Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу
Інститут інформаційних технологій

Кафедра комп'ютерних систем і мереж

Гречанюк Богдан Миколайович

УДК 004.4

БАКАЛАВРСЬКА РОБОТА

Розробка інформаційного web-сайту для кінотеатру засобами C# та фреймворків ASP. NET і SSMS

Комп'ютерна інженерія
(назва освітньої програми)

123 – Комп'ютерна інженерія
(шифр і назва спеціальності)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Здобувач освітнього ступеня _____ *Гречанюк Б.М.* _____
(підпис, ініціали та прізвище здобувача)

Науковий керівник _____ *Гарасимів В.М., доцент* _____
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
Завідувач кафедри
д.т.н., професор _____ /С. І. Мельничук/
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2025 рік

Івано-Франківський національний технічний університет нафти і газу

Інститут *Інформаційних технологій*

Кафедра *Комп'ютерних систем і мереж*

Освітньо-кваліфікаційний рівень *бакалавр*

Спеціальність *123 – Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ:

Зав. кафедрою КСМ

С.І. Мельничук

« 05 » травня 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ.

Гречанюку Богдану Миколайовичу

, (прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Розробка інформаційного web-сайту для кінотеатру засобами C# та фреймворків ASP. NET і SSMS*

керівник проєкту (роботи) *Гарасимів Віра Михайлівна, доцент* .

затверджені наказом вищого навчального закладу від *05. 05. 2025 №275/7*

2. Строк подання студентом проекту (роботи) *12 червня 2025р.*

3. Вихідні дані до роботи *Методичні вказівки, технічна література* .

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) *1.Інформаційне забезпечення інтернет-сервісу продажу квитків 2. Розробка структурних рішень веб-сайту продажу квитків 3.Реалізація графічного інтерфейсу та бази даних веб-сайту продажу квитків*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) .

6. Консультанти розділів роботи

7. Дата видачі завдання *29 січня 2025 р.* .

№ з/п	Назва етапів дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	*Збір інформації, вивчення літератури та пошук додаткової інформації*	*Лютий 2025р*	
2	*Інформаційне забезпечення інтернет-сервісу продажу квитків*	*Березень 2025р*	
3	*Розробка структурних рішень веб-сайту продажу квитків*	*Квітень 2025р*	
4	*Реалізація графічного інтерфейсу та бази даних веб-сайту продажу квитків*	*Травень 2025р*	
5	*Оформлення додатків, дипломної роботи*	*Червень 2025р*	

Студент _____ *Гречанюк Б. М.*

Керівник роботи _____ *Гарасимів В. М.*

АНОТАЦІЯ

Бакалаврська робота присвячена проєктуванню та розробці платформи для надання інформації про фільми та організації продажів квитків.

Метою роботи є проєктування та реалізація інформаційної, структурної, алгоритмічної та програмної складових веб-додатку для зберігання, відображення, пошуку та організації процесу купівлі квитків засобами C#, ASP. NET, Sql Server.

В ході виконання бакалаврської роботи здійснено огляд основних властивостей фільмів, проаналізовано веб-сайти аналогів і сформовано технічне завдання на проект. Також розроблено архітектуру та функціонал веб-сайту, таблиці, структуру бази даних і діаграми, що ілюструють порядок взаємодії між складовими частинами сервісу. Здійснено розробку USE-CASE діаграми взаємодії з користувачем, а також інтерфейс та функціонал веб-сайту. В кінцевому результаті отримаємо готовий веб-сайт для продажу квитків на фільми.

Ключові слова: інформаційний сервіс, веб-додаток, база даних.

ANNOTATION

The bachelor's thesis is dedicated to the design and development of a platform for providing information about films and organizing tickets sales.

The purpose of the work is to design and implement the information, structural, algorithmic and functional software components of a web application for storing, displaying, searching and organizing the ticket purchase process using C#, ASP. NET, Sql Server.

During the bachelor's thesis, a review of the main properties of films was carried out, analogue websites were analyzed and a technical task for the project was formed. The architecture and functionality of the website, tables, database structure and diagrams illustrating the order of interaction between the components of the service were also developed. A USE-CASE diagram of user interaction, as well as the interface and functionality of the website were developed. The end result will be a ready-made website for selling movie tickets.

Keywords: information service, web application, database.

ЗМІСТ

ВСТУП

У наш час все більшу роль відіграє впровадження цифрових технологій та автоматизованих рішень у повсякденні сервіси. Люди очікують, що онлайн-платформи дозволять їм легко та швидко вирішувати різноманітні завдання, зокрема й організацію дозвілля. Однією з таких сфер є кіноіндустрія, де ефективне програмне забезпечення може значно покращити користувацький досвід, надаючи можливість бронювання квитків, перегляду розкладу сеансів і отримання інформації про фільми.

Актуальність роботи обумовлена потребою створити унікальний веб-сервіс для інформування користувачів, демонстрації продукції та забезпечення процесу продажу квитків інструментами мови програмування C# та бази даних реляційного типу, яка у порівнянні з існуючими сервісами матиме нижчі вимоги до апаратних ресурсів та буде набагато зручнішою у використанні для користувачів.

Об'єктом дослідження є процес створення інформаційного сервісу, підтримки та забезпечення продажі квитків через Інтернет.

Предметом дослідження є удосконалення функціоналу інформаційного сервісу продажі квитків в інтернеті.

Метою роботи є проєктування та реалізація інформаційної, структурної, алгоритмічної та програмної складових веб-додатку для зберігання, відображення, пошуку та організації процесу купівлі квитків.

Методи дослідження є теорія інформації, теорія реляційних баз даних, методи розробки користувацьких інтерфейсів.

Практичне значення - розроблено веб-сервіс організації інформаційного супроводу та продаж квитків.

1 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ ІНТЕРНЕТ-СЕРВІСУ ПРОДАЖУ КВИТКІВ

1.1 Опис інформаційного процесу

У межах даної роботи досліджується веб-сайт для продажу квитків у кінотеатрі як приклад галузевої інформаційної системи, що автоматизує процес бронювання та купівлі квитків на кіносеанси. Система реалізує інформаційний процес, який охоплює:

1. Збирання інформації — адміністратор сайту вводить дані про фільми, кінотеатри, продсюсерів, акторів та ціни;

2. Зберігання даних — інформація структуровано зберігається в базі даних і охоплює фільми, кінотеатри, продсюсерів, акторів та ціни;

3. Обробка даних — сайт забезпечує можливість пошуку фільмів за назвою; здійснює формування замовлення після оплати;

4. Надання інформації користувачу — відображення фільмів, їх деталей, вартості квитків.

Таким чином, веб-сайт продажу квитків у кінотеатрі є спеціалізованою інформаційною системою в галузі кінопрокату, що дозволяє автоматизувати взаємодію між клієнтом та адміністрацією закладу.

Функціонування таких інформаційних систем повинно відповідати чинній нормативно-правовій базі України, а також технічним стандартам розробки ПЗ. Зокрема, було проаналізовано такі документи:

1. Закон України «Про захист персональних даних» — регулює обробку персональної інформації користувачів, які купують квитки;

2. Закон України «Про електронну комерцію» — визначає порядок укладання електронних договорів купівлі-продажу, включаючи оплату квитків онлайн;

3. Національні стандарти в галузі IT (ДСТУ ISO/IEC 25010:2016) — використовуються для оцінки якості програмного забезпечення(зокрема зручності, функціональності, доступності);

4. Методичні рекомендації з побудови інформаційних систем, зокрема щодо баз даних, інтерфейсів, безпеки інформації(включно з OWASP-рекомендаціями);

У межах розробки та функціонування веб-сайту ці документи забезпечують правове, організаційне й технічне підґрунтя для реалізації інформаційного процесу.

Основні інформативні параметри системи, які забезпечують її функціонування:

1. Назва фільму: кожен фільм має свою оригінальну назву, що надає можливість розрізняти його поміж інших фільмів;

2. Зображення фільму: кожен фільм має постер, що презентує зовнішній вигляд та деталі прокату;

3. Ціна за квиток: кожен квиток на фільм має ціну, що дає можливість користувачам ознайомитись з ціною квитка;

4. Опис фільму: кожен фільм містить короткий опис, у якому розповідається короткий сюжет фільму;

5. Категорія: фільми можуть бути різних категорій: бойовик, комедія, хоррор тощо. Завдяки цьому користувач, не читавши опис, буде знати жанр фільму;

6. Початок прокату: інформація про початок прокату фільму в кінотеатрі задля інформування користувача про доступність квитків;

7. Кінець прокату: інформація про кінець прокату фільму, після чого користувач не матиме змоги купити квиток;

8. Продюсер: кожен фільм має свого продюсера, що допоможе людям розуміти , яка людина стоїть за створенням;

9. Кінотеатр: кожен фільм має інформацію про кінотеатр, де він буде відбуватися;

10. Актори: кожен фільм має інформацію про акторів, які брали участь в його зйомці.

Ці параметри фільмів є основною інформацією, яка ознайомлює користувача з фільмом, допомагає при прийнятті рішення про покупку та забезпечує інформаційний процес на веб-сайті.

Основою функціоналу веб-сайту є інструменти, що забезпечують швидкий та функціональний доступ до інформації про фільм. Наприклад, пошук фільму по його назві. Якщо користувач введе половину назви, то йому всеодно покаже результат, у назві фільму якого міститься введений текст.

1.2 Порівняльний огляд веб-сайтів кінотеатрів-аналогів

Важливим етапом у процесі створення інформаційного ресурсу є аналіз типових веб-ресурсів за відповідним напрямом, зокрема сайтів кінотеатрів, що здійснюють продаж квитків. Також потрібно зважати на уподобання користувачів, які мають мають індивідуальні уявлення про дизайн і функціонал онлайн-платформи. У ході дослідження веб-застосунків аналогів вибрано декілька варіантів, які варто взяти як фундамент для створення веб-сайту, зокрема:

1. Компанія: Linia Kino (інтернет ресурс: https: //liniakino. com)
2. Компанія: Lumiere (інтернет ресурс: https: //kinolumiere. com/)
3. Компанія: MovieLand (інтернет ресурс: https: //m. vkino. com. ua/ua)

Основною задачею було створити сайт, у якого функціонал буде схожим до вищеперелічених. Тому спочатку перед постанкою детального завдання на проєкт, треба виконати точний аналіз дизайну та структури сайтів-конкурентів.

Якщо проаналізувати сайт компанії «Linia Kino» (рис. 1.1), то відразу помітно, що у основі візуальної стилістики сайту є темні відтінки та фотографії високої якості.

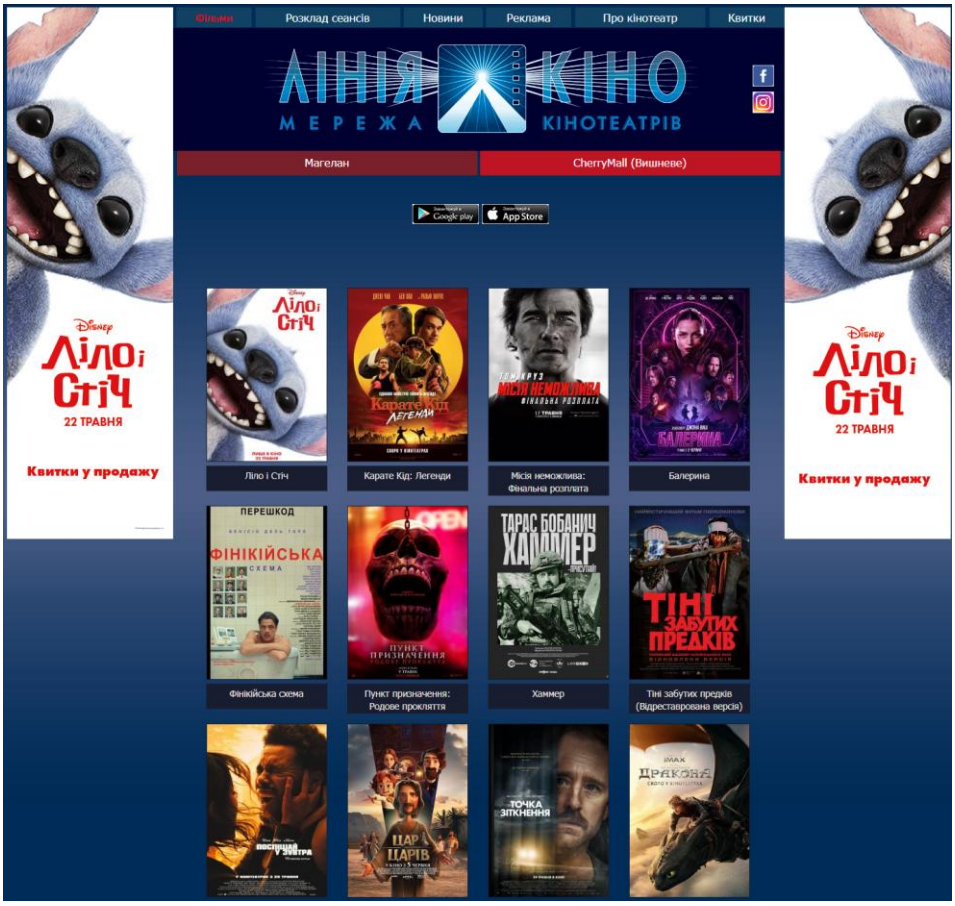


Рисунок 1.1 – Головна сторінка компанії “Linia Kino”

Головна сторінка містить список різноманітних фільмів, постери головного фільму зліва та справа. Меню сайту розташоване вгорі сторінки, де є такі вкладки: «Фільми», «Розклад сеансів», «Новини», «Реклама», «Про кінотеатр», «Квитки». Також є можливість вибрати кінотеатри, перейти в соцмережі «Facebook» та «Instagram», завантажити мобільну версію додатку з «Google Play або «App Store».

Деталі продуктів відображаються на окремих сторінках з докладними фото, цінами та описами. На сторінці фільму є можливість детально ознайомитись з назвою фільму, його жанром, тривалістю, режисером, країною випуску, датою прем’єри та списком найпопулярніших акторів (рис. 1.2). На сторінці також можна побачити короткий опис фільму та кнопку з активним посиланням, яка перенаправить нас на сторінку з сенсами. Якщо вікове обмеження буде присутнє, то воно відобразиться під постером фільму, а якщо ні то буде написано «3А+».

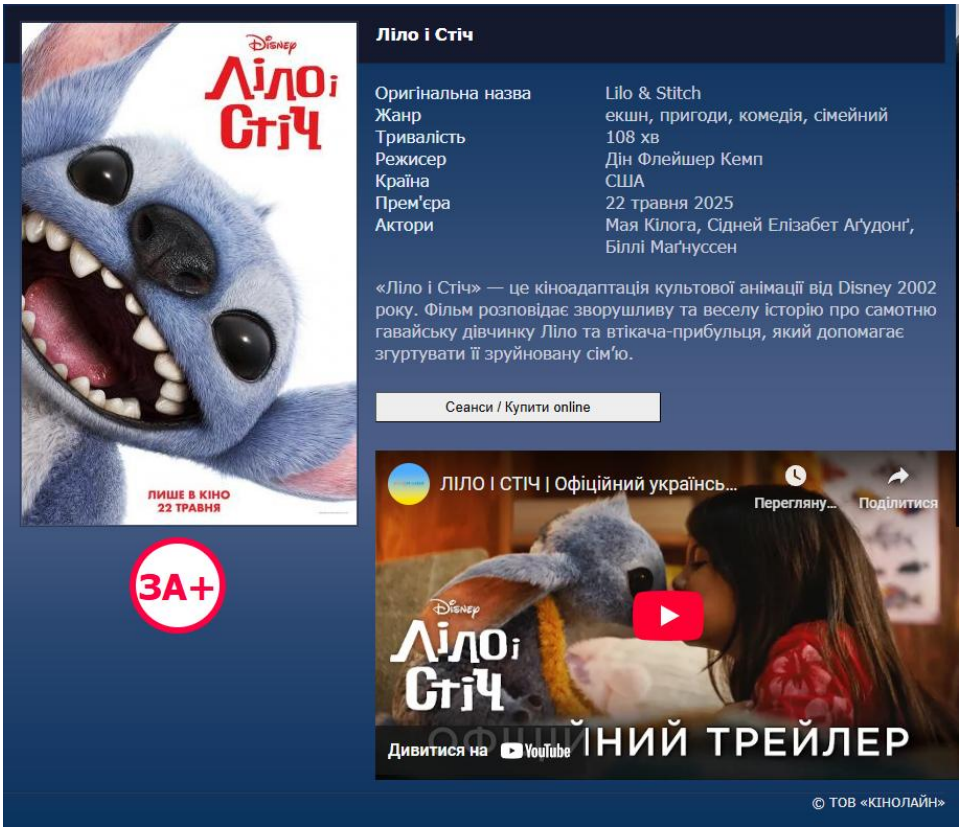


Рисунок 1.2 – Сторінка деталей фільму «Ліло і Стіч»

Тепер перейдемо до аналізу сайту компанії «Lumiere» (рис. 1.3) Загальна стилістика дизайну сайту містить темні відтінки.

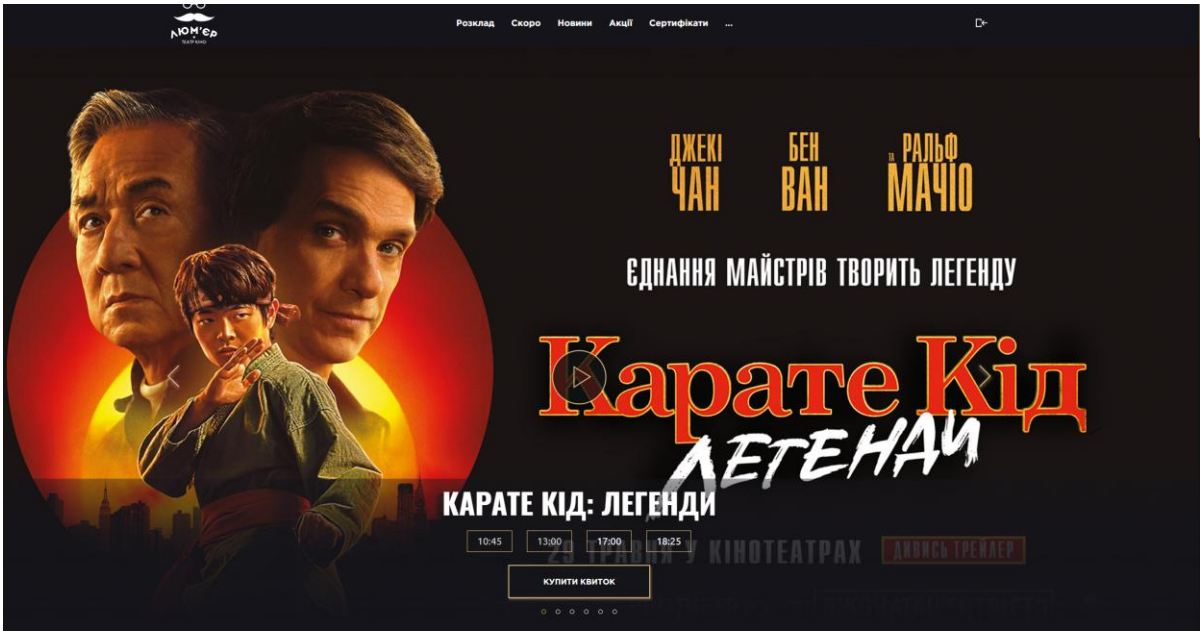


Рисунок 1.3 – Головна сторінка компанії «Lumiere»

На головній сторінці знаходиться постер найпопулярнішого фільму та пропозиції для покупки квитків на найближчу дату. Меню містить вкладки, де користувач може відкрити розклад сеансів, побачити список фільмів які скоро появлятся в прокаті, новини кінотеатру, різноманітні акції на товари та фільми і можливість покупки сертифікатів.

Деталі фільмів можна побачити на окремих сторінках з постером, сеансами і їх фільтрами, описами, трейлерами (рис. 1.4). Тут присутній також блок «Про фільм», де є жанр фільму, його тривалість, рік випуску, країна випуску, режисери, студія та список авторів, що його створили. Інший блок – фільтр, де є періоди(сьогодні, завтра, тиждень, місяць) запису на фільм та формат перегляду фільму(будь-який, 2D, 3D).

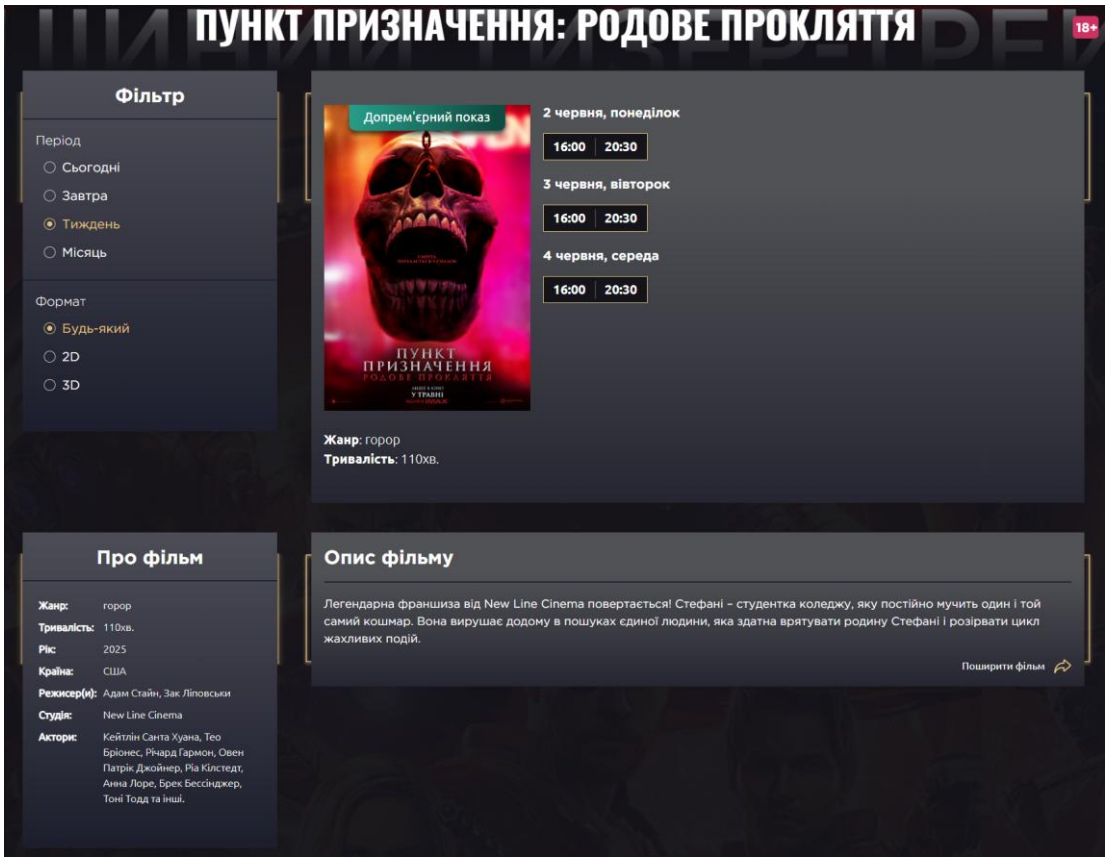


Рисунок 1.4 – Деталі фільму «Пункт призначення: родове прокляття»

Перейдемо до аналізу сайту компанії «Movie Land» (рис. 1.5). Кольорова гама дизайну сайту містить яскраві кольори. По центру сторінки можна побачити список фільмів, які зараз є в прокаті і на них можна купити квиток.

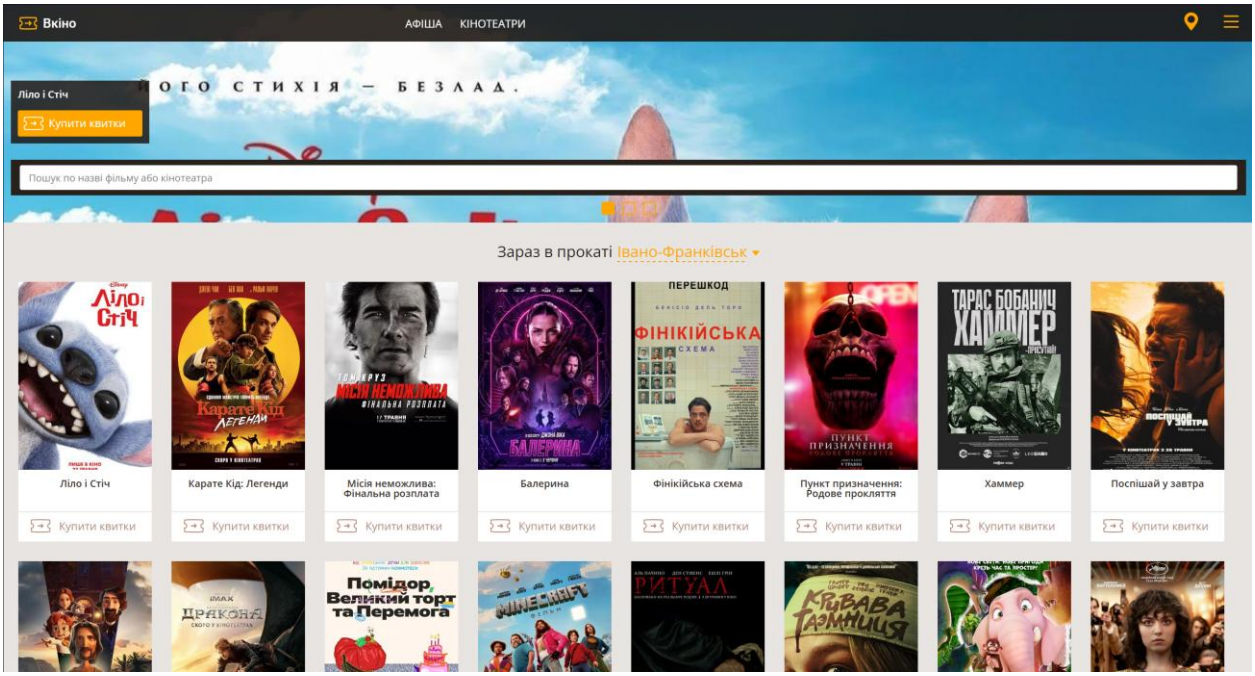


Рисунок 1.5 – Головна сторінка сайту компанії «Movie Land»

У правій верхній частині сайту знаходиться іконка меню, де користувача зустрінуть такі пункти: Мої квитки, Питання-відповідь, Як купити квиток, Повернення квитків, Конфіденційність, Наші контакти та Публічна оферта. Також тут реалізовані пошук фільму по назві. На сторінці деталей фільму «Ліло і Стіч» (рис. 1.6) є назва фільму, жанр, тривалість, вибір дати сеансу, кінотеатру, опис та коментарі.

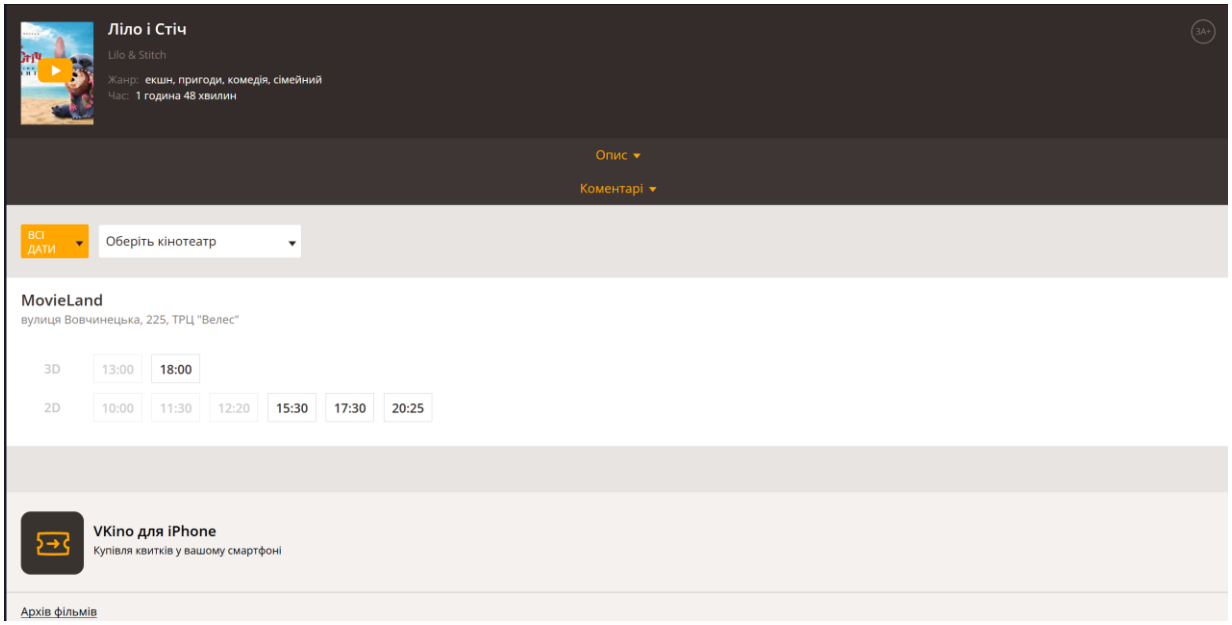


Рисунок 1.6 – Деталі фільму «Ліло і Стіч»

Таблиця 1.1 - Порівняльна таблиця трьох компаній

Функціональність	Назва компанії		
	Лінія Кіно	**Люм'єр**	**Movie Land**
Наявність найновіших фільмів	Так	Так	Так
Інформація про фільми та трейлери	Так	Так	Так
Пошук фільмів по назві	Ні	Ні	Так
Можливість онлайн-оплати та бронювання квитків	Так	Так	Так
Додаткові послуги(оренда залів, кафе тощо)	Так	Так	Ні
Відгуки та оцінка фільмів	Так	Ні	Ні

В таблиці показані основні функції для сайту кінотеатрів, які можуть бути присутніми або відсутніми.

Отже, всі три сайти містять такі спільні елементи:

- Наявність найсвіжіших фільмів, які нещодавно вийшли в прокат.
- Інформація про фільми та трейлери є на кожному з сайтів.
- Можливість онлайн-оплати та бронювання квитків.
- Можливість вибору місць у залі.

На основі пройденого огляду вищевказаних сайтів зроблено висновки, дизайн сайту кінотеатру буде складатися з кольорової гами, в основі якої будуть яскраві світлі кольори, що надасть користувачу відчуття простору та легкості. Дизайн веб-сайту реалізований у світлій кольоровій гамі, яка забезпечує комфортне візуальне сприйняття контенту та інтерфейс у стилі мінімалізму, що дозволяє користувачу швидко орієнтуватися на веб-сайті.

1.3 Постановка завдання на проєкт

При постановці завдання сформовано вимоги, притримуючись яких буде розроблено основна структуру та функціонал сайту. Після успішного проведення

аналізу веб-сайтів, які є конкурентоспроможними у сфері кінотеатрів, прийняте рішення про початок розробки проєкту під назвою PREMIERE. Виокремлено основні плюси та мінуси кожного з 3 варіантів, в результаті чого було сформовано найоптимальніший варіант, який буде найкраще виконувати поставлені задачі для веб-сайту кінотеатру. Метою роботи є розробка веб-сайту , де користувач зможе купити квитки на фільм і отримати інформаційну довідку про кожен з них. Щоб це реалізувати, потрібно вирішити такі задачі:

- Опираючись на технічне завдання, розробити дизайн майбутнього сайту;

- Здійснити опис логічної моделі бази даних, структуру її таблиць та зв’язків між таблицями;

- Здійснити розробкмау структури бази даних.

Задачі, які повинні вирішуватись:

- Реалізувати функціональну панель адміністратора, в якій користувач з роллю адміністратор матиме змогу змінювати інформацію про фільми, акторів, продюсерів, кінотеатри, видаляти старі і додавати нові фільми, кінотеатри, продюсери та актори банерів на сайті, переглядати замовлення усіх користувачів та бачити список користувачів та їх електронні пошти;

- Реалізувати механізм легкої можливості оформити покупку квитків;

- Розробити зручний та інтуїтивно зрозумілий графічний інтерфейс, що забезпечить ефективність використання функціоналу веб-сайту. Отже, основний функціонал на сайті складається з:

1. Header (верхня частина сайту). Даний розділ містить:

- назву веб-сайту;

- сторінки ("Фільми", "Профіль", "Замовлення");

- пошук(за назвою фільму);

- кошик з кількістю товарів в ньому;

- налаштування(у адміністратора є можливість редагувати дані кінотеатрів, продюсерів та акторів).

2. Footer (нижня частина сайту). Даний розділ містить:

- назву продукту;

- авторські права;

3. Головна сторінка (контент сайту). Даний розділ містить:

- Блок фільмів(кожен фільм тут має свої властивості, такі як назва, опис, продюсер, кінотеатр, статус, початок і кінець прокату, актори, фото).

4. Сторінка деталей фільму. Даний розділ містить:

- Назву фільму;

- Фото;

- Опис;

- Кінотеатр;

- Продюсер;

- Актори;

- Початок прокату;

- Кінець прокату;

5. Замовлення користувача:

- Назва товару;

- Ціна товару;

- Кількість товару;

- Фінальна сума.

6. Поле пошуку фільмів. Даний розділ містить:

- Поле для вводу;

- При вводі тексту находить усі фільми, які починаються з введеної фрази

7. Сторінка входу в акаунт:

- Електронна пошта користувача(при вводі неіснуючої виб’є помилку «Такої пошти не існує»;

- Пароль (при вводі неправильного пароля, який прив’язаний до конкретної пошти, покаже помилку «неправильно введений пароль»).

8. Сторінка реєстрації:

- Поле вводу повного імені користувача(від 5 до 30 символів);

- Поле вводу електронної пошти(пошта повинна закінчутись «@....com»);

- Поле вводу пароля(пароль повинен бути добре захищений – містити малу, велику літери, символ і цифру);

- поле вводу пароля(пароль повинен бути добре захищений – містити малу, велику літери, символ і цифру);

- поле вводу повторного пароля(потрібен задля безпеки вводу паролю, який користувач має ввести).

2 РОЗРОБКА СТРУКТУРНИХ РІШЕНЬ ВЕБ-САЙТУ ПРОДАЖУ КВИТКІВ

2.1 Розробка структури та функціоналу веб-сайту продажу квитків

У більшості випадків для веб-сайтів інформаційної підтримки онлайн-продаж застосовують деревовидну структуру, яка передбачає поглиблення при переході на кожен наступний рівень . Під час роботи з сайтом користувач має змогу рухатись в межах певного напрямку, що є обмежений так званою «гілкою», а зміна «гілки» можлива тільки при поверненні до «кореня».

Також можна зустріти структури типу павутини, яка передбачає, що кожна сторінка застосунку має посилання на інші сторінки. Мається на увазі, що користувач з будь якої сторінки може перейти на будь яку сторінку.

Аналізуючи розроблений дизайн сторінок веб-сайту і попередньо проаналізовані аналоги, можна дійти висновку, що найбільш поширеними у даному випадку є деревовидні структури, що бере свій початок з головної(«кореневої») сторінки, від якої йдуть різні категорії «гілки».

Наприклад, головна сторінка сайту має сторінки-«гілки», такі як "Замовлення", "Деталі фільму", "Корзина" тощо. Нижче розглянемо структуру сайту з продажу квитків, яка є деревовидною, тому що в нашому випадку цей варіант ідеально підійде для реалізації вище поставлених задач.

Крім того, веб-сайт обмежує функціонал користувачу, який не увійшов акаунт. Він матиме змогу переглядати деталі фільмів, але купити квиток або зайти в замовлення чи вийти з акаунта не матиме можливості. При таких діях його перенаправить на сторінку авторизації в акаунт. Тому типовий функціонал веб-сайту майже не є можливим реалізувати без використання баз даних , в яких зберігається інформація про основні сутності: фільми, кінотеатри, актори та продюсери.

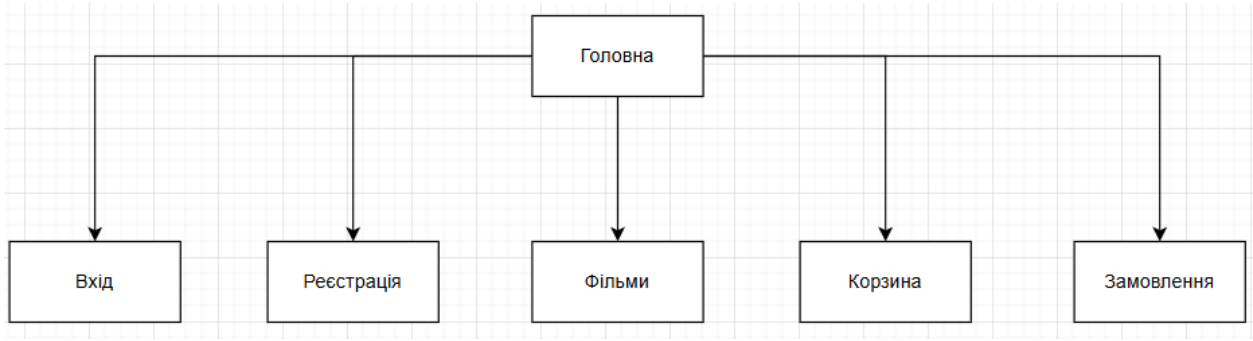


Рисунок 2.1 - структура сайту з продажу ювелірних прикрас

2.2 Розробка таблиць та структури бази даних

Наступним кроком для успішного створення веб-сайту для продажу квитків є створення таблиць бази даних(у нашому випадку реляційної), які будуть заповнені даними про сутності, що потрібні для правильної роботи системи інформаційного супроводу та організації продажів.

Для створення бази даних використано SQL-інструкцію CREATE SCHEMA і написана така команда:

```
CREATE SCHEMA IF NOT EXISTS `Premiere` DEFAULT CHARACTER SET utf8mb4 ;
```

Спочатку опишемо таблиці кінотеатрів, акторів, продюсерів, оскільки їхні ключі будуть виступати зовнішніми в основній таблиці фільмів. . Зв’язок між залежними таблицями буде реалізовано використовуючи первинні і зовнішні ключі. Для початку створимо таблицю кінотеатрів і заповнимо її властивостями, які вони мають. Ця таблиця містить ідентифікатор кінотеатру, що є первинним ключом типу, логотип кінотеатру, назву кінотеатру та опис кінотеатру. Ідентифікатор типу int , бо це є ціле число і цей тип буде найзручнішим в даному випадку, Logo Nvarchar(200), бо посилання може бути різної довжини і оптимальним числом вибрано саме 200. Назва кінотеатру і його опис типу Nvarchar, оскільки це є найкращим вибором для текстового поля в цьому випадку.

Таблиця 2.1 - Дані про кінотеатри

Назва поля	Тип поля	Опис	Ключ
Id	int	Ідентифікатор кінотеатру	Первинний ключ
Logo	Nvarchar (MAX)	Посилання на логотип	-
Name	Nvarchar (50)	Назва кінотеатру	-
Description	Nvarchar (200)	Опис кінотеатру	-

Створення таблиці Cinemas в базі даних здійснюється за допомогою SQL-команди CREATE:

```
CREATE TABLE dbo.Cinemas (
Id INT NOT NULL PRIMARY KEY,
Logo NVARCHAR(MAX) NOT NULL,
Name NVARCHAR(50) NOT NULL,
Description NVARCHAR(200) NOT NULL);
```

Наступною таблицею створимо Producers і пропишемо певні властивості.

Таблиця 2.2 - Дані про продюсерів

Назва поля	Тип поля	Опис	Ключ
Id	int	Ідентифікатор продюсера	Первинний ключ
ProfilePicture URL	Nvarchar(MAX)	Посилання на логотип	-
FullName	Nvarchar(50)	Ім’я і прізвище продюсера	-
Bio	Nvarchar(200)	Опис продюсера	-

У ній буде ідентифікатор, який буде виступати первинним ключем, тому що потім з таблицею Movies буде здійснюватись зв’язок за допомогою цього

ключа. Поля цієї таблиці як і попередньої є досить поширеними для багатьох об’єктів при створенні веб-сайту, а саме: ідентифікатор, ім’я, фото, опис. В цій таблиці передбачено зв’язок з таблицею фільмів – один до багатьох , оскільки один продюсер може приймати участь у створенні багатьох фільмів, а в фільмі може бути тільки один продюсер.

Створення таблиці Producers в базі даних здійснюється за допомогою SQL-команди CREATE:

```
CREATE TABLE dbo.Producers (
Id INT NOT NULL PRIMARY KEY,
ProfilePictureURL NVARCHAR(MAX) NOT NULL,
FullName NVARCHAR(50) NOT NULL,
Bio NVARCHAR(200) NOT NULL);
```

Тепер розглянемо таблицю Actors. У неї з таблицею фільмів реалізовано зв’язок багато до багатьох , тому що один актор може зніматись в багатьох фільмах і в одному фільмі може зніматись декілька акторів. У ній також є первинний ключ, яким виступає поле Id. Поля – ідентифікатор, посилання на логотип, повне ім’я та опис є такі як і в попередніх.

Таблиця 2.3 - Дані про акторів

Назва поля	Тип поля	Опис	Ключ
Id	int	Ідентифікатор актора	Первинний ключ
ProfilePictureURL	Nvarchar(MAX)	Посилання на логотип	-
FullName	Nvarchar(50)	Повне ім’я актора	-
Bio	Nvarchar(200)	Опис актора	-

Створення таблиці Actors в базі даних здійснюється за допомогою SQL-команди CREATE:

```
CREATE TABLE dbo.Actors (
```

```
Id INT NOT NULL PRIMARY KEY,
ProfilePictureURL NVARCHAR(MAX) NOT NULL,
FullName NVARCHAR(50) NOT NULL,
Bio NVARCHAR(200) NOT NULL
);
```

Тепер можна перейти до створення основної таблиці фільмів – Movies. У ній значно більше параметрів, оскільки вона є основною і найбільше часу користувач приділяє перегляду деталей саме фільму.

Таблиця 2.4 - Дані про фільми

Назва поля	Тип поля	Опис	Ключ
Id	int	Ідентифікатор фільму	Первинний ключ
Name	Nvarchar(50)	Назва фільму	-
Description	Nvarchar(50)	Опис фільму	-
Price	float	Ціна квитка	-
ImageURL	Nvarchar(MAX)	Посилання на фото	-
StartDate	Datetime2(7)	Початок прокату	-
EndDate	Datetime2(7)	Кінець прокату	-
MovieCategory	int	Ідентифікатор категорії	Зовнішній ключ
CinemaId	int	Ідентифікатор кінотеатру	Зовнішній ключ
ProducerId	int	Ідентифікатор продюсера	Зовнішній ключ

Actors_Movies – таблиця для реалізації зв’язку Many-To-Many між акторами та фільмами. Тут є два поля-ідентифікатори, які потрібні для того, щоб реалізувати зв’язок Many-To-Many.Спочатку прописуються два ключі, а потім їх об’єднюють і робиться В цьому випадку цей ключ стає складеним(або ще можна назвати композитним).

Таблиця 2.5 – Проміжна таблиця Actors_Movies

Назва поля	Тип поля	Опис	Ключ
MovieId	int	Ідентифікатор фільму	Первинний ключ
ActorId	int	Ідентифікатор актора	Первинний ключ

Створення таблиці Actors_Movies:

```
Create TABLE dbo.Actors_Movies (
MovieId INT NOT NULL, ActorId INT NOT NULL,
CONSTRAINT PK_Actors_Movies PRIMARY KEY (ActorId, MovieId),
CONSTRAINT FK_Actors_Movies_Actors FOREIGN KEY (ActorId) REFERENCES dbo.Actors(Id) ON DELETE CASCADE,
CONSTRAINT FK_Actors_Movies_Movies FOREIGN KEY (MovieId) REFERENCES dbo.Movies(Id) ON DELETE CASCADE);
```

Створимо таблицю AspNetUsers, де будуть користувачі.

Таблиця 2.6 – Таблиця користувачів

Назва поля	Тип поля	Опис	Ключ
Id	int	Ідентифікатор користувача	Первинний ключ
FullName	Nvarchar(MAX)	Повне ім’я	-
Email	Nvarchar(256)	Електронна пошта	-
PasswordHash	Nvarchar(MAX)	Захешований пароль	-

Створення таблиці Users в базі даних здійснюється за допомогою SQL-команди CREATE:

```
CREATE TABLE dbo.Users (
Id INT NOT NULL PRIMARY KEY,
FullName NVARCHAR(50) NOT NULL,
Email NVARCHAR(256) NOT NULL,
PasswordHash NVARCHAR(MAX)
);
```

Розглянемо наступну таблицю для створення – Orders. У ній при створенні замовлення буде передаватися інформація . Таким чином у нас буде працювати функціонал оформлення замовлення користувчем.

Таблиця 2.7 – Таблиця замовлень

Назва поля	Тип поля	Опис	Ключ
Id	int	Ідентифікатор замовлення	Первинний ключ
Amount	int	Кількість товару	-
Email	Nvarchar(256)	Електронна пошта	-
MovieId	int	Ідентифікатор фільму	Зовнішній ключ

Створення таблиці Orders в базі даних здійснюється за допомогою SQL-команди CREATE:

```
CREATE TABLE dbo.Orders (
Id INT NOT NULL PRIMARY KEY,
Amount INT NOT NULL,
EMAIL NVARCHAR(50) NOT NULL,
MovieId INT NOT NULL,
CONSTRAINT PK_Orders PRIMARY KEY CLUSTERED (Id),
CONSTRAINT FK_Orders_Movies_MovieId FOREIGN KEY(MovieId)
REFERENCES dbo.Movies(Id) ON DELETE CASCADE
);
GO
CREATE NONCLUSTERED INDEX IX_Orders_MovieId
ON dbo.Orders (MovieId ASC);
```

Розглянемо процес створення останньої таблиці – ShoppingCartItems. Це є корзина товарів, які користувач додає. Структура її схожа до таблиці Orders, просто корзина містить товари , які користувач може видаляти і добавляти , а замовлення це уже сформовані товари.

Таблиця 2.8 – Таблиця корзини

Назва поля	Тип поля	Опис	Ключ
Id	int	Ідентифікатор замовлення	Первинний ключ
Amount	int	Кількість товару	-
MovieId	int	Ідентифікатор фільму	Зовнішній ключ

Створення таблиці ShoppingCartItems в базі даних здійснюється за допомогою SQL-команди CREATE:

```
CREATE TABLE dbo.ShoppingCartItems (
Id INT NOT NULL PRIMARY KEY,
Amount INT NOT NULL,
MovieId INT NOT NULL);
CREATE NONCLUSTERED INDEX IX_ShoppingCartItems_MovieId ON dbo.ShoppingCartItems (MovieId ASC);
```

На основі розроблених таблиць створено базу даних реляційного типу на рисунку 2.2.

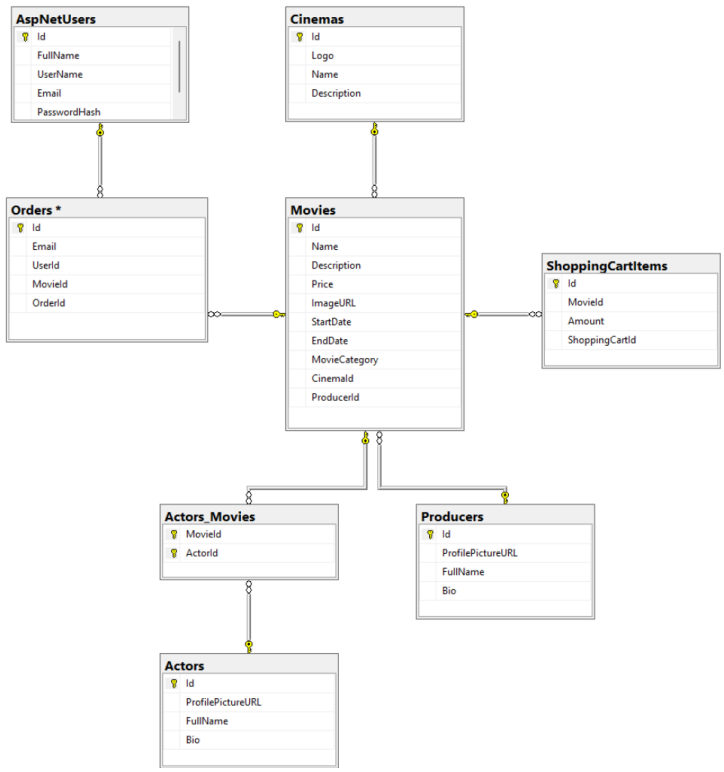


Рисунок 2.2 - Структура бази даних

Як ми бачимо, усі таблиці пов’язані між собою за допомогою первинних та зовнішніх ключів та продемонстровано поля кожної таблиці.

2.3 Розробка діаграм послідовності взаємодії компонентів сервісу

Під час розробки веб-сайту продажу квитків потрібно здійснити аналіз взаємодії користувача з програмою на різних етапах її функціонування. У нас передбачено 2 сценарія послідовності.

Спочатку розглянемо послідовність взаємодії користувача при реалізації функції перегляду фільмів (рис. 2.3)

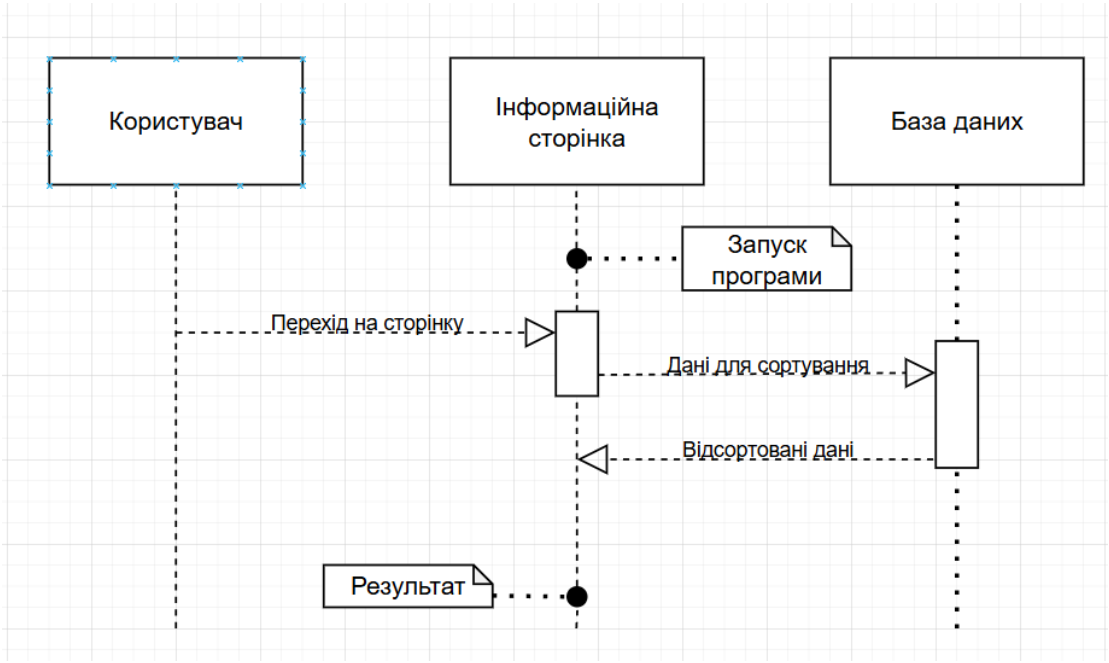


Рисунок 2.3 – Діаграма послідовності перегляду фільмів

Користувач при вході на сайт потрапляє на головну сторінку, де він побачить список фільмів, які є у доступі. Додаток отримує запит від користувача і відображає список фільмів, які або доступні, або закінчені, або очікуються. Також користувач має можливість подивитись деталі фільму, який йому сподобається, натиснувши на кнопку «Деталі фільму». Йому відкриється сторінка з повною інформацією про вибраний фільм, а саме: фото, назва, опис,

кінотеатр, актори, продюсер, категорія, ціна. Передбачена функція повернення до усіх фільмів , яка під'єднана до кнопки «Назад до фільмів».

Діаграма послідовності перегляду інформації на сайті продажу квитків демонструє послідовність дій користувача та системи під час перегляду списку фільмів і деталей окремого фільму з поверненням до списку усіх фільмів.

Тепер розглянемо порядок дій користувача при пошуку фільмів (рис. 2.4) Користувач, який при відкриванні сайту захоче швидше найти бажаний фільм, здійснить пошуковий запит на веб-сайті шляхом вводу назви в поле для вводу. В цей час асихронно система отримує запит і відображає список фільмів, які задовільняють умови пошуку.

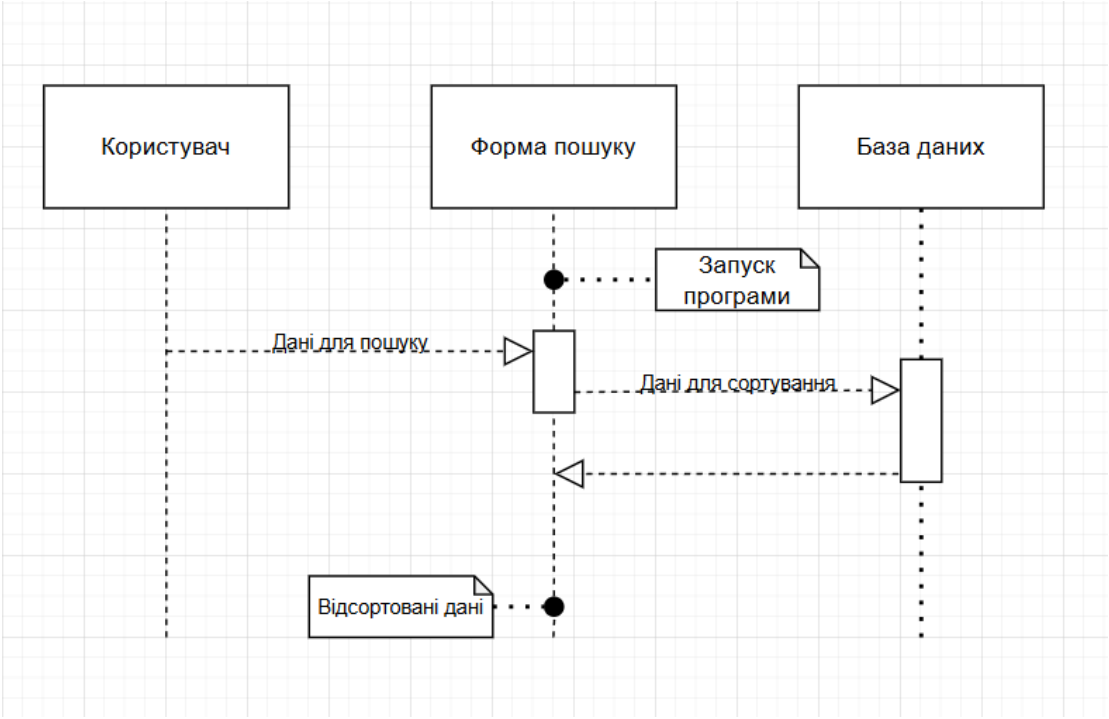


Рисунок 2.4 – Діаграма послідовності пошуку фільма

Зрештою система здійснює пошук результатів в таблиці фільмів, де назва фільму буде сходитись декількома або усіма літерами з тією, що ввели. Після цього користувача перенаправляє на сторінку результатів, де буде відображено список фільмів, які задовільняють умову пошуку. Якщо введена повна назва, то результатом буде 1 фільм.

Таким чином, реалізація попередньо згаданих діаграм послідовностей, які відображають взаємодію з користувачем, потрібна для повного функціонування веб-сайту.

2.4 Розробка USE-CASE діаграми взаємодії з користувачем

Спираючись на структуру веб-сайту продажу квитків, структури бази даних, а також діаграм послідовностей дій з користувачем, здійснено розробку USE-CASE діаграми для акторів(учасників взаємодії), що згруповано і описано в таблиці 2.9.

Таблиця 2.9 - Опис акторів

Назва актора	Опис
Користувач	Користувач веб-сайту продажу квитків
Адміністратор	Користувач, що виконує функції «адміністратора»

Також розглянуто можливі варіанти використання з функціоналом, який є доступним для наявних акторів, таблиця 2.10. Тут описано дії, які може виконати актор (користувач) залежно від його ролі. Авторизація потрібна для усіх акторів.

Таблиця 2.10 - Опис варіантів використання

Назва	Опис
Авторизація	Функція входу в систему
Перегляд фільмів	Перегляд списку фільмів на сайті
Пошук фільму	Можливість пошуку фільму
Реєстрація	Функція створення акаунта
Додавання товару в кошик	Функція додавання товару в кошик і подальше оформлення замовлення
Редагування даних	Адміністратор може редагувати дані сайту
Перегляд замовлень	Адміністратор може переглядати замовлення
Оформлення замовлень	Створення замовлення на сайті

Для початку розглянемо роль адміністратора на веб-сайті продажу квитків, який може керувати різними функціями веб-сайту на рисунку 2.5.

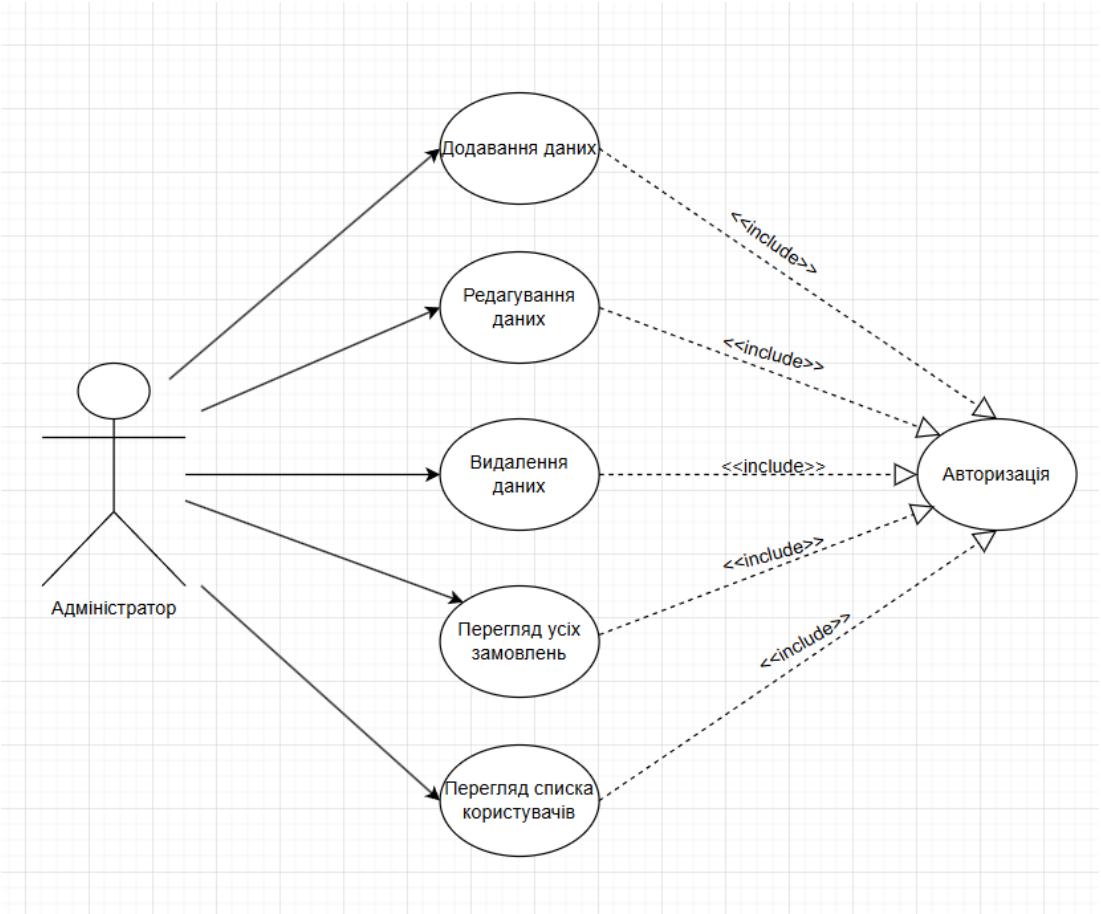


Рисунок 2.5 – USE-CASE діаграми функцій адміністратора

Адміністратор:

• Додавання даних: Адміністратор може додавати нові фільми, кінотеатри, продюсерів та акторів.;

• Редагування даних: Адміністратор може редагувати інформацію про фільми, кінотеатри, продюсерів та акторів;

• Видалення даних: Адміністратор може видаляти інформацію про фільми, кінотеатри, продюсерів та акторів;

• Перегляд замовлень: Адміністратор отримує доступ до інформації про замовлення;

• Перегляд списку користувачів: Адміністратор має можливість побачити інформацію про користувачів(електронна пошта та ім’я).

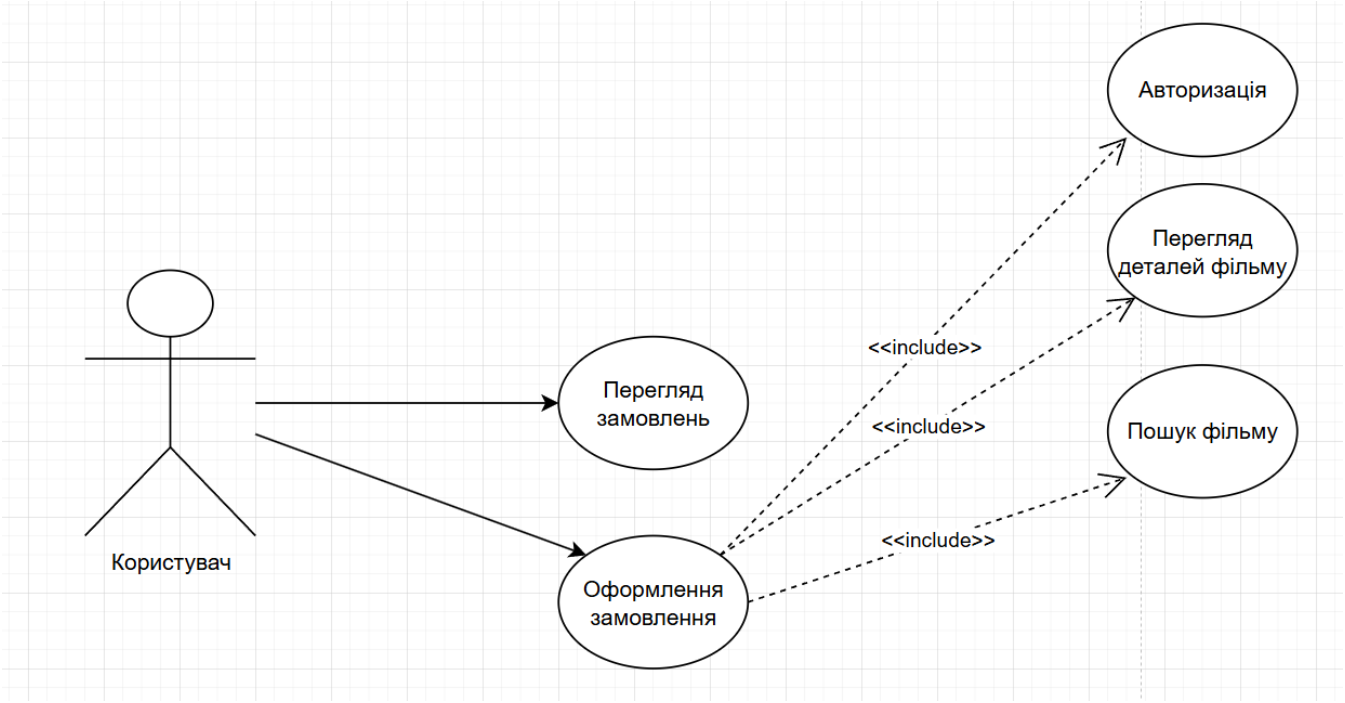


Рисунок 2.6 - Use-case діаграма роботи користувача

Користувач:

- Відкриває веб-браузер і вводить адресу веб-сайту продажу квитків;
- Переглядає основну сторінку з фільмами;
- Здійснює пошук фільму за його назвою;
- Обирає фільм для перегляду детальної інформації про нього;
- Додає квитки на фільм в корзину;
- Підтверджує замовлення і отримує підтвердження про успішну покупку.

3 РЕАЛІЗАЦІЯ ГРАФІЧНОГО ІНТЕРФЕЙСУ ТА БАЗИ ДАНИХ ВЕБ-САЙТУ ПРОДАЖУ КВИТКІВ

3.1 Реалізація інтерфейсу та функціоналу веб-сайту продажу квитків

Щоб створити проєкт, вибрано шаблон в Visual Studio ASP . NET MVC. Він містить набір файлів і папок, які потрібні, щоб створити і запустити сайт. Архітектурний шаблон MVC передбачає три основні папки: Models, Views, Controllers. У папці Models зберігаються сутності проєкту, а саме їх властивості і також сутності, які допомагають при створенні зв'язків в таблиці.

Фундамент програми находиться в файлі Program. cs. Звідси здійснюється запуск програми, здійснення логування для можливості при помилці оперативно бачити що і коли сталося.

Бібліотеки , які використовуються в цьому файлі:

```
using eTickets.Data;
using eTickets.Data.Cart;
using eTickets.Data.Services;
using eTickets.Models;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using NLog.Web;
```

Папка eTickets.Data містить усі допоміжні файли та структури, які необхідні для коректного функціонування та запуску веб-проєкту (рис. 3.1). Вона виконує роль ядра логіки даних і взаємодії між різними частинами системи.

BaseRepository потрібен для того щоб прописати логіку для CRUD операцій основних сутностей. Таким чином , коли треба поміняти логіку виконання певних операцій, у нас не буде проблем , що код треба буде всюди міняти або зміна логіки поламає весь код. Змінивши в EntityBaseRepository метод, він зміниться в класах, які його наслідують(такі як Movies).

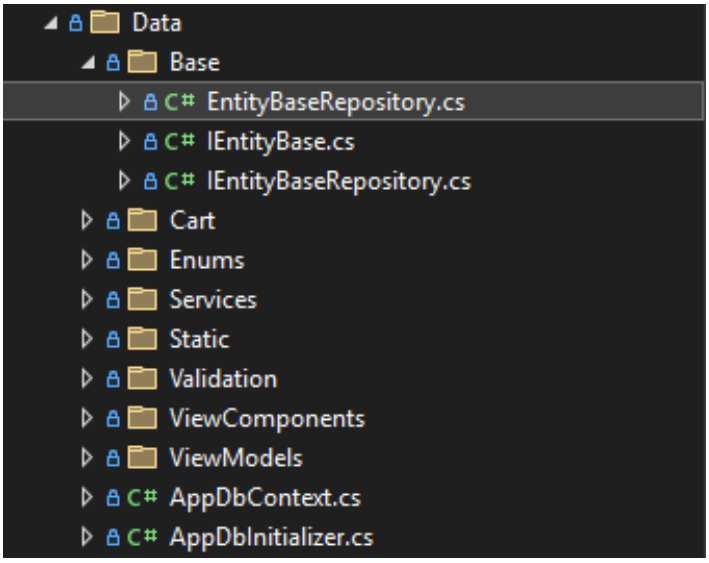


Рисунок 3.1 – Структура папки Data

В папці Cart прописана логіка роботи корзини (додавання, перегляд, підрахунок загальної суми). Папка Enums містить файл MovieCategory.cs , у якому прописані категорії фільмів. В папці Services – сервіси для усіх сутностей. Сервіси допомагають уникати повторювання коду, оскільки у нас багато сутностей мають схожі дії(CRUD-операції). Папка Static містить файл UserRoles.cs у якому прописані 2 ролі – користувач і адміністратор. У папці Validation є файл для валідування паролю:

```
    public PasswordValidator1()
    { RuleFor(x => x.Password).Length(5,
15).Must(HasValidPassword);}
    private bool HasValidPassword(string pw)
    {
        var lowercase = new Regex("[a-z]+");
        var uppercase = new Regex("[A-Z]+");
        var digit = new Regex("(\\d)+");
        var symbol = new Regex("(\\W)+");
        return (lowercase.IsMatch(pw) && uppercase.IsMatch(pw) &&
digit.IsMatch(pw) && symbol.IsMatch(pw));
    }
```

Пароль повинен містити малі, великі букви, символи і цифри, щоб

користувач зміг за допомогою нього зареєструватися.

ViewModels – містить 4 файли: LoginVM, RegisterVM, ShoppingCartVM, NewMovieDropdownsVM. Ці елементи використовуються для того, щоб відділяти представлення(Views) від бізнес логіки. Оскільки деякі сутності можуть містити багато зайвих або конфіденційних полів, які не потрібно показувати на сторінці, то цей підхід ідеально підійде для цієї роботи.

Далі в файлі Program.cs здійснюється ініціалізація програми, створення білдера(об'єкт, який налаштовує та створює веб-застосунок):

```
logger.Debug("Initialization of program");
var builder = WebApplication.CreateBuilder(args);
builder.Host.UseNLog();
// Add services to the container.
builder.Services.AddDbContext<AppDbContext>(options => options.
UseSqlServer(builder.Configuration.GetConnectionString("DefaultCon
nection")));
builder.Services.AddControllersWithViews();
builder.Services.AddScoped<IActorsService, ActorsService>();
builder.Services.AddScoped<IMoviesService, MoviesService>();
builder.Services.AddScoped<IProducersService, ProducersService>();
```

Тут здійснюється реєстрація інтерфейсів сервісів і їх реалізація для роботи з акторами, фільмами і іншими сутностями. Також тут реєструють аутентифікацію та авторизацію для користувачів, створюється застосунок в рядку: `var app = builder.Build();`

Зрештою наприкінці прописується код, щоб накінець використовувати функціонал, який попередньо зареєстровано і прописано:

```
app.UseHttpsRedirection();
app.UseStaticFiles();app.UseRouting();
app.UseSession();
app.UseAuthentication();
app.UseAuthorization();
app.UseAuthorization();
app.MapControllerRoute(name:"default",pattern:"{controller=Movies}
```

```
/{action=Index}/{id?}");
 AppDbInitializer.Seed(app);
 await AppDbInitializer.SeedUsersAndRolesAsync(app);
 app.Run();
```

Як бачимо, добавлено використання статичних файлів, роутинг (найменування певних сторінок, щоб мати можливість називати посилання до кожної сторінки), прописано базову сторінку при вході і нарешті запуск програми.

Тепер розглянемо вигляд основної сторінки сайту (рис. 3.2). На ній можемо побачити список фільмів і дані про них: назва, опис, кінотеатр, категорія, початок, кінець прокату і статус. Є 3 види статусу: доступно, очікуваний, закінчений. Коли фільм закінчений, користувач не має змоги купити квиток і кнопка стає неклікабельною.

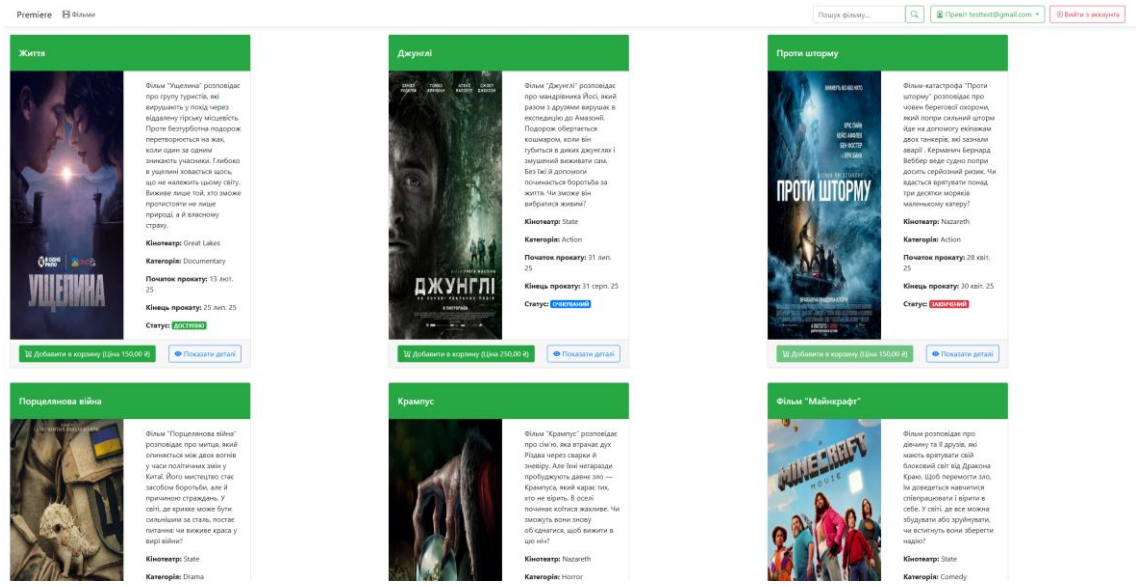


Рисунок 3.2 – Вигляд основної сторінки

У проєкті файл MoviesController відповідає за весь функціонал, який може виконувати фільм. CRUD операції тут виконані і їх може застосовувати на сайті тільки адміністратор.

Код методу Index() контролера MoviesController:

```
public async Task<IActionResult> Index()
```

```
{
    var allMovies = await _service.GetAllAsync(n=>n.Cinema);
    return View(allMovies);
}
```

Наведений вище код реалізовує функціонал для виводу усіх фільмів на основну сторінку. Рядок “return View(allMovies);” передає усі фільми, які згодом циклічно виведуться на основній сторінці. Контролер передбачає також перегляд деталей кожного фільму:

```
public async Task<IActionResult> Details(int id)
{
    var movieDetail = await _service.GetMovieByIdAsync(id);
    return View(movieDetail);
}
```

В окремому сервісі попередньо прописано логіку, щоб витягувати ідентифікатор фільму асихронно, що дозволяє швидко перенести користувача на сторінку з детальним описом фільму. В подальшому “return View();” буде використовуватись щоб викликати певні сторінки з передачою певних даних.

Наступний метод створено для пошуку фільмів за назвою:

```
[AllowAnonymous]
 public async Task<IActionResult> Filter(string searchString)
 {
     var allMovies = await _service.GetAllAsync(n => n.Cinema);
     if (!string.IsNullOrEmpty(searchString))
     {
        var filteredResult = allMovies.Where(n =>
n.Name.ToLower().Contains(searchString.ToLower()));
         return View("Index", filteredResult);
     }
     return View("Index", allMovies);
 }
```

У ньому використано LINQ – мову запитів до структурованих даних, яку створили в C#. Вона значно спрощує точну вибірку інформації з різними умовами, що можна вмістити в один рядок. В цьому випадку це рядок «var

`filteredResult = allMovies.Where(n => n.Name.ToLower().Contains (searchString.ToLower()));`» В ньому з усіх фільмів вибирається той, у якого ім’я буде збігатися буквами з тим, яке ввів користувач. Тобто навіть якщо користувач введе 2-3 букви , то покаже усі фільми , які містять ці букви (рис. 3.3)

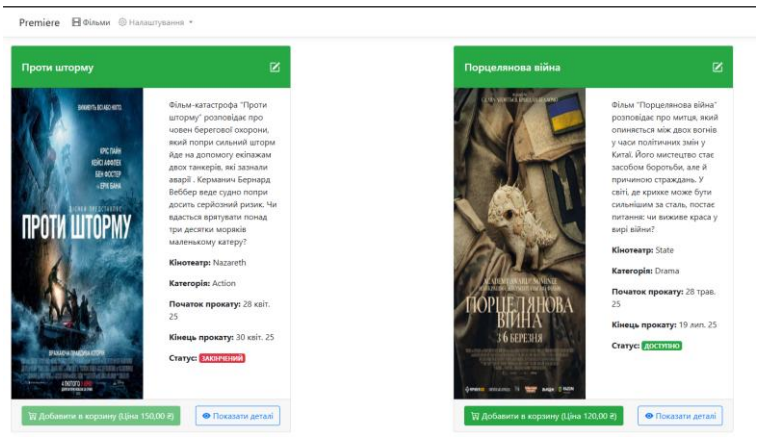


Рисунок 3.3 – Результат після пошуку фільмів, які починаються з «П»

Наступною сторінкою розглянемо «Деталі фільму» (рис. 3.4)

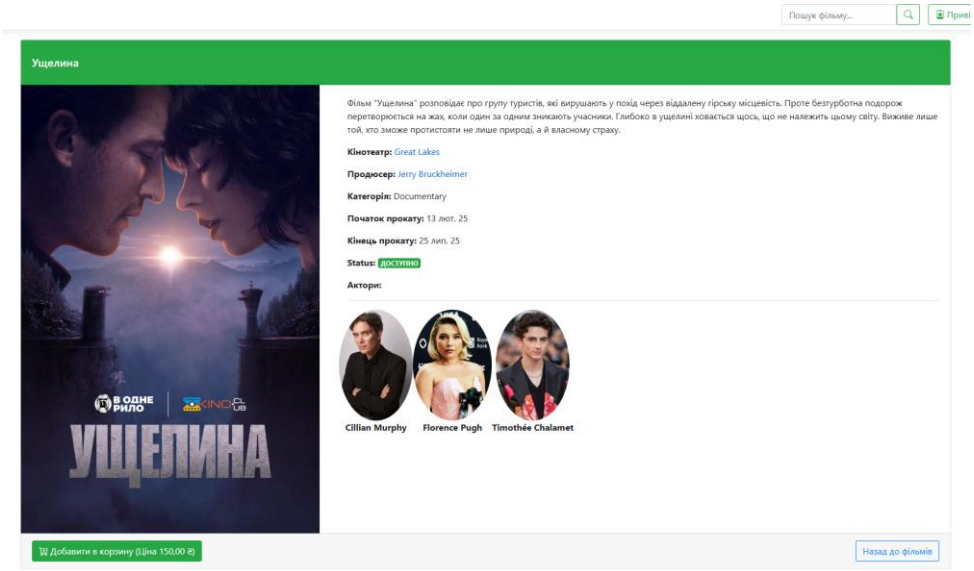


Рисунок 3.4 – Вигляд сторінки фільму

Крім інформації , яка є на основній сторінці(назва фільму, опис і інші властивості) також тут є фото акторів, які знімались в фільмі і їх імена. Користувач має змогу повернутись на головну сторінку натисканням на кнопку «Назад до фільмів» та добавити квиток на фільм кнопкою «Добавити в

корзину(Ціна 150 грн)».

Код методу Details() класу MoviesController:

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

Майже усі методи додатку є асихронними, оскільки в більшості випадків потрібна швидка відповідь сервера що асихронність позволяє зробити. В метод передається ідентифікатор фільму, щоб потім в методі була можливість найти дані про фільм методом GetMovieByIdAsync() . Потім ми в метод View() передаєм інформацію про фільм, який потрібно відобразити.

У адміністратора є можливість редагування даних про фільм (рис. 3.5) Тут у нас наявні усі поля , які має фільм(назва, початок і кінець прокату, ціна, посилання на зображення фільму, опис фільму) та вибір кінотеатру, категорії, продюсера та акторів. Також є 3 кнопки: «Оновити», «Видалити» та «Усі фільми».

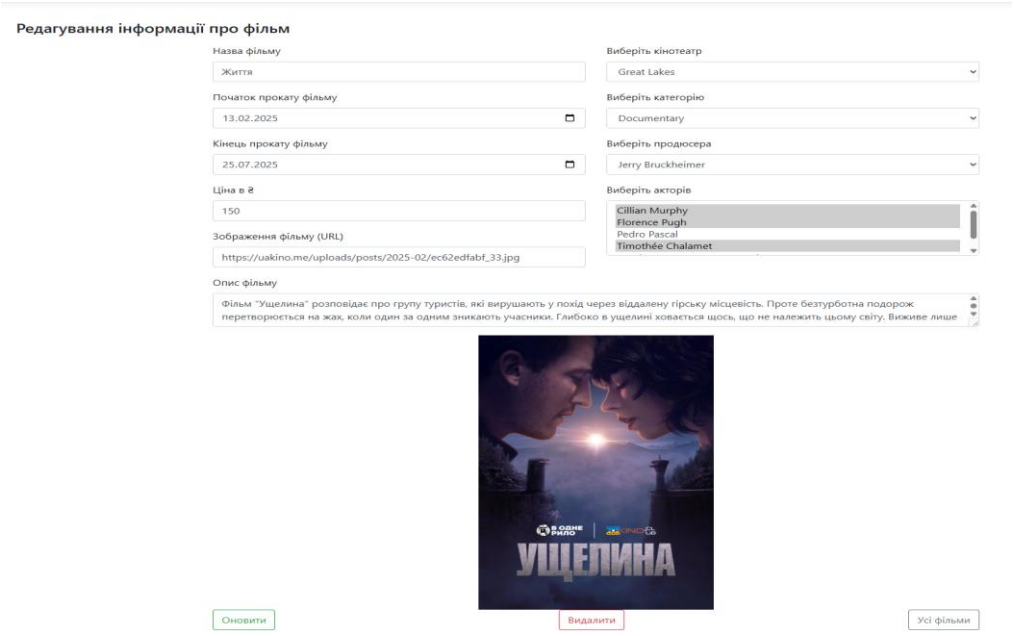


Рисунок 3.5 – Сторінка редагування інформації про фільм

Код методу Edit() класу MoviesController:

```
[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.Cinemas = new SelectList(movieDropdownsData.Cinemas, "Id",
"Name");
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));
}
```

Якщо ж ми захочемо видалити фільм, то нас перекине на сторінку з підтвердженням видалення фільму (рис. 3.6). Тут у нас вказані 3 основні поля фільму та кнопки «Назад» і «Видалити».

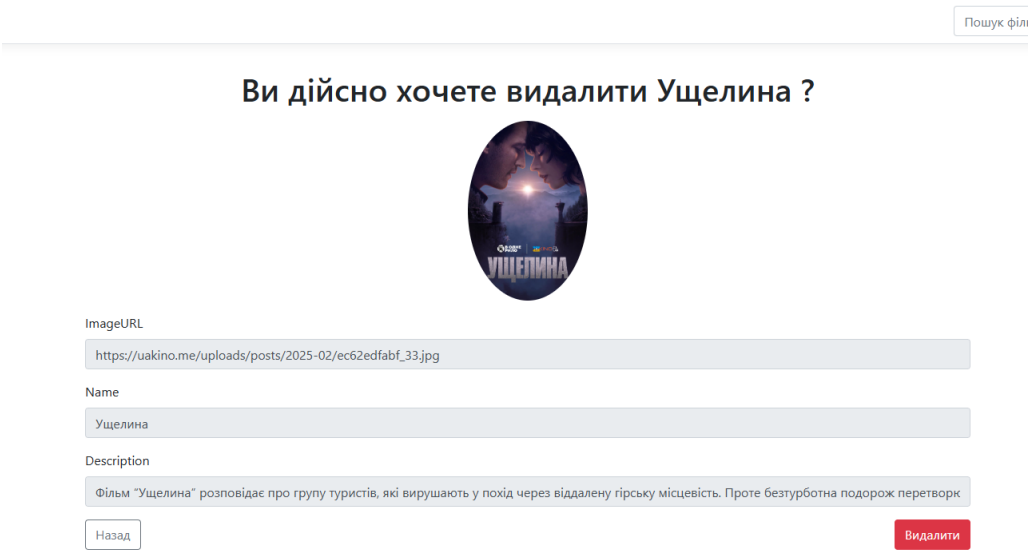


Рисунок 3.6 – Сторінка підтвердження видалення фільму

Код методу Delete(int) класу MoviesController , де здійснюється видалення інформації про фільм:

```
[HttpPost, ActionName("Delete")]
 public async Task<IActionResult> DeleteConfirm(int id)
 {
      var movieDetails = await _service.GetByIdAsync(id);
      if (movieDetails == null) return View("NotFound");
      await _service.DeleteAsync(id);
      return RedirectToAction(nameof(Index));
       }
```

При натисканні на кнопку “Добавити в корзину(Ціна 150 грн))” при не залогіненому статусі, нас перекине на сторінку входу в акаунт (рис. 3.7)

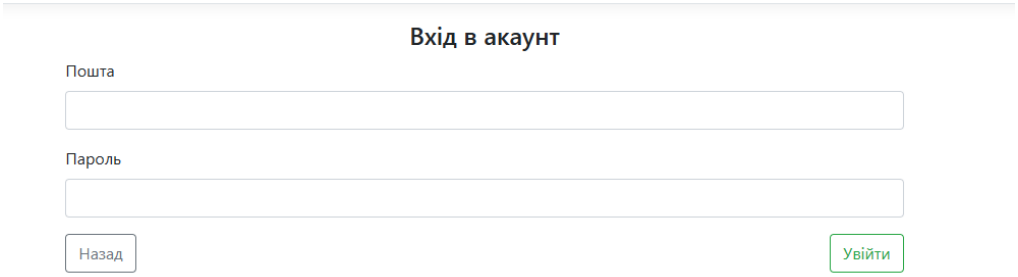


Рисунок 3.7 – Вигляд сторінки Login. cshtml

На цій сторінці ми бачимо два поля для вводу: пошта і пароль. Якщо ми введемо дані акаунта, якого немає в базі, нам видасться результат (рис. 3.8).

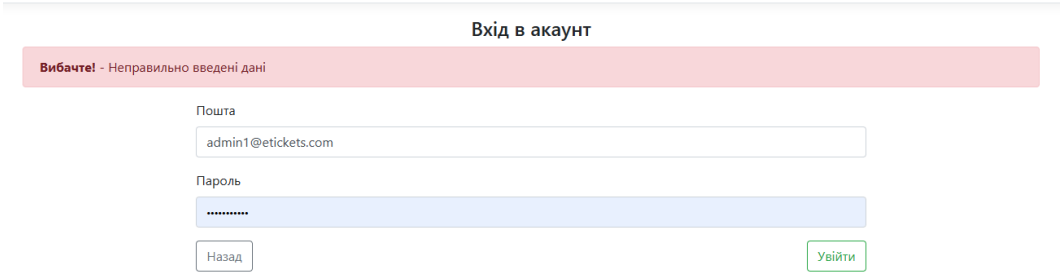


Рисунок 3.8 – Вигляд сторінки у разі вводі неправильної інформації

Вся логіка, яка зв'язання з процесом входу в акаунт прописана в файлі AccountController в методі Login():

```
[HttpPost]
public async Task<IActionResult> Login(LoginVM loginVM)
{
var user = await   _userManager. FindByEmailAsync
(loginVM.EmailAddress);
if (user != null)
{
var passwordCheck = await _userManager .CheckPasswordAsync(user,
loginVM.Password);
```

Цей метод містить атрибут [HttpPost], оскільки цей метод обробляє запит типу пост, тобто дані, які надсилаються з форми. В метод передається зміна типу LoginVM , яка містить в собі поля логіну і паролю користувача. Створюється зміна user, в якій асихронно здійснюється пошук користувача за поштою, яка передається в метод. Потім якщо користувач містить значення, здійснюється перевірка паролю методом CheckPasswordAsync(), в який передається електронна пошта користувача і введений пароль.

```
if (passwordCheck)
{var result = await _signInManager.PasswordSignInAsync(user,
loginVM.Password, false, false);
if (result.Succeeded)
{return RedirectToAction("Index", "Movies");}
         }
         TempData["Error"] = "Неправильні дані!Спробуйте ще раз!";
 return View(loginVM);
     }
 TempData["Error"] = "Неправильні дані!Спробуйте ще раз!";
     return View(loginVM);
 }
```

У цьому відрізку коду, якщо дані користувача збіглися з даними в базі даних, то здійснюється процес входу в акаунт. В методі PasswordSignInAsync 3 аргумент означає чи потрібно запам’ятати користувача, а 4 – чи заблокувати при

невдачах. Якщо метод відпрацьовує успішно, користувач входить в акаунт, а якщо ні – бачить помилку. Передбачено дві кнопки – «Назад» та «Увійти». Перша здійснює повернення на головну сторінку, а друга -вхід в акаунт.

Наступною розглянемо сторінку реєстрації, на якій є 4 поля вводу: Повне ім’я, електронна пошта, пароль і підтвердження паролю (рис. 3.9). Тут користувачу потрібно ввести в поля вводу дані про себе, для того щоб він мав змогу зареєструвати свій акаунт в системі. Усі дані , які передадуться формою, при успішній реєстрацію попадуть в базу даних і створиться новий акаунт, в який можна буде увійти.

Рисунок 3.9 – Вигляд сторінки реєстрації

В контролері AccountController в методі Register() передбачено різноманітні перевірки при створенні акаунта:

```
[HttpPost]
public async Task<IActionResult> Register(RegisterVM registerVM)
{
    if (registerVM.Password==null)
    {

        return View(registerVM);
    }
    var validator = new PasswordValidator1();
```

```
    var validationResult = validator.Validate(registerVM);
    if (!validationResult.IsValid)
    {
        TempData["Error"] = "Пароль повинен містити малі літери,
великі літери символи та цифри";
    }
    if (!ModelState.IsValid) return View(registerVM);

    var user = await
_userManager.FindByEmailAsync(registerVM.EmailAddress);
    if (user != null)
    {
        TempData["Error"] = "Цю пошту уже використовують!";
        return View(registerVM);
    }
```

Метод як і попередній метод входу є асихронним і з атрибутом [HttpPost] і приймає аргументом RegisterVM, яка містить параметри , які вводить користувач в формі. Якщо пароль не введено, нам покаже помилку (рис. 3.10)

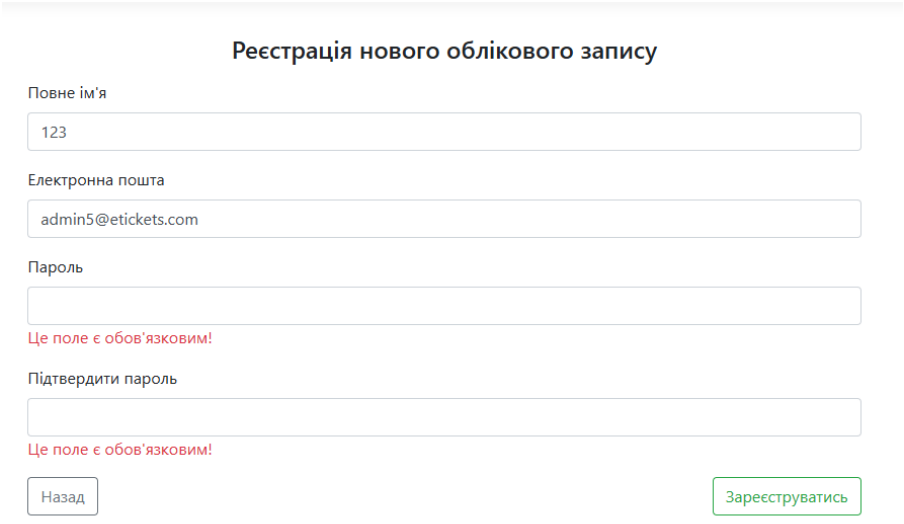


Рисунок 3.10 – Вигляд сторінки при відсутності паролю при реєстрації

Потім створюється валідатор, який буде здійснювати подальшу перевірку паролю на ступінь захищеності. Якщо хоча б одна з умов не буде дотримана (наявність малої та великої літери, цифри і символів), то нам покаже помилку

(рис. 3.11). Це потрібно для того, щоб забезпечити захист облікового запису користувача шляхом додавання в нього великі, малі літери, цифри та символи. Ці дії допоможуть зберегти обліковий запис користувча в безпеці. Також тут здійснюється хешування паролю і коли він уже відхешований, то він відправляється в таблицю користувачів і там зберігається результат хешування.

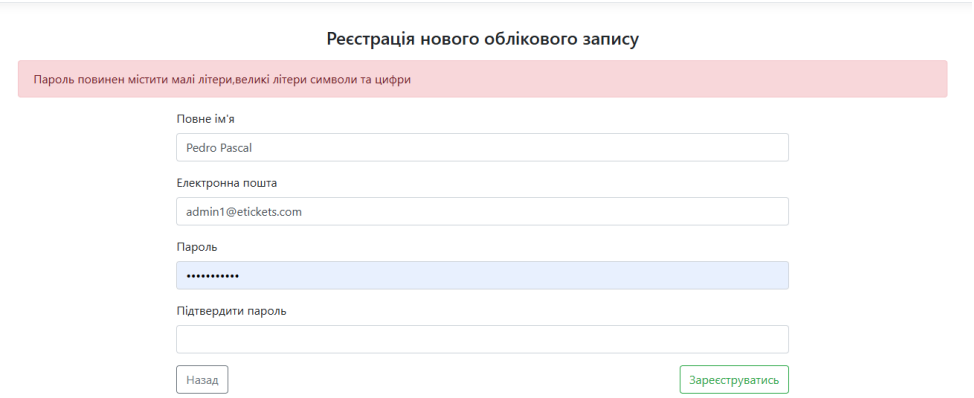


Рисунок 3.11 – Вигляд сторінки при вводі недостатньо захищеного паролю

Наступною дією є створення змінної user , якій передадуть значення пошти, яку користувач вводить в пошті. Якщо це значення не буде null, то користувач отримає помилку «Цю пошту уже використовують». В іншому випадку створюється користувач і йому передають поля, які ввели в формі:

```
var newUser = new ApplicationUser()
 {
     FullName = registerVM.FullName,
     Email = registerVM.EmailAddress,
     UserName = registerVM.EmailAddress
 };
 var newUserResponse = await _userManager.CreateAsync(newUser,
registerVM.Password);
```

Далі виконується створення користувача в системі. Наприкінці методу виконується відрізок коду:

```
if (newUserResponse.Succeeded)
```

```
{
    await _userManager.AddToRoleAsync(newUser, UserRoles.User);
    var result = await
 _signInManager.PasswordSignInAsync(newUser, registerVM.Password,
 false, false);
    if (result.Succeeded)
    {
        return RedirectToAction("Index", "Movies");
    }
    return View(registerVM);
}
return View(registerVM);
```

У разі якщо користувач створився успішно, йому надається роль користувача і здійснюється вхід в акаунт. Після того як ми увійдем в акаунт, у нас видозміниться меню справа зверху сторінки фільмів (рис. 3.12)

Рисунок 3.12– Кнопки виходу з акаунта і привітання

Логіка виходу з акаунта прописана в методі LogOut(), який є асихронним і не приймає параметрів:

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

Після натискання на кнопку “Привіт admin1@etickets. com” нас перекине на сторінку замовлень (рис. 3.13). Тут будуть відображенні актуальні замовлення користувача, якщо такі є.

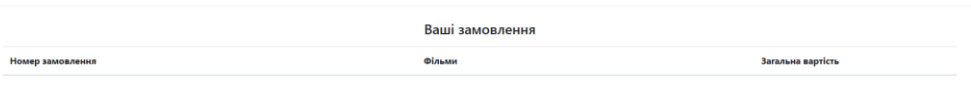


Рисунок 3.13 – Сторінка замовлень

Після додавання квитка в корзину, у нас появиться кількість, назва фільму, ціна і загальна вартість корзини. Також появляться дві кнопки червного і зеленого кольору для додавання або видалення квитка і кількість об’єктів справа зверху (рис. 3.14)

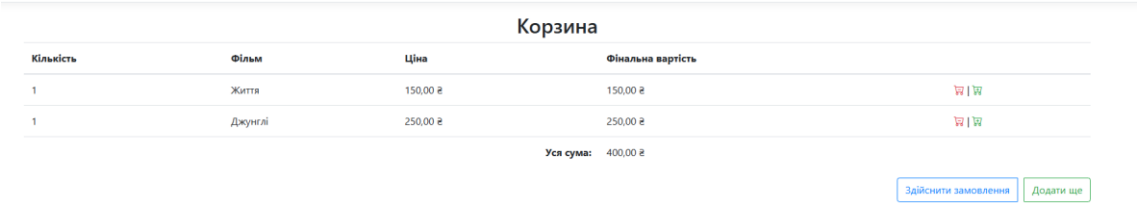


Рисунок 3.14 – Сторінка замовлень при додаванні нового елементу

Основний функціонал корзини прописаний у контролері OrdersController. У всіх контролерів є метод Index(), який відкриває сторінку:

```
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);
 }
```

Тут здійснюється перевірка ролі користувача(якщо адміністратор, він бачить замовлення усіх) і відображається інформація про корзину користувача. Також як і у інших контролерах, тут є CRUD-операції. Додавання елементів прописано в методі AddItemToShoppingCart(int):

```
public async Task<IActionResult> AddItemToShoppingCart(int id)
 {
     var item = await _moviesService.GetMovieByIdAsync(id);
     if (item != null)
     {
         _shoppingCart.AddItemToCart(item);
     }
```

```
    return RedirectToAction(nameof(ShoppingCart));}
```

Тут спочатку передається ідентифікатор фільму, здійснюється пошук фільму і присвоєння інформації про нього в змінну item. Також тут цей метод виконується також при натисканні зеленої кнопки корзини з «+», він асихронно відпрацьовує і в режимі реального часу оновлює дані на сторінці і в базі даних.

Наступним методом розглянемо видалення елементу:

```
public async Task<IActionResult> RemoveItemFromShoppingCart(int
id)
{
    var item = await _moviesService.GetMovieByIdAsync(id);
    if (item != null)
    {
        _shoppingCart.RemoveItemFromCart(item);}
    return RedirectToAction(nameof(ShoppingCart));}
```

Переглянемо корзину після видалення квитка (рис. 3.15):

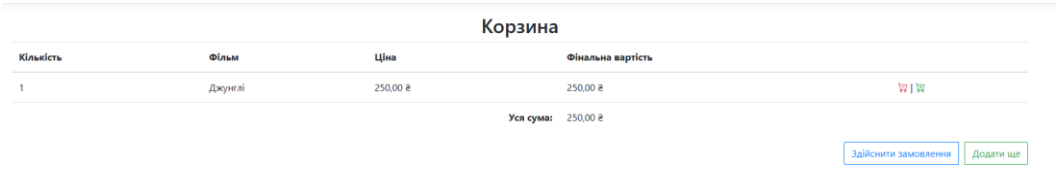


Рисунок 3.15 – Сторінка замовлень при видаленні квитка

Також на цій сторінці присутні кнопки «Здійснити замовлення» та «Додати ще». Перша кнопка сформує замовлення і воно буде відображатись на однойменній сторінці, а друга перекине нас на основну сторінку. При здійсненні замовлення користувач побачить сторінку успішного замовлення (рис. 3.16)

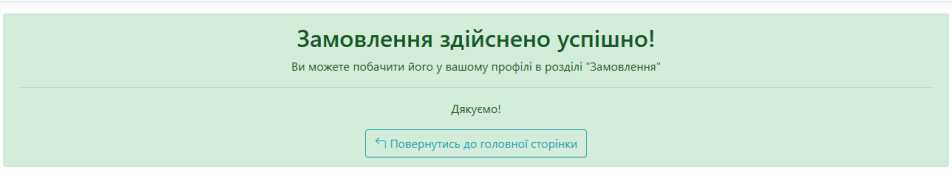


Рисунок 3.16 – Вигляд сторінки успішного здійснення замовлення

На ній він побачить інформацію про те, що його замовлення успішно сформувалось і він зможе його побачити у розділі «Замовлення» свого профілю. Також передбачено кнопку для повернення на головну сторінку.

Тепер розглянемо сторінку замовлень (рис. 3.17)

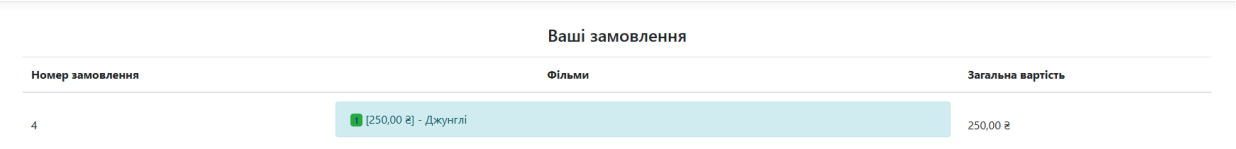


Рисунок 3.17 – Вигляд сторінки замовлень

На ній ми бачимо номер замовлення, ціну, назву фільму і загальну вартість замовлень.

Тепер розглянемо функціонал акаунта адміністратора, почнемо з головної сторінки (рис. 3.18)

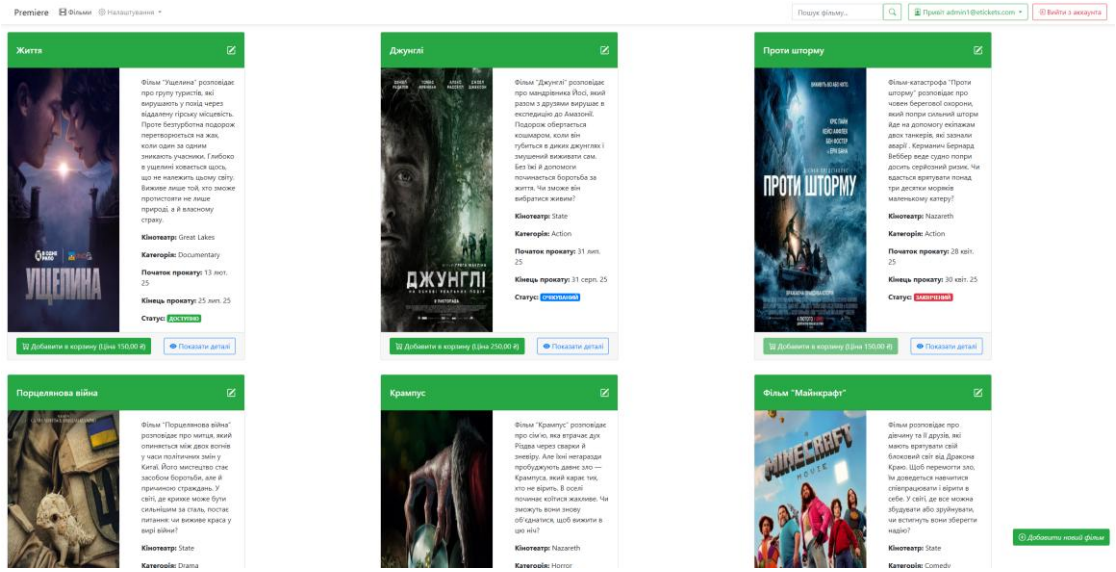


Рисунок 3.18 – Вигляд сторінки фільмів у адміністратора

Адміністратор має можливість створювати нові фільми, редагувати попередні і добавляти акторів, кінотеатри, продюсерів. При натисканні на кнопку «Налаштування» , появляється меню вибору (рис. 3.19)

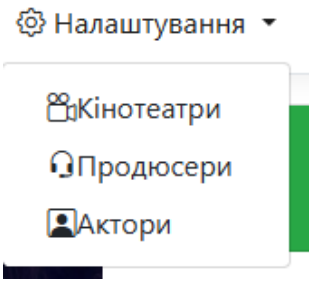


Рисунок 3.19 – Вигляд меню “Management”

Для початку виберемо сторінку з кінотеатрами (рис. 3.20). На ній ми можемо побачити список кінотеатрів які є і їх параметри: логотип, назва, опис та дії, які з ними передбачені.

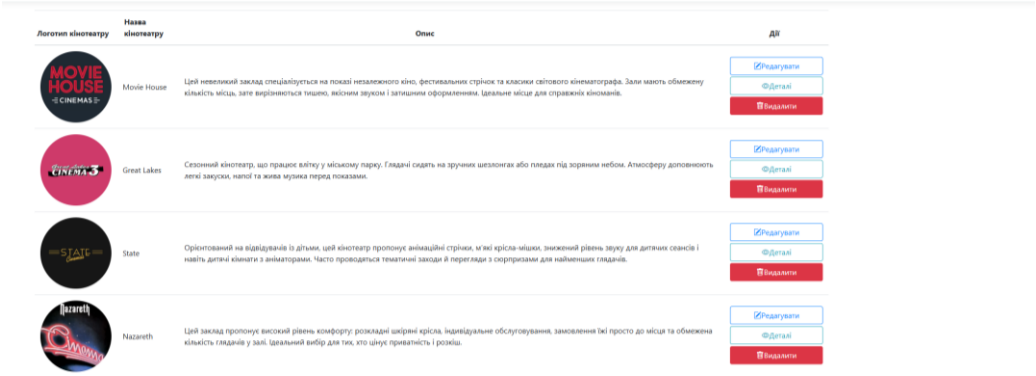


Рисунок 3.20 – Вигляд сторінки кінотеатрів

Адміністратор може редагувати, видаляти , дивитися деталі і добавляти нові(CRUD операції) кінотеатри. Кінотеатр має фото, назву, опис. На прикладі першого кінотеатру, розглянемо функціонал, який тут передбачено. Спочатку подивимось деталі фільму. Для цього реалізовано метод Details(int id):

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

У ньому при кліку передається ідентифікатор кінотеатру, деталі якого ми

хочемо переглянути. Потім в змінну cinemaDetails присвоюється дані про кінотеатр, в якого ідентифікатор збігається з тим, який передали. У разі якщо такого не існує, нас перекине на сторінку з помилкою, а при успіху – на сторінку деталей (рис. 3.21)

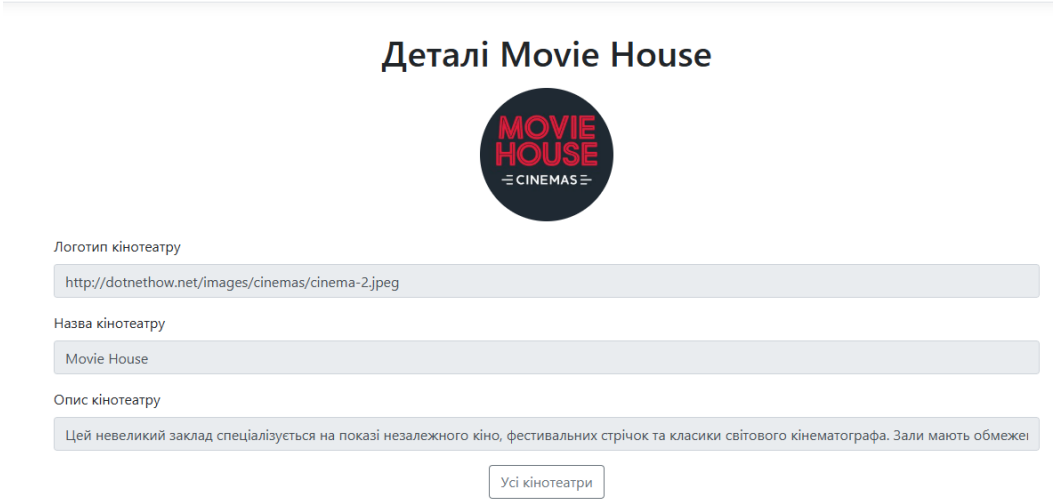


Рисунок 3.21 – Деталі кінотеатру Movie House

Як ми бачимо, тут прописано логотип кінотеатру, назва і опис. Ці 3 поля тут неможливо змінити, тільки переглядати. Також на цій сторінці є кнопка «Усі кінотеатри», яка поверне на сторінку , де список усіх кінотеатрів.

Наступним переглянемо сторінку редагування кінотеатру (рис. 3.22). Поля тут ті самі, що й на попередній сторінці, але тут є можливість змінювати значення полів.

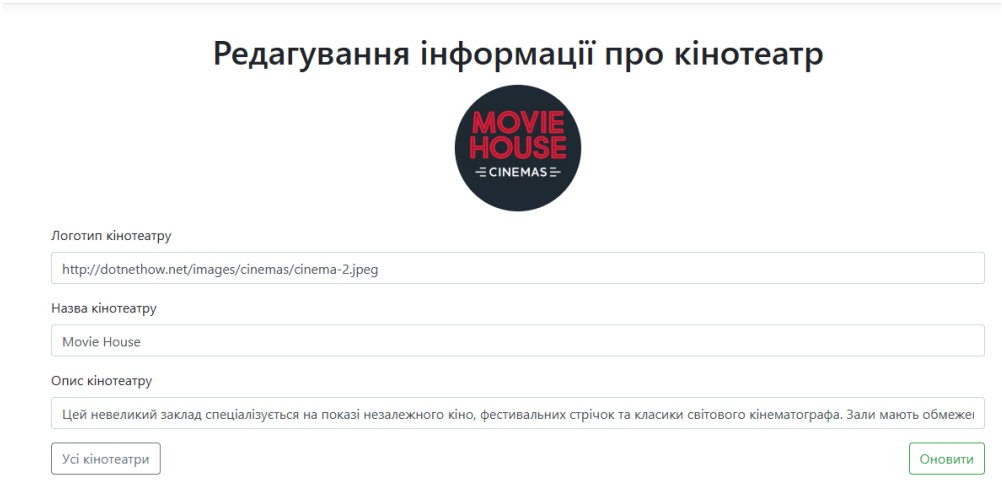


Рисунок 3.22 – Редагування інформації про кінотеатр

Код методу Edit() класу CinemasController для відкривання сторінки:

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

У ньому ми при кліку передаємо ідентифікатор кінотеатру, щоб потім в методі в змінну cinemaDetails присвоїти дані про кінотеатр з таким ідентифікатором. Потім як і на сторінці деталей, нам відкривається сторінка.

Код методу Edit() класу CinemasController для відправлення даних форми:

```
[HttpPost]
public async Task<IActionResult> Edit(int id, [Bind("Id, Logo,
Name, Description")] Cinema cinema)
{
    if (!ModelState.IsValid) return View(cinema);
    await _service.UpdateAsync(id, cinema);
    return RedirectToAction(nameof(Index));
}
```

Тут ми здійснюємо прив’язку ідентифікатора, логотипу, імені та опису і оновляємо дані в базі, здійснюється оновлення.

Тепер розглянемо сторінку видалення інформації про кінотеатр (рис. 3.23)

Рисунок 3.23 – Видалення інформації про кінотеатр

На ній у нас є надпис «Ви точно хочете видалити Movie House Test?», логотип кінотеатру, назва і його опис. Також присутні дві кнопки – «Показати усі» та «Видалити». Код реалізовано в методі DeleteConfirm() класу CinemasController:

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

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

Тут у нас в метод передається ідентифікатор кінотеатру, який видаляється. Потім з бази даних витягується інформація про кінотеатр з таким ідентифікатором, якщо він є. І нарешті здійснюється асихронне видалення кінотеатру з бази даних і повернення на сторінку, де у нас всі кінотеатри.

Тепер розглянемо сторінку зі списком продюсерів (рис. 3.24). На ній ми можемо побачити фото, повне ім'я, біографію кожного продюсера, дії з ними(редагувати, деталі, видалити) та кнопку «Добавити нового продюсера»

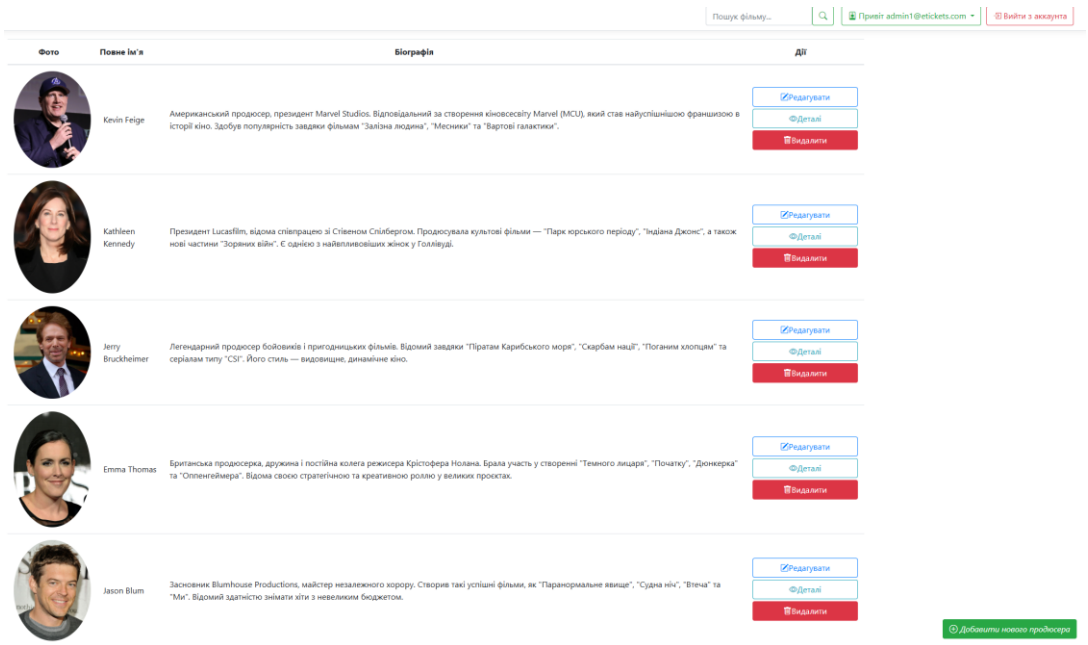


Рисунок 3.24 – Вигляд сторінки зі списком продюсерів

За допомогою методу Index() ми запускаємо цю сторінку і передаємо їй список продюсерів, що реалізовано таким способом:

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

Спочатку в змінну allProducers ми присвоюємо з _service інформацію про усіх продюсерів, а потім відкриваємо View з передачею аргументом список продюсерів і інформацію про них.

Розглянемо алгоритм додавання нового продюсера. Після натискання на кнопку «Додати нового продюсера», нас перекине на сторінку (рис. 3.25) з пустими полями для вводу посилання на фото, повного імені та біографії.

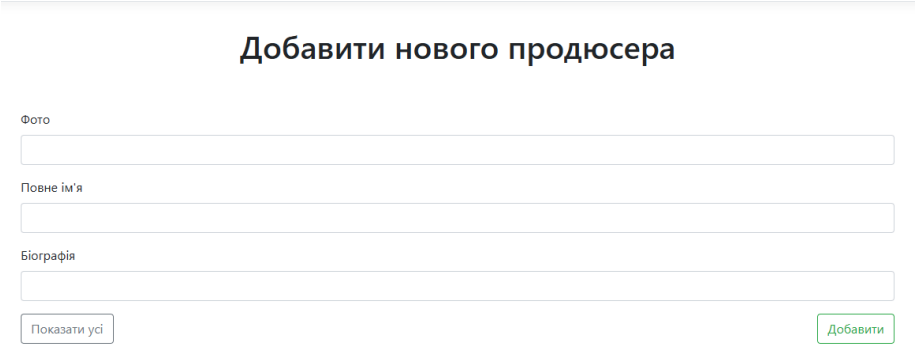


Рисунок 3.25 – Сторінка додавання нового продюсера

Після того як ми введемо дані і натиснемо «Добавити», спрацює метод додавання продюсера в базу даних і нас перекине на сторінку з усіма продюсерами. Також тут є можливість натиснути «Показати усі» , яка перекине нас назад до списку продюсерів.

Код методу Create() класу ProducersController:

```
[HttpPost]
public async Task<IActionResult>
Create([Bind("ProfilePictureURL, FullName, Bio")] Producer
producer)
{
```

```
    if (!ModelState.IsValid) return View(producer);

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

Логіка полягає в тому, щоб асихронно добавити продюсера з його даними в базу даних. Дані передаються параметром і потім додаються. Після додавання нас перекине до списку усіх продюсерів (рис. 3.26)

Рисунок 3.26 – Новий продюсер

Розглянемо код методу Edit() класу ProducersController та сторінку редагування (рис. 3.27):

```
[HttpPost]
public async Task<IActionResult>
Edit([Bind("Id, ProfilePictureURL, FullName, Bio")] Producer
producer)
{
    if (!ModelState.IsValid) return View(producer);
        await _service.UpdateAsync(producer.id, producer);
        return RedirectToAction(nameof(Index));
}
```

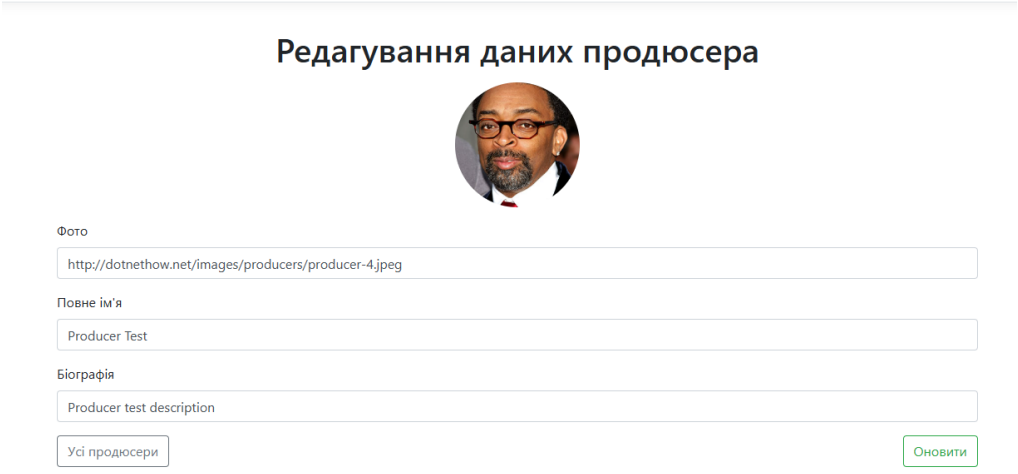


Рисунок 3.27 – Вигляд сторінки редагування продюсера

У ньому ми передаємо змінну типу Producer , в якій містимуться дані з форми та асихронно оновляємо дані в базі даних . Ця вся реалізація починається після натискання кнопки «Оновити», після чого нас перекине на сторінку з усіма продюсерами і дані буде змінено (рис. 3.28)

Рисунок 3.28 – Вигляд зміненого продюсера

Сторінка з деталями продюсера містить поля продюсера, які не можна на тут змінити і одну кнопку «Усі продюсери» (рис. 3.29)

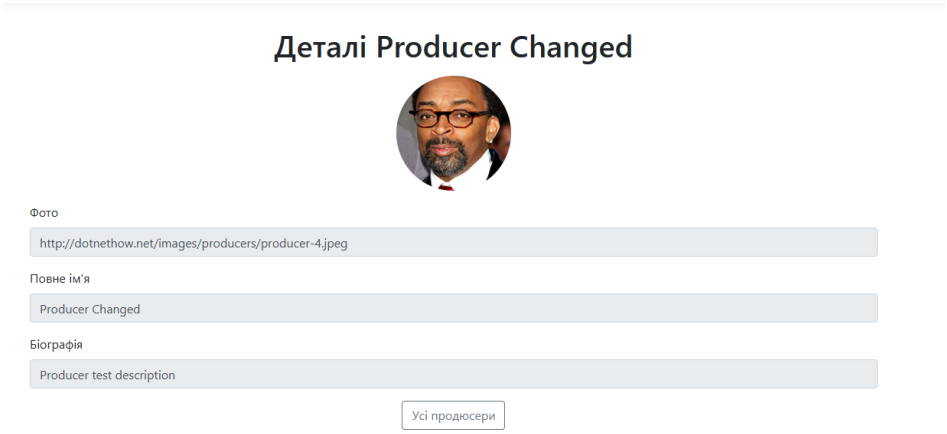


Рисунок 3.29 – Вигляд сторінки деталей продюсера

Код методу Details класу ProducersController():

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

У ньому передається ідентифікатор продюсера і на сторінці відображається інформація з бази даних про продюсера, у якого ідентифікатор збігається з тим, який передали.

Тепер розглянемо як виконано видалення продюсера (рис. 3.30). Тут є

незмінні поля, оскільки на цій сторінці адміністратор не має права щось змінювати, бо тут здійснюється тільки підтвердження видалення продюсера і є дві кнопки «Показати усі», яка поверне на сторінку з усіма продюсерами та «Видалити».

Рисунок 3.30 – Сторінка підтвердження видалення продюсера

Код методу Delete() класу ProducersController:

```
[HttpPost, ActionName("Delete")]public async Task<IActionResult>
DeleteConfirmed(int id)
{
    var producerDetails = await _service.GetByIdAsync(id);
    await _service.DeleteAsync(id);
    return RedirectToAction(nameof(Index));
}
```

Останнім в налаштуваннях є сторінка з акторами (рис. 3.31)

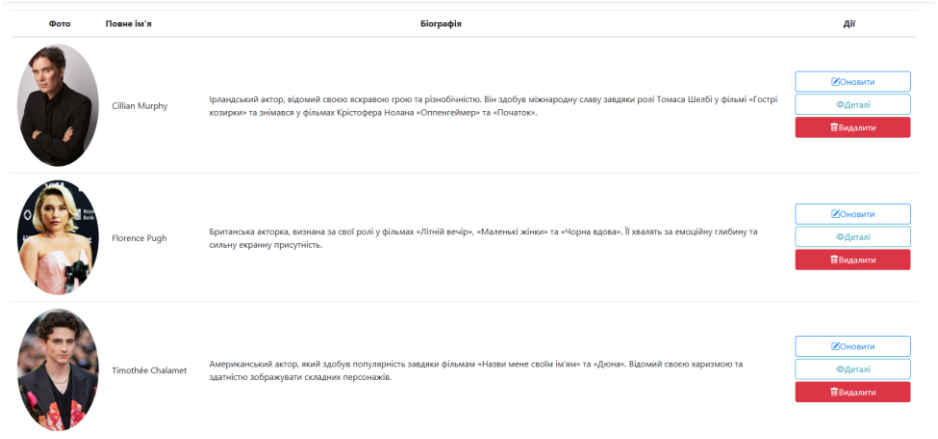


Рисунок 3.31 – Сторінка з акторами

Почнемо зі створення нового актора (рис. 3.32). Тут є 3 пустих поля для вводу – посилання на фото, повне ім’я та біографія. При натисканні на кнопку «Показати усі» нас поверне на сторінку зі списком акторів, а при натисканні на «Додати» передасться інформація про нового актора і відпрацює метод Create(Actor):

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

Як і для попередніх сутностей, тут передається зміна типу Actor і асихронно додається в базу даних з даними цієї змінної, після чого нас перекидає на сторінку з усіма акторами (рис. 3.31)

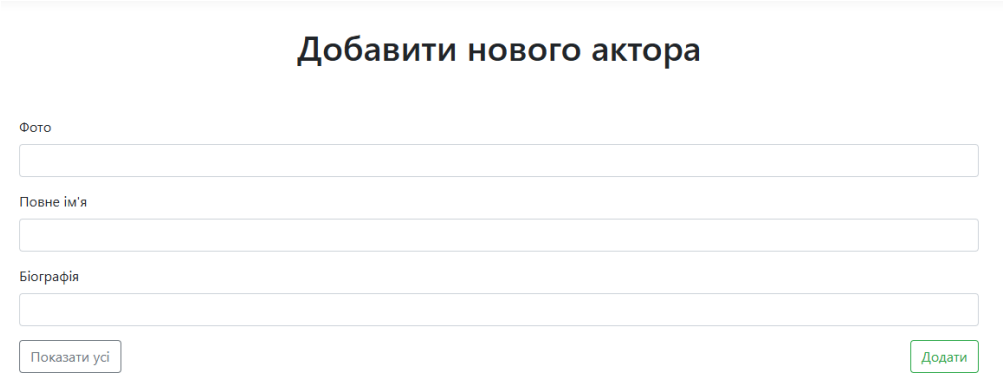


Рисунок 3.32 – Сторінка додавання нового актора

Розглянемо код методу Edit() класу ActorsController та сторінку з редагуванням (рис. 3.33):

```
[HttpPost]
public async Task<IActionResult> Edit(int id,
```

```
[Bind("Id, FullName, ProfilePictureURL, Bio")] Actor actor)
{
    if (!ModelState.IsValid)
    {
        return View(actor);
    }
    await _service.UpdateAsync(id, actor);
    return RedirectToAction(nameof(Index));
}
```

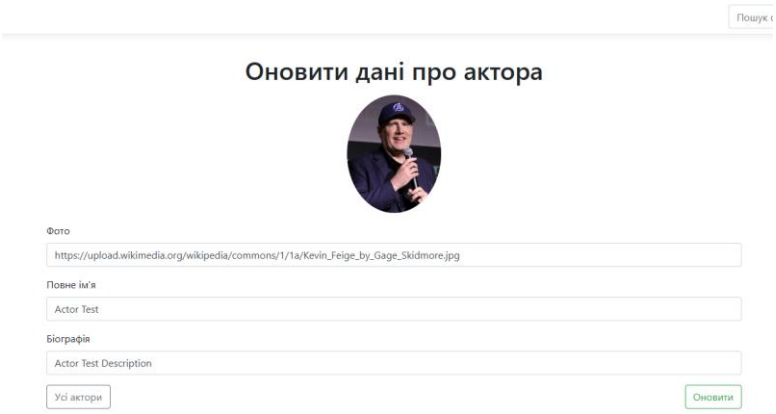


Рисунок 3.33 – Сторінка редагування даних про актора

При натисканні на кнопку «Оновити» передаються дані з форми в метод і викликається метод UpdateAsync(int, Actor) у якому продюсер з цим ідентифікатором отримає зміну даних.

Рисунок 3.34 – Сторінка деталей про актора

Сторінка з деталями (рис. 3.34) містить поля , які не можна змінювати та кнопку «Усі актори». На ній ми можемо побачити напис «Деталі про Actor Test», де замість Actor Test буде відображатись назва іншого актора, ідентифікатор якого передасться в метод, за допомогою якого відкривається ця сторінка.

При натисканні на кнопку «Видалити» нас перекине на сторінку з підтвердженням видалення (рис. 3.35). На ній у нас будуть незмінні поля, кнопка повернення до списку акторів та видалення. У незмінних полях вказана інформація про актора, якого хочуть видалити, щоб уникнути випадкового видалення не того актора.

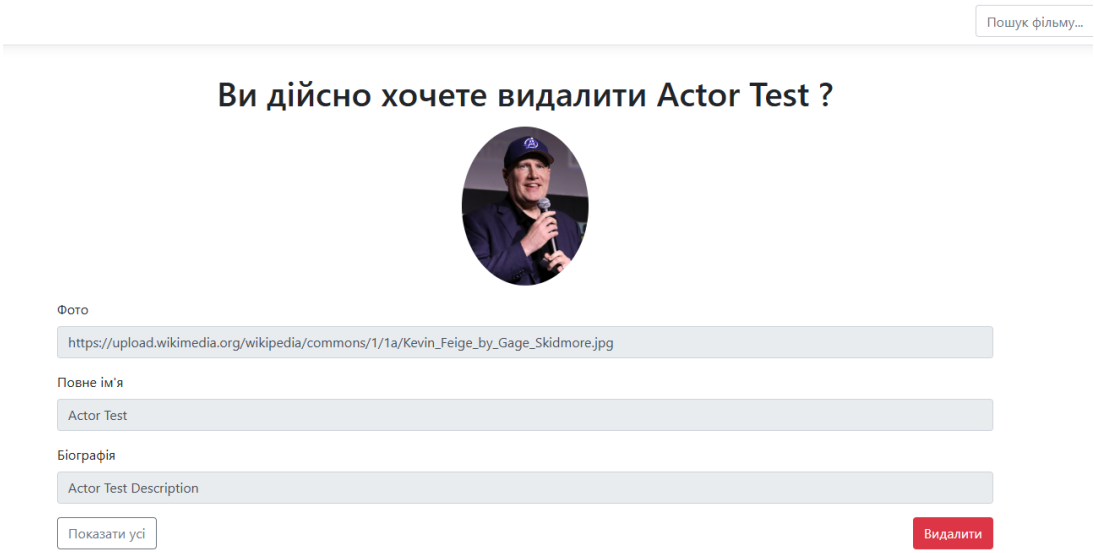


Рисунок 3.35 – Сторінка видалення актора

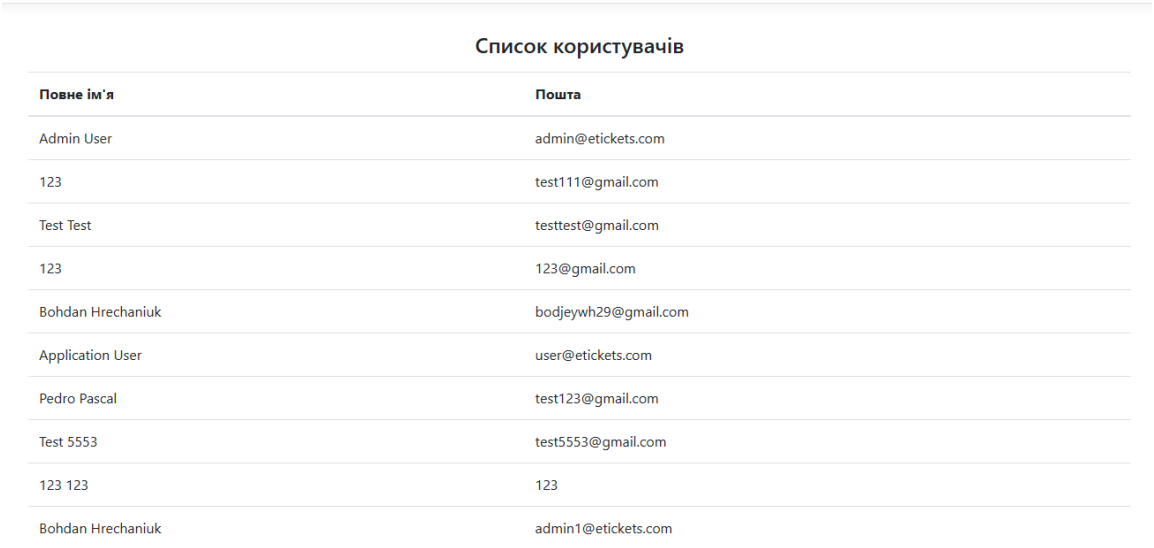
Код методу DeleteConfirmed() класу ActorsController:

```
[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));
}
```

Тут передається ідентифікатор актора і потім в методі здійсюється видалення актора з бази даних за цим ідентифікатором.

Адміністратор у меню профіля має додаткову кнопку «Users» , при натисканні на яку появиться список усіх користувачів сайту (рис. 3.36)

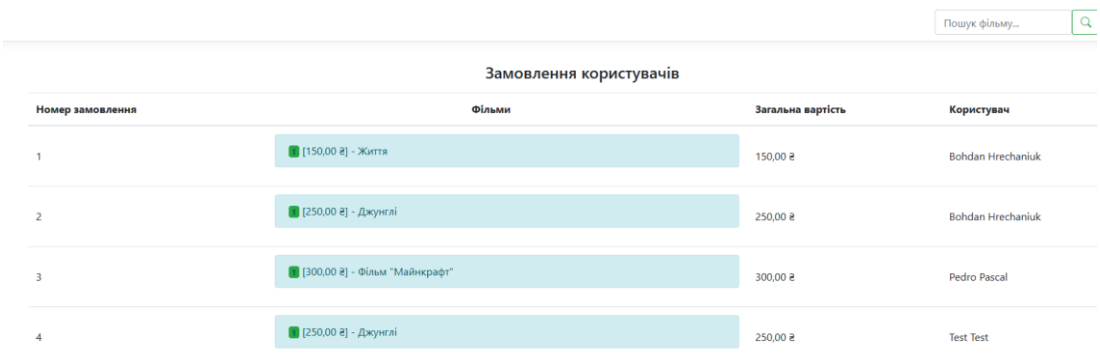


Список користувачів

Повне ім'я	Пошта
Admin User	admin@etickets.com
123	test111@gmail.com
Test Test	testtest@gmail.com
123	123@gmail.com
Bohdan Hrechaniuk	bodjeywh29@gmail.com
Application User	user@etickets.com
Pedro Pascal	test123@gmail.com
Test 5553	test5553@gmail.com
123 123	123
Bohdan Hrechaniuk	admin1@etickets.com

Рисунок 3.36 – Вигляд сторінки користувачів

При натисканні на кнопку «Замовлення», у адміністратора появиться сторінка з замовленнями усіх користувачів (рис. 3.37), в той час як в звичайного користувача тут будуть тільки його замовлення.



Пошук фільму...

Замовлення користувачів

Номер замовлення	Фільми	Загальна вартість	Користувач
1	[150,00 ₴] - Життя	150,00 ₴	Bohdan Hrechaniuk
2	[250,00 ₴] - Джунглі	250,00 ₴	Bohdan Hrechaniuk
3	[300,00 ₴] - Фільм "Майнкрафт"	300,00 ₴	Pedro Pascal
4	[250,00 ₴] - Джунглі	250,00 ₴	Test Test

Рисунок 3.37 – Вигляд сторінки замовлень користувачів

ВИСНОВКИ

У ході виконання бакалаврської роботи здійснено розробку веб-сайту організації інформаційного супроводу, демонстрації та організації продажів квитків на фільми засобами C#, бази даних реляційного типу SSMS та фреймворку ASP. NET.

У першому розділі описано інформаційне забезпечення веб-сайту продажу квитків, а саме проведено опис інформаційного процесу, здійснено порівняльний огляд веб-сайтів аналогів внаслідок чого вибрано найкращі функції кожного з них та здійснена постановка завдання на проєкт: реалізувати функціональну панель адміністратора, механізм легкої можливості оформити покупку квитків та розробити зручний та інтуїтивно зрозумілий графічний інтерфейс , що забезпечить ефективність використання функціоналу веб-сайту.

У другому розділі розроблено структурні рішення для даного інтернет-сервісу, а саме розроблено структуру та функціонал веб-сайту продажу квитків, таблиці та структуру бази даних, розроблено та описано діаграми послідовності взаємодії компонентів сервісу та розроблено USE-CASE діаграми взаємодії з користувачем і адміністратором.

У третьому розділі здійснено проектування графічного інтерфесу та реалізація баз даних веб-сайту продажу квитків , інтерфейс та функціонал веб-сайту продажу квитків. У результаті отримано функціональний веб-сайт продажу квитків, де користувач має змогу переглянути список фільмів, зареєструвати обліковий запис, додавати квитки в корзину та оформлювати замовлення. Адміністратор має можливість змінювати інформацію про фільми, кінотеатри, продюсерів та акторів , додавати нових та видаляти інформацію про них. Також здійснена перевірка працездатності додатку.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Adam Freeman Pro ASP.NET Core MVC: підручник, ред. Apress, 2016. 456-845 с.

2. Джон Шарп Visual Studio 2022 Profesional: підручник, ред. Microsoft , 2022. 7-142 с.

3. “SQL Server 2019 для професіоналів” / Воррен Г. Метц, Сьоюзен Дж. Робертс, Брайан Ларсен; Видавництво “Діалектика”, 2021.

4. J.Liberty Programming C# 8.0: підручник, O’Reilly Media, 2020. 704 с.

5. D.Esposito Modern Web Development with ASP.NET Core 3: підручник, Microsoft Press, 2020. 432 с.

6. R.Strahl West Wind WebSurge and Web Performance with ASP.NET Core: підручник, West Wind Technologies, 2021. 284 с.

7. C.Nagel Professional C# and .NET – 2021 Edition: підручник, Wrox, 2021. 1504 с.

8. B.Watson Writing High-Performance .NET Code: підручник, Kindle Edition, 2020. 425 с.

9. K.Delaney SQL Server Internals: підручник, Microsoft Press, 2022. 960 с.

10. J.Price, Freeman A. Microsoft ASP.NET CORE in Action, 2021. 624с.

ДОДАТКИ

ДОДАТОК А

Вміст файлу «Program. cs»

```
using eTickets.Data;

using eTickets.Data.Cart;

using eTickets.Data.Services;

using eTickets.Models;

using Microsoft.AspNetCore.Authentication.Cookies;

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using NLog.Web;


var logger = NLogBuilder

            .ConfigureNLog("nlog.config")

            .GetCurrentClassLogger();

try

{

    logger.Debug("Initialization of program");

    var builder = WebApplication.CreateBuilder(args);

    builder.Host.UseNLog();

    // Add services to the container.

    builder.Services.AddDbContext<AppDbContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString("De
faultConnection")));

    builder.Services.AddControllersWithViews();

    builder.Services.AddScoped<IActorsService, ActorsService>();

    builder.Services.AddScoped<IMoviesService, MoviesService>();

    builder.Services.AddScoped<IProducersService,
ProducersService>();

    builder.Services.AddScoped<ICinemasService,
```

```
CinemasService>();, ,

    builder.Services.AddScoped<IOrdersService, OrdersService>();

    builder.Services.AddSingleton<IHttpContextAccessor,
HttpContextAccessor>();

    builder.Services.AddScoped(sc =>
ShoppingCart.GetShoppingCart(sc));

    //Authentication and authorization

    builder.Services.AddIdentity<ApplicationUser,
IdentityRole>().AddEntityFrameworkStores<AppDbContext>();

    builder.Services.AddMemoryCache();

    builder.Services.AddSession();

    builder.Services.AddAuthentication(options =>

    {

        options.DefaultScheme =
CookieAuthenticationDefaults.AuthenticationScheme;

    });

    var app = builder.Build();

    // Configure the HTTP request pipeline.

    if (!app.Environment.IsDevelopment())

    {

        app.UseExceptionHandler("/Home/Error");

        // The default HSTS value is 30 days.You may want to
change this for production scenarios, see https:
//aka.ms/aspnetcore-hsts.

        app.UseHsts();

    }

    app.UseHttpsRedirection();

    app.UseStaticFiles();

    app.UseRouting();
```

```
    app.UseSession();
    app.UseAuthentication();
    app.UseAuthorization();
    app.UseAuthorization();
    app.MapControllerRoute(
        name:  "default",
        pattern:  "{controller=Movies}/{action=Index}/{id?}");
    AppDbInitializer.Seed(app);
    await AppDbInitializer.SeedUsersAndRolesAsync(app);
    app.Run();
}
catch (Exception exception)
{
    // NLog:  catch setup errors
    logger.Error(exception, "Stopped program because of
exception");
    throw;
}
finally
{
    // Ensure to flush and stop internal timers/threads before
application-exit (Avoid segmentation fault on Linux)
    NLog.LogManager.Shutdown();
}
```

Вміст файлу «MoviesController. cs»

```
using eTickets.Data;
using eTickets.Data.Services;
using eTickets.Data.Static;
```

```
using eTickets.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
namespace eTickets.Controllers
{
    [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.GetAllAsync(n=>n.Cinema);
            return View(allMovies);
        }
        [AllowAnonymous]
        public async Task<IActionResult> Details(int id)
        {
            var movieDetail = await
_service.GetMovieByIdAsync(id);
            return View(movieDetail);
```

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

            await _service.DeleteAsync(id);
            return RedirectToAction(nameof(Index));
        }
        //GET:  Movies/Create
        public async Task<IActionResult> Create()
        {
            var movieDropdownsData = await
_service.GetNewMovieDropdownsValues();

            ViewBag.Cinemas = new
SelectList(movieDropdownsData.Cinemas, "Id", "Name");
            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.Cinemas = new
SelectList(movieDropdownsData.Cinemas, "Id", "Name");
                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()
            {
                Id = movieDetails.Id,
                Name = movieDetails.Name,
```

```
                Description = movieDetails.Description,
                Price = movieDetails.Price,
                StartDate = movieDetails.StartDate,
                EndDate = movieDetails.EndDate,
                ImageURL = movieDetails.ImageURL,
                MovieCategory = movieDetails.MovieCategory,
                CinemaId = movieDetails.CinemaId,
                ProducerId = movieDetails.ProducerId,
                ActorIds = movieDetails.Actors_Movies.Select(n =>
n.ActorId).ToList(),
            };
            var movieDropdownsData = await
_service.GetNewMovieDropdownsValues();
            ViewBag.Cinemas = new
SelectList(movieDropdownsData.Cinemas, "Id", "Name");
            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.Cinemas = new
SelectList(movieDropdownsData.Cinemas, "Id", "Name");

                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));

        }

        [AllowAnonymous]

        public async Task<IActionResult> Filter(string
searchString)

        {

            var allMovies = await _service.GetAllAsync(n =>
n.Cinema);

            if (!string.IsNullOrEmpty(searchString))

            {

               var filteredResult = allMovies.Where(n =>
n.Name.ToLower().StartsWith(searchString.ToLower()));

                return View("Index", filteredResult);

            }

            return View("Index", allMovies);

        }

    }

}
```

Вміст файлу «ActorsController. cs»

```
using eTickets.Data;
```

```
using eTickets.Data.Services;
using eTickets.Data.Static;
using eTickets.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
namespace eTickets.Controllers
{
    [Authorize(Roles = UserRoles.Admin)]
    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,
ProfilePictureURL, Bio")] 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
        [HttpGet]
        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, Bio")] Actor actor)
        {
            if (!ModelState.IsValid)
            {
                return View(actor);
            }
            await _service.UpdateAsync(id, actor);
            return RedirectToAction(nameof(Index));
        }
        //Get:  Actors/Delete
        public async Task<IActionResult> Delete(int id)
        {
            var actorDetails = await _service.GetByIdAsync(id);
            if (actorDetails == null)
            {
                return View("NotFound");
            }
```

Кінець додатку А

```
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));
        }

    }
}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: ***Розробка інформаційного web-сайту для кінотеатру засобами C# та фреймворків ASP.NET і SSMS***

Обсяг пояснювальної записки 59 аркушів:

11 таблиць;

48 рисунків;

1 додаток.

Дата завершення роботи: *12 червня 2025р*.

Підпис студента- _____*Гречанюк Б.М.*