минобрнауки россии

федеральное государственное бюджетное

образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

|  |  |
| --- | --- |
| Институт (факультет) | Информационных технологий |
| Кафедра | Математическое и программное обеспечение ЭВМ |

КУРСОВАЯ РАБОТА

|  |
| --- |
| по дисциплине С# - программирование |

|  |  |
| --- | --- |
| на тему | Разработка программного обеспечения на языке C# |
|  | |

|  |
| --- |
| Выполнил студент группы 1ПИб-01-1оп-21 |
| *группа* |
| направления подготовки (специальности) |
| 09.03.04 Программная инженерия |
| *шифр, наименование* |
| Фомина Ксения Сергеевна |
| *фамилия, имя, отчество* |

|  |
| --- |
| Руководитель |
| Шаханов Н.И. |
| *фамилия, имя, отчество* |
| Доцент |
| *должность* |

|  |
| --- |
| Дата представления работы |
| «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г. |
|  |
| Заключение о допуске к защите |
|  |
|  |
|  |
|  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| количество баллов |
| Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Череповец, 2024

*Год*

Оглавление

[Введение 3](#_Toc168656056)

[1. Описание предметной области 4](#_Toc168656057)

[2. Организация работы в команде 5](#_Toc168656058)

[3. Обзор инструментов 8](#_Toc168656059)

[3.1. Выбор платформы 8](#_Toc168656060)

[3.2. Фреймворк Prism 8](#_Toc168656061)

[3.3. MVVM 9](#_Toc168656062)

[3.4. Внедрение зависимостей (Unity) 11](#_Toc168656063)

[3.5. Навигация 12](#_Toc168656064)

[3.6. Интеграция с XAML 14](#_Toc168656065)

[4. Проектирование приложения 16](#_Toc168656066)

[4.1. Разработка диаграммы классов и диаграммы пакетов с учетом инверсии управления (IoC) 16](#_Toc168656067)

[4.2. Внедрение зависимостей между пакетами 17](#_Toc168656068)

[5. Разработка базы данных 18](#_Toc168656069)

[5.1. Создание базы данных 18](#_Toc168656070)

[5.2. Разработка логики взаимодействия с базой данных 20](#_Toc168656071)

[Заключение 23](#_Toc168656072)

[Список литературы 24](#_Toc168656073)

[Приложение 1. Техническое задание 25](#_Toc168656074)

[Приложение 2. Руководство пользователя 32](#_Toc168656075)

[Приложение 3. Текст программы 37](#_Toc168656076)

# Введение

В эпоху цифровизации и активного развития технологий обработки изображений, возникает необходимость в создании эффективных инструментов для автоматизации процессов, связанных с анализом визуальной информации. Одним из ключевых аспектов такой автоматизации является автоматическая разметка изображений — процесс, при котором машина самостоятельно идентифицирует и классифицирует различные объекты на цифровых фотографиях и графических изображениях.

Цель данной работы — разработка программного приложения, способного автоматически распознавать и маркировать объекты на изображениях для их дальнейшего анализа или использования в системах, основанных на визуальных данных.

# Описание предметной области

Автоматическая разметка и обработка сгенерированных изображений — это комплексная область, занимающаяся изучением и разработкой методов и алгоритмов для создания, анализа и модификации визуального контента. Она охватывает процессы генерации изображений с помощью компьютерных технологий и последующей их автоматизированной разметки для различных целей, включая, но не ограничиваясь, машинное обучение, компьютерное зрение и автоматизированное тестирование.

Кадр обычно относится к изображению в контексте компьютерной графики или видео. Он представляет собой отдельное изображение, обычно с определенными размерами и содержанием.

Маска представляет собой изображение, которое используется для определения областей или пикселей в другом изображении. Обычно маска содержит информацию о прозрачности, маске или областях интереса в изображении.

Разметка — это процесс или результат размечивания данных, часто в контексте машинного обучения или анализа данных. Он включает в себя разметку объектов или атрибутов в данных для последующего использования в обучении моделей или анализе.

# Организация работы в команде

Правильная организация работы в команде важна для достижения общих целей проекта и повышения эффективности работы.

Для работы в команде существует большое количество различных инструментов.

Для отслеживания прогресса в решении задач использовалась доска задач в сервисе Notion — это относительно новый сервис для создания заметок и текстовых документов, списков дел, баз данных, таблиц, канбан-досок, баз знаний, ведения проектов и совместной работы. Наиболее значимыми в рамках работы стали возможности совместной работы и ведения досок задач (рис.1).

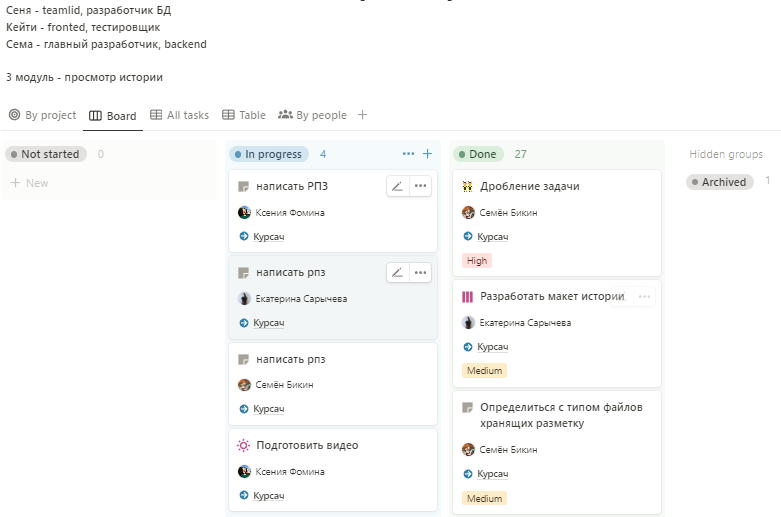


Рис.1. Доска задач в Notion

Лидер команды должен формулировать задачи, следить за их выполнением, назначать ответственных и инициировать организационные задачи с определенной периодичностью, для отслеживания прогресса и назначения новых задач, определения векторов последующих действий.

В качестве платформы для собраний был выбран Discord — это бесплатный мессенджер, который позволяет обмениваться голосовым, видео и текстовым чатом с друзьями, игровыми сообществами и разработчиками. Помимо возможностей организовывать собрания в голосовых чатах, в текстовых чатах можно делать разветвление на различные ветки (рис.2).

Кроме необходимости организовывать собрания и отслеживания прогресса в решении задач команде разработчиков нужна платформа для контроля версий проекта. Git — это консольная утилита, для отслеживания и ведения истории изменения файлов в проекте (рис.3). Чаще всего его используют для кода, но можно и для других файлов.

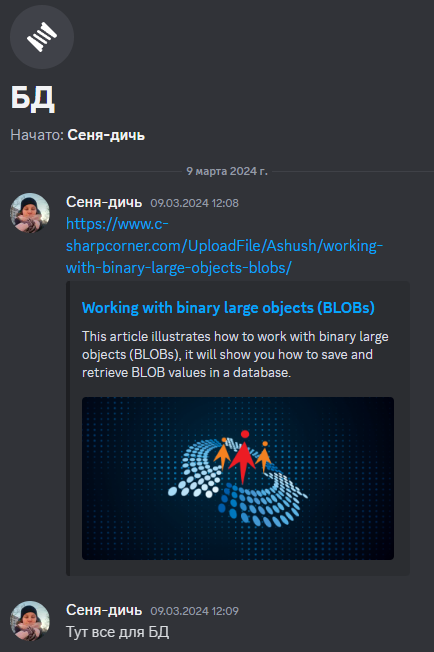


Рис.2. Ветка разработчика баз данных

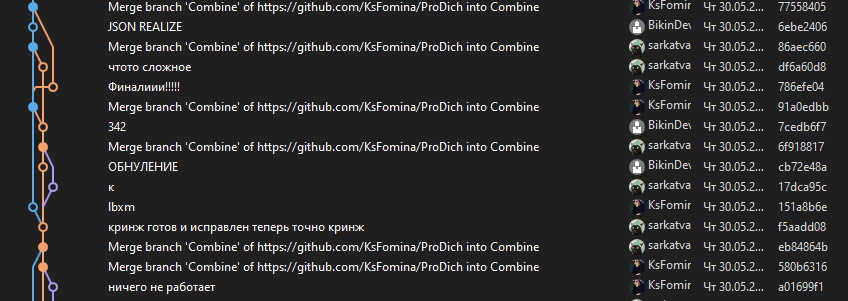


Рис.3. Действия на ветке

# Обзор инструментов

## Выбор платформы

В качестве среды разработки была выбрана Visual Studio, которая обладает рядом весомых преимуществ:

1. Интегрированная среда разработки (IDE): Visual Studio — это мощная и удобная в использовании IDE с богатым набором функций для разработки, отладки и тестирования кода.

2. Совместимость с C#: Visual Studio является родной средой для разработки на C#, что обеспечивает глубокую интеграцию с языком и .NET Framework, позволяя использовать все преимущества языка.

3. Продуктивность и эффективность: Visual Studio предлагает множество инструментов для повышения продуктивности, таких как IntelliSense (автодополнение кода), серверы сборки, профилировщики и инструменты для работы с базами данных.

4. Широкие возможности для командной работы: Visual Studio поддерживает интеграцию с системами контроля версий, такими как Git, и предлагает инструменты для совместной работы и непрерывной интеграции.

5. Расширяемость: IDE может быть расширена с помощью плагинов и расширений, что позволяет адаптировать среду под конкретные задачи и потребности проекта [2].

## Фреймворк Prism

Prism предназначено для разработчиков WPF приложений с многочисленными экранами, расширенным взаимодействием с пользователем, визуализацией данных, и со сложным взаимодействие представлений и бизнес логики. Эти приложения обычно взаимодействуют с многочисленными back-end системами и службами и, используя многоуровневую архитектуру, могут быть физически развернуты на нескольких уровнях.

Основная идея Prism заключается в разделении приложения на отдельные модули, которые могут быть разработаны и поддерживаться независимо друг от друга. Каждый модуль имеет свою функциональность и может быть интегрирован в основное приложение. Это позволяет создавать гибкие и масштабируемые приложения, которые легко изменяются и расширяются.

Prism предоставляет различные инструменты и функции для упрощения разработки. Некоторые из них включают модель просмотра (MVVM), инверсию управления (IoC), навигацию, внедрение зависимостей и обработку событий [3].

## MVVM

Паттерн MVVM (Model-View-ViewModel) позволяет отделить логику приложения от визуальной части (представления). Данный паттерн является архитектурным, то есть он задает общую архитектуру приложения. MVVM состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View) (рис.4).

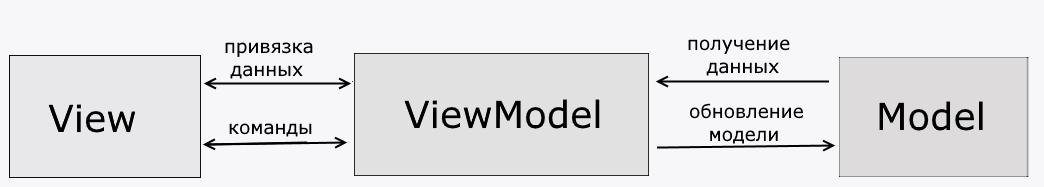


Рис.4. Паттерн MVVM

**Model** описывает используемые в приложении данные. Модели могут содержать логику, непосредственно связанную этими данными, например, логику валидации свойств модели. В то же время модель не должна содержать никакой логики, связанной с отображением данных и взаимодействием с визуальными элементами управления.

Нередко модель реализует интерфейсы INotifyPropertyChanged или INotifyCollectionChanged, которые позволяют уведомлять систему об изменениях свойств модели. Благодаря этому облегчается привязка к представлению, хотя опять же прямое взаимодействие между моделью и представлением отсутствует.

**View** или представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Применительно к WPF представление — это код в xaml, который определяет интерфейс в виде кнопок, текстовых полей и прочих визуальных элементов.

**ViewModel** или модель представления связывает модель и представление через механизм привязки данных. Если в модели изменяются значения свойств, при реализации моделью интерфейса INotifyPropertyChanged автоматически идет изменение отображаемых данных в представлении, хотя напрямую модель и представление не связаны.

Итогом применения паттерна MVVM является функциональное разделение приложения на три компонента, которые проще разрабатывать и тестировать, а также в дальнейшем модифицировать и поддерживать [1].

Связь между этими тремя составными частями осуществляется с помощью двух ключевых механизмов:

• Связывание данных (Data Binding): Механизм связывания данных позволяет установить связь между свойствами модели представления и элементами интерфейса представления. Таким образом, изменения в модели представления автоматически отражаются в представлении, и наоборот.

• Команды (Commands): Команды позволяют модели представления определить действия, которые могут быть выполнены в ответ на пользовательские действия. Например, кнопка может быть привязана к команде в модели представления, и при нажатии кнопки будет выполнена соответствующая команда. Команды позволяют централизованно управлять логикой приложения и обрабатывать пользовательские действия.

Prism предоставляет базовые классы и инфраструктуру для реализации MVVM:

• BaseBindableBase — это базовый класс в Prism, который реализует интерфейс INotifyPropertyChanged. Он предоставляет функциональность для уведомления представления об изменении свойств модели представления. При изменении значения свойства, модель представления автоматически уведомляет представление о необходимости обновиться.

• DelegateCommand — это класс, который реализует интерфейс ICommand и предоставляет механизм для определения команд, которые могут быть выполнены из представления. DelegateCommand позволяет определить методы, которые будут вызываться при выполнении команды, а также определить условия, при которых команда может быть выполнена (CanExecute).

• EventAggregator — это класс, предоставляемый Prism, который служит для обмена сообщениями (событиями) между различными компонентами приложения. Он предоставляет механизм публикации и подписки на события, что позволяет модулям и компонентам взаимодействовать между собой без явной привязки. EventAggregator помогает реализовать слабую связанность между различными частями приложения [2].

## Внедрение зависимостей (Unity)

Внедрение зависимостей (Dependency Injection) — это метод, который используется для управления связями между различными частями программы. Вместо того чтобы жестко кодировать связи, программа получает доступ к другим компонентам через интерфейсы или абстрактные классы. Это упрощает тестирование и поддержку кода, а также делает программу более гибкой и масштабируемой.

Один из инструментов для реализации внедрения зависимостей в .NET-приложениях — это Unity Application Block (Unity). Unity — это контейнер внедрения зависимостей, разработанный компанией Microsoft. Он предоставляет мощные функциональные возможности для управления связями между компонентами в программе.

Некоторые ключевые особенности и концепции Unity включают:

1. Контейнер внедрения зависимостей: Unity представляет собой контейнер, который управляет созданием, разрешением и уничтожением связей между компонентами в программе. Он обеспечивает реализацию принципа инверсии управления, где компоненты не создают и не управляют своими связями самостоятельно, а полагаются на контейнер для этого.

2. Регистрация зависимостей: перед использованием Unity необходимо зарегистрировать связи. Регистрация указывает Unity, какой тип связи должен быть создан и как ее разрешить. Связи могут быть зарегистрированы для интерфейсов, абстрактных классов или конкретных типов.

3. Разрешение зависимостей: Unity автоматически разрешает связи между компонентами во время выполнения программы. Когда компонент запрашивает связь, Unity обеспечивает ее создание и передачу в компонент. Разрешение может быть выполнено с помощью конструктора, свойства или метода компонента.

4. Жизненный цикл зависимостей: Unity позволяет управлять жизненным циклом связей. Можно указать, как долго должна существовать связь — одиночная инстанция (singleton), новая инстанция для каждого разрешения (transient) или другие варианты.

5. Автоматическое разрешение зависимостей: Unity может автоматически разрешать связи, анализируя типы и их связи через рефлексию. Это означает, что вам не нужно явно указывать каждую связь, Unity может найти и разрешить их автоматически.

Таким образом, Unity является популярным инструментом в .NET-сообществе и предоставляет гибкий и мощный механизм для реализации внедрения зависимостей в программах. Он позволяет создавать модульные и слабо связанные системы, облегчает тестирование и повторное использование кода, а также способствует более гибкому и расширяемому проектированию программ [3].

## Навигация

Навигация в Prism предоставляет средства для управления переходами между представлениями в приложении. Она позволяет определять набор экранов и переходить между ними в ответ на действия пользователя или внутренние события.

Prism предлагает различные способы навигации, включая навигацию с использованием регионов (Region-based navigation). Навигация на основе регионов основана на определении областей (регионов) на пользовательском интерфейсе, в которых представления могут быть отображены. При использовании этого способа навигации в Prism используются следующие термины:

• Регион (Region) —это область на пользовательском интерфейсе, в которой может быть отображено одно или несколько представлений. Регион может быть определен на различных элементах пользовательского интерфейса, таких как окна, страницы или пользовательские элементы управления. Регионы Prism, по большей части, являются именованными заполнителями, в которых отображаются представления. Любой элемент управления может быть объявлен как регион, с помощью простого добавления присоединенного свойства RegionName [3].

• Менеджер регионов (Region Manager) — это инфраструктура в Prism, которая отвечает за управление регионами в вашем приложении. Менеджер регионов позволяет добавлять, удалять и заменять представления в регионах, а также управлять их жизненным циклом.

Часто возникает необходимость участия представлений и моделей представлений в навигации. Интерфейс INavigationAware позволяет это сделать.

Интерфейс INavigationAware является частью инфраструктуры навигации в Prism и предоставляет методы для уведомления представления о своем состоянии навигации. Этот интерфейс позволяет представлениям реагировать на различные этапы навигации, такие как инициализация, активация, деактивация и завершение навигации.

Вот некоторые из методов, определенных в интерфейсе INavigationAware:

• OnNavigatedTo(NavigationContext navigationContext): Этот метод вызывается, когда представление активируется и отображается в результате навигации. Он принимает объект NavigationContext, который содержит информацию о навигации, такую как параметры, переданные во время навигации. В методе OnNavigatedTo можно выполнить необходимую логику и инициализацию для представления на этапе активации.

• OnNavigatingFrom(NavigationContext navigationContext, ref bool isNavigationCancelled): Этот метод вызывается перед тем, как представление будет деактивировано и покинуто в результате навигации. Он также принимает объект NavigationContext, который содержит информацию о навигации. Параметр isNavigationCancelled является ссылкой на логическую переменную, которую можно использовать для отмены навигации, если требуется. В методе OnNavigatingFrom можно выполнять необходимую проверку и подготовку представления перед его деактивацией.

• OnNavigatedFrom(NavigationContext navigationContext): Этот метод вызывается после того, как представление было деактивировано и покинуто в результате навигации. Он также принимает объект NavigationContext, содержащий информацию о навигации. В методе OnNavigatedFrom можно выполнять очистку ресурсов или сохранение состояния представления перед его завершением.

Использование интерфейса INavigationAware позволяет представлениям взаимодействовать с процессом навигации в Prism и реагировать на изменения состояния навигации. Это может быть полезно, например, для загрузки данных при активации представления, сохранения изменений перед его деактивацией или выполнения других действий, связанных с жизненным циклом представления в рамках навигации.

## Интеграция с XAML

Prism обеспечивает глубокую интеграцию с XAML, что позволяет разрабатывать приложения с использованием декларативного подхода к созданию пользовательского интерфейса.

Основной способ интеграции с XAML в Prism — это использование паттерна MVVM (Model-View-ViewModel) для разделения логики приложения и пользовательского интерфейса. С помощью MVVM разработчик может создать отдельные классы ViewModel для каждого представления, которые связываются с соответствующими XAML-файлами представлений.

В XAML-файлах представлений Prism предоставляет дополнительные элементы и функциональность, которые облегчают интеграцию с фреймворком. Некоторые из этих функций включают:

• Регионы: Prism предоставляет Region элемент, который может быть добавлен в XAML-разметку, чтобы определить область, где должно размещаться представление. Регионы позволяют динамически добавлять и удалять представления из заданных областей внутри пользовательского интерфейса.

• ViewModelLocator: это паттерн, используемый в Prism для обеспечения связи между представлениями и соответствующими ViewModel. Он предоставляет механизм для нахождения и создания экземпляров ViewModel для каждого представления. ViewModelLocator позволяет разработчикам объявить связь между представлением и его ViewModel в XAML-разметке. Обычно в ресурсах приложения или ресурсах конкретного модуля определяется ViewModelLocator.

# Проектирование приложения

В рамках разработки WPF приложения были организован репозиторий git, разработаны база данных и класс для работы с ней, а также объединены все части, разработанные другими участниками команды.

## Разработка диаграммы классов и диаграммы пакетов с учетом инверсии управления (IoC)

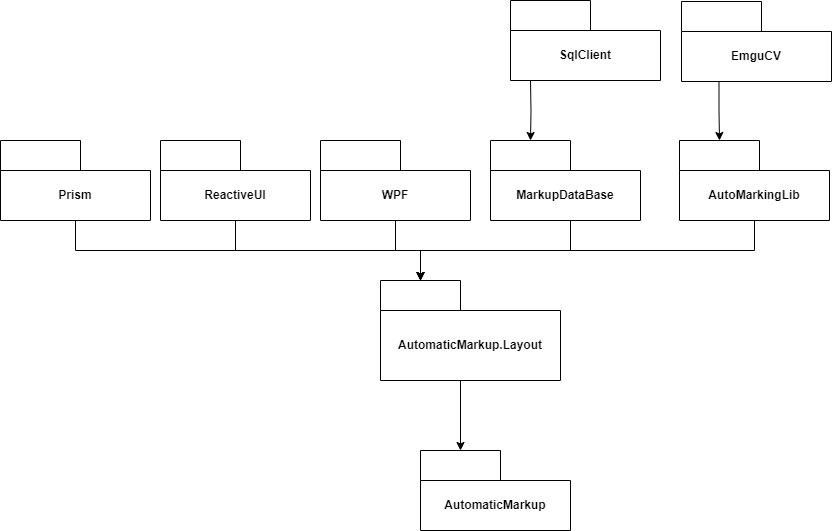


Рис.5. Диаграмма пакетов

На диаграмме классов (рис.5) отражены созданные пакеты и сторонние пакеты, которые использовались при написании программы. Готовые внешние пакеты Prism, ReactiveUI, WPF, SqlClient, EmguCV подключаются к другим пакетам, которые были разработаны участниками команды.

Пакет AutomaticMarkup – это оболочка, которая содержит основные инструкции запуска приложения. Пакет AutomaticMarkingLib содержит логику разметки. Пакет DataBase содержит логику работы с базой данных. AutomaticMarkup.Layout - пакет, в котором расписана разметка и взаимодействие с другими пакетами.

## Внедрение зависимостей между пакетами

Чтобы связать пакеты друг с другом в файле настройки пакета необходимо прописать подключение (рис.6), также это можно сделать, используя возможности обозревателя решений, код сгенерируется автоматически.

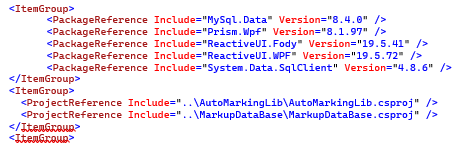


Рис.6. Установление связей для пакета AutomaticMarkup.Layout

Аналогично и для других пакетов. Чтобы подключить различные модули необходимо в пакете AutomaticMarkup в классе App (рис.7) его прописать. Там же прописывается начальное представление.



Рис.7. Класс App пакета AutomaticMarkup

# Разработка базы данных

## Создание базы данных

База данных — это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно управляется системой управления базами данных (СУБД).

SQL Server Management Studio (SSMS) — это интегрированная среда для управления любой инфраструктурой SQL, от SQL Server до баз данных SQL Azure. SSMS предоставляет средства для настройки, наблюдения и администрирования экземпляров SQL Server и баз данных. Используйте SSMS для развертывания, мониторинга и обновления компонентов уровня данных, используемых приложениями, и создания запросов и скриптов [5].

В рамках программы для работы с базой данных используется библиотека SqlClient.

SQL-запросы необходимы для работы с информацией из базы данных. Это может быть внесение, извлечение, сортировка, удаление и ряд других операций. При этом не указывается способ осуществления запрашиваемого действия [6].

База данных (рис.8-9) была создана с помощью следующего SQL-запроса:

Create table history(

id int IDENTITY(1,1) NOT NULL,

name\_file varchar(20) NOT NULL,

time\_create time NOT NULL,

date\_create date NOT NULL,

file\_marking image NOT NULL,

file\_source image NOT NULL,

file\_mask image

marki image

);

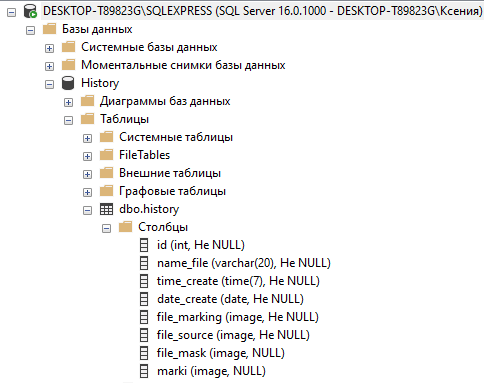


Рис.8. База данных в SSMS

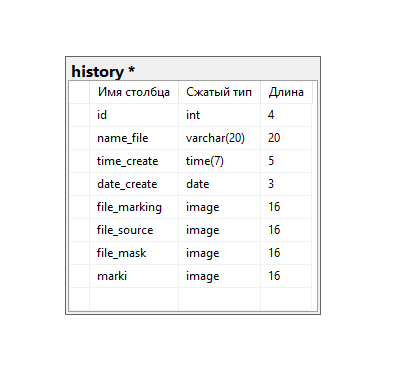


Рис.9. Диаграмма базы данных

На диаграмме базы данных обычно отображаются все таблицы и их отношения, но поскольку в базе данных истории одна таблица, то и на диаграмме (рис.9) сущность тоже одна, а отношения отсутствуют. В ней отражены все атрибуты, их типы и размер. Создавать большее количество таблиц в рамках данной задаче имеет малый смысл, так как это усложнит взаимодействие с базой данных. Эта база данных предназначена только для хранения информации.

## Разработка логики взаимодействия с базой данных

База данных отображается в окне истории (рис.10), которое открывается после нажатия на кнопку «History». Пользователю приложения видны только общая информация, ему не видны поля, где хранятся изображения и разметка.

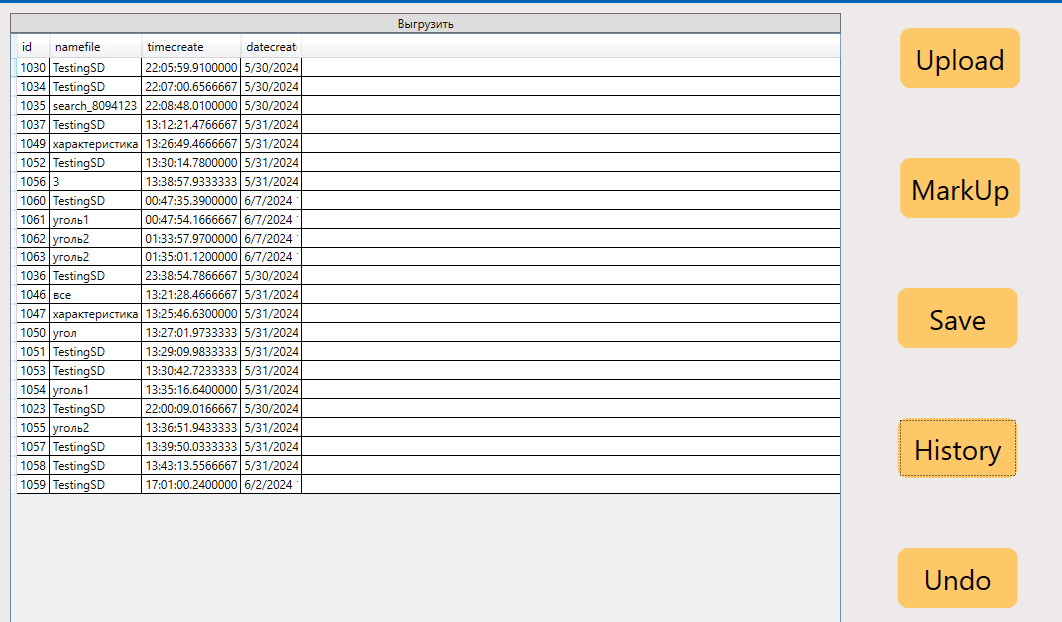


Рис.10. Окно истории с загруженной базой данных

Для того, чтобы подключить базу данных к программе используется следующий код:

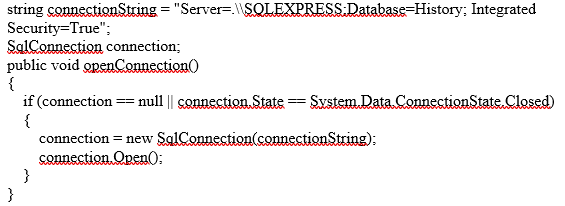


Рис.11. Код подключение базы данных

База данных хранит уникальный Id разметки, имя исходного файла, время создания разметки, дата создания разметки, саму разметку, исходный файл и маску изображения. Используя базу данных, можно восстановить, обратиться к любой ранее созданной разметке, даже если пользователь забыл сохранить результаты или удалил файлы. Новая запись в базу данных добавляется сразу после разметки независимо от пользователя.

Подключение к базе данных и функции с этим связанны находятся в пакете MarkupDataBase. Код для добавления новой записи в базу данных при создании новой разметки:

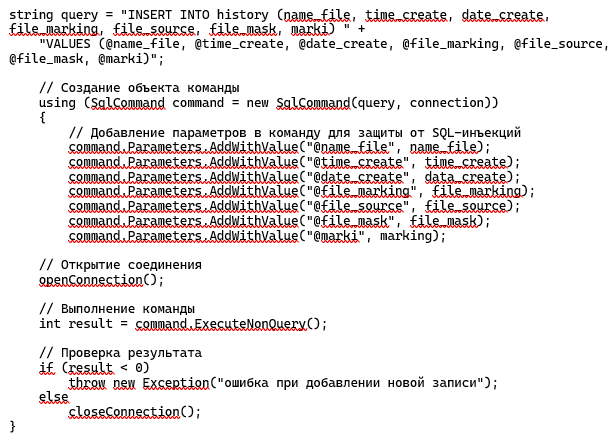


Рис.12. Код добавления записи в базу данных

Более подробный код, связанный с базой данных, представлен в листинге (прил.3). Описание функций класса BaseConnection представлены в табл.1.

Таблица 1

Функции класса BaseConnection

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Атрибуты | Описание |
| openConnection | void | - | Функция, которая открывает соединение с базой данных |
| closeConnection | void | - | Функция, которая закрывает соединение с базой данных |
| AddData | void | string name\_file, DateTime time\_create, DateTime data\_create, byte[] file\_marking, byte[] file\_source, byte[] file\_mask, byte[] marking | Функция добавления новых записей в базу данных |
| GetHistory | (byte[] DataOrig, byte[] DataMark, byte[] DataMask ) | int Id | Функция получения файлов из истории |
| start | void | DataTable dataTable | Функция, выгружающая историю в таблицу |

Тестирование всех модулей производилось тестировщиком Сарычевой Екатериной.

# Заключение

В ходе выполнения работы было создано приложение на c# для автоматической разметки, которое на вход принимает сгенерированные изображение и его маску и создает на их основе разметку, а также изображение с созданной разметкой. Помимо этого, программа поддерживает историю, с помощью которой можно восстановить размеченные изображения, при необходимости.

Разработка велась в команде из 3 человек, каждый выполнял задачи, соответствующие определенной роли. Применены были соответствующие инструменты для совместной работы: был создан репозиторий git, доска задач в Notion и регулярно организовывались собрания команды.

# Список литературы

1. Паттерн MVVM. Электронный ресурс: https://metanit.com (дата обращения 10.05.2024)
2. Visual Studio. Электронный ресурс https://metanit.com/sharp/tutorial (дата обращения 10.05.2024)
3. Руководство разработчика Prism — часть 8.1, навигация на основе представлений (View-Based Navigation). Электронный ресурс: https://habr.com/ru/articles/182052/ (дата обращения: 20.05.2024)
4. Ключевые механизмы. Электронный ресурс https://habr.com (дата обращения 15.05.2024)
5. SQL Server Management Studio (SSMS). Электронный ресурс https://learn.microsoft.com (дата обращения 15.05.2024)
6. SQL-запросы. Электронный ресурс: https://gb.ru (дата обращения 18.05.2024)
7. Ершов, Е.В. Методика и организация самостоятельной работы: учебное пособие. [Текст] / Ершов Е.В., Виноградова Л.Н., Селивановских В.В. // Череповец: ЧГУ, 2015. – 243 с.

# Приложение 1. Техническое задание

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий

(наименование структурного подразделения)

Кафедра математического и программного обеспечения ЭВМ

(наименование кафедры)

С# - программирование

(наименование дисциплины в соответствии с учебным планом)

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ,

д.т.н., профессор Ершов Е.В.

«\_\_\_» апреля 2024 г.

Разработка программного обеспечения на языке C#

Техническое задание на курсовую работу

Листов \_\_

Руководитель: Шаханов Н.И.

Исполнитель: студент гр. 1ПИб-01-01оп-21

Фомина Ксения Сергеевна

2024 год

Введение

Курсовая работа посвящена командной разработке программы для автоматической на языке С#, которая будет иметь возможность обрабатывать входные данные (изображение и маска) и создавать по ним разметку, сохранять результаты и поддерживать историю.

Данная документация выдвигает требования к разработке и оформлению сопутствующей документации.

1. Основания для разработки

Основанием для разработки является задание на курсовую работу по дисциплине «С# - программирование», выданное на кафедре МПО ЭВМ ИИТ ЧГУ.

Дата утверждения: 14 марта 2024 года.

Наименование темы разработки: «Разработка программного обеспечения на языке C#»

2. Назначение разработки

Основная задача курсовой работы: освоить на практике материал, полученный в ходе изучения дисциплины «С# - программирование».

3. Требования к программе

3.1. Требования к функциональным характеристикам

Разработать приложение для авторазметки. Приложение должно выделять шарообразные камни угля на сгенерированном изображении, на основе его маски. Помимо этого, приложение должно поддерживать историю разметки, которая будет хранить дату и время разметки, у пользователя должна быть возможность восстановить файлы разметки с помощью истории. В ходе разработки учесть следующие требования:

* В качестве базы спроектировать WPF-приложение с использованием паттерна MVVM и/или использованием Фреймворка Prism.
* Придерживаться принципам SOLID2 и DRY3.
* Избегать наследования там, где его можно заменить ассоциацией. Особое внимание уделяйте классам «Моделям» и классам «Сервисам».
* Избегать написание кода в code-behind4.
* Выделить стили и темы оформления элементов пользовательского интерфейса в отдельные словари ресурсов.
* Реализовать файловый ввод/вывод данных.
* Предусмотреть обработку различных исключительных ситуаций.
* Работа всех функций должна быть проверена и результаты проверки оформлены протоколом тестирования.

3.2. Требования к надёжности

В местах, где могут возникнуть неожиданные ошибки предусмотреть обработку исключений.

3.3. Условия эксплуатации

Программу следует запускать на компьютере в закрытом помещении с обогревом и (или) охлаждением при следующих условиях окружающей среды:

• Уровень атмосферного давления: от 70 кПа до 106 кПа;

• Предел абсолютной влажности воздуха должен быть равен 25 гр/м;

• Температура окружающего воздуха от +16°C до +28°C;

• Запыленность воздуха не более 0,75 мг/м³;

• Знание основ работы в операционной системе Windows.

3.4. Требования к составу и параметрам технических средств

Для нормального функционирования программного средства минимальный состав и параметры технических средств должны соответствовать нижеследующему:

• Процессор с тактовой частотой не менее 2000 MHz, частота 2,3 Ghz;

• Оперативная память 5Gb и выше;

• Архитектура с разрядностью 32 бит или 64 бит;

• Наличие компьютерной мыши, клавиатуры, монитора (для персонального компьютера).

3.5. Требования к информационной и программной совместимости

Минимальные требования для информационной и программной совместимости:

• Операционная система (Windows 7 и выше);

• Наличие подключения к серверу базы данных или среды SQL Server Management Studio (SSMS).

3.6. Требования к маркировке и упаковке

Распространение через средства коммуникации (для некоммерческого использования).

3.7. Требования к транспортированию и хранению

Файлы, необходимые для корректной работы, необходимо записать на CD-диск.

3.8. Специальные требования

Отсутствуют.

4. Требования к программной документации

4.1. Содержание расчётно-пояснительной записки:

* Оглавление;
* введение;
* Описание предметной области;
* Организация работы в команде;
* Проектирование приложения;
* Разработка базы данных;
* заключение;
* список литературы;
* техническое задание;
* руководство пользователя;
* текст программы.

4.2. Требования к оформлению

Требования к оформлению, установленные ГОСТ, должны быть выполнены на протяжении всей работы без каких-либо изменений (в табл. П1.1).

Таблица П1.1

Требования к оформлению

|  |  |
| --- | --- |
| Документ | Печать на отдельных листах формата А4 (20х297 мм); оборотная сторона не заполняется; листы нумеруются. Печать возможна ч/б. |
| Страницы | Ориентация — книжная; отдельные страницы, при необходимости, альбомная. Поля: верхнее, нижнее — по 2 см, левое — 3 см, правое — 2 см. |
| Абзацы | Межстрочный интервал — 1,5, перед и после абзаца — 0. |
| Шрифты | Кегль — 14. В таблицах шрифт 12. Шрифт листинга — 8 (возможно в 2 колонки). |
| Рисунки | Подписывается под ним по центру: «Рис.Х. Название В» приложениях: «Рис.П.3. Название» |
| Таблицы | Подписывается: над таблицей, выравнивание по правому: «Таблица Х». В следующей строке по центру Название Надписи в «шапке» (имена столбцов, полей) — по центру. В теле таблицы (записи) текстовые значения — выравнены по левому краю, числа, даты — по правому. |

5. Стадии и этапы разработки

Стадии и этапы разработки представлены в таблице П1.2.

Таблица П1.2

Этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапа разработки | Сроки разработки | Результат выполнения | Отметка о выполнении |
| Определение темы для курсовой работы | 20.03.2024 | Утверждена тема разработки | Выполнено |
| Оформление техническое задания | 10.04.2024 | Выполненное техническое задание | Выполнено |
| Выбор инструментов организации работы в команде | 10.04.2024 | Выбраны инструменты для совместной работы | Выполнено |
| Проектирование приложения | апрель | Разработана концепция приложения, созданы макеты интерфейсов | Выполнено |
| Реализация | Апрель-май | Готовая программа | Выполнено |
| Написание РПЗ | 6.06.2024 | Создано РПЗ по курсовой работе | Выполнено |

6. Порядок контроля и приёмки

Порядок контроля и приёма представлены в таблице П1.3.

Таблица П1.3

Порядок контроля и приемки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование контрольного этапа выполнения курсовой работы | Сроки контроля | Результат выполнения | Отметка о приемке результата контрольного этапа |
| 1 | 2 | 3 | 4 |
| Утверждение технического задания | 19.05.2024 | Документ «Техническое задание» проверен |  |

Продолжение таблицы П.1.3

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| Демонстрация работы первой версии программы | Апрель-май | Работа программы проверена |  |
| Демонстрация окончательной версии программы | 30.05.2024 | Работа программы проверена |  |
| Защита курсовой работы | 8.06.2024 | Курсовая работа защищена |  |

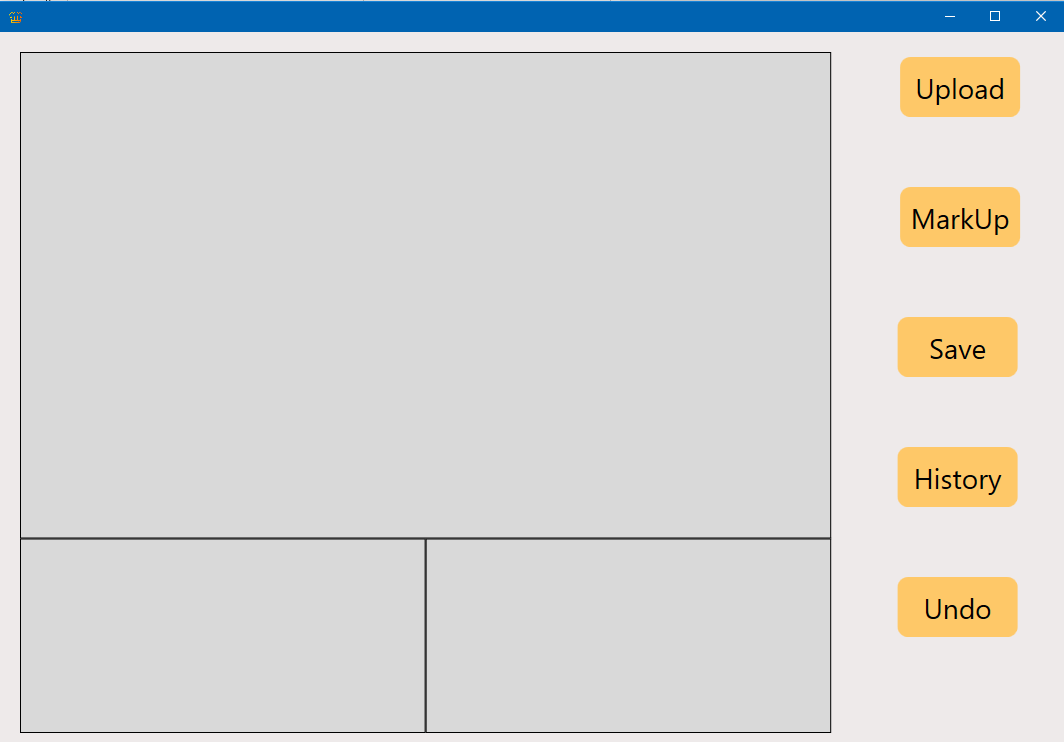
# Приложение 2. Руководство пользователя

Для начала работы с приложением необходимо запустить файл приложения(рис. П2.1)



Рис.П2.1. Запускающий файл

Откроется стартовое окно (рис. П2.2)



3

2

1

Рис.П2.2. Окно приложения при запуске

Пользователь может загрузить изображение и его маску после взаимодействия с кнопкой «Upload». После нажатия на кнопку откроется диалоговое окно для выбора файла(рис.П2.3), выбрав нужное изображение, нужно будет после выбрать и маску для данного изображения. Исходное изображение сразу отобразится в поле 1, а его маска в поле 2 (рис.П2.4).

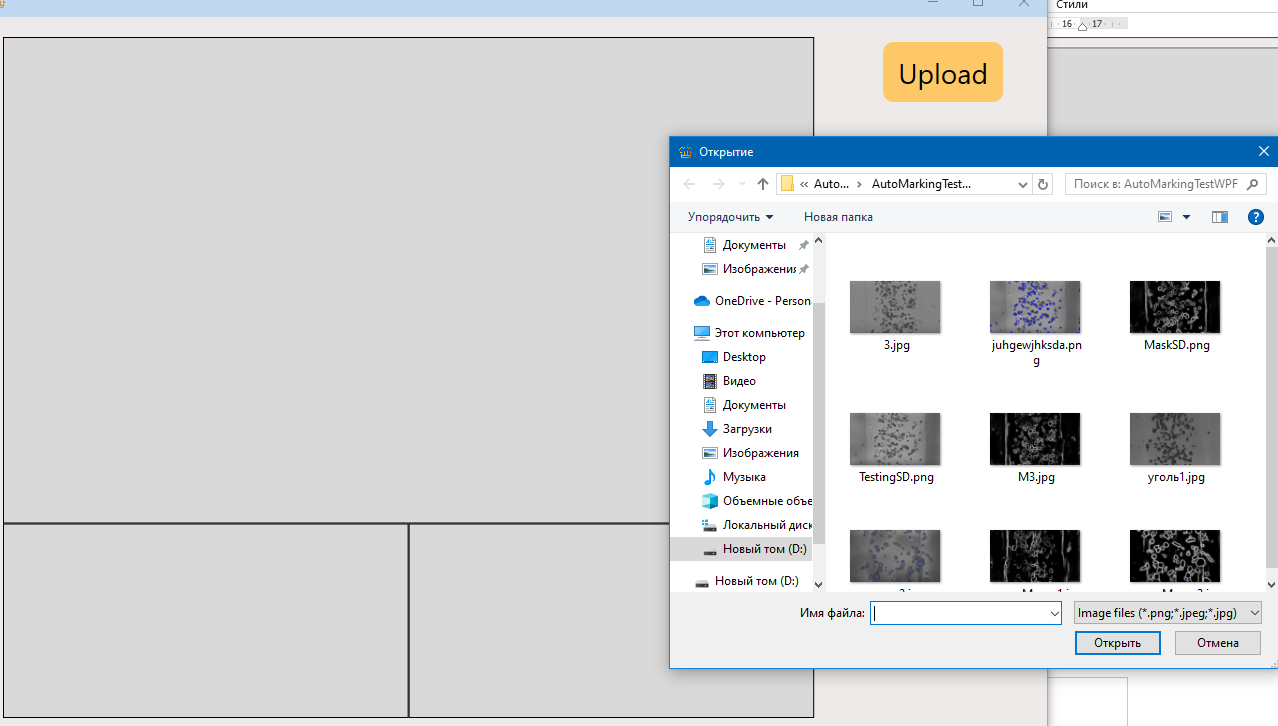


Рис.П2.3. Выбор исходного изображения

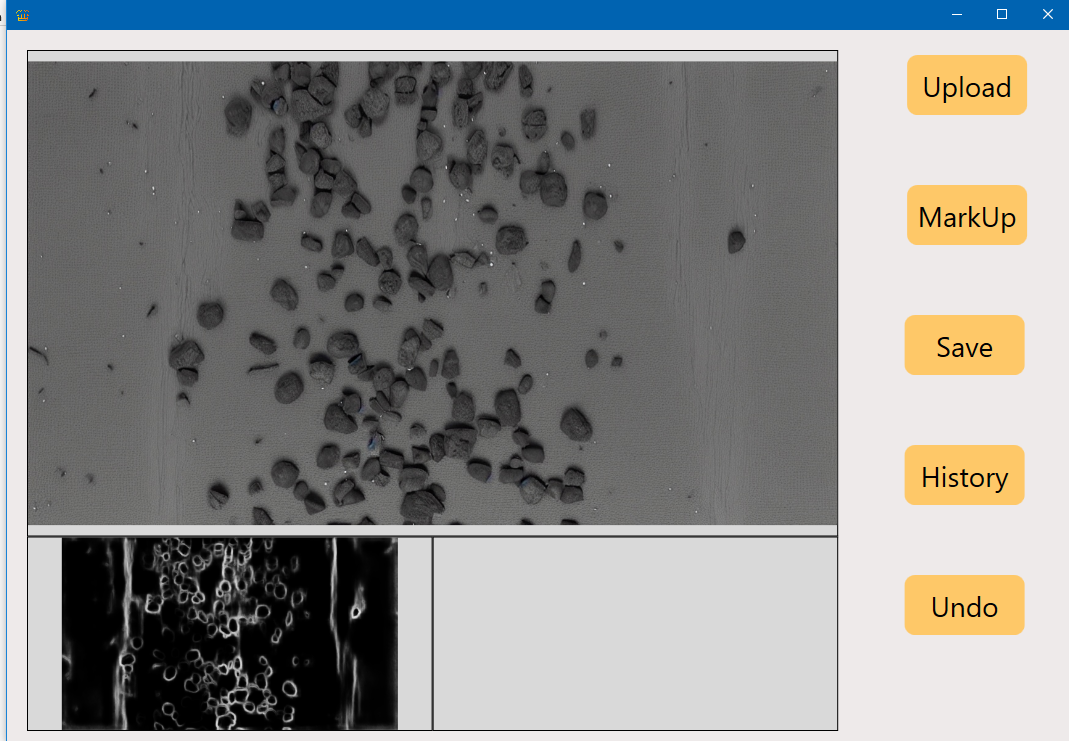


Рис.П2.4. Загруженное изображение и его маска

После выбора нужных исходных файлов пользователь может нажать кнопку «MurkUp» для создания разметки. После завершения операции появится размеченный файл в 3 поле (рис.П2.5).

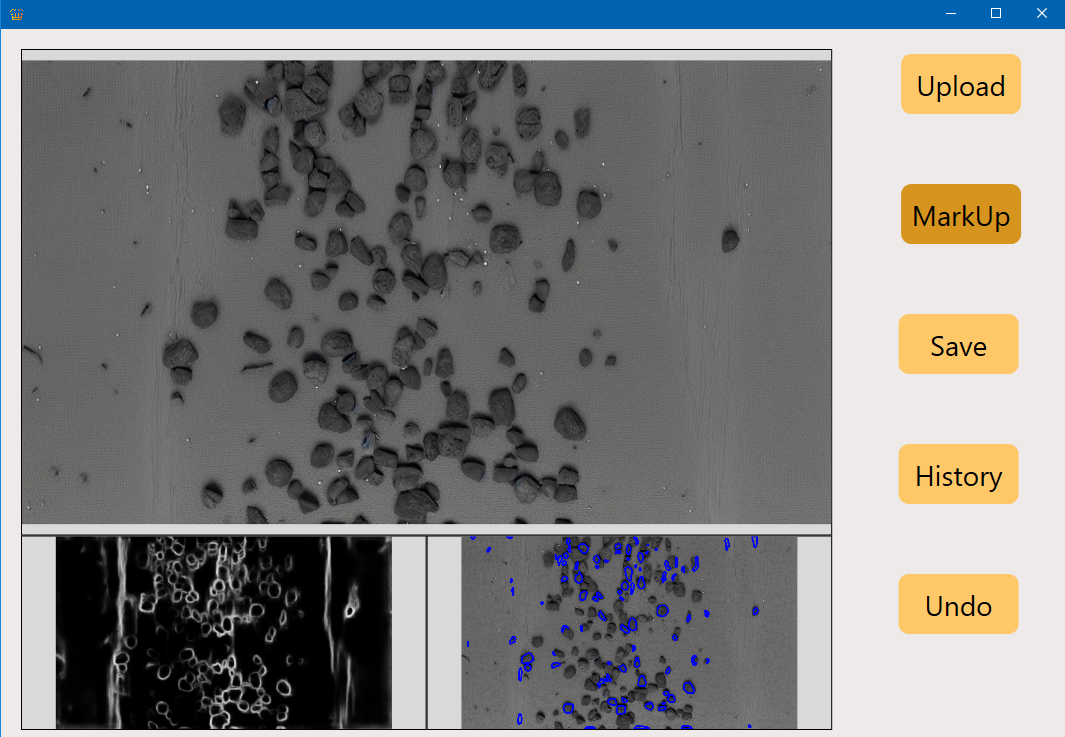


Рис.П2.5. Разметка завершена

Если пользователю необходимо вывести какое-то изображение в поле 1, достаточно нажать на это изображение, и оно поменяется места с изображением из поля 1 (рис.П2.6).

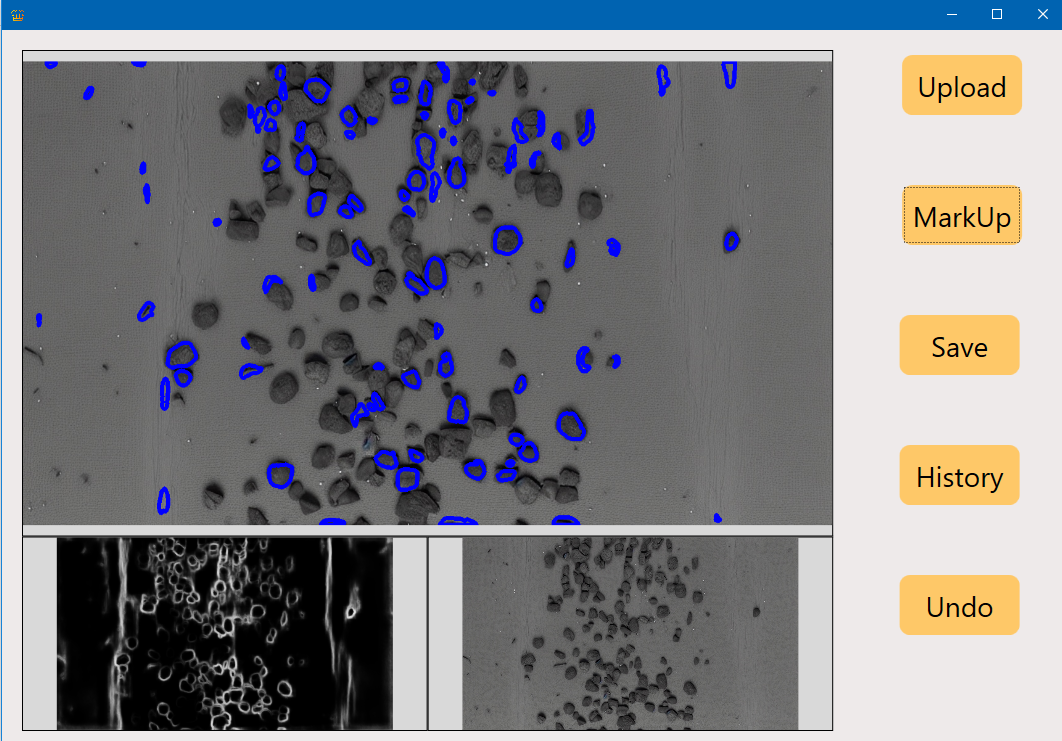


Рис.П2.6. Исходное изображение и изображение с разметкой поменялись местами

Теперь результаты можно сохранить, для этого необходимо воспользоваться соответствующей кнопкой «Save», выбрать папку, в которую необходимо сохранить файл, и как-то назвать. Будет сохранено 2 файла: изображение с разметкой и файл разметки (рис.П2.7).

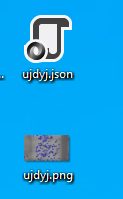


Рис.П2.7. Сохраненные файлы

Также приложение подцеживает историю (рис.П2.8) с помощью которой можно восстановить прежде размеченные файлы.

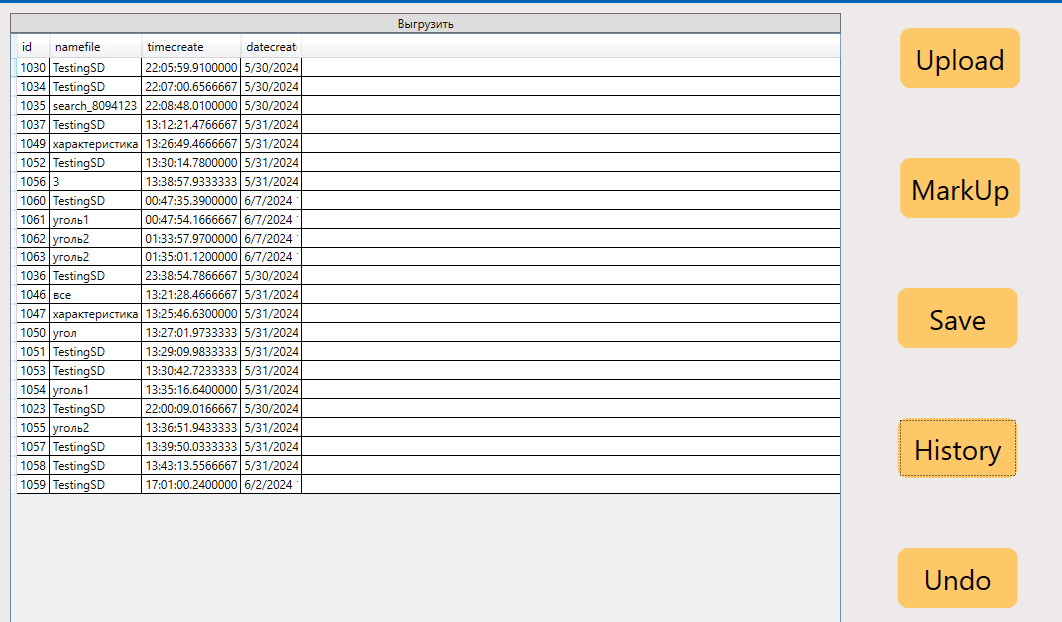


Рис.П2.8. Окно истории

Чтобы восстановить файлы нужно выбрать соответствующую строку в таблице и нажать на кнопку «Выгрузить», она расположена сверху (рис.П2.9). Далее нужно вернуться на прежнюю страницу, нажав на кнопку «Undo», на ней будут отображены восстановленные изображения (рис.П2.10).

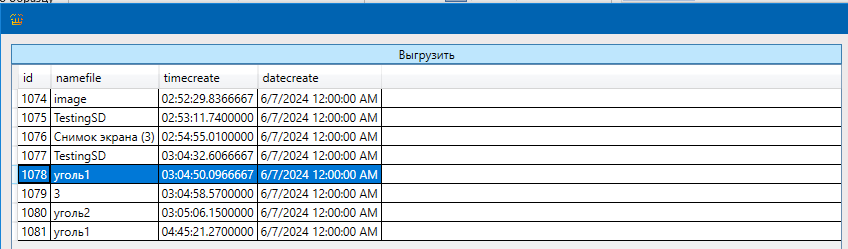


Рис.П2.9. Выбор

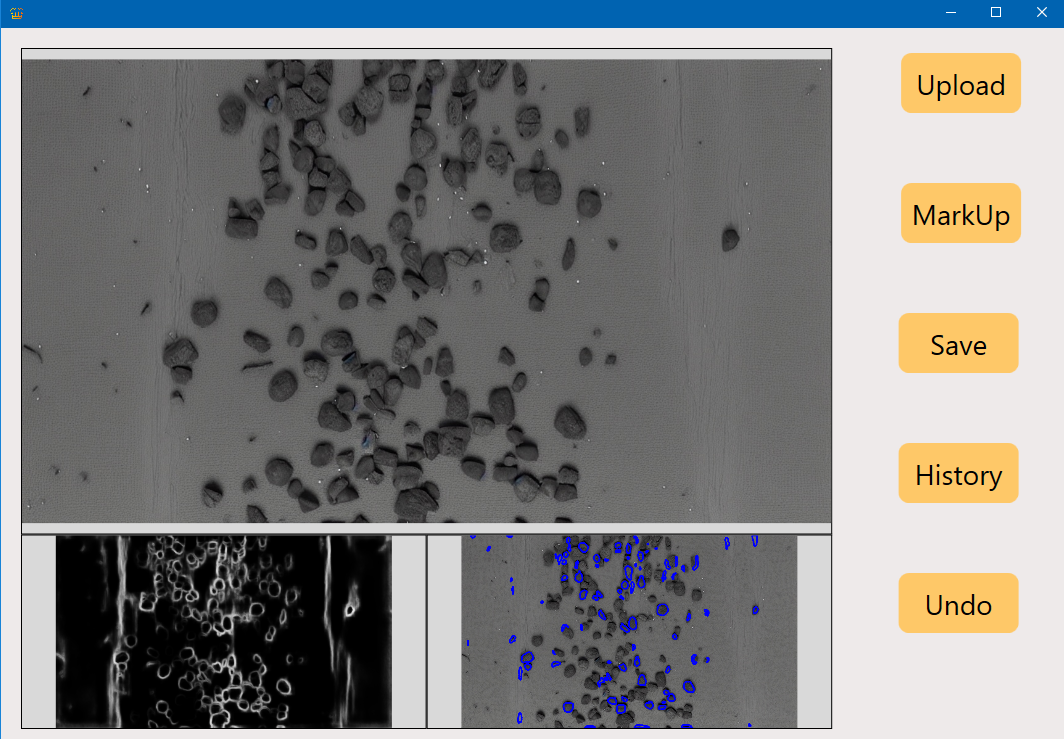


Рис.П2.10. Восстановленные изображения

К руководству создано видео с демонстрацией работы программы (рис.П2.11).



Рис.П2.11. Файл видео с демонстрацией

# Приложение 3. Текст программы

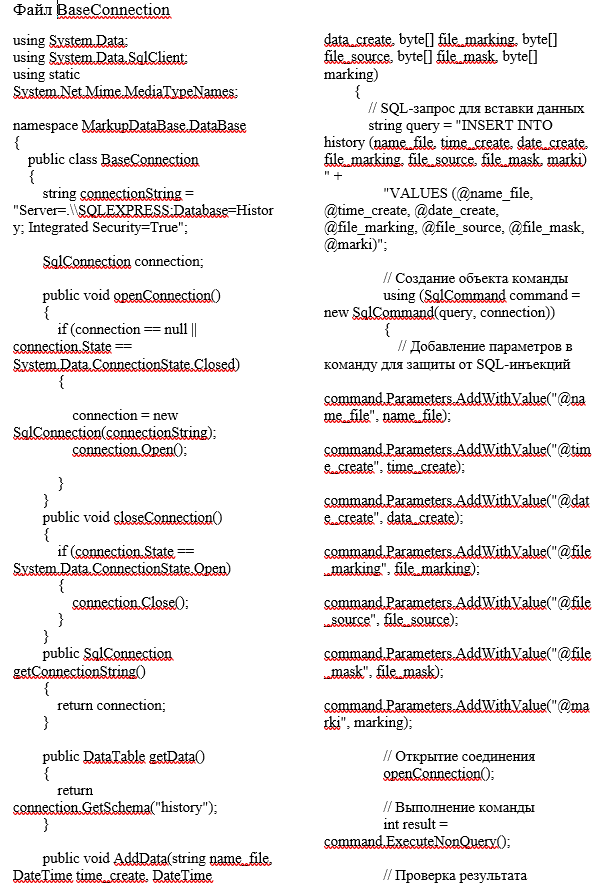


Рис.П3.1 Код программы

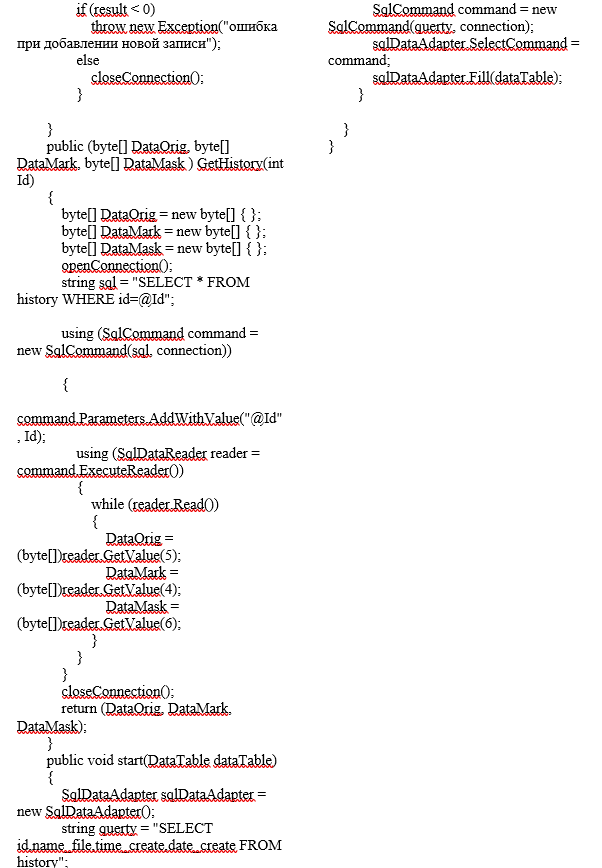


Рис.П3.2. Код программы

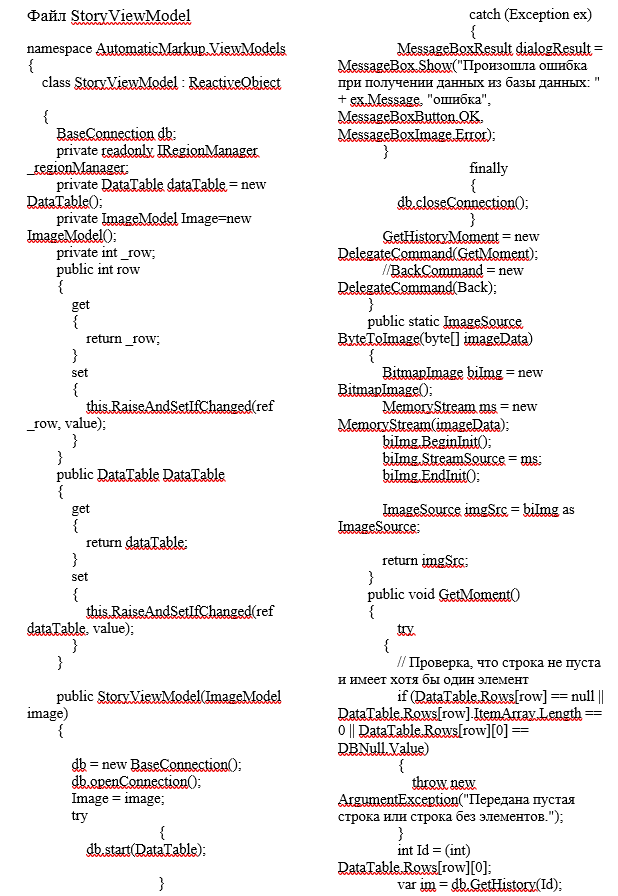
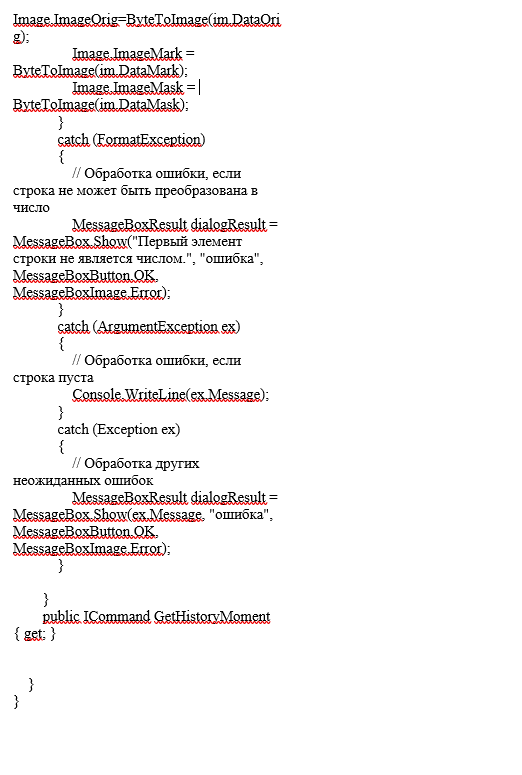


Рис.П3.3. Код программы



**Функция разметки из AutomaticMarkup.ViewModels.** **MainViewModel**

private async void AutoMarking()

{

marking = null;

Bitmap orig = ConvertToBitmap((BitmapSource)SelectedImage.ImageOrig);

Bitmap mask = ConvertToBitmap((BitmapSource)SelectedImage.ImageMask);

await Task.Run(() => { marking = GetMark(orig, mask); });

SelectedImage.ImageOrig = ConvertToImageSource(marking.GetBitmap());

SelectedImage.ImageMark = ConvertToImageSource(marking.GetMarkBitmap());

SelectedImage.ImageMask = ConvertToImageSource(marking.GetMaskBitmap());

try

{

var BaseConnection = new BaseConnection();

BaseConnection.openConnection();

DateTime time = DateTime.Now;

var img1 = ImageToByte(marking.GetBitmap());

var img2 = ImageToByte(marking.GetMarkBitmap());

var img3 = ImageToByte(marking.GetMaskBitmap());

var mark = ReadFully(marking.SaveJson("marking.json"));

BaseConnection.AddData(file\_name, time, time.Date, img2, img1,img3, mark);

}

catch (Exception ex)

{

MessageBoxResult dialogResult = MessageBox.Show(ex.Message, "ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

Рис.П3.4. Код программы