**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 2](#_Toc197354979)

[1 Обзор литературы 3](#_Toc197354980)

[1.1 Анализатор и редактор файловой системы 3](#_Toc197354981)

[1.2 Анализ существующих аналогов 3](#_Toc197354982)

[1.3 Анализ использованных средств разработки 4](#_Toc197354983)

[1.4 Постановка задачи 5](#_Toc197354984)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 7](#_Toc197354985)

[2.1 Интерфейс командной строки (CLI) 7](#_Toc197354986)

[2.2 Менеджер файловой системы 7](#_Toc197354987)

[2.3 Модуль просмотра файлов 7](#_Toc197354988)

[2.4 Модуль редактирования файлов 7](#_Toc197354989)

[2.5 Модуль поиска файлов 7](#_Toc197354990)

[3 функциональное проектирование 9](#_Toc197354991)

[3.1 Описание ключевых функций 9](#_Toc197354992)

[2.2 Описание ключевых функций интерфейса 10](#_Toc197354993)

[4 разработка программных модулей 12](#_Toc197354994)

[4.1 Описание программных модулей 12](#_Toc197354995)

[4.2 Примеры кода 13](#_Toc197354996)

[ЗАКЛЮЧЕНИЕ 15](#_Toc197354997)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 16](#_Toc197354998)

[Приложение А 17](#_Toc197354999)

[Приложение Б 18](#_Toc197355000)

[Приложение В 19](#_Toc197355001)

[Приложение Г 20](#_Toc197355002)

ВВЕДЕНИЕ

В условиях стремительного развития технологий и увеличения объема информации, управление файловыми системами становится ключевым аспектом для пользователей и разработчиков. Эффективный анализ и редактирование файловых систем требует создания инструментов, которые обеспечивают удобный и интуитивно понятный интерфейс для взаимодействия с данными. Проект «Анализатор и редактор файловой системы» на основе библиотеки ncurses направлен на удовлетворение этих потребностей, предлагая пользователям мощные функции для работы с файловыми системами в текстовом режиме.

В современных условиях существует множество приложений и утилит для управления файлами, однако многие из них имеют ограниченный функционал или сложны в использовании. Программа, разрабатываемая в рамках данного проекта, будет включать в себя такие возможности, как навигация по файловой системе, просмотр и редактирование файлов, а также выполнение различных операций с данными. Основное внимание будет уделено созданию пользовательского интерфейса, который обеспечит простоту и удобство работы, а также поддержит работу в терминале, что является важным для пользователей, предпочитающих текстовые интерфейсы.

Анализ существующих решений показывает, что потребность в эффективных инструментах для работы с файлами остается актуальной. Наша разработка имеет целью улучшение взаимодействия пользователей с файловыми системами, что, в свою очередь, позволит повысить продуктивность и упростить выполнение рутинных задач. В результате, приложение станет полезным инструментом как для новичков, так и для опытных пользователей, стремящихся повысить свою эффективность в работе с данными.

1 Обзор литературы

* 1. Анализатор и редактор файловой системы

Основной целью проекта является разработка интуитивно понятного и надежного инструмента, который позволит пользователям эффективно анализировать и редактировать файловые системы. Система будет включать в себя различные модули, обеспечивающие комплексный подход к управлению файлами и данными.

Архитектура системы:

‒ Клиентская часть: включает приложение, предоставляющее пользователю доступ к функционалу анализа и редактирования файлов. Интерфейс будет реализован на основе библиотеки ncurses, что позволит создать удобный консольный интерфейс для навигации и выполнения операций с файлами.

‒ Пользовательский интерфейс: разрабатывается с акцентом на простоту и удобство, что обеспечит легкость взаимодействия пользователя с системой. Интерфейс будет включать элементы навигации, такие как дерево каталогов и меню, позволяющие быстро находить и управлять нужными файлами.

Таким образом, проект направлен на создание мощного инструмента, который будет полезен как для обычных пользователей, так и для специалистов, работающих с большими объемами данных.

* 1. Анализ существующих аналогов

Среди аналогов приложения для учёта личных финансов в основном выступают мобильные приложения. Рассмотрим несколько популярных вариантов.

* + 1. Midnight Commander

Midnight Commander — это классический файловый менеджер для консоли, который предоставляет пользователям возможность управлять файлами и каталогами с использованием текстового интерфейса. Он включает такие функции, как копирование, перемещение и удаление файлов, а также поддержку работы с архивами и FTP.

Набор функций приложения:

‒ Двухпанельный интерфейс для удобной навигации.

‒ Поиск файлов и каталогов по имени.

‒ Поддержка работы с различными файловыми системами.

‒ Возможность редактирования файлов с помощью встроенного текстового редактора.

Несмотря на свои возможности, Midnight Commander имеет ограниченные возможности по кастомизации интерфейса и не предлагает расширенной аналитики работы с файлами. Тем не менее, многие пользователи отмечают его простоту и надежность в повседневной работе.

1.2.2 Ranger

Ranger — это файловый менеджер, который акцентирует внимание на клавиатурной навигации и минимализме. Он использует трехпанельный интерфейс, что позволяет пользователям быстро перемещаться между каталогами и просматривать содержимое файлов.

Ключевые функции:

‒ Поддержка предварительного просмотра файлов в различных форматах.

‒ Возможность работы с плагинами для расширения функционала.

‒ Интуитивно понятные горячие клавиши для быстрого доступа к функциям.

‒ Поддержка работы с несколькими файлами одновременно.

Однако Ranger может быть сложен для пользователей, не привыкших к работе в командной строке. Он требует определенной подготовки, чтобы максимально использовать все его возможности.

* + 1. nnn

nnn — это минималистичный файловый менеджер, который фокусируется на высокой производительности и простоте использования. Он занимает минимум ресурсов и отлично подходит для работы на системах с ограниченными возможностями.

Функционал включает:

‒ Быстрый доступ к файлам и каталогам.

‒ Возможность запуска приложений и скриптов непосредственно из интерфейса.

‒ Интеграция с различными утилитами для работы с файлами.

‒ Поддержка плагинов для расширения возможностей.

Несмотря на свою простоту, nnn может не предоставлять всех необходимых функций для глубокого анализа файловой системы, что может ограничить его применение для более сложных задач.

* 1. Анализ использованных средств разработки

Для разработки данного проекта был выбран язык программирования C в сочетании с библиотекой ncurses. Эта библиотека предоставляет мощные инструменты для создания текстовых интерфейсов в консольных приложениях, что идеально подходит для реализации анализа и редактирования файловых систем.

Библиотека ncurses включает в себя следующие ключевые компоненты:

‒ Управление окнами – позволяет разбивать интерфейс на несколько окон для удобства отображения информации.

‒ Обработка ввода – обеспечивает удобный способ работы с клавиатурным вводом, включая поддержку функциональных клавиш и сочетаний клавиш.

‒ Цветовая палитра – позволяет настраивать цветовые схемы, улучшая визуальное восприятие интерфейса.

‒ Отображение текста – предоставляет функции для форматирования и вывода текста, включая поддержку многострочных сообщений и выравнивания.

Использование языка C в сочетании с библиотекой ncurses предоставляет несколько значительных преимуществ. Во-первых, C обеспечивает высокую производительность и эффективное использование ресурсов, что критично для приложений, работающих с файловыми системами. Библиотека ncurses, созданная для работы в текстовом режиме, позволяет разрабатывать интерфейс, который будет интуитивно понятен и удобен для пользователей.

Кроме того, структура C позволяет создавать модульные и легко расширяемые системы, что идеально соответствует требованиям проекта. Модульность кода позволяет добавлять новые функции и улучшения без значительных затрат времени и усилий.

Библиотека ncurses также обладает кроссплатформенными возможностями, что позволяет запускать приложение на различных операционных системах, таких как Linux и macOS, без необходимости в значительных изменениях в коде. Это делает проект более доступным для широкой аудитории пользователей, что является важным аспектом его разработки.

* 1. Постановка задачи

Опираясь на поставленные цели и требования к разработке анализатора и редактора файловой системы, формируется список задач, необходимых для реализации данного проекта. Итак, консольное приложение обязуется выполнять следующие функции:

‒ Навигация по файловой системе: пользователи могут перемещаться между каталогами, просматривать их содержимое и возвращаться к предыдущим уровням.

‒ Просмотр информации о файлах: для каждого файла отображается информация, такая как размер, дата последнего изменения и тип файла.

‒ Редактирование текстовых файлов: приложение предоставляет встроенный редактор, позволяющий пользователям открывать и редактировать текстовые файлы без необходимости покидать интерфейс.

‒ Управление файлами: пользователи могут выполнять основные операции, такие как копирование, перемещение и удаление файлов и каталогов, что упрощает управление содержимым файловой системы.

‒ Поиск файлов: встроенная функция поиска позволяет быстро находить файлы по имени, что делает работу с большими объемами данных более эффективной.

‒ Создание и удаление каталогов: пользователи могут создавать новые каталоги и удалять существующие, что позволяет организовывать файловую структуру по своему усмотрению.

‒ Работа с архивами: программа может поддерживать работу с архивами, позволяя пользователям извлекать и добавлять файлы в архивы без выхода из приложения.

‒ Настройка интерфейса: возможность изменения цветовых схем и настройки интерфейса для улучшения восприятия информации пользователями.

Таким образом, проект «Анализатор и редактор файловой системы» направлен на создание мощного и удобного инструмента, который значительно упростит работу с файлами и каталогами, обеспечивая пользователям все необходимые функции для эффективного управления данными.

1. СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Системное проектирование консольного анализатора и редактора файловой системы направлено на создание функционального и интуитивно понятного приложения для работы с файловой системой из командной строки. Основной акцент сделан на реализации базовых операций с файлами и каталогами. Ниже представлена структурная схема и описание основных модулей.

* 1. Интерфейс командной строки (CLI)

Модуль интерфейса командной строки (CLI) отвечает за взаимодействие с пользователем, принимая команды и отображая результаты их выполнения. Он анализирует введенные пользователем команды и передает запросы соответствующим модулям для обработки. CLI обеспечивает удобный текстовый интерфейс с возможностью навигации по файловой системе.

2.2 Менеджер файловой системы

Менеджер файловой системы отвечает за выполнение операций с файлами и каталогами, таких как копирование, перемещение, удаление и создание. Он получает команды от CLI и использует системные вызовы для взаимодействия с файловой системой. Менеджер также обрабатывает ошибки и предоставляет пользователю информацию о статусе операций.

2.3 Модуль просмотра файлов

Модуль просмотра файлов реализует функциональность для отображения информации о файлах и каталогах. Он получает запросы от CLI и извлекает данные, такие как размер, дата последнего изменения и тип файла. Этот модуль также отвечает за форматирование вывода информации для удобства восприятия пользователем.

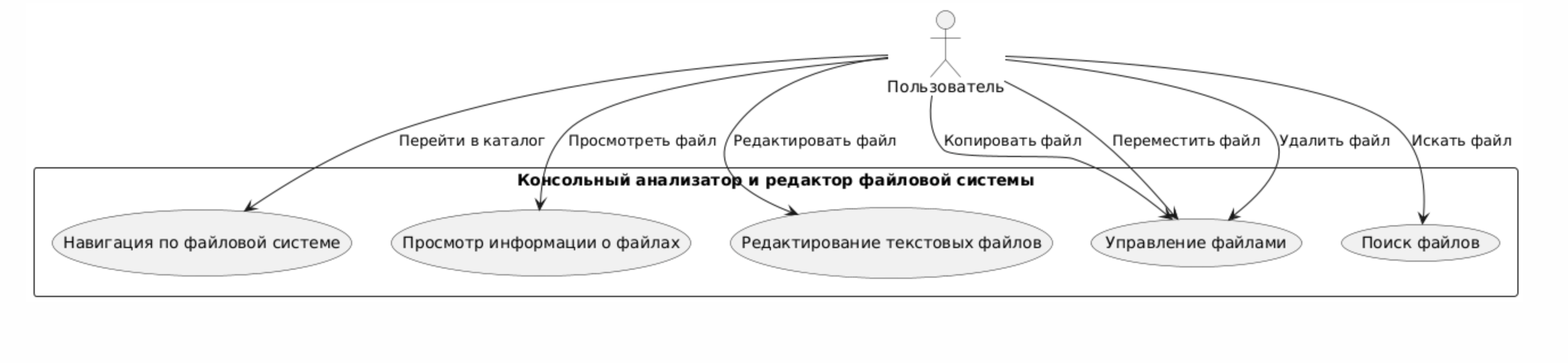
2.4 Модуль редактирования файлов

Модуль редактирования файлов предоставляет возможность пользователям открывать и редактировать текстовые файлы. Он включает встроенный текстовый редактор, который позволяет вносить изменения без выхода из интерфейса. Модуль отвечает за сохранение изменений и управление состоянием файла во время редактирования.

2.5 Модуль поиска файлов

Модуль поиска файлов отвечает за выполнение запросов на поиск файлов по имени. Он принимает параметры поиска от CLI и осуществляет поиск в заданных каталогах, возвращая результаты пользователю. Этот модуль обеспечивает эффективный доступ к данным, что особенно полезно при работе с большими объемами информации.

В заключение, данная архитектура обеспечивает простое и функциональное консольное приложение для анализа и редактирования файловой системы. Каждый модуль выполняет определенную функцию и взаимодействует с другими модулями для достижения общей цели.



3 функциональное проектирование

Проект представляет собой консольное приложение на языке C, использующее библиотеку ncurses для анализа и редактирования файловой системы. Программа предоставляет пользователям удобный текстовый интерфейс для навигации по файловым каталогам и позволяет просматривать структуру файлов и папок в реальном времени. Основные функции приложения включают в себя:

‒ Навигация по файловой системе: пользователи могут перемещаться между каталогами, просматривать их содержимое и возвращаться к предыдущим уровням.

‒ Просмотр информации о файлах: для каждого файла отображается информация, такая как размер, дата последнего изменения и тип файла.

‒ Редактирование текстовых файлов: приложение предоставляет встроенный редактор, позволяющий пользователям открывать и редактировать текстовые файлы без необходимости покидать интерфейс.

‒ Управление файлами: пользователи могут выполнять основные операции, такие как копирование, перемещение и удаление файлов и каталогов.

‒ Поиск файлов: встроенная функция поиска позволяет быстро находить файлы по имени, что делает работу с большими объемами данных более эффективной.

Для более детального понимания архитектуры проекта используется блок-схема (Приложение А), которая визуализирует структуру и принцип работы приложения.

3.1 Описание ключевых функций

2.1.1 Навигация по файловой системе

Эта функция отвечает за просмотр и перемещение по структуре каталогов. Пользователь может просматривать содержимое текущего каталога и переходить на уровень вверх или вниз в иерархии файлов. Система предоставляет возможность выбирать файлы и каталоги для дальнейшего взаимодействия.

Особенности реализации:

Используется функция list\_directory, которая отображает содержимое каталога, включая навигационные команды для пользователя.

Реализован механизм обработки ввода с использованием библиотеки ncurses, что обеспечивает плавный интерфейс.

Пользователь может возвращаться на уровень выше или выбирать файлы для просмотра их информации.

2.1.2 Просмотр информации о файлах

Эта функция позволяет пользователю получать информацию о каждом файле, включая его размер, дату последнего изменения и тип. Это важный аспект, помогающий пользователю оценить состояние файловой системы.

Особенности реализации:

Для отображения информации используется функция display\_file\_info, которая извлекает данные о файле с помощью системных вызовов.

Все данные выводятся на экран в удобном формате, что позволяет быстро ориентироваться в информации.

2.1.3 Редактирование текстовых файлов

Эта функция предоставляет возможность открывать и редактировать текстовые файлы непосредственно в приложении. Это значительно упрощает процесс работы с файлами и позволяет избежать выхода из интерфейса.

Особенности реализации:

Встроенный редактор позволяет пользователю редактировать текст, используя стандартные команды редактирования.

Функции для сохранения изменений интегрированы в основной поток программы, что упрощает процесс редактирования.

2.1.4 Поиск файлов

Эта функция предоставляет пользователю возможность быстро находить файлы по имени, что особенно полезно при работе с большими каталогами.

Особенности реализации:

Реализован механизм поиска с помощью функции search\_file, которая принимает имя файла для поиска и выводит результаты на экран.

Пользователь может вводить часть имени файла, и система будет отображать все соответствующие результаты.

* 1. Описание ключевых функций интерфейса

2.2.1 Интерфейс приложения

Главное окно приложения служит центральным элементом интерфейса, позволяющим пользователю взаимодействовать с файловой системой через консольный интерфейс. Оно включает в себя следующие элементы:

‒ Обновление содержимого каталога: при перемещении между каталогами информация о текущем содержимом обновляется автоматически.

Навигация и управление записями: Пользователь может добавлять, редактировать или удалять файлы и каталоги, используя текстовые команды.

‒ Отображение структуры файлов: приложение выводит структуру каталогов и информацию о файлах в текстовом формате, что облегчает восприятие.

‒ Функции поиска: пользователь может выполнять поиск файлов по имени, что упрощает работу с большими объемами данных.

Особенности реализации:

Приложение использует библиотеку ncurses для создания текстового интерфейса, что позволяет эффективно управлять вводом и выводом данных.

Поддержка клавиатурного ввода для навигации и взаимодействия с файлами делает консольный интерфейс удобным и интуитивно понятным.

4 разработка программных модулей

4.1 Описание программных модулей

1 Функция: list\_directory

Функция list\_directory отвечает за отображение содержимого указанного каталога. Она считывает файлы и подкаталоги, фильтрует скрытые элементы и выводит их на экран с возможностью навигации по страницам. Пользователь может выбрать файл или перейти в подкаталог.

Параметры:

‒ const char \*path: Путь к каталогу, содержимое которого нужно отобразить.

Переменные:

‒ DIR \*dir: Указатель на структуру каталога, используемый для открытия и чтения содержимого указанного каталога.

‒ struct dirent \*entry: Указатель на структуру, представляющую запись о файле или каталоге.

‒ char \*files[1024]: Массив указателей на строки для хранения имен файлов и каталогов.

‒ char next\_path[1024]: Буфер для хранения полного пути к выбранному файлу или подкаталогу.

‒ int count: Счетчик для отслеживания количества найденных файлов и каталогов.

2 Функция: display\_file\_info

Функция display\_file\_info отображает информацию о выбранном файле, включая его размер, дату последнего изменения и тип. Это помогает пользователю понять характеристики файла.

Параметры:

‒ const char \*file\_path: Путь к файлу, информацию о котором нужно отобразить.

Переменные:

‒ struct stat file\_info: Структура, используемая для хранения информации о файле.

‒ const char \*file\_path: Параметр функции, представляющий путь к файлу.

3 Функция: search\_file

Функция search\_file реализует механизм поиска файлов в указанном каталоге по имени. Она позволяет пользователю вводить поисковый запрос и отображает найденные файлы.

Параметры:

‒ const char \*path: Путь к каталогу, в котором будет выполняться поиск.

Переменные:

‒ char search\_term[256]: Массив символов для хранения поискового запроса.

‒ DIR \*dir: Указатель на структуру каталога для открытия каталога.

‒ struct dirent \*entry: Указатель на структуру для перебора найденных файлов.

‒ int found: Логическая переменная, указывающая, были ли найдены файлы.

4 Главная функция: main

Главная функция main инициализирует консольное приложение и запускает основной цикл навигации по файловой системе. Она обрабатывает пользовательский ввод и вызывает соответствующие функции для выполнения действий.

Переменные:

‒ char current\_path[1024]: Массив символов для хранения текущего пути к каталогу.

4.2 Примеры кода

4.2.1 Функция list\_directory

void list\_directory(const char \*path) {

// Открытие каталога и чтение его содержимого

DIR \*dir = opendir(path);

// Проверка на ошибки

if (!dir) {

printw("Ошибка при открытии каталога: %s\n", path);

refresh();

return;

}

// Считывание и отображение файлов

// ...

}

4.2.2 Функция display\_file\_info

void display\_file\_info(const char \*file\_path) {

// Извлечение информации о файле

struct stat file\_info;

if (stat(file\_path, &file\_info) == 0) {

clear();

printw("Информация о файле: %s\n", file\_path);

// Вывод информации

// ...

}

}

4.2.3 Функция search\_file

void search\_file(const char \*path) {

// Ввод поискового запроса

char search\_term[256];

printw("Введите имя файла для поиска: ");

// ...

// Поиск и отображение результатов

}

4.2.4 Главная функция main

int main(void) {

setlocale(LC\_ALL, "");

initscr();

cbreak(); // Включаем режим cbreak

noecho(); // Отключаем вывод вводимых символов

keypad(stdscr, TRUE); // Включаем поддержку дополнительных клавиш

char current\_path[1024];

getcwd(current\_path, sizeof(current\_path));

while (1) {

list\_directory(current\_path);

char ch = getch(); // Ожидание ввода

if (ch == 'q') {

break; // Выход

}

else if (ch == 'b') {

char \*last\_slash = strrchr(current\_path, '/');

if (last\_slash != NULL && last\_slash != current\_path) {

\*last\_slash = '\0'; // Возврат на уровень вверх

}

}

else if (ch == 'f') {

search\_file(current\_path);

}

}

endwin(); // Завершение работы ncurses

return 0;

}

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Habr [Электронный ресурс]. – Режим доступа: https://habr.com. – Дата доступа: 12.10.2024.

[2] Stackoverflow [Электронный ресурс]. – Режим доступа: https://stackoverflow.com/questions/. – Дата доступа: 10.11.2023.

[3] Qt Documentation [Электронный ресурс]. – Режим доступа: https://doc.qt.io. – Дата доступа: 09.10.2024.

[4] Sqlite3 Documentation [Электронный ресурс]. – Режим доступа: https://www.sqlite.org/docs.html. – Дата доступа: 25.09.2024.

[5] Конструирование программ и языки программирования: метод. Указания по К65 курсовому проектированию “Вычислительные машины, системы и сети” для всех форм обуч. / сост. А. В. Бушкевич , А. М. Ковальчук , И. В. Лукьянова. - Минск: БГУИР , 2010- 30с.

[6] Луцик, Ю. А. Основы алгоритмизации и программирования : язык Си : учебно-метод. пособие / Ю. А. Луцик, А. М. Ковальчук, Е. А. Сасин. – Минск : БГУИР, 2015. – 170с. : ил.

Приложение А

Диаграмма классов

Приложение Б

Схема алгоритма addIncome

Приложение В

Схема алгоритмма addDebt

Приложение Г

Листинг кода

Файл main.cpp

#include <QApplication>

#include "MainWindow.h"

#include "WelcomeWindow.h"

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

QApplication app(argc, argv);

WelcomeWindow welcomeWindow;

QObject::connect(&welcomeWindow, &WelcomeWindow::start, [&]() {

MainWindow \*mainWindow = new MainWindow();

mainWindow->show();

});

if (welcomeWindow.exec() == QDialog::Accepted) {

return app.exec();

}

return 0;

}

Файл ActionManager.h

#ifndef ACTIONMANAGER\_H

#define ACTIONMANAGER\_H

#include "FinanceService.h"

#include "Stack.h"

#include "CustomException.h"

#include <string>

#include <memory>

struct Action {

enum class Type { AddIncome, AddExpense, DeleteRecord };

Type type;

std::string description;

double amount;

int recordId;

};

class ActionManager {

public:

explicit ActionManager(FinanceService &service) : financeService(service) {}

void addIncome(const std::string &description, double amount) {

int recordId = financeService.createIncome(description, amount);

actionStack.push(Action{Action::Type::AddIncome, description, amount, recordId});

}

void addExpense(const std::string &description, double amount) {

int recordId = financeService.createExpense(description, amount);

actionStack.push(Action{Action::Type::AddExpense, description, amount, recordId});

}

void deleteRecord(int id) {

auto record = financeService.getRecordById(id);

actionStack.push(Action{Action::Type::DeleteRecord, record->getDescription(), record->getAmount(), id});

financeService.deleteRecord(id);

}

void undo() {

if (actionStack.isEmpty()) {

throw CustomException("Нет действий для отмены.");

}

Action lastAction = actionStack.top();

actionStack.pop();

using enum Action::Type;

switch (lastAction.type) {

case AddIncome:

case AddExpense:

financeService.deleteRecord(lastAction.recordId);

break;

case DeleteRecord:

financeService.createExpense(lastAction.description, lastAction.amount);

break;

}

}

private:

FinanceService &financeService;

Stack<Action> actionStack;

};

#endif

Файл Budget.h

#ifndef BUDGET\_H

#define BUDGET\_H

#include "IncomeRecord.h"

#include "ExpenseRecord.h"

#include <vector>

#include <sqlite3.h>

#include <iostream>

#include <format>

class Budget

{

private:

double targetAmount;

double totalIncome {0.0};

double totalExpenses {0.0};

std::vector<IncomeRecord> incomeRecords;

std::vector<ExpenseRecord> expenseRecords;

public:

Budget();

explicit Budget(double target);

void addIncome(const std::string &description, double amount);

void addExpense(const std::string &description, double amount);

double getTotalIncome() const;

double getTotalExpenses() const;

double getTargetAmount() const;

double getRemainingBudget() const;

void displayBudgetSummary() const;

void displayIncomeRecords() const;

void displayExpenseRecords() const;

void saveBudgetToDatabase(sqlite3 \*db) const;

void loadBudgetFromDatabase(sqlite3 \*db);

~Budget() = default;

};

#endif

Файл Budget.cpp

#include "Budget.h"

Budget::Budget() : targetAmount(0.0) {}

Budget::Budget(double target) : targetAmount(target) {}

void Budget::addIncome(const std::string &description, double amount) {

IncomeRecord income(description, amount, incomeRecords.size() + 1);

incomeRecords.push\_back(income);

totalIncome += amount;

}

void Budget::addExpense(const std::string &description, double amount) {

ExpenseRecord expense(description, amount, expenseRecords.size() + 1);

expenseRecords.push\_back(expense);

totalExpenses += amount;

}

double Budget::getTotalIncome() const {

return totalIncome;

}

double Budget::getTotalExpenses() const {

return totalExpenses;

}

double Budget::getRemainingBudget() const {

return targetAmount - totalExpenses;

}

double Budget::getTargetAmount() const {

return targetAmount;

}

void Budget::displayBudgetSummary() const {

std::cout << std::format("БЮДЖЕТ: {:.2f}\n\n", targetAmount);

std::cout << std::format("ДОХОДЫ: {:.2f}\n", totalIncome);

std::cout << std::format("РАСХОДЫ: {:.2f}\n", totalExpenses);

std::cout << std::format("ОСТАТОК: {:.2f}\n", getRemainingBudget());

}

void Budget::displayIncomeRecords() const {

std::cout << "Записи доходов:\n";

for (const auto &record : incomeRecords) {

record.display();

}

}

void Budget::displayExpenseRecords() const {

std::cout << "Записи расходов:\n";

for (const auto &record : expenseRecords) {

record.display();

}

}

void Budget::saveBudgetToDatabase(sqlite3 \*db) const {

std::string sql = std::format("INSERT OR REPLACE INTO Budget (ID, TargetAmount) VALUES (1, {:.2f});", targetAmount);

char \*errMsg = nullptr;

if (sqlite3\_exec(db, sql.c\_str(), nullptr, nullptr, &errMsg) != SQLITE\_OK) {

std::cerr << "SQL error: " << (errMsg ? errMsg : "Unknown error") << std::endl;

sqlite3\_free(errMsg);

}

}

void Budget::loadBudgetFromDatabase(sqlite3 \*db) {

const char \*sql = "SELECT TargetAmount FROM Budget WHERE ID = 1;";

sqlite3\_stmt \*stmt = nullptr;

if (sqlite3\_prepare\_v2(db, sql, -1, &stmt, nullptr) == SQLITE\_OK) {

if (sqlite3\_step(stmt) == SQLITE\_ROW) {

targetAmount = sqlite3\_column\_double(stmt, 0);

}

} else {

std::cerr << "Failed to prepare statement: " << sqlite3\_errmsg(db) << std::endl;

}

sqlite3\_finalize(stmt);

}

Файл CurrencyConeverterWindow.h

#ifndef CURRENCYCONVERTERWINDOW\_H

#define CURRENCYCONVERTERWINDOW\_H

#include <QDialog>

#include <QLineEdit>

#include <QNetworkAccessManager>

#include <QNetworkReply>

#include <QMap>

#include <QMainWindow>

#include <QMouseEvent>

#include <QMessageBox>

#include <QJsonDocument>

#include <QJsonObject>

#include <QJsonValue>

#include <QMainWindow>

#include <QMouseEvent>

QT\_BEGIN\_NAMESPACE

namespace Ui { class CurrencyConverterWindow; }

QT\_END\_NAMESPACE

class CurrencyConverterWindow : public QDialog {

Q\_OBJECT

public:

explicit CurrencyConverterWindow(const QString &apiKey, QWidget \*parent = nullptr);

~CurrencyConverterWindow();

private:

Ui::CurrencyConverterWindow \*ui;

QMap<QString, double> rates;

QNetworkAccessManager \*networkManager;

QString apiKey;

double usdToByn;

double eurToByn;

double rubToByn;

double convert(double amount, const QString &fromCurrency, const QString &toCurrency);

void fetchCurrencyRates(); //

void updateCurrencyRates();

void mousePressEvent(QMouseEvent \*event);

private slots:

void on\_convertButton\_clicked();

void onNetworkReply(QNetworkReply \*reply);

};

#endif

Файл CurrencyConverterWindow.cpp

#include "CurrencyConverterWindow.h"

#include "ui\_CurrencyConverterWindow.h"

CurrencyConverterWindow::CurrencyConverterWindow(const QString &apiKey, QWidget \*parent)

: QDialog(parent), ui(new Ui::CurrencyConverterWindow), networkManager(new QNetworkAccessManager(this)), apiKey(apiKey) {

ui->setupUi(this);

ui->fromCurrencyComboBox->addItems({"USD", "EUR", "RUB","BYN"});

ui->toCurrencyComboBox->addItems({"USD", "EUR", "RUB","BYN"});

connect(networkManager, &QNetworkAccessManager::finished, this, &CurrencyConverterWindow::onNetworkReply);

connect(ui->convertButton, &QPushButton::clicked, this, &CurrencyConverterWindow::on\_convertButton\_clicked);

fetchCurrencyRates();

}

CurrencyConverterWindow::~CurrencyConverterWindow() {

delete ui;

}

void CurrencyConverterWindow::fetchCurrencyRates() {

if (apiKey.isEmpty()) {

QMessageBox::warning(this, "Ошибка", "Введите действительный API-ключ.");

return;

}

QString url = QString("https://api.exchangeratesapi.io/v1/latest?access\_key=%1").arg(apiKey);

qDebug() << "Формируемый URL:" << url;

QNetworkRequest request((QUrl(url)));

networkManager->get(request);

}

void CurrencyConverterWindow::onNetworkReply(QNetworkReply \*reply) {

if (reply->error() == QNetworkReply::NoError) {

QJsonDocument doc = QJsonDocument::fromJson(reply->readAll());

QJsonObject jsonObj = doc.object();

QJsonObject ratesObj = jsonObj["rates"].toObject();

rates.clear();

for (const QString &currency : ratesObj.keys()) {

rates[currency] = ratesObj[currency].toDouble();

}

ui->usdToByn->setText(QString("1 USD = %1 BYN").arg(QString::number(rates.value("BYN") / rates.value("USD"), 'f', 2)));

ui->eurToByn->setText(QString("1 EUR = %1 BYN").arg(QString::number(rates.value("BYN") / rates.value("EUR"), 'f', 2)));

ui->rubToByn->setText(QString("100 RUB = %1 BYN").arg(QString::number(rates.value("BYN") / rates.value("RUB") \* 100, 'f', 2)));

} else {

QMessageBox::warning(this, "Ошибка сети", reply->errorString());

}

reply->deleteLater();

}

double CurrencyConverterWindow::convert(double amount, const QString &fromCurrency, const QString &toCurrency) {

if (!rates.contains(fromCurrency) || !rates.contains(toCurrency)) {

throw std::invalid\_argument("Currency not found");

}

double amountInBase = amount / rates[fromCurrency];

return amountInBase \* rates[toCurrency];

}

void CurrencyConverterWindow::on\_convertButton\_clicked() {

bool ok;

double amount = ui->amountLineEdit->text().toDouble(&ok);

if (!ok) {

QMessageBox::warning(this, "Ошибка", "Введите корректную сумму.");

return;

}

QString fromCurrency = ui->fromCurrencyComboBox->currentText();

QString toCurrency = ui->toCurrencyComboBox->currentText();

try {

double convertedAmount = convert(amount, fromCurrency, toCurrency);

ui->resultLineEdit->setText(QString::number(convertedAmount, 'f', 2));

} catch (const std::invalid\_argument &e) {

QMessageBox::warning(this, "Ошибка", e.what());

}

}

void CurrencyConverterWindow::mousePressEvent(QMouseEvent \*event) {

if (!ui->amountLineEdit->geometry().contains(event->pos()) &&

!ui->resultLineEdit->geometry().contains(event->pos())) {

this->clearFocus();

}

QWidget::mousePressEvent(event);

}

Файл CustomException.h

#ifndef CUSTOM\_EXCEPTION\_H

#define CUSTOM\_EXCEPTION\_H

#include <exception>

#include <string>

class CustomException : public std::exception {

public:

explicit CustomException(const std::string& message)

: message\_(message) {}

const char\* what() const noexcept override {

return message\_.c\_str();

}

private:

std::string message\_;

};

#endif

Файл DatabaseService.h

#ifndef DATABASESERVICE\_H

#define DATABASESERVICE\_H

#include <sqlite3.h>

#include <string>

#include <memory>

#include <iostream>

#include <filesystem>

#include <memory>

#include "FinanceEntry.h"

#include "ExpenseRecord.h"

#include "IncomeRecord.h"

#include "CustomException.h"

class DatabaseService {

private:

sqlite3 \*db = nullptr;

bool isOpen = false;

DatabaseService() = default;

public:

DatabaseService(const DatabaseService &) = delete;

DatabaseService &operator=(const DatabaseService &) = delete;

DatabaseService(DatabaseService &&other) noexcept = default;

DatabaseService &operator=(DatabaseService &&other) noexcept = default;

static inline DatabaseService& getInstance() {

static DatabaseService instance;

return instance;

}

bool openDatabase(const std::string &dbName);

void closeDatabase();

void clearRecords() const;

int createRecord(const std::string &description, double amount, const std::string &type) const;

std::unique\_ptr<FinanceEntry> getRecordById(int id) const;

void executeSQL(const std::string &sql) const;

sqlite3\_stmt \*prepareStatement(const std::string &sql) const;

int getLastInsertedId() const;

sqlite3 \*getDb() const { return db; }

void createBudgetTable();

};

#endif

Файл DatabaseService.cpp

#include "DatabaseService.h"

bool DatabaseService::openDatabase(const std::string &dbName) {

std::cout << "Attempting to open database: " << dbName << std::endl;

if (!std::filesystem::exists(dbName)) {

std::cerr << "Database file does not exist at: " << dbName << std::endl;

return false;

}

std::cout << "Current working directory: " << std::filesystem::current\_path() << std::endl;

if (isOpen) {

std::cerr << "Database is already open." << std::endl;

return true;

}

if (sqlite3\_open(dbName.c\_str(), &db) != SQLITE\_OK) {

std::cerr << "Can't open database: " << sqlite3\_errmsg(db) << std::endl;

return false;

} else {

isOpen = true;

std::cout << "Opened database successfully" << std::endl;

executeSQL("CREATE TABLE IF NOT EXISTS FinanceRecords (ID INTEGER PRIMARY KEY AUTOINCREMENT, Description TEXT, Amount REAL, Type TEXT);");

executeSQL("CREATE TABLE IF NOT EXISTS Budget (ID INTEGER PRIMARY KEY AUTOINCREMENT, TargetAmount REAL NOT NULL);");

executeSQL("CREATE TABLE IF NOT EXISTS Debts (ID INTEGER PRIMARY KEY AUTOINCREMENT, Creditor TEXT, Amount REAL, DueDate TEXT, Payments TEXT, ReminderSet INTEGER);");

return true;

}

}

void DatabaseService::closeDatabase() {

if (db) {

sqlite3\_close(db);

db = nullptr;

isOpen = false;

std::cout << "Database closed." << std::endl;

}

}

void DatabaseService::executeSQL(const std::string &sql) const {

char \*errMsg = nullptr;

if (sqlite3\_exec(db, sql.c\_str(), nullptr, nullptr, &errMsg) != SQLITE\_OK) {

std::cerr << "SQL error: " << (errMsg ? errMsg : "Unknown error") << std::endl;

sqlite3\_free(errMsg);

} else {

std::cout << "SQL executed successfully: " << sql << std::endl;

}

}

sqlite3\_stmt \*DatabaseService::prepareStatement(const std::string &sql) const {

sqlite3\_stmt \*stmt = nullptr;

if (sqlite3\_prepare\_v2(db, sql.c\_str(), -1, &stmt, nullptr) != SQLITE\_OK) {

std::cerr << "Failed to prepare statement: " << sqlite3\_errmsg(db) << std::endl;

return nullptr;

}

return stmt;

}

void DatabaseService::clearRecords() const {

const char \*deleteSql = "DELETE FROM FinanceRecords;";

std::cout << "Executing SQL: " << deleteSql << std::endl;

char \*errMsg = nullptr;

if (sqlite3\_exec(db, deleteSql, nullptr, nullptr, &errMsg) != SQLITE\_OK) {

std::cerr << "Error clearing records: " << (errMsg ? errMsg : "Unknown error") << std::endl;

sqlite3\_free(errMsg);

} else {

std::cout << "All records cleared successfully." << std::endl;

const char \*resetSql = "DELETE FROM sqlite\_sequence WHERE name='FinanceRecords';";

if (sqlite3\_exec(db, resetSql, nullptr, nullptr, &errMsg) != SQLITE\_OK) {

std::cerr << "Error resetting ID: " << (errMsg ? errMsg : "Unknown error") << std::endl;

sqlite3\_free(errMsg);

} else {

std::cout << "ID reset successfully." << std::endl;

}

}

}

int DatabaseService::createRecord(const std::string &description, double amount, const std::string &type) const {

if (!isOpen) {

std::cerr << "Database is not open. Cannot create record." << std::endl;

return -1;

}

executeSQL("BEGIN TRANSACTION;");

const char \*sql = "INSERT INTO FinanceRecords (Description, Amount, Type) VALUES (?, ?, ?) RETURNING ID;";

sqlite3\_stmt \*stmt = prepareStatement(sql);

if (!stmt) {

std::cerr << "Failed to prepare statement for creating record." << std::endl;

executeSQL("ROLLBACK;");

return -1;

}

sqlite3\_bind\_text(stmt, 1, description.c\_str(), -1, SQLITE\_STATIC);

sqlite3\_bind\_double(stmt, 2, amount);

sqlite3\_bind\_text(stmt, 3, type.c\_str(), -1, SQLITE\_STATIC);

int recordId = -1;

if (sqlite3\_step(stmt) == SQLITE\_ROW) {

recordId = sqlite3\_column\_int(stmt, 0);

} else {

std::cerr << "Error inserting record: " << sqlite3\_errmsg(db) << std::endl;

sqlite3\_finalize(stmt);

executeSQL("ROLLBACK;");

return -1;

}

sqlite3\_finalize(stmt);

executeSQL("COMMIT;");

return recordId;

}

std::unique\_ptr<FinanceEntry> DatabaseService::getRecordById(int id) const {

const char \*sql = "SELECT Description, Amount, Type FROM FinanceRecords WHERE ID = ?;";

sqlite3\_stmt \*stmt = prepareStatement(sql);

std::unique\_ptr<FinanceEntry> record = nullptr;

if (!stmt) {

std::cerr << "Failed to prepare statement." << std::endl;

return nullptr;

}

sqlite3\_bind\_int(stmt, 1, id);

if (sqlite3\_step(stmt) == SQLITE\_ROW) {

std::string description = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 0));

double amount = sqlite3\_column\_double(stmt, 1);

std::string type = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 2));

if (type == "Income") {

record = std::make\_unique<IncomeRecord>(description, amount, id);

} else if (type == "Expense") {

record = std::make\_unique<ExpenseRecord>(description, amount, id);

}

} else {

std::cerr << "No record found with ID: " << id << std::endl;

}

sqlite3\_finalize(stmt);

return record;

}

int DatabaseService::getLastInsertedId() const {

const char \*sql = "SELECT last\_insert\_rowid();";

sqlite3\_stmt \*stmt;

int lastId = -1;

if (sqlite3\_prepare\_v2(db, sql, -1, &stmt, nullptr) != SQLITE\_OK) {

throw CustomException("Ошибка подготовки запроса: " + std::string(sqlite3\_errmsg(db)));

}

if (sqlite3\_step(stmt) == SQLITE\_ROW) {

lastId = sqlite3\_column\_int(stmt, 0);

}

sqlite3\_finalize(stmt);

return lastId;

}

Файл Debt.h

#ifndef DEBT\_H

#define DEBT\_H

#include <string>

struct Debt {

double amount;

std::string creditor;

std::string dueDate;

};

#endif

Файл DebtManager.h

#ifndef DEBT\_MANAGER\_H

#define DEBT\_MANAGER\_H

#include <string>

#include <vector>

#include "DatabaseService.h"

#include "Debt.h"

#include <iostream>

#include <sqlite3.h>

class DebtManager {

public:

explicit DebtManager(const std::string& dbName);

int addDebt(const std::string& creditor, double amount, const std::string& dueDate);

bool makePayment(int debtId, double paymentAmount);

std::vector<Debt> getDebts() const;

bool deleteAllDebts();

private:

DatabaseService& dbService;

};

#endif

Файл DebtManager.cpp

#include "DebtManager.h"

DebtManager::DebtManager(const std::string& dbName)

: dbService(DatabaseService::getInstance()) {

if (!dbService.openDatabase(dbName)) {

throw CustomException("Не удалось открыть базу данных!");

}

}

int DebtManager::addDebt(const std::string& creditor, double amount, const std::string& dueDate) {

const char\* sql = "INSERT INTO Debts (Creditor, Amount, DueDate, Payments, ReminderSet) VALUES (?, ?, ?, '', 0);";

sqlite3\_stmt\* stmt = dbService.prepareStatement(sql);

if (!stmt) {

std::cerr << "Failed to prepare statement for adding debt." << std::endl;

return -1;

}

sqlite3\_bind\_text(stmt, 1, creditor.c\_str(), -1, SQLITE\_STATIC);

sqlite3\_bind\_double(stmt, 2, amount);

sqlite3\_bind\_text(stmt, 3, dueDate.c\_str(), -1, SQLITE\_STATIC);

if (sqlite3\_step(stmt) != SQLITE\_DONE) {

std::cerr << "Failed to execute statement: " << sqlite3\_errmsg(dbService.getDb()) << std::endl;

sqlite3\_finalize(stmt);

return -1;

}

int lastId = dbService.getLastInsertedId();

sqlite3\_finalize(stmt);

return lastId;

}

bool DebtManager::makePayment(int debtId, double paymentAmount) {

const char\* sql = "UPDATE Debts SET Amount = Amount - ?, Payments = COALESCE(Payments, '') || ? || ';' WHERE ID = ?;";

sqlite3\_stmt\* stmt = dbService.prepareStatement(sql);

if (!stmt) {

std::cerr << "Failed to prepare statement for making payment." << std::endl;

return false;

}

sqlite3\_bind\_double(stmt, 1, paymentAmount);

sqlite3\_bind\_text(stmt, 2, std::to\_string(paymentAmount).c\_str(), -1, SQLITE\_STATIC);

sqlite3\_bind\_int(stmt, 3, debtId);

if (sqlite3\_step(stmt) != SQLITE\_DONE) {

std::cerr << "Failed to execute statement: " << sqlite3\_errmsg(dbService.getDb()) << std::endl;

sqlite3\_finalize(stmt);

return false;

}

sqlite3\_finalize(stmt);

return true;

}

std::vector<Debt> DebtManager::getDebts() const {

std::vector<Debt> debts;

const char\* sql = "SELECT ID, Creditor, Amount, DueDate FROM Debts;";

sqlite3\_stmt\* stmt = dbService.prepareStatement(sql);

if (!stmt) {

std::cerr << "Failed to prepare statement for retrieving debts." << std::endl;

return debts;

}

while (sqlite3\_step(stmt) == SQLITE\_ROW) {

Debt debt;

debt.creditor = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 1));

debt.amount = sqlite3\_column\_double(stmt, 2);

debt.dueDate = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 3));

debts.push\_back(debt);

}

sqlite3\_finalize(stmt);

return debts;

}

bool DebtManager::deleteAllDebts() {

const char\* sql = "DELETE FROM Debts;";

sqlite3\_stmt\* stmt = dbService.prepareStatement(sql);

if (!stmt) {

std::cerr << "Failed to prepare statement for deleting all debts." << std::endl;

return false;

}

if (sqlite3\_step(stmt) != SQLITE\_DONE) {

std::cerr << "Failed to execute statement: " << sqlite3\_errmsg(dbService.getDb()) << std::endl;

sqlite3\_finalize(stmt);

return false;

}

sqlite3\_finalize(stmt);

return true;

}

Файл DepositWindow.h

#ifndef DEPOSITWINDOW\_H

#define DEPOSITWINDOW\_H

#include <QDialog>

#include <QLineEdit>

#include <QPushButton>

#include <QLabel>

#include <QMessageBox>

#include <cmath>

#include "ui\_DepositWindow.h"

class DepositWindow : public QDialog {

Q\_OBJECT

public:

explicit DepositWindow(QWidget \*parent = nullptr);

private slots:

void onCalculateButtonClicked();

private:

Ui::DepositWindow \*ui;

double calculateInvestment(double initialAmount, double interestRate, int years);

};

#endif

Файл DepositWindow.cpp

#include "DepositWindow.h"

DepositWindow::DepositWindow(QWidget \*parent)

: QDialog(parent),

ui(new Ui::DepositWindow)

{

ui->setupUi(this);

connect(ui->calculateButton, &QPushButton::clicked, this, &DepositWindow::onCalculateButtonClicked);

}

void DepositWindow::onCalculateButtonClicked() {

double initialAmount = ui->initialAmountEdit->text().toDouble();

double interestRate = ui->interestRateEdit->text().toDouble();

int years = ui->yearsEdit->text().toInt();

double finalAmount = calculateInvestment(initialAmount, interestRate, years);

ui->resultLabel->setText(QString("Результат: %1").arg(finalAmount));

}

double DepositWindow::calculateInvestment(double initialAmount, double interestRate, int years) {

return initialAmount \* std::pow((1 + interestRate / 100), years);

}

Файл ExpenseRecord.h

#ifndef EXPENSERECORD\_H

#define EXPENSERECORD\_H

#include "FinanceEntry.h"

#include <stdexcept>

class ExpenseRecord : public FinanceEntry {

public:

ExpenseRecord(const std::string &description, double amount, int id)

: FinanceEntry(description, amount, id) {

if (description.empty()) {

throw std::invalid\_argument("Описание не может быть пустым.");

}

if (amount < 0) {

throw std::invalid\_argument("Сумма не может быть отрицательной.");

}

}

ExpenseRecord(const ExpenseRecord& other) = default;

ExpenseRecord(ExpenseRecord&& other) noexcept = default;

ExpenseRecord& operator=(const ExpenseRecord& other) = default;

ExpenseRecord& operator=(ExpenseRecord&& other) noexcept = default;

std::string getType() const override {

return "Expense";

}

};

#endif

Файл IncomeRecord.h

#ifndef INCOMERECORD\_H

#define INCOMERECORD\_H

#include "FinanceEntry.h"

#include <stdexcept>

class IncomeRecord : public FinanceEntry {

public:

IncomeRecord(const std::string &description, double amount, int id)

: FinanceEntry(description, amount, id) {

if (description.empty()) {

throw std::invalid\_argument("Описание не может быть пустым.");

}

if (amount < 0) {

throw std::invalid\_argument("Сумма не может быть отрицательной.");

}

}

IncomeRecord(const IncomeRecord& other) = default;

IncomeRecord(IncomeRecord&& other) noexcept = default;

IncomeRecord& operator=(const IncomeRecord& other) = default;

IncomeRecord& operator=(IncomeRecord&& other) noexcept = default;

std::string getType() const override {

return "Income";

}

};

#endif

Файл FinanceEntry.h

#ifndef FINANCEENTRY\_H

#define FINANCEENTRY\_H

#include <string>

#include <iostream>

class FinanceEntry {

private:

std::string description;

double amount;

int id;

public:

FinanceEntry(const std::string &description, double amount, int id)

: description(description), amount(amount), id(id) {}

FinanceEntry(const FinanceEntry& other) = default;

FinanceEntry(FinanceEntry&& other) noexcept = default;

FinanceEntry& operator=(const FinanceEntry& other) = default;

FinanceEntry& operator=(FinanceEntry&& other) noexcept = default;

virtual ~FinanceEntry() = default;

virtual std::string getType() const = 0;

virtual void display() const {

std::cout << "ID: " << id << ", Description: " << description

<< ", Amount: " << amount << ", Type: " << getType() << std::endl;

}

double getAmount() const { return amount; }

std::string getDescription() const { return description; }

int getId() const { return id; }

};

#endif

Файл FinanceService.h

#ifndef FINANCESERVICE\_H

#define FINANCESERVICE\_H

#include <string>

#include <iostream>

#include <memory>

#include <vector>

#include <format>

#include "DatabaseService.h"

#include "Budget.h"

#include "sqlite3.h"

#include "FinanceEntry.h"

#include <QString>

#include "DatabaseService.h"

#include "Budget.h"

#include "Stack.h"

#include <stdexcept>

#include "CustomException.h"

class FinanceService {

private:

Budget budget;

double targetBudget;

double balanceDelta;

public:

explicit FinanceService(double initialBudget = 0.0);

void setTargetBudget(double targetBudget);

void createRecord(const std::string &description, double amount, const std::string &type);

void readRecords() const;

void updateRecord(int id, const std::string &description, double amount) const;

void deleteRecord(int id) const;

double calculateTotalBalance() const;

void clearRecords() const;

std::unique\_ptr<FinanceEntry> getRecordById(int id) const;

int createIncome(const std::string &description, double amount);

int createExpense(const std::string &description, double amount);

void displayBudgetSummary() const;

void displayIncomeRecords() const;

void displayExpenseRecords() const;

QString getBudgetSummary() const;

double getBalance() const;

std::vector<std::unique\_ptr<FinanceEntry>> getRecords() const;

double calculateTotalIncome() const;

double calculateTotalExpenses() const;

int getLastRecordId() const;

double updateBalance(double winning);

void loadBudget();

void saveBudget() const;

};

#endif

Файл FinanceService.cpp

#include "FinanceService.h"

template<typename T>

void bindParameter(sqlite3\_stmt \*stmt, int index, const T &value);

template<>

void bindParameter<std::string>(sqlite3\_stmt \*stmt, int index, const std::string &value) {

sqlite3\_bind\_text(stmt, index, value.c\_str(), -1, SQLITE\_STATIC);

}

template<>

void bindParameter<double>(sqlite3\_stmt \*stmt, int index, const double &value) {

sqlite3\_bind\_double(stmt, index, value);

}

template<>

void bindParameter<int>(sqlite3\_stmt \*stmt, int index, const int &value) {

sqlite3\_bind\_int(stmt, index, value);

}

FinanceService::FinanceService(double initialBudget)

: budget(initialBudget), targetBudget(initialBudget) {

DatabaseService &dbService = DatabaseService::getInstance();

if (!dbService.openDatabase("Records.db")) {

throw CustomException("Не удалось открыть базу данных!");

}

budget.loadBudgetFromDatabase(dbService.getDb());

}

void FinanceService::setTargetBudget(double newTargetBudget) {

this->targetBudget = newTargetBudget;

const DatabaseService &dbService = DatabaseService::getInstance();

const char \*sql = "UPDATE Budget SET TargetAmount = ?;";

sqlite3\_stmt \*stmt = dbService.prepareStatement(sql);

if (!stmt) {

throw CustomException("Ошибка подготовки SQL-запроса для обновления бюджета.");

}

bindParameter(stmt, 1, newTargetBudget);

if (sqlite3\_step(stmt) != SQLITE\_DONE) {

throw CustomException("Ошибка обновления бюджета: " + std::string(sqlite3\_errmsg(dbService.getDb())));

}

sqlite3\_finalize(stmt);

}

int FinanceService::createIncome(const std::string &description, double amount) {

const DatabaseService &dbService = DatabaseService::getInstance();

if (amount <= 0) {

throw CustomException("Сумма дохода должна быть положительной.");

}

try {

budget.addIncome(description, amount);

return dbService.createRecord(description, amount, "Income");

} catch (const std::system\_error &e) {

throw CustomException("Ошибка при создании дохода: " + std::string(e.what()));

}

}

void FinanceService::readRecords() const {

const DatabaseService &dbService = DatabaseService::getInstance();

const char \*sql = "SELECT ID, Description, Amount, Type FROM FinanceRecords;";

sqlite3\_stmt \*stmt = dbService.prepareStatement(sql);

if (!stmt) {

throw CustomException("Ошибка подготовки SQL-запроса для чтения записей.");

}

Stack<std::unique\_ptr<FinanceEntry>> recordStack;

while (sqlite3\_step(stmt) == SQLITE\_ROW) {

int id = sqlite3\_column\_int(stmt, 0);

std::string description = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 1));

double amount = sqlite3\_column\_double(stmt, 2);

std::string type = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 3));

std::unique\_ptr<FinanceEntry> record;

if (type == "Income") {

record = std::make\_unique<IncomeRecord>(description, amount, id);

} else if (type == "Expense") {

record = std::make\_unique<ExpenseRecord>(description, amount, id);

} else {

throw CustomException("Неизвестный тип записи: " + type);

}

recordStack.push(std::move(record));

}

while (!recordStack.isEmpty()) {

auto record = std::move(recordStack.top());

record->display();

recordStack.pop();

}

sqlite3\_finalize(stmt);

}

void FinanceService::updateRecord(int id, const std::string &description, double amount) const {

const DatabaseService &dbService = DatabaseService::getInstance();

std::string sql = "UPDATE FinanceRecords SET Description = ?, Amount = ? WHERE ID = ?;";

sqlite3\_stmt \*stmt = dbService.prepareStatement(sql);

if (!stmt) {

throw CustomException("Ошибка подготовки SQL-запроса для обновления записи.");

}

bindParameter(stmt, 1, description);

bindParameter(stmt, 2, amount);

bindParameter(stmt, 3, id);

if (sqlite3\_step(stmt) != SQLITE\_DONE) {

throw CustomException("Ошибка обновления записи: " + std::string(sqlite3\_errmsg(dbService.getDb())));

}

sqlite3\_finalize(stmt);

}

void FinanceService::deleteRecord(int id) const {

const DatabaseService &dbService = DatabaseService::getInstance();

std::string sql = std::format("DELETE FROM FinanceRecords WHERE ID = {}; ", id);

try {

dbService.executeSQL(sql);

} catch (const std::exception &e) {

throw CustomException(std::format("Ошибка удаления записи с ID {}: {}", id, e.what()));

}

}

std::unique\_ptr<FinanceEntry> FinanceService::getRecordById(int id) const {

const DatabaseService &dbService = DatabaseService::getInstance();

auto record = dbService.getRecordById(id);

if (!record) {

throw CustomException(std::format("Запись с ID {} не найдена!", id));

}

return record;

}

int FinanceService::createExpense(const std::string &description, double amount) {

const DatabaseService &dbService = DatabaseService::getInstance();

if (amount <= 0) {

throw CustomException("Сумма расхода должна быть положительной.");

}

try {

budget.addExpense(description, amount);

return dbService.createRecord(description, amount, "Expense");

} catch (const std::system\_error &e) {

throw CustomException("Ошибка при создании расхода: " + std::string(e.what()));

}

}

void FinanceService::displayIncomeRecords() const {

const DatabaseService &dbService = DatabaseService::getInstance();

const char \*sql = "SELECT \* FROM FinanceRecords WHERE Type = 'Income' ORDER BY Description;";

sqlite3\_stmt \*stmt = dbService.prepareStatement(sql);

if (!stmt) {

throw CustomException("Ошибка подготовки SQL-запроса для отображения доходов.");

}

std::cout << "Записи доходов (отсортированные по описанию):" << std::endl;

while (sqlite3\_step(stmt) == SQLITE\_ROW) {

int id = sqlite3\_column\_int(stmt, 0);

std::string description = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 1));

double amount = sqlite3\_column\_double(stmt, 2);

std::string type = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 3));

std::cout << "ID: " << id

<< ", Описание: " << description

<< ", Сумма: " << amount

<< ", Тип: " << type << std::endl;

}

sqlite3\_finalize(stmt);

}

void FinanceService::displayExpenseRecords() const {

const DatabaseService &dbService = DatabaseService::getInstance();

const char \*sql = "SELECT \* FROM FinanceRecords WHERE Type = 'Expense' ORDER BY Description;";

sqlite3\_stmt \*stmt = dbService.prepareStatement(sql);

if (!stmt) {

throw CustomException("Ошибка подготовки SQL-запроса для отображения расходов.");

}

std::cout << "Записи расходов (отсортированные по описанию):" << std::endl;

while (sqlite3\_step(stmt) == SQLITE\_ROW) {

int id = sqlite3\_column\_int(stmt, 0);

std::string description = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 1));

double amount = sqlite3\_column\_double(stmt, 2);

std::string type = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 3));

std::cout << "ID: " << id

<< ", Описание: " << description

<< ", Сумма: " << amount

<< ", Тип: " << type << std::endl;

}

sqlite3\_finalize(stmt);

}

void FinanceService::saveBudget() const {

const DatabaseService &dbService = DatabaseService::getInstance();

budget.saveBudgetToDatabase(dbService.getDb());

}

double FinanceService::calculateTotalIncome() const {

const DatabaseService &dbService = DatabaseService::getInstance();

double totalIncome = 0.0;

const char \*sql = "SELECT SUM(Amount) FROM FinanceRecords WHERE Type = 'Income';";

sqlite3\_stmt \*stmt = dbService.prepareStatement(sql);

if (!stmt) {

throw CustomException("Ошибка подготовки SQL-запроса для расчета общего дохода.");

}

if (sqlite3\_step(stmt) == SQLITE\_ROW) {

totalIncome = sqlite3\_column\_double(stmt, 0);

}

sqlite3\_finalize(stmt);

return totalIncome;

}

double FinanceService::calculateTotalExpenses() const {

const DatabaseService &dbService = DatabaseService::getInstance();

double totalExpenses = 0.0;

const char \*sql = "SELECT SUM(Amount) FROM FinanceRecords WHERE Type = 'Expense';";

sqlite3\_stmt \*stmt = dbService.prepareStatement(sql);

if (!stmt) {

throw CustomException("Ошибка подготовки SQL-запроса для расчета общих расходов.");

}

if (sqlite3\_step(stmt) == SQLITE\_ROW) {

totalExpenses = sqlite3\_column\_double(stmt, 0);

}

sqlite3\_finalize(stmt);

return totalExpenses;

}

QString FinanceService::getBudgetSummary() const {

double targetAmount = budget.getTargetAmount();

double totalIncome = calculateTotalIncome();

double totalExpenses = calculateTotalExpenses();

double balance = targetAmount + totalIncome - totalExpenses + balanceDelta;

return QString("БЮДЖЕТ: %1\n\nДОХОДЫ: %2\nРАСХОДЫ: %3\nОСТАТОК: %4")

.arg(targetAmount)

.arg(totalIncome)

.arg(totalExpenses)

.arg(balance);

}

double FinanceService::getBalance() const {

double targetAmount = budget.getTargetAmount();

double totalIncome = calculateTotalIncome();

double totalExpenses = calculateTotalExpenses();

return targetAmount + totalIncome - totalExpenses;

}

double FinanceService::updateBalance(double winning) {

balanceDelta = 0.0;

balanceDelta += winning;

return balanceDelta;

}

std::vector<std::unique\_ptr<FinanceEntry>> FinanceService::getRecords() const {

std::vector<std::unique\_ptr<FinanceEntry>> records;

const DatabaseService &dbService = DatabaseService::getInstance();

const char \*sql = "SELECT ID, Description, Amount, Type FROM FinanceRecords;";

sqlite3\_stmt \*stmt = dbService.prepareStatement(sql);

if (!stmt) {

throw CustomException("Ошибка подготовки SQL-запроса для получения записей.");

}

while (sqlite3\_step(stmt) == SQLITE\_ROW) {

int id = sqlite3\_column\_int(stmt, 0);

std::string description = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 1));

double amount = sqlite3\_column\_double(stmt, 2);

std::string type = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 3));

std::unique\_ptr<FinanceEntry> record;

if (type == "Income") {

record = std::make\_unique<IncomeRecord>(description, amount, id);

} else {

record = std::make\_unique<ExpenseRecord>(description, amount, id);

}

records.push\_back(std::move(record));

}

sqlite3\_finalize(stmt);

return records;

}

void FinanceService::clearRecords() const {

const DatabaseService &dbService = DatabaseService::getInstance();

dbService.clearRecords();

}

void FinanceService::loadBudget() {

const DatabaseService& dbService = DatabaseService::getInstance();

budget.loadBudgetFromDatabase(dbService.getDb());

}

int FinanceService::getLastRecordId() const {

const DatabaseService &dbService = DatabaseService::getInstance();

return dbService.getLastInsertedId();

}

Файл MainWindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTableWidget>

#include "FinanceService.h"

#include <QInputDialog>

#include <QMessageBox>

#include <QVBoxLayout>

#include <QLabel>

#include <QPushButton>

#include <QListWidget>

#include "DatabaseService.h"

#include "ui\_MainWindow.h"

#include <QtCharts/QChartView>

#include <QtCharts/QPieSeries>

#include <QFile>

#include <QTextStream>

#include <QDebug>

#include "ActionManager.h"

#include "СustomTitleBar.h"

#include "DebtManager.h"

#include "ui\_MainWindow.h"

#include "FinanceService.h"

#include "CustomException.h"

#include "ReportService.h"

#include "DebtManager.h"

#include <QtCharts/QChart>

#include <QPropertyAnimation>

#include <QGraphicsOpacityEffect>

#include <QDateEdit>

#include "CurrencyConverterWindow.h"

#include "InvestmentCalculator.h"

#include "DepositWindow.h"

#include "Budget.h"

class SlotMachine;

class MainWindow : public QMainWindow {

Q\_OBJECT

public:

explicit MainWindow(QWidget \*parent = nullptr);

~MainWindow() override;

void updateBudgetSummary();

private:

Ui::MainWindow \*ui;

Budget budget;

QListWidget \*incomeListWidget;

QListWidget \*expenseListWidget;

FinanceService financeService;

DebtManager debtManager;

QLabel \*budgetSummaryLabel;

QListWidget \*debtListWidget;

QLabel \*totalDebtLabel;

QLabel \*lastDueDateLabel;

CurrencyConverterWindow \*currencyConverterWindow;

ActionManager actionManager;

DepositWindow \*depositWindow;

double totalDebt;

QString lastDueDate;

QWidget\* savingsWidget;

QLabel \*warningLabel;

void loadRecords();

void loadDebts();

void onCategoryChanged();

void showDebts();

void showDeposits();

void showIncomeExpense();

void toggleSideMenu();

void showSideMenu();

void hideSideMenu();

void handleCategorySelection();

void hideChartWidgets();

void showChartWidgets();

void fadeOutWidget(QWidget \*widget);

void fadeInWidget(QWidget \*widget);

void updateDebtSummary();

void showDailySavingsRecommendation();

void mousePressEvent(QMouseEvent \*event) override;

void showExpenseWarning();

void setInputDialogStyle(QInputDialog &dialog);

private slots:

void setTargetBudget();

void addIncome();

void addExpense();

void updateRecord();

void clearAllRecords();

void createPieChart();

void undoLastAction();

void generateJsonReport();

void addDebt();

void onDeleteAllDebts();

void openConverter();

void openCalculator();

void openSlots();

signals:

void expensesAdded();

protected:

};

#endif

Файл MainWindow.cpp

#include "MainWindow.h"

#include "SlotMachine.h"

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent), ui(new Ui::MainWindow), financeService(0.0), budget(budget), actionManager(financeService), debtManager("Records.db"), currencyConverterWindow(nullptr) {

ui->setupUi(this);

setFixedSize(757, 580);

ui->sideMenuWidget->setVisible(false);

ui->debtListWidget->setVisible(false);

ui->addDebtButton->setVisible(false);

ui->deleteAllDebtsButton->setVisible(false);

ui->debtLabelMenu->setVisible(false);

ui->budgetLabelMenu->setVisible(true);

incomeListWidget = ui->incomeListWidget;

expenseListWidget = ui->expenseListWidget;

debtListWidget = ui->debtListWidget;

savingsWidget = new QLabel(this);

savingsWidget->setFixedSize(600, 50);

savingsWidget->setGeometry(63, 45, 450, 200);

savingsWidget->setVisible(false);

budgetSummaryLabel = new QLabel(this);

budgetSummaryLabel->setGeometry(280, 125, 550, 110);

budgetSummaryLabel->setAlignment(Qt::AlignCenter);

budgetSummaryLabel->setStyleSheet("font-size: 18px; color: white;");

totalDebtLabel = new QLabel("Общая сумма долга: 0", this);

totalDebtLabel->setGeometry(180, 440, 400, 40);

totalDebtLabel->setAlignment(Qt::AlignCenter);

totalDebtLabel->setStyleSheet("font-size: 18px; color: white;");

totalDebtLabel->setVisible(false);

lastDueDateLabel = new QLabel("Последняя дата просрочки: нет", this);

lastDueDateLabel->setGeometry(180, 470, 400, 40);

lastDueDateLabel->setAlignment(Qt::AlignCenter);

lastDueDateLabel->setStyleSheet("font-size: 18px; color: white;");

lastDueDateLabel->setVisible(false);

connect(ui->setTargetBudgetButton, &QPushButton::clicked, this, &MainWindow::setTargetBudget);

connect(ui->addIncomeButton, &QPushButton::clicked, this, &MainWindow::addIncome);

connect(ui->addExpenseButton, &QPushButton::clicked, this, &MainWindow::addExpense);

connect(ui->clearButton, &QPushButton::clicked, this, &MainWindow::clearAllRecords);

connect(ui->updateRecordButton, &QPushButton::clicked, this, &MainWindow::updateRecord);

connect(ui->undoButton, &QPushButton::clicked, this, &MainWindow::undoLastAction);

connect(this, &MainWindow::expensesAdded, this, &MainWindow::createPieChart);

connect(ui->generateJsonReportButton, &QPushButton::clicked, this, &MainWindow::generateJsonReport);

connect(ui->hamburgerButton, &QPushButton::clicked, this, &MainWindow::toggleSideMenu);

connect(ui->incomeExpenseButton, &QPushButton::clicked, this, &MainWindow::showIncomeExpense);

connect(ui->debtButton, &QPushButton::clicked, this, &MainWindow::showDebts);

connect(ui->depositsButton, &QPushButton::clicked, this, &MainWindow::openCalculator);

connect(ui->addDebtButton, &QPushButton::clicked, this, &MainWindow::addDebt);

connect(ui->deleteAllDebtsButton, &QPushButton::clicked, this, &MainWindow::onDeleteAllDebts);

connect(ui->currencyButton, &QPushButton::clicked, this, &MainWindow::openConverter);

connect(ui->slotButton,&QPushButton::clicked, this, &MainWindow::openSlots);

loadRecords();

createPieChart();

updateBudgetSummary();

loadDebts();

showExpenseWarning();

}

MainWindow::~MainWindow() {

DatabaseService::getInstance().closeDatabase();

delete ui;

delete currencyConverterWindow;

}

void MainWindow::mousePressEvent(QMouseEvent \*event) {

if (ui->sideMenuWidget->isVisible() && !ui->sideMenuWidget->geometry().contains(event->pos())) {

hideSideMenu();

}

if (ui->incomeListWidget->geometry().contains(event->pos())) {

QListWidgetItem\* item = ui->incomeListWidget->itemAt(event->pos());

if (item) {

for (int i = 0; i < ui->incomeListWidget->count(); ++i) {

ui->incomeListWidget->item(i)->setSelected(false);

}

item->setSelected(true);

}

} else {

ui->incomeListWidget->clearSelection();

}

if (ui->expenseListWidget->geometry().contains(event->pos())) {

QListWidgetItem\* item = ui->expenseListWidget->itemAt(event->pos());

if (item) {

for (int i = 0; i < ui->expenseListWidget->count(); ++i) {

ui->expenseListWidget->item(i)->setSelected(false);

}

item->setSelected(true);

}

} else {

ui->expenseListWidget->clearSelection();

}

}

void MainWindow::handleCategorySelection() {

hideSideMenu();

}

void MainWindow::showSideMenu() {

ui->sideMenuWidget->setVisible(true);

QRect startRect = ui->sideMenuWidget->geometry();

QRect endRect = startRect;

endRect.setX(0);

QPropertyAnimation \*animation = new QPropertyAnimation(ui->sideMenuWidget, "geometry");

animation->setDuration(300);

animation->setStartValue(startRect);

animation->setEndValue(endRect);

animation->setEasingCurve(QEasingCurve::OutCubic);

animation->start();

}

void MainWindow::hideSideMenu() {

ui->sideMenuWidget->setVisible(false);

QRect startRect = ui->sideMenuWidget->geometry();

QRect endRect = startRect;

endRect.setX(-startRect.width());

QPropertyAnimation \*animation = new QPropertyAnimation(ui->sideMenuWidget, "geometry");

animation->setDuration(300);

animation->setStartValue(startRect);

animation->setEndValue(endRect);

animation->setEasingCurve(QEasingCurve::OutCubic);

animation->start();

}

void MainWindow::toggleSideMenu() {

qDebug() << "Side menu visible:" << ui->sideMenuWidget->isVisible();

if (ui->sideMenuWidget->isVisible()) {

hideSideMenu();

} else {

showSideMenu();

}

}

void MainWindow::showIncomeExpense() {

if (!ui->sideMenuWidget->isVisible()) {

showSideMenu();

}

showChartWidgets();

fadeInWidget(ui->incomeListWidget);

fadeInWidget(ui->expenseListWidget);

fadeInWidget(ui->addIncomeButton);

fadeInWidget(ui->addExpenseButton);

fadeInWidget(ui->setTargetBudgetButton);

fadeInWidget(ui->updateRecordButton);

fadeInWidget(ui->undoButton);

fadeInWidget(ui->clearButton);

fadeInWidget(ui->generateJsonReportButton);

fadeInWidget(ui->budgetSummaryTextLabel);

fadeInWidget(budgetSummaryLabel);

fadeInWidget(ui->expenseLabel);

fadeInWidget(ui->incomeLabel);

fadeInWidget(ui->currencyButton);

fadeInWidget(ui->budgetLabelMenu);

fadeOutWidget(debtListWidget);

fadeOutWidget(savingsWidget);

fadeOutWidget(ui->deleteAllDebtsButton);

fadeOutWidget(ui->addDebtButton);

fadeOutWidget(totalDebtLabel);

fadeOutWidget(lastDueDateLabel);

fadeOutWidget(ui->debtLabelMenu);

hideSideMenu();

}

void MainWindow::showDebts() {

if (!ui->sideMenuWidget->isVisible()) {

showSideMenu();

}

hideChartWidgets();

fadeOutWidget(ui->incomeListWidget);

fadeOutWidget(ui->expenseListWidget);

fadeOutWidget(ui->addIncomeButton);

fadeOutWidget(ui->addExpenseButton);

fadeOutWidget(ui->setTargetBudgetButton);

fadeOutWidget(ui->updateRecordButton);

fadeOutWidget(ui->undoButton);

fadeOutWidget(ui->clearButton);

fadeOutWidget(ui->generateJsonReportButton);

fadeOutWidget(ui->budgetSummaryTextLabel);

fadeOutWidget(budgetSummaryLabel);

fadeOutWidget(ui->expenseLabel);

fadeOutWidget(ui->incomeLabel);

fadeOutWidget(ui->budgetLabelMenu);

fadeInWidget(debtListWidget);

fadeInWidget(savingsWidget);

fadeInWidget(ui->deleteAllDebtsButton);

fadeInWidget(ui->addDebtButton);

fadeInWidget(totalDebtLabel);

fadeInWidget(lastDueDateLabel);

fadeInWidget(ui->debtLabelMenu);

hideSideMenu();

}

void MainWindow::showDeposits() {

if (!ui->sideMenuWidget->isVisible()) {

showSideMenu();

}

fadeOutWidget(ui->incomeListWidget);

fadeOutWidget(ui->expenseListWidget);

fadeOutWidget(ui->addIncomeButton);

fadeOutWidget(ui->addExpenseButton);

fadeOutWidget(ui->setTargetBudgetButton);

fadeOutWidget(ui->updateRecordButton);

fadeOutWidget(ui->undoButton);

fadeOutWidget(ui->clearButton);

fadeOutWidget(ui->generateJsonReportButton);

fadeOutWidget(ui->budgetSummaryTextLabel);

fadeOutWidget(budgetSummaryLabel);

fadeOutWidget(ui->expenseLabel);

fadeOutWidget(ui->incomeLabel);

fadeOutWidget(ui->currencyButton);

hideChartWidgets();

hideSideMenu();

}

void MainWindow::fadeInWidget(QWidget \*widget) {

if (widget) {

qDebug() << "Fading in widget:" << widget->objectName();

widget->setVisible(true);

QGraphicsOpacityEffect \*effect = new QGraphicsOpacityEffect(widget);

widget->setGraphicsEffect(effect);

QPropertyAnimation \*animation = new QPropertyAnimation(effect, "opacity");

animation->setDuration(300);

animation->setStartValue(0.0);

animation->setEndValue(1.0);

animation->setEasingCurve(QEasingCurve::InOutQuad);

animation->start();

} else {

qDebug() << "Widget is null!";

}

}

void MainWindow::fadeOutWidget(QWidget \*widget) {

if (widget) {

qDebug() << "Fading out widget:" << widget->objectName();

widget->setVisible(false);

QGraphicsOpacityEffect \*effect = new QGraphicsOpacityEffect(widget);

widget->setGraphicsEffect(effect);

QPropertyAnimation \*animation = new QPropertyAnimation(effect, "opacity");

animation->setDuration(300);

animation->setStartValue(1.0);

animation->setEndValue(0.0);

animation->setEasingCurve(QEasingCurve::InOutQuad);

connect(animation, &QPropertyAnimation::finished, widget, &QWidget::hide);

animation->start();

} else {

qDebug() << "Widget is null!";

}

}

void MainWindow::hideChartWidgets() {

for (int i = 0; i < ui->chartLayout->count(); ++i) {

QLayoutItem \*item = ui->chartLayout->itemAt(i);

if (item && item->widget()) {

QWidget \*widget = item->widget();

QGraphicsOpacityEffect \*effect = new QGraphicsOpacityEffect(widget);

widget->setGraphicsEffect(effect);

QPropertyAnimation \*animation = new QPropertyAnimation(effect, "opacity");

animation->setDuration(300);

animation->setStartValue(1.0);

animation->setEndValue(0.0);

animation->setEasingCurve(QEasingCurve::InOutQuad);

connect(animation, &QPropertyAnimation::finished, widget, &QWidget::hide);

animation->start();

}

}

}

void MainWindow::showChartWidgets() {

for (int i = 0; i < ui->chartLayout->count(); ++i) {

QLayoutItem \*item = ui->chartLayout->itemAt(i);

if (item && item->widget()) {

QWidget \*widget = item->widget();

widget->setVisible(true);

QGraphicsOpacityEffect \*effect = new QGraphicsOpacityEffect(widget);

widget->setGraphicsEffect(effect);

QPropertyAnimation \*animation = new QPropertyAnimation(effect, "opacity");

animation->setDuration(300);

animation->setStartValue(0.0);

animation->setEndValue(1.0);

animation->setEasingCurve(QEasingCurve::InOutQuad);

animation->start();

}

}

}

void MainWindow::clearAllRecords() {

try {

financeService.clearRecords();

loadRecords();

updateBudgetSummary();

createPieChart();

QMessageBox::information(this, "Success", "All records have been cleared.");

} catch (const CustomException &e) {

QMessageBox::critical(this, "Error", "An error occurred: " + QString::fromStdString(e.what()));

} catch (const std::exception &e) {

QMessageBox::critical(this, "Error", "An unexpected error occurred: " + QString::fromStdString(e.what()));

}

}

void MainWindow::setTargetBudget() {

bool ok;

QInputDialog dialog(this);

setInputDialogStyle(dialog);

double amount = QInputDialog::getDouble(this, "Установить целевой бюджет", "Введите целевой бюджет:", 0, 0, 1000000, 2, &ok);

if (ok) {

try {

financeService.setTargetBudget(amount);

financeService.loadBudget();

updateBudgetSummary();

QMessageBox::information(this, "Успех", "Целевой бюджет установлен на " + QString::number(amount, 'f', 2));

} catch (const CustomException &e) {

QMessageBox::critical(this, "Ошибка", "Не удалось установить целевой бюджет: " + QString::fromStdString(e.what()));

} catch (const std::exception &e) {

QMessageBox::critical(this, "Ошибка", "Неизвестная ошибка: " + QString::fromStdString(e.what()));

}

}

}

void MainWindow::addIncome() {

bool ok;

QInputDialog inputDialog(this);

inputDialog.setWindowTitle("Income Description");

inputDialog.setLabelText("Enter description:");

setInputDialogStyle(inputDialog);

QString description = inputDialog.getText(this, "Income Description", "Enter description:", QLineEdit::Normal, QString(), &ok);

if (ok && !description.isEmpty()) {

double amount = QInputDialog::getDouble(this, "Income Amount", "Enter amount:", 0, 0, 100000, 2, &ok);

if (ok) {

try {

actionManager.addIncome(description.toStdString(), amount);

loadRecords();

updateBudgetSummary();

QMessageBox::information(this, "Success", "Income added: " + description + " - " + QString::number(amount, 'f', 2));

} catch (const CustomException &e) {

QMessageBox::critical(this, "Error", "Failed to add income: " + QString::fromStdString(e.what()));

} catch (const std::exception &e) {

QMessageBox::critical(this, "Error", "Unexpected error: " + QString::fromStdString(e.what()));

}

}

}

}

void MainWindow::addExpense() {

bool ok;

QInputDialog inputDialog(this);

inputDialog.setWindowTitle("Expense Description");

inputDialog.setLabelText("Enter description:");

setInputDialogStyle(inputDialog);

QString description = inputDialog.getText(this, "Expense Description", "Enter description:", QLineEdit::Normal, QString(), &ok);

if (ok && !description.isEmpty()) {

double amount = QInputDialog::getDouble(this, "Expense Amount", "Enter amount:", 0, 0, 100000, 2, &ok);

if (ok) {

try {

actionManager.addExpense(description.toStdString(), amount);

loadRecords();

updateBudgetSummary();

emit expensesAdded();

QMessageBox::information(this, "Success", "Expense added: " + description + " - " + QString::number(amount, 'f', 2));

} catch (const CustomException &e) {

QMessageBox::critical(this, "Error", "Failed to add expense: " + QString::fromStdString(e.what()));

} catch (const std::exception &e) {

QMessageBox::critical(this, "Error", "Unexpected error: " + QString::fromStdString(e.what()));

}

}

}

}

void MainWindow::updateRecord() {

bool ok;

QInputDialog inputDialog(this);

inputDialog.setWindowTitle("Update Record");

inputDialog.setLabelText("Enter record ID to update:");

setInputDialogStyle(inputDialog);

int recordId = inputDialog.getInt(this, "Update Record", "Enter ID of the record to update:", 0, 0, 1000000, 1, &ok);

if (ok) {

QString newDescription = QInputDialog::getText(this, "New Description", "Enter new description:", QLineEdit::Normal, QString(), &ok);

if (ok && !newDescription.isEmpty()) {

double newAmount = QInputDialog::getDouble(this, "New Amount", "Enter new amount:", 0, 0, 1000000, 2, &ok);

if (ok) {

try {

financeService.updateRecord(recordId, newDescription.toStdString(), newAmount);

loadRecords();

updateBudgetSummary();

QMessageBox::information(this, "Success", "Record updated.");

} catch (const CustomException &e) {

QMessageBox::critical(this, "Error", "Failed to update record: " + QString::fromStdString(e.what()));

} catch (const std::exception &e) {

QMessageBox::critical(this, "Error", "Unexpected error: " + QString::fromStdString(e.what()));

}

}

}

}

}

void MainWindow::loadRecords() {

incomeListWidget->clear();

expenseListWidget->clear();

try {

auto records = financeService.getRecords();

for (const auto &record : records) {

// Убираем ID из отображаемого текста

QString recordText = QString("%1 - %2")

.arg(QString::fromStdString(record->getDescription()))

.arg(QString::number(record->getAmount(), 'f', 2));

if (record->getType() == "Income") {

auto \*item = new QListWidgetItem(recordText);

incomeListWidget->addItem(item);

} else if (record->getType() == "Expense") {

auto \*item = new QListWidgetItem(recordText);

expenseListWidget->addItem(item);

}

}

} catch (const CustomException &e) {

QMessageBox::warning(this, "Error", "Failed to load records: " + QString::fromStdString(e.what()));

} catch (const std::exception &e) {

QMessageBox::warning(this, "Error", "Unexpected error: " + QString::fromStdString(e.what()));

}

}

void MainWindow::updateBudgetSummary() {

try {

QString budgetSummary = financeService.getBudgetSummary();

budgetSummaryLabel->setText(budgetSummary);

} catch (const CustomException &e) {

QMessageBox::warning(this, "Ошибка", "Не удалось обновить сводку бюджета: " + QString::fromStdString(e.what()));

} catch (const std::exception &e) {

QMessageBox::warning(this, "Ошибка", "Неизвестная ошибка: " + QString::fromStdString(e.what()));

}

showExpenseWarning();

}

void MainWindow::createPieChart() {

QLayoutItem \*item;

while ((item = ui->chartLayout->takeAt(0)) != nullptr) {

delete item->widget();

delete item;

}

QPieSeries \*series = new QPieSeries();

auto records = financeService.getRecords();

QMap<QString, double> expenseMap;

QStringList colors = {

"#007BFF",

"#339FFF",

"#66B3FF",

"#99D6FF",

"#0066CC",

"#005BB5",

"#004999"

};

for (const auto &record : records) {

if (record->getType() == "Expense") {

QString description = QString::fromStdString(record->getDescription());

double amount = record->getAmount();

expenseMap[description] += amount;

}

}

for (auto it = expenseMap.constBegin(); it != expenseMap.constEnd(); ++it) {

auto slice = series->append(it.key(), it.value());

slice->setBrush(QBrush(QColor(colors[series->count() % colors.size()])));

slice->setLabel(QString("%1\n%2%").arg(it.key()).arg(QString::number(slice->percentage() \* 100, 'f', 1)));

slice->setLabelBrush(QBrush(Qt::white));

}

if (series->count() == 0) {

QMessageBox::information(this, "Нет данных", "Нет доступных записей расходов для отображения.");

return;

}

series->setLabelsVisible(true);

QChart \*chart = new QChart();

chart->addSeries(series);

chart->setTitle("");

chart->legend()->setVisible(false);

chart->setBackgroundBrush(Qt::transparent);

chart->setPlotAreaBackgroundBrush(Qt::transparent);

chart->setPlotAreaBackgroundPen(QPen(Qt::transparent));

QChartView \*chartView = new QChartView(chart);

chartView->setRenderHint(QPainter::Antialiasing);

chartView->setBackgroundBrush(Qt::transparent);

ui->chartLayout->addWidget(chartView);

QWidget \*parentWidget = ui->chartLayout->parentWidget();

if (parentWidget) {

parentWidget->setStyleSheet("background: transparent;");

}

ui->chartLayout->update();

}

void MainWindow::undoLastAction() {

try {

actionManager.undo();

loadRecords();

updateBudgetSummary();

createPieChart();

QMessageBox::information(this, "Undo", "Last action undone.");

} catch (const CustomException &e) {

QMessageBox::critical(this, "Error", "Failed to undo action: " + QString::fromStdString(e.what()));

}

}

void MainWindow::generateJsonReport() {

QStringList reportTypes = {"Доходы", "Расходы", "Бюджет", "Описание"};

bool ok;

QString selectedType = QInputDialog::getItem(this, "Выбор типа отчета", "Выберите тип отчета:", reportTypes, 0, false, &ok);

if (ok && !selectedType.isEmpty()) {

try {

if (selectedType == "Описание") {

bool okDescription;

QString description = QInputDialog::getText(this, "Введите описание", "Описание:", QLineEdit::Normal, "", &okDescription);

if (okDescription && !description.isEmpty()) {

ReportService reportService(financeService);

reportService.generateDescriptionReport(description.toStdString());

QMessageBox::information(this, "Успех", "Отчет по описанию успешно сгенерирован.");

} else {

QMessageBox::warning(this, "Предупреждение", "Описание не должно быть пустым.");

}

} else {

ReportService reportService(financeService);

reportService.generateJsonReport(selectedType.toStdString());

QMessageBox::information(this, "Успех", "Отчет успешно сгенерирован.");

}

} catch (const CustomException &e) {

QMessageBox::critical(this, "Ошибка", "Не удалось сгенерировать отчет: " + QString::fromStdString(e.what()));

} catch (const std::exception &e) {

QMessageBox::critical(this, "Ошибка", "Неизвестная ошибка: " + QString::fromStdString(e.what()));

}

}

}

void MainWindow::loadDebts() {

debtListWidget->clear();

try {

auto debts = debtManager.getDebts();

for (const auto& debt : debts) {

QString itemText = QString("Кредитор: %1 | Сумма: %2 | Дата просрочки: %3")

.arg(QString::fromStdString(debt.creditor))

.arg(debt.amount, 0, 'f', 2)

.arg(QString::fromStdString(debt.dueDate));

auto \*item = new QListWidgetItem(itemText);

debtListWidget->addItem(item);

}

} catch (const CustomException &e) {

QMessageBox::warning(this, "Ошибка", "Не удалось загрузить долги: " + QString::fromStdString(e.what()));

} catch (const std::exception &e) {

QMessageBox::warning(this, "Ошибка", "Неизвестная ошибка: " + QString::fromStdString(e.what()));

}

updateDebtSummary();

}

void MainWindow::addDebt() {

bool ok;

QString creditor = QInputDialog::getText(this, "Кредитор долга", "Введите имя кредитора:", QLineEdit::Normal, QString(), &ok);

if (ok && !creditor.isEmpty()) {

double amount = QInputDialog::getDouble(this, "Сумма долга", "Введите сумму:", 0, 0, 1000000, 2, &ok);

if (ok) {

QDialog dateDialog(this);

dateDialog.setWindowTitle("Выберите дату погашения");

QDateEdit dueDateEdit;

dueDateEdit.setCalendarPopup(true);

dueDateEdit.setDate(QDate::currentDate());

QPushButton \*confirmButton = new QPushButton("Подтвердить", &dateDialog);

QVBoxLayout \*layout = new QVBoxLayout;

layout->addWidget(&dueDateEdit);

layout->addWidget(confirmButton);

dateDialog.setLayout(layout);

connect(confirmButton, &QPushButton::clicked, &dateDialog, &QDialog::accept);

if (dateDialog.exec() == QDialog::Accepted) {

QString dueDate = dueDateEdit.date().toString("yyyy-MM-dd");

try {

int debtId = debtManager.addDebt(creditor.toStdString(), amount, dueDate.toStdString());

if (debtId != -1) {

loadDebts();

QMessageBox::information(this, "Успех", "Долг добавлен: " + creditor + " - " + QString::number(amount, 'f', 2));

} else {

QMessageBox::warning(this, "Ошибка", "Не удалось добавить долг.");

}

} catch (const CustomException &e) {

QMessageBox::critical(this, "Ошибка", "Не удалось добавить долг: " + QString::fromStdString(e.what()));

} catch (const std::exception &e) {

QMessageBox::critical(this, "Ошибка", "Неизвестная ошибка: " + QString::fromStdString(e.what()));

}

}

}

}

updateDebtSummary();

}

void MainWindow::updateDebtSummary() {

double totalDebt = 0.0;

QDate lastDueDate;

try {

auto debts = debtManager.getDebts();

for (const auto& debt : debts) {

totalDebt += debt.amount;

QDate dueDate = QDate::fromString(QString::fromStdString(debt.dueDate), "yyyy-MM-dd");

if (!dueDate.isValid()) {

qDebug() << "Некорректная дата:" << debt.dueDate;

continue;

}

if (!lastDueDate.isValid() || dueDate > lastDueDate) {

lastDueDate = dueDate;

}

}

totalDebtLabel->setText(QString("Общая сумма долга: %1").arg(totalDebt, 0, 'f', 2));

lastDueDateLabel->setText(QString("Последняя дата просрочки: %1").arg(lastDueDate.isValid() ? lastDueDate.toString("yyyy-MM-dd") : "нет"));

totalDebtLabel->lower();

lastDueDateLabel->lower();

showDailySavingsRecommendation();

} catch (const CustomException &e) {

QMessageBox::warning(this, "Ошибка", "Не удалось загрузить долги: " + QString::fromStdString(e.what()));

} catch (const std::exception &e) {

QMessageBox::warning(this, "Ошибка", "Неизвестная ошибка: " + QString::fromStdString(e.what()));

}

}

void MainWindow::showDailySavingsRecommendation() {

auto debts = debtManager.getDebts();

if (debts.empty()) {

QMessageBox::information(this, "Информация", "Нет долгов для расчета.");

return;

}

totalDebt = 0.0;

lastDueDate = "";

for (const auto& debt : debts) {

totalDebt += debt.amount;

if (lastDueDate.isEmpty() || QString::fromStdString(debt.dueDate) < lastDueDate) {

lastDueDate = QString::fromStdString(debt.dueDate);

}

}

if (lastDueDate.isEmpty()) {

QMessageBox::information(this, "Информация", "Нет долгов для расчета.");

return;

}

QDate due = QDate::fromString(lastDueDate, "yyyy-MM-dd");

if (!due.isValid()) {

QMessageBox::warning(this, "Ошибка", "Некорректная дата погашения.");

return;

}

QDate today = QDate::currentDate();

int daysLeft = today.daysTo(due);

QString recommendationText;

double dailyAmount = 0.0;

if (daysLeft > 0) {

dailyAmount = (totalDebt / daysLeft);

recommendationText = QString("Рекомендуемая сумма для откладывания ежедневно: %1")

.arg(dailyAmount, 0, 'f', 2);

} else if (daysLeft == 0) {

recommendationText = "Срок погашения сегодня.";

} else {

recommendationText = "Срок погашения уже прошел.";

}

QLabel\* recommendationLabel = savingsWidget->findChild<QLabel\*>();

if (!recommendationLabel) {

recommendationLabel = new QLabel(recommendationText, savingsWidget);

recommendationLabel->setAlignment(Qt::AlignCenter);

recommendationLabel->setStyleSheet("font-size: 18px; color: white;");

QVBoxLayout\* layout = new QVBoxLayout(savingsWidget);

layout->addWidget(recommendationLabel);

savingsWidget->setLayout(layout);

} else {

recommendationLabel->setText(recommendationText);

}

savingsWidget->lower();

}

void MainWindow::onDeleteAllDebts() {

if (QMessageBox::warning(this, "Удаление всех долгов", "Вы уверены, что хотите удалить все долги?", QMessageBox::Yes | QMessageBox::No) == QMessageBox::Yes) {

if (debtManager.deleteAllDebts()) {

QMessageBox::information(this, "Успех", "Все долги успешно удалены.");

updateDebtSummary();

loadDebts();

} else {

QMessageBox::critical(this, "Ошибка", "Не удалось удалить все долги.");

}

}

}

void MainWindow::openConverter() {

if (!currencyConverterWindow) {

QString apiKey = "b755d25aa24d5b03f7f924a0d974b61d";

currencyConverterWindow = new CurrencyConverterWindow(apiKey, this);

}

currencyConverterWindow->exec();

}

void MainWindow::showExpenseWarning() {

QString budgetSummary = financeService.getBudgetSummary();

QStringList lines = budgetSummary.split("\n");

double balance = lines[4].split(": ")[1].toDouble();

if (balance < 0) {

ui->warningLabel->setText("Внимание: расходы превышают доходы!");

ui->warningLabel->setStyleSheet("color: #ffcccc; font-weight: bold;");

ui->warningLabel->show();

} else {

ui->warningLabel->hide();

}

}

void MainWindow::setInputDialogStyle(QInputDialog &dialog) {

dialog.setStyleSheet(R"(

QInputDialog {

background-color: #1e2734;

}

QLineEdit {

background-color: white;

color: black;

}

QLabel {

color: black;

}

QPushButton {

background-color: #0056b3;

color: white;

border: none;

border-radius: 5px;

padding: 8px 12px;

}

QPushButton:hover {

background-color: #004494;

}

QPushButton:pressed {

background-color: #003366;

}

)");

}

void MainWindow::openCalculator() {

DepositWindow \*depositWindow = new DepositWindow(this);

depositWindow->exec();

}

void MainWindow::openSlots() {

SlotMachineDialog \*slotDialog = new SlotMachineDialog(&financeService, this);

connect(slotDialog, &SlotMachineDialog::budgetUpdated, this, &MainWindow::updateBudgetSummary);

slotDialog->exec();

}

Файл ReportService.h

#ifndef REPORTSERVICE\_H

#define REPORTSERVICE\_H

#include "FinanceService.h"

#include <string>

#include <fstream>

#include <iostream>

#include <vector>

#include <memory>

#include <string>

class ReportService {

public:

ReportService(const FinanceService& financeService);

void generateJsonReport(const std::string& reportType) const;

void generateDescriptionReport(const std::string& description) const;

private:

const FinanceService& financeService;

std::vector<FinanceEntry\*> filterByDescription(const std::string& description) const;

};

#endif

Файл ReportService.cpp

#include "ReportService.h"

ReportService::ReportService(const FinanceService& financeService)

: financeService(financeService) {}

void ReportService::generateJsonReport(const std::string& reportType) const {

std::vector<std::unique\_ptr<FinanceEntry>> records = financeService.getRecords();

std::string jsonContent = "{\n \"reportType\": \"" + reportType + "\",\n \"records\": [\n";

bool hasRecords = false;

if (reportType == "Бюджет") {

QString budgetSummary = financeService.getBudgetSummary();

jsonContent += " {\n \"summary\": \"" + budgetSummary.toStdString() + "\"\n }\n";

jsonContent += " ]\n}";

std::ofstream outFile("Бюджет\_Отчет.json");

if (outFile.is\_open()) {

outFile << jsonContent;

outFile.close();

std::cout << "Отчет сохранен в Бюджет\_Отчет.json" << std::endl;

} else {

std::cerr << "Не удалось открыть файл для записи." << std::endl;

}

return;

}

for (const auto& record : records) {

if (record && ((reportType == "Доходы" && record->getType() == "Income") ||

(reportType == "Расходы" && record->getType() == "Expense"))) {

jsonContent += " {\n";

jsonContent += " \"id\": " + std::to\_string(record->getId()) + ",\n";

jsonContent += " \"description\": \"" + record->getDescription() + "\",\n";

jsonContent += " \"amount\": " + std::to\_string(record->getAmount()) + "\n";

jsonContent += " },\n";

hasRecords = true;

}

}

if (hasRecords) {

jsonContent.pop\_back();

jsonContent.pop\_back();

}

jsonContent += " ]\n}";

if (!hasRecords) {

jsonContent = "{\n \"reportType\": \"" + reportType + "\",\n \"message\": \"Нет записей для данного типа.\"\n}";

}

std::ofstream outFile(reportType + "\_Отчет.json");

if (outFile.is\_open()) {

outFile << jsonContent;

outFile.close();

std::cout << "Отчет сохранен в " << reportType << "\_Отчет.json" << std::endl;

} else {

std::cerr << "Не удалось открыть файл для записи." << std::endl;

}

}

void ReportService::generateDescriptionReport(const std::string& description) const {

std::vector<std::unique\_ptr<FinanceEntry>> records = financeService.getRecords();

std::string jsonContent = "{\n \"description\": \"" + description + "\",\n \"records\": [\n";

bool hasRecords = false;

for (const auto& record : records) {

if (record && record->getDescription().find(description) != std::string::npos) {

jsonContent += " {\n";

jsonContent += " \"id\": " + std::to\_string(record->getId()) + ",\n";

jsonContent += " \"description\": \"" + record->getDescription() + "\",\n";

jsonContent += " \"amount\": " + std::to\_string(record->getAmount()) + "\n";

jsonContent += " },\n";

hasRecords = true;

}

}

if (hasRecords) {

jsonContent.pop\_back();

jsonContent.pop\_back();

}

jsonContent += " ]\n}";

if (!hasRecords) {

jsonContent = "{\n \"description\": \"" + description + "\",\n \"message\": \"Нет записей для данного описания.\"\n}";

}

std::ofstream outFile("Description\_Report.json");

if (outFile.is\_open()) {

outFile << jsonContent;

outFile.close();

std::cout << "Отчет сохранен в Description\_Report.json" << std::endl;

} else {

std::cerr << "Не удалось открыть файл для записи." << std::endl;

}

}

Файл SlotMachine.h

#ifndef SLOTMACHINE\_H

#define SLOTMACHINE\_H

#include <QDialog>

#include <QLabel>

#include <QPushButton>

#include <QVBoxLayout>

#include <QHBoxLayout>

#include <QRandomGenerator>

#include <QPropertyAnimation>

#include <QSpinBox>

#include "financeservice.h"

#include "Budget.h"

#include <sqlite3.h>

#include "MainWindow.h"

class SlotMachineDialog : public QDialog {

Q\_OBJECT

public:

explicit SlotMachineDialog(FinanceService \*financeService, QWidget \*parent = nullptr);

double balance;

protected:

void closeEvent(QCloseEvent \*event) override;

private slots:

void spin();

private:

void animateSymbol(QLabel \*label);

QList<QLabel\*> symbolLabels;

QLabel \*resultLabel;

MainWindow \*mainWindow;

QLabel \*netResultLabel;

QPushButton \*spinButton;

QSpinBox \*betSpinBox;

FinanceService \*financeService;

Budget \*budget;

double currentBet;

double totalWinnings;

double totalLosses;

double netResult;

signals:

void budgetUpdated();

};

#endif

Файл SlotMachine.cpp

#include "SlotMachine.h"

#include "DatabaseService.h"

#include "CustomException.h"

SlotMachineDialog::SlotMachineDialog(FinanceService \*financeService, QWidget \*parent)

: QDialog(parent), financeService(financeService), currentBet(0), totalWinnings(0), totalLosses(0), netResult(0) {

setWindowTitle("Слот Казино");

resize(300, 200);

DatabaseService &dbService = DatabaseService::getInstance();

if (!dbService.openDatabase("Records.db")) {

throw CustomException("Не удалось открыть базу данных!");

}

QVBoxLayout \*layout = new QVBoxLayout(this);

QHBoxLayout \*symbolLayout = new QHBoxLayout();

for (int i = 0; i < 3; ++i) {

QLabel \*symbolLabel = new QLabel("💎", this);

symbolLabel->setFont(QFont("Arial", 40));

symbolLabels.append(symbolLabel);

symbolLayout->addWidget(symbolLabel);

}

layout->addLayout(symbolLayout);

balance = financeService->getBalance();

resultLabel = new QLabel("Баланс: " + QString::number(balance), this);

layout->addWidget(resultLabel);

netResultLabel = new QLabel("Результат: 0", this);

layout->addWidget(netResultLabel);

betSpinBox = new QSpinBox(this);

betSpinBox->setRange(1, 100);

betSpinBox->setValue(1);

betSpinBox->setStyleSheet(

"QSpinBox {"

" font-size: 10px;"

" color: white;"

" background-color: #404040;"

" border: 1px solid #007BFF;"

" border-radius: 5px;"

" padding: 2px;"

"}"

"QSpinBox::up-button, QSpinBox::down-button {"

" background-color: #007BFF;"

" border: none;"

" border-radius: 5px;"

"}"

"QSpinBox::up-button:hover, QSpinBox::down-button:hover {"

" background-color: #0056b3;"

"}"

"QSpinBox::up-button:pressed, QSpinBox::down-button:pressed {"

" background-color: #004494;"

"}"

"QLineEdit {"

" font-size: 10px;"

" color: white;"

" background-color: #404040;"

" border: none;"

"}"

"QLineEdit:focus {"

" border: 1px solid #0056b3;"

"}"

"QLineEdit::placeholder {"

" color: #aaaaaa;"

" font-style: italic;"

"}"

);

layout->addWidget(betSpinBox);

spinButton = new QPushButton("Крутить", this);

layout->addWidget(spinButton);

spinButton->setStyleSheet(

"QPushButton {"

" font-size: 14px;"

" background-color: #007BFF;"

" color: white;"

" border: none;"

" border-radius: 5px;"

" padding: 10px;"

"}"

"QPushButton:hover {"

" background-color: #0056b3;"

"}"

"QPushButton:pressed {"

" background-color: #004494;"

" padding-left: 12px;"

" padding-top: 12px;"

"}"

);

connect(spinButton, &QPushButton::clicked, this, &SlotMachineDialog::spin);

}

void SlotMachineDialog::spin() {

currentBet = betSpinBox->value();

if (balance < currentBet) {

resultLabel->setText("Недостаточно средств для ставки!");

return;

}

netResult = 0.0;

balance -= currentBet;

totalLosses += currentBet;

for (int i = 0; i < 3; ++i) {

animateSymbol(symbolLabels[i]);

}

}

void SlotMachineDialog::animateSymbol(QLabel \*label) {

QPropertyAnimation \*animation = new QPropertyAnimation(label, "pos");

animation->setDuration(1000);

animation->setStartValue(label->pos());

animation->setEndValue(QPoint(label->pos().x(), label->pos().y() - 100));

connect(animation, &QPropertyAnimation::finished, [this, label]() {

QStringList symbols = {"💎", "🧊", "🥏"};

QString newSymbol = symbols[QRandomGenerator::global()->bounded(symbols.size())];

label->setText(newSymbol);

label->move(label->pos().x(), label->pos().y() + 100);

if (symbolLabels[0]->text() == symbolLabels[1]->text() && symbolLabels[1]->text() == symbolLabels[2]->text()) {

double winnings = 2 \* (currentBet/3);

balance += winnings;

totalWinnings += winnings;

netResult += winnings;

} else {

netResult -= currentBet/3;

}

resultLabel->setText(QString("Баланс: %1").arg(balance));

netResultLabel->setText(QString("Выигрыш: %1").arg(netResult));

if (netResult > 0) {

resultLabel->setText("Вы выиграли! " + resultLabel->text());

} else if (netResult < 0) {

resultLabel->setText("Вы проиграли. " + resultLabel->text());

}

});

animation->start(QAbstractAnimation::DeleteWhenStopped);

}

void SlotMachineDialog::closeEvent(QCloseEvent \*event) {

double oldBalance = financeService->getBalance();

financeService->updateBalance(balance - oldBalance);

emit budgetUpdated();

QDialog::closeEvent(event);

}

Файл Stack.h

#ifndef STACK\_H

#define STACK\_H

#include <vector>

#include <stdexcept>

#include <memory>

#include "StackIterator.h"

template<typename T>

class Stack {

public:

Stack() = default;

void push(T value) {

data.push\_back(std::move(value));

}

void pop() {

if (isEmpty()) {

throw std::underflow\_error("Стек пуст, невозможно удалить элемент.");

}

data.pop\_back();

}

T& top() {

if (isEmpty()) {

throw std::underflow\_error("Стек пуст, невозможно получить верхний элемент.");

}

return data.back();

}

const T& top() const {

if (isEmpty()) {

throw std::underflow\_error("Стек пуст, невозможно получить верхний элемент.");

}

return data.back();

}

bool isEmpty() const {

return data.empty();

}

size\_t size() const {

return data.size();

}

void clear() {

data.clear();

}

using Iterator = StackIterator<T>;

Iterator begin() {

return Iterator(data.rbegin());

}

Iterator end() {

return Iterator(data.rend());

}

private:

std::vector<T> data;

};

#endif

Файл StackIterator.h

#ifndef STACKITERATOR\_H

#define STACKITERATOR\_H

#include <vector>

template<typename T>

class StackIterator {

public:

StackIterator(typename std::vector<T>::reverse\_iterator it) : current(it) {}

T& operator\*() {

return \*current;

}

StackIterator& operator++() {

++current;

return \*this;

}

bool operator!=(const StackIterator& other) const {

return current != other.current;

}

private:

typename std::vector<T>::reverse\_iterator current;

};

#endif

Файл WelcomeWindow.h

#ifndef WELCOMEWINDOW\_H

#define WELCOMEWINDOW\_H

#include <QDialog>

#include <QLabel>

#include "ui\_WelcomeWindow.h"

class WelcomeWindow : public QDialog {

Q\_OBJECT

public:

explicit WelcomeWindow(QWidget \*parent = nullptr);

~WelcomeWindow();

private slots:

void onStartButtonClicked();

signals:

void start();

private:

Ui::WelcomeWindow \*ui;

};

#endif

Файл WelcomeWindow.cpp

#include "WelcomeWindow.h"

WelcomeWindow::WelcomeWindow(QWidget \*parent)

: QDialog(parent), ui(new Ui::WelcomeWindow)

{

ui->setupUi(this);

setFixedSize(650, 400);

connect(ui->startButton, &QPushButton::clicked, this, &WelcomeWindow::onStartButtonClicked);

}

WelcomeWindow::~WelcomeWindow() {

delete ui;

}

void WelcomeWindow::onStartButtonClicked() {

emit start();

accept();

}