Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Объектно-ориентированное программирование

К защите допустить:

И.О. Заведующего кафедрой информатики
_____С. И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ДОКУМЕНТООБОРОТА

БГУИР КП 1-40 04 01

Студент Н. С. Сенько

Руководитель Е. В. Тушинская

Нормоконтролер Е. В. Тушинская

Минск 2024

СОДЕРЖАНИЕ

Введение 5
1 Анализ предметной области 6
1.1 Обзор аналогов 6
1.2 Постановка задачи 7
2 Проектирование программного средства 9
2.1 Разработка функциональности программного средства 9
2.2 Архитектура программного средства 9
3 Разработка программного средства 12
3.1 Разработка уровня представления 12
3.2 Разработка уровня приложения 14
3.3 Разработка уровня данных 14
4 Проверка работоспособности приложения 16
Заключение 18
Список использованных источников 19
Приложение А (обязательное) Исходный код программы 20
Приложение Б (обязательное) Схемы алгоритмов программного средства 26

ВВЕДЕНИЕ

Современный мир стал таким благодаря технологиям. В различных сферах жизни и работы технологии играют ключевую роль, повышая эффективность и упрощая выполнение задач. Одной из таких важных и востребованных технологий является система документооборота. С ростом объема информации и документов, требующих обработки, управление документами становится все более сложной задачей. Однако, благодаря современным программным решениям, эта задача значительно упрощается.

Системы документооборота позволяют автоматизировать процессы создания, хранения, поиска и управления документами. Одной из особенностей таких систем является возможность создания документов из шаблонов, что значительно ускоряет процесс подготовки документов и уменьшает вероятность ошибок.

Особенно это важно для компаний, которые часто составляют различные договоры. У таких компаний могут быть типовые договоры для различных ситуаций: аренда, поставка, оказание услуг и т.д. При составлении конкретного договора пользователю нужно лишь заполнить определенные поля, такие как имена сторон, даты, суммы и другие специфические детали. Это позволяет существенно сократить время на подготовку документов и минимизировать ошибки, связанные с ручным вводом данных.

В данной курсовой работе был разработан программный продукт, обеспечивающий эффективное управление документами с возможностью их создания на основе шаблонов. Пользователи могут использовать данное приложение для создания, редактирования и управления документами, что позволяет значительно упростить и ускорить документооборот в организации.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Обзор аналогов

В данном пункте рассмотрим примеры популярных систем документооборота с возможностью создания документов из шаблонов. Приведем примеры четырех таких систем, которые являются аналогами разрабатываемого в курсовом проекте приложения. DocuWare — это облачная система управления документами, которая позволяет автоматизировать все процессы документооборота. Система предоставляет широкий функционал для создания документов из шаблонов, что позволяет пользователям быстро и без ошибок создавать необходимые документы. DocuWare также обеспечивает надежное хранение документов, возможность их поиска и доступа к ним из любого места, что делает систему удобной для использования в современных условиях работы. Laserfiche — это система управления документами и бизнес-процессами, которая позволяет автоматизировать процессы создания, хранения и обработки документов. Система поддерживает создание документов из шаблонов, что позволяет пользователям быстро создавать стандартизированные документы. Laserfiche также предоставляет инструменты для управления бизнес-процессами, что позволяет автоматизировать и оптимизировать различные процессы в организации. SharePoint — это платформа для совместной работы и управления документами от компании Microsoft. SharePoint предоставляет возможность создания документов на основе шаблонов, что позволяет стандартизировать процесс подготовки документов и уменьшить вероятность ошибок. Платформа также поддерживает функции совместной работы, управление рабочими процессами и интеграцию с другими продуктами Microsoft, что делает ее мощным инструментом для управления информацией в организации. Анализ аналогов показал, что современные системы управления документами предоставляют широкий функционал для автоматизации документооборота и создания документов из шаблонов. Однако на основе проведенного исследования аналогов не было найдено решения с интегрированной поддержкой шаблонов и составления документов на их основе, которое бы в полной мере удовлетворяло все потребности пользователей в удобстве и эффективности. В рамках курсового проекта была поставлена задача разработать приложение, обеспечивающее эффективное управление документами и создание документов на основе шаблонов. Основные требования к проекту включают удобный интерфейс для создания и редактирования документов, надежное хранение и возможность поиска документов, а также интеграцию с другими системами для повышения эффективности работы пользователей.

1.2 Постановка задачи

В рамках данного проекта необходимо разработать систему документооборота, доступ к функциям которой осуществляется через графический интерфейс пользователя (UI). Основная цель проекта — создание системы, которая позволяет создавать, редактировать, просматривать и управлять документами и шаблонами документов. Система должна предоставлять возможности для создания, редактирования, просмотра и управления документами и шаблонами документов без необходимости регистрации и авторизации пользователей. Это обеспечит простоту и доступность использования системы для всех пользователей, исключая необходимость выполнения лишних действий для входа в систему и управления документами.

Среди основных функций можно выделить:

– функция добавления шаблона – позволяет пользователю создавать новый шаблон документа через UI;

– функция редактирования шаблона – позволяет пользователю изменять содержимое существующего шаблона через UI;

– функция переименования шаблона – позволяет пользователю изменять название существующего шаблона через UI;

– функция выгрузки шаблона – позволяет пользователю сохранять шаблон на локальное устройство;

– функция добавления документа – позволяет пользователю создавать новый документ на основе шаблона через UI;

– функция редактирования документа – позволяет пользователю изменять содержимое созданного документа через UI;

– функция переименования документа – позволяет пользователю изменять название существующего документа через UI;

– функция выгрузки документа – позволяет пользователю сохранять документ на локальное устройство;

– функция генерации документа по шаблону – позволяет пользователю автоматически создавать документ на основе выбранного шаблона.

Пользователь может создавать новый шаблон документа через UI, задавая его название и содержание. Это позволяет пользователям легко и быстро создавать структурированные и стандартизированные документы на основе заранее подготовленных шаблонов. Пользователь может изменять существующие шаблоны через UI, что дает возможность вносить изменения и обновления в шаблоны в соответствии с текущими требованиями и стандартами. Также пользователь может удалять существующий шаблон через UI, что позволяет поддерживать актуальность и порядок в базе шаблонов, удаляя устаревшие или ненужные шаблоны.

Пользователь может создавать новый документ на основе шаблона через UI, что упрощает процесс создания документа, так как основные элементы и структура уже заданы. Пользователь может просматривать созданные документы через UI, что позволяет удобно работать с документами, обеспечивая быстрый доступ к их содержимому. Пользователь может редактировать свои документы через UI, что дает возможность вносить изменения и корректировки в документы в любое время. Пользователь может удалять свои документы через UI, что позволяет управлять содержимым системы, удаляя ненужные или устаревшие документы. Пользователь может переименовывать свои документы через UI, что обеспечивает удобство в организации и хранении документов, позволяя давать им понятные и логичные названия.

Данные о шаблонах и документах должны надежно сохраняться в базе данных, что обеспечивает их безопасность и доступность в любое время. Система должна иметь удобный и интуитивно понятный интерфейс для управления документами и шаблонами, что повышает эффективность работы пользователей с системой, делая процесс управления документами простым и понятным. Разработав техническое задание и определив базовые необходимые функции для работы приложения с графическим интерфейсом пользователя, можно перейти к проектированию программного продукта. Основные требования к проекту включают надежное хранение данных, удобный интерфейс и функционал, удовлетворяющий потребности пользователей. Учитывая все эти аспекты, система документооборота будет полезным инструментом для эффективного управления документами и шаблонами, обеспечивая пользователям простой и удобный способ создания, редактирования и хранения документов.

2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Разработка функциональности программного средства

Для разработки программного продукта был выбран такой язык программирования как C++ и интегрированная среда разработки Clion 2024. Данная платформа позволяет создавать приложения с графическим интерфейсом на этом языке.

При анализе требований для реализации программного продукта для тестирования была построена use-case диаграмма, которая отображает взаимодействие между пользователем и приложением. На рисунке 2.1, представленном ниже отображены основные возможности, используемые в приложении от лица обычного пользователя.

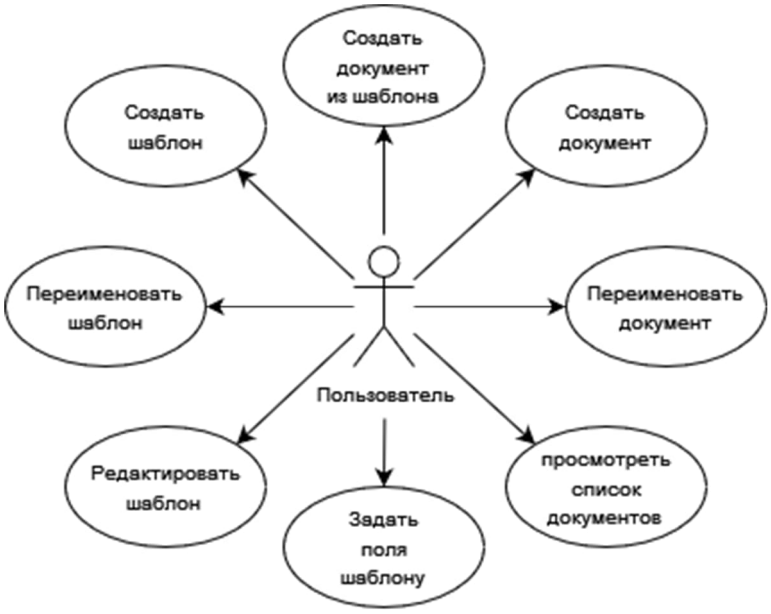

Рисунок 2.1 – Use-case диаграмма приложения

2.2 Архитектура программного средства

Для создания программного средства была выбрана многоуровневая архитектура, включающая три уровня: представления, приложения и данных.

Самый верхний уровень, уровень представления, отвечает за отображение и получение контента от пользователя. К этому слою можно получить доступ через любое клиентское устройство. В разрабатываемом приложении этот слой будет представлен библиотекой, которая обрабатывает

информацию, получаемую от пользователя. Таким образом, этот слой обеспечивает интерактивность и удобство взаимодействия для конечных пользователей, предоставляя им необходимый интерфейс для работы с приложением.

Средний уровень, уровень приложения или бизнес-логики, не знает о происхождении данных и их дальнейшем использовании. Он обеспечивает общую логику работы приложения и преобразование данных. В данном приложении этот слой будет реализован классами, обрабатывающими данные из уровня данных и передающими обработанную информацию на уровень представления. Этот уровень играет ключевую роль в обеспечении правильной работы приложения, так как здесь происходит основная обработка данных и выполнение бизнес-правил.

Нижний уровень, уровень данных, занимается сохранением данных, поступающих из уровня бизнес-логики или внешних источников, таких как файлы или базы данных. В этом программном продукте будет подключена база данных, и данный слой будет представлен классами, обеспечивающими бизнес-логику информацией и сохраняющими данные, поступающие оттуда. Этот уровень отвечает за долговременное хранение информации и её доступность для других уровней приложения.

Так как данные могут передаваться только между соседними уровнями, можно представить общую структуру взаимодействия компонентов приложения в виде диаграммы классов, показанной на рисунке 2.2. Эта структура позволяет поддерживать четкое разделение обязанностей между уровнями, обеспечивая модульность и упрощая дальнейшее обслуживание и развитие приложения.

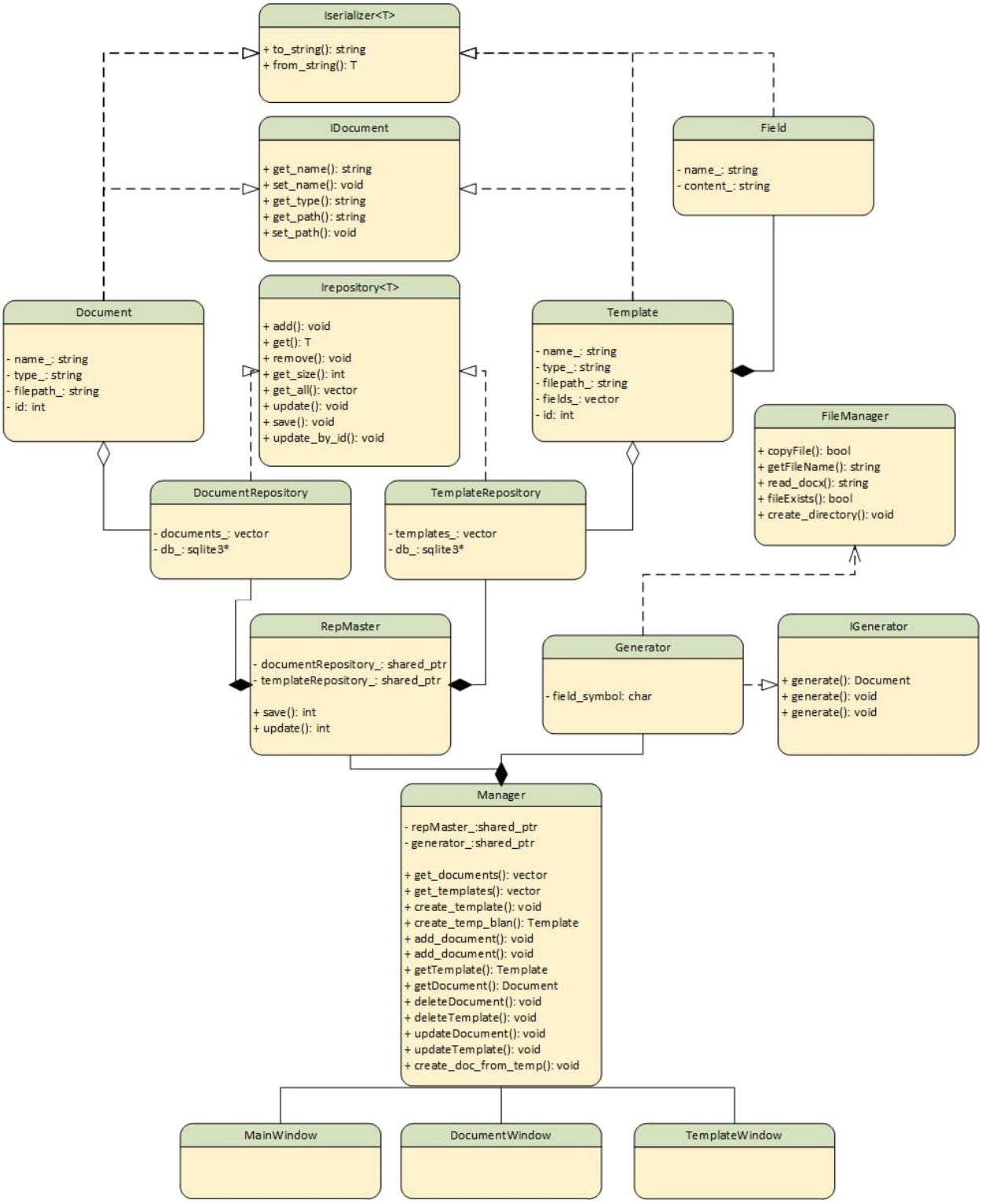

Рисунок 2.2 – Диаграмма классов приложения

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

3.1 Разработка уровня представления

В данном уровне представления реализованы три основных класса: DocumentWindow, TemplateWindow и MainWindow. Эти классы отвечают за отображение и взаимодействие с пользователем в рамках приложения, которое управляет документами и шаблонами. Рассмотрим каждый из них подробнее.

Класс MainWindow представляет главное окно приложения, представлен на рисунке 3.1 и реализует функции: обновления списка шаблонов, обновляет списка документов, обработки нажатия кнопки открытия нового документа или шаблона, обработки нажатия на элемент списка шаблонов для открытия окна шаблона, обработки нажатия на элемент списка документов для открытия окна документа.

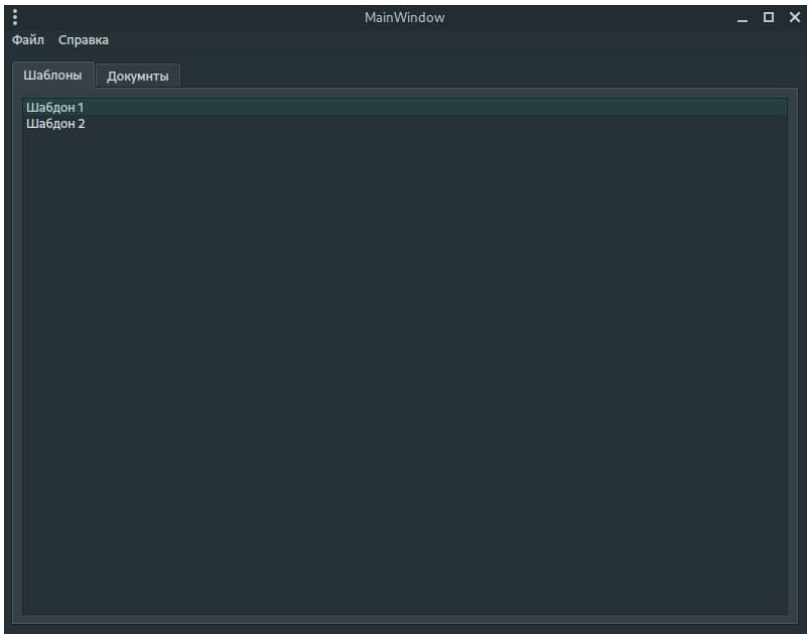

Рисунок 3.1 – Демонстрация окна Main MainWindow

Класс DocumentWindow представляет окно для работы с документами, главное окно которого представлено на рисунке 3.2. Он позволяет переименовывать и сохранять на компьютер созданные документы.

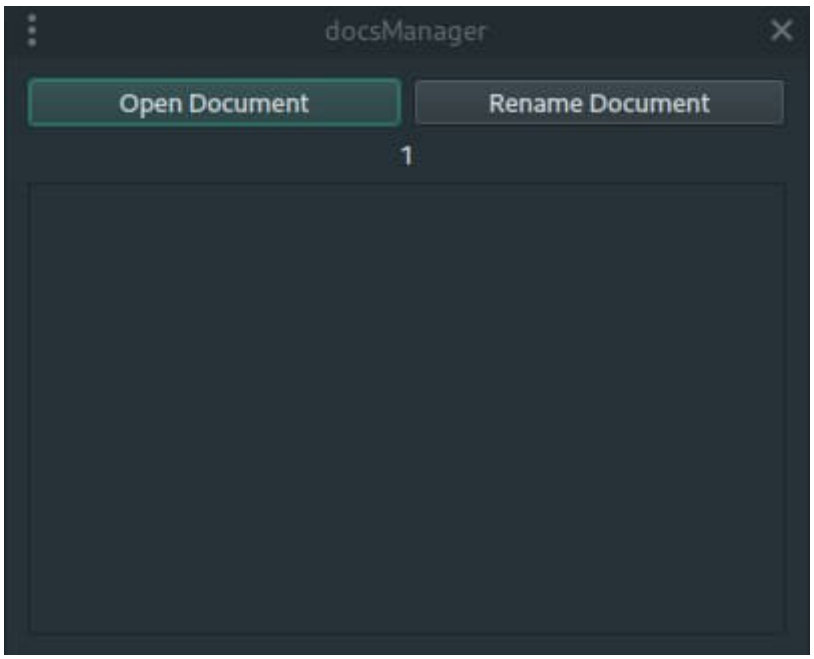

Рисунок 3.2 – Демонстрация окна DocumentWindow

Класс InfoWindow представляет окно для работы с шаблонами, прелсьавленное на рисунке 3.3. Он включает в себя функции обработки нажатия кнопок сохранение, переименования, редактирования, сохранения шаблона.

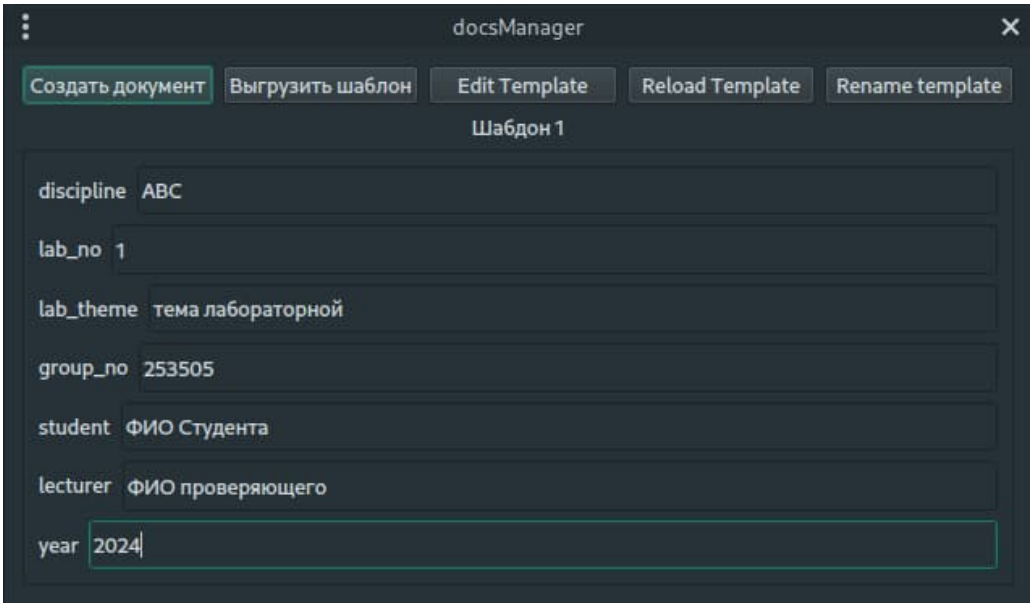

Рисунок 3.3 – Демонстрация окна TemplateWindow

3.2 Разработка уровня приложения

Уровень приложения для генерации документов реализует принципы объектно-ориентированного программирования, используя композицию для организации взаимодействия между классами.

Класс Manager использует композицию для управления генерацией документов. Он содержит два поля экземпляра класса RepMaster, который является оболочкой и агрегатором для репозиториев и экземпляр класса Generator, который наследуется от интерфейса IGenerator и отвечает за генерацию документов.

Уровень представления взаимодействует только с классом Manager, который уже управляет Generator и RepMaster. Модели Document, Template и Field представляют данные, используемые в приложении для управления документами и шаблонами. Эти модели инкапсулируют структуру и поведение данных, обеспечивая удобный интерфейс для работы с ними.

Класс Manager может возвращать вектор документов, возвращать вектор шаблонов, создавать новый шаблон, создавать новый шаблон из пустого файла, добавлять новый документ, возвращать документ по ID, удалять документ по ID и создавать новый документ из шаблона.

Класс Generator отвечает за создание шаблонов и документов, основываясь на .docx файлах. Он реализует интерфейс IGenerator, который определяет методы для генерации документов и шаблонов. Основные функции класса Generator включают: создание нового шаблона на основе .docx файла, создание нового документа на основе шаблона, заполнение полей документа данными, сохранение документа в формате .docx.

Класс RepMaster хранит в себе экземпляры классов, работающих непосредственно с базой данных. Он является оболочкой и агрегатором для репозиториев, предоставляя централизованный доступ к данным. Основные функции RepMaster включают: создание и управление экземплярами DocumentRepository и TemplateRepository, предоставление методов для добавления, обновления и удаления документов и шаблонов, выполнение запросов к базе данных через репозитории.

3.3 Разработка уровня данных

Уровень данных состоит из двух основных компонентов: репозиториев для управления документами (DocumentRepository) и шаблонами (TemplateRepository). Каждый репозиторий взаимодействует с базой данных через библиотеку SQLite, предоставляя методы для выполнения SQL-запросов.

Для того чтобы объект был сериализуемым, в приложении используется интерфейс ISerializer, который определяет методы to_string и from_string. Эти методы используются в репозиториях для записи объектов в базу данных и их последующего извлечения. Метод to_string преобразует объект в строку, которая может быть сохранена в базе данных, а метод from_string восстанавливает объект из строки, извлеченной из базы данных.

Репозиторий документов отвечает за управление данными документов. Основные функции репозитория включают: добавление нового документа, удаление документа по его идентификатору, обновление существующего документа, получение списка всех документов, получение документа по его идентификатору. Класс DocumentRepository инкапсулирует логику взаимодействия с таблицей documents в базе данных SQLite. Основные методы класса предоставляют интерфейс для выполнения CRUD-операций. Репозиторий использует библиотеку SQLite для выполнения SQL-запросов и обработки результатов.

Репозиторий шаблонов отвечает за управление данными шаблонов. Основные функции репозитория включают: добавление нового шаблона, удаление шаблона по его идентификатору, обновление существующего шаблона, получение списка всех шаблонов, получение шаблона по его идентификатору. Класс имеет аналогичную структуру репозиторию документов и инкапсулирует логику взаимодействия с таблицей templates в базе данных SQLite.

Для взаимодействия с базой данных используется объект, предоставляемый библиотекой SQLite. Репозитории используют этот объект для выполнения SQL-запросов, обеспечивая тем самым доступ к данным.

4 ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРИЛОЖЕНИЯ

Осуществлялось функциональное тестирование. Результаты проведенного тестирования приведены в таблице 4.1.

Таблица 4.1 – Тестирование программного средства

№	Тестируемая функция	Ожидаемый результат	Полученный результат
1	Добавление документа	При добавлении, документ должен отображаться в списке документов, при нажатии на него в списке открывается диалоговое окно документа	Соответствует ожидаемому
2	Добавление шаблона	При добавлении, шаблон должен отображаться в списке шаблонов, при нажатии на него в списке открывается диалоговое окно шаблона	Соответствует ожидаемому
3	Редактирование шаблона	При выборе функции редактирования шаблона, должен открыться внешний редактор с открытым шаблоном для редактирования	Соответствует ожидаемому
4	Создание документа по шаблону	При заполнении полей шаблона и нажатии кнопки создания документа, документ появляется в списке документов	Соответствует ожидаемому
5	Переименование шаблона	При выборе функции переименовании шаблона должно появиться диалоговое окно ввода нового названия и при вводе обновиться главное окно программы	Соответствует ожидаемому
6	Переименование документа	При выборе функции переименовании документа должно появиться диалоговое окно ввода нового названия и при вводе обновиться главное окно программы	Соответствует ожидаемому
7	Добавление файла с пустым именем	Появление диалогового окна об ошибке	Соответствует ожидаемому
8	Обновление шаблона	При наличии измененного файла шаблона, обновление полей и их названий на новые, соответствующие новому шаблону	Соответствует ожидаемому

Проведенное функциональное тестирование программного средства показало, что все основные функции выполняются в соответствии с ожидаемыми результатами. Результаты тестирования приведены в таблице 4.1, где для каждой протестированной функции указаны как ожидаемые, так и полученные результаты. Все тесты завершились успешно, что свидетельствует о корректной работе программного обеспечения.

Таким образом, на основании проведенных тестов можно сделать вывод о том, что программное средство прошло проверку на функциональность и готово к внедрению в рабочие процессы. Будущие пользователи могут рассчитывать на стабильную и надежную работу программы, что является важным фактором для ее эффективного применения.

ЗАКЛЮЧЕНИЕ

В результате работы над курсовым проектом было разработано программное средство для системы документооборота, которое полностью удовлетворяет условиям поставленной задачи. Разработка приложения включала несколько ключевых этапов: анализ требований и проектирование, разработка и реализация, тестирование и отладка.

На этапе анализа требований и проектирования были определены основные функциональные требования к приложению, такие как возможность добавления и редактирования шаблонов, создание документов по шаблонам, и управление документами.

Особый упор был сделан на разработку архитектуры приложения, что позволило обеспечить его гибкость и масштабируемость. Было разработано проектное решение, включая архитектуру системы и пользовательский интерфейс. На этапе разработки и реализации использованы современные технологии и инструменты, такие как язык программирования C++ и SQLite для хранения данных. Основное внимание уделялось обеспечению удобства использования и стабильности работы приложения.

На этапе тестирования и отладки проведено тестирование всех компонентов приложения, выявление и исправление ошибок. Также проводилось пользовательское тестирование для оценки удобства интерфейса и функциональности приложения.

Приложение было разработано с учетом принципов программирования, таких как ООП и SOLID. За счет этого продукт получился легко масштабируемым и открытым для дальнейшего совершенствования и использования. С помощью данного программного средства уже можно эффективно управлять документооборотом, добавлять и редактировать шаблоны, создавать документы по шаблонам. Это способствует упрощению и автоматизации рабочих процессов, что особенно важно в учебной и профессиональной деятельности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Лафоре Р. Объектно-ориентированное программирование в С++:[пер. с англ.]. – Издательский дом" Питер", 2013. – 902 с.

[2] Макс Шлее, Профессиональное программирование на С++ – М.: БХВ-Петербург, 2010. – 883 с.

[3] Мартин Р., Чистый код. Создание, анализ и рефакторинг: Петербург, 2015. – 464 с.

[4] The Qt Company. Qt Documentation [Электронный ресурс]. – Режим доступа: https://doc.qt.io, свободный. – Загл. с экрана.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код программы

```
class VectorStrings {
    public:
        std::vector<std::string> vector_strings;
        int size;

        void init() {
            size = 0;
            for (auto x: vector_strings) {
                size += x.size();
            }
        }

        char operator[](int cnt) {
            while (true) {
                if (cnt > size || cnt < 0)
                    break;
                int tmp = 0;
                int num = 0;
                while (tmp < cnt) {
                    tmp += vector_strings[num].size();
                    num++;
                    if (tmp >= cnt) {
                        int k = vector_strings[num - 1].size() - 1;
                        while (tmp != cnt) {
                            k--;
                            tmp--;
                        }
                        return vector_strings[num - 1][k];
                    }
                }
                return 0;
            }
            throw std::invalid_argument("Index is out of range🙂");
        }

        void operator()(int cnt) {
            while (true) {
                if (cnt > size || cnt < 0)
                    break;
                int tmp = 0;
                int num = 0;
                while (tmp < cnt) {
                    tmp += vector_strings[num].size();
                    num++;
                    if (tmp >= cnt) {
                        int k = vector_strings[num - 1].size() - 1;
                        while (tmp != cnt) {
                            k--;
                            tmp--;
                        }
                        vector_strings[num - 1].erase(k, 1);
                        --size;
```

```
                    }
                }
                return;
            }
            throw std::invalid_argument("Index is out of range🙂");
        }

        void operator()(int cnt, std::string str) {
            while (true) {
                if (cnt > size || cnt < 0)
                    break;

                int tmp = 0;
                int num = 0;
                while (tmp < cnt) {
                    tmp += vector_strings[num].size();
                    num++;
                    if (tmp >= cnt) {
                        int k = vector_strings[num - 1].size() - 1;
                        while (tmp != cnt) {
                            k--;
                            tmp--;
                        }
                        vector_strings[num - 1].insert(k, str);
                        size += str.size();
                    }
                }
                return;
            }
            throw std::invalid_argument("Index is out of range🙂");

        }
    };

Document Generator::generate(const Template &aTemplate, const std::string
&name) {
    while (true) {
        FileManager::copyFile(aTemplate.get_path(), "./documents/" + name +
".docx");
        auto document = Document(name, "./documents/" + name + ".docx");
        duckx::Document doc(document.get_path());
        doc.open();
        int k = 0;
        for (auto p = doc.paragraphs(); p.has_next(); p.next()) {
            VectorStrings vectorStrings;
            for (auto r = p.runs(); r.has_next(); r.next()) {
                vectorStrings.vector_strings.emplace_back(r.get_text());
            }
            vectorStrings.init();
            for (int i = 0; i < vectorStrings.size - 1; i++) {
                if (vectorStrings[i] == field_symbol && vectorStrings[i + 1] ==
field_symbol) {

                    vectorStrings(i);
                    vectorStrings(i);
                    auto x = aTemplate.fields_[k++].get_content();
                    vectorStrings(i, x);
                    i += x.size();
```

```
                    for (int j = i; j < vectorStrings.size; j++) {
                        if (vectorStrings[j] != field_symbol || vectorStrings[j
+ 1] != field_symbol) {
                            vectorStrings(j--);
                        } else {
                            vectorStrings(j);
                            vectorStrings(j);
                            break;
                        }
                    }
                }
            }
            int i = 0;
            for (auto r = p.runs(); r.has_next(); r.next()) {
                r.set_text(vectorStrings.vector_strings[i++].c_str());
            }
        }
        doc.save();
        return document;
    }

}


void Generator::generate(Template &aTemplate) {
    int flag = 0;
    while (true) {
        if (!FileManager::fileExists(aTemplate.get_path())) {
            flag = 1;
            break;
        }
        duckx::Document doc(aTemplate.get_path());
        try {
            doc.open();
            for (auto p = doc.paragraphs(); p.has_next(); p.next()) {

                VectorStrings vectorStrings;
                for (auto r = p.runs(); r.has_next(); r.next()) {
                    vectorStrings.vector_strings.emplace_back(r.get_text());
                }
                vectorStrings.init();
                for (int i = 0; i < vectorStrings.size - 1; i++) {
                    if (vectorStrings[i] == field_symbol && vectorStrings[i +
1] == field_symbol) {
                        std::string tmp;
                        for (int j = i + 2; j < vectorStrings.size - 1; j++) {
                            if (vectorStrings[j] != field_symbol || vector-
Strings[j + 1] != field_symbol)
                                tmp += vectorStrings[j];
                            else {
                                i = j + 2;
                                break;
                            }
                        }
                        aTemplate.fields_.emplace_back(tmp);
                    }
                }
            }
```

```
        } catch (...) {
            flag = 2;
            break;
        }
        return;
    }
    switch (flag) {
        case 1:
            throw std::invalid_argument("Invalid template: no such file in di-
rectory🙂");
        case 2:
            throw std::invalid_argument("Invalid template: error while pro-
cessing file🙂");
        default:
            throw std::invalid_argument("Unhandled exception");
    }

}

Manager::Manager() {
    repMaster_ = std::make_shared<RepMaster>();
    generator_ = std::make_shared<Generator>();
    folder_name_ = "documents";
    FileManager::create_directory("documents");
    FileManager::create_directory("templates");
    FileManager::create_directory("data");
    repMaster_->update();
}


void Manager::create_template(std::string path, std::string name) {
    for (auto x: repMaster_->templateRepository_->get_all()) {
        if (x.get_name() == name)
            throw new std::invalid_argument("There is already exists template
with this name🙂");
    }
    std::string filePath = "./templates/" + name + ".docx";
    FileManager::copyFile(path, filePath);
    auto temp = Template(std::move(name), filePath);
    generator_->generate(temp);
    repMaster_->templateRepository_->add(temp);
}

void Manager::create_document_from_template(Template temp, std::string name) {
    auto doc = generator_->generate(temp, name);
    repMaster_->documentRepository_->add(doc);
}

void Manager::add_document(int template_id, std::string name) {
    for (auto x: repMaster_->documentRepository_->get_all()) {
        if (x.get_name() == name)
            throw std::invalid_argument("There is already exists document with
this name🙂");
    }
    auto const &templ = repMaster_->templateRepository_->get(template_id);
    std::string    filePath    =    "./templates/"    +    FileManager::getFile-
Name(templ.get_path());
```

```
    FileManager::copyFile(templ.get_path(), filePath);
    generator_->generate(templ, name);
}

void Manager::add_document(std::string filepath, std::string name) {
    for (auto x: repMaster_->documentRepository_->get_all()) {
        if (x.get_name() == name)
            throw std::invalid_argument("There is already exists document with
this name🙂");
    }
    std::string filePath = "./documents/" + FileManager::getFileName(filepath);
    FileManager::copyFile(filepath, filePath);
    auto doc = Document(name, filePath);
    repMaster_->documentRepository_->add(doc);
}

std::vector<Document> Manager::get_documents() {
    return repMaster_->documentRepository_->get_all();
}

std::vector<Template> Manager::get_templates() {

    return repMaster_->templateRepository_->get_all();
}

Template Manager::getTemplate(int id) {
    return repMaster_->templateRepository_->get(id);
}

Document Manager::getDocument(int id) {
    return repMaster_->documentRepository_->get(id);
}

void Manager::updateDocument(int id, Document document) {
    repMaster_->documentRepository_->update_by_id(id, document);
}

void Manager::updateTemplate(int id, Template temp) {
    repMaster_->templateRepository_->update_by_id(id, temp);
}

void Manager::create_template_form_blank(std::string name) {
    for (auto x: repMaster_->templateRepository_->get_all()) {
        if (x.get_name() == name)
            throw std::invalid_argument("There is already exists template with
this name🙂");
    }

    std::string filePath = "./data/blank.docx";
    std::string filepath = "./template/" + name + ".docx";
    FileManager::copyFile(filePath, filepath);
    auto temp = Template(std::move(name), filePath);
    generator_->generate(temp);
    repMaster_->templateRepository_->add(temp);
}

void Manager::deleteDocument(int id) {
    repMaster_->documentRepository_->remove(id);
```

```
}

void Manager::deleteTemplate(int id) {
    repMaster_->templateRepository_->remove(id);
}
int RepMaster::save() {

    documentRepository_->save();

    templateRepository_->save();

    return 0;

}


int RepMaster::update() {

    documentRepository_->update();

    templateRepository_->update();

    return 0;

}


RepMaster::RepMaster() {

    db_ = nullptr;

    if (sqlite3_open("docsmanager.db", &db_)) {

        throw std::runtime_error("Can't open database: " +
std::string(sqlite3_errmsg(db_)));

    }


    templateRepository_ = std::make_shared<TemplateRepository>(db_);

    documentRepository_ = std::make_shared<DocumentRepository>(db_);

}
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Схемы алгоритмов программного средства

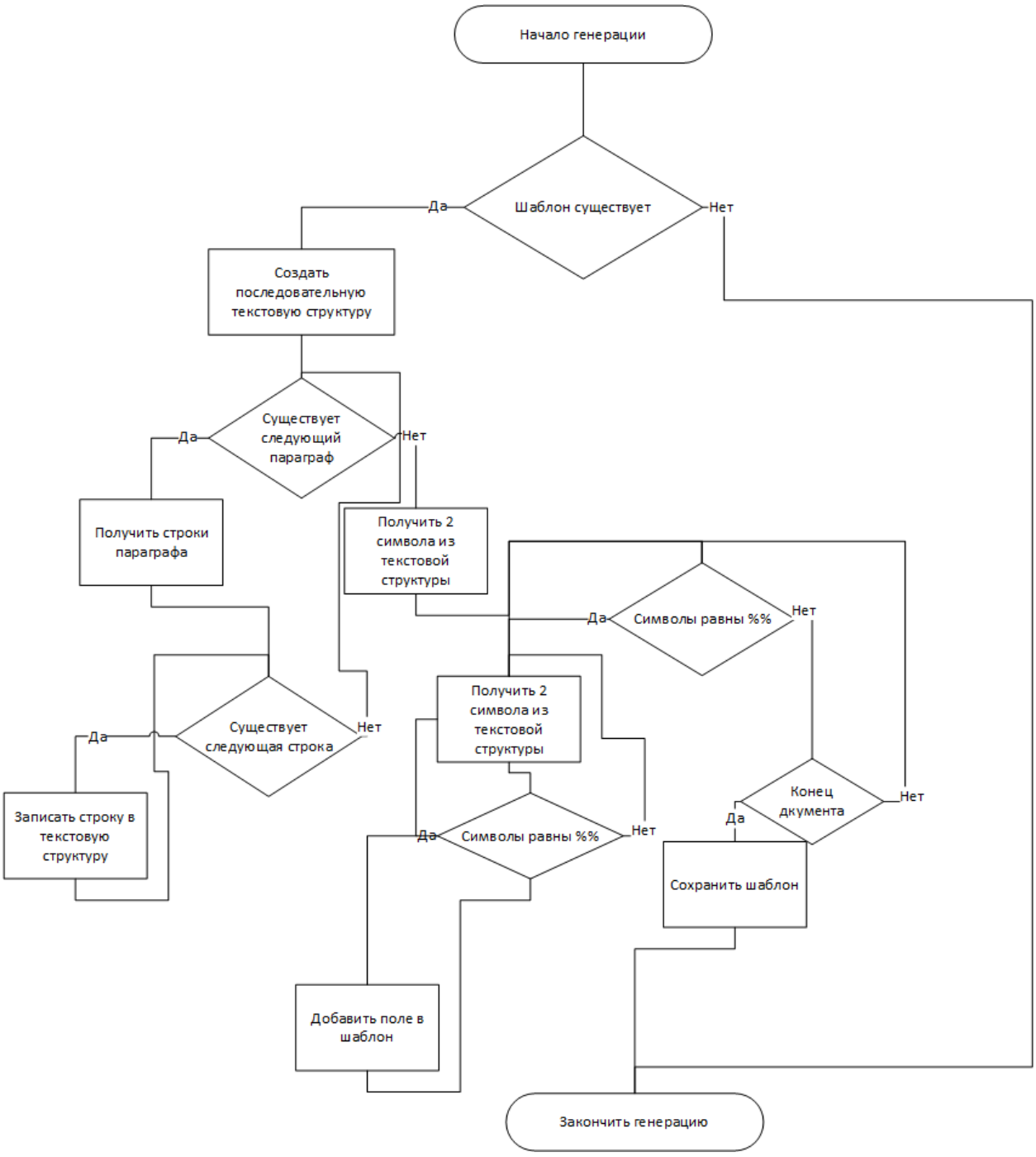

Рисунок Б.1 – Блок-схема функции генерации шаблона

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КП 1-40 04 01	Пояснительная записка	27 с.
	Графические документы	
ГУИР 253505 014 СА	Программное средство для документооборота. Схемы алгоритмов	Формат А4

					БГУИР КП 1-40 04 01			
Изм.	Л.	№ докум.	Подп.	Дата	Программное средство для документооборота		Лист	Листов
Разраб.		*Сенько Н. С.*		04.06.24	Ведомость курсового проекта	Т	27	27
Пров.		*Тушинская Е.В.*		04.06.24		Кафедра Информатики гр. 253505		