Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра Информатики

Дисциплина: Программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

ГЛУБОКАЯ МОДИФИКАЦИЯ СТАНДАРТНОЙ КОНСОЛИ WINDOWS И СОЗДАНИЕ НА ЕЁ БАЗЕ ФАЙЛОВОГО МЕНЕДЖЕРА

Студент гр. 053501 Слуцкий Н. С.

Руководитель:
ассистент кафедры Информатики Удовин И. А.

Минск 2021

Белорусский государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

«УТВЕРЖДАЮ»

Заведующий кафедрой Информатики

_____ Н. А. Волорова

«__» _____ 2021 года

ЗАДАНИЕ

по курсовой работе

студенту Слуцкому Н. С.

1. Тема курсовой работы « Глубокая модификация стандартной консоли Windows и создание на её базе файлового менеджера».
2. Дата защиты курсовой работы «__» _____ 2021 г.
3. Исходные данные для курсовой работы

3.1. Операционная система Windows

3.2. Язык программирования C++

3.3. Windows API

4. Содержание пояснительной записки

4.1. Введение.

4.2. Windows API

4.3. Подходы раздельного рендера компонентов и отдельного хранения состояния приложения

4.4. Реализация вышеупомянутых подходов посредством C++ и WinAPI в стандартной консоли Windows

4.5. Заключение.

4.6. Список используемой литературы.

5. Консультант по курсовой работе: Удовин И. А.

6. Дата выдачи задания «___» _____ 2021 г.

7. Календарный график выполнения курсовой работы.

7.1. к 15.03.2021 г. – 10% готовности

7.2. к 15.04.2021 г. – 40% готовности

7.3. к 15.05.2021 г. – 60% готовности

7.4. Оформление пояснительной записки к 02.06.2021 г. — 100% готовности

Руководитель курсовой работы _____ И. А. Удовин

Задание принял для исполнения _____ Н. С. Слуцкий

(дата и подпись студента)

ОГЛАВЛЕНИЕ

Введение

Цели и задачи

Целями данного курсового проекта являются:

1. Научить стандартную консоль операционной системы Windows 10 полноценно работать с пользовательскими событиями, производимыми с помощью клавиатуры или компьютерной мыши
2. На практике попытаться самостоятельно с нуля реализовать и использовать подход из веб-программирования, используемый с привлечением таких технологий как React, Redux, RxJS
3. Научить консоль выглядеть и вести себя как полноценное оконное десктопное приложение для операционной системы Windows (в том числе и за счёт выполнения пункта 1)

Потенциальные сложности при попытке создания приложения в консоли Windows

Стандартная консоль Windows — это программное обеспечение, которое используется для ввода команд и вывода уведомлений системы. Консоль еще называют командной строкой. Это специальная утилита, обеспечивающая поддержку прямой связи пользователя с операционной системой. Внешний вид стандартной консоли операционной системы Windows 10 представлен на рисунке 1. Максимально простой интерфейс лишь с текстовым полем и с отсутствием привычных элементов управления и визуализации:

- button,
- label,
- input.

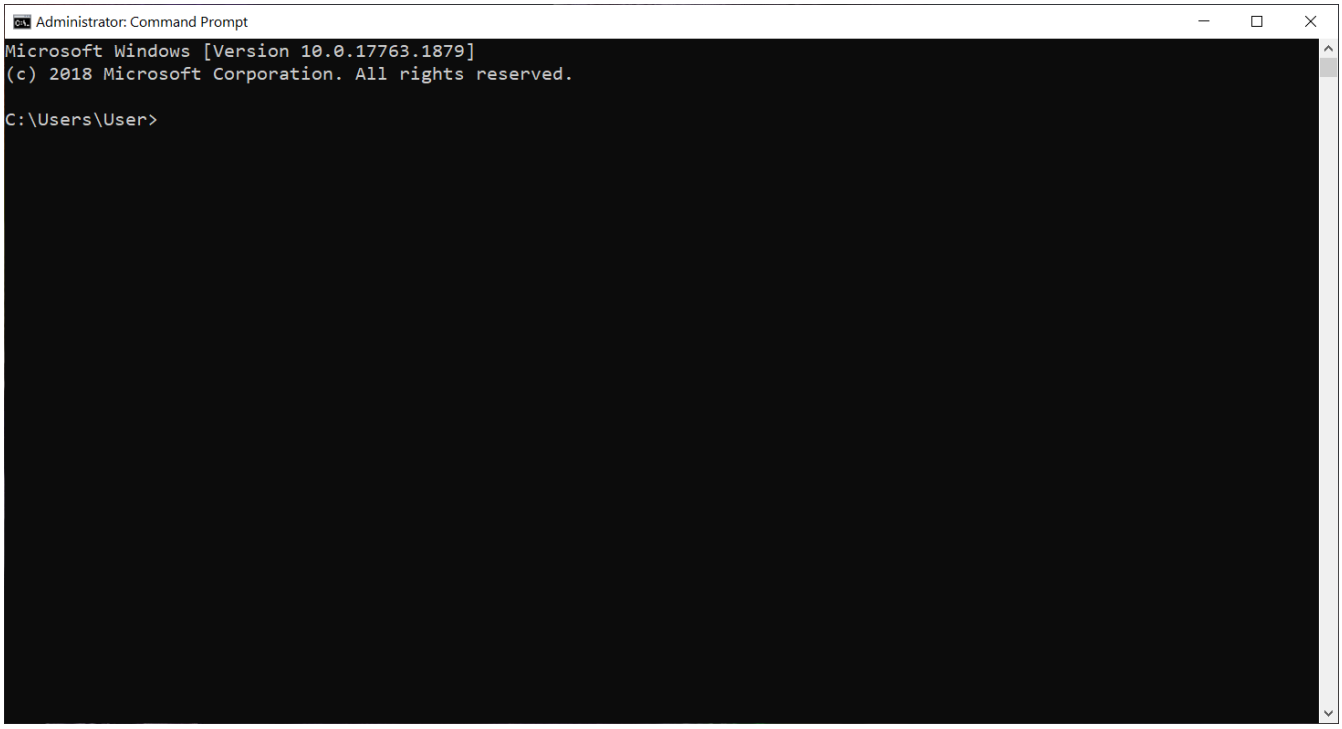

Рисунок 1 – Консоль (командная строка) в Windows 10

Перед и во время разработки курсового проекта анализировались некоторые потенциальные сложности при работе с консолью. Основные из них:

- невозможность представления (корректного вывода) всех символов Unicode внутри командной строки,
- невозможность реализовать классические компоненты привычного графического интерфейса (объекты Label, Button и так далее) в поле консоли,
- сложность в подборе хорошей цветовой палитры приложения в связи с ограниченной поддержкой цветов для цвета фона или текста в консоли,
- в принципе нераспространённость и почти неактуальность подобной схемы и подобных подходов к разработке десктопных приложений в последние два десятилетия.

Существующий пример полноценного приложения в консоли — FAR

Одним из самых популярных и актуальных примеров хорошего и очень функционального консольного приложения является FAR Manager [1]. 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.

Скриншот программы представлен на рисунке 2.

Рисунок 2 - Far Manager

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.

Windows API

Windows API (API — application programming interfaces) — общее наименование набора базовых функций интерфейсов программирования приложений операционных систем семейств Microsoft Windows корпорации «Майкрософт». 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.

История

Windows API [2] 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 представлен список всех версий 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 ныне версия. Базовые функции реализованы в динамически подключаемых библиотеках `kernel32.dll` и `advapi32.dll`; базовые модули графического интерфейса пользователя — 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.

Возможные сценарии использования

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.

Подход раздельного рендера компонентов и наличия состояния приложения

Раздельный рендеринг компонентов. Псевдоподход из React JS

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 популярном JavaScript-фреймворке React[3]. Пример разделения на компоненты веб-приложения представлен на рисунке 3.

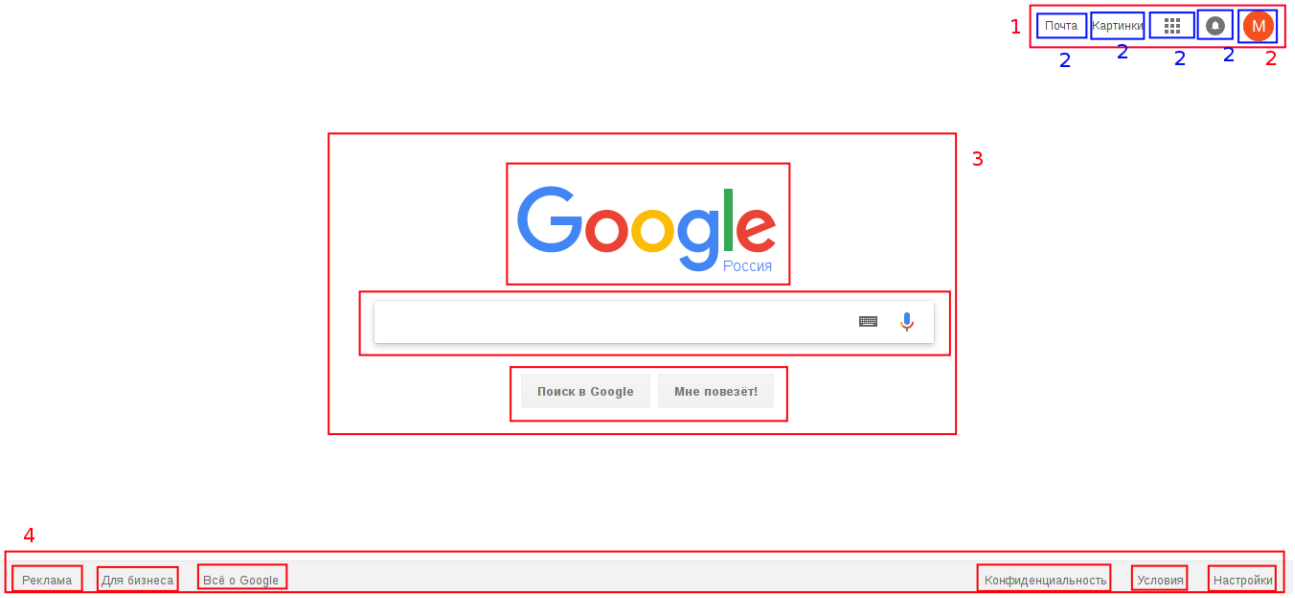

Рисунок 3 — пример разделения на компоненты

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. Пример такой связи состояния и компонентов изображён на рисунке 4. Состояние всего приложения хранится в едином хранилище. Единственный способ изменить состояние — это вызвать соответствующий метод у хранилища с необходимым описанием.

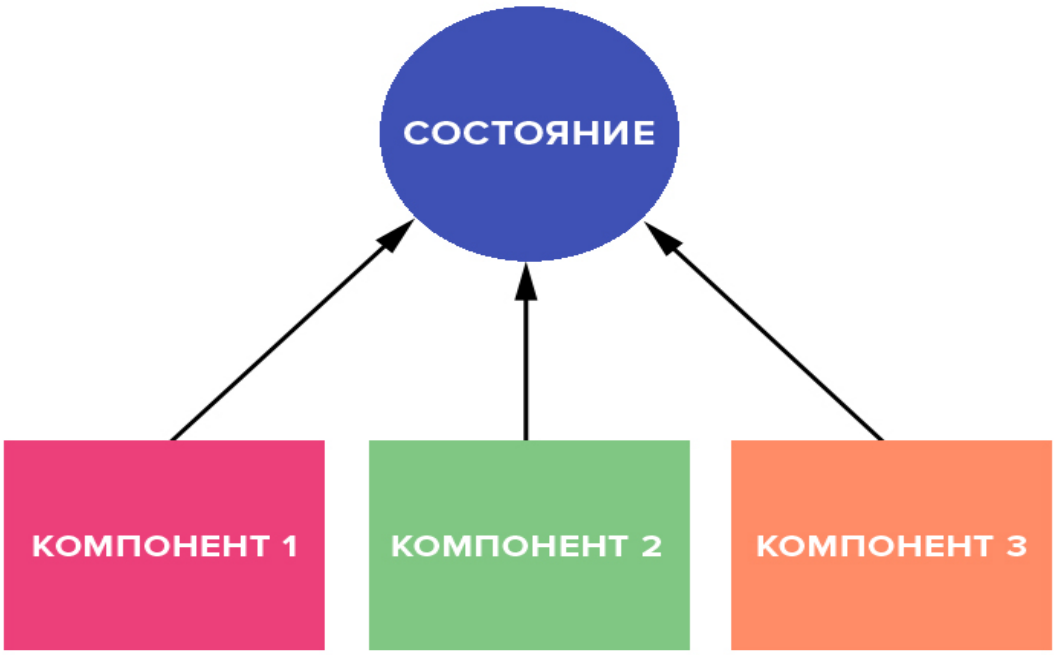

Рисунок 4 — пример связи компонентов и состояния

Бесконечное 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 (например, в ходе перехода в альбомную ориентацию на планшете или смартфоне).

Реализация этих подходов посредством C++, WinAPI в стандартной консоли

В данной курсовой работе в некоторой степени были использованы и реализованы все концепции и подходы, которые были описаны в предыдущем разделе. Далее приведено описание непосредственно внутри разрабатываемого приложения

Описание сущностей и общей схемы жизненного цикла приложения

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. На рисунке 5 изображена примерная схема взаимодействия этих классов. Её 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.

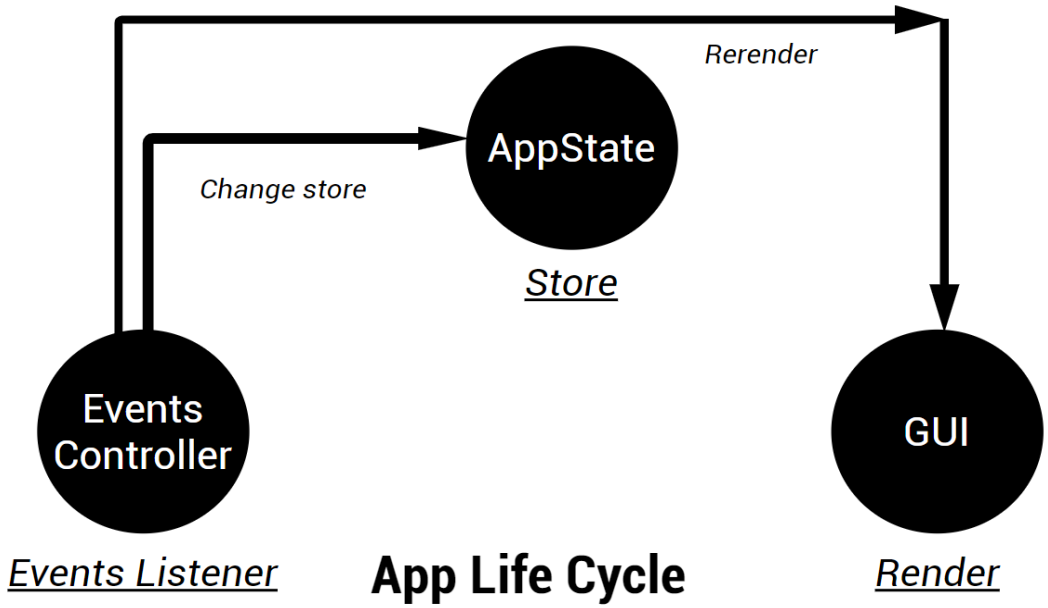

Рисунок 5 — общая схема жизненного цикла разработанного приложения

Состояние приложения AppState

AppState — один из трёх главных классов в приложении. Он отвечает за хранения 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.

Список полей для AppState приведён на рисунке 6. На этом и последующих скриншотах с кодом можно наблюдать одну из особенностей проекта — следование правильному официальному стилю написания кода — Google C++ Style Guide.

```
25    static std::stack<std::string> history;
26
27    static std::string current_directory;
28    static std::string parent_directory;
29
30    static std::map<size_t, std::filesystem::directory_entry> currently_rendered_with_coordinates; // according to screen coordinates
31    static std::vector<std::filesystem::directory_entry> currently_rendered_files_list; // just currently rendered list
32    static std::vector<std::filesystem::directory_entry> files_list; // all in current directory
33    static size_t render_from, render_to; // in global vector
34    static size_t current_position; // in currently_rendered
```

Рисунок 6 — структура полей состояния приложения

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 + кнопка 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. Часть из них для примера приведена на рисунке 7. На нём также можно наблюдать пример правильного названия константных полей в соответствии с Google C++ Style Guide [4].

```
25      static const std::string kWindowTitle;
26
27      static const std::string kFirstLineTexture;
28      static const std::string kSecondLineTexture;
29      static const std::string kArrowBackTexture;
30
31      static const size_t kMenuItemsCount = 4;
32      static const std::array<const std::string, GUI::kMenuItemsCount> kMenuItemsTitles;
33      static const std::array<const std::string, GUI::kMenuItemsCount> kMenuItemsKeys;
```

Рисунок 7 —некоторые константные поля в конфигурации интерфейса

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. Пример первого окна изображён на рисунке 8, скриншоте самого приложения.

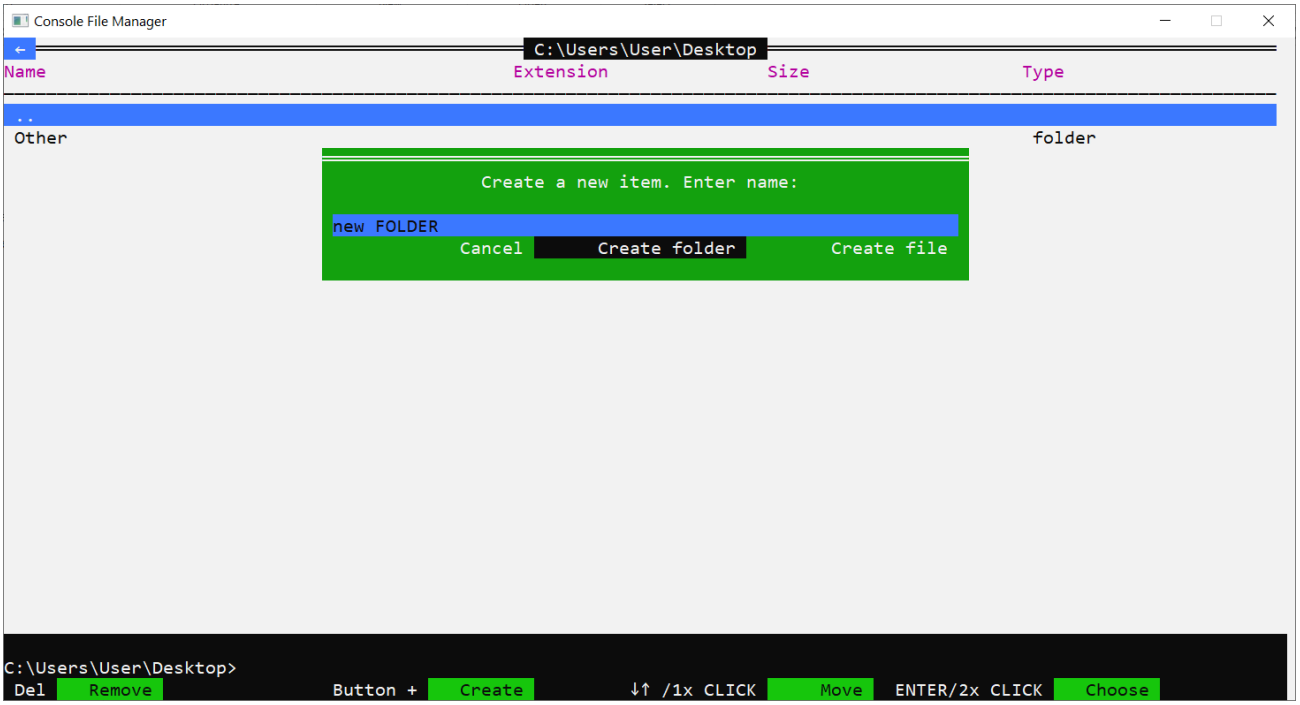

Рисунок 8 — скриншот приложения с демонстрацией всплывающего окна

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. На рисунке 9 изображены классы, которые имеют доступ к внутреннему устройству класса GUI.

```
18      private:
19          friend class EventsController;
20
21          friend class ModalDelete;
22
23          friend class ModalCreate;
```

Рисунок 9 — классы-"друзья" для GUI

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. Пример распределения событий для разных сущностей с игнорированием неактивных сущностей изображён на рисунке 10.

```
207 ⇆ void EventsController::ProcessMouseEvent(const MOUSE_EVENT_RECORD &mouse_event)
208   {
209       if (ModalDelete::IsLaunched())
210           EventsController::ProcessMouseEventInModalDelete(mouse_event);
211       else if (ModalCreate::IsLaunched())
212           EventsController::ProcessMouseEventInModalCreate(mouse_event);
213       else
214           EventsController::ProcessMouseEventInMainGUI(mouse_event);
215   }
```

Рисунок 10 — скриншот кода с распределением работы с событием компьютерной мышью

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).

Общий лаунчер для запуска всех сущностей

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). На рисунке 11, скриншоте из кода, изображена основная логика метода, который, вызываясь из main.cpp, запускает всё приложение полностью.

```
8  void FileManagerLauncher::Launch()
9  {
10     FileManagerLauncher::LoadConfiguration();
11     AppState::Launch(FileManagerLauncher::kStartDirectory);
12     GUI::Launch();
13     EventsController::RunEventLoop();
14 }
```

Рисунок 11 — процесс запуска всех сущностей в приложении

Заключение

Возникшие трудности при написании курсовой работы

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 полностью функционирующее приложение без видимых проблем.

Другие особенности проекта

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

- использование типа auto,
- использование range-based циклов для обхода коллекций,
- использование enum-классов вместо обычных перечислений,
- использование структуры std::array вместо обыкновенных статических массивов,
- использование анонимных функций (лямбда-выражений),
- активное использование STL-контейнеров,

В C++17 добавлено новое пространство имен std::filesystem. 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.

В течение всего написания проекта было проведено несколько рефакторингов кода, которые существенно улучшили его читаемость и эффективность. Автор следовал официальному и правильному стилю именования переменных, файлов, методов и так далее — обобщённо правильному codestyle. Все правила взяты из Google C++ Style Guide.

Было создано описание всех целей и результатов с демонстрацией работы в файле readme.md в репозитории на GitHub полностью на английском.

Результаты и оценка объёма проделанной работы

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.

Демонстрация работы программы приведена в репозитории на GitHub, а также на скриншотах рисунков 12 – 13.

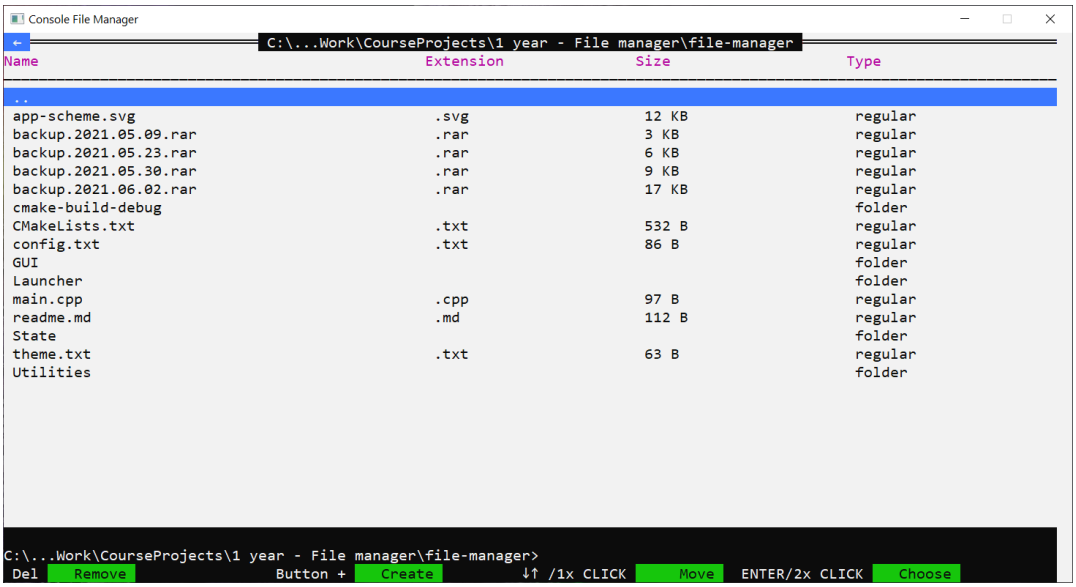

Рисунок 12 — скриншот из программы

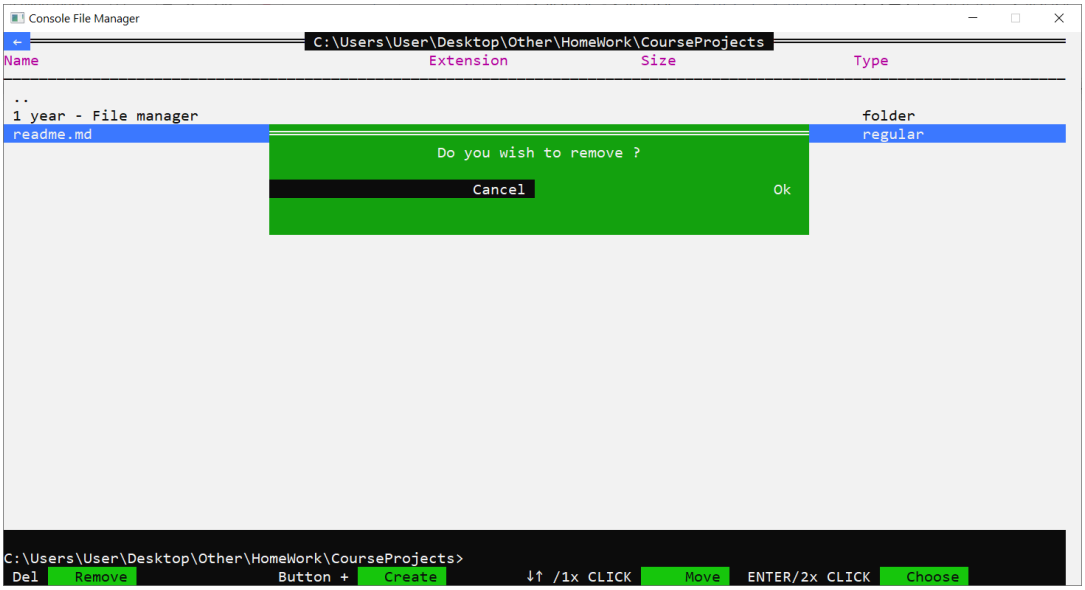

Рисунок 13 — скриншот из программы

Исходный код проекта находится в репозитории GitHub автора.

Статистические данные:

- Около 12 масштабных коммитов в GIT (всего 29, включая совсем мелкие)
- Более 1200 строчек чистого кода (без учёта пустых строк)

Список источников

Ссылки из текста:

- www.farmanager.com/ [1]
- en.wikipedia.org/wiki/Windows_API [2]
- reactjs.org/ [3]
- google.github.io/styleguide/cppguide.html [4]

В ходе написания курсового проекта использовались материалы:

- Книги:
  - “C++ 17 STL Стандартная библиотека шаблонов. 2018” – Я. Галовиц
  - “Эффективный и современный C++. 42 рекомендации по использованию C++11 и C++14. 2016” — С. Мейерс
- Интернет ресурсы:
  - Microsoft documentation C++ : docs.microsoft.com/en-us/cpp
  - C++ Forum: stackoverflow.com/
  - C++ Guide: en.cppreference.com/
  - Naming Style Guide: google.github.io/styleguide/cppguide.html#Naming