МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-40 05 02 Информационные системы и технологии

Направление специальности Издательско-полиграфический комплекс

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТ:**

по дисциплине «Объектно-ориентированное программирование»

Тема «Файловый менеджер»

Исполнитель

студент (ка) 2 курса группы 2 Юзефчик Дмитрий Петрович

(Ф.И.О.)

Руководитель работы ст. препод. Наркевич А.С.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель Наркевич А.С.

(подпись)

Минск 2018

Оглавление

[Введение 3](#_Toc514929749)

[**1** **Аналитическая часть** 4](#_Toc514929750)

[**1.1** **Постановка задачи и описание используемых технологий** 4](#_Toc514929751)

[**1.2** **Обзор аналогов** 4](#_Toc514929752)

[**2** **Проектирование системы** 7](#_Toc514929753)

[**2.1** **Выбор системной архитектуры** 7](#_Toc514929754)

[**2.2** **Составление диаграммы базы данных** 8](#_Toc514929755)

[**2.3** **Структура классов** 9](#_Toc514929756)

[**2.4** **Диаграмма использований** 10](#_Toc514929757)

[**1.** **Тестирование** 11](#_Toc514929758)

[**4** **Руководство пользователя** 15](#_Toc514929759)

[**Заключение** 18](#_Toc514929760)

[**Список источников:** 19](#_Toc514929761)

[**Приложение** 20](#_Toc514929762)

Введение

Файловый менеджер – самая часто используемая программа, он позволяет упростить манипуляцию с файлами и ускорить работу пользователя при помощи разнообразных функций. Также необходимость в данном ПО ставит следующую задачу: необходимо удовлетворить как можно больше разнообразных потребностей пользователей.

Целью курсового проекта является создание одной из самой важной программы на компьютере каждого пользователя. Он должен обеспечить комфортную и быструю работу с файлами. В результате должно получится приложение, которое сочетает в себе два противоположных качества: простоту и многофункциональность. Изучить при разработке курсового проекта архитектурный паттерн MVVM

Введение содержит общую информацию о направлении разработки проекта.

Аналитическая часть содержит постановку задачи и разбор аналогов, для расширения направления проекта.

В проектирование описывается структура приложения.

Руководство пользователя описывает взаимодействие с ПО.

В заключении подводится итог, исходя из тестирования.

Приложение содержит листинг ключевых моментов приложения.

1. **Аналитическая часть**
   1. **Постановка задачи и описание используемых технологий**

Программное средство файловый менеджер будет реализовывать следующие функции:

* + Выполнять открытие файлов.
  + Возможность проигрывать, просматривать, редактирование, перемещение, переименование, копирование, удаление файлов;
  + Изменение атрибутов и свойств, поиск файлов и назначение прав;
  + Возможность работы с несколькими файлами одновременно;
  + Отображение и хранение списка действий пользователя;
  + Архивирование файлов;

Данный программное обеспечение разрабатывалось в операционной системе Windows 10.

Данное ПО разрабатывалось в ОС Windows 10 в среде разработки Visual Studio 2017.

Приложение будет являться десктопным и не будет иметь онлайн сервисов, реализация на WPF 4.6, что позволит создать приятный и не требовательный к ресурсам PC интерфейс.

В программном средстве будет применена база данных, в которой фиксируются действия пользователя. Используется БД SQL lite, ее внедрение не скажется на производительности так как является более упрощенной версией, но этого будет достаточно для реализации необходимого.

Работа с БД будет осуществляться через EF6. Выбор в сторону EF6 был сделан, так как БД получается с 1 таблицей и малым количеством записей, разница в скорости работы между ADO.NET не будет такой существенной. В отличие от работы с MS SQL Server по отношению к SQLite EF 6 не поддерживает автоматическое создание базы данных и ее таблиц через Code First в соответствии со структурой моделей приложения. И в этом случае нужно самому создавать базу данных и ее таблицы. Для работы с СУБД использовалась DB Browser for SQLite.

Для стилизации приложения была выбрана библиотека MahApps.Metro

* 1. **Обзор аналогов**

Рассмотрим стандартный проводник Windows, интерфейс которого представлена на рисунке 1.1.

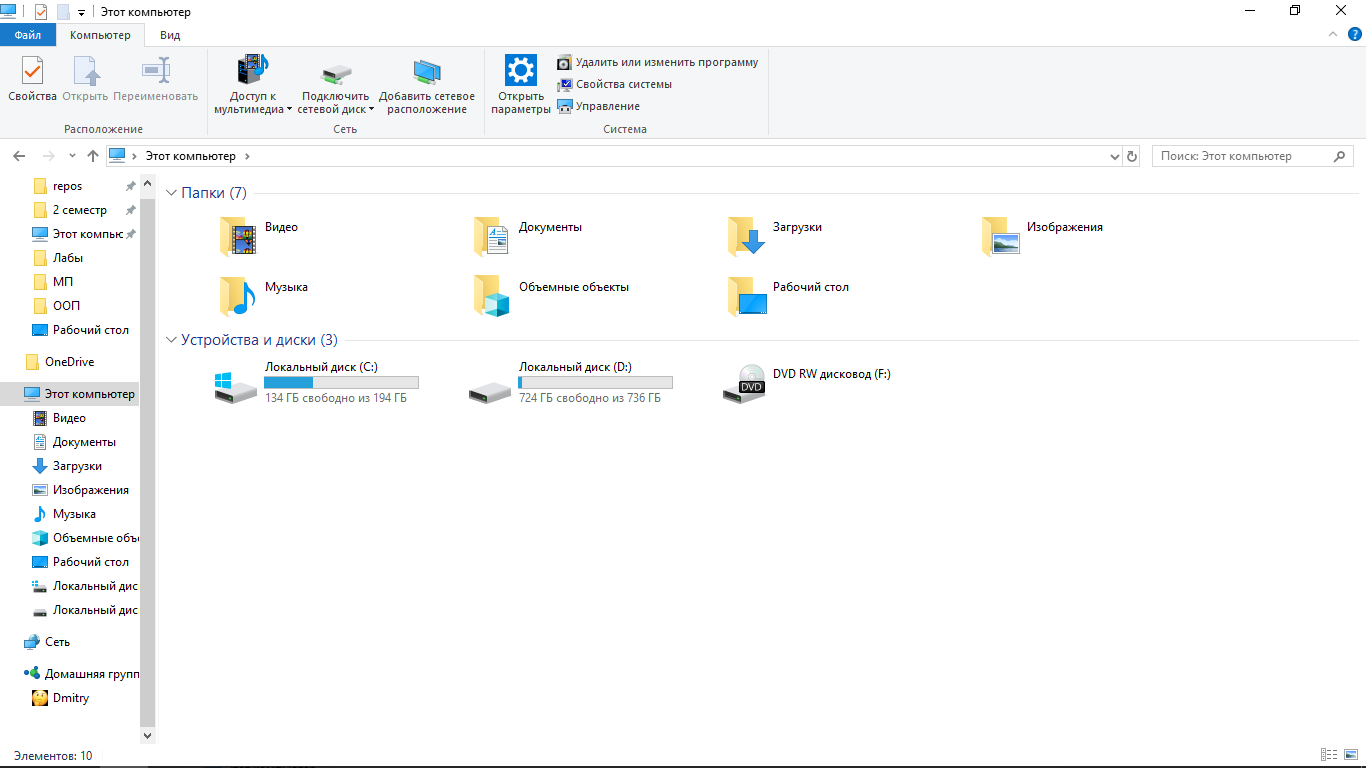


Рисунок 1.1 ­ интерфейс проводника Windows

Является типичным представителем навигационных файловых менеджеров. Интерфейс представляет из себя одно окно, в котором отображается список папок и файлов. Является очень простым, что крайне важно для таких массовых приложений, но и предоставляет весь необходимый функционал.

Также его популярность обусловлена и тем, что он является стандартным приложением.

Выделим основные плюсы:

* Очень прост в освоении;
* Легкий и приятный интерфейс;

Минусы:

* Интерфейс практически нельзя изменить;
* Имеет только базовые функции управления файлами.

Total Commander

Рассмотрим интерфейс Total Commander на рисунке 1.2.

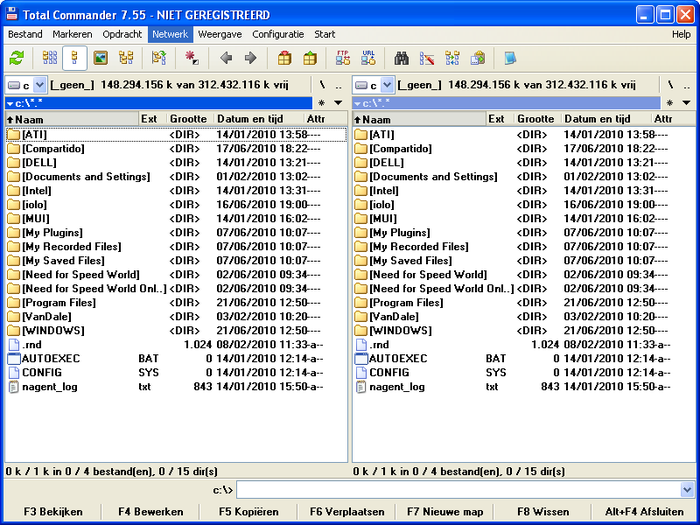


Рисунок 1.2 – интерфейс Total Commander

Total commander очень популярен и представляет собой совершенно другой класс файловых менеджеров, а именно, двух панельные. Имеет широкие функциональные возможности, настраиваемый интерфейс.

Плюсы:

* Имеет встроенный FTP клиент;
* Можно устанавливать пользовательские модификации;
* Очень много инструментов для работы с файлами;

Минусы:

* Интерфейс изначально кажется громоздким и сложным;

Проанализировав основных представителей, я сделал следующие выводы по своему будущему приложению.

Нужно еще отметить, что файловый менеджер это одна из базовых программ на компьютере, поэтому он должен экономно расходовать ресурсы PC, поэтому проект будет легковесным, иметь простой и понятный интерфейс, базовое количество функций для рядового пользователя.

1. **Проектирование системы**
   1. **Выбор системной архитектуры**

Существует несколько типов системной архитектуры проекта, в частности одноуровневая, двухуровневая, трехуровневая. Для трехуровневой архитектуры характерно наличие User Interface, Business Logic, Data Base. Данный тип архитектуры применятся в программном обеспечении, которое осуществляет работу, с какой либо базой данных. Одноуровневая системная архитектура не предусматривает деление на уровни. В данном случаи все действия реализации графического интерфейса и решения поставленной задачи выполняются на трех уровнях.

В данном проекте будет рационально использовать трехуровневую архитектуру. В этом случае разрабатываемая система разделяется на три уровня User Interface, Business Logic, Data Base (рис. 2.1).

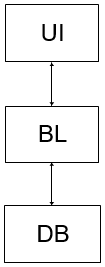


Рисунок 2.1 – трехуровневая архитектура

User Interface – уровень, отвечающий за реализацию графического интерфейса проекта.

Business Logic – уровень системной архитектуры, который отвечает за реализацию алгоритма работы памяти.

Data Base – уровень, хранящий модели, описывающие используемые сущности.

В данном курсовом проекте для реализации трёхуровневой архитектуры использовался паттерн MVVM, где в качестве слоя Business Logic используется т.н. ViewModel.

* 1. **Составление диаграммы базы данных**

Для данного приложения не требуется объемная БД так как она служит log-ом, в который заносятся все изменения в файловой системе и действия пользователя. Ее представление на рисунке 2.2.

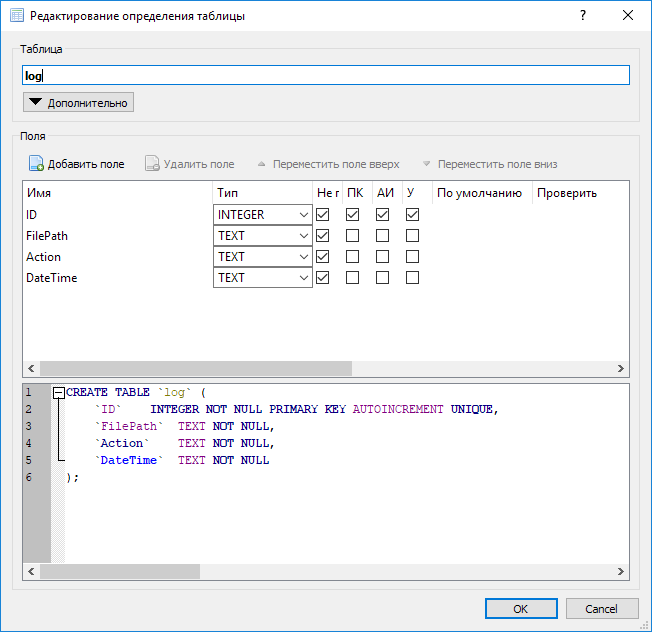


Рисунок 2.2 – таблица log

Поле ID является уникальным и служит для определения записи в таблице.

Action - отображает действие, произведенное над файлом.

FilePath – путь к файлу.

DateTime – содержит время начала операций.

Все поля не могут иметь нулевых значений.

* 1. **Структура классов**

Приложения реализовано при помощи архитектурного паттерна MVVM: каждое окно связано со своим классом, в котором описана бизнес-логика, каждая ViewModel посылает информацию через DataContext в базу данных.

Взаимосвязь всех классов представлена на рисунке 2.3.

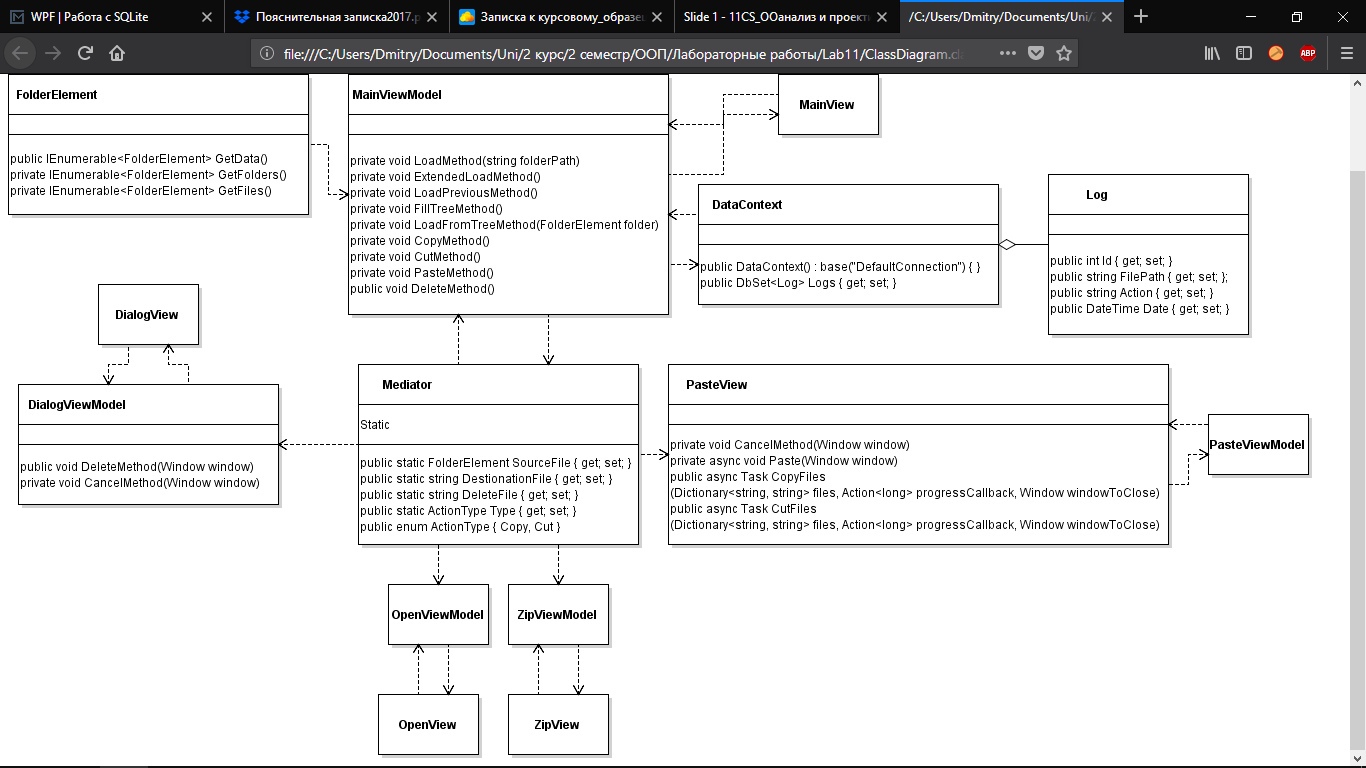


Рисунок 2.3 – структура классов

MainWindow отвечает за отображение главного окна приложения.

MainViewModel содержит все базовые методы, которые отвечают за взаимодействие с файлами (их добавление в коллекции, поиск, переход по папкам и т.д.) файлового менеджера и передачу корректных данных в интерфейс. Посылает команды на открытие других окон приложения.

FolderElement – ключевой класс. Описывает структуру файла и хранит всю информацию (полный путь, расширение, тип файла, размер и т.д.). Формирует итоговые коллекции файлов и папок.

DataContext – связывает бд с приложением, осуществляет отправку сообщение в бд.

Log – описывает структуру таблицы бд.

Mediator – связующее звено между всеми ViewModel. Статический класс, хранит необходимую информацию для осуществления остальных действий, передает ее в необходимую ViewModel.

DialogViewModel – отвечает за корректное удаление файлов.

DialogView – представляет собой диалоговое окно, с запросом на подтверждение удаления файла.

PasteViewModel – отвечает за копирование и перемещение файлов.

PasteView – окно отображает текущий процесс копирования/вставки, также имеет кнопку для отмены действия.

OpenViewModel – расширенное открытие файлов, позволяет выбрать программу для открытия.

ZipViewModel – отвечает за архивацию файлов.

PropertyViewModel – обрабатывает и изменяет свойства доступа и параметры файлов, отображает более подробную информацию о файле.

LogView – отображает лог-записи из таблицы.

* 1. **Диаграмма использований**

Варианты использования приложения показаны на рисунке 2.4.

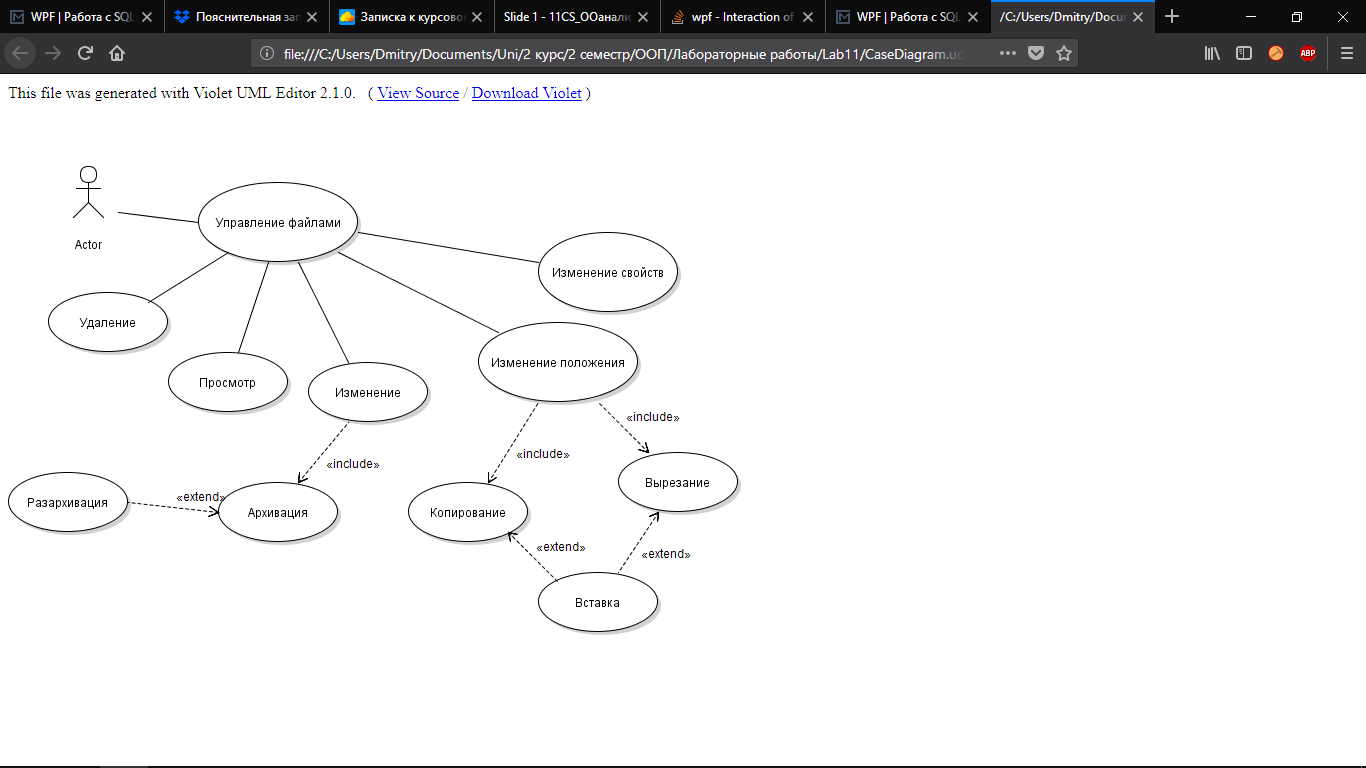


Рисунок 2.4 ­­­­– схема вариантов использования

Здесь можно оценить всю работу пользователя с приложением как отношения между актёрами и прецедентами. Где в качестве актёра выступает сам пользователь, а прецедентами являются возможности системы, благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат.

При запуске приложения пользователь сразу имеет главное окно, из которого можно выполнить любую операцию.

1. **Тестирование**

Тестирование производилось на операционной системе Windows 10.

В программе содержатся следующие окна: главное окно, диалоговое окно удаления, окно перемещения, окно архивации, окно свойств, окно вывода лога.

Тест главного окна.

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Отображение списка файлов из бокового списка дисков. | Выбрать доступный диск. | Данные были занесены в коллекцию. Пользователю была отображена корневая папка. |
| Отображение списка файлов из строки пути. | Ввести в строку адреса валидный путь к папке. | Данные были занесены в коллекцию. Пользователю была отображена папка |
| Отображение списка файлов при переходе по папкам. | Выбрать папку в списке. | Данные были занесены в коллекцию. Пользователю была отображена выбранная папка |
| Поиск в папке. | Ввести часть имени файла в строку поиска. | Были отображены все походящие файлы. |
| Кнопка «Назад». | Находясь не в корневой директории нажать на кнопку. | Был выполнен переход в родительскую папку. |
| Контекстное меню | Выбрать файл и вызвать контекстное меню. | Было отображено контекстное меню. Все пункты выполняли свои действия и открывали окна. |

Диалогового окна удаления.

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Подтверждение операции. | Нажать «Да» | Файл был удален из папки, окно закрылось. |
| Отмена операции. | Нажать «Нет» | Окно закрылось без удаления файла. |
| Проверка строки вывода информации | Вызвать окно удаления. | Строка отображала корректное действие «Удаление» и путь файла. |

Диалогового окна перемещения.

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Перемещение | Скопировать и вставить файл. | Копирование было завершено, окно закрылось |
| Отмена операции. | Скопировать и вставить файл, нажать отмену | Операция прерывается, не удаляя частично скопированный файл, в случае перемещения не удалился исходный файл, окно закрылось. |
| Проверка строки вывода информации | Вызвать окно копирования. | Строка отображала корректное действие и путь файла. |

Окно свойств

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Смена атрибутов. | Выбрать файл и поменять атрибуты. | У файла изменились атрибуты. |
| Проверка строки вывода информации | Вызвать свойств. | Строки отображают корректную информацию. |

Окна архивации.

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Архивация | Выбрать файл и выполнить архивацию. | Архивация была завершена, окно закрылось |
| Отмена операции. | Выбрать файл и выполнить архивацию., нажать отмену | Операция прерывается, окно закрылось. |
| Проверка строки вывода информации | Вызвать окно архивации. | Строка отображала корректное действие и путь файла. |

Окна деархивации.

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Деархивация | Выбрать файл и выполнить деархивацию. | Деархивация была завершена, окно закрылось |
| Отмена операции. | Выбрать файл и выполнить деархивацию., нажать отмену | Операция прерывается, окно закрылось. |
| Проверка строки вывода информации | Вызвать окно архивации. | Строка отображала корректное действие и путь файла. |

Окна вывода лога.

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Проверка вывода информации | Вызвать окно логов. | Из бд взята вся информация, она корректно отображена. |

Окна архивации.

|  |  |  |
| --- | --- | --- |
| **Название теста** | **Действие** | **Результат** |
| Архивация | Выбрать файл и выполнить архивацию. | Архивация была завершена, окно закрылось |
| Отмена операции. | Выбрать файл и выполнить архивацию., нажать отмену | Операция прерывается, окно закрылось. |
| Проверка строки вывода информации | Вызвать окно архивации. | Строка отображала корректное действие и путь файла. |

1. **Руководство пользователя**

Для инсталляции программы на персональный компьютер необходимо запустить установочный файл и указать папку для установки программы. После чего возможен запуск программы нажатием двойного клика левой мыши на исполняемом файле. Теперь программа запущена и готова к использованию рисунок 4.1

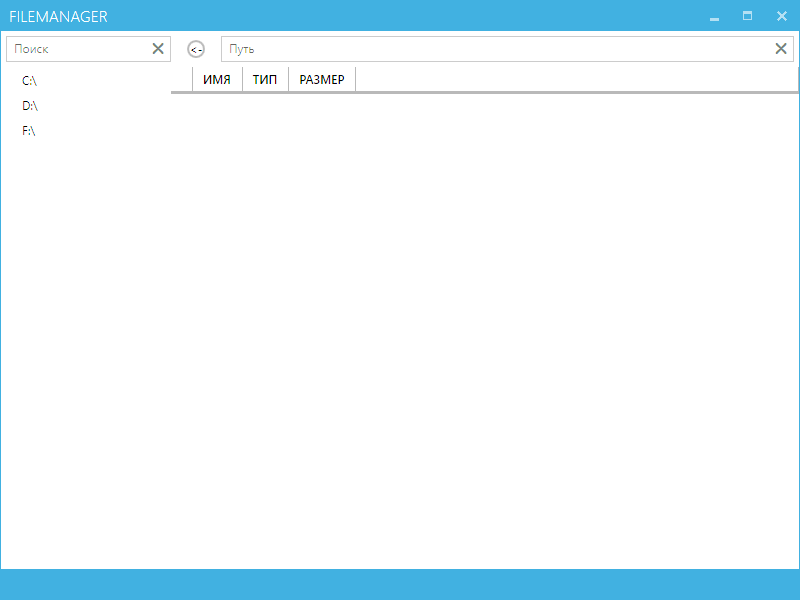


Рисунок 4.1 – интерфейс

На данном этапе пользователь может получить список файлов в корневой папке или перейти по нужному пути через строку «Путь», пример на рисунке 4.2.

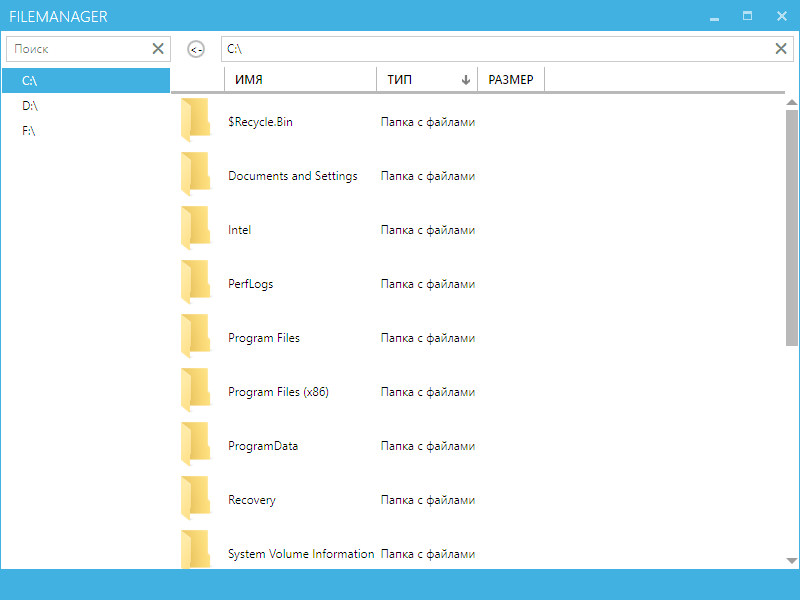


Рисунок 4.2 – интерфейс

Далее пользователь может вызвать контекстное меню правым щелчком по файлу рисунок 4.3.

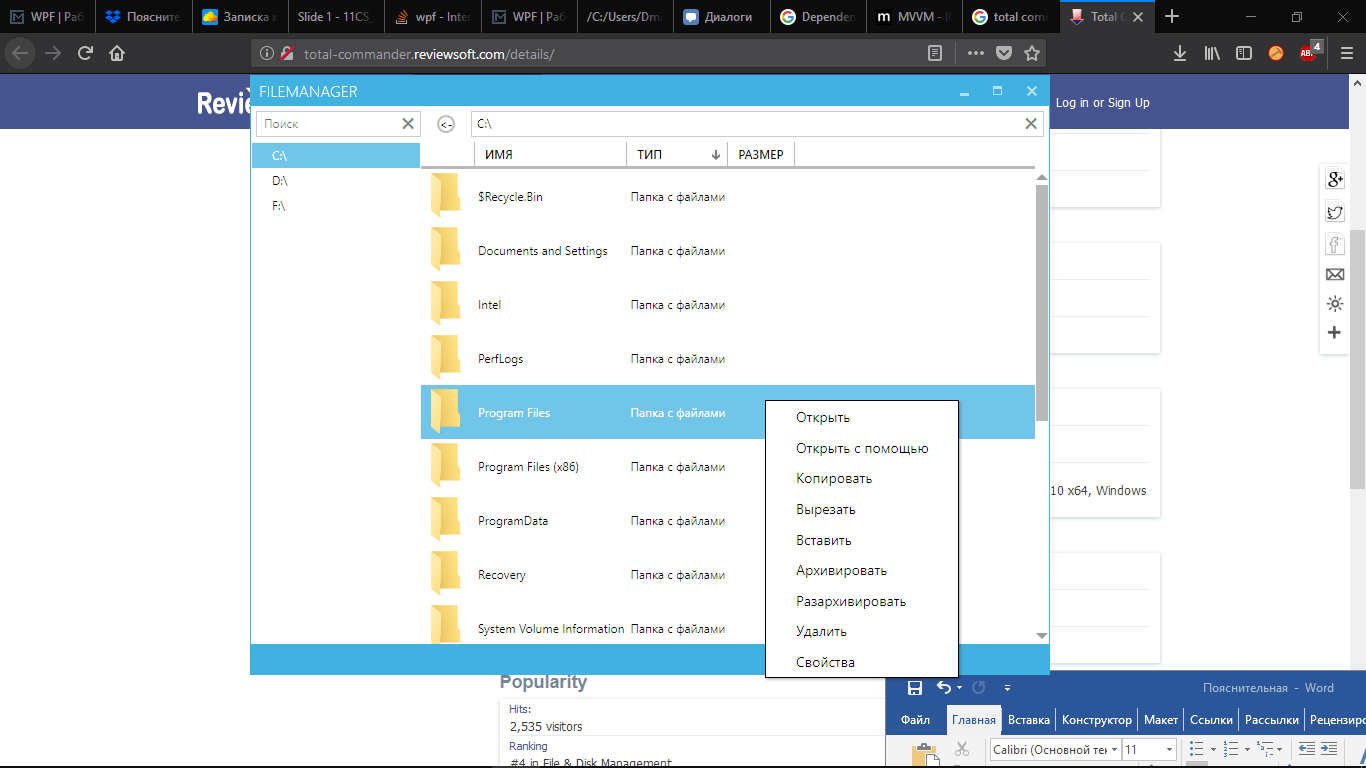


Рисунок 4.3 – вызов контекстного меню

Все функции однотипны и выполняются в один этап, кроме копирования и перемещения. Для их завершения нужно в конечной папке выбрать вставить, тогда появится окно с возможностью отмены, изображенное на рисунке 4.4.



Рисунок 4.4 – окно копирования

Здесь пользователь может изменять имя файлов и папок, просматривать путь и размер, изменять их параметры.

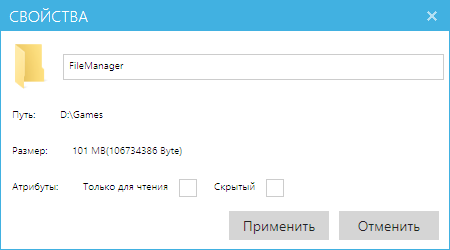


Рисунок 4.5 – окно свойств

**Заключение**

При разработке курсового проекта была создана программа для манипуляции с файлами. При разборе аналогов было выбрано направление в сторону простого и быстрого приложения. Оно имеет базовые функции, необходимые каждому пользователю. В результате его тестирования не было выявлено критических ошибок.

Для разработки программы использовался язык программирования C# и фреймворк WPF, для реализации бизнес логики использовался паттерн MVVM. В качестве СУБД была выбрана SQLite. В качестве среды разработки были использованы: Visual Studio 2017.

**Список источников:**

1. <https://metanit.com/sharp/wpf/21.1.php>
2. <https://msdn.microsoft.com/ru-ru/library/system.io.file(v=vs.110).aspx>
3. <https://msdn.microsoft.com/ru-ru/library/system.io.fileinfo(v=vs.110).aspx>
4. <https://mahapps.com/>
5. <https://msdn.microsoft.com/ru-ru/library/hh191443(v=vs.120).aspx>

**Приложение**

Получение списка файлов:

private IEnumerable<FolderElement> GetFiles()

{

return Directory.GetFiles(\_folderpath).Select(x => new FolderElement { Name = x, Type = ElementType.File });

}

private IEnumerable<FolderElement> GetFolders()

{

return Directory.GetDirectories(\_folderpath).Select(x => new FolderElement { Name = x, Type = ElementType.Folder });

}

public IEnumerable<FolderElement> GetData()

{

try

{

return GetFiles().Union(GetFolders());

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

return Enumerable.Empty<FolderElement>();

}

}

Выполнение асинхронного копирования папки:

public async Task FolderCopy(string from, string parentDirName, string to, Action<long> progressCallback, Window windowToClose)

{

await DirectoryCopy

(

from,

parentDirName,

to,

prog => Progress = prog

);

db.Logs.Add(new Log

(from,

"Файл копирован в " + to,

DateTime.Now.ToString("h:mm:ss tt"))

);

db.SaveChanges();

MainViewModel.GetInstance().AddToCollection(new FolderElement { FullName = to, Type = ElementType.Folder });

if (windowToClose != null)

{

windowToClose.Close();

}

}

public async Task DirectoryCopy(string from, string parentDirName, string to, Action<long> progressCallback)

{

if (!token.IsCancellationRequested)

{

DirectoryInfo directory = new DirectoryInfo(from);

DirectoryInfo[] dirs = directory.GetDirectories();

FileInfo[] files = directory.GetFiles();

if (!Directory.Exists(to + parentDirName))

{

Directory.CreateDirectory(to);

}

foreach (FileInfo file in files)

{

using (var outStream = new FileStream(Path.Combine(to, file.Name), FileMode.Create, FileAccess.Write, FileShare.Read))

{

using (var inStream = new FileStream(file.FullName, FileMode.Open, FileAccess.Read, FileShare.Read))

{

await inStream.CopyToAsync(outStream, 128, token);

}

}

progressCallback((long)(100));

}

foreach (DirectoryInfo subDir in dirs)

{

string newPath = Path.Combine(to, subDir.Name);

await DirectoryCopy(subDir.FullName, subDir.Name, newPath, prog => Progress = prog);

}

}

}

Поиск файлов:

public async void SearchMethod(string filter)

{

if (filter != "")

{

Regex regex = new Regex(@"\w\*" + filter + @"\w\*", RegexOptions.IgnoreCase);

if (currentFolderPath != null)

{

string searchPath = currentFolderPath;

File.Clear();

try

{

await RegexSearch(regex, searchPath);

}

catch (System.IO.IOException ex)

{

MessageBox.Show(ex.Message);

}

catch(Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

}

public async Task RegexSearch(Regex regex, string destinationPath)

{

DirectoryInfo directory = new DirectoryInfo(destinationPath);

DirectoryInfo[] dirs = directory.GetDirectories();

FileInfo[] files = directory.GetFiles();

BackgroundWorker worker = new BackgroundWorker();

foreach (FileInfo file in files)

{

if (regex.IsMatch(file.Name))

{

File.Add(new FolderElement { Name = file.FullName, Type = ElementType.File });

}

}

foreach (DirectoryInfo dir in dirs)

{

try

{

await RegexSearch(regex, dir.FullName);

}

catch (Exception)

{

continue;

}

if (regex.IsMatch(dir.Name))

{

File.Add(new FolderElement { Name = dir.FullName, Type = ElementType.Folder });

}

}

CurrentFolder = "";

}

Обращение к базе данных на примере архивации:

public async void ZipMethod()

{

try

{

Mediator.SourceFile = selectedElement;

Mediator.Type = Mediator.ActionType.Zip;

ZipWindow zipWindow = new ZipWindow();

zipWindow.Title = "Архивация";

await SendMessage

(

new Log

(Mediator.SourceFile.FullName,

"Попытка Архивации",

DateTime.Now.ToString("h:mm:ss tt"))

);

zipWindow.Show();

}

catch (NullReferenceException ex)

{

MessageBox.Show("Выберите файл");

}

}

public Task SendMessage(Log logInfo)

{

return Task.Run(() =>

{

db.Logs.Add(logInfo);

db.SaveChanges();

});

}