

СОДЕРЖАНИЕ

Введение	3
1 Аналитический обзор.....	4
1.1 Анализ предметной области	4
1.2 Обзор существующих аналогов.....	4
1.3 Обзор технологий.....	7
1.4 Постановка задачи.....	8
2 Проектирование приложения.....	10
2.1 Функциональное проектирование	10
2.2 Информационное обеспечение	12
2.3 Архитектура приложения.....	15
3 Реализация приложения	17
3.1 Описание классов.....	17
3.2 Описание интерфейса пользователя	20
4 Тестирование программного обеспечения	29
Заключение	31
Список использованных источников	32
Приложение А (Листинг программы).....	33

ВВЕДЕНИЕ

Информационные технологии оказывают значительное влияние на человеческую деятельность в различных сферах жизни. Они позволяют людям быстро и эффективно обмениваться информацией, автоматизировать процессы и повышать производительность труда.

Онлайн-сервисы распространения цифрового контента становятся все более популярными, предоставляя пользователям удобный способ получения доступа к музыке, фильмам, книгам и другим видам контента. Такие сервисы позволяют не только быстро и удобно получать доступ к нужному контенту, но и экономить время и деньги, не выходя из дома.

В данном проекте представлен онлайн-сервис распространения цифрового контента, который позволит пользователям получать доступ к широкому выбору контента.

Цель данной курсовой работы – изучить предметную область и разработать приложение для покупки фильмов. В работе будут рассмотрены основные требования к веб-приложениям, анализ рынка и конкурентов.

В результате выполнения данной работы будет создано приложение, которое позволит клиентам быстро и удобно выбирать понравившиеся фильмы, а также просматривать всю необходимую информацию о фильме или составе съемочной и производственной команды.

1 АНАЛИТИЧЕСКИЙ ОБЗОР

1.1 Анализ предметной области

Современный мир стремительно развивается в сторону цифровой экономики, в которой цифровой контент играет ключевую роль. Онлайн сервисы распространения цифрового контента предоставляют возможность создания, хранения и распространения цифрового контента в электронном виде. Это позволяет пользователям получать доступ к контенту из любой точки мира, а также авторам быстро и эффективно распространять свои произведения.

Одной из главных причин необходимости онлайн сервисов распространения цифрового контента является удобство использования. Пользователи могут получать доступ к контенту из любой точки мира, без необходимости покупки физических носителей. Это экономит время и деньги.

Онлайн сервисы позволяют авторам контролировать распространение своих произведений и защищать их от незаконного использования. Это помогает сохранять авторские права.

В целом, онлайн сервисы распространения цифрового контента являются необходимыми в современном мире.

Для разработки онлайн сервиса распространения цифрового контента необходимо четкое понимание предметной области.

В ходе курсового проектирования будет разрабатываться информационная система, которая позволит распространять такой цифровой контент как фильмы.

1.2 Обзор существующих аналогов

В ходе обзора существующих методов решения поставленной задачи были рассмотрены уже готовые решения веб-приложений для онлайн систем для аренды специальной техники. Подобного вида веб-приложения помогают сильно облегчить процесс аренды, поскольку это очень сильно экономит время, а также успешно помогает в ведении бизнеса.

В качестве аналогов были выбраны сайты «Кинопоиск» и «IMDB».

Начнём изучение первого сайта «IMDB». Рассмотрим главную страницу сайта на рисунке 1.1.

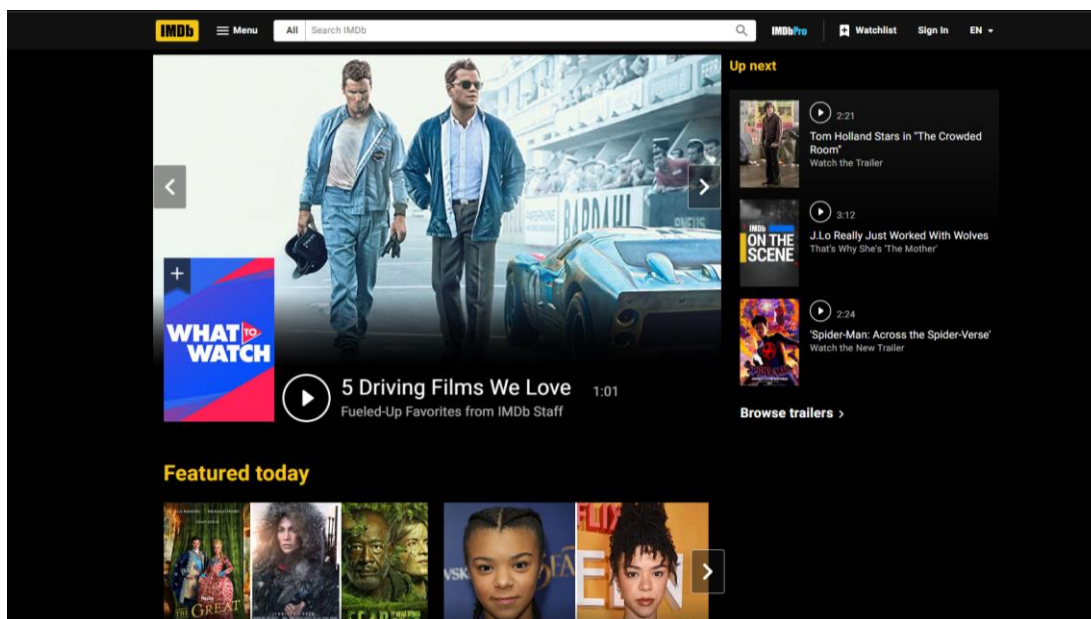


Рисунок 1.1 – Главная страница сайта «IMDB»

Как можно увидеть, на главной странице нам представлены новости мира кино. Нажав на кнопку «Меню», осуществляется переход к окну меню, в котором можно просмотреть всю доступную информацию.

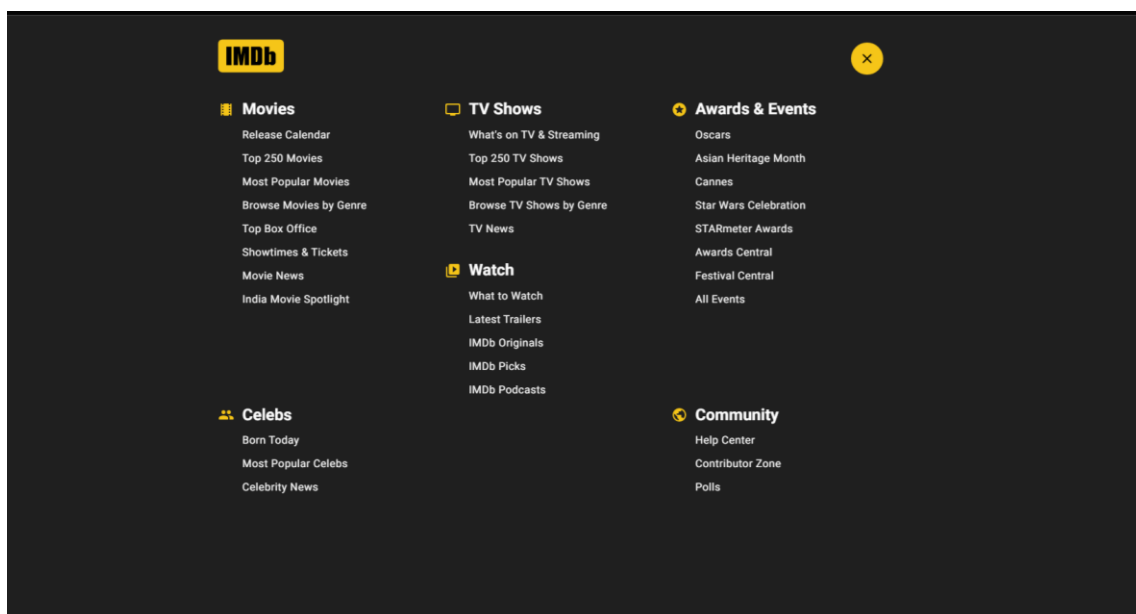


Рисунок 1.2 – Страница «Меню» сайта «IMDB»

Интересующим нас функционалом будет возможность не только просмотреть информацию о фильме, но и о людях, принявших участие в съемках и производстве картины.

Согласно [4], одним из основных преимуществ «IMDB» является удобный и быстрый поиск фильмов и отзывов на них.

Начнём изучение второго сайта «Кинопоиск». Рассмотрим главную страницу на рисунке 1.3.

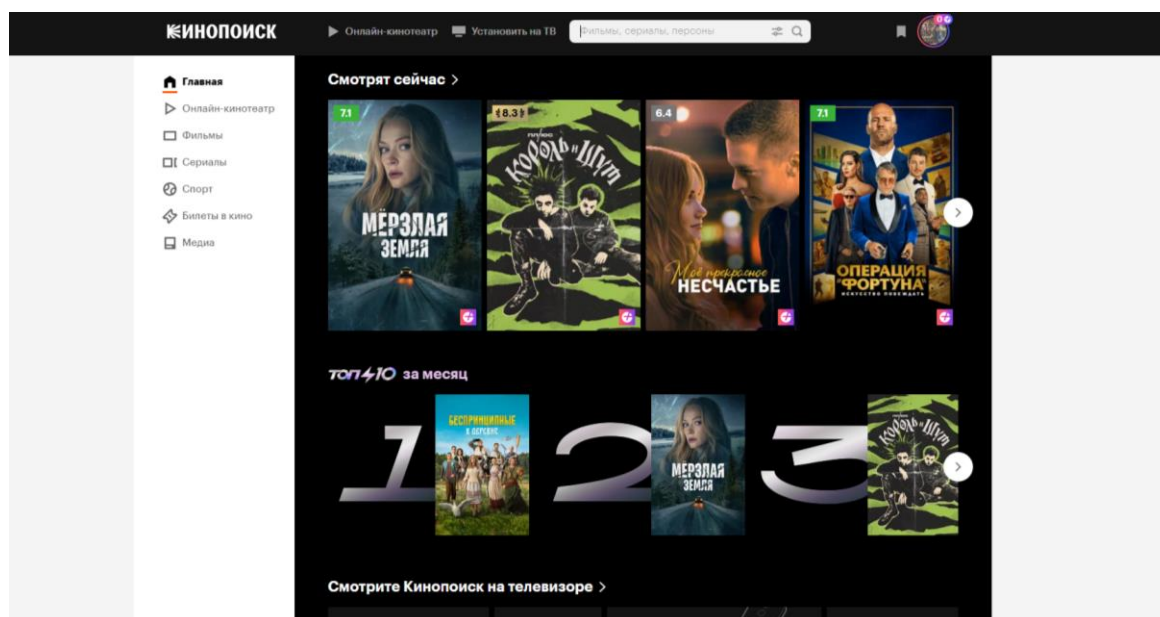


Рисунок 1.3 – Главная страница сайта «Кинопоиск»

На этой странице мы видим список фильмов, которые могут заинтересовать пользователя. Этот функционал будет взят за основу [2]. Открыв информацию о фильме, мы можем наблюдать информацию, которая может заинтересовать пользователя перед просмотром. Вид страницы представлен на рисунке 1.4.

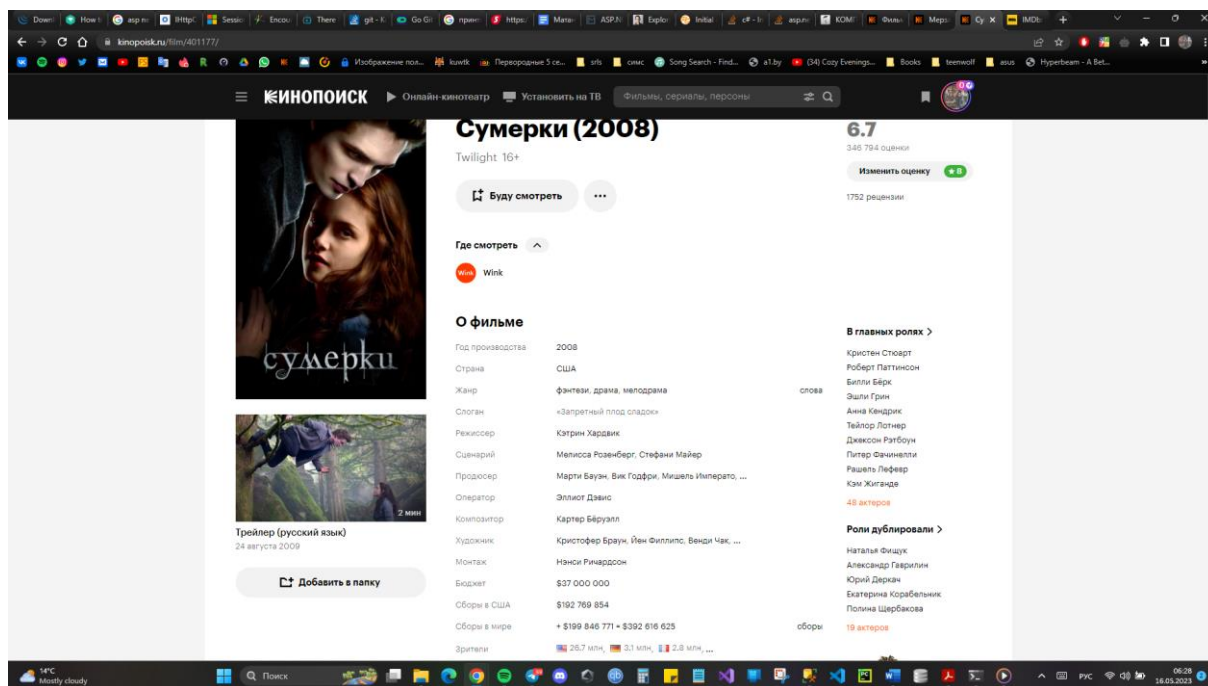


Рисунок 1.4 – Страница фильма

Открыв информацию о сценаристе, мы можем наблюдать информацию. Вид страницы представлен на рисунке 1.5.

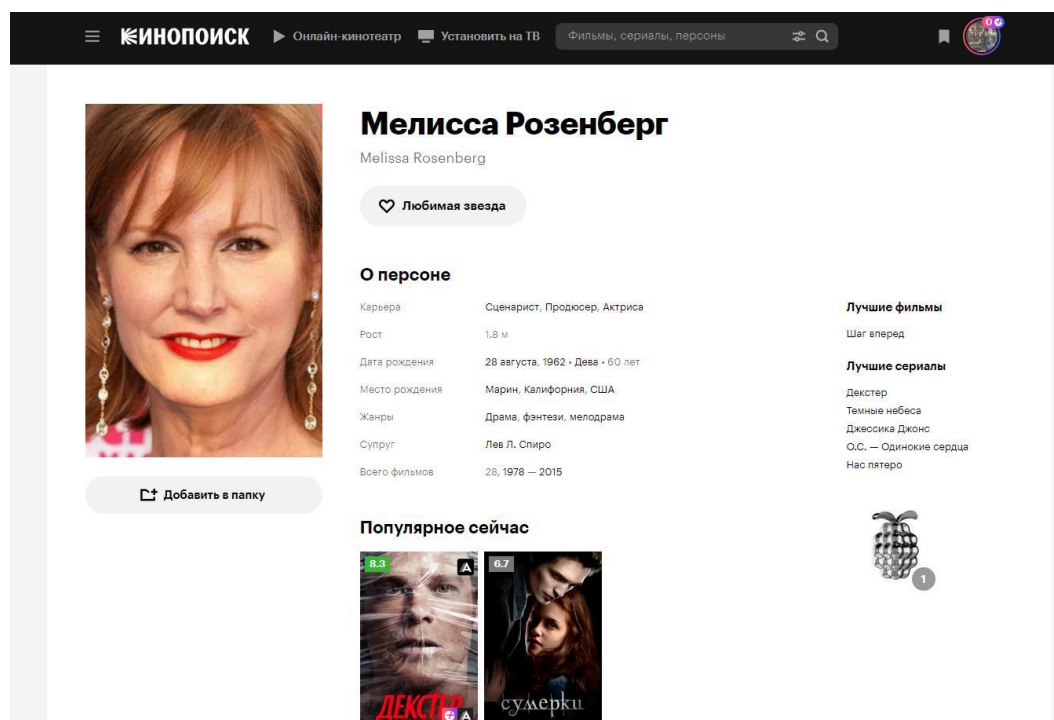


Рисунок 1.5 – Страница сценариста»

Согласно [3], одним из основных преимуществ Кинопоиска является удобный и быстрый поиск фильмов на сайте.

1.3 Обзор технологий

Для решения поставленной задачи в качестве СУБД используется *Ms SQL Server*. Данная СУБД обеспечивает поддержку баз данных очень большого объёма и обработку сложных запросов, а также имеет эффективные алгоритмы для работы с памятью и автоматизированным контролем размера файлов баз данных.

В качестве технологии для разработки веб-приложения используется платформа *ASP.NET*.

ASP.NET является одной из самых популярных платформ для создания веб-приложений. Она была разработана компанией *Microsoft* и является основой для создания многих веб-сайтов и приложений, которые мы используем каждый день.

ASP.NET предоставляет разработчикам мощный инструментальный для создания профессиональных веб-приложений. Он использует языки программирования, такие как *C#* и *Visual Basic*, а также другие технологии *Microsoft*, такие как *SQL Server* и *Windows Server*.

Одной из особенностей *ASP.NET* является его модель программирования на основе событий. В этой модели веб-страницы и компоненты реагируют на события, такие как щелчок мыши или изменение текста в поле ввода, вызывая

определенный код. Это позволяет создавать более динамические и интерактивные приложения.

ASP.NET также обеспечивает масштабируемость приложений. Система автоматически управляет выделением ресурсов и распределением нагрузки на серверах, что позволяет обеспечивать высокую производительность даже при большом количестве пользователей [1].

Одним из ключевых компонентов *ASP.NET* является его фреймворк *MVC* (*Model-View-Controller*). Он предоставляет разработчикам гибкую архитектуру, которая позволяет разделить приложение на три отдельных слоя: модель, представление и контроллер. Это упрощает разработку и поддержку приложений.

С помощью *ASP.NET* разработчики могут создавать различные типы приложений, включая веб-сайты, веб-службы, веб-API и даже мобильные приложения.

Одной из главных преимуществ *ASP.NET* является его интеграция с *Visual Studio*, интегрированной средой разработки (*IDE*) от *Microsoft*. *Visual Studio* предоставляет множество инструментов для разработки приложений на *ASP.NET*, включая отладчик, инструменты тестирования и управление версиями.

ASP.NET также поддерживает различные сторонние библиотеки и плагины, которые могут расширить его функциональность. Некоторые из этих библиотек включают в себя *jQuery*, *AngularJS* и *Bootstrap*.

Для доступа к данным будет использована технология *Entity Framework Core*.

Entity Framework (EF) — это технология объектно-ориентированного доступа к данным для *.NET Framework*. Она предоставляет удобный способ работать с базами данных, используя объекты *.NET*, а не *SQL*-запросы. *EF* позволяет разработчикам работать с данными на более высоком уровне абстракции, что упрощает процесс разработки и обслуживания приложений.

Одной из ключевых концепций *Entity Framework* является *ORM* (*Object-Relational Mapping*). *ORM* — это технология, которая позволяет разработчикам работать с базой данных, используя объекты *.NET*. *EF* предоставляет *ORM*-функциональность, которая позволяет создавать и обновлять таблицы в базе данных, а также выполнять запросы к данным. Она также позволяет использовать *LINQ* (*Language Integrated Query*) для выполнения запросов к данным [5].

EF также предоставляет *Code First* подход к разработке приложений. *Code First* — это способ создания базы данных, который позволяет разработчикам определять модели данных в коде и затем автоматически создавать базу данных на основе этих моделей. Это упрощает процесс разработки и уменьшает количество кода, необходимого для создания базы данных.

1.4 Постановка задачи

Необходимо разработать веб-приложение, которое содержит в себе следующий функционал:

- содержание информации о фильмах;

- содержание информации об актерах;
- содержание информации о сценаристах;
- содержание информации о продюсерах;
- содержание информации о всех пользователях;
- содержание информации о всех заказах.

На основании произведённого анализа существующих методов, можно сформировать список требований, для разрабатываемой системы:

- приложение не должно иметь лишнего функционала;
- доступ к приложению должен быть удобным и простым.

Программный продукт должен предоставлять:

- справочники, включающие в себя фильмы, актеры, кинотеатры и продюсеры;
- формирование отчетов, статистики;
- необходимо разделение доступа к функционалу приложения по ролям:

Менеджер, Пользователь.

Обзор темы показал, что для работы с приложением необходимо будет определить ролевую политику, где каждой роли будет соответствовать свой функционал.

К функционалу Администратора будет относиться:

- добавление, обновление фильмов;
- добавление, обновление, удаление актеров;
- добавление, обновление, удаление сценаристов;
- добавление, обновление, удаление продюсеров;
- просмотр всех пользователей в системе;
- просмотр всех заказов в системе.

К функционалу Пользователя будет относиться:

- поиск фильма;
- просмотр подробной информации о фильме;
- просмотр информации о сценаристах;
- просмотр информации о продюсерах;
- просмотр информации о сценаристах;
- добавление фильма в корзину;
- покупка фильмов;
- просмотр совершенных заказов.

К функционалу Гостя будет относиться:

- просмотр информации о фильмах;
- просмотр информации о сценаристах;
- просмотр информации о продюсерах;
- просмотр информации о сценаристах;
- регистрация;
- авторизация.

2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

2.1 Функциональное проектирование

Исходя из представленной предметной области были выявлены необходимые роли. Были разработана ролевая политика, включающая в себя Клиента, Администратора и Гостя. Каждый из актеров содержит в себе свой собственный функционал.

Для формирования представления о предметной области используют *UML* диаграммы, которые представлены на рисунках 2.1 – 2.3.

Диаграмма прецедента и актёра «Гость», предоставлена на рисунке 2.1.

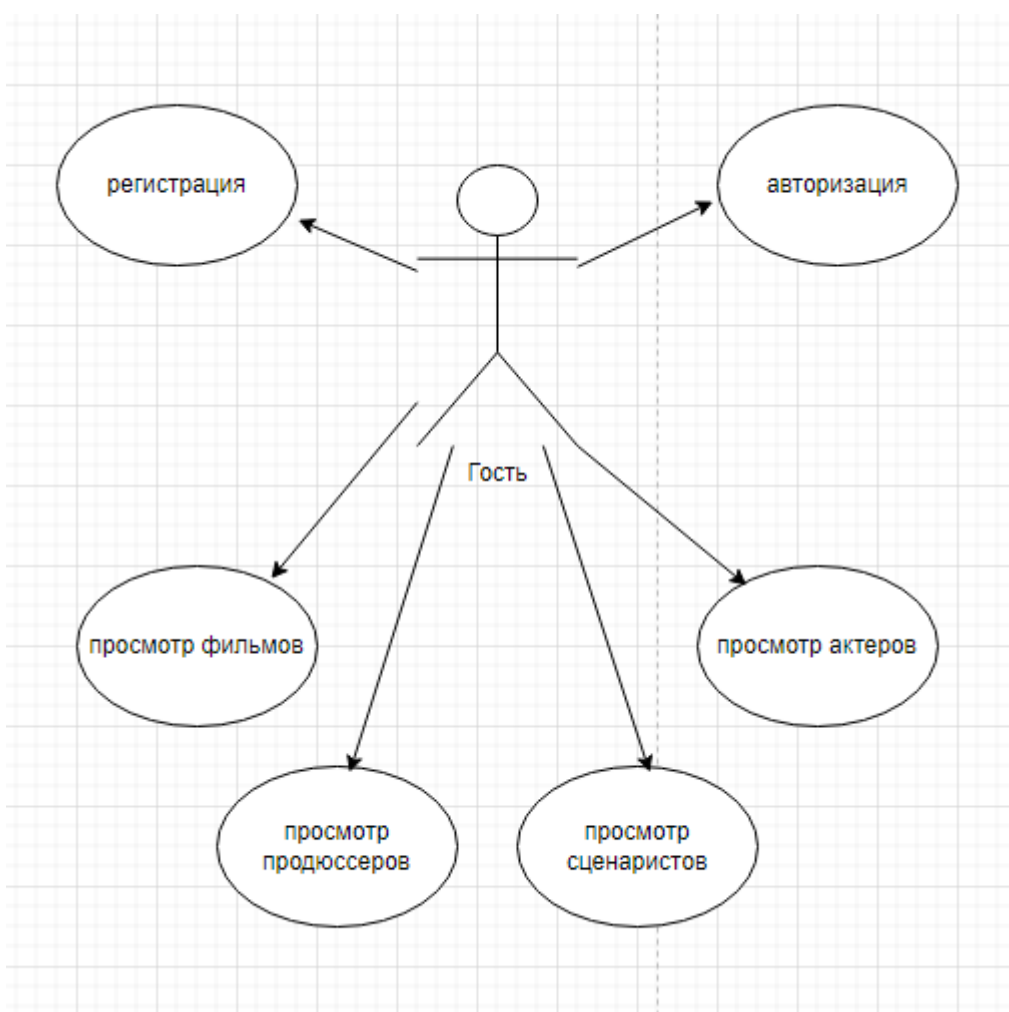


Рисунок 2.1 – Диаграмма гостя

На данной диаграмме представлен весь функционал веб-приложения, который доступен для гостя. Гости в приложении еще не имеют свой личный профиль, но могут его создать, имеют доступ ко всей информации сайта, но не могут делать покупки.

Диаграмма прецедента и актёра «Клиент», предоставлена на рисунке 2.2.

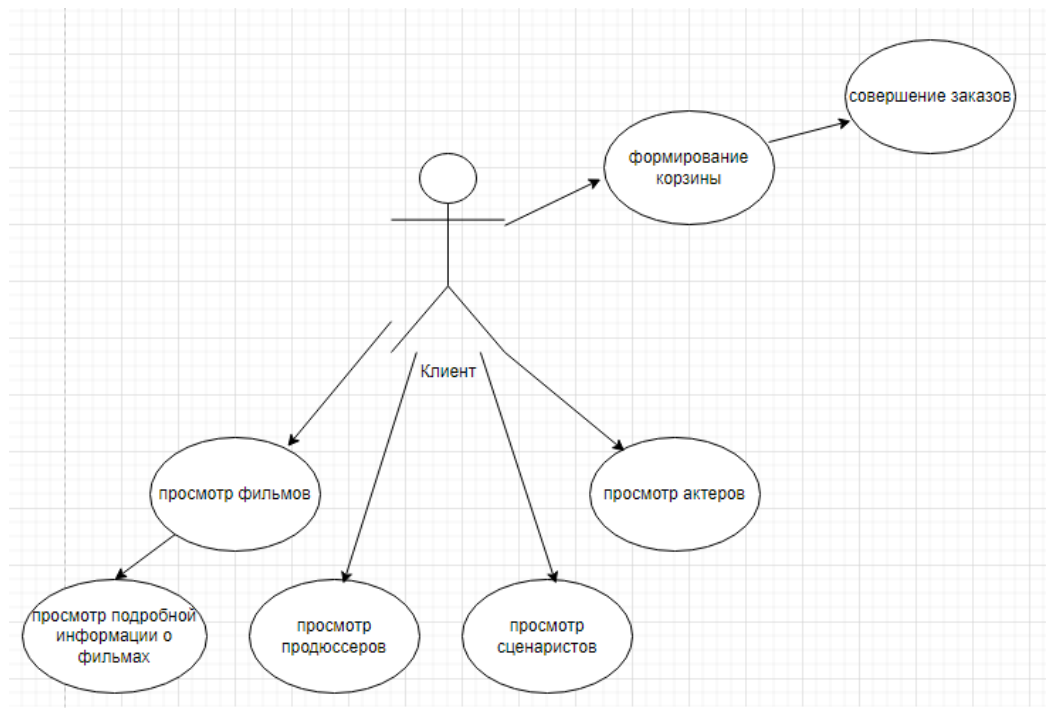


Рисунок 2.2 – Диаграмма клиента

На данной диаграмме представлен весь функционал веб-приложения, который доступен для клиента. Клиенты могут просматривать всю информацию на сайте, а также клиенты обладают возможностью совершения покупок.

Диаграмма прецедента и актёра «Администратор», предоставлена на рисунке 2.3.

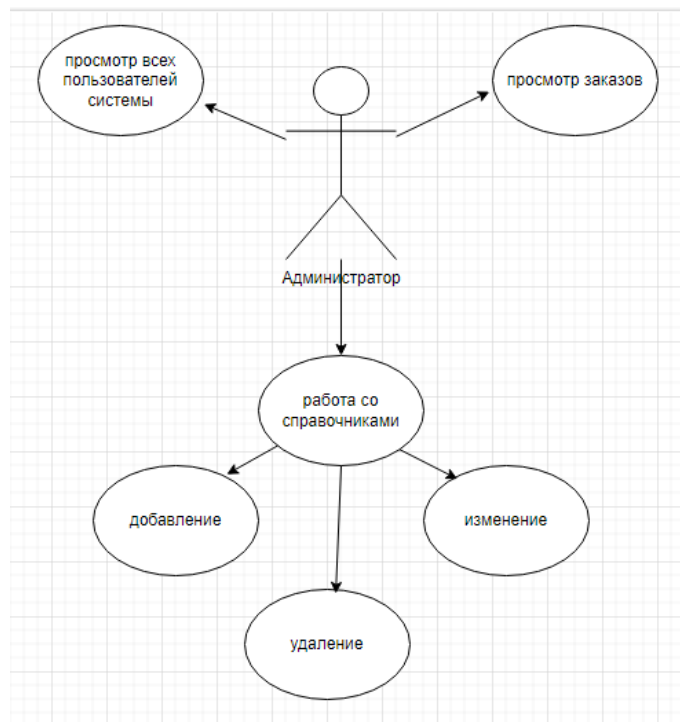


Рисунок 2.3 – Диаграмма администратора

На данной диаграмме представлен весь функционал веб-приложения, который доступен для администратора. Администратор может просматривать и пользователей системы, обновлять информацию всех справочников.

2.2 Информационное обеспечение

На основе результатов анализа предметной области, результатов концептуального и логического проектирований, была создана физическая модель базы данных со следующими таблицами:

- *Actors*;
- *Actors_Movies*;
- *Movies*;
- *Orders*;
- *OrderItems*;
- *Producers*;
- *Screenwriters*;
- *ShoppingCartItems*.

Данная модель имеет 8 сущностей. Рассмотрим таблицы, которые необходимы для полноценного функционирования подсистемы. Описание полей таблиц приведено в таблицах 2.1-2.8.

Описание атрибутов таблицы *Actors* представлено в таблице 2.1.

Таблица 2.1 – Сущность *Actors*

№	Имя	Ключевое поле, тип ключа	Тип данных
1	<i>Id</i>	Первичный	<i>int</i>
2	<i>ProfilePictureURL</i>	–	<i>nvarchar</i>
3	<i>FullName</i>	–	<i>nvarchar</i>
4	<i>DateOfBirth</i>	–	<i>datetime2</i>
5	<i>PlaceOfBirth</i>	–	<i>nvarchar</i>

Описание атрибутов таблицы *Actors_Movies* представлено в таблице 2.2.

Таблица 2.2 – Сущность *Actors_Movies*

№	Имя	Ключевое поле, тип ключа	Тип данных
1	<i>MovieId</i>	Первичный, Внешний	<i>int</i>
2	<i>ActorId</i>	Первичный, Внешний	<i>int</i>

Описание атрибутов таблицы *Producers* представлено в таблице 2.3.

Таблица 2.3 – Сущность *Producers*

№	Имя	Ключевое поле, тип ключа	Тип данных
1	<i>Id</i>	Первичный	<i>int</i>
2	<i>ProfilePictureURL</i>	—	<i>nvarchar</i>
3	<i>FullName</i>	—	<i>nvarchar</i>
4	<i>DateOfBirth</i>	—	<i>datetime2</i>
5	<i>PlaceOfBirth</i>	—	<i>nvarchar</i>

Описание атрибутов таблицы *Movies* представлено в таблице 2.4.

Таблица 2.4 – Сущность *Movies*

№	Имя	Ключевое поле, тип ключа	Тип данных
1	<i>Id</i>	Первичный	<i>int</i>
2	<i>Name</i>	—	<i>nvarchar</i>
3	<i>Description</i>	—	<i>nvarchar</i>
4	<i>ReleaseDate</i>	—	<i>datetime2</i>
5	<i>Country</i>	—	<i>nvarchar</i>
6	<i>Tagline</i>	—	<i>nvarchar</i>
7	<i>Budget</i>	—	<i>float</i>
8	<i>Price</i>	—	<i>float</i>
9	<i>Duration</i>	—	<i>float</i>
10	<i>RatingSystem</i>	—	<i>int</i>
11	<i>ImageURL</i>	—	<i>nvarchar</i>
12	<i>MovieCategory</i>	—	<i>int</i>
13	<i>ProducerId</i>	Внешний	<i>int</i>
14	<i>ScreenwriterId</i>	Внешний	<i>int</i>

Описание атрибутов таблицы *Screenwriters* представлено в таблице 2.5.

Таблица 2.5 – Сущность *Screenwriters*

№	Имя	Ключевое поле, тип ключа	Тип данных
1	2	3	4
1	<i>Id</i>	Первичный	<i>int</i>
2	<i>ProfilePictureURL</i>	—	<i>nvarchar</i>

Продолжение таблицы 2.5

1	2	3	4
3	<i>FullName</i>	—	<i>nvarchar</i>
4	<i>DateOfBirth</i>	—	<i>datetime2</i>
5	<i>PlaceOfBirth</i>	—	<i>nvarchar</i>

Описание атрибутов таблицы *ShoppingCartItems* представлено в таблице 2.6.

Таблица 2.6 – Сущность *ShoppingCartItems*

№	Имя	Ключевое поле, тип ключа	Тип данных
1	<i>Id</i>	Первичный	<i>int</i>
2	<i>MovieId</i>	Внешний	<i>int</i>
3	<i>Amount</i>	—	<i>int</i>

Описание атрибутов таблицы *OrderItems* представлено в таблице 2.7.

Таблица 2.7 – Сущность *OrderItems*

№	Имя	Ключевое поле, тип ключа	Тип данных
1	<i>Id</i>	Первичный	<i>int</i>
2	<i>Amount</i>	—	<i>int</i>
3	<i>Price</i>	—	<i>float</i>
	<i>MovieId</i>	Внешний	<i>int</i>
	<i>OrderId</i>	Внешний	<i>int</i>

Описание атрибутов таблицы *Orders* представлено в таблице 2.8.

Таблица 2.8 – Сущность *Orders*

№	Имя	Ключевое поле, тип ключа	Тип данных
1	<i>Id</i>	Первичный	<i>int</i>
2	<i>Email</i>	—	<i>nvarchar</i>
3	<i>UserId</i>	Внешний	<i>nvarchar</i>

На рисунке 2.4 представлена структура базы данных.

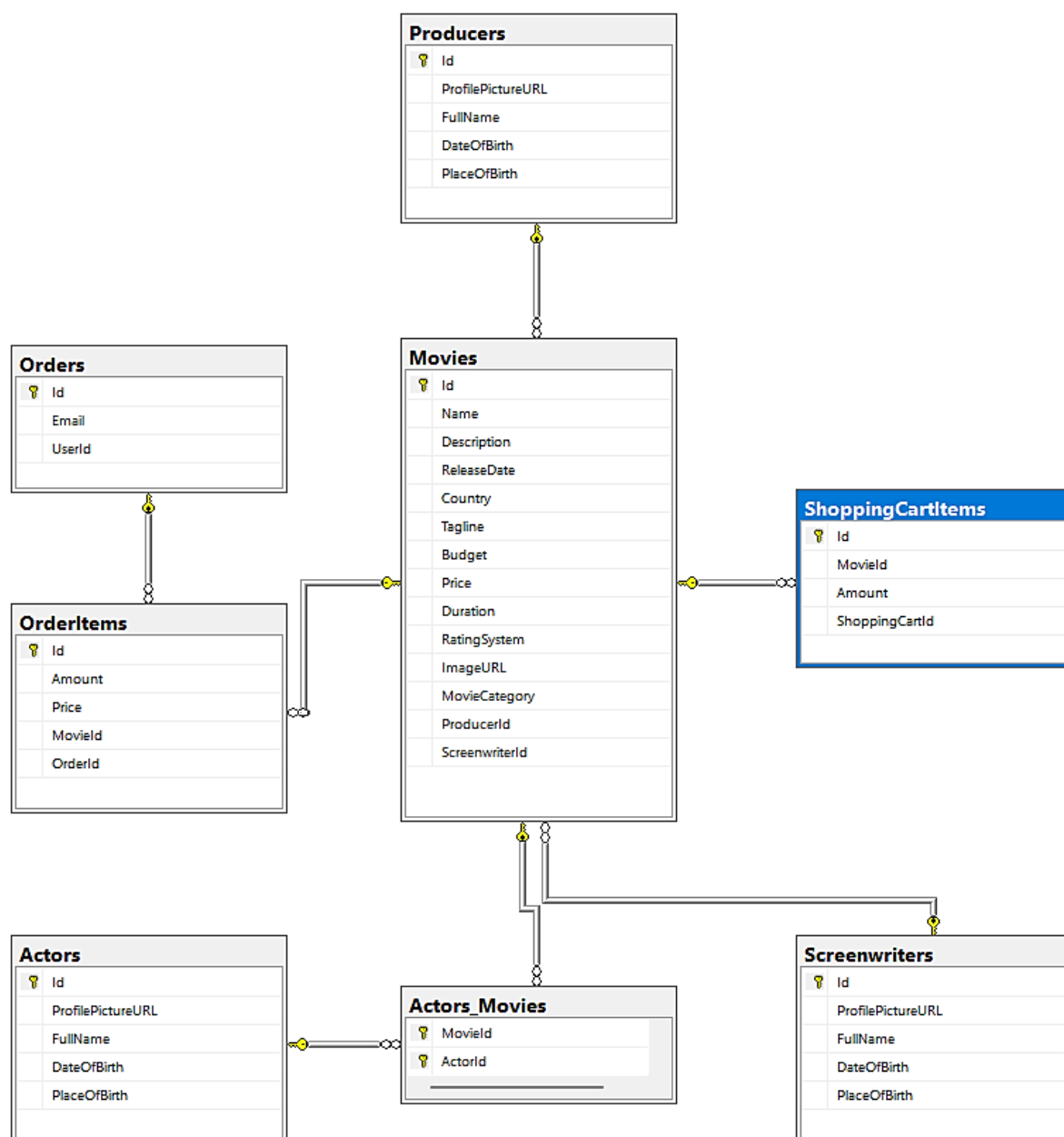


Рисунок 2.4 – Структура Базы данных

2.3 Архитектура приложения

В качестве архитектуры было выбрано трехслойное приложение. Которое состоит из трех уровней.

Data Access layer (уровень доступа к данным): хранит модели, описывающие используемые сущности, также здесь размещаются специфичные классы для работы с разными технологиями доступа к данным, например, класс контекста данных *Entity Framework*. Здесь также хранятся репозитории, через которые уровень бизнес-логики взаимодействует с базой данных.

Business layer (уровень бизнес-логики): содержит набор компонентов, которые отвечают за обработку полученных от уровня представлений данных,

реализует всю необходимую логику приложения, все вычисления, взаимодействует с базой данных и передает уровню представления результат обработки.

Presentation layer (уровень представления): это тот уровень, с которым непосредственно взаимодействует пользователь. Этот уровень включает компоненты пользовательского интерфейса, механизм получения ввода от пользователя. Применительно к *asp.net mvc* на данном уровне расположены представления и все те компоненты, который составляют пользовательский интерфейс (стили, статичные страницы *html*, *javascript*), а также модели представлений, контроллеры, объекты контекста запроса.

Этот подход обладает рядом преимуществ:

- позволяет полностью контролировать отображаемый *HTML*;
- обеспечивает чистое разделение проблем;
- включает *Test Driven Development (TDD)*;
- простая интеграция с фреймворками *JavaScript*;
- следуя дизайну безгражданности веб-сайта;
- *URL*-адреса *RESTful*, которые позволяют *SEO*;
- нет событий *ViewState* и *PostBack*.

А также нельзя не отметить возможность быстрой разработки больших проектов в группе.

Уровень доступа к данным не зависит от других уровней, уровень бизнес-логики зависит от уровня доступа к данным, а уровень представления – от уровня бизнес-логики.

Таким образом приложение будет иметь следующий набор проектов, предоставленный на рисунке 2.5.

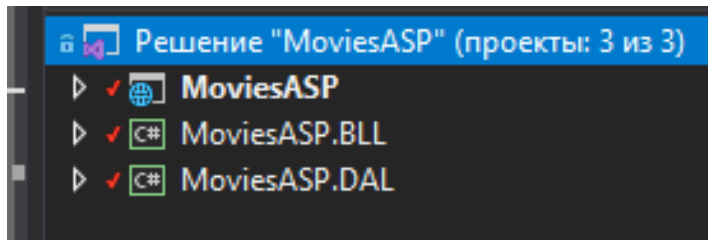


Рисунок 2.5 – Архитектура веб-приложения

3 РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

3.1 Описание классов

Описание классов будет начинаться со слоя *DAL*, описанный в приложении А, потом будет описываться слой *BLL*, а после будет описываться слой *UI*.

Для работы с данными в проекте *DAL* также реализуется паттерн репозиторий, который позволяет абстрагироваться от деталей реализации доступа к данным и предоставляет более высокий уровень абстракции для бизнес-логики приложения.

В целом, проект *DAL* является важной частью архитектуры приложения, так как обеспечивает надежный и эффективный доступ к данным, который необходим для работы других слоев приложения.

В папке *Configuration* находится класс, который помогает настроить зависимости. Они необходимы для работы со слоем *DAL*.

В папке *Context* находится контекст базы данных *EF Core*. Он необходим для подключения к базе данных и работы с ней с помощью *EF Core*.

В папке *Base* находится базовый репозиторий и интерфейсы для репозитория и моделей. Они необходимы для реализации более гибкого функционала репозитория.

Базовый интерфейс «*IEntityBaseRepository*» приведен в Приложении А. Описание методов интерфейса приведено в таблице 3.1. Он имеет все основные операции, которые свойственны репозиторию.

Таблица 3.1 – Таблица, описывающая базовый интерфейс репозитория

Возвращаемый тип	Имя	Описание
<i>Task<IEnumerable<T>></i>	<i>GetAllAsync</i>	Метод, отвечающий за получение всех объектов
<i>Task<T></i>	<i>GetByIdAsync</i>	Метод, отвечающий за получение объекта
<i>Task</i>	<i>DeleteAsync</i>	Метод, отвечающий за удаление данных
<i>Task</i>	<i>AddAsync</i>	Метод, отвечающий за добавление данных
<i>Task</i>	<i>UpdateAsync</i>	Метод, отвечающий за обновление данных

Все классы, которые реализуют этот интерфейс используют эти методы для работы с базой данных. Также, данные классы используются в бизнес-слое.

Не все репозитории должны выполнять основные операции над сущностями. В некоторых случаях может быть достаточно реализации только

определенных методов, например, для получения количества элементов, поиска элементов по заданным критериям, выполнения специализированных запросов.

Таким образом, реализация только необходимых методов может повысить эффективность работы репозитория и улучшить читаемость и поддерживаемость кода.

Слой *BLL* реализует набор классов для работы с бизнес логикой приложения. На данном слое находятся *DTO* модели, сервисы и профили.

Содержимое *DTO* классов почти полностью соответствуют классам моделей предметной области. *DTO* классы используются для передачи данных из классов-сервисов на слой представления.

В каталоге *Profiles* проекта *BLL*, используется библиотека *AutoMapper* для преобразования классов-моделей в классы-*DTO* и обратно.

В каталоге *Services* находятся классы, которые описывают интерфейсы сервисов и классы, которые их реализуют.

Интерфейсы приведены в Приложении А.

Рассмотрим интерфейс «*IMoviesService*». Описание методов интерфейса приведено в таблице 3.2.

Таблица 3.2 – Таблица, описывающая «*IMoviesService*»

Возвращаемый тип	Имя	Описание
<i>Task<Movie></i>	<i>GetMovieByIdAsync</i>	Метод, отвечающий за получение объекта
<i>Task<NewMovie-DropdownsVM></i>	<i>GetNewMovieDropdownsValues</i>	Метод, отвечающий за получение данных, используемых для связи таблиц
<i>Task</i>	<i>AddNewMovieAsync</i>	Метод, отвечающий за добавление данных
<i>Task</i>	<i>UpdateMovieAsync</i>	Метод, отвечающий за обновление данных
<i>Task<IEnumerable<Movie>></i>	<i>GetMovies</i>	Метод, отвечающий за получение всех объектов

В папке *Controllers* находятся классы-контроллеры, необходимые для функционирующей логики приложения на стороне веб-сервера. Контроллеры в приложении отвечают за обработку запросов пользователя и взаимодействие с моделью и сервисами для формирования ответа пользователю.

В таблице 3.3 представлены основные контроллеры слоя представления.

Таблица 3.3 – Таблица, описывающая контроллеры слоя отображения

Название контроллера	Описание
<i>AccountController</i>	Данный контроллер отвечает за обработку запросов, связанных с аккаунтами и соответствующими страницами
<i>ActorsController</i>	Данный контроллер отвечает за обработку запросов, связанных с актерами
<i>MoviesController</i>	Данный контроллер отвечает за обработку запросов, связанных с фильмами
<i>OrdersController</i>	Данный контроллер отвечает за обработку запросов, связанных с заказами
<i>ProducersController</i>	Данный контроллер отвечает за обработку запросов, связанных с продюсерами
<i>ScreenwritersController</i>	Данный контроллер отвечает за обработку запросов, связанных со сценаристами

Описанные модули и классы являются основой приложения и определяют его функциональность, а их правильная организация обеспечивает эффективное взаимодействие с пользователем и обеспечивает удобство использования приложения.

В любом типе проектов *ASP.NET Core*, как и в проекте консольного приложения, мы можем найти файл *Program.cs*, в котором определен одноименный класс *Program*, с которого начинается выполнение приложения.

Чтобы запустить приложение *ASP.NET Core*, необходим объект *IHost*, в рамках которого разворачивается веб-приложение. В классе *Program* происходит создание объекта этого класса.

При создании объекта *IHost* указывается класс *Startup*. В нём определены 2 метода: *ConfigureServices* и *Configure*. В первом подключаются зависимости, во втором настраивается конвейер обработки входящих *HTTP*-запросов.

Папка *Views* в *ASP.NET* используется для хранения файлов представлений (*View*), которые отображают данные, полученные из модели приложения. В этой папке создаются подпапки для каждого контроллера, в которых хранятся файлы представлений, соответствующие каждому действию контроллера. При запросе определенного действия контроллера, *ASP.NET* ищет соответствующий файл представления в папке *Views* и отображает его содержимое пользователю. Таким образом, папка *Views* играет важную роль в разработке веб-приложений на *ASP.NET*, позволяя разделить логику приложения, модель и представление.

Представления представляют собой файлы, содержащие *html*-разметку со вставками кода на языке *C#*.

3.2 Описание интерфейса пользователя

Поэтому, при разработке приложения, следует уделить достаточное внимание пользовательскому интерфейсу. Необходимо создавать удобный, интуитивно понятный интерфейс, который позволит пользователям легко находить нужные функции и взаимодействовать с приложением. В результате, это повысит удовлетворенность пользователей и улучшит репутацию приложения на рынке.

Как уже было установлено ранее, в данном веб-приложении были выделены 3 роли:

- гость;
- администратор;
- клиент.

Для гостя сайт носит справочный характер, он может просмотреть Фильмы, Актеров, Сценаристов и Продюсеров.

Главная страница, отображаемая гостю представлена на рисунке 3.7.

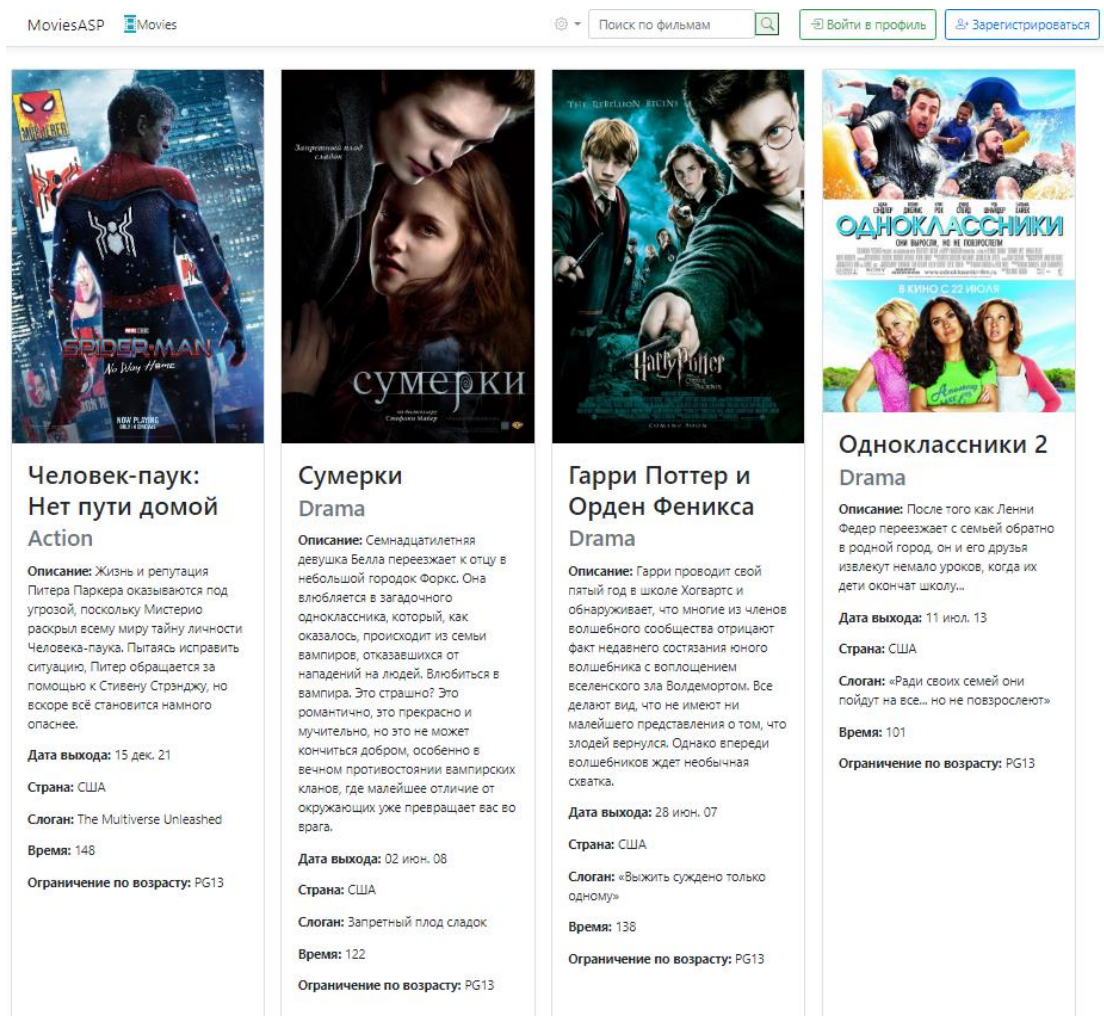


Рисунок 3.7 – Главная страница сайта для гостя

Страница с актерами представлена на рисунке 3.8.

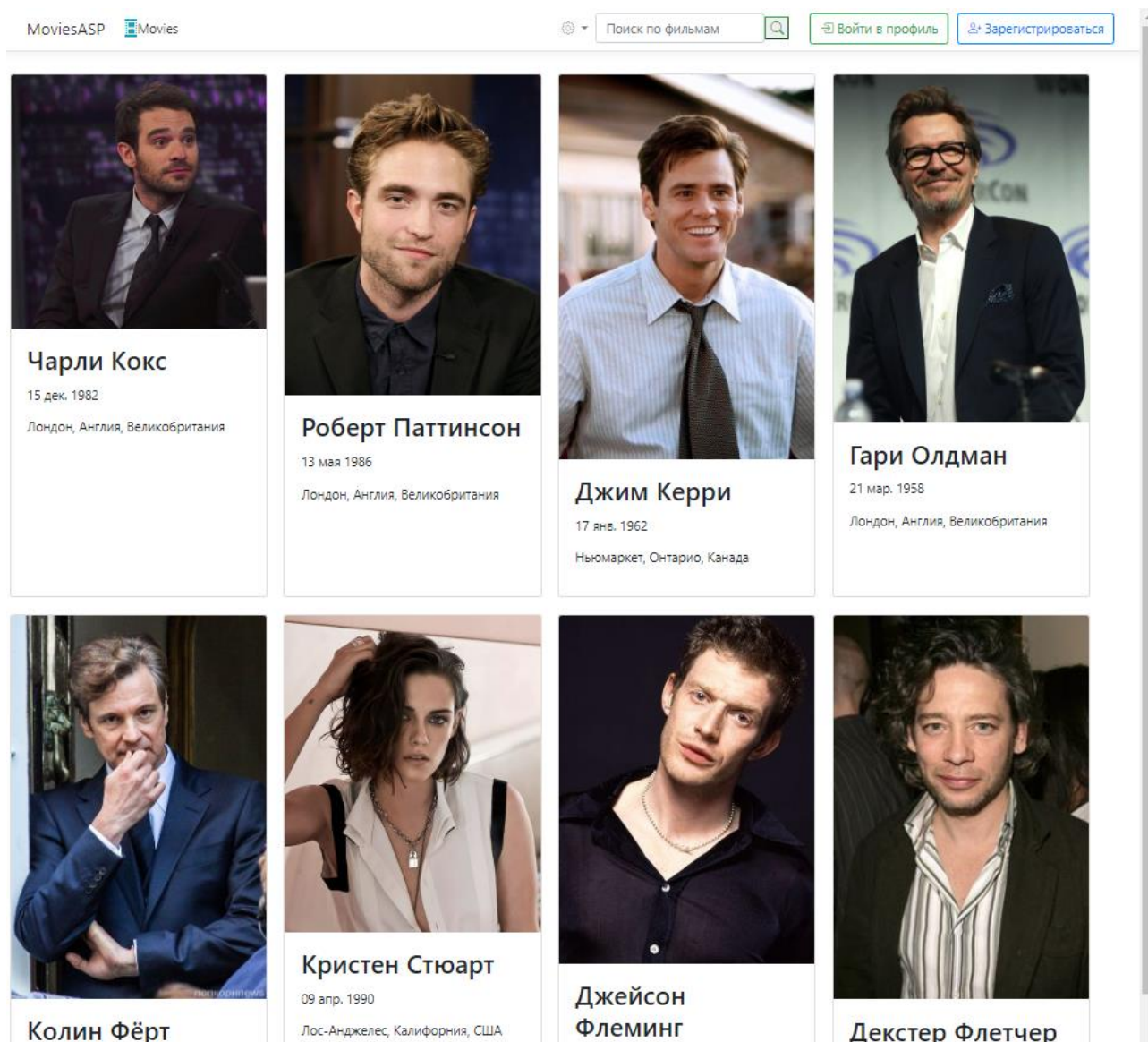


Рисунок 3.8 – Страница актеров

Также на панели присутствуют кнопки позволяющие передвигаться по сайту. Вид панели представлен на рисунке 3.9.

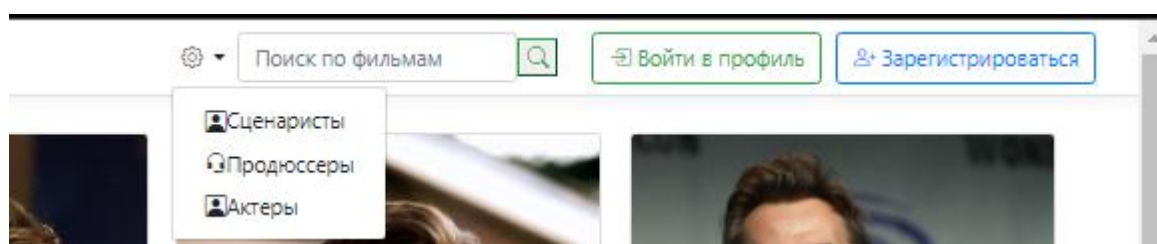


Рисунок 3.9 – Панель доступа

Нажав на кнопку, «Зарегистрироваться» откроется страница регистрации, где гость может зарегистрироваться в системе как клиент.

Страница регистрации представлена на рисунке 3.10.

The screenshot shows a registration form with the following elements:

- Имя** (Name): Input field containing "Белла Свон".
- Email**: Input field containing "bellaaaswan".
- Password**: Input field with masked characters (dots).
- Подтвердите пароль** (Confirm password): Input field with masked characters (dots).
- Назад** (Back): Button with a blue border.
- Подтвердить** (Confirm): Button with a green border.

Рисунок 3.10 – Страница регистрации

Если пользователь нажмет на кнопку «Подтвердить», то пользователя перенаправит на страницу, информирующую об успешной регистрации.

Данная страница представлена на рисунке 3.11.

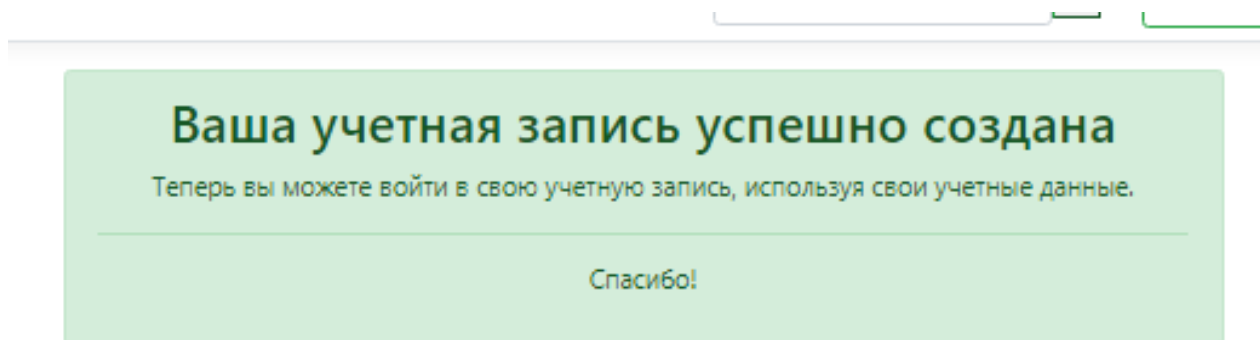


Рисунок 3.11 – Страница успешной регистрации

После создания аккаунта пользователь может войти в него нажав кнопку «Войти в профиль». Почта и пароль были сохранены при регистрации, поэтому пользователя встречают поля, заполненные данными.

Страница входа в аккаунт представлена на рисунке 3.12.

Вход в аккаунт

Email

bellaaaswan

Password

Назад

Войти

Рисунок 3.12 – Страница авторизации

После авторизации клиент может просматривать подробную информацию о фильмах, а также добавлять их в корзину для совершения покупки. Вид главной страницы для клиента представлен на рисунке 3.13.

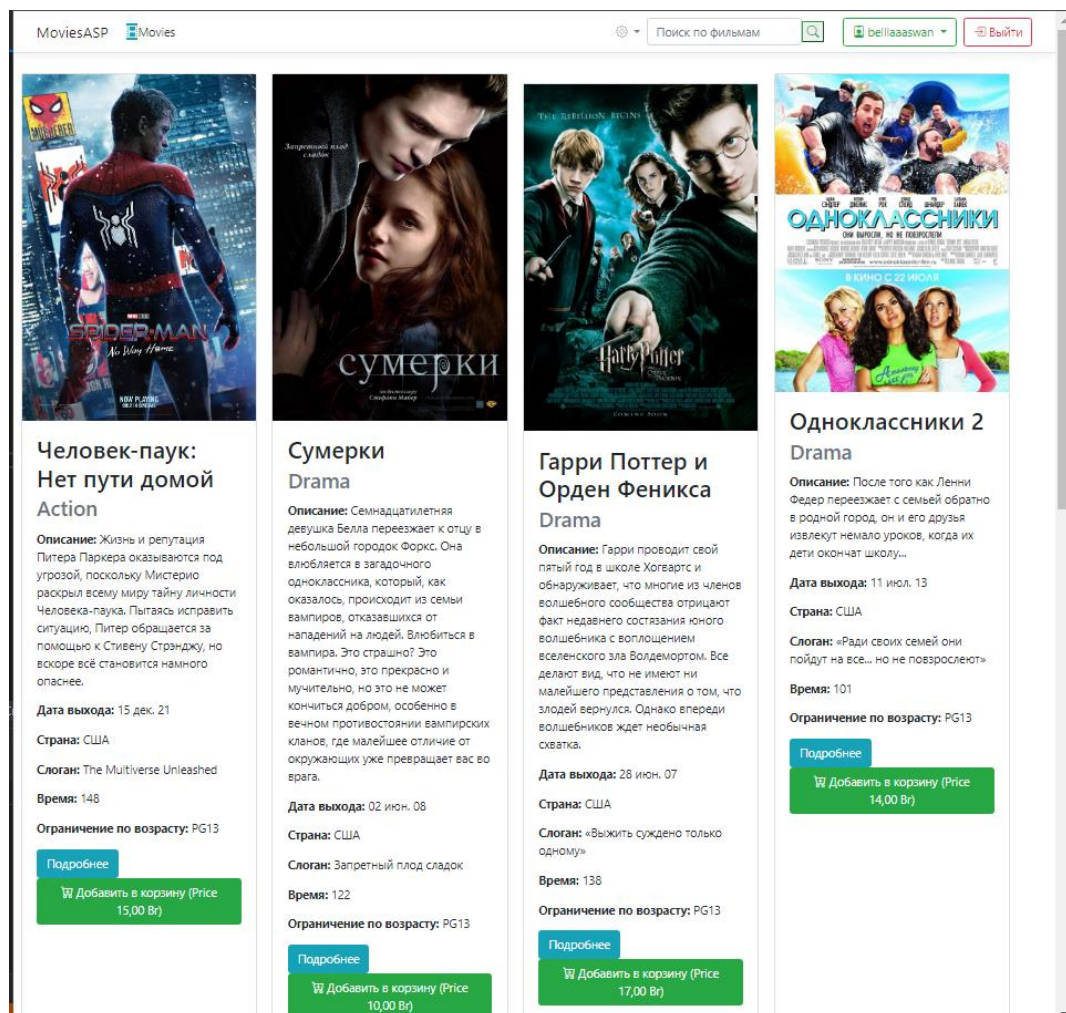


Рисунок 3.13 – Главная страница для клиента

Страница подробной информации представлена на рисунке 3.14.

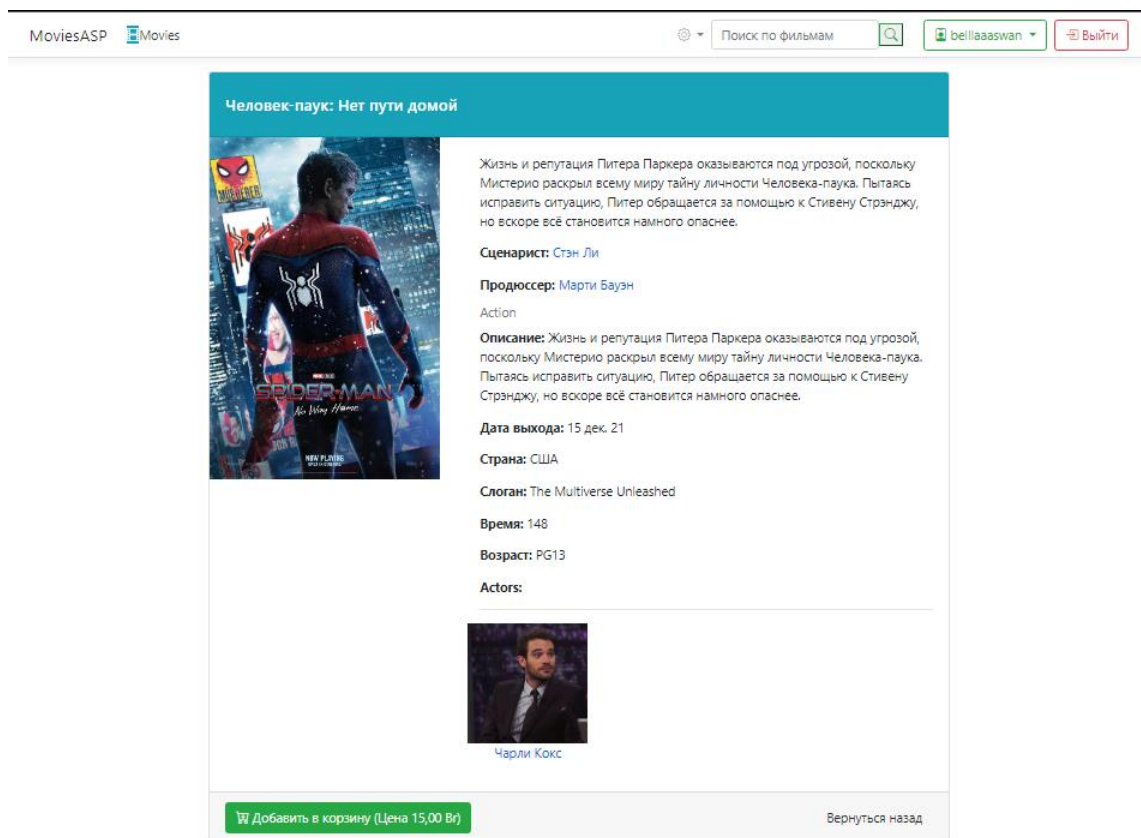


Рисунок 3.14 – Страница подробной информации о фильме

При добавлении в корзину пользователю открывается страница, на которой он может просмотреть или удалить выбранные варианты.

Страница корзины представлена на рисунке 3.15.

Ваша корзина				
Количество	Фильм	Цена	Итого	
1	Человек-паук: Нет пути домой	15,00 Br	15,00 Br	Удалить Добавить
1	Сумерки	10,00 Br	10,00 Br	Удалить Добавить
1	Одноклассники 2	14,00 Br	14,00 Br	Удалить Добавить
Total:			39,00 Br	
Завершить покупку Добавить еще				

Рисунок 3.15 – Страница корзины

При нажатии на кнопку «Завершить покупку», пользователя перенаправляет на страницу, уведомляющую об успешной покупке.

Страница выполнения заказа представлена на рисунке 3.16.

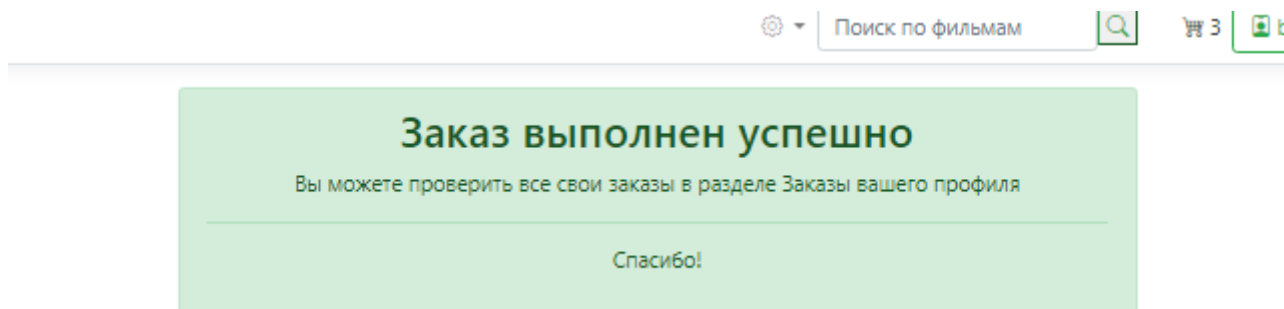


Рисунок 3.16 – Страница входа в систему

Пользователь так же может найти интересующий его фильм, используя «Поиск по фильмам» в навигационной панели. Пример поиска представлен на рисунке 3.17.

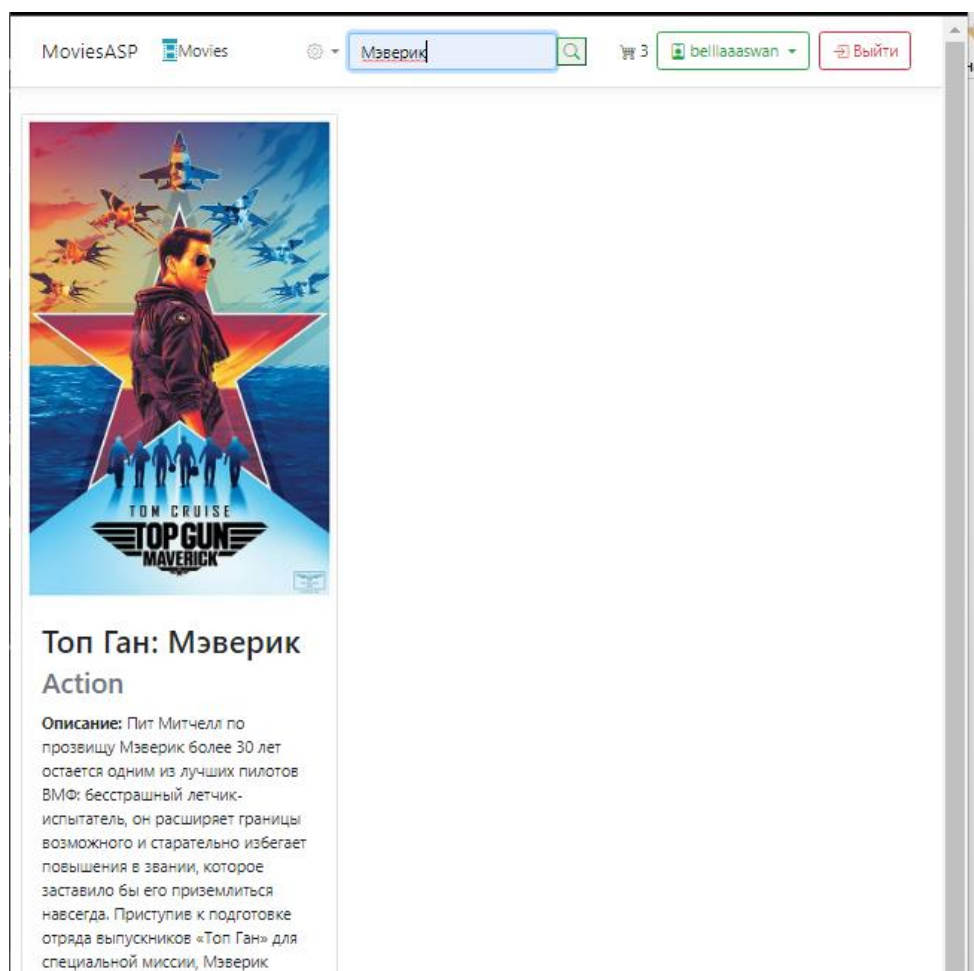


Рисунок 3.17 – Поиск по названию

Пользователь так же может просматривать историю своих заказов. Страница заказов представлена на рисунке 3.18.

Order ID	Items	Total
7	1 [15,00 Br] - Человек-паук: Нет пути домой	46,00 Br
	1 [14,00 Br] - Одноклассники 2	
	1 [17,00 Br] - Гарри Поттер и Орден Феникса	

Рисунок 3.18 – Страница заказов

При входе в аккаунт под администратором, открывается новый функционал веб-приложения. В навигационной панели теперь присутствует кнопка «Пользователи». Навигационная панель представлена на рисунке 3.19.

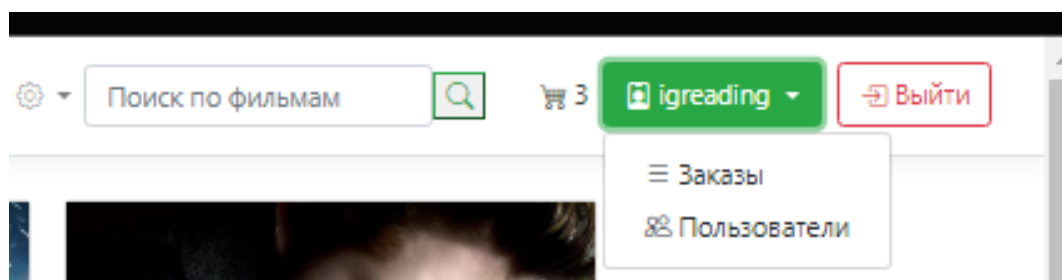


Рисунок 3.19 – Навигационная панель администратора


Нажав на кнопку, «Пользователи», администратору открывается страница с информацией о пользователях. Страница представлена на рисунке 3.20.

Список пользователей		
FullName	UserName	Email
1	1	1
Белла Свон	belllaaswan	belllaaswan
44	44	44
3	3	3
nestea	igreading	igreading
me	meeeeee	meeeeee

Рисунок 3.20 – Список пользователей

Администратору доступна возможность добавлять, удалять и обновлять информацию на сайте. Страница добавления нового сценариста представлена на рисунке 3.21.

Добавить нового сценариста



ProfilePictureURL

<https://avatars.mds.yandex.net/get-kinopoisk-image/1777765/6ffa06f4-eab0-4bfc->

FullName

Гай Ричи

DateOfBirth

10.09.1968

PlaceOfBirth

Хатфилд, Хартфордшир, Англия, Великобритания

[Вернуться назад](#) [Create](#)

Рисунок 3.21 – Добавление сценариста

Страница добавления нового фильма представлена на рисунке 3.22.

Create a new movie

Фильм

Сумерки

Дата выхода

02.06.2008

Страна происхождения

США

Слоган

Запретный плод сладок

Бюджет

37000000

Цена

10

Длительность фильма

122

Фото

https://i.pinimg.com/originals/b5/82/b8/b582b8

Описание

Семнадцатилетняя девушка Белла переезжает к отцу в небольшой городок Форкс. Она влюбляется в загадочного одноклассника, который, как оказалось, происходит из семьи

Сценаристы

Мелисса Розенберг

Жанр фильма

Drama

Возраст

PG13

Продюсеры

Кевин Файги

Актёры

Джейсон Чиленинг

Джим Керри

Коллин Фёрт

Кристен Стюарт

Роберт Паттинсон

Возврат

Create



Рисунок 3.22 – Добавление фильма

Просмотр всех заказов представлен на рисунке 3.23.

List of all your orders

Order ID	Items	Total	User
5	<div> <div></div> <div>[15,00 Br] - Человек-паук: Нет пути домой</div> </div>	15,00 Br	44
6	<div> <div></div> <div>[15,00 Br] - Человек-паук: Нет пути домой</div> </div> <div> <div></div> <div>[10,00 Br] - Сумерки</div> </div>	25,00 Br	44
7	<div> <div></div> <div>[15,00 Br] - Человек-паук: Нет пути домой</div> </div> <div> <div></div> <div>[14,00 Br] - Одноклассники 2</div> </div> <div> <div></div> <div>[17,00 Br] - Гарри Поттер и Орден Феникса</div> </div>	46,00 Br	Белла Свон

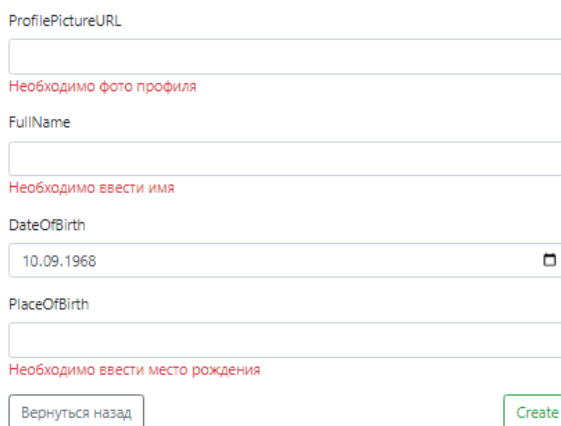
Рисунок 3.23 – Страница заказов

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Ручное тестирование веб-приложения на *ASP.NET Core MVC* заключается в проверке функциональности приложения, его интерфейса и юзабилити вручную, путем взаимодействия с ним как обычный пользователь.

Протестируем форму добавления информации о сценаристе. Если при добавлении какое-то из полей не заполнено или заполнено неправильно, то на экран выведется предупреждение, показанное на рисунке 4.1.

Добавить нового сценариста



ProfilePictureURL

Необходимо фото профиля

FullName

Необходимо ввести имя

DateOfBirth

PlaceOfBirth

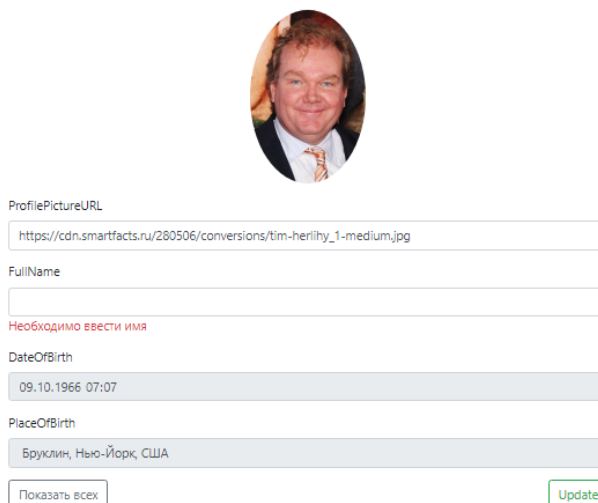
Необходимо ввести место рождения

[Вернуться назад](#) [Create](#)

Рисунок 4.1 – Ошибка добавления данных

Протестируем форму обновления информации о сценаристе. Если при добавлении какое-то из полей не заполнено или заполнено неправильно, то на экран выведется предупреждение, показанное на рисунке 4.2.

Обновить сценариста





ProfilePictureURL

FullName

Необходимо ввести имя

DateOfBirth

PlaceOfBirth

[Показать всех](#) [Update](#)

Рисунок 4.2 – Ошибка обновления данных

Проверим работу регистрации пользователя. Если при обновлении какое-то из полей не заполнено или заполнено неправильно, то на экран выведется предупреждение, показанное на рисунке 4.3.

The screenshot shows a registration form titled "Регистрация в новом аккаунте". It contains four input fields: "Имя" (Name), "Email", "Password", and "Подтвердите пароль" (Confirm password). Each field is empty, and below each one is a red error message: "Необходимо ввести имя" (Name is required), "Необходимо ввести Email" (Email is required), "Необходимо ввести пароль" (Password is required), and "Необходимо подтверждение пароля" (Password confirmation is required). At the bottom, there are two buttons: "Назад" (Back) and "Подтвердить" (Confirm).

Рисунок 4.3 – Ошибка ввода данных при регистрации

Проверим работу входа. Если при вводе какое-то из полей не заполнено или заполнено неправильно, то на экран выведется предупреждение, показанное на рисунке 4.4.

The screenshot shows a login form titled "Вход в аккаунт". At the top, there is a red error message: "Sorry! - Wrong credentials. Please, try again!". Below this, there are two input fields: "Email" and "Password". The "Email" field contains the text "igreading", and the "Password" field contains a series of dots ".....". At the bottom, there are two buttons: "Назад" (Back) and "Войти" (Login).

Рисунок 4.4 – Ошибка ввода данных при регистрации

ЗАКЛЮЧЕНИЕ

В результате разработки приложения на *ASP.NET Core MVC* с использованием *Microsoft Identity*, *Bootstrap*, *AutoMapper* и *MS Server* были успешно применены современные подходы к разработке веб-приложений, что позволило создать качественный и удобный продукт для конечных пользователей.

Одним из ключевых аспектов в разработке был выбор *ASP.NET Core*, который позволяет создавать масштабируемые веб-приложения с высокой производительностью.

Bootstrap был использован для создания красивого интерфейса приложения, что обеспечивает лучший пользовательский опыт и повышает удобство использования. *AutoMapper*, в свою очередь, значительно упрощает маппинг между сущностями базы данных и моделями представления.

MS Server, в свою очередь, был использован для управления базой данных и обеспечения безопасности приложения.

В ходе разработки приложения были изучены новые подходы в программировании, такие как внедрение зависимостей.

Продукт был успешно отлажен и протестирован. Поставленные в курсовой работе задачи были решены полностью.

Список использованных источников

1. *Top Front-End and Back-End ASP .NET Frameworks*. – Электронные данные. – Режим доступа: https://www.onestopdevshop.io/net-framework/top-front-end-and-back-end-asp-net-frameworks/#What_Is_ASPNET_With_Example. – Дата доступа – 10.05.2023
2. Эндрю Троелсен. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание Pro C# 5.0 and the .NET 4.5 Framework, 6th edition. – М.: «Вильямс», 2013. – 1312 с.
3. Кинопоиск. Все фильмы планеты. – Электронные данные. – Режим доступа: <https://www.kinopoisk.ru/> – Дата доступа: 10.05.2023
4. *IMDB: Ratings, Reviews, and Where to Watch the Best Movies*. – Электронные данные. – Режим доступа: <https://www.imdb.com/> – Дата доступа: 10.05.2023
5. Стилмен Э. Изучаем C#. 4-е изд. – СПб.: Питер, 2022. – 816 с.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг программы

```
public class Actor : IEntityTypeBase
{
    public int Id { get; set; }
    [Required(ErrorMessage = "Необходимо фото профиля")]
    [Display(Name = "Фото")]
    public string ProfilePictureURL { get; set; }

    [Display(Name = "Имя")]
    [Required(ErrorMessage = "Необходимо ввести имя")]
    [StringLength(50, MinimumLength = 3, ErrorMessage = "У неверная длина")]
    public string FullName { get; set; }

    [Display(Name = "Дата рождения")]
    [Required(ErrorMessage = "Необходимо ввести дату рождения")]
    public DateTime DateOfBirth { get; set; }

    [Display(Name = "Место рождения")]
    [Required(ErrorMessage = "Необходимо ввести место рождения")]
    [StringLength(50, MinimumLength = 3, ErrorMessage = "У неверная длина")]
    public string PlaceOfBirth { get; set; }

    //Relationships
    public List<Actor_Movie> Actor_Movies { get; set; }
}
public class Actor_Movie: IEntityTypeBase
{
    public int MovieId { get; set; }
    public Movie Movie { get; set; }
    public int ActorId { get; set; }
    public Actor Actor { get; set; }
    [NotMapped]
    public int Id { get; set; }
}
public class Movie : IEntityTypeBase
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public DateTime ReleaseDate { get; set; }
    public string Country { get; set; }
    public string Tagline { get; set; }
    public double Budget { get; set; }
    public double Price { get; set; }
    public double Duration { get; set; }
    public RatingSystem RatingSystem { get; set; }
    public string ImageURL { get; set; }
    public MovieCategory MovieCategory { get; set; }

    //Relationships
    //Actor
    public List<Actor_Movie> Actors_Movies { get; set; }

    //Producer
    public int ProducerId { get; set; }
    [ForeignKey("ProducerId")]
    public Producer Producer { get; set; }
}
```



```

//Screenwriter
public int ScreenwriterId { get; set; }
[ForeignKey("ScreenwriterId")]
public Screenwriter Screenwriter { get; set; }
}
public class Order : IEntityTypeBase
{
    [Key]
    public int Id { get; set; }

    public string Email { get; set; }

    public string UserId { get; set; }
    [ForeignKey(nameof(UserId))]
    public ApplicationUser User { get; set; }

    public List<OrderItem> OrderItems { get; set; }
}
public class OrderItem : IEntityTypeBase
{
    public int Id { get; set; }

    public int Amount { get; set; }
    public double Price { get; set; }

    public int MovieId { get; set; }
    public Movie Movie { get; set; }

    public int OrderId { get; set; }
    public Order Order { get; set; }
}
public class Producer : IEntityTypeBase
{
    public int Id { get; set; }
    [Required(ErrorMessage = "Необходимо фото профиля")]
    public string ProfilePictureURL { get; set; }

    [Required(ErrorMessage = "Необходимо ввести имя")]
    [StringLength(50, MinimumLength = 3, ErrorMessage = "У неверная длина")]
    public string FullName { get; set; }
    [Required(ErrorMessage = "Необходимо ввести дату рождения")]
    public DateTime DateOfBirth { get; set; }
    [Required(ErrorMessage = "Необходимо ввести место рождения")]
    [StringLength(50, MinimumLength = 3, ErrorMessage = "У неверная длина")]
    public string PlaceOfBirth { get; set; }

    //Relationships
    public List<Movie> Movies { get; set; }
}
public class Screenwriter : IEntityTypeBase
{
    public int Id { get; set; }
    [Required(ErrorMessage = "Необходимо фото профиля")]
    public string ProfilePictureURL { get; set; }

    [Required(ErrorMessage = "Необходимо ввести имя")]
    [StringLength(50, MinimumLength = 3, ErrorMessage = "У неверная длина")]
    public string FullName { get; set; }
    [Required(ErrorMessage = "Необходимо ввести дату рождения")]
    public DateTime DateOfBirth { get; set; }
    [Required(ErrorMessage = "Необходимо ввести место рождения")]
    [StringLength(50, MinimumLength = 3, ErrorMessage = "У неверная длина")]
    public string PlaceOfBirth { get; set; }
}

```

```

        //Relationships
        public List<Movie> Movies { get; set; }
    }
    public class ShoppingCartItem
    {
        public int Id { get; set; }
        public Movie Movie { get; set; }
        public int Amount { get; set; }
        public string ShoppingCartId { get; set; }
    }
    public class EntityBaseRepository<T> : IEntityBaseRepository<T> where T : class, IEntityBase, new()
    {
        private readonly AppDbContext _context;
        public EntityBaseRepository(AppDbContext context)
        {
            _context = context;
        }

        public async Task AddAsync(T entity)
        {
            await _context.Set<T>().AddAsync(entity);
            await _context.SaveChangesAsync();
        }

        public async Task DeleteAsync(int id)
        {
            var entity = await _context.Set<T>().FirstOrDefaultAsync(n => n.Id == id);
            EntityEntry entityEntry = _context.Entry<T>(entity);
            entityEntry.State = EntityState.Deleted;

            await _context.SaveChangesAsync();
        }

        public async Task<IEnumerable<T>> GetAllAsync() => await _context.Set<T>().ToListAsync();

        public async Task<IEnumerable<T>> GetAllAsync(params Expression<Func<T, object>>[] includeProperties)
        {
            IQueryable<T> query = _context.Set<T>();
            query = includeProperties.Aggregate(query, (current, includeProperty) => current.Include(includeProperty));
            return await query.ToListAsync();
        }

        public async Task<T> GetByIdAsync(int id) => await _context.Set<T>().FirstOrDefaultAsync(n => n.Id == id);

        public async Task<T> GetByIdAsync(int id, params Expression<Func<T, object>>[] includeProperties)
        {
            IQueryable<T> query = _context.Set<T>();
            query = includeProperties.Aggregate(query, (current, includeProperty) => current.Include(includeProperty));
            return await query.FirstOrDefaultAsync(n => n.Id == id);
        }

        public async Task UpdateAsync(int id, T entity)
        {
            EntityEntry entityEntry = _context.Entry<T>(entity);
            entityEntry.State = EntityState.Modified;
        }
    }

```

```

        await _context.SaveChangesAsync();
    }
}
public interface IEntityBaseRepository<T> where T : class, IEntityBase, new()
{
    Task<IEnumerable<T>> GetAllAsync();
    Task<IEnumerable<T>> GetAllAsync(params Expression<Func<T, object>>[] includeProperties);
    Task<T> GetByIdAsync(int id);
    Task<T> GetByIdAsync(int id, params Expression<Func<T, object>>[] includeProperties);
    Task AddAsync(T entity);
    Task UpdateAsync(int id, T entity);
    Task DeleteAsync(int id);
}
public class SampleContext : IDesignTimeDbContextFactory<AppDbContext>
{
    AppDbContext IDesignTimeDbContextFactory<AppDbContext>.CreateDbContext(string[] args)
    {
        var optionsBuilder = new DbContextOptionsBuilder<AppDbContext>();

        // получаем конфигурацию из файла appsettings.json
        ConfigurationBuilder builder = new ConfigurationBuilder();
        builder.SetBasePath(Directory.GetCurrentDirectory());
        builder.AddJsonFile("appsettings.json");
        IConfigurationRoot config = builder.Build();
        // получаем строку подключения из файла appsettings.json
        string connectionString = config.GetConnectionString("DefaultConnectionString");
        optionsBuilder.UseSqlServer(connectionString, opts => opts.CommandTimeout((int)TimeSpan.FromMinutes(10).TotalSeconds));
        return new AppDbContext(optionsBuilder.Options);
    }
}
public static class ConfigurationExtensions
{
    public static void ConfigureDAL(this IServiceCollection services, IConfiguration configuration)
    {
        services.AddDbContext<AppDbContext>(options => options.UseSqlServer(configuration.GetConnectionString("DefaultConnectionString")));

        services.AddScoped<IEntityBaseRepository<Actor>, EntityBaseRepository<Actor>>();
        services.AddScoped<IEntityBaseRepository<Movie>, EntityBaseRepository<Movie>>();
        services.AddScoped<IEntityBaseRepository<Producer>, EntityBaseRepository<Producer>>();
        services.AddScoped<IEntityBaseRepository<Screenwriter>, EntityBaseRepository<Screenwriter>>();
        services.AddScoped<IEntityBaseRepository<Actor_Movie>, EntityBaseRepository<Actor_Movie>>();
        services.AddScoped<IEntityBaseRepository<Order>, EntityBaseRepository<Order>>();
        services.AddScoped<IEntityBaseRepository<OrderItem>, EntityBaseRepository<OrderItem>>();
    }
}
public class AppDbContext : IdentityDbContext<ApplicationUser>
{
    public AppDbContext(DbContextOptions<AppDbContext> options) : base(options)
    {
    }
}

```

```

protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Actor_Movie>().HasKey(am => new
    {
        am.ActorId,
        am.MovieId
    });

    modelBuilder.Entity<Actor_Movie>().HasOne(m => m.Movie)
        .WithMany(am => am.actors_movies)
        .HasForeignKey(m => m.MovieId);

    modelBuilder.Entity<Actor_Movie>().HasOne(m => m.Actor)
        .WithMany(am => am.actors_movies)
        .HasForeignKey(m => m.ActorId);

    base.OnModelCreating(modelBuilder);
}
public DbSet<Actor> Actors { get; set; }
public DbSet<Movie> Movies { get; set; }
public DbSet<Screenwriter> Screenwriters { get; set; }
public DbSet<Actor_Movie> Actors_movies { get; set; }
public DbSet<Producer> Producers { get; set; }

//Orders related tables
public DbSet<Order> Orders { get; set; }
public DbSet<OrderItem> OrderItems { get; set; }
public DbSet<ShoppingCartItem> ShoppingCartItems { get; set; }
}
public class MoviesService : IMoviesService
{
    private readonly IEntityBaseRepository<Movie> entityBaseRepositoryMovie;
    private readonly IMapper mapper;
    private readonly IEntityBaseRepository<Actor_Movie> entityBaseRepositoryActor_Movie;
    private readonly IEntityBaseRepository<Actor> entityBaseRepositoryActor;
    private readonly IEntityBaseRepository<Producer> entityBaseRepositoryProducer;
    private readonly IEntityBaseRepository<Screenwriter> entityBaseRepositoryScreen-
writer;

    public MoviesService(IEntityBaseRepository<Movie> entityBaseRepository, IMapper map-
per, IEntityBaseRepository<Actor> entityBaseRepositoryActor, IEntityBaseRepository<Ac-
tor_Movie> entityBaseRepositoryActor_Movie, IEntityBaseRepository<Producer> entityBaseRepos-
itoryProducer, IEntityBaseRepository<Screenwriter> entityBaseRepositoryScreenwriter)
    {
        this.entityBaseRepositoryMovie = entityBaseRepository;
        this.mapper = mapper;
        this.entityBaseRepositoryActor_Movie = entityBaseRepositoryActor_Movie;
        this.entityBaseRepositoryActor = entityBaseRepositoryActor;
        this.entityBaseRepositoryProducer = entityBaseRepositoryProducer;
        this.entityBaseRepositoryScreenwriter = entityBaseRepositoryScreenwriter;
    }

    public async Task AddNewMovieAsync(NewMovieVM data)
    {
        var newMovie = new MovieDTO()
        {
            Name = data.Name,
            Description = data.Description,
            ReleaseDate = data.ReleaseDate,
            Country = data.Country,
            Tagline = data.Tagline,
            Budget = data.Budget,
            Price = data.Price,

```

```

        Duration = data.Duration,
        RatingSystem = data.RatingSystem,
        ImageURL = data.ImageURL,
        MovieCategory = data.MovieCategory,
        ProducerId = data.ProducerId,
        ScreenwriterId = data.ScreenwriterId
    };
    await entityBaseRepositoryMovie.AddAsync(mapper.Map<Movie>(newMovie));

    //Add Movie Actors
    foreach (var actorId in data.ActorIds)
    {
        var newActorMovie = new Actor_MovieDTO()
        {
            MovieId = newMovie.Id,
            ActorId = actorId
        };
        await entityBaseRepositoryActor_Movie.AddAsync(mapper.Map<Actor_Movie>(newActorMovie));
    }

    public async Task<Movie> GetMovieByIdAsync(int id)
    {
        var bbb = await entityBaseRepositoryMovie.GetAllAsync(x => x.Screenwriter, x =>
x.Producer, x => x.actors_Movies);
        foreach (var item in bbb)
        {
            foreach (var i in item.actors_Movies)
            {
                var actor = await entityBaseRepositoryActor.GetByIdAsync(i.ActorId);
                i.Actor = actor;
            }
        }
        var movieDetails = bbb
            .FirstOrDefault(n => n.Id == id);

        return movieDetails;
    }
    public async Task<IEnumerable<Movie>> GetMovies()
    {
        return await entityBaseRepositoryMovie.GetAllAsync();
    }

    public async Task<NewMovieDropdownsVM> GetNewMovieDropdownsValues()
    {
        var Actors = await entityBaseRepositoryActor.GetAllAsync();
        var Screenwriters = await entityBaseRepositoryScreenwriter.GetAllAsync();
        var Producers = await entityBaseRepositoryProducer.GetAllAsync();
        var response = new NewMovieDropdownsVM()
        {
            Actors = Actors.OrderBy(n => n.FullName).ToList(),
            Screenwriters = Screenwriters.OrderBy(n => n.FullName).ToList(),
            Producers = Producers.OrderBy(n => n.FullName).ToList()
        };

        return response;
    }

    public async Task UpdateMovieAsync(NewMovieVM data)
    {
        var dbMovies = await entityBaseRepositoryMovie.GetAllAsync();
        var dbMovie = dbMovies.FirstOrDefault(n => n.Id == data.Id);
    }

```

```

        if (dbMovie != null)
        {
            dbMovie.Name = data.Name;
            dbMovie.Description = data.Description;
            dbMovie.ReleaseDate = data.ReleaseDate;
            dbMovie.Country = data.Country;
            dbMovie.Tagline = data.Tagline;
            dbMovie.Budget = data.Budget;
            dbMovie.Price = data.Price;
            dbMovie.Duration = data.Duration;
            dbMovie.RatingSystem = data.RatingSystem;
            dbMovie.ImageURL = data.ImageURL;
            dbMovie.MovieCategory = data.MovieCategory;
            dbMovie.ProducerId = data.ProducerId;
            dbMovie.ScreenwriterId = data.ScreenwriterId;

            await entityBaseRepositoryMovie.UpdateAsync(dbMovie.Id, dbMovie);
        }

        //Remove existing actors
        var existingActorsDBs = await entityBaseRepositoryActor_Movie.GetAllAsync();
        var existingActorsDb = existingActorsDBs.Where(n => n.MovieId ==
data.Id).ToList();
        foreach (var item in existingActorsDb)
        {
            await entityBaseRepositoryActor_Movie.DeleteAsync(item.Id);
        }

        //Add Movie Actors
        foreach (var actorId in data.ActorIds)
        {
            var newActorMovie = new Actor_MovieDTO()
            {
                MovieId = data.Id,
                ActorId = actorId
            };
            await entityBaseRepositoryActor_Movie.AddAsync(mapper.Map<Ac-
tor_Movie>(newActorMovie));
        }
    }

    public class OrdersService : IOrdersService
    {
        private readonly IEntityBaseRepository<Order> entityBaseRepositoryOrder;
        private readonly IEntityBaseRepository<OrderItem> entityBaseRepositoryOrderItem;
        private readonly IEntityBaseRepository<Movie> entityBaseRepositoryMovie;

        private readonly IMapper mapper;
        public OrdersService(IEntityBaseRepository<Order> entityBaseRepositoryOrder, IMapper
mapper, IEntityBaseRepository<OrderItem> entityBaseRepositoryOrderItem, IEntityBaseReposi-
tory<Movie> entityBaseRepositoryMovie)
        {
            this.entityBaseRepositoryOrder = entityBaseRepositoryOrder;
            this.mapper = mapper;
            this.entityBaseRepositoryOrderItem = entityBaseRepositoryOrderItem;
            this.entityBaseRepositoryMovie = entityBaseRepositoryMovie;
        }

        public async Task<List<Order>> GetOrdersByUserIdAndRoleAsync(string userId, string
userRole)
        {

```

```

x.User);
    var order = await entityBaseRepositoryOrder.GetAllAsync(x => x.OrderItems, x =>
foreach(var item in order)
{
    foreach (var i in item.OrderItems)
    {
        var movieId = await entityBaseRepositoryMovie.GetByIdAsync(i.MovieId);
        i.Movie = movieId;
    }
}
//var order = await entityBaseRepositoryOrder.GetAllAsync(x=>x.OrderItems, x=>
"OrderItems.Movie", x=> x.User);
var orders = order.ToList();

if (userRole != "Admin")
{
    orders = orders.Where(n => n.UserId == userId).ToList();
}

return orders;
}

public async Task StoreOrderAsync(List<ShoppingCartItem> items, string userId,
string userEmailAdress)
{
    var order = new OrderDTO()
    {
        UserId = userId,
        Email = userEmailAdress,
    };
    await entityBaseRepositoryOrder.AddAsync(mapper.Map<Order>(order));

    var orderr = await entityBaseRepositoryOrder.GetAllAsync();
    var bbb = orderr.ToList().Last().Id;

    foreach (var item in items)
    {
        var orderItem = new OrderItemDTO()
        {
            Amount = item.Amount,
            MovieId = item.Movie.Id,
            OrderId = bbb,
            Price = item.Movie.Price
        };
        await entityBaseRepositoryOrderItem.AddAsync(mapper.Map<OrderItem>(order-
Item));
    }
}

public class AccountController : Controller
{
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly SignInManager<ApplicationUser> _signInManager;
    private readonly ApplicationDbContext _context;
    private readonly RoleManager<IdentityRole> _roleManager;

    public AccountController(UserManager<ApplicationUser> userManager, SignInManager<Ap-
plicationUser> signInManager, ApplicationDbContext context, RoleManager<IdentityRole> roleManager)
    {
        _userManager = userManager;
        _signInManager = signInManager;
        _context = context;
    }
}

```

```

        _roleManager = roleManager;
    }

    public async Task<IActionResult> Users()
    {
        var users = _context.Users.ToList();
        return View(users);
    }

    public IActionResult Login() => View(new LoginVM());

    [HttpPost]
    public async Task<IActionResult> Login(LoginVM loginVM)
    {
        //await _roleManager.CreateAsync(new IdentityRole(UserRoles.User));

        if (!ModelState.IsValid) return View(loginVM);

        var user = await _userManager.FindByEmailAsync(loginVM.EmailAddress);
        if (user != null)
        {
            var passwordCheck = await _userManager.CheckPasswordAsync(user,
loginVM.Password);
            if (passwordCheck)
            {
                var result = await _signInManager.PasswordSignInAsync(user,
loginVM.Password, false, false);
                if (result.Succeeded)
                {
                    return RedirectToAction("Index", "Movies");
                }
            }
            TempData["Error"] = "Wrong credentials. Please, try again!";
            return View(loginVM);
        }

        TempData["Error"] = "Wrong credentials. Please, try again!";
        return View(loginVM);
    }

    public IActionResult Register() => View(new RegisterVM());

    [HttpPost]
    public async Task<IActionResult> Register(RegisterVM registerVM)
    {
        if (!ModelState.IsValid) return View(registerVM);

        var user = await _userManager.FindByEmailAsync(registerVM.EmailAddress);
        if (user != null)
        {
            TempData["Error"] = "This email address is already in use";
            return View(registerVM);
        }

        var newUser = new ApplicationUser()
        {
            FullName = registerVM.FullName,
            Email = registerVM.EmailAddress,
            UserName = registerVM.EmailAddress
        };
    }

```



```

word);

        var newUserResponse = await _userManager.CreateAsync(newUser, registerVM.Pass-

        if (newUserResponse.Succeeded)
            await _userManager.AddToRoleAsync(newUser, UserRoles.User);

        return View("RegisterCompleted");
    }

    [HttpPost]
    public async Task<IActionResult> Logout()
    {
        await _signInManager.SignOutAsync();
        return RedirectToAction("Index", "Movies");
    }

    public IActionResult AccessDenied(string returnUrl)
    {
        return View();
    }
}

public class ActorsController : Controller
{
    private readonly IActorsService _service;

    public ActorsController(IActorsService service)
    {
        _service = service;
    }

    [AllowAnonymous]
    public async Task<IActionResult> Index()
    {
        var data = await _service.GetAllAsync();
        return View(data);
    }

    //Get: Actors/Create
    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Create([Bind("FullName,ProfilePic-
tureURL,DateOfBirth,PlaceOfBirth")] Actor actor)
    {
        if (!ModelState.IsValid)
        {
            return View(actor);
        }
        await _service.AddAsync(actor);
        return RedirectToAction(nameof(Index));
    }

    //Get: Actors/Details/1
    [AllowAnonymous]
    public async Task<IActionResult> Details(int id)
    {
        var actorDetails = await _service.GetByIdAsync(id);
    }
}

```

```

        if (actorDetails == null) return View("NotFound");
        return View(actorDetails);
    }

    //Get: Actors/Edit/1
    public async Task<IActionResult> Edit(int id)
    {
        var actorDetails = await _service.GetByIdAsync(id);
        if (actorDetails == null) return View("NotFound");
        return View(actorDetails);
    }

    [HttpPost]
    public async Task<IActionResult> Edit(int id, [Bind("Id,FullName,ProfilePictureURL,DateOfBirth,PlaceOfBirth")] Actor actor)
    {
        if (!ModelState.IsValid)
        {
            return View(actor);
        }
        await _service.UpdateAsync(id, actor);
        return RedirectToAction(nameof(Index));
    }

    //Get: Actors/Delete/1
    public async Task<IActionResult> Delete(int id)
    {
        var actorDetails = await _service.GetByIdAsync(id);
        if (actorDetails == null) return View("NotFound");
        return View(actorDetails);
    }

    [HttpPost, ActionName("Delete")]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var actorDetails = await _service.GetByIdAsync(id);
        if (actorDetails == null) return View("NotFound");

        await _service.DeleteAsync(id);
        return RedirectToAction(nameof(Index));
    }
}

[Authorize(Roles = UserRoles.Admin)]
public class MoviesController : Controller
{
    private readonly IMoviesService _service;

    public MoviesController(IMoviesService service)
    {
        _service = service;
    }

    [AllowAnonymous]
    public async Task<IActionResult> Index()
    {
        var allMovies = await _service.GetMovies();
        return View(allMovies);
    }

    [AllowAnonymous]
    public async Task<IActionResult> Filter(string searchString)
    {
        var allMovies = await _service.GetMovies();
    }
}

```

```

        if (!string.IsNullOrEmpty(searchString))
        {
            var filteredResult = allMovies.Where(n => n.Name.ToLower().Contains(searchString.ToLower()) || n.Description.ToLower().Contains(searchString.ToLower())).ToList();

            //var filteredResultNew = allMovies.Where(n => string.Equals(n.Name, searchString, StringComparison.CurrentCultureIgnoreCase) || string.Equals(n.Description, searchString, StringComparison.CurrentCultureIgnoreCase)).ToList();

            return View("Index", filteredResult);
        }

        return View("Index", allMovies);
    }

    //GET: Movies/Details/1
    [AllowAnonymous]
    public async Task<IActionResult> Details(int id)
    {
        var movieDetail = await _service.GetMovieByIdAsync(id);
        return View(movieDetail);
    }

    //GET: Movies/Create
    public async Task<IActionResult> Create()
    {
        var movieDropdownsData = await _service.GetNewMovieDropdownsValues();

        ViewBag.Screenwriters = new SelectList(movieDropdownsData.Screenwriters, "Id", "FullName");
        ViewBag.Producers = new SelectList(movieDropdownsData.Producers, "Id", "FullName");
        ViewBag.Actors = new SelectList(movieDropdownsData.Actors, "Id", "FullName");

        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Create(NewMovieVM movie)
    {
        if (!ModelState.IsValid)
        {
            var movieDropdownsData = await _service.GetNewMovieDropdownsValues();

            ViewBag.Screenwriters = new SelectList(movieDropdownsData.Screenwriters, "Id", "FullName");
            ViewBag.Producers = new SelectList(movieDropdownsData.Producers, "Id", "FullName");
            ViewBag.Actors = new SelectList(movieDropdownsData.Actors, "Id", "FullName");

            return View(movie);
        }

        await _service.AddNewMovieAsync(movie);
        return RedirectToAction(nameof(Index));
    }

    //GET: Movies/Edit/1
    public async Task<IActionResult> Edit(int id)
    {
        var movieDetails = await _service.GetMovieByIdAsync(id);

```

```

        if (movieDetails == null) return View("NotFound");

        var response = new NewMovieVM()
        {
            Name = movieDetails.Name,
            Description = movieDetails.Description,
            ReleaseDate = movieDetails.ReleaseDate,
            Country = movieDetails.Country,
            Tagline = movieDetails.Tagline,
            Budget = movieDetails.Budget,
            Price = movieDetails.Price,
            Duration = movieDetails.Duration,
            RatingSystem = movieDetails.RatingSystem,
            ImageURL = movieDetails.ImageURL,
            MovieCategory = movieDetails.MovieCategory,
            ProducerId = movieDetails.ProducerId,
            ScreenwriterId = movieDetails.ScreenwriterId,
            ActorIds = movieDetails.Actors_Movies.Select(n => n.ActorId).ToList(),
        };

        var movieDropdownsData = await _service.GetNewMovieDropdownsValues();

        ViewBag.Screenwriters = new SelectList(movieDropdownsData.Screenwriters, "Id",
"FullName");
        ViewBag.Producers = new SelectList(movieDropdownsData.Producers, "Id",
"FullName");
        ViewBag.Actors = new SelectList(movieDropdownsData.Actors, "Id", "FullName");

        return View(response);
    }

    [HttpPost]
    public async Task<IActionResult> Edit(int id, NewMovieVM movie)
    {
        if (id != movie.Id) return View("NotFound");

        if (!ModelState.IsValid)
        {
            var movieDropdownsData = await _service.GetNewMovieDropdownsValues();

            ViewBag.Screenwriters = new SelectList(movieDropdownsData.Screenwriters,
"Id", "FullName");
            ViewBag.Producers = new SelectList(movieDropdownsData.Producers, "Id",
"FullName");
            ViewBag.Actors = new SelectList(movieDropdownsData.Actors, "Id",
"FullName");

            return View(movie);
        }

        await _service.UpdateMovieAsync(movie);
        return RedirectToAction(nameof(Index));
    }
}

[Authorize]
public class OrdersController : Controller
{
    private readonly IMoviesService _moviesService;
    private readonly ShoppingCart _shoppingCart;
    private readonly IOOrdersService _ordersService;

    public OrdersController(IMoviesService moviesService, ShoppingCart shoppingCart,
IOOrdersService ordersService)
    {

```

```

        _moviesService = moviesService;
        _shoppingCart = shoppingCart;
        _ordersService = ordersService;
    }

    public async Task<IActionResult> Index()
    {
        string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        string userRole = User.FindFirstValue(ClaimTypes.Role);

        var orders = await _ordersService.GetOrdersByUserIdAndRoleAsync(userId,
userRole);
        return View(orders);
    }

    public IActionResult ShoppingCart()
    {
        var items = _shoppingCart.GetShoppingCartItems();
        _shoppingCart.ShoppingCartItems = items;

        var response = new ShoppingCartVM()
        {
            ShoppingCart = _shoppingCart,
            ShoppingCartTotal = _shoppingCart.GetShoppingCartTotal()
        };

        return View(response);
    }

    public async Task<IActionResult> AddItemToShoppingCart(int id)
    {
        var item = await _moviesService.GetMovieByIdAsync(id);

        if (item != null)
        {
            _shoppingCart.AddItemToCart(item);
        }
        return RedirectToAction(nameof(ShoppingCart));
    }

    public async Task<IActionResult> RemoveItemFromShoppingCart(int id)
    {
        var item = await _moviesService.GetMovieByIdAsync(id);

        if (item != null)
        {
            _shoppingCart.RemoveItemFromCart(item);
        }
        return RedirectToAction(nameof(ShoppingCart));
    }

    public async Task<IActionResult> CompleteOrder()
    {
        var items = _shoppingCart.GetShoppingCartItems();
        string userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
        string userEmailAddress = User.FindFirstValue(ClaimTypes.Email);

        await _ordersService.StoreOrderAsync(items, userId, userEmailAddress);
        await _shoppingCart.ClearShoppingCartAsync();

        return View("OrderCompleted");
    }
}
[Authorize(Roles = UserRoles.Admin)]

```

```

public class ProducersController : Controller
{
    private readonly IProducersService _service;

    public ProducersController(IProducersService service)
    {
        _service = service;
    }

    [AllowAnonymous]
    public async Task<IActionResult> Index()
    {
        var allProducers = await _service.GetAllAsync();
        return View(allProducers);
    }

    //GET: producers/details/1
    [AllowAnonymous]
    public async Task<IActionResult> Details(int id)
    {
        var producerDetails = await _service.GetByIdAsync(id);
        if (producerDetails == null) return View("NotFound");
        return View(producerDetails);
    }

    //GET: producers/create
    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Create([Bind("ProfilePictureURL,FullName,DateOfBirth,PlaceOfBirth")] Producer producer)
    {
        if (!ModelState.IsValid) return View(producer);

        await _service.AddAsync(producer);
        return RedirectToAction(nameof(Index));
    }

    //GET: producers/edit/1
    public async Task<IActionResult> Edit(int id)
    {
        var producerDetails = await _service.GetByIdAsync(id);
        if (producerDetails == null) return View("NotFound");
        return View(producerDetails);
    }

    [HttpPost]
    public async Task<IActionResult> Edit(int id, [Bind("Id,ProfilePictureURL,FullName,DateOfBirth,PlaceOfBirth")] Producer producer)
    {
        if (!ModelState.IsValid) return View(producer);

        if (id == producer.Id)
        {
            await _service.UpdateAsync(id, producer);
            return RedirectToAction(nameof(Index));
        }
        return View(producer);
    }

    //GET: producers/delete/1

```

```

public async Task<IActionResult> Delete(int id)
{
    var producerDetails = await _service.GetByIdAsync(id);
    if (producerDetails == null) return View("NotFound");
    return View(producerDetails);
}

[HttpPost, ActionName("Delete")]
public async Task<IActionResult> DeleteConfirmed(int id)
{
    var producerDetails = await _service.GetByIdAsync(id);
    if (producerDetails == null) return View("NotFound");

    await _service.DeleteAsync(id);
    return RedirectToAction(nameof(Index));
}
}
[Authorize(Roles = UserRoles.Admin)]
public class ScreenwritersController : Controller
{
    private readonly IScreenwritersService _service;

    public ScreenwritersController(IScreenwritersService service)
    {
        _service = service;
    }

    [AllowAnonymous]
    public async Task<IActionResult> Index()
    {
        var allScreenwtiters = await _service.GetAllAsync();
        return View(allScreenwtiters);
    }

    //GET: screenwtiters/details/1
    [AllowAnonymous]
    public async Task<IActionResult> Details(int id)
    {
        var producerDetails = await _service.GetByIdAsync(id);
        if (producerDetails == null) return View("NotFound");
        return View(producerDetails);
    }

    //GET: screenwtiters/create
    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    public async Task<IActionResult> Create([Bind("ProfilePic-
tureURL,FullName,DateOfBirth,PlaceOfBirth")] Screenwriter screenwtiter)
    {
        if (!ModelState.IsValid) return View(screenwtiter);

        await _service.AddAsync(screenwtiter);
        return RedirectToAction(nameof(Index));
    }

    //GET: screenwtiters/edit/1
    public async Task<IActionResult> Edit(int id)
    {
        var screenwtiterDetails = await _service.GetByIdAsync(id);
        if (screenwtiterDetails == null) return View("NotFound");

```

```

        return View(screenwtiterDetails);
    }

    [HttpPost]
    public async Task<IActionResult> Edit(int id, [Bind("Id,ProfilePictureURL,FullName,DateOfBirth,PlaceOfBirth")] Screenwriter screenwtiter)
    {
        if (!ModelState.IsValid) return View(screenwtiter);

        if (id == screenwtiter.Id)
        {
            await _service.UpdateAsync(id, screenwtiter);
            return RedirectToAction(nameof(Index));
        }
        return View(screenwtiter);
    }

    //GET: screenwtiters/delete/1
    public async Task<IActionResult> Delete(int id)
    {
        var screenwtiterDetails = await _service.GetByIdAsync(id);
        if (screenwtiterDetails == null) return View("NotFound");
        return View(screenwtiterDetails);
    }

    [HttpPost, ActionName("Delete")]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        var screenwtiterDetails = await _service.GetByIdAsync(id);
        if (screenwtiterDetails == null) return View("NotFound");

        await _service.DeleteAsync(id);
        return RedirectToAction(nameof(Index));
    }
}

public class ShoppingCart
{
    public ApplicationDbContext _context { get; set; }

    public string ShoppingCartId { get; set; }
    public List<ShoppingCartItem> ShoppingCartItems { get; set; }

    public ShoppingCart(ApplicationDbContext context)
    {
        _context = context;
    }

    public static ShoppingCart GetShoppingCart(IServiceProvider services)
    {
        ISession session = services.GetRequiredService<IHttpContextAccessor>()?.HttpContext.Session;
        var context = services.GetService<ApplicationDbContext>();

        string cartId = session.GetString("CartId") ?? Guid.NewGuid().ToString();
        session.SetString("CartId", cartId);

        return new ShoppingCart(context) { ShoppingCartId = cartId };
    }

    public void AddItemToCart(Movie movie)
    {
        var shoppingCartItem = _context.ShoppingCartItems.FirstOrDefault(n => n.Movie.Id == movie.Id && n.ShoppingCartId == ShoppingCartId);
    }
}

```



```

        if (shoppingCartItem == null)
        {
            shoppingCartItem = new ShoppingCartItem()
            {
                ShoppingCartId = ShoppingCartId,
                Movie = movie,
                Amount = 1
            };

            _context.ShoppingCartItems.Add(shoppingCartItem);
        }
        else
        {
            shoppingCartItem.Amount++;
        }
        _context.SaveChanges();
    }

    public void RemoveItemFromCart(Movie movie)
    {
        var shoppingCartItem = _context.ShoppingCartItems.FirstOrDefault(n => n.Movie.Id
== movie.Id && n.ShoppingCartId == ShoppingCartId);

        if (shoppingCartItem != null)
        {
            if (shoppingCartItem.Amount > 1)
            {
                shoppingCartItem.Amount--;
            }
            else
            {
                _context.ShoppingCartItems.Remove(shoppingCartItem);
            }
        }
        _context.SaveChanges();
    }

    public List<ShoppingCartItem> GetShoppingCartItems()
    {
        return ShoppingCartItems ?? (ShoppingCartItems = _context.ShoppingCartItems.Where(n => n.ShoppingCartId == ShoppingCartId).Include(n => n.Movie).ToList());
    }

    public double GetShoppingCartTotal() => _context.ShoppingCartItems.Where(n => n.ShoppingCartId == ShoppingCartId).Select(n => n.Movie.Price * n.Amount).Sum();

    public async Task ClearShoppingCartAsync()
    {
        var items = await _context.ShoppingCartItems.Where(n => n.ShoppingCartId == ShoppingCartId).ToListAsync();
        _context.ShoppingCartItems.RemoveRange(items);
        await _context.SaveChangesAsync();
    }

    public static void ConfigureBLL(this IServiceCollection services, IConfiguration configuration)
    {
        MoviesASP.DAL.Configuration.ConfigurationExtensions.ConfigureDAL(services, configuration);

        IMapper mapper = new MapperConfiguration(mc =>
        {
            mc.AddProfile(new ActorProfile());
            mc.AddProfile(new Actor_MovieProfile());
        }

```

```

        mc.AddProfile(new MovieProfile());
        mc.AddProfile(new ProducerProfile());
        mc.AddProfile(new ScreenwritersProfile());
        mc.AddProfile(new OrderProfile());
        mc.AddProfile(new OrderItemProfile());

    }).CreateMapper();
    services.AddSingleton(mapper);
    services.AddTransient<IMoviesService, MoviesService>();
    services.AddTransient<IActorsService, ActorsService>();
    services.AddTransient<IOrdersService, OrdersService>();
    services.AddTransient<IProducersService, ProducersService>();
    services.AddTransient<IScreenwritersService, ScreenwritersService>();

}

}
public class Actor_MovieProfile : Profile
{
    public Actor_MovieProfile()
    {
        CreateMap<Actor_Movie, Actor_MovieDTO>().ReverseMap();
    }
}
public class ActorProfile:Profile
{
    public ActorProfile()
    {
        CreateMap<Actor, ActorDTO>().ReverseMap();
    }
}
public class MovieProfile: Profile
{
    public MovieProfile()
    {
        CreateMap<Movie, MovieDTO>().ReverseMap();
    }
}
public class OrderItemProfile : Profile
{
    public OrderItemProfile()
    {
        CreateMap<OrderItem, OrderItemDTO>().ReverseMap();
    }
}
public class OrderProfile : Profile
{
    public OrderProfile()
    {
        CreateMap<Order, OrderDTO>().ReverseMap();
    }
}
public class ProducerProfile : Profile
{
    public ProducerProfile()
    {
        CreateMap<Movie, MovieDTO>().ReverseMap();
    }
}
public class ScreenwritersProfile : Profile
{
    public ScreenwritersProfile()
    {
        CreateMap<Movie, MovieDTO>().ReverseMap();
    }
}

```

```

    }
}
public class Actor_MovieDTO
{
    public int MovieId { get; set; }
    public MovieDTO Movie { get; set; }

    public int ActorId { get; set; }
    public ActorDTO Actor { get; set; }
    public int Id { get; set; }
}
public class ActorDTO : IEntityBase
{
    public int Id { get; set; }
    [Display(Name = "Фото")]
    [Required(ErrorMessage = "Введите ссылку на постер")]
    public string ProfilePictureURL { get; set; }
    [Display(Name = "Продюссер")]
    [Required(ErrorMessage = "Введите имя")]
    public string FullName { get; set; }
    [Display(Name = "Дата рождения")]
    [Required(ErrorMessage = "Введите дату")]
    public DateTime DateOfBirth { get; set; }

    [Display(Name = "Место рождения")]
    [Required(ErrorMessage = "Введите место рождения")]
    public string PlaceOfBirth { get; set; }

    //Relationships
    public List<MovieDTO> Movies { get; set; }
}
public class ApplicationUserDTO
{
    [Display(Name = "Full name")]
    public string FullName { get; set; }
}
public class MovieDTO
{
    public int Id { get; set; }
    [Display(Name = "Фильм")]
    [Required(ErrorMessage = "Введите название")]
    public string Name { get; set; }
    [Display(Name = "Описание")]
    [Required(ErrorMessage = "Введите описание")]
    public string Description { get; set; }

    [Display(Name = "Дата выхода")]
    [Required(ErrorMessage = "Введите дату")]
    public DateTime ReleaseDate { get; set; }

    [Display(Name = "Страна происхождения")]
    [Required(ErrorMessage = "Введите страну происхождения")]
    public string Country { get; set; }

    [Display(Name = "Слоган")]
    [Required(ErrorMessage = "Введите слоган")]
    public string Tagline { get; set; }

    [Display(Name = "Бюджет")]
    [Required(ErrorMessage = "Введите бюджет")]
    public double Budget { get; set; }

    [Display(Name = "Цена")]
    [Required(ErrorMessage = "Введите цену")]

```

```

    public double Price { get; set; }

    [Display(Name = "Длительность фильма")]
    [Required(ErrorMessage = "Введите длительность фильма")]
    public double Duration { get; set; }

    [Display(Name = "Возраст")]
    [Required(ErrorMessage = "Введите возрастное ограничение")]
    public RatingSystem RatingSystem { get; set; }

    [Display(Name = "Фото")]
    [Required(ErrorMessage = "Введите ссылку на постер")]
    public string ImageURL { get; set; }
    [Display(Name = "Жанр фильма")]
    [Required(ErrorMessage = "Введите жанр")]
    public MovieCategory MovieCategory { get; set; }

    //Relationships
    //Actor
    [Display(Name = "Актеры")]
    [Required(ErrorMessage = "Выберете актера(ов)")]
    public List<int> ActorIds { get; set; }

    //Producer
    [Display(Name = "Продюссеры")]
    [Required(ErrorMessage = "Выберете продюссера(ов)")]
    public int ProducerId { get; set; }

    //Screenwriter
    [Display(Name = "Сценаристы")]
    [Required(ErrorMessage = "Выберете сценариста(ов)")]
    public int ScreenwriterId { get; set; }
}
public class OrderDTO
{
    public int Id { get; set; }

    public string Email { get; set; }

    public string UserId { get; set; }
    [ForeignKey(nameof(UserId))]
    public ApplicationUserDTO User { get; set; }

    public List<OrderItemDTO> OrderItems { get; set; }
}
public class OrderItemDTO
{
    public int Id { get; set; }

    public int Amount { get; set; }
    public double Price { get; set; }
    public int MovieId { get; set; }
    public MovieDTO Movie { get; set; }

    public int OrderId { get; set; }
    public OrderDTO Order { get; set; }
}
public class ProducerDTO: IEntityBase
{
    public int Id { get; set; }
    [Display(Name = "Фото")]
    [Required(ErrorMessage = "Введите ссылку на постер")]
    public string ProfilePictureURL { get; set; }
}

```

```

        [Display(Name = "Продюссер")]
        [Required(ErrorMessage = "Введите имя")]
        public string FullName { get; set; }
        [Display(Name = "Дата рождения")]
        [Required(ErrorMessage = "Введите дату")]
        public DateTime DateOfBirth { get; set; }

        [Display(Name = "Место рождения")]
        [Required(ErrorMessage = "Введите место рождения")]
        public string PlaceOfBirth { get; set; }

        //Relationships
        public List<MovieDTO> Movies { get; set; }
    }
    public class ScreenwriterDTO : IEntityBase
    {
        public int Id { get; set; }
        [Display(Name = "Фото")]
        [Required(ErrorMessage = "Введите ссылку на постер")]
        public string ProfilePictureURL { get; set; }
        [Display(Name = "Продюссер")]
        [Required(ErrorMessage = "Введите имя")]
        public string FullName { get; set; }
        [Display(Name = "Дата рождения")]
        [Required(ErrorMessage = "Введите дату")]
        public DateTime DateOfBirth { get; set; }

        [Display(Name = "Место рождения")]
        [Required(ErrorMessage = "Введите место рождения")]
        public string PlaceOfBirth { get; set; }

        //Relationships
        public List<MovieDTO> Movies { get; set; }
    }
    public class ShoppingCartItemDTO
    {
        public int Id { get; set; }
        public MovieDTO Movie { get; set; }
        public int Amount { get; set; }
        public string ShoppingCartId { get; set; }
    }
}

```