Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина «Операционные системы и среды»

«К ЗАЩИТЕ ДОПУСТИТЬ»
Руководитель курсового проекта
ассистент кафедры Информатики
_____В. Д. Владымцев
___.____.2023

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
на тему:

«ПРОСТОЙ ФАЙЛОВЫЙ МЕНЕДЖЕР ДЛЯ WINDOWS 10»

БГУИР КП 1-40 04 01 020 ПЗ

Выполнил студент группы 053505
Слуцкий Никита Сергеевич

_____
(подпись студента)

Курсовой проект представлен на проверку ___.____.2023

_____
(подпись студента)

Минск 2023

СОДЕРЖАНИЕ

Введение 5
1 Архитектура программного продукта 6
1.1 Структура и архитектура операционной системы 6
1.2 Краткая история ОС и версии 7
1.3 Обоснование выбора 7
2 Платформа программного обеспечения 8
2.1 GNU и MinGW 8
2.2 JetBrains Clion 8
2.3 Windows API 9
3 Теоретическое обоснование разработки программного продукта 11
3.1 FAR Manager 11
3.2 Используемые технологии программирования 12
3.2.1 Раздельный рендеринг компонентов. Псевдоподход из React 12
3.2.2 Хранение состояния приложения 13
3.2.3 Прослушивание событий 14
3.3 Некоторые принципы, на которых основана разработка 15
3.3.1 DRY – Don't Repeat Yourself or DIE – Duplication Is Evil 16
3.3.2 KISS – keep it short simple / keep it simple, stupid 16
3.3.3 YAGNI – You ain't gonna need it 16
3.3.4 Комментарии 16
3.3.5 Именование сущностей 16
4 Проектирование функциональных возможностей программы 17
4.1 Описание сущностей и общей схемы жизненного цикла 17
4.2 Состояние приложения AppState 18
4.3 Псевдографический интерфейс GUI 18
4.4 Прослушка событий 20
5 Архитектура разрабатываемой программы 22
Заключение 23
Список используемой литературы 24
Приложение А (обязательное) Листинг кода 26
Приложение Б (обязательное) Функциональная схема 55
Приложение В (обязательное) Блок схема алгоритма 56

Приложение Г (обязательное) Графический интерфейс 57
Приложение Д (обязательное) Ведомость документов 58

ВВЕДЕНИЕ

Целью данного курсового проекта ставится попытка научить стандартную консоль операционной системы Windows 10 полноценно работать с ограниченным выбранным набором пользовательских событий, производимых с помощью клавиатуры или компьютерной мыши. Для того, чтобы данная цель имела относительную практическую ценность для потенциального пользователя будущего программного продукта, на базе этой модифицированной консоли будет разработан простой файловый менеджер, позволяющий просматривать каталоги, открывать файлы программами по умолчанию, создавать новые папки и файлы, а также удалять существующие.

Соответственно, в итоге должен получиться консольный файловый менеджер, чьё поведение должно напоминать взаимодействие с полноценным оконным приложением.

В ходе разработки дополнительными целями ставится:

– разделение ответственности в коде;
– создание аналогии с веб-разработкой на стеке React + MobX;
– обеспечение предыдущего пункта за счёт введения отдельных сущностей для хранилища данных, отлавливания событий и отрисовки интерфейса.

Исходя из вышеописанных целей задачами ставятся:

– разработка программного продукта на языке программирования C++;
– использование компилятора GNU для компиляции и линковки;
– изучение и использование библиотеки windows.h;
– проектирование «дружелюбного» интерфейса пользователя;
– создание условий для того, чтобы такие события, как скролл, клик, нажатие на клавиатуру, корректно и ожидаемо отрабатывали.

1 АРХИТЕКТУРА ПРОГРАММНОГО ПРОДУКТА

1.1 Структура и архитектура операционной системы

Windows – одна из наиболее многогранных и гибких ОС, она работает на совершенно разных архитектурах и доступна в разных вариантах [1]. На сегодня она поддерживает архитектуры x86, x64, ARM и ARM64. Windows в своё время поддерживала Itanium, PowerPC, DEC Alpha и MIPS. Кроме того, Windows поддерживает целый набор SKU, работающих в различных условиях; от дата-центров, ноутбуков, Xbox и телефонов до встраиваемых версий для интернета вещей, например, в банкоматах.

Самый удивительный аспект состоит в том, что ядро Windows практически не меняется в зависимости от всех этих архитектур и SKU. Ядро динамически масштабируется в зависимости от архитектуры и процессора, на котором оно работает, так, чтобы пользоваться всеми возможностями оборудования. Конечно, в ядре присутствует определённое количество кода, связанного с конкретной архитектурой, однако его там минимальное количество, что позволяет Windows запускаться на разнообразных архитектурах.

Архитектура операционной системы Windows представляет собой слоистый дизайн, состоящий из двух основных компонентов: режима пользователя и режима ядра. Режим пользователя содержит приложения и подсистемы, которые предоставляют услуги, такие как графический пользовательский интерфейс, сетевые, веб-службы и т.д. Режим ядра содержит ядро операционной системы, такие как ядро, аппаратный уровень абстракции, драйверы и исполнительные службы, которые управляют процессами, потоками, памятью, безопасностью и т.д. Ядро Windows NT является гибридным ядром, которое сочетает в себе особенности микроядерной и монолитной архитектур.

Первые версии Windows не были полноценными операционными системами, а являлись надстройками над операционной системой DOS и были по сути многофункциональным расширением, добавляющим поддержку новых режимов работы процессора, поддержку многозадачности, обеспечивали стандартизацию интерфейсов аппаратного обеспечения, обмен данными между приложениями и единообразие пользовательских интерфейсов программ. Для создания графического интерфейса использовались встроенные средства GDI и USER. Первые версии Windows

вообще состояли из трёх модулей – KERNEL, GDI и USER. Первый из них обеспечивал управление памятью, запуск исполняемых файлов и загрузку динамических библиотек DLL, второй отвечал за графику, третий – за окна. Они работали с процессорами начиная с Intel 8086.

1.2 Краткая история ОС и версии

Windows – это группа семейств проприетарных операционных систем корпорации Microsoft, ориентированных на управление с помощью графического интерфейса. Они пришли на смену MS-DOS, текстовой однозадачной операционной системе.

Первая независимая версия Windows, Windows 1.0, была выпущена 20 ноября 1985 года и не получила большой популярности. Она была всего лишь графической программой-надстройкой для MS-DOS и имела ограниченные возможности.

Последней на данный момент операционной системой Microsoft является Windows 11, представленная 24 июня 2021 года. Она имеет новый дизайн интерфейса, поддержку приложений Android и улучшенную производительность.

1.3 Обоснование выбора

Ввиду распространённости выбранной операционной системы среди как разработчиков, так и пользователей было принято решение разработки программного продукта именно под эту операционную систему. WinAPI – комплекс процедур для взаимодействия с системой, хорошо представлен на различных интернет-ресурсах, имеет хорошую документацию и обширную базу знаний в сети интернет. В связи с вышеописанными факторами и тем фактом, что на используемой рабочей машине установлены ОС Windows 10 и Linux Ubuntu 20, и было принято решение разрабатывать приложение под первую из упомянутых установленных операционных систем.

2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В используемом рабочем компьютере используется архитектура на основе процессора AMD Ryzen 7 4800H (7 nm). В качестве ОС выступает OS Windows 10. Для разработки описанного выше программного решения требуется компилятор, линковщик и среда разработки (IDE). Компилятором выступает GNU с поддержкой современного стандарта языка программирования C++ – C++ 17. В качестве инструмента для написания программного кода была выбрана JetBrains Clion Ide.

2.1 GNU и MinGW

GNU компилятор для C++ – это один из компиляторов языка программирования C++, который входит в состав GNU Compiler Collection (GCC) [2]. GCC – это набор компиляторов и библиотек для разных языков программирования, разработанный проектом GNU. GNU – это операционная система, созданная для того, чтобы быть 100% свободным программным обеспечением, уважающим свободу пользователей.

GNU компилятор C++ может работать на разных платформах и операционных системах, таких как Linux, Windows, Mac OS и других. Он поддерживает разные стандарты C++, такие как C++98, C++11, C++14 и C++17. Он также имеет множество опций и расширений для настройки процесса компиляции.

MinGW для Windows – это набор инструментов для компиляции и запуска программ на языке C или C++ в среде Windows [3]. MinGW означает Minimalist GNU for Windows, то есть минимальный набор GNU для Windows.

MinGW позволяет создавать нативные приложения для Windows, используя стандарты и расширения языков C и C++. MinGW включает в себя порт компилятора GCC (GNU Compiler Collection) для Windows, а также набор заголовочных файлов и библиотек для работы с Win32 API.

Для того, чтобы использовать MinGW для Windows, необходимо скачать и установить его на компьютер.

2.2 JetBrains Clion

JetBrains CLion – это кросс-платформенная интегрированная среда разработки (IDE) для языков программирования C и C++ [4]. Она предоставляет умный редактор кода, который понимает синтаксис и семантику C и C++, а также поддерживает разные стандарты и библиотеки.

CLion также имеет множество других функций, которые помогают разработчикам писать качественный и эффективный код. Например, CLion предлагает:

- автоматическое генерирование и рефакторинг кода;
- анализ кода на лету и исправление потенциальных ошибок;
- отладчик с поддержкой GDB или LLDB;
- поддержку разных систем сборки, таких как CMake, Makefile, Gradle и других;
- поддержку разных инструментов для тестирования, профилирования и статического анализа кода;
- поддержку разных плагинов для расширения функциональности IDE.

Для того, чтобы использовать JetBrains CLion для разработки настоящего образовательного курсового проекта, необходимо получить образовательную лицензию на использование этого программного обеспечения, скачать и установить его на компьютер.

2.3 Windows API

Windows API – общее наименование набора базовых функций интерфейсов программирования приложений операционных систем семейств Microsoft Windows корпорации «Майкрософт» [5]. Предоставляет прямой способ взаимодействия приложений пользователя с операционной системой Windows. Для создания программ, использующих Windows API, корпорация «Майкрософт» выпускает комплект разработчика программного обеспечения, который называется Platform SDK и содержит документацию, набор библиотек, утилит и других инструментальных средств для разработки.

Windows API спроектирован для использования в языке Си для написания прикладных программ, предназначенных для работы под управлением операционной системы MS Windows. Работа через Windows API – это наиболее близкий к операционной системе способ взаимодействия с ней из прикладных программ. Более низкий уровень доступа, необходимый только для драйверов устройств, в текущих версиях Windows предоставляется через Windows Driver Model.

Windows API представляет собой множество функций, структур данных и числовых констант, следующих соглашениям языка Си. В то же время конвенция вызова функций отличается от cdecl, принятой для языка C: Windows API использует stdcall (winapi). Все языки программирования, способные вызывать такие функции и оперировать такими типами данных в

программах, исполняемых в среде Windows, могут пользоваться этим API. В частности, это языки C++, Pascal, Visual Basic и многие другие.

Ниже вкратце описаны основные версии WinAPI.

Win16 – первая версия WinAPI для 16-разрядных версий Windows. Изначально назывался Windows API, позднее был ретроспективно переименован в Win16 для отличия от Win32. Описан в стандарте ECMA-234.

Win32 – 32-разрядный API для современных версий Windows. Самая популярная на данный момент версия. Базовые функции реализованы в динамически подключаемых библиотеках kernel32.dll и advapi32.dll. Базовые модули графического интерфейса пользователя реализованы в user32.dll и gdi32.dll. Win32 появился вместе с Windows NT и затем был перенесён в несколько ограниченном виде в системы серии Windows 9x. В современных версиях Windows, происходящих от Windows NT, работу Win32 GUI обеспечивают два модуля: csrss.exe (процесс исполнения клиент-сервер), работающий в пользовательском режиме, и win32k.sys в режиме ядра. Работу же системы обеспечивает ядро – ntoskrnl.exe.

Win32s – подмножество Win32, устанавливаемое на семейство 16-разрядных систем Windows 3.x и реализующее ограниченный набор функций Win32 для этих систем.

Win64 – 64-разрядная версия Win32, содержащая дополнительные функции Windows на платформах x86-64 и IA-64.

Win32 API – это набор функций Windows на все случаи жизни. Вызывая функции WinAPI, можно добиться желаемого поведения системы: создания окон и другие графических объектов, взаимодействия с подключенными устройствами, выполнения обработки данных, работы с сетью, безопасностью и так далее. WinAPI – это «переходник» между программой и операционной системой, то есть теми возможностями, которые она предоставляет. С помощью WinAPI можно создавать различные оконные процедуры, диалоговые окна, программы и даже игры. Эта «библиотека» является базовой в освоении программирования Windows.

3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

С точки зрения реальной практической пользы разработка представляет интерес лишь в образовательных целях. В современном мире у вычислительных машин достаточно производительности, чтобы обеспечивать комфортную работу в проводнике или других оконных файловых менеджерах.

Тем не менее большой популярностью пользуется консольный файловый менеджер FAR, по подобию которого и создаётся вышеописанный разрабатываемый программный продукт.

3.1 FAR Manager

Одним из самых популярных и актуальных примеров хорошего и очень функционального консольного приложения является FAR Manager [6]. FAR Manager – консольный файловый менеджер для операционных систем семейства Microsoft Windows. С 2000 года разработкой FAR Manager занимается группа FAR Group. Он работает в текстовом (то есть консольном) режиме и благодаря этому обеспечивает очень простой и интуитивный интерфейс для совершения ряда базовых операций:

- просмотр файлов и директорий;
- редактирование, копирование;
- переименование и перемещение файлов;
- многое другое.

Программа FAR Manager написана на языке программирования C++. Она является отличным примером того, как можно максимально приблизить стандартную консоль Windows по внешнему виду к обыкновенному десктопному приложению с графическим интерфейсом, а также как можно реализовать работу с событиями (компьютерной мышью и клавиатурой) для обеспечения максимально комфортного опыта использования приложения.

Итак, автором данной курсовой работы была поставлена цель провести все необходимые модификации стандартной консоли C++ для возможности создания приложения, приближенного по уровню пользовательского опыта взаимодействия к вышеописанному FAR Manager.

Far Manager имеет многоязычный, легко настраиваемый интерфейс. Простую навигацию по файловой системе обеспечивают цветовое выделение и группы сортировки файлов.

Функциональность Far Manager существенно расширяется за счет внешних подключаемых DLL-модулей – плагинов (этому способствует набор

специальных интерфейсов – Plugins API). Например, работа с архивами, FTP-клиент, временная панель и просмотр сети реализованы с помощью плагинов, включенных в стандартную поставку Far Manager. На рисунке 1 представлен скриншот.

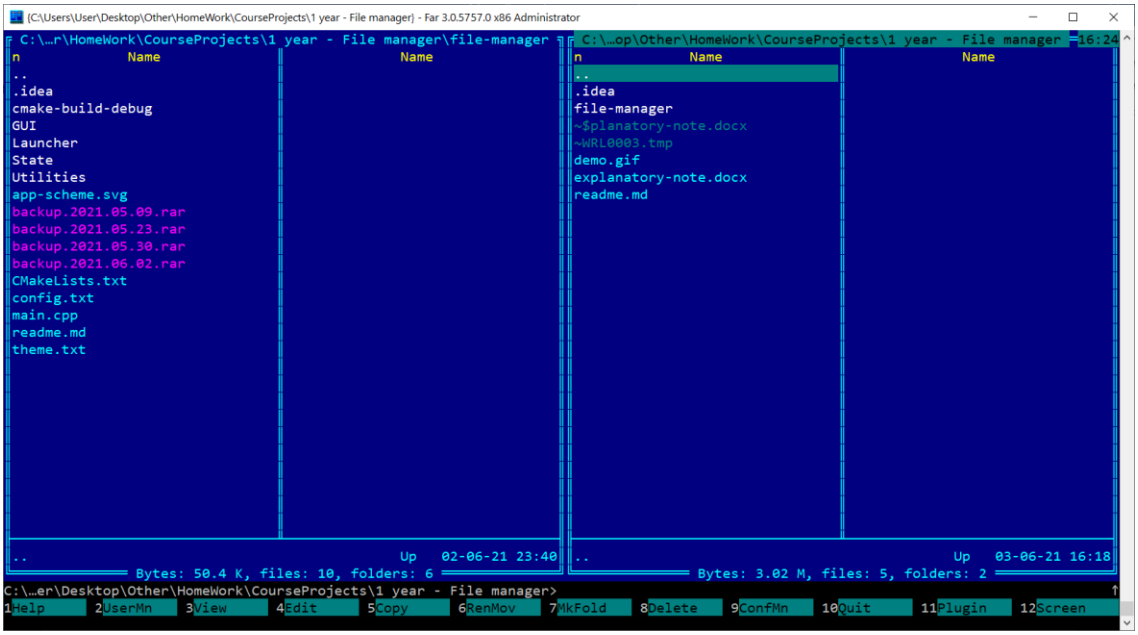

Рисунок 1 – Скриншот из приложения FAR

3.2 Используемые технологии программирования

Ниже будут описаны несколько концепций, взятых из клиентского веб-программирования, на которые будет опираться подход к разработке программного продукта.

3.2.1 Раздельный рендеринг компонентов. Псевдоподход из React

Почти любое приложение и в частности его графическую составляющую можно рассматривать как совокупность некоторых сущностей, которые при отрисовке никак не зависят друг от друга, но при изменении себя влекут изменения других компонентов. Это представляет собой разбиение на компоненты и их отрисовку только по необходимости. Такой подход отражается на производительности приложения, так как не требует постоянной перерисовки статических или неизменившихся частей интерфейса. И приложение не являет собой единственную сущность интерфейса со всеми компонентами в одном, а построено на модульности.

Именно такая концепция используется в популярной JavaScript-библиотеке React.

React.js – это сторонняя библиотека языка JavaScript, созданная для разработки интерактивных пользовательских интерфейсов [7]. Благодаря этой библиотеке развертывание веб-приложений и интерактивных UI-интерфейсов значительно ускоряется (по сравнению с программированием на «ванильном» JS). Разработчику на React необходимо писать гораздо меньше кода, чем если бы он пытался запрограммировать интерфейс только на стандартном JavaScript. Чтобы задать функциональность компоненту веб-приложения в React – достаточно описать то, как будут выглядеть определённые части интерфейса в различных состояниях. Изменять существующий код, при этом, не обязательно. Кроме того, логика описывается на стандартном JS, а не в шаблонах, поэтому можно передавать любые данные по всему веб-приложению. На рисунке 2 представлен пример разбиения страницы на компоненты.

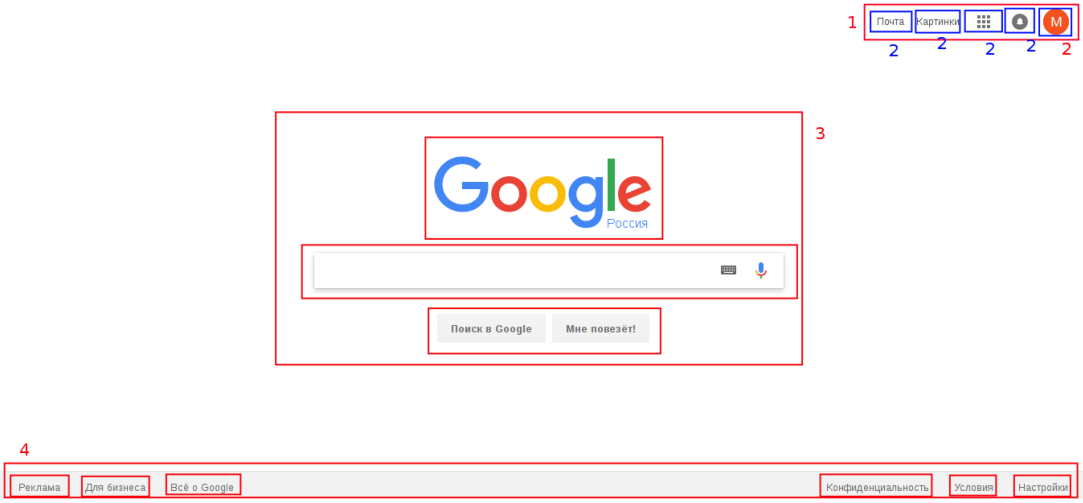

Рисунок 2 – Пример разбиения на компоненты

3.2.2 Хранение состояния приложения

Состояние приложения – это просто состояние, в котором находится программа относительно того, где и как она выполняется в данный момент. Например, в игре это может быть текущее положение игрока, текущий счёт и так далее. Общепринятым считается хранение состояния приложения отдельно от сущностей для его отрисовки. Компоненты приложения имеют частичный или полный доступ к полям состояния и, в зависимости от этого

состояния, могут по-разному отрисовываться. На рисунке 3 изображена упрощённая схема доступа нескольких компонентов к одному общему состоянию Состояние всего приложения хранится в едином хранилище. Единственный способ изменить состояние – вызвать соответствующий метод у хранилища с необходимым описанием.

Сущностями для хранения и изменения хранилищ в клиентском программировании являются просто классы, сервисы или объекты. Подход можно реализовать за счёт паттернов observer, proxy или ручной перерисовки.

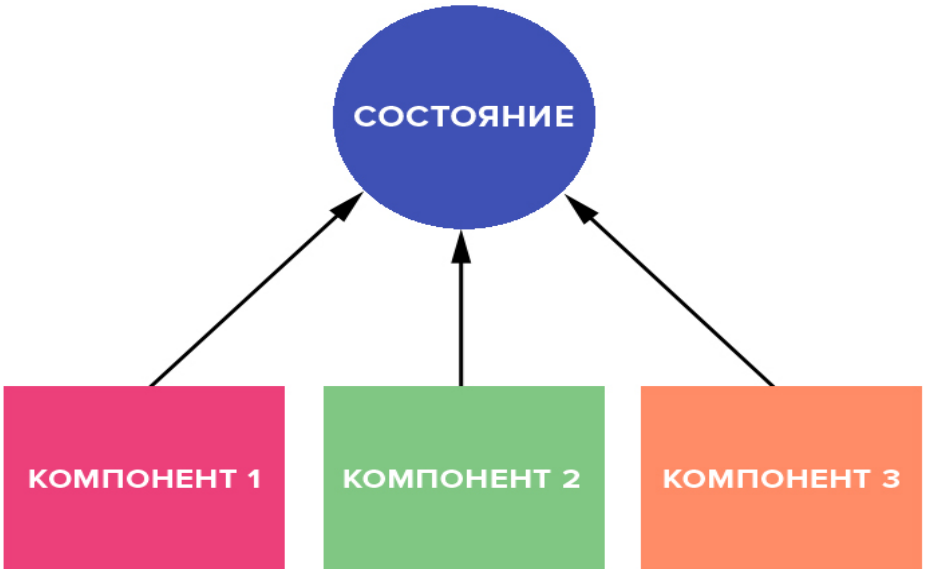

Рисунок 3 – Доступ компонентов к общему стору

3.2.3 Прослушивание событий

Одной из основных частей приложения является обработка пользовательских событий. Подразумевается некоторая реакция программы на те или иные действия, совершаемые пользователем. Например, действия компьютерной мышкой, трекпадом или клавиатурой. Подобные события должны постоянно отлавливаться или, как это называют, прослушиваться, обрабатываться, а пользователь должен видеть какую-то отдачу от своего действия. В большинстве программ, которые пишутся с нуля, необходимо уметь отлавливать базовые события, такие как:

– единичный клик левой кнопки мыши;
– двойной клик левой кнопки мыши;
– прокрутка колёсика мыши;
– нажатие клавиш на клавиатуре;

– другие по необходимости и желанию.

Клики мышью могут быть использованы при взаимодействиях с модальными окнами, кнопками и другими компонентами интерфейса. Нажатия клавиш на клавиатуре могут быть использованы при навигации, вводе текста и фильтре этого ввода. Прокрутка колёсика компьютерной мыши может быть использована для пролистывания списка картинок, увеличения / уменьшения масштаба карты и других целей. Также могут отлавливаться и другие события. Примеры таких событий это:

– изменение размеров окна;
– сворачивание и разворачивание окна;
– изменение положения окна;
– нажатие правой кнопки мыши;
– изменение размера (разрешения) экрана исходного устройства.

3.3 Некоторые принципы, на которых основана разработка

В соответствии с поставленной целью, функциональность программы будет максимально декомпозирована на независимые и неделимые в рамках математической подзадачи «методы», чтобы чётко следовать принципам функционального программирования, принципам единой ответственности, модульности и расширяемости. Поэтому после точки входа будут реализованы процедуры, соответствующие нуждам решения поставленной задачи. Каждая процедура может сопровождаться понятным пояснением насчёт:

– требуемое размещение входных параметров;
– размещение выходных параметров;
– затрагиваемые регистры при отработке процедуры;
– будет ли задействован стек при работе процедуры;
– затрагиваемые переменные для работы процедуры;
– другие пояснения по необходимости.

Ставится целью написать в том числе как можно более «чистый» код.

Легкоподдерживаемый читаемый код – то, к чему стремится любой опытный разработчик. Это код, который легко читать через 2 месяца, полгода, год и больше после его написания, причём не только автору, но и любому другому программисту. А так как в большинстве случаев код разрабатывается в командах – участники команды должны иметь возможность легко разбираться в кусочке приложения, не прилагая усилий, чтобы расшифровать написанную логику.

3.3.1 DRY – Don't Repeat Yourself or DIE – Duplication Is Evil

Принцип [8] призывает не повторяться при написании кода. При несоблюдении этого принципа программист будет вынужден вносить изменения в несколько повторяющихся фрагментов кода, вместо одного. Также дублирующийся код приводит к разрастанию программы, а значит, усложняет ее понимание, читабельность.

3.3.2 KISS – keep it short simple / keep it simple, stupid

Принцип KISS [9] подразумевает следующее. Чем проще код, тем легче в нём разобраться. Под простотой подразумевается отказ от использования хитроумных приемов и ненужного усложнения.

3.3.3 YAGNI – You ain't gonna need it

Всё, что не предусмотрено заданием проекта, не должно быть в нём.

3.3.4 Комментарии

Необходимо пояснять, комментировать код, где это возможно.

Комментарии могут использоваться для пояснения следующих моментов:

– задача кода;

– предпочтительность выбранного решения.

В то же время не стоит задача покрыть комментариями весь код. Использование значимых названий переменных и функций, разбитие кода на логические фрагменты с помощью функции и другие практики помогают сделать код максимально читаемым и понятным, не прибегая к комментариям (самодокументирующийся код).

3.3.5 Именование сущностей

Необходимо придерживаться единого стиля именования файлов в проекте. В рамках данного курсового проекта будет использоваться именование переменных и процедур буквами нижнего регистра с разделением в виде символа нижнего подчеркивания. Это так называемый стиль «Snake case». Это один из официальных подходов к именованию, диктуемый Google Code Style Guide [10].

4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

Создание программного продукта начато с базовой настройки проекта, прогнозирования необходимого набора переменных для хранения некоторых данных, возможно, набора необходимых для работы констант, списка используемых библиотек с учётом того, что программный продукт разрабатывается для работы в среде операционной системы Windows.

4.1 Описание сущностей и общей схемы жизненного цикла

В разрабатываемом приложении за основу взяты три основные сущности и одна дополнительная. Все они выделены в отдельные классы. Была организована правильная схема их взаимодействия таким образом, чтобы втроём они самостоятельно и независимо обеспечивали правильную работу приложения без возможности вмешательства из вне.

Итак, работу всего приложения обеспечивают три одновременно независимых, но при этом в какой-то степени взаимоконтролируемых, класса: AppState, GUI и EventsController. Ниже будут подробно описаны права и обязанности каждого из них. Ниже на рисунке 4 изображён жизненный цикл классов в совокупности друг с другом.

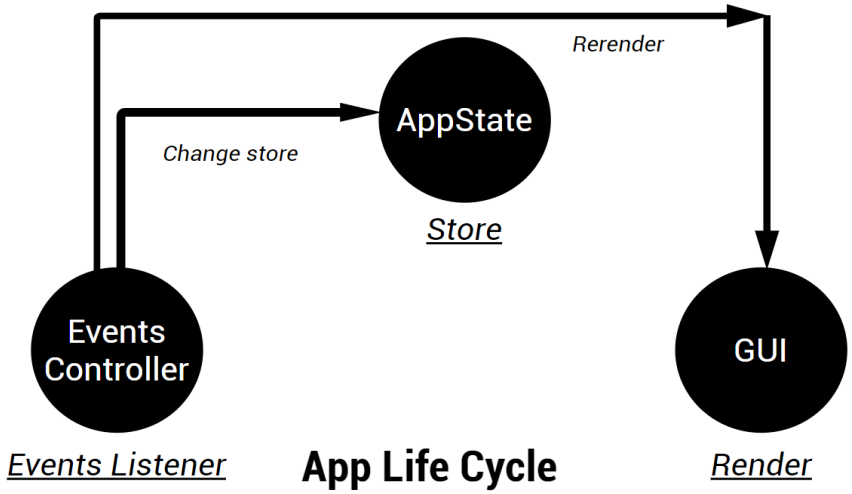

Рисунок 4 – Жизненный цикл приложения

Схему взаимодействия классов можно растолковать следующим образом. В приложении постоянно «крутится» и работает класс, отвечающий за прослушку событий. После обнаружения события этот класс выполняет с

ним определённые действия согласно прописанной логике, соответствующим образом при необходимости изменяет состояние приложения и, в случае его изменения, перерисовывает изменившуюся часть в графическом интерфейсе. Некоторые события происходят с помощью графического интерфейса – поэтому получается замкнутый круг.

Это и есть жизненный цикл приложения. Четвёртый класс, FileManagerLauncher, отвечает за стартовую настройку и конфигурацию приложения, запуск всех трёх вышеописанных сущностей и более не принимает участие в жизненном цикле программы.

4.2 Состояние приложения AppState

AppState – один из трёх главных классов в приложении. Он отвечает за хранение текущего состояния приложения. В случае данного проекта (файлового менеджера) это такие данные, как:

- путь к текущей директории;
- путь к родительской директории;
- список всех сущностей в текущей директории;
- список отрисованных на экране в данный момент сущностей из текущей директории;
- ассоциативный массив отрисованных на экране в данный момент сущностей и закреплённых за ними координат в интерфейсе консоли;
- стек с историей посещения папок.

Этот класс отвечает за все аспекты работы с файловой системой, которые могут понадобиться в данный момент времени. Это такая функциональность, как, например:

- возвращение в предыдущую директорию;
- получение списка сущностей в папке по её пути;
- вход в подпапку.

Со списком полей класса можно ознакомиться в листинге кода. Можно наблюдать одну из особенностей проекта – следование правильному официальному стилю написания кода – Google C++ Style Guide.

4.3 Псевдографический интерфейс GUI

GUI – самый обширный и масштабный класс в приложении. Он включает в себя всю функциональность по отрисовке компонентов в консоли. Перемещение по консоли, изменение цветов, установка размеров окна

осуществляется посредством WinAPI. Реализованы внутренние приватные методы:

– перемещение по координатам;
– установка тем оформления (загружается из файла);
– раскраска областей в консоли;
– отрисовка компонентов и их частей;
– другие необходимые служебные методы.

Реализован принцип единой ответственности. Каждый метод отвечает за что-то одно. Это может быть, например, только покраска фона. Или только рендеринг части какого-то компонента. Реализованы 2 основных компонента в GUI:

– Body;
– Footer.

В свою очередь они разбиты на дочерние компоненты. Их можно поделить на статические и динамические. В случае компонента Body это:

– верхняя граница (одна из них является статической);
– текущий путь вместе с кнопкой возврата к предыдущей директории (динамический);
– список файлов (динамический).

В случае Footer это:

– текущий путь (динамический);
– список горячих клавиш (статический).

Статические части компонентов рендерятся единожды при запуске приложения и более не перерисовываются. В классе присутствует много константных полей, задающих структуру интерфейса, положения компонентов и так далее.

Была реализована попытка создать псевдомодальные окна в приложении. За счёт плоскости и «однопоточности» интерфейса и возможностей консоли эти окна отрисовываются непосредственно поверх уже нарисованного контента, затирая его. После закрытия окна компонент, поверх которого был нарисован прямоугольник с окном, перерисовывается. В момент запуска окна вся текущая активность сосредотачивается на нём. Это значит, что события в основном окне игнорируются (файлы более не пролистываются, не открываются и так далее), а вводятся новые события для конкретного активного окна.

События для модального окна также предполагают взаимодействие как компьютерной мышью, так и клавиатурой. В разработанной курсовой работе имеются два псевдомодальных окна для создания папок или файлов и для

удаления сущностей в файловой системе. Внешний вид модального окна можно наблюдать на рисунке 5.

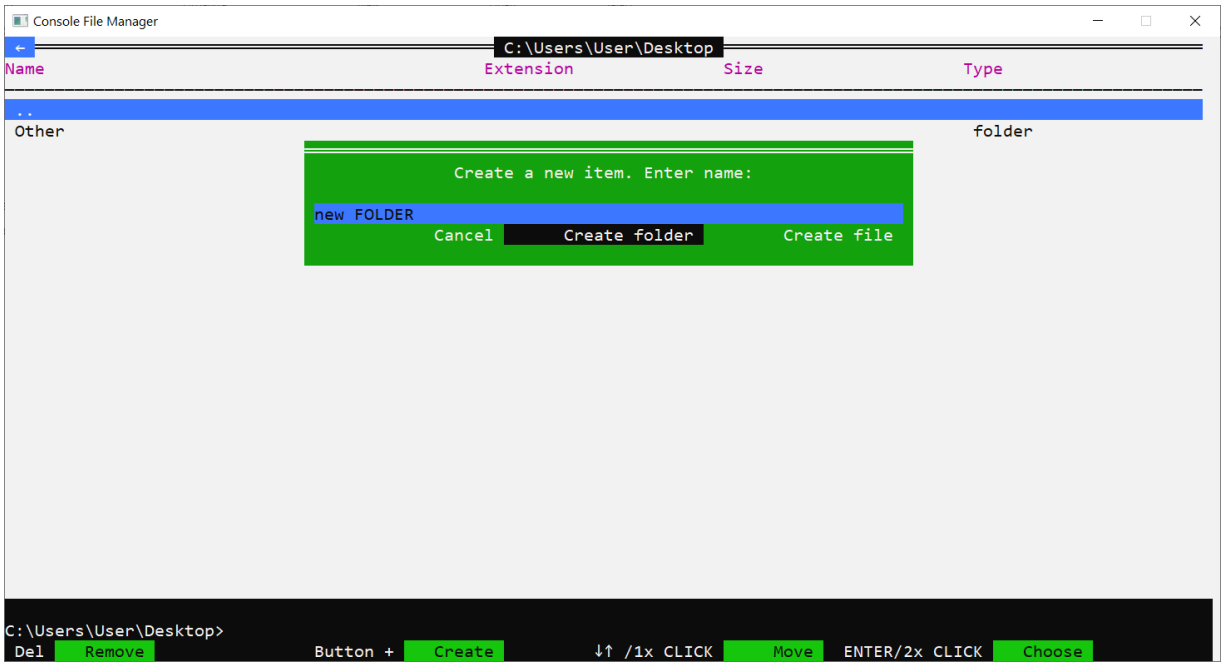

Рисунок 5 – Модальное окно

Всплывающие окна были вынесены в отдельные классы, но сохранили доступ к функциональности класса GUI для возможности пользования его методами по отрисовке на экране, изменению цветов и так далее. Это было реализовано за счёт возможности «подружить» классы ключевым словом friend.

4.4 Прослушка событий

Ещё одной сущностью в приложении является класс для мониторинга событий пользователя. Изначально эта ответственность лежала на классе GUI, но после была вынесена в отдельную сущность. В основе лежит бесконечный цикл, который при помощи инструментов, предоставляемых WinAPI и C++ имеет возможность отлавливать все виды событий, производимых в текущем окне консоли. В случае разработанной курсовой работы это события:

- единичный клик левой кнопки мыши;
- двойной клик левой кнопки мыши;
- прокрутка колеса мыши;
- нажатия клавиш на клавиатуре.

Работа с событиями организована так, что события не перекрывают друг друга во время, например, открытого модального окна. Это значит, что во время активного всплывающего окна события в основном интерфейсе «блокируются», а правильнее сказать – игнорируются, а обрабатываются уже для непосредственно открытого окна.

С таким подходом удалось избежать возникших на раннем этапе внедрения событий проблем. Сейчас с такой реализацией можно добавлять сколько угодно типов модальных окон (или других сущностей, реагирующих по-другому на события) – и они не будут перекрывать друг друга, так как одновременно активной может быть только одна сущность и только она сама в состоянии себя сделать неактивной (и, следовательно, вернуть «права на события» основному интерфейсу).

5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

В main.cpp присутствует одна строчка, которая запускает написанное приложение. Это метод класса FileManagerLauncher. Его цель – это собрать стартовую конфигурацию приложения из сторонних файлов с настройками и запустить по отдельности 3 главных класса. Под стартовой конфигурацией подразумевается путь по умолчанию, в который ведёт файловый менеджер (файл config.txt), и цветовая тема приложения (набор обычных и акцентных цветов для компонентов) (файл theme.txt).

Таким образом всё приложение построено на нескольких статических классах, которые выступают даже скорее гарантом сокрытия и объединения логики, чем именно классом в привычном смысле (для порождения объектов).

полностью функционирующее приложение без видимых проблем.

Из особенностей проекта необходимо описать несколько наиболее важных. Разработка велась исключительно на чистом C++ без привлечения сторонних фреймворков и с использованием WinAPI (встроенных библиотек windows.h [11] и winuser.h [12]). При разработке использовался современный стандарт C++ версии 17 [13], а также использовались все нововведения этого языка, начиная с версии C++ 11. Это такие крайне важные моменты как:

- использование типа auto [14];
- использование range-based циклов для обхода коллекций [15];
- использование enum-классов вместо обычных перечислений [16];
- использование структуры std::array вместо обыкновенных статических массивов [17];
- использование анонимных функций (лямбда-выражений);
- активное использование STL-контейнеров.

В C++17 добавлено новое пространство имен std::filesystem [18]. Оно предоставляет удобный интерфейс для работы с файловой системой. Тогда как раньше приходилось складывать строки и вызывать функции, пришедшие из языка программирования C, для манипуляций атрибутами файлов. В курсовой работе использовалось именно это новое пространство имён и его API, а не устаревшее API из заголовочного файла direct.h [19].

В течение всего написания проекта было проведено несколько рефакторингов кода, которые существенно улучшили его читаемость и эффективность [20].

ЗАКЛЮЧЕНИЕ

В ходе написания курсовой работы автор столкнулся со следующими проблемами:

– невозможность отобразить абсолютно все символы из кодовой таблицы в консоли;
– хаотичная перерисовка консоли (сдвиг всех символов на случайное число позиций) после сворачивания и разворачивания окна;
– различная ширина вывода для латинских и кириллических символов.

Тем не менее, проделанный анализ и разработанное приложение показали, что эти проблемы, вероятно, можно решать и на выходе получать

В результате написания курсового проекта было создано консольное приложение со следующим функционалом:

– навигация по папкам;
– просмотр файлов с помощью программ, зарегистрированных по умолчанию для данных типов файлов;
– запоминание истории посещений;
– возможность создавать новые файлы и директории, а также удалять их;
– полная поддержка событий с компьютерной мышью и клавиатурой.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

[1] Операционная система Windows [Электронный ресурс]. – Режим доступа: https://www.microsoft.com/en-us/windows?r=1/. – Дата доступа: 03.02.2023.

[2] Семейство компиляторов GNU [Электронный ресурс]. – Режим доступа: https://www.gnu.org/home.en.html/. – Дата доступа: 10.02.2023.

[3] GNU для Windows [Электронный ресурс]. – Режим доступа: https://www.mingw-w64.org/. – Дата доступа: 12.02.2023.

[4] Среда разработки Clion [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/clion/. – Дата доступа: 12.02.2023.

[5] Введение в Windows API [Электронный ресурс]. – Режим доступа: https://users.physics.ox.ac.uk/~Steane/cpp_help/winapi_intro.htm/. – Дата доступа: 05.03.2023.

[6] Файловый менеджер FAR [Электронный ресурс]. – Режим доступа: https://www.farmanager.com/. – Дата доступа: 17.03.2023.

[7] Библиотека React [Электронный ресурс]. – Режим доступа: https://react.dev/. – Дата доступа: 26.03.2023.

[8] Принцип DRY [Электронный ресурс]. – Режим доступа: www.digitalocean.com/community/tutorials/what-is-dry-development. – Дата доступа: 30.03.2023.

[9] Принцип KISS [Электронный ресурс]. – Режим доступа: https://www.interaction-design.org/literature/article/kiss-keep-it-simple-stupid-a-design-principle. – Дата доступа: 03.04.2023.

[10] Google Code Style Guide [Электронный ресурс]. – Режим доступа: https://google.github.io/styleguide/. – Дата доступа: 01.04.2023.

[11] Заголовочный файл windows.h [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Windows.h/. – Дата доступа: 17.04.2023.

[12] Заголовочный файл winuser.h [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/winuser/. – Дата доступа: 17.04.2023.

[13] Галовиц Я. C++ 17 STL Стандартная библиотека шаблонов / Я. Галовиц – Спб: Питер, 2018. – 18 с.

[14] Тип auto [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/cpp/cpp/auto-cpp?view=msvc-170. – Дата доступа: 31.01.2023.

[15] Range-based циклы [Электронный ресурс]. – Режим доступа: https://en.cppreference.com/w/cpp/language/range-for. – Дата доступа: 26.04.2023.

[16] Enum-классы [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/enum-classes-in-c-and-their-advantage-over-enum-datatype/. – Дата доступа: 07.05.2023.

[17] Контейнер std::array [Электронный ресурс]. – Режим доступа: https://en.cppreference.com/w/cpp/container/array. – Дата доступа: 07.05.2023.

[18] Галовиц Я. C++ 17 STL Стандартная библиотека шаблонов / Я. Галовиц – Спб: Питер, 2018. – 400 с.

[19] Заголовочный файл direct.h [Электронный ресурс]. – Режим доступа: https://digitalmars.com/rtl/direct.html. – Дата доступа: 02.02.2023.

[20] Понятие рефакторинга кода [Электронный ресурс]. – Режим доступа: https://www.techtarget.com/searchapparchitecture/definition/refactoring. – Дата доступа: 09.05.2023.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

```
#include "./Launcher/launcher.h"

int main()
{
    FileManagerLauncher::Launch();
    return 0;
}
const std::string Trim(std::string s)
{
    s.erase(std::find_if(std::rbegin(s), std::rend(s), [](unsigned char ch) {
        return !std::isspace(ch);
    }).base(), std::end(s));
    s.erase(std::begin(s), std::find_if(std::begin(s), std::end(s), [](unsigned char ch) {
        return !std::isspace(ch);
    }));

    return s;
}
const Color StringToColor(const std::string &color)
{
    auto trimmedString = Trim(color);

    if (trimmedString == "Black")
        return Color::Black;
    if (trimmedString == "Blue")
        return Color::Blue;
    if (trimmedString == "Green")
        return Color::Green;
    if (trimmedString == "Cyan")
        return Color::Cyan;
    if (trimmedString == "Red")
        return Color::Red;
    if (trimmedString == "Magenta")
        return Color::Magenta;
    if (trimmedString == "Brown")
        return Color::Brown;
    if (trimmedString == "LightGray")
        return Color::LightGray;
    if (trimmedString == "DarkGray")
```

```
        return Color::DarkGray;
    if (trimmedString == "LightBlue")
        return Color::LightBlue;
    if (trimmedString == "LightGreen")
        return Color::LightGreen;
    if (trimmedString == "LightCyan")
        return Color::LightCyan;
    if (trimmedString == "LightRed")
        return Color::LightRed;

    if (trimmedString == "LightMagenta")
        return Color::LightMagenta;
    if (trimmedString == "Yellow")
        return Color::Yellow;
    if (trimmedString == "White")
        return Color::White;
    return Color::Black;
}

const size_t GetMaximumWordLength(const std::array<const std::string, 4>
&array)
{
    size_t response = array.front().size();
    for (const auto &word : array)
        response = std::max(response, word.size());
    return response;
}

const std::string CutDirectoryString(const std::string &query, const size_t
max_path_length)
{
    if (query.size() <= max_path_length)
        return query;
    std::string response;
    response = query;
    size_t index = response.find_first_of('\\') + 1;
    while (response.size() > max_path_length - 3)
        response.erase(index, 1);
    response.insert(index, "...");
    return response;
}


const std::string FileTypeToString(const std::filesystem::file_type &type)
```

```
{
   switch (type)
   {
      case std::filesystem::file_type::block:
         return "block";
      case std::filesystem::file_type::character:
         return "character";
      case std::filesystem::file_type::directory:
         return "folder";
      case std::filesystem::file_type::fifo:
         return "fifo";
      case std::filesystem::file_type::none:
         return "none";
      case std::filesystem::file_type::not_found:
         return "not found";
      case std::filesystem::file_type::regular:
         return "regular";
      case std::filesystem::file_type::socket:
         return "socket";
      case std::filesystem::file_type::symlink:
         return "symlink";
      case std::filesystem::file_type::unknown:
         return "unknown";
      default:
         return "other";
   }
}

const std::string CutFileNameString(const std::string &query, const size_t
to_length)
{
   if (query.size() < to_length)
      return query;
   return query.substr(0, to_length - 3) + "...";
}

const bool IsFileHidden(const std::filesystem::directory_entry &file)
{
   if (file.path().filename().string() != ".." && file.path().filename().string() != "."
&&
      file.path().filename().string()[0] == '.')
      return true;
   return false;
}
```

```
const std::string GetParentDirectory(const std::string &current_directory)
{
   if (current_directory == "C:\\")
      return current_directory;
   std::string response = current_directory;
   auto last_slash_position = response.find_last_of('\\');
   response.erase(last_slash_position, response.size() - last_slash_position + 1);
   if (std::find(std::begin(response), std::end(response), '\\') == std::end(response))
      response.push_back('\\');
   return response;
}

const std::string GetAdaptiveSize(const size_t bytes_value)
{
   std::string bytes = std::to_string(bytes_value);
   std::string kilobytes = std::to_string(bytes_value / 1024);
   std::string megabytes = std::to_string((bytes_value / 1024) / 1024);
   std::string gigabytes = std::to_string(((bytes_value / 1024) / 1024) / 1024);

   if (bytes.size() <= 3)
      return bytes + " B";
   if (kilobytes.size() <= 3)
      return kilobytes + " KB";
   if (megabytes.size() <= 3)
      return megabytes + " MB";
   return gigabytes + " GB";
}
std::string AppState::current_directory;
std::string AppState::parent_directory;
std::vector<std::filesystem::directory_entry> AppState::files_list;
std::map<size_t, std::filesystem::directory_entry>
AppState::currently_rendered_with_coordinates;
std::vector<std::filesystem::directory_entry>
AppState::currently_rendered_files_list;
const size_t AppState::kFilesListLength;
std::stack<std::string> AppState::history;
size_t AppState::render_from;
size_t AppState::render_to;
size_t AppState::current_position = 0;
void AppState::Launch(const std::string &path)
{
   AppState::UpdateDirectory(path);
}
```

```
std::filesystem::directory_iterator AppState::GetDirectoryByPath(const std::string
&path)
{
   return std::filesystem::directory_iterator(path);
}
void
AppState::GetFilesListFromDirectoryIterator(std::filesystem::directory_iterator
&files)
{
   AppState::files_list.clear();

   if (AppState::parent_directory != AppState::current_directory)
      AppState::files_list.emplace_back(

std::filesystem::directory_entry(std::filesystem::path(AppState::parent_directory)))
;

   for (const auto &item : files)
      if (!IsFileHidden(item))
         AppState::files_list.emplace_back(item);
}
void AppState::CreateCurrentlyRenderedList()
{
   AppState::currently_rendered_files_list.clear();

   for (size_t counter = AppState::render_from; counter <= render_to; ++counter)
AppState::currently_rendered_files_list.push_back(AppState::files_list[counter]);
}
void AppState::UpdateDirectory(const std::string &path)
{
   AppState::current_directory = path;
   AppState::parent_directory = GetParentDirectory(AppState::current_directory);
   auto raw_files_iterator =
AppState::GetDirectoryByPath(AppState::current_directory);
   AppState::GetFilesListFromDirectoryIterator(raw_files_iterator);

   AppState::current_position = 0;
   AppState::render_from = 0;
   AppState::render_to = std::min(AppState::files_list.size() - 1,
static_cast<size_t>(AppState::kFilesListLength));
   AppState::CreateCurrentlyRenderedList();
}
bool AppState::GoBack()
{
```

```
    if (AppState::history.empty())
        return false;

    AppState::UpdateDirectory(AppState::history.top());
    AppState::history.pop();

    return true;
}
void AppState::Move(const std::string &new_path)
{
    AppState::history.push(AppState::current_directory);
    AppState::UpdateDirectory(new_path);
}
const std::string FileManagerLauncher::kFileManagerSettingsFilePath =
"config.txt";
const std::string FileManagerLauncher::kFileManagerDefaultThemeFilePath =
"theme.txt";
std::string FileManagerLauncher::kStartDirectory;
void FileManagerLauncher::Launch()
{
    FileManagerLauncher::LoadConfiguration();
    AppState::Launch(FileManagerLauncher::kStartDirectory);
    GUI::Launch();
    EventsController::RunEventLoop();
}
void FileManagerLauncher::LoadConfiguration()
{
    FileManagerLauncher::LoadSettings();
    FileManagerLauncher::LoadTheme();
}
void FileManagerLauncher::LoadTheme()
{
    auto read =
std::ifstream(FileManagerLauncher::kFileManagerDefaultThemeFilePath);
    std::string body_bgc, body_fgc, body_bgc_acc, body_fgc_acc,
            footer_bgc, footer_fgc, footer_bgc_acc, footer_fgc_acc;
    std::getline(read, body_bgc);
    std::getline(read, body_fgc);
    std::getline(read, body_bgc_acc);
    std::getline(read, body_fgc_acc);
    std::getline(read, footer_bgc);
    std::getline(read, footer_fgc);
    std::getline(read, footer_bgc_acc);
    std::getline(read, footer_fgc_acc);
```

```
    GUI::SetTheme({StringToColor(body_bgc),
                   StringToColor(body_fgc),
                   StringToColor(body_bgc_acc),
                   StringToColor(body_fgc_acc),
                   StringToColor(footer_bgc),
                   StringToColor(footer_fgc),
                   StringToColor(footer_bgc_acc),
                   StringToColor(footer_fgc_acc)});

    read.close();
}

void FileManagerLauncher::LoadSettings()
{
    auto read = std::ifstream(FileManagerLauncher::kFileManagerSettingsFilePath);
    std::getline(read, FileManagerLauncher::kStartDirectory);
    read.close();
}

#include "modal_delete.h"
const std::array<const std::string, 2> ModalDelete::kChoiceItems = {"Cancel",
"Ok"};
const std::string ModalDelete::kWarning = "Do you wish to remove ?";
std::filesystem::path ModalDelete::deleted_item_path;
std::filesystem::path ModalDelete::current_path;
size_t ModalDelete::modal_width;
size_t ModalDelete::left_padding;
bool ModalDelete::is_launched = false;
size_t ModalDelete::currently_selected = 0;
size_t ModalDelete::choice_line_y = kModalMarginTop + 3;


void ModalDelete::RenderChoicePositions()
{
    GUI::MoveToCoordinate(ModalDelete::left_padding,
ModalDelete::choice_line_y);

    for (size_t counter = 0; counter < ModalDelete::kChoiceItems.size(); ++counter)
    {
        GUI::SetConsoleColors(counter == ModalDelete::currently_selected
                              ? kModalSelectionColor
                              : kModalBackgroundColor,
                              kModalForegroundColor);
        std::cout << std::setw(ModalDelete::modal_width /
```

```
ModalDelete::kChoiceItems.size()) << std::right
            << (ModalDelete::kChoiceItems.at(counter) + " ");
    }
}

void ModalDelete::Render()
{
    GUI::PaintBackground(kModalMarginTop, ModalDelete::left_padding,
                kModalBottomBorderCoordinate, ModalDelete::left_padding +
ModalDelete::modal_width,
                kModalBackgroundColor);
    GUI::MoveToCoordinate(ModalDelete::left_padding, kModalMarginTop);
    GUI::SetConsoleColors(kModalBackgroundColor, kModalForegroundColor);
    ModalDelete::PrintBorder();

    GUI::MoveToCoordinate(ModalDelete::left_padding +
(ModalDelete::modal_width / 2 - ModalDelete::kWarning.size() / 2),
                 kModalMarginTop + 1);


    std::cout << ModalDelete::kWarning;

    ModalDelete::RenderChoicePositions();
}

void ModalDelete::Launch(const std::filesystem::path &path, const
std::filesystem::path &current)
{
    ModalDelete::deleted_item_path = path;
    ModalDelete::current_path = current;
    ModalDelete::modal_width = GUI::console_width / 2;
    ModalDelete::left_padding = ModalDelete::modal_width / 2;

    ModalDelete::is_launched = true;

    ModalDelete::currently_selected = 0;
    ModalDelete::Render();
}

const bool ModalDelete::IsLaunched()
{
    return ModalDelete::is_launched;
}
```

```
void ModalDelete::PrintBorder()
{
   for (size_t counter = 0; counter <= ModalDelete::modal_width; ++counter)
      std::cout << kModalBorderTexture;
}

void ModalDelete::MoveSelection(const short new_item)
{
   if (new_item >= 0 && new_item < ModalDelete::kChoiceItems.size())
   {
      ModalDelete::currently_selected = new_item;
      ModalDelete::RenderChoicePositions();
   }
}

void ModalDelete::ProcessChoice()
{
   switch (ModalDelete::currently_selected)
   {
      case 0:
         ModalDelete::is_launched = false;
         GUI::RenderBodyDynamicFilesList();
         break;
      case 1:
         std::filesystem::remove_all(ModalDelete::deleted_item_path);
         AppState::UpdateDirectory(ModalDelete::current_path.string());
         ModalDelete::is_launched = false;
         GUI::RenderBodyDynamicFilesList();
         break;
   }
}

void ModalDelete::Close()
{
   ModalDelete::is_launched = false;
   GUI::RenderBodyDynamicFilesList();
}

void ModalDelete::ComputeSingleMouseClick(const size_t y, const size_t x)
{
   if (y < kModalMarginTop ||
      y > kModalBottomBorderCoordinate ||
      x < ModalDelete::left_padding ||
      x > ModalDelete::left_padding + ModalDelete::modal_width)
```

```
    {
        ModalDelete::Close();
        return;
    }

    else if (y == ModalDelete::choice_line_y)
    {
        if (x >= ModalDelete::left_padding &&
            x < ModalDelete::left_padding + ModalDelete::modal_width /
ModalDelete::kChoiceItems.size())
            ModalDelete::MoveSelection(0);
        else
            ModalDelete::MoveSelection(1);
    }
}

void ModalDelete::ComputeDoubleMouseClick(const size_t y, const size_t x)
{
    if (y < kModalMarginTop ||
        y > kModalBottomBorderCoordinate ||
        x < ModalDelete::left_padding ||
        x > ModalDelete::left_padding + ModalDelete::modal_width)
    {
        ModalDelete::Close();
        return;
    }

    else if (y == ModalDelete::choice_line_y)
    {
        if (x >= ModalDelete::left_padding &&
            x < ModalDelete::left_padding + ModalDelete::modal_width /
ModalDelete::kChoiceItems.size())
        {
            ModalDelete::MoveSelection(-1);
            ModalDelete::ProcessChoice();
        }
        else
        {
            ModalDelete::MoveSelection(1);
            ModalDelete::ProcessChoice();
        }
    }
}
```

```
#include "modal_create.h"


const std::array<const std::string, 3> ModalCreate::kChoiceItems = {"Cancel",
"Create folder", "Create file"};
const std::string ModalCreate::kWarning = "Create a new item. Enter name:";

size_t ModalCreate::modal_width;

size_t ModalCreate::left_padding;
size_t ModalCreate::choice_line_y = kModalMarginTop + 4;
bool ModalCreate::is_launched = false;

std::string ModalCreate::new_file_name = "new file";
std::filesystem::path ModalCreate::current_path;
size_t ModalCreate::currently_selected = 0;


void ModalCreate::PrintBorder()
{
   for (size_t counter = 0; counter <= ModalCreate::modal_width; ++counter)
      std::cout << kModalBorderTexture;
}

void ModalCreate::RenderChoicePositions()
{
   GUI::MoveToCoordinate(ModalCreate::left_padding,
ModalCreate::choice_line_y);

   for (size_t counter = 0; counter < ModalCreate::kChoiceItems.size(); ++counter)
   {
      GUI::SetConsoleColors(counter == ModalCreate::currently_selected
                        ? kModalSelectionColor
                        : kModalBackgroundColor,
                        kModalForegroundColor);

      std::cout << std::setw(ModalCreate::modal_width /
ModalCreate::kChoiceItems.size()) << std::right
               << (ModalCreate::kChoiceItems.at(counter) + " ");
   }
}

void ModalCreate::RenderInput()
{
```

```
   GUI::MoveToCoordinate(ModalCreate::left_padding,
ModalCreate::choice_line_y - 1);
   GUI::PaintBackground(ModalCreate::choice_line_y - 1,
ModalCreate::left_padding + 1,
                        ModalCreate::choice_line_y - 1, ModalCreate::left_padding +
ModalCreate::modal_width - 1,
                        kModalInputBackground);

   GUI::MoveToCoordinate(ModalCreate::left_padding + 1,
ModalCreate::choice_line_y - 1);
   GUI::SetConsoleColors(kModalInputBackground, kModalInputForeground);
   std::cout << std::setw(ModalCreate::modal_width - 1) << std::left <<
ModalCreate::new_file_name;

}

void ModalCreate::Render()
{
   GUI::PaintBackground(kModalMarginTop, ModalCreate::left_padding,
                        kModalBottomBorderCoordinate, ModalCreate::left_padding +
ModalCreate::modal_width,
                        kModalBackgroundColor);

   GUI::MoveToCoordinate(ModalCreate::left_padding, kModalMarginTop);
   GUI::SetConsoleColors(kModalBackgroundColor, kModalForegroundColor);
   ModalCreate::PrintBorder();

   GUI::MoveToCoordinate(ModalCreate::left_padding +
(ModalCreate::modal_width / 2 - ModalCreate::kWarning.size() / 2),
                         kModalMarginTop + 1);
   std::cout << ModalCreate::kWarning;

   ModalCreate::RenderInput();
   ModalCreate::RenderChoicePositions();
}

void ModalCreate::ProcessChoice()
{
   switch (ModalCreate::currently_selected)
   {
      case 0:
         ModalCreate::Close();
         GUI::RenderBodyDynamicFilesList();
         break;
```

```
        case 1:
            if (std::find_if(std::begin(AppState::files_list),
std::end(AppState::files_list),
                             [&](const auto &item) { return item.path() ==
ModalCreate::new_file_name; }) ==
                std::end(AppState::files_list))
            {
                std::filesystem::create_directory(
                      std::filesystem::path(ModalCreate::current_path.string() + '\\' +
ModalCreate::new_file_name));
                AppState::UpdateDirectory(ModalCreate::current_path.string());
            }

            ModalCreate::Close();
            GUI::RenderBodyDynamicFilesList();
            break;
        case 2:
            auto response = std::ofstream(ModalCreate::current_path.string() +
ModalCreate::new_file_name);
            AppState::UpdateDirectory(ModalCreate::current_path.string());
            ModalCreate::Close();
            GUI::RenderBodyDynamicFilesList();
            break;
    }
}

void ModalCreate::ComputeSingleMouseClick(const size_t y, const size_t x)
{
    if (y < kModalMarginTop ||
        y > kModalBottomBorderCoordinate ||
        x < ModalCreate::left_padding ||
        x > ModalCreate::left_padding + ModalCreate::modal_width)
    {
        ModalCreate::Close();
        return;
    }

    else if (y == ModalCreate::choice_line_y && x >= ModalCreate::left_padding
&&
             x < ModalCreate::left_padding + ModalCreate::modal_width)
    {
        if (x >= ModalCreate::left_padding &&
            x < ModalCreate::left_padding + ModalCreate::modal_width /
ModalCreate::kChoiceItems.size())
```

```
            ModalCreate::MoveSelection(0);
        else if (x > ModalCreate::left_padding + 2 * ModalCreate::modal_width /
ModalCreate::kChoiceItems.size())
            ModalCreate::MoveSelection(2);
        else
            ModalCreate::MoveSelection(1);
    }
}

void ModalCreate::ComputeDoubleMouseClick(const size_t y, const size_t x)
{
    if (y < kModalMarginTop ||
        y > kModalBottomBorderCoordinate ||
        x < ModalCreate::left_padding ||
        x > ModalCreate::left_padding + ModalCreate::modal_width)
    {
        ModalCreate::Close();
        return;
    }

    else if (y == ModalCreate::choice_line_y && x >= ModalCreate::left_padding
&&
             x < ModalCreate::left_padding + ModalCreate::modal_width)
    {
        if (x >= ModalCreate::left_padding &&
            x < ModalCreate::left_padding + ModalCreate::modal_width /
ModalCreate::kChoiceItems.size())
            ModalCreate::MoveSelection(0);
        else if (x > ModalCreate::left_padding + 2 * ModalCreate::modal_width /
ModalCreate::kChoiceItems.size())
            ModalCreate::MoveSelection(2);
        else
            ModalCreate::MoveSelection(1);

        ModalCreate::ProcessChoice();
    }
}

void ModalCreate::UpdateNewFileName(const size_t code)
{
    if (code == 13)
    {
        if (!ModalCreate::new_file_name.empty())
        {
```

```
            ModalCreate::new_file_name.pop_back();
            ModalCreate::RenderInput();
        }
    }
    else
    {
        if (ModalCreate::new_file_name.size() < ModalCreate::modal_width - 2)
        {
            ModalCreate::new_file_name += static_cast<char>(code);
            ModalCreate::RenderInput();
        }
    }
}


void ModalCreate::Launch(const std::filesystem::path &current)
{
    ModalCreate::current_path = current;
    ModalCreate::modal_width = GUI::console_width / 2;
    ModalCreate::left_padding = ModalCreate::modal_width / 2;

    ModalCreate::is_launched = true;

    ModalCreate::Render();
}

const bool ModalCreate::IsLaunched()
{
    return ModalCreate::is_launched;
}


void ModalCreate::MoveSelection(const short new_item)
{
    if (new_item >= 0 && new_item < ModalCreate::kChoiceItems.size())
    {
        ModalCreate::currently_selected = new_item;
        ModalCreate::RenderChoicePositions();
    }
}

void ModalCreate::Close()
{
    ModalCreate::is_launched = false;
```

```
    GUI::RenderBodyDynamicFilesList();
}
#include "main_GUI.h"


const std::string GUI::kWindowTitle = "Console File Manager";

const std::string GUI::kFirstLineTexture = "═";
const std::string GUI::kSecondLineTexture = "—";
const std::string GUI::kArrowBackTexture = "←";

const size_t GUI::kMenuItemsCount;
const std::array<const std::string, GUI::kMenuItemsCount>
GUI::kMenuItemsTitles = {"Remove", "Create", "Move", "Choose"};
const std::array<const std::string, GUI::kMenuItemsCount>
GUI::kMenuItemsKeys = {"Del", " Button +",
                                                    "↓↑ /1x CLICK", "ENTER/2x
CLICK"};
const size_t GUI::kColumnsCount;
const std::array<const std::string, GUI::kColumnsCount> GUI::kColumnsTitles =
{"Name", "Extension", "Size", "Type"};
const std::array<const size_t, GUI::kColumnsCount> GUI::kColumnsPrecisions =
{2, 1, 1, 1};

const size_t GUI::kFooterStartPositionFromBottom;

size_t GUI::kMaxPathLength;
size_t GUI::kMaximumMenuItemLength;

size_t GUI::console_width;
const size_t GUI::console_height;
const size_t GUI::kFilesListLength;

HANDLE GUI::console_handle;
CONSOLE_SCREEN_BUFFER_INFO GUI::console_info;

bool GUI::was_first_render = false;

Theme GUI::kTheme;


void GUI::SetTheme(const Theme &new_theme)
{
    GUI::kTheme = new_theme;
```

```
}

void GUI::Launch()
{
   GUI::kMaximumMenuItemLength =
GetMaximumWordLength(GUI::kMenuItemsTitles) + 2;

   GUI::ConfigureConsoleWindow();
   GUI::PaintBackground(0, 0, GUI::console_height - 1, GUI::console_width - 1,
GUI::kTheme.body_background);
   GUI::RenderFooter();
   GUI::RenderBody();
   GUI::was_first_render = true;
}

void GUI::ResizeConsole()
{
   GUI::console_handle = GetStdHandle(STD_OUTPUT_HANDLE);
   GetConsoleScreenBufferInfo(GUI::console_handle, &GUI::console_info);

   GUI::console_width = GUI::console_info.dwSize.X;
   SetConsoleScreenBufferSize(GUI::console_handle, {(short)
GUI::console_width, (short) GUI::console_height});

   SetWindowLong(GetConsoleWindow(), GWL_STYLE,
            GetWindowLong(GetConsoleWindow(), GWL_STYLE) &
~WS_MAXIMIZEBOX & ~WS_SIZEBOX);

   GUI::kMaxPathLength = GUI::console_width / 2;
}

void GUI::HideCursor()
{
   auto cursor = CONSOLE_CURSOR_INFO();
   cursor.bVisible = false;
   cursor.dwSize = 20;
   SetConsoleCursorInfo(GUI::console_handle, &cursor);
}


void GUI::ConfigureConsoleWindow()
{
   SetConsoleOutputCP(CP_UTF8);
   SetConsoleCP(CP_UTF8);
```

```
    SetConsoleTitle(GUI::kWindowTitle.c_str());
    GUI::ResizeConsole();
    GUI::HideCursor();
}

void GUI::MoveToCoordinate(const size_t x, const size_t y)
{
    SetConsoleCursorPosition(GUI::console_handle, {(short) x, (short) y});
}

void GUI::SetConsoleColors(const Color &back, const Color &fore)
{
    SetConsoleTextAttribute(GUI::console_handle, (WORD)
((static_cast<int>(back) << 4) | static_cast<int>(fore)));
}

void GUI::PaintBackground(const size_t y_start, const size_t x_start, const size_t
y_end, const size_t x_end,
                          const Color &background)
{
    GUI::MoveToCoordinate(x_start, y_start);
    GUI::SetConsoleColors(background, GUI::kTheme.footer_foreground);

    for (size_t y = y_start; y <= y_end; ++y)
    {
        for (size_t x = x_start; x <= x_end; ++x)
        {
            GUI::MoveToCoordinate(x, y);
            std::cout << ' ';
        }
    }
}

void GUI::PaintFooterBackground()
{
    GUI::PaintBackground(GUI::console_height -
GUI::kFooterStartPositionFromBottom, 0,
                         GUI::console_height - 1, GUI::console_width - 1,
                         GUI::kTheme.footer_background);
}

void GUI::PaintBodyBackground()
{
```

```
    GUI::PaintBackground(3, 0,
                         GUI::console_height - GUI::kFooterStartPositionFromBottom - 2,
console_width - 1,
                         GUI::kTheme.body_background);
}

void GUI::RenderFooterFixedInterface()
{
    GUI::MoveToCoordinate(0, GUI::console_height - 1);

    size_t precision = GUI::console_width / GUI::kMenuItemsCount - 1;
    size_t counter = 0;
    for (const auto &key : GUI::kMenuItemsKeys)
    {
        GUI::SetConsoleColors(GUI::kTheme.footer_background,
GUI::kTheme.footer_foreground);
        std::cout << std::setw(key.size() + 2) << (" " + key + " ");

        GUI::SetConsoleColors(GUI::kTheme.footer_background_accent,
GUI::kTheme.footer_foreground_accent);
        std::cout << std::setw(GUI::kMaximumMenuItemLength + 2) << (" " +
GUI::kMenuItemsTitles.at(counter) + " ");

        GUI::SetConsoleColors(GUI::kTheme.footer_background,
GUI::kTheme.footer_foreground);
        std::cout << std::setw(precision - GUI::kMaximumMenuItemLength -
key.size() - 4) << " ";

        ++counter;
    }
}

void GUI::RenderBodyFixedInterface()
{
    GUI::MoveToCoordinate(0, 1);
    GUI::SetConsoleColors(GUI::kTheme.body_background,
Color::LightMagenta);

    for (size_t counter = 0; counter < GUI::kColumnsCount; ++counter)
        std::cout << std::setw(GUI::console_width *
GUI::kColumnsPrecisions.at(counter) / 5)
                  << std::left << GUI::kColumnsTitles.at(counter);

    GUI::SetConsoleColors(GUI::kTheme.body_background,
```

```
GUI::kTheme.body_foreground);
    GUI::MoveToCoordinate(0, 2);
    for (size_t counter = 0; counter < GUI::console_width; ++counter)
        std::cout << GUI::kSecondLineTexture;
}

void GUI::RenderBodyDynamicPath()
{
    GUI::MoveToCoordinate(0, 0);
    GUI::SetConsoleColors(GUI::kTheme.body_background,
GUI::kTheme.body_foreground);
    for (size_t counter = 0; counter < GUI::console_width; ++counter)
        std::cout << GUI::kFirstLineTexture;

    std::string path = CutDirectoryString(AppState::current_directory,
GUI::kMaxPathLength);
    size_t left_margin = GUI::console_width / 2 - (path.size() + 2) / 2;
    GUI::MoveToCoordinate(left_margin, 0);
    GUI::SetConsoleColors(Color::Black, Color::White);
    std::cout << ' ' << path << ' ';

    GUI::MoveToCoordinate(0, 0);
    GUI::SetConsoleColors(GUI::kTheme.body_background_accent,
GUI::kTheme.body_foreground_accent);
    std::cout << ' ' << GUI::kArrowBackTexture << ' ';
}

void GUI::RenderBodySingleFileLine(const std::filesystem::directory_entry &file,
const bool is_link_to_parent)
{
    if (file.path().string() == AppState::parent_directory)
    {
        std::cout << std::setw(GUI::console_width) << std::left << " .. ";
        return;
    }

    std::cout << std::setw(GUI::kColumnsPrecisions.at(0) * GUI::console_width /
5) << std::left
              << (" " + CutFileNameString(file.path().filename().string(),
                                          GUI::kColumnsPrecisions.at(0) * GUI::console_width /
5 - 1));
    std::cout << std::setw(GUI::kColumnsPrecisions.at(1) * GUI::console_width /
5) << std::left
              << (" " + file.path().extension().string());
```

```
   std::cout << std::setw(GUI::kColumnsPrecisions.at(2) * GUI::console_width /
5) << std::left
             << (file.is_regular_file() ? (" " + GetAdaptiveSize(file.file_size())) : "");
   std::cout << std::setw(GUI::kColumnsPrecisions.at(3) * GUI::console_width /
5) << std::left
             << (" " + FileTypeToString(file.status().type()));
}

void GUI::RenderBodyDynamicFilesList()
{
   GUI::PaintBodyBackground();

   AppState::currently_rendered_with_coordinates.clear();

   GUI::SetConsoleColors(GUI::kTheme.body_background, Color::DarkGray);
   GUI::MoveToCoordinate(0, 3);

   size_t counter = 0;
   for (const auto &file : AppState::currently_rendered_files_list)
   {
      AppState::currently_rendered_with_coordinates.insert(std::make_pair(3 +
counter, file));

      if (counter == AppState::current_position)
         GUI::SetConsoleColors(GUI::kTheme.body_background_accent,
GUI::kTheme.body_foreground_accent);
      else
         GUI::SetConsoleColors(GUI::kTheme.body_background,
GUI::kTheme.body_foreground);

      GUI::RenderBodySingleFileLine(file, (counter == 0));

      ++counter;
   }
}

void GUI::RenderBody()
{
   if (GUI::was_first_render == false)
      GUI::RenderBodyFixedInterface();

   GUI::RenderBodyDynamicPath();
   GUI::RenderBodyDynamicFilesList();
}
```

```
void GUI::RenderFooter()
{
   if (!GUI::was_first_render)
      GUI::PaintFooterBackground();

   GUI::MoveToCoordinate(0, GUI::console_height -
GUI::kFooterStartPositionFromBottom);
   GUI::SetConsoleColors(GUI::kTheme.footer_background,
GUI::kTheme.footer_foreground);
   std::cout << std::setw(GUI::console_width - 1) << "";
   GUI::MoveToCoordinate(0, GUI::console_height -
GUI::kFooterStartPositionFromBottom);
   std::cout << CutDirectoryString(AppState::current_directory,
GUI::kMaxPathLength) << '>';

   if (!GUI::was_first_render)
      GUI::RenderFooterFixedInterface();
}


void GUI::ChangeSelection(const size_t previous, const size_t current)
{
   GUI::MoveToCoordinate(0, 3 + previous);
   GUI::SetConsoleColors(GUI::kTheme.body_background,
GUI::kTheme.body_foreground);

GUI::RenderBodySingleFileLine(AppState::currently_rendered_with_coordinates[
3 + previous],
                              AppState::currently_rendered_with_coordinates.at(3 +
previous) ==
                              AppState::files_list.at(0));

   GUI::MoveToCoordinate(0, 3 + current);
   GUI::SetConsoleColors(GUI::kTheme.body_background_accent,
GUI::kTheme.body_foreground_accent);

GUI::RenderBodySingleFileLine(AppState::currently_rendered_with_coordinates.
at(3 + current),
                              AppState::currently_rendered_with_coordinates.at(3 +
current) ==
                              AppState::files_list.at(0));
}
```

```
void GUI::MoveSelection(const short delta)
{
   if (AppState::current_position + delta >= 0 &&
      AppState::current_position + delta <
AppState::currently_rendered_files_list.size())
   {
      auto prev = AppState::current_position;
      AppState::current_position = prev + delta;
      GUI::ChangeSelection(prev, AppState::current_position);
      return;
   }

   if (AppState::current_position == AppState::currently_rendered_files_list.size()
- 1 && delta == 1 &&
      AppState::render_to < AppState::files_list.size() - 1)
   {
      AppState::render_from += GUI::kFilesListLength;
      AppState::render_to = std::min(AppState::render_to + GUI::kFilesListLength,
                                     AppState::files_list.size() - 1);
      AppState::current_position = 0;
      AppState::CreateCurrentlyRenderedList();
      GUI::RenderBodyDynamicFilesList();
   }
   else if (AppState::current_position == 0 && delta == -1 &&
AppState::render_from > 0)
   {
      AppState::render_from = std::max(AppState::render_from -
GUI::kFilesListLength, static_cast<size_t>(0));
      AppState::render_to = std::min(AppState::render_from +
GUI::kFilesListLength,
                                     AppState::files_list.size() - 1);
      AppState::CreateCurrentlyRenderedList();
      AppState::current_position = AppState::currently_rendered_files_list.size() -
1;
      GUI::RenderBodyDynamicFilesList();
   }
}
#include "events_controller.h"

void EventsController::RunEventLoop()
{
   const size_t kInputRecordBufferSize = 128;

   auto fdwSaveOldMode = DWORD(), input_records_number = DWORD(), i =
```

```
DWORD();
   INPUT_RECORD input_record_buffer[kInputRecordBufferSize];

   auto handle_stdin = GetStdHandle(STD_INPUT_HANDLE);
   GetConsoleMode(handle_stdin, &fdwSaveOldMode);

   auto fdwMode = ENABLE_MOUSE_INPUT | ENABLE_INSERT_MODE;
   SetConsoleMode(handle_stdin, fdwMode);

   while (true)
   {
      ReadConsoleInput(handle_stdin, input_record_buffer,
kInputRecordBufferSize, &input_records_number);

      for (size_t counter = 0; counter < input_records_number; ++counter)
         switch (input_record_buffer[i].EventType)
         {
            case KEY_EVENT:

EventsController::ProcessKeyEvent(input_record_buffer[i].Event.KeyEvent);
               break;
            case MOUSE_EVENT:

EventsController::ProcessMouseEvent(input_record_buffer[i].Event.MouseEvent);
               break;
         }
   }

   SetConsoleMode(handle_stdin, fdwSaveOldMode);
}

void EventsController::ProcessSelection()
{
   if
(AppState::currently_rendered_files_list.at(AppState::current_position).status().typ
e() ==
      std::filesystem::file_type::directory)
   {

AppState::Move(AppState::currently_rendered_files_list.at(AppState::current_posi
tion).path().string());
      GUI::RenderBody();
      GUI::RenderFooter();
   }
```

```
   else if
(AppState::currently_rendered_files_list.at(AppState::current_position).status().type() ==
            std::filesystem::file_type::regular)
   {
      ShellExecute(HWND(), "open",

AppState::currently_rendered_files_list.at(AppState::current_position).path().string().c_str(),
                  NULL, NULL, SW_SHOWNORMAL);
   }
}

void EventsController::ArrowBackPressed(const MOUSE_EVENT_RECORD &mouse_event)
{
   if (mouse_event.dwMousePosition.Y == 0 && mouse_event.dwMousePosition.Y >= 0 && mouse_event.dwMousePosition.X <= 2)
      if (AppState::GoBack())
      {
         GUI::RenderBody();
         GUI::RenderFooter();
      }
}

void EventsController::ProcessKeyEventInMainGUI(const KEY_EVENT_RECORD &key_event)
{
   switch (key_event.wVirtualKeyCode)
   {
      case 38: // Up
         GUI::MoveSelection(-1);
         break;
      case 40: // Down
         GUI::MoveSelection(1);
         break;

      case 13: // Enter
         EventsController::ProcessSelection();
         break;
      case 8: // BackSpace
         if (AppState::GoBack())
         {
```

```
            GUI::RenderBody();
            GUI::RenderFooter();
        }
        break;
    case 46: // Delete
        if (AppState::currently_rendered_files_list.at(AppState::current_position)
!= AppState::files_list.at(0))

ModalDelete::Launch(AppState::currently_rendered_files_list.at(AppState::current
_position),
                            std::filesystem::path(AppState::current_directory));
        break;
    case 107: // +
        ModalCreate::Launch(AppState::current_directory);
        break;
  }
}

void EventsController::ProcessKeyEventInModalCreate(const
KEY_EVENT_RECORD &key_event)
{
  switch (key_event.wVirtualKeyCode)
  {
    case 37: // Left
        ModalCreate::MoveSelection(ModalCreate::currently_selected - 1);
        break;
    case 39: // Right
        ModalCreate::MoveSelection(ModalCreate::currently_selected + 1);
        break;
    case 13: // Enter//
        ModalCreate::ProcessChoice();
        break;
    case 8: // BackSpace
        ModalCreate::UpdateNewFileName(13);
        break;
    case 27: //Escape
        ModalCreate::Close();
        break;
    case 65 ... 90:

ModalCreate::UpdateNewFileName(static_cast<size_t>(key_event.wVirtualKeyC
ode));
        break;
  }
```

```
}

void EventsController::ProcessKeyEventInModalDelete(const
KEY_EVENT_RECORD &key_event)
{
   switch (key_event.wVirtualKeyCode)
   {
      case 37: // Left
         ModalDelete::MoveSelection(ModalDelete::currently_selected - 1);
         break;
      case 39: // Right
         ModalDelete::MoveSelection(ModalDelete::currently_selected + 1);
         break;
      case 13: // Enter
         ModalDelete::ProcessChoice();
         break;
      case 8: // BackSpace
         ModalDelete::Close();
         break;
      case 27: //Escape
         ModalDelete::Close();
         break;
   }
}

void EventsController::ProcessKeyEvent(const KEY_EVENT_RECORD
&key_event)
{
   if (!key_event.bKeyDown)
      return;

   if (ModalDelete::IsLaunched())
      EventsController::ProcessKeyEventInModalDelete(key_event);
   else if (ModalCreate::IsLaunched())
      EventsController::ProcessKeyEventInModalCreate(key_event);
   else
      EventsController::ProcessKeyEventInMainGUI(key_event);
}


void EventsController::ProcessMouseEventInMainGUI(const
MOUSE_EVENT_RECORD &mouse_event)
{
   switch (mouse_event.dwEventFlags)
```

```
   {
      case 0:
         if (mouse_event.dwButtonState ==
FROM_LEFT_1ST_BUTTON_PRESSED)
         {
            EventsController::ArrowBackPressed(mouse_event);
            GUI::MoveSelection((mouse_event.dwMousePosition.Y - 3) - (int)
AppState::current_position);
         }
         break;
      case DOUBLE_CLICK:
         if (mouse_event.dwButtonState ==
FROM_LEFT_1ST_BUTTON_PRESSED &&

AppState::currently_rendered_with_coordinates.contains(mouse_event.dwMouseP
osition.Y))
            EventsController::ProcessSelection();
         break;
      case MOUSE_WHEELED:
         if (static_cast<size_t>(HIWORD(mouse_event.dwButtonState)) == 120)
            GUI::MoveSelection(-1);
         else
            GUI::MoveSelection(1);
         break;
   }
}

void EventsController::ProcessMouseEventInModalDelete(const
MOUSE_EVENT_RECORD &mouse_event)
{
   switch (mouse_event.dwEventFlags)
   {
      case 0:

ModalDelete::ComputeSingleMouseClick(mouse_event.dwMousePosition.Y,
mouse_event.dwMousePosition.X);
         break;
      case DOUBLE_CLICK:

ModalDelete::ComputeDoubleMouseClick(mouse_event.dwMousePosition.Y,
mouse_event.dwMousePosition.X);
         break;
   }
}
```

```
void EventsController::ProcessMouseEventInModalCreate(const
MOUSE_EVENT_RECORD &mouse_event)
{
   switch (mouse_event.dwEventFlags)
   {
      case 0:

ModalCreate::ComputeSingleMouseClick(mouse_event.dwMousePosition.Y,
mouse_event.dwMousePosition.X);
         break;
      case DOUBLE_CLICK:

ModalCreate::ComputeDoubleMouseClick(mouse_event.dwMousePosition.Y,
mouse_event.dwMousePosition.X);
         break;
   }
}

void EventsController::ProcessMouseEvent(const MOUSE_EVENT_RECORD
&mouse_event)
{
   if (ModalDelete::IsLaunched())
      EventsController::ProcessMouseEventInModalDelete(mouse_event);
   else if (ModalCreate::IsLaunched())
      EventsController::ProcessMouseEventInModalCreate(mouse_event);
   else
      EventsController::ProcessMouseEventInMainGUI(mouse_event);
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Функциональная схема

ПРИЛОЖЕНИЕ В

(обязательное)

Блок схема алгоритма

ПРИЛОЖЕНИЕ Г

(обязательное)

Графический интерфейс

ПРИЛОЖЕНИЕ Д

(обязательное)

Ведомость документов