ОБЛАСТНОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

«КУРСКИЙ ТЕХНИКУМ СВЯЗИ»

|  |  |
| --- | --- |
| Рецензент  *ОАО «Курскрезинатехника», начальник управления информационных технологий*  / *В. И. Горелухин*  « » 2023 г | Допущен к защите Зам. директора по учебно- производственной работе  / *В.В. Малинников*  « » 2024 г |

Специальность: 09.02.03 Программирование в компьютерных системах

*(шифр) (наименование специальности)*

Форма обучения: *очная/заочная*

Дипломная работа на тему

Разработка web-приложения для просмотра фильмов и чтения книг

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  |  |  |
|  | (подпись) |  | Фамилия И.О.  « »\_\_\_\_\_\_\_\_\_ 20\_\_г. |
| Руководитель дипломной  работы |  |  | Рыжков В. В. |
|  | (подпись) |  | Фамилия И.О.  « »\_\_\_\_\_\_\_\_\_ 20\_\_г. |
| Нормоконтроль дипломной работы |  |  | Котов С. С. |
|  | (подпись) |  | Фамилия И.О  « »\_\_\_\_\_\_\_\_\_ 20\_\_г. |

Курск, 2024г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc168441932)

[ГЛАВА 1: ПРЕДПРОЕКТНОЕ ОБСЛЕДОВАНИЕ 6](#_Toc168441933)

[1.1 Исследование предметной области 6](#_Toc168441934)

[1.2 Формирование комплекса требований к программному продукту 7](#_Toc168441935)

[ГЛАВА 2 ТЕХНИЧЕСКОЕ ЗАДАНИЕ 8](#_Toc168441936)

[2.1 Постановка задачи 8](#_Toc168441937)

[2.2 Концепция информационной базы и её структура 10](#_Toc168441938)

[2.3 Неформальное описание алгоритма работы ПП 16](#_Toc168441939)

[2.4 Функции подсистем (модулей) 17](#_Toc168441940)

[2.5 Оценка экономической эффективности от внедрения программы 23](#_Toc168441941)

[ГЛАВА 3 ЭСКИЗНЫЙ ПРОЕКТ 28](#_Toc168441942)

[3.1 Разработка эскиза проекта 28](#_Toc168441943)

[3.2 Верстка интерфейса 31](#_Toc168441944)

[ГЛАВА 4 РАЗРАБОТКА ПРОЕКТНЫХ РЕШЕНИЙ 33](#_Toc168441945)

[4.1 Разработка API 33](#_Toc168441946)

[4.2 Разработка интерфейса 37](#_Toc168441947)

[ЗАКЛЮЧЕНИЕ 41](#_Toc168441948)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 43](#_Toc168441949)

[ПРИЛОЖЕНИЕ А 44](#_Toc168441950)

[ПРИЛОЖЕНИЕ Б 45](#_Toc168441951)

[ПРИЛОЖЕНИЕ В 46](#_Toc168441952)

[ПРИЛОЖЕНИЕ Г 47](#_Toc168441953)

# ВВЕДЕНИЕ

В наше время цифровые технологии играют ключевую роль в удовлетворении повседневных потребностей, таких как получение информации и развлечение. Большую популярность приобретают web-приложения, позволяющие пользователям просматривать фильмы и читать книги онлайн. Эти приложения становятся неотъемлемой частью цифрового пространства благодаря своей удобности, доступности и широкому функционалу. Разработка web-приложения для просмотра фильмов и чтения книг отвечает современным запросам пользователей, стремящихся к быстрому и качественному доступу к контенту.

Цель разрабатываемого проекта:

Создать универсальную платформу, предоставляющую пользователям доступ к разнообразному ассортименту фильмов и книг. Приложение будет обладать интуитивно понятным интерфейсом, позволяющим легко находить и просматривать интересующий контент, а также персонализированными функциями, такими как создание списков избранного, рекомендации на основе предпочтений пользователя, и возможность оставлять отзывы и оценки.

Обеспечить пользователям удобный и функциональный инструмент для потребления цифрового контента, который может использоваться как индивидуальными пользователями, так и коммерческими организациями.

Для достижения цели необходимо выполнить следующие задачи:

* Определить требования к разрабатываемому приложению на основе потребностей пользователей.
* Спроектировать архитектуру приложения, учитывая его масштабируемость и надежность.
* Реализовать ключевые функциональные модули, включая систему поиска, рекомендаций и персонализации.
* Провести тестирование приложения для выявления и устранения возможных ошибок.

Актуальность выбранной темы:

Актуальность разработки обусловлена несколькими ключевыми факторами. Во-первых, растет спрос на комплексные решения, объединяющие различные виды мультимедийного контента. Пользователи предпочитают иметь доступ к фильмам и книгам в рамках одной платформы, что экономит время и усилия на использование различных сервисов. Во-вторых, в условиях пандемии и вынужденной самоизоляции цифровые развлечения стали еще более востребованными, подчеркивая необходимость создания качественных и удобных web-приложений.

Практическая значимость: Практическая значимость проекта заключается в создании удобного и функционального инструмента для потребителей цифрового контента. Приложение может использоваться как индивидуальными пользователями, так и коммерческими организациями, предоставляющими услуги по распространению мультимедийного контента. Это способствует повышению удовлетворенности пользователей и увеличению числа клиентов.

Основные отличия от подобных программ: Ключевое отличие разработанного приложения – его универсальность. В отличие от многих существующих решений, оно объединяет функции просмотра фильмов и чтения книг, позволяя пользователям использовать одну платформу для удовлетворения различных потребностей.

Описание предметной области: Предметная область разработки включает цифровое распространение мультимедийного контента, такого как фильмы и книги. Эта область характеризуется высоким уровнем конкуренции и быстрым развитием технологий. Основными пользователями являются люди всех возрастов, активно использующие интернет для получения развлечений и информации.

Информационная база и средства разработки:

Все данные, включая фильмы и книги, хранятся на сервере, что обеспечивает высокую скорость доступа и надежность работы приложения. В качестве средств разработки для backend использовались технологии Asp.Net Core и .Net 8.0 с использованием языка программирования C#. Для управления данными применена ORM-библиотека EntityFramework Core, а база данных PostgreSQL служит для хранения всей необходимой информации.

Frontend-разработка велась с использованием HTML и CSS для создания структуры и стилей приложения, а также TypeScript для написания логики клиентской части. В качестве фреймворков использовались React для построения пользовательского интерфейса, Axios для обработки HTTP-запросов, и Redux RTK для управления состоянием приложения. Эти технологии обеспечивают высокую производительность, отзывчивость и удобство использования приложения.

Условия функционирования:

Приложение разработано для работы в веб-браузерах, что делает его доступным для пользователей на различных устройствах . Для обеспечения стабильной работы и высокой скорости загрузки контента использовались серверные решения, обеспечивающие надежное хранение и быструю доставку данных пользователям.

Таким образом, разработка web-приложения для просмотра фильмов и чтения книг является актуальной задачей, решающей многие проблемы современных пользователей и предоставляющей им удобный и универсальный инструмент для потребления цифрового контента.

# ГЛАВА 1: ПРЕДПРОЕКТНОЕ ОБСЛЕДОВАНИЕ

## 1.1 Исследование предметной области

При разработке web-приложения для просмотра фильмов и чтения книг важно учитывать особенности данной сферы и потребности пользователей. Исследование предметной области включает описание основных понятий и функциональной модели, а также анализ связей между функциями и моделирование потоков данных.

Основные понятия и функциональная модель

В разработке web-приложения для просмотра фильмов и чтения книг ключевыми объектами являются:

* Пользователь: конечный пользователь приложения, который ищет разнообразный контент для просмотра и чтения.
* Фильмы и книги: основной контент, который предоставляется пользователю для просмотра и чтения.
* Интерфейс: пользовательский интерфейс приложения, который обеспечивает удобство взаимодействия с контентом.

Связи между функциями приложения представлены следующим образом:

1. Пользовательское взаимодействие с интерфейсом приложения.
2. Запросы на сервер для получения информации о фильмах и книгах.
3. Отображение полученной информации пользователю.
4. Возможность выбора фильма или книги для просмотра или чтения.
5. Поток данных с сервера к клиенту и обратно для обеспечения непрерывности работы приложения.

Использованные источники

Для проведения исследования предметной области были использованы различные источники, включая:

1. Специализированные литературные и онлайн-ресурсы, посвященные веб-разработке и мультимедийному контенту.
2. Аналогичные web-приложения для просмотра фильмов и чтения книг с целью изучения их функционала и пользовательского опыта.

## 1.2 Формирование комплекса требований к программному продукту

При формировании требований к web-приложению учитываются как функциональные, так и нефункциональные аспекты, а также требования на различных этапах разработки.

Функциональные требования

1. Поиск и фильтрация контента: Пользователь должен иметь возможность осуществлять поиск фильмов и книг.
2. Просмотр фильмов и чтение книг: Пользователь должен иметь возможность просматривать фильмы и читать книги непосредственно в приложении.
3. Персонализация: Предоставление пользователю возможности создания профиля, добавления контента в избранное.
4. Управление контентом: Администратору приложения должна быть предоставлена возможность добавления, удаления и редактирования фильмов и книг и прочих второстепенных данных.

Нефункциональные требования

1. Интерфейс: Приложение должно иметь интуитивно понятный и привлекательный пользовательский интерфейс.
2. Производительность: Приложение должно обеспечивать быструю загрузку контента и отзывчивость интерфейса.
3. Надежность и безопасность: Приложение должно быть защищено от угроз безопасности и обеспечивать сохранность данных пользователей.
4. Совместимость: Приложение должно корректно функционировать на различных устройствах и в различных веб-браузерах.

Использование общероссийских и других классификаторов

Для обеспечения стандартизации данных и удобства использования, приложение будет использовать классификаторы жанров и категорий фильмов и книг, а также соответствовать общепринятым стандартам безопасности данных и веб-разработки.

Организация данных и информационный обмен

Данные о фильмах и книгах будут храниться на сервере приложения, а обмен информацией между клиентом и сервером будет осуществляться посредством API запросов. Для хранения информации о пользователях и их действиях будет использована база данных с соблюдением принципов безопасности и конфиденциальности.

Возможности пользователя представлены в UseCase диаграмме, которою можно посмотреть в приложении B.

# ГЛАВА 2 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

## 2.1 Постановка задачи

Важность постановки задач в проекте неоспорима, поскольку каждое действие направлено на достижение конечной цели. Цель определяет желаемый результат разработки проекта, а задачи представляют собой конкретные шаги и методы, необходимые для достижения этой цели. Хаотичные и несогласованные действия могут привести к задержкам или даже к провалу проекта, в то время как планирование и организация помогают эффективно реализовывать поставленные задачи и достигать поставленных целей.

«Web-приложение для просмотра фильмов и чтения книг» должно представлять собой не только удобный инструмент, но и интересное развлечение, а главное – быть актуальным. В процессе использования пользователь должен ощущать не просто техническую функциональность, а настоящее удовольствие от контента, который может оказаться полезным в будущем. Этот проект призван привлечь внимание не только молодежи, но и взрослых, предоставляя доступ к разнообразному контенту и помогая новичкам в мире кино и литературы. Важно понимать, что приложение не является универсальным средством решения всех проблем, но оно может стать надежным компаньоном, помогающим освоить основы этих искусств.

Разработка веб-приложения для просмотра фильмов и чтения книг требует определенных целей и задач:

1. Создание удобного интерфейса: Разработать пользовательский интерфейс, который был бы интуитивно понятен и привлекателен для пользователей.
2. Реализация функциональности поиска: Обеспечить возможность пользователям искать фильмы и книги.
3. Интеграция с базой данных: Создать и подключить базу данных PostgreSQL для хранения информации о фильмах, книгах, пользователях и их предпочтениях.
4. Разработка логики работы с данными: Написать код, который обрабатывает запросы пользователей и взаимодействует с базой данных для получения и отображения контента.
5. Обеспечение безопасности: Реализовать меры безопасности, такие как аутентификация пользователей и защита данных.
6. Оптимизация производительности: Оптимизировать код и ресурсы приложения для обеспечения быстрой загрузки и плавной работы.

Благодаря поставленным задачам разработка проекта будет осуществляться быстрее, чем без составленного плана работы. Вышеперечисленные пункты должны быть выполнены и полностью действительны в конечной версии проекта.

## 2.2 Концепция информационной базы и её структура

При разработке информационной базы для проекта использовалась реляционная модель данных, которая обеспечивает удобное и эффективное хранение информации. В данной базе данных применяется модель "сущность-связь" (ER-модель), которая позволяет описать сущности и их взаимосвязи.

При разработке была создана БД на PostgreSQL которая включает в себя такие данные как:

1. Таблица "Authors" (AuthorId, FirstName, SecondName, Patronymic) – содержит информацию об авторах книг;
2. Таблица "BookAuthors" (BookId, AuthorId) – устанавливает связь между книгами и их авторами;
3. Таблица "BookGenres" (BookId, GenreId) – устанавливает связь между книгами и их жанрами;
4. Таблица "Books" (BookId, BookName, BookDescription, PosterFilePath, BookFilePath, ReleaseDate) – содержит информацию о книгах;
5. Таблица "Directors" (DirectorId, FirstName, SecondName, Patronymic) – содержит информацию о режиссерах фильмов;
6. Таблица "FavoriteBooksUsers" (UserId, BookId) – устанавливает связь между пользователями и их любимыми книгами;
7. Таблица "FavoriteMoviesUsers" (UserId, MovieId) – устанавливает связь между пользователями и их любимыми фильмами;
8. Таблица "Genres" (GenreId, GenreName) – содержит информацию о жанрах;
9. Таблица "MovieFiles" (MovieFileId, MovieId, FilePath, EpisodNumber) – содержит информацию о файлах фильмов;
10. Таблица "MovieTypes" (MovieTypeId, MovieTypeName) – содержит информацию о типах фильмов;
11. Таблица "Movies" (MovieId, MovieName, MovieDescription, MovieTypeId, ReleaseDate, PosterFilePath) – содержит информацию о фильмах;
12. Таблица "MoviesDirectors" (MovieId, DirectorId) – устанавливает связь между фильмами и их режиссерами;
13. Таблица "MoviesGenres" (MovieId, GenreId) – устанавливает связь между фильмами и их жанрами;
14. Таблица "Roles" (RoleId, RoleName) – содержит информацию о ролях пользователей;
15. Таблица "Users" (UserId, Login, Password, RoleId, Email, PhoneNumber, CreatedDate) – содержит информацию о пользователях.

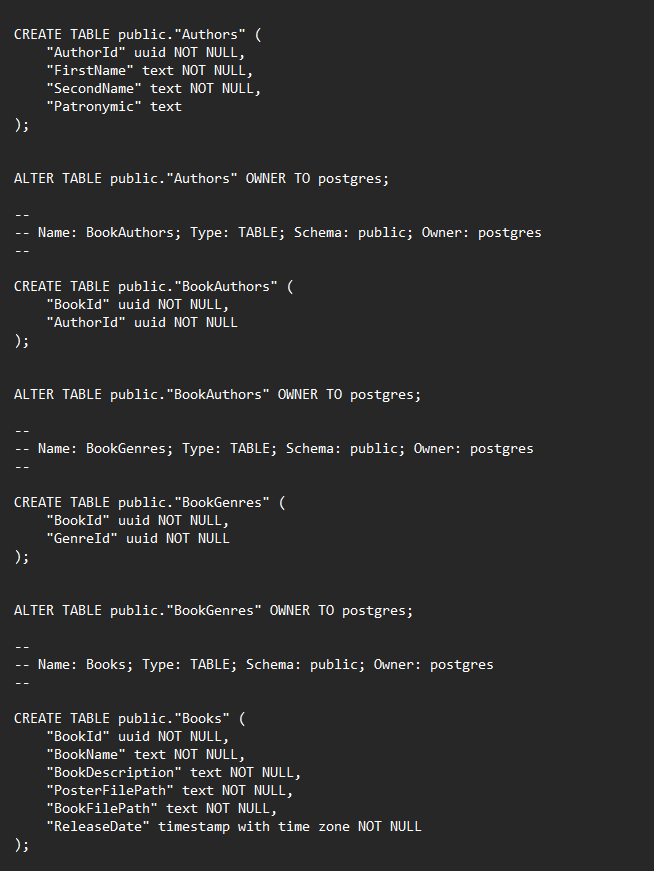


Рисунок 1 – Часть скрипта создания БД.

Некоторые таблицы в базе данных связаны через внешние ключи (FK) с атрибутом ID, который является первичным ключом (PK). Учитывая, что приложение будет обрабатывать большой объем данных и потребуется эффективное хранение и доступ к ним, база данных представляет собой достаточно крупную структуру. Ниже приведены описания сущностей для каждой таблицы.

Таблица 1 - "Authors" (Таблица данных об авторах)

| Поле | Тип данных | Атрибут |
| --- | --- | --- |
| AuthorId | UUID | PK |
| FirstName | TEXT |  |
| SecondName | TEXT |  |
| Patronymic | TEXT |  |

Таблица 2 - "BookAuthors" (Таблица для связи многие ко многим хранит идентификаторы книг и авторов)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| BookId | UUID | FK |
| AuthorId | UUID | FK |

Таблица 3 - "BookGenres" (Таблица для связи многие ко многим хранит идентификаторы книг и жанров)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| BookId | UUID | FK |
| GenreId | UUID | FK |

Таблица 4 - "Books" (Таблица данных о книгах)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| BookId | UUID | PK |
| BookName | TEXT |  |
| BookDescription | TEXT |  |
| PosterFilePath | TEXT |  |
| BookFilePath | TEXT |  |
| ReleaseDate | TIMESTAMP WITH TIME ZONE |  |

Таблица 5 - "Directors" (Таблица данных о режиссерах)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| DirectorId | UUID | PK |
| FirstName | TEXT |  |
| SecondName | TEXT |  |
| Patronymic | TEXT |  |

Таблица 6 - "FavoriteBooksUsers" (Таблица данных о избранных книгах у пользователя)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| UserId | UUID | FK |
| BookId | UUID | FK |

Таблица 7 - "FavoriteMoviesUsers" (Таблица данных о избранных фильмах у пользователя)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| UserId | UUID | FK |
| MovieId | UUID | FK |

Таблица 8 - "Genres" (Таблица данных о жанрах)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| GenreId | UUID | PK |
| GenreName | TEXT |  |

Таблица 9 - "MovieFiles" (Таблица данных о файлах фильмов)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| MovieFileId | UUID | PK |
| MovieId | UUID | FK |
| FilePath | TEXT |  |
| EpisodNumber | INTEGER |  |

Таблица 10 - "MovieTypes" (Таблица данных о типах фильмов)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| MovieTypeId | UUID | PK |
| MovieTypeName | TEXT |  |

Таблица 11 - "Movies" (Таблица данных о фильмах)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| MovieId | UUID | PK |
| MovieName | TEXT |  |
| MovieDescription | TEXT |  |
| MovieTypeId | UUID | FK |
| ReleaseDate | TIMESTAMP WITH TIME ZONE |  |
| PosterFilePath | TEXT |  |

Таблица 12 - "MoviesDirectors" (Таблица для связи многие ко многим хранит идентификаторы фильма и режиссера)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| MovieId | UUID | FK |
| DirectorId | UUID | FK |

Таблица 13 - "MoviesGenres" (Таблица для связи многие ко многим хранит идентификаторы фильма и жанра)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| MovieId | UUID | FK |
| GenreId | UUID | FK |

Таблица 14 - "Roles" (Таблица данных о ролях)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| RoleId | UUID | PK |
| RoleName | TEXT |  |

Таблица 15 - "Users" (Таблица данных о пользователях)

| **Поле** | **Тип данных** | **Атрибут** |
| --- | --- | --- |
| UserId | UUID | PK |
| Login | VARCHAR(30) |  |
| Password | TEXT |  |
| RoleId | UUID | FK |
| Email | TEXT |  |
| PhoneNumber | TEXT |  |
| CreatedDate | TEXT |  |

ER диаграмму можно посмотреть в приложении A.

## 2.3 Неформальное описание алгоритма работы ПП

База данных написана в СУБД PostgreSQL, содержит 15 таблиц и связи меду некоторыми таблицами. В данной БД возможны следующие действия:

Регистрация и авторизация пользователя:

* Проверка введенных данных.
* Создание новой учетной записи или проверка существующей.

Поиск контента

* Получение данных из базы данных.

Отображение контента

* Вывод списка фильмов и книг.
* Переход к детальной информации при выборе элемента списка.

Просмотр контента

* Воспроизведение видеофайлов.
* Отображение текста книг.

Персонализация

* Управление списком избранного.

При открытии главной страницы пользователь видит список последних 9 добавленных фильмов. Он может воспользоваться поиском по названию фильма или перейти к просмотру списка последних 9 добавленных книг. В разделе книг также предусмотрен поиск по названию. Неавторизованным пользователям доступны опции регистрации или входа в систему. После авторизации пользователь может перейти в свой профиль и просмотреть список избранных элементов. Если у пользователя роль Admin или Owner, он получает доступ к "Админ панели".

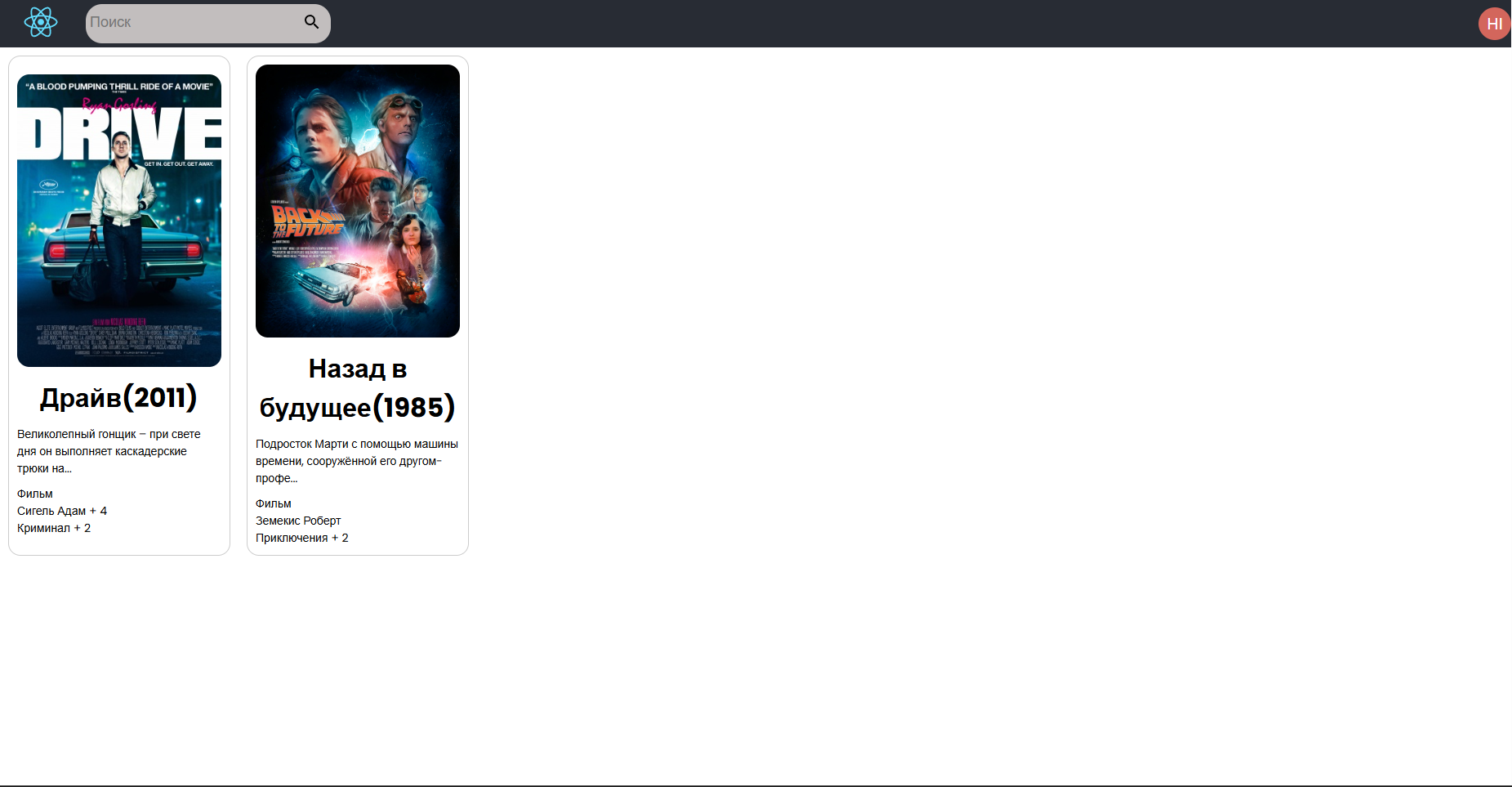


Рисунок 2 – Главная страница

При регистрации требуется заполнить обязательные поля, а также указать информацию, соответствующую определенным форматам символов. Если пользователь попытается войти в "Админ панель", не обладая соответствующей ролью или не авторизовавшись, он будет перенаправлен на главную страницу. Аналогично, если неавторизованный пользователь попытается просмотреть список избранных, он будет перенаправлен на страницу входа в систему.

Рисунок 3 – Страница авторизации

Для наглядности всего вышеперечисленного была создана блок-схема, которую можно посмотреть в приложении Б.

## 2.4 Функции подсистем (модулей)

Для улучшения понимания, доступ к базе данных был организован через определенные модули.

Модуль аутентификации

* Регистрация пользователей.
* Авторизация пользователей.
* Смена пароля.

Процесс регистрации пользователя начинается с заполнения определенной информации на веб-странице, после чего эти данные добавляются в базу данных. Код, отвечающий за добавление пользователя, представлен на рисунке 4, а внешний вид страницы регистрации можно увидеть на рисунке 5.

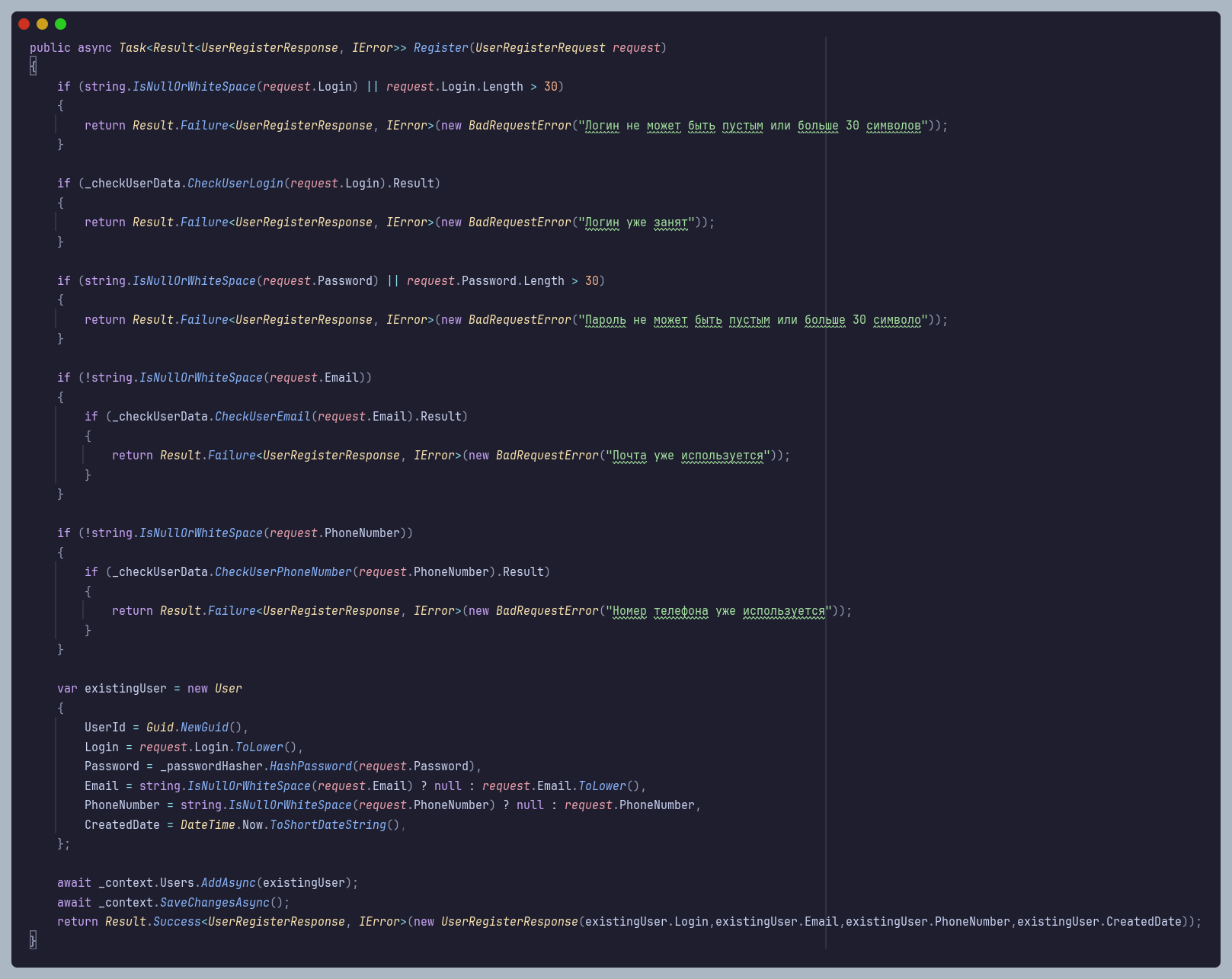


Рисунок 4 – Скрипт добавления пользователя в БД при регистрации

Рисунок 5 – Страница регистрации

Для авторизации пользователь вводит логин и пароль на веб-странице, после чего система осуществляет поиск соответствующего пользователя в базе данных по логину и проверяет введенный пароль. Код, отвечающий за поиск пользователя в базе данных и проверку пароля, представлен на рисунке 6, а внешний вид страницы авторизации можно увидеть на рисунке 7.



Рисунок 6 – Скрипт авторизации пользователя

Рисунок 7 – Страница авторизации

Модуль управления контентом

* Поиск фильмов и книг.
* Отображение списков контента.
* Отображение детальной информации.

Для поиска фильмов и книг по их названию пользователь может использовать специальное поле на веб-странице. Код, ответственный за выполнение поиска фильмов, представлен на рисунке 8, а код для поиска книг, представлен на рисунке 9.

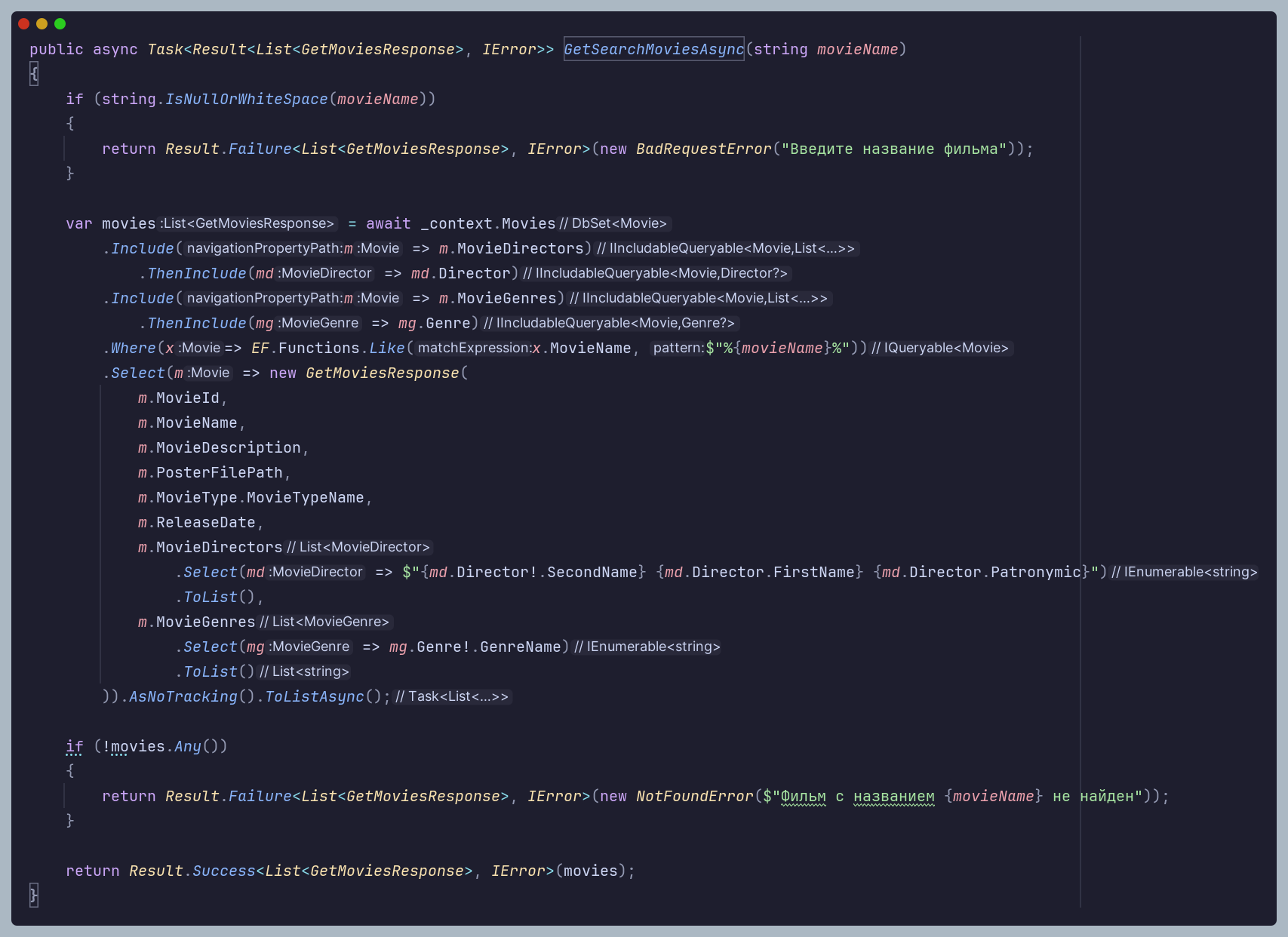


Рисунок 8 – Скрипт поиска фильмов в БД по названию

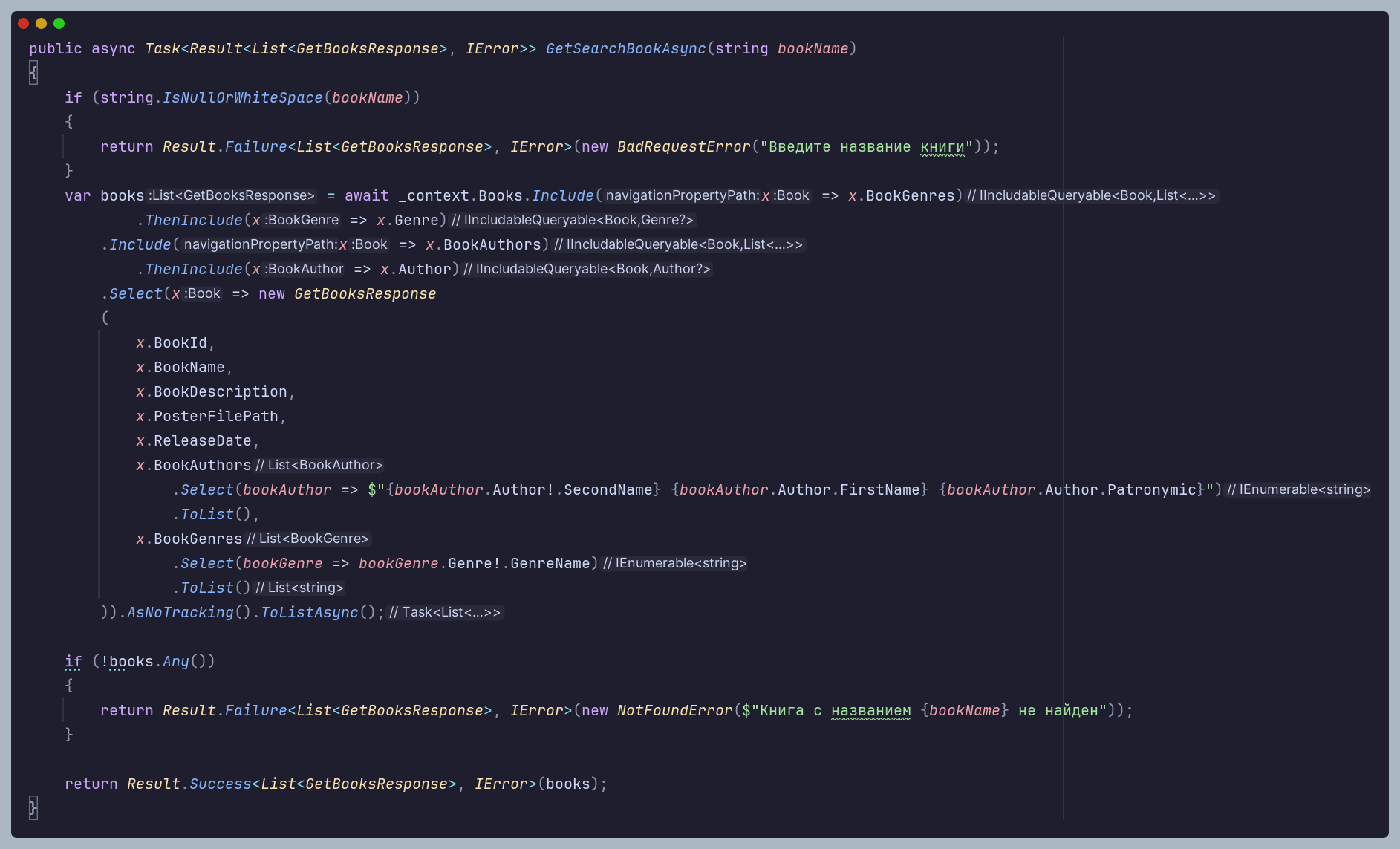


Рисунок 9 – Скрипт поиска книг в БД по названию

Чтобы просмотреть подробную информацию о конкретном фильме или книге, нужно кликнуть на их карточку, что перенаправит на страницу с деталями и возможностью просмотра. Код, обеспечивающий получение подробной информации о книге, представлен на рисунке 10, а пример страницы с этой информацией показан на рисунке 11. Код, отвечающий за получение детальной информации о фильме, представлен на рисунке 12, а страница с этой информацией показана на рисунке 13.

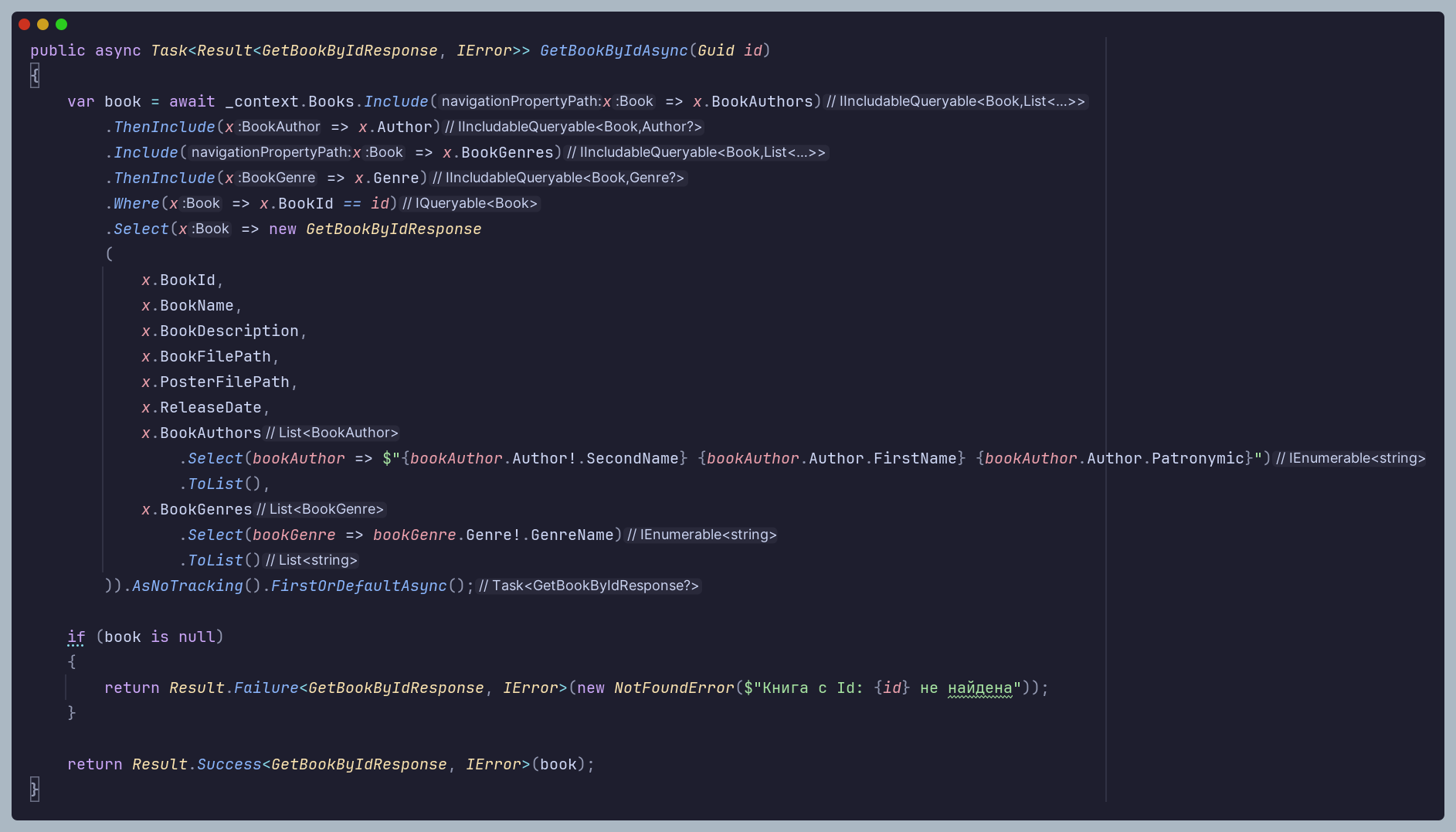


Рисунок 10 – Скрипт получения информации о конкретной книге

Рисунок 11 – Страница с детальной информацией о книге

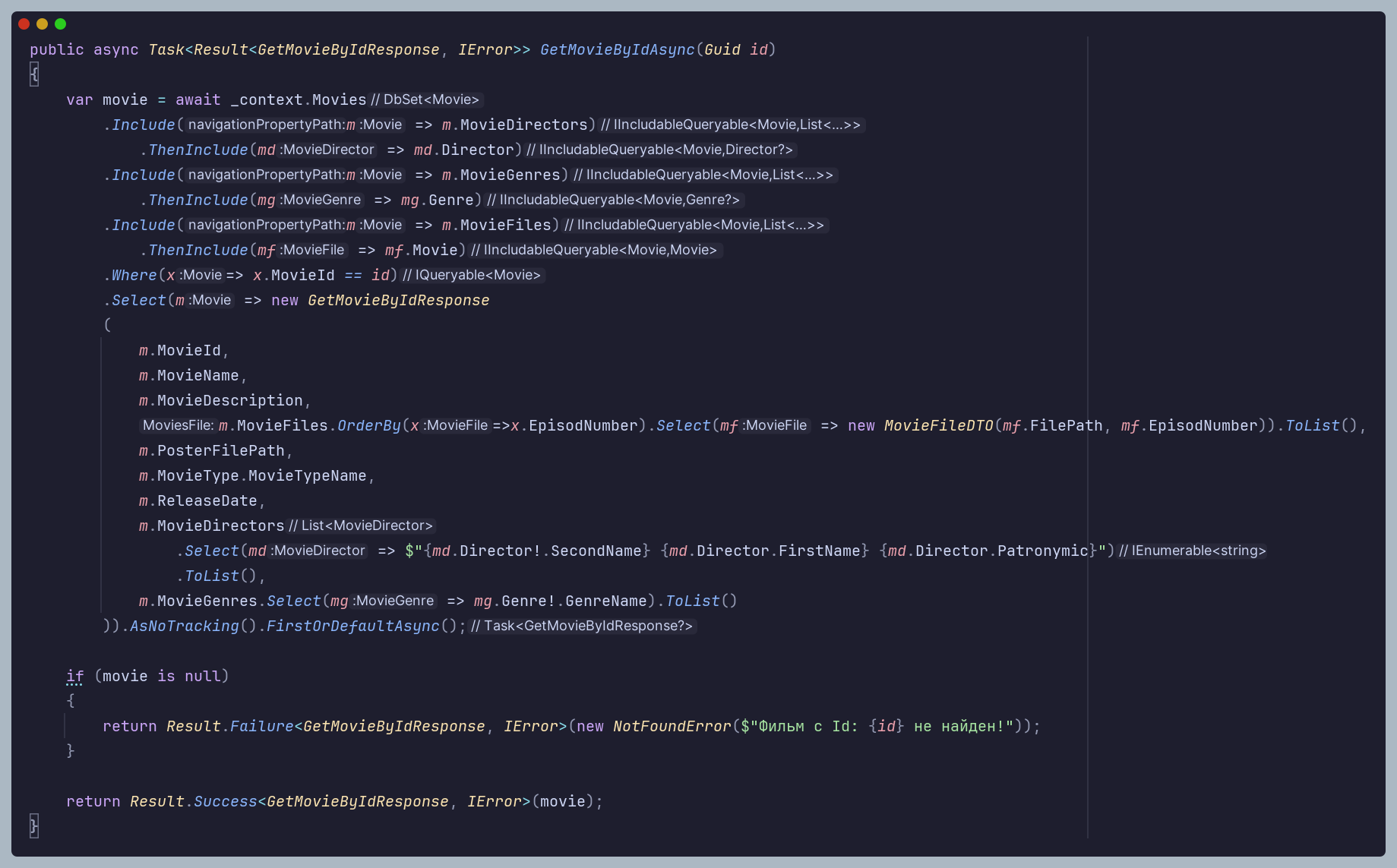


Рисунок 12 - Скрипт получения информации о конкретном фильме

Рисунок 13 – Страница с детальной информацией о фильме

Модуль просмотра контента

* Воспроизведение видеофайлов.
* Чтение текстовых файлов.

На странице с подробной информацией о фильме размещен видеоплеер, который пользователь может использовать для просмотра фильма. На странице с детальной информацией о книге имеется кнопка "Читать", при нажатии на которую пользователь будет перенаправлен на страницу для чтения книги.

## 2.5 Оценка экономической эффективности от внедрения программы

Оценка экономической эффективности показывает все положительные результаты при использовании проекта. Всегда необходимо сравнивать выгоды и затраты, которые будут возникать при разработке и реализации проекта. Это помогает понять, рационально или нерационально использовались предоставленные ресурсы для достижения целей. Данная оценка выявляет причины снижения эффективности работы над проектом и позволяет предпринять необходимые меры для их решения.

Чтобы оценить экономическую эффективность от внедрения веб-приложения «для управления мультимедийным контентом», необходимо сначала составить перечень всех этапов работ, которые будут реализованы. Для расчётов также понадобятся различные формулы, которые будут перечислены ниже. Распределение работ по этапам приведено в таблице 16.

|  |  |  |
| --- | --- | --- |
| Этап проведения | Вид работы на данном этапе | Трудоемкость  выполнения, чел.-ч. |
| Получение  информации о  предметной  области | Разработка концепта приложения | 3 |
| Сбор данных о предметной  области | 10 |
| Обработка данных | 4 |
| Разработка  программы | Разработка алгоритма приложения | 5 |
| Написание кода | 223 |
| Тестирование | 41 |
| Описания  программы | Подготовка руководства для администраторов | 1 |
| Итого трудоемкость выполнения дипломной работы  (проекта) | | 287 |

Таблица 16 – Распределение работ по этапам и видам

Для определения затрат на разработку web-приложения необходимо составить смету со следующими статьями:

* затраты на оплату труда;
* отчисления на социальные нужды;
* амортизация основных фондов;
* прочие затраты.

Общую сумму затрат на оплату труда можно посмотреть в таблице 4.

Таблица 4 – Затраты на оплату труда

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория работника | Квалификация | Трудоемкость  разработки  игры,  чел.-ч. | Часовая  ставка,  руб/ч | Сумма,  руб |
| Разработчик  программы | Студент-  программист | 287 | 259 | 74 333 |
| Итого |  |  |  | 74 333 |

Общая сумма затрат на оплату труда определяется по формуле (1):

, (1)

Обозначения данной формулы:

* – часовая ставка i-го работника, руб.;
* - время на разработку, час.;
* i – категория работника;
* n – количество работников, разрабатывающих приложение.

Среднечасовая заработная плата разработчика рассчитывается по формуле (2):

, (2)

Обозначения данной формулы:

* – среднемесячная заработная плата программиста (в данном случае берется средняя заработная плата программиста без опыта работы по Курской области, с учетом НДФЛ 13% - 43500 руб.);
* – среднемесячный фонд рабочего времени, 168 часов в месяц.

Получается, стоимость одного часа работы программиста равна:

Общая сумма затрат на оплату труда равна:

Отчисления на социальные нужды включают в себя сумму единого социального налога и взносы на страхование от несчастных случаев и профессиональных заболеваний, которые составляют 35,6% и 0,2% от затрат на оплату труда всех работников, занятых разработкой игры.

Отчисления на социальные нужды составят:

Расчет амортизационных отчислений приведен в таблице 17.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Наименование  оборудования | Стоимость  оборудования,  руб. | Годовая норма  амортизации,  % | Время работы  оборудования во  время разработки  игры, ч | Сумма,  руб. |
| Компьютер (корпус) | 4600 | 20 | 287 | 920 |
| Оперативная  память | 3399 | 679,8 |
| Видеокарта | 13650 | 2730 |
| Процессор | 18700 | 3740 |
| [Жесткий диск](https://www.dns-shop.ru/product/4546a647f761ed20/2-tb-zestkij-disk-wd-blue-wd20ezaz/) | 4500 | 900 |
| Материнская  плата | 7800 | 1560 |
| Блок питания | 2500 | 500 |
| Монитор | 10000 | 2000 |
| Куллер | 1000 | 200 |
| Итого: | | | | 13 229,8 |

Таблица 17 − Расчет амортизационных отчислений

Общая сумма амортизационных отчислений определяется по формуле (3):

, (3)

Обозначения данной формулы:

* – стоимость i-го оборудования, руб.;
* – годовая норма амортизации i-го оборудования, %;
* i – вид оборудования;
* n – количество оборудования.

Сумма амортизационных отчислений составит:

Прочие затраты содержат расходы на содержание административно-управленческого и учебно-вспомогательного персонала, на отопление, освещение и текущий ремонт помещений, канцелярию, командировочные и разные хозяйственные расходы. Затраты по этой статье рассчитываются в размере 70% от затрат на оплату труда.

Исходя из следующих данных была составлена смета затрат на разработку web-приложения которая приведена в таблице 18.

|  |  |
| --- | --- |
| Статьи затрат | Сумма, руб. |
| 1. Затраты на оплату труда | 74333 |
| 2. Отчисления на социальные нужды | 27 327,21 |
| 3. Амортизация основных фондов | 13 229,8 |
| 4. Прочие затраты | 53 528,3 |
| Итого по смете | 168418,31 |

Таблица 18 − Смета затрат на разработку

Договорная цена для игры рассчитывается по формуле (4):

, (4)

Обозначения данной формулы:

* – затраты на разработку, руб.;
* – средний уровень рентабельности, 25%.

Исходя из этого, цена данной игры, будет следующей:

# ГЛАВА 3 ЭСКИЗНЫЙ ПРОЕКТ

## 3.1 Разработка эскиза проекта

Эскизный проект даёт общее представление о принципе работы программного продукта, об информации по устройству этого изделия. Составлять его стоит до разработки технического проекта и документации.

Эскизный проект позволяет:

* выявить инвестиционную ценность объекта;
* определить нужный объём работ для реализации проекта;
* избежать дальнейших ошибок;
* наглядно представить проект.

При заходе на сайт открывается главная страница, состоящая из следующих элементов:

Header состоящий из

* Поле ввода «Поле поиска»
* Кнопка «Вход»
* Кнопка «Регистрация»
* Кнопка «Профиль»

Main состоит из

* Карточки фильмов
* Карточки книг

Footer состоит из

* Кнопки «контактов»
* Текст «Email»

На рисунке 14 показан макет главной страницы.

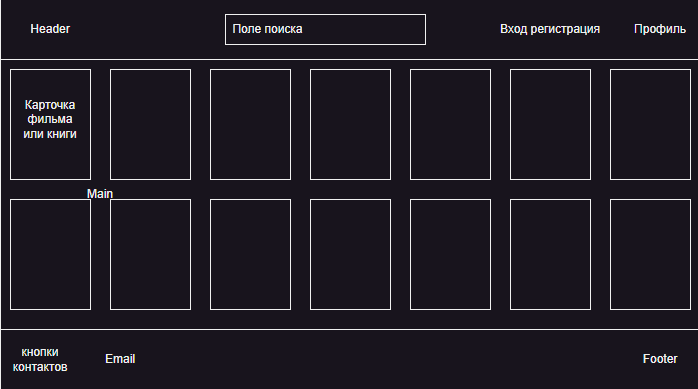


Рисунок 14 – Макет главной страницы

При нажатии на кнопку «Вход», пользователя перенаправляет на страницу авторизации, если он не авторизован. Кнопка «Регистрация» также отображается только для неавторизованных пользователей и при нажатии ведет на страницу регистрации. Кнопка «Профиль» видна только авторизованным пользователям и при нажатии перенаправляет на страницу профиля. При клике на карточку фильма пользователь попадает на страницу соответствующего фильма, аналогично при нажатии на карточку книги — на страницу этой книги. Нажатие на кнопки контактов перенаправляет пользователя на страницы социальных сетей или других контактных страниц.

В разрабатываемом проекте присутствуют так же индивидуальная страница для фильма и книги, макет индивидуальных страниц можно посмотреть на рисунках 15 и 16.

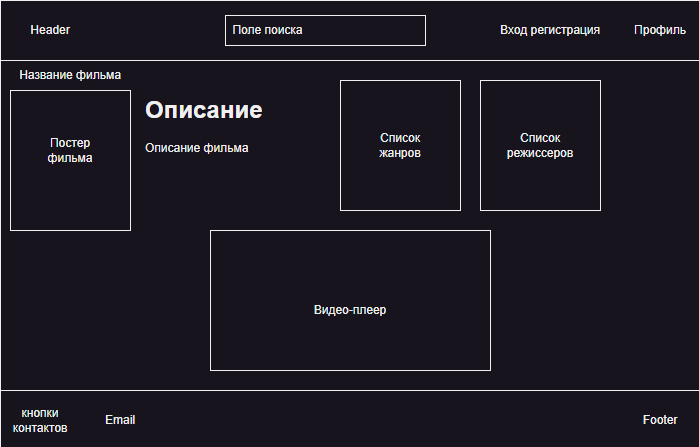


Рисунок 15 – Макет индивидуальной страницы для фильма

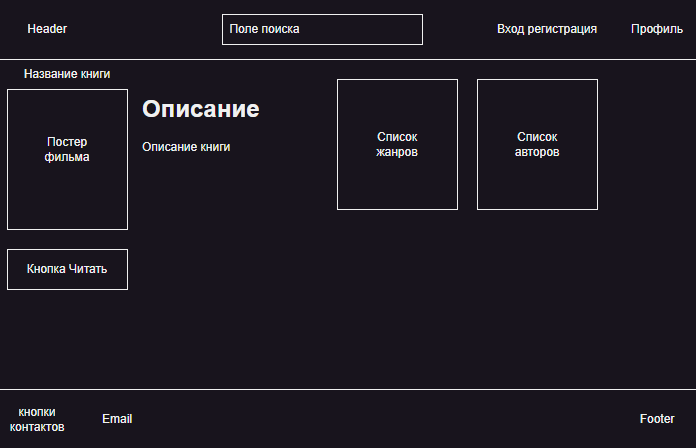


Рисунок 16 – Макет индивидуальной страницы для книги

Так же присутствует админ панель с ее макетом можно ознакомиться на рисунке 17.

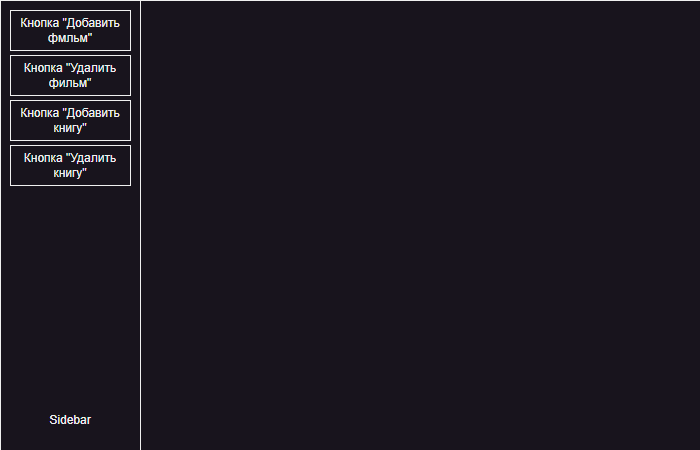


Рисунок 17 – Макет админ панели

## 3.2 Верстка интерфейса

Вёрстка интерфейса является важным этапом в разработке web-приложения, так как от её качества зависит восприятие проекта пользователями. Разработчик должен учитывать потребности пользователя, создавая удобный, привлекательный и интуитивно понятный интерфейс. Главная страница, включает в себя различные элементы управления.

Смотреть рисунок 18;

Рисунок 18 – Главная страница

Главная страница, включает в себя кнопку регистрации. При данном выборе пользователю предлагается ввести логин, пароль, номер телефона, электронную почту. Смотреть рисунок 19.

Рисунок 19 – Страница регистрации

Главная страница так же включает в себя кнопку входа. При данном выборе пользователю предлагается ввести логин и пароль. Смотреть рисунок 20.

Рисунок 20 – Страница входа

Еще главная страница включает в себя кнопку профиля. При данном выборе пользователь попадет в свой профиль. Смотреть рисунок 21.

Рисунок 21 – страница профиля

Главная страница также содержит основную секцию, где представлены карточки фильмов и книг. При нажатии на любую из этих карточек пользователь будет перенаправлен на отдельную страницу с подробной информацией о выбранной книге или фильме. Смотреть рисунок 22 и 23.

Рисунок 22 – Страница для книги

Рисунок 23 – Страница для фильма

# ГЛАВА 4 РАЗРАБОТКА ПРОЕКТНЫХ РЕШЕНИЙ

## 4.1 Разработка API

Разработка web-приложения осуществлялась с использованием фреймворка ASP.Net Core и языка программирования C#. Код писался в среде разработки Rider. Для взаимодействия с базой данных использовался EntityFramework Core, а в качестве СУБД применялся PostgreSQL.

ASP.NET Core – это удобный и мощный фреймворк для веб-разработки. Он предлагает большое количество различных библиотек, а также является кроссплатформенным, что позволяет вести разработку на Linux, macOS и Windows, что является несомненным преимуществом.

Rider оснащён редактором кода, множеством настроек и обширной библиотекой плагинов, которые упрощают работу с данной IDE. Его главные преимущества включают простоту, гибкость и возможность кастомизации, что позволяет пользователю полностью адаптировать IDE под свои нужды. Интерфейс максимально прост и понятен.

Использование EntityFramework Core в связке с СУБД PostgreSQL для написания дипломной работы обусловлено легкостью подключения, возможностью создания базы данных непосредственно из кода на C#, а также удобной настройкой связей между таблицами и свойств полей прямо в коде. На рисунке 24 можно увидеть настройку конфигурации для 1 из таблиц.

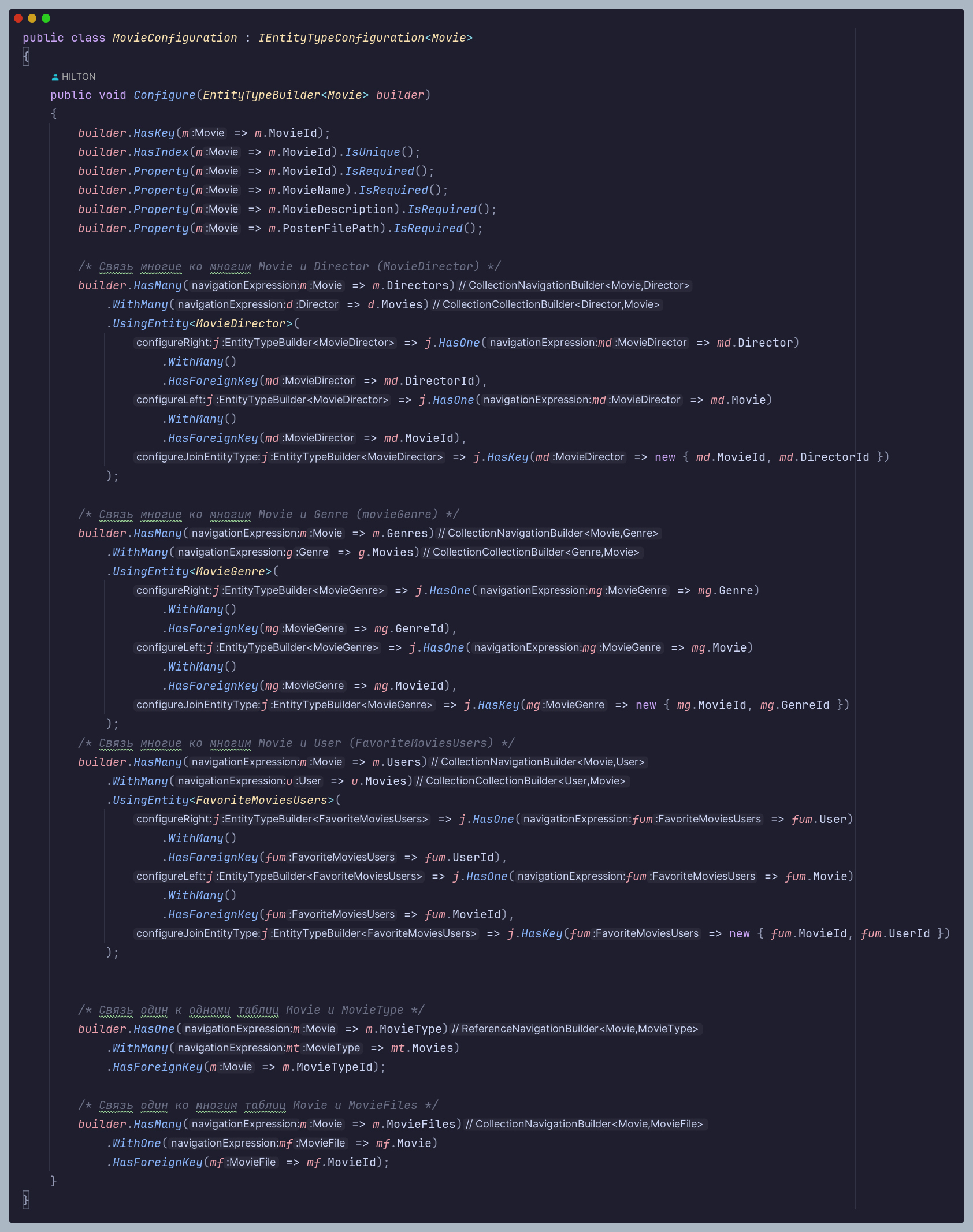


Рисунок 24 – Код настройки конфигурации для таблицы Movies

Этот код определяет свойства полей таблицы Movies и устанавливает связи с внешними таблицами.

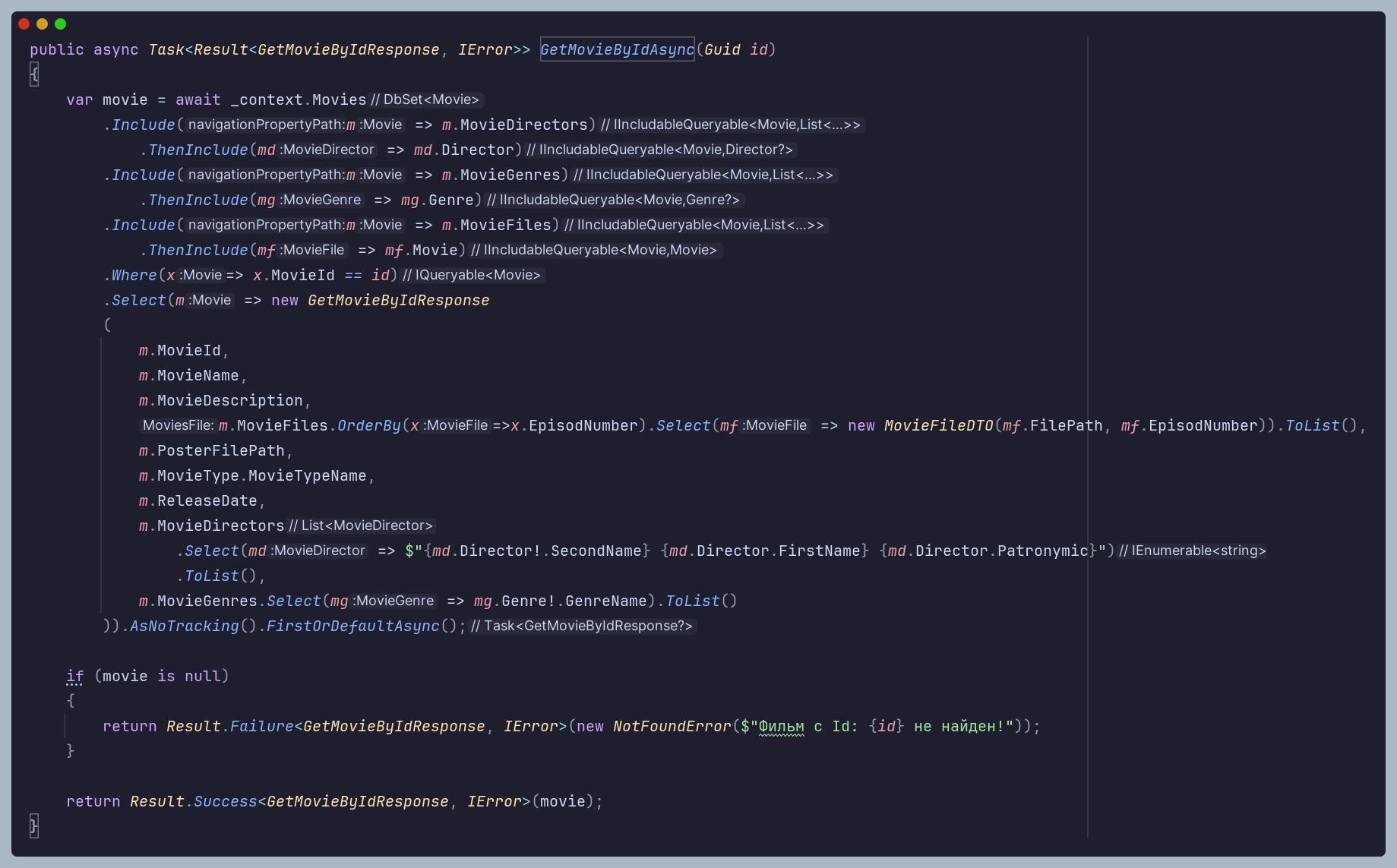
На рисунке 25 представлен код для получения информации о фильме по ее идентификатору. Сначала мы пытаемся получить фильм с нужными данными по его идентификатору. Затем условие проверяет, найден ли фильм. Если фильм не найден, возвращается результат с сообщением об ошибке. Если фильм найден, возвращается успешный результат вместе с данными о фильме.

Рисунок 25 – Код получения информации о фильме по его идентификатору

На рисунке 26 представлен код API, который обрабатывает результат получения фильма по идентификатору. Сначала мы передаем в переменную result результат, возвращенный из метода GetMovieByIdAsync. Затем условие проверяет, является ли результат ошибочным. В случае ошибки мы определяем, какой объект ошибки был возвращен, и возвращаем соответствующий статус-код с сообщением об ошибке. Если результат успешен, мы возвращаем успешный результат и данные фильма.

Рисунок 26 – Код API для обработки получения фильма по id

На рисунке 27 представлена часть всех путей API в приложении. Видно, что для некоторых URL-адресов установлены различные политики доступа, что позволяет разграничивать возможности разных ролей в нашем приложении.

Рисунок 27 – Код части путей API

На рисунке 28 представлено подключение всех API в нашем приложении. В этом коде мы видим, как различные маршруты добавляются к приложению с помощью расширений. Это позволяет структурировать код и упрощает управление маршрутами для различных сущностей, таких как пользователи, фильмы, жанры, авторы, книги и многие другие сущности.

Рисунок 28 – Код подключения всех API в приложении

## 4.2 Разработка интерфейса

Разработка интерфейса производилась с использованием фреймворка React в связке с языком TypeScript. Код писался в WebStrom. Для связью с API использовались такие технологии как RTK Query и Axios. А для хранения данных Redux.

React - библиотека для создания пользовательских интерфейсов. Ее компонентный подход позволяет эффективно организовывать код, а виртуальный DOM улучшает производительность приложений.

TypeScript - расширение JavaScript с статической типизацией, что улучшает надежность кода и делает его более понятным и поддерживаемым.

WebStorm - интегрированная среда разработки для веб-приложений. Ее мощный редактор кода и множество инструментов улучшают процесс разработки.

RTK Query - библиотека для работы с сетевыми запросами в Redux Toolkit. Она предоставляет простой API и автоматически генерирует код для управления состоянием.

Axios - библиотека для выполнения HTTP-запросов. Ее удобный интерфейс и поддержка различных функций делают работу с сетью более эффективной.

Redux - библиотека для управления состоянием приложения. Она предоставляет единое хранилище для данных и делает управление состоянием более предсказуемым и удобным. На рисунке 29 можно увидеть код создания хранилища Redux.



Рисунок 29 – Код создания хранилища Redux.

На рисунке 30 можно увидеть код для получения информации по конкретному фильму. Можно увидеть что здесь применяется Axios для отправки запроса. А так же HTML для создания интерфейса.

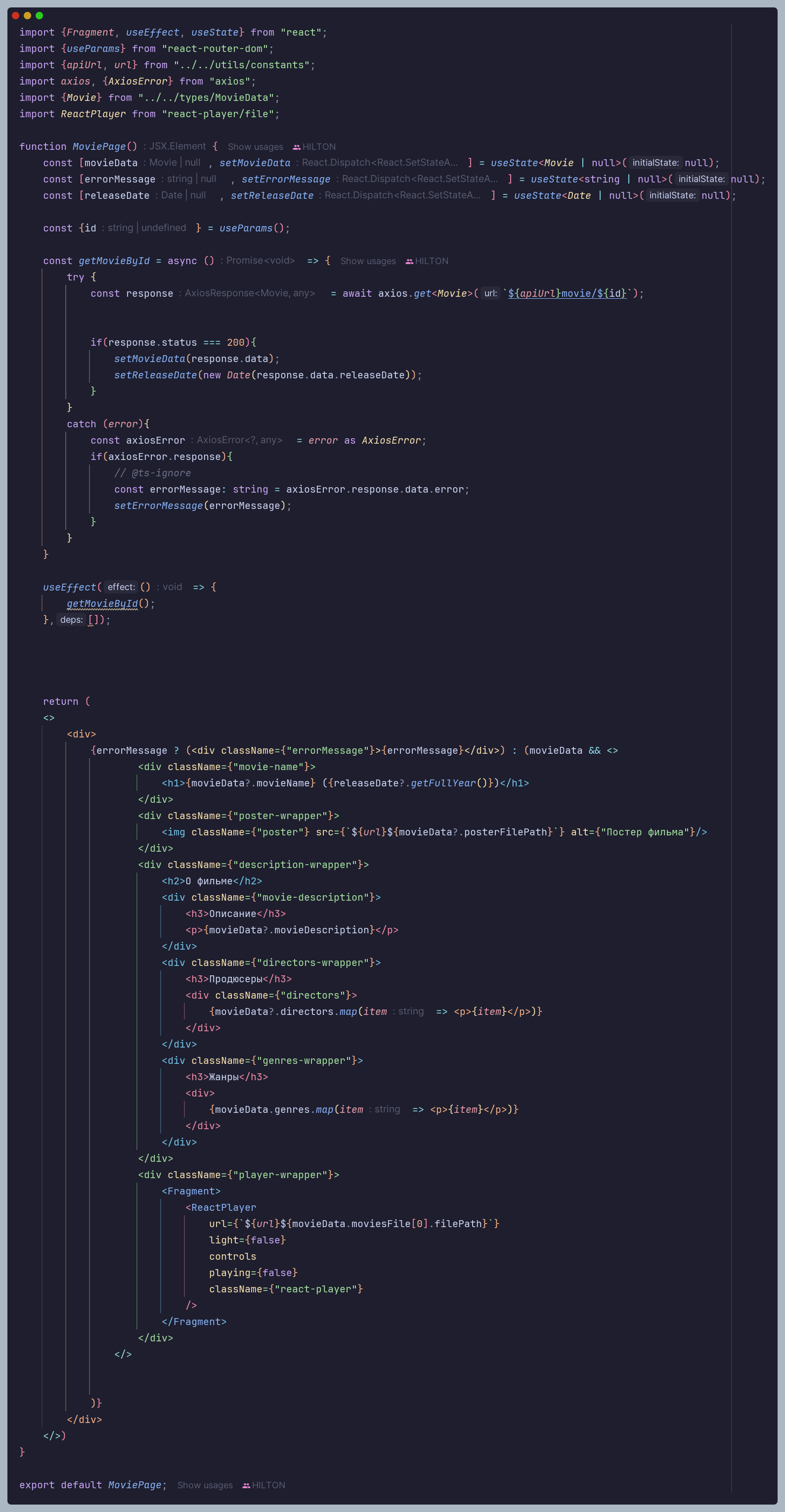


Рисунок 30 – Код страницы для определенного фильма.

# ЗАКЛЮЧЕНИЕ

Web-приложение для просмотра фильмов и чтения книг разработано и полностью функционирует. В процессе разработки были учтены все требования к дипломной работе. Приложение успешно выполняет свои основные задачи, предоставляя пользователям удобный интерфейс для доступа к коллекциям фильмов и книг.

Web-приложение реализовано с использованием современных технологий и инструментов. На стороне backend использован фреймворк ASP.NET Core .NET 8.0 и язык программирования C#. Для работы с базой данных применен EntityFramework Core, а в качестве СУБД выбрана PostgreSQL. В результате, приложение обладает высокой производительностью и возможностью масштабирования.

Для frontend-разработки были использованы HTML, CSS, и TypeScript. Основным фреймворком для создания пользовательского интерфейса стал React, а для упрощения работы с состоянием приложения применены библиотеки Redux и RTK. Для взаимодействия с API сервера использован Axios.

В итоге, разработанное web-приложение включает в себя:

* Удобный интерфейс для просмотра фильмов и чтения книг;
* Возможность авторизации и регистрации пользователей;
* Персонализированные профили пользователей;
* Интерактивные карточки фильмов и книг с подробной информацией;
* Возможность добавления фильмов и книг в избранное;
* Поиск фильмов и книг;
* Поддержку различных ролей пользователей с разграничением доступа к функционалу;

Процесс разработки следовал популярной методологии Scrum, что позволило эффективно управлять задачами и ресурсами. Каждый спринт имел четкую цель и завершался выпуском новой рабочей версии продукта. Периодическое тестирование помогло выявить и исправить большинство ошибок на ранних этапах разработки, что способствовало созданию качественного и надежного продукта.

Выбор ASP.NET Core и React был сделан на основе их гибкости, производительности и обширного сообщества разработчиков. Эти технологии обеспечили высокую скорость разработки и позволили реализовать все необходимые функции приложения.

В итоге, разработанное web-приложение полностью соответствует заявленным требованиям и успешно выполняет поставленные задачи. Оно обеспечивает удобный и интуитивно понятный интерфейс для пользователей любого возраста, предоставляя им возможность наслаждаться просмотром фильмов и чтением книг.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Руководство по ASP.NET Core 8// METANIT.COM URL: https://metanit.com/sharp/aspnet6/ (дата обращения: 08.03.2024).
2. Руководство по React// METANIT.COM URL: https://metanit.com/web/react/?ysclid=lwttwb20im715565816 (дата обращения: 09.04.2024).
3. Руководство по Entity Framework Core 8// METANIT.COM URL: https://metanit.com/sharp/efcore/lwttwb20im715565816 (дата обращения: 01.02.2024).
4. Полное руководство по языку программирования С# 12 и платформе .NET 8// METANIT.COM URL: https://metanit.com/sharp/tutorial/ (дата обращения: 05.01.2024).

# ПРИЛОЖЕНИЕ А

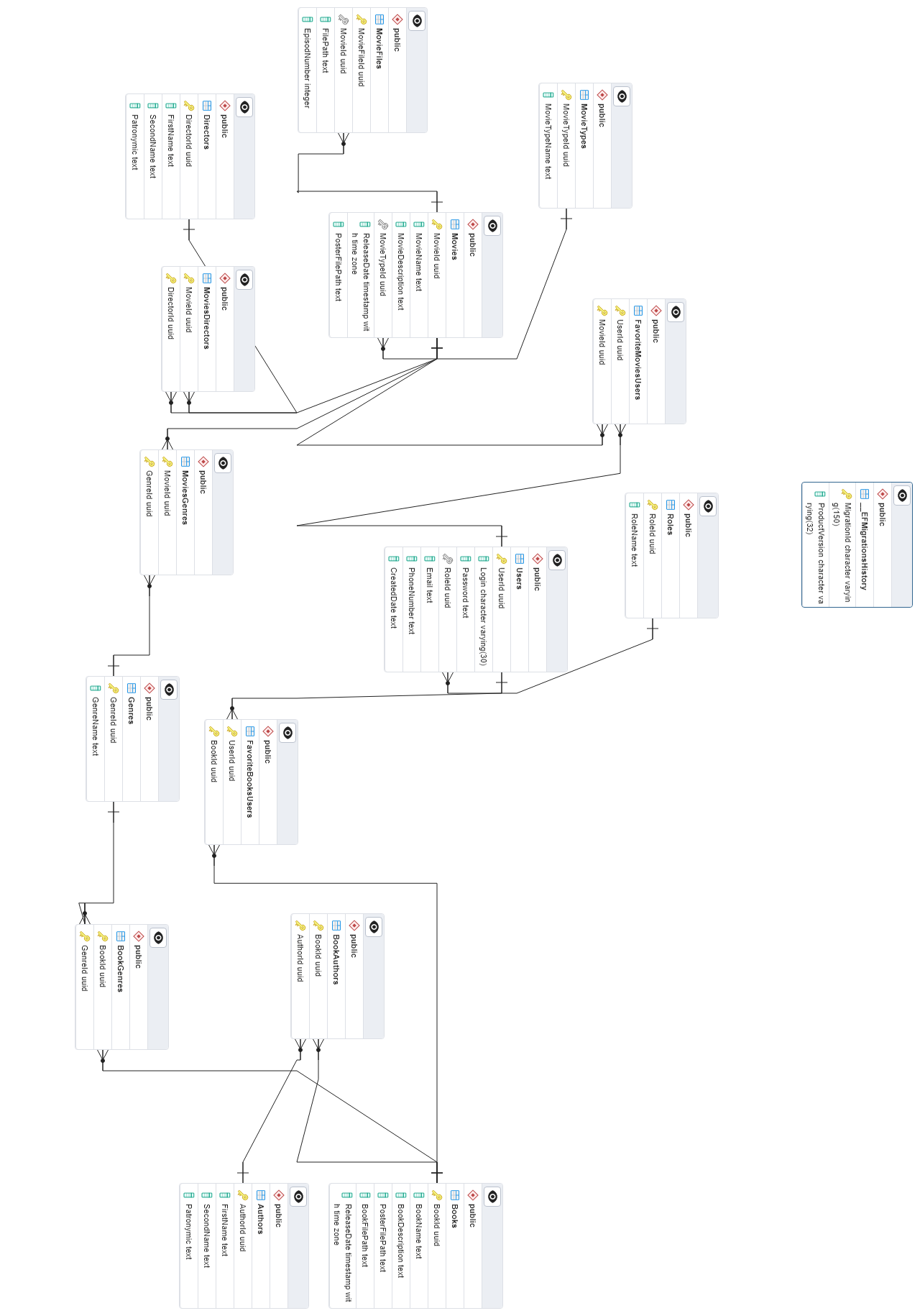


Рисунок 31 – ER диаграмма

# ПРИЛОЖЕНИЕ Б

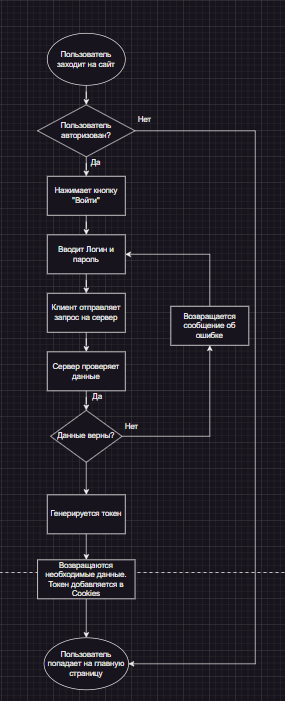


Рисунок 32 – Блок-схема алгоритма авторизации пользователя

# ПРИЛОЖЕНИЕ В



Рисунок 33 – UseCase диаграмма страницы фильма

# ПРИЛОЖЕНИЕ Г

Листинг 1 MovieManager.cs Методы для взаимодействия с БД для сущности Movie.

using CSharpFunctionalExtensions;  
using Microsoft.EntityFrameworkCore;  
using UserService.Domain.Interfaces;  
using UserService.Infrastructure.Context;  
using UserService.Infrastructure.ErrorObjects;  
using UserService.Domain.Contracts;  
using UserService.Domain.DTO.MovieDTO;  
using UserService.Domain.Models;  
  
namespace UserService.Infrastructure.Repositories;  
  
public class MovieManager : IMovieManager  
{  
 private readonly ApplicationDbContext \_context;  
 private readonly IFileHelper \_fileHelper;  
 private readonly ISorting \_sorting;  
  
 private readonly int \_getMovie = 9;  
 public MovieManager(ApplicationDbContext context, IFileHelper fileHelper, ISorting sorting)  
 {  
 \_context = context;  
 \_fileHelper = fileHelper;  
 \_sorting = sorting;  
 }  
   
 */// <summary>  
 /// Добавление фильма  
 /// </summary>  
 /// <param name="request">Данные для добавления фильма</param>  
 /// <returns>Статус</returns>* public async Task<Result<AddMovieRequest, IError>> AddMovieAsync(AddMovieRequest request)  
 {  
 if (string.*IsNullOrWhiteSpace*(request.MovieName))  
 {  
 return Result.*Failure*<AddMovieRequest, IError>(new BadRequestError("Введите название фильма"));  
 }  
   
 if (string.*IsNullOrWhiteSpace*(request.MovieDescription))  
 {  
 return Result.*Failure*<AddMovieRequest, IError>(new BadRequestError("Введите описание"));  
 }  
   
 if (string.*IsNullOrWhiteSpace*(request.MovieType.ToString()))  
 {  
 return Result.*Failure*<AddMovieRequest, IError>(new BadRequestError("Выберите тип"));  
 }  
  
 if (request.Directors.Count == 0)  
 {  
 return Result.*Failure*<AddMovieRequest, IError>(new BadRequestError("Выберите режиссера"));  
 }  
  
 if (request.Genres.Count == 0)  
 {  
 return Result.*Failure*<AddMovieRequest, IError>(new BadRequestError("Выберите жанр"));  
 }  
  
 var movieFilePath = await \_fileHelper.LoadVideoFileAsync(request.MovieFile, request.MovieName);  
 if (movieFilePath.IsFailure)  
 {  
 return Result.*Failure*<AddMovieRequest, IError>(new BadRequestError($"{movieFilePath.Error}"));  
 }  
   
 var posterFilePath = await \_fileHelper.LoadImageFileAsync(request.PosterFile, request.MovieName);  
 if (movieFilePath.IsFailure)  
 {  
 return Result.*Failure*<AddMovieRequest, IError>(new BadRequestError($"{posterFilePath.Error}"));  
 }  
  
 using (var transaction = await \_context.Database.BeginTransactionAsync())  
 {  
 try  
 {  
 var movie = new Movie  
 {  
 MovieId = Guid.*NewGuid*(),  
 MovieName = request.MovieName,  
 MovieDescription = request.MovieDescription,  
 PosterFilePath = posterFilePath.Value,  
 MovieTypeId = request.MovieType,  
 ReleaseDate = request.ReleaseData  
 };  
  
 await \_context.Movies.AddAsync(movie);  
  
 var movieFile = new MovieFile  
 {  
 MovieFileId = Guid.*NewGuid*(),  
 MovieId = movie.MovieId,  
 FilePath = movieFilePath.Value  
 };  
  
 await \_context.MovieFiles.AddAsync(movieFile);  
  
 var movieGenres = request.Genres.Select(genre => new MovieGenre  
 {  
 MovieId = movie.MovieId,  
 GenreId = genre  
 }).ToList();  
  
 await \_context.MoviesGenres.AddRangeAsync(movieGenres);  
  
 var movieDirectors = request.Directors.Select(director => new MovieDirector  
 {  
 MovieId = movie.MovieId,  
 DirectorId = director  
 }).ToList();  
  
 await \_context.MoviesDirectors.AddRangeAsync(movieDirectors);  
  
 await \_context.SaveChangesAsync();  
  
 await transaction.CommitAsync();  
 }  
 catch (Exception e)  
 {  
 await transaction.RollbackAsync();  
 return Result.*Failure*<AddMovieRequest, IError>(new BadRequestError("Что-то пошло не так"));   
 }  
  
 return Result.*Success*<AddMovieRequest,IError>(request);  
 }  
 }  
   
 */// <summary>  
 /// Получение всех фильмов   
 /// </summary>  
 /// <returns>Список фильмов (Для рендеринга карточек фильмов на клиенте)</returns>* public async Task<Result<List<GetMoviesResponse>, IError>> GetSearchMoviesAsync(string movieName)  
 {  
 if (string.*IsNullOrWhiteSpace*(movieName))  
 {  
 return Result.*Failure*<List<GetMoviesResponse>, IError>(new BadRequestError("Введите название фильма"));  
 }  
   
 var movies = await \_context.Movies  
 .Include(m => m.MovieDirectors)  
 .ThenInclude(md => md.Director)  
 .Include(m => m.MovieGenres)  
 .ThenInclude(mg => mg.Genre)  
 .Where(x=> EF.Functions.Like(x.MovieName, $"%{movieName}%"))  
 .Select(m => new GetMoviesResponse(  
 m.MovieId,  
 m.MovieName,  
 m.MovieDescription,  
 m.PosterFilePath,  
 m.MovieType.MovieTypeName,  
 m.ReleaseDate,  
 m.MovieDirectors  
 .Select(md => $"{md.Director!.SecondName} {md.Director.FirstName} {md.Director.Patronymic}")  
 .ToList(),  
 m.MovieGenres  
 .Select(mg => mg.Genre!.GenreName)  
 .ToList()  
 )).AsNoTracking().ToListAsync();  
  
 if (!movies.Any())  
 {  
 return Result.*Failure*<List<GetMoviesResponse>, IError>(new NotFoundError($"Фильм с названием {movieName} не найден"));  
 }  
   
 return Result.*Success*<List<GetMoviesResponse>, IError>(movies);  
 }  
  
 public async Task<Result<List<GetMoviesResponse>>> GetFavoriteMoviesByUserIdAsync(Guid userId)  
 {  
 var movies = await \_context.Movies.Include(x=> x.MovieDirectors)  
 .ThenInclude(x=> x.Director)  
 .Include(x=> x.MovieGenres)  
 .ThenInclude(x=> x.Genre)  
 .Where(x=> x.FavoriteMoviesUsers.Any(fmu=> fmu.UserId == userId))  
 .Select(m => new GetMoviesResponse  
 (  
 m.MovieId,  
 m.MovieName,  
 m.MovieDescription,  
 m.PosterFilePath,  
 m.MovieType.MovieTypeName,  
 m.ReleaseDate,  
 m.MovieDirectors  
 .Select(md =>  
 $"{md.Director!.SecondName} {md.Director.FirstName} {md.Director.Patronymic}")  
 .ToList(),  
 m.MovieGenres.Select(mg => mg.Genre!.GenreName).ToList()  
 )).AsNoTracking().ToListAsync();  
   
 return Result.*Success*(movies);  
 }  
  
 public async Task<Result<List<GetMoviesResponse>>> GetMoviesAsync()  
 {  
 var movies = await \_context.Movies  
 .Include(m => m.MovieDirectors)  
 .ThenInclude(md => md.Director)  
 .Include(m => m.MovieGenres)  
 .ThenInclude(mg => mg.Genre)  
 .OrderByDescending(m=>m.ReleaseDate)  
 .Select(m => new GetMoviesResponse  
 (  
 m.MovieId,  
 m.MovieName,  
 m.MovieDescription,  
 m.PosterFilePath,  
 m.MovieType.MovieTypeName,  
 m.ReleaseDate,  
 m.MovieDirectors  
 .Select(md => $"{md.Director!.SecondName} {md.Director.FirstName} {md.Director.Patronymic}")  
 .ToList(),  
 m.MovieGenres.Select(mg => mg.Genre!.GenreName).ToList()  
 )).AsNoTracking().Take(\_getMovie).ToListAsync();  
   
 return Result.*Success*(movies);  
 }  
  
 */// <summary>  
 /// Получение всех данных фильма по Id  
 /// </summary>  
 /// <param name="id"> Идентификатор нужного нам фильма</param>  
 /// <returns>Все данные нужного нам фильма</returns>* public async Task<Result<GetMovieByIdResponse, IError>> GetMovieByIdAsync(Guid id)  
 {  
 var movie = await \_context.Movies  
 .Include(m => m.MovieDirectors)  
 .ThenInclude(md => md.Director)  
 .Include(m => m.MovieGenres)  
 .ThenInclude(mg => mg.Genre)  
 .Include(m => m.MovieFiles)  
 .ThenInclude(mf => mf.Movie)  
 .Where(x=> x.MovieId == id)  
 .Select(m => new GetMovieByIdResponse  
 (  
 m.MovieId,  
 m.MovieName,  
 m.MovieDescription,  
 m.MovieFiles.OrderBy(x=>x.EpisodNumber).Select(mf => new MovieFileDTO(mf.FilePath, mf.EpisodNumber)).ToList(),  
 m.PosterFilePath,  
 m.MovieType.MovieTypeName,  
 m.ReleaseDate,  
 m.MovieDirectors  
 .Select(md => $"{md.Director!.SecondName} {md.Director.FirstName} {md.Director.Patronymic}")  
 .ToList(),  
 m.MovieGenres.Select(mg => mg.Genre!.GenreName).ToList()  
 )).AsNoTracking().FirstOrDefaultAsync();  
  
 if (movie is null)  
 {  
 return Result.*Failure*<GetMovieByIdResponse, IError>(new NotFoundError($"Фильм c Id: {id} не найден!"));  
 }  
   
 return Result.*Success*<GetMovieByIdResponse, IError>(movie);  
 }  
  
 public async Task<Result<Movie, IError>> DeleteMovieByIdAsync(Guid id)  
 {  
 var movie = await \_context.Movies  
 .Include(x => x.MovieFiles)  
 .FirstOrDefaultAsync(x => x.MovieId == id);  
 if (movie is null)  
 {  
 return Result.*Failure*<Movie, IError>(new NotFoundError($"Фильм с Id: {id} не найлен"));  
 }  
  
 var movieFiles = movie.MovieFiles.Select(x => x.FilePath).ToList();  
 movieFiles.Add(movie.PosterFilePath);  
 \_fileHelper.DeleteFilesByPath(movieFiles);  
   
 \_context.Movies.Remove(movie);  
 await \_context.SaveChangesAsync();  
 return Result.*Success*<Movie, IError>(movie);  
 }  
  
 public async Task<Result<Movie, IError>> UpdateMovieByIdAsync(UpdateMovieRequest request)  
 {  
 var movie = await \_context.Movies  
 .Include(m => m.MovieGenres)  
 .Include(m => m.MovieDirectors)  
 .Include(m => m.MovieFiles)  
 .Where(m => m.MovieId == request.MovieId)  
 .FirstOrDefaultAsync();  
  
 if (movie is null)  
 {  
 return Result.*Failure*<Movie, IError>(new NotFoundError($"Фильм с Id: {request.MovieId} не найден"));  
 }  
  
 movie.MovieName = request.MovieName;  
 movie.MovieDescription = request.MovieDescription;  
 movie.ReleaseDate = request.ReleaseDate;  
 movie.MovieTypeId = request.MovieType;  
   
 *// Удаление жанров* if (request.RemovedGenresId is not null)  
 {  
 movie.MovieGenres.RemoveAll(mg => request.RemovedGenresId.Contains(mg.GenreId));  
 }  
   
 *// Добавление жанров* if (request.AddedGenresId is not null)  
 {   
 movie.MovieGenres.AddRange(request.AddedGenresId.Select(g => new MovieGenre()  
 {  
 MovieId = request.MovieId,  
 GenreId = g  
 }));  
 }  
   
 *// Удаление режиссеров* if (request.RemovedDirectorsId is not null)  
 {  
 movie.MovieDirectors.RemoveAll(md=> request.RemovedDirectorsId.Contains(md.DirectorId));  
 }  
   
 *// Добавление режиссеров* if (request.AddedDirectorsId is not null)  
 {  
 movie.MovieDirectors.AddRange(request.AddedDirectorsId.Select(d=> new MovieDirector()  
 {  
 MovieId = request.MovieId,  
 DirectorId = d  
 }));  
 }  
  
 \_context.Update(movie);  
 await \_context.SaveChangesAsync();  
  
  
 return Result.*Success*<Movie, IError>(movie);  
 }  
   
 */// <summary>  
 /// Добавление фильма в избранные пользователем  
 /// </summary>  
 /// <param name="request">Данные для добавления</param>  
 /// <returns>Результат операции</returns>* public async Task<Result> AddMovieToFavoritesAsync(MovieToFavoriteRequest request)  
 {  
 await \_context.FavoriteMoviesUsers.AddAsync(new FavoriteMoviesUsers()  
 {  
 UserId = request.userId,  
 MovieId = request.movieId  
 });  
  
 await \_context.SaveChangesAsync();  
  
 return Result.*Success*();  
 }  
   
 */// <summary>  
 /// Удаление фильма из избранных у пользователя  
 /// </summary>  
 /// <param name="request">Данные для добавления</param>  
 /// <returns>Результат операции</returns>* public async Task<Result<FavoriteMoviesUsers, IError>> DeleteMovieFromFavoritesAsync(MovieToFavoriteRequest request)  
 {  
 var existData = await \_context.FavoriteMoviesUsers  
 .Where(fum => fum.UserId == request.userId && fum.MovieId == request.movieId).FirstOrDefaultAsync();  
 if (existData is null)  
 {  
 return Result.*Failure*<FavoriteMoviesUsers, IError>(new NotFoundError("Запись не найдена"));  
 }  
  
 \_context.FavoriteMoviesUsers.Remove(existData);  
 await \_context.SaveChangesAsync();  
 return Result.*Success*<FavoriteMoviesUsers, IError>(existData);  
 }  
}

Листинг2 Program.cs Конфигурация всего приложения

using AuthService.Infrastructure;  
using Microsoft.AspNetCore.CookiePolicy;  
using Microsoft.AspNetCore.Http.Features;  
using Microsoft.Extensions.FileProviders;  
using UserService.Infrastructure.Extensions;  
using UserSevice.Host.Extensions;  
  
var builder = WebApplication.*CreateBuilder*(args);  
  
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");  
  
builder.Services.Configure<JwtOptions>(builder.Configuration.GetSection(nameof(JwtOptions)));  
builder.Services.AddBusinessLogic(builder.Configuration, connectionString);  
builder.WebHost.ConfigureKestrel(options => options.Limits.MaxRequestBodySize = long.*MaxValue*);  
const string *allowCorsPolicy* = "allowCorsPolicy";  
builder.Services.AddCors(options =>  
{  
 options.AddPolicy(name: *allowCorsPolicy*, policyBuilder =>  
 {  
 policyBuilder.WithOrigins("https://localhost:3000");  
 policyBuilder.AllowAnyHeader();  
 policyBuilder.AllowAnyMethod();  
 policyBuilder.AllowCredentials();  
 } );  
});  
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen();  
builder.Services.Configure<FormOptions>(options =>  
{  
 options.MultipartBodyLengthLimit = long.*MaxValue*;  
});  
  
var app = builder.Build();  
var staticFilePath = Path.*Combine*(AppContext.BaseDirectory, @"C:\Diplom\data\");  
app.UseStaticFiles(new StaticFileOptions  
{  
 FileProvider = new PhysicalFileProvider(staticFilePath),  
 RequestPath = "/data"  
});  
app.UseCors(*allowCorsPolicy*);  
  
app.AddRouting();  
  
if (app.Environment.IsDevelopment())  
{  
 app.UseSwagger();  
 app.UseSwaggerUI(options =>  
 {  
 options.SwaggerEndpoint("/swagger/v1/swagger.json", "v1");  
 options.RoutePrefix = string.*Empty*;  
 });  
}  
  
app.UseCookiePolicy(new CookiePolicyOptions  
{  
 HttpOnly = HttpOnlyPolicy.*Always*,  
 MinimumSameSitePolicy = SameSiteMode.*Strict*,  
 Secure = CookieSecurePolicy.*Always*,  
   
});  
  
app.UseAuthentication();  
app.UseAuthorization();  
  
app.Run();