



**Министерство образования и науки России
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Казанский национальный исследовательский
технологический университет»
(ФГБОУ ВО «КНИТУ»)**

Кафедра Интеллектуальных систем и управления информационными ресур-
сами

Направление 01.03.02

Группа 4381-22

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Уровень образования бакалавр

(бакалавр, специалист, магистр)

Вид ВКР _____

(проектный, исследовательский, комбинированный)

Тема «Разработка веб-приложения для чтения комиксов с функцией распо-
знавания текста»

Зав. кафедрой _____ (_____)

Нормоконтролер _____ (_____)

Руководитель _____ (_____)

Студент _____ (_____)

2022 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	3
ЗАДАНИЕ	4
ЛИСТ НОРМОКОНТРОЛЕРА	5
1. РАЗРАБОТКА ПРОЕКТА ВЕБ-ПРИЛОЖЕНИЯ.....	6
1.1. Исследование предметной области.....	6
1.2. Функциональные требования	6
1.3. Нефункциональные требования	7
1.4. Существующие аналоги системы.....	7
1.5. Mind-карта системы.....	8
1.6. Макеты системы.....	10
1.7. Логическое проектирование базы данных	16
2. РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ	18
2.1. Инструменты разработки	18
2.2. Создание моделей	20
2.3. Создание контроллеров	25
2.4. Создание представлений	33
2.5. Распознавание текста с изображений	39
ЗАКЛЮЧЕНИЕ	47
СПИСОК ЛИТЕРАТУРЫ.....	48

ВВЕДЕНИЕ

В последнее время комиксы становятся все популярнее в мире, а люди все чаще используют веб-приложения для их чтения. Также очень популярны комиксы, выходящие на иностранных языках, однако перевод произведений на родной язык выходит через некоторое время. Чтобы людям не приходилось ждать перевод, было создано веб-приложение, позволяющее читать комикс и распознавать текст с изображения, что позволит в дальнейшем перевести его на родной язык с помощью машинного перевода.

Цель данной работы – разработать веб-приложение для чтения комиксов и реализовать функцию распознавания текста с изображений.

Для достижения этой цели необходимо выполнить следующий перечень задач:

1. Анализ предметной области;
2. Определение функциональных и нефункциональных требований к системе;
3. Построение макетов системы;
4. Создание базы данных;
5. Разработка клиентской и серверной частей веб-приложения.

Кафедра ИСУИР
Направление 01.03.02
Группа 4381-22

УТВЕРЖДАЮ
Зав. кафедрой _____
_____ 2022 г.

ЗАДАНИЕ

на выпускную квалификационную работу студента _____

Тема: Разработка веб-приложения для чтения комиксов с функцией распознавания текста

Срок представления работы к защите "___" _____ 20__ г.

Цель, задачи и исходные данные работы: _____

Задание по разделам работы: _____

Содержание графической части (иллюстративного материала): _____

Консультанты: _____

Дата выдачи задания "___" _____ 20__ г.

Руководитель _____ (_____)

Задание принял к исполнению _____ (_____)

ЛИСТ НОРМОКОНТРОЛЕРА

1. Лист является обязательным приложением к пояснительной записке дипломного (курсового) проекта.

2. Нормоконтролер имеет право возвращать документацию без рассмотрения в случаях:

- нарушения установленной комплектности,
- отсутствия обязательных подписей,
- нечеткого выполнения текстового и графического материала.

3. Устранение ошибок, указанных нормоконтролером, обязательно.

П е р е ч е н ь

Замечаний и предложений нормоконтролера по дипломному
проекту студента

4381-22, Р.А. Галиевой

(группа, инициалы, фамилия)

Лист (страница)	Условное обозначение (код ошибок)	Содержание замечаний и предложений со ссылкой на нормативный документ, стан- дарт или типовую документацию

Дата « ____ » _____ 20 ____ г.

Нормоконтролер _____
(подпись)

(инициалы, фамилия)

1. РАЗРАБОТКА ПРОЕКТА ВЕБ-ПРИЛОЖЕНИЯ

1.1. Исследование предметной области

Предметной областью веб-приложения для чтения комиксов является хранилище графических романов.

Для людей, увлекающихся комиксами, будет полезно приложение, отображающее их список, позволяющее их читать и просматривать о них информацию. К информации о комиксах относятся их название, жанры, авторы, главы, обложка, описание, год выхода, язык текста и сами изображения страниц. Также пользователь сможет добавлять комикс в один из списков для чтения: «читаю», «прочитано» или «хочу прочитать», добавлять в категорию любимых, оставлять отзыв и ставить оценку от одного до пяти.

Для удобства пользователей необходимо предусмотреть поиск по названию, а также фильтрацию комиксов по жанрам, языкам и сортировку по дате выхода, названию или средней оценке читателей.

1.2. Функциональные требования

Для данной информационной системы были выделены следующие функциональные требования:

- Аутентификация и авторизация пользователя;
- Чтение комиксов;
- Возможность оставлять отзывы;
- Добавление в список для чтения;
- Распознавание текста с изображений.

При чтении комикса пользователь сможет менять размер изображения с помощью ползунка, выбрать режим чтения: горизонтальный либо вертикальный. При горизонтальном режиме чтения будет отображаться одна страница

комикса, а пользователь сможет переключаться между страницами с помощью кнопок со стрелками либо с помощью клавиатуры.

Немаловажной является функция распознавания текста. С ее помощью пользователь сможет выделить конкретный участок изображения, программа распознает текст, находящийся на выделенном участке и выведет его на экран. Также отдельно будет выводиться лишь выделенный фрагмент изображения.

1.3. Нефункциональные требования

В данном проекте имеются следующие нефункциональные требования:

- дружественный интерфейс;
- привлекательный дизайн, соответствующий ожиданиям целевой аудитории;
- легкость в навигации;
- легкость в использовании;
- ограничение между пользовательским интерфейсом и серверной части.

1.4. Существующие аналоги системы

Существует большое количество аналогов приложений для чтения комиксов, в которых реализовано множество полезных функций. К примеру, в таком приложении, как MangaLib, есть возможность оставлять комментарии под каждой страницей, как продемонстрировано на рисунке 1.1.

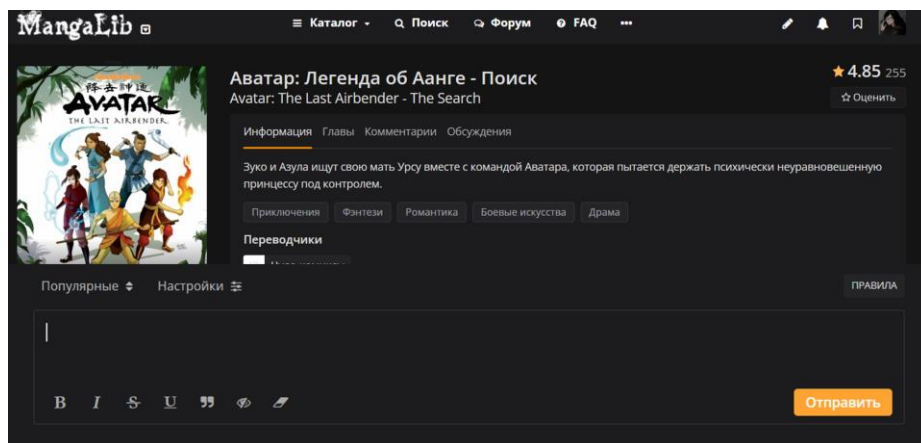


Рисунок 1.1 – Обзор приложения MangaLib

А в приложении WebToon, как показано на рисунке 1.2, пользователь может сам загружать комиксы на сервер.

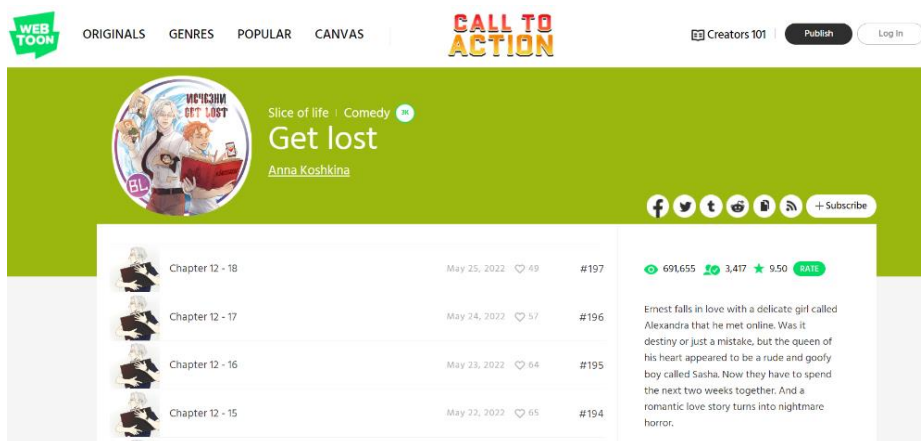


Рисунок 1.2 – Обзор приложения WebToon

Однако ни в одном из них нет функции распознавания текста с изображения, которая даст возможность переводить текст с иностранного языка на родной. Поэтому выбранная тема проекта является актуальной на сегодняшний день.

1.5. Mind-карта системы

На рисунке 1.3 представлена mind-карта разрабатываемой системы. Сайт состоит из шести страниц, каждая из которых имеет навигационный блок и основной контент. На главной странице отображается список комиксов, который можно отфильтровать по жанрам и языкам, а также отсортировать по

дате выхода, названию и средней оценке. На странице «Вход» или «Регистрация» располагаются формы – для входа в систему и для регистрации новых пользователей соответственно. На странице «Профиль» пользователь может просмотреть личные данные и изменить их. Также там отображаются комиксы, добавленные в один список для чтения – «Читаю», «Прочитано», «Хочу прочитать», «Любимое», а также отображаются все отзывы данного пользователя. На странице «Комикс» имеется информация о комиксе, главы и отзывы. На странице «Чтение» расположены навигация по комиксу, настройки читалки и сами страницы комикса. Также имеется страница «Распознавание» для распознавания текста.

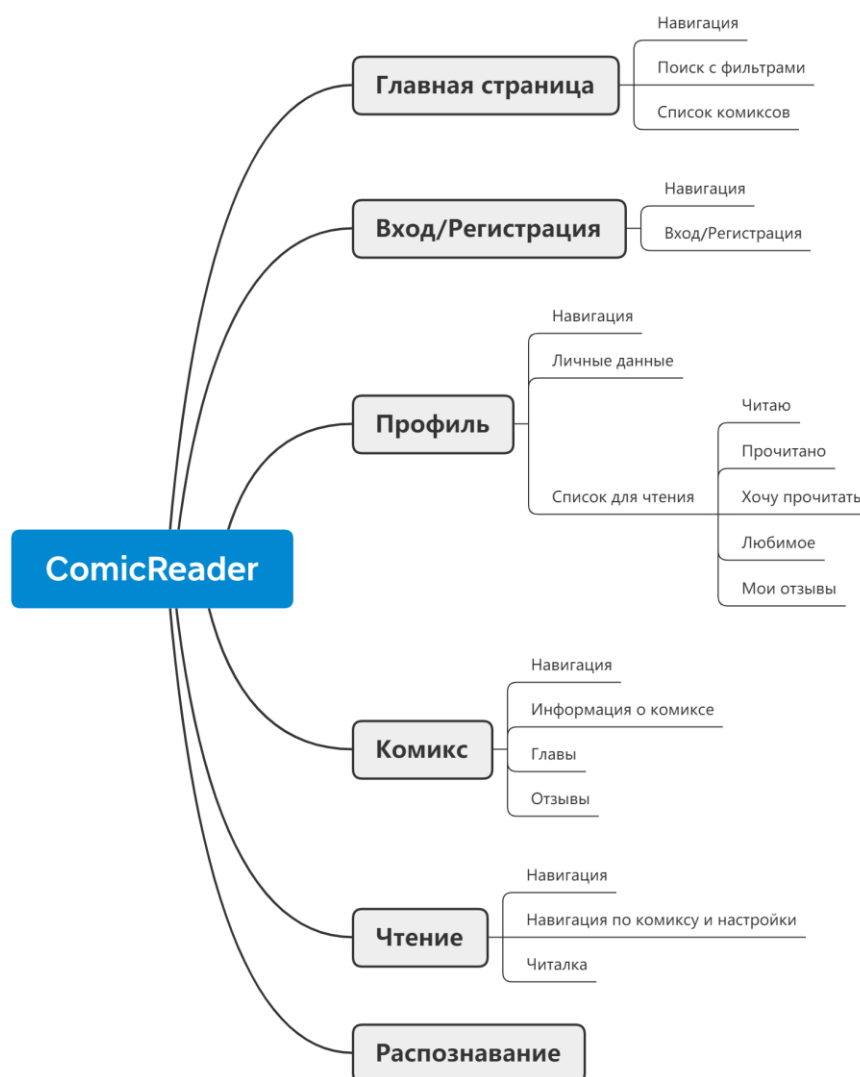


Рисунок 1.3 – Mind-карта системы

1.6. Макеты системы

С помощью приложения “Photoshop” были построены макеты системы.

На каждой странице сайта имеется навигационная панель и основной контент. В навигационную панель входят логотип, название сайта, строка поиска и кнопка для перехода на страницу пользователя. При нажатии на логотип открывается главная страница.

На главной странице, как показано на рисунке 1.4, отображается список комиксов в порядке даты выхода по убыванию. Также имеется строка поиска с фильтрами по жанрам, языку и сортировка по дате выхода, названию и оценке пользователей.

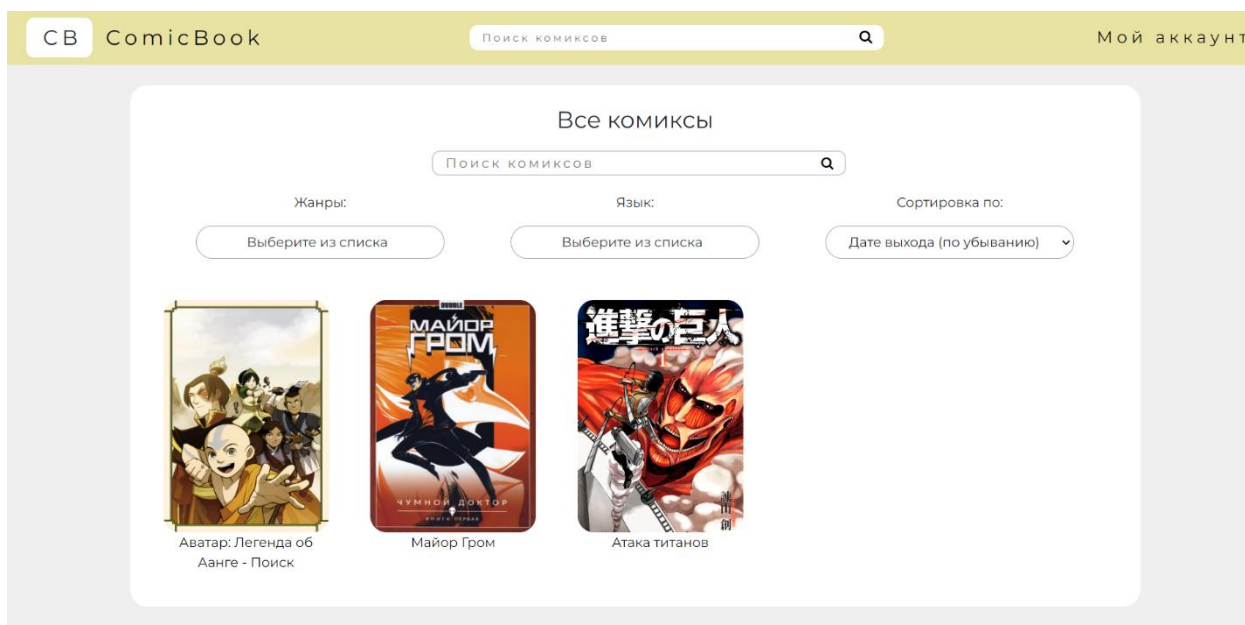


Рисунок 1.4 – Макет главной страницы

На странице комикса в соответствии с рисунком 1.5 располагается такая информация о комиксе, как обложка, название, год выхода, список авторов и жанров, описание. Справа располагается информация о средней оценке пользователей и общее количество оценок. Там же находятся кнопки «Добавить в любимые» и «Оставить отзыв». При нажатии на последнюю, как показано на

рисунке 1.6, открывается модальное окно с формой для отправки отзыва. Следом идет форма для добавления в один из списков: «Хочу прочитать», «Читаю» и «Прочитано».

Ниже располагается список глав со ссылкой на конкретную главу, а нажатии на кнопку «Отзывы» отображаются все отзывы, оставленные пользователями на данный комикс.

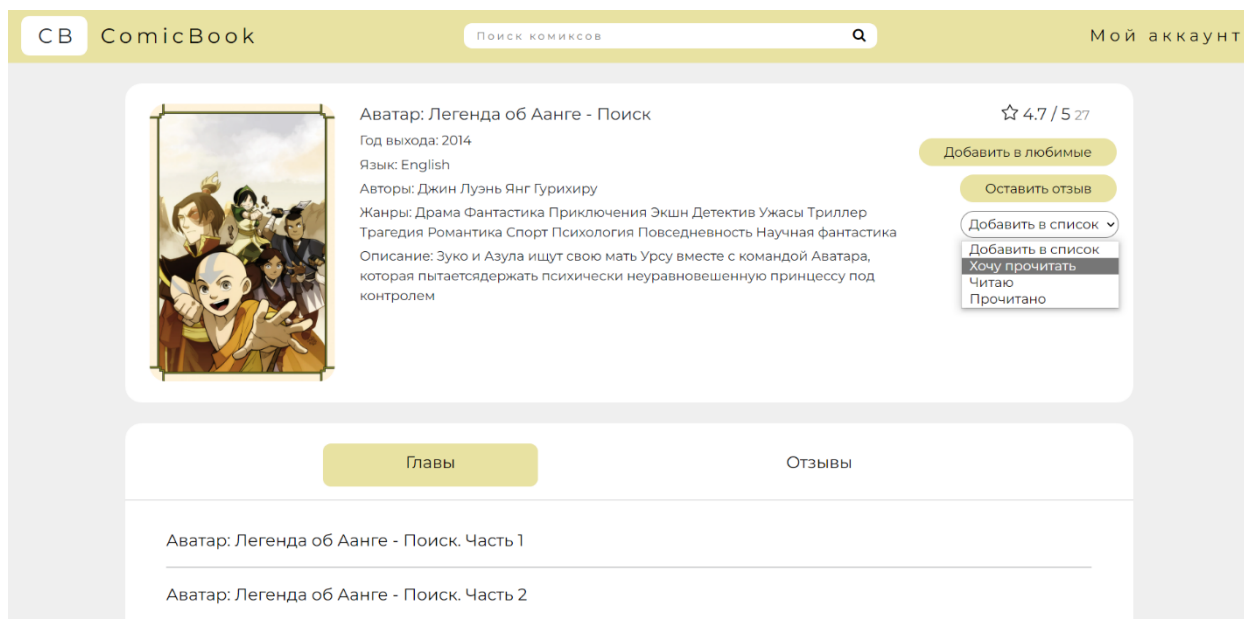


Рисунок 1.5 – Макет страницы комикса



Рисунок 1.6 – Модальное окно с формой для отправки отзыва

Страница чтения комикса открывается при нажатии на ссылку с определенной главой. Как показано на рисунке 1.4, она имеет дополнительную навигационную панель для перехода на другую главу или другую страницу либо для возвращения на страницу комикса. Также имеется кнопка настроек, при нажатии на которые открывается окно с настройками размера изображения и режима чтения.

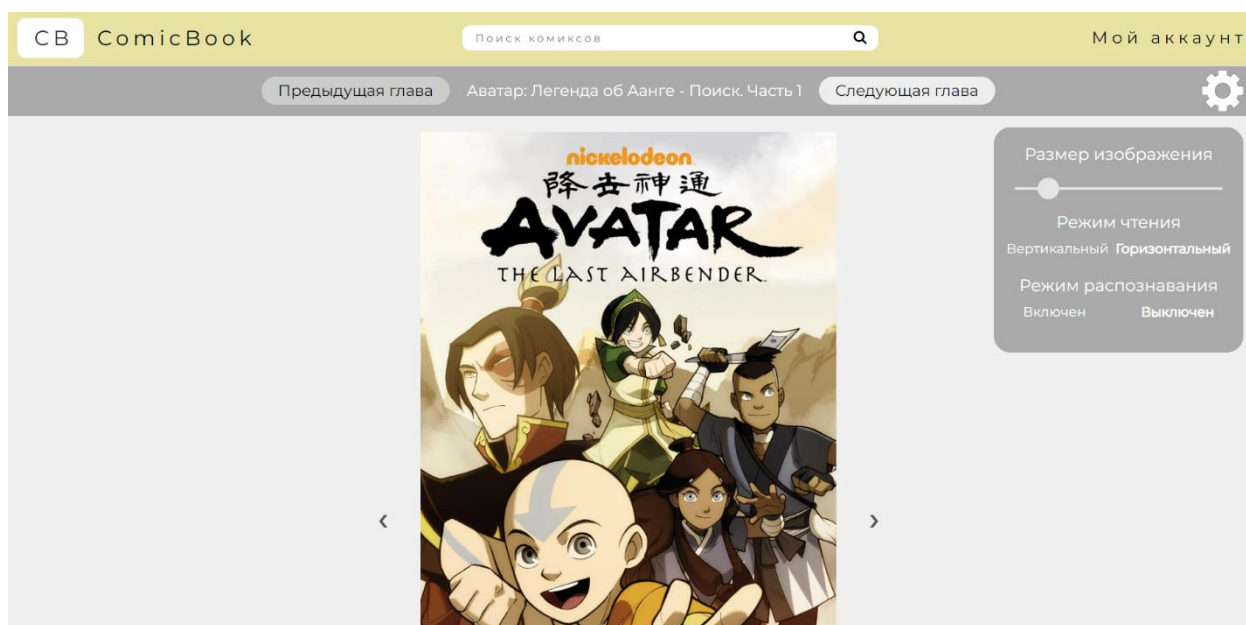


Рисунок 1.7 – Макет страницы чтения комикса

Под каждой страницей комикса имеется кнопка «Распознать», при нажатии на которую открывается страница с распознаванием. В соответствии с рисунком 1.8 пользователь имеет возможность выделить конкретный участок изображения, тогда в окне появится распознанный текст, а также выведется окошко с выделенным фрагментом.



Рисунок 1.8 – Макет страницы распознавания

На странице входа, согласно рисунку 1.9, запрашивается e-mail и пароль пользователя. При нажатии на кнопку «Войти» происходит переход на страницу профиля пользователя. А при нажатии на кнопку «Зарегистрироваться» открывается страница регистрации.

Q

Вход

Email

theamydave@gmail.com

Пароль

.....

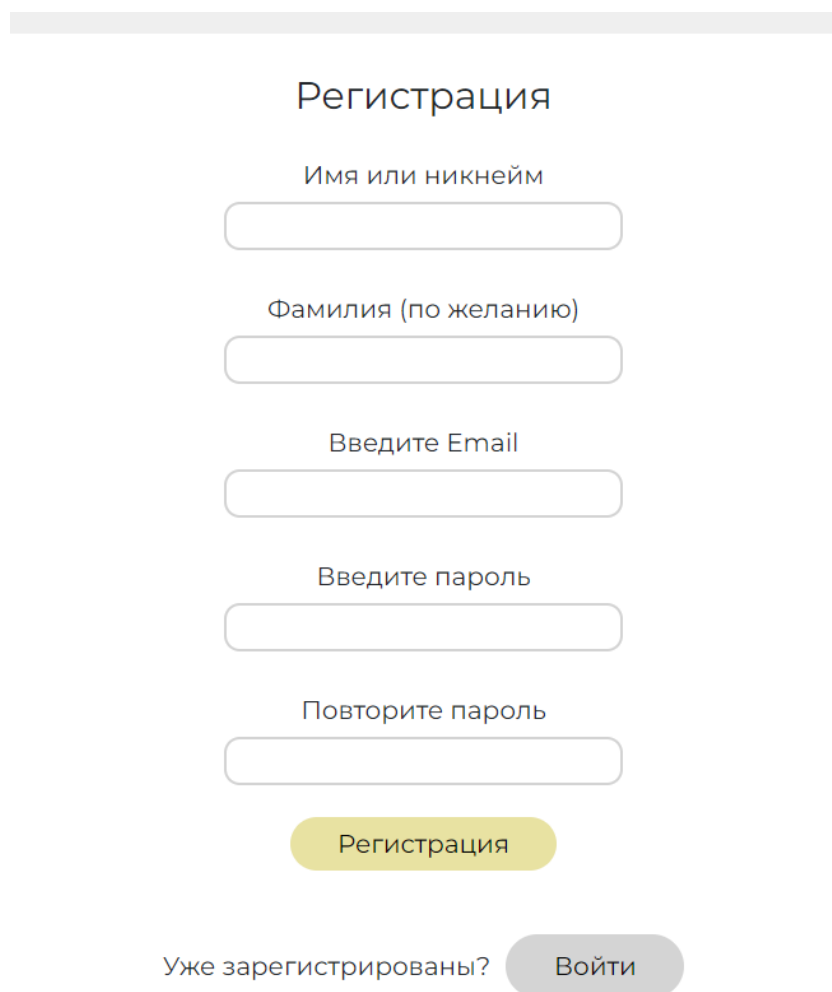
Войти

Ещё не зарегистрированы?

Зарегистрироваться

Рисунок 1.9 – Макет страницы «Вход»

На странице регистрации, как показано на рисунке 1.10, запрашивается имя, фамилия, e-mail, пароль и подтверждение пароля. При отсутствии заполнения обязательных полей будет выводиться ошибка с просьбой заполнить данное поле. При нажатии на кнопку «Зарегистрироваться» происходит регистрация пользователя в системе и переход на страницу его профиля. А при нажатии на кнопку «Войти» откроется страница входа в систему.



Регистрация

Имя или никнейм

Фамилия (по желанию)

Введите Email

Введите пароль

Повторите пароль

Регистрация

Уже зарегистрированы? Войти

Рисунок 1.10 – Макет страницы «Регистрация»

На странице профиля в соответствии с рисунком 1.11 находится блок с личными данными пользователя, которые можно изменить при нажатии на кнопку «Изменить информацию». Как показано на рисунке 1.12, откроется модальное окно для изменения информации. В нем можно добавить фото профиля, изменить имя, фамилию, e-mail и обновить пароль. Для последнего по-

требуется ввести предыдущий пароль для подтверждения личности пользователя. Также на странице отображаются комиксы, добавленные в один из списков для чтения и все оставленные им отзывы.

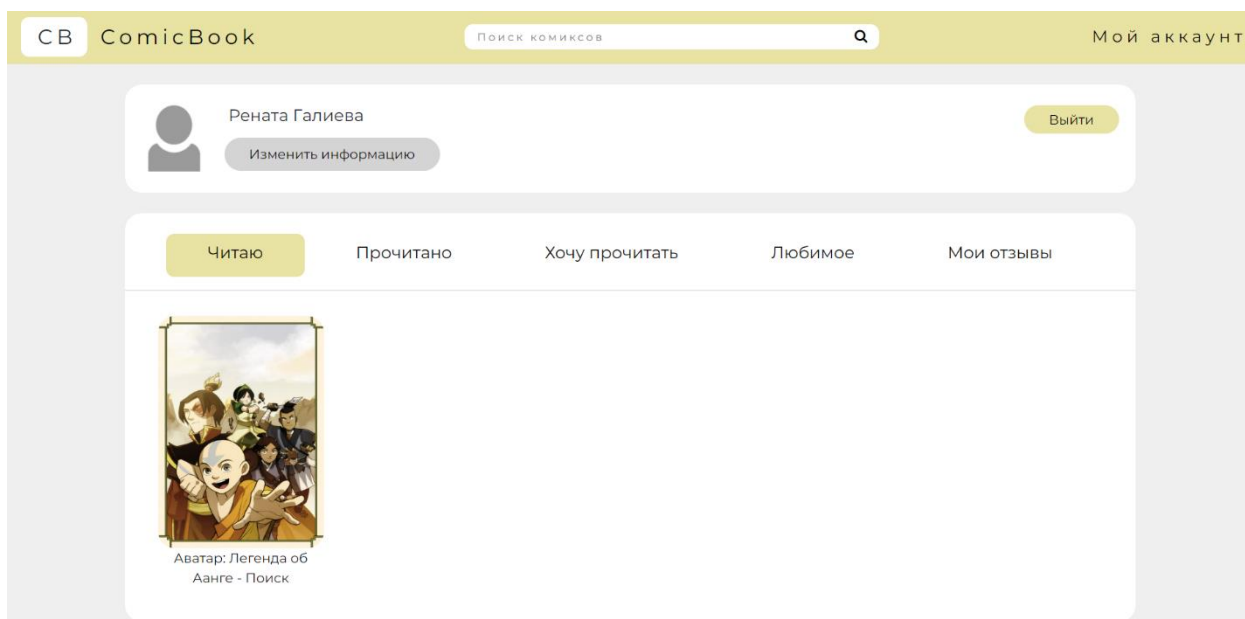


Рисунок 1.11 – Макет страницы «Профиль»

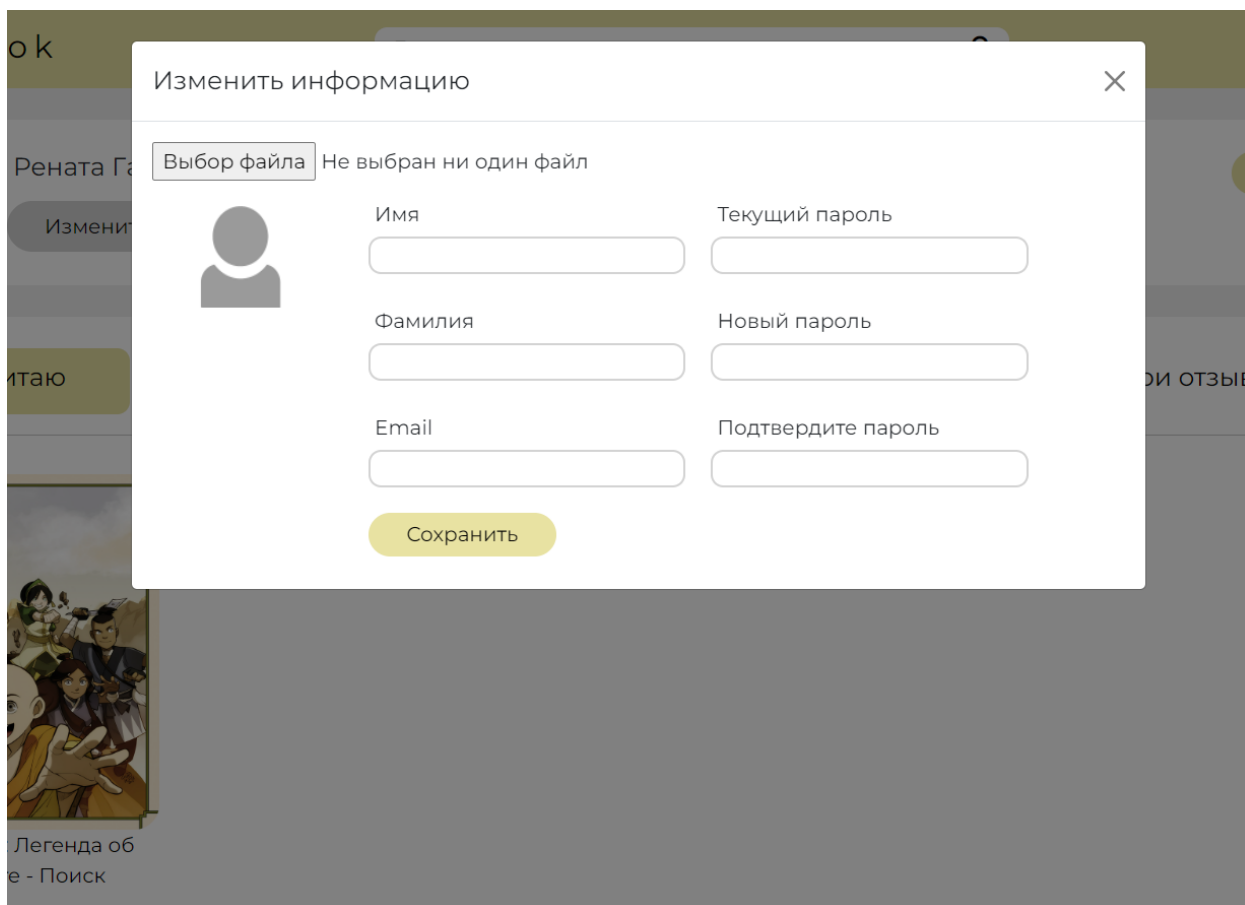


Рисунок 1.12 – Макет модального окна для изменения информации

1.7. Логическое проектирование базы данных

Как показано на рисунке 1.13, была спроектирована логическая модель базы данных. Всего имеется семь сущностей:

- Комиксы
- Жанры
- Авторы
- Главы
- Страницы
- Пользователи
- Комикс_Пользователь

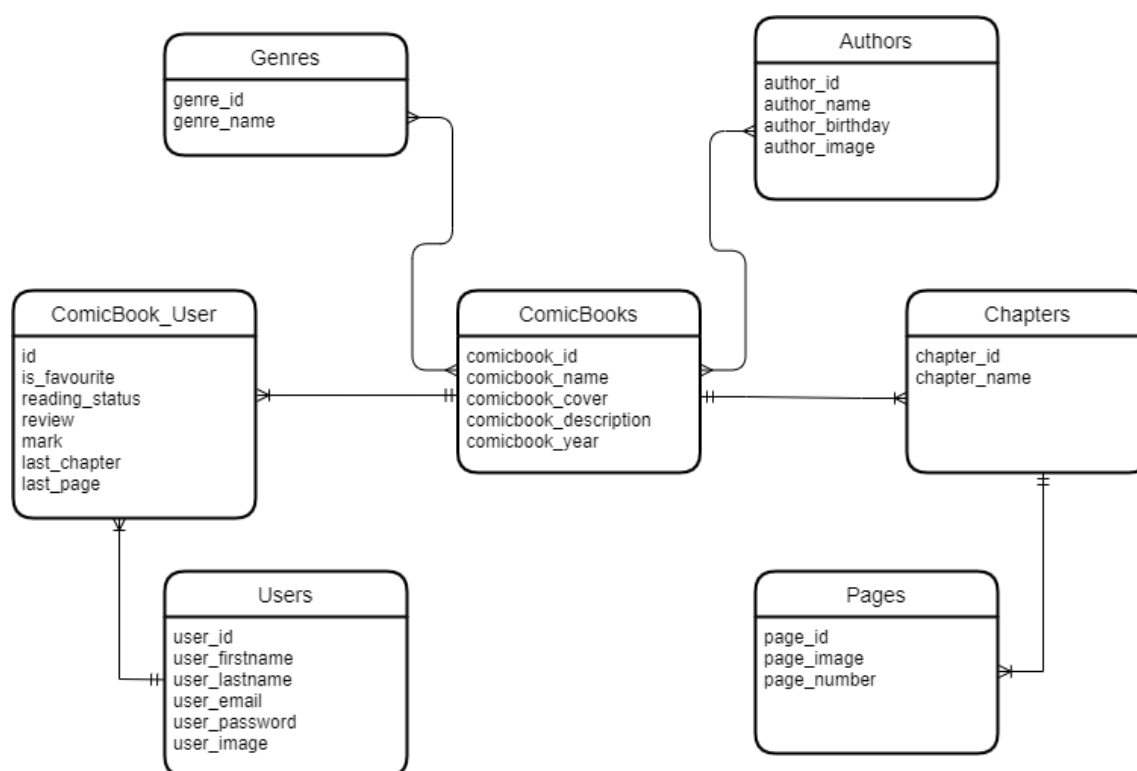


Рисунок 1.13 – ER-модель базы данных

Сущность «Комиксы» содержит в себе информацию о комиксах и имеет такие атрибуты, как код комикса, название, обложка, описание, год выхода и язык текста.

Сущность «Жанры» содержит в себе информацию о жанрах и имеет такие атрибуты, как код жанра и его название. Она связана с сущностью «Комиксы» отношением «Многие ко многим».

Сущность «Авторы» содержит в себе информацию об авторах и имеет такие атрибуты, как код автора, его имя, дата рождения и фото. Она связана с сущностью «Комиксы» отношением «Многие ко многим».

Сущность «Главы» содержит в себе информацию о главах и имеет такие атрибуты, как код главы и ее название. Она связана с сущностью «Комиксы» отношением «Один ко многим».

Сущность «Страницы» содержит в себе информацию о страницах и имеет такие атрибуты, как код страницы, ее номер и изображение. Она связана с сущностью «Главы» отношением «Один ко многим».

Сущность «Пользователи» содержит в себе информацию о пользователях и имеет такие атрибуты, как код пользователя, его имя, фамилия, e-mail, пароль и фото профиля.

Сущность «Комикс_Пользователь» является промежуточной, содержит в себе информацию о конкретном пользователе и комиксе и имеет такие атрибуты, как код комикса, код пользователя, является ли комикс любимым, статус чтения, отзыв, оценка, последняя страница и последняя глава. Она связана с сущностями «Комиксы» и «Пользователи» отношением «Один ко многим».

2. РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ

2.1. Инструменты разработки

Веб-приложение было разработано на платформе ASP.NET Core MVC.

Паттерн MVC подразумевает деление системы на три компонента:

- Модель (model): описывает применяемые в системе данные и логику, которая сопряжена с ними напрямую. Обычно их объекты хранятся в базе данных. Модели представлены двумя главными видами: модели представлений и модели домена. Первые нужны для того, чтобы представления могли передавать и отображать данные, а вторые для того, чтобы описывать логику обработки данных.
- Представление (view): соответствует визуалу или пользовательскому интерфейсу, зачастую это html-страница, посредством которой пользователь пользуется системой. Кроме того, представление имеет логику, связанную с отражением данных. Однако в представление нельзя включать обработку запроса или изменение данных.
- Контроллер (controller): предполагает главный компонент MVC, который гарантирует связь между пользователем и сервером, то есть представлением и моделью. Он включает логику обработки запроса пользователя. Контроллер принимает на вход данные пользователя и занимается их обработкой. И в зависимости от ее результатов посылает пользователю конкретный ответ.

Отношения между компонентами паттерна можно описать схемой на рисунке 2.1:

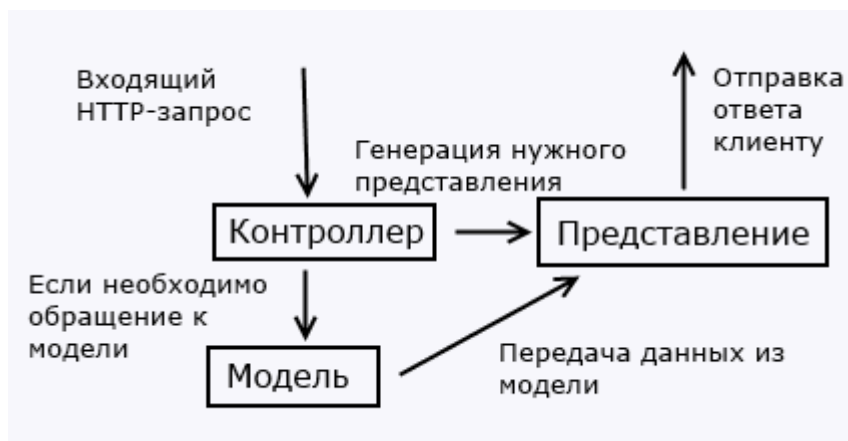


Рисунок 2.1 – Схема паттерна MVC

В качестве системы управления базами данных был использован MS SQL Server, а в качестве ORM (объектно-реляционного отображения) – Entity Framework Core.

При разработке клиентской части веб-приложения использовались следующие инструменты:

- HTML (HyperText Markup Language) – стандартизированный язык разметки документов в интернете. Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства. HTML состоит из ряда элементов, которые используются, чтобы вкладывать или оборачивать различные части контента, чтобы заставить контент отображаться или действовать определённым образом.
- CSS (Cascading Style Sheets) – формальный язык описания внешнего вида документа (веб-страницы), написанного с использованием языка разметки. В отличие от HTML, который служит для определения структуры и семантики содержимого, CSS отвечает за его внешний вид и отображение. К примеру, с помощью CSS можно изменять шрифт, цвет, размер, межстрочный интервал, разделять содержимое на колонки, а также добавлять анимацию и другие декоративные элементы.
- JavaScript – это язык программирования, который даёт возможность реализовывать сложное поведение веб-страницы. Веб-страница может

не только отображать статическое содержимое, но и своевременно показывать обновление контента, выводить интерактивные карты, 2D/3D анимацию, прокручивать видео и т.д.

Для распознавания текста с изображений был использован движок OCR Tesseracti уже обученные модели, а также библиотека для обработки изображений EmguCV.

2.2. Создание моделей

При создании проекта ASP.NET Core с паттерном MVC в нем автоматически создаются нужные папки для моделей, контроллеров и представлений. Для создания моделей нужно поместить в папку Models классы, соответствующие названиям сущностей базы данных. Таким образом, как показано на рисунке 2.2, был создан класс ComicBook, имеющий свойства:

- ComicBookId – код комикса, имеющий тип integer и являющийся первичным ключом;
- ComicBookTitle – название комикса, имеющее тип string;
- ComicBookLanguage – язык текста комикса, имеющий тип string;
- ComicBookCover – обложка комикса, имеющая тип string;
- ComicBookDescription – описание комикса, имеющее тип string;
- ComicBookYear – год выхода комикса, имеющий тип integer;
- ComicBookUsers – свойство, имеющее тип List<ComicBook_User>, показывающее отношение «Один ко многим» данной сущности к сущности «Комикс_Пользователь»;
- Authors – свойство, имеющее тип List<Author>, показывающее отношение «Многие ко многим» данной сущности к сущности «Авторы»;
- Genres – свойство, имеющее тип List<Genres>, показывающее отношение «Многие ко многим» данной сущности к сущности «Жанры»;

- Chapters – свойство, имеющее тип List<Chapters>, показывающее отношение «Один ко многим» данной сущности к сущности «Главы»;

```
public class ComicBook
{
    — ссылки
    public int ComicBookId { get; set; }
    [Required]
    — ссылки
    public string ComicBookTitle { get; set; }
    — ссылки
    public string ComicBookLanguage { get; set; }
    — ссылки
    public string ComicBookCover { get; set; }
    — ссылки
    public string ComicBookDescription { get; set; }
    — ссылки
    public int ComicBookYear { get; set; }
    — ссылки
    public List<ComicBook_User> ComicBookUsers { get; set; } = new List<ComicBook_User>();
    — ссылки
    public List<Author> Authors { get; set; } = new List<Author>();
    — ссылки
    public List<Genre> Genres { get; set; } = new List<Genre>();
    — ссылки
    public List<Chapter> Chapters { get; set; } = new List<Chapter>();
}
```

Рисунок 2.2 – Создание модели ComicBook

Подобным образом, как показано на рисунках, были созданы остальные модели приложения.

```
public class Author
{
    public int AuthorId { get; set; }

    public string AuthorName { get; set; }

    public string AuthorBirthday { get; set; }
    Ссылка: 2
    public string AuthorImage { get; set; }
    Ссылка: 1
    public List<ComicBook> ComicBooks { get; set; } = new List<ComicBook>();
}
```

Рисунок 2.3 – Создание модели Author

```

public class Chapter
{
    public int ChapterId { get; set; }

    public int ComicBookId { get; set; }

    public virtual ComicBook ComicBook { get; set; }

    public string ChapterName { get; set; }
    Ссылка: 9
    public List<ComicPage> ComicPages { get; set; } = new List<ComicPage>();
}

```

Рисунок 2.4 – Создание модели Chapter

```

public class ComicPage
{
    public int ComicPageId { get; set; }

    public int ChapterId { get; set; }

    public string PageImage { get; set; }
    Ссылка: 5
    public int PageNumber { get; set; }
    Ссылка: 0
    public string PageTranslatedImage { get; set; }
    Ссылка: 3
    public virtual Chapter Chapter { get; set; }
}

```

Рисунок 2.5 – Создание модели ComicPage

```

    Ссылка: 32
public class Genre
{
    Ссылка: 1
    public int GenreId { get; set; }
    [Required]
    Ссылка: 15
    public string GenreName { get; set; }
    Ссылка: 1
    public List<ComicBook> ComicBooks { get; set; } = new List<ComicBook>();
}

```

Рисунок 2.6 – Создание модели Genre

```

public class User
{
    public int UserId { get; set; }
    [Required]

    public string UserFirstName { get; set; }

    public string UserLastName { get; set; }
    [Required]

    public string Email { get; set; }
    [Required]

    public string Password { get; set; }

    public string UserImage { get; set; }
    Ссылка: 0
    public List<ComicBook_User> ComicBookUsers { get; set; } = new List<ComicBook_User>();
}

```

Рисунок 2.7 – Создание модели User

```

public class ComicBook_User
{
    [Key, Column(Order = 0)]
    Ссылка: 4
    public int ComicBookId { get; set; }
    [Key, Column(Order = 1)]
    Ссылка: 3
    public int UserId { get; set; }
    Ссылка: 0
    public virtual ComicBook ComicBook { get; set; }
    Ссылка: 0
    public virtual User User { get; set; }
    Ссылка: 0
    public bool IsFavourite { get; set; }
    Ссылка: 0
    public string ReadingStatus { get; set; }
    Ссылка: 4
    public string ComicBookReview { get; set; }
    Ссылка: 4
    public int ComicBookMark { get; set; }
    Ссылка: 0
    public int LastChapter { get; set; }
    Ссылка: 0
    public int LastPage { get; set; }
}

```

Рисунок 2.8 – Создание модели ComicBook_User

Далее, согласно рисунку 2.9, был создан класс-контекст AppDBContext, унаследованный от класса DbContext. В нем создаются свойства, имеющие тип DbSet<T>. В качестве T указываются названия созданных моделей. Данные свойства помогают получать из базы набор данных определенного типа. Так

как по умолчанию база данных отсутствует, то в конструктор класса AppDbContext вызывается метод Database.EnsureCreated(), чтобы база данных была создана.

```
public class AppDbContext : DbContext
{
    Ссылка: 6
    public DbSet<User> Users { get; set; }
    Ссылка: 5
    public DbSet<ComicBook> ComicBooks { get; set; }
    Ссылка: 3
    public DbSet<Author> Authors { get; set; }
    Ссылка: 5
    public DbSet<Chapter> Chapters { get; set; }
    Ссылка: 7
    public DbSet<ComicPage> ComicPages { get; set; }
    Ссылка: 3
    public DbSet<Genre> Genres { get; set; }
    Ссылка: 7
    public DbSet<ComicBook_User> ComicBook_Users { get; set; }

    Ссылка: 0
    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
    {
        //Database.EnsureDeleted();
        Database.EnsureCreated();
    }
}
```

Рисунок 2.9 – Создание класса AppDbContext

В методе OnModelCreating() описываются первичные ключи сущностей, а также добавляются нужные значения по умолчанию, в данном случае это фото профиля пользователя.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    // использование Fluent API
    base.OnModelCreating(modelBuilder);
    modelBuilder.Entity<ComicBook_User>().HasKey(cbu => new
    {
        cbu.ComicBookId,
        cbu.UserId
    });
    modelBuilder.Entity<User>().Property(u => u.UserImage).HasDefaultValue(value: "/img/user_logo.png");
}
```

Рисунок 2.10 – Метод OnModelCreating()

В соответствии с рисунком 2.11 был создан класс InitialData, в котором имеется метод Initialize, принимающий на вход контекст и предназначенный для заполнения базы начальными данными. Таким образом, при запуске программы эти данные внесутся в базу.


```

public class InitialData
{
    Ссылка: 1
    public static void Initialize(AppDbContext context)
    {
        if (!context.ComicBooks.Any())
        {
            ComicBook ComicBookAvatarTheSearch = new ComicBook
            {
                ComicBookTitle = "Аватар: Легенда об Аанге - Поиск",
                ComicBookLanguage = "English",
                ComicBookCover = "/img/ComicBooks/Avatar_The_Promise/cover.jpg",
                ComicBookDescription = "Зуко и Азула ищут свою мать Урсу вместе с командой Аватара, которая пытается" +
                    "держать психически неуравновешенную принцессу под контролем",
                ComicBookYear = 2014
            };

            context.ComicBooks.Add(ComicBookAvatarTheSearch);

            Author AuthorAvatarTheSearch1 = new Author
            {
                AuthorName = "Джин Луэнь Янг",
                AuthorBirthday = "9.08.1973",
                AuthorImage = ""
            };

            context.Genres.AddRange(Drama, Fantasy, Adventure, Action, Detective, Horror, Thriller, Tragedy, Romance, Sport);

            ComicBookAvatarTheSearch.Authors.Add(AuthorAvatarTheSearch1);
            ComicBookAvatarTheSearch.Authors.Add(AuthorAvatarTheSearch2);

            ComicBookAvatarTheSearch.Genres.Add(Drama);
            ComicBookAvatarTheSearch.Genres.Add(Fantasy);
            ComicBookAvatarTheSearch.Genres.Add(Adventure);
            ComicBookAvatarTheSearch.Genres.Add(Action);

            Chapter ChapterAvatarTheSearch1 = new Chapter
            {
                ComicBook = ComicBookAvatarTheSearch,
                ChapterName = "Аватар: Легенда об Аанге - Поиск. Часть 1"
            };

            Chapter ChapterAvatarTheSearch2 = new Chapter
            {
                ComicBook = ComicBookAvatarTheSearch,
                ChapterName = "Аватар: Легенда об Аанге - Поиск. Часть 2"
            };

            Chapter ChapterAvatarTheSearch3 = new Chapter
            {
                ComicBook = ComicBookAvatarTheSearch,
                ChapterName = "Аватар: Легенда об Аанге - Поиск. Часть 3"
            };

            context.Chapters.AddRange(ChapterAvatarTheSearch1, ChapterAvatarTheSearch2, ChapterAvatarTheSearch3);
        }
    }
}

```

Рисунок 2.11 – Внесение начальных данных в базу

2.3. Создание контроллеров

В проекте было имеется два контроллера – HomeController и AccountController. Первый предназначен для работы с главной страницей, страницей с комиксом, страницей для чтения и распознавания, а второй – для работы со страницами входа, регистрации и профиля пользователя.

Как показано на рисунке 2.12, класс наследуется от класса Controller. Также в нем создается конструктор, передающий контекст базы данных для работы с ней.

```
public class HomeController : Controller
{
    AppDbContext db;

    Ссылка: 0
    public HomeController(AppDbContext context)
    {
        db = context;
    }
}
```

Рисунок 2.12 – Создание HomeController

Затем, согласно рисунку 2.13, был создан метод ComicBook, принимающий на вход id комикса, возвращающий тип IActionResult. Это означает, что данный метод является действием. В нем был создан список cb_users с типом ComicBook_User, которому с помощью LINQ-запросов передаются все значения сущности ComicBook_Users, где ComicBookId равен id текущего комикса. Далее этот список присваивается переменной ViewBag.CB_Users, позволяющей передавать объекты из контроллеров в представления.

Затем был создан экземпляр класса ComicBook, которому присваивается комикс с нужным id. Данное действие осуществляется с помощью метода Find(). Далее этому экземпляру класса присваиваются нужные главы, авторы и жанры. Метод возвращает представление, в которое передана модель текущего комикса.

```
public IActionResult ComicBook(int id)
{
    List<ComicBook_User> cb_users = db.ComicBook_Users.Where(cbu => cbu.ComicBookId == id).ToList();
    ViewBag.CB_Users = cb_users;

    ComicBook comicBook = db.ComicBooks.Find(id);
    comicBook.Chapters = db.Chapters.Where(ch => ch.ComicBookId == id).ToList();
    comicBook.Authors = db.Authors.Include(a => a.ComicBooks).ToList();
    comicBook.Genres = db.Genres.Include(g => g.ComicBooks).ToList();
    return View(comicBook);
}
```

Рисунок 2.13 – Создание метода ComicBook

Как показано на рисунке 2.14, был создан метод `ComicChapter`, принимающий на вход `id` главы и также являющийся действием. В нем создан экземпляр класса `Chapter`, которому присваивается глава с нужным `id`. Таким же образом находится комикс по этой главе и присваиваются главы. С помощью метода `Count()` подсчитывается количество глав, находится `id` первой главы и передается в представление. Затем с помощью LINQ-запроса этому экземпляру класса присваиваются нужные страницы и сортируются в порядке возрастания по номерам страниц. Метод возвращает представление, в которое передана модель текущей главы.

```
public IActionResult ComicChapter(int id)
{
    Chapter chapter = db.Chapters.Find(id);
    int cbId = chapter.ComicBookId;
    ComicBook comicBook = db.ComicBooks.Find(cbId);
    comicBook.Chapters = db.Chapters.Where(ch => ch.ComicBookId == cbId).ToList();
    int chaptersCount = comicBook.Chapters.Count();
    int firstChapter = comicBook.Chapters.FirstOrDefault().ChapterId;
    ViewBag.ChapterCount = chaptersCount;
    ViewBag.FirstChapter = firstChapter;
    chapter.ComicPages = db.ComicPages.Where(cp => cp.ChapterId == id).OrderBy(cp => cp.PageNumber).ToList();
    return View(chapter);
}
```

Рисунок 2.14 – Создание метода `ComicChapter`

На рисунке 2.15 показано создание метода отправки отзыва. Атрибут `[Authorize]` перед объявлением метода означает, что вызвать его может только авторизированный пользователь, а `[HttpPost]` указывает, что метод обрабатывает тип запроса `Post`. На вход подаются три аргумента: `id` комикса, оценка и текст отзыва. Далее, если пользователь прошел аутентификацию, создается экземпляр класса `User`, которому присваивается текущий пользователь. Затем в базе данных ищется строка с нужными пользователем и комиксом. Если она не найдена, то создается новая, ей присваиваются оценка и текст отзыва. Если найдена, то эти же данные обновляются. С помощью метода `SaveChanges()` сохраняются изменения в базу данных.

```

[Authorize]
[HttpPost]
Ссылка: 0
public IActionResult SendReview(int mark, int cbId, string reviewValue)
{
    if (User.Identity.IsAuthenticated)
    {
        User user = db.Users.FirstOrDefault(u => u.Email == User.Identity.Name);
        int userId = user.UserId;
        ComicBook_User cb_user = db.ComicBook_Users.Find(cbId, userId);
        if (cb_user == null)
        {
            db.ComicBook_Users.Add(entity: new ComicBook_User
            {
                ComicBookId = cbId,
                UserId = userId,
                ComicBookMark = mark,
                ComicBookReview = reviewValue
            });
            db.SaveChanges();
        }
        else
        {
            cb_user.ComicBookReview = reviewValue;
            cb_user.ComicBookMark = mark;
            db.ComicBook_Users.Update(cb_user);
            db.SaveChanges();
        }
    }
    return RedirectToAction(actionName: "ComicBook", new {id = cbId});
}

```

Рисунок 2.15 – Создание метода SendReview

Таким же образом в соответствии с рисунком 2.16 создается метод для добавления в список для чтения и добавления в любимые.

```

[Authorize]
[HttpPost]
Ссылка: 0
public IActionResult AddStatus(string status, int cbId)
{
    if (User.Identity.IsAuthenticated)
    {
        User user = db.Users.FirstOrDefault(u => u.Email == User.Identity.Name);
        int userId = user.UserId;
        ComicBook_User cb_user = db.ComicBook_Users.Find(cbId, userId);

        if (cb_user == null)
        {
            db.ComicBook_Users.Add(entity: new ComicBook_User
            {
                ComicBookId = cbId,
                UserId = userId,
                ReadingStatus = status
            });
            db.SaveChanges();
        }
        else
        {
            cb_user.ReadingStatus = status;
            db.ComicBook_Users.Update(cb_user);
            db.SaveChanges();
        }
    }
}

```

Рисунок 2.16 – Создание метода AddStatus

Создание контроллера аккаунта, как показано на рисунке 2.17, происходит аналогичным образом.

```

public class AccountController : Controller
{
    private AppDBContext db;
    Ссылка: 0
    public AccountController(AppDBContext context)
    {
        db = context;
    }
}

```

Рисунок 2.17 – Создание контроллера аккаунта

Также были созданы модели для регистрации – RegisterModel и входа – LoginModel в соответствии с рисунками 2.18 и 2.19.

```

public class RegisterModel
{
    [Required(ErrorMessage = "Не указано имя")]
    Ссылка: 4
    public string UserFirstName { get; set; }

    Ссылка: 4
    public string UserLastName { get; set; }

    [Required(ErrorMessage = "Не указан Email")]
    Ссылка: 6
    public string Email { get; set; }

    [Required(ErrorMessage = "Не указан пароль")]
    [DataType(DataType.Password)]
    Ссылка: 4
    public string Password { get; set; }

    [DataType(DataType.Password)]
    [Compare(otherProperty: "Password", ErrorMessage = "Пароль введен неверно")]
    Ссылка: 3
    public string ConfirmPassword { get; set; }
}

```

Рисунок 2.18 – Модель для регистрации

```

public class LoginModel
{
    [Required(ErrorMessage = "Не указан Email")]
    Ссылка: 5
    public string Email { get; set; }

    [Required(ErrorMessage = "Не указан пароль")]
    [DataType(DataType.Password)]
    Ссылка: 4
    public string Password { get; set; }
}

```

Рисунок 2.19 – Модель для входа

В методе `Authenticate()`, согласно рисунку 2.20, происходит аутентификация пользователя. В нем создается новый `Claim`, который ответственен за создание набора данных, их шифрование и добавление в куки. Далее вызывается метод `SignInAsync`, отправляющий пользователю куки, которые при дальнейших запросах будут поступать обратно на сервер и использоваться для аутентификации пользователя.

```
private async Task Authenticate(string userName)
{
    // создаем один claim
    var claims = new List<Claim>
    {
        new Claim(ClaimsIdentity.DefaultNameClaimType, userName)
    };
    // создаем объект ClaimsIdentity
    ClaimsIdentity id = new ClaimsIdentity(claims, authenticationType: "ApplicationCookie", ClaimsIdentity.DefaultNameClaimType,
    // установка аутентификационных куки
    await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, principal: new ClaimsPrincipal(id));
}
```

Рисунок 2.20 – Метод для аутентификации

Далее был создан метод Login(), как показано на рисунке 2.20, который проверяет аутентифицирован ли пользователь. Если да, то происходит переход на страницу пользователя.

```
[HttpGet]
Ссылка: 0
public IActionResult Login()
{
    if (User.Identity.IsAuthenticated)
    {
        return RedirectToAction(actionName: "UserProfile", controllerName: "Account");
    }
    return View();
}
```

Рисунок 2.21 – Метод Login()

На рисунке 2.21 показан метод Login(LoginModel model), который ответствен за вход пользователя. Если нужный пользователь найден, вызывается метод аутентификации и происходит переход на страницу пользователя. Иначе высвечивается ошибка: "Некорректные логин и(или) пароль".

```
[HttpPost]
[ValidateAntiForgeryToken]
Ссылка: 0
public async Task<IActionResult> Login(LoginModel model)
{
    if (ModelState.IsValid)
    {
        User user = await db.Users.FirstOrDefaultAsync(u => u.Email == model.Email && u.Password == model.Password);
        if (user != null)
        {
            await Authenticate(model.Email); // аутентификация
            return RedirectToAction(actionName: "UserProfile", controllerName: "Account");
        }
        ModelState.AddModelError(key: "", errorMessage: "Некорректные логин и(или) пароль");
    }
    return View(model);
}
```

Рисунок 2.22 – Метод Login(LoginModel model)

В методе регистрации проверяется корректность введенных данных. Затем происходит поиск пользователя в базе данных по e-mail. Если таковой не

найден, происходит добавление новых данных пользователя в базу, их сохранение и переход на страницу пользователя. Иначе выводится ошибка о некорректных данных.

```
[HttpGet]
Ссылка: 0
public IActionResult Register()
{
    return View();
}
//== null ? "" : model.UserLastName
[HttpPost]
[ValidateAntiForgeryToken]
Ссылка: 0
public async Task<IActionResult> Register(RegisterModel model)
{
    if (ModelState.IsValid)
    {
        User user = await db.Users.FirstOrDefaultAsync(u => u.Email == model.Email);
        if (user == null)
        {
            // добавляем пользователя в бд
            db.Users.Add(entity: new User { Email = model.Email, Password = model.Password,
            UserFirstName = model.UserFirstName, UserLastName = model.UserLastName });
            await db.SaveChangesAsync();

            await Authenticate(model.Email); // аутентификация

            return RedirectToAction(actionName: "Index", controllerName: "Home");
        }
        else
        {
            ModelState.AddModelError(key: "", errorMessage: "Некорректные логин и(или) пароль");
        }
    }
    return View(model);
}
```

Рисунок 2.23 – Метод регистрации

На рисунке 2.24 отображен метод Logout(), отвечающий за выход пользователя из аккаунта.

```
public async Task<IActionResult> Logout()
{
    await HttpContext.SignOutAsync(CookieAuthenticationDefaults.AuthenticationScheme);
    return RedirectToAction(actionName: "Login", controllerName: "Account");
}
```

Рисунок 2.24 – Метод выхода из аккаунта

2.4. Создание представлений

Представления были созданы с помощью страниц Razor Pages, которые позволяют использовать код на С# внутри html-страниц. На рисунке 2.25 представлен код страницы Layout. Здесь подключаются все нужные стили, шрифты и скрипты. Также здесь описана навигационная панель. Внутри тега «main» вызывается метод `RenderBody()`, который позволяет передать в него другие представления.

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="preconnect" href="https://fonts.gstatic.com">
  <link href="https://fonts.googleapis.com/css2?family=Montserrat&display=swap" rel="stylesheet">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-awesome/4.5.0/css/font-awesome.min.css">
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-1BmE4kWBq7I" />
  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/az" />
  <link href="~/css/style.css" rel="stylesheet" />

  <link href="~/css/comicbook_page.css" rel="stylesheet" />
  <title>ComicBook</title>
</head>
<body>
  <header>
    <div class="topnav">
      <div class="logo">
        <div class="logo-cb">CB</div>
        <a class="logo-comicbook" asp-area="" asp-controller="Home" asp-action="Index">ComicBook</a>
      </div>
      <div class="search-container">
        <form action="/">
          <input type="text" placeholder="Поиск комиксов" name="search">
          <button class="search-btn" type="submit"><i class="fa fa-search"></i></button>
        </form>
      </div>
      <div>
        <a class="username item-nav" asp-action="Login" asp-controller="Account">Мой аккаунт</a>
      </div>
    </div>
  </header>

  <div>
    <main role="main" class="pb-3">
      @RenderBody()
    </main>
  </div>

  <footer class="border-top footer text-muted">
    <div class="container">
      &copy; 2022 - ComicBookReader - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </div>
  </footer>

  @*
  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
  <script src="~/js/site.js" asp-append-version="true"></script>
  @await RenderSectionAsync("Scripts", required: false)*@
</body>
</html>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-MrcW" />
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js"></script>
<script src="/js/home.js"></script>
@await RenderSectionAsync("Scripts", required: false)
```

Рисунок 2.25 – Страница Layout

Представление Index, как показано на рисунках 2.26 и 2.27, отображает главную страницу, на которой имеется список комиксов, сортировка, строка и фильтры поиска. Для строки поиска используется тег «input» и кнопкой поиска из библиотеки «Font awesome». Для создания фильтра по жанрам и языкам был использован js-скрипт. Всего имеется 13 жанров: драма, фантастика, приключения, экшн, детектив, ужасы, триллер, трагедия, романтика, спорт, психология, повседневность и научная фантастика, а также 3 языка: русский, английский и японский.

Также представление Index реализует модель ComicBook. В цикле foreach, согласно рисунку 2.27, происходит перебор всех комиксов в базе данных и вывод названия и обложки каждого на экран. В атрибут «href» тега «a» передается нужный id и при нажатии на название или обложку происходит переход на страницу данного комикса.

```
<html>
  <head>
    <title>ComicBookReader</title>
  </head>
  <body>
    <section class="content home-page">
      <h3>Все комиксы</h3>
      <div class="search-container-content">
        <form action="/">
          <input class="input-item" type="text" placeholder="Поиск комиксов" name="search">
          <button class="search-btn" type="submit"><i class="fa fa-search"></i></button>
        </form>
      </div>
      <div class="filters">
        <div class="filter-item">
          <p>Жанры:</p>
          <div class="checkselect">
            <label><input type="checkbox" name="brands[]" value="1"> Драма</label>
            <label><input type="checkbox" name="brands[]" value="2"> Фантастика</label>
            <label><input type="checkbox" name="brands[]" value="3"> Приключения</label>
            <label><input type="checkbox" name="brands[]" value="4"> Экшн</label>
            <label><input type="checkbox" name="brands[]" value="5"> Детектив</label>
            <label><input type="checkbox" name="brands[]" value="6"> Ужасы</label>
            <label><input type="checkbox" name="brands[]" value="7"> Триллер</label>
            <label><input type="checkbox" name="brands[]" value="8"> Трагедия</label>
            <label><input type="checkbox" name="brands[]" value="9"> Романтика</label>
          </div>
        </div>
      </div>
    </section>
  </body>
</html>
```

Рисунок 2.26 – Представление Index

```

        <label><input type="checkbox" name="brands[]" value="10"> Спорт</label>
        <label><input type="checkbox" name="brands[]" value="11"> Психология</label>
        <label><input type="checkbox" name="brands[]" value="12"> Повседневность</label>
        <label><input type="checkbox" name="brands[]" value="13"> Научная фантастика</label>
    </div></div>

    <div class = "filter-item">
        <p>Язык:</p>
        <div class="checkselect">
            <label><input type="checkbox" name="brands[]" value="1"> Русский</label>
            <label><input type="checkbox" name="brands[]" value="2"> Английский</label>
            <label><input type="checkbox" name="brands[]" value="3"> Японский</label>
        </div>
    </div>

    <div class = "filter-item">
        <p>Сортировка по:</p>
        <select>
            <option>Дате выхода (по убыванию)</option>
            <option>Дате выхода</option>
            <option>Названию</option>
            <option>Рейтингу</option>
        </select>
    </div>
</div>

<div class="comicbook-previews">
@foreach (var cb in Model)
{
    <a href="~/Home/ComicBook/@cb.ComicBookId" class="comicbook-preview">
        
        <p>@cb.ComicBookTitle</p>
    </a>
}

```

Рисунок 2.27 – Представление Index

В представление ComicBook передается модель комикса. В цикле foreach, как показано на рисунке 2.28, вычисляется средняя оценка комикса. Затем с помощью тегов «img», «h5» и «p» значения средней оценки и количества отзывов выводятся на экран. Также с помощью тега «a» создаются кнопки «Добавить в любимые» и «Оставить отзыв», а с помощью «select» создается выпадающий список с возможностью выбрать одно из значений: «Хочу прочитать», «Читаю», «Прочитано».

```

@foreach (var cbu in ViewBag.CB_Users)
{
    sum += cbu.ComicBookMark;
    c++;
}
@{av = Math.Round((double) sum / c, 2);}
<section class="content comicbook-page">
    <div class="cb-info-user">
        <div class="cb-info cbuser-info">
            
            <h5 class="rev-star">@av / 5</h5>
            <p class="rev-count">@c</p>
        </div>
        <div><a class="review-btn">Добавить в любимые</a></div>
        <div><a class="review-btn col change-info-btn btn btn-primary" type="button" data-bs-toggle="modal"
        <div class="cb-status">
            <form method="post">
                <select name="listStatus">
                    <option value="">Добавить в список</option>
                    <option value="1">Хочу прочитать</option>
                    <option value="2">Читаю</option>
                    <option value="3">Прочитано</option>
                </select>
                <br />
                <input type="submit" class="sign-in" value="Подтвердить" style="margin-top: 10px;" />
            </form>
        </div>
    </div>
</div>

```

Рисунок 2.28 – Представление ComicBook

На рисунке 2.29 демонстрируется вывод на экран следующей информации о комиксе: обложка, название, год выхода, язык, авторы, жанры и описание. Данные поступают из переданной ранее модели.

```

<div class="cb-info">
    
    <div class="cb-info-text">
        <h5>@Model.ComicBookTitle</h5>
        <p class="cb-info-item">Год выхода: @Model.ComicBookYear </p>
        <p class="cb-info-item">Язык: @Model.ComicBookLanguage</p>
        <p class="cb-info-item">Авторы:
            @foreach (var author in @Model.Authors)
            {
                <a>@author.AuthorName</a>
            }
        </p>
        <p class="cb-info-item">Жанры:
            @foreach (var genre in @Model.Genres)
            {
                <a>@genre.GenreName</a>
            }
        </p>
        <p class="cb-info-item">Описание: @Model.ComicBookDescription</p>
    </div>
</div>

```

Рисунок 2.28 – Представление ComicBook

На рисунке 2.29 иллюстрируется создание модального окна, которое открывается при нажатии на кнопку «Оставить отзыв». В нем с помощью тега «form» создается форма с полями для оценки по пятибалльной шкале и текста отзыва. При нажатии на кнопку «Отправить» происходит вызов действия «SendReview» из контроллера «Home».

```
<div class="modal fade" id="exampleModal" tabindex="-1" aria-labelledby="exampleModalLabel" aria-hidden="true">
  <div class="modal-dialog modal-lg">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">Оставить отзыв</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body">
        <div class="info-change">
          <div>
            <form action="/Home/SendReview" method="post">
              <div class="img-username">
                <p>Ваша оценка:</p>
                @*
                <a href="#"></a>
                <a href="#"></a>
                <a href="#"></a>
                <a href="#"></a>
                <a href="#"></a>
                *@
                <input class="mark" type="number" name="mark"/>
                <input class="cbId" type="number" name="cbId" style="display: none;" value="@Model.ComicBookId"/>
              </div>
              <textarea class="review-input" name="reviewValue"></textarea>
              <div class="form-group">
                <input class="sign-in" type="submit" value="Отправить" />
              </div>
            </form>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Рисунок 2.29 – Представление ComicBook

На рисунке 2.30 показан вывод соответствующих глав и отзывов.

```
<section class="content">
  <div class="chapters-reviews">
    <a class="ch" onClick="showChapters()"><h5>Главы</h5></a>
    <a class="rev" onClick="showReviews()"><h5>Отзывы</h5></a>
  </div>
  <hr />
  <div class="chapters">
    @for (int i = 0; i < @Model.Chapters.Count; i++)
    {
      <a href="~/Home/ComicChapter/@Model.Chapters.ElementAt(i).ChapterId" class="chapter"><h5>@Model.Chapters.ElementAt(i)
      <hr />
    }
  </div>
  <div class="reviews">
    @foreach (var cbu in ViewBag.CB_Users)
    {
      <div class="review">
        <div class="review-mark">
          
          <p class="rev-star">@cbu.ComicBookMark / 5</p>
        </div>
        <div class="img-username">
          
          <p>Username</p>
        </div>
        <p class="review-value">@cbu.ComicBookReview</p>
      </div>
    }
  </div>
</div>
```

Рисунок 2.30 – Представление ComicBook

Как демонстрируется на рисунке 2.31, был написан код на языке JavaScript для обработки нажатия на кнопки «Главы» и «Отзывы». Сначала создаются переменные, которым присваиваются нужные элементы html-документа: блоки глав и отзывов, а также сами кнопки. Далее создаются функции, которые вызываются при нажатии на кнопки «Главы» и «Отзывы». В первом случае отображается блок с главами, а во втором – блок с отзывами. Также происходит смена цвета фона кнопок.

```
var reviews = document.getElementsByClassName("reviews");
var chapters = document.getElementsByClassName("chapters");

var rev = document.getElementsByClassName("rev");
var ch = document.getElementsByClassName("ch");

function showReviews() {
    chapters[0].style.display = "none";
    ch[0].style.backgroundColor = "#fff";

    ch[0].addEventListener("mouseover", function() {
        this.style.backgroundColor = "#e8e2a2";
    })

    reviews[0].style.display = "block";
    rev[0].style.backgroundColor = "#e8e2a2";
}

function showChapters() {
    chapters[0].style.display = "block";
    ch[0].style.backgroundColor = "#e8e2a2";

    reviews[0].style.display = "none";
    rev[0].style.backgroundColor = "#fff";

    rev[0].addEventListener("mouseover", function() {
        this.style.backgroundColor = "#e8e2a2";
    })
}
```

Рисунок 2.31 – Код JavaScript для представления ComicBook

Аналогичным образом были написаны остальные представления.

2.5. Распознавание текста с изображений

Для распознавания текста с изображений был использован движок OCR Tesseract. Он имеет открытый исходный код и является наиболее популярной и качественной OCR-библиотекой. Также Tesseract использует нейронные сети для поиска и распознавания текста на изображениях.

Данный движок находит шаблоны в пикселях, буквах, словах и предложениях. Он использует двухэтапный подход, именуемый адаптивным распознаванием. Первый проход по данным нужен для распознавания символов, а второй проход – для заполнения букв, в которых он более уверен в соответствии контексту.

В настоящее время последней версией является Tesseract 5.0, основанная на нейросетях LSTM, которые и используются в данной работе.

Длинная цепь элементов краткосрочной памяти (англ. Long short-term memory; LSTM) — разновидность архитектуры рекуррентных нейронных сетей, предложенная в 1997 году Зеппом Хохрайтером и Юргеном Шмидхубером. Как и многие рекуррентные нейронные сети, сеть LSTM универсальна, поскольку при наличии достаточного количества элементов, она способна выполнять любые вычисления, на которые способен нормальный компьютер, что требует соответствующей матрицы весов.

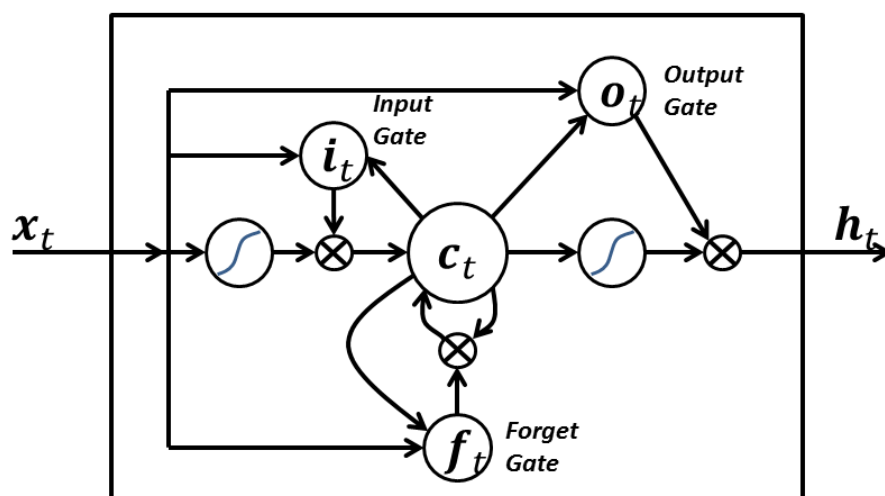


Рисунок 2.32 – Структура нейросети

LSTM-сеть, помимо других сетевых модулей, также содержит LSTM-модули. Они являются рекуррентными модулями сети и могут запоминать значения не только на короткий, но и на длинный период времени. Это происходит потому, что LSTM-модуль не использует функцию активации внутри своих рекуррентных компонентов. Поэтому значение, хранящееся в модуле, не размывается во времени, и градиент или штраф не исчезает при использовании метода обратного распространения ошибки во времени при обучении искусственной нейронной сети.

LSTM-блоки содержат три или четыре «вентилей», которые используются для контроля потоков информации на входах и на выходах памяти данных блоков. Эти вентили реализованы в виде логистической функции для вычисления значения в диапазоне $[0; 1]$. Умножение на это значение используется для частичного допуска или запрещения потока информации внутрь и наружу памяти. Например, «входной вентиль» контролирует меру вхождения нового значения в память, а «вентиль забывания» контролирует меру сохранения значения в памяти. «Выходной вентиль» контролирует меру того, в какой степени значение, находящееся в памяти, используется при расчёте выходной функции активации для блока.

Веса в LSTM-блоке (W и U) используются для задания направления оперирования вентилей. Эти веса определены для значений, которые подаются в блок (включая x_t и выход с предыдущего временного шага h_{t-1}) для каждого из вентилей. Таким образом, LSTM-блок определяет, как распоряжаться своей памятью как функцией этих значений, и тренировка весов позволяет LSTM-блоку выучить функцию, минимизирующую потери. LSTM-блоки обычно тренируют при помощи метода обратного распространения ошибки во времени.

Традиционная LSTM с вентилями забывания $c_0 = 0$ и $h_0 = 0$ (\circ обозначает произведение Адамара):

$$\begin{aligned}f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\h_t &= o_t \circ \sigma_h(c_t)\end{aligned}$$

Переменные:

- x_t — входной вектор,
- h_t — выходной вектор,
- c_t — вектор состояний,
- W , U и b — матрицы параметров и вектор,
- f_t , i_t , и o_t — векторы вентиляей,
- f_t — вектор вентиля забывания, вес запоминания старой информации,
- i_t — вектор входного вентиля, вес получения новой информации,
- o_t — вектор выходного вентиля, кандидат на выход.

Функции активации:

σ_g : на основе сигмоиды.

σ_c : на основе гиперболического тангенса.

σ_h : на основе гиперболического тангенса, но в работе о глазках (смотровых отверстиях) для LSTM предполагается, что $\sigma_h(x) = x$.

Для создания нейронной сети была использована нейросетевая библиотека EmguCV, предназначенная для работы на языке C#, как показано на рисунке 2.33.

```
using Emgu;  
using Emgu.CV;  
using Emgu.CV.Util;  
using Emgu.CV.OCR;  
using Emgu.CV.Structure;  
using Emgu.Util;
```

Рисунок 2.33 – Подключение библиотеки

В методе обработки нажатия на кнопку «Распознать текст» сначала проверяется наличие ошибок, в случае которых выбрасывается исключение. Если же ошибки не были найдены, создается новый объект Tesseract, который принимает на вход в качестве аргументов путь к обученным данным, язык текста и режим распознавания – в нашем случае это нейросеть LSTM. Затем экземпляру объекта Tesseract задается путь к выбранному изображению. Далее используется метод Recognize(), который выполняет распознавание текста. Распознанный текст выводится на экран, а после выполняется очистка экземпляра объекта Tesseract с помощью метода Dispose(). Ниже на рисунке 2.34 приведен код данного метода.

```
Tesseract tesseract = new Tesseract(dataPath: "wwwroot/TrainedData",  
    lang, OcrEngineMode.TesseractLstmCombined);  
  
tesseract.SetImage(new Image<Bgr, byte>(cropFilePath));  
  
tesseract.Recognize();  
  
string text = tesseract.GetUTF8Text();  
ViewBag.Text = text;  
  
tesseract.Dispose();  
  
return View(viewName: "RecognizeText", cp);
```

Рисунок 2.34 – Метод распознавания текста с изображения

Для того, чтобы можно было распознать лишь часть изображения, выбранную пользователем, была использована библиотека JQuery и встроенный метод Jcrop, позволяющий выделить фрагмент изображения. На рисунке 2.35 и 2.36 показан код использования данного метода. В него передается функция showCoords, которая принимает значения координат выделенного фрагмента и передает их значения полям формы по id, как показано на рисунке 2.37.

```

var jcrop_api,
boundx,
boundy,
// Grab some information about the preview pane
$preview = $('#preview-pane'),
$pcnt = $('#preview-pane .preview-container'),
$img = $('#preview-pane .preview-container img'),
$imgR = $('#imgRec'),
imgRWidth = $imgR.width(),
imgRHeight = $imgR.height(),

xsize = $pcnt.width(),
ysize = $pcnt.height();

jQuery('#imgW').val(imgRWidth);
jQuery('#imgH').val(imgRHeight);

console.log(imgRWidth);
console.log(imgRHeight);

$(function() {
    $('#imgRec').Jcrop({
        onChange: showCoords,
        onSelect: showCoords
    }, function() {
        // Use the API to get the real image size
        var bounds = this.getBounds();
        boundx = bounds[0];
        boundy = bounds[1];
        // Store the API in the jcrop_api variable
        jcrop_api = this;

        // Move the preview into the jcrop container for css positioning
        $preview.appendTo(jcrop_api.ui.holder);
    });
});

```

Рисунок 2.35 – Использование метода Jcrop

```

function showCoords(c) {
    if (parseInt(c.w) > 0)
    {
        $pcnt.css({
            width: c.w,
            height: c.h
        });

        $pimg.css({
            width: boundx + 'px',
            height: boundy + 'px',
            marginLeft: '-' + c.x + 'px',
            marginTop: '-' + c.y + 'px'
        });
    }

    jQuery('#x1').val(c.x);
    jQuery('#y1').val(c.y);
    jQuery('#x2').val(c.x2);
    jQuery('#y2').val(c.y2);
    jQuery('#w').val(c.w);
    jQuery('#h').val(c.h);
};

```

Рисунок 2.36 – Использование метода Jcrop

```

<form action="/Home/CropImage" method="post">
  <label>X1 <input type="number" size="4" id="x1" name="x1" style="width: 60px;"/></label>
  <label>Y1 <input type="number" size="4" id="y1" name="y1" style="width: 60px;"/></label>
  <label>X2 <input type="number" size="4" id="x2" name="x2" style="width: 60px;"/></label>
  <label>Y2 <input type="number" size="4" id="y2" name="y2" style="width: 60px;"/></label>
  <label>W <input type="number" size="4" id="w" name="w" style="width: 60px;"/></label>
  <label>H <input type="number" size="4" id="h" name="h" style="width: 60px;"/></label>
  <input type="number" size="4" name="imgW" id="imgW" style="display: none;"/>
  <input type="number" size="4" name="imgH" id="imgH" style="display: none;"/>
  <input type="number" size="4" name="cpId" id="cpId" value="@Model.ComicPageId" style="display: none;"/>
  <input type="submit" value="Распознать"/>
</form>



<div id="recTextBlock" style="position: fixed; top: 400px; right: 40px;">
  <p style="font-weight: bold; background-color: #fff; width: 200px;">Распознанный текст:</p>
  <textarea id="recText" type="text" style="width: 400px; height: 300px;">@ViewBag.Text</textarea>
</div>

```

Рисунок 2.37 – Создание формы для координат

Для обработки изображения по координатам внутри контроллера «Home» был создан метод CropImage(). Согласно рисунку 2.38, он принимает на вход координаты и id изображения. Далее он копирует это изображение в отдельную папку. Затем по имеющимся координатам создается экземпляр объекта Rectangle под названием CropArea и с помощью метода DrawImage происходит обрезка изображения по данному экземпляру. Файл сохраняется, и его путь передается на вход методу tesseract.SetImage() для дальнейшего распознавания. Полученный распознанный текст выводится на экран с помощью объекта ViewBag.

```
[HttpPost]
Ссылка: 0
public IActionResult CropImage(int x1, int y1, int x2, int y2, int w, int h, int imgW, int imgH, int cpId)
{
    string lang = "eng";

    ComicPage cp = db.ComicPages.Find(cpId);

    string newFileName = Guid.NewGuid() + ".jpg";
    string newFilePath = "wwwroot/img/crop/";

    string filePath = Path.Combine(newFilePath, newFileName);

    System.IO.File.Copy("wwwroot" + cp.PageImage, filePath);
    Image orgImg = Image.FromFile(filePath);
    Rectangle CropArea = new Rectangle(
        Convert.ToInt32(x1),
        Convert.ToInt32(y1),
        Convert.ToInt32(w),
        Convert.ToInt32(h));
    Bitmap bitMap = new Bitmap(CropArea.Width, CropArea.Height);
    using (Graphics g = Graphics.FromImage(bitMap))
    {
        g.DrawImage(orgImg, destRect: new Rectangle(x: 0, y: 0, bitMap.Width, bitMap.Height), CropArea);
    }
    var cropFileName = "crop_" + newFileName;
    var cropFilePath = Path.Combine(newFilePath, cropFileName);
    bitMap.Save(cropFilePath);

    ViewBag.Preview = "/img/crop/" + cropFileName;
}
```

Рисунок 2.38 – Код метода CropImage()

Результаты распознавания текста с изображений показаны на рисунках 2.39, 2.40 и 2.41 на английском, русском и японском языках соответственно.



Рисунок 2.39 – Результат распознавания с английского языка

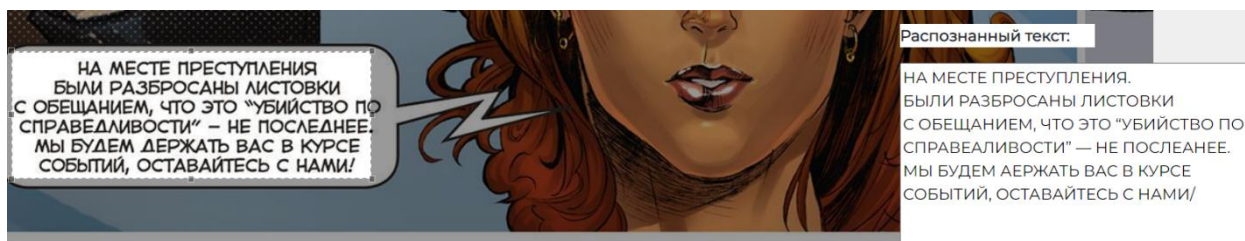


Рисунок 2.40 – Результат распознавания с русского языка

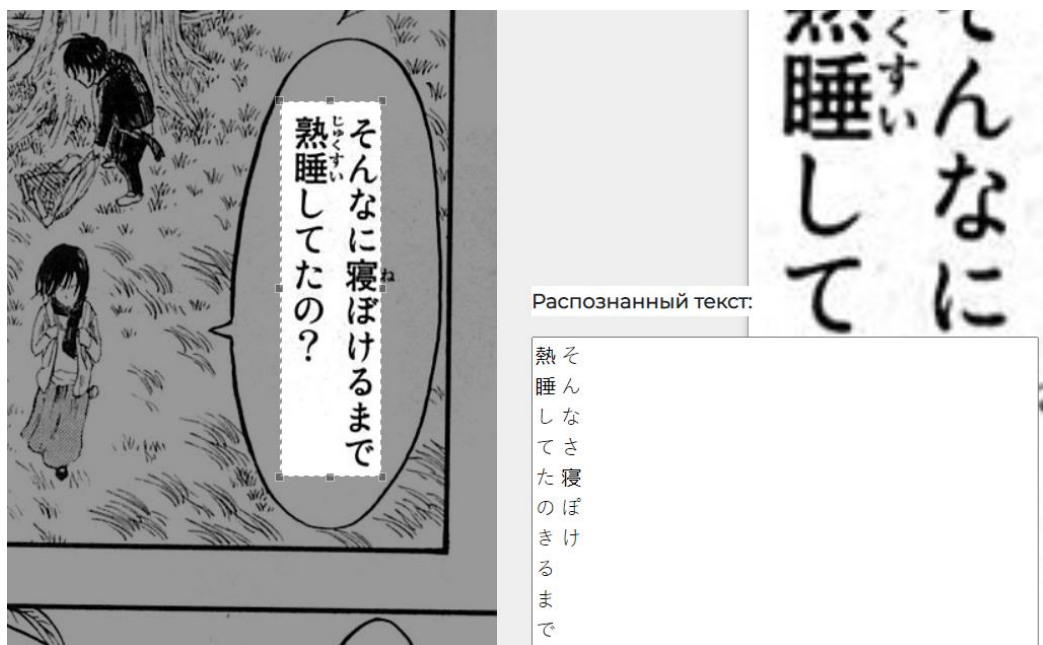


Рисунок 2.41 – Результат распознавания с японского языка

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы было разработано веб-приложение для чтения комиксов с возможностью распознавания текста с изображения. Были реализованы все задачи проводимого исследования. Была проанализирована предметная область, определены функциональные и нефункциональные требования к системе, построена карта информационной системы, созданы макеты приложения; разработаны и представлены клиентская и серверная части информационной системы. Также был разработан привлекательный и эффективный дизайн, соответствующий ожиданиям целевой аудитории. В процессе были использованы современные методы разработки сайта, а также инструменты последних версий, такие как ASP.NET Core 5, MS SQL Server, EntityFramework Core, HTML5, CSS3, JavaScript, OCR Tesseract и EmguCV. Полученное веб-приложение позволяет читать комиксы. Также сайт предполагает возможность регистрации пользователей, внесения изменений в личные данные, просмотр информации о комиксе, их оценивание и добавление в список для чтения. Система отвечает исходным требованиям по оперативности, достоверности, надежности и адекватности.

Результаты исследования могут быть применены для облегчения получения пользователями актуальной информации о фильмах.

В процессе исследования были получены знания и сформированы навыки практической работы, связанной с разработкой и реализацией информационных систем.

СПИСОК ЛИТЕРАТУРЫ

1. Фримен Адам ASP.NET MVC 4 с примерами на C# 5.0 для профессионалов; Вильямс - Москва, 2013. - 688 с.
2. Фримен Адам, Сандерсон Стивен ASP.NET MVC 3 Framework с примерами на C# для профессионалов; Вильямс - Москва, 2011. - 672 с.
3. Чедвик Джесс, Снайдер Тодд , Панда Хришикеш ASP.NET MVC 4. Разработка реальных веб-приложений с помощью ASP.NET MVC; Вильямс - Москва, 2013. - 432 с.
4. Эспозито Дино Программирование на основе Microsoft ASP.NET MVC; БХВ-Петербург, Русская Редакция - Москва, 2012. - 464 с.
5. Adam Freeman Pro ASP.NET MVC 4; Apress -, 2013. - 756 с.
6. Bill Evjen, Christian Nagel, Joe Duffy, Tod Golding, Scott Hanselman .NET 2.0 Wrox Box: Professional ASP.NET 2.0, Professional C# 2005, Professional .NET 2.0 Generics, and Professional .NET Framework 2.0; Wrox - Москва, 2006. - 709 с.
7. Chris Sutton Programming ASP. NET MVC - Москва, 2008. - 350 с.
8. D Esposito Building Web Solutions with ASP.Net and ADO .Net - Москва, 2002. - 416 с.
9. David Gefen, Chittibabu Govindarajulu Advanced Visual Basic.NET: Programming Web and Desktop Applications in ADO.NET and ASP.NET - Москва, 2000. - 595 с.
10. Богданов В., Ахметов К. Системы распознавания текстов в офисе. // Компьютер-пресс -- 1999 №3, с.40-42.
11. Павлидис Т. Алгоритмы машинной графики и обработки изображений. М.; Радиоисвязь, 1986
12. Shani U. Filling Regions in Binary Raster Images -- a Graph-theoretic Approach. // SIGGRAPH'80, pp 321-327.

13. Merrill R.D. Representation of Contours and Regions for Efficient Computer Search. // CACM, 16 (1973), pp. 69-82.
14. Pavlidis T. Filling Algorithms for Raster Graphics. // CGIP, 10 (1979), pp. 126141.
15. Албахари, Джозеф С# 6.0. Справочник. Полное описание языка / Джозеф Албахари, Бен Албахари. - М.: Вильямс, 2016. - 418 с.