciLexerJava()

# case 'css':

# lexer = lexers.QsciLexerCSS()

# case 'js':

# lexer = lexers.QsciLexerJavaScript()

# case \_:

# lexer = None

# 

# if not lexer:

# return

# 

# font = QFont('Fira Code')

# font.setPointSize(10)

# lexer.setFont(font)

# 

# if self.theme == Theme.DARK:

# blue = QColor('#9ecbf2')

# magenta = QColor('#b389c9')

# 

# lexer.setColor(QColor('white'), lexers.QsciLexerPython.Default)

# lexer.setColor(QColor('white'), lexers.QsciLexerPython.Identifier)

# lexer.setColor(QColor('white'), lexers.QsciLexerPython.Operator)

# 

# lexer.setColor(QColor(blue), lexers.QsciLexerPython.Number)

# 

# lexer.setColor(QColor('#f97578'), lexers.QsciLexerPython.Keyword)

# 

# lexer.setColor(QColor('#4c6a7d'), lexers.QsciLexerPython.Comment)

# lexer.setColor(QColor(blue), lexers.QsciLexerPython.CommentBlock)

# 

# lexer.setColor(QColor(blue), lexers.QsciLexerPython.SingleQuotedString)

# lexer.setColor(QColor(blue), lexers.QsciLexerPython.SingleQuotedFString)

# lexer.setColor(QColor(blue), lexers.QsciLexerPython.DoubleQuotedString)

# lexer.setColor(QColor(blue), lexers.QsciLexerPython.DoubleQuotedFString)

# 

# lexer.setColor(QColor(blue), lexers.QsciLexerPython.TripleSingleQuotedFString)

# lexer.setColor(QColor(blue), lexers.QsciLexerPython.TripleSingleQuotedString)

# lexer.setColor(QColor(blue), lexers.QsciLexerPython.TripleDoubleQuotedFString)

# lexer.setColor(QColor(blue), lexers.QsciLexerPython.TripleDoubleQuotedString)

# lexer.setColor(QColor(magenta), lexers.QsciLexerPython.ClassName)

# lexer.setColor(QColor(magenta), lexers.QsciLexerPython.FunctionMethodName)

# lexer.setColor(QColor(magenta), lexers.QsciLexerPython.Decorator)

# 

# lexer.setPaper(QColor("#353535"))

# 

# 

# self.setLexer(lexer)

# def changeEvent(self, event):

# """

# The changeEvent function is called whenever the user changes a text in the editor.

# It is used to track whether there are unsaved changes in the code.

# """

# self.file.saved = False

# print(self.file)

# def keyPressEvent(self, e: QKeyEvent) -> None:

# """

# The keyPressEvent function is called whenever the user presses a key.

# If the control modifier and spacebar are pressed at the same time,

# the autoCompleteFromAll function is called to show all possible completions.

# 

# :param e:QKeyEvent: Pass the event to the parent class

# """

# if self.key\_press\_handler:

# self.key\_press\_handler()

# if e.modifiers() == Qt.KeyboardModifier.ControlModifier and e.key(

# ) == Qt.Key.Key\_Space:

# self.autoCompleteFromAll()

# return

# super().keyPressEvent(e)

# Компонент globals

# import os

# import logging

# PROJECT\_DIRECTORY = os.path.dirname(\_\_file\_\_).replace('\\', '/').replace('c:/', 'C:/')

# SETTINGS\_PATH = os.path.join(PROJECT\_DIRECTORY, 'settings.json')

# WINDOW\_ICON = os.path.join(PROJECT\_DIRECTORY, 'logo.png')

# LOGGING\_LEVEL = logging.NOTSET

# Компонент main

# import sys

# from PyQt6.QtWidgets import QApplication, QStyleFactory

# from palettes import get\_dark\_palette, Theme

# from utils import SettingsInstance

# from window import CustomMainWindow

# if \_\_name\_\_ == '\_\_main\_\_':

# app = QApplication(sys.argv)

# QApplication.setStyle(QStyleFactory.create('Fusion'))

# 

# settings = SettingsInstance()

# if settings.default\_theme == Theme.DARK:

# app.setPalette(get\_dark\_palette())

# gui = CustomMainWindow(settings)

# sys.exit(app.exec())

# Компонент palettes

# import enum

# from PyQt6.QtCore import Qt

# from PyQt6.QtGui import QColor, QPalette

# class Theme(enum.Enum):

# LIGHT = 'light'

# DARK = 'dark'

# @staticmethod

# def get\_theme(value):

# return {'dark': Theme.DARK, 'light': Theme.LIGHT}[value]

# def get\_dark\_palette():

# dark\_palette = QPalette()

# dark\_palette.setColor(QPalette.ColorRole.Window, QColor(53, 53, 53))

# dark\_palette.setColor(QPalette.ColorRole.WindowText, Qt.GlobalColor.white)

# dark\_palette.setColor(QPalette.ColorRole.Base, QColor(25, 25, 25))

# dark\_palette.setColor(QPalette.ColorRole.AlternateBase, QColor(53, 53, 53))

# dark\_palette.setColor(QPalette.ColorRole.ToolTipBase, Qt.GlobalColor.white)

# dark\_palette.setColor(QPalette.ColorRole.ToolTipText, Qt.GlobalColor.white)

# dark\_palette.setColor(QPalette.ColorRole.Text, Qt.GlobalColor.white)

# dark\_palette.setColor(QPalette.ColorRole.Button, QColor(53, 53, 53))

# dark\_palette.setColor(QPalette.ColorRole.ButtonText, Qt.GlobalColor.white)

# dark\_palette.setColor(QPalette.ColorRole.BrightText, Qt.GlobalColor.red)

# dark\_palette.setColor(QPalette.ColorRole.Link, QColor(42, 130, 218))

# dark\_palette.setColor(QPalette.ColorRole.Highlight, QColor(42, 130, 218))

# dark\_palette.setColor(QPalette.ColorRole.HighlightedText,

# Qt.GlobalColor.white)

# return dark\_palette

# Компонент tabmanager

# from PyQt6.QtWidgets import QTabWidget

# from editor import CustomEditor, File

# from db\_view import DatabaseEditor, Database

# class TabManager(QTabWidget):

# def \_\_init\_\_(self, theme):

# super(TabManager, self).\_\_init\_\_()

# self.theme = theme

# def editors\_states(self) -> dict:

# """

# The editors\_states function returns a dictionary of the unsaved\_changes attribute for each editor.

# The unsaved\_changes attribute is True if there are any changes to the editor's document that have not been saved.

# """

# return {i: self.widget(i).file.saved for i in range(self.count())}

# 

# def get\_files(self):

# """

# The get\_fiels function returns a list of the file paths for each editor.

# """

# return [self.widget(i).file for i in range(self.count())]

# def show\_file(self, filepath=None, is\_new=False):

# """

# The show\_file function creates a new tab in the editor and displays the file contents.

# If no filepath is provided, it creates an empty editor instead.

# 

# :param filepath=None: Determine whether the file is opened from a file or created new

# :param is\_new=False: Determine whether the file should be opened in a new tab or not

# 

# :return: True if the file was opened successfully, False otherwise

# """

# if filepath:

# file = File(filepath)

# if file.extention in File.DATABASE\_EXTENTIONS:

# db = Database(filepath)

# newtab = DatabaseEditor(db)

# else:

# newtab = CustomEditor(self.theme, file\_object=file)

# newtab.setCursorPosition(0, 0)

# newtab.ensureCursorVisible()

# newtab.setFocus()

# if newtab.error\_while\_reading:

# return False

# newtabName = newtab.file.name

# elif is\_new:

# newtab = CustomEditor(self.theme)

# newtabName = "Без имени"

# self.addTab(newtab, newtabName)

# self.setCurrentWidget(newtab)

# 

# 

# return True

# Компонент toolbar

# from PyQt6.QtWidgets import QTabWidget

# from editor import CustomEditor, File

# from db\_view import DatabaseEditor, Database

# class TabManager(QTabWidget):

# def \_\_init\_\_(self, theme):

# super(TabManager, self).\_\_init\_\_()

# self.theme = theme

# def editors\_states(self) -> dict:

# """

# The editors\_states function returns a dictionary of the unsaved\_changes attribute for each editor.

# The unsaved\_changes attribute is True if there are any changes to the editor's document that have not been saved.

# """

# return {i: self.widget(i).file.saved for i in range(self.count())}

# 

# def get\_files(self):

# """

# The get\_fiels function returns a list of the file paths for each editor.

# """

# return [self.widget(i).file for i in range(self.count())]

# def show\_file(self, filepath=None, is\_new=False):

# """

# The show\_file function creates a new tab in the editor and displays the file contents.

# If no filepath is provided, it creates an empty editor instead.

# 

# :param filepath=None: Determine whether the file is opened from a file or created new

# :param is\_new=False: Determine whether the file should be opened in a new tab or not

# 

# :return: True if the file was opened successfully, False otherwise

# """

# if filepath:

# file = File(filepath)

# if file.extention in File.DATABASE\_EXTENTIONS:

# db = Database(filepath)

# newtab = DatabaseEditor(db)

# else:

# newtab = CustomEditor(self.theme, file\_object=file)

# newtab.setCursorPosition(0, 0)

# newtab.ensureCursorVisible()

# newtab.setFocus()

# if newtab.error\_while\_reading:

# return False

# newtabName = newtab.file.name

# elif is\_new:

# newtab = CustomEditor(self.theme)

# newtabName = "Без имени"

# self.addTab(newtab, newtabName)

# self.setCurrentWidget(newtab)

# 

# 

# return True

# Компонент tree

# from PyQt6.QtWidgets import QTreeView, QApplication

# from PyQt6.QtGui import QFileSystemModel

# from PyQt6.QtCore import QDir, QModelIndex, QSortFilterProxyModel

# FILE\_SYSTEM\_ROOT = QDir.rootPath()

# class ProxyModel(QSortFilterProxyModel):

# def \_\_init\_\_(self, parent=None):

# super().\_\_init\_\_(parent)

# self.\_root\_path = ""

# def filterAcceptsRow(self, source\_row, source\_parent):

# source\_model = self.sourceModel()

# if self.\_root\_path and isinstance(source\_model, QFileSystemModel):

# root\_index = source\_model.index(self.\_root\_path).parent()

# if root\_index == source\_parent:

# index = source\_model.index(source\_row, 0, source\_parent)

# return index.data(

# QFileSystemModel.Roles.FilePathRole) == self.\_root\_path

# return True

# @property

# def root\_path(self):

# return self.\_root\_path

# @root\_path.setter

# def root\_path(self, p):

# self.\_root\_path = p

# self.invalidateFilter()

# class FileTree(QTreeView):

# def \_\_init\_\_(self, path=FILE\_SYSTEM\_ROOT):

# QTreeView.\_\_init\_\_(self)

# 

# self.dirModel = QFileSystemModel()

# self.dirModel.setRootPath(QDir.rootPath())

# self.dirModel.setFilter(QDir.Filter.Files | QDir.Filter.Dirs | QDir.Filter.NoDotAndDotDot)

# root\_index = self.dirModel.index(path).parent()

# self.proxy = ProxyModel(self.dirModel)

# self.proxy.setSourceModel(self.dirModel)

# self.proxy.root\_path = path

# self.setModel(self.proxy)

# proxy\_root\_index = self.proxy.mapFromSource(root\_index)

# self.setRootIndex(proxy\_root\_index)

# 

# self.setHeaderHidden(True)

# self.hideColumn(1)

# self.hideColumn(2)

# self.hideColumn(3)

# self.setMaximumWidth(1000)

# self.setMinimumWidth(0)

# if \_\_name\_\_ == '\_\_main\_\_':

# import sys

# app = QApplication(sys.argv)

# w = FileTree()

# w.show()

# sys.exit(app.exec())

# Компонент utils

# import json

# from pprint import pprint

# from globals import SETTINGS\_PATH

# from palettes import Theme

# class SettingsInstance:

# def \_\_init\_\_(self):

# with open(SETTINGS\_PATH, 'r', encoding='utf8') as f:

# self.\_\_settings = json.load(f)

# self.refresh()

# def refresh(self):

# self.hotkeys\_settings = self.\_\_settings.get('Горячие клавиши')

# self.default\_theme = Theme.get\_theme(self.\_\_settings.get('Тема по умолчанию'))

# self.run\_settings = self.\_\_settings.get('Настройки запуска')

# self.recent\_files = self.\_\_settings.get('Последние файлы')

# def save(self):

# with open(SETTINGS\_PATH, 'w', encoding='utf8') as f:

# json.dump(self.\_\_settings, f, ensure\_ascii=False)

# def set(self, key: str, value):

# self.\_\_settings[key] = value

# self.save()

# def update(self, chains: dict):

# self.\_\_settings.update(chains)

# self.save()

# 

# def add\_recent(self, file\_path):

# recent: list = self.\_\_settings['Последние файлы']

# if file\_path in recent:

# return

# 

# if len(recent) >= 10:

# recent.pop(0)

# 

# self.\_\_settings.update({'Последние файлы': recent + [file\_path]})

# self.refresh()

# self.save()

# def describe(self):

# pprint(self.\_\_settings)

# if \_\_name\_\_ == '\_\_main\_\_':

# settings = SettingsInstance()

# settings.describe()

# Компонент window

# import os

# import subprocess

# from functools import partial

# from typing import Tuple

# from PyQt6.QtCore import QCoreApplication, QModelIndex, Qt, pyqtSlot

# from PyQt6.QtGui import QAction, QFileSystemModel, QIcon

# from PyQt6.QtWidgets import (QDockWidget, QFileDialog, QFrame, QLabel,

# QMainWindow, QMessageBox, QVBoxLayout)

# from globals import WINDOW\_ICON

# from tabmanager import TabManager

# from tree import FileTree

# class CustomMainWindow(QMainWindow):

# def \_\_init\_\_(self, settings):

# super(CustomMainWindow, self).\_\_init\_\_()

# self.setWindowIcon(QIcon(WINDOW\_ICON))

# self.setGeometry(300, 300, 800, 400)

# self.setWindowTitle("Редактор кода")

# self.settings = settings

# self.theme = self.settings.default\_theme

# layout = QVBoxLayout()

# frame = QFrame(self)

# frame.setLayout(layout)

# self.setCentralWidget(frame)

# self.tab\_manager = TabManager(theme=self.theme)

# self.tab\_manager.setVisible(False)

# self.file\_tree = FileTree()

# self.file\_tree.doubleClicked.connect(self.openFromTree)

# self.directory\_sidebar = QDockWidget("Проводник", self)

# self.directory\_sidebar.setFloating(False)

# self.directory\_sidebar.setAllowedAreas(

# Qt.DockWidgetArea.LeftDockWidgetArea

# | Qt.DockWidgetArea.RightDockWidgetArea)

# self.directory\_sidebar.setWidget(self.file\_tree)

# self.directory\_sidebar.setVisible(False)

# self.addDockWidget(Qt.DockWidgetArea.LeftDockWidgetArea,

# self.directory\_sidebar)

# self.placeholder = QLabel(

# 'Создайте новый файл (Ctrl + n)\nили\nоткройте существующий (Ctrl + o).'

# )

# self.placeholder.setAlignment(Qt.AlignmentFlag.AlignCenter)

# self.\_createActions()

# self.\_createMenuBar()

# layout.addWidget(self.tab\_manager)

# layout.addWidget(self.placeholder)

# self.show()

# def get\_editors(self):

# """

# Возвращает список всех редакторов.

# """

# return [

# self.tab\_manager.widget(i) for i in range(self.tab\_manager.count())

# ]

# def closeEvent(self, event):

# """

# Переопределение метода закрытия окна.

# """

# are\_all\_saved = all(e.file.saved for e in self.get\_editors())

# if not are\_all\_saved:

# msgBox = QMessageBox()

# msgBox.setIcon(QMessageBox.Icon.Warning)

# msgBox.setText(

# "Один из файлов не был сохранен, вы действительно хотите выйти?"

# )

# msgBox.setWindowTitle("Несохраненные изменения")

# msgBox.addButton('Сохранить', QMessageBox.ButtonRole.ActionRole)

# msgBox.addButton('Да', QMessageBox.ButtonRole.YesRole)

# msgBox.addButton('Нет', QMessageBox.ButtonRole.NoRole)

# reply = msgBox.exec()

# # 0 - Сохранить

# # 1 - Да

# # 2 - Нет

# if reply == 0:

# for editor in self.get\_editors():

# if not editor.file.saved:

# self.saveActionHandler(editor)

# event.accept()

# elif reply == 1:

# event.accept()

# else:

# event.ignore()

# def \_createActions(self):

# """

# Описание действий, которые будут выполняться при нажатии на кнопки в меню, а так же

# инициализация горячих клавиш для них.

# """

# self.newAction = QAction("&Создать новый файл", self)

# self.newAction.setShortcut(

# self.settings.hotkeys\_settings['Создать новый файл'])

# self.newAction.triggered.connect(self.newActionHandler)

# 

# self.newDatabaseAction = QAction("&Создать новую базу данных", self)

# self.newDatabaseAction.setShortcut(

# self.settings.hotkeys\_settings['Создать новую базу данных'])

# self.newDatabaseAction.triggered.connect(self.newDatabaseActionHandler)

# # TODO: сделать чтобы хоть как-то работало

# self.newWindowAction = QAction("&Новое окно", self)

# self.newWindowAction.setShortcut(

# self.settings.hotkeys\_settings['Новое окно'])

# self.newWindowAction.triggered.connect(self.newWindowActionHandler)

# self.openAction = QAction("&Открыть файл...", self)

# self.openAction.setShortcut(

# self.settings.hotkeys\_settings['Открыть файл'])

# self.openAction.triggered.connect(self.openActionHandler)

# self.openDirectoryAction = QAction("&Открыть директорию...", self)

# self.openDirectoryAction.setShortcut(

# self.settings.hotkeys\_settings['Открыть директорию'])

# self.openDirectoryAction.triggered.connect(

# self.openDirectoryActionHandler)

# self.saveAction = QAction("&Сохранить", self)

# self.saveAction.setShortcut(

# self.settings.hotkeys\_settings['Сохранить'])

# self.saveAction.triggered.connect(self.saveActionHandler)

# self.saveAsAction = QAction("&Сохранить как...", self)

# self.saveAsAction.setShortcut(

# self.settings.hotkeys\_settings['Сохранить как'])

# self.saveAsAction.triggered.connect(self.saveAsActionHandler)

# self.closeAction = QAction("&Закрыть файл", self)

# self.closeAction.setShortcut(

# self.settings.hotkeys\_settings['Закрыть файл'])

# self.closeAction.triggered.connect(self.closeActionHandler)

# self.exitAction = QAction("&Выйти", self)

# self.exitAction.setShortcut(self.settings.hotkeys\_settings['Выйти'])

# self.exitAction.triggered.connect(QCoreApplication.instance().quit)

# self.copyAction = QAction("&Копировать", self)

# self.pasteAction = QAction("&Вставить", self)

# self.cutAction = QAction("&Вырезать", self)

# self.runAction = QAction("&Запуск открытого файла", self)

# self.runAction.setShortcut(

# self.settings.hotkeys\_settings['Запуск файла'])

# self.runAction.triggered.connect(self.runActionHandler)

# self.helpContentAction = QAction("&Документация", self)

# self.aboutAction = QAction("&О редакторе", self)

# self.aboutAction.setShortcut(

# self.settings.hotkeys\_settings['О редакторе'])

# self.aboutAction.triggered.connect(self.aboutActionHandler)

# def \_createMenuBar(self):

# """

# \_createMenuBar создает меню и добавляет в них действия.

# """

# menuBar = self.menuBar()

# fileMenu = menuBar.addMenu("&Файл")

# fileMenu.addAction(self.newAction)

# fileMenu.addAction(self.newDatabaseAction)

# fileMenu.addAction(self.newWindowAction)

# fileMenu.addSeparator()

# fileMenu.addAction(self.openAction)

# fileMenu.addAction(self.openDirectoryAction)

# self.openRecentMenu = fileMenu.addMenu("&Открыть последние")

# self.openRecentMenu.aboutToShow.connect(self.populateOpenRecent)

# fileMenu.addSeparator()

# fileMenu.addAction(self.saveAction)

# fileMenu.addAction(self.saveAsAction)

# fileMenu.addAction(self.closeAction)

# fileMenu.addSeparator()

# fileMenu.addAction(self.exitAction)

# editMenu = menuBar.addMenu("&Правка")

# editMenu.addAction(self.copyAction)

# editMenu.addAction(self.pasteAction)

# editMenu.addAction(self.cutAction)

# editMenu = menuBar.addMenu("&Запуск")

# editMenu.addAction(self.runAction)

# helpMenu = menuBar.addMenu("&Справка")

# helpMenu.addAction(self.aboutAction)

# def populateOpenRecent(self):

# """

# Обновляет список последних открытых файлов.

# """

# self.openRecentMenu.clear()

# actions = []

# self.settings.refresh()

# paths = self.settings.recent\_files

# for path in paths:

# action = QAction(path, self)

# action.triggered.connect(partial(self.openActionHandler, path))

# actions.append(action)

# self.openRecentMenu.addActions(actions)

# @pyqtSlot(QModelIndex)

# def openFromTree(self, index):

# """

# openFromTree открывает файл, который был выбран в дереве файлов.

# """

# ix = self.file\_tree.proxy.mapToSource(index)

# path = ix.data(QFileSystemModel.Roles.FilePathRole)

# if not os.path.isdir(path):

# self.openActionHandler(path)

# def add\_recent(self):

# """

# add\_recent добавляет открытый файл в список последних файлов.

# """

# new\_recent = self.tab\_manager.currentWidget().file.path

# self.settings.add\_recent(new\_recent)

# def add\_tab(self, path='', is\_new\_file=False):

# """

# add\_tab добавляет новую вкладку в редактор.

# """

# if self.tab\_manager.count() == 0:

# self.placeholder.setVisible(False)

# self.tab\_manager.setVisible(True)

# return self.tab\_manager.show\_file(path, is\_new\_file)

# def remove\_tab(self, idx):

# """

# remove\_tab удаляет вкладку с индексом idx.

# """

# self.tab\_manager.removeTab(idx)

# if self.tab\_manager.count() == 0:

# self.placeholder.setVisible(True)

# self.tab\_manager.setVisible(False)

# def newActionHandler(self):

# """

# The newActionHandler function creates a new file in the editor.

# """

# self.add\_tab(is\_new\_file=True)

# 

# def newDatabaseActionHandler(self):

# """

# newDatabaseActionHandler создает новую базу данных.

# """

# import sqlite3

# path = QFileDialog.getSaveFileName(

# self, "Создать базу данных", "", "База данных (\*.db)")[0]

# if path:

# sqlite3.connect(path)

# self.openActionHandler(path)

# else:

# QMessageBox.warning(self, "Ошибка", "Не удалось создать базу данных")

# def newWindowActionHandler(self):

# """

# Метод создает новое окно редактора.

# """

# raise NotImplementedError

# def openActionHandler(self, file\_path=None):

# """

# openActionHandler открывает диалоговое окно выбора файла и открывает его в редакторе, или

# открывает файл, который был передан в качестве аргумента.

# Так же является обработчиком действия "Открыть файл".

# """

# if not file\_path:

# file\_path, \_ = self.choose\_file()

# if file\_path:

# if self.add\_tab(path=file\_path):

# self.add\_recent()

# self.tab\_manager.currentWidget().file.saved = True

# def choose\_directory(self):

# directory = QFileDialog.getExistingDirectory(self, "Выберите папку",

# os.path.expanduser('~'))

# return directory

# def openDirectoryActionHandler(self):

# """

# Этот метод открывает диалоговое окно выбора папки и открывает дерево файлов в док панели.

# Так же является обработчиком действия "Открыть папку".

# """

# directory = self.choose\_directory()

# if directory:

# self.file\_tree = FileTree(directory)

# self.file\_tree.doubleClicked.connect(self.openFromTree)

# self.directory\_sidebar.setWidget(self.file\_tree)

# self.directory\_sidebar.setVisible(True)

# def openRecentActionHandler(self):

# pass

# def saveFile(self):

# """

# saveFile сохраняет файл в зависимости от его типа: новый или уже сущестующий.

# Так же является обработчиком действия "Сохранить".

# """

# editor = self.tab\_manager.currentWidget()

# if editor.file.new:

# self.tab\_manager.currentWidget().reload\_lexer(

# editor.file.extention)

# current\_tab\_index = self.tab\_manager.currentIndex()

# self.tab\_manager.setTabText(current\_tab\_index, editor.file.name)

# self.add\_recent()

# code = editor.text()

# code = code.replace('\r', '')

# editor.file.save(code)

# def runActionHandler(self):

# """

# Запуск файла в соответствии с его расширением.

# Так же является обработчиком действия "Запустить".

# """

# if editor := self.tab\_manager.currentWidget():

# file = editor.file

# command = self.settings.run\_settings.get(file.extention[1:], None)

# if command:

# self.saveActionHandler()

# command = command.replace('$file\_path', file.path)

# subprocess.Popen('start /wait ' + command,

# shell=True,

# creationflags=subprocess.CREATE\_NEW\_CONSOLE)

# else:

# msg = QMessageBox()

# msg.setIcon(QMessageBox.Icon.Warning)

# msg.setText(

# "Нет настроек запуска для данного файла. Их можно установить в файле settings.json в папке редактора."

# )

# msg.setWindowTitle("Ошибка")

# msg.setStandardButtons(QMessageBox.StandardButton.Ok)

# msg.exec()

# def saveActionHandler(self, editor=None):

# """

# Сохранение файла в запрошенной директории, а так же обработчик действия "Сохранить как"

# """

# if not editor:

# editor = self.tab\_manager.currentWidget()

# if editor.file.new:

# file\_path, \_ = self.choose\_file\_save()

# if file\_path:

# editor.file.update\_path(file\_path)

# self.saveFile()

# else:

# self.saveFile()

# def saveAsActionHandler(self):

# """

# Сохранение файла с выбором пути, а так же обработчик действия "Сохранить как"

# """

# editor = self.tab\_manager.currentWidget()

# file\_path, \_ = self.choose\_file\_save()

# if file\_path:

# editor.file.update\_path(file\_path)

# self.saveFile(file\_path)

# def closeActionHandler(self):

# """

# Метод закрытия файла, а так же обработчик действия "Закрыть"

# """

# self.remove\_tab(self.tab\_manager.currentIndex())

# def copyActionHandler(self):

# """

# Обработчик действия "Копировать"

# """

# self.tab\_manager.currentWidget().copy()

# def pasteActionHandler(self):

# """

# Обработчик действия "Вставить"

# """

# self.tab\_manager.currentWidget().paste()

# def cutActionHandler(self):

# """

# Обработчик действия "Вырезать"

# """

# self.tab\_manager.currentWidget().cut()

# def aboutActionHandler(self):

# """

# The aboutActionHandler function displays a message box containing information about the editor.

# """

# text = "<center>" \

# "<h1>Редактор кода Light</h1>" \

# "<p>Light - это редактор кода с подсветкой синтаксиса, автодополнением и множеством других функций.</p>" \

# "<p>Он был создан в качестве курсового проекта для колледжа, и построен на виджетах PyQt6 и Qscintilla</p>" \

# f"<img src={WINDOW\_ICON}>" \

# "</center>" \

# "<p>Вресия 1.0.0<br/>" \

# "Copyright &copy; Жужелица Inc.</p>"

# QMessageBox.about(self, "О программе", text)

# def choose\_file(self) -> Tuple[str, str]:

# """

# Метод choose\_file открывает диалоговое окно выбора файла и возвращает путь к нему.

# 

# `path, filters = self.choose\_file()`

# :return: Путь к файлу и фильтры

# """

# return QFileDialog.getOpenFileName(self, 'Открыть файл',

# \_\_file\_\_) or \_\_file\_\_

# def choose\_file\_save(self) -> Tuple[str, str]:

# return QFileDialog.getSaveFileName(self, 'Save File')

# Приложение Б

Документ о концепции и границах проекта

# Бизнес-требования

## Исходные данные

Решение о создании продукта было принято из идеологических предпочтений разработчика.

## Возможности бизнеса

Большинство современного программного  обеспечения имеет закрытый исходнй код, стоит денег, является невероятно перегруженным, и в принципе не является свободным. Такая ситуация на рынке заставляет крупные корпорации чувствовать себя монополистами в какой-либо сфере, подсаживая конечного потребителя на собственные продукты, добавляя туда вредные функции, такие как шпионаж. Редактор кода Light является полностью свободным программным продуктом с открытым исходным кодом. Это позволяет использовать данную разработку не только для безопасного домашнего использования, но и малого бизнеса.

## Бизнес-цели и критерии успеха

Достигнуть широкого распространения продукта, повысить количество пожертвований. Привлечение пользователей будет осуществляться через интернет, т.к. продукт будет размещаться на соответствующих платформах.

## Потребности клиентов или рынка

Зачастую программисты при выборе редактора кода предпочитают удобство и функциональность. «Light» был создан исходя из данных критериев. Он совмешает в себе минималистичный дизайн, наличие всех необходимых функций для удобного редактирования кода, и гибкие настройки. А главный плюс в том, что разработка является мультиплатформенной, что позволяет ей работать практически на любых операционных системах.

## Бизнес-риски

Данный продукт имеет риск быть не востребованным т.к. многие привыкли к интерфесу и функционалу проприетарных решений, и не желают менять свое рабочее пространство.

# Образ решения

## Положение об образе проекта

* для программистов
* Для тех, кто работает с текстом.

## Основные функции

## Основные функции продукта:

Редактирование текстовых файлов

Редактирование базы данных SQLite

Подсветка синтаксиса кода

Возможность переключение между светлой и темной темами

# Масштабы и ограничения проекта

Система проекта обладает красивым и привлекательным внешним видом и не сложным, но фундаментальным функционалом текстового редактора, который понятен для каждого. Но, в системе отсутствуют пользовательские плагины, из-за чего рашираться функционал редактора можно лишь изменяя исходный код.

## Объем проекта

Редактор кода с возможностью редактирования текстовых файлов, базы данных SQLite, а так же редактирование базы данных SQLite.

# Бизнес-контекст

## Профили заинтересованных лиц

Студент: Шатохин

Преподаватель: Манакова О.П.

Программисты

## Приоритеты проекта

Функции проекта должны быть освоены на 100%, качество проекта должно быть на наивысшем возможном уровне, графическая составляющая должна быть интуитивно понятна и красива, затраты проекта должны быть сведены к минимуму, достигается это набором высоко квалифицированных программистов с правильным использованием их возможностей.

## Операционная среда

Если говорить в целом, то пользователь любой операционной системы может использовать наш продукт.

# Приложение В

Руководство пользователя

1. Для запуска приложения следует запустить соответсвующий исполняемый файл.
2. Чтобы создать новый файл, нажмите комбинацию клавиш «Ctrl + C», либо «Файл» > «Создать новый файл»
3. Чтобы открыть существующий файл, нажмите комбинацию клавиш «Ctrl + O», либо «Файл» > «Открыть файл...»
4. Чтобы открыть директорию, перейдите в «Файл» > «Создать новый файл»
5. Чтобы сохранить редактируемый файл, нажмите комбинацию клавиш «Ctrl + S», либо «Файл» > «Сохранить» или «Сохранить как...».