**Министерство Образования, Культуры и Исследований**

**Технический университет Республики Молдова**

**Факультет Вычислительной техники, Информатики и Микроэлектроники**

**Департамент программной инженерии и автоматики**

# Курсовой проект

## по предмету

**«Проектирование информационных систем»**

**Тема: «РАЗРАБОТКА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ЧТЕНИЯ И ПРОДАЖИ КНИГ»**

**« DEZVOLTAREA UNEI APLICAȚII WEB PENTRU CITIREA ȘI VÂNZAREA CĂRȚILOR»**

**« DEVELOPMENT OF WEB APPLICATION FOR READING AND SELLING BOOKS»**

***Выполнил:*** *студент группы TI-209 Корман Александр*

***Проверила:*** *ассист. унив. Скороходова Т. А.*

Кишинев 2023

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

В современном мире, где Интернет есть у каждого, где он заменяет человеку большую часть жизни, где компьютеры и телефоны стали неотъемлемой частью повседневной жизни, многие люди забывают, что есть что-то вне Интернета, например, бумажные книги. Сейчас мало кто читает книги, тем более бумажные, ведь это так неудобно, книга громоздкая и ее нельзя брать куда захочется, но все же читающие люди остались и электронные книги стали для них спасением, ведь электронную книгу можно читать где угодно. С целью предоставить пользователям возможность читать те книги, которые им нравятся, в данной курсовом проекте будет проведена работа над созданием веб-приложения для чтения электронных книг.

В ходе написания курсового проекта будет рассмотрена разработка веб-приложения, которое позволит не только читать, а также покупать, писать и скачивать книги.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В этой главе идет речь об анализе предметной области, которая включает в себя анализ предметной области дипломного проекта. Осуществлен анализ существующей информации о аналогах разрабатываемого приложения, включая их краткое описание, преимущества и недостатки, представленные в сравнительной таблице. Этот этап предоставляет ответы на вопросы о характере приложения, его предполагаемой структуре, целевой платформе и области применения в рамках дипломного проекта. Анализ занимает важное место в процессе планирования системы, представляя собой комплексный обзор различных аналитических аспектов разрабатываемого приложения.

## 1.1 Важность темы

Веб-приложения для чтения книг является прогрессивным способом чтения книг. Один из важнейших аспектов сайтов для чтения книг - это обеспечение доступа к литературе разных стран и культур. Сайты предоставляют доступ к классическим произведениям, позволяя читателям познакомиться с литературным наследием разных эпох и культур. Это способствует культурному обогащению и пониманию разнообразия мировой литературы.

Сайты для чтения книг также помогают расширить аудиторию для авторов. Электронные книги доступны во всем мире и не ограничены тиражами и физическими магазинами. Это дает возможность молодым или малоизвестным авторам представить свои произведения широкой публике, а читателям - найти новых авторов и жанры, которые могли бы им понравиться. Сайты для чтения книг также играют важную роль в образовании.

Студенты, учителя и исследователи могут находить необходимую литературу онлайн и изучать ее без дополнительных затрат. Это также удобно для тех, кто изучает иностранные языки, так как они могут читать тексты на оригинальном языке и одновременно изучать перевод.

Популярность сайтов для чтения книг способствует сокращению использования бумаги и уменьшению экологической нагрузки. Электронные книги не требуют древесных ресурсов для производства и не создают отходов, связанных с утилизацией бумажных книг. Это способствует сохранению природных ресурсов и снижению воздействия на окружающую среду.

В данном случае будут собраны все преимущества различных площадок в разрабатываемое приложение, убрав соответствующие минусы. В таком случае приложение будет в разы лучше выглядеть в глазах пользователей, чем другие.

### 1.2 Существующие аналоги

Перед тем как разработать веб-приложеие читалку, важно провести анализ уже существующих аналогов. Это позволит определить особенности, преимущества и недостатки конкурирующих продуктов.

Одним из аналогов, который был рассмотрен, является "Author Today" (рисунок 1.1) — веб-приложение с похожим функционалом и большой библиотекой книг. В "Author Today" пользователь может читать книги, используя веб-приложение или мобильное приложение. Книги могут платными и бесплатными, автор сам задает это, автор может ограничивать чтение по главам, одни бесплатные, другие нет. Автор может обьединять книги по циклам, писать свой блог. Читатель же может оценивать произведения, писать комментарии и благодарить автора.

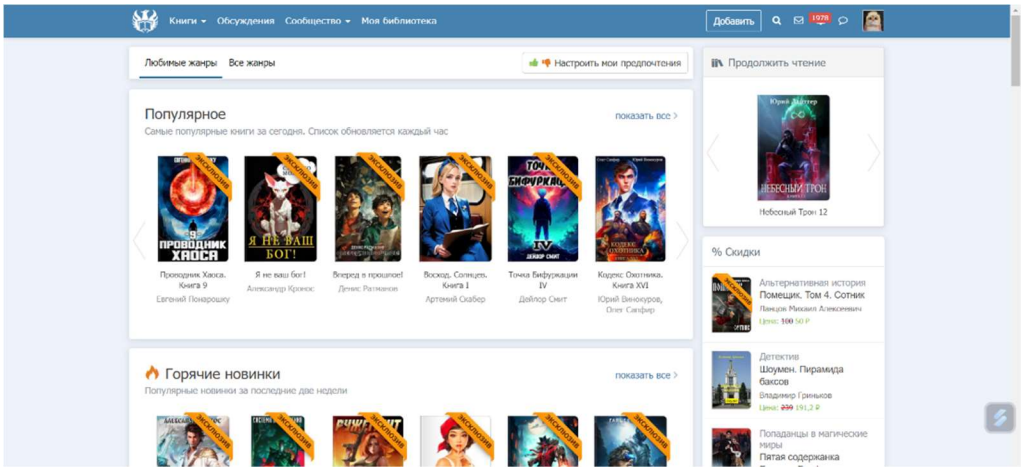


**Рисунок 1.1 – "Author Today"**

Еще одним аналогом является " Litres" (рисунок 1.2). Обычная читалка ничем особым не выделяющаяся. Посредственный дизайн усложняет взаимодействие пользователя с приложениям. Предоставляет стандартные функции для читателя такие как: чтения книг, комментирование.

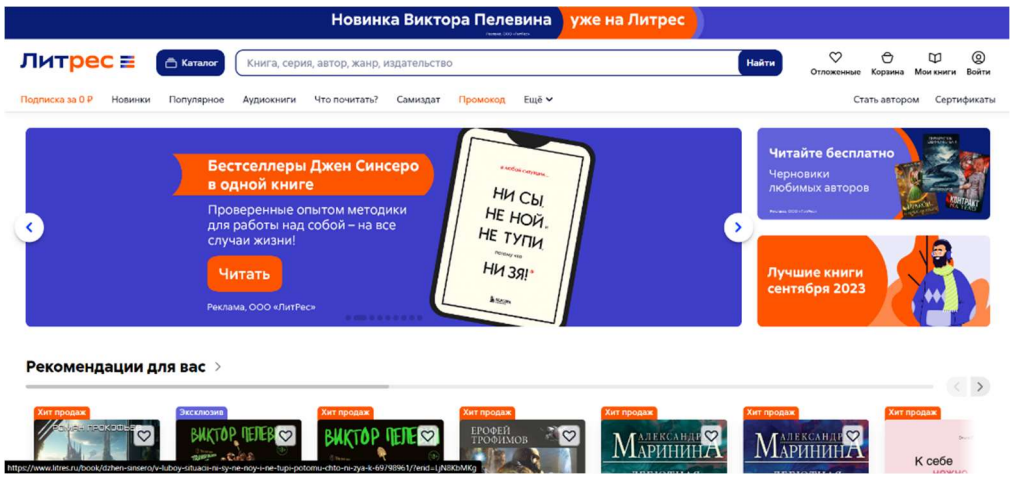


**Рисунок 1.2 – "Litres"**

Еще одним интересным аналогом является "Литнет" (рисунок 1.3). Приятный дизайн, множество понятных пользователю функции как в других читалках. Отличается совсем незначительно от других.

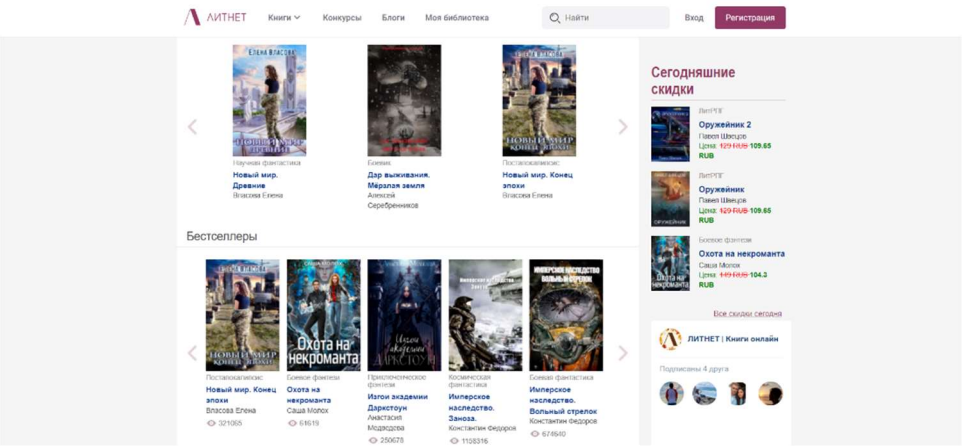


**Рисунок 1.3 – "Литнет"**

В таблице 1.1 представлены основные критерии, которые используются для сравнения нашего разрабатываемого приложения с существующими аналогами. Эта таблица предоставляет инструмент для более наглядного выявления преимуществ и недостатков наших систем относительно конкурентов.

**Таблица 1.1 – Сравнительный анализ аналогов**

| Критерии | Author Today | Litres | Litnet | Разрабатываемое приложение |
|---|---|---|---|---|
| **Чтение книги** | **Да** | **Да** | **Да** | **Да** |
| **Создание комментариев** | **Да** | **Да** | **Да** | **Да** |
| **Мобильное приложение** | **Да** | **Нет** | **Нет** | **Да** |
| **Назойливая реклама** | **Да** | **Да** | **Нет** | **Нет** |
| **Уведомления при выходе нового** | **Да** | **Нет** | **Нет** | **Да** |
| **Аудио книги** | **Да** | **Нет** | **Нет** | **Да** |
| **Умный подбор рекомендаций** | **Нет** | **Нет** | **Нет** | **Да** |

### 1.3 Цель, задачи и требования к системе

Цель дипломного проекта является веб-приложения. Для создания приложения было принято решение выбрать для бэкенда “ASP .NET CORE” – кроссплатформенныый фреймворк для содания MVC, API приложений. В данном приложение используется MVC принцип разработки бэкенд приложения и немного RESTful API. Для фронтэнда используется чистый javascript совместно с Vue.js для реактивности и немного jQuery. В качестве библеотек для дизайна были выбраны Fomantic UI и Bootstrap.

Можно выделить несколько задач, по результату выполнения которых может быть достигнута поставленная цель:

- создание бэкенда;
- создание фронтенда;
- настройка базы данных;
- создание дополнительных возможностей;
- тестирование приложения на разных этапах реализации.

Создание сложных программных приложений требует разделения жизненного цикла программного обеспечения на определенные этапы. Жизненный цикл программного приложения включает следующие ключевые этапы:

- анализ предметной области и создание технического задания – на этом этапе проект существует лишь на бумаге, происходит изучение предметной области и формирование технического задания;
- анализ и проектирование структуры приложения – тот этап включает создание документации, описывающие внутренние взаимодействия и взаимодействие с системой;
- реализация – на данном этапе осуществляется непосредственная реализация системы в соответствии с проектной документацией;
- тестирование и отладка – этап проверки насколько проект соответствует требованиям, выявления ошибок и их последующего устранения;
- утилизация – непосредственно использование продукта;
- обновления приложения – этап подразумевает выход обновлений для игры, после её релиза и поддержка приложения в целом.

### 1.4 Язык программирования

Для создания веб-приложения будет использоваться C# (рисунок 1.4). C# — это мощный и универсальный объектно-ориентированный язык программирования, разработанный компанией Microsoft.

Объектно-ориентированный язык: C# построен вокруг концепций объектно-ориентированного программирования (ООП), что облегчает создание модульных и структурированных программ. Интеграция с платформой .NET: C# является основным языком для разработки приложений на платформе .NET (Microsoft .NET Framework и .NET Core). Это обеспечивает высокую переносимость кода и возможность работы на различных операционных системах. Сильная типизация: C# предоставляет строгую, статическую типизацию, что помогает обнаруживать ошибки на этапе компиляции, а не во время выполнения программы. Управление памятью: Язык C# использует систему управления памятью, основанную на сборке мусора, что облегчает разработку и предотвращает многие проблемы, связанные с утечками памяти. Многозадачность: C# поддерживает асинхронное программирование и многозадачность, что позволяет эффективно работать с асинхронными операциями и параллельными вычислениями. Богатая стандартная библиотека: C# поставляется с обширной стандартной библиотекой классов, предоставляющей широкий спектр функциональности для работы с файлами, сетевыми протоколами, базами данных и многими другими аспектами программирования. Современные возможности языка: C# постоянно развивается, добавляя новые функции и синтаксические конструкции. Введение асинхронного программирования, лямбда-выражений, LINQ (Language Integrated Query) и других возможностей делает его современным и эффективным языком программирования. C# широко используется для создания различных типов приложений, включая веб-приложения, настольные приложения, мобильные приложения, игры и многое другое.

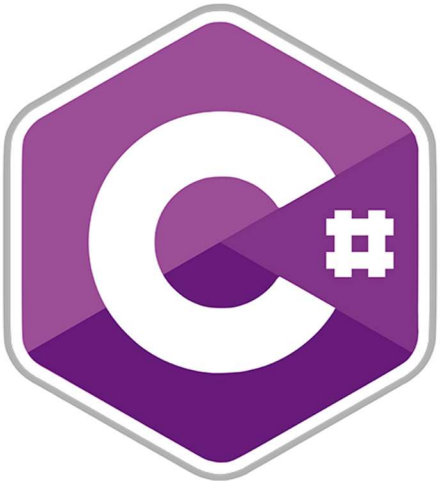

**Рисунок 1.4 – Язык C#**

### 1.5 Среда разработки

Для создания веб-приложения будет использоваться среда разработки Visual Studio (рисунок. 1.5).

Visual Studio - это интегрированная среда разработки (IDE) от Microsoft, предназначенная для создания различных типов приложений, включая веб-приложения, настольные приложения, мобильные приложения, игры и другие программные решения. Многозадачность и поддержка множества языков: Visual Studio предоставляет возможность разработки на различных языках программирования, таких как C#, Visual Basic, F#, C++, Python, и другие. Это позволяет разработчикам выбирать язык в зависимости от требований проекта. Интеграция с платформой .NET: Visual Studio тесно интегрирована с платформой .NET, что облегчает создание, отладку и развертывание приложений для .NET Framework и .NET Core. Отладка и профилирование: Инструменты отладки Visual Studio обеспечивают широкие возможности для выявления и устранения ошибок в коде. Также предоставляются средства профилирования для анализа производительности приложений. Графический дизайнер интерфейса: Visual Studio содержит инструменты для визуального проектирования пользовательского интерфейса приложений, что облегчает создание привлекательных и функциональных пользовательских интерфейсов. Управление версиями и командная работа: Встроенная поддержка систем контроля версий, таких как Git, облегчает управление и отслеживание изменений в коде. Интеграция с платформой Azure DevOps обеспечивает совместную работу в командах. Расширяемость: Visual Studio поддерживает расширения и плагины, что позволяет разработчикам интегрировать сторонние инструменты и расширять функциональность среды разработки. Поддержка различных платформ: Visual Studio позволяет разрабатывать приложения для различных платформ, включая Windows, Linux, macOS, мобильные устройства и облачные сервисы. Кросс-платформенная разработка: С Visual Studio можно создавать приложения для различных операционных систем, в том числе использовать .NET Core для разработки кросс-платформенных приложений. Visual Studio является одним из наиболее распространенных инструментов разработки в индустрии программного обеспечения и широко используется профессиональными разработчиками.


**Рисунок 1.5- Visual Studio**

# 2 ТРЕБОВАНИЯ К СИСТЕМЕ

В данной главе разбираются функциональные и нефункциональные требования, а также поток данных к нашей системе, которые помогут лучше разобраться в том, как работает каждый режим игры и как он реализуется.

## 2.1 Функциональные требования

Функциональные требования - это часть спецификации системы или программного продукта, которая описывает функции, операции и возможности, которые должны быть реализованы в продукте. Они фокусируются на том, что система должна делать, и какие функциональные возможности она должна предоставить пользователю. Функциональные требования часто определяются как "что" система должна делать.

Примеры функциональных требований могут включать:

- Регистрация пользователя: Система должна предоставлять форму для регистрации новых пользователей. При этом требуется сбор информации, такой как имя, адрес электронной почты и пароль.
- Поиск функции: Система должна обеспечивать возможность поиска информации на основе заданных критериев с использованием поискового механизма.
- Добавление товара в корзину: В электронной коммерции система должна позволять пользователям добавлять товары в корзину покупок, а затем осуществлять оформление заказа.
- Выполнение арифметических операций: В приложении калькулятора функциональные требования могут включать выполнение базовых арифметических операций, таких как сложение, вычитание, умножение и деление.
- Генерация отчетов: Система управления базами данных должна предоставлять возможность генерации отчетов по определенным критериям.

В первую очередь система предоставляет пользователю возможность читать книги в онлайн, но только бесплатные. Для чтения платных книг, соответственно и их покупка доступна только зарегистрированным пользователям. Зарегистрированные пользователи так же могут оставлять комментарии, подписываться на обновления книг и авторов, скачивать книги, так же сами писать свои книги. То есть можно смело утверждать, что незарегистрированный пользователь сильно ограничен в возможностях использования системы.

### 2.2 Поток данных

Поток данных (Data Flow) - это концепция, которая описывает передачу данных в системе, программе или процессе. Он представляет собой путь, по которому данные перемещаются от одного места к другому внутри системы. Концепция потока данных часто используется в контексте программирования, проектирования информационных систем, а также в области обработки данных и анализа. В программировании поток данных может быть представлен следующим образом:

- Ввод данных (Input): Получение данных из внешних источников, таких как пользовательский ввод, файлы или сенсоры.
- Обработка данных (Processing): Применение логики и алгоритмов к входным данным для получения желаемого результата. Этот этап может включать операции фильтрации, сортировки, вычислений и другие манипуляции с данными.
- Вывод данных (Output): Предоставление результатов обработки в виде вывода, который может быть направлен на экран, в файл, базу данных или другое место.

Проектирование потока данных позволяет легко представлять и отслеживать перемещение информации в системе, что важно для понимания работы приложений и оптимизации процессов обработки данных. В области обработки данных и анализа поток данных также может означать непрерывный поток информации, который поступает и обрабатывается в режиме реального времени. Например, в системах обработки потоков данных (stream processing) данные поступают и обрабатываются по мере их поступления, а не после того, как весь объем данных собран. Таким образом, поток данных представляет собой путь передвижения информации внутри системы, процесса или программы от начального источника до конечного пункта назначения, проходя через различные этапы обработки и манипуляций.

Система использует множество данных для своей работы. В первую очередь используется данные пользователя, которые попадают в систему через форму регистрации во фронтенд части и попадают в базу данных через бэкенд часть. Так же часть данных может изменяться или дополняться через форму аккаунта в личном кабинете. В систему попадает книги в полном обьеме загруженные в веб-приложение, комментарии к этим книгам, потом эти данные используются для отображения в некоторых частях веб-приложения. В системе используются алгоритмы для подбора книг по предпочтениям для пользователей основываясь на уже прочитанных книгах, добавленных в библиотеку.

### 2.3 Нефункциональные требования

Нефункциональные требования - это аспекты, характеристики или ограничения системы, которые не описывают конкретные функции, которые система должна выполнять, а скорее определяют свойства, качества или ограничения, которые должны присутствовать в системе или в ее компонентах. Эти требования описывают "как" система должна выполнять свои функции, а не "что" она должна делать. Примеры нефункциональных требований включают:

- Производительность: определение требований по скорости работы, времени отклика и обработки данных системы в различных условиях.
- Надежность: указание на уровень стабильности, надежности и устойчивости системы, включая требования к отказоустойчивости и восстановлению после сбоев.
- Безопасность: описывает требования по обеспечению защиты данных, аутентификации пользователей, управлению доступом и другим аспектам безопасности.
- Масштабируемость: устанавливает требования по способности системы масштабироваться для обработки увеличения объема данных или пользователей.
- Удобство использования (Usability): включает требования по дизайну интерфейса, простоте использования, интуитивности и удобству взаимодействия с пользователем.
- Сопровождаемость (Maintainability): определение того, насколько легко можно поддерживать, модифицировать и обновлять систему.
- Совместимость: описывает требования к совместимости системы с другими системами, стандартами и аппаратным обеспечением.
- Эффективность использования ресурсов: определение эффективного использования ресурсов, таких как память, процессорное время, энергия и т. д.

Нефункциональные требования являются важной частью общего набора требований, так как они влияют на аспекты системы, которые могут быть критическими для ее успеха, надежности и удовлетворения потребностей пользователей и бизнеса.

Разрабатывая веб-приложение, были выделены нефункциональные требования необходимые для комфортного использования системы пользователями.

a) Производительность:
    1) Время загрузки страницы должно быть не более 2 секунд для основного контента.
    2) Система должна поддерживать одновременное обслуживание не менее 500 активных пользователей без существенного снижения производительности.
b) Доступность:

1) Сайт должен обеспечивать доступность не менее 99% времени в течение месяца.
2) Веб-приложение должно поддерживать чтение книги в оффлайн-режиме после их предварительной загрузки.

c) Безопасность:
   1) Все пользовательские данные должны передаваться по протоколу HTTPS.
   2) Система должна обеспечивать защиту от угроз безопасности, такие как SQL-инъекции и межсайтовые атаки (XSS, CSRF).

d) Удобство использования:
   1) Интерфейс должен быть интуитивно понятным для пользователей всех уровней.
   2) Система должна предоставлять возможности настройки шрифта, цветовой схемы и других параметров для удовлетворения индивидуальных потребностей пользователей.

# 3 АНАЛИЗ И МОДЕЛИРОВАНИЕ СИСТЕМЫ

Процесс моделирования является один из важнейших компонентов в разработке программного обеспечения. Моделирование системы является важным инструментом для эффективного проектирования, анализа и внедрения сложных систем, позволяя участникам проекта иметь общее понимание целевой системы. Моделирование системы помогает участникам проекта и заинтересованным сторонам лучше понять структуру, функциональность и взаимодействие компонентов системы.

## 3.1 Разработка логической модели БД

Чтобы посторить модель базы данных использовались мощности программы SSMS.

Для создания диаграммы зависимостей модели базы данных в SSMS необходимо сначала создать эту базу данных, задать все поля, так же добавить ключи для связей.

**Создание сущностей и добавление определений**

В первую очередь определяется, что есть пользователь в системе. Это характеризуется таблицей User: UserId (Primary Key) – уникальный идентификатор пользователя, Username – имя пользователя в системе, Email – электронный почтовый адрес пользователя для связи с ним вне системы, Password – захэшированный пароль пользователя для входа в систему, CreatedAt – дата регистрации / создания аккаунта.  На рисунке 3.1 представлен пример создания таблицы.

```
CREATE TABLE [dbo].[User](
    [UserId] [int] IDENTITY(1,1) NOT NULL,
    [Username] [nvarchar](255) NOT NULL,
    [Email] [nvarchar](255) NOT NULL,
    [Password] [nvarchar](255) NOT NULL,
    [CreatedAt] [datetime] NOT NULL,
 CONSTRAINT [PK_User] PRIMARY KEY CLUSTERED
(
    [UserId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [IX_Email] UNIQUE NONCLUSTERED
(
    [Email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [IX_Username] UNIQUE NONCLUSTERED
(
    [Username] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

**Рисунок 3.1 – Пример создания таблицы для модели БД**

Таблица Book характеризуется следующими полями: BookId (Primary Key) – уникальный идентификатор книги, Title – название книги, Description – аннотация книги, Author (Foreign Key) – внешний ключ ссылающийся на таблицу User (UserId) и характеризующий автора книги, CreatedAt – дата публикации книги.

Таблица Comment характеризуется следующими полями: CommentId (Primary Key) – уникальный идентификатор комментария, Text – содержание комментария, AuthorId (Foreign Key) – внешний ключ ссылающийся на таблицу User (UserId) и характеризующий автора комментария, BookId (Foreign Key) – внешний ключ ссылающийся на таблицу Book (BookId) и характеризующий к какой книге принадлежит комментарий, CreatedAt – дата публикации комментария.

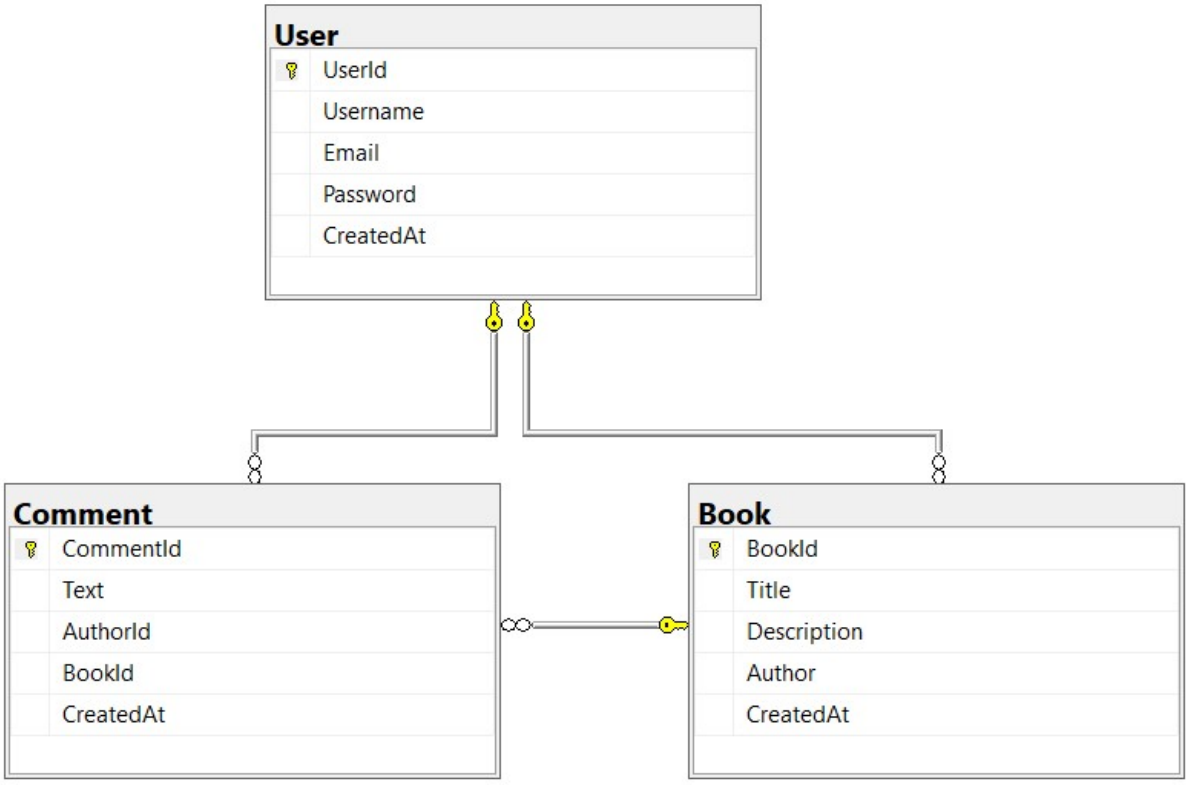


**Рисунок 3.2 – Логическая схема базы данных**

### 3.2 Разработка контекстной диаграммы и её декомпозиция

Контекстная диаграмма - это вид диаграммы, который позволяет визуализировать взаимодействие между системой и её окружением. Она описывает систему как один блок (чаще всего прямоугольник) и внешние сущности или компоненты, с которыми система взаимодействует. Контекстная диаграмма является высокоуровневым представлением и обычно используется на начальном этапе анализа системы. Контекстная диаграмма предоставляет участникам проекта и заинтересованным сторонам общее представление о системе и её месте в окружающей среде.

Контекстная диаграмма проекта (рисунок 3.3) состоит из требований необходимых веб-приложению для его разработки.

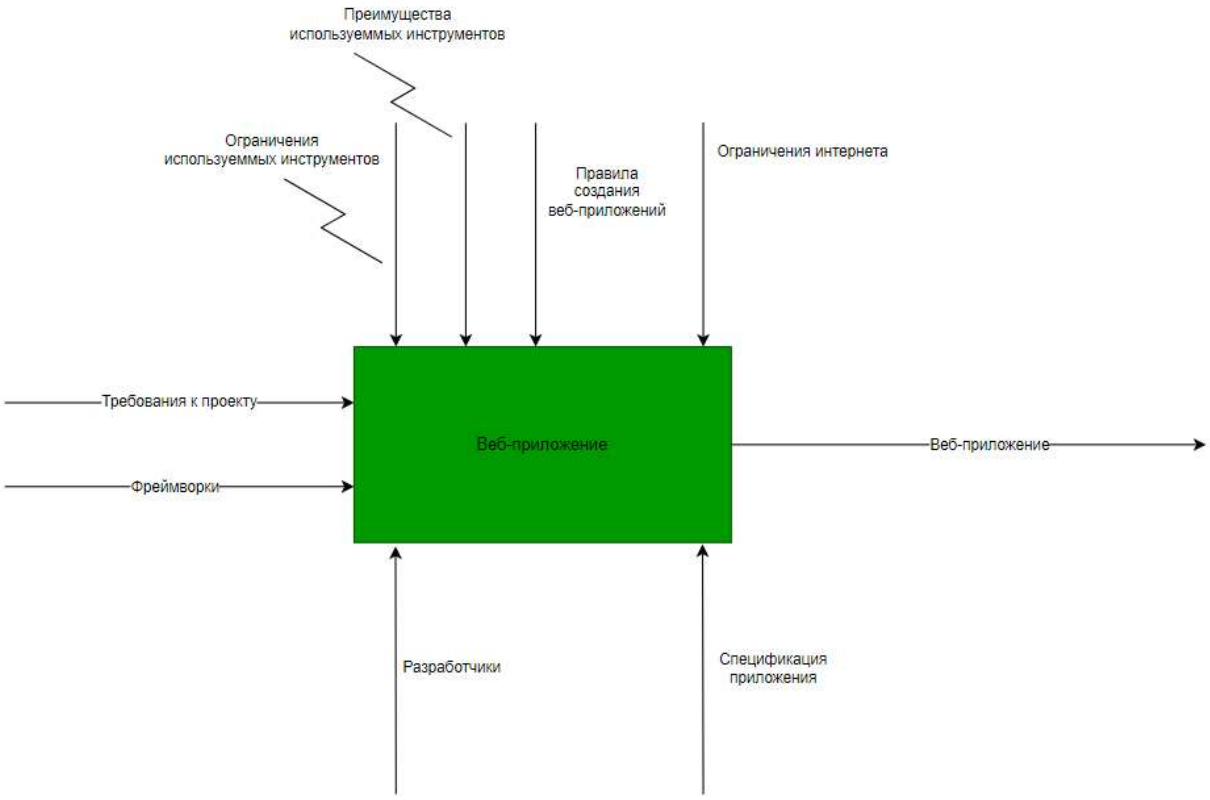


**Рисунок 3.3 – Контекстная диаграмма**

Входные стрелки:

– требования к проекту;

– фреймворки.

Управляющие воздействия:

– ограничения используеммых инструментов;

– преимузества используеммых инструментов;

– правила создания веб-приложений;

– ограничения интернета.

Механизмы:

– разработчики;

– спецификация приложения.

Выходные стрелки:

– веб-приложение.

Декомпозиция контекстной диаграммы - это процесс разбиения общего представления системы на более детализированные компоненты, уточняя внутреннюю структуру и взаимосвязи между её частями (рисунок 3.4). Этот процесс позволяет перейти от высокоуровневого представления системы к более подробному описанию её компонентов и взаимосвязей, что является важным этапом в анализе и проектировании системы. Вот несколько причин, почему декомпозиция контекстной диаграммы может быть полезной:

a) уточнение структуры системы: декомпозиция позволяет более детально определить внутренние компоненты и подсистемы системы, что важно для точного понимания её структуры;
b) более детальное проектирование: детализация контекстной диаграммы является этапом, предшествующим более подробному проектированию компонентов системы;
c) лучшее понимание требований: декомпозиция контекстной диаграммы помогает лучше понять требования к системе, учитывая их внутренний контекст и взаимосвязи;
d) определение зависимостей: процесс декомпозиции помогает выявить зависимости между компонентами системы, что важно для управления изменениями и понимания влияния изменений на другие части системы.

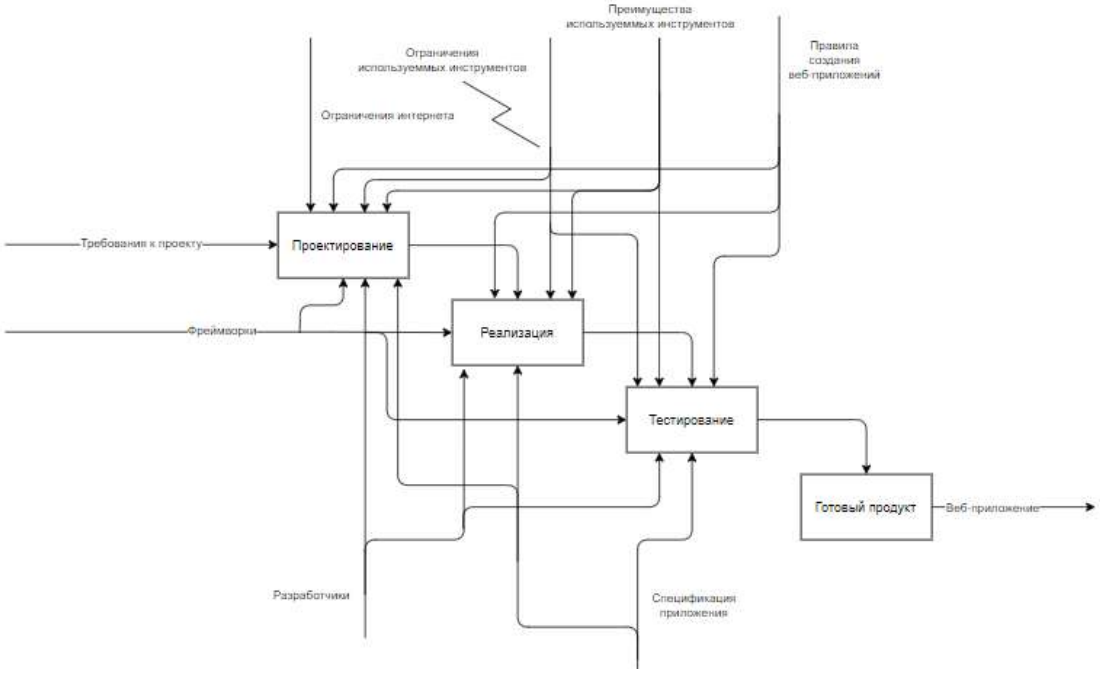


**Рисунок 3.4 – Декомпозиция контекстной диаграммы**

### 3.3 Функциональное назначение системы

Диаграммы вариантов использования предоставляют общее представление о том, как система взаимодействует с внешними акторами и какие функциональные возможности предоставляет. На диаграмме вариантов использования (рисунок 3.5) изображены действия, которые может выполнить пользователь в системе. Не зарегистрированный пользователь может только читать бесплатные книги, тот кто вошел в систему уже может оставлять комментарии, купить какое-то произведение.

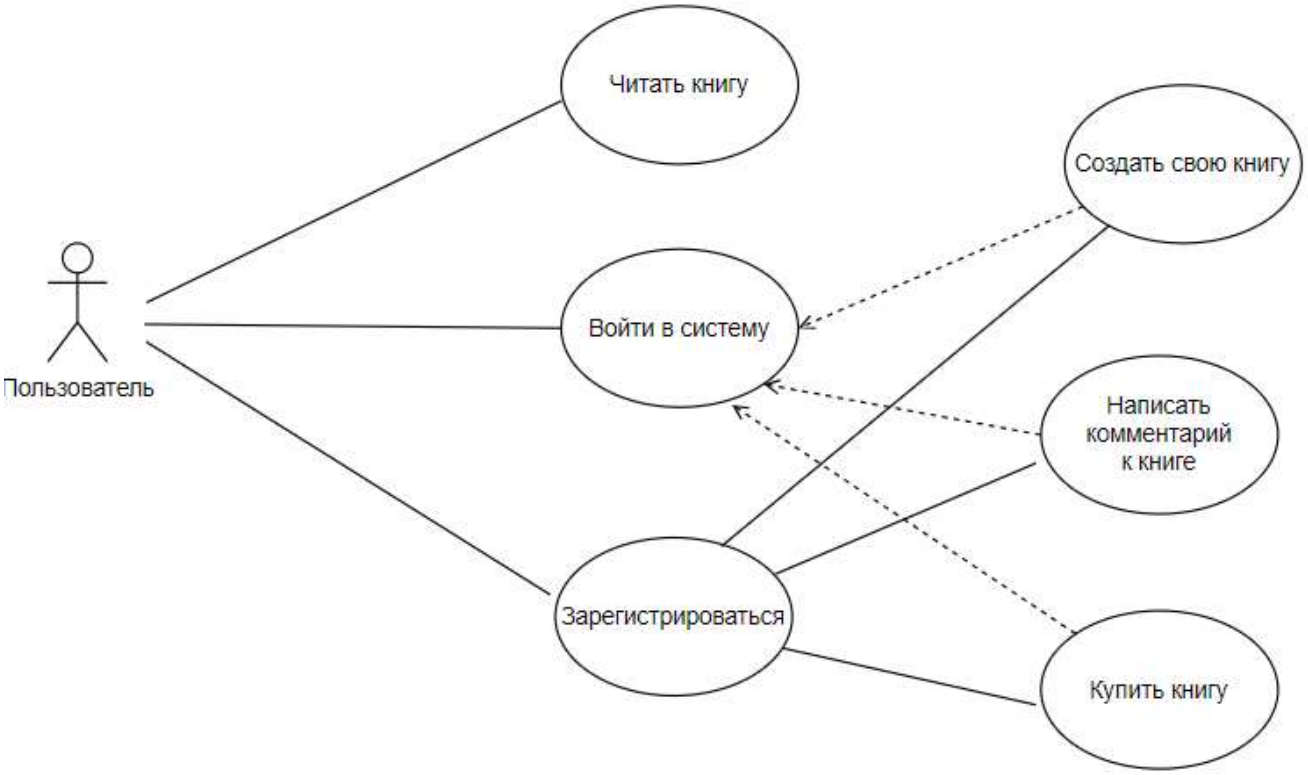


**Рисунок 3.5 – Диаграмма прецедентов для взаимодействия с системой**

### 3.4 Взаимодействие элементов системы

В информационном плане рассматривается взаимодействие между элементами, где коммуникация осуществляется обменом информацией между взаимодействующими объектами. Для моделирования такого взаимодействия в языке UML применяются соответствующие диаграммы взаимодействия. Диаграммы взаимодействия — это один из видов диаграмм в языке моделирования UML. Они используются для визуализации и описания взаимодействия между различными элементами системы, такими как объекты, компоненты и акторы.

На диаграмме последовательности (рисунок 3.6) показывается как происходит изменение счёта при соревновательной игре.

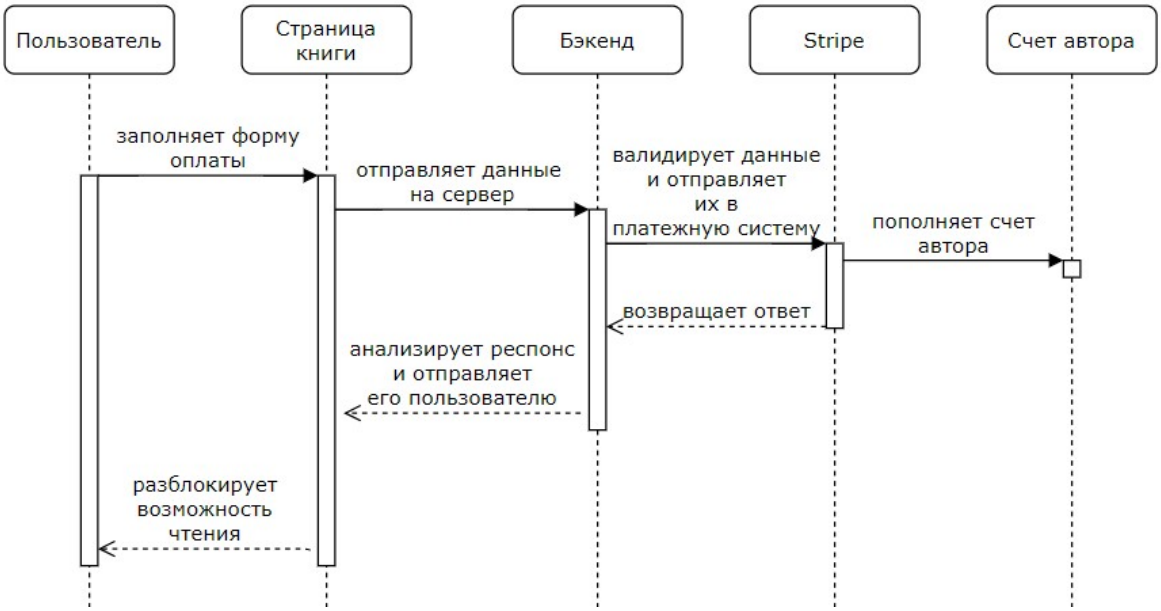


**Рисунок 3.6 – Диаграмма последовательности оплаты книги**

На рисунке 3.6 представлен процесс оплаты книги в веб-приложении. Сначала пользователь должен заполнить форму, где указывает данные своей карты. Эти данные отправляются на сервер, а потом в платежную систему Stripe, она их валидирует и пополняет счет автора. Далее Stripe предоставляет ответ серверу, который в свою очередь возвращает ответ на фроненд, где пользователю предоставляется возможность чтения.

На рисунке 3.7 представлен процесс начала чтения книги. Пользователь нажимает кнопку «Читать» потом его перенаправляет на страницу с текстовы содержанием книги. Но сначала нужно этот текст получить. Сначала фроненд делает запрос на бэкенд, который в свою очередь делает запрос в базу данных. От туда бэкенд достает уникальную ссылку на книгу и отправляет ее в хранилище Azure. Используя эту уникальную ссылку Azure, находит файл с книгой и отправляет ее на обратно серверу, а он фронтенду, который отображает текст пользователю.

На рисунке 3.8 представлена диаграмма кооперации для создания нового пользователя приложения. Сначала юзер должен заполнить форму на странице сайта, потом эти данные отправляются на сервер, там валидируются и после успешной валидации отправляются в базу данных. Потом ответ об успешной операции отправляется пользователю, где ему дается доступ к закрытой части приложения.

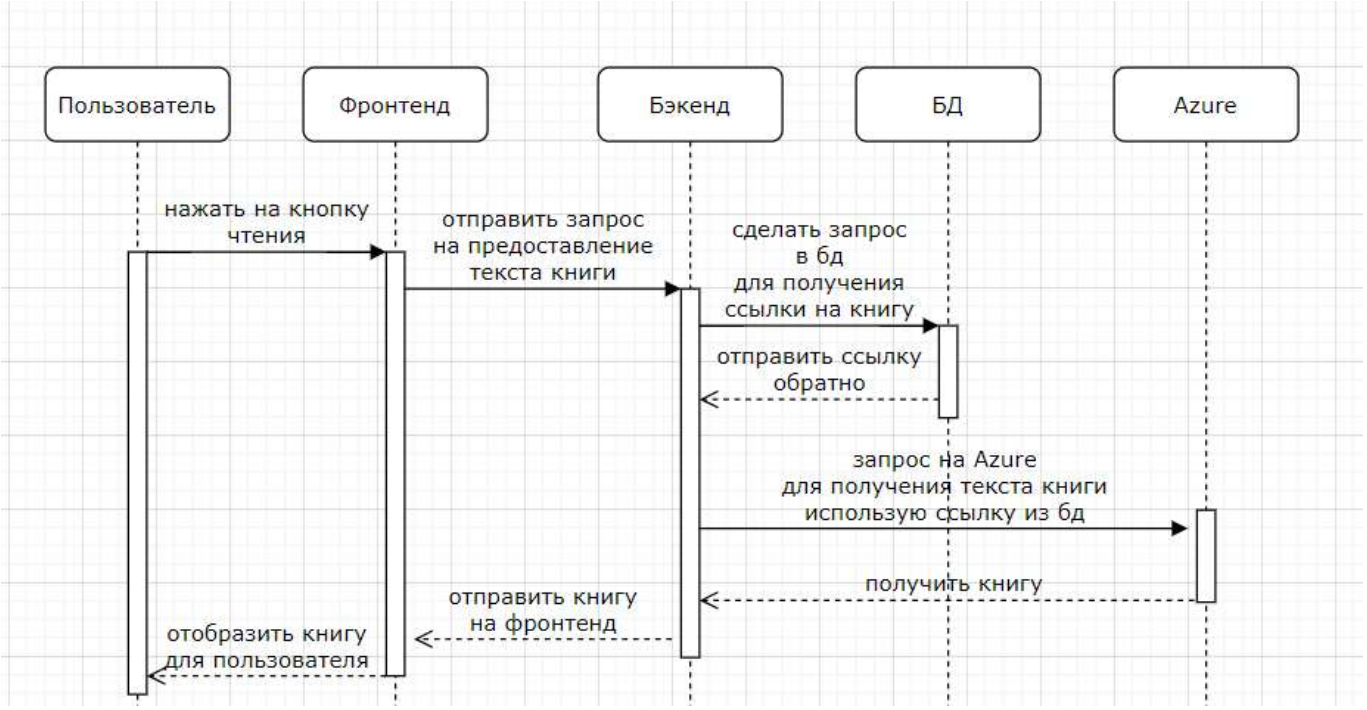


**Рисунок 3.7 – Диаграмма последовательности начала чтения книги**

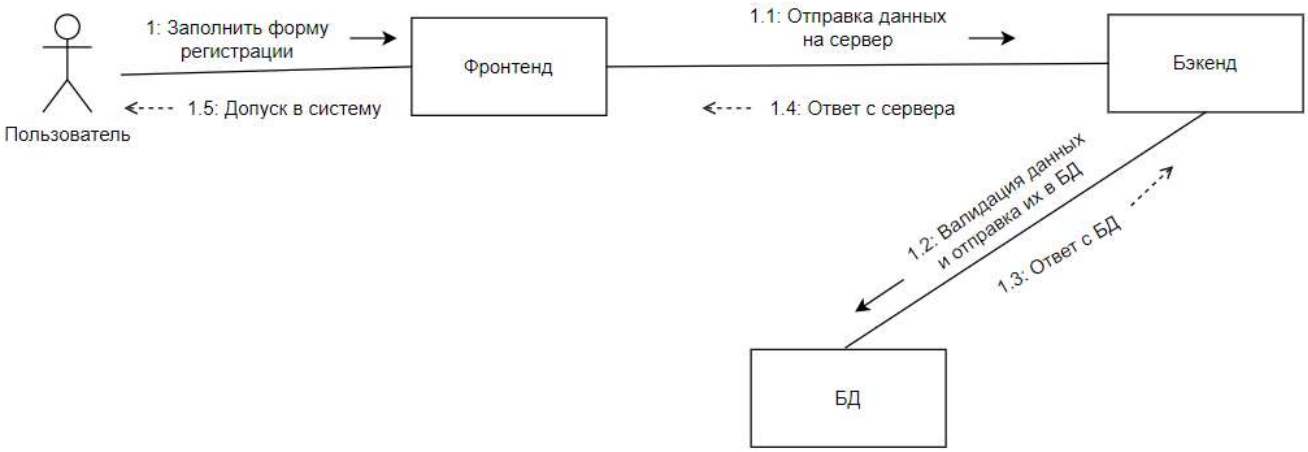


**Рисунок 3.8 – Диаграмма кооперации регистрации нового пользователя**

Диаграмма кооперации, представленная на рисунке 3.9 для создания нового комментария для книги в приложении. Сначала юзер должен заполнить поле ввода на странице книги, потом данные отправляются на сервер, там валидируются и после успешной валидации отправляются в базу данных. Потом ответ об успешной операции отправляется пользователю, где он уведомляется посредством пуш уведомления и сам комментарий отображается на странице.

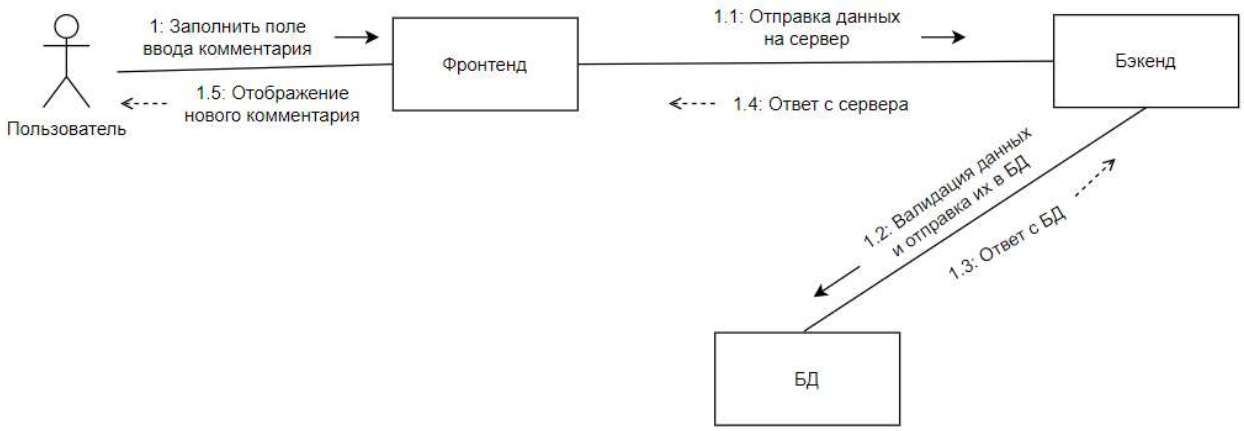


**Рисунок 3.9 – Диаграмма кооперации добавления нового комментария**

### 3.5 Статическая структура модели системы

Диаграмма классов (Class Diagram) - это один из ключевых видов диаграмм в языке моделирования UML. Диаграммы классов используются для визуализации структуры системы, описывая классы, их атрибуты, методы и отношения между ними. Эти диаграммы помогают моделировать структуру объектно-ориентированных систем и служат основой для разработки кода.

На рисунке 3.10 представлена диаграмма классов для начала чтения книги. Пользователь нажимает кнопку «Читать» на странице Book.cshtml. Отправляется запрос на бэкенд. Запрос попадает в BookController, потом данные отправляются в BookService, сервис делает запрос в базу данных для получения уникальной ссылки на файл с книгой через BookRepository. Потом сервис используя FileService (передавая ему уникальную ссылку на книгу) получает файл с Azure.

На рисунке 3.11 представлена диаграмма классов для создания нового аккаунта. Пользователь заполняет форму на странице SignIn.cshtml, запрос попадает в SignInController, потом данные отправляются в UserService и там валидируются, после валидации они отправляюся в бд с помощью UserRepository.

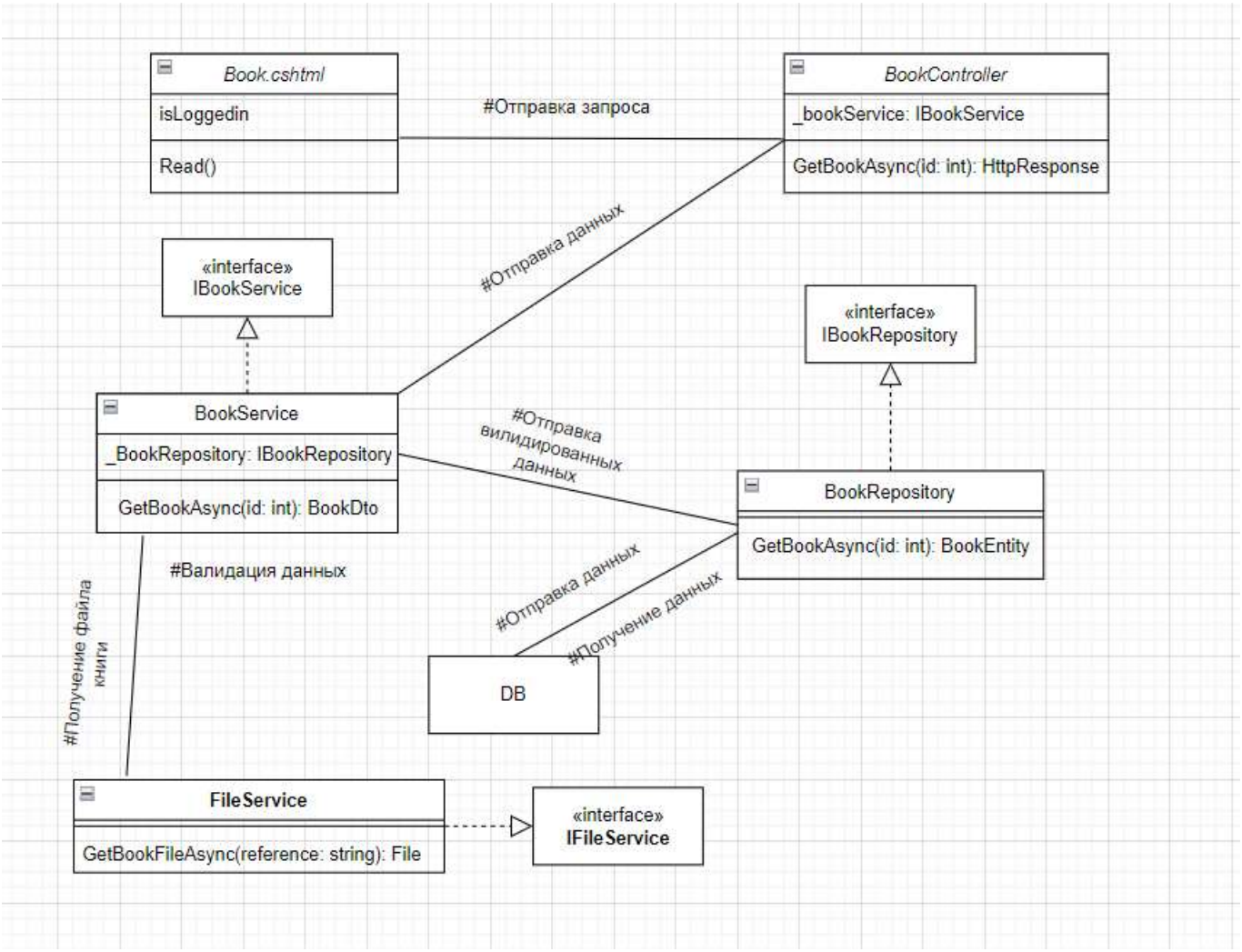


**Рисунок 3.10 – Диаграмма классов начала чтения книги**

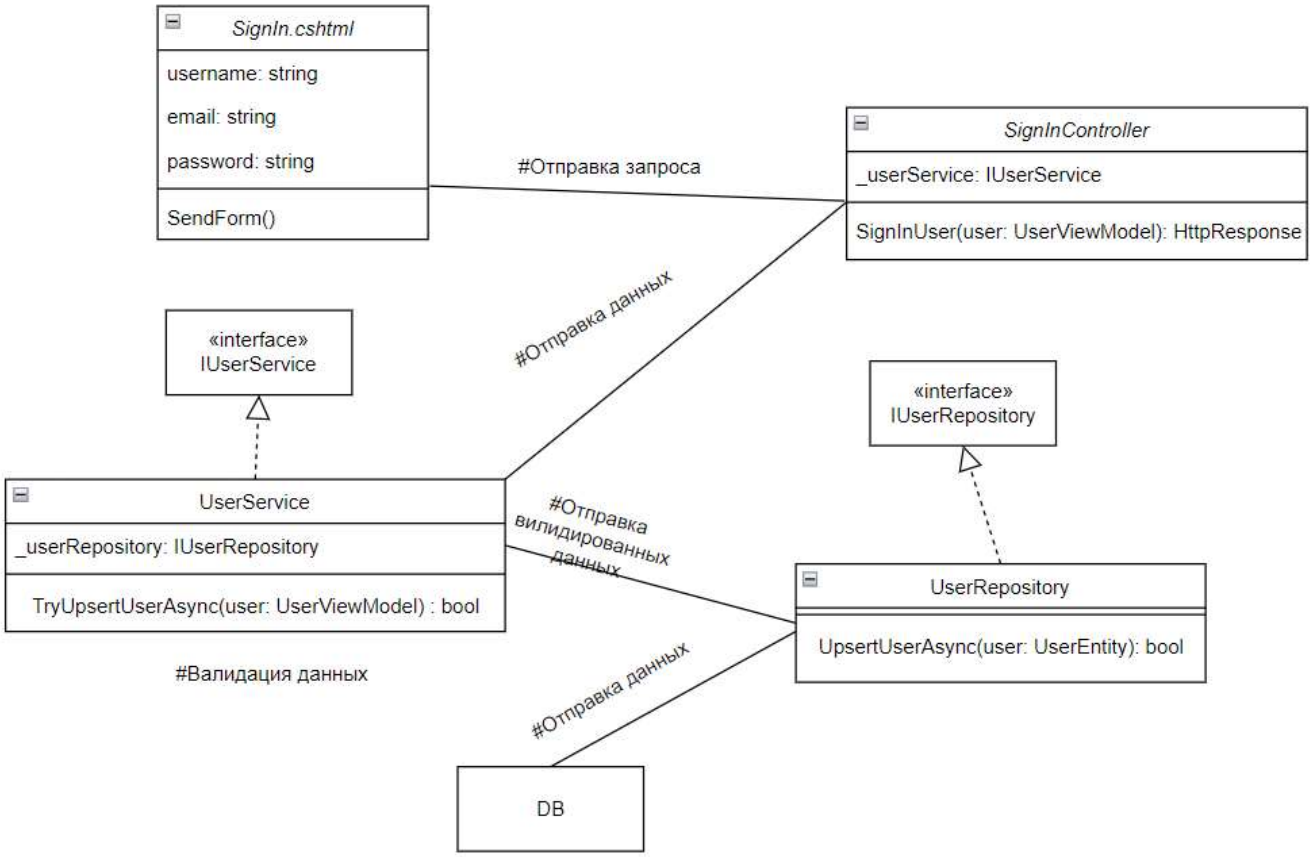


**Рисунок 3.11 – Диаграмма для классов создание нового аккаунта**

### 3.6 Моделирование поведения и процессов системы

Диаграммы состояний - это вид диаграмм в языке моделирования UML, который используется для визуализации и описания различных состояний, переходов между ними и событий, которые вызывают эти переходы в объекте или системе. Диаграммы состояний особенно полезны при моделировании поведения объектов, где важно представить их динамическое состояние. Диаграммы состояний являются мощным инструментом для моделирования и анализа динамического поведения объектов, а также обеспечивают важный вклад в разработку программных систем.

На рисунке 3.12 представлена диаграмма состояния. Для регистрации пользователь должен заполнить форму регистрации, потом она отправляется на сервер и там валидируется, если валидация проходит успешно, то пользователь вносится в базу данных, если нет, то ему высвечивает ошибку.

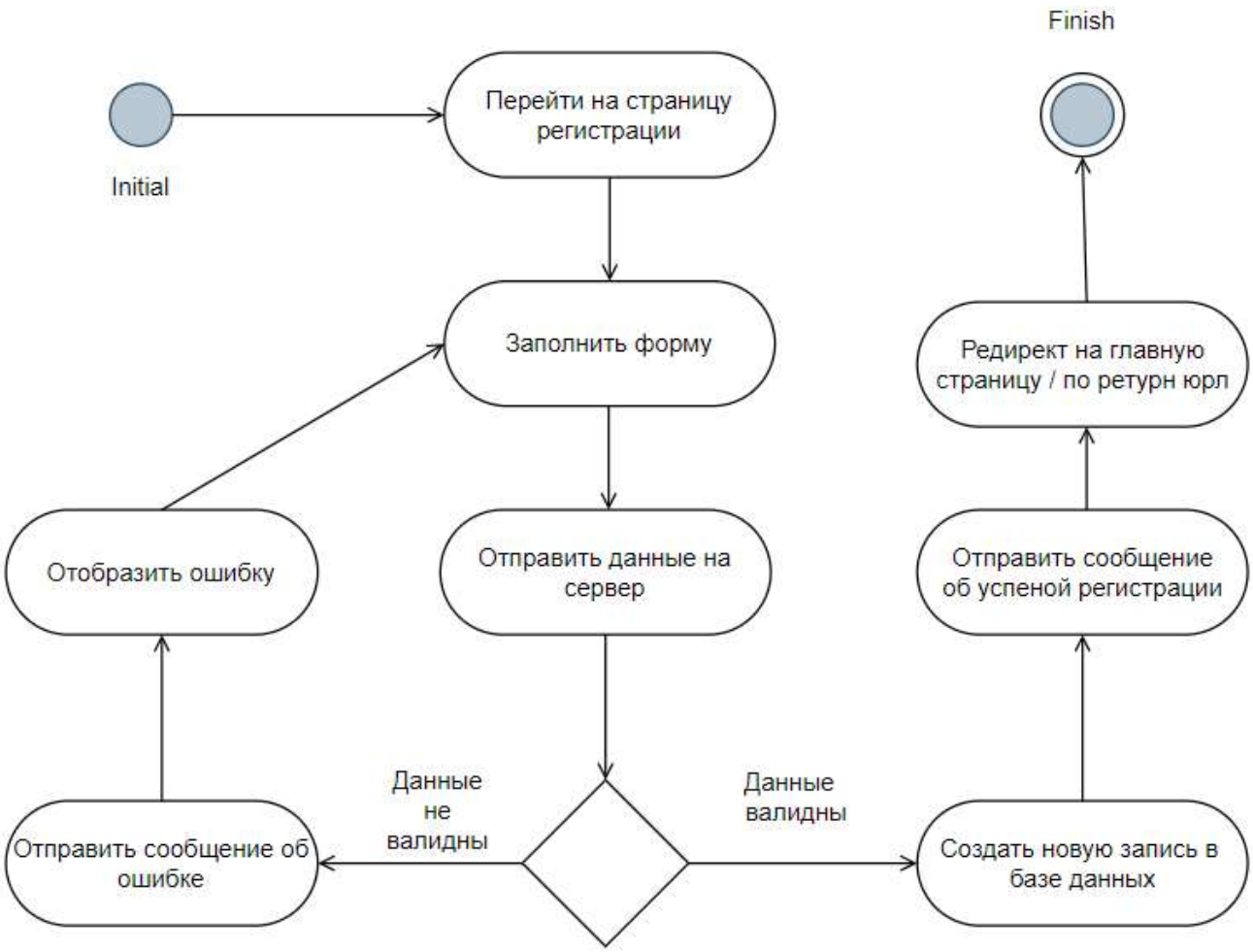


**Рисунок 3.12 – Диаграмма состояния для создания нового аккаунта**

Диаграмма состояния (рисунок 3.13) отображает процесс входа в аккаунт. Пользователь попадает на страницу авторизации, заполняет форму, потом она отправляется на сервер и там валидируется, если валидация проходит успешно, то созадется куки/токен авторизации, если нет, то ему высвечивает ошибку. Куки/токен сохраняется в браузере, чтобы дальнейшие запросы не нуждались в повторной авторизации.

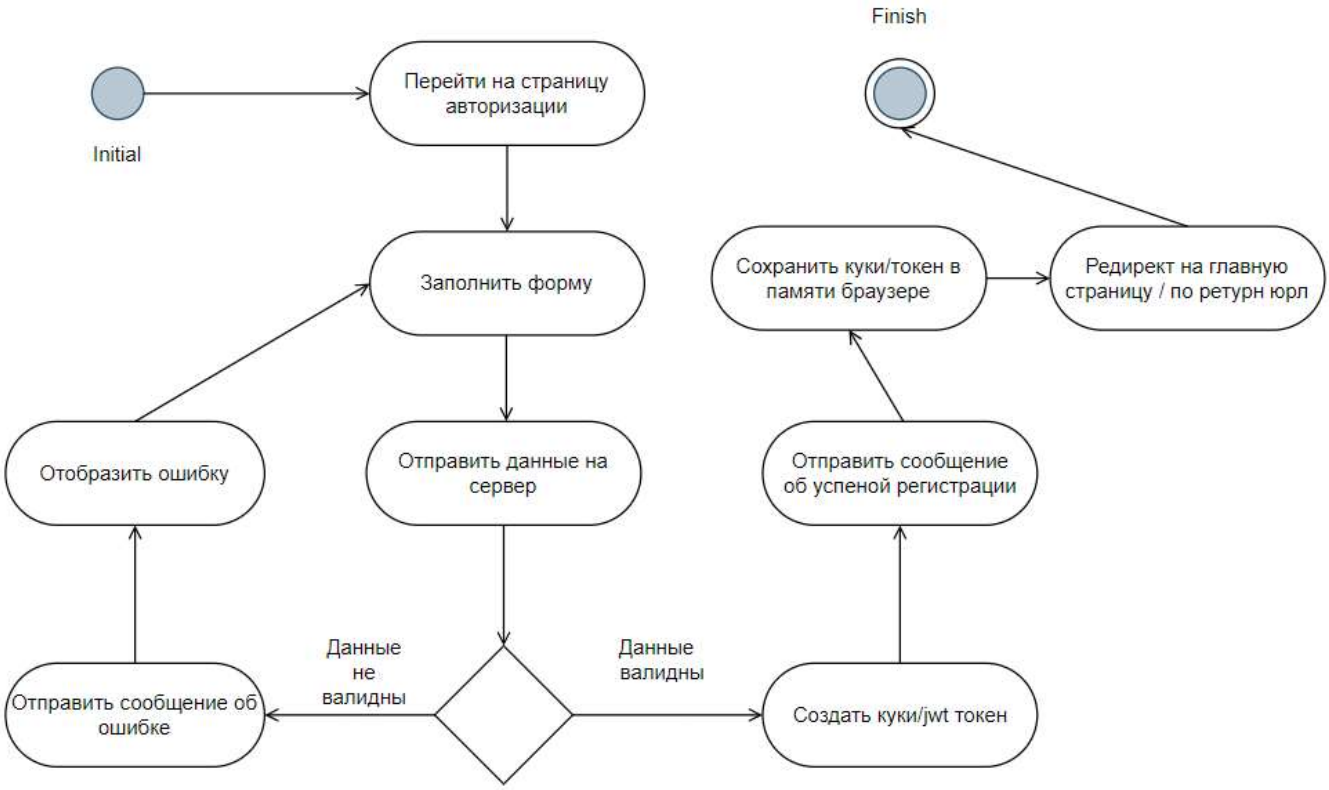


**Рисунок 3.13 – Диагрмма состояния для входа в аккаунт**

### 3.7 Моделирование физического представления системы

Диаграммы компонентов являются частью языка моделирования UML и используются для визуализации и описания высокоуровневой структуры системы, выделяя компоненты, их взаимосвязи и взаимодействие. Диаграммы компонентов часто применяются на этапе проектирования системы для представления её модульной архитектуры. Диаграммы компонентов являются важным инструментом в процессе проектирования и разработки программных систем, предоставляя архитекторам и разработчикам ясное представление о структуре системы.

На рисунке 3.14 представлена диаграмма компонентов страницы книги. По верхнему меню (нав бар) пользователь может открывать другие отделы веб-приложения. Это стандартная часть интерфейса для любой страницы приложения (кроме страницы регистрации/авторизации). Пользователь может посмотреть краткую информацию об авторе, почитать коментарии, посмотреть аннотацию к книге, ее жанр, тэги, чтобы понять, стоит ли ее читать или покупать.

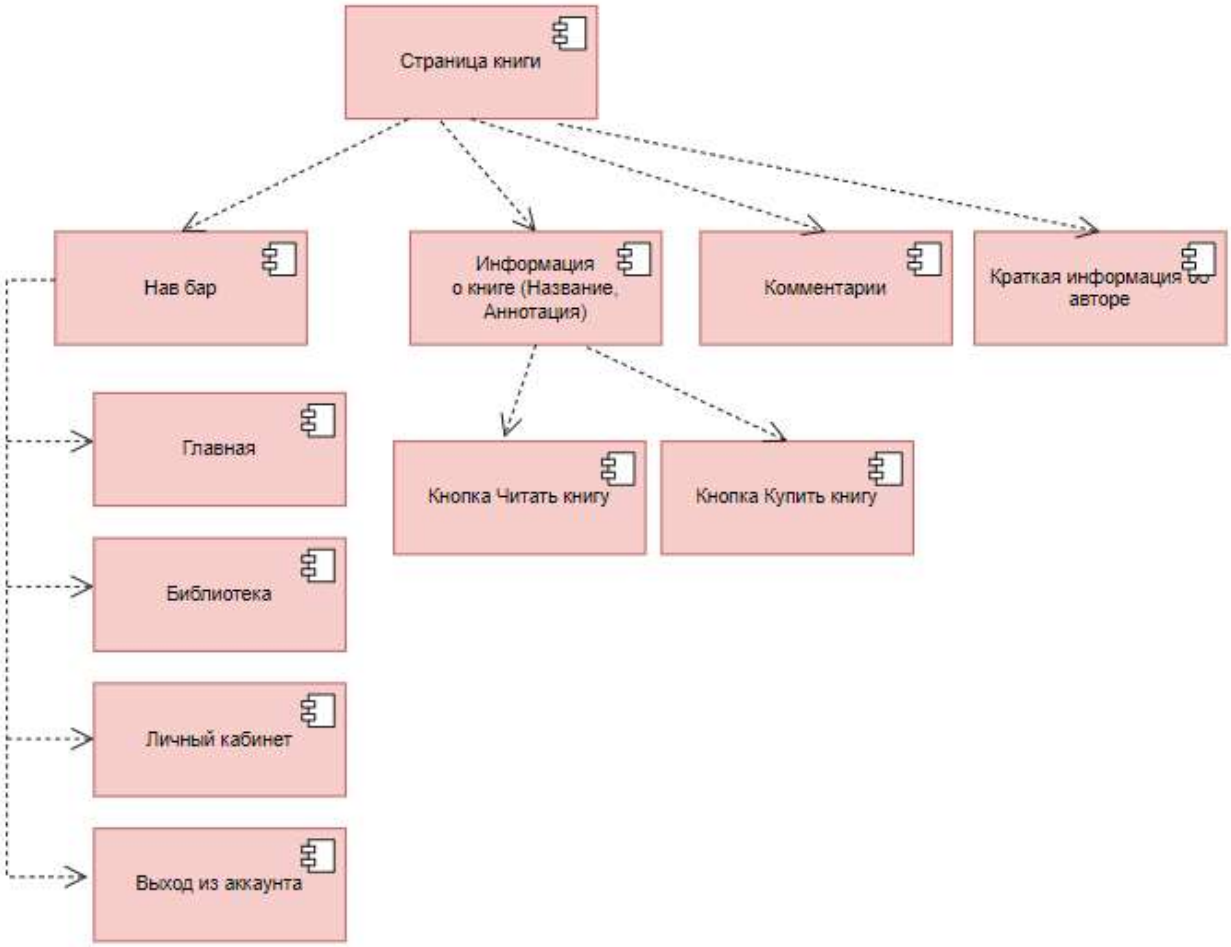


**Рисунок 3.14 – Диаграмма компонентов меню**

Диаграмма компонентов регистрации/авторизации (рисунок. 3.15) предоставляет описание интерфейса регистрации/авторизации. Пользователь может переключаться между авторизацией и регистрацией, используя вкладки над формами (рисунок. 3.16), это позволяет использовать одну страницу для двух действий.

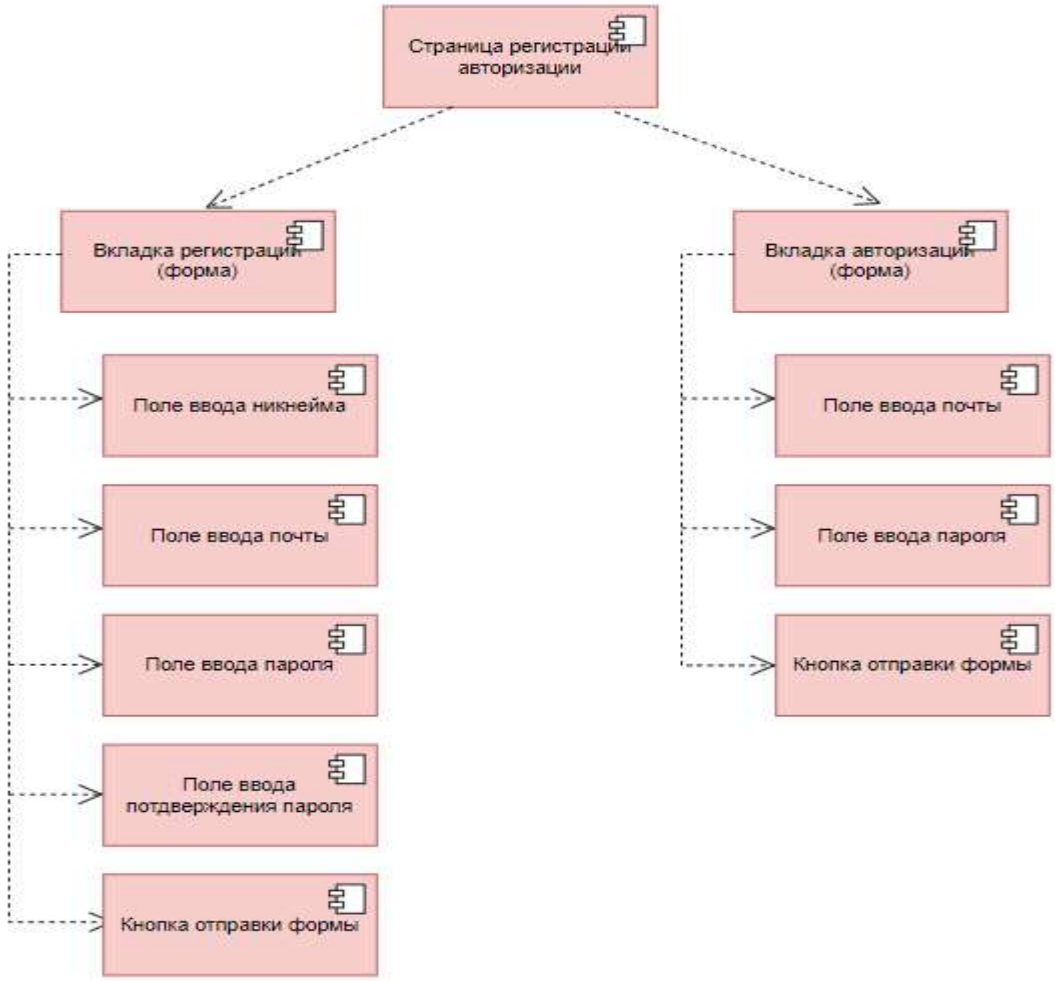


**Рисунок 3.15 – Диаграмма компонентов регистрации/авторизации**

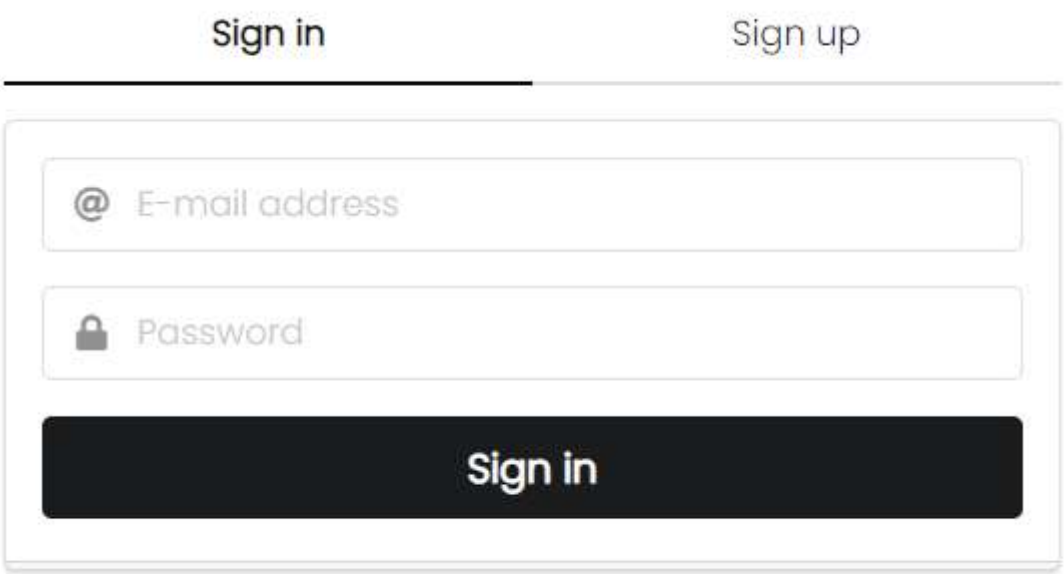


**Рисунок 3.16 – Форма авторизации/регистрации**

# 4 ОЦЕНКА СТОИМОСТИ

Work Breakdown Structure (WBS) — это методология структурирования и организации проекта на более управляемые и управляемые компоненты. WBS представляет собой древовидную декомпозицию проекта на более мелкие и управляемые элементы, что облегчает понимание общего объема работы и планирование ресурсов. Каждый уровень WBS представляет собой более детализированный уровень задач.

## 4.1 Декомпозиция разрабатываемой системы

Разработка практического любого веб-приложения (мой проект не исключение) состоит из 4-х основных этапов (рисунок 4.1):

- разработка фроненда: разработка пользовательской части приложения, например UI (визуальная часть приложения);
- разработка бэкенда: разработка серверной части приложение, которое обеспечивает прием запросов с фроненда, их обработки и ответ. Отвечает за взаимодействие с базой данных и сторонними сервисами;
- проектирование базы данных: является одним из важнейших аспектов разработки приложения, хорошо спроектированная база данных влияет на производительность всего приложения, а на это влияет оптимизацию запросов, оптимизация sql, правильные индексы, правильная иерархия таблиц;
- интеграция сторонних сервисов и библиотек: важный этап, который описывает как приложение взаимодействует с другими приложениями, например Azure DevOps, Microsoft Entra ID.

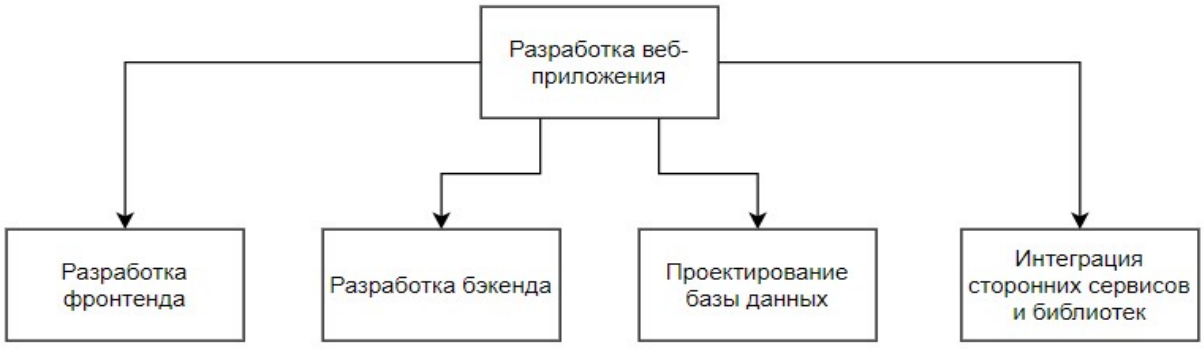


**Рисунок 4.1 – Основные этапы разработки веб-приложения**

Ниже представлена диаграмма декомпозиции проекта на иерархию (рисунок 4.2), где есть подготовка (проектирвоание) проекта, проектирование базы данных, разработка проекта и его тестирование с последующим запуском.

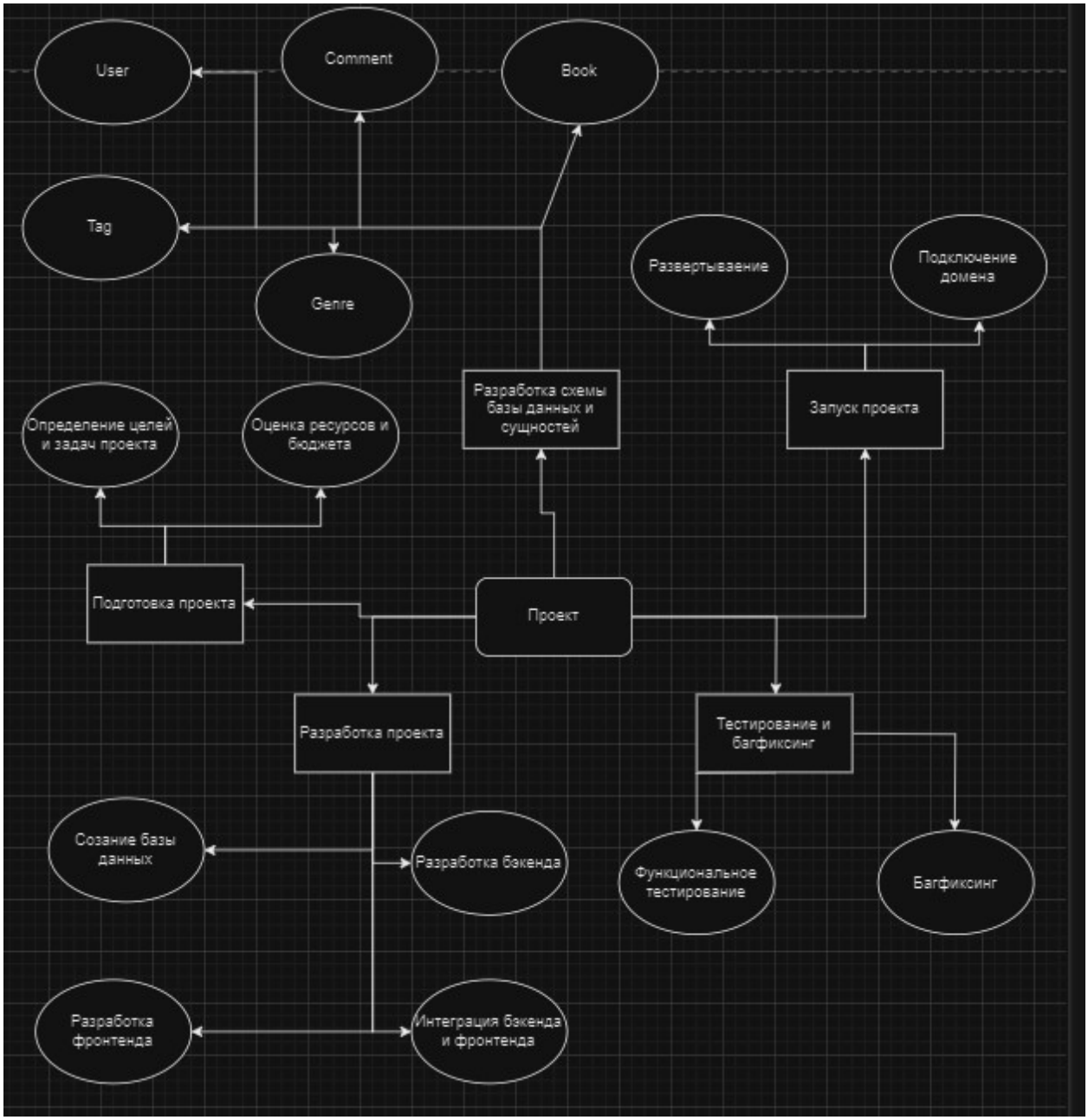


**Рисунок 4.2 – WBS (Work Breakdown Structure) диаграмма декомпозиции проекта на иерархию**

### 4.2 Планирование реализации требований

На данном этапе определяем временные рамки для каждого этапа разработки. Выделяем такие ресурсы как разработчики и дизайнеры, а также проводим тестирование нашего продукта. Ниже, на рисунке 4.3, представлена диаграмма Ганта по WBS, на которой каждому этапу выделили определённое время на его реализацию.


**Рисунок 4.3 – Диаграмма Ганта по WBS**

### 4.3 Оценка стоимости проекта

Определение стоимости проекта — это процесс оценки и вычисления всех расходов, необходимых для выполнения проекта. Этот процесс включает в себя анализ всех затрат, включая трудозатраты, материальные ресурсы, оборудование, программное обеспечение, внешние услуги и другие расходы, связанные с реализацией проекта. Цель определения стоимости проекта заключается в создании бюджета, который будет использоваться для управления финансами, контроля затрат и обеспечения финансовой устойчивости проекта.

**Расчет**

Для каждого этапа разработки нужно подсчитать дни и часы, затраченные одним работником. В среднем один рабочий день длится 8 часов. Оценим один час работы в $10.

- подготовка проекта: 7 дней × 8 часов/день × $10/час = $560;
- разработка базы данных и сущностей: 21 дней × 8 часов/день × $10/час = $1680;
- разработка проекта: 26 день × 8 часов/день × $10/час = $2080;
- тестирование и багфиксинг: 12 дней × 8 часов/день × $10/час = $960;
- запуск проекта: 5 дней × 8 часов/день × $10/час = $400.

Общая стоимость за разработку функциональности составляет:

$$О = П + Р1 + Р2 + Т + З, \quad (4.1)$$

где

О – общая стоимость зарплат;

П – проектирование;

Р1 – разработка базы данных;

Р2 – реализация самого проекта;

Т – тестирование;

З – запуск проекта.

О = $560 + $2080 + $1680 + $960 + $400 = $5680.

**Оценочную стоимость проекта можно вычислить по формуле:**

C = S + Э + E + Eprob,

где

C – оценочная стоимость;

S – затраты на заработную плату за разработку функциональности и дизайна;

E – лицензии или оборудование;

Eprob – затраты на дополнительные расходы.

**Затраты на электричество**

Для определения окончательной стоимости проекта необходимо учесть не только расходы на оплату труда сотрудников, но также затраты на электроэнергию, особенно учитывая, что в рамках данного проекта, связанного с разработкой программного продукта, существуют затраты на рабочие ноутбуки, потребляющие значительное количество электроэнергии. При оценке стоимости проекта следует учесть эти факторы. Для расчета затрат на электроэнергию важно учитывать следующие аспекты:

- потребление энергии каждым устройством, задействованным в проекте;
- время, в течение которого каждое устройство будет использоваться в рамках проекта;
- стоимость потребленной электроэнергии на местном энергорынке.

Для определения затрат на электроэнергию в рамках данного проекта необходимо выполнить следующие шаги:

- составить список всех используемых в проекте устройств и указать их потребление электроэнергии в ваттах (W) или киловатт-часах (kWh);
- определить общее время использования каждого устройства в рамках проекта;
- установить стоимость потребленной электроэнергии на местном рынке, уточнив цену за киловатт-час или за тарифную единицу.

Путем анализа этих данных можно определить примерные затраты на электроэнергию и включить их в общую стоимость проекта.

Затраты на электроэнергию определяются с учётом тарифа в Бельцах – 4 лея за 1 КВт*ч и по формуле (4.2):

$$q = W * t, \quad (4.2)$$

где

q – расчёт затраченной электроэнергии за все время;

W – мощность одного компьютера, кВт;

t – время работы в часах.

t = 568 часов,

q = W *t = 0,60 кВт * 568 часа = 346,48 кВт*час,

$$Э = Т * q, \quad (4.3)$$

где

Э – затраты на электроэнергию;

Т – тариф на электроэнергию.

На основе полученной информации можно подсчитать расходы в молдавских леях на электроэнергию, которая была затрачена в процессе разработки проекта. Используя формулу (4.3):

Э = Т * q = 4 * 346,48 = 1385,92 (лея).

Для разработки дипломного проекта затраты по электроэнергии составляют 1385,92 лей.

**Оценка дополнительных расходов**

Здесь учитываются различные дополнительные расходы, которые могут возникнуть в результате непредвиденных обстоятельств, например, поломка компьютера:

- нижняя оценка: 0% от затрат на оплату труда;
- наиболее вероятная оценка: 10% от затрат на оплату труда;
- верхняя оценка: 25% от затрат на оплату труда.

Оценка дополнительных расходов вычисляется по формуле (4.4):

$$Eprob = 0 * A + * 0,1 * A + 0,25 * A, \quad (4.4)$$

где

А – затраты, которые включаю в себя затраты на заработную плату сотрудников, лицензии и электроэнергию. Так как в разработке проекта использовались только бесплатные инструменты, библиотеки и программы, то затраты на лицензию составляют 0 лей. Так же, в разработке проекта не нужны компоненты и материалы, поэтому Е = 0 лей.

А = S + Э + Е;

А = 102240 + 1385,92 + 0 = 103625,92 (лей).

Подставляя полученные затраты (А) в формулу (4.4), можно рассчитать стоимость дополнительных расходов:

Eprob = (0% * 103625,92 + 10% * 103625,92 + 25% * 103625,92):100% = 36268,98 (лей).

Учитывая все перечисленные затраты, такие как заработные платы сотрудников, расходы на электроэнергию, затраты на лицензии для создания программного продукта, а также неожиданные расходы, окончательная оценочная стоимость проекта может быть рассчитана с использованием формулы (1), где происходит суммирование всех учтенных компонентов.

С = 102240 + 1385,92  + 0 + 36268,98 = 139894,93 (лей).

Таким образом, окончательная стоимость данной логической мобильной игры оценивается в 139894,93  лей.

# ЗАКЛЮЧЕНИЕ

Разработка веб-приложения на языке программирования C# для чтения книг представляет собой результат комплексного инженерного подхода. Реализованный проект предоставляет пользователю удобный интерфейс для доступа к литературным произведениям, обеспечивая при этом эффективное взаимодействие с контентом. Отдельное внимание уделено анализу требований и проектированию системы, что позволило создать структурированное и масштабируемое веб-приложение.

Процесс разработки включал в себя эффективное использование технологий и принципов языка C#, что способствовало высокой производительности и надежности приложения. Результаты тестирования подтвердили соответствие функциональности приложения заявленным требованиям, а внедрение современных практик безопасности обеспечивает устойчивость к возможным угрозам.

Обнаруженные в процессе разработки проблемы были тщательно исследованы и приняты меры по их решению. Отмечается, что разработанное веб-приложение предоставляет пользователю интуитивно понятный интерфейс, а использование технологии C# дает возможность эффективного масштабирования и дальнейшего развития системы.

В целом, создание веб-приложения на C# для чтения книг подчеркивает важность современных методологий разработки и применения передовых технологий для достижения высокого уровня функциональности и удовлетворения потребностей пользователей. Полученные результаты могут служить основой для дальнейших исследований в области веб-разработки и оптимизации пользовательского опыта.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ, Описание предметной области, Обзор существующих программных продуктов, Выбор и обоснование варианта реализации задач. Студенческая библиотека онлайн, ©2023.
   Режим доступа: https://studbooks.net/2247494/informatika/analiz_predmetnoy_oblasti
2. Microsoft Documentation, Microsoft. ASP.NET Core Documentation.
   Режим доступ: https://docs.microsoft.com/en-us/aspnet/core/
3. Metanit. ASP.NET, Руководство по ASP.NET Core 8.
   Режим доступ: https://metanit.com/sharp/aspnet6/
4. Metanit. Полное руководство по языку программирования C# 12 и платформе .NET 8
   Режим доступ: https://metanit.com/sharp/tutorial/
5. БЕШЛИУ, В., КИРЕВ, П., СКОРОХОДОВА, Т. Методическое руководство по реализации лабораторных работ по курсу: Проектирование информационных систем (инструменты, руководства и методологические вопросы). Кишинёв, 2023. с.2-73.