

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ВЕБЗАСТОСУНОК ДЛЯ ПРОДАЖУ КВИТКІВ У КІНО
WEB APPLICATION TO SELL MOVIE TICKETS

Виконав: студент 4 курсу, групи КНТ-110сп
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ДЕМЧЕНКО І.Д.

(ПРИЗВИЩЕ та ініціали)

Керівник КОЛПАКОВА Т.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент КОРОТКИЙ О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
« » 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ДЕМЧЕНКА Івана Дмитровича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Вебзастосунок для продажу квитків у кіно. Web Application to Sell Movie Tickets

керівник проєкту (роботи) к.т.н., доцент, КОЛПАКОВА Тетяна Олексіївна

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «04» квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 1 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Характеристика об'єкта проєктування. 2. Вибір і обґрунтування структури системи. 3. Основні рішення щодо реалізації компонентів. 4. Керівництво системного адміністратора. 5. Керівництво користувача.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	КОЛПАКОВА Т.О., доцент		
Нормоконтроль	ЛИПОВЕЦЬ М.В, асистент		

7. Дата видачі завдання « 04 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка структури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділи 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділи 4, 5
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Іван ДЕМЧЕНКО
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна КОЛПАКОВА
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 120 с., 4 табл., 39 рис., 3 дод., 15 джерел.

ВЕБЗАСТОСУНОК, ОНЛАЙН ПРОДАЖ, АВТОМАТИЧНЕ ОБСЛУГОВУВАННЯ, КІНОТЕАТРИ, E-COMMERCE.

Об'єкт дослідження – процес купівлі квитків онлайн.

Предмет дослідження – вебзастосунок для продажу квитків у кіно.

Мета роботи – створення вебзастосунку для обслуговування клієнтів кінотеатру, автоматизації продажів квитків, автоматичної роботи з базою фільмів та місць в кінотеатрі.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, технологія ASP.NET, мови програмування C#, JavaScript, персональний комп'ютер з процесором Intel Core 5 Duo під управлінням операційної системи Microsoft Windows 10.

Галузь використання – проєкт може бути використаний кінотеатрами, які хочуть впровадити автоматичне обслуговування та продаж квитків.

Результат – розроблено вебзастосунок для автоматичного обслуговування відвідувачів кінотеатру.

Висновки: в процесі розробки вебзастосунку для купівлі квитків онлайн на мові програмування C# було досліджено сучасні технології, що дозволили створити зручний для користувачів застосунок. Розроблений застосунок надає можливість купувати квитки без відвідування кас, що суттєво заощаджує час відвідувачів кінотеатрів.

ABSTRACT

Explanatory note to the final qualifying work of the bachelor: 120 pages, 4 tables, 39 pictures, 3 appendixes, 15 sources.

WEB APPLICATION, ONLINE SALES, AUTOMATED SERVICE, CINEMA THEATERS, E-COMMERCE.

The research object is a web application for e-commerce.

The subject of the study is a web application for selling movie tickets.

The purpose of the work is to create a web application for servicing cinema customers, automating ticket sales, and automatically working with the database.

Materials, methods and technical means: structural and object-oriented programming, ASP.NET technology, programming languages C#, JavaScript, a personal computer with an Intel Core 5 Duo processor running the Microsoft Windows 10 operating system.

Field of use - the project can be used by cinemas that want to implement automatic service and ticket sales.

The result is a web application developed for automatic service of cinema visitors.

Conclusions: in the process of developing a web application for buying tickets online in the C# programming language, modern technologies were investigated, which allowed to create a user-friendly application. The developed application provides an opportunity to buy tickets without visiting the box office, which significantly saves the time of cinema visitors.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	8
Вступ	9
1 Характеристика об'єкта проектування.....	10
1.1 Поняття інтернет магазину	10
1.2 Основні способи ведення бізнесу за допомогою онлайн-покупок	11
1.3 Основні функціональні можливості інтернет-магазину.....	15
1.4 Висновки за розділом.....	16
2 Вибір і обґрунтування структури системи.....	17
2.1 Вибір середовища програмування для серверної частини сайту.....	17
2.2 Вибір середовища програмування для клієнтської частини сайту.....	25
2.3 Обґрунтування вибору середовища функціонування системи	27
2.4 Бази даних.....	30
2.5 Висновки за розділом	32
3 Основні рішення щодо реалізації компонентів системи.....	34
3.1 Основні рішення щодо реалізації компонентів системи	34
3.2 Алгоритм замовлення у онлайн сервісі продажу квитків	37
3.3 Алгоритм роботи головної сторінки сервісу з продажу квитків	39
3.4 Алгоритм роботи адміністративної частини.....	41
3.5 Реалізація бази даних	42
3.6 Характеристики розробленого продукту	50
3.7 Структурна схема компонентів магазину	50
3.8 Висновки за розділом	51
4 Керівництво системного адміністратора	52
4.1 Розгортання інтернет-магазину	52
4.2 Розгортання бази даних вебсайту	52
4.3 Адміністративна панель інтернет-магазину	52
4.4 Висновки за розділом.....	56
5 Керівництво користувача.....	57

5.1 Призначення сайту	57
5.2 Вхід до вебсайту та процес авторизації й замовлення товару	57
5.3 Висновки за розділом	59
Висновки.....	60
Перелік джерел посилань.....	61
Додаток А Технічне завдання.....	63
Додаток Б Текст програми	65
Додаток В Слайди презентації	117

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

ASP – Active Server Pages;

B2B – Business To Business;

B2C – Business To Consumer;

C2B – Consumer To Business;

C2C – Consumer To Consumer;

CRUD – Create Read Update Delete;

DB – Database;

IDE – Integrated Development Environment;

REST – Representational State Transfer;

ОС – операційна система.

ВСТУП

Нині онлайн-сервіси стають дедалі популярнішими, і кінотеатри не є винятком. Інтернет-сайти для продажу квитків у кіно надають споживачам можливість купівлі квитків на фільми із зручністю та комфортом, не виходячи з дому. Цей підхід має ряд переваг, таких як економія часу та зручність, оскільки покупці можуть вибрати фільм та забронювати місця на нього онлайн, а також заощадити гроші, оскільки багато кінотеатрів надають знижки на онлайн-бронювання.

Крім того, такий сайт може бути корисним як для кінотеатрів, так і для кіноманів, які можуть вибрати та забронювати місця на улюблений фільм онлайн. Кінотеатри можуть використовувати сайти для продажу квитків, щоб збільшити прибуток, задовольнити потреби споживачів та покращити якість обслуговування. У цьому процесі сайти для продажу квитків стають невід'ємною частиною бізнесу кінотеатру та дозволяють керувати продажами квитків у реальному часі, обробляти замовлення та надавати інформацію про фільми та сеанси.

Метою моєї дипломної роботи є створення ефективного та зручного сайту для продажу квитків у кіно, який задовольнить потреби та бажання користувачів та допоможе збільшити прибуток кінотеатру. В рамках роботи буде проведено аналіз існуючих сайтів для продажу квитків у кіно, розроблено концепцію нового сайту, створено його структуру, розроблено та реалізовано дизайн-концепцію. Крім того, буде проведено тестування сайту для продажу квитків у кіно з метою визначення його ефективності та зручності використання, а також розроблено план заходів щодо просування сайту та залучення нових клієнтів.

1 ХАРАКТЕРИСТИКА ОБ'ЄКТА ПРОЄКТУВАННЯ

1.1 Поняття онлайн сервісу для продажів

Онлайн сервіс для продажів – це віртуальний магазин, де послуги або товари продаються через інтернет. Сьогодні інтернет-сервіси стали дуже популярними, і багато людей вважають за краще купувати в інтернеті, ніж у традиційний спосіб. Це пов'язано з тим, що онлайн-сервіси надають безліч переваг, таких як зручність, швидкість та доступність.

Через онлайн можна купити практично все, що завгодно: від побутової техніки та електроніки до продуктів харчування та одягу. Покупці можуть легко і швидко знайти потрібний товар або послугу, ознайомитися з його характеристиками та відгуками інших покупців, порівняти ціни та зробити вибір. Оформлення замовлення в онлайн-сервісах також відбувається швидко і легко, і зазвичай вимагає всього кількох кліків миші.

Для створення онлайн-сервісу для продажів потрібно мати добре розуміння того, як працює інтернет і які технології використовуються для створення вебсайтів. Також важливо розуміти, як влаштовані процеси продажу та логістики, щоб забезпечити швидке та якісне обслуговування покупців.

Для успішного створення онлайн-сервісу необхідно проаналізувати конкурентів та визначити свої переваги перед ними. Однією з ключових переваг може бути висока якість товарів, конкурентні ціни, швидке та зручне оформлення замовлення, високий рівень обслуговування клієнтів тощо.

Одним з найпоширеніших видів онлайн сервісів є онлайн-магазини, які продають товари через інтернет, але не мають фізичних магазинів. Це дозволяє значно скоротити витрати на оренду приміщень, оплату персоналу і т.д. Натомість вони інвестують у розробку та просування своїх сайтів, щоб залучити більше покупців.

Важливим аспектом онлайн-сервісу є також забезпечення безпеки покупців. Для цього використовуються різні механізми, такі як захищені

протоколи передачі даних (HTTPS), системи шифрування та аутентифікації, а також механізми захисту від шахрайства.

Незважаючи на деякі проблеми та виклики, інтернет магазини продовжують зростати та розвиватися. Завдяки своїй зручності, доступності та швидкості, вони стають все більш популярними серед покупців, і майбутнє інтернет магазинів, швидше за все, буде лише більш успішним та процвітаючим.

1.2 Основні способи ведення бізнесу за допомогою системи онлайн-покупок

В даний час, у зв'язку з широким поширенням інтернету, все більше людей стають користувачами онлайн-магазинів та сервісів для придбання товарів та послуг. Існує кілька способів інтернет-покупок, таких як B2B, B2C, C2C та C2B, кожен з яких має свої особливості та переваги [1].

B2B (business-to-business) – це спосіб інтернет-купівлі, коли одне підприємство купує товари чи послуги в іншого підприємства. Цей спосіб інтернет-покупки зазвичай використовується для закупівлі товарів або послуг у великих обсягах для використання у бізнесі. B2B-покупки можуть бути зроблені через спеціалізовані сайти, де постачальники товарів чи послуг пропонують свої продукти для покупки.

B2C (business-to-consumer) – це спосіб інтернет-купівлі, коли компанія продає товари чи послуги безпосередньо кінцевому споживачеві. Цей спосіб інтернет-покупки найпоширеніший і доступніший для багатьох людей. B2C-купівлі можуть бути зроблені через сайти онлайн-магазинів, які пропонують широкий асортимент товарів та послуг.

C2C (consumer-to-consumer) – це спосіб інтернет-купівлі, коли один споживач продає товари чи послуги іншому споживачеві. Цей спосіб інтернет-купівлі використовується для продажу товарів та послуг між окремими людьми, наприклад, на вебсайтах онлайн-аукціонів. C2C-покупки

зазвичай надають більше гнучкості в ціноутворенні і можуть бути вигідними для покупців, але можуть мати ризики.

C2B (consumer-to-business) – це спосіб інтернет-купівлі, коли споживач продає товари чи послуги компанії. Цей спосіб інтернет-купівлі використовується для продажу товарів чи послуг, що виробляються чи надаються окремими людьми, компаніям. C2B-купівлі можуть бути вигідними для компаній, оскільки вони можуть отримати товари або послуги за нижчою ціною, ніж у постачальника.

При порівнянні способів інтернет-покупок необхідно враховувати не тільки їх особливості, але й переваги та недоліки кожного з них.

B2B-купівлі мають свої переваги у вигляді можливості купувати товари або послуги за нижчими цінами, особливо якщо це робиться на регулярній основі та у великих обсягах. Крім того, компанії, що займаються B2B-продажами, можуть надавати додаткові послуги, такі як консультації щодо вибору товарів, навчання персоналу та технічна підтримка.

B2C-покупки також мають свої переваги, такі як зручність та доступність. Клієнти можуть робити покупки у будь-який час та місце, а також вибирати з великої кількості товарів та послуг. Онлайн-магазини можуть надавати знижки та бонуси для постійних клієнтів, що є додатковим стимулом для покупок.

C2C-купівлі дозволяють споживачам продавати та купувати товари та послуги один в одного, що може бути вигідним для обох сторін. Крім того, C2C-купівлі можуть бути набагато більш гнучкими в ціноутворенні, що може призвести до більш вигідних угод.

C2B-покупки можуть бути вигідними для компаній, оскільки вони можуть отримувати товари та послуги за нижчими цінами, ніж при покупці у постачальника. Крім того, C2B-купівлі можуть також сприяти зміцненню бренду компанії, якщо товари або послуги будуть використовуватись у рамках бізнес-діяльності.

Однак кожен спосіб інтернет-покупки має свої недоліки. Наприклад, B2B-купівлі можуть бути обмежені вибором постачальників, а C2C-купівлі можуть мати ризики у зв'язку з несумлінними продавцями та неперевіреними товарами. Крім того, в деяких випадках повернення товарів і послуг може бути більш складним при інтернет-покупках необхідно враховувати свої потреби та переваги. Якщо ви віддаєте перевагу великому вибору товарів та послуг, то B2C-купівлі можуть бути найбільш підходящим варіантом. Якщо ж ви займаєтеся бізнесом і вам необхідно купувати товари на регулярній основі, то B2B-покупки можуть бути вигіднішими [1].

Також варто враховувати довіру до продавців та якість товарів та послуг. У випадку з C2C-покупками, необхідно бути обережнішими і перевіряти продавців перед угодами. У будь-якому випадку, перед здійсненням покупки, варто вивчити відгуки про продавців та товари, а також перевірити умови доставки та повернення.

Нарешті, при виборі способу інтернет-покупки, варто враховувати свої можливості та обмеження. Деякі способи можуть бути доступні лише для певних категорій покупців, а також можуть мати додаткові вимоги та обмеження.

В цілому, кожен спосіб інтернет-купівлі має свої переваги та недоліки, і вибір повинен ґрунтуватися на індивідуальних потребах та уподобаннях покупця. Однак важливо пам'ятати про необхідність дбайливого ставлення до особистих даних та захисту від шахрайства, щоб уникнути можливих проблем та негативного досвіду інтернет-покупок.

Нижче розглянемо приклади B2C сайтів, які продають квитки в кино і реалізують схожий функціонал. В них приблизно однакові переваги які відрізняються особливими можливостями при онлайн замовленні, але серед недоліків, які би я визначив онлайн сервісів те, що онлайн сервіс все одно доведеться постійно обслуговувати, та при несправності сервера, на якому був запущений вебзастосунок, користувачі не зможуть ним користуватися.

Нижче я приведу аналоги українських проєктів, які зробили можливість бронювати квитки онлайн.

1.2.1 "Мультиплекс"

“Мультиплекс” [2] – це найбільша мережа багатозальних кінотеатрів України заснована у 2004 році. Нижче я приведу рисунок 1.1, на котрому зображений сервіс онлайн бронювання квитків сервісу “Мультиплекс”.

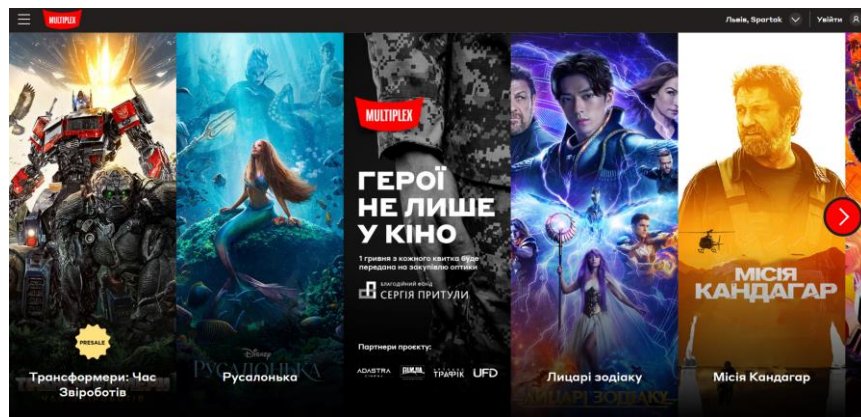


Рисунок 1.1 – Вигляд онлайн сервісу “Мультиплекс” [2]

Основними перевагами сервісу "Мультиплекс" є те, що для перегляду актуальних фільмів та бронювання квитків не потрібно замовляти квиток на касі, сервіс робить можливість бронювати квитки онлайн. Що заощаджує час, робить зручнішим та привабливим користування саме цим сервісом.

1.2.2 “Планета Кіно”

“Планета Кіно” [3] – це українська мережа кінотеатрів з передовими технологіями у кінозалах. Заснована у 2008 році. Нижче я приведу рисунок 1.2, на якому зображено онлайн сервіс “Планета Кіно”.

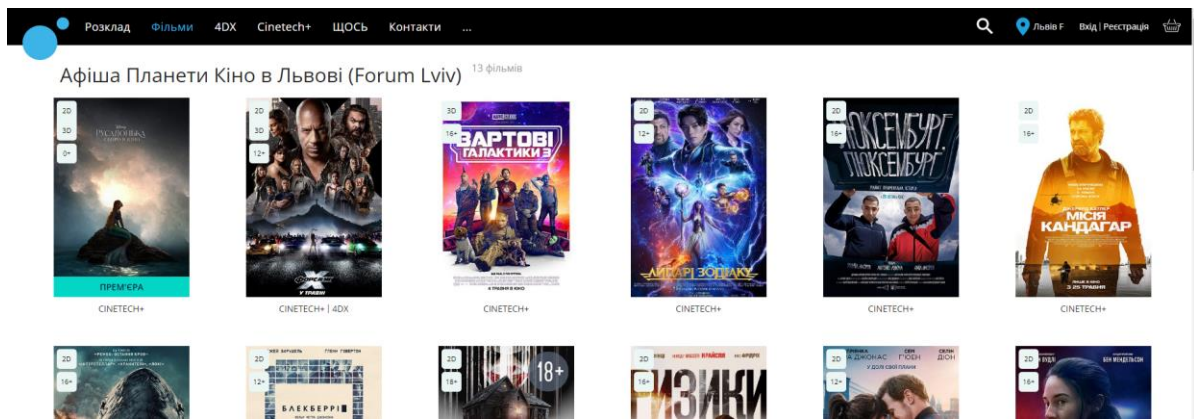


Рисунок 1.2 – Онлайн сервіс “ПланетаКіно” [3]

Доречі, ця мережа кінотеатрів першою почала продавати квитки у кіно у режимі онлайн, без необхідності роздруковувати квиток та показувати його у паперовому вигляді.

Серед переваг цього онлайн сервісу я би виділив особливі клієнто орієнтовані можливості, а саме придбання подарункових сертифікатів, автоматизоване отримання знижок та бонусів на особистий кабінет на сайті, можливість переглядати опис майбутніх фільмів та дату їх появи.

1.3 Основні функціональні можливості інтернет-магазину

Ключовим моментом під час створення інтернет-магазину є його функціональність. Необхідно врахувати вимоги та потреби клієнтів, щоб створити магазин, який задовольнить їхні очікування та залучить нових покупців. У зв'язку з цим, список основного функціоналу інтернет-магазину може бути корисним інструментом для розробки та покращення магазину.

Основний функціонал інтернет-магазину:

- каталог товарів з описом та цінами;
- зручний пошук товарів за різними параметрами;
- фільтри для точного підбору товарів;
- кошик для збирання вибраних товарів;
- оформлення замовлення з мінімальною кількістю кліків;

- різні варіанти оплати та доставки замовлень;
- система повідомлень про статус замовлення;
- система відгуків та рейтингу товарів;
- система лояльності та бонусів для клієнтів;
- мобільна версія сайту для зручності використання на мобільних пристроях;
- технології захисту даних клієнтів;
- система управління замовленнями та обліку товарів на складі;
- можливість швидкої реєстрації та авторизації на сайті;
- різні варіанти знижок та акцій для залучення клієнтів.

Основний функціонал інтернет-магазину включає безліч можливостей, які допомагають магазину залучати клієнтів, підвищувати якість обслуговування і збільшувати прибуток. Важливо враховувати потреби клієнтів та адаптувати функціональність магазину під їх очікування. Регулярне оновлення та покращення функціоналу допоможуть інтернет-магазину залишатися конкурентоспроможним та успішним у довгостроковій перспективі [4].

1.4 Висновки за розділом

Було проведено огляд об'єкта проєктування, під час якої було складено набір термінів, що характеризують процеси електронної комерції та основні принципи функціонування інтернет-магазинів. Було викладено етапи ведення бізнесу під час використання систем онлайн-покупок. Крім того, були проаналізовані основні функціональні можливості кінотеатрів, які віддають перевагу саме онлайн сервісам для продажу квитків. На основі цих даних було ухвалено рішення про створення вебзастосунку для продажу квитків у кіно з урахуванням актуальності, зручності та ефективності онлайн-бізнесу.

2 ВИБІР І ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ

2.1 Вибір середовища програмування для серверної частини сайту

Розробка буде проводитись у IDE Visual Studio 2022.

Для розробки цього магазину з продажу квитків у кіно я вирішив використовувати мову C#, технологію ASP.NET від Microsoft. ASP розшифровується як Active Server Pages.

Visual Studio і мова програмування C# надають кілька переваг розробки сервера.

По-перше, Visual Studio має потужні інструменти розробки, які значно спрощують процес створення серверних додатків. Інтегроване середовище розробки (IDE) надає функціональності, такі як автодоповнення коду, налагоджувач та інструменти для управління проєктом, прискорюючи процес розробки та підвищуючи ефективність. [5]

По-друге, C# є потужною та елегантною мовою програмування, спеціально розробленою для створення масштабованих та надійних серверних додатків. Він володіє багатими можливостями об'єктно-орієнтованого програмування, підтримкою багатопоточності та широким набором бібліотек, що робить його ідеальним вибором для розробки сервера.

Третя перевага полягає у великій підтримці спільноти та документації. Visual Studio і C# мають активні спільноти розробників, які готові поділитися знаннями та досвідом. Це дозволяє розробникам швидко отримати допомогу, знайти відповіді на питання та обмінюватися best practices.

У результаті, вибір Visual Studio та C# для розробки сервера забезпечує комфортне та професійне середовище, інструменти розробки та потужну мову програмування, що дозволяє створювати високоякісні та масштабовані серверні програми.

C# — потужна та гнучка мова програмування, яка стала стандартом для створення вебсайтів на основі ASP.NET. Він пропонує ряд функцій і

переваг, які роблять його популярним вибором для розробників, які хочуть створювати безпечні, ефективні та гнучкі вебпрограми.

C# — це мова програмування, розроблена Microsoft як частина .NET framework. Вона була вперше випущена в 2000 році і з тих пір стала однією з найпопулярніших мов програмування в світі, яка використовується для широкого спектру програм, включаючи веброзробку, програмне забезпечення для настільних ПК, мобільні програми та ігри.

2.1.1 Історія C#

C# створила команда під керівництвом Андерса Хейлсберга, який раніше працював над розробкою популярної мови програмування Turbo Pascal. Мова була розроблена як сучасна об'єктно-орієнтована мова, якою розробники могли б легко користуватися та вивчати її. C# також було розроблено як мову, яку можна використовувати для широкого спектру програм, включаючи веброзробку, програмне забезпечення для настільних комп'ютерів і мобільні програми [6].

2.1.2 Приклади використання

C# використовується для широкого спектру програм, включаючи веброзробку, програмне забезпечення для настільних ПК, мобільні програми та ігри. Деякі з найпопулярніших програм, створених на C#, включають Microsoft Office, Visual Studio та саму операційну систему Windows [6].

У веброзробці C# зазвичай використовується для створення вебдодатків на основі ASP.NET. Це включає як зовнішню, так і бек-енд розробку, причому C# використовується для створення коду на стороні сервера, який забезпечує роботу вебдодатків. C# також використовується в поєднанні з такими популярними фреймворками веброзробки, як AngularJS і ReactJS.

У розробці програмного забезпечення для настільних ПК C# зазвичай використовується для створення додатків Windows, включаючи інструменти, утиліти та ігри. C# є популярним вибором для розробки програмного забезпечення для настільних комп'ютерів через його простий у вивченні синтаксис і здатність створювати потужні, багатофункціональні програми.

У розробці програм для мобільних пристроїв C# зазвичай використовується для створення програм для платформ Windows Phone і Windows Store. C# також використовується в поєднанні з міжплатформенними фреймворками розробки, такими як Xamarin, що дозволяє розробникам створювати програми, які працюють на кількох платформах, використовуючи одну кодову базу [7].

2.1.3 Стандартизм

C# — це стандартизована мова програмування, розроблена Microsoft як частина .NET framework. Це означає, що він має послідовний і передбачуваний синтаксис, що полегшує його вивчення та використання розробниками [8]. Це також зріла мова, яку багато років перевіряли та вдосконалювали, що робить її надійним вибором для створення вебдодатків.

2.1.4 Простота використання

Однією з ключових особливостей C# є його простота. Він має чистий синтаксис, який легко читати та розуміти, що робить його доступним для розробників будь-якого рівня кваліфікації. Мова розроблена так, щоб її було легко вивчати та використовувати, з акцентом на продуктивність та ефективність [9].

2.1.5 Ефективність

C# — це високоефективна мова, розроблена для швидкої роботи та ефективного використання системних ресурсів. Це робить його ідеальним вибором для створення великомасштабних вебдодатків, які потребують високої продуктивності та масштабованості. Він також пропонує ряд функцій і оптимізацій, які допомагають розробникам писати швидкий і ефективний код.

2.1.6 Безпека використання

Безпека є критичною проблемою для будь-якої вебпрограми, і C# пропонує ряд функцій, які допомагають забезпечити безпеку та цілісність вебпрограм, створених за допомогою цієї мови. Наприклад, мова включає вбудовану підтримку шифрування та хешування, що полегшує захист конфіденційних даних. Він також містить такі функції, як безпека доступу до коду, що дозволяє розробникам визначати та застосовувати політики безпеки у своїх програмах.

2.1.7 Гнучкість процесу розробки

C# — це дуже гнучка мова, яку можна використовувати для створення широкого спектру вебдодатків. Він пропонує підтримку низки парадигм програмування, включаючи об'єктно-орієнтоване, процедурне та функціональне програмування. Це означає, що розробники можуть вибрати стиль програмування, який найкраще відповідає їхнім потребам і потребам їх програми [10].

2.1.8 Доступ до відкритого джерела

Майкрософт розробила C# з відкритим вихідним кодом, що означає, що тепер розробники можуть робити внесок у розвиток мови та пов'язаних з нею фреймворків. Це призвело до створення яскравої та активної спільноти розробників, які постійно працюють над покращенням і вдосконаленням мови.

2.1.9 Бази даних

C# пропонує чудову підтримку для роботи з базами даних із вбудованою підтримкою популярних систем баз даних, таких як SQL Server і MySQL. Це дозволяє легко створювати потужні та гнучкі вебдодатки, які можуть зберігати та маніпулювати великими обсягами даних. Мова також містить такі функції, як LINQ, що дозволяє розробникам запитувати бази даних за допомогою знайомого синтаксису [11].

2.1.10 Порівняння аналогів та аналіз шаблону проєктування

Нижче я приведу таблицю порівняння C# з іншими можливими аналогами для розробки серверних рішень (таблиця 2.1).

Таблиця 2.1 – Порівняння мов програмування клієнт-серверних рішень

№	Критерій	C#	JavaScript	Python
1	2	3	4	5
1	Використання фреймворків	має .NET Framework і ASP.NET 3	має Node.js і Express.js	має популярні фреймворки, такі як Django та Flask

Продовження таблиці 2.1

1	2	3	4	5
2	Популярність використання	широко використовувався для розробки на стороні сервера на основі Windows	стає все більш популярним для розробки на стороні сервера з появою Node.js	стає все більш популярним для розробки на стороні сервера
3	Робота з базою даних	Entity Framework забезпечує високорівневу абстракцію над базами даних	Має фреймворк Node.js	Має фреймворк Django
4	Мовний поріг входу	має крутішу криву навчання, ніж деякі інші мови, але забезпечує міцну основу для розробки складних програм	його синтаксис схожий на інші популярні мови програмування	відомий своєю простотою та читабельністю, завдяки чому його легко вивчити
5	Майбутні перспективи мови	має потужну екосистему та підтримується корпорацією Майкрософт	Останніми роками суттєво зріс популярність, завдяки зростанню Node.js	популярність значно зросла в останні роки
6	Читабельність	має синтаксис, подібний до інших мов на основі C	може бути схильний до заплутаного коду через свою гнучкість, але може бути написаний у зрозумілий для читання спосіб	відомий своєю читабельністю та простотою
7	Синтаксис	подібна до інших мов на основі C, з акцентом на фігурні дужки та крапку з комою	має синтаксис, подібний до інших мов на основі C, з акцентом на фігурні дужки та крапку з комою	має чистий і читабельний синтаксис із акцентом на відступи та пробіли
8	Відома кампанія з використанням мови	Microsoft, Dell, Visual Studio	Netflix, LinkedIn, Uber, PayPal	Instagram, Google, Spotify, Dropbox

Шаблон проєктування компонентів я обрав MVC (Model-View-Controller). Нижче на рисунку 2.1 наведено базову схему його роботи.

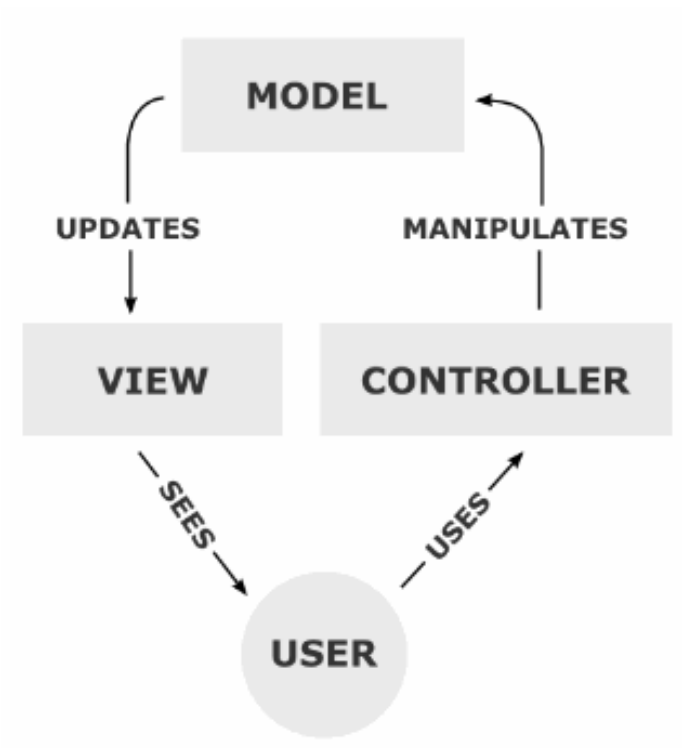


Рисунок 2.1 – Базова схема роботи MVC

MVC (Model-View-Controller) — це шаблон проєктування, який використовується в ASP.NET для створення вебдодатків. Він забезпечує чіткий розподіл завдань між логікою програми та рівнем презентації, що дозволяє краще організувати та підтримувати код [12].

Шаблон MVC складається з трьох основних компонентів:

- модель представляє дані та логіку програми. Він відповідає за отримання та зберігання даних, а також за виконання будь-яких необхідних обчислень або бізнес-логіки;
- представлення відповідає за відображення даних користувачеві в певному форматі. Він взаємодіє з моделлю, щоб отримати дані та відобразити їх користувачеві;

– контролер діє як посередник між моделлю та видом. Він отримує дані користувача та відповідно оновлює модель, а потім вибирає відповідне подання для відображення оновлених даних.

Шаблон MVC в ASP.NET працює, розділяючи програму на окремі модулі на основі їх функції. Модель містить дані програми, представлення відображає дані, а контролер обробляє введені користувачем дані та відповідно оновлює модель і представлення.

В ASP.NET контролер реалізовано як клас, який обробляє HTTP-запити та генерує відповіді. Він взаємодіє з моделлю для отримання або зміни даних і вибирає відповідне подання для відображення оновлених даних. Подання реалізовано як файл HTML, який містить розмітку та будь-яку необхідну логіку програмування для відтворення даних користувачеві.

Принцип роботи вебдодатку розробленому на C#, ASP.NET MVC представлений на рисунку 2.2.

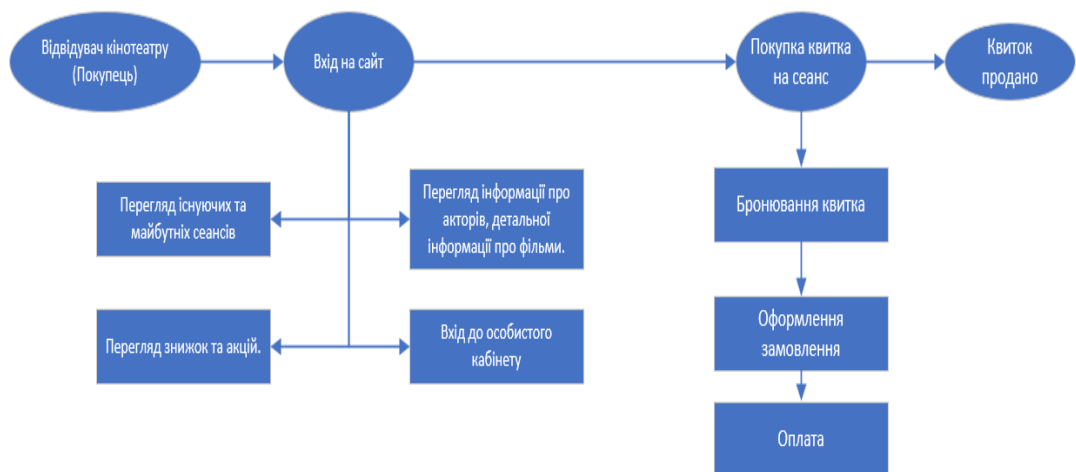


Рисунок 2.2 – Блок-схема роботи вебзастосунку на ASP.NET MVC

Таким чином, C# є потужною та гнучкою мовою програмування, яка добре підходить для створення вебдодатків на основі ASP.NET. Він пропонує ряд функцій і переваг, включаючи простоту, ефективність, безпеку, гнучкість

і відмінну підтримку роботи з базами даних. Завдяки своїй природі з відкритим кодом і живій спільноті C# є чудовим вибором для розробників, які хочуть створювати потужні, безпечні та масштабовані вебпрограми.

2.2 Вибір середовища програмування для клієнтської частини сайту

Вебзастосунок – це програма, яка працює через браузер та дозволяє користувачам взаємодіяти з вебсервером. Вони розробляються за допомогою вебтехнологій, таких як HTML, CSS, JavaScript та різні серверні мови програмування, такі як PHP, ASP.NET або Python. Вебпрограми можуть виконувати широкий спектр завдань, від створення простих блогів до складних бізнес-додатків. Вони можуть бути доступні в будь-який час та з будь-якого пристрою з підключенням до Інтернету. Вебпрограми надають користувачеві зручний і легко доступний інтерфейс, що робить їх популярними та зручними для використання.

Клієнтська частина сайту – це частина вебсайту, що відображається за клієнта, тобто. на комп'ютері або іншому пристрої користувача. Вона включає всі елементи, які бачить і використовує користувач, такі як графічні зображення, текст, посилання, кнопки і форми. Клієнтська частина сайту написана мовами веброзмітки, таких як HTML, CSS та JavaScript, та відповідає за візуальне подання та інтерактивність сайту. Ця частина сайту працює на стороні клієнта, використовуючи ресурси браузера користувача та обмінюючись даними із сервером за допомогою протоколів, таких як HTTP та WebSocket.

Кінцева схема роботи сайту від надсилання запиту з боку клієнта до обробки на ASP.NET може виглядати так:

Клієнт відправляє HTTP-запит на сервер, вказуючи URL-адресу сторінки, яку потрібно отримати. Цей запит може містити дані в запиті, якщо це необхідно. Послідовність обробки запитів зазвичай є наступною:

- сервер приймає запит та надсилає його на ASP.NET;
- ASP.NET отримує запит та створює новий потік для обробки запиту;
- ASP.NET розбирає запит та визначає, який обробник запиту необхідно викликати, на основі URL-адреси та інших параметрів запиту;
- обробник запиту виконує необхідні дії, такі як читання даних із бази даних, обробку введених користувачем даних тощо;
- обробник запиту генерує HTML-код, який відображатиметься на сторінці веббраузера клієнта;
- ASP.NET повертає HTML-код клієнту у вигляді HTTP-відповіді, який містить статус відповіді, заголовки та тіло відповіді;
- браузер клієнта отримує відповідь та відображає HTML-код на сторінці.

ASP.NET надає різні способи відображення даних на вебсторінці. Розглянемо деякі з них [14]:

- .cshtml: Це мова розмітки для створення динамічних вебсторінок в ASP.NET. У файлі .cshtml можна використовувати синтаксис C# для відображення даних на сторінці. Щоб відобразити дані на вебсторінці за допомогою .cshtml, можна використовувати HTML-теги та розмітку, а також C#-код для вилучення даних із бази даних або іншого джерела;
- .razor: Це нова мова розмітки, введена в ASP.NET Core 3.0. Він використовується для створення динамічних вебсторінок і забезпечує більш простий та інтуїтивно зрозумілий синтаксис, ніж .cshtml. Файли .razor також можуть містити код C# для відображення даних на сторінці;
- .css: Це мова таблиць стилів, яка використовується для керування зовнішнім виглядом вебсторінки. Файли .css містять інструкції щодо форматування елементів вебсторінки, таких як текст, зображення та кнопки. Вони можуть бути використані для зміни кольору, шрифту, розміру та інших аспектів елементів на сторінці;

– .js: Це мова програмування, яка використовується для створення інтерактивних елементів на вебсторінці. Файли .js можуть використовуватися для обробки подій, таких як клацання миші або надсилення форми, та для взаємодії з елементами на сторінці. Наприклад, скрипти можуть бути використані для перевірки правильності заповнення форми перед її відправкою на сервер.

Використання цих способів відображення даних на вебсторінці в ASP.NET може суттєво покращити інтерфейс користувача та функціональність вебсайту. Вибір способу залежить від конкретних потреб проєкту та переваг розробника.

2.3 Обґрунтування вибору середовища функціонування системи

ASP.NET – це платформа для розробки вебзастосунків, розроблена компанією Microsoft. Вона надає розробникам потужний інструментарій для створення високопродуктивних та масштабованих вебдодатків. Одним із ключових рішень, які необхідно прийняти під час роботи з ASP.NET, є вибір середовища функціонування сайту. Ця стаття розгляне основні фактори, які слід враховувати під час вибору середовища функціонування сайту на ASP.NET [13].

2.3.1 Браузери

ASP.NET дозволяє створювати вебпрограми, які можуть працювати в різних браузерах. Однак, різні браузери можуть інтерпретувати HTML, CSS та JavaScript по-різному, що може призвести до проблем з відображенням та функціонуванням сайту. Тому, при виборі браузерів, на яких буде працювати сайт, необхідно враховувати їхню частку на ринку, тестувати роботу сайту на різних браузерах і переконатися, що він коректно відображається та функціонує у всіх браузерах, що підтримуються.

Деякі з найпопулярніших браузерів, які можна використовувати для запуску сайту на ASP.NET, включають Google Chrome, Mozilla Firefox, Safari, Microsoft Edge і Opera. Однак, варто пам'ятати, що кожен із цих браузерів має свої особливості та вимагає певного підходу при розробці сайту.

2.3.2 Операційні системи

ASP.NET може бути запущений на різних операційних системах, включаючи Windows та Linux. При виборі системи на якій запускатиметься сайт, необхідно враховувати безліч факторів, таких як продуктивність, вартість, зручність використання та сумісність з іншими системами. Одним з головних факторів є зручність розгортання програми та її масштабованість.

Windows є основною платформою для розробки та запуску ASP.NET програм. Однак, Linux стає все більш популярною платформою для розробки вебдодатків і має безліч переваг, таких як безкоштовність, висока продуктивність та безпека.

2.3.3 Порівняння середовищ функціонування системи

Нижче, у таблиці 2.2, я представлю інформацію з підтримки ОС різними браузерами.

Таблиця 2.2 – Підтримка ОС різними браузерами

Браузер	Google Chrome	Mozilla Firefox	Microsoft Edge	Opera
Підтримувані ОС	Windows, macOS, Linux, Chrome OS, Android и iOS	Windows, macOS, Linux, Android и iOS	Windows, macOS, Linux, Android и iOS	Windows и macOS

Нижче в таблиці 2.3 я предоставив інформацію про функціональні можливості браузерів.

Таблиця 2.3 – Можливості браузерів

Критерій	Chrome	Mozilla	Edge	Opera
Вкладки	+	+	+	+
Менеджер завантажень	+	+	+	+
Панель пошуку	+	+	+	+
Перевірка орфографії	±	+	+	+
Блокування банерів	+	+	-	+
Блокування Pop-Up	+	+	+	+

Також існують відмінності у підтримці веб технологій та протоколів, нижче у таблиці 2.4 я приведу порівняння підтримки технологій та протоколів.

Таблиця 2.4 – Порівняння підтримки технологій і протоколів

Критерій	Chrome	Mozilla	Edge	Opera
CSS3	+	+	+	+
Фрейми	+	+	+	+
JavaScript	+	+	+	+
XHTML	-	+	-	+
RSS	-	+	+	+

У висновку можу сказати, що вирішив вибрати Google Chrome в якості рекомендованого браузера для відображення клієнтської частини сайту, так як в ньому підтримуються усі потрібні технології останніх версій.

2.4 Бази даних

Для реалізації збереження даних я обрав реляційну базу даних SQLite, де в основі лежить мова доступу до таблиць SQL.

SQLite – це легковажна реляційна база даних, що вбудовується, яка не вимагає наявності окремого сервера управління базами даних. Вона зберігає дані в локальному файлі і може бути використана в широкому спектрі програм, включаючи вебдодатки, настільні програми та мобільні програми.

SQLite підтримує більшість функцій, які можна знайти у повному реляційному СУБД, таких як індекси, транзакції, підтримку SQL-запитів тощо. Крім того, SQLite є дуже надійною та стабільною базою даних, що робить її ідеальним вибором для додатків з високими вимогами до надійності та безпеки даних.

У контексті проєкту ASP.NET використання SQLite має кілька переваг. По-перше, так як SQLite зберігає дані в локальному файлі, його легко впровадити в програму без необхідності налаштування та керування окремим сервером баз даних. Це робить процес розгортання проєкту більш простим та швидким, а також економічно вигідним.

По-друге, SQLite має гарну продуктивність, що особливо важливо для вебдодатків, де висока швидкість відповіді сервера є критичним фактором. SQLite підтримує багатопоточність та асинхронні операції, що дозволяє йому забезпечувати високу швидкість обробки запитів та паралельну обробку запитів від кількох користувачів.

Нарешті, SQLite є дуже надійним і стабільним рішенням, що робить його ідеальним вибором для додатків, де потрібна збереження даних. SQLite

має механізм транзакцій, який дозволяє забезпечувати цілісність даних у разі збоїв у роботі програми або раптового відключення живлення.

У SQLite ви можете використовувати кілька типів даних для зберігання своїх даних, зокрема:

- NULL – представляє нульове значення;
- INTEGER – представляє ціле число зі знаком;
- REAL – представляє число з плаваючою комою;
- TEXT – представляє текстовий рядок;
- BLOB – представляє двійкові дані;
- Крім того, SQLite має кілька інших типів, які базуються на цих основних типах, наприклад;
- NUMERIC – представляє числове значення, яке може бути цілим чи числом з плаваючою комою, залежно від значення;
- DATA – представляє дату у форматі «PPPP-MM-ДД»;
- ЧАС – представляє час у форматі «ГГ:ХХ:СС»;
- TIMESTAMP – представляє позначку часу у форматі «PPPP-MM-ДД ГГ:ХХ:СС»;
- BOOLEAN – представляє логічне значення, яке може бути істинним або хибним.

Серед переваг SQLite я можу виділити :

- простий у використанні: SQLite легко встановити, налаштувати та використовувати. Це легка система керування базами даних, яка не потребує налаштування або адміністрування сервера;
- портативність: SQLite — це автономна база даних, яка працює на різних платформах, таких як Windows, Mac, Linux, Android та iOS. Він ідеально підходить для програм, яким потрібна локальна база даних;
- невеликий розмір: SQLite має невеликі розміри та вимагає мінімальних системних ресурсів. Він підходить для вбудованих систем, мобільних пристроїв та інших програм з обмеженим обсягом пам'яті;

- сумісність з ACID: SQLite сумісний з ACID, що означає, що він гарантує надійну передачу даних навіть у разі втрати живлення або збою системи;

- немає єдиної точки відмови: оскільки SQLite є базою даних на основі файлів, немає єдиної точки відмови. У разі пошкодження або пошкодження одного файлу доступ до інших файлів усе одно буде доступним;

- економічно: SQLite є відкритим вихідним кодом і вільним для використання, що робить його чудовим вибором для розробників та організацій з обмеженим бюджетом;

- висока продуктивність: SQLite — це швидка та ефективна система керування базами даних, яка може легко обробляти великі обсяги даних.

Також серед можливостей SQLite я виділю ці фактори:

- підтримка SQL: SQLite підтримує підмножину SQL (Structured Query Language), включаючи загальні операції, такі як SELECT, INSERT, UPDATE і DELETE;

- транзакції: SQLite підтримує транзакції для забезпечення цілісності даних. Транзакції дозволяють виконувати кілька операцій як одну одиницю роботи, і їх можна відкотити у разі помилки або збою;

- індексування: SQLite забезпечує функцію індексування для покращення продуктивності запитів до бази даних. Індеси створюються для певних стовпців у таблиці, що забезпечує швидший пошук і сортування;

- тригери: SQLite дозволяє визначати тригери, які є автоматизованими процедурами, які виконуються у відповідь на певні події бази даних. Тригери можна використовувати для аудиту, перевірки та інших цілей;

- перегляди: SQLite підтримує створення представлень, які є віртуальними таблицями, заснованими на результатах запиту;

- представлення можна використовувати для спрощення складних запитів і підвищення продуктивності бази даних;

- оптимізація запитів: SQLite автоматично оптимізує запити для підвищення продуктивності шляхом аналізу структури запиту та вибору найкращого плану виконання;

- резервне копіювання та відновлення: SQLite забезпечує функції резервного копіювання та відновлення, дозволяючи легко створювати резервні копії та відновлювати файли бази даних у разі втрати чи пошкодження даних.

Загалом SQLite надає широкий спектр функціональних можливостей для підтримки ефективного та надійного зберігання та пошуку даних. Його легкий дизайн, крос-платформна підтримка та простота використання роблять його чудовим вибором для широкого спектру додатків, включаючи мобільні додатки, настільні додатки та вбудовані системи.

2.5 Висновки за розділом

У цьому розділі описується процес вибору відповідного середовища розробки на основі критеріїв, встановлених під час постановки завдання, а також збірка необхідних інструментів для проєкту. Пояснюється обґрунтування вибору мови програмування та середовища функціонування системи. Зокрема, ASP.NET був обраний для розробки серверних компонентів вебсайту, .cshtml і Razor Pages – для клієнтського інтернет-магазину. Крім того, СУБД SQLite була обрана як оптимальне рішення для баз даних для вимог проєкту.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Основні відомості про розробку сайту для продажу квитків у кіно

Сайт для продажу квитків у кіно дозволяє робити бронювання та оплату квитків у режимі онлайн, також у режимі онлайн співробітники кінотеатру мають змогу вносити зміни, додавати, видаляти сеанси які доступні для відвідувачів.

Мова програмування котру я обрав для реалізації онлайн магазину для продажу квитків у кіно я обрав C#, з технологією подубування серверних рішень ASP.NET MVC з базою даних SQLite під управлінням Entity Framework Core.

Основною бізнес логіки на сайті інтернет продажу є саме запис, редагування бази даних, власне я хочу описати базу даних SQLite, приклади роботи з даними, та використання EntityFramework Core.

Entity Framework Core (EF Core) – це об'єктно-реляційний мапер (ORM), який дозволяє працювати з даними у базі даних у вигляді об'єктів. Він надає безліч функцій, які полегшують доступ до даних і полегшують взаємодію з базою даних.

EF Core інтегрується з ASP.NET і дозволяє розробникам використовувати об'єкти C# для доступу до даних у базі даних. EF Core використовує модель Code First, яка дозволяє визначити модель даних C# і генерувати базу даних на основі цієї моделі.

EF Core підтримує різні провайдери баз даних, такі як Microsoft SQL Server, MySQL, PostgreSQL, SQLite та інші. Він також підтримує багато функцій, таких як відносини між таблицями, транзакції, агрегування даних та інші [15].

Принцип роботи EF Core полягає у створенні моделі даних C# і використанні цієї моделі для взаємодії з базою даних. EF Core автоматично

генерує SQL-запити на основі операцій, що виконуються з об'єктами моделі, та виконує ці запити на базі даних. EF Core також дозволяє використовувати мову LINQ для запитів до даних.

Я приведу приклад використання EF Core, з SQL запитом який він автоматично генерує.

Код C#:

```
var result = dbContext.Movies
    .Where(c => c.Name == "Spider Man 2")
    .ToList();
```

SQL запит до бази даних:

```
SELECT * FROM Movies WHERE Name = 'Spider Man 2'
```

Отже, використання Entity Framework Core дозволяє зручніше використовувати базу даних та заощаджувати час при розробці, ще великою перевагою EF Core є вбудований захист від SQL ін'єкцій.

Далі я буду приводити приклади запитів з функціоналу інтернет магазину для продажу квитків до бази даних саме через використання EF Core та LINQ.

Архітектурне рішення для зручної роботи з базою даних я обрав через сервіси DI і наслідування, для економії часу та зменшення коду. Принцип роботи такий : є базовий клас-сервіс який працює з базою даних з Generic моделлю даних з базовими операціями CRUD, та для кожної моделі у базі я унаслідував цей клас, і завантажив у Dependency Injection. Та на сторінках сайту, де ми взаємодіємо з певною моделлю, ми виграємо з DI потрібні сервіси та маємо зручний доступ до бази.

CRUD (Create Read Update Delete) – це принцип проєктування роботи з базою даних, який значить, що на кожен модель у базі даних потрібно обов'язково реалізувати 4 операції : створення моделі, читання моделі, редагування та видалення.

Dependency Injection (DI) – це патерн проєктування, який використовується для керування залежностями об'єктів у додатку. У DI залежності впроваджуються в об'єкти, замість того, щоб створювати їх всередині об'єктів самостійно. Це дозволяє гнучкіше керувати об'єктами та їх залежностями, а також полегшує тестування програми. DI може бути реалізований різними способами, включаючи конструктори, сетери та інтерфейси. Добре реалізований DI полегшує підтримку коду, розширення та повторне використання.

Ініціалізація бази даних та сервісів для роботи з моделями у проєкті через DI:

```
public void ConfigureServices(IServiceCollection services)
{
    //DbContext configuration
    services.AddDbContext<AppDbContext>(options =>
options.UseSqlite(Configuration.GetConnectionString("DefaultConnectionString")));

    //Services configuration
    services.AddScoped<IActorsService, ActorsService>();
    services.AddScoped<IProducersService, ProducersService>();
    services.AddScoped<ICinemasService, CinemasService>();
    services.AddScoped<IMoviesService, MoviesService>();
    services.AddScoped<IOrdersService, OrdersService>();
}
```

Приклади та реалізації взаємодії з базою даних я приведу у додатку Б.8.

Інтерфейс базової моделі для нашої системи бази даних знаходиться у додатку Б.8.1, він забезпечує те, що кожна модель з якою ми будемо працювати, буде містити у собі ідентифікатор у вигляді хеш ключа, для забезпечення індексації моделей у базі даних.

Інтерфейс сервісу який буде взаємодіяти з базою даних я приведу у додатку Б.8.2, він забезпечує, що будь який сервіс для бази даних з яким

працює вебзастосунок буде зобов'язаний мати у собі реалізовані методи пошуку, редагування та додавання моделі.

Головний сервіс з реалізованими методами описаними у інтерфейсі я приведу у додатку Б.8.3, у цьому сервісі було реалізовано методи доступу, редагування та додавання моделей за допомогою Entity Framework.

Власне, цей великий об'єм коду на архітектурному рівні дозволяє нам додавати у систему нові моделі написавши лише декілька строк коду. Нижче я приведу приклад сервісу який працює в базі даних з моделлю Actor, для інших моделей код буде той самий.

```
public class ActorsService : EntityBaseRepository<Actor>, IActorsService
{
    public ActorsService(AppDbContext context) : base(context) { }
}
```

Приклад отримання моделей з бази даних у середовищі розробки:

```
IMoviesService _moviesService;
IActorsService _actorsService;
IMoviesService _ordersService;
var movies = await _moviesService.GetAllAsync();
var actors = await _actorsService.GetAllAsync();
var orders = await _ordersService.GetAllAsync();
```

3.2 Алгоритм замовлення у онлайн сервісі продажу квитків

Користувач сайту з продажу квитків може здійснити покупку, просто обравши потрібні фільми у кошик та оформивши замовлення.

Оформлення замовлення проходить таким чином : користувач заходить на сайт, на головній сторінці відображаються усі поточні фільми які доступні для покупки, вибирає фільми до кошика, заходить на сторінку кошика та оформляє покупку.

Нижче я приведу код для реалізації роботи кошика.

Додавання фільму до кошика:

```

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

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

```

Видалення фільму з кошика:

```

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

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

```

Підтвердження замовлення:

```

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");
}

```

Також я додав до цього процесу, оплату за допомогою сервісу PayPal. Код для цієї можливості я привів у додатку Б.9.19.

На місці змінної PAYPAL_API_KEY потрібно вставити ключ доступу API до сервісу PayPal. Нижче я приведу рисунок 3.1, на якому можна побачити схему роботи оформлення замовлення.

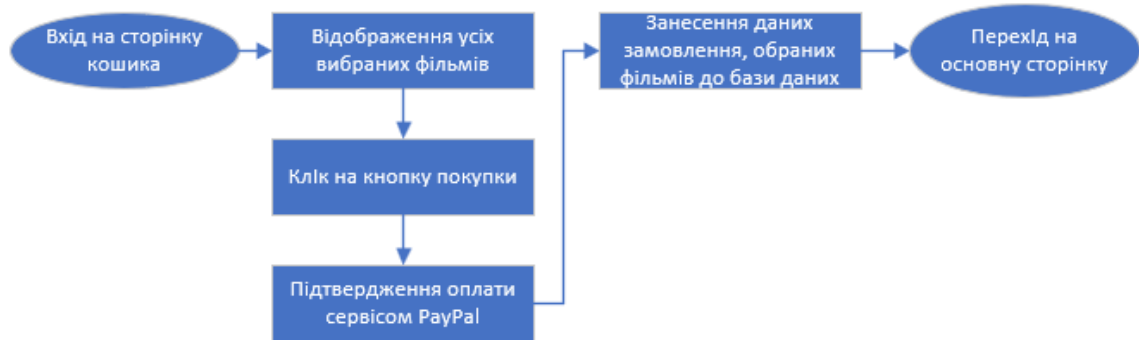


Рисунок 3.1 – Схема роботи оформлення замовлення

3.3 Алгоритм роботи головної сторінки сервісу з продажу квитків

Після того, як користувач перейшов на сайт, ASP.NET шукає головний компонент для відображення – `_Layout.cshtml`. У ньому вказана головна структура відображення сайту (шапка сайту, підвал, навігація), та на місці `RenderBody()` буде відображено контент, який потрібно користувачу. У нашому випадку при переході на головну сторінку, у `RenderBody()` підставиться те що я вказав у методі `Configure()` у файлі `Startup.cs`.

```

app.UseEndpoints(endpoints =>
{
    endpoints.MapControllerRoute(
        name: "default",
        pattern: "{controller=Movies}/{action=Index}/{id?}");
});

```

Це означає, що про завантаженні сторінки стандартний контент для відображення буде надходити з файлу `Movies.cshtml`, з шаблоном сторінки `_Layout.cshtml`, код який описує ці компоненти знаходяться у додатках Б.9.17 і Б.9.19

Також у алгоритмі головної сторінки та компоненту відображення фільмів існує перевірка на доступ до даних. Наприклад для адміністратора буде доступно редагування, видалення та додавання фільмів. Для звичайного користувача ці можливості не будуть відображені.

Нижче я приведу приклад з розрізненням даних для різних ролей у системі.

```
@if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
{
<a class="text-white float-right" asp-action="Edit" asp-route-id="@item.Id"><i
class="bi bi-pencil-square"></i></a>
}
```

Цей код вказує на те що якщо користувач адміністратор – то для нього напроти кожного фільма буде відображена кнопка редагування, такі можливості реалізовані і у інших компонентах відображення.

На рисунку 3.2 я зобразив блок схему роботи головної сторінки.



Рисунок 3.2 – Блок схема роботи головної сторінки

3.4 Алгоритм роботи адміністративної частини

Адміністративна частина дуже важлива при розробці рішень для бізнесу, для спрощення редагування актуальних даних. У вебзастосунку я реалізував адміністративну частину через авторизацію і визначення ролі користувача. Користувачі та їх ролі зберігаються у базі даних під керуванням допоміжної бібліотеки для введення системи юзерів та ролей – ASP.NET Identity. Так, у адміністратора відображаються сторінки для зручного редагування актуальних даних. У адміністратора є панель доступу до редагування усіх моделей з бази даних. Я приведу приклад для однієї моделі – Фільм.

Також приклад коду, який описує розрізнення користувачів по ролям я приведу у додатку Б.2, цим сервісом може користуватись як і адміністратор так і звичайний користувач, але особливі функції помічені атрибутом [Authorize(“Admin”)], що визначає, що доступ до цього запросу буде тільки у адміністратора, будь яка спроба користувача скористатись методами з цим атрибутом буде завершатись помилкою запита 403.

Нижче, я приведу рисунок блок схеми адміністративної частини вебзастосунку на рисунку 3.3.



Рисунок 3.3 – Блок схема роботи адміністративної частини

3.5 Реалізація бази даних

В основі проєктування бази даних лежить структура, зображена на рисунку 3.4.

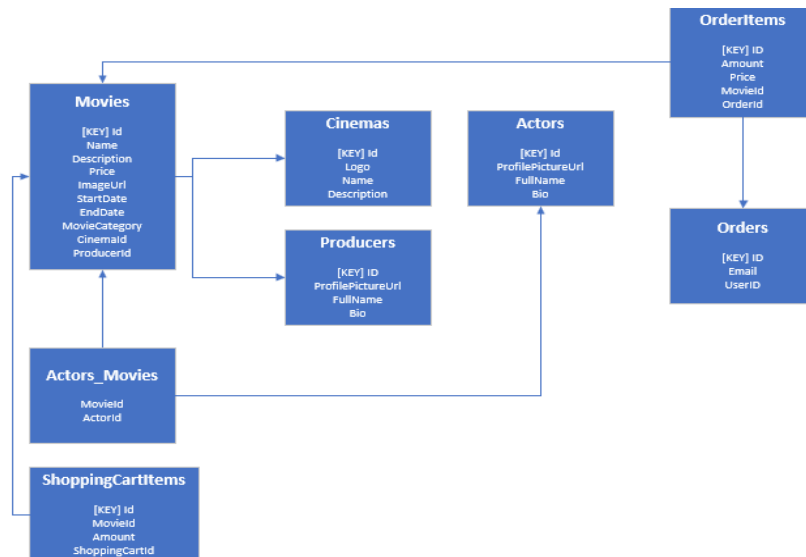


Рисунок 3.4 – Блок схема структури бази даних

В структурі існують такі набори даних (таблиці);

- Movies (Кінофільми) ;
- Cinemas (Кінотеатри);
- Actors (Актори);
- OrderItems (замовлення одного квитка);
- Orders (замовлення);
- ShoppingCartItems (поточна корзина користувача).

3.6 Характеристики розробленого продукту

При розробці інтернет сайту для продажу квитків було реалізовано необхідні логічні модулі, які я зараз опишу нижче з прикладом програмного коду.

3.6.1 Ідентифікація користувача

3.6.1.1 Авторизація

Авторизація проходить таким чином, що з компоненти відображення Login.cshtml, який я привів у додатку Б.9.2, після підтвердження форми дані

надсилаються до сервісу AccountController.cs, методу Login(), який можна побачити у додатку Б.2.

Коли система дійде до цього метода, спочатку алгоритм перевірить модель на валідність, потім перевірить : чи існує користувач у системі, якщо існує, то перевіряємо чи успішна авторизація за паролем який вказав користувач, або ні. Якщо ні, то видаємо помилку, якщо так – відображаємо головну сторінку.

Код методу Login:

```
[HttpPost]
public async Task<IActionResult> Login(LoginVM loginVM)
{
    if (!ModelState.IsValid) return View(loginVM);

    var user = await _userManager.FindByEmailAsync(loginVM.EmailAddress);
    var users = await _userManager.Users.ToListAsync();
    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);
}
```

Нижче, на рисунку 3.5 я приведу блок схему методу авторизації.

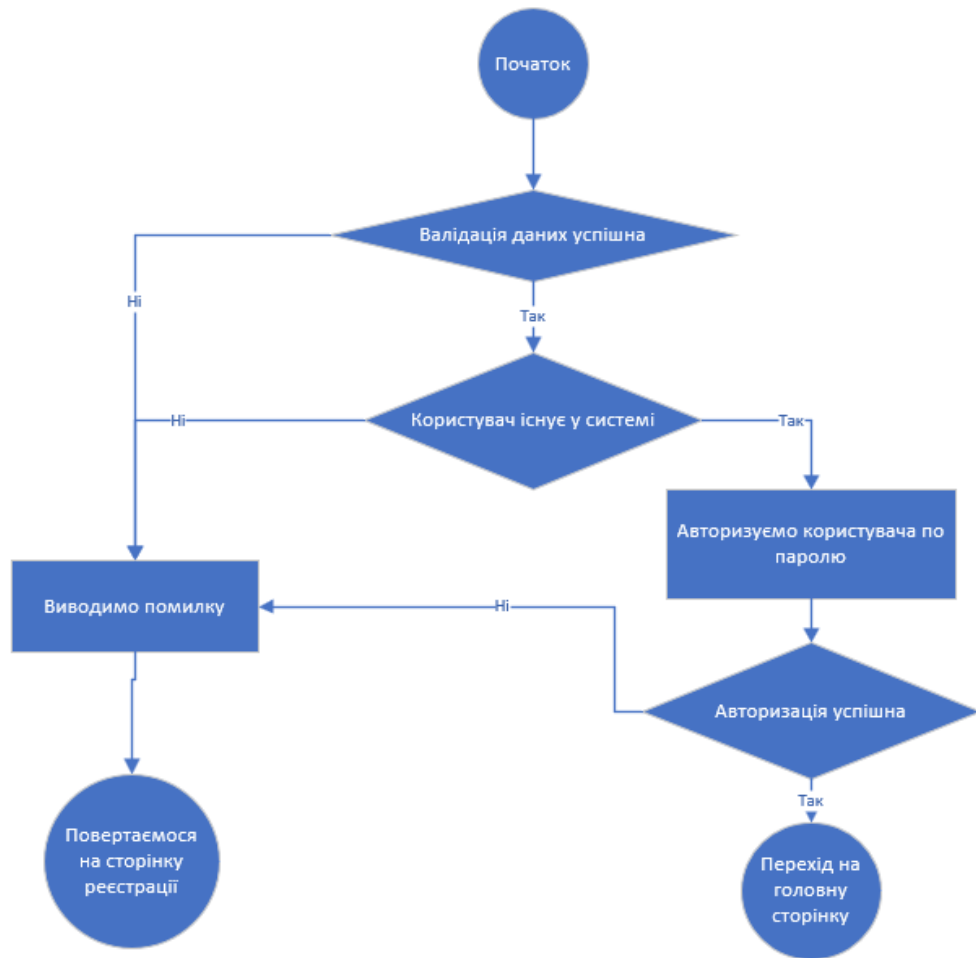


Рисунок 3.5 – Блок схема авторизації користувача

3.6.1.2 Реєстрація

Після того як користувач підтвердить форму реєстрації, яку можна побачити у додатку Б.9.3, дані передадуться у метод `AccountController.Register()`.

Код методу `Register()`:

```

[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
};
var newUserResponse = await _userManager.CreateAsync(newUser, registerVM.Password);

if (newUserResponse.Succeeded)
{
    string role = registerVM.EmailAddress.Contains("admin") ? UserRoles.Admin : UserRoles.User;
    await _userManager.AddToRoleAsync(newUser, role);
}
else
{
    TempData["Error"] = string.Join("\n", newUserResponse.Errors);
    return View(registerVM);
}

return View("RegisterCompleted");
}

```

Нижче на рисунку 3.6 зображена блок схема реєстрації.



Рисунок 3.6 – Блок схема реєстрації користувача

3.6.2 Робота з базою даних

Реалізовані компоненти для роботи з базою на основі класів EntityBaseRepository.cs (ActorsService.cs, MoviesService.cs, и т.д.)

Код EntityBaseRepository.cs:

```
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();
    }
}
```

Цей клас є базовим класом для роботи з базою даних на Entity Framework, також цей клас може працювати з будь якими моделями тому що він generic.

В цьому класі реалізовані методи:

- AddAsync – додає до бази даних модель T;
- DeleteAsync – видаляє з бази даних модель за її ID;
- GetAllAsync – метод який повертає усі моделі вказанного типу, та перевантажений метод який повертає усі моделі за вказанною умовою;
- GetByIdAsync – повертає модель з вказаним ID;
- UpdateAsync – редагує модель з вказаним ID.

Виходячи з цього коду для роботи з базою даних на прикладі моделі Actor, треба усього створити такий клас, і функціонал для роботи з базою з моделю Actor буде працювати через використання ActorsService:

```
public class ActorsService : EntityBaseRepository<Actor>, IActorsService
{
    public ActorsService(AppDbContext context) : base(context) { }
}
```

3.6.3 Компоненти для відображення інтерфейсу

Для відображення інтерфейсу були реалізовані такі компоненти відображення:

_Layout.cshtml, Movies/Index.cshtml, Movies/Edit.cshtml, Movies/Create.cshtml і т.д. для інших моделей. Основою відображення є саме компонент _Layout.cshtml, він задає базове тіло сторінки, та поточний контент, такий як список фільмів(Movies/Index.cshtml), відображається на місці методу RenderBody() в _Layout.cshtml.

Приклади всіх компонентів для відображення інтерфейсу можна подивитись у додатку Б.9.

3.6.4 Процес придбання квитка

У компоненті ShoppingCart.cshtml, код якого можна побачити у додатку Б.9.19 відображається кошик користувача, за допомогою компонента PayPal генерується кнопка покупки. Після натискання, і успішної оплати викликається метод OrdersController.CompleteOrder(), проданий квиток заноситься у базу даних. Нижче я приведу код метода CompleteOrder():

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

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

    return View("OrderCompleted");
}
```

3.6.5 Адміністративна частина

Адміністративна частина реалізована у кожному компоненті, так, що якщо система ідентифікує користувача як адміністратора. То в нього з'являється доступ до списку усіх користувачів. Також напроти кожного компонента з фільмом буде значок редагування. Після його натискання адміністратора переводить на сторінку редагування яка реалізована у компоненті Movies/Edit.cshtml.

Приклад ідентифікування адміністратора при перегляді списку фільмів:

```
<h5 class="card-title">
    @item.Name
    @if (User.Identity.IsAuthenticated &&
    User.IsInRole("Admin"))
    {
        <a class="text-white float-right" asp-action="Edit" asp-
        route-id="@item.Id"><i class="bi bi-pencil-square"></i></a>
```

</h5>

Нижче на рисунку 3.7 я приведу блок схему відображення адміністративної частини на прикладі перегляду списку фільмів.



Рисунок 3.7 – Блок схема роботи визначення адміністратора та функціоналу

Для додавання фільмів, акторів і інших даних у кожному компоненті реалізовані сторінки Create.cshtml, доступ до них є тільки у користувача з роллю адміністратора.

3.7 Структурна схема компонентів магазину

Якщо розглядати інтернет-магазин як вебсайт, його можна розділити на дві основні логічні частини: адміністративну і клієнтську. Адміністративна частина, до якої можна отримати доступ після входу в систему адміністрування, дозволяє адміністратору редагувати базу даних та додавати нові фільми чи акторів. У клієнтській частині, доступ до

можливості редагування даних відсутній і закритий для загального огляду. Клієнтська частина дозволяє користувачам та відвідувачам переглядати, шукати та вибирати фільми з різних жанрів, що спрощує пошук необхідної інформації. Клієнт може оформити замовлення на квиток, та оплатити за допомогою сервіса PayPal. Структура компонентів вебсайту зображена на схемі 3.8.

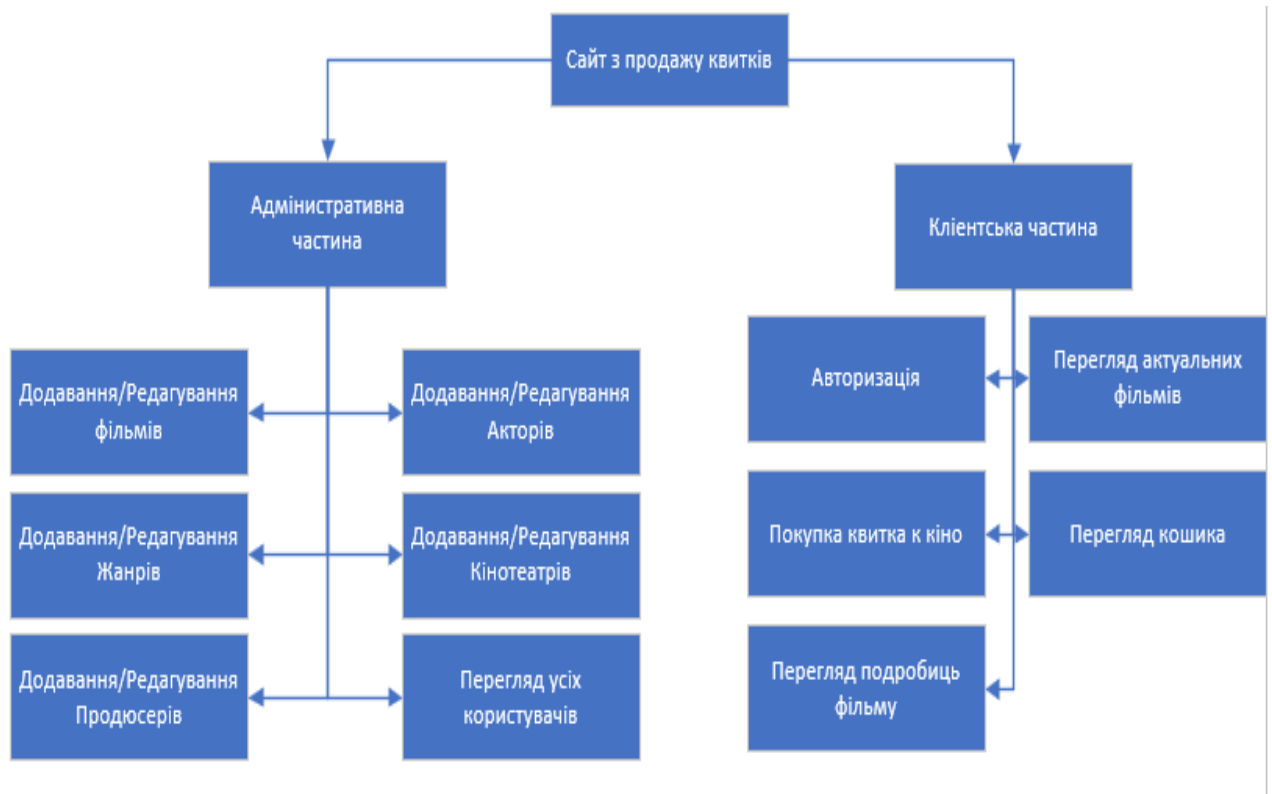


Рисунок 3.8 – Блок схема роботи інтернет магазину

У цьому розділі були представлені основні цілі та умови використання у розробці інтернет-магазину з продажу квитків. Також була описана структура бази даних, включаючи первинні ключі та зв'язки таблиць. Було представлено структурну схему компонентів інтернет-магазину, що показує логічний взаємозв'язок між компонентами, які були розроблені. Нарешті, було створено структуру бази даних інтернет-магазину та описано структурну схему розробленої системи.

3.8 Висновки за розділом

В даному розділі було описано основну інформацію щодо функціонування вебзастосунку, алгоритми роботи користувацької та адміністративної частини. Були розроблені основні алгоритми які необхідні для функціонування вебзастосунку для продажу квитків у кіно. Були описані технології які були використані при розробці функціоналу.

4 КЕРІВНИЦТВО СИСТЕМНОГО АДМІНІСТРАТОРА

4.1 Розгортання інтернет-магазину

Інтернет магазин знаходиться у CinemaTickets.zip

Можливо як і створення локального хоста, так і викладення до мережі інтернет.

Щоб розгорнути сайт в інтернеті, треба виконати наступні кроки:

- завантажити файл CinemaTickets.zip на виділений сервер з Windows 10;
- розпакувати архів у будь-яку папку;
- відкрити IIS;
- додати новий сервер, вибрати порт, та обрати шлях до папки файлами.

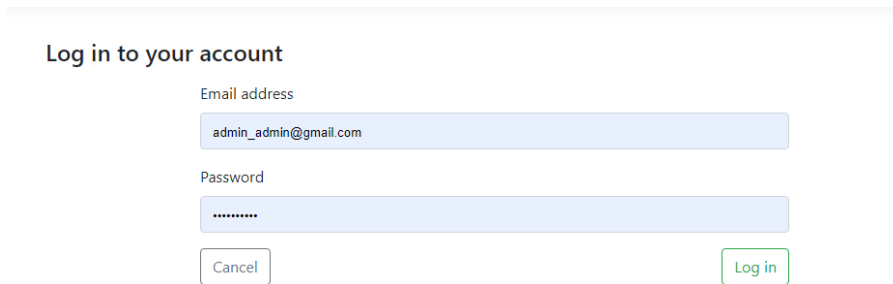
4.2 Розгортання бази даних вебсайту

База даних автоматично розгортається при першому запуску сама у файл .db. Для цього на віртуальному сервері потрібні бути завантажені такі драйвера як sqlite-tools.

4.3 Адміністративна панель інтернет магазину

Вхід до адміністративного режиму знаходиться у вікні входу до аккаунту.

Зображення входу до адміністративної панелі я зображу на рисунку 4.1.



Log in to your account

Email address

admin_admin@gmail.com

Password

Cancel Log in

Рисунок 4.1 – Вхід до адміністративної частини

Після ідентифікації користувача як адміністратора, на головному вікні з’явиться кнопки редагування фільмів, панель усіх користувачів та усіх заказів. Нижче на рисунку 4.2 зображений адміністративний інтерфейс з кнопкою для редагування даних фільму.

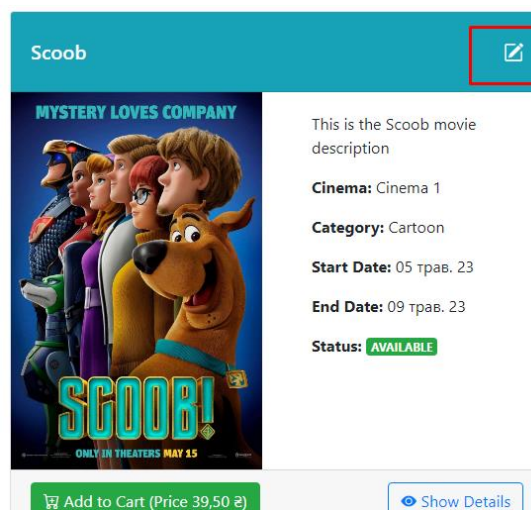


Рисунок 4.2 – Кнопка редагування даних фільму

Після натискання цієї кнопки, адміністратор має можливість повністю редагувати усі дані з фільму. Нижче на рис. 4.3 я приведу рисунок з панеллю редагування фільму.

Update movie

Movie name Scoob	Select a cinema Cinema 1
Movie start date 05/05/2023	Select a category Cartoon
Movie end date 05/09/2023	Select a producer Producer 3
Price in \$ 	Select actor(s) Actor 1 Actor 2 Actor 3 Actor 4
Movie poster URL http://dotmethow.net/images/movies/movie-7.jpeg	
Movie description This is the Scoob movie description	



Рисунок 4.3 – Адміністративна панель редагування фільму

Адміністратор має можливість змінити назву фільму, зміїтини жанр, продюсерів, акторів, фотографію, кінозал, додаткову інформацію, ціну та час початку і кінця продажу квитків на фільм.

Також у адміністратора з'являється додатковий функціонал такий як додавання фільму, після натискання кнопки на рисунку 4.4, адміністратора перенаправляє на сторінку додавання фільму.

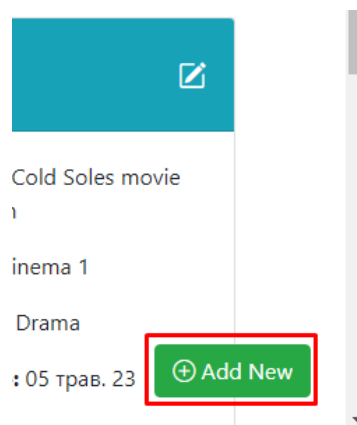


Рисунок 4.4 – Кнопка додавання фільму

Окрім додавання фільму у адміністратора є можливість редагувати інші моделі даних такі як актори, кінозали, продюсери. Функціонал адміністратора я покажу на рисунку 4.5.

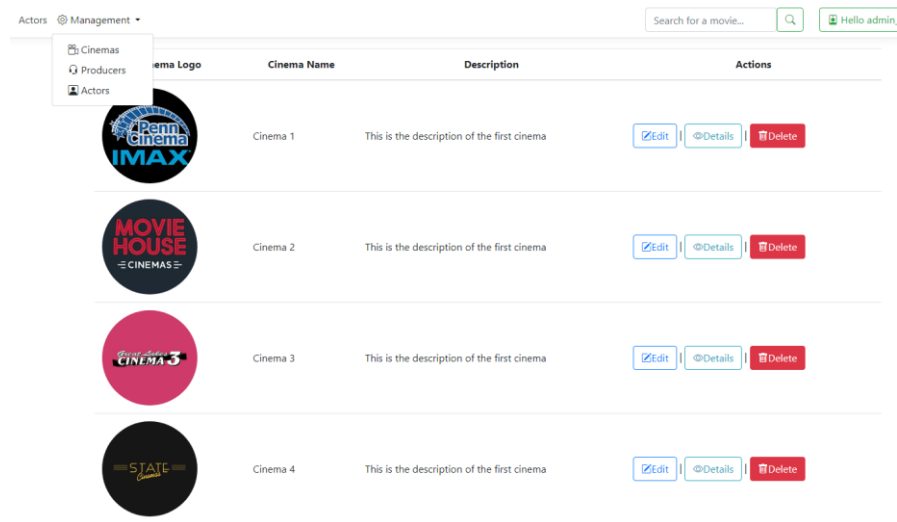


Рисунок 4.5 – Адміністративна панель редагування даних

Також у адміністративній частині передбачено перегляд поточних користувачів. Ця панель дозволяє адміністратору переглядати дані про потік відвідувачів кінотеатру, та для ведення бізнес статистики.

Панель перегляду користувачів сайту я приведу на рисунку 4.6.

List of all users

Full name	UserName	Email
Admin User	admin-user	admin@etickets.com
Application User	app-user	user@etickets.com
Ivan Demchenko	ivan.demchenko.01@gmail.com	ivan.demchenko.01@gmail.com
Admin Admin	admin@gmail.com	admin@gmail.com
Ivan Demchenko	ivan.demchenko.02@gmail.com	ivan.demchenko.02@gmail.com
Admin Fullname	admin_admin@gmail.com	admin_admin@gmail.com

Рисунок 4.6 – Панель перегляду користувачів

Адміністратори інтернет сайту з продажу квитків мають можливість переглядати поточні заброньовані квитки у панелі перегляду замовлень. Нижче, на рисунку 4.7 зображена панель перегляду замовлень.

2	 [0,00 ₴] - Scoob	79,00 ₴	Admin User
1069	 [0,00 ₴] - Life	79,00 ₴	Admin Admin
1468	 [0,00 ₴] - Ghost	79,00 ₴	Application User
2594	 [0,00 ₴] - Scoob	79,00 ₴	Ivan Demchenko
3459	 [0,00 ₴] - Ghost	79,00 ₴	Ivan Demchenko

Рисунок 4.7 – Панель перегляду замовлень

4.4 Висновки за розділом

Цей розділ містить інформацію про призначення та умови використання вебсайту. Тут також описано процес розгортання електронного магазину з продажу квитків та створення віртуального хоста на сервері. Також було розглянуто способи управління адміністративною панеллю магазину, включаючи додавання нових товарів, редагування товарів через адміністративну панель та управління замовленнями.

5 КЕРІВНИЦТВО КОРИСТУВАЧА

5.1 Призначення сайту

Даний веб сайт призначений для відвідувачів кінотеатру, які хочуть заощаджити час та придбати квиток онлайн у зручному вигляді.

Серверна частина сайту була розроблена з використанням технологій ASP.NET MVC, EntityFramework, та базою даних SQLite.

Умовою надійної експлуатації вебсайту є наявність надійного з'єднання з мережею інтернет, а також наявність пакету .NET Framework, та SQLite tools для коректної роботи додатку да бази даних.

Для роботи з веб сайтом потрібен – комп'ютер з пристроями перегляду та введення, з завантаженим браузером, і інтернет з'єднанням.

5.2 Вхід до вебсайту та процес авторизації й замовлення товару

Для відкриття веб сайту потрібно ввести у адресній строці браузеру адресу розгорнутого сайту.

Вікно входу на сайт зображено на рисунку 5.1.

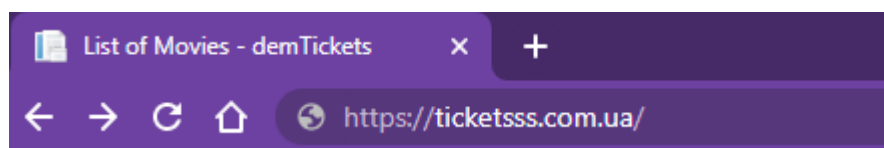


Рисунок 5.1 – Процес входу на сайт

Далі, після завантаження сторінки користувач має побачити головну сторінку з списком актуальних фільмів, їх ціни, та можливість зареєструватися або авторизуватися.

Нижче на рисунку 5.2 я приведу зображення головної сторінки.

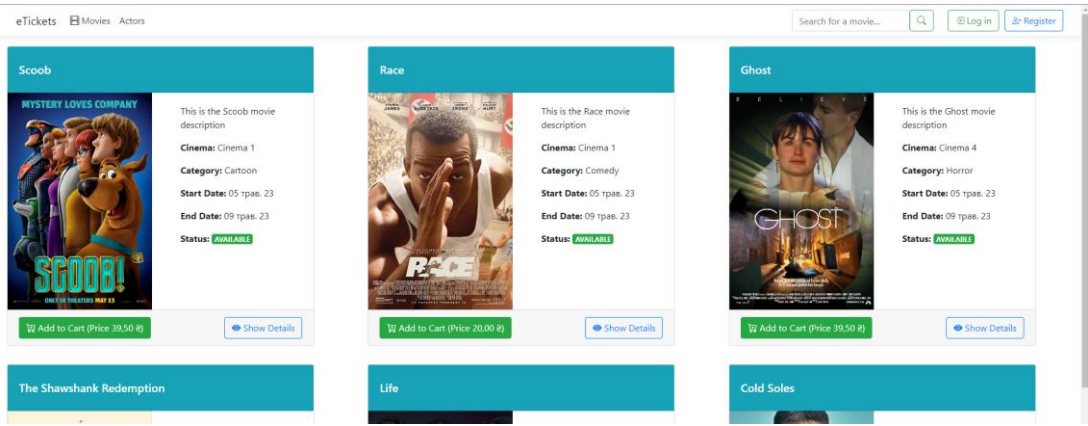


Рисунок 5.2 – Головна сторінка

Здійснити покупку, необхідно увійти в систему. Вікно авторизації у системі зображено на рисунку 5.3.

Log in to your account

Email address

admin_admin@gmail.com

Password

Cancel Log in

Рисунок 5.3 – Вікно авторизації у системі.

Після вдалої авторизації у системі у користувача з’явиться можливість обрати фільм для покупки. Після додавання фільму до кошика, користувача переводить на сторінку кошика. Нижче на рисунку 5.4 зображено кошик користувача.

You shopping cart summary			
Selected amount	Movie	Price	Subtotal
1	Race	20,00 ₪	20,00 ₪
		Total:	20,00 ₪

Shop now. Pay over time.

Add more items

Рисунок 5.4 – Кошик користувача

У кошику користувач повинен натиснути на кнопку оплати PayPal. Після успішного підтвердження транзакції, користувач отримає фільм.

5.3 Висновки за розділом

У цьому розділі описується процес взаємодії користувача з інтернет-магазином, включаючи способи реєстрації, авторизації, пошуку, замовлення та оплати товарів. Інтерфейс вебдодатку, розробленого для цього магазину, є стандартним і зрозумілим для звичайних користувачів, і не повинен викликати складнощів при використанні відповідно до основної мети.

ВИСНОВКИ

У процесі виконання дипломної кваліфікаційної роботи було розроблено інтернет-додаток для продажу квитків у кіно.

Після повного тестування та налагодження функціонування сайту з продажу квитків можна зробити наступні висновки:

- розроблений функціонал інтернет магазину повністю відповідає технічному завданню;
- інтернет магазин працює стабільно та без необхідності постійного обслуговування людиною;
- сайт має зручний та зрозумілий інтерфейс для користувачів.

Для розробки проєкту було обрано мову C#, технологію ASP.NET Core MVC. Для реалізації функціоналу роботи з базою даних була використана технологія EntityFramework та база даних SQLite.

Було проаналізовано середовище розробки, що відповідає зазначеним у постановці задачі критеріям, та необхідний набір інструментів для вирішення проєктної задачі.

Було обгрунтовано вибір мови програмування та середовища функціонування системи.

В результаті випробувань розроблений сайт з продажу квитків дозволяє відвідувачам кінотеатру зручно переглядати актуальні фільми та здійснювати покупки у мережі інтернет з будь якого зручного пристрою.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Пономаренко, Л. А. Електронна комерція [Текст] : підруч. для студ. екон. спец. / Л. А. Пономаренко, В. О. Філатов; Київський національний торговельно-економічний ун-т. - К. : КНТЕУ, 2002. – 442 с.
2. Мультиплекс / [Електрон. ресурс]. – Режим Доступу: <https://multiplex.ua/>.
3. ПланетаКіно / [Електрон. ресурс]. – Режим Доступу: <https://planetakino.ua/lvov2/movies/>.
4. Краус, К. М. Електронна комерція та інтернет-торгівля [Текст]: навч.-метод. посіб. / К. М. Краус, Н. М. Краус, О. В. Манжура. – Київ : Аграр Медіа Груп, 2021. – 454 с.
5. Прайс, М. С# 9 і .NET 5. Розробка і оптимізація. / М. Прайс. – O'Reilly Media, 2019. – 35 с.
6. Браун, Д. Опанування ASP.NET MVC: створюйте надійні вебдодатки за допомогою ASP.NET MVC Framework / Д. Браун. – Packt Publishing, 2018. – 84 с.
7. Макдональд, М. Pro ASP.NET Core MVC 2 / Метью Макдональд. – Apress, 2017. – 121 с.
8. Japiske, P. Pro C# 7: With .NET and .NET Core. / P. Japiske, A. Troelsen. – Apress, 2017. – 83 с.
9. Freeman, A. Pro ASP.NET MVC 3 Framework 3rd Edition / A. Freeman. – Apress, 2011. – 74 с.
10. Seemann, M. Dependency Injection .NET / M. Seemann. – Manning, 2011. – 45 с.
11. Schwichtenberg, H. Modern Data Access with Entity Framework Core: Database Programming Techniques for .NET, .NET Core, UWP, and Xamarin with C# / Schwichtenberg, H. – Apress, 2018. – 43 с.

12. Cwalina, K. Framework Design Guidelines: Conversations, Idioms and Patterns for Resuable .NET Libraries. / Krzysztof Cwalina, Brad Abrams. – Addison-Wesley Professional, 2009. – 28 с.
13. Lock, A. ASP.NET Core in Action: second edition (Annotated Edition) / A. Lock. – Manning, 2021. – 34 с.
14. Албахарі, Б. С# 7.0. Довідник: Повний опис. / Б. Албахарі, Дж. Албахарі. – O'Reilly Media, 2021. – 242 с.
15. Сміт, Д. Entity Framework Core in Action, Second Edition / Сміт, Д. – Manning, 2021. – 53 с.

ДОДАТОК А
Технічне завдання

A.1 Підстава для розробки

Підставою для розробки послужило завдання на дипломну роботу на тему "Розробка вебзастосунку для продажу квитків у кіно", по Національному університету «Запорізька політехніка» .

A.2 Призначення розробки

Мета роботи – розробка програмного продукту, який реалізує функції Інтернет магазину з індивідуальним наповненням та працює в мережі Інтернет, і створення індивідуального наповнення Вебсайту відповідно до певної предметної області.

A.3 Основні вимоги до розроблювальної програми

A.3.1 Вимоги до надійності

Програма повинна забезпечувати коректну обробку помилок у наступних ситуаціях:

- при введенні користувачем неприпустимих значень;
- при невірній роботі з файлами (спробі доступу до файлу неприпустимого формату, спробі запису на захищений від запису диск та ін.)

A.3.2 Умови експлуатації

Рекомендується не перезавантажувати комп'ютер або вимкнути його під час роботи програми. Для виходу із програми необхідно послідовно зупинити клієнтський та серверний сервіс. Під час експлуатації програмного продукту необхідно керуватися експлуатаційною документацією, яка

відповідає стандартам та містить інформацію, необхідну для освоєння та використання програмного продукту.

А.3.3 Вимоги до складу й параметрів технічних засобів

Необхідними вимогами до складу й параметрів технічних засобів, які повинні бути застосовані для роботи Інтернет-магазин, є:

- Оперативна пам'ять – не менш 2Гб;
- HDD (жорсткий диск)- не менш 40Гб;
- процесор – не менш 2 ГГц;
- мережний адаптер;
- операційна система – Windows 10, Windows 10 Server.

А.3.4 Вимоги до маркування й пакування

Програма може бути записана як на Flash-накопичувач, компакт-диску або на жорсткому диску комп'ютера. На пакуванні повинна бути назва програми – "Вебзастосунок для продажу квитків у кіно".

А.3.5 Вимоги до транспортування й зберігання

Вимоги до транспортування й зберігання по більшій частині аналогічні вимогам, які пропонуються до носія, на якому записана програма.

А.3.6 Вимоги до програмної документації

Програма повинна поставлятися з "Керівництвом оператора".

ДОДАТОК Б
Текст програми

Б.1 Startup конфігураційний файл з налагодженням усіх сервісів для проєкту

```

using eTickets.Data;
using eTickets.Data.Cart;
using eTickets.Data.Services;
using eTickets.Models;

namespace eTickets
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            //DbContext configuration
            services.AddDbContext<AppDbContext>(options =>
options.UseSqlite(Configuration.GetConnectionString("DefaultConnectionString")));

            //Services configuration
            services.AddScoped<IActorsService, ActorsService>();
            services.AddScoped<IProducersService, ProducersService>();
            services.AddScoped<ICinemasService, CinemasService>();
            services.AddScoped<IMoviesService, MoviesService>();
            services.AddScoped<IOrdersService, OrdersService>();

            services.AddSingleton<IHttpContextAccessor, HttpContextAccessor>();
            services.AddScoped(sc => ShoppingCart.GetShoppingCart(sc));

            //Authentication and authorization
            services.AddIdentity<ApplicationUser, IdentityRole>()
                .AddEntityFrameworkStores<AppDbContext>();

            services.AddMemoryCache();
            services.AddSession();
            services.AddAuthentication(options =>
            {
                options.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;
            });

            services.AddControllersWithViews();
        }
    }
}

```

```

    }

    // This method gets called by the runtime. Use this method to configure the HTTP request
    pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            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();

        //Authentication & Authorization
        app.UseAuthentication();
        app.UseAuthorization();

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllerRoute(
                name: "default",
                pattern: "{controller=Movies}/{action=Index}/{id?}");
        });

        //Seed database
        AppDbInitializer.Seed(app);
        AppDbInitializer.SeedUsersAndRolesAsync(app).Wait();
    }
}

```

Б.2 AccountController класс с функціями для авторизації та реєстрації

```

public class AccountController : Controller
{
    private readonly UserManager<ApplicationUser> _userManager;

```

```

private readonly SignInManager<ApplicationUser> _signInManager;
private readonly ApplicationDbContext _context;

public AccountController(UserManager<ApplicationUser> userManager,
SignInManager<ApplicationUser> signInManager, ApplicationDbContext context)
{
    _userManager = userManager;
    _signInManager = signInManager;
    _context = context;
}

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

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

[HttpPost]
public async Task<IActionResult> Login(LoginVM loginVM)
{
    if (!ModelState.IsValid) return View(loginVM);

    var user = await _userManager.FindByEmailAsync(loginVM.EmailAddress);
    var users = await _userManager.Users.ToListAsync();
    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
    };
    var newUserResponse = await _userManager.CreateAsync(newUser,
registerVM.Password);

    if (newUserResponse.Succeeded)
    {
        string role = registerVM.EmailAddress.Contains("admin") ? UserRoles.Admin :
UserRoles.User;
        await _userManager.AddToRoleAsync(newUser, role);
    }
    else
    {
        TempData["Error"] = string.Join("\n", newUserResponse.Errors);
        return View(registerVM);
    }

    return View("RegisterCompleted");
}

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

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

```

Б.3 ActorsController клас з функціями редагування акторів у базі даних

```
[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/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,Bio")] 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));
}
}

```

Б.4 CinemasController клас для редагування кінозалів у базі даних для адміністратора

```

[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/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,Bio")] 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));
    }
}

```

Б.5 MoviesController клас для взаємодії з кінофільмами для користувача та адміністратора

```

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> Filter(string searchString)
    {
        var movies = await _service.GetAllAsync();

        if (!string.IsNullOrEmpty(searchString))
        {

```

```

        //var filteredResult = allMovies.Where(n =>
n.Name.ToLower().Contains(searchString.ToLower()) ||
n.Description.ToLower().Contains(searchString.ToLower())).ToList();

```

```

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

```

```

        return View("Index", filteredResultNew);
    }

```

```

    return View("Index", movies);
}

```

```

//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.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));
}
}

```

Б.6 OrdersController клас для взаємодії з замовленнями

```

[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");
}
}

```

Б.7 ProducersController клас для взаємодії з продюсерами

```

[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,Bio")]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,Bio")] 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));
}
}

```

Б.8 Реалізація взаємодії з базою даних

```

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 ActorsService : EntityBaseRepository<Actor>, IActorsService
{
    public ActorsService(AppDbContext context) : base(context) { }
}

```

Б.8.1 IEntityBase.cs інтерфейс моделі у базі даних

```

public interface IEntityBase
{
    int Id { get; set; }
}

```

Б.8.2 IEntityBaseRepository.cs інтерфейс сервісу який взаємодіє з базою даних

```
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);
}
```

Б.8.3 EntityBaseRepository базовий клас який реалізує інтерфейс IRepository та функції взаємодії з базою даних через Entity Framework

```
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();
    }
}

```

Б.9 Реалізація інтерфейсу

Б.9.1 AccessDenied.cshtml компонент відображення помилки

```

<div class="row">
    <div class="col-md-6 offset-3 alert alert-danger text-center">
        <h2>You are not allowed to access this page</h2>
        <p>Please, click the button below to be redirected to the home page.</p>
        <hr />
        <p>
            <a asp-controller="Movies" asp-action="Index" class="btn btn-outline-success"
>Home Page</a>
        </p>
    </div>
</div>

```

Б.9.2 Login.cshtml компонент відображення авторизації

```

@using eTickets.Data.ViewModels;
@model LoginVM;

@{
    ViewData["Title"] = "Log in to your account";
}

<div class="row">
    <div class="col-md-6 offset-3">
        <p>
            <h4>Log in to your account</h4>
        </p>

        @if(TempData["Error"] != null)
        {
            <div class="col-md-12 alert alert-danger">
                <span><b>Sorry!</b> – @TempData["Error"] </span>
            </div>
        }

        <div class="row">
            <div class="col-md-8 offset-2">
                <form asp-action="Login">
                    <div asp-validation-summary="ModelOnly" class="text-danger"></div>

                    <div class="form-group">
                        <label asp-for="EmailAddress" class="control-label"></label>
                        <input asp-for="EmailAddress" class="form-control" />
                        <span asp-validation-for="EmailAddress" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Password" class="control-label"></label>
                        <input asp-for="Password" class="form-control" />
                        <span asp-validation-for="Password" class="text-danger"></span>
                    </div>

                    <div class="form-group">
                        <input class="btn btn-outline-success float-right" type="submit" value="Log in"
/>
                        <a class="btn btn-outline-secondary" asp-controller="Movies" asp-
action="Index">Cancel</a>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>

```

Б.9.3 Register.cshtml компонент відображення авторизації

```

@using eTickets.Data.ViewModels;
@model RegisterVM;

@{
    ViewData["Title"] = "Sign up for a new account";
}

<div class="row">
    <div class="col-md-6 offset-3">
        <p>
            <h4>Sign up for a new account</h4>
        </p>

        @if(TempData["Error"] != null)
        {
            <div class="col-md-12 alert alert-danger">
                <span><b>Sorry!</b> – @TempData["Error"] </span>
            </div>
        }

        <div class="row">
            <div class="col-md-8 offset-2">
                <form asp-action="Register">
                    <div asp-validation-summary="ModelOnly" class="text-danger"></div>

                    <div class="form-group">
                        <label asp-for="FullName" class="control-label"></label>
                        <input asp-for="FullName" class="form-control" />
                        <span asp-validation-for="FullName" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="EmailAddress" class="control-label"></label>
                        <input asp-for="EmailAddress" class="form-control" />
                        <span asp-validation-for="EmailAddress" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Password" class="control-label"></label>
                        <input asp-for="Password" class="form-control" />
                        <span asp-validation-for="Password" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="ConfirmPassword" class="control-label"></label>
                        <input asp-for="ConfirmPassword" class="form-control" />
                        <span asp-validation-for="ConfirmPassword" class="text-danger"></span>
                    </div>

                    <div class="form-group">
                        <input class="btn btn-outline-success float-right" type="submit" value="Sign up"

```

```

        <a class="btn btn-outline-secondary" asp-controller="Movies" asp-
action="Index">Cancel</a>
    </div>
</form>
</div>
</div>
</div>
</div>

```

Б.9.4 RegisterCompleted.cshtml компонент відображення завершення реєстрації

```

<div class="row">
    <div class="col-md-6 offset-3 alert alert-success text-center">
        <h2>Your account was created successfully</h2>
        <p>You can now log in to your account using your credentials.</p>
        <hr />
        <p>Thank you!</p>
    </div>
</div>

```

Б.9.5 Users.cshtml компонент відображення користувачів для адміністратора

```
@model IEnumerable<ApplicationUser>
```

```
@{
    ViewData["Title"] = "List of all users";
}
```

```

<div class="row">
    <div class="col-md-6 offset-3">
        <p>
            <h4>List of all users</h4>
        </p>

        <table class="table">
            <thead>
                <tr class="text-center">
                    <th>@Html.DisplayNameFor(model => model.FullName)</th>
                    <th>@Html.DisplayNameFor(model => model.UserName)</th>
                    <th>@Html.DisplayNameFor(model => model.Email)</th>
                </tr>
            </thead>
            <tbody>

```

```

@foreach (var item in Model)
{
|  |  |  |
| --- | --- | --- |
| @Html.DisplayFor(modelItem => item.FullName) | @Html.DisplayFor(modelItem => item.UserName) | @Html.DisplayFor(modelItem => item.Email) |

}
</tbody>

</table>

</div>
</div>

```

Б.9.6 Actors/Create.cshtml компонент додавання акторів

@model Actor

```

@{
    ViewData["Title"] = "Add a new actor";
}

```

```

<div class="row text">
    <div class="col-md-8 offset-2">
        <p>
            <h1>Add a New Actor</h1>
        </p>
        <div class="row">
            <div class="col-md-8 offset-2">
                <form asp-action="Create">
                    <div asp-validation-summary="ModelOnly" class="text-danger"></div>

                    <div class="form-group text-center">
                        <img id="ProfilePicturePreview" class="border-info rounded-circle"
style="max-width: 150px" />
                    </div>
                    <div class="form-group">
                        <label asp-for="ProfilePictureURL" class="control-label"></label>
                        <input asp-for="ProfilePictureURL" class="form-control" />
                        <span asp-validation-for="ProfilePictureURL" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="FullName" class="control-label"></label>

```



```

<div class="form-group text-center">
  <img class="border-info rounded-circle" style="max-width: 150px"
id="ProfilePicturePreview" />
</div>
<div class="form-group">
  <label asp-for="ProfilePictureURL" class="control-label"></label>
  <input asp-for="ProfilePictureURL" class="form-control" readonly />
  <span asp-validation-for="ProfilePictureURL" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="FullName" class="control-label"></label>
  <input asp-for="FullName" class="form-control" readonly />
  <span asp-validation-for="FullName" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="Bio" class="control-label"></label>
  <input asp-for="Bio" class="form-control" readonly />
  <span asp-validation-for="Bio" class="text-danger"></span>
</div>

<div class="form-group">
  <form asp-action="Delete">
    <input type="hidden" asp-for="Id" />
    <input type="submit" value="Confirm" class="btn btn-danger float-right" />
  </form>
  <a class="btn btn-outline-secondary" asp-action="Index">Show All</a>
</div>
</div>
</div>
</div>
</div>

@section Scripts{
  <script>
    $(document).ready(function () {
      var output = document.getElementById('ProfilePicturePreview');
      output.src = $("#ProfilePictureURL").val();
    })
  </script>
}

```

Б.9.8 Actors/Details.cshtml компонент подробиць акторів

@model Actor

```

@{
  ViewData["Title"] = "Details of the actor";
}

```

```

<div class="row text">
  <div class="col-md-8 offset-2">

```

```

<p>
  <h1>Details for @Model.FullName</h1>
</p>
<div class="row">
  <div class="col-md-8 offset-2">
    <div class="form-group text-center">
      <img class="border-info rounded-circle" style="max-width: 150px"
id="ProfilePicturePreview" />
    </div>
    <div class="form-group">
      <label asp-for="ProfilePictureURL" class="control-label"></label>
      <input asp-for="ProfilePictureURL" class="form-control" readonly />
      <span asp-validation-for="ProfilePictureURL" class="text-danger"></span>
    </div>
    <div class="form-group">
      <label asp-for="FullName" class="control-label"></label>
      <input asp-for="FullName" class="form-control" readonly />
      <span asp-validation-for="FullName" class="text-danger"></span>
    </div>
    <div class="form-group">
      <label asp-for="Bio" class="control-label"></label>
      <input asp-for="Bio" class="form-control" readonly />
      <span asp-validation-for="Bio" class="text-danger"></span>
    </div>

    <div class="form-group">
      @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
      {
        <a class="btn btn-outline-success float-right" asp-action="Edit" asp-route-
id="@Model.Id">Edit</a>
      }
      <a class="btn btn-outline-secondary" asp-action="Index">Show All</a>
    </div>
  </div>
</div>
</div>
</div>
</div>

@section Scripts{
  <script>
    $(document).ready(function () {
      var output = document.getElementById('ProfilePicturePreview');
      output.src = $("#ProfilePictureURL").val();
    })
  </script>
}

```

Б.9.9 Actors/Edit.cshtml компонент редагування акторів

```

@model Actor

@{
    ViewData["Title"] = "Update actor";
}

<div class="row text">
    <div class="col-md-8 offset-2">
        <p>
            <h1>Update Actor</h1>
        </p>
        <div class="row">
            <div class="col-md-8 offset-2">
                <form asp-action="Edit">
                    <div asp-validation-summary="ModelOnly" class="text-danger"></div>

                    <div class="form-group text-center">
                        <img id="ProfilePicturePreview" class="border-info rounded-circle"
style="max-width: 150px" />
                    </div>
                    <div class="form-group">
                        <label asp-for="ProfilePictureURL" class="control-label"></label>
                        <input asp-for="ProfilePictureURL" class="form-control" />
                        <span asp-validation-for="ProfilePictureURL" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="FullName" class="control-label"></label>
                        <input asp-for="FullName" class="form-control" />
                        <span asp-validation-for="FullName" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Bio" class="control-label"></label>
                        <input asp-for="Bio" class="form-control" />
                        <span asp-validation-for="Bio" class="text-danger"></span>
                    </div>

                    <div class="form-group">
                        <input type="submit" value="Update" class="btn btn-outline-success float-right"

/a class="btn btn-outline-secondary" asp-action="Index">Show All</a>
                </div>
            </div>
        </div>
    </div>
</div>

@section Scripts{
    <script>

```

```

$(document).ready(function () {
    var output = document.getElementById('ProfilePicturePreview');
    output.src = $('#ProfilePictureURL').val();
})

$('#ProfilePictureURL').on("change", function () {
    var output = document.getElementById('ProfilePicturePreview');
    output.src = $(this).val();
})
</script>
}

```

Б.9.10 Actors/Index.cshtml компонент перегляду акторів

```

@model IEnumerable<Actor>

@{
    ViewData["Title"] = "List of Actors";
}

<div class="row">
    <div class="col-md-8 offset-md-2">

        <table class="table">
            <thead>
                <tr class="text-center">
                    <th>@Html.DisplayNameFor(model => model.ProfilePictureURL)</th>
                    <th>@Html.DisplayNameFor(model => model.FullName)</th>
                    <th>@Html.DisplayNameFor(model => model.Bio)</th>
                    @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
                    {
                        <th>Actions</th>
                    }
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model)
                {
                    <tr>
                        <td class="align-middle">
                            
                        </td>
                        <td class="align-middle">
                            @Html.DisplayFor(modelItem => item.FullName)
                        </td>
                        <td class="align-middle">
                            @Html.DisplayFor(modelItem => item.Bio)
                        </td>

```

```

        @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
        {
            <td class="align-middle">
                <a class="btn btn-outline-primary" asp-action="Edit" asp-route-
id="@item.Id"><i class="bi bi-pencil-square"></i>Edit </a> |
                <a class="btn btn-outline-info" asp-action="Details" asp-route-
id="@item.Id"><i class="bi bi-eye"></i>Details </a> |
                <a class="btn btn-danger text-white" asp-action="Delete" asp-route-
id="@item.Id"><i class="bi bi-trash"></i>Delete </a>
            </td>
        }
    </tr>
}
</tbody>

</table>

</div>
</div>

@await Html.PartialAsync("_CreateItem", "Actors")

```

Б.9.11 Cinemas/Create.cshtml компонент додавання кінозалів

```

@model Cinema

@{
    ViewData["Title"] = "Add a new cinema";
}

<div class="row text">
    <div class="col-md-8 offset-2">
        <p>
            <h1>Add a New Cinema</h1>
        </p>
        <div class="row">
            <div class="col-md-8 offset-2">
                <form asp-action="Create">
                    <div asp-validation-summary="ModelOnly" class="text-danger"></div>

                    <div class="form-group text-center">
                        <img id="CinemaLogoPreview" class="border-info rounded-circle" style="max-
width: 150px" />
                    </div>
                    <div class="form-group">
                        <label asp-for="Logo" class="control-label"></label>
                        <input asp-for="Logo" class="form-control" />
                        <span asp-validation-for="Logo" class="text-danger"></span>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>

```

```

        <div class="form-group">
            <label asp-for="Name" class="control-label"></label>
            <input asp-for="Name" class="form-control" />
            <span asp-validation-for="Name" class="text-danger"></span>
        </div>
        <div class="form-group">
            <label asp-for="Description" class="control-label"></label>
            <input asp-for="Description" class="form-control" />
            <span asp-validation-for="Description" class="text-danger"></span>
        </div>

        <div class="form-group">
            <input type="submit" value="Create" class="btn btn-outline-success float-right"
/>

            <a class="btn btn-outline-secondary" asp-action="Index">Show All</a>
        </div>
    </form>
</div>
</div>
</div>
</div>
</div>

@section Scripts{
    <script>

        $(document).ready(function () {
            var output = document.getElementById('CinemaLogoPreview');
            output.src = $("#Logo").val();
        })

        $("#Logo").on("change", function () {
            var output = document.getElementById('CinemaLogoPreview');
            output.src = $(this).val();
        })
    </script>
}

```

Б.9.12 Cinemas /Delete.cshtml компонент виделення кінозалів

```

@model Cinema

@{
    ViewData["Title"] = "Delete cinema";
}

<div class="row text">
    <div class="col-md-8 offset-2">
        <p>
            <h1>Delete confirmation for @Model.Name</h1>
        </p>
    </div>

```

```

<div class="row">
  <div class="col-md-8 offset-2">
    <div class="form-group text-center">
      <img class="border-info rounded-circle" style="max-width: 150px"
id="LogoPreview" />
    </div>
    <div class="form-group">
      <label asp-for="Logo" class="control-label"></label>
      <input asp-for="Logo" class="form-control" readonly />
      <span asp-validation-for="Logo" class="text-danger"></span>
    </div>
    <div class="form-group">
      <label asp-for="Name" class="control-label"></label>
      <input asp-for="Name" class="form-control" readonly />
      <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="form-group">
      <label asp-for="Description" class="control-label"></label>
      <input asp-for="Description" class="form-control" readonly />
      <span asp-validation-for="Description" class="text-danger"></span>
    </div>

    <div class="form-group">
      <form asp-action="Delete">
        <input type="hidden" asp-for="Id" />
        <input type="submit" value="Confirm" class="btn btn-danger float-right" />
      </form>
      <a class="btn btn-outline-secondary" asp-action="Index">Show All</a>
    </div>
  </div>
</div>
</div>
</div>
</div>

@section Scripts{
  <script>
    $(document).ready(function () {
      var output = document.getElementById('LogoPreview');
      output.src = $("#Logo").val();
    })
  </script>
}

```

Б.9.13 Cinemas/Details.cshtml компонент подробиць кінозалів

@model Cinema

```

@{
  ViewData["Title"] = "Details of the cinema";
}

```

```

<div class="row text">
  <div class="col-md-8 offset-2">
    <p>
      <h1>Details for @Model.Name</h1>
    </p>
    <div class="row">
      <div class="col-md-8 offset-2">
        <div class="form-group text-center">
          <img class="border-info rounded-circle" style="max-width: 150px"
id="LogoPreview" />
        </div>
        <div class="form-group">
          <label asp-for="Logo" class="control-label"></label>
          <input asp-for="Logo" class="form-control" readonly />
          <span asp-validation-for="Logo" class="text-danger"></span>
        </div>
        <div class="form-group">
          <label asp-for="Name" class="control-label"></label>
          <input asp-for="Name" class="form-control" readonly />
          <span asp-validation-for="Name" class="text-danger"></span>
        </div>
        <div class="form-group">
          <label asp-for="Description" class="control-label"></label>
          <input asp-for="Description" class="form-control" readonly />
          <span asp-validation-for="Description" class="text-danger"></span>
        </div>

        <div class="form-group">
          @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
          {
            <a class="btn btn-outline-success float-right" asp-action="Edit" asp-route-
id="@Model.Id">Edit</a>
          }
          <a class="btn btn-outline-secondary" asp-action="Index">Show All</a>
        </div>
      </div>
    </div>
  </div>
</div>

@section Scripts{
  <script>
    $(document).ready(function () {
      var output = document.getElementById('LogoPreview');
      output.src = $("#Logo").val();
    })
  </script>
}

```

Б.9.14 Cinemas /Edit.cshtml компонент редагування кінозалів

```

@model Cinema

@{
    ViewData["Title"] = "Update cinema";
}

<div class="row text">
    <div class="col-md-8 offset-2">
        <p>
            <h1>Update cinema</h1>
        </p>
        <div class="row">
            <div class="col-md-8 offset-2">
                <form asp-action="Edit">
                    <div asp-validation-summary="ModelOnly" class="text-danger"></div>

                    <div class="form-group text-center">
                        <img id="CinemaLogoPreview" class="border-info rounded-circle" style="max-
width: 150px" />
                    </div>
                    <div class="form-group">
                        <label asp-for="Logo" class="control-label"></label>
                        <input asp-for="Logo" class="form-control" />
                        <span asp-validation-for="Logo" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Name" class="control-label"></label>
                        <input asp-for="Name" class="form-control" />
                        <span asp-validation-for="Name" class="text-danger"></span>
                    </div>
                    <div class="form-group">
                        <label asp-for="Description" class="control-label"></label>
                        <input asp-for="Description" class="form-control" />
                        <span asp-validation-for="Description" class="text-danger"></span>
                    </div>

                    <div class="form-group">
                        <input type="submit" value="Update" class="btn btn-outline-success float-right"
/>

                        <a class="btn btn-outline-secondary" asp-action="Index">Show All</a>
                    </div>
                </form>
            </div>
        </div>
    </div>
</div>

@section Scripts{
    <script>

```

```

$(document).ready(function () {
    var output = document.getElementById('CinemaLogoPreview');
    output.src = $("#Logo").val();
})

$("#Logo").on("change", function () {
    var output = document.getElementById('CinemaLogoPreview');
    output.src = $(this).val();
})
</script>
}

```

Cinemas /Index.cshtml – компонент перегляду кінозалів

```
@model IEnumerable<Cinema>
```

```
@{
    ViewData["Title"] = "List of Cinemas";
}
```

```

<div class="row">
    <div class="col-md-8 offset-md-2">

        <table class="table">
            <thead>
                <tr class="text-center">
                    <th>@Html.DisplayNameFor(model => model.Logo)</th>
                    <th>@Html.DisplayNameFor(model => model.Name)</th>
                    <th>@Html.DisplayNameFor(model => model.Description)</th>
                    @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
                    {
                        <th>Actions</th>
                    }
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Model)
                {
                    <tr>
                        <td class="align-middle">
                            
                        </td>
                        <td class="align-middle">
                            @Html.DisplayFor(modelItem => item.Name)
                        </td>
                        <td class="align-middle">
                            @Html.DisplayFor(modelItem => item.Description)
                        </td>
                        @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
                        {

```

```

        <td class="align-middle">
            <a class="btn btn-outline-primary" asp-action="Edit" asp-route-
id="@item.Id"><i class="bi bi-pencil-square"></i>Edit </a> |
            <a class="btn btn-outline-info" asp-action="Details" asp-route-
id="@item.Id"><i class="bi bi-eye"></i>Details </a> |
            <a class="btn btn-danger text-white" asp-action="Delete" asp-route-
id="@item.Id"><i class="bi bi-trash"></i>Delete </a>
        </td>
    }
</tr>
}
</tbody>

</table>

</div>
</div>

```

```
@await Html.PartialAsync("_CreateItem", "Cinemas")
```

Б.9.15 Movies/Create.cshtml компонент додавання фільмів

```

@model NewMovieVM
@using eTickets.Data;

@{
    ViewData["Title"] = "Create a new movie";
}

<div class="row">
    <div class="col-md-8 offset-2">
        <p>
            <h4>Create a new movie</h4>
        </p>

        <div class="row">
            <div class="col-md-8 offset-2">
                <form asp-action="Create">
                    <div asp-validation-summary="ModelOnly" class="text-danger"></div>

                    <div class="row">
                        <div class="col-md-6">
                            <div class="form-group">
                                <label asp-for="Name" class="control-label"></label>
                                <input asp-for="Name" class="form-control" />
                                <span asp-validation-for="Name" class="text-danger"></span>
                            </div>
                        <div class="form-group">
                            <label asp-for="StartDate" class="control-label"></label>

```

```

        <input asp-for="StartDate" type="date" class="form-control" />
        <span asp-validation-for="StartDate" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="EndDate" class="control-label"></label>
        <input asp-for="EndDate" type="date" class="form-control" />
        <span asp-validation-for="EndDate" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Price" class="control-label"></label>
        <input asp-for="Price" type="number" class="form-control" />
        <span asp-validation-for="Price" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="ImageUrl" class="control-label"></label>
        <input asp-for="ImageUrl" class="form-control" />
        <span asp-validation-for="ImageUrl" class="text-danger"></span>
    </div>
</div>
<div class="col-md-6">
    <div class="form-group">
        <label asp-for="CinemaId" class="control-label"></label>
        <select asp-for="CinemaId" class="form-control" asp-
items="ViewBag.Cinemas"></select>
        <span asp-validation-for="CinemaId" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="MovieCategory" class="control-label"></label>
        <select asp-for="MovieCategory" class="form-control" asp-
items="Html.GetEnumSelectList<MovieCategory>()"></select>
        <span asp-validation-for="MovieCategory" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="ProducerId" class="control-label"></label>
        <select asp-for="ProducerId" class="form-control" asp-
items="ViewBag.Producers"></select>
        <span asp-validation-for="ProducerId" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="ActorIds" class="control-label"></label>
        <select asp-for="ActorIds" class="form-control" asp-
items="ViewBag.Actors" multiple></select>
        <span asp-validation-for="ActorIds" class="text-danger"></span>
    </div>
</div>
<div class="col-md-12">
    <div class="form-group">
        <label asp-for="Description" class="control-label"></label>
        <textarea asp-for="Description" class="form-control"></textarea>
        <span asp-validation-for="Description" class="text-danger"></span>
    </div>
</div>

```

```

        </div>

        <div class="col-md-4 offset-4 text-center">
            <img width="100%" id="ImageUrlPreview" />
        </div>

        <div class="form-group">
            <input type="submit" value="Create" class="btn btn-outline-success float-right"
/>

            <a class="btn btn-outline-secondary" asp-action="Index">Show all</a>
        </div>

    </form>

</div>
</div>

</div>
</div>

@section Scripts{
    <script>
        $("#ImageUrl").on("change", function () {
            var output = document.getElementById("ImageUrlPreview");
            output.src = $(this).val();
        })
    </script>
}

```

Б.9.16 Movies /Details.cshtml компонент подробиць фільмів

```

@model Movie

@{
    ViewData["Title"] = "Movie details";
}

<div class="row">
    <div class="col-md-8 offset-2">
        <div class="card mb-12">
            <div class="row g-0">
                <div class="col-md-12">
                    <div class="card-header text-white bg-info">
                        <p class="card-text">
                            <h5 class="card-title">
                                @Model.Name
                            </h5>
                        </p>
                    </div>
                </div>
            </div>
        </div>
    </div>

```

```

<div class="col-md-4">
  
</div>
<div class="col-md-8">
  <div class="card-body">
    <p class="card-text">@Model.Description</p>
    <p class="card-text">
      <b>Cinema: </b>
      <a asp-controller="Cinemas" asp-action="Details" asp-route-
id="@Model.CinemaId">
        @Model.Cinema.Name
      </a>
    </p>
    <p class="card-text">
      <b>Producer: </b>
      <a asp-controller="Producers" asp-action="Details" asp-route-
id="@Model.ProducerId">
        @Model.Producer.FullName
      </a>
    </p>
    <p class="card-text"><b>Category: </b>@Model.MovieCategory</p>
    <p class="card-text"><b>Start Date: </b>@Model.StartDate.ToString("dd MMM
yy")</p>
    <p class="card-text"><b>End Date: </b>@Model.EndDate.ToString("dd MMM
yy")</p>
    <p class="card-text ">
      <b>Status: </b>
      @if (DateTime.Now >= Model.StartDate && DateTime.Now <=
Model.EndDate)
      {
        <span class="badge bg-success text-white">AVAILABLE</span>
      }
      else if (DateTime.Now > Model.EndDate)
      {
        <span class="badge bg-danger text-white">EXPIRED</span>
      }
      else
      {
        <span class="badge bg-primary text-white">UPCOMING</span>
      }
    </p>
    <p class="card-text">
      <b>Actors: </b>
      <hr />
      <div class="row">
        @foreach (var actor in Model.Actors_Movies)
        {
          <div class="column text-center">
            
            <br />

```

```

        <a asp-controller="Actors" asp-action="Details" asp-route-
id="@actor.ActorId">@actor.Actor.FullName</a>
    </div>
}
</div>
</div>
<div class="col-md-12">
    <div class="card-footer">
        <p class="card-text">
            <a class="btn btn-outline-primary float-right" asp-action="Index">
                Back to List
            </a>
            <a class="btn btn-success text-white @((DateTime.Now > Model.EndDate) ?
"disabled" : "")"
                asp-controller="Orders"
                asp-action="AddItemToShoppingCart" asp-route-id="@Model.Id"><i
class="bi bi-cart-plus"></i> Add to Cart (Price @Model.Price.ToString("c"))</a>
        </p>
    </div>
</div>
</div>
</div>
</div>
</div>

```

Б.9.17 Movies /Edit.cshtml компонент редагування фільмів

```

@model NewMovieVM
@using eTickets.Data;

@{
    ViewData["Title"] = "Update movie";
}

<div class="row">
    <div class="col-md-8 offset-2">
        <p>
            <h4>Update movie</h4>
        </p>

        <div class="row">
            <div class="col-md-8 offset-2">
                <form asp-action="Edit">
                    <div asp-validation-summary="ModelOnly" class="text-danger"></div>

                    <div class="row">
                        <div class="col-md-6">

```

```

<div class="form-group">
  <label asp-for="Name" class="control-label"></label>
  <input asp-for="Name" class="form-control" />
  <span asp-validation-for="Name" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="StartDate" class="control-label"></label>
  <input asp-for="StartDate" type="date" class="form-control" />
  <span asp-validation-for="StartDate" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="EndDate" class="control-label"></label>
  <input asp-for="EndDate" type="date" class="form-control" />
  <span asp-validation-for="EndDate" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="Price" class="control-label"></label>
  <input asp-for="Price" type="number" class="form-control" />
  <span asp-validation-for="Price" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="ImageUrl" class="control-label"></label>
  <input asp-for="ImageUrl" class="form-control" />
  <span asp-validation-for="ImageUrl" class="text-danger"></span>
</div>
</div>
<div class="col-md-6">
  <div class="form-group">
    <label asp-for="CinemaId" class="control-label"></label>
    <select asp-for="CinemaId" class="form-control" asp-
items="ViewBag.Cinemas"></select>
    <span asp-validation-for="CinemaId" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="MovieCategory" class="control-label"></label>
    <select asp-for="MovieCategory" class="form-control" asp-
items="Html.GetEnumSelectList<MovieCategory>()"></select>
    <span asp-validation-for="MovieCategory" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="ProducerId" class="control-label"></label>
    <select asp-for="ProducerId" class="form-control" asp-
items="ViewBag.Producers"></select>
    <span asp-validation-for="ProducerId" class="text-danger"></span>
  </div>
  <div class="form-group">
    <label asp-for="ActorIds" class="control-label"></label>
    <select asp-for="ActorIds" class="form-control" asp-
items="ViewBag.actors" multiple></select>
    <span asp-validation-for="ActorIds" class="text-danger"></span>
  </div>
</div>

```

```

        <div class="col-md-12">
            <div class="form-group">
                <label asp-for="Description" class="control-label"></label>
                <textarea asp-for="Description" class="form-control"></textarea>
                <span asp-validation-for="Description" class="text-danger"></span>
            </div>
        </div>
    </div>

    <div class="col-md-4 offset-4 text-center">
        <img width="100%" id="ImageUrlPreview" />
    </div>

    <div class="form-group">
        <input type="submit" value="Update" class="btn btn-outline-success float-right"
/>

        <a class="btn btn-outline-secondary" asp-action="Index">Show all</a>
    </div>

</form>

</div>
</div>

</div>
</div>

@section Scripts{
    <script>
        $(document).ready(function () {
            var output = document.getElementById("ImageUrlPreview");
            output.src = $("#ImageUrl").val();
        });

        $("#ImageUrl").on("change", function () {
            var output = document.getElementById("ImageUrlPreview");
            output.src = $(this).val();
        })
    </script>
}

```

Б.9.18 Movies/Index.cshtml компонент перегляду фільмів

```

@model IEnumerable<Movie>

@{
    ViewData["Title"] = "List of Movies";
}

<div class="row">
    @foreach (var item in Model)
    {

```

```

<div class="col-md-4 col-xs-6 border-primary mb-3">
  <div class="card mb-3" style="max-width: 540px;">
    <div class="row g-0">
      <div class="col-md-12">
        <div class="card-header text-white bg-info">
          <p class="card-text">
            <h5 class="card-title">
              @item.Name
              @if (User.Identity.IsAuthenticated && User.IsInRole("Admin"))
              {
                <a class="text-white float-right" asp-action="Edit" asp-route-
id="@item.Id"><i class="bi bi-pencil-square"></i></a>
              }
            </h5>
          </p>
        </div>
      </div>
      <div class="col-md-6">
        
      </div>
      <div class="col-md-6">
        <div class="card-body">
          <p class="card-text">@item.Description</p>
          <p class="card-text"><b>Cinema: </b>@item.Cinema.Name</p>
          <p class="card-text"><b>Category: </b>@item.MovieCategory</p>
          <p class="card-text"><b>Start Date: </b>@item.StartDate.ToString("dd
MMM yy")</p>
          <p class="card-text"><b>End Date: </b>@item.EndDate.ToString("dd MMM
yy")</p>
          <p class="card-text ">
            <b>Status: </b>
            @if (DateTime.Now >= item.StartDate && DateTime.Now <=
item.EndDate)
            {
              <span class="badge bg-success text-white">AVAILABLE</span>
            }
            else if (DateTime.Now > item.EndDate)
            {
              <span class="badge bg-danger text-white">EXPIRED</span>
            }
            else
            {
              <span class="badge bg-primary text-white">UPCOMING</span>
            }
          </p>
        </div>
      </div>
    </div>
  </div>
  <div class="col-md-12">
    <div class="card-footer ">
      <p class="card-text">

```

```

        <a class="btn btn-outline-primary float-right" asp-action="Details" asp-
route-id="@item.Id">
            <i class="bi bi-eye-fill"></i> Show Details
        </a>
        <a class="btn btn-success text-white @((DateTime.Now > item.EndDate) ?
"disabled" : "")"
            asp-controller="Orders"
            asp-action="AddItemToShoppingCart"
            asp-route-id="@item.Id">
            <i class="bi bi-cart-plus"></i> Add to Cart (Price
@item.Price.ToString("c"))</a>
        </p>
    </div>
</div>
</div>
</div>
</div>
}
</div>

@await Html.PartialAsync("_CreateItem", "Movies")

```

Б.9.19 _Layout.cshtml – головна компонента для відображення тіла вебзастосунку

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] – demTickets</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
icons@1.4.1/font/bootstrap-icons.css">
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-
white border-bottom box-shadow mb-3">
            <div class="container-fluid">
                <a class="navbar-brand" asp-controller="Movies" asp-
action="Index">eTickets</a>
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target=".navbar-collapse" aria-controls="navbarSupportedContent"

```

```

        aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="navbar-collapse collapse d-sm-inline-flex justify-content-
between">
        <ul class="navbar-nav flex-grow-1">
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area="" asp-controller="Movies"
asp-action="Index"><i class="bi bi-film"></i> Movies</a>
            </li>

            <li class="nav-item">
                <a class="nav-link text-dark" asp-area="" asp-controller="Actors"
asp-action="Index"><i class="bi bi-user"></i> Actors</a>
            </li>

            @if(User.Identity.IsAuthenticated && User.IsInRole("Admin"))
            {
                <li class="nav-item dropdown">
                    <a class="nav-link dropdown-toggle" href="#"
id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true"
aria-expanded="false">
                        <i class="bi bi-gear"></i> Management
                    </a>

                    <div class="dropdown-menu" aria-
labelledby="navbarDropdown">
                        <a class="dropdown-item" asp-controller="Cinemas" asp-
action="Index"><i class="bi bi-camera-reels"></i> Cinemas</a>
                        <a class="dropdown-item" asp-controller="Producers" asp-
action="Index"><i class="bi bi-headset"></i> Producers</a>
                        <a class="dropdown-item" asp-controller="Actors" asp-
action="Index"><i class="bi bi-person-square"></i> Actors</a>
                    </div>
                </li>
            }
        </ul>

        <form class="form-inline my-2 my-lg-0" asp-controller="Movies" asp-
action="Filter" style="padding-right: 25px;">
            <input name="searchString" type="text" class="form-control"
placeholder="Search for a movie..." aria-label="Recipient's username" aria-
describedby="button-addon2">
            <button class="btn btn-outline-success my-2 my-sm-0"
type="submit"><i class="bi bi-search"></i></button>

```

```

</form>

<ul class="navbar-nav">
    @await Component.InvokeAsync("ShoppingCartSummary")
</ul>

<partial name="_Identity.cshtml" />
</div>
</div>
</nav>
</header>
<div class="container-fluid">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>

</div>

<footer class="border-top footer text-muted">
    <div class="container">
        &copy; 2023 – demTickets
    </div>
</footer>
<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>

    @await RenderSectionAsync("Scripts", required: false)
</body>
</html>

```

Б.9.20 ShoppingCart.cshtml компонент для відображення та оброблення бронювань

```

@using eTickets.Data.ViewModels;
@model ShoppingCartVM

@{
    ViewData["Title"] = "Shopping cart summary";
}

<div class="row">
    <div class="col-md-8 offset-2">

```

```

<div class="text-center">
  <h2>You shopping cart summary</h2>
</div>

<table class="table">
  <thead>
    <tr>
      <th>Selected amount</th>
      <th>Movie</th>
      <th>Price</th>
      <th>Subtotal</th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    @foreach (var item in Model.ShoppingCart.ShoppingCartItems)
    {
      <tr>
        <td class="align-middle">@item.Amount</td>
        <td class="align-middle">@item.Movie.Name</td>
        <td class="align-middle">@item.Movie.Price.ToString("c")</td>
        <td class="align-
middle">@((item.Amount*item.Movie.Price).ToString("c"))</td>
        <td class="align-middle">
          <a class="text-danger" asp-controller="Orders" asp-
action="RemoveItemFromShoppingCart"
          asp-route-id="@item.Movie.Id">
            <i class="bi bi-cart-dash"></i>
          </a>
          |
          <a class="text-success" asp-controller="Orders" asp-
action="AddItemToShoppingCart"
          asp-route-id="@item.Movie.Id">
            <i class="bi bi-cart-plus"></i>
          </a>
        </td>
      </tr>
    }
  </tbody>
  <tfoot>
    <tr>
      <td colspan="2"></td>
      <td><b>Total:</b></td>
      <td>
        @Model.ShoppingCartTotal.ToString("c")
      </td>
      <td></td>
    </tr>
  </tfoot>
</table>

<div class="text-right">

```

```

<div class="btn-group">
  @*<a class="btn btn-outline-primary" asp-action="CompleteOrder">
    Complete order
  </a>*@
<div id="paypal-btn"></div>
<span style="padding:5px"></span>
<a class="btn btn-outline-success" asp-controller="Movies" asp-action="Index">
  Add more items
</a>
</div>
</div>
</div>

@section Scripts{

<script src="https://www.paypalobjects.com/api/checkout.js"></script>

<script>

  let _total = @Html.Raw(Model.ShoppingCartTotal);

  paypal.Button.render({
    //Configure environment
    env: 'sandbox',
    client: {
      sandbox:
'AcKRtgL4i68EZ2ZYuYUx3ilihAg2YhL5dtF_m19A8MjBWVOn6ewNS4M_uV8gA3xi98zTc
Uq38gUHbVFx'
    },

    //Customize button
    locale: 'en_US',
    style: {
      size: 'small',
      color: 'gold',
      shape: 'pill'
    },
    commit: true,

    //Set up a payment
    payment: function (data, actions) {
      return actions.payment.create({
        transactions: [{
          amount: {
            total: _total,
            currency: 'USD'
          }
        }]
      });
    },
  },

```

```
//Execute the payment
onAuthorize: function (data, actions) {
    return actions.payment.execute().then(function () {
        var url = '@Url.Action("CompleteOrder", "Orders", new { })';
        window.location.href = url;
    });
}

}, '#paypal-btn')
</script>
}
```

ДОДАТОК В
Слайди презентації

Кваліфакаційна робота бакалавра

ТЕМА : ВЕБ ДОДАТОК ДЛЯ ПРОДАЖУ КВИТКІВ У КІНО

Виконав ДЕМЧЕНКО Іван
ст. гр. КНТ 110СП

Керівник КОЛПАКОВА Тетяна
к.т.н. доцент кафедри

Рисунок В.1 – Титульний лист

Об'єкт та мета роботи

- ▶ Об'єкт розробки дипломної роботи – інтернет сайт для продажу квитків у кіно
- ▶ Мета роботи – розробка програмного продукту у вигляді веб додатку, який реалізує функцію онлайн покупок квитків і є доступним у мережі інтернет.

2

Рисунок В.2 – Об'єкт та мета роботи

Задачі

- ▶ Проаналізувати об'єкт розробки;
- ▶ Проаналізувати середовище розробки;
- ▶ Розробити структуру об'єкту дослідження;
- ▶ Розробити структуру бази даних;
- ▶ Розробити основні програмні модулі;
- ▶ Протестувати та відлагодити розроблену програму.

3

Рисунок В.3 – Задачі

Матеріали розробки та технічні засоби

- ▶ Мови програмування : C#
- ▶ Технології : ASP.NET MVC, EntityFramework
- ▶ Технології для інтерфейсу : razor, .cshtml, HTML
- ▶ Середовище розробки : Visual Studio 2022
- ▶ Персональний комп'ютер з процесором більше 2ггц, з встановленою системою Windows 10
- ▶ Необхідні для функціонування бібліотеки : .NET Framework 6.0, Sqlite tools.

4

Рисунок В.4 – Матеріали розробки та технічні засоби

Структурна схема компонентів інтернет магазину

5



Рисунок В.5 – Структурна схема компонентів інтернет магазину

Функціональна схема роботи сайту з продажу квитків

6

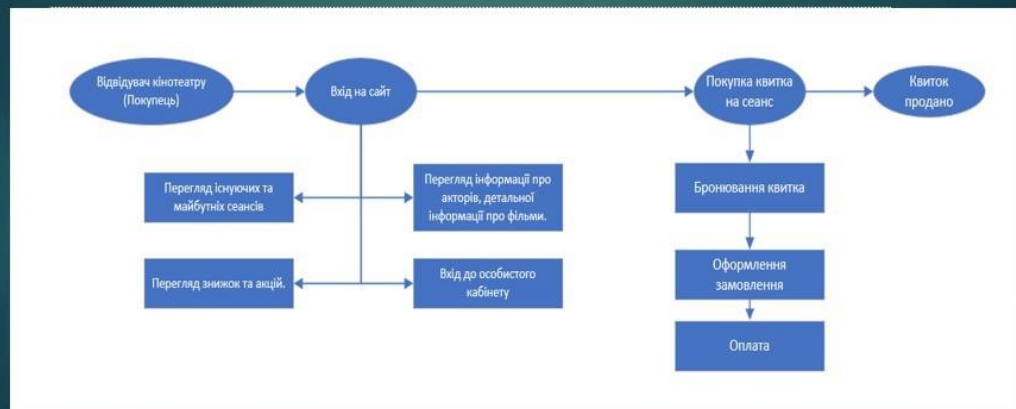


Рисунок В.6 – Функціональна схема роботи сайту з продажу квитків

Структурна схема бази даних

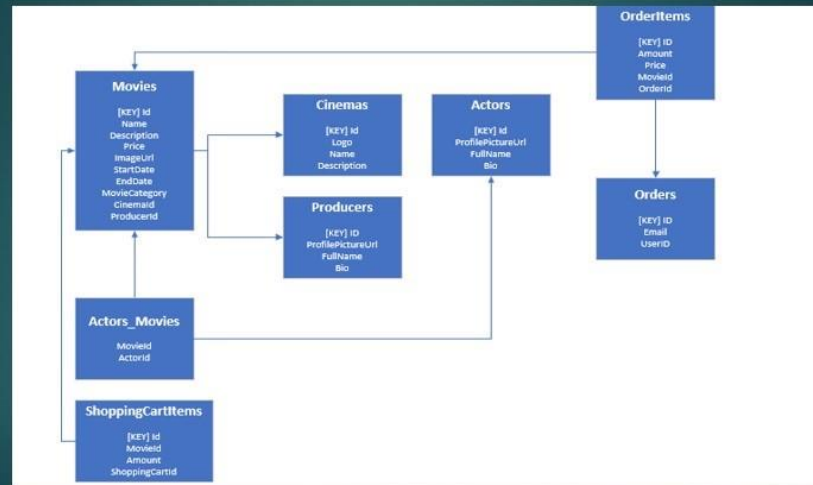


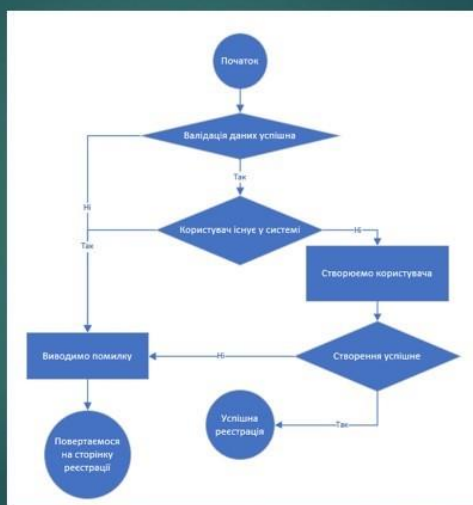
Рисунок В.7 – Структурна схема бази даних

Блок-схема генерації головної сторінки



Рисунок В.8 – Блок-схема генерації головної сторінки

Блок схема реєстрації користувача



9

Рисунок В.9 – Блок схема реєстрації користувача

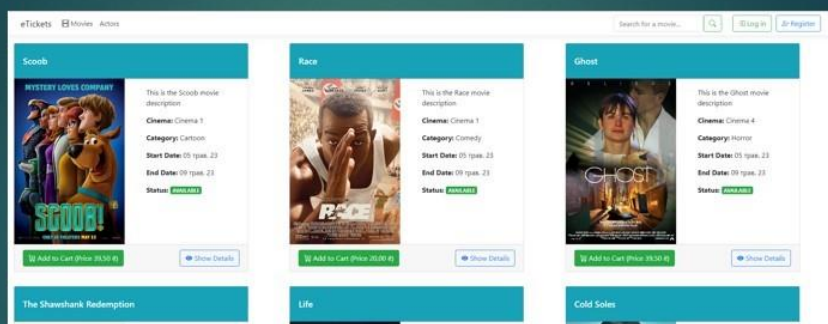
Блок схема роботи кошика



10

Рисунок В.10 – Блок схема роботи кошика

Головна сторінка



11

Рисунок В.11 – Головна сторінка

Вікна авторизації та реєстрації

Log in to your account

Email address

Password

Sign up for a new account

Full name

Email address

Password

Confirm password

12

Рисунок В.12 – Вікна авторизації та реєстрації

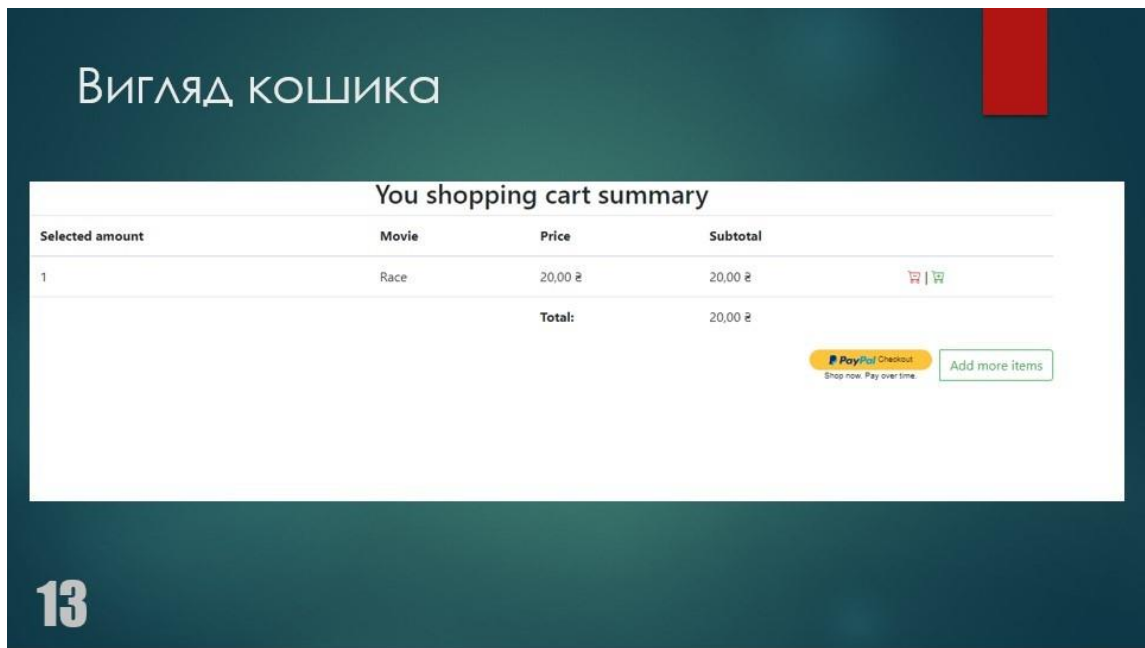


Рисунок В.13 – Вигляд кошика

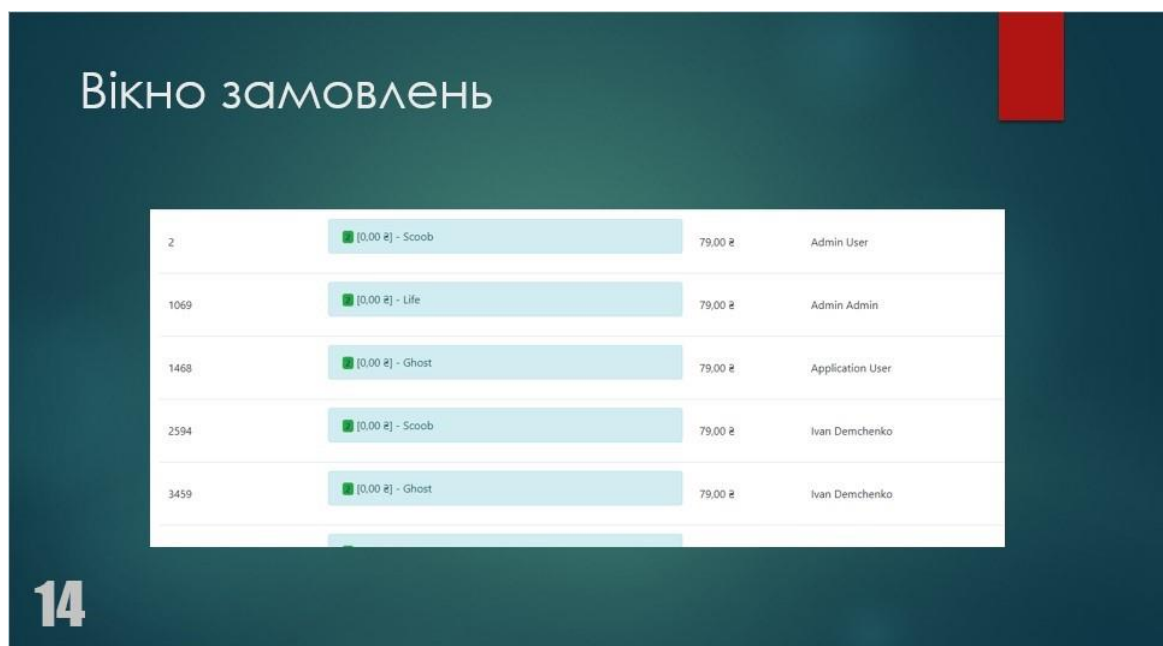


Рисунок В.14 – Вікно замовлень

Висновки

- ▶ розроблений функціонал інтернет магазину повністю відповідає технічному завданню;
- ▶ інтернет магазин працює стабільно та без необхідності постійного обслуговування людиною;
- ▶ сайт має зручний та зрозумілий інтерфейс для користувачів.
- ▶ Додаток забезпечує коректність даних при вводі та виводі.
- ▶ Для розробки було обрано мову C#, та технологію ASP.NET MVC.
- ▶ Для бази було обрано SQLite, та EntityFramework, було створено основну структуру бази даних
- ▶ Було обґрунтовано вибір мови програмування та середовища розробки
- ▶ В результаті випробувань розроблений веб додаток дозволяє відображати інформацію про поточні фільми у кінотеатрах, редагувати інформацію для відвідувачів, забезпечує функціонал придбання квитка через PayPal

15

Рисунок В.15 – Висновки

Дякую за увагу!

16

Рисунок В.16 – Останній лист