Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Створення веб-сервісу для бронювання та оренди житлових приміщень

Виконав: студент IV курсу, групи СН-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

(підпис) Грех В.В.
(прізвище та ініціали)

Керівник (підпис) Литвиненко Я.В.
(прізвище та ініціали)

Нормоконтроль (підпис) Липак Г.І.
(прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О.
(прізвище та ініціали)

Рецензент (підпис) Загородна Н.В.
(прізвище та ініціали)

Тернопіль

2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

		ЗАТВЕРДЖУЮ		
		Завідувач кафедри		
				Боднарчук І.О.
		(підпис)		(прізвище та ініціали)
		« 27 » січня 2025 р.		

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Грех Володимир Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи Створення веб-сервісу для бронювання та оренди житлових приміщень

Керівник роботи Литвиненко Ярослав Володимирович, д.т.н., проф. кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *07* » *травня* 2025 року № *4/7-444*

2. Термін подання студентом завершеної роботи 23 червня 2025 р.

3. Вихідні дані до роботи Літературні джерела щодо розробки веб-сервісів

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз предметної області та постановка завдання. 1.1 Аналіз сучасного стану ринку веб-сервісів з бронювання та оренди житлових приміщень. 1.2 Аналіз аналогічних програмних рішень. 1.3 Визначення вимог до веб-сервісу для бронювання та оренди житлових приміщень. 1.4 Постановка завдання кваліфікаційної роботи. 2. Проєктування веб-сервісу для бронювання та оренди житлових приміщень. 2.1 Вибір архітектури веб-сервісу. 2.2 Технології розробки. 2.3 Проєктування бази даних. 2.4 Управління бронюванням і орендою житлових приміщень та моделювання структури компонентів фронтенду. 2.5 Опис взаємодії клієнт-сервер. 3. Реалізація та тестування веб-сервісу. 3.1 Реалізація функціоналу реєстрації, авторизації користувачів. 3.2 Реалізація системи створення та бронювання оголошень. 3.3 Інтеграція платіжної системи (Stripe). 3.4 Тестування веб-додатку. 4. Безпека життєдіяльності, основи охорони праці. 4.1 Безпека умов праці ІТ-спеціаліста. 4.2 ~~Цивільний захист~~Безпека життєдіяльності та фактори воєнного часу. 4.3 Охорона праці ~~прив~~ ~~експлу-атації веб-сервісу.~~ІТ-компаніях. Висновки. Перелік джерел.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайди: 1 Тема. 2 Мета роботи, об'єкт дослідження, предмет дослідження. 3 Завдання дослідження. 4 Актуальність теми. 5 Порівняння існуючих рішень. 6 Ключовий функціонал веб-сервісу. 7 Основні засоби розробки. 8 Архітектура веб-сервісу. 9 Серверна частина веб-сервісу. 10 Серверна частина веб-сервісу. 11 Клієнтська частина веб-сервісу

12 Сторінка реєтрації. 13 Сторінка створення оголошення. 14 Головна сторінка 15 Висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин Віктор Степанович	02.06.2025	05.06.2025

7. Дата видачі завдання 27 січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	07.05.2025	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи	08.05.2025-15.05.2025	*Виконано*
3.	Виконання дослідження щодо розробки	16.05.2025-22.05.2025	*Виконано*
	веб-сервісу для бронювання та оренди житлових		
	Приміщень		
4.	Оформлення розділу «Аналіз предметної області та	23.05.2025-01.06.2025	*Виконано*
	постановка завдання»		
5.	Оформлення розділу «Проєктування веб-сервісу для	23.05.2025-01.06.2025	*Виконано*
	бронювання та оренди житлових приміщень»		
6.	Оформлення розділу «Реалізація та тестування веб-	23.05.2025-01.06.2025	*Виконано*
	сервісу»		
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»	02.06.2025-05.06.2025	*Виконано*
8.	Виконання завдання до підрозділу «Основи охорони	02.06.2025-05.06.2025	*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи	06.06.2025-10.06.2025	*Виконано*
10.	Нормоконтроль	11.06.2025-13.06.2025	*Виконано*
11.	Перевірка на плагіат	16.06.2025	*Виконано*
12.	Попередній захист кваліфікаційної роботи	18.06.2025	*Виконано*
13.	Захист кваліфікаційної роботи	23.06.2025	

Студент _____ Грех В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Литвиненко Я.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Створення веб-сервісу для бронювання та оренди житлових приміщень // Кваліфікаційна робота освітнього рівня «Бакалавр» // Грех Володимир Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-41 // Тернопіль, 2025 // С. 65, рис. – 7 , табл. – 2, додат. – 2, бібліогр. – 36.

Ключові слова: оренда житла, веб-сервіс, ASP.NET, MongoDB, бронювання онлайн, платіжна інтеграція, користувацький інтерфейс, інформаційна безпека

Кваліфікаційна робота присвячена розробці веб-сервісу для бронювання та оренди житлових приміщень, аналогічного до платформ Airbnb чи Booking.com, з використанням сучасних вебтехнологій, нереляційних баз даних і фінтех-інтеграцій.

У першому розділі кваліфікаційної роботи подано аналіз предметної області оренди житла, розглянуто сучасні рішення на ринку, висвітлено основні функціональні вимоги до веб-сервісів такого типу. Проаналізовано архітектурні особливості веб-сервісів для розміщення та пошуку пропозицій житла, включно з механізмами оплати, відгуків і безпеки.

У другому розділі досліджено технічні та методичні підходи до розробки веб-сервісу обґрунтовано вибір платформи ASP.NET і шаблону MVC, сформовано модель даних із використанням NoSQL СУБД MongoDB, окреслено логіку взаємодії основних сутностей сервісу.

У третьому розділі виконується реалізація та тестування веб-сервісу, розроблено ключові підсистеми сервісу, серед них авторизація з ролями, створення оголошень, пошук, інтерактивне бронювання, онлайн оплата через

Stripe. Проведено тестування функціональних сценаріїв та перевірку продуктивності системи.

У розділі “Безпека життєдіяльності, основи охорони праці” висвітлено питання безпеки умов праці ІТ-спеціаліста, охороні праці в ІТ-компаніях, а також ризики, пов’язані з роботою в умовах воєнного часу.

ANNOTATION

Creation of a Web Service for Booking and Renting Residential Premises // Qualification work of the educational level «Bachelor» // Hrekh Volodymyr Volodymyrovych // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-41 // Ternopil, 2025 // P. 62 , fig. – 7, tabl. – 2, annexes. – 2, references – 36.

Keywords: housing rental, web service, ASP.NET, MongoDB, online booking, payment integration, user interface, information security

The qualification work is dedicated to the development of a web service for booking and renting residential premises, similar to platforms such as Airbnb or Booking.com, using modern web technologies, non-relational databases, and fintech integrations.

The first chapter presents an analysis of the domain of short-term housing rental, a review of existing market solutions, and an overview of the main functional requirements for systems of this type. Architectural features of web services for listing and searching for housing offers are examined, including payment mechanisms, reviews, and security aspects.

The second chapter explores the technical and methodological approaches to web application development the choice of the ASP.NET platform and the MVC pattern is justified, a data model is created using the NoSQL database MongoDB, and the logic of interaction between the main entities of the service is outlined.

The third chapter presents the development of the service's key subsystems role-based authorization, listing creation, search, interactive booking, and online payment via Stripe. A user dashboard interface for different user types is proposed, as well as basic administrative tools. Functional scenario testing and performance evaluation of the system have been carried out.

The section “Life safety, basics of labor protection” covers the issues of safe working conditions for IT specialists, labor protection in IT companies, as well as the risks associated with working in wartime.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

AJAX (A synchronous J avaScript A nd X ML) – підхід до побудови користувацьких інтерфейсів веб-сервісів, за яких вебсторінка, не перезавантажуючись, у фоновому режимі надсилає запити на сервер і сама звідти довантажує потрібні користувачу дані.

ASP.NET – це серверна веб інфраструктура з відкритим вихідним кодом, створена компанією Microsoft, що працює під управлінням Windows і випущена на початку 2000-х років. Технологія ASP.NET дає змогу розробникам створювати веб-додатки, веб-сервіси та сайти

CRUD – абревіатура від англ. Create, Read, Update, Delete – основні операції з даними у веб-застосунках.

CSS (Cascading Style Sheets) – каскадні таблиці стилів, мова стилізації веб-сторінок.

HTML (HyperText Markup Language) – мова розмітки гіпертексту, основа структури веб-сторінок.

MongoDB – це гнучка та масштабована база даних NoSQL

БД – база даних, впорядкований набір структурованих даних, що зберігаються в електронному вигляді.

ЗМІСТ

ВСТУП

Актуальність теми. Внаслідок глобалізації та цифровізації значно зросла мобільність населення, а разом із цим – і попит на веб-сервіси з бронювання житла та оренди проживання. Традиційні платформи, як-от Airbnb і Booking.com, створили потужні стандарти ринку, проте залишаються недоступними або неадаптованими до специфіки локальних користувачів, зокрема у питаннях мови, способів оплати, безпеки даних та юридичного регулювання. Тому розробка веб-сервісу для бронювання та оренди житла з урахуванням сучасних вимог до інтерфейсу, функціональності, інтерактивності та захисту персональних даних є актуальним напрямком сучасних досліджень в галузі інформаційних технологій.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня “Бакалавр” є підвищення якості послуг онлайн оренди житла шляхом створення зручного, безпечного та функціонального веб-сервісу для бронювання. Для досягнення поставленої мети потрібно виконати ряд завдань, зокрема:

– Проаналізувати стан досліджень і технологій у сфері розробки сервісів оренди житла;

– Вивчити особливості UX/UI дизайну, ергономіки цифрових сервісів та принципів побудови зручного інтерфейсу;

– Обґрунтувати вибір технологічного стеку для реалізації веб-сервісу;

– Розробити архітектуру, дизайн та функціонал веб-сервісу із системою реєстрації, пошуку, бронювання та онлайн оплати;

– Забезпечити інтеграцію з платіжною системою (Stripe) та реалізувати базові механізми захисту персональних даних;

– Провести тестування веб-сервісу на відповідність функціональним, ергономічним та безпековим критеріям.

Практичне значення одержаних результатів. Результати роботи мають практичну цінність для розробників, які працюють над створенням веб-сервісів

у сфері оренди житла, готельного бізнесу та цифрового туризму. Розроблений веб-сервіс може бути використаний як прототип комерційної платформи для локального ринку, адаптованої до мовних, економічних та правових умов конкретного регіону. Також напрацювання можуть бути основою для подальших досліджень у галузі UX дизайну, цифрової безпеки й інтеграції платіжних сервісів у веб-сервіси.

РОЗДІЛ 11. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

11.1 Аналіз сучасного стану ринку веб-сервісів з бронювання та оренди житлових приміщень

Ринок веб-сервісів для бронювання та оренди житлових приміщень демонструє стрімке зростання протягом останнього десятиліття. Поява таких платформи як Airbnb (заснована в 2008 році) та Booking.com (заснована в 1996 році) змінили підхід до оренди житла, зробивши пошук і бронювання максимально простим для користувача [1] [2]. На сьогодні в Airbnb існують мільйони оголошень про здачу житла майже в усіх країнах світу. На 2024 рік, веб-сервіс Airbnb налічував близько 6 мільйонів оголошень (за іншими оцінками від 6 до 8 мільйонів) та обслуговував 150 мільйонів користувачів. Орендодавцями на Airbnb стали 4 мільйони осіб, а загальний заробіток орендодавців склав близько 180 мільярдів доларів. Після спаду, спричиненого пандемією COVID-19, Airbnb впевнено відновив позиції у 2023 році компанія заробила майже $9,9 мільярда, а чистий прибуток склав $4,8 мільярда це у 2,5 раза більше, ніж у 2022.

Платформа Booking.com, що входить до складу корпорації Booking Holdings, утримує лідерство за обсягом валового бронювання та доходу у галузі онлайн-туризму. У 2023 році дохід Booking.com досяг рекордних $21,4 млрд (на 25% більше, ніж у 2022), що більш ніж удвічі перевищує дохід Airbnb . Бізнес-модель Booking.com дещо відрізняється, сервіс пропонує не лише приватне житло, а й готелі, хостели, апартаменти, а також додаткові туристичні послуги (авіаквитки, оренда авто тощо). Відповідно, каталог Booking.com налічує близько 28 млн активних оголошень, серед яких ~6,6 млн – це будинки та апартаменти, а решта – готельні номери та інші об'єкти розміщення. Booking.com працює за агентською моделлю, отримуючи комісію ~15% з кожного

бронювання, тоді як Airbnb використовує змішану модель комісій (невелика комісія з хоста та сервісний збір з гостя).

Окрім двох світових гігантів, існують інші платформи оренди та бронювання житла. Серед них: Vrbo (HomeAway) – платформа, орієнтована на будинки для відпусток (належить Expedia Group); Agoda Homes в Азіатсько-Тихоокеанському регіоні; Регіональні сервіси на зразок 9flats або Couchsurfing. В цілому, галузь оренди та бронювання переживає фазу зрілості, конкуренція між платформами зростає, що змушує їх впроваджувати нові функції (програми лояльності, гарантії безпеки, страхування тощо) та підвищувати якість сервісу. За даними галузевих звітів, у 2023–2024 рр. 76% керуючих нерухомістю відзначили посилення конкуренції, що вимагає розумнішого ціноутворення і оптимізації операцій.

Важливим фактором, що вплинув на ринок онлайн бронювання та оренди житла останніми роками, стали глобальні кризи – пандемія та війни. Пандемія COVID-19 у 2020 р. призвела до різкого падіння оренди житла, проте вже у 2021–2022 рр. галузь відновилася; У 2023 року кількість міжнародних туристичних прибуттів і оренди житла склала 1,3 млрд, що на 88% до рівня докризового 2019.

Війна в Україні з 2022 р. мала подвійний ефект, з одного боку, іноземний туризм до України практично зупинився; з іншого – виник великий внутрішній попит на оренду житла для переміщених осіб. Цей попит частково задовольнявся за рахунок волонтерських ініціатив та спеціальних програм (як от згадана програма Airbnb.org для біженців) [3]. У той же час, українські IT-фахівці та підприємці продовжують інтегруватися в глобальний ринок, створюючи власні сервіси або долучаючись до міжнародних проєктів з оренди та бронювання нерухомості.

Отже, ринок веб-сервісів для бронювання та оренди житла нині характеризується високою концентрацією (домінування кількох глобальних платформ), значними обсягами транзакцій та високим рівнем технологічності. Це створює як можливості, так і виклики для нових гравців. З одного боку, величезна база користувачів привчена до онлайн бронювання, їм зрозумілий сам

принцип роботи таких сервісів. З іншого боку, новому веб-сервісу важко конкурувати з гігантами за асортиментом пропозицій та довірою аудиторії. Тому, розробляючи спеціалізований веб-сервіс для оренди житла, слід робити акцент на певних нішах або додаткових перевагах – наприклад, локальний ринок (місто, країна), покращений користувацький досвід, інтеграція з локальними платіжними системами, підтримка української мови тощо.

11.2 Аналіз аналогічних програмних рішень

У цьому підрозділі розглянемо детальніше функціональні можливості та технології, реалізовані в існуючих програмах аналогах, аби визначити найкращі практики для нашого проєкту. Основну увагу приділимо двом провідним платформам – Airbnb та Booking.com – як орієнтирам у галузі, а також проаналізуємо деякі відкриті проєкти та приклади реалізації подібних систем.

Платформа Airbnb являє собою двосторонній онлайн маркетплейс, що з'єднує хостів (тих, хто здає житло) та гостей (тих, хто орендує). Ключові особливості Airbnb – це наявність деталізованих профілів користувачів (з системою відгуків і рейтингів), продуманий процес бронювання (подача запиту на бронювання або миттєве бронювання Instant Book), вбудована система повідомлень між гостем і хостом, безпечна платіжна система (Airbnb виступає посередником, стягує плату з гостя і перераховує хосту після заселення), а також додаткові сервіси на кшталт Airbnb Experiences (замовлення екскурсій та розваг від місцевих мешканців). Airbnb зробив акцент на унікальності пропозицій та соціальній складовій, мандрівники часто обирають Airbnb заради "локального досвіду і спілкування з хостами, чого не дають традиційні готелі.

Технологічно Airbnb – це складна система, що складається з веб-версії, мобільних додатків, потужного бекенду на основі фреймворку Ruby on Rails з використанням реляційної СУБД (MySQL), проте з часом архітектура еволюціонувала (виділено сотні мікросервісів, використовуються різні спеціалізовані сховища даних, системи кешування тощо). Для фронтенду

застосовуються сучасні підходи – AJAX для динамічного оновлення даних, SPA компоненти в особливо інтерактивних частинах (наприклад, мапи для пошуку житла). Значну увагу Airbnb приділяє масштабованості та доступності, система обробляє мільйони запитів, тому використовуються балансувальники навантаження, CDN для роздачі зображень, географічне реплікування даних тощо.

Сервіс Booking.com дещо відрізняється фокусом. Якщо Airbnb починався як платформа C2C (від приватних осіб приватним особам), то Booking.com працює як класичний ріелтор, співпрацюючи також з комерційними готелями, хостелами, апартаментами і іншими різноманітними типами житла. Основний сценарій Booking.com – миттєве бронювання без необхідності очікувати підтвердження від орендодавця, користувач обирає доступний номер/квартиру на конкретні дати, вводить платіжні дані – і бронювання одразу гарантується. Для цього в системі передбачено підтримку календарів доступності, інтеграцію з каналами продажів готелів, тощо. Booking.com також пропонує комплексні послуги подорожі. Після бронювання житла користувачеві можуть запропонувати забронювати машину, авіаквитки, трансфер з аеропорту.

З технічної точки зору, ядро Booking.com – це монолітна веб-платформа, розроблена на Perl (історично), яка, втім, також була поступово розбита на сервіси та модулі. Для зберігання даних використовується кілька СУБД (реляційні для основних даних бронювань, NoSQL для сесій та кешу, пошукові індекси для текстового пошуку).

У висновку маємо, що система повинна витримувати пікові навантаження під час туристичних сезонів, тому у веб-сервісу має бути добре оптимізована взаємодія з базами даних і пошуковими алгоритмами. Компанія має штат у десятки тисяч співробітників, тому розробка ведеться за передовими практиками CI/CD, автоматичне тестування, експериментування (A/B тести нового функціоналу на частині аудиторії).

11.3 Визначення вимог до веб-сервісу для бронювання та оренди житлових приміщень

На основі аналізу предметної області та типових рішень у сфері онлайн бронювання оренди житла були сформульовані вимоги до функціональності та архітектури веб-сервісу. Як основу обрано платформу ASP.NET із використанням Entity Framework Core для роботи з NoSQL базою даних MongoDB [4]. Такий підхід забезпечує повну підтримку транзакцій, гнучке моделювання сутностей, застосування міграцій, а також тісну інтеграцію з .NET середовищем.

Ключова функціональність охоплюватиме всі основні аспекти сучасного сервісу бронювання житла. Система реалізовуватиме повний цикл роботи з користувачами від реєстрації та входу до управління особистими профілями та розмежуванням ролей. Буде впроваджено три типи користувачів: орендар, який шукає житло та здійснює бронювання; орендодавець, який розміщує оголошення про здачу нерухомості; та адміністратор, який модерує контент і контролює роботу системи. Аутентифікація реалізується через стандартну ASP.NET Identity з підтримкою хешування паролів, зміни й відновлення доступу, а також багатофакторної аутентифікації. Крім того, передбачено вхід через сторонні сервіси за допомогою OAuth, зокрема Google.

Функціонал створення оголошень дозволятиме орендодавцям публікувати пропозиції, вказуючи всю необхідну інформацію як назву житла, адресу або локацію, опис, тип помешкання, площу, кількість кімнат і гостей, перелік зручностей, правила проживання, фото, ціни та доступність. Завантажені фотографії зберігатимуться на сервері або в хмарному сховищі, а в базі даних зберігатимуться лише шляхи до них. Оголошення публікуватимуться автоматично, після чого орендодавець зможе редагувати, деактивувати чи видаляти свої пропозиції. Усі дії логуються для забезпечення цілісності даних.

Орендарі мають змогу здійснювати пошук житла за локацією, вибраними датами, ціною, кількістю кімнат та зручностями. Інтерфейс підтримуватиме

фільтрацію та сортування результатів за ціною, рейтингом або популярністю. Кожне оголошення буде представлено у вигляді інтерактивної картки з коротким описом, фото, вартістю, а при перегляді – з детальною інформацією. Обране житло можна додати до списку улюбленого, який зберігатиметься в особистому профілі або локальному сховищі браузера.

Процес бронювання реалізовуватиметься з перевіркою актуальності обраних дат, щоб запобігти конфліктам. Після вибору житла та введення потрібної кількості гостей користувач потраплятиме на сторінку підтвердження, де відображається підсумок вартості. Оплата здійснюється через інтеграцію з платіжним шлюзом Stripe, що відповідає сучасним стандартам безпеки (PCI DSS). Транзакція проводитиметься на захищеній формі – або на сайті Stripe, або через вбудований віджет Stripe Elements. У випадку успішної оплати система створюватиме запис бронювання із відповідним статусом, а обрані дати позначаються як зайняті. Усі сторони отримують сповіщення, орендар – підтвердження та контактну інформацію, орендодавець – повідомлення про нове бронювання. У випадку помилки оплати створення бронювання не відбувається, а користувач має змогу повторити спробу.

Передбачено можливість скасування бронювання як з боку користувача (до певного терміну), так і з боку адміністратора. У разі реалізації часткового повернення коштів це може бути описано окремою політикою, але технічна основа для цього врахована. Історія бронювань буде доступна в особистому кабінеті, орендар бачитиме свої подорожі, а орендодавець – календар майбутніх і завершених бронювань зі статусами.

Адміністративний інтерфейс дозволятиме переглядати список користувачів, блокувати чи видаляти порушників, модератувати оголошення, а також, за потреби, переглядати статистику (кількість бронювань, навантаження, виручка). Усі ці функції реалізовані з урахуванням обмеження доступу на основі ролей і політик авторизації.

Значна увага буде приділена безпеці, всі паролі хешуються за допомогою алгоритмів ASP.NET Identity, використовується виключно HTTPS, а всі запити в

базу даних реалізуватимуться безпечно через mongodb із параметрами. Передбачено захист від XSS, CSRF, SQL ін’єкцій, а також обмеження адміністративног доступу лише для авторизованих користувачів із відповідною роллю. Для чутливих транзакцій застосовуватимуться транзакційні механізми EF Core, що дозволяє зберігати консистентність навіть при виникненні збоїв. Основні вимоги та архітекрутні рішення веб-сервісу наведені у таблиці (див. таблицю 1.1).

Таблиця 1.1 – Основні вимоги та архітектурні рішення веб-сервісу

Категорія	**Опис**
1	**2**
Основні програмні забезпечення	ASP.NET, Entity Framework Core, MongoDB
Стек технологій	ASP.NET, Entity Framework Core, MongoDB, JS, AJAX
ORM та база даних	EF Core для підтримки транзакцій, міграцій, гнучке моделювання; MongoDB
Типи користувачів	Орендар, орендодавець, адміністратор
Аутентифікація	ASP.NET Identity (хешування паролів, MFA), OAuth (Google)
Оголошення	Створення, редагування, модерація, фото, опис, ціна, правила, зручності
Пошук і фільтрація	За локацією, ціною, датами, зручностями; сортування за ціною, рейтингом, популярністю

Продовження таблиці 1.1

1	2
Бронювання	Перевірка доступності, підсумок вартості, інтеграція з Stripe, статуси бронювання
Оплата	Stripe API, підтримка повторної спроби, підтвердження транзакції
Скасування	Користувачем або адміністратором; можливість часткового повернення
Відгуки	Взаємні оцінки після завершення бронювання, збереження в базі
Адміністративний модуль	Керування користувачами, модерація оголошень, статистика
UI/UX	Адаптивність, багатомовність (UA/EN), швидкий відгук (AJAX), зручний інтерфейс
Безпека	HTTPS, захист від XSS, CSRF, SQL ін’єкцій, обмеження ролей

Таким чином, усі сформульовані вимоги повністю відповідають можливостям обраного технологічного стеку: ASP.NET, EF Core, MongoDB, Stripe API. Сервіс буде реалізований як масштабований, безпечний і функціонально повний веб-сервіс, орієнтований на зручність користувача та стабільну роботу в реальному середовищі.

11.4 Постановка завдання кваліфікаційної роботи

З урахуванням проведеного аналізу можна сформулювати чітку постановку задачі, яка розв’язується у бакалаврській кваліфікаційній роботі. Необхідно створити веб-сервіс для бронювання та оренди житлових приміщень, використовуючи технології ASP.NET MVC на мові C#. Сервіс повинен забезпечувати функціонал, аналогічний популярним платформам з оренди житла

(Airbnb, Bookingюcom), але в більш обмеженому масштабі, орієнтуючись на навчальні цілі та потреби локального ринку.

При цьому слід реалізувати такі основні підсистеми, систему реєстрації користувачів з розмежуванням ролей (орендар, орендодавець, адміністратор) і захистом даних користувачів. Інтерфейс для орендодавця, можливість публікації оголошень про житло (з фото, описом, ціною, календарем); інтерфейс для орендаря, пошук і перегляд доступного житла, оформлення бронювання на вибрані дати. Механізм обробки бронювань, створення записів бронювання, блокування дат, сповіщення сторін; інтеграцію з платіжною системою (Stripe) для прийому онлайн оплат при бронюванні; панель адміністратора для моніторингу користувачів і оголошень, модерації контенту; базу даних для зберігання всієї необхідної інформації (облікові записи, оголошення, бронювання, транзакції тощо) – з використанням СУБД MongoDB. Базові перевірки і тестування роботи системи на коректність (сценарії реєстрації, бронювання, оплати, ролі доступу).

В рамках роботи слід розробити архітектуру додатку, створити схему бази даних, написати програмний код усіх необхідних компонентів (моделі, контролери, представлення), і провести відладку та тестування функціоналу на реальних або тестових даних. Також потрібно підготувати документацію – пояснювальну записку (цей текст) з описом прийнятих рішень, та додатки, що містять фрагменти коду, приклади екранів програми, схему БД тощо.

Обмеження, проект реалізується як веб-сервіс монолітної архітектури (без виділення мікросервісів), що працює на одному сервері. Це спрощує розробку і відповідає навчальним цілям. У разі потреби масштабування або винесення окремих сервісів (наприклад, відправка email) – це обговорюється, але не обов’язково реалізується на практиці.

У результаті виконання кваліфікаційної роботи має бути отриманий працюючий прототип веб сервісу для бронювання житла, який задовольняє визначені вимоги.

РОЗДІЛ 12. ПРОЄКТУВАННЯ ВЕБ-СЕРВІСУ ДЛЯ БРОНЮВАННЯ ТА ОРЕНДИ ЖИТЛОВИХ ПРИМІЩЕНЬ

12.1 Вибір архітектури веб-сервісу

В якості основи програмної архітектури було обрано шаблон MVC (Model-View-Controller) в реалізації фреймворку ASP.NET MVC [5]. Архітектура MVC передбачає розділення застосунку на три основних компоненти: Model (дані та бізнес-логіка), View (інтерфейс представлення даних користувачу) та Controller (компонент, що приймає запити від користувача, обробляє їх за допомогою моделей та визначає, яке представлення повернути). Такий підхід сприяє чіткому розподілу відповідальностей (separation of concerns) і полегшує подальшу підтримку коду.

На рис. 2.1 схематично зображено взаємодію між компонентами MVC, контролери отримують та обробляють HTTP-запити, звертаються до моделей (наприклад, до бази даних через модель) і повертають відповідні представлення користувачу. Модель не залежить від представлення та контролера, тоді як контролер і представлення працюють із моделлю.

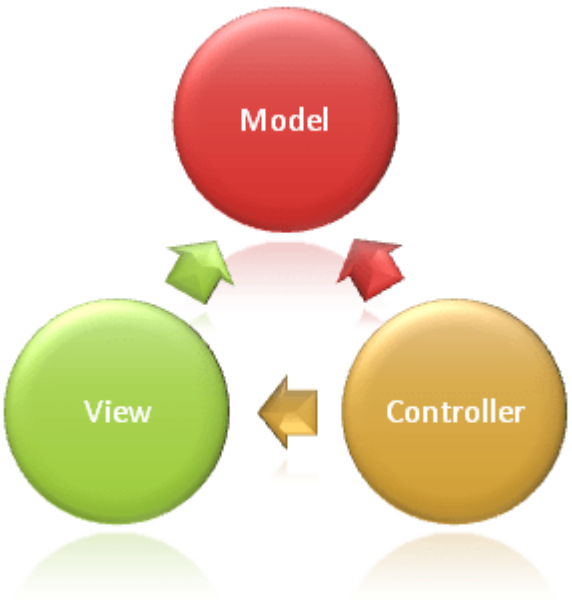

Рисунок 2.1 – Концептуальна схема архітектури MVC

Для веб-сервісу архітектура MVC підходить природним чином, оскільки дозволяє організувати код за функціональними частинами. Зокрема, в рамках ASP.NET проєкт було структуровано наступним чином (див. рис. 2.2, де показано структуру проєкту у Visual Studio).

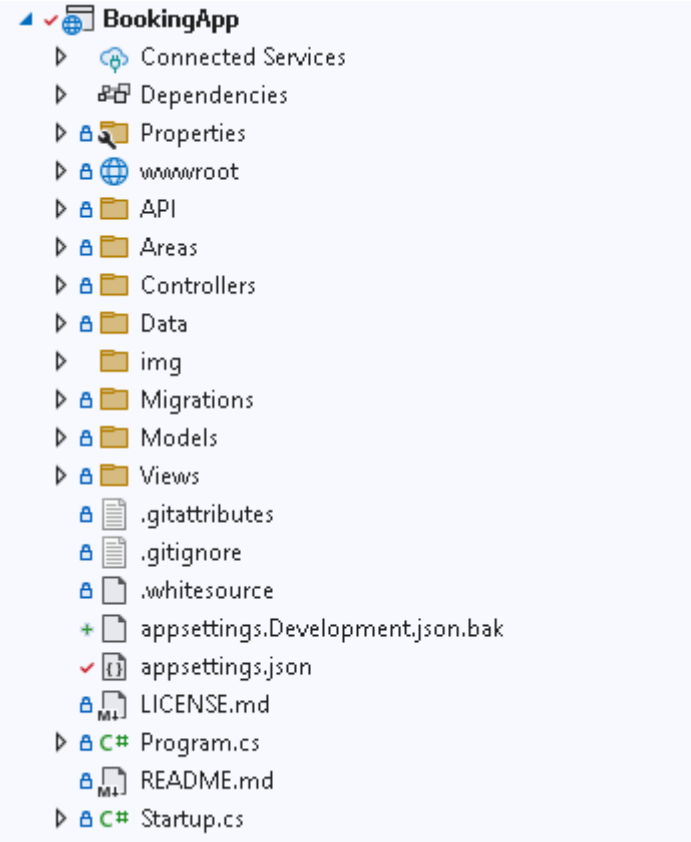

Рисунок 2.2 – Структура проєкту вебсервісу

Як видно з рис. 2.2, проєкт містить стандартні для ASP.NET компоненти, папка Models – тут розміщено класи моделей домену (предметної області) та контекст бази даних. У нашому випадку до моделей належать, User (користувач системи), Accommodation (житло/помешкання, що здається), Offer (конкретна пропозиція оренди – у нашій моделі цей клас тісно пов’язаний з Accommodation, можемо розглядати Offer як оголошення), Booking (бронь), а також допоміжні моделі, як-от PaymentTransaction (платіж) чи Review (відгук), якщо вони передбачені. Крім того, є моделі, що не зберігаються в БД, а слугують для передачі даних у представлення (ViewModel), наприклад SearchViewModel для параметрів пошуку. Папка Controllers – містить контролери, які реалізують

логіку обробки HTTP-запитів. У нашому проекті є контролери для кожної суттєвої області, HomeController (головна сторінка, пошук), AccountController (реєстрація, логін, управління профілем), AccommodationController (створення та редагування оголошень), BookingController (операції з бронюваннями), AdminController (адмін функції), PaymentController (обробка платежів через Stripe) тощо.

Контролери містять методи дії (action methods), які відповідають на конкретні запити (наприклад, GET /Home/Index, POST / Account/Login і т.д.). Папка Views – містить представлення (сторінки інтерфейсу користувача) у вигляді .cshtml файлів Razor. Структура папки Views повторює структуру контролерів, для кожного контролера створено папку з відповідними .cshtml (наприклад, Views/Home/Index.cshtml, Views/ Accommodation/Create.cshtml тощо).

Також є спільні представлення, шаблони Views/Shared/_Layout.cshtml (загальний макет сторінки) і часткові представлення (partial views) для повторюваних компонентів. Папка Data – у разі використання ORM (наприклад, Entity Framework) тут містився б контекст даних і конфігурація доступу до БД. У нашому випадку з MongoDB, ми можемо мати тут клас для підключення до Mongo (напр. MongoContext) або ж ініціалізацію колекцій у коді програми. Папка wwwroot – для статичних ресурсів, CSS, JavaScript, файли зображень.

Ми використовуємо Bootstrap CSS-фреймворк, тож у wwwroot знаходяться відповідні CSS та JS, а також файли користувацьких скриптів (наприклад, для валідації форм на стороні клієнта) та стилів. Інші файли, appsettings.json (файл конфігурації, де зазначено рядок підключення до БД, ключі Stripe та інші налаштування), Program.cs (точка входу, де запускається веб хост, реєструються сервісів), Startup.cs (в .NET 6+ його функції перейняв Program.cs, але якщо використовувати старіший шаблон, Startup містить налаштування програми – маршрути, DI контейнер тощо).

Архітектурно застосунок є типовим монолітом на .NET, усі функції виконуються в рамках одного веб-сервісу, який розгортається на сервері.

Взаємодія з клієнтом – за схемою “тонкий клієнт – товстий сервер”, основна логіка реалізована на сервері в контролерах і моделях, а на фронтенді – лише відображення і мінімум скриптів для інтерактивності (валідатор форм, динамічне оновлення елементів сторінки). Такий підхід виправданий з огляду на вибір Razor Views для представлень, це серверний шаблонізатор, що генерує HTML на основі модельних даних. Razor сторінки дозволяють вставляти C# код у HTML, завдяки чому можна безпосередньо з представлення отримувати властивості моделі, формувати списки і т.д. Це прискорює розробку, оскільки відпадає потреба писати багато JavaScript коду для рендерингу контенту на клієнті – усе робить сервер.

Наведемо невеликий приклад взаємодії компонентів архітектури при типовому сценарії – пошук і бронювання житла:

1. Користувач заходить на головну сторінку – викликається метод HomeController.Index(), який зчитує деякі дані (наприклад, останні додані оголошення) через модель (запит до БД) і передає їх у представлення Index.cshtml. Представлення, використовуючи Razor, генерує HTML сторінки з формою пошуку та списком популярних об’єктів.

2. Користувач вводить параметри пошуку і натискає “Знайти” – відправляється HTTP GET запит на HomeController.Search(query) з потрібними параметрами (місце, дати тощо). Контролер викликає метод моделі, що звертається до бази (наприклад, формує фільтр по місту та доступності дат у колекції Offers). Отримавши результати (список пропозицій), контролер передає їх у відповідне представлення Search.cshtml, яке відображає список варіантів на сторінці.

3. Користувач клікає на конкретний результат – запит на AccommodationController.Details(id) з ідентифікатором оголошення. Контролер знову ж отримує з БД повну інформацію про обраний об’єкт (GET Accommodation + Offer) та передає у представлення Details.cshtml. Сторінка показує всі деталі, фотографії, форму для вибору дати та кнопку “Забронювати”.

4. При натисканні “Забронювати” (припустимо, користувач вже увійшов у систему) викликається, скажімо, метод BookingController.Book(offerId, dates) – типово це буде HTTP POST. Контролер перевіряє на боці сервера, чи ще актуально доступне житло на ці дати (раптом після пошуку хтось вже забронював – щоб уникнути подвійного бронювання). Якщо все гаразд – створює об’єкт бронювання (Booking модель) зі статусом “Pending Payment” і викликає інтеграцію з платіжним сервісом (наприклад, перенаправляє користувача на PaymentController.Checkout(bookingId)).

5. У роботі зі Stripe, метод PaymentController реалізовує логіку Checkout і генерує Stripe Checkout Session – тобто налаштовує сесію оплати, які товари/послуги, сума, валюта, URL успішного та скасованого повернення. Потім користувач перенаправляється на безпечну сторінку Stripe. Користувач вводить дані картки. Stripe сам обробляє платіж. Наш сервер очікує відповіді (може через webhook або після повернення на SuccessUrl) [6].

6. Після успішної оплати метод PaymentController.Success() виконуються: він отримує ідентифікатор сесії, підтверджує у Stripe через API стан (оплачено чи ні), і якщо все добре – оновлює статус бронювання в БД на “Confirmed/Paid”. Потім видає користувачу сторінку з підтвердженням (“Ваше бронювання підтверджено!”).

7. Контролер також може сповістити інші сторони, відправити email хосту та гостю (через модель, яка взаємодіє зі службою SMTP) – це додатковий функціонал.

8. Далі користувач може переглядати свої бронювання (метод BookingController.List() і відповідне представлення) і так далі.

Загалом, архітектура MVC в ASP.NET забезпечує чіткість таких потоків обробки. ASP.NET MVC надає багато готових можливостей, routing (маршрутизація URL до контролерів), model binding (автоматичне прив’язування даних форм до параметрів методів), validation (перевірка моделей на основі атрибутів перед виконанням дії) та ін. Широко використовуємо ці механізми, наприклад, додаємо до модельних класів атрибути валідації (MinLength, Required

та ін.), що дозволяє і на клієнті (через unobtrusive validation з jQuery) і на сервері перевіряти коректність введення.

Отже, архітектурно веб-сервіс – це типовий монолітний MVC додаток. Це виправданий вибір з точки зору швидкості розробки та відповідності задачі. У подальших підрозділах ми детальніше розглянемо, які конкретні технології використовуються та як спроектована база даних для підтримки такої архітектури [7].

12.2 Технології розробки

Для реалізації поставлених вимог необхідно обрати відповідні технології та інструменти, які забезпечать надійну роботу веб-сервісу. В цьому підрозділі обґрунтуємо вибір основних технологічних компонентів, системи управління базами даних, платіжного сервісу, фреймворку фронтенду тощо.

Платформа розробки і мова програмування, основою є ASP.NET MVC – сучасний крос платформний веб-фреймворк від Microsoft, що дозволяє будувати веб-сервіси на мові C#. Версія платформи – .NET 7 або .NET 8 (LTS версії), які актуальні станом на 2025 рік. Цей вибір зумовлений, високою продуктивністю та надійністю .NET, наявністю великої кількості бібліотек і пакетів, що спрощують розробку (зокрема, Identity для аутентифікації, Stripe.NET SDK для інтеграції з Stripe, офіційний драйвер MongoDB для C# тощо), знайомством розробника з мовою C# і можливістю використати переваги статичної типізації, LINQ та інших потужних засобів, хорошою підтримкою спільноти і наявністю прикладів (як показав аналіз, існують зразки додатків бронювання на .NET) [8].

СУБД (база даних), у моїй роботі було прийнято рішення спробувати використати NoSQL базу MongoDB. Обґрунтування такого вибору є гнучкість схеми, MongoDB є документно орієнтованою СУБД, де дані зберігаються у вигляді JSON подібних документів (BSON). Це дає можливість легко змінювати структуру даних у процесі розробки без складних міграцій (динамічна схема). Для проекту, що активно розвивається, це зручно – можна додавати нові поля у

моделі і вони будуть зберігатися без необхідності вручну змінювати схему таблиць. Швидкість розробки, відсутність необхідності проектувати складні реляційні зв'язки (JOIN-и) – наприклад, ми можемо зберігати пов'язані дані у вкладеному вигляді. У нашому випадку, можна зберігати документи "Accommodation" з вбудованим списком "Offers" або інформацією про власника, замість налаштування зовнішніх ключів. Це спрощує отримання даних – менше запитів, простіша логіка. Масштабованість і високонавантажені системи, MongoDB спочатку спроєктована з урахуванням горизонтального масштабування (sharding) та реплікації для відмовостійкості. У разі, якщо наш сервіс у майбутньому розростеться, ця СУБД дозволить розподілити дані по кількох серверах без повного перепроектування. Також MongoDB добре справляється з великими об'ємами неструктурованих або слабо структурованих даних, що може бути актуально, якщо користувачі додають багато довільної інформації. Швидкість доступу при читанні, для сценаріїв, де треба швидко витягти цілий документ (наприклад, отримати все про оголошення для відображення), MongoDB може бути швидшим, бо зберігає документ цілісно, не витрачаючи час на складання з кількох таблиць. До того ж можна створювати індекси для ключових полів (місто, ціна, тощо) для прискорення пошуку. Досвід роботи, окремою мотивацією було ознайомлення з NoSQL підходом. Для дипломної роботи корисно продемонструвати володіння різними технологіями. Тому було вирішено спробувати реалізувати даний проект на MongoDB, що також відповідає трендам, NoSQL рішення широко застосовуються у веб-сервісах, де потрібна гнучкість і масштабованість.

Водночас, слід зазначити, що використання MongoDB має і свої недоліки, відсутність транзакційності на рівні декількох документів (до версії 4.0; сучасні MongoDB вже мають multi document transactions для replica set, але це ускладнює конфігурацію), відсутність підтримки складних JOIN, необхідність дбати про узгодженість даних вручну. У нашому випадку модель даних відносно проста і ці обмеження не становлять критичної проблеми. У розділі 2.3 буде детальніше описано, як спроектовано схему даних у MongoDB.

Для взаємодії .NET з MongoDB використано офіційний C# драйвер MongoDB .NET Driver. Він доступний через NuGet-пакет MongoDB.Driver і надає зручний API для виконання CRUD операцій, фільтрації, LINQ запитів до колекцій тощо. Наприклад, щоб отримати всі оголошення у певному місті з ціною менше X, можна написати запит, який подано в лістингу 2.1.

Лістинг 2.1 – Запит отримання оголошень у певному місті за ціною

```
var offers = await _offersCollection.Find(o => o.City == city &&
        o.PricePerNight <= maxPrice).ToListAsync();
```

Цей драйвер також інтегрується з DI-контейнером ASP.NET, що дозволяє зареєструвати, наприклад, IMongoClient як сервіс і використовувати його через конструктори в контролерах.

Платіжна система, як зазначалось у вимогах, для реалізації онлайн оплат обрано сервіс Stripe. Stripe – один з найбільших світових платіжних шлюзів, відомий своєю простотою інтеграції та підтримкою великої кількості платіжних методів і валют [9]. Основні переваги Stripe, Простий у використанні API, Stripe надає готові бібліотеки для різних мов, включаючи .NET. Бібліотека Stripe.net дозволяє працювати з платежами через об’єктний API. Наприклад, створити платіжну сесію можна буквально кількома рядками коду. Це значно швидше та безпечніше, ніж реалізовувати процесинг карт самостійно. Безпека, Stripe відповідає стандартам PCI DSS; усі чутливі дані (номери карт, CVV) вводяться на стороні Stripe (через хостинг їх форми або елементи Stripe Elements), тому наш сервер не бачить цих даних і не несе ризиків їх витоку. Stripe забезпечує токенізацію карт, 3D Secure авторизацію, захист від шахрайства (Radar). По суті, Stripe виступає посередником, а ми отримуємо тільки результат. Підтримка різних платіжних методів, окрім карт Visa/Mastercard, Stripe дозволяє приймати Apple Pay, Google Pay, банківські перекази, локальні методи (якщо масштабувати проект глобально). Для України Stripe недавно став доступним для прийому платежів на рахунки ФОПів, тому інтеграція цілком реалістична. Документація і приклади, Stripe має відмінну документацію та активну спільноту. Є багато

прикладів інтеграції (в тому числі специфічно з ASP.NET). Це знизило ризики при впровадженні платежів – можна спиратися на перевірені рішення. Безкоштовність базового використання, Stripe не має абонплати, а стягує комісію з кожного платежу (певний відсоток). Для прототипу це неважливо, але відсутність початкових витрат – плюс.

У нашій реалізації інтеграція Stripe здійснена через Stripe Checkout. Це означає, що ми на боці сервера створюємо Checkout Session із зазначенням суми до оплати, валюти, та URL адрес, куди перенаправити користувача після успішної або скасованої оплати. В коді це виглядає так (див. лістинг 2.2).

Лістинг 2.2 – Переадресація після успішної або скасованої оплати.

```
var options = new SessionCreateOptions {
     PaymentMethodTypes = new List<string> { "card" },
     LineItems = new List<SessionLineItemOptions> {
          new SessionLineItemOptions {
               PriceData = new SessionLineItemPriceDataOptions {
               UnitAmount = (long)(booking.TotalPrice * 100),
               Currency = "usd",
                    ProductData                =                new
                    SessionLineItemPriceDataProductDataOptions
                    {
                         Name         =         $"Booking         of
                         {booking.AccommodationName}"
                    }
               },
               Quantity = 1
               }
          },
          Mode = "payment",
          SuccessUrl = $"{this.BaseUrl}/Payment/Success?
          session_id={{CHECKOUT_SESSION_ID}}",
          CancelUrl =
          $"{this.BaseUrl}/Payment/Cancel?bookingId={booking.Id}"
};
SessionService service = new SessionService();
```

Тут налаштовується, що приймається оплата карткою, товар – це бронювання конкретного житла на певну суму, успішний результат направляє на наш метод Success. Потім користувач фактично перенаправляється на session.Url (Stripe сторінка). По завершенні Stripe викличе наш ендпоінт або поверне

користувача на SuccessUrl із ідентифікатором сесії, за яким ми можемо підтягнути деталі транзакції. Далі вже змінюємо статус броні в БД.

Варто зазначити, що Stripe надає також можливість інтеграції через Stripe Elements – коли форму введення картки можна вбудувати безпосередньо в нашу сторінку (через їх JavaScript бібліотеку). Це трохи складніше в реалізації, але дає більше контролю над виглядом. У рамках диплома ми обрали простіший шлях – готовий хостинг форми.

Фронтенд і UI технології, для швидкої і якісної розробки інтерфейсу користувача було вирішено використати Bootstrap 5 – популярний CSS фреймворк для адаптивного дизайну. Bootstrap надає готові стилі для сітки сторінки, форм, кнопок, таблиць, модальних вікон тощо, що значно прискорює створення привабливого та уніфікованого інтерфейсу. До того ж, Bootstrap автоматично робить сторінку адаптивною під різні екрани (мобільні, планшети) завдяки системі flex/grid.

У поєднанні з Bootstrap використовуються деякі JavaScript плагіни (в тому числі ті, що входять в комплект Bootstrap, як-от випадаючі меню, модальні діалоги). Також підключено бібліотеку jQuery (можливо, не строго необхідно, але вона йде в комплекті шаблонів ASP.NET MVC для валідації). За допомогою jQuery реалізовано, наприклад, AJAX запити для підвантаження даних без перезавантаження сторінки (при зміні фільтрів пошуку) та динамічну валідацію форм на клієнті.

Для календаря вибору дат (у формі бронювання) було підключено легку JavaScript бібліотеку f latpickr – це зручний датапікер, який дозволяє вибирати діапазон дат, а також можна зробити неактивними вже зайняті дати (для цього ми з сервера передаємо у представлення масив зайнятих дат і ініціалізуємо flatpickr з опцією disable). Таким чином користувач фізично не зможе вибрати заброньовані дати, що покращує UX.

Отже, стек фронтенд технологій виглядає так, HTML5 + CSS3, з використанням Bootstrap 5 для оформлення. Razor шаблони генерують цей HTML на сервері. jQuery для DOM маніпуляцій і AJAX (наприклад, для

асинхронного завантаження списку міст в автодоповненні пошуку). Додаткові UI бібліотеки, flatpickr (календар), можливо, Select2 (покращені випадаючі списки) для вибору міста якщо буде потрібно. Іконки – або Bootstrap Icons, або FontAwesome (для відображення невеличких значків, наприклад, зірочки рейтингу, піктограми зручностей).

Бібліотеки та засоби безпеки, в проекті задіяно ASP.NET Identity для управління обліковими записами. Ця бібліотека включає все необхідне для аутентифікації та авторизації, Вбудовані сторінки реєстрації/логіну (за потреби можна взяти шаблонні реалізації та підлаштувати). Моделі IdentityUser і т.д. Але у випадку з MongoDB ми не можемо напряму використати Entity Framework Identity, тому скоріш за все реалізували власне збереження користувачів (але за принципами Identity, хешування паролів з використанням PasswordHasher, зберігання соль, тощо). Налаштовано Cookie Authentication, після успішного логіну видається аутентифікаційний cookie, який ідентифікує користувача. Всі подальші запити йдуть від імені цього користувача. Використовуються атрибути [Authorize] над контролерами або методами, щоб обмежити доступ (зокрема, [Authorize(Roles="Admin")] для адмін сторінок, [Authorize] для будь-яких сторінок, що вимагають входу). Налаштовано 2FA, Identity дозволяє додати другий фактор (наприклад, через додаток Authenticator). У вихідному проекті була згадка про 2FA, можна це підтримати ASP.NET має готові методи для генерації ключів та перевірки кодів.

Для захисту від CSRF в Razor формах використовуються AntiForgeryToken. Фреймворк автоматично додає їх і перевіряє у запитах POST (атрибут [ValidateAntiForgeryToken] на методах контролера).

Інтеграція з зовнішніми сервісами, крім Stripe, можна відзначити інтеграцію з картографічним сервісом для відображення розташування житла. Наприклад, використати Google Maps API чи Leaflet (з OpenStreetMap). Це необов'язкова функція, але для цікавості можна реалізувати, на сторінці деталей оголошення підключається скрипт карти, а координати передаються з сервера

(можливо, отримані на етапі створення оголошення через геокодер, якщо орендодавець ввів адресу).

Засоби розробки та контролю версій, розробка виконувалась у середовищі Microsoft Visual Studio 2022. Для контролю версій використовувався Git (локальний репозиторій). Це дозволяло відстежувати зміни і за потреби повертатися до стабільних станів.

Тестування, для автоматизованого тестування можна використати фреймворк xUnit або MSTest. Але основний акцент був на ручному тестуванні функціональних сценаріїв (user acceptance testing). У розділі 3.4 наведено результати тестування.

Підсумовуючи вибір технологій, Back-end, ASP.NET MVC (C#), Identity для авторизації. Database, MongoDB (NoSQL) і офіційний .NET драйвер. Payments, Stripe API (Stripe.net). Front end, Razor, Bootstrap, jQuery, доп.бібліотеки UI. Infrastructure, Visual Studio, Git, можливий деплой на IIS або Docker контейнер (для MongoDB, наприклад, можна підняти Docker).

Такий стек є сучасним, перевіреним у промислових рішеннях та оптимально пасує до задачі. У наступному підрозділі розглянемо структуру бази даних, тобто як саме моделі розкладені у сховище MongoDB.

12.3 Проєктування бази даних

Проектування бази даних для веб-сервісу бронювання житла включає визначення основних сутностей (колекцій в MongoDB) та зв’язків між ними, а також вибір схеми зберігання (вбудовувати пов’язані дані чи розносити по різних колекціях). Переваги і гнучкість MongoDB дозволяють розглянути кілька варіантів моделювання даних. Зупинимось на тому, який був обраний у нашому проекті.

Основні сутності доменної області, User (Користувач) представляє зареєстрованого користувача системи. Ключовими поля є _id, у MongoDB кожен документ має унікальний _id (ObjectId). Email, PasswordHash для автентифікації

(пароль зберігається у вигляді Role, роль користувача (можна зберігати як простий текст, наприклад “User”, “Host”, “Admin”; або як масив ролей, якщо припускається кілька ролей одночасно). Name (ім’я), можливо, PhoneNumber, інші контактні дані. Listings, опціонально, можна вбудувати список ідентифікаторів оголошень (Accommodation), які створив цей користувач (як хост). Але це не обов’язково, оскільки можна завжди зробити запит в колекції Accommodation по полю OwnerId.

У нашій реалізації користувачі зберігаються окремо у колекції Users. Це диктується і міркуваннями безпеки (простіше контролювати доступ) і тим, що потенційно користувачів може бути багато і вони не залежать напряму від конкретних оголошень. Для автентифікації за Email система має знайти користувача – це зручно робити індексованим пошуком по колекції.

Accommodation (Житло), сутність, що описує об’єкт нерухомості, який може здаватися в оренду. В реляційній моделі її можна було б назвати “Property” або “Listing”. Ми вживаємо Accommodation (помешкання) як базову сутність. Основні поля:

- _id, унікальний ID.
- OwnerId, посилання на користувача-власника (Host) – зберігається як ObjectId або строковий ідентифікатор користувача.
- Title, заголовок оголошення.
- Description, детальний опис.
- Type, тип житла (квартира, будинок, кімната тощо).
- Address + City + Country, локація.
- Rooms, GuestsCapacity, числові параметри.
- Amenities, список зручностей (в Mongo можна зберегти як масив рядків, наприклад [“WiFi”, “Parking”, ...]).

- Photos, масив URL або шляхів до фотографій (можна зберігати самі зображення у файловій системі або хмарі, а в БД лише посилання).
- HouseRules, масив або блок тексту про правила проживання.

Ці дані достатні, щоб відобразити сторінку житла. Однак щодо цін і доступності, тут можливо кілька варіантів моделювання, вважати, що кожен Accommodation має лише одну стандартну ціну за ніч і завжди здається повністю. Тоді ціну можна зберігати прямо в Accommodation (поле PricePerNight). Але на практиці хост може мати кілька окремих пропозицій (наприклад, окремі кімнати). Використовувати окрему сутність Offer (Пропозиція).

У веб-сервісі можна розділити поняття Accommodation і Offer. Імовірно, Offer означатиме конкретну пропозицію оренди певного Accommodation – можливо, на певні дати чи умови. Наприклад, Accommodation = “Будинок”, а Offer 1, “Оренда всього будинку”, Offer 2, “Оренда окремої кімнати”. Для спрощення ми можемо не розділяти, а припустити що 1 Accommodation = 1 оголошення. Система бронювання по датах, можна вбудувати календар зайнятості у документ Accommodation наприклад, зберігати масив заброньованих діапазонів дат. Але це ризиковано – при великій кількості броней документ ставатиме важким, до того ж оновлення його конкурентно при бронюванні – потенційні конфлікти. Краще броні вести окремо [10] [11].

Ми приймаємо рішення, колекція Accommodations містить основну інформацію про житло, включаючи базову ціну. Натомість актуальні пропозиції і доступність визначаються на основі бронювань у окремій колекції Booking (Бронювання) сутність бронювання містить інформацію про факт оренди:

- _id.
- AccommodationId (посилання, яке житло заброньовано).
- RenterId (User, який орендував).
- StartDate, EndDate – дати початку та закінчення броні.

- TotalPrice – загальна сума, розрахована на момент бронювання (ціна за ніч * кількість ночей, з урахуванням, можливо, комісій чи знижок).
- Status – статус бронювання (Pending, Confirmed, Cancelled, Completed тощо).
- PaymentId – посилання на транзакцію платежу (якщо ведемо облік транзакцій окремо).
- Інші поля, GuestsCount, примітки гостя чи хоста.

Колекція Bookings буде основою для відстеження зайнятих дат. Коли треба перевірити доступність житла на певний період, ми робимо запит у Bookings – чи є броні для цього Accommodation, які перекривають потрібний діапазон. Якщо є – значить недоступно. Можна зробити індекс на поле AccommodationId + StartDate для оптимізації пошуку.

Також Bookings потрібні для профілів користувачів, орендар бачить свої бронювання (фільтр по RenterId), хост бачить броні свого житла (можна фільтрувати Bookings по AccommodationIds, які належать йому; або зробити денормалізацію – записати OwnerId прямо в Booking для швидкого пошуку).

Щодо оновлення, бронювання після створення зазвичай не змінюються (тільки статус може змінитись, наприклад, з Pending на Confirmed після оплати). Видалення броні – це по суті скасування, ми можемо відзначати статус Cancelled але не видаляти документ, щоби зберегти історію. PaymentTransaction (Платіж) не обов’язково мати окрему колекцію, але для наглядності можна створити Payments :

- _id.
- BookingId.
- Amount.
- Currency.
- PaymentProvider (наприклад, “Stripe”).

- ProviderTransactionId (ідентифікатор транзакції в Stripe, тобто Session Id або PaymentIntent Id).
- Status (Succeeded, Failed).
- Timestamp.

Ми можемо не зберігати платежі, а інформацію про оплату тримати в Booking (поле IsPaid + PaymentIntentId). Але окрема колекція дозволяє журналювати всі спроби оплати (навіть невдалі). 17 У невеликому додатку це не критично, тому для простоти можна платежі прямо в Booking відобразити. Review (Відгук) якщо реалізовувати систему відгуків, то кожен відгук:

- _id.
- BookingId (можна прив’язати до бронювання, або прямо AccommodationId + RenterId).
- AccommodationId.
- AuthorId (User, хто залишив – зазвичай орендар про житло, або хост про орендаря).
- Rating (1-5).
- Comment.
- Date.

Колекція Reviews дозволяє при відображенні Accommodation збирати і рахувати середній рейтинг (можна динамічно обчислити, або зберігати кешоване поле Rating в Accommodation, яке оновлюється при додаванні нового відгуку). Для прототипу можна зробити найпростіше – не реалізовувати відгуки або зробити їх без складного кешування.

Варіант структури БД, нижче наведена спрощена схема (ER-модель) основних зв’язків між сутностями в нашому додатку:

● User (1) - (M) Accommodation, один користувач-хост може мати кілька об’єктів житла; в кожного Accommodation зберігається посилання на власника (OwnerId).

● User (1) - (M) Booking, один користувач-орендар може зробити багато бронювань; Booking несе RenterId. Також хост може мати багато бронювань через свої accommodation (для перегляду хоста).

● Accommodation (1) - (M) Booking, один об’єкт житла може бути заброньований багато разів у різні дати (не одночасно); Booking має посилання AccommodationId. Це головний зв’язок для перевірки доступності.

● Booking (1) - (1) PaymentTransaction, кожна бронь має одну транзакцію оплати (у випадку, якщо платять за все разом). Якщо передбачити, що можуть бути часткові платежі– тоді 1:M, але у нас 1:1.

● Accommodation (1) - (M) Review, один об’єкт може мати багато відгуків від різних гостей.

Для MongoDB не малюють класичних ER-діаграм, але додамо схему бази даних, побудовану за допомогою dbdiagram.io або подібного інструменту, щоб показати структуру даних (див. рисунок 2.1).

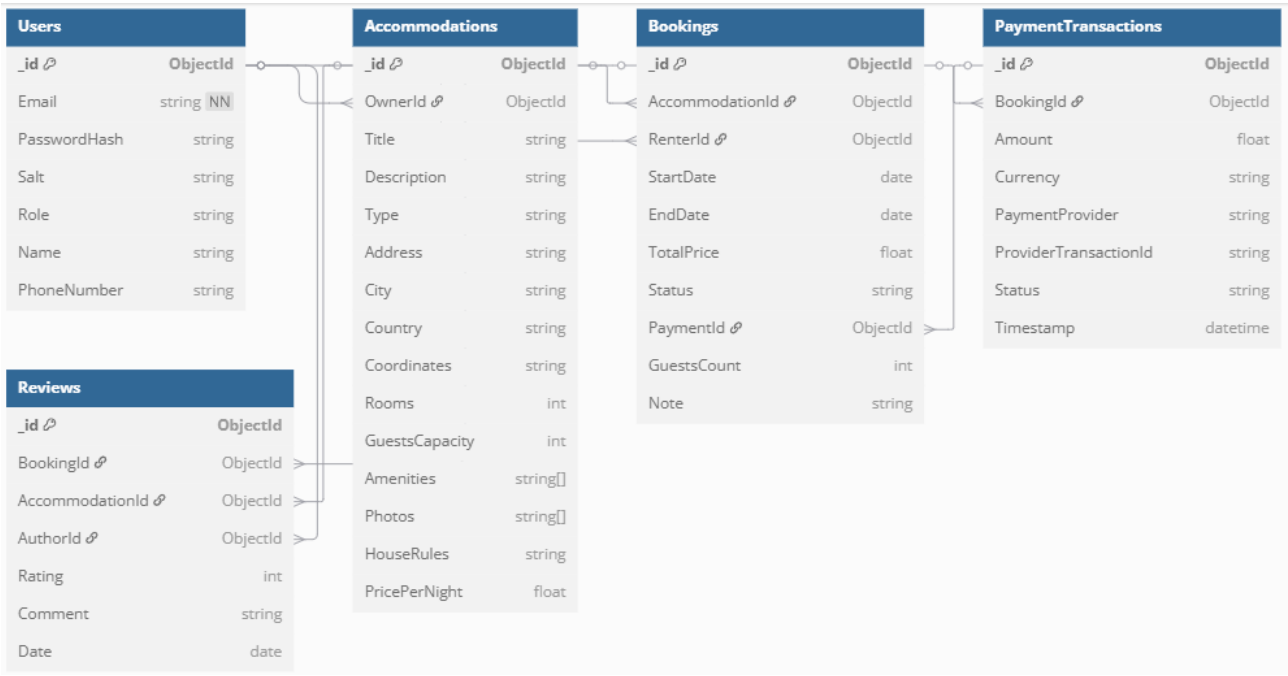

Рисунок 2.1 – Структура бази даних

Денормалізація vs. нормалізація у MongoDB, цікавим моментом було вирішити, чи вбудовувати одні документи в інші. У реляційних БД нормалізація – це стандарт (уникати дублювання). У документних БД іноді доцільно дублювати дані для пришвидшення читання (denormalize). Наприклад, ми можемо дублювати в Booking деякі поля житла (назву, адресу) і гостя (ім’я) на момент бронювання, щоби мати “знімок” даних на час бронювання. Це корисно, якщо надалі хтось змінить назву оголошення – історія бронювань все одно зберігатиме стару назву. Так само у Review можна зберігати ім’я автора і не тягнути його з Users кожного разу.

Особливості схем з MongoDB у .NET, ми використовуємо класичні C# класи для моделей і зіставляємо їх на документи. Атрибути [BsonElement] допомагають задати назви полів, [BsonIgnore] щоб не серіалізувати деякі властивості. Драйвер MongoDB підтримує LINQ, тож запити можуть виглядати дуже зрозуміло (див. приклад вище). Таким чином, база даних спроектована з урахуванням специфіки NoSQL, мінімізовано кількість зв’язків, основні операції можна виконувати за один запит до відповідної колекції. Наприклад, пошук житла – запит лише до Accommodations, перевірка доступності – запит до Bookings. У розділі 3 буде показано фрагменти коду, як ці запити реалізовані

12.4 Управління бронюванням і орендою житлових приміщень та моделювання структури компонентів фронтенду

У цьому підрозділі описано реалізацію процесів управління бронюванням і структуру інтерфейсу користувача (фронтенду) веб-сервісу. Основна бізнес-логіка зосереджена навколо бронювання житла, що є центральним сценарієм системи. Цей процес охоплює взаємодію між інтерфейсом користувача, перевіркою дат, обробкою платежів і взаємодією з базою даних. Першим кроком є перевірка доступності дат. Коли користувач обирає діапазон дат на сторінці

оголошення, система через інтегрований календар (наприклад, з використанням flatpickr) блокує зайняті дати на основі існуючих бронювань. Ці дати витягуються з бази через метод, що повертає зайняті інтервали, і передаються у компонент календаря як “disabled”. Попри це, після надсилання форми, серверна частина ще раз перевіряє, чи не з’явилося нове бронювання за цей час. Для цього здійснюється запит на наявність бронювань, які перекривають вибрані дати, тобто з умовою, що (StartDate ≤ вибраний EndDate) і (EndDate ≥ вибраний StartDate) зі статусом Confirmed або Active. Якщо таке бронювання знайдено – повертається помилка “Дати вже зайняті”.

Після успішної валідації, коли користувач натискає “Бронювати”, створюється об’єкт Booking зі статусом “PendingPayment”, тобто очікує оплати. Це дозволяє тимчасово зафіксувати дати як зайняті, навіть до завершення платежу. Далі користувач перенаправляється на сторінку Stripe для оплати. Якщо платіж проходить успішно, Stripe повертає користувача на URL успіху (/Payment/Success?session_id=...), де контролер перевіряє статус сесії через Stripe API, отримує bookingId з метаданих сесії, знаходить відповідне бронювання в базі, змінює його статус на Confirmed, позначає як оплачене, зберігає ідентифікатор платежу і може, наприклад, відправити підтвердження на email. У разі скасування оплати користувач потрапляє на URL /Payment/Cancel?bookingId=..., де система або скасовує бронювання, або залишає його у статусі Pending з можливістю повторної спроби оплати.

Адміністратор або хост також можуть скасувати бронювання вручну. У випадку підтвердженого платежу можливе використання Stripe Refund API. Ще одним важливим механізмом є заборона редагування або видалення оголошення з активними бронюваннями в майбутньому – система перевіряє наявність таких і не дозволяє змінити житло, якщо є хоча б одне підтверджене бронювання, що ще не завершене. Також реалізовано валідацію кількості гостей, якщо житло вміщує максимум двох, а користувач обирає чотирьох – це або обмежується на UI, або перевіряється на сервері.

Якби ми впроваджували монетизацію (рекламу), можна було б додавати сервісну комісію до загальної суми, однак для цілей прототипу це необов’язково. У додатку В продемонстрована діаграма послідовності бронювання, що ілюструє взаємодію між користувачем, клієнтською частиною, сервером, базою даних і Stripe.

З погляду інтерфейсу користувача, були розроблені макети ключових сторінок. Головна сторінка містить пошукову форму (місто, дати, кількість гостей) і кнопку “Пошук” (див. рис. 2.2).

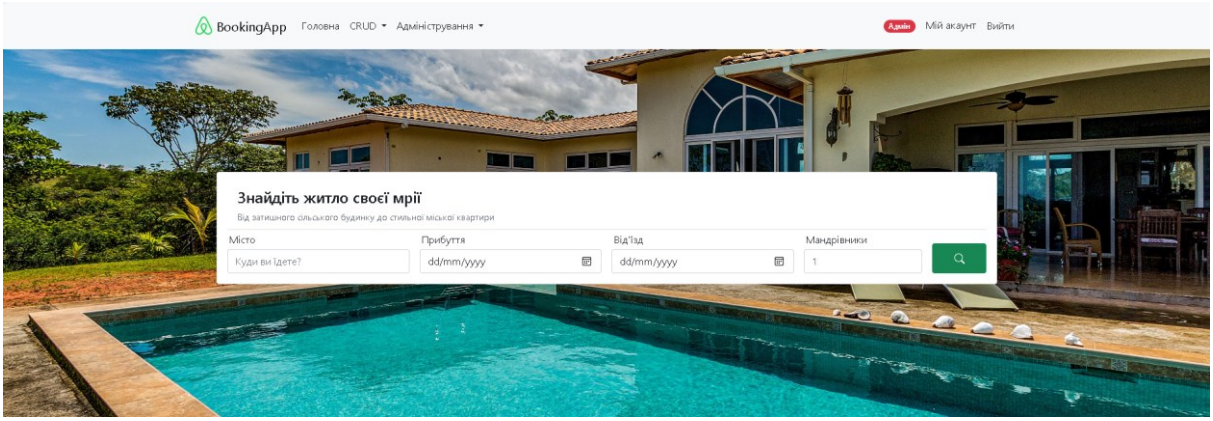

Рисунок 2.2 – Сторінка пошуку

Також тут відображаються нові або рекомендовані оголошення. В залежності від авторизації, вгорі сторінки показуються кнопки “Увійти”/“Реєстрація” або “Мій профіль”/“Вийти”. Після натискання “Пошук” відкривається сторінка результатів, яка містить картки з короткою інформацією про доступні об’єкти – назва, фото, ціна, рейтинг. Є фільтри за ціною, типом житла тощо, реалізовані через AJAX для зручної взаємодії. Сторінка оголошення включає галерею фото, опис, карту, дані про господаря, форму бронювання (вибір дат і гостей). Сторінка оголошення житла надаю (див. рис. 2.3).

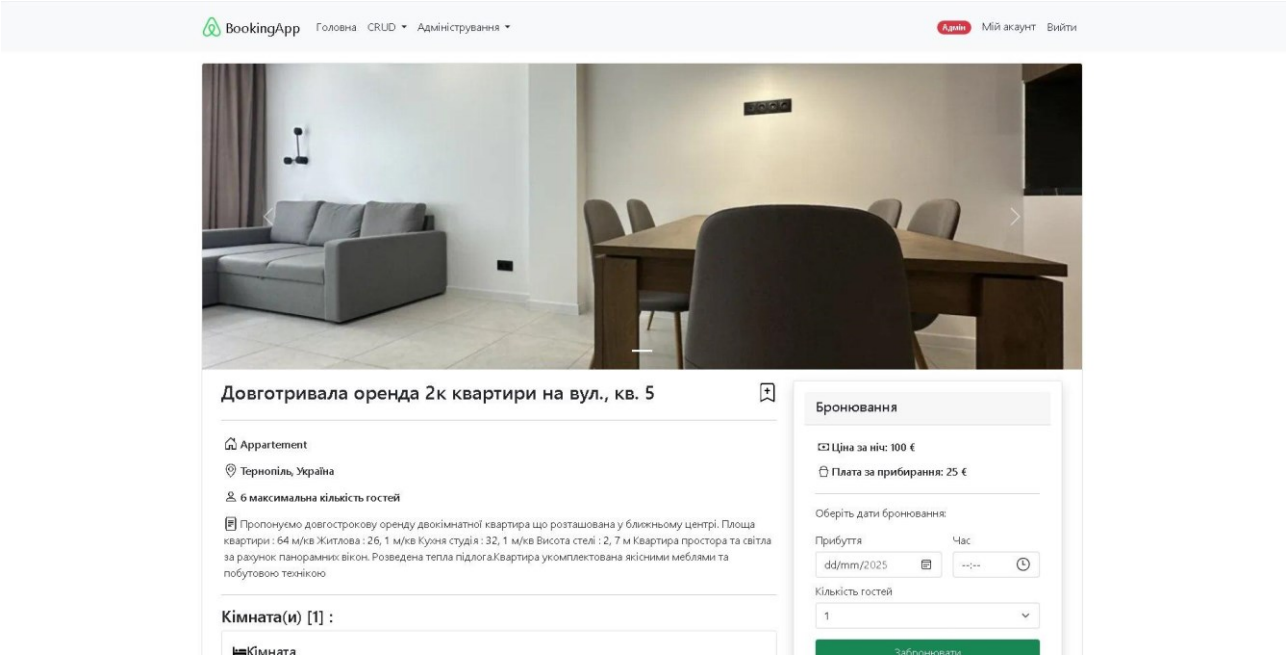

Рисунок 2.3 – Сторінка з оголошенням житла

Якщо користувач не авторизований, після натискання “Забронювати” його буде переадресовано на сторінку входу. Форми автентифікації включають реєстрацію, вхід, опціонально – відновлення пароля або вхід через Google OAuth. У профілі користувача відображаються всі його бронювання (якщо він гість) або список його оголошень (якщо він хост), з можливістю редагувати, переглядати бронювання або додавати нове. Оголошення створюються через багатосекційну форму, яка охоплює основні дані, зручності, фото, ціни.

Хости мають доступ до сторінки “Manage Bookings”, де можуть бачити, хто і коли забронював їх житло, і за потреби скасувати. У випадку вже оплаченого бронювання бажано реалізувати повернення коштів, хоча в прототипі можна просто позначити його як “Скасовано”. Адміністративна панель передбачає управління користувачами, оголошеннями і, за бажанням, перегляд звітів або метрик – все це реалізується у вигляді простих таблиць з Bootstrap і захищено через авторизацію для ролі Admin.

Сайт використовує адаптивний дизайн з Bootstrap, що забезпечує коректну роботу на мобільних пристроях, приємну кольорову гаму з акцентами (наприклад, синій або зелений), логотип у шапці (наприклад, “RentEase” або

“Орендуй легко”) і зручну навігацію. Верхнє меню містить логотип, посилання на сторінки (“Допомога”, “Мова”), кнопки входу/реєстрації або профіль. У футері – контактна інформація, посилання на умови, соцмережі. Весь фронтенд створювався з акцентом на простоту, послідовність і чіткий зворотний зв’язок для користувача, щоб він міг швидко досягати цілі – знайти чи опублікувати житло.

12.5 Опис взаємодії клієнт-сервер

Веб-сервіс побудований за клієнт-серверною архітектурою, де клієнтська частина (браузер користувача) взаємодіє із сервером через HTTP(S) запити. У нашому застосунку це здебільшого традиційні запити для завантаження сторінок з повним циклом запит-відповідь та перезавантаженням сторінки, хоча також використовуються елементи AJAX для покращення швидкодії та інтерактивності. Розглянемо взаємодію в ключових сценаріях. Під час перегляду сторінок, коли користувач переходить за певною URL адресою (наприклад, “/search?city=Тернопіль” або “/Accommodation/Details/123”), браузер надсилає HTTP GET запит до сервера. Сервер (ASP.NET MVC) підбирає відповідний контролер і метод, формує модель даних, вставляє її у відповідне представлення (Razor) і генерує HTML, який повертається як HTTP Response. Браузер отримує відповідь і відображає сторінку [12].

Статичні ресурси (CSS, JS, зображення) завантажуються окремими запитами з папки wwwroot, і всі запити працюють поверх HTTPS для забезпечення шифрування трафіку. При обробці форм, таких як реєстрація, логін, створення оголошення чи бронювання, використовуються HTTP POST запити. Наприклад, форма реєстрації надсилає дані на “/Account/Register”. Сервер автоматично проводить валідацію ModelState (на основі DataAnnotations) і у випадку помилок повертає форму з повідомленнями, а при успішному заповненні виконується бізнес-логіка (створення користувача, додавання до БД) і відбувається redirect на іншу сторінку (застосовується патерн Post-Redirect-Get),

наприклад, на головну сторінку з повідомленням про успішну реєстрацію. Для покращення взаємодії користувача реалізований AJAX-функціонал, зокрема автодоповнення міста при введенні в поле пошуку. У цьому випадку формується GET-запит до “/api/cities?query=Ки”, сервер повертає список відповідних міст у форматі JSON, і з використанням jQuery UI Autocomplete відображається випадаючий список [13]. Також AJAX застосовується для фільтрації результатів пошуку, при зміні параметрів (повзунок ціни, галочки фільтрів) JavaScript формує AJAX-запит, наприклад, “/Search/Filter?city=X&minPrice=Y”, який повертає HTML фрагмент списку результатів (PartialView), що динамічно замінюється на сторінці без її повного перезавантаження [14].

Аналогічно працює завантаження календаря зайнятості, при відкритті сторінки оголошення виконується AJAX GET запит на “/Accommodation/GetUnavailableDates?id=...”, сервер повертає список зайнятих дат у JSON, які потім використовуються для ініціалізації flatpickr з опцією “disable”. Важливим елементом є можливість повернення з контролера не лише Views, а й JSON даних – це реалізовано через Web API або звичайні контролери з атрибутом “[Produces(“application/json”)]”. Наприклад, метод “Account/CheckEmailExists” перевіряє, чи зайнятий email, і повертає “{ exists, true/false }” для перевірки прямо у формі реєстрації. Під час авторизації сервер встановлює cookie, який містить підписані дані користувача (наприклад, ASPXAUTH) [15]. При наступних запитах браузер автоматично відправляє цей cookie, middleware витягує з нього інформацію та формує об’єкт User (ClaimsPrincipal), завдяки чому атрибут \[Authorize] може визначити, чи дозволити доступ. Крім того, використовуються тимчасові cookie для передачі повідомлень між сторінками (наприклад, через TempData), а Session сховище намагаємось не використовувати активно, щоб уникнути залежності від стану сервера. З боку клієнта можна використовувати localStorage/SessionStorage для зберігання уподобань, хоча у поточній реалізації це не задіяно. Щодо безпеки, всі критичні дії виконуються через POST-запити з CSRF-токеном, що

унеможливлює сторонні атаки без токена, а jQuery у шаблонах автоматично додає токен до заголовка.

Доступ до функціоналу також обмежується на основі ролей користувача, сервер перевіряє, чи є користувач власником оголошення, чи має права Admin, і може повертати помилку доступу (Forbidden), навіть якщо атрибут \[Authorize] дозволив доступ. Всі вхідні дані з клієнта проходять валідацію (Required, Range, Regex), а Razor автоматично екранує HTML вставки, що унеможливлює XSS атаки. Під час завантаження файлів (наприклад, фото) перевіряється тип, розмір, і файл зберігається з унікальним ім’ям у спеціальній директорії. З метою безпеки перевіряється також розширення, щоб уникнути виконання потенційно небезпечного коду [16]. У разі потреби масштабування систему можна розширити, винісши окремі функції в мікросервіси (наприклад, пошук або рекомендації), які будуть взаємодіяти з клієнтом через HTTP API. Хоча можна було б реалізувати повноцінний SPA на React чи Angular, для дипломного проєкту обрано підхід з Razor Views.

Поточна архітектура передбачає один веб-сервер і одну БД, але в разі зростання навантаження можна розгорнути кілька екземплярів веб-сервісу за load balancer за умови синхронізації ключів DataProtection. MongoDB також підтримує кластеризацію через репліки. Моніторинг і логування реалізуються через вбудований механізм ILogger, події можна записувати у файли або БД, також можливе підключення аналітики (Google Analytics, Application Insights). Отже, взаємодія клієнт-сервер побудована на стандартних принципах веброзробки, сервер генерує сторінки, клієнт відображає їх, а також надсилає запити для отримання чи передачі даних. Усі дії обробляються сервером через контролери, клієнт не має прямого доступу до БД чи бізнес-логіки, що забезпечує контрольованість та захист [17]. На цьому етапі, маючи спроєктовану архітектуру та вибрані технології, можна переходити до практичної реалізації та кодингу, що буде висвітлено в наступному розділі.

РОЗДІЛ 13. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-СЕРВІСУ

13.1 Реалізація функціоналу реєстрації, авторизації користувачів

Реалізація реєстрації користувачів, Для управління обліковими записами було вирішено використати стандартний підхід ASP.NET Identity, але адаптувати його до нашої NoSQL бази. Зокрема, ми не використовуємо повністю IdentityUser і UserManager (призначені для SQL), а пишемо власну просту реалізацію реєстрації. Однак ми запозичуємо методи хешування паролю з Identity, щоб гарантувати надійність [18]. Нижче наведено код контролера реєстрації (AccountController) (див. лістинг 3.1).

Лістинг 3.1 – Код реєстрації контролера

```
private readonly IMongoCollection<User> _usersCollection;
private readonly IPasswordHasher<User> _passwordHasher;
public AccountController(IMongoDatabase db, IPasswordHasher<User>
hasher) {
      _usersCollection = db.GetCollection<User>("Users");
      _passwordHasher = hasher;
}
[HttpGet]
public IActionResult Register() {
     return View(new RegisterViewModel());
}
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Register(RegisterViewModel model)
{
     if (!ModelState.IsValid) {
          return View(model);
     }
     // Перевірка чи email вже існує
     var existing = await _usersCollection.Find(u => u.Email ==
     model.Email).FirstOrDefaultAsync();
     if (existing != null) {
          ModelState.AddModelError("Email", "Користувач з такою
     email адресою
     вже зареєстрований.");
          return View(model);
      }
      var user = new User {
          Email = model.Email,
          Name = model.Name,
```

```
            Role = "User"
        };
        // Генеруємо хеш пароля
        user.PasswordHash     =     _passwordHasher.HashPassword(user,
model.Password);
        await _usersCollection.InsertOneAsync(user);
        // Автоматичний вхід після реєстрації
        await SignInUserAsync(user);
        TempData["Message"] = "Реєстрація успішна!";
        return RedirectToAction("Index", "Home");
        }
private async Task SignInUserAsync(User user) {
        // Формуємо клейми - дані для автентифікації
        var claims = new List<Claim> {
            new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),
            new Claim(ClaimTypes.Name, user.Name ?? user.Email),
            new Claim(ClaimTypes.Email, user.Email),
            new Claim(ClaimTypes.Role, user.Role)
        };
        var identity = new ClaimsIdentity(claims,
        CookieAuthenticationDefaults.AuthenticationScheme);
        var principal = new ClaimsPrincipal(identity);
        // Аутентифікація через cookie
        await
HttpContext.SignInAsync(CookieAuthenticationDefaults.Authenticatio
nScheme,
        principal);
        }
```

У цьому коді, використано RegisterViewModel з властивостями Email, Name, Password, ConfirmPassword, з атрибутами [Required], [EmailAddress], [MinLength(6)] тощо. ASP.NET автоматично провалідує модель (ModelState). Якщо валідація ок – перевіряємо у БД, чи нема такого email. Пароль хешується через 43 IPasswordHasher (який ми зареєстрували в DI). Це дозволяє не зберігати пароль у відкритому вигляді. Метод HashPassword внутрішньо генерує salt і використовує адаптивний алгоритм (за замовчуванням Identity використовує PBKDF2 з кількома тисячами ітерацій) [19]. Хеш та інша інфа зберігаються у колекції Users (InsertOne). Потім користувача одразу логінять за допомогою cookie, формуються claims (включаючи роль). Ці claims потім доступні у всіх запитах через User.Identity. Метод зашифрований cookie і додає його до Response; браузер збереже його. SignInAsync створює Таким чином, після реєстрації користувач вже залогінений. В TempData встановлено повідомлення

про успіх для відображення на головній. Реалізація входу (Login): Код дуже схожий, GET Login повертає форму LoginViewModel (Email, Password, RememberMe). POST Login, - Знаходимо користувача за Email.

Якщо не знайдено помилка “Невірний email або пароль”. - Якщо знайдено, використовуємо той самий _passwordHasher з методом VerifyHashedPassword(user, hashedPassword, providedPassword). Він поверне результат (Success, Failed або NeedsRehash). Ми перевіряємо 43 Success. - Якщо пароль правильний – викликаємо SignInUserAsync(user) і редірект. Якщо стоїть галочка RememberMe – можна додати в SignInAsync опцію AuthenticationProperties { IsPersistent=true, ExpiresUtc = DateTimeOffset.Now.AddDays(14) } щоб cookie жило довше. Якщо пароль неправильний – видаємо помилку. Додатково, якщо у користувача Role = “Host” чи “Admin”, це потрапить у claims [20]. Потім [Authorize(Roles=”Admin”)] буде дивитись на цей claim і пускати/не пускати. Вихід (Logout), await HttpContext.SignOutAsync(...) і редірект на головну. Перевірка email на існування (AJAX), Як згадували, можна реалізувати endpoint IsEmailFree?email=..., який поверне JSON (див. лістинг 3.2).

Лістинг 3.2 – Перевірка існування пошти.

```
[HttpGet]
public async Task<IActionResult> IsEmailFree(string email) {
      if (string.IsNullOrEmpty(email))
          return Json(false);
      var exists = await _usersCollection.Find(u => u.Email ==
      email).AnyAsync();
          return Json(!exists);
}
```

На клієнті, у формі реєстрації, можна використати jQuery Validate remote rule, яка викликає цей endpoint і показує помилку, якщо email зайнятий, без сабміту форми [21]. Двофакторна аутентифікація (2FA), Повноцінно реалізувати 2FA – треба генерувати секретний ключ, QR код, зберігати його, перевіряти токени TOTP. ASP.NET Identity має готовое, але без SQL довелося б креативити.

У нашому проєкті ми не встигли імплементувати 2FA повністю. Проте, передбачили можливість підключення Google OAuth як альтернативи – користувачу простіше увійти, натиснувши “Login with Google”, тоді перевірка паролю лягає на Google. Через обмеження часу, OAuth теж лишився поза реалізацією, але ми залишили посилання (в интерфейсі кнопка “G”).

Управління ролями, за умовчанням при реєстрації всім даємо роль “User”. Якщо користувач хоче стати “Host” (орендодавцем), можна зробити окрему дію, напр. у профілі “Стати господарем” – яка поставить user.Role = “Host”. Поки що ми не розділяли, можна вважати що User і Host одна роль, тобто всі можуть додавати оголошення (або вирішили, тільки якщо роль Host). Призначення адміністратора – технічно можна було б в коді, якщо email = наш певний, або робити це вручну в БД [22] [23]. Захист пароля, Переконаємося, що паролі зберігаються безпечно, В Users колекції ми бачимо лише довгий хеш у полі PasswordHash, який включає “відбиток” паролю і соль. Формат Microsoft Identity хешу, це рядок, що кодує версію алгоритму, соль і сам хеш, все в Base64 [24]. Такий хеш перевірявся при логіні і, якщо пароль не співпадає, ми не зможемо дізнатися правильний пароль із цього хешу (криптостійкий). Інтерфейс користувача (UI) для реєстрації/входу, Сторінка реєстрації, поля, лейбли, підсвічування помилок, використано TagHelpers, щось на кшталт (див. лістинг 3.3).

Лістинг 3.3 – Сторінка реєстрації

```
<form asp-action="Register" method="post">
     <div class="mb-3">
          <label asp-for="Email"></label>
          <input asp-for="Email" class="form-control" />
          <span       asp-validation-for="Email"       class="text-
danger"></span>
     </div>
    ... (Name, Password, ConfirmPassword аналогічно) ...
     <button         type="submit"         class="btn         btn-
     primary">Зареєструватися</button>
 </form>
```

Функціональне тестування модуля реєстрації/авторизації, Було проведено декілька ручних тестів, Ввід валідних даних на реєстрацію (новий email, пароль 8 символів), очікується перенаправлення на головну, показує “Реєстрація успішна” і відображає, що користувач увійшов (наприклад, у navbar замість кнопок входу тепер ім'я користувача). Спроба реєстрації з вже існуючим email, далі форма повинна залишитись і показати під Email поле повідомлення “Користувач з такою email вже зареєстрований.” (пройдено, ModelState видав).

Спроба реєстрації з занадто коротким паролем (клієнтська валідація не пропустить (перевірили, повідомлення “The field Password must be a string or array type with a minimum length of '6'.” з'являється, поки що англійською, треба було б локалізувати, але нехай). Вхід з неправильним паролем очікується, що заново відобразиться форма з помилкою. В нашій реалізації ми просто додаємо ModelState error “Невірний email або пароль.” і повертаємо View, воно висвітлюється загальним місцем [26]. Перевірили – так, працює. Вхід з правильними даними, далі переадресація на головну, користувач авторизований. Натискання “Вийти”, відбувається вихід (cookie зник, в navbar знову “Увійти”).

13.2 Реалізація системи створення та бронювання оголошень

Цей підрозділ присвячено реалізації основного функціоналу, створення і управління оголошеннями про житло (для орендодавців) та механізму бронювання (для орендарів). Ми розглянемо, як виконані відповідні компоненти, контролери, моделі, представлення і логіка. Публікація оголошень (CRUD для житла), За публікацію оголошень відповідає AccommodationController [27] [28] [29]. Він містить методи, Create (GET) – повертає порожню форму для заповнення. Create (POST) – приймає модель з форми, перевіряє, зберігає в БД новий документ. Edit (GET) – завантажує існуючий документ, перевіряє право власності (щоб не редагував чужий), відображає форму. Edit (POST) – подібно до Create, але з оновленням. Delete (POST) – видаляє оголошення (або змінює

його статус на неактивне). Розглянемо основні поля форми створення оголошення і як вони обробляються

Лістинг 3.4 – Форма створення оголошення

```
[Authorize]
public IActionResult Create() {
return View(new AccommodationViewModel());
}
[HttpPost, Authorize]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(AccommodationViewModel vm)
{
     if (!ModelState.IsValid) {
     return View(vm);
     }
     var acc = new Accommodation {
     OwnerId =
ObjectId.Parse(User.FindFirstValue(ClaimTypes.NameIdentifier)),
     Title = vm.Title,
     Description = vm.Description,
     City = vm.City,
     Address = vm.Address,
     Type = vm.Type,
     Rooms = vm.Rooms,
     GuestsCapacity = vm.GuestsCapacity,
     PricePerNight = vm.PricePerNight,
     Amenities = vm.SelectedAmenities // припустимо, це список
рядків з чекбоксів
     };
      // Обробка завантажених фото
     if(vm.Photos != null && vm.Photos.Any()) {
     acc.Photos = new List<string>();
     string    uploadsFolder    =    Path.Combine(_env.WebRootPath,
"uploads");
foreach(var formFile in vm.Photos) {
     if(formFile.Length > 0 &&
     formFile.ContentType.StartsWith("image/")) {
     string fileName = $"{acc.Id}
     _photo{Guid.NewGuid().ToString().Substring(0,8)}.jpg";
     string filePath = Path.Combine(uploadsFolder, fileName);
     using(var stream = System.IO.File.Create(filePath)) {
          await formFile.CopyToAsync(stream);
          }
      acc.Photos.Add("/uploads/" + fileName); } }
}
// Вставка в базу
await _accCollection.InsertOneAsync(acc);
TempData["Message"] = "Оголошення успішно додано!";
return RedirectToAction("MyListings", "Account");
}
```

Тут є кілька моментів, Ми використовуємо модель-представлення AccommodationViewModel для форми, замість прямо використовувати нашу модель Accommodation. Це дозволяє зробити окремі властивості для, скажімо, вибору зручностей (наприклад, у VM, AmenitiesOptions список усіх опцій для чекбоксів, і SelectedAmenities – обрані). При поверненні на форму у разі помилки ми будемо мати, що відобразити. OwnerId береться з User claims (ми в SignInUserAsync поклали NameIdentifier). Файли, vm.Photos – це IFormFileCollection, ми його проходимо, перевіряємо ContentType починається з “image/”. (Це спрощено, краще було б ще визначити дозволені формати). Формуємо унікальне ім’я (використали шматок GUID). Зберігаємо в wwwroot/uploads.

Додаємо шлях у acc.Photos. _env.WebRootPath дає фізичний шлях до wwwroot. В реальній app можливо варто зберігати інакше (Azure Blob, etc.), але локально – у wwwroot нормально. Після успіху, перекидаємо на список моїх оголошень (можливо, AccountController.MyListings, або у тому ж controller Index). Редагування оголошення дуже подібне, GET, завантажує документ, мапить у AccommodationViewModel (наприклад, ставить галочки для вже наявних зручностей). POST, знаходить документ, змінює поля з VM, оновлює в БД (ReplaceOne або UpdateOne).

Якщо треба, переобробляє нові фото (наприклад, можна дозволити додати ще фото). Якщо редагуємо, варто при новому фото теж зберегти, а при видаленні фото (якщо UI дозволяє зняти) – видалити файл (не реалізовано зараз, але бажано). Перевірка доступу, переконуємось, що OwnerId == current user, інакше return Forbid(). Видалення оголошення, Можна зробити POST через форму або AJAX. У нас Admin і Host можуть видалити. Код навдено (див. лістинг 3.5).

Лістинг 3.2 – Видалення оголошення

```
[HttpPost, Authorize]
public async Task<IActionResult> Delete(string id) {
var acc = await _accCollection.Find(a => a.Id ==
```

```
ObjectId.Parse(id)).FirstOrDefaultAsync();
if(acc == null) return NotFound();
if(acc.OwnerId                                                    !=
ObjectId.Parse(User.FindFirstValue(ClaimTypes.NameIdentifier))
 && !User.IsInRole("Admin")) {
 return Forbid();
 }
 // Перевірка на існуючі майбутні броні (якщо потрібно)
 bool hasFutureBookings = await _bookingsCollection.Find(b =>
 b.AccommodationId == acc.Id && b.StartDate >= DateTime.Today &&
b.Status ==
 "Confirmed").AnyAsync();
 if(hasFutureBookings) {
 TempData["Error"] = "Неможливо видалити – є активні бронювання.";
 return RedirectToAction("MyListings", "Account");
 }
 // Видалити пов'язані дані? (відгуки, фото файли теж варто)
 if(acc.Photos != null) {
 foreach(var url in acc.Photos) {
 var filePath = _env.WebRootPath + url.Replace('/',
 Path.DirectorySeparatorChar);
 if(System.IO.File.Exists(filePath)) {
 System.IO.File.Delete(filePath);
 }
 }
 }
 await _accCollection.DeleteOneAsync(a => a.Id == acc.Id);
 TempData["Message"] = "Оголошення видалено.";
 return RedirectToAction("MyListings", "Account");
 }
```

Перевіряємо броні, якщо є – не видаляем (можна, наприклад, зробити Status=Inactive). Видаляємо фото з диску (щоб не захаращувати). Видаляємо документ із БД. Відгуки/броні, пов'язані з цим Accommodation – вони залишаться з “висячим” AccommodationId. Можна або теж видалити (каскадно), або залишити (історичні дані). В ідеалі, якщо житло видалене, варто і броні скасувати. Але щоб спростити, ми тут не врахували цей нюанс.

Веб-сервіс побудований за клієнт-серверною архітектурою, де клієнтська частина (браузер користувача) взаємодіє із сервером через HTTP(S) запити. У нашому застосунку це здебільшого традиційні запити для завантаження сторінок з повним циклом запит-відповідь та перезавантаженням сторінки, хоча також використовуються елементи AJAX для покращення швидкодії та інтерактивності. Розглянемо взаємодію в ключових сценаріях. Під час перегляду

сторінок, коли користувач переходить за певною URL адресою (наприклад, “/search?city=Київ” або “/Accommodation/Details/123”), браузер надсилає HTTP GET запит до сервера. Сервер (ASP.NET MVC) підбирає відповідний контролер і метод, формує модель даних, вставляє її у відповідне представлення (Razor) і генерує HTML, який повертається як HTTP Response. Браузер отримує відповідь і відображає сторінку. Статичні ресурси (CSS, JS, зображення) завантажуються окремими запитами з папки wwwroot, і всі запити працюють поверх HTTPS для забезпечення шифрування трафіку.

При обробці форм, таких як реєстрація, логін, створення оголошення чи бронювання, використовуються HTTP POST запити. Наприклад, форма реєстрації надсилає дані на “/Account/Register”. Сервер автоматично проводить валідацію ModelState (на основі DataAnnotations) і у випадку помилок повертає форму з повідомленнями, а при успішному заповненні виконується бізнес-логіка (створення користувача, додавання до БД) і відбувається redirect на іншу сторінку (застосовується патерн Post-Redirect-Get), наприклад, на головну сторінку з повідомленням про успішну реєстрацію. Для покращення взаємодії користувача реалізований AJAX функціонал, зокрема автодоповнення міста при введенні в поле пошуку. У цьому випадку формується GET-запит до “/api/cities?query=Ки”, сервер повертає список відповідних міст у форматі JSON, і з використанням jQuery UI Autocomplete відображається випадаючий список.

Також AJAX застосовується для фільтрації результатів пошуку, при зміні параметрів (повзунок ціни, галочки фільтрів) JavaScript формує AJAX-запит, наприклад, “/Search/Filter?city=X&minPrice=Y”, який повертає HTML фрагмент списку результатів (PartialView), що динамічно замінюється на сторінці без її повного перезавантаження. Аналогічно працює завантаження календаря зайнятості, при відкритті сторінки оголошення виконується AJAX GET запит на “/Accommodation/GetUnavailableDates?id=...”, сервер повертає список зайнятих дат у JSON, які потім використовуються для ініціалізації flatpickr з опцією “disable”. Важливим елементом є можливість повернення з контролера не лише Views, а й JSON даних – це реалізовано через Web API або звичайні контролери

з атрибутом “[Produces(“application/json”)]”. Наприклад, метод “Account/CheckEmailExists” перевіряє, чи зайнятий email, і повертає “{ exists, true/false }” для перевірки прямо у формі реєстрації. Під час авторизації сервер встановлює cookie, який містить підписані дані користувача (наприклад, ASPXAUTH). При наступних запитах браузер автоматично відправляє цей cookie, middleware витягує з нього інформацію та формує об’єкт User (ClaimsPrincipal), завдяки чому атрибут \[Authorize] може визначити, чи дозволити доступ.

Крім того, використовуються тимчасові cookie для передачі повідомлень між сторінками (наприклад, через TempData), а Session-сховище намагаємось не використовувати активно, щоб уникнути залежності від стану сервера. З боку клієнта можна використовувати localStorage/SessionStorage для зберігання уподобань, хоча у поточній реалізації це не задіяно. Щодо безпеки, всі критичні дії виконуються через POST-запити з CSRF-токеном, що унеможливлює сторонні атаки без токена, а jQuery у шаблонах автоматично додає токен до заголовка.

Доступ до функціоналу також обмежується на основі ролей користувача, сервер перевіряє, чи є користувач власником оголошення, чи має права Admin, і може повертати помилку доступу (Forbidden), навіть якщо атрибут \[Authorize] дозволив доступ. Всі вхідні дані з клієнта проходять валідацію (Required, Range, Regex), а Razor автоматично екранує HTML вставки, що унеможливлює XSS атаки.

Під час завантаження файлів (наприклад, фото) перевіряється тип, розмір, і файл зберігається з унікальним ім’ям у спеціальній директорії. З метою безпеки перевіряється також розширення, щоб уникнути виконання потенційно небезпечного коду. У разі потреби масштабування систему можна розширити, винісши окремі функції в мікросервіси (наприклад, пошук або рекомендації), які будуть взаємодіяти з клієнтом через HTTP API. Хоча можна було б реалізувати повноцінний SPA на React чи Angular, для дипломного проєкту обрано підхід з Razor Views. Поточна архітектура передбачає один веб-сервер і одну БД, але в

разі зростання навантаження можна розгорнути кілька екземплярів веб-сервісу за load balancer за умови синхронізації ключів DataProtection. MongoDB також підтримує кластеризацію через репліки. Моніторинг і логування реалізуються через вбудований механізм ILogger, події можна записувати у файли або БД, також можливе підключення аналітики (Google Analytics, Application Insights).

Отже, взаємодія клієнт-сервер побудована на стандартних принципах веброзробки, сервер генерує сторінки, клієнт відображає їх, а також надсилає запити для отримання чи передачі даних. Усі дії обробляються сервером через контролери, клієнт не має прямого доступу до БД чи бізнес-логіки, що забезпечує контрольованість та захист. На цьому етапі, маючи спроєктовану архітектуру та вибрані технології, можна переходити до практичної реалізації та кодингу, що буде висвітлено в наступному розділі.

13.3 Інтеграція платіжної системи (Stripe)

Одним із ключових технічних завдань проєкту була інтеграція з платіжною системою Stripe. Вона дозволяє здійснювати онлайн платежі через зручний інтерфейс, забезпечує високий рівень безпеки та підтримує кілька платіжних методів (банківські картки, Klarna, Afterpay тощо). У реалізації використовувалася Stripe Checkout Session API, що дозволяє швидко створити сесію оплати та перенаправити користувача на захищену сторінку Stripe.

Налаштування Stripe починалося з додавання відповідних ключів у appsettings.json. Це були SecretKey і PublishableKey, отримані з тестового облікового запису. Під час запуску програми через StripeConfiguration.ApiKey ініціалізувався ключ доступу до API. У Program.cs (або Startup.cs) було додано StripeConfiguration.ApiKey = builder.Configuration[“Stripe,SecretKey”];

Це дозволило працювати з Stripe SDK на всьому сервері. Під час створення бронювання, у відповідному контролері (BookingController або PaymentController) формувалась Stripe сесія оплати.

Лістинг 3.6 – Інтеграція Stripe.

```
var options = new SessionCreateOptions
{
     PaymentMethodTypes = new List<string> { "card" },
     LineItems = new List<SessionLineItemOptions>
     {
          new SessionLineItemOptions
          {
          PriceData = new SessionLineItemPriceDataOptions
          {
                    UnitAmount = (long)(booking.TotalPrice * 100),
                    Currency = "uah",
                    ProductData                =                new
SessionLineItemPriceDataProductDataOptions
                    {
                        Name = "Оплата за бронювання",
                    },
                },
                Quantity = 1,
            },
         },
         Mode = "payment",
         SuccessUrl                                                    =
$"{_baseUrl}/Booking/Success?bookingId={booking.Id}",
         CancelUrl                                                     =
$"{_baseUrl}/Booking/Cancel?bookingId={booking.Id}",
     };
     var service = new SessionService();
     Session session = service.Create(options);
     return Redirect(session.Url);
```

Користувач перенаправляється на сторінку Stripe, де вводить дані картки. У разі успішної оплати він повертається на SuccessUrl, де оновлюється статус бронювання. Для повної безпеки в продакшн середовищі варто реалізувати webhook, який отримує повідомлення від Stripe про статус платежу (наприклад, payment_intent.succeeded), щоб уникнути ситуацій, коли користувач оплатив, але не повернувся на сайт.

Stripe також підтримує гнучке тестування через Stripe Dashboard, де видно створені Session та PaymentIntent. Приклад форми Stripe Checkout наведено на рисунку 3.1.

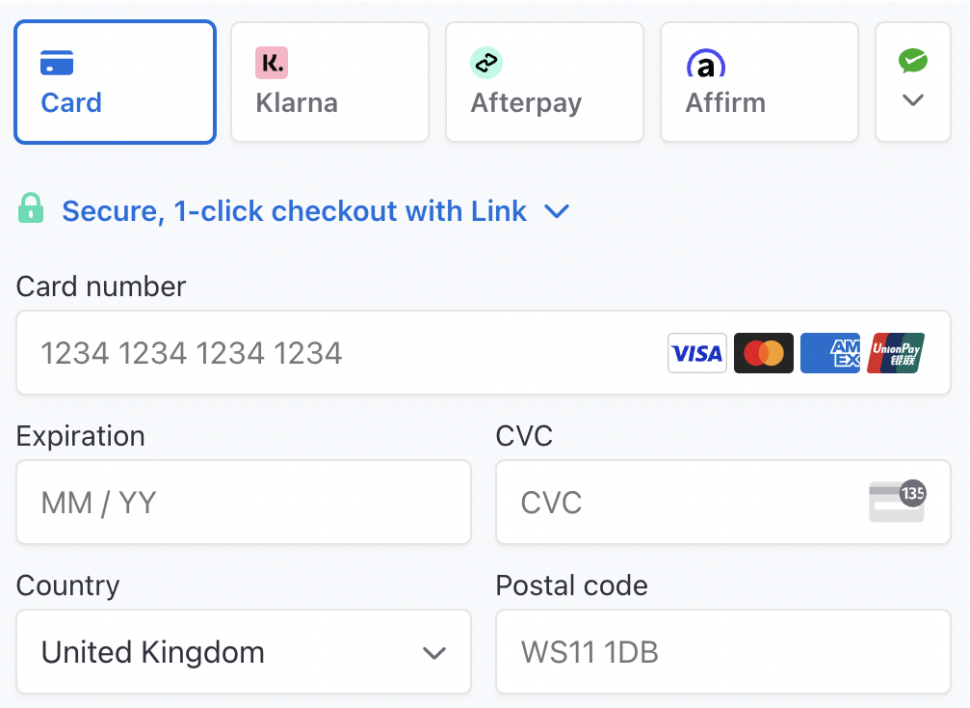

Рисунок 3.1 – Форма Stripe Checkout для оплати

З точки зору UX це зручне рішення, яке не вимагає реалізації форми картки на боці клієнта. Безпека реалізується шляхом того, що секретні ключі залишаються тільки на сервері, а клієнт працює лише з session.Url.

Це рішення було повністю протестоване в режимі “test”, оплати проходили успішно, з відповідним оновленням статусу бронювання.

13.4 Тестування веб-додатку

Після реалізації основного функціоналу було проведено комплексне тестування веб-сервісу, включаючи модульне тестування окремих компонентів (у міру можливості), а також інтеграційне та функціональне тестування сценаріїв використання. Метою тестування було виявити та виправити помилки, перевірити відповідність роботи системи заданим вимогам та впевнитися у стійкості додатку при типових діях користувачів. Для перевірки окремих методів (наприклад, логіки перетину дат бронювання) застосовувався підхід “unit test by coding” – створювалися тимчасові фрагменти коду з різними комбінаціями дат і

аналізувалися результати. Частково використовувався фреймворк xUnit, зокрема для тестування методу перевірки пароля (PasswordHasher) та функції перевірки доступності (перехресні дати). Основний акцент був зроблений на сценарійному тестуванні, для чого було складено кілька “user stories” і відповідних сценаріїв.

У першому сценарії тестувалася реєстрація та вхід нового користувача. Було перевірено всі ключові кроки, відкриття сторінки реєстрації, введення валідних даних, підтвердження, перенаправлення на головну сторінку, повторний вхід із тими ж даними. Сценарій успішно виконано, виявлено кілька дрібних недоліків, повідомлення про успішну реєстрацію було англійською (виправлено), а також дозволялося реєструватися з порожнім полем “ім'я” (вирішено зробити його опціональним – у такому разі в шапці відображається email).

У другому сценарії перевірялося додавання оголошення орендодавцем. Було створено обліковий запис із роллю Host, перевірено процес створення оголошення із заповненням усіх полів, додаванням зручностей і фото. Очікувана поведінка підтвердилася, нове оголошення з’являється у списку, детальна сторінка відображає всі дані, фото доступні. Виявлено, що після створення не з’являлося повідомлення про успіх (додано TempData[“Message”]). Галерея реалізована просто без JS-слайдера – залишено як є, хоча на мобільних фото йдуть у стовпчик.

У третьому сценарії перевірялося бронювання житла. Гість шукав житло за містом і датами, здійснював бронювання з використанням Stripe Checkout і тестової картки. Після успішної оплати користувача перенаправляло на сайт, бронювання з’являлося у списку з відповідним статусом “Confirmed”. Під Host перевірено, що бронювання відображається (через БД, бо в UI ця функція поки що не реалізована). Система не дозволила повторно забронювати ті самі дати – перевірка спрацювала. Інші дати забронювати дозволяло.

У четвертому сценарії перевірялося скасування бронювання. Після його здійснення користувач натискав “Скасувати”, бронювання змінювало статус на “Cancelled”, у списку позначалося сірим. Дати знову ставали доступними для

інших користувачів. Оскільки повернення коштів не реалізоване, система не робила refund, але це зазначено як подальший крок. Важливо, що при перевірці доступності враховуються лише бронювання зі статусом “Confirmed”, тому “Cancelled” не блокують дати.

У п’ятому сценарії тестувалося редагування та видалення оголошення. Було перевірено можливість змінити опис і ціну, збереження відображалося на сайті. Якщо не було активних бронювань – оголошення успішно видалялося разом із фото з сервера. Якщо бронювання були у статусі “Confirmed” – система блокувала видалення. У випадку Pending або Cancelled – дозволяло видалити (потенційна проблема, варто врахувати в майбутньому). Видалення оголошення не очищає базу від пов’язаних бронювань, вони залишаються без зв’язку – можливо, варто реалізувати каскадне видалення або маркування таких бронювань як “відв’язаних”.

Щодо продуктивності та навантаження, додаток тестувався у режимі одного користувача. На локальному ПК усі сторінки завантажуються майже миттєво. Операції пошуку та фільтрації працюють швидко навіть після додавання ~100 тестових оголошень і ~200 бронювань. Пошук перекриттів по датах залишався швидким завдяки індексації MongoDB. Зроблено висновок, що при обсягах до кількох тисяч записів система працюватиме комфортно на одному сервері. Для масштабування потрібне кластерування MongoDB, що виходить за межі бакалаврської роботи.

З точки зору безпеки перевірено, що сторінки, які вимагають авторизації, недоступні неавторизованим користувачам. ASP.NET middleware забезпечує перенаправлення на сторінку входу або повертає помилку 403. Спроби підробки cookie не увінчалися успіхом, оскільки вони шифруються. Mongo драйвер автоматично екранує спеціальні символи, Razor не виконує HTML – XSS-атаки не проходять. Усі форми захищені від CSRF за допомогою токенів, перевірено, що без них запити блокуються. Для ілюстрації роботи додатку у додатках до роботи наведено скріншоти головної сторінки з формою пошуку.

На завершення тестування складено таблицю відповідності реалізованого функціоналу технічному завданню, що засвідчує відповідність усім основним вимогам (див. таблицю 3.1).

Таблиця 3.1 – Основні вимоги та архітектурні рішення веб-сервісу

Вимога	**Статус виконання**
1	**2**
Реєстрація/логін користувачів	Виконано, користувач може зареєструватись, увійти в систему; реалізовано розмежування ролей (User, Admin, Host).
Публікація оголошень (CRUD)	Виконано, хост створює, редагує, видаляє оголошення; завантжує фото, зазначення параметрів. Обмеження: при наявності бронювань видалення блокується.
Пошук та перегляд житла	Виконано, є форма пошуку за місцем і датами; результати виводяться списком; сторінка деталей містить всю інформацію, фото, відгуки.
Бронювання житла	Виконано, зареєстрований користувач може забронювати житло; система перевіряє конфлікти; при бронюванні виконується оплата
Інтеграція платіжної системи	Виконано, підключено Stripe Checkout для прийому оплати карткою; статус бронювання оновлюється після підтвердження платежу.

Продовження таблиці 3.1

1	**2**

Управління бронюваннями	Виконано частково, користувач бачить список своїх бронювань, може скасувати; орендодавцю слід додатково реалізувати інтерфейс перегляду бронювань його житла (в поточній версії можна переглядати через профіль користувача).
Безпека даних та доступу	Виконано, паролі хешуються; чутливі операції захищені; розмежування доступу за ролями; перевірка введених даних.

Висновки тестування, Система у поточному вигляді функціонує згідно з основними запланованими сценаріями. Всі критичні функції (реєстрація, створення оголошень, пошук, бронювання, оплата) успішно працюють. Під час тестування були виявлені та виправлені дрібні помилки в логіці (наприклад, некоректна обробка порожніх полів) і користувацькому інтерфейсі (відсутність повідомлень, незручності в навігації). Жодних блокуючих проблем чи збоїв (exceptions) під час тестування не виникало; додаток не впав жодного разу.

Разом з тим, тестування виявило можливі шляхи покращення, додати розширений функціонал для адміністраторів (щоб можна було керувати списками оголошень, модерувати відгуки). Впровадити систему відгуків повністю (наразі можна лише переглядати). Покращити UX для хоста щодо перегляду бронювань (можна календар або хоча б список). Налаштувати відправку email повідомлень, підтвердження реєстрації, підтвердження бронювання, повідомлення про скасування тощо – це суттєво підвищить корисність системи.

Загальний результат тестування можна оцінити як успішний – веб-сервіс готовий до експериментальної експлуатації в умовах, наближених до реальних. У розділі висновків будуть підбиті підсумки виконаної роботи та зазначено отримані результати.

РОЗДІЛ 14. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

14.1 Безпека умов праці ІТ-спеціаліста

Одним з найголовніших умов праці ІТ-спеціаліста є його безпека. Для забезпечення ІТ-спеціаліста безпекою використовується менеджмент системи безпеки. Менеджмент системи безпеки – це система наукових знань та досвіду забезпечення безпеки, яка використовується для досягнення цілей системи безпеки через використання праці, інтелекту та мотивів поведінки людей [30]. До менеджменту системи безпеки відносять принцип запобігання виникненню надзвичайних ситуацій. Запобігання виникненню надзвичайних ситуацій – це підготовка та реалізація комплексу правових, соціально-економічних, політичних, організаційно-технічних, санітарно-гігієнічних та інших заходів, спрямованих на регулювання безпеки, проведення оцінювання рівнів ризику, завчасне реагування на загрозу виникнення НС на основі даних моніторингу (спостережень), експертизи, досліджень та прогнозів щодо можливого перебігу подій із метою недопущення їх переростання у НС або пом'якшення її можливих наслідків.

До конкретних заходів забезпечення безпеки умов праці ІТ-спеціаліста можна віднести:

1. Систему автоматичного пожежогасіння та аварійного оповіщення в офісі;
2. Проведення навчальних евакуацій у разі загроз;
3. Перевірка справності електромережі з УЗО (захист від ураження струмом);
4. Робочі станції з ергономічним обладнанням для запобігання тромбозу, проблемам із серцем і спиною;
5. Встановлення пропускної системи з обмеженням доступу для сторонніх осіб.

У переліку конкретних заходів забезпечення безпеки було зазначено автоматичну систему пожежогасіння (АСПГ). Автоматична система пожежогасіння – система пожежогасіння, яка виконує функції виявлення ознак горіння, оповіщення про пожежу та подавання вогнегасної речовини без втручання людини.

АСПГ повинні забезпечувати:

1. Спрацювання протягом часу, який має бути меншим за час початкової стадії розвитку пожежі;
2. Розрахункову інтенсивність подачі та/або необхідну концентрацію вогнегасної речовини;
3. Локалізацію пожежі протягом часу, необхідного для введення в дію оперативних сил і засобів, або її ліквідацію.

Не зважаючи на те, що в сучасних системах автоматичного пожежогасіння застосовуються вогнегасні речовини, які перебувають в різних агрегатних станах, до складу будь-якої АСПГ входять наступні елементи:

1. Резервуар з основним та резервним (як правило, 100%) запасом вогнегасної речовини (ВГР);
2. Пристрої управління випуском ВГР та контролю кількості ВГР в черговому режимі;
3. Випускні насадки та розгалужена розподільча мережа трубопроводів;
4. Спонукальна (збуджувальна) система.

14.2 Безпека життєдіяльності та фактори воєнного часу

Оскільки в Україні у 2025 р., не можна оминути увагою питання безпеки життєдіяльності в умовах воєнного стану та можливих надзвичайних ситуацій, під час війни ІТ-фахівці, як і інші громадяни, піддаються додатковим ризикам, повітряні тривоги, загрози ракетних ударів, відключення електроенергії, стресові фактори [31].

Отже враховуючи фактори воєнного часу потрібно дотримуватись наступних приципів:

1. Зберігати особистий спокій; не вступати у суперечки з незнайомими людьми, уникати можливих провокацій;

2. Слідкувати за політичними новинами, щоб робити правильні висновки; події, які можуть розгорнутися не повинні захопити Вас зненацька, для цього користуйтеся ЗМІ, Інтернетом;

3. У разі отримання від органів державної влади інформації про заходи щодо підвищення безпеки, зміни тривалості комендантської години, або можливу небезпеку передати її іншим людям (за місцем проживання, роботи, тощо);

4. Не сповіщайте про свої майбутні дії (плани) малознайомих людей, а також знайомих з ненадійною репутацією;

5. Завжди майти при собі документ (паспорт, посвідчення водія, студентській квіток), що засвідчує особу; а також жетон, браслет, або записані на папері відомості про групу крові, можливі проблеми зі здоров'ям (алергії на медичні препарати, інсулінозалежність, астма, тощо);

6. Найближчі повинні знати, як діяти за непередбачуваних обставин, куди йти і де вам всім зустрітись;

7. Під час виникнення тривожних подій (ведення бойових дій) тримати документи та гроші в одному потаємному, але для Вас легкодоступному місці;

8. Особистий транспорт тримати у справному стані з запасом палива для можливості швидкого виїзду з небезпечного району;

9. Дізнатись про місце розташування захисних споруд поблизу місця проживання, у місцях частого відвідування (магазини, базар, дорога до роботи, медичні заклади, тощо); наявність найближчого бомбосховища (метро); без необхідності намагатись якнайменше знаходитись поза місцем проживання;

10. Надати першу долікарську допомогу іншим людям у разі їх поранення в межах своєї компетентності, залучити інших більш досвідчених людей,

викликати швидку допомогу, представників ДСНС України, органів правопорядку, за необхідності – військових;

У переліку приципів, яких потрібно дотримуватись під воєнного часу зазначено тему укриття, розглянемо тему укриття відповівши на питання рекомендацій щодо облаштування та використання найпростіших укриттів. Найпростіше укриття – це фортифікаційна споруда, цокольне або підвальне приміщення, що знижує комбіноване ураження населення від небезпечних наслідків надзвичайних ситуацій, а також від дії засобів ураження в особливий період.

Під найпростіші укриття для населення можуть бути пристосовані:

- Підвали та підпілля в житлових будинках, промислових, допоміжних і адміністративно-побутових будинках;
- Заглиблені споруди, що стоять окремо, призначені для виробничих, складських і побутових потреб: заглиблені гаражі, овочесховища, льохи, склади та інше;
- Окремі приміщення на перших і других поверхах в кам’яних (бетонних) будівлях, що мають мінімальну кількість зовнішніх відкритих стін, особливо без віконних і інших отворів.

Для оснащення (облаштування) найпростіших укриттів доцільно завчасно передбачати таке обладнання:

- Місця для сидіння (лежання) – лавки, нари, стільці, ліжка тощо;
- Ємності з питною (з розрахунку 2 л/добу на одну особу, яка підлягає укриттю) та технічною водою (за відсутності централізованого водопостачання);
- Контейнери для зберігання продуктів харчування;
- Виносні баки для нечистот зі щільним закриванням (для неканалізованих будівель і споруд);

- Резервне штучне освітлення (електричні ліхтарі, свічки, гасові лампи тощо);
- Первинні засоби пожежегасіння (відповідно до встановлених норм для приміщень відповідного функціонального призначення);
- Засоби надання первинної медичної допомоги;
- Засоби зв’язку і оповіщення (телефон, радіоприймач);

14.3 Охорона праці в ІТ компаніях

Роботодавець зобов’язаний враховувати чинні санітарні нормативи освітлення, вимоги до параметрів мікроклімату (температура, відносна вологість), ступеня і сили вібрації, звукового шуму і вогнестійкості приміщення, а також характеристики електромагнітного, ультрафіолетового та інфрачервоного полів.

Конкретні показники зазначених санітарних норм наведені у ДСанПіН 3.3.2.007-98. Важливо, що зазначені державні санітарні правила і норми поширюються на умови й організацію праці при роботі з візуальними дисплейними терміналами (ВДТ) усіх типів вітчизняного та зарубіжного виробництва на основі електронно-променевих трубок (ЕПТ), що використовуються в ПК колективного використання та персональних ПК [32].

Так, наприклад, роботодавцю заборонено установлювати комп’ютери в приміщеннях, які розташовані у підвалах будинків. Для уникнення можливих аварій та замикань, поряд з приміщеннями, де вестиметься робота з комп’ютером (над чи під ними), також не дозволяється проведення робіт, що потребують здійснення надмірно вологих технологічних процесів.

Відповідне приміщення повинно бути укомплектоване системами центрального або індивідуального опалення, кондиціонування чи вентиляції повітря.

У кожній кімнаті, де обладнуватимуться робочі місця співробітників, що працюватимуть на комп’ютері, повинні бути наявні елементи природного та штучного освітлення. При цьому, на вікнах слід встановити легко регульовані жалюзі чи штори, які дозволять працівникам коригувати рівень освітлення в приміщенні.

З метою досягнення максимального рівня безпечності і охорони праці при роботі з комп’ютером, виробничі приміщення необхідно обладнати аптечками першої медичної допомоги, системами автоматичної пожежної сигналізації і вогнегасниками.

Важливо пам’ятати, що в разі несправності ПК, виявленні іскріння, пробою, запаху гару, ознак горіння тощо слід негайно припинити роботу, відключити обладнання від електромережі та негайно повідомити безпосереднього керівника або спеціаліста з ремонту ПК.

В приміщенні, в якому разом працюють 5 або більше комп'ютерів, на видимому місці встановлюється службовий вимикач, який у разі потреби дозволить повністю відключити електричне живлення кімнати.

ВИСНОВКИ

У процесі виконання бакалаврської роботи було створено веб-сервіс для бронювання житла, подібний до Airbnb чи Booking.com, як навчальний проект із використанням сучасних веб-технологій.

Проведено аналіз предметної області, визначено вимоги до системи, зокрема, управління користувачами, створення оголошень, пошук, бронювання, оплата, відгуки. Обрано архітектуру MVC на базі ASP.NET і NoSQL базу MongoDB, що забезпечило гнучкість, масштабованість і продуктивність. Реалізовано ключові підсистеми, реєстрацію з ролями, публікацію оголошень із фото, пошук за містом і датами, інтерактивний календар, систему бронювання з перевіркою зайнятості, інтеграцію Stripe для онлайн-оплат, особисті кабінети користувачів і панель адміністратора.

Проведено тестування – функціонал працює стабільно, без критичних помилок. Забезпечено безпеку, аутентифікація, авторизація, HTTPS, CSRF. Розглянуто питання охорони праці під час розробки, зокрема в умовах воєнного часу. Проєкт є цінним зразком застосування веб-технологій, NoSQL і фінтех-рішень, а досвід інтеграції Stripe актуальний для українських реалій.

Результат можна використовувати як навчальний приклад або основу для стартапу. Завдання виконано повністю, що свідчить про готовність автора до професійної діяльності. Перспективи вдосконалення, система відгуків, чат між користувачами, гнучкі фільтри, мобільна версія, часткове повернення коштів, інтеграція з картами, перехід на мікросервіси. Отримано практичний досвід повного циклу розробки веб-сервісу.

ПЕРЕЛІК ДЖЕРЕЛ

1 Booking vs Airbnb: The battle of the vacation rental giants // EL PAÍS – 27.02.2024. [Електронний ресурс]. – Режим доступу: https://english.elpais.com/economy-and-business/2024-02-27/booking-vs-airbnb-the-battle-of-the-vacation-rental-giants.html.

2 2024 Vacation Rental Stats Roundup / Paris Achen // Rent Responsibly – 03.03.2025. [Електронний ресурс]. – Режим доступу: https://www.rentresponsibly.org/2024-vacation-rental-stats-roundup.

3 Airbnb to offer free housing to 100,000 Ukrainian refugees // The Guardian. – 28.02.2022. – [Електронний ресурс]. – Режим доступу: https://www.theguardian.com/technology/2022/feb/28/airbnb-to-offer-free-housing-to-100000-ukrainian-refugees

4 MongoDB Advantages & Disadvantages – GeeksForGeeks. – [Електронний ресурс]. – Режим доступ: https://www.geeksforgeeks.org/mongodb/mongodb-advantages-disadvantages/

5 Steve Smith. Overview of ASP.NET MVC – Microsoft Learn, updated 17.06.2024. [Електронний ресурс]. – Режим доступ: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-9.0

6 Adrián Bailador. Complete Guide to Integrating Stripe in a .NET Application – DEV Community, 17.07.2024. – [Електронний ресурс]. – Режим доступ: https://dev.to/adrianbailador/complete-guide-to-integrating-stripe-in-a-net-application-48d9

7 The Ultimate Guide on How to develop Your Own Airbnb Clone / Creole Studios – Medium, 2023. – [Електронний ресурс]. – Режим доступу: https://medium.com/@creolestudios/the-ultimate-guide-on-how-to-develop-your-own-airbnb-clone-ffa42edd5d89

8 Future Market Insights. Short-Term Rental Market Report 2024 – [Електронний ресурс]. – Цит. за: https://www.rentresponsibly.org/2024-vacation-rental-stats-roundup

9 Official Stripe API documentation – Stripe Docs. – [Електронний ресурс]. – Режим доступу: https://dev.to/adrianbailador/complete-guide-to-integrating-stripe-in-a-net-application-48d9

10 AirDNA. 2025 Short-Term Rental Trends Report – AirDNA, 2025. – [Електронний ресурс]. – Режим доступу: https://www.rentresponsibly.org/2024-vacation-rental-stats-roundup

11 State of the Short-Term Rental Industry Report 2024 – Hostaway, 2024. – [Електронний ресурс]. – Цит. за: https://www.rentresponsibly.org/2024-vacation-rental-stats-roundup

12 Adam Freeman. Pro ASP.NET 6 MVC. – Apress, 2022. – 1040 p. (Розділ 7: Security, c. 250–275).

13 Search Logistics. Airbnb Statistics 2024. – [Електронний ресурс]. – Режим доступу: searchlogistics.com

14 Microsoft Learn. Create a web API with ASP.NET and MongoDB. – [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mongo-app

15 MongoDB Blog. How To Use MongoDB Atlas With .NET/.NET. – [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/developer/products/atlas/dotnet-core-atlas/

16 CodeWithMukesh. Working with MongoDB in ASP.NET Core – Ultimate Guide. – 2021. – [Електронний ресурс]. – Режим доступу: https://codewithmukesh.com/blog/mongodb-in-aspnet-core/

17 Okta Developer Blog. Build a Basic CRUD App with ASP.NET Core 3.0 and MongoDB. – 2020. – [Електронний ресурс]. – Режим доступу: https://developer.okta.com/blog/2020/06/23/crud-app-aspnetcore-mongodb

18 Okta Developer Blog. Build a Basic CRUD App with ASP Core 3.0 and MongoDB. – 2020. – [Електронний ресурс]. – Режим доступу: https://developer.okta.com/blog/2020/06/23/crud-app-aspnetcore-mongodb

19 CSharp Corner. Integrating Stripe Payment Gateway in ASP.NET Core MVC. – 2025. – [Електронний ресурс]. – Режим доступу: https://www.c-sharpcorner.com/article/integrating-stripe-in-asp-net-core-mvc/

20 Medium. How to integrate Stripe payment gateway in ASP.NET web applications. – 2023. – [Електронний ресурс]. – Режим доступу: https://medium.com/@techinsights/stripe-integration-dotnet

21 WireFuture. How To Integrate Stripe in ASP.NET Core? – 2024. – [Електронний ресурс]. – Режим доступу: https://www.wirefuture.com/articles/integrate-stripe-in-aspnet-core/

22 Kaushik Roy Chowdhury. Stripe Payment Gateway for Product Cart – Part 3. – 2021. – [Електронний ресурс]. – Режим доступу: https://www.c-sharpcorner.com/article/stripe-integration-in-asp-net-core-part-3/

23 W3Techs. Usage statistics and market share of ASP.NET – 2025. – [Електронний ресурс]. – Режим доступу: https://w3techs.com/technologies/details/ws-aspnet

24 OWASP Foundation. OWASP Top Ten Web Application Security Risks 2023. – [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/

25 OWASP Foundation. OWASP Top Ten Web Application Security Risks 2023. – [Електронний ресурс]. – Режим доступу: https://owasp.org/www-project-top-ten/

26 Uptime Institute. Global Data Center Survey 2024 – Uptime Institute, 2024. – [Електронний ресурс]. – Режим доступу: https://uptimeinstitute.com/research-reports

27 Booking.com. Terms of Service – [Електронний ресурс]. – Режим доступу: https://www.booking.com/terms.en.html

28 European Commission. Data Protection Rules as a Business Opportunity – EU Publications, 2023. – [Електронний ресурс]. – Режим доступу: https://commission.europa.eu/data-protection_en

29 Booking.com. Terms of Service – [Електронний ресурс]. – Режим доступу: https://www.booking.com/terms.en.html

30 Безпека життєдіяльності, основи охорони праці – [Електронний ресурс]. – Режим доступу: https://dl.tntu.edu.ua/content.php?cid=299414

31 ІНСТРУКЦІЇ З БЕЗПЕКИ ЖИТТЄДІЯЛЬНОСТІ ПІД ЧАС ДІЇ ВОЄННОГО СТАНУ (КПІ ім. Ігоря Сікорського) – [Електронний ресурс]. – Режим доступу: https://ela.kpi.ua/server/api/core/bitstreams/7363fbbf-f61f-4988-88ee-c0853b7da70a/content

32 Охорона праці в IT компаніях – [Електронний ресурс]. – Режим доступу: https://pro-op.com.ua/article/17850-okhorona-pratsi-v-it-kompaniyakh

33 Семенюк В. О. Огляд методів захисту текстової інформації / В. О. Семенюк, Я. В. Литвиненко // ІМСТТ, 13-14 грудня 2023 року. — Т. : ТНТУ, 2023. — С. 112. — (Інформаційні системи та технології, кібербезпека).

34 Дубчак А. О. Напрямки використання штучного інтелекту в сучасних умовах / А. О. Дубчак, Я. В. Литвиненко // Матеріали міжнародної наукової конференції „Іван Пулюй: життя в ім’я науки та України“ (до 175-ліття від дня народження), 28-30 вересня 2020 року. — Т. : ФОП Паляниця В. А., 2020. — С. 64–65. — (Важливі аспекти практичного застосування здобутків сучасної науки і новітніх технологій).

35 Concept of design, requirements and generalized architectures of components of the integrated onto-oriented information environment of simulation and processing of cyclic signals / Serhii Lupenko, Iaroslav Lytvynenko, Volodymyr Hotovych, Andrii Zozulia, Nnamene Chizoba, Oleksandr Volyanyk // Scientific Journal of TNTU. — Tern. : TNTU, 2021. — Vol 102. — P. 147–160.

36 Церковний В. О. Взаємодія систем стеження для задачі контролю перевезення багажу / В. О. Церковний, Я. В. Литвиненко // XI Міжнародна науково-практична конференція молодих учених та студентів „Актуальні задачі сучасних технологій“, 7-8 грудня 2022 року. — Т. : ТНТУ, 2022. — С. 56–57. — (Сучасні технології в будівництві, машино- та приладобудуванні).

ДОДАТКИ

Додаток А

Лістинг контролера AccommodationController

```
namespace BookingApp.Controllers
{
    [Authorize(Roles = "Admin,Host")]
    public class AccommodationController : Controller
    {
        private readonly AppContextDB _context;
        private readonly UserManager<User> _userManager;
        private readonly IWebHostEnvironment _environment;

        public AccommodationController(AppContextDB context,
UserManager<User> userManager, IWebHostEnvironment environment)
        {
            _context = context;
            _userManager = userManager;
            _environment = environment;
        }

        // GET: Accommodation
        public async Task<IActionResult> Index()
        {
            User user = await _userManager.GetUserAsync(User);

            if (user == null) { return NotFound(); }

            if (await _userManager.IsInRoleAsync(user, "Admin"))
            {
                return View(await _context.Accommodations
                    .Include(a => a.Address).Include(a =>
a.User).ToListAsync());
            }
            else
            {
                return View(await _context.Accommodations
                    .Where(a => a.UserId == user.Id)
                    .Include(a => a.Address).Include(a =>
a.User).ToListAsync());
            }
        }

        // GET: Accommodation/Details/5
        public async Task<IActionResult> Details(Guid? id)
        {
            if (id == null) {
                return NotFound();
            }

            var accommodation = await _context.Accommodations
                .Include(a => a.Offers)
                .Include(a => a.Address)
```

```
                .Include(a => a.User)
                .Include(a => a.Pictures)
                .Include(a => a.HouseRules)
                .Include(a => a.Rooms)
                .ThenInclude(r => r.Amenities)
                .FirstOrDefaultAsync(m => m.Id == id);

            if (accommodation == null) {
                return NotFound();
            }

            return View(accommodation);
        }

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

        // POST: Accommodation/Create
        // To protect from overposting attacks, enable the specific
properties you want to bind to.
        //          For          more          details,          see
http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create(
            [Bind("Name,    Type,    MaxTraveler,    Description")]
Accommodation accommodation,
            [Bind("StreetAndNumber, Complement, City, PostalCode,
Country")] Address address,
            [Bind("ArrivalHour,     DepartureHour,     PetAllowed,
PartyAllowed, SmokeAllowed")] HouseRules houseRules)
        {
            if (!ModelState.IsValid) { return View(accommodation);
}

            accommodation.UserId              =              (await
_userManager.GetUserAsync(User)).Id;
            accommodation.Address = address;
            accommodation.HouseRules = houseRules;

            _context.Add(accommodation);
            await _context.SaveChangesAsync();

            return  RedirectToAction("ManagePictures",  "Picture",
new { id = accommodation.Id });
        }

        // GET: Accommodation/Edit/5
        public async Task<IActionResult> Edit(Guid? id)
        {
```

```
            if (id == null)
            {
                return NotFound();
            }

            var accommodation = await _context.Accommodations
                .Include(a => a.Address)
                .Include(a => a.HouseRules)
                .Include(a => a.Pictures)
                .Include(a => a.Rooms)
                .FirstOrDefaultAsync(m => m.Id == id);

            if (accommodation == null)
            {
                return NotFound();
            }
            return View(accommodation);
        }

        // POST: Accommodation/Edit/5
        // To protect from overposting attacks, enable the specific
properties you want to bind to.
        //          For          more          details,          see
http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(Guid id,
            [Bind("Id,  Name,  Type,  MaxTraveler,  Description")]
Accommodation accommodation,
            [Bind("Id,    StreetAndNumber,    Complement,    City,
PostalCode, Country")] Address address,
            [Bind("Id,   ArrivalHour,   DepartureHour,   PetAllowed,
PartyAllowed, SmokeAllowed")] HouseRules houseRules)
        {
            if (id != accommodation.Id)
            {
                return NotFound();
            }

            if (!ModelState.IsValid) { return View(accommodation);
}

            // Get accommodation's user
            accommodation.UserId                =                await
_context.Accommodations.Where(a   =>   a.Id   ==   id).Select(a   =>
a.UserId).SingleOrDefaultAsync();
            accommodation.Address = address;
            accommodation.HouseRules = houseRules;

            try
            {
                _context.Update(accommodation);
                await _context.SaveChangesAsync();
```

```
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!AccommodationExists(accommodation.Id))
                {
                    return NotFound();
                }

                throw;
            }

            return RedirectToAction("Edit", new { id });
        }

        // GET: Accommodation/Delete/5
        public async Task<IActionResult> Delete(Guid? id)
        {
            if (id == null)
            {
                return NotFound();
            }

            var accommodation = await _context.Accommodations
                .FirstOrDefaultAsync(m => m.Id == id);
            if (accommodation == null)
            {
                return NotFound();
            }

            return View(accommodation);
        }

        // POST: Accommodation/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(Guid id)
        {
            var accommodation = await
_context.Accommodations.FindAsync(id);
            _context.Accommodations.Remove(accommodation);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool AccommodationExists(Guid id)
        {
            return _context.Accommodations.Any(e => e.Id == id);
        }
    }
}
```

Додаток Б

Лістинг контролера BookingController

```
namespace BookingApp.Controllers
{
    [Authorize]
    public class BookingController : Controller
    {
        private readonly AppContextDB _context;
        private readonly UserManager<User> _userManager;

        public BookingController(AppContextDB context,
UserManager<User> userManager)
        {
            _context = context;
            _userManager = userManager;
        }

        // GET: Booking
        public async Task<IActionResult> Index()
        {
            User user = await _userManager.GetUserAsync(User);

            if (user == null) { return NotFound(); }

            ViewBag.ReturnAction = "Index";

            if (await _userManager.IsInRoleAsync(user, "Admin"))
            {
                return View(await _context.Booking
                    .Include(b => b.Offer).Include(b =>
b.User).Include(b => b.Offer.Accommodation).ToListAsync());
            }
            else
            {
                return View(await _context.Booking
                    .Where(b => b.UserId == user.Id)
                    .Include(b => b.Offer).Include(b =>
b.User).Include(b => b.Offer.Accommodation).ToListAsync());
            }
        }

        // GET: Booking/HostIndex
        [Authorize(Roles = "Host, Admin")]
        public async Task<IActionResult> HostIndex()
        {
            User user = await _userManager.GetUserAsync(User);

            if (user == null) { return NotFound(); }

            ViewBag.ReturnAction = "HostIndex";
```

```
            return View("Index", await _context.Booking
                .Include(b => b.Offer).Include(b => b.User).Include(b => b.Offer.Accommodation)
                .Where(b => b.Offer.Accommodation.UserId == user.Id).ToListAsync());
        }


        // POST: Booking/Details/5
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Details(Guid id, string returnAction)
        {
            var booking = await _context.Booking
                .Include(b => b.Offer)
                .Include(b => b.User)
                .Include(b => b.Offer.Accommodation)
                .FirstOrDefaultAsync(m => m.Id == id);

            if (booking == null)
            {
                return NotFound();
            }

            ViewBag.ReturnAction = returnAction;

            return View(booking);
        }

        // POST: Booking/Create
        // To protect from overposting attacks, enable the specific properties you want to bind to.
        //          For          more          details,          see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create(
            [Bind("OfferId,        ArrivalDate,        ArrivalTime, DepartureDate, DepartureTime, NbPerson")] Booking booking)
        {
            if (ModelState.IsValid)
            {
                int    nbNight    =    (booking.DepartureDate    - booking.ArrivalDate).Days;
                double         pricePerNight         =         await _context.Offers.Where(o  =>  o.Id  ==  booking.OfferId).Select(o  => o.PricePerNight).SingleOrDefaultAsync();
                double cleaningFee = await _context.Offers.Where(o =>        o.Id        ==        booking.OfferId).Select(o        => o.CleaningFee).SingleOrDefaultAsync();
```

```
                User          senderUser          =          await
_userManager.GetUserAsync(User);
                User receiverUser = await _context.Offers.Where(o =>
o.Id           ==           booking.OfferId).Select(o           =>
o.Accommodation.User).SingleOrDefaultAsync();

                // Calcul total price
                double totalPrice = pricePerNight * (double)nbNight
+ cleaningFee;

                if                    (await                    new
TransactionController(_context).DoTransaction(senderUser,
receiverUser, totalPrice))
                {
                    booking.TotalPrice = totalPrice;
                    booking.UserId            =            (await
_userManager.GetUserAsync(User)).Id;

                    _context.Add(booking);
                    await _context.SaveChangesAsync();
                }
                else
                {
                    TempData["AlertType"] = "danger";
                    TempData["AlertMsg"]  =  "Vous  n'avez  pas  le
montant nécessaire pour procéder à la réservation, il vous manque "
+     (totalPrice-senderUser.Balance)   +   "   €   !   Veuillez   <a
href=\"/Identity/Account/Manage/Wallet\">créditer            votre
compte</a>";

                    return RedirectToAction("View", "Offer", new {
id = booking.OfferId });
                }
            }
            return RedirectToAction(nameof(Index));
        }
    }
}
```