Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

РОЗРОБКА WEB-РЕСУРСУ ТЕАТРАЛЬНОЇ УСТАНОВИ

Виконав: студент (4) курсу, групи 3КН–21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Тітарчук О.М.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф.КН
Денисюк В.О.
(прізвище та ініціали)

«04» червня 2025 р.

Рецензент: к.т.н, професор каф. АІІТ
Папінов В.М,
(прізвище та ініціали)

«05.» червня 2025 р.

Допущено до захисту
Зав. кафедри Яровий А.А.
«06» червня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» червня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Тітарчуку Олександру Максимовичу

1. Тема роботи: Розробка WEB-ресурсу театральної установи.
Керівник роботи: Денисюк Валерій Олександрович, к.т.н., доцент
затверджені наказом вищого навчального закладу "20" 03 2025 року №97
2. Строк подання студентом роботи 30 Травня 2025р.
3. Вихідні дані до роботи :
об'єктно-орієнтована мова програмування; середовище розробки Visual Studio; зручний інтерфейс; швидке завантаження сторінок; інформаційне наповнення; забезпечення статичного та динамічного наповнення сайту.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
аналіз можливостей HTML; аналіз можливостей каскадних таблиць стилів CSS; аналіз можливостей клієнтської мови програмування JavaScript; постановка завдання; розробка структури сайту; реалізація web-сайтів; розробка інтерфейсу сайту; вибір колірної гами; розробка динамічного контенту; тестування роботи сайту.
5. Перелік графічного матеріалу:
схема алгоритму роботи модуля; діаграми класів та компонентів програмного забезпечення; загальний вигляд інтерфейсу користувача; приклад роботи програми.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Денисюк В.О., к.т.н., доцент		

7. Дата видачі завдання 5 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
	Аналіз предметної області WEB-ресурсу театральної установи	05.05.25	06.05.25	
	Постановка задачі	07.05.25	09.05.25	
	Розробка програмного модуля WEB-ресурсу театральної установи	10.05.25	13.05.25	
	Обгрунтування засобів розробки WEB-ресурсу театральної установи	14.05.25	17.05.25	
	Програмна реалізація WEB-ресурсу театральної установи	18.05.25	22.05.25	
	Тестування web-ресурсу WEB-ресурсу театральної установи	23.05.25	25.05.25	
	Розробка інструкції користувача	26.05.25	27.05.25	
	Оформлення матеріалів до захисту БКР	01.06.25	04.06.25	

Студент _____ Тітарчук О.М..
(підпис) (прізвище та ініціали)

Керівник роботи _____ Денисюк В.О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 75 сторінок формату А4, на яких є 19 рисунків, список використаних джерел містить 34 найменувань.

У роботі розглянуто актуальну задачу розробки програмного модуля, який забезпечує роботу WEB-ресурсу театральної установи. Досліджено та розроблено систему замовлення квитків у театрі. Розглянуто основні можливості застосування веб-сайтів у галузі. Проаналізовано та досліджено основні методи розв'язання задачі. Було розглянуто існуючі структури сайтів, обрано найкращу, спроектовано базу даних, в якій зберігаються дані, необхідні для роботи системи. Проведене тестування веб-додатку за допомогою модульного тестування. Для всіх найважливіших операцій було написані окремі модульні тести, що дозволили впевнитись в правильності виконання цих операцій. Втілено основні шаблони проектування та використано IoC-контейнер Ninject за допомогою засобів ASP.NET MVC Framework. Для взаємодії з базою даних обрано ORM Entity Framework, через її високу інтеграцію з платформою .NET.

Ключові слова: веб-додаток, театральна установа, платформа .NET, фреймворк ASP.NET MVC, ORM Entity Framework.

ABSTRACT

The bachelor's qualification work consists of 75 pages of A4 format, on which there are 19 figures, the list of used sources contains 34 names.

The paper considers the current task of developing a software module that ensures the operation of a WEB resource of a theater institution. A distributed ticket ordering system in a theater is studied and developed. The main possibilities of using websites in the industry are considered. The main methods for solving the problem are analyzed and researched. The existing structures of sites were considered, the best one was selected, and a database was designed in which the data necessary for the system to work is stored. The web application was tested using unit testing. Separate unit tests were written for all the most important operations, which allowed us to make sure that these operations were performed correctly. The main design patterns were implemented and the Ninject IoC container was used using the ASP.NET MVC Framework. ORM Entity Framework was chosen for interaction with the database, due to its high integration with the .NET platform.

Keywords: web application, theater institution, .NET platform, ASP.NET MVC framework, ORM Entity Framework.

ЗМІСТ

ВСТУП

Актуальність досліджень. Сучасні досягнення у сфері інтернет-технологій сприяють перенесенню дедалі більшої кількості сфер людської діяльності до віртуального простору. Надання послуг театральними установами є яскравою рисою сучасності. Найважливішою послугою є процес купівлі-продажу квитків. Завдяки розвитку засобів розробки веб-ресурсів є можливим створення якісного та привабливого сайту, який буде надавати необхідні послуги. За схемою представлення контенту, його об’єму та категоріях вирішуваних задач розрізняють такі типи веб-ресурсів, як портали, інформаційні ресурси, представництва, сервіси та комбіновані, які можуть поєднувати елементи інших ресурсів. Системи замовлення квитків в театр є зразками систем електронної комерції та різновидами інтернет-магазинів. Інтернет-магазин являє собою веб-ресурс, що забезпечує можливість продажу товарів через Інтернет. Користувачі можуть у режимі онлайн за допомогою браузера або мобільного застосунку оформити покупку, обрати зручні способи оплати та доставки, а також здійснити оплату. Усі дії — від надання інформації споживачем до оформлення покупки та укладання угоди — виконуються безпосередньо на сайті в межах інтернет-середовища. В системі замовлення квитків в театр товаром виступає квиток на певну виставу.

Отже, з вищенаведеного можна зробити висновок, що розробка WEB-ресурсу театральної установи є актуальною та потребує подальшого дослідження.

Метою дослідження є розширення функціональних можливостей WEB-ресурсу театральної установи за рахунок додавання функії замовлення квитків та створення дружнього і комфортного інтерфейса.

Об’єктом дослідження є процеси, які виконуються WEB-ресурсом театральної установи.

Предметом дослідження є алгоритми та програмне забезпечення, що втілює WEB-ресурс театральної установи.

Задачі дослідження:

- дослідження вже наявних аналогічних рішень;
- вибір та обґрунтування підходу до вирішення поставленого завдання;
- розробка структури та дизайну WEB-ресурсу театральної установи;
- програмне створення WEB-ресурсу театральної установи;
- перевірка функціонування ресурсу та аналіз результатів тестування;
- створення користувацької інструкції з експлуатації WEB-ресурсу.

Апробація роботи.

Робота пройшла апробацію на конференції «LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації, 2025р.», доповідь «Розробка веб-додатку театральної установи» [1].

Публікація: електронні тези конференції «LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації., 2025р.» [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ WEB-РЕСУРСУ ТЕАТРАЛЬНОЇ УСТАНОВИ

1.1 Аналіз галузі WEB-ресурсів

Розвиток інтернет-технологій на сучасному етапі сприяв тому, що значна частина повсякденних процесів почала переходити у цифровий простір. Зокрема, до таких процесів належить і здійснення купівлі та продажу товарів і послуг. Завдяки розвитку засобів розробки веб-ресурсів є можливим створення якісного та привабливого сайту, який буде надавати необхідні послуги.

За схемою представлення контенту, його об’єму та категоріях вирішуваних задач розрізняють такі типи веб-ресурсів:

- портали;
- інформаційні ресурси;
- представництва;
- сервіси;
- комбіновані, які можуть поєднувати елементи інших ресурсів.

Системи для онлайн-бронювання театральних квитків можна вважати окремим видом електронної комерції та однією з форм інтернет-магазинів.

Такі сервіси функціонують як веб-ресурси, що забезпечують реалізацію товарів через Інтернет. Користувачі мають змогу через браузер або мобільний застосунок здійснити оформлення покупки, обрати зручний варіант оплати та способу отримання, а також провести оплату онлайн.

Усі етапи — від надання особистих даних до укладання угоди — реалізуються безпосередньо на веб-сайті. У випадку з театральними квитками товаром виступає квиток на конкретну подію або виставу.

1.2 Аналіз відомих програм аналогів

У процесі створення веб-ресурсу театральної організації центральним елементом виступає система онлайн-бронювання квитків. Саме навколо неї вибудовується основна логіка сайту. Така система по суті є формою інтернет-магазину, адже передбачає продаж квитків — специфічного виду товару — через Інтернет.

Інтернет-магазином прийнято вважати веб-сайт, за допомогою якого користувач може вибрати потрібний товар або послугу, оформити покупку, обрати спосіб оплати та доставки. На відміну від інформаційних ресурсів, інтернет-магазини орієнтовані на здійснення фінансових операцій. У контексті системи продажу квитків на вистави, квиток виконує роль товару, доступного для замовлення через Інтернет.

До ключових компонентів інтернет-магазину належать: актуалізація асортименту (в даному випадку – вільних місць та вистав), можливість додавання квитків до "кошика", облік зареєстрованих користувачів тощо. Такі сервіси зазвичай реалізуються у вигляді веб-додатків.

Веб-додаток — це розподілений програмний продукт, в якому клієнтською частиною виступає браузер користувача, а обробка запитів здійснюється сервером. Браузер, як правило, є «тонким клієнтом» і виконує функцію відображення інформації, що надходить із сервера, та передачі введених даних назад. Увесь функціонал програми зазвичай зосереджується на серверному боці, що дозволяє веб-застосункам бути незалежними від операційної системи користувача та забезпечує кросплатформеність.

Перевагою такого підходу є можливість створення універсального додатку, який не потребує адаптації для кожної окремої ОС — Windows, macOS, Linux тощо. Однак, певні складнощі викликають відмінності у реалізації стандартів HTML, CSS, DOM у різних браузерах, а також

варіативність налаштувань з боку користувача (наприклад, зміна розміру шрифтів, кольорової схеми чи вимкнення JavaScript).

Веб-додатки працюють через обробку HTTP-запитів. Отримавши запит від клієнта, сервер виконує необхідні обчислення, формує сторінку і передає її назад користувачу. Крім того, веб-додаток може звертатися до зовнішніх джерел даних — наприклад, баз даних або сторонніх API.

Для підвищення швидкодії та зручності користування в сучасній розробці активно застосовується технологія AJAX. Вона дозволяє надсилати запити до сервера у фоновому режимі, без повного перезавантаження сторінки, довантажуючи лише необхідні фрагменти даних. Це суттєво покращує взаємодію з користувачем.

Одним з найбільш поширених архітектурних підходів при розробці веб-додатків є використання шаблону Model-View-Controller (MVC). Цей підхід дозволяє розділити програму на три логічні частини.

- Модель (Model) - відповідає за роботу з даними і бізнес-логікою.
- Вигляд (View) - забезпечує візуалізацію інформації для користувача.
- Контролер (Controller) - обробляє дії користувача, взаємодіє з моделлю і оновлює вигляд.

Такий поділ покращує структуризацію програмного забезпечення, полегшує підтримку і масштабування проекту, а також сприяє повторному використанню окремих його компонентів. Наприклад, модель може бути змінена або доповнена без потреби втручання у вигляд, і навпаки.

Контролер реагує на дії користувача (натискання кнопок, введення даних), інтерпретує ці дії як події та транслює відповідні запити до моделі. Зміни, що відбулися в моделі, автоматично відображаються у вигляді. Такий підхід забезпечує чітке розмежування обов’язків і робить код додатку більш зрозумілим і підтримуваним.

Порівняння сайтів українських театрів з точки зору зручності користувача (UX) виявляє як позитивні тенденції, так і сфери, що потребують вдосконалення. Основні параметри сайту, з огляду на зручність користувача, включають такі критерії [3-10]:

- навігація - оцінюється за легкістю знаходження інформації, такої як афіша, репертуар, квитки та контакти;
- дизайн - враховується естетика, відповідність стилю театру та сучасність оформлення;
- швидкість завантаження - визначається часом відкриття сторінок та загальною продуктивністю сайту;
- адаптивність - оцінюється здатність сайту коректно відображатися на різних пристроях (комп'ютер, планшет, смартфон);
- онлайн-квитки - наявність та зручність системи онлайн-продажу квитків;
- доступність - відповідність сайту стандартам доступності для людей з інвалідністю;
- UX-рейтинг Оцінка загального користувацького досвіду.

Перелік одних із найвідоміших театрів України та їх сайти [11-21]:

- Національна опера України – http://opera.com.ua ;
- Львівська національна опера - http://opera.lviv.ua ;
- Одеський національний театр опери та балету - http://operahouse.od.ua ;
- Харківський національний театр опери та балету- http://kharkivopera.com ;
- Дніпровський театр драми та комедії - http://dramicom.dp.ua ;
- Сумський театр драми і музкомедії- http://musicdrama.com.ua ;
- Запорізький театр юного глядача - http://teatrmolodi.zp.ua ;
- Вінницький театр ім. М. Садовського - http://sadovskogo.vn.ua ;
- Тернопільський драмтеатр ім. Шевченка - http://dramteatr.te.ua ;
- Закарпатський театр братів Шерегіїв - http://dramteatr.uz.ua .

Оцінка зручності користування (UX) веб-сайтами українських театрів є важливою для забезпечення комфортного досвіду відвідувачів.

У таблиц 1.1 надано порівняння сайтів театрів України за зручністю користувача.

Таблиця 1.1 - Порівняння сайтів театрів України за зручністю користувача

Театр	Навігація	Дизайн	Швидкість	Адаптивність	Бронювання квитків	Купівля квитків
Національна опера України opera.com.ua	+	+	+/- Помірна	+ Повна	+ Прямо на сайті	+ Прямо на сайті
Львівська національна опера opera.lviv.ua	+	+	+/- Помірна	+ Повна	+Через TicketsBox	+ Онлайн
Одеський національний театр опери та балету operahouse.od.ua	+	+	+ Висока	+ Повна	+ Прямо на сайті	+ Прямо на сайті
Харківський національний театр опери та балету kharkivopera.com	+/-	+/-	+/- Повільна	+/- Часткова	- Відсутнє	-Відсутнє
Дніпровський театр драми та комедії dramicom.dp.ua	+	-	+ Висока	+/- Часткова	+/- Через сервіс	+/- Через сервіс
Сумський театр драми і музкомедії musicdrama.com.ua	+/-	+/-	+/- Повільна	+/-Часткова	- Відсутнє	- Відсутнє
Запорізький театр юного глядача teatrmolodi.zp.ua	+/-	+/-	+/- Повільна	+/-Часткова	+/- Відсутнє	- Відсутнє
Вінницький театр ім. М. Садовського sadovskogo.vn.ua	+	+/-	+ Висока	+ Часткова	+/- Через karabas.com	+ Онлайн
Тернопільський драмтеатр ім. Шевченка dramteatr.te.ua	+	+/-	+ Висока	+/- Часткова	+/- Через сервіс	+ Онлайн
Закарпатський театр братів Шерегіїв dramteatr.uz.ua	+/-	+/-	+/- Повільна	+/- Часткова	- Відсутнє	- Відсутнє

Позначення у таблиці 1.1 такі:

«+» – повноцінна можливість (на сайті або через перевірений сервіс);
«+/-» – є, але через сторонні ресурси або неінтуїтивно;
«-» – відсутня можливість.

Отже, аналіз таблиці 1.1 дозволяє зробити висновки.

1. Лідери за зручністю: сайти Одеської та Національної опери України відзначаються сучасним дизайном, повною адаптивністю та зручною навігацією.
2. Потребують оновлення: сайти Харківського, Сумського, Запорізького та Закарпатського театрів мають застарілий дизайн, обмежену інформацію та низьку адаптивність.
3. Кращі сайти з точки зору повного циклу купівлі (зручність + бронювання + онлайн-оплата): Одеський, Національна опера України, Львівська опера, Вінницький театр.
4. Театри, які втрачають користувачів через відсутність купівлі онлайн: Харківський, Сумський, Запорізький, Закарпатський.

Згальні спостереження.

Позитивні риси:

- адаптивний дизайн: більшість сайтів театрів коректно відображаються на мобільних пристроях, що забезпечує зручність користування;
- зрозуміла навігація: меню структуроване логічно, що дозволяє користувачам легко знаходити потрібну інформацію;
- онлайн-продаж квитків: багато театрів впровадили системи онлайн-продажу квитків, що спрощує процес придбання.

Проблемні моменти:

- доступність - не всі сайти повністю відповідають стандартам веб-доступності, що може ускладнювати користування для осіб з обмеженими можливостями;

- технічні збої - деякі користувачі повідомляють про збої в роботі сайтів, особливо під час високого навантаження.

Загальні тенденції використтання веб-ресурсів театральними установами мають позитивні та проблемні аспекти.

Позитивні аспекти такі.

1. Україномовний інтерфейс: Згідно з моніторингом, 96 із 97 театрів мають сайти українською мовою, що відповідає вимогам мовного законодавства [8].
2. Цифрова доступність: Міністерство культури та інформаційної політики (МКІП) спільно з UNDP в Україні працюють над забезпеченням цифрової доступності сайтів театрів для всіх користувачів, включаючи людей з інвалідністю та літніх осіб [9].
3. Електронні квитки: Багато театрів, зокрема Національна опера України, впровадили системи онлайн-продажу квитків, що дозволяє глядачам зручно обирати місця та купувати квитки через інтернет.

Проблемні моменти.

1. Недостатня адаптація для людей з інвалідністю: не всі сайти театрів повністю відповідають стандартам вебдоступності, що може ускладнювати користування для осіб з обмеженими можливостями [9].
2. Технічні збої: користувачі скаржаться на збої в роботі сайтів, особливо під час високого навантаження, що може призводити до труднощів при покупці квитків онлайн.

Рекомендації для покращення UX.

1. Впровадження стандартів веб-доступності (WCAG 2.1) [10]: забезпечення доступності контенту для всіх користувачів, включаючи людей з інвалідністю.

2. Оптимізація мобільних версій сайтів: Забезпечення повної функціональності та зручності користування на мобільних пристроях.
3. Покращення технічної стабільності: Забезпечення безперебійної роботи сайтів, особливо під час пікових навантажень.
4. Розширення мовних версій: Додавання англомовної версії сайту для залучення іноземних відвідувачів.
5. Зворотний зв'язок з користувачами: Впровадження механізмів для збору відгуків та пропозицій від відвідувачів з метою постійного вдосконалення UX.

Загалом, українські театри демонструють позитивні зрушення у напрямку покращення зручності користування своїми сайтами. Проте, для досягнення високого рівня UX необхідно продовжувати роботу над впровадженням сучасних стандартів вебдоступності, технічної стабільності та орієнтації на потреби всіх категорій користувачів

Наприклад, більш детально проаналізовано офіційний сайт Вінницького академічного музично-драматичного театру ім. М. Садовського [19] за основними вимогами.

1. Навігація:

- структура - сайт має чітку структуру з основними розділами: "Контакти", "Про театр", "Репертуар", "Афіша";
- зручність - інтерфейс інтуїтивно зрозумілий, що дозволяє користувачам легко знаходити потрібну інформацію.

2. Дизайн:

- візуальне оформлення - сайт має сучасний та приємний дизайн з використанням якісних зображень вистав та акторів;
- кольорова гама - використання спокійних кольорів сприяє комфортному перегляду.

3. Швидкість завантаження:

- оптимізація - сторінки сайту завантажуються швидко, що забезпечує позитивний досвід користувача.

4. Адаптивність:

- мобільна версія - сайт адаптований для перегляду на різних пристроях, включаючи смартфони та планшети;
- функціональність: -усі функції сайту доступні та зручні у використанні на мобільних пристроях.

5. бронювання та купівля квитків:

- онлайн-купівля - користувачі можуть придбати квитки безпосередньо на сайті через інтеграцію з сервісом Karabas.com ;
- процес покупки - процедура купівлі квитків є простою та зрозумілою, з можливістю вибору місць та оплати онлайн.

Переваги сайту:

- інтуїтивна навігація - користувачі можуть легко орієнтуватися на сайті;
- сучасний дизайн - візуальне оформлення приваблює та утримує увагу відвідувачів;
- швидке завантаження - сторінки сайту відкриваються без затримок;
- адаптивність - сайт коректно відображається на різних пристроях;
- онлайн-купівля квитків - можливість придбати квитки через інтернет спрощує процес для користувачів.

Можливі покращення:

- мультимовність - додавання англомовної версії сайту може залучити іноземних відвідувачів;

- інтерактивність - впровадження інтерактивних елементів, таких як віртуальні тури або відеоанонси, може підвищити залученість користувачів.

Отже, офіційний сайт Вінницького академічного музично-драматичного театру ім. М. Садовського є зручним та функціональним ресурсом для глядачів. Він забезпечує легкий доступ до інформації про вистави, дозволяє швидко та зручно придбати квитки онлайн, але через сторонні сайти, а також адаптований для перегляду на різних пристроях.

Завдяки сучасному дизайну та інтуїтивній навігації, сайт сприяє позитивному досвіду користувачів.

1.3 Постановка задачі

Задачею досліджень та розробки є розробка WEB-ресурсу театральної установи. Для реалізації поставленої мети необхідно виконати наступні етапи:

- сформувати логічну та навігаційну структуру веб-сайту театрального закладу;
- здійснити проєктування та створення бази даних, яка відповідатиме потребам функціоналу;
- визначити архітектуру веб-застосунку;
- розробити безпосередньо сам веб-застосунок;
- реалізувати функціонал для додавання та оновлення інформації про вистави й ціни на квитки;
- надати можливість користувачам здійснювати онлайн-бронювання квитків;
- провести тестування веб-застосунку для перевірки його працездатності та коректної взаємодії компонентів.

1.4 Висновок до розділу 1

Проведені дослідження та аналіз предметної області WEB-ресурсу театральної установи, аналіз відомих програм аналогів дозволив підтвердити актуальність розробки WEB-ресурсу театральної установи, з’ясувати основні задачі подальшого дослідження.

2 РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ WEB-РЕСУРСУ ТЕАТРАЛЬНОЇ УСТАНОВИ

2.1 Розробка структури WEB-ресурсу

Розробка структури веб-ресурсу передбачає врахування як теоретичних, так і практичних аспектів, що забезпечують ефективну, зручну, безпечну та масштабовану роботу сайту. Ось їхній детальний розподіл:

Теоретичні аспекти місять п'ять основних компонент.

1. Аналіз цільової аудиторії відповідає на питання:
   - «Які пристрої та браузери використовуватимуться?»;
   - «Які задачі має вирішувати веб-ресурс?»;
   - «Хто користуватиметься сайтом?».
2. Інформаційна архітектура (IA) містить:
   - ієрархія сторінок - головна сторінка, підрозділи, тематичні категорії;
   - побудова навігаційної структури (меню, підменю);
   - карта сайту (sitemap) - логічна модель всіх сторінок.
3. Юзабіліті та UX-дизайн повинен забезпечувати тикі аспекти як:
   - інтуїтивна навігація;
   - мінімізація кліків до важливої інформації;
   - візуальна послідовність (eye-tracking, принципи гештальту).
4. SEO-орієнтоване проєктування враховує наявність таких компонент:
   - семантична структура HTML (заголовки, теги);
   - URL-структура (читаємі, ключові слова);
   - мікророзмітка (Schema.org, OpenGraph).
5. Модульність і масштабованість передбачає, що:
   - веб-ресурс має легко доповнюватися новими розділами;
   - уникають надмірної складності на старті.

Практичні аспекти складаються із шести компонент.

1. Планування технологій:
   - frontend - HTML5, CSS3, JavaScript (React, Vue, або Vanilla);
   - backend - PHP, Python (Django/Flask), Node.js тощо;
   - база даних - MySQL, PostgreSQL, MongoDB.
2. Приклад проєктування структури папок надано на рисунку 2.1.

```
/ (корінь)
├── index.html (або index.php)
├── /css/
├── /js/
├── /images/
├── /includes/ (шаблони, хедери, футери)
├── /pages/ (окремі сторінки)
├── /admin/ (якщо є)
└── /api/ (якщо REST)
```

Рисунок 2.1 – Приклад структури папок розробки

3. Реалізація шаблонів та компонентів:
   - використання шаблонних двигунів (Blade, Twig, EJS);
   - компонентна структура в React/Vue: повторно використовувані блоки UI.
4. Контроль доступу та безпека:
   - аутентифікація/авторизація;
   - захист форм (CSRF, XSS);
   - HTTPS, захищене зберігання паролів.
5. Тестування структури:
   - перевірка роботи навігації;
   - адаптивність (Responsive Design);
   - тестування на різних пристроях/браузерах.
6. Засоби розгортання та обслуговування:
   - CI/CD (наприклад, GitHub Actions);

- хостинг: Apache, Nginx, Docker;
- моніторинг та логування (Sentry, Google Analytics).

Розробка сайту для театральної установи має свою специфіку, яка враховує культурну місію, потреби глядачів, розклад вистав, продаж квитків та естетику мистецтва. Такий веб-ресурс виконує функції як інформаційного порталу, так і інструменту комунікації та продажу.

Ось детально сім параметрів, які потрібно врахувати.

1. Цільова аудиторія:
   - глядачі різного віку (студенти, родини, пенсіонери);
   - туристи (бажано багатомовність);
   - постійні відвідувачі та шанувальники театру;
   - журналісти, критики, партнери.
2. Ключові функції сайту театру надані у таблиці 2.1.
3. Дизайн та естетика:
   - візуальний стиль: драматичний, витончений, емоційний;
   - адаптивність: для смартфонів, планшетів;
   - контрастність: темні кольори + яскраві акценти;
   - типографіка: використання театральних шрифтів, афішного стилю;
   - відео - фон чи анімовані елементи (з обережністю).
4. Технічна реалізація (приклад основної структури сайту надано на рисунку 2.2).

Технології:

- Frontend: HTML5, CSS3 (або Tailwind), JavaScript (можливо React);
- Backend: PHP 8 (або Node.js / Python Flask);
- CMS (опціонально): WordPress (з плагінами подій і білетів);
- база даних - MySQL або PostgreSQL;

- API - якщо інтеграція з зовнішньою системою продажу квитків.

Таблиця 2.1 - Ключові функції сайту театру

Розділ	Опис
Афіша / Розклад	Календар подій, вистави за датами, жанрами.
Купівля квитків	Онлайн-продаж, інтеграція з білетними сервісами (або власна система).
Про театр	Історія, місія, команда, трупа, режисери.
Репертуар	Інформація про вистави: опис, актори, трейлери, рецензії.
Новини / Блог	Анонси, прес-релізи, статті.
Контакти / Карта	Адреса, схеми проїзду, контактні форми.
Галерея / Відео	Фото, трейлери, закулісся, архів подій.
Мовна підтримка	Українська / Англійська / Інші мови — залежно від регіону.
Особистий кабінет	Для постійних глядачів: історія покупок, бонуси, підписки.

5. Особливості адміністрування:

- додавання нових вистав;
- завантаження афіш, відео, галерей;
- оновлення розкладу;
- вивантаження продажів / звітів.
- масова e-mail розсилка.

```
/
├── index.php                    # Головна сторінка з
├                                # афішею
├── about.php                    # Про театр
├── schedule.php                 # Афіша (розклад)
├── play.php?id=                 # Сторінка конкретної
├                                # вистави
├── tickets.php                  # Продаж квитків
├── contact.php                  # Контакти
├── news.php                     # Новини
├── /admin/                      # Панель
├                                # адміністрування
├── /css/
├── /js/
├── /images/
└── /db/                         # Підключення до бази
                                 # MySQL/PostgreSQL)
```

Рисунок 2.2 - Приклад основної структури сайту

6. Приклад UX сценарію: купівля квитка:

- користувач заходить на головну → бачить афішу;
- клікає на виставу → відкривається детальна сторінка;
- натискає "купити квиток" → бачить схему залу;
- вибирає місце → переходить до оплати;
- отримує електронний квиток.

7. Додатково враховуємо:

- інтеграція з соцмережами.
- розсилка новин / email-підписка;
- банери з майбутніми прем’єрами;
- підтримка людей з вадами зору (контрастний режим, більший шрифт).

Отже, сайт театру - це не просто сторінка з інформацією. Це віртуальний простір мистецтва, який має:

- передати атмосферу закладу,
- заохотити відвідування вистав,
- спростити купівлю квитків,
- і створити емоційний контакт із користувачем.

Подальші покращення веб-сайту:

- підтвердження оплати через LiqPay / WayForPay / Stripe;
- ліміти на час утримання броні (наприклад, 15 хв.);
- автоматичне очищення непридбаних броней;
- авторизація користувача;
- адмінпанель з додаванням вистав і розкладу.

У таблиці 2.2 надано основні етапи розробки сайту для театральної установи.

Розглянемо етапи детальніше.

1. Постановка цілей та аналіз вимог:
    - визначення мети сайту (інформування, продаж квитків, створення іміджу);
    - аналіз цільової аудиторії;
    - вивчення конкурентів, сучасних трендів;
    - формулювання функціональних та нефункціональних вимог.
2. Створення технічного завдання (ТЗ):
    - структура сайту (меню, сторінки, модулі);
    - функціонал (бронювання, реєстрація, пошук, фільтри);
    - база даних, CMS (чи без неї);
    - вимоги до дизайну, адаптивності, мовних версій.
3. Проєктування (UX/UI):
    - складання структурної карти сайту (sitemap);
    - розробка wireframes (каркасів сторінок);

- створення прототипу користувацького інтерфейсу;
- узгодження макетів з замовником.

Таблиця 2.2 - Етапи розробки сайту для театральної установи

№	Етап	Опис	Результат
1	Постановка цілей	Визначення мети сайту, аналіз аудиторії, вивчення конкурентів	Збір вимог, аналітичний звіт
2	Технічне завдання (ТЗ)	Формування структури, функцій, дизайну, технологій	Затверджене ТЗ
3	Проєктування (UX/UI)	Створення карти сайту, прототипів інтерфейсу	Wireframes, макети
4	Дизайн	Розробка візуального стилю, шрифтів, кольорової гами	Готові графічні макети
5	Розробка (frontend/ backend)	Верстка сторінок, програмування логіки, підключення бази даних	Прототип або демо сайту
6	Тестування	Перевірка роботи всіх функцій, адаптивності, безпеки	Виправлений, стабільний сайт
7	Запуск	Завантаження на хостинг, підключення домену, SEO	Сайт в мережі, доступний для всіх
8	Підтримка	Оновлення контенту, виправлення помилок, розвиток функціоналу	Живий, актуальний ресурс

4. Дизайн інтерфейсу:

- візуальна стилістика відповідно до теми театру;
- підбір кольорів, шрифтів, графіки, анімацій;
- розробка адаптивного дизайну (для мобільних, планшетів).

5. Розробка (frontend + backend):

- HTML/CSS/JS верстка;
- програмування логіки (PHP, Python, Node.js тощо);
- інтеграція з базою даних (MySQL, PostgreSQL);
- реалізація особистого кабінету, бронювання, CMS-модуля.

6. Тестування:

- функціональне тестування (всі кнопки, форми, оплати);
- тестування адаптивності (на різних пристроях);
- перевірка безпеки (введення даних, SQL-ін’єкції);
- усунення помилок.

7. Запуск сайту:

- перенесення на хостинг;
- підключення домену;
- налаштування SEO (title, meta, sitemap);
- реєстрація в Google Search Console.

8. Підтримка та розвиток:

- регулярне оновлення контенту (афіша, новини);
- моніторинг роботи сайту;
- оптимізація швидкодії;
- додавання нового функціоналу за потребою.

Якщо сайт театру, додатково враховується: афіша, зв’язок з квитковою системою, креативний дизайн, медіа-галереї та підтримка декількох мов.

2.2 Розробка діаграми класів WEB-ресурсу театральної установи

UML-діаграма класів відображає структуру системи управління театральними сеансами та квитками (рис 2.3).

Розглянемо опис основних класів і зв’язків між ними.

Клас `Show`:

- Атрибути - `Id`, `Name`, `ReleaseDate`, `Duration`, `Genre`, `Director`, `Description`, `ImageData`, `ImageMimeType`;
- зв’язки - має колекцію `Seances` (1 до багатьох).

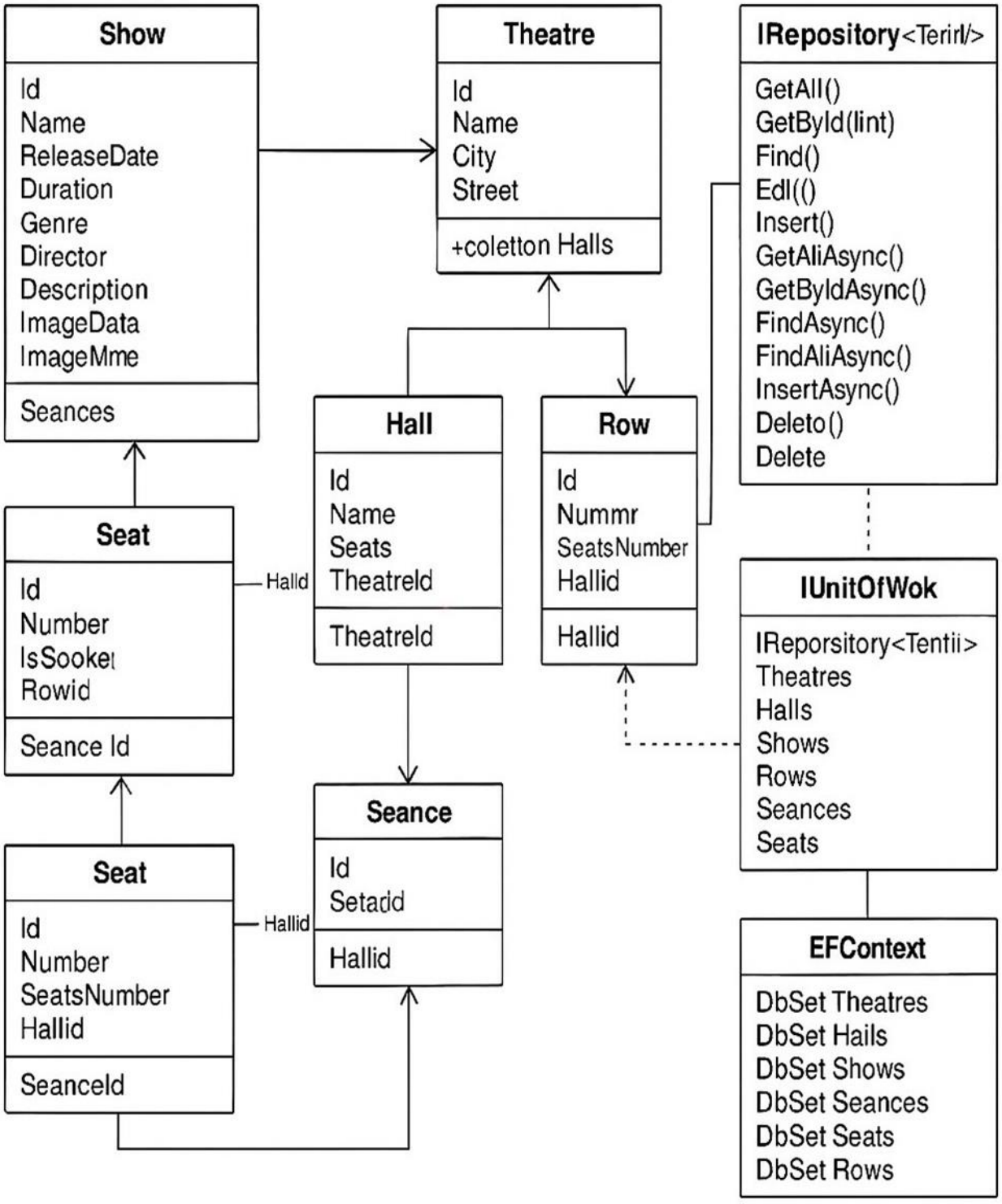

Рисунок 2.3 - UML-діаграма класів WEB-ресурсу театральної установи

Клас `Theatre`:

- атрибути - `Id`, `Name`, `City`, `Street`;
- зв’язки - має колекцію `Halls` (1 до багатьох).

Клас `Hall`:

- атрибути: `Id`, `Name`, `Seats`, `TheatreId`;
- зв’язки - належить одному `Theatre`; має колекції `Seances` та `Rows`.

Клас `Row`:

- атрибути: `Id`, `Number`, `SeatsNumber`, `HallId`;
- зв’язки - належить одному `Hall`; має колекцію `Seats`.

Клас `Seat`:

- атрибути - `Id`, `Number`, `IsBooked`, `RowId`, `SeanceId`;
- зв’язки - належить одному `Row`; належить одному `Seance`.

Клас `Seance`:

- атрибути - `Id`, `Date`, `Time`, `SeatsLeft`, `Price`, `HallId`, `ShowId`;
- зв’язки - належить одному `Hall`; належить одному `Show`; має колекцію `Seats`.

Інтерфейс `IRepository<TEntity>`:

- методи CRUD - `GetAll()`, `GetById()`, `Find()`, `Insert()`, `Edit()`, `Delete()`, а також асинхронні версії.

Інтерфейс `IunitOfWork`:

- властивості - репозиторії для `Theatres`, `Halls`, `Shows`, `Rows`, `Seances`, `Seats`.

Клас `EFContext` (DbContext):

- властивості: `DbSet<Theatre>`, `DbSet<Hall>`, `DbSet<Show>`, `DbSet<Seance>`, `DbSet<Seat>`, `DbSet<Row>;`
- призначення: відображення бази даних у контексті Entity Framework.

Відносини:

- 1 до багатьох: між `Theatre–Hall`, `Hall–Row`, `Row–Seat`, `Show–Seance`, `Seance–Seat;`
- Композиції / агрегації: `EFContext` включає колекції (через `DbSet`), `UnitOfWork` — агрегує репозиторії.

Ця структура добре відповідає предметній області театральних сеансів, дозволяючи ефективно керувати розкладом, квитками та місцями.

2.3 Послідовність дій WEB-ресурсу театральної установи

UML-діаграма послідовність дій WEB-ресурсу театральної установи зображає процес створення сеансу (Seance) для конкретної вистави (Show) у певному залі (Hall) театру (Theatre), а також пов'язану взаємодію з сутностями Seat та Row (рис.2.4).

Розглянемо опис UML-діаграми послідовності.

Учасники (Actors / Lifelines):

- користувач / адміністратор – ініціатор дії (наприклад, додає новий сеанс);
- ShowRepository – надає доступ до об'єкта вистави;
- HallRepository – забезпечує доступ до зали;
- SeanceRepository – створює об'єкт сеансу;
- SeatRepository / RowRepository – зберігають інформацію про місця та ряди, що пов'язані із сеансом.

Основна послідовність дій містить 7 етапів.

1. Користувач ініціює створення нового сеансу.
2. Запит до ShowRepository: отримати інформацію про відповідну виставу.

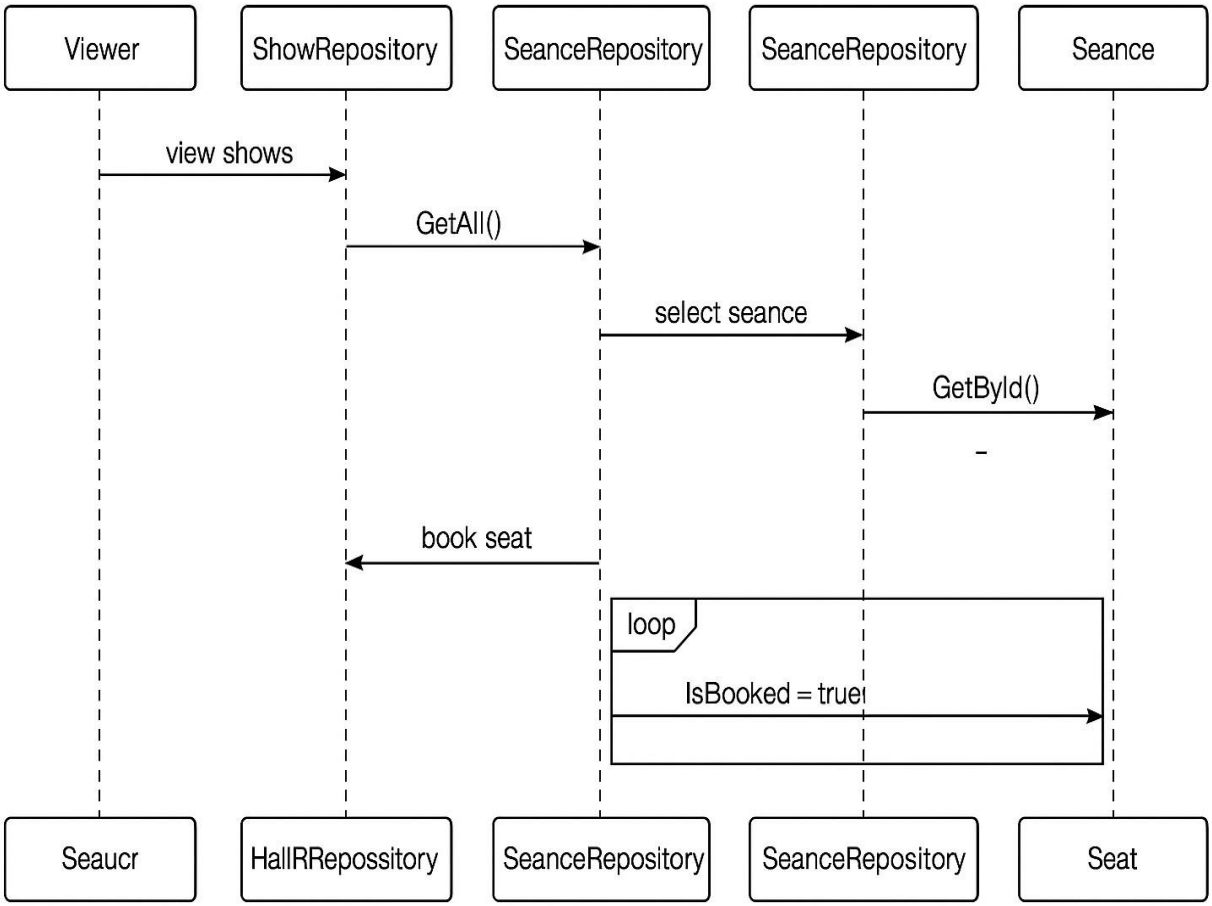

Рисунок 2.4 - UML-діаграма послідовність дій WEB-ресурсу театральної установи.

3. Запит до HallRepository: отримати інформацію про залу, де буде проходити сеанс.
4. Створення нового об’єкта Seance, який зв’язується із Show і Hall.
5. Збереження Seance у базі через SeanceRepository.
6. Ініціалізація рядів (Row) та місць (Seat) у залі для цього сеансу:

- для кожного ряду створюється відповідна кількість місць.
- кожне місце отримує прив'язку до конкретного сеансу.

7. Збереження рядів та місць у базу через RowRepository та SeatRepository.

Мета діаграми - показати, як різні об’єкти (класи) взаємодіють між собою для реалізації функціональності додавання нового сеансу в систему кінотеатру. Це наочно демонструє принципи репозиторного патерну та використання зв’язків між сутностями.

2.4 Висновок до розділу 2

У результаті дослідження розроблено структуру програмного модуля WEB-ресурсу театральної установи, діаграму класів програмного модуля WEB-ресурсу, послідовність дій WEB-ресурсу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-РЕСУРСУ ТЕАТРАЛЬНОЇ УСТАНОВИ

3.1 Обґрунтування вибору мови програмування

Обґрунтування вибору засобів реалізації веб-сайту.

До засобів реалізації сайту відносять методи ручного та автоматизованого створення. Перше передбачає програмування сайту на допомогою мови розмітки, мов сценаріїв з використанням каскадних таблиць стилів. Друге – використання конструкторів сайту [22].

Використання мов веб-програмування та стилів не обмежує можливості по створенню власного дизайну. Крім того, можна писати код як у простих редакторах коду, так і в середовищах розробки, які полегшують процес написання засобами автоматичного доповнення коду, підказками, детальною документацію по синтаксису і методам програмування. Також у мережі Інтернет існують професійні форуми та інші ресурси, які розглядають вузькоспеціалізовані питання, розміщують шаблони та методи реалізації елементів дизайну, дають поради і рекомендації з написання коду та створення інтерфейсу користувача. Недоліком ручного створення є відносно повільніший процес розробки та її складність.

Архітектурний підхід Model-View-Controller (MVC) є широко застосовуваним шаблоном у процесі створення програмних систем. Його суть полягає у логічному поділі застосунку на три окремі компоненти:

- Model (модель) – відповідає за управління даними та бізнес-логікою;
- View (вигляд) – відповідає за візуальне представлення даних користувачу;
- Controller (контролер) – забезпечує зв’язок між інтерфейсом та логікою, реагуючи на дії користувача та оновлюючи модель і

вигляд.

Головна мета MVC — створення гнучкої, масштабованої структури, яка полегшує внесення змін, повторне використання модулів та підтримку коду. Такий підхід сприяє зменшенню складності в великих інформаційних системах, завдяки чітко розмежованим функціям складових частин.

На базі цього архітектурного шаблону реалізовано багато сучасних фреймворків для веб-розробки. Серед найпоширеніших:

- ASP.NET MVC – рішення від Microsoft, орієнтоване на розробку веб-сервісів і застосунків;
- Symfony – PHP-фреймворк з відкритим кодом, підтримується компанією SensioLabs;
- Ruby on Rails – популярне рішення для мови Ruby, орієнтоване на швидку розробку;
- Express.js – мінімалістичний фреймворк на базі Node.js;
- Django – Python-фреймворк з багатим набором можливостей, підтримується Django Software Foundation.

Архітектура Django близька до MVC, але має власну термінологію: контролер у цій моделі представлений рівнем Подань (Views), а за візуальне відображення відповідають Шаблони (Templates). У зв’язку з цим, архітектурний підхід Django часто описують як Model-Template-View (MTV).

Основні особливості Django:

- ORM-система, яка дозволяє працювати з базами даних через Python-класи;
- інтерфейс адміністратора, який забезпечує керування вмістом без додаткової розробки;
- гнучкий механізм маршрутизації через регулярні вирази;
- вбудоване кешування та підтримка багатомовності;
- структура додатків**, що дозволяє легко підключати та масштабувати функціональність;

- система фільтрів**, зокрема для кешу, стиснення, сесій тощо;
- гнучка система шаблонів**, яку можна замінювати на сторонні рішення (наприклад, Jinja, Mako).

Слід зазначити, що хоча компоненти Django є доволі незалежними, деякі з них (наприклад, ORM) замінити складно. Первісно Django створювався для потреб інформаційних порталів, тому містить інструменти, які значно спрощують розробку контентних сайтів. Наприклад, модуль адміністративного інтерфейсу дозволяє керувати об'єктами бази даних без потреби написання додаткового коду для інтерфейсу управління.

Для взаємодії з базою даних фреймворк використовує власну ORM-технологію, де модель описується як Python-клас, а її структура автоматично транслюється у схему БД.

Symfony є сучасним PHP-фреймворком, що спрощує створення та адміністрування веб-додатків, зокрема шляхом автоматизації стандартних задач веб-розробки. Фреймворк сумісний із різноманітними системами управління базами даних, включно з MySQL, PostgreSQL, SQLite, а також будь-якими іншими, що підтримують інтерфейс PDO. Для інтеграції реляційної структури бази даних з об’єктною моделлю застосовуються ORM-рішення. Symfony підтримує два основні ORM-інструменти — Propel та Doctrine — які дозволяють легко створювати, зберігати й модифікувати об’єкти, синхронізовані з таблицями в БД.

Фреймворк ASP.NET MVC, розроблений Microsoft, забезпечує побудову як простих, так і високонавантажених веб-сайтів, що здатні обробляти мільйони запитів. Він є зручним для розробників, які мають досвід роботи з Windows, проте не знайомі з Unix-середовищем. Серед його недоліків можна назвати обмежений вибір недорогих хостингів у порівнянні з Unix-платформами, а також необхідність придбання ліцензій при використанні окремих серверів.

Ruby on Rails (або просто Rails) є потужним фреймворком, який

дозволяє максимально швидко створити повнофункціональний веб-додаток. Це комплексне середовище розробки, яке охоплює всі основні аспекти: управління базою даних через систему міграцій, реалізацію бізнес-логіки через контролери та дії, а також створення тестів. Rails ґрунтується на кількох базових принципах.

1. MVC-структура застосовується як стандартна архітектура для всіх проєктів.
2. Синтаксис Ruby сприяє більш інтуїтивному опису поведінки додатків та зв’язків між сутностями.
3. Дотримання принципу Don't Repeat Yourself (DRY) дозволяє уникати дублювання коду, підвищуючи підтримуваність проєктів.
4. Концепція Convention over Configuration означає, що більшість налаштувань мають стандартні значення, що скорочує обсяг конфігураційного коду.

Rails підходить для проєктів, де пріоритетом є висока швидкість розробки, хоча недоліком залишається нижча продуктивність у порівнянні з деякими іншими рішеннями.

На відміну від Rails, ASP.NET MVC концентрується виключно на реалізації обробки запитів через контролери та дії в межах шаблону MVC. Сам по собі фреймворк не містить засобів для роботи з базами даних (ORM), не має вбудованого механізму тестування або інструментів для міграції даних. Проте, у межах платформи .NET доступно багато сторонніх рішень, які дозволяють розширити функціональність відповідно до потреб проєкту.

Express.js – Node.js серверний web-застосунок, призначений для створення одинарних сторінок, багатосторінкових і гібридних веб-додатків. Це є де-факто стандартом серверного фреймворка для Node.js. Має мінімальну кількість функцій, доступних у вигляді плагінів.

У таблиці 3.1 представлено порівняльний аналіз різних MVC-

фреймворків.

З урахуванням проведеного аналізу для реалізації веб-додатку було прийнято рішення використовувати ASP.NET MVC. Головною перевагою цього фреймворку є його незалежність від конкретного ORM-інструменту, що дає розробнику свободу у виборі найбільш зручного засобу для роботи з базою даних, на відміну від фреймворків Ruby on Rails чи Django, де ORM зазвичай жорстко інтегрований у середовище.

Окрім цього, ASP.NET MVC дозволяє інтегрувати різноманітні фреймворки для модульного тестування, що забезпечує більшу гнучкість у виборі інструментів тестування.

Вибір бази даних та засобів доступу.

Одним із ключових етапів під час створення веб-додатку є проектування структури бази даних та вибір способу взаємодії з нею. У випадку використання ASP.NET MVC найчастіше застосовується Microsoft SQL Server (MS SQL Server) — одна з основних СУБД, сумісних з екосистемою Microsoft.

MS SQL Server — це комерційна система управління базами даних, яка розробляється компанією Microsoft. В якості основної мови запитів вона використовує Transact-SQL (T-SQL) — діалект SQL, розроблений у співпраці з компанією Sybase.

T-SQL базується на стандарті ANSI/ISO SQL, однак містить низку розширень, які забезпечують більш гнучке управління даними, зокрема підтримку процедур, тригерів, обробки помилок та складних транзакцій.

Застосування MS SQL Server у поєднанні з ASP.NET MVC дозволяє отримати надійне, масштабоване та ефективне рішення для обробки даних у веб-додатку.

Таблиця 3.1 – Порівняння MVC-фреймворків.

Критерій	ASP.NET MVC	Django	Ruby on Rails	Symfony	Express.js
Мова	ASP.NET	Python	Ruby	PHP	JavaScript
Використання ORM	ORM-незалежний	Так	Так	Так	Так
Механізми інтернаціоналізації та локалізації	Так	Так	Так	Так	Так
Використання шаблонів при створенні користувацьких інтерфейсів	Так	Так	Відсутнє	Так	Відсутнє
Створення та перевірка форм	Так	Так		Так	Відсутнє
Керування доступом на основі ролей	Так	Так	Відсутнє	Так	Відсутнє
Використання Ajax	Так	Так	Відсутнє	Так	Відсутнє
Модульне тестування	Так	Так	Відсутнє	Так	Так
Система кешування	Так	Так		Так	Так

Microsoft SQL Server застосовується для обслуговування як невеликих і середніх, так і масштабних баз даних на рівні підприємств. Протягом тривалого часу ця система успішно змагається з іншими провідними СУБД

на ринку.

Для обробки запитів у SQL Server використовується мова Transact-SQL (T-SQL) — розширений варіант стандарту SQL-92, доповнений можливістю створення збережених процедур, підтримкою транзакцій та додатковими операторами керування даними. Це дозволяє ефективно організовувати взаємодію бази даних із зовнішніми додатками.

Однією з особливостей SQL Server є використання протоколу **TDS (Tabular Data Stream)**, спільно з **Sybase ASE**, який забезпечує обмін табличними даними через мережу на прикладному рівні.

SQL Server підтримує розподілену обробку запитів завдяки таким функціям, як кластеризація та дзеркалювання баз даних. Кластер являє собою групу серверів з ідентичною конфігурацією, які працюють під єдиним віртуальним ім’ям. Такий підхід дозволяє рівномірно розподіляти навантаження, забезпечує високу доступність системи та автоматичне переключення у разі збоїв одного з вузлів.

Починаючи з версії 2005, SQL Server отримав інтеграцію з платформою .NET Framework, що надало можливість створювати збережені процедури мовами .NET, наприклад, C# чи VB.NET. При цьому використовуються повноцінні бібліотеки платформи .NET, що значно розширює можливості обробки даних всередині СУБД.

Ця інтеграція дозволяє SQL Server працювати автономно від базових механізмів Windows, використовуючи власні засоби управління ресурсами, які оптимізовані під вимоги баз даних, що забезпечує кращу продуктивність у порівнянні зі стандартними підходами Windows.

Для зручнішої роботи з даними часто застосовуються технології об'єктно-реляційного відображення (ORM), які дозволяють працювати з даними через об'єкти, використовуючи мови об'єктно-орієнтованого програмування. У середовищі .NET найбільш популярними ORM-рішеннями є Entity Framework та NHibernate.

NHibernate — це безкоштовний інструмент з відкритим кодом, який дозволяє здійснювати об’єктно-реляційне зіставлення у середовищі .NET. Він розповсюджується за ліцензією GNU GPL і надає широкі можливості для гнучкого доступу до реляційних даних, мінімізуючи необхідність у ручному написанні SQL-запитів [27].

Ключова особливість NHibernate полягає у здатності автоматично відображати класи, створені у середовищі .NET, на відповідні таблиці реляційної бази даних. Це включає також перетворення типів даних .NET (CLR) у SQL-типи. Завдяки цьому NHibernate формує SQL-запити самостійно, звільняючи розробника від необхідності вручну керувати наборами результатів і перетворювати їх у об'єкти. Такий підхід забезпечує високу переносимість додатків між різними SQL-сумісними базами даних, при цьому втрати продуктивності залишаються мінімальними.

Відображення між класами і таблицями конфігурується за допомогою XML-файлів. На основі цих файлів NHibernate здатний автоматично згенерувати шаблон коду для класів, що відповідають збереженим об'єктам. Крім XML, можливе також використання атрибутів (анотацій) у коді для опису структури та взаємозв'язків.

NHibernate дозволяє налаштовувати відображення для користувацьких типів даних, що відкриває додаткові можливості:

- змінювати тип SQL, який за замовчуванням відповідає конкретній властивості;
- відображати перерахування (.NET Enum) як прості значення у базі;
- зіставляти одну властивість класу з кількома колонками в таблиці.

Entity Framework — це об'єктно-реляційний механізм доступу до даних, вбудований у .NET, який пропонує більш високий рівень абстракції. Він дозволяє розробнику працювати з даними не на рівні таблиць та зв’язків, а як

із об’єктами, приховуючи деталі реалізації сховища.

Основна одиниця в Entity Framework — це сутність (entity), яка представляє логічну модель об’єкта з набором пов’язаних властивостей. Наприклад, сутність «Користувач» може містити такі характеристики, як ім’я, прізвище, дата народження або адреса. Унікальні поля, що однозначно ідентифікують об’єкт, виступають у ролі ключів.

Таким чином, замість прямої роботи з таблицями, індексами чи зовнішніми ключами, розробник маніпулює сутностями та їх колекціями. Цей підхід дозволяє будувати додатки, які залишаються гнучкими щодо змін бази даних, оскільки бізнес-логіка орієнтована на об'єкти, а не на структуру сховища [28, 29].

У Entity Framework сутності можуть бути пов’язані між собою асоціативними зв’язками типу «один до одного», «один до багатьох» або «багато до багатьох». Ці взаємозв’язки моделюються аналогічно до зовнішніх ключів у традиційних реляційних базах даних, що дозволяє зручно передавати логіку структури з БД у програмний код.

Однією з характерних можливостей Entity Framework є підтримка LINQ-запитів (Language Integrated Query). Вони надають інструмент для гнучкої вибірки інформації з бази даних у вигляді об'єктів. За допомогою LINQ можна не тільки отримувати окремі сутності, а й легко витягувати пов’язані з ними об'єкти, завдяки чіткому визначенню зв’язків між ними.

Іншою ключовою складовою технології є Entity Data Model (EDM) — модель, яка описує, як класи у коді співвідносяться з елементами бази даних. Вона має три логічних рівні.

1. Концептуальний рівень — визначає об'єкти-сутності, з якими працює додаток.
2. Рівень зберігання — описує фізичну структуру бази даних: таблиці, поля, типи даних, ключі та зв'язки.
3. Рівень відображення (мапінгу) — встановлює відповідність між

властивостями об’єктів і стовпцями у таблицях бази даних.

Завдяки такій трирівневій архітектурі розробник може ефективно керувати даними, взаємодіючи з ними через об'єкти, не відволікаючись на технічні деталі зберігання інформації.

Entity Framework підтримує три підходи до створення і взаємодії з базою даних.

1. Database First — модель формується на основі вже існуючої бази даних, а класи генеруються автоматично.
2. Model First — спочатку створюється модель у вигляді діаграми, після чого система на її основі генерує схему бази даних.
3. Code First — розробник спочатку створює класи сутностей, а база даних створюється автоматично відповідно до цієї моделі.

У таблиці 3.2 представлено детальне порівняння двох популярних ORM-систем — NHibernate і Entity Framework — з урахуванням їхніх ключових характеристик та підходів до роботи з базами даних.

Таблиця 3.2 – Порівняння ORM для .NET

Критерій	NHibernate	Entity Framework
Підтримка та генерація запитів до бази даних	Так	Так
Генерація бази даних з існуючої об’єктної моделі	Так	Так
Генерації об’єктної моделі з існуючої бази даних	Ні	Так

Для реалізації веб-додатку було обрано ORM-технологію Entity Framework, оскільки вона є нативним рішенням, спеціально розробленим для платформи .NET. Це забезпечує глибшу інтеграцію з екосистемою .NET порівняно з NHibernate, який є стороннім інструментом. Завдяки цьому Entity

Framework надає ширший спектр інструментів і можливостей для розробки, що дозволяє реалізовувати функціональність додатку більш ефективно та з меншою кількістю зовнішніх налаштувань.

Розробка структури.

Структура сайту – це система взаємного розміщення і взаємозв'язків.

При організації структури сайту необхідно дотримуватись певних правил і рекомендацій:

- дотримання чіткої структури посилань: кожний документ повинен відноситись до свого розділу;
- для середніх і великих проектів слід створювати карту сайту, яка допоможе зорієнтуватись серед його розділів;
- кожна сторінка повинна мати свою унікальну адресу (URL);
- навігація по сайту повинна однаково виглядати на всіх сторінках сайту;
- кількість вкладень у навігації не повинна перевищувати третій рівень. На більшості сайтів реалізовано два рівні навігації: глобальний і локальний, що аналогічно розділам і вкладеним підрозділам. Подальші рівні вкладеності ускладнюють навігацію по сайту;
- для назв розділів слід використовувати загальноприйняту термінологію, крім того, потрібно уникати довгих назв.

Розробляючи структуру сайту, можна використати діаграми визначення сторінок (PDD), які відповідають на питання, який контент належить певній сторінці та який пріоритет кожної частини матеріалу.

Загальна схема PDD використовує пріоритет горизонтальний доступу. Наприклад, сторінка може мати три колонки: перша колонка має найвищий пріоритет і в ній повинна розміщуватись найважливіша інформація, далі пріоритет спадає, і, відповідно, у наступних колонках розміщується менш важлива інформація. Такі діаграми дозволяють на етапі проектування

відділити дизайн сайту від контенту, що прискорює розробку.

Структура сайту є індикатором того, яким чином виконуватиметься побудова сторінок усередині ресурсу і як буде розташована інформація. Залежно від цього також визначається зручність вивчення проекту. Тому підбирати структуру необхідно украй відповідально.

Розрізняють лінійну, лінійну з відгалуженнями, ієрархічну, мережну та комбіновану структури сайту.

Найпершим в історії видом структур сайту був лінійний. Якщо шукати аналогію, найвлучніше порівняння — це книга. Побудова інтерфейсу нагадує її структуру: користувач послідовно переглядає сторінки, переходячи від однієї до іншої у вигляді логічного ланцюжка. Тобто, можна вийти на головну сторінку, на попередню або на подальшу, але ніяких поділів тут немає. Усі відомості слідують один за одним в звичній послідовності і не діляться на якісь розділи або підпункти. Одним словом, назва лінійного виду структур сайту повністю виправдана, тут дійсно все відбувається в лінійному форматі з однією гілкою. Зазвичай такий тип застосовують для виконання презентації, представлення книги в онлайн-форматі і т.д (рис.3.1).

Рисунок 3.1 – Лінійна структура сайту

Проте є розвиненіший вид структур сайту - лінійний з відгалуженнями. Тут на головній сторінці користувачам пропонується вибір, вони можуть піти по будь-якому необхідному напряму.

Наприклад, на корпоративному веб-ресурсі представлено окремі розділи з інформацією, що орієнтована на різні категорії користувачів — партнерів, клієнтів та постачальників (рис. 3.2).

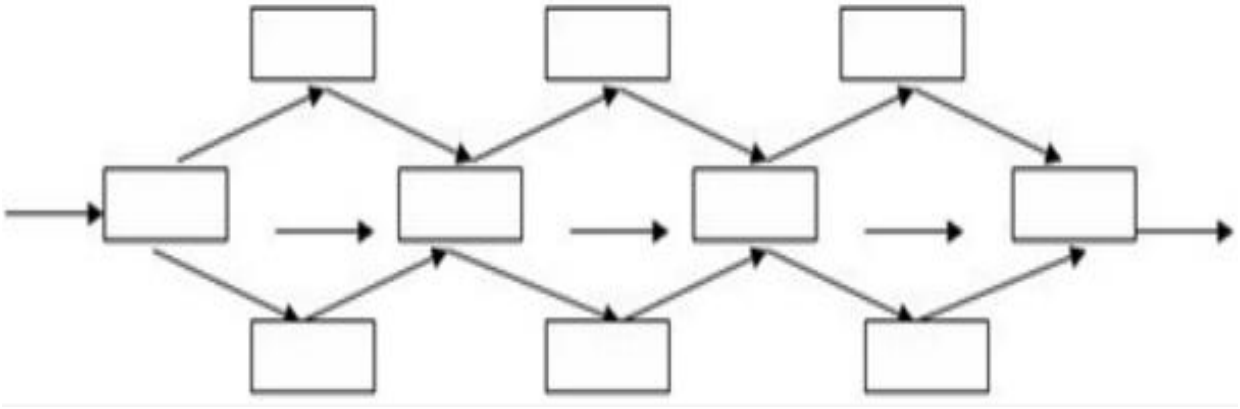

Рисунок 3.2 – Структура сайту лінійна з відгалуженнями

Але якщо говорити про найсучасніший вигляд структур сайту, то тут слід зазначити саме ієрархічний. Ця структура дозволяє укласти величезну безліч розділів, підрозділів і сторінок, які розміщуватимуться в довільній або фіксованій послідовності. Але тут слід правильно розподіляти усі матеріали, щоб в майбутньому не допустити плутанини. Особливо потрібно стежити за навігацією, адже якщо усередині сайту буде безліч "перехресть" і поділів, то треба продумати такі умови, щоб навіть гість, що уперше потрапив, тут не заблукав. Потрібно чітко ділити усю інформацію, а щоб не відбувалося засмічення, треба вчасно видаляти неактуальні відомості в архів (рис. 3.3).

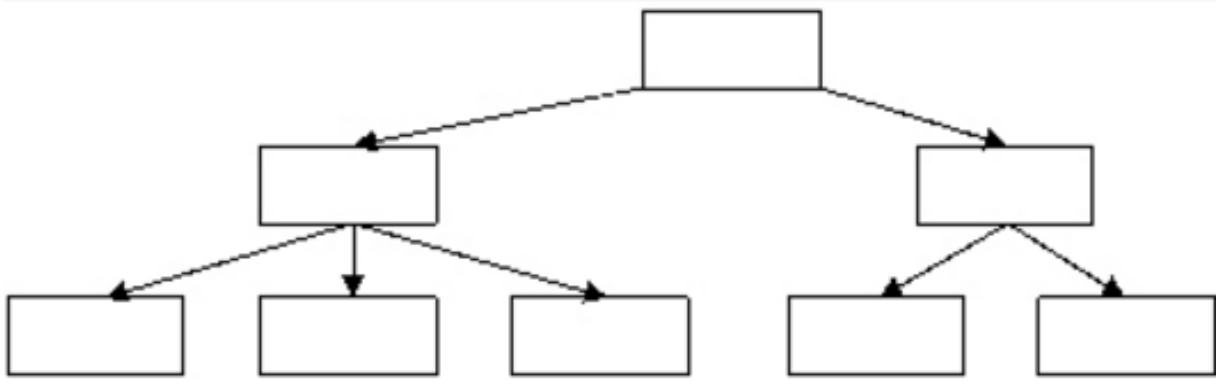

Рисунок 3.3 – Ієрархічна структура

Структура типу «мережа» (рис. 3.4) є однією з найпоширеніших у

середовищі Інтернету. Такий підхід дозволяє здійснювати навігацію між будь-якими сторінками, які мають змістовний або функціональний зв’язок. У подібній організації зв’язки між розділами формуються динамічно – відповідно до поведінки або логіки користувача. Прикладами ресурсів із такою структурою є інтернет-магазини чи масштабні новинні портали.

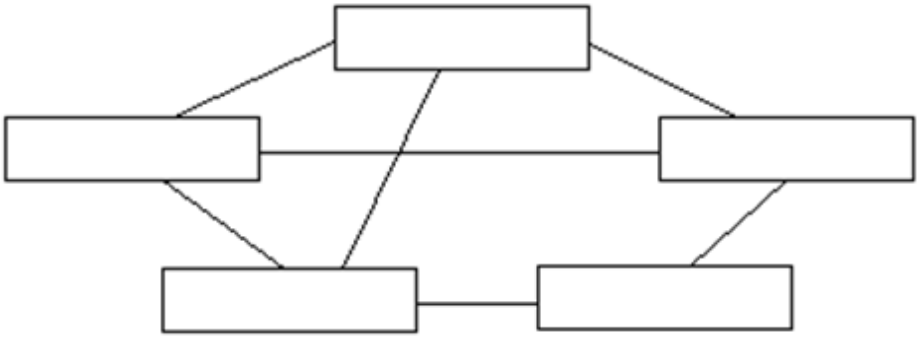

Рисунок 3.4 – Мережева структура

Комбінована структура поєднує елементи кількох типів організації навігації. Наприклад, загальна схема сайту може бути побудована за ієрархічним принципом, але в окремих розділах реалізовувати послідовне виконання дій, що є характерним для лінійної структури.

У процесі розробки веб-сайту була обрана ієрархічно-мережна модель організації, оскільки вона забезпечує високу зручність користування та адаптивність навігації відповідно до потреб відвідувачів (рис. 3.5).

Проектування бази даних.

Процес розробки проєктів, заснованих на архітектурному шаблоні MVC, зазвичай стартує з формування моделей предметної області, оскільки саме вони є центральною складовою будь-якого MVC-додатку. Моделі представляють собою ключові об'єкти, з якими працює система. В контексті інформаційного наповнення системи були визначені наступні основні об'єкти: вистава, театр, зал, ряд, місце та сеанс.

Уся інформація, яку користувач бачить на сайті, зберігається в базі даних. Це дозволяє сайту функціонувати як динамічний ресурс — тобто, його

зміст може оновлюватися в режимі реального часу. Для внесення змін у систему бронювання театральних квитків достатньо скористатися панеллю адміністрування сайту, через яку легко редагується потрібний контент без необхідності втручання в програмний код.

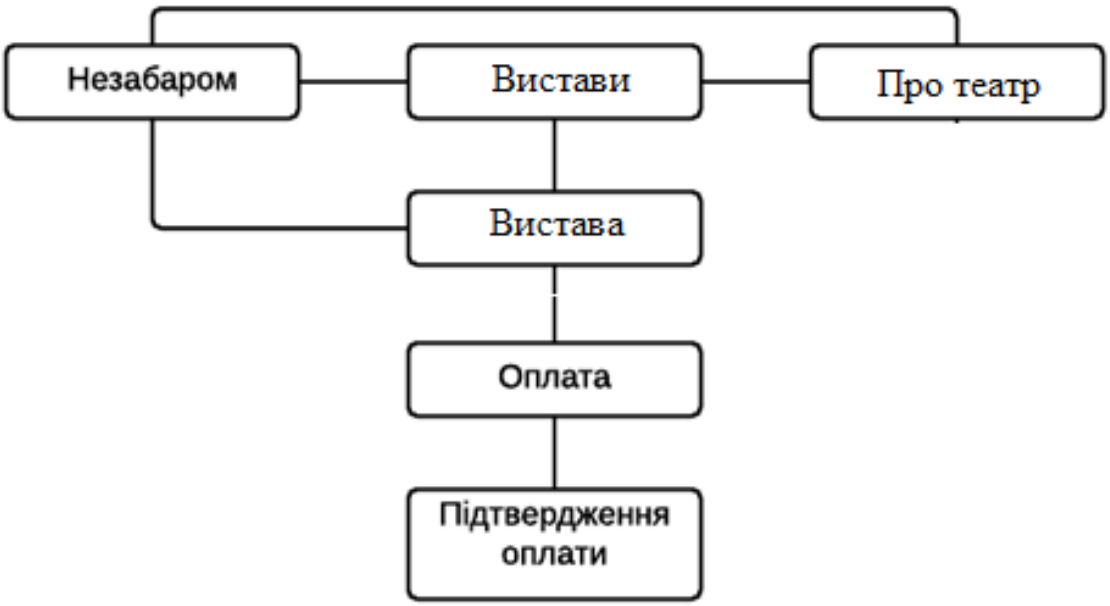

Рисунок 3.5 – Структура сайту

Для забезпечення коректної роботи системи у базі даних необхідно створити відповідні таблиці, які відображають попередньо визначені об'єкти предметної області.

Таблиця «Вистави» містить інформацію, необхідну для створення, редагування та перегляду даних про театральні постановки. У ній зберігаються такі поля.

1. Id — унікальний ідентифікатор кожної вистави, що виконує роль первинного ключа.
2. Name — назва вистави.
3. ReleaseDate — дата першого показу (прем'єри).
4. Duration — тривалість вистави у хвилинах.
5. Actors — перелік провідних акторів, задіяних у постановці.

6. Description — стислий опис змісту вистави, що має викликати інтерес у глядача.
7. ImageData — графічне зображення (постер), збережене у форматі байтового масиву.
8. ImageMimeType — тип зображення, що зберігається (наприклад, PNG, JPEG тощо).

Таблиця «Про театр» включає атрибути, які характеризують театри, в яких відбуваються покази вистав. Основні поля цієї таблиці такі.

1. Id — унікальний ідентифікатор театру (первинний ключ).
2. Name — назва театру.
3. City — населений пункт, у якому розташовано театр.
4. Street — адреса, яка включає назву вулиці та номер будинку.

Таблиця «Зали» призначена для зберігання даних про глядацькі зали в межах конкретних театрів. Вона містить інформацію, необхідну для подальшого формування схеми розсадки та організації перегляду вистав.

Опис атрибутів бази даних.

Таблиця «Зали». Ця таблиця містить дані про зали, у яких проводяться вистави. Її структура включає такі атрибути.

1. Id — унікальний ідентифікатор залу, який виступає первинним ключем. Завдяки йому кожен запис можна чітко відрізнити.
2. Name — назва залу.
3. Seats — кількість місць, що доступні у залі.
4. TheatreId — зовнішній ключ, який пов’язує запис із відповідним театром у таблиці «Про театр».

Таблиця «Ряди». Ця таблиця зберігає інформацію про ряди в межах конкретного залу.

1. Id — ідентифікатор ряду, є унікальним і виступає первинним ключем.
2. Number — номер ряду у залі.

3. SeatsNumber — кількість сидінь у конкретному ряду.
4. HallId — зовнішній ключ, який вказує на зал, до якого належить ряд. Посилається на первинний ключ у таблиці «Зали».

Таблиця «Сеанси». У цій таблиці зберігаються дані про театральні сеанси.

1. Id — унікальний ідентифікатор сеансу (первинний ключ).
2. Date — дата проведення вистави (день, місяць, рік).
3. Time — час початку сеансу.
4. SeatsLeft — кількість вільних місць, які залишилися на момент сеансу.
5. Price — вартість квитка на відповідну подію.
6. HallId — зовнішній ключ, що пов’язує сеанс із конкретним залом.
7. ShowId — зовнішній ключ, який відповідає виставі у таблиці «Вистави».

Таблиця «Місця». Призначена для опису місць у рядах театру та фіксації їхнього бронювання на певні сеанси.

1. Id — унікальний ідентифікатор місця (первинний ключ).
2. Number — номер місця у відповідному ряду.
3. IsBooked — логічний атрибут, що вказує, чи заброньоване це місце.
4. RowId — зовнішній ключ, що вказує на ряд, у якому розташоване місце.
5. SeanceId — зовнішній ключ, що визначає, до якого сеансу належить бронювання місця.

Модель даних: сутність–зв’язок. Щоб описати взаємозв’язки між об’єктами в базі даних, застосовано модель сутність-зв’язок. Ця модель дозволяє формально представити логічні зв’язки між сутностями реального світу. Основу моделі складають сутності (об’єкти, що зберігають дані) та

зв’язки між ними.

Архітектура системи. Для побудови системи бронювання квитків у театрі була використана ієрархічно-мережева структура. Було створено базу даних, яка відображає необхідні таблиці та логічні зв’язки між ними у межах розподіленої системи.

Реалізація доступу до даних. Розробка модуля роботи з базою даних здійснюється з використанням Entity Framework та підходу Code First. На початковому етапі слід підключити бібліотеку Entity Framework до проєкту через NuGet Package Manager.

Install-Package EntityFramework. Далі створюються класи-сутності, які представляють відповідні таблиці, а також клас `DbContext`, що реалізує доступ до бази (рис. 3.6).

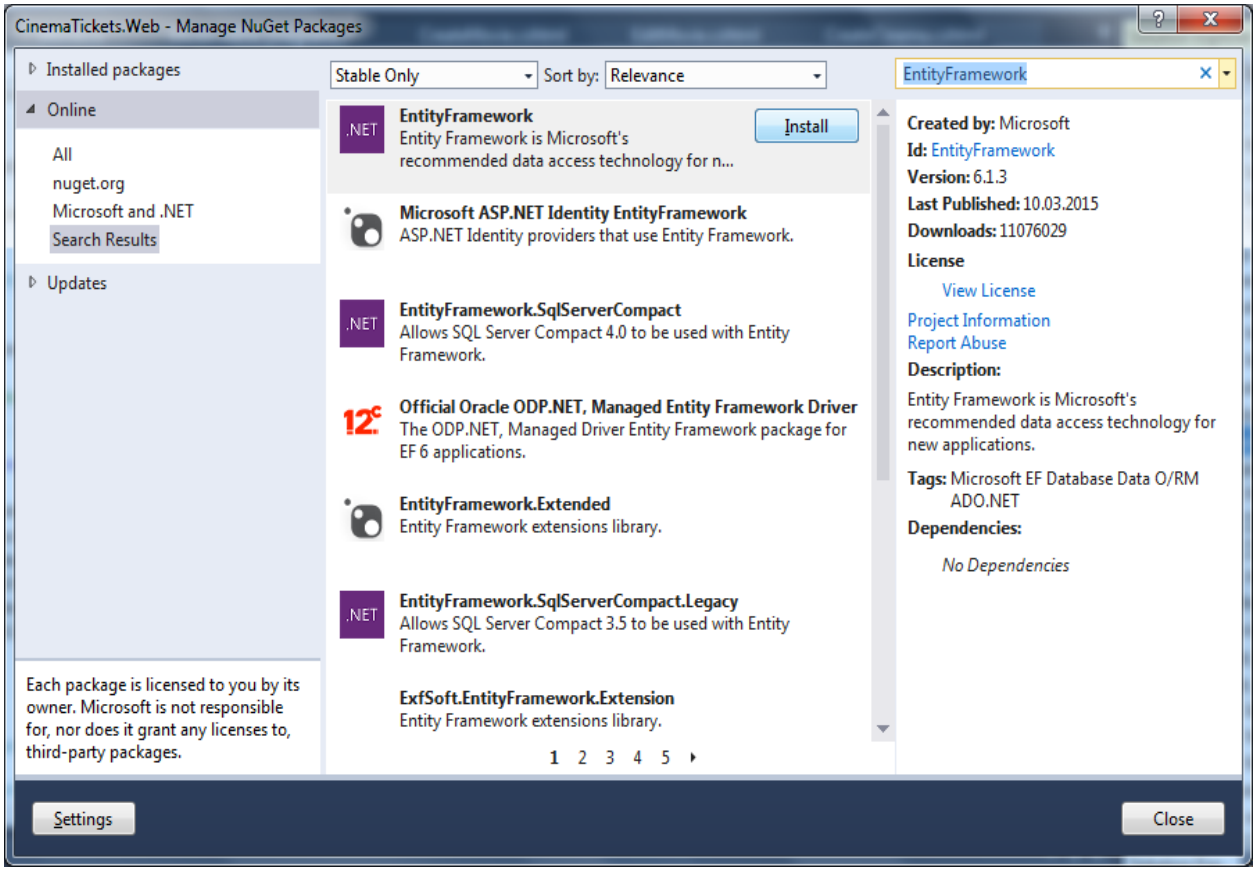

Рисунок 3.6 – Інтерфейс керування пакетами NuGet із результатами пошуку за запитом "EntityFramework"

Після того як бібліотека Entity Framework була додана до проєкту, наступним етапом є визначення класів, які будуть відображати сутності, що зберігатимуться в базі даних. Клас у цьому контексті виступає як логічна одиниця, яка дозволяє формувати власні типи, об’єднуючи поля, методи та події під одним ім’ям.

Окрім стандартних властивостей, що відповідають колонкам таблиць бази даних, у створені класи також включаються навігаційні властивості. Ці властивості забезпечують доступ до пов’язаних даних через об'єкти і дозволяють працювати з іншими сутностями, пов’язаними зовнішніми ключами. Варто зазначити, що самі навігаційні властивості безпосередньо в базі даних не зберігаються — вони є частиною логіки моделі даних.

Один із популярних механізмів взаємодії з пов’язаними даними — це ліниве завантаження (lazy loading).

При використанні цього підходу об’єкти, пов’язані між собою, завантажуються з бази лише в момент звернення до них, що дозволяє зменшити початкове навантаження на систему. Однак для коректної роботи цього механізму слід дотримуватись певних умов: класи мають бути публічними, а їхні навігаційні властивості — оголошеними з модифікаторами public та virtual [30].

У кожному застосунку, який використовує Entity Framework для роботи з базою даних, необхідно створити спеціальний клас, що наслідує DbContext. Такий клас виступає як центральний елемент для взаємодії з базою та дозволяє отримати доступ до таблиць за допомогою властивостей типу DbSet.

У межах розробки було створено клас EFContext, який успадковується від DbContext. Саме він виконує роль основного зв’язку між програмною логікою та базою даних, забезпечуючи доступ до таблиць та керування операціями з даними:

```
public class EFContext : DbContext
    {
        public EFContext() : base("TheatreTicketsDB")
        {   }
        protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<Hall>()
                .HasMany(s => s.Seances)
                .WithRequired(s => s.Hall)
                .HasForeignKey(s => s.HallId)
                .WillCascadeOnDelete(false);
        }
```

У нашому проєкті, під час реалізації класу контексту бази даних, було здійснено перевизначення методу OnModelCreating, який належить до базового класу DbContext. Цей метод дозволяє вручну налаштовувати конфігурацію моделі та змінювати стандартну поведінку Entity Framework при використанні підходу Code First.

Наступним кроком після створення класу контексту є налаштування підключення до бази даних. Це здійснюється шляхом додавання елемента add у секцію connectionStrings конфігураційного файлу, де вказується рядок з параметрами для підключення до необхідної БД [31].

Отже, для розробки ресурсу використовуємо C# , ASP.NET MVC, Entity Framework та SQL Server

3.2 Створення web-ресурсу

Розробка модуля замовлення квитків. Однією з головних функцій розподіленої системи замовлення білетів для театру є функція замовлення

квитків.

Процес бронювання квитка починається з вибору клієнтом потрібної вистави. Для цього користувач має перейти до сторінки з інформацією про виставу або переглянути афішу певного театру, де він може натиснути на посилання із зазначеним часом початку сеансу. Після цього система автоматично перенаправить його на сторінку оформлення замовлення, приклад якої наведено на рисунку 3.7.

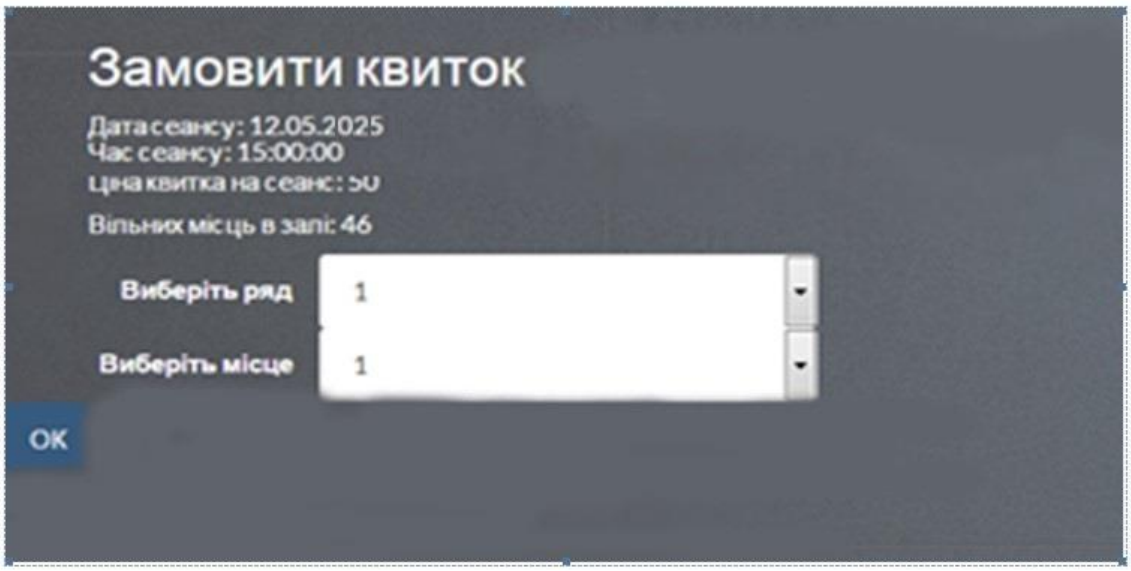

Рисунок 3.7 – Сторінка замовлення квитка на сеанс

На цій сторінці відображається повна інформація про вибраний сеанс: дата та час проведення, назва вистави, вартість квитка, загальна кількість місць у залі, а також кількість доступних для бронювання місць.

У даному фрагменті інтерфейсу формується елемент select, який містить список доступних місць, згенерований на основі переданих даних. Для отримання цього списку застосовується функція .ajax із бібліотеки jQuery. Вона відправляє AJAX-запит до відповідного методу контролера, який здійснює запит до бази даних та повертає масив об’єктів типу «Місце». Код, що реалізує зазначену функціональність, подано нижче [32]:

```
$('#row').change(function () {
        var id = $(this).val();
        |       $.ajax({
            type: 'GET',
            url: '@Url.Action("GetSeatsForRow")' +
'?seanceId='+@Model.Id+'&rowId='+id,
            success: function (data) {
                $('#seat').replaceWith(data);
            }
        });
    });
```

Кожному користувачеві має бути призначено власний екземпляр об'єкта замовлення, який зберігається між запитами. Оскільки об'єкти, пов'язані із сеансом, автоматично видаляються після завершення його життєвого циклу (наприклад, у разі бездіяльності користувача протягом певного часу), немає необхідності вручну керувати збереженням чи видаленням об'єктів типу Order.

У методі AddTicket відбувається наповнення об’єкта Order інформацією про поточне бронювання. Далі користувач автоматично перенаправляється на сторінку Checkout, де йому потрібно ввести контактні дані. На цьому етапі обов’язковим є лише введення адреси електронної пошти, на яку буде доставлено електронний квиток. Після введення даних та підтвердження дії (натискання кнопки «ОК»), керування передається методу Checkout, що знаходиться в OrderController.

Для генерації квитка у форматі PDF використовувалась безкоштовна бібліотека iTextSharp. Щоб створити PDF-документ у пам’яті, необхідно ініціалізувати об’єкт Document, який є основною структурною одиницею при

роботі з файлами у цій бібліотеці [33].

Для початку повноцінної роботи з PDF-документом необхідно виконати його відкриття, після чого можна додати новий елемент вмісту, а завершальним етапом є закриття документа:

```
doc1.Open();
doc1.Add(new Paragraph("My first PDF"));
doc1.Close();
```

Наступним етапом є ініціалізація об'єкта класу `MailMessage`, в якому встановлюється тема листа та його вміст. Для прикріплення файлу до листа застосовується об'єкт класу `Attachment`.

Після формування повідомлення його можна надіслати клієнту за допомогою методу `Send`, який належить об'єкту класу `SmtpClient`.

Розробка модуля авторизації.

У вже реалізованому фрагменті розподіленої системи досі не було впроваджено механізм авторизації. На поточному етапі будь-який користувач має змогу відкрити сторінки, що забезпечують доступ до функцій роботи з базою даних. Це створює ризик того, що сторонні особи можуть змінювати дані, що може негативно вплинути на стабільність функціонування системи. Щоб запобігти таким загрозам, було прийнято рішення впровадити модуль авторизації, який би забезпечував захист від несанкціонованого доступу.

Після встановлення необхідних бібліотек потрібно внести зміни у файл `Web.config`, додавши новий рядок підключення у секцію `connectionStrings`. Це необхідно для налаштування зв'язку з базою даних, у якій зберігатиметься інформація про зареєстрованих користувачів та інші пов’язані дані. Приклад зміненої секції з підключенням до бази `Identity` наведено на рисунку 3.8.

```
<connectionStrings>
  <add name="IdentityDb" providerName="System.Data.SqlClient"
       connectionString="Data Source=(localdb)\v11.0; AttachDbFilename='|DataDirectory|\IdentityDb.mdf';
       Integrated Security=True; Connect Timeout=15; Encrypt=False; TrustServerCertificate=False;
       MultipleActiveResultSets=True" />
  <add name="CinemaTicketsDB" providerName="System.Data.SqlClient"
       connectionString="Data Source=(LocalDB)\v11.0; AttachDbFilename='|DataDirectory|\CinemaTicketsDatabase.mdf';
       Integrated Security = true" />
</connectionStrings>
```

Рисунок 3.8 – Рядок підключення до бази даних Identity у файлі Web.config

У розділі `appSettings` додається новий ключ, який вказує, який клас має використовуватись OWIN-інфраструктурою під час запуску для конфігурації:

```xml
<add key="owin:AppStartup" value="TheatreTickets.Web.IdentityConfig" />
```

Подальшим кроком у впровадженні ASP.NET Identity є створення класу користувача, в якому зберігається інформація про обліковий запис. Після цього створюється контекст бази даних, який використовує `Entity Framework` та працює з вищезгаданим класом. Це дозволяє застосувати підхід Code First для створення схеми бази даних і керування нею.

Завершальним етапом конфігурації є визначення стартового класу, що містить метод `Configuration`, який викликається системою OWIN. У цей метод передається об’єкт, що реалізує інтерфейс `Owin.IAppBuilder`, який забезпечує налаштування необхідних компонентів.

Метод `CreatePerOwinContext` відповідає за створення нових екземплярів класів `AppUserManager` та `AppIdentityDbContext` для кожного запиту [34].

Після налаштування інфраструктури можна переходити до створення облікових записів користувачів. Для цього створюється контролер `AdminController` з дією `Create`. Для виводу списку зареєстрованих

користувачів реалізується метод `Index`. Для обох дій створюються відповідні представлення.

Для створення нового облікового запису не обов'язково використовувати всі поля, які надає `IdentityUser`. Було прийнято рішення створити спеціальний клас-модель для представлення `Create`, який містить лише найнеобхідніші поля: ім’я користувача, email та пароль. Всі вони позначені атрибутом `[Required]`, що робить їх обов’язковими для заповнення. Реєстрація користувача відбувається через об'єкт `AppUserManager`.

Під час створення першого користувача система автоматично формує наступні таблиці:

1. `AspNetRoles` — таблиця для зберігання ролей.
2. `AspNetUserClaims` — зберігає список клеймів (claims) користувача.
3. `AspNetUserLogins` — містить дані про логіни з використанням зовнішніх сервісів.
4. `AspNetUserRoles` — встановлює зв’язок між користувачами та їхніми ролями.
5. `AspNetUsers` — основна таблиця, що містить облікові записи.

Щоб обмежити доступ до функцій, пов’язаних із базою даних, необхідно застосовувати атрибут `[Authorize]` до відповідних методів. Якщо доступ потрібно обмежити для більшості методів контролера, доцільно вказати атрибут на рівні всього класу. Методи, які повинні залишатися доступними для неавторизованих користувачів, позначаються атрибутом `AllowAnonymous`.

Крім того, контролер `AdminController` також було захищено за допомогою атрибуту `[Authorize]`, що гарантує доступ до адміністрування тільки для авторизованих користувачів. Для уникнення ситуації, коли неможливо створити адміністратора після встановлення обмежень, кілька

облікових записів було створено до активації перевірки прав доступу.

У випадку, коли користувач без авторизації намагається перейти до сторінки з обмеженим доступом, система автоматично перенаправляє його на сторінку входу. Приклад такої сторінки показано на рисунку 3.9.

Після реалізації всіх необхідних модулів формується повноцінна система. Зовнішній вигляд прикладу вебсайту представлено на рисунку 3.10.

Рисунок 3.9 – Сторінка входу в систему

Рисунок 3.10 – Інтерфейс вебзастосунку

3.3 Тестування web-ресурсу

Тестування програмного забезпечення — це процес оцінки працездатності розробленої програми як у статичному режимі (аналіз коду, перегляди, налагодження), так і в динамічному (з використанням тестових даних), з метою перевірки логіки виконання та співставлення фактичних результатів із очікуваними.

Перевірка вебсайту є складовою частиною забезпечення якості. Вона полягає у верифікації відповідності фактичної поведінки ресурсу передбаченій, використовуючи набір заздалегідь визначених тестів. Такий підхід дає можливість виявити логічні помилки, а також порушення в коректності відображення або функціонуванні елементів сайту.

Основні рівні тестування.

- Системне тестування охоплює всю систему в цілому, оцінюючи її відповідність до функціональних та нефункціональних вимог.
- Інтеграційне тестування спрямоване на оцінку взаємодії між окремими компонентами системи або підсистемами. Часто реалізується поетапно: до вже протестованих модулів поступово додаються нові.
- Модульне тестування (unit testing) передбачає перевірку найменших елементів системи — окремих методів чи класів. Зазвичай цим видом тестування займаються самі розробники під час реалізації функціоналу.

Під час розробки веб-застосунку були реалізовані модульні тести, які перевіряли:

- правильну реалізацію розрахунків та завершення замовлень.
- правильність редагування та видалення даних;
- коректність додавання квитків до кошика;

Операції редагування та видалення записів у базі даних мають

критичне значення, тому вони були піддані окремому тестуванню. Приклад реалізації тестів цих дій наведено на рисунку 3.11.

```
[TestMethod]
0 references
public async Task Can_Delete_Valid_Movies()
{
    Movie movie = new Movie {Id = 2, Name = "Test" };
    Mock<IUnitOfWork> mock = new Mock<IUnitOfWork>();
    mock.Setup(m => m.Movies.GetAll()).Returns(new Movie[] {
        new Movie {Id = 1, Name = "P1"},
        movie
    });

    MovieController target = new MovieController(mock.Object);
    await target.DeleteMovie(movie.Id);
    mock.Verify(m => m.Movies.Delete(movie));
}
```

```
public async Task Can_Edit_Movie()
{
    Mock<IUnitOfWork> mock = new Mock<IUnitOfWork>();
    mock.Setup(m => m.Movies.GetAll()).Returns(new Movie[]
    {
        new Movie {Id = 1, Name = "1"},
        new Movie {Id = 2, Name = "2"}
    });
    // Arrange - create the controller
    MovieController target = new MovieController(mock.Object

    Movie m1 = (Movie)(await target.EditMovie(1)).ViewData.M
    Movie m2 = (Movie)(await target.EditMovie(2)).ViewData.M
    Assert.AreEqual(1, m1.Id);
    Assert.AreEqual(2, m2.Id);
}
```

Рисунок 3.11 – Тести для перевірки редагування і видалення записів

Для перевірки процедури додавання квитків до замовлення було створено окремий тест, показаний на рисунку 3.12.

```
[TestMethod]
0 references
public void Can_Add_New_Lines()
{
    Ticket p1 = new Ticket { Row = 1, Seat = 3 };
    Ticket p2 = new Ticket { Row = 2, Seat = 15 };
    Order target = new Order();
    target.AddTicket(p1);
    target.AddTicket(p2);
    Ticket[] results = target.OrderedTickets.ToArray();
    // Assert
    Assert.AreEqual(results.Length, 2);
    Assert.AreEqual(results[0], p1);
    Assert.AreEqual(results[1].Seat, p2.Seat);
}
```

Рисунок 3.12 – Тест для перевірки додавання квитка до кошика

Результат виконання тесту вказує на успішне додавання квитка до замовлення (рис.3.13).

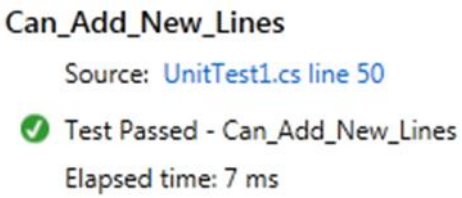

Рисунок 3.13 – Успішне виконання тесту додавання квитка

Окрема увага приділялася перевірці процедури розрахунку, яка є ключовою для системи. Тест, який відповідає за валідацію цього процесу, представлений на рисунку 3.14. У цьому тесті перевіряється, чи правильно здійснюється перенаправлення до сторінки `Completed` у випадку успішної обробки.

Виконання цього тесту підтвердило коректність функціоналу (рис.3.15).

Усі тести створювались у процесі розробки за допомогою інструментів модульного тестування. Оскільки всі перевірки завершились успішно, це свідчить про правильність реалізації ключових компонентів системи.

3.4 Висновок до розділу 3

У результаті проведених досліджень і розробки обґрунтовано вибір мови програмування, створено WEB-ресурс театральної установи та проведено його успішне тестування. WEB-ресурс реалізує систему управління театральними виставами, залами, сеансами та бронюванням місць. Тестування підтвердило правильність методів та ідей, використаних у розробці.

```
[TestMethod]
0 references
public void Can_Checkout_And_Submit_Order()
{
    Mock<IOrderProcessor> mock = new Mock<IOrderProcessor>();
    Mock<IRepository<Seat>> mock_seats = new Mock<IRepository<Seat>>();
    Mock<IRepository<Cinema>> mock_cinema = new Mock<IRepository<Cinema>>();
    Mock<IRepository<Movie>> mock_movie = new Mock<IRepository<Movie>>();
    Mock<IRepository<Seance>> mock_seance = new Mock<IRepository<Seance>>();
    Mock<IRepository<Hall>> mock_hall = new Mock<IRepository<Hall>>();
    Mock<IUnitOfWork> mock_uow = new Mock<IUnitOfWork>();
    mock_cinema.Setup(m => m.GetAll()).Returns(new Cinema[] { new Cinema { Id = 1, Name = "C" } });
    mock_movie.Setup(m => m.GetAll()).Returns(new Movie[] { new Movie { Id = 1, Name = "M" } });
    mock_seance.Setup(m => m.GetAll()).Returns(new Seance[] {new Seance { Id = 1, HallId = 1, SeatsLeft = 34
    mock_hall.Setup(m => m.GetAll()).Returns(new Hall[] {new Hall { Id = 1, CinemaId = 1, Name = "H" }});
    mock_uow.Setup(m => m.Halls).Returns(mock_hall.Object);
    mock_uow.Setup(m => m.Cinemas).Returns(mock_cinema.Object);
    mock_uow.Setup(m => m.Movies).Returns(mock_movie.Object);
    mock_uow.Setup(m => m.Seances).Returns(mock_seance.Object);
    OrderController target = new OrderController(mock_uow.Object, mock.Object);
    Order order = new Order();
    Seance seance = new Seance { Id = 1, HallId = 1, SeatsLeft = 34, MovieId = 1 };
    order.AddTicket(new Ticket { Row = 1, Seat = 1, Seance = seance });
    ViewResult result = target.Checkout(order, new CustomerDetails());
    mock.Verify(m => m.ProcessOrder(It.IsAny<Order>(), It.IsAny<CustomerDetails>()),
        Times.Once());
    Assert.AreEqual("Completed", result.ViewName);
    Assert.AreEqual(true, result.ViewData.ModelState.IsValid);
}
```

Рисунок 3.14 – Тест для перевірки операції розрахунку

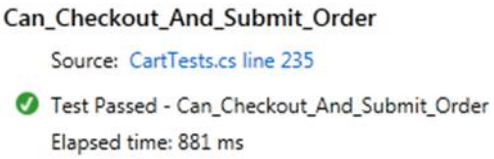

Рисунок 3.15 – Позитивний результат тесту розрахунку

ВИСНОВКИ

Поставлені перед бакалаврською кваліфікаційною роботою задачі виконано в повному обсязі [2], а саме:

- досліджено вже наявні аналогічні рішення;
- обрано та обґрунтовано підхід до вирішення поставленого завдання;
- розроблено структури та дизайн WEB-ресурсу театральної установи;
- створено програму WEB-ресурсу театральної установи;
- протестовано та перевірено функціонування ресурсу;
- створено користувацьку інструкцію з експлуатації WEB-ресурсу.

В ході виконання бакалаврської роботи були проаналізовані найпопулярніші системи для розборки WEB-ресурсу театральної установи. Відповідно до завдання було проведено аналіз проблеми та обґрунтування необхідності даної розробки. Розроблено структуру та систему навігації сайту. Наступним етапом проектування було створення інформаційного забезпечення, одного із важливих етапів, який відповідає за наявність на сайті якісного контенту, що призводить до популярності сторінки серед користувачів. Відповідно до всіх вимог було проведено розробку графічного забезпечення, що включало в себе розробку графіки та анімації а саме: текстур, логотипів, «шапки» сайту, різноманітних розділювачів та інших елементів дизайну. Було проаналізовано різні типи структур веб сайтів, обрано і розроблено оптимальну структуру веб ресурсу та розроблено алгоритми функціонування веб ресурсу. Відповідно з завданням до бакалаврської роботи був розроблений WEB-ресурс театральної установи. Також було проведено тестування, яке підтвердило роботоспроможність WEB-ресурсу. Мета роботи - розширення функціональних можливостей WEB-ресурсу театральної установи за рахунок додавання функії замовлення квитків та створення дружнього і комфортного інтерфейса досягнута.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. О.М. Тітарчук, В.О. Денисюк. Розробка веб-додатку театральної установи. LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2025). Вінницький національний технічний університет. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23947/19750 .
2. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» [Електронний ресурс] / уклад.: А. А. Яровий, О. В. Сілагін, І. В. Богач. Вінниця: ВНТУ, 2023. (PDF, 54 с.). URL: https://iq.vntu.edu.ua/method/getfile.php?fname=124285.pdf&x=1&card_id=46522&id=124285 .
3. Що таке зручний вебсайт та як його зробити у 2024. URL: https://wizeclub.education/blog/shho-take-zruchnij-vebsajt-ta-yak-jogo-zrobiti-u-2024/
4. Юзабіліті сайту. URL: https://webtune.com.ua/statti/web-rozrobka/yuzabiliti-sajtu/
5. UX/UI дизайн для конверсій: Як зробити сайт зручним і прибутковим. URL: https://megasite.ua/ua/uxui-dizayn-dlya-konversiy-kak-sdelat-sayt-udobnim-i-pribilnim?utm_source=chatgpt.com
6. Особливості веб-сайтів, проведення аудиту та ключові аспекти юзабіліті. URL: https://tto-studio.com.ua/osoblyvosti-veb-saytiv-provedennya-audytu-ta-klyuchovi-aspekty-yuzabiliti/?utm_source=chatgpt.com
7. Юзабіліті сайту і чекліст по ним. URL: https://cityhost.ua/uk/blog/ yuzabiliti-sayta-i-cheklist-po-nim.html?utm_source=chatgpt.com
8. Омбудсмен розповів, чи дотримуються мовного закону українські театри. URL: https://novynarnia.com/2023/10/17/ombudsmen-rozpoviv-chy-dotrymuyutsya-movnogo-zakonu-ukrayinski-teatry/?utm_source= chatgpt.com

9. МКІП займається діджіталізацією українських театрів. URL: https://shotam.info/mkip-zaymaietsia-didzhitalizatsiieiu-ukrainskykh-teatriv/?utm_source=chatgpt.com
10. Web Content Accessibility Guidelines (WCAG) 2.1. URL: https://www.w3.org/Translations/WCAG21-ua/
11. Найпопулярніші театри. URL:https://www.dramox.com.ua/teatry
12. Національна опера України. URL: http://opera.com.ua .
13. Львівська національна опера. URL:http://opera.lviv.ua .
14. Одеський національний театр опери та балету. URL: http://operahouse.od.ua
15. Харківський національний театр опери та балету. URL: http://hatob.com.ua/ukr/
16. Дніпровський театр драми та комедії. URL: http://dramicom.dp.ua
17. Сумський театр драми і музкомедії. URL: http://musicdrama.com.ua
18. Запорізький театр юного глядача. URL: http://teatrmolodi.zp.ua
19. Вінницький театр ім. М. Садовського. URL: https://teatr.vn.ua/
20. Тернопільський драмтеатр ім. Шевченка. URL: https://www.theatre.te.ua/
21. Закарпатський театр братів Шерегіїв. URL: http://dramteatr.uz.ua.
22. Романюк О.Н. Основи WEB-дизайну. URL: https://iq.vntu.edu.ua/method/getfile.php?fname=146471.doc&x=1&card_id =71888&id=146471 .
23. ASP.NET MVC Pattern. URL: https://dotnet.microsoft.com/en-us/apps/aspnet/mvc
24. Symfony. URL: https://symfony.com/ .
25. Compress the complexity of modern web apps. URL: https://rubyonrails.org/
26. Express. URL: https://expressjs.com/
27. Christian Nagel, Jay Glynn, Morgan Skinner. Professional C# 5.0 and .NET 4.5.1. URL: https://www.tahlildadeh.com/files/professional_csharp_5.0_and_dotnet_4.5.1.pdf

28. Романюк О. Н. Організація баз даних і знань. / О.Н. Романюк, Т.О. Савчук /Навчальний посібник. URL: https://pdf.lib.vntu.edu.ua/books/2016/Romanuk_2001_123.pdf .
29. Берко А.Ю., Верес О.М., Пасічник В.В. Системи баз даних та знань. Книга 2. Системи управління базами даних та знань: навч. посібник. Львів : ПП «Магнолія2006», 2018. 584 с.
30. Adam Freeman. Pro ASP.NET Core 7, Tenth Edition 10th ed. Edition. Apress, 2023. 1256. URL: https://ebin.pub/qdownload/pro-aspnet-core-7-10nbsped-1633437825-9781633437821-z-7147640.html .
31. Jess Chadwick. Programming ASP.NET MVC 4 URL: https://anhtan1987.wordpress.com/wp-content/uploads/2014/12/programming-asp-net-mvc-4.pdf
32. jQuery Підручник. URL: https://w3schoolsua.github.io/jquery/index.html#gsc.tab=0
33. Create PDFs in ASP.NET - getting started with iTextSharp. URL: http://www.mikesdotnetting.com/article/80/create-pdfs-in-asp-net-getting-started-with-itextsharp
34. Jon Galloway. Professional ASP.NET MVC 5. URL: https://library.uniq.edu.iq/storage/books/file/Professional%20Asp.Net%20Mvc%205/1667803069Professional%20ASP.NET%20MVC%205%20(Jon%20Galloway,%20Brad%20Wilson,%20K.%20Scott%20Allen%20etc.)%20(z-lib.org).pdf

ДОДАТОК А (ОБОВ'ЯЗКОВИЙ)
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка WEB-ресурсу театральної установи

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 2,12 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- ✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту
- ☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- ☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Денисюк В.О.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Тітарчук О.М.
(підпис) (прізвище, ініціали)

ДОДАТОК В (ОБОВ'ЯЗКОВИЙ)

ЛІСТИНГ ПРОГРАМИ

```
    public class Show
    {
        public int Id { get; set; }
        public string Name { get; set; }
        [DataType(DataType.Date)]
        public DateTime ReleaseDate { get; set; }
        public int Duration { get; set; }
        public string Genre { get; set; }
        public string Director { get; set; }
        public string Description { get; set; }
        public byte[] ImageData { get; set; }
        public string ImageMimeType { get; set; }
        public virtual ICollection<Seance> Seances { get; set; }
        public Show()
        {Seances = new List<Seance>();}
    }
public class Theatre
    {
        [HiddenInput(DisplayValue=false)]
        public int Id { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        public string City { get; set; }
        [Required]
        public string Street { get; set; }
        public virtual ICollection<Hall> Halls { get; set; }
        public Theatre()
        {    Halls = new List<Hall>();}
    }
public class Hall
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public int Seats { get; set; }
        public virtual ICollection<Seance> Seances { get; set; }
        public virtual ICollection<Row> Rows { get; set; }
[HiddenInput(DisplayValue = false)]
        public int TheatreId { get; set; }
        public Theatre Theatre { get; set; }
        public Hall()
        {
            Seances = new List<Seance>();
            Rows = new List<Row>();
        }
    }
public class Row{
        public int Id { get; set; }
        public int Number { get; set; }
        public int SeatsNumber { get; set; }
        public virtual ICollection<Seat> Seats { get; set; }
```

```
        public int HallId { get; set; }
        public virtual Hall Hall { get; set; }
        public Row()
        { Seats = new List<Seat>();}
    }
    public class Seat{
        public int Id { get; set; }
        public int Number { get; set; }
        public bool IsBooked { get; set; }
        public int RowId { get; set; }
        public virtual Row Row { get; set; }
        public int SeanceId { get; set; }
        public virtual Seance Seance { get; set; }
    }
    public class Seance{
        public int Id { get; set; }
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:dd'/'MM'/'yyyy}", ApplyFormatInEditMode
= true)]
        public DateTime Date { get; set; }
        [DataType(DataType.Time)]
        [DisplayFormat(DataFormatString = "{0:HH:mm}", ApplyFormatInEditMode = true)]
        public TimeSpan Time { get; set; }
        public int SeatsLeft { get; set; }
        public int Price { get; set; }
        public int HallId { get; set; }
        public virtual Hall Hall { get; set; }
        public int ShowId { get; set; }
        public virtual Show Show { get; set; }
        public virtual ICollection<Seat> Seats { get; set; }
        public Seance()
        { Seats = new List<Seat>(); }
    }
    public interface IRepository<TEntity> where TEntity : class{
        ICollection<TEntity> GetAll();
        TEntity GetById(int? id);
        TEntity Find(Expression<Func<TEntity, bool>> match);
        void Edit(TEntity entity, int id);
        void Insert(TEntity entity);
        Task<ICollection<TEntity>> GetAllAsync();
        Task<TEntity> GetByIdAsync(int? id);
        Task<TEntity> FindAsync(Expression<Func<TEntity, bool>> match);
        Task<ICollection<TEntity>> FindAllAsync(Expression<Func<TEntity, bool>>
predicate);
        Task EditAsync(TEntity entity, int id);
        Task InsertAsync(TEntity entity);
        Task<int> DeleteAsync(TEntity entity);
        int Delete(TEntity entity);
    }
public interface IUnitOfWork{
        IRepository<Theatre> Theatres
        {get; }
        IRepository<Hall> Halls
        {get; }
        IRepository<Show> Shows
        {get; }
        IRepository<Row> Rows
        {get; }
        IRepository<Seance> Seances
```

```
        {get; }
        IRepository<Seat> Seats
        {get; }
    }
public class EFContext : DbContext {
        public EFContext() : base("TheatreTicketsDB"){}
        protected override void OnModelCreating(DbModelBuilder modelBuilder){
            base.OnModelCreating(modelBuilder);
            modelBuilder.Entity<Hall>()
                .HasMany(s => s.Seances)
                .WithRequired(s => s.Hall)
                .HasForeignKey(s => s.HallId)
                .WillCascadeOnDelete(false);

        }
        public DbSet<Theatre> Theatres { get; set; }
        public DbSet<Hall> Halls { get; set; }
        public DbSet<Show> Shows { get; set; }
        public DbSet<Seance> Seances { get; set; }
        public DbSet<Seat> Seats { get; set; }
        public DbSet<Row> Rows { get; set; }
    }
public class GenericRepository<TEntity> : IRepository<TEntity> where TEntity : class
{
        protected EFContext context;// = new EFContext();
        protected SemaphoreSlim mutex = new SemaphoreSlim(1);
        public GenericRepository(){
            context = new EFContext();
        }
        public GenericRepository(EFContext context)
        {
            this.context = context;
        }
        public ICollection<TEntity> GetAll()
        {
            try{
                return context.Set<TEntity>().ToList();
            }catch (SqlException e)
            { throw e; }        }

        public async Task<ICollection<TEntity>> GetAllAsync()
        {
            await mutex.WaitAsync().ConfigureAwait(false);
            try{
                return await context.Set<TEntity>().ToListAsync();
            }
            catch (SqlException e)
            { throw e; }            finally
            {
                mutex.Release();
            }
        }
        public async Task<TEntity> GetByIdAsync(int? id)
        {
            await mutex.WaitAsync().ConfigureAwait(false);
            try
            {
                TEntity result = await context.Set<TEntity>().FindAsync(id);
                return result;
            }
```

```
            catch (SqlException e)
            { throw e; }            finally
            {
                mutex.Release();
            }
        }
        public TEntity GetById(int? id)
        {
            try
            {
                TEntity result = context.Set<TEntity>().Find(id);
                return result;
            }
            catch (SqlException e)
            { throw e; }        }
        public async Task<TEntity> FindAsync(Expression<Func<TEntity, bool>>
predicate)
public static class NinjectWebCommon
    {
        private static readonly Bootstrapper bootstrapper = new Bootstrapper();
        public static void Start()
        {
            DynamicModuleUtility.RegisterModule(typeof(OnePerRequestHttpModule));
            DynamicModuleUtility.RegisterModule(typeof(NinjectHttpModule));
            bootstrapper.Initialize(CreateKernel);
        }
        public static void Stop()
        {
            bootstrapper.ShutDown();
        }
        private static IKernel CreateKernel()
        {
            var kernel = new StandardKernel();
            try
            {
                kernel.Bind<Func<IKernel>>().ToMethod(ctx => () => new
Bootstrapper().Kernel);

kernel.Bind<IHttpModule>().To<HttpApplicationInitializationHttpModule>();

                RegisterServices(kernel);
                return kernel;
            }
            catch
            {
                kernel.Dispose();
                throw;
            }
        }
        private static void RegisterServices(IKernel kernel)
        {
            System.Web.Mvc.DependencyResolver.SetResolver(
              new
TheatreTickets.Web.Infrastructure.NinjectDependencyResolver(kernel));
        }
    }
```

ДОДАТОК В (ОБОВ'ЯЗКОВИЙ)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА WEB-РЕСУРСУ ТЕАТРАЛЬНОЇ УСТАНОВИ

Виконав: студент 4-го курсу,
групи 3КН-22б
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Тітарчук О.М.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Денисюк В.О.
(прізвище та ініціали)

«03» червня 2025 р.

Вінниця ВНТУ - 2025 рік

```
/ (корінь)
├── index.html (або index.php)
├── /css/
├── /js/
├── /images/
├── /includes/ (шаблони, хедери, футери)
├── /pages/ (окремі сторінки)
├── /admin/ (якщо є)
└── /api/ (якщо REST)
```

Рисунок В.1 – Приклад структури папок розробки

```
/
├── index.php                  # Головна сторінка з
├                              # афішею
├── about.php                  # Про театр
├── schedule.php               # Афіша (розклад)
├── play.php?id=               # Сторінка конкретної
├                              # вистави
├── tickets.php                # Продаж квитків
├── contact.php                # Контакти
├── news.php                   # Новини
├── /admin/                    # Панель
├                              # адміністрування
├── /css/
├── /js/
├── /images/
└── /db/                       # Підключення до бази
                               # MySQL/PostgreSQL)
```

Рисунок В.2 - Приклад основної структури сайту

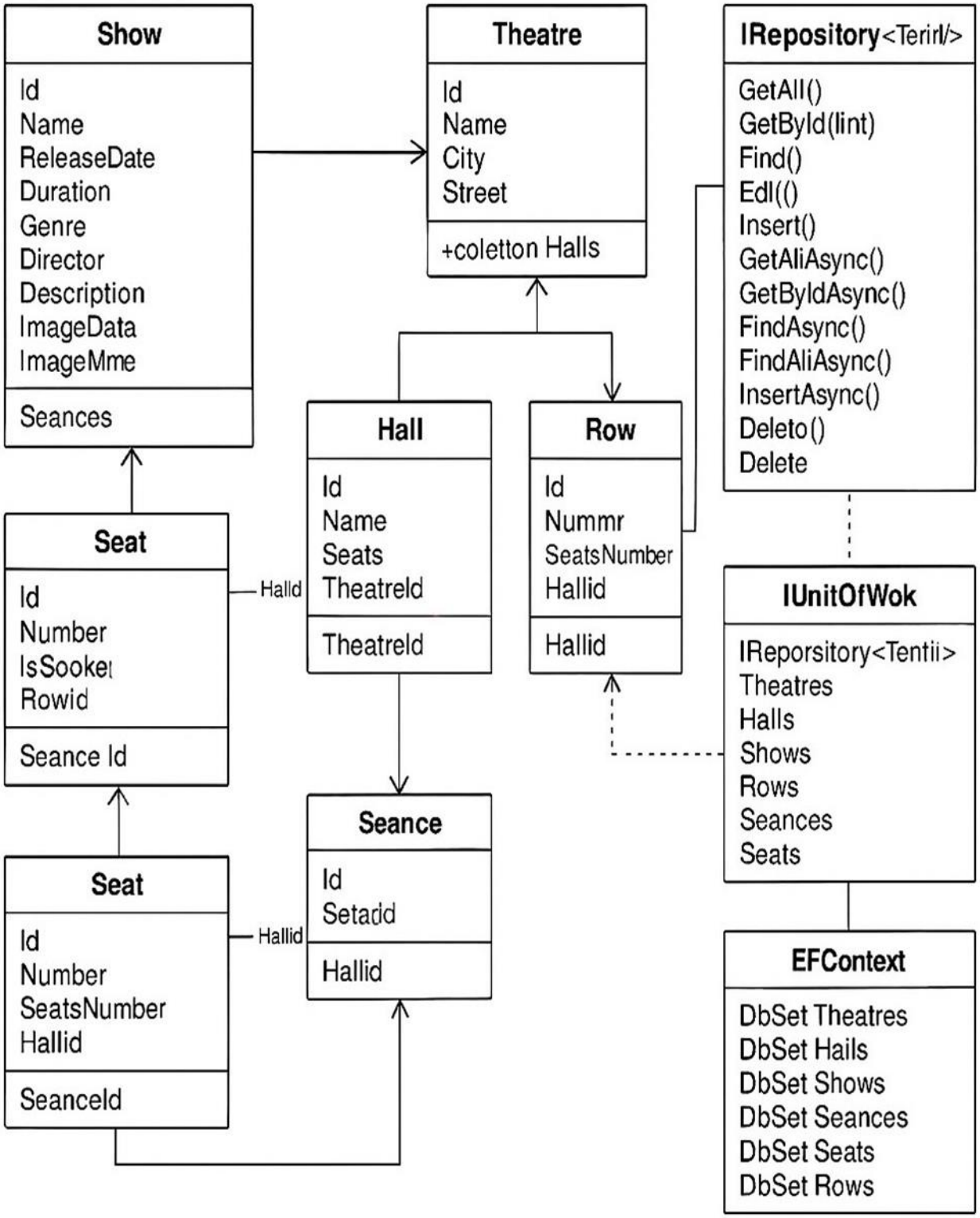

Рисунок В.3 - UML-діаграма класів WEB-ресурсу театральної установи

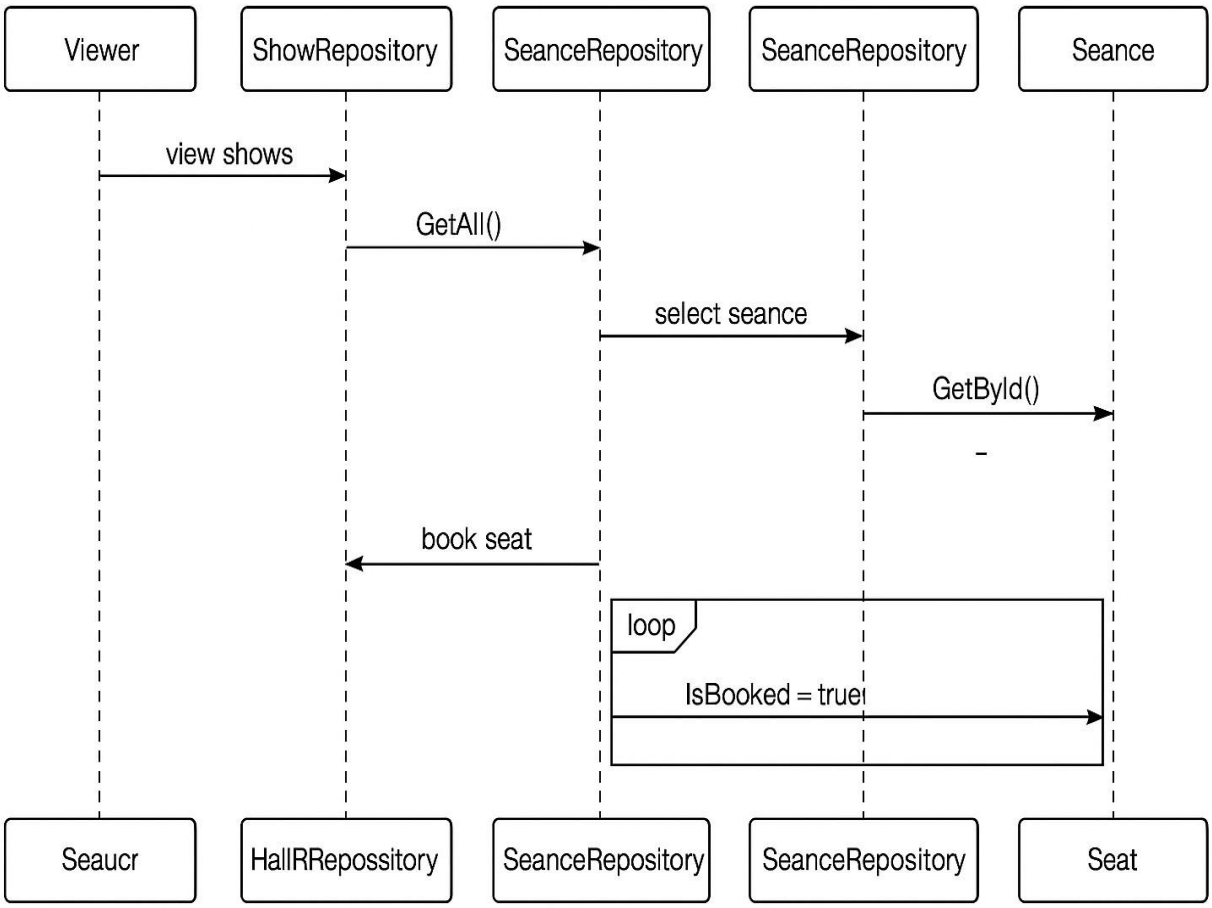

Рисунок В.4 - UML-діаграма послідовність дій WEB-ресурсу театральної установи.

Рисунок В.5 – Лінійна структура сайту

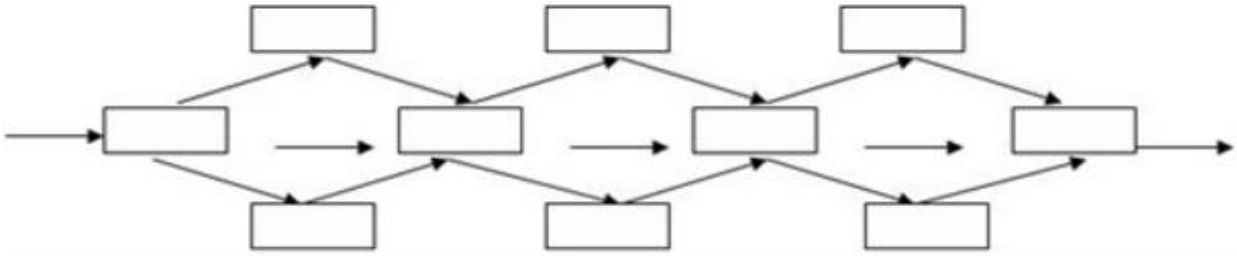

Рисунок В.6 – Структура сайту лінійна з відгалуженнями

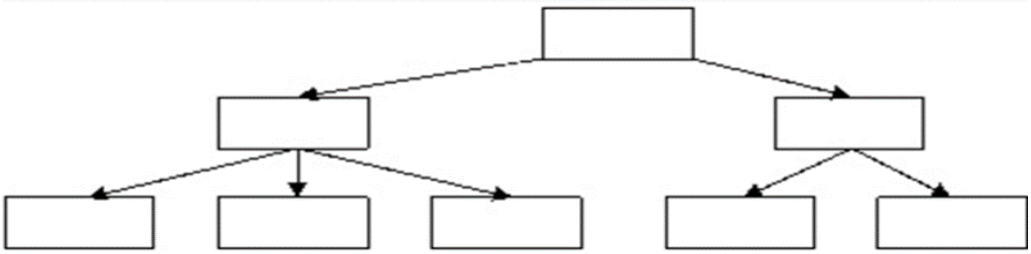

Рисунок В.7 – Ієрархічна структура

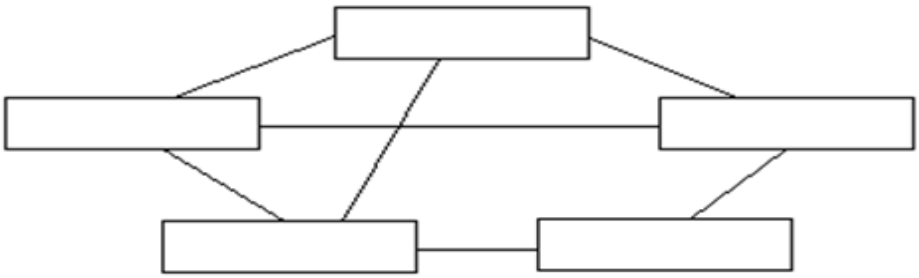

Рисунок В.8 – Мережна структура

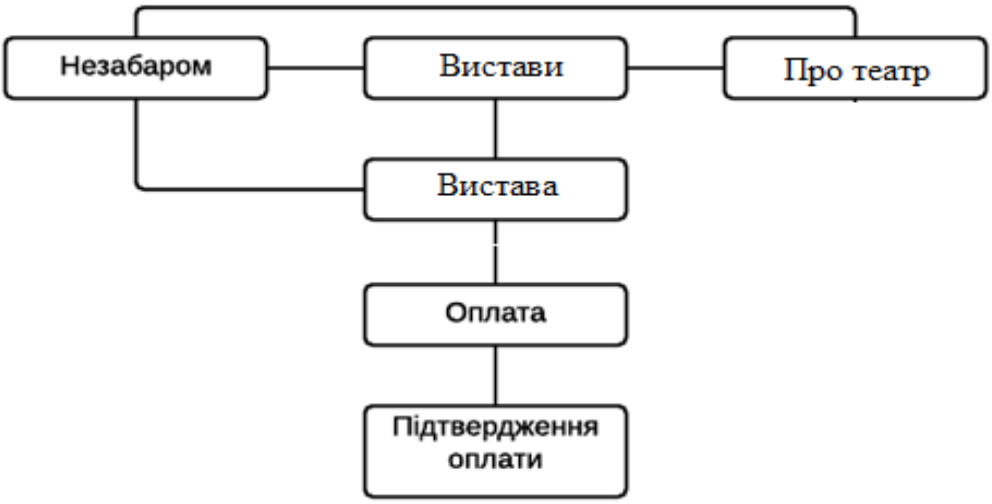

Рисунок В.9 – Структура сайту

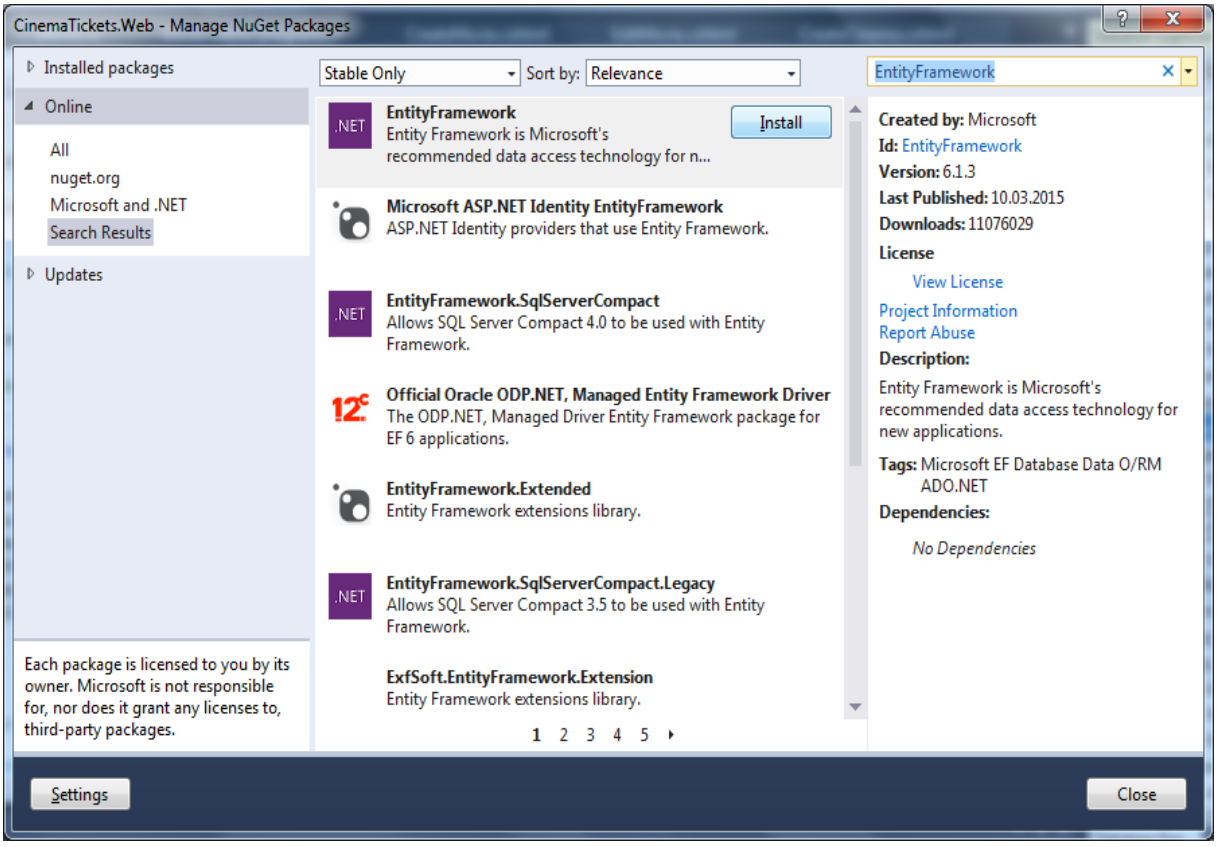

Рисунок В.10 – Вікно керування пакетами NuGet з відображеними результатами для запиту «EntityFramework»

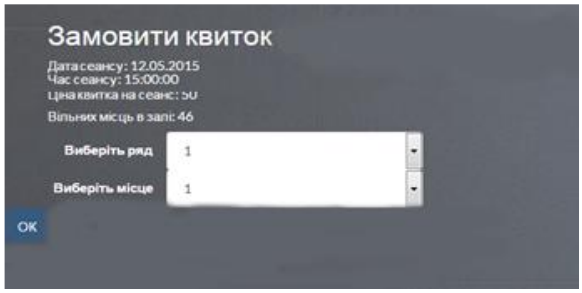

Рисунок В.11 – Сторінка замовлення квитка на сеанс

Рисунок В.12 – Сторінка авторизації

Рисунок В.13 – Зовнішній вигляд сайту

ДОДАТОК Г (ДОВІДНИКОВИЙ)
ІНСТРУКЦІЯ КОРИСТУВАЧА

Перед завантаженням і запуском програми необхідно переконатися, що встановлені такі компоненти:

- .NET Framework або .NET Core ;
- Visual Studio (рекомендується Community Edition) - 2019 або новіше;
- SQL Server / SQL Express - будь-яка версія;
- NuGet Package Manager - вбудовано у Visual Studio.

Запуск з Visual Studio:

1. Встановити проєкт `Web` як стартап-проєкт.
2. Натисніть `Ctrl + F5` або кнопку ▶ "Start Without Debugging".
3. Програма відкриється у браузері

Передумови.

1. Місця не додаються до сеансу автоматично. Потрібно створити ряди та місця в залі перед створенням сеансу - тоді вони копіюються.
2. Щоб оновити виставу треба знайти виставу за `Id`, внести зміни, зберегти.
3. Налаштування БД.

   Створіити базу даних Автоматично (Code First):

   У `Web.config` або `App.config` переконатися, що є правильний connection string :

```
<connectionStrings>
<add name="TheatreTicketsDB" connectionString="Data
Source=.\SQLEXPRESS;Initial Catalog=TheatreDB;Integrated
Security=True" providerName="System.Data.SqlClient" />
</connectionStrings>
```

   У Visual Studio відкрийте Package Manager Console і виконати:

```
Update-Database
```

   Це застосує всі міграції та створить базу даних автоматично.