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: Розробка інформаційної системи бронювання робочих місць у коворкінгах

Виконав: студент IV курсу, групи СТ-41
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)

_____ Семенюк В. В.
(підпис) (прізвище та ініціали)

Керівник _____ Марценко С.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

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

Рецензент _____ _____
(підпис) (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)
« 20 » _червня_ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 126 Інформаційні системи та технології
(шифр і назва спеціальності)

Студенту Семенюку Владиславові Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка інформаційної системи бронювання робочих місць у коворкінгах

Керівник роботи Марценко Сергій Володимирович, к. т. н., доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *27* » *січня* 2025 року № *4/7-444*

2. Термін подання студентом завершеної роботи 20 червня 2025 р.

3. Вихідні дані до роботи Літературні та інтернет джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз предметної області та постановка завдання. 1.1 Огляд існуючих рішень для бронювання робочих місць у коворкінгу. 1.2 Аналіз вибору мови програмування та середовища розробки, що використовуються для створення інформаційної системи. 1.3 Постановка задачі для розробки інформаційної системи бронювання робочих місць у коворкінгах. 1.4 Висновок до першого розділу. 2. Реалізація та проектування інформаційної системи. 2.1 Архітектура інформаційної системи бронювання робочих місць у коворкінгах. 2.2 Пошук актантів та варіантів використання. 2.3 Проектування та реалізація екранів користувацького інтерфейсу інформаційної системи бронювання робочих місць. 2.4 Проектування бази даних. 2.5 Реалізація логіки інформаційної системи бронювання робочих місць у коворкінгах. 2.6 Висновки до другого розділу. 3. Практична частина. Демонстрація та тестування інформаційної системи. 3.1 Демонстрація роботи інформаційної системи бронювання робочих місць у коворкінгах. 3.2 Тестування та валідація інформаційної системи бронювання робочих місць у коворкінгах. 3.3 Висновки третього розділу. 4. Безпека життєдіяльності, основи охорони праці. 4.1 Долікарська допомога при ураженні електричним струмом. 4.2 Організація праці на робочому місці. Висновки. Перелік джерел. Додатки.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В.С., к.т.н., доцент		

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	27.01.2025	*Виконано*
2.	Підбір та опрацювання літературних джерел по темі кваліфікаційної роботи		*Виконано*
3.	Виконання дослідження щодо вибору програмних	01.02.2025-03.02.2025	*Виконано*
	засобів та архітектури.		
4.	Розроблення інформаційної системи бронювання	04.02.2025-14.02.2025	*Виконано*
	робочих місць у коворкінгах із використанням		
	JavaScript		
5.	Оформлення розділу «Аналіз предметної області та	15.02.2025-27.02.2025	*Виконано*
	постановка задачі»		
6.	Оформлення розділу «Реалізація та проектування	15.02.2025-27.02.2025	*Виконано*
	Інформаційної системи бронювання робочих місць у		
	коворкінгах»		
7.	Оформлення розділу «Практична частина.	15.02.2025-27.02.2025	*Виконано*
	Демонстрація та тестування інформаційної системи»		
8.	Виконання завдання до підрозділу «Безпека життєдіяльності»	04.03.2025-07.03.2025	*Виконано*
9.	Виконання завдання до підрозділу «Основи охорони праці»	04.03.2025-07.03.2025	*Виконано*
10.	Оформлення кваліфікаційної роботи		*Виконано*
11.	Нормоконтроль		*Виконано*
12.	Перевірка на плагіат		*Виконано*
13.	Попередній захист кваліфікаційної роботи		*Виконано*
14.	Захист кваліфікаційної роботи		

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

Керівник роботи _____ Марценко С.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка інформаційної системи бронювання робочих місць у коворкінгах // Кваліфікаційна робота освітнього рівня «Бакалавр» // Семенюк Владислав Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СТ-41 // Тернопіль, 2025 // С. 67, рис. – 16 , табл. – 1 , кресл. – 0 , додат. – 15, бібліогр. – 40 .

Ключові слова: коворкінг, бронювання, web, застосунок, react, node.js, mongoDB, інтерфейс, база даних.

Кваліфікаційна робота присвячена розробці інформаційної системи бронювання робочих місць у коворкінгах.

У першому розділі обґрунтовано актуальність теми, проаналізовано сервіси бронювання, визначено вимоги до системи та обрано компонентно-орієнтовану архітектуру.

У другому розділі описано проєктування структури веб-застосунку, моделювання бази даних на MongoDB, реалізацію серверної частини з використанням Node.js та Express, а також створення адаптивного інтерфейсу користувача на базі React та Tailwind CSS.

У третьому розділі подано опис реалізованої функціональності веб-застосунку, наведено результати ручного тестування та виправлення помилок, що підтверджують працездатність системи в типових сценаріях використання.

У четвертому розділі розглянуто питання охорони праці та безпеки життєдіяльності. Наведено долікарські допомогу при ураженні електричним струмом та організацію праці на робочому місці.

ANNOTATION

Development of an Information System for Booking Workspaces in Coworking Spaces // Qualification work of the educational level «Bachelor» // Semeniuk Vladyslav Volodymyrovych // Ternopil Ivan Pulyu National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group ST-41 // Ternopil, 2025 // P. 67 , fig. – 16 , tabl. – 1 , chair. – 0 , annexes. – 15, references – 40.

Keywords: coworking, booking, web, application, react, node.js, mongoDB, interface, database.

The qualification thesis is dedicated to the development of an information system for booking workspaces in coworking spaces.

The first chapter justifies the relevance of the topic, analyzes existing booking services, defines the system requirements, and explains the choice of a component-based architecture.

The second chapter describes the design of the web application structure, database modeling using MongoDB, backend implementation with Node.js and Express, as well as the creation of a responsive user interface using React and Tailwind CSS.

The third chapter presents the implemented functionality of the web application, the results of manual testing, and bug fixes that confirm the system's performance in typical usage scenarios.

The fourth chapter addresses occupational safety and life safety issues. It presents first aid measures for electric shock injuries and the organization of workplace safety.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ СКОРОЧЕНЬ І ТЕРМІНІВ

API (Application Programming Interface) — набір протоколів та інструментів для взаємодії між програмними компонентами.

CRUD (Create, Read, Update, Delete) — основні операції для роботи з даними.

CSS (Cascading Style Sheets) — мова стилів для оформлення вебсторінок.

DOM (Document Object Model) — модель документа, яка дозволяє програмно змінювати структуру вебсторінки.

JSX — синтаксичне розширення JavaScript, що дозволяє створювати інтерфейс у React.

MongoDB — документно-орієнтована нереляційна база даних NoSQL.

Node.js — середовище виконання JavaScript на стороні сервера.

React — бібліотека JavaScript для побудови користувацького інтерфейсу.

REST API — архітектурний стиль обміну даними між клієнтом і сервером за допомогою HTTP-запитів.

UI (User Interface) — інтерфейс користувача.

UX (User Experience) — досвід користувача при взаємодії з інтерфейсом.

Адміністратор — користувач з розширеними правами, який має доступ до керування вмістом системи.

БД — база даних, організоване сховище структурованої інформації.

Веб-застосунок — клієнт-серверна програма, що працює через веббраузер.

Користувач — особа, яка використовує веб-застосунок для взаємодії з системою.

Коворкінг — робочий простір, який користувач може забронювати на певний час.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі коворкінги стають все більш популярними серед фахівців різних галузей, фрілансерів, стартапів та малих підприємств. Це зумовлено потребою у гнучких умовах праці, економії ресурсів і створенні комфортного робочого простору. Зі збільшенням кількості коворкінгів зростає і потреба в ефективному управлінні їхніми ресурсами, зокрема у бронюванні робочих місць. Традиційні методи бронювання вже не відповідають сучасним вимогам до швидкості, зручності та доступності сервісів. У зв’язку з цим виникає потреба у впровадженні сучасних інформаційних систем, які дозволяють автоматизувати процеси бронювання, підвищити ефективність управління коворкінгом та покращити користувацький досвід.

Розробка інформаційної системи бронювання робочих місць у коворкінгу дозволяє автоматизувати рутинні процеси управління простором, надати користувачам зручний інтерфейс для перегляду доступних місць та здійснення бронювання у реальному часі. Система забезпечує ефективну організацію робочих місць, що особливо важливо в умовах гнучкої зайнятості та популярності віддаленої роботи. Для реалізації функціонального та зручного користувацького інтерфейсу було запропоновано створити веб-застосунок із використанням мови програмування JavaScript та фреймворку React.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є розробка інформаційної системи для бронювання робочих місць у коворкінгах, що забезпечує зручний інтерфейс для користувачів та ефективне адміністрування ресурсів. Для досягнення поставленої мети необхідно виконати такі завдання:

- Проаналізувати існуючі інформаційні системи для бронювання у коворкінгах та суміжних сферах.
- Визначити основні вимоги користувачів до системи бронювання та сформулювати постановку задачі.

- Розробити архітектуру інформаційної системи з урахуванням принципів масштабованості, безпеки та зручності у використанні.
- Реалізувати функціональні модулі системи, зокрема реєстрацію та авторизацію користувачів, перегляд доступних місць, бронювання, перегляд історії бронювання, адміністрування ресурсів.
- Забезпечити інтеграцію фронтенду та бекенду за допомогою сучасних веб-технологій.
- Провести тестування розробленої системи, оцінити її функціональність, надійність та зручність для користувачів.

Практичне значення одержаних результатів. Розроблений веб-застосунок для бронювання робочих місць у коворкінгах має значне практичне значення, оскільки забезпечує ефективну цифрову підтримку процесу організації спільного простору для роботи. Завдяки автоматизації процесу бронювання, система дозволяє зменшити адміністративне навантаження на персонал, уникнути дублювання бронювань, а також забезпечити користувачам зручний і швидкий доступ до інформації про доступні робочі місця, дати та часові слоти.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Огляд існуючих рішень для бронювання робочих місць у коворкінгу

Для визначення оптимального функціоналу та інтерфейсу розроблюваної інформаційної системи бронювання робочих місць коворкінгу було проаналізовано три популярні сервіси: програма для бронювання – EasyWeek, Carpe Diem Coworking, ITEAHub.

Почнемо з першого сервісу EasyWeek. EasyWeek – це SaaS-платформа з різноманітними інструментами для онлайн-бронювання, CRM, аналітики та платежів, яка підтримує як окремі локації, так і коворкінги [1]. Серед ключових можливостей: бронювання в реальному часі, гнучке налаштування правил, управління декількома локаціями, інтеграція з календарями, аналітикою та платежами. На рисунку 1.1 можем побачити дизайн веб-застосунку.

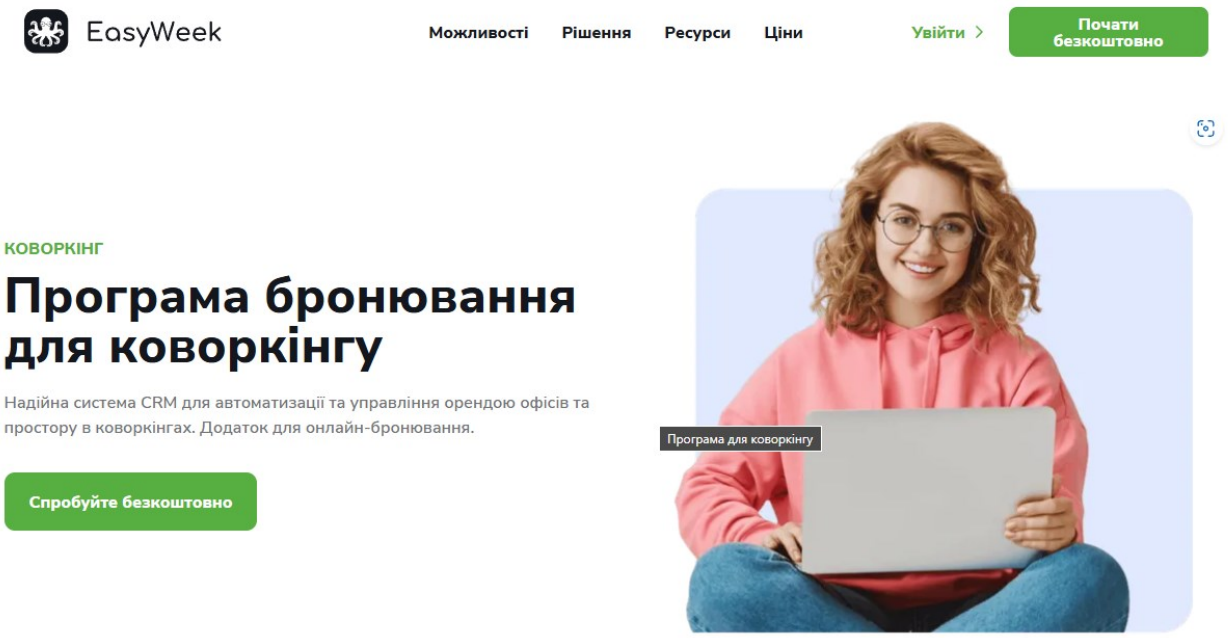

Рисунок 1.1 – Дизайн веб-застосунку EasyWeek

Серед плюсів цього додатку виділяють: інтуїтивний інтерфейс, цілодобова підтримка клієнтів. Серед мінусів користувачі виділяють проблеми з обробкою скасувань, відсутність автоматичних листів про скасування.

Далі перейдемо до наступного веб-застосунку Carpe Diem Coworking. Пропонує як загальні робочі зони, так і приватні, звукоізольовані кімнати, орієнтуючись на фрілансерів, студентів та невеликі команди [2]. На рисунку 1.2 можемо побачити дизайн веб-застосунку.

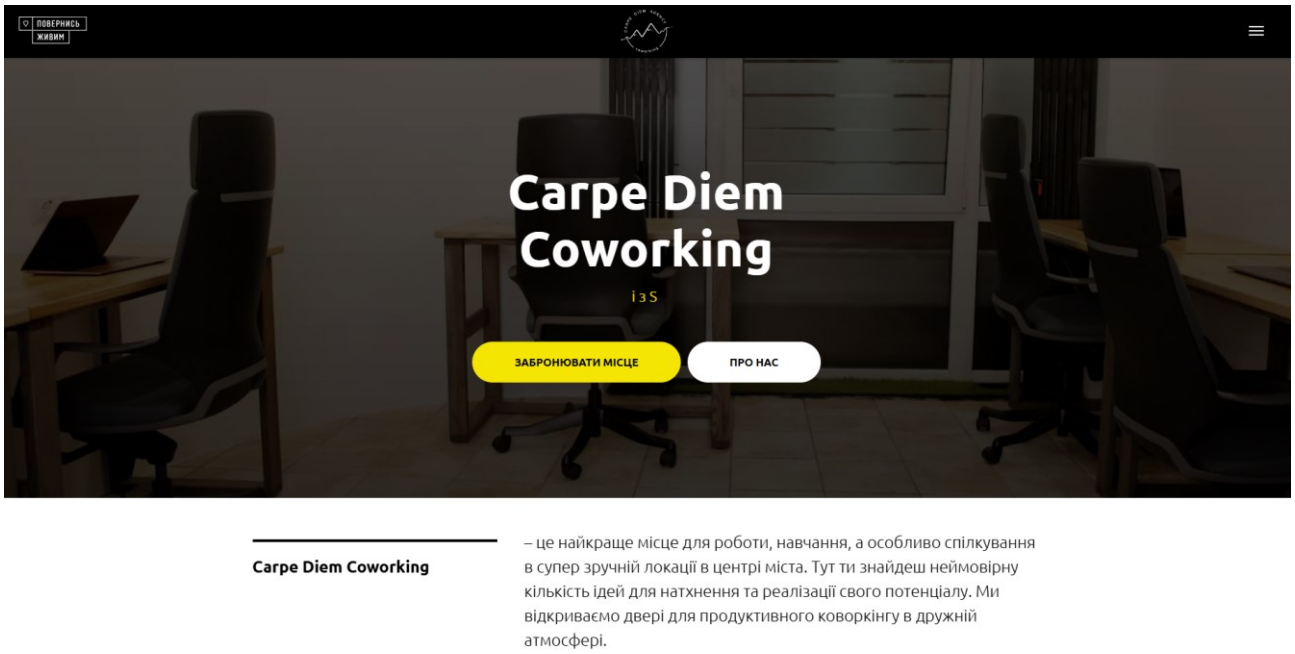

Рисунок 1.2 – Дизайн веб-застосунку Carpe Diem Coworking

Серед плюсів веб-застосунку виділяють: доступні ціни; приватні кімнати для різних потреб.

З недоліків користувачі жаліються на відсутність аналітики чи віджетів бронювання на сайті, не видно інтеграцій з CRM, а також немає вбудованої системи online booking.

Перейдемо до останнього третього веб-застосунку ITEAHub. ITEAHub – сильний бренд із продуманими тарифами та спільнотою, який орієнтований на IT-спеціалістів, студентів та фрілансерів [3]. На рисунку 1.3 можемо побачити дизайн веб-застосунку.

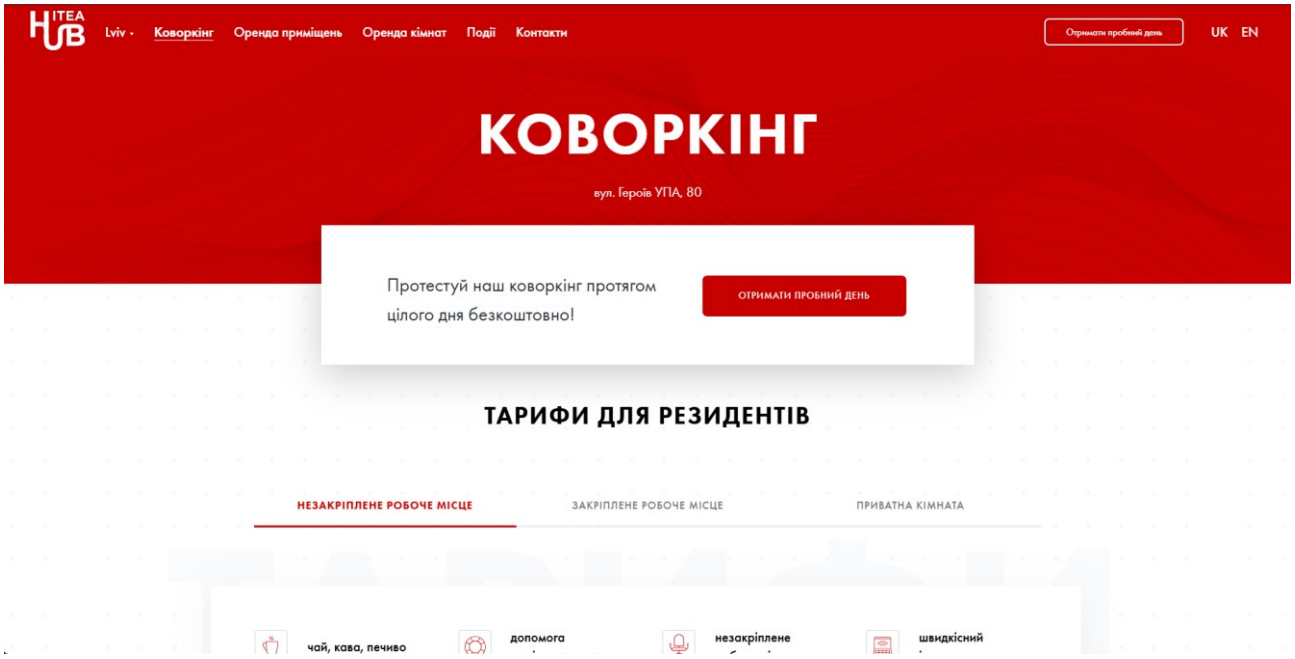

Рисунок 1.3 – Дизайн веб-застосунку ITEAHub

До особливостей цього веб-застосунку належать: пробний день для ознайомлення з послугами; прозора система тарифів з чіткими цінами й послугами; широкий спектр послуг.

Однак веб-застосунок також має і недоліки, у нього відсутня автоматизована онлайн система бронювання, користувачі жаліються на затримки відповіді менеджера, а також не видно реального стану доступності робочих місць.

Отже, незважаючи на наявність великої кількості існуючих рішень для бронювання робочих місць у коворкінгах, користувачі часто стикаються з низкою проблем – відсутністю інтерактивного інтерфейсу для вибору робочого місця, неможливістю здійснити бронювання в реальному часі, до необхідності ручної обробки заявок через контактні форми. Більшість платформ також не пропонують інтеграцій із платіжними сервісами, календарями або не мають аналітики завантаженості. Тому актуальним є створення зручної, інтуїтивно зрозумілого веб-застосунку бронювання, який автоматизує ключові процеси, враховує потреби локальних коворкінгів і надає повноцінний функціонал без надмірних обмежень для користувача.

1.2 Аналіз вибору мови програмування та середовища розробки, що використовується для створення інформаційної системи

Розробка веб-застосунків є однією з найактуальніших і найпоширеніших задач у сучасній IT-галузі. У зв'язку зі стрімкою цифровізацією, більшість користувачів віддають перевагу онлайн-сервісам, які доступні з будь-якого пристрою без необхідності встановлення додаткового програмного забезпечення. Особливо це актуально для сервісів бронювання, де оперативність доступу до інформації та простота взаємодії з інтерфейсом мають ключове значення [29].

Сучасні веб-застосунки працюють через браузери та є незалежними від операційної системи – будь то Windows, macOS, Linux, Android чи iOS. Це забезпечує універсальність доступу для користувачів із різними пристроями: ноутбуками, планшетами чи смартфонами.

Для створення таких застосунків використовуються спеціалізовані мови програмування та середовища розробки. Основною мовою розробки веб-застосунків є JavaScript, яка підтримується всіма сучасними браузерами та має велику екосистему фреймворків і бібліотек [4]. Однією з найпопулярніших бібліотек для створення інтерфейсів є React, розроблений компанією Meta [5]. Він дозволяє ефективно створювати динамічні та масштабовані користувацькі інтерфейси, що особливо важливо для інформаційних систем із великою кількістю взаємодій – таких як системи онлайн-бронювання.

Однак, окрім JavaScript, для веб розробки можуть також використовуватись інші мови, такі як: TypeScript – над множина JavaScript із підтримкою статичної типізації, яка дозволяє зменшити кількість помилок і підвищити масштабованість коду; Python – популярна мова для серверної частини застосунків, однак вона також може використовуватись для рендерингу інтерфейсів у поєднанні з HTML/CSS; PHP – класична мова для створення веб-сайтів, проте її популярність знижується у порівнянні з JavaScript [6].

Серед альтернатив фреймворків для побудови інтерфейсів можна виділити: Vue.js – легкий і гнучкий фреймворк, орієнтований на простоту та швидке освоєння; Angular – фреймворк від Google, що надає повноцінну архітектуру для масштабних застосунків, проте вимагає більше часу на вивчення; Svelte – сучасний фреймворк, що компілює код у мінімальний JavaScript без обов'язкового середовища виконання.

На відміну від мобільної розробки, де для операційних систем (Android та iOS) часто потрібно створювати окремі додатки або використовувати кросплатформенні інструменти, веб-застосунок створюється один раз і є доступним для всіх користувачів незалежно від їхньої платформи. Це дозволяє суттєво скоротити витрати часу та ресурсів на розробку і супровід, зберігаючи при цьому високий рівень доступності та функціональності.

Таким чином, для реалізації інформаційної системи бронювання робочих місць у коворкінгах із використанням JavaScript як основної мови програмування та React як основного фреймворку. Такий вибір забезпечує ефективну розробку, зручність для користувача, гнучке керування даними, а також легкість у подальшому розширенні та підтримці системи.

Розглянемо першу мову програмування JavaScript. Класична і найбільш поширена мова для розробки веб-застосунків. Вона є стандартом для створення інтерактивних інтерфейсів у браузерах і підтримується всіма сучасними веб платформами [4]. Завдяки JavaScript було створено величезну кількість веб-застосунків і сайтів, які щоденно використовуються мільйонами користувачами по всьому світі.

Однією з ключових переваг JavaScript є її універсальність і гнучкість. Мова дозволяє реалізовувати як прості анімації та взаємодії, так і складні одно сторінкові застосунки (SPA) з багатим функціоналом. JavaScript має надзвичайно велику і активну спільноту розробників, що забезпечує доступ до численних ресурсів, бібліотек і фреймворків які значно полегшують і пришвидшують розробку. Коли виникають труднощі, важливою перевагою є можливість легко

знайти відповіді на питання або готові рішення серед широкої бази знань та прикладів.

Тепер про складнощі, однією з головних проблем під час використання JavaScript у порівнянні з сучасними надбудовами, такими як TypeScript, є те, що JavaScript іноді вимагає написання більшого обсягу коду для реалізації складної логіки з перевіркою типів і структурованістю. Це може призвести до ускладнення підтримки коду, особливо у великих проектах.

Додатково, синтаксис JavaScript не завжди забезпечує статичну типізацію, що ускладнює виявлення помилок на ранніх етапах розробки. Це збільшує ризик появи багів у роботі застосунку, особливо при масштабуванні коду.

Також JavaScript, як мова з динамічною типізацією, часом менш передбачуваний у великих проектах, що може подовжувати час розробки через необхідність додаткового тестування та відкоригування.

Розглянемо тепер TypeScript. Це сучасна надмножина JavaScript, яка додає до неї підтримку статичної типізації і розширені можливості для розробників. TypeScript компілюється у JavaScript, що дозволяє виконувати код у будь-якому браузері або середовищі, яке підтримує JavaScript [7].

Однією з ключових переваг TypeScript є можливість виявляти помилки ще на етапі компіляції завдяки статичній типізації, що значно підвищує надійність і якість коду. Це особливо важливо у великих і складних веб-застосунках, таких як інформаційні системи бронювання, де потрібно керувати великою кількістю даних і взаємодій.

TypeScript також пропонує сучасний синтаксис і потужні інструменти для організації коду, що полегшує розробку, масштабування та підтримку проєкту. Крім того, TypeScript сумісний із усіма популярними бібліотеками і фреймворками JavaScript, зокрема React, що робить його відмінним вибором для розробки клієнтської частини веб-застосунків.

Незважаючи на численні переваги, використання TypeScript має і певні складнощі. По-перше, додавання статичної типізації та суворих правил синтаксису вимагає від розробників додаткового часу на вивчення мови та

налаштування середовища розробки. Це може ускладнити початковий етап проєкту. По-друге, написання типів та анотацій часто збільшує обсяг коду, що може призводити до більш об’ємних файлів у порівнянні з JavaScript. Це вимагає додаткових зусиль на підтримку і перепроєктування коду, особливо коли типи змінюються або проєкт активно розвивається. Ще одним викликом є те, що не всі сторонні бібліотеки мають повну або актуальну підтримку типів для TypeScript, що може призвести до необхідності створювати власні типові оголошення або використовувати менш точні типи, знижуючи тим самим ефективність типізації.

Розглянемо ще середовище розробки – Visual Studio Code, яке є одним із найпопулярніших редакторів коду для створення веб-застосунків. VS Code було розроблено компанією Microsoft і вперше представлено у квітні 2015 року [8]. Це легке, але водночас функціонально потужне інтегроване середовище розробки, яке підтримує велику кількість мов програмування, зокрема JavaScript та TypeScript. Однією з основних переваг VS Code є його висока гнучкість завдяки розширенням, які дозволяють адаптувати редактор під будь-які потреби розробника. Приклад інтерфейсу Visual Studio Code зображено на рисунку 1.4.

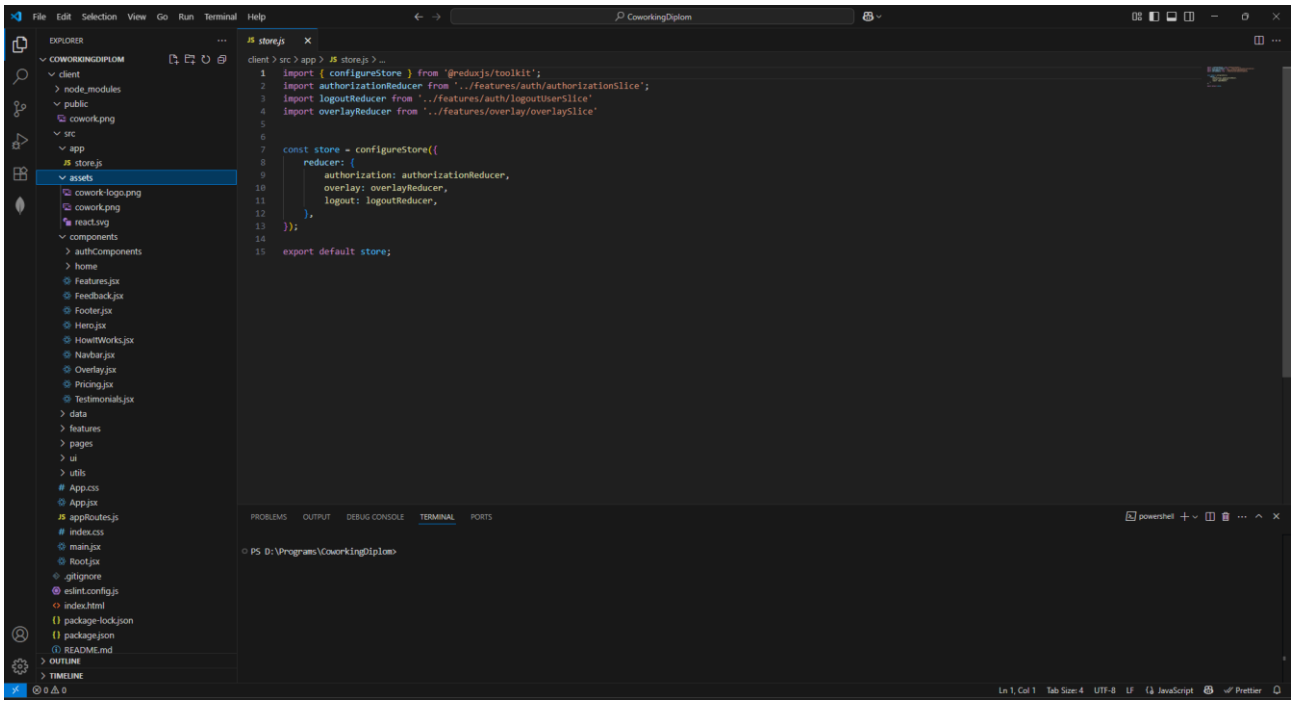

Рисунок 1.4 – Інтерфейс Visual Studio Code

Таким чином, комбінація Visual Studio Code + JavaScript є оптимальним вибором для розробки веб-застосунку інформаційної системи бронювання робочих місць у коворкінгах. Його гнучкість, розширюваність та підтримка сучасних технологій забезпечують ефективність розробки та комфорт користувача.

1.3 Постановка задачі

Інформаційна система бронювання робочих місць у коворкінгах призначена для автоматизації процесу резервування, управління і контролю доступності робочих зон у спільних офісних просторах [9]. Основна мета системи забезпечити користувачам зручний інтерфейс для пошуку, вибору та бронювання робочих місць з урахуванням їхніх потреб і доступності, а також надати адміністраторам ефективні інструменти для управління ресурсами коворкінгу.

На етапі проектування можна виділити наступні сутності:

- Користувач (людина, яка використовує систему для бронювання робочих місць; характеризується ідентифікатором, іменем, електронною поштою, роллю, контактними даними).
- Робоче місце (окрема робоча зона або робочий стіл у коворкінгу; характеризується унікальним номером, типом, розташуванням, наявністю додаткових опцій).
- Бронювання (інформація про резервування робочого місця; характеризується користувачем, робочим місцем, датою і часом початку і завершення бронювання, статусом).
- Коворкінг (об’єкт, що включає набір робочих місць, розташування, графік роботи, правила користування).
- Оплата (дані про оплату бронювання; включає суму, спосіб оплати, статус платежу, дату проведення транзакції).

Проаналізувавши предметну область і додатки, можна сформувати загальні вимоги до розробки інформаційної системи:

1. Функціональні вимоги:

- Реєстрація та авторизація користувачів із розмежуванням ролей.
- Можливість пошуку та фільтрації робочих місць за різними параметрами.
- Створення, перегляд, редагування і скасування бронювання.
- Автоматичне оновлення інформації про доступність робочих місць у режимі реального часу.
- Надання адміністратором інструментів для керування робочими зонами та бронюванням.
- Повідомлення користувачів про підтвердження, нагадування або зміну статусу бронювання.
- Можливість оплати бронювання.
- Збереження історії бронювання користувача.

2. Нефункціональні вимоги:

- Зручний, інтуїтивно зрозумілий користувацький інтерфейс, реалізований із використанням React.
- Висока продуктивність і швидкість відгуку системи.
- Масштабованість із можливістю подальшого розширення функціоналу.
- Безпека даних користувачів і конфіденційність інформації.
- Забезпечення крос браузерної сумісності та адаптивності інтерфейсу.

Таким чином, розроблена інформаційна система має забезпечити ефективне і зручне керування процесом бронювання робочих місць у коворкінгах, що сприятиме підвищенню комфорту користувачів і оптимізації роботи адміністрації.

1.4 Висновок до першого розділу

В першому розділі кваліфікаційної роботи був проведений огляд існуючих рішень у сфері інформаційних систем бронювання робочих місць у коворкінгах. На основі проведеного аналізу можна зробити висновок, що попри наявність основного функціоналу, більшість таких рішень мають спільні недоліки: обмежена гнучкість у пошуку місць, недостатня інтерактивність інтерфейсу, відсутність повноцінного інструментарію для адміністрування, а також слабка адаптація під мобільні пристрої. Також було здійснено аналіз вибору мови програмування та середовища розробки, що використовуються для створення веб-застосунків. Встановлено, що JavaScript у поєднанні з фреймворком React є доцільний рішенням завдяки своїй гнучкості, розвинутій екосистемі та зручності реалізації інтерактивного інтерфейсу. Як середовище розробки обрано Visual Studio Code, що забезпечує ефективність роботи над проєктом. У межах постановки задачі були визначені основні функціональні й нефункціональні вимоги до системи та окреслені ключові сутності, необхідні для реалізації повноцінної інформаційної системи бронювання робочих місць.

РОЗДІЛ 2. РЕАЛІЗАЦІЯ ТА ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Архітектура інформаційної системи

Під час вибору середовища розробки було ухвалено рішення на користь створення веб-застосунку, важливою умовою для нього є забезпечення структурованого, масштабованого та зручного для підтримки програмного коду [30]. Досягти цього дозволяє застосування відповідних архітектурних підходів, які сприяють чіткому розмежуванню відповідальностей між окремими логічними компонентами системи. У цьому проєкті було прийнято рішення використовувати архітектурну модель Component-Based Architecture, яка є типовою для веб-застосунків, реалізованих за допомогою фреймворку React [10].

Цей підхід передбачає побудову інтерфейсу у вигляді окремих ізольованих компонентів, кожен з яких виконує визначену функцію: відображення елементів, обробку подій або логіку управління станом. Така модульність забезпечує гнучкість при оновленні, тестуванні або розширенні застосунку.

До основних рівнів архітектури можна віднести:

Компоненти представлення (Presentation Layer) – відповідають за візуалізацію інтерфейсу користувача. У даному випадку це компоненти React, які відображають форму бронювання, список доступних робочих місць, календар та інші елементи.

Логіка управління станом (State Management) – реалізована через React hooks (useState, useEffect) та за потреби може бути масштабована до використання глобального стану за допомогою Redux або Context API.

Шар доступу до даних (Data Layer) – включає функції для обміну даними з серверною частиною через HTTP-запити. Для цього використовується бібліотека Axios, яка дозволяє взаємодіяти з REST API для створення, перегляду та видалення бронювання [11].

Навігація – організована за допомогою бібліотеки React Router, яка забезпечує маршрутизацію між сторінками веб-застосунку (головна сторінка, панель адміністратора, сторінка авторизації).

На рисунку 2.1 можем побачити візуальне представлення цієї архітектури.

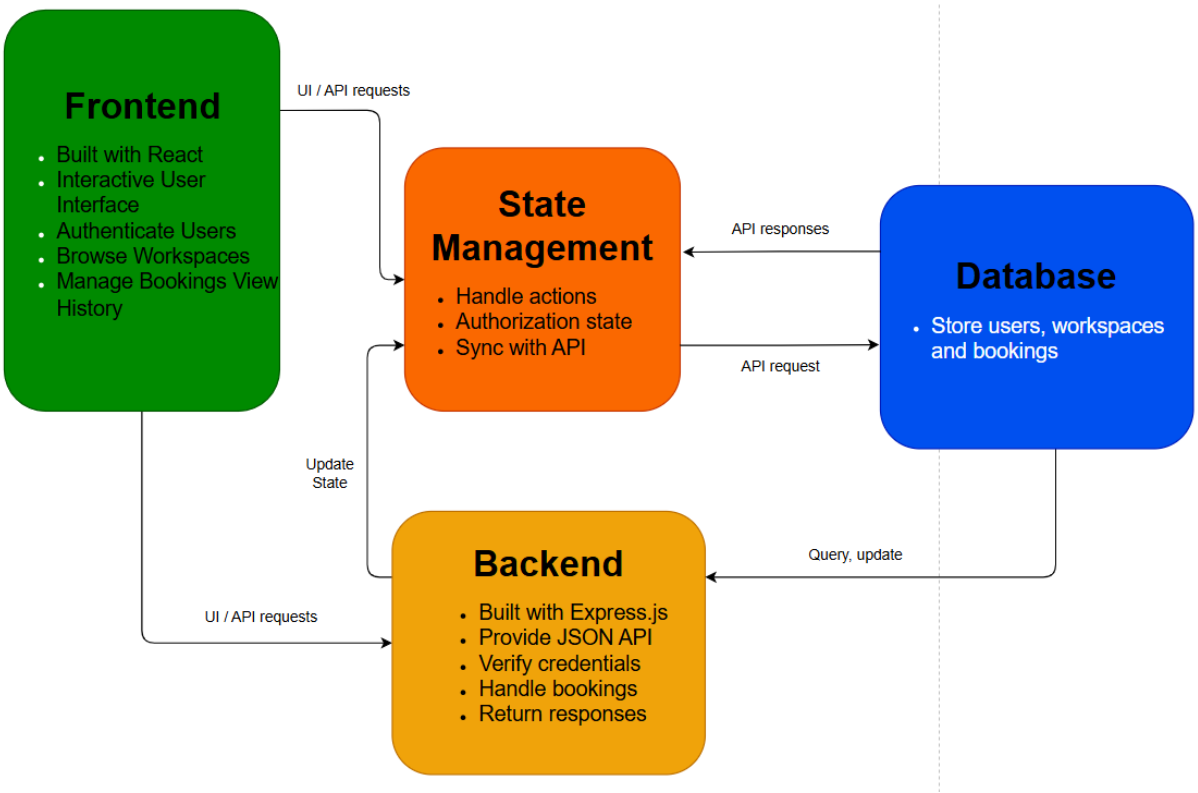

Рисунок 2.1 – Принцип роботи архітектури

У рамках розробки веб-застосунку для бронювання робочих місць у коворкінгах було обрано компонентно-орієнтовану архітектуру, що реалізована з використанням фреймворку React для клієнтської частини та Express.js для серверної [12]. Такий підхід дозволяє ефективно розділяти відповідальність між логікою відображення, керування станом та обробкою даних, що сприяє масштабованості й супроводжуваності системи.

Архітектура веб-застосунку поділяється на такі рівні:

1. Frontend (Клієнтська частина) реалізована з використанням React. Вона відповідає за формування інтерактивного інтерфейсу, який включає:

- компоненти для авторизації користувачів;

- перегляду вільних робочих місць;
- створення та скасування бронювання;
- перегляду історії користування системою;

2. State Management (Управління станом) забезпечується за допомогою Context API або зовнішніх бібліотек (Zustand або Redux) – залежно від конкретної реалізації). Цей рівень відповідає:

- обробку подій користувача;
- зберігання стану авторизації;
- інформації про бронювання, а також асинхронну взаємодію з API;

3. Backend (Серверна частина) побудована на основі Express.js – мінімалістичного Node.js-фреймворку. Сервер обробляє REST-запити від клієнта, забезпечує маршрутизацію, взаємодіє з базою даних (MongoDB або PostgreSQL) та відповідає за:

- реєстрацію та автентифікацію користувачів;
- CRUD-операції з бронюванням та робочими місцями;
- валідацію даних та обробку помилок;
- захист ресурсів через middleware (перевірка токенів доступу).

4. Database (База даних) – на цьому рівні зберігається інформація про користувачів, робочі місця та історія бронювання. Структура бази даних спроєктована таким чином, щоб забезпечити швидкий доступ до актуальної інформації та гарантувати цілісність транзакцій.

Обрана архітектура Component-Based Architecture забезпечує чітке розділення обов’язків, сприяє повторному використанню компонентів, дозволяє масштабувати застосунок та реалізовувати нові функціональні модулі з мінімальними змінами в існуючій структурі. Такий підхід є доцільним для інформаційних систем середнього рівня складності з активною взаємодією між користувачем та даними. На рисунку 2.2 можемо побачити структуру нашого веб-застосунку.

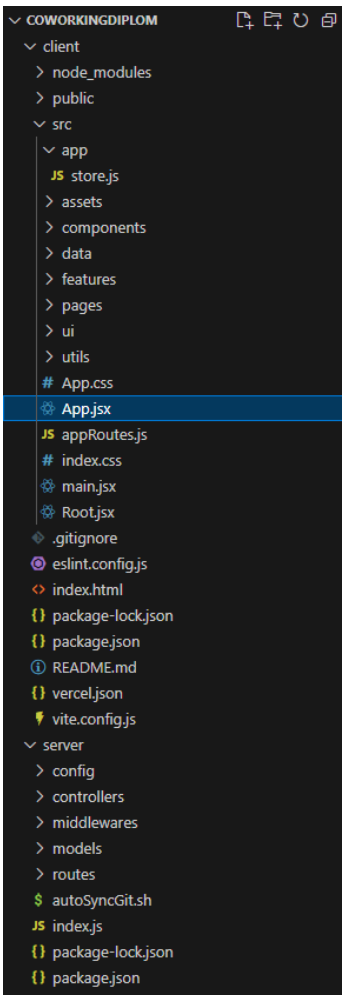

Рисунок 2.2 – Структура веб-застосунку бронювання робочих місць у коворкінгах

Обрана структура є достатньо простою та зручною для веб-застосунків середнього масштабу, де немає потреби у впровадженні складної багаторівневої архітектури. Відсутність окремого шару репозиторію компенсується чітким розмежуванням відповідальностей між модулями components, services, store та pages, що забезпечує підтримуваність і розширюваність веб-застосунку.

2.2 Пошук актантів та варіантів використання

Під час проєктування інформаційної системи для бронювання робочих місць у коворкінгах важливим етапом стало визначення ключових актантів системи та формування основних варіантів використання. Це дозволяє

побудувати чітку логіку функціонування системи, а також спроєктувати інтерфейс, орієнтований на кінцевого користувача.

Актантами (або користувачами системи) у контексті даного веб-застосунку є:

- Користувач – відвідувач коворкінгу, який здійснює реєстрацію, переглядає доступні робочі місця, бронює їх на певні дати та керує своїм бронюванням.
- Адміністратор – представник коворкінгу, який має змогу керувати користувачами, робочими місцями, переглядати загальну статистику та адмініструвати систему.
- Система – забезпечує функціонування механізмів перевірки, авторизації, збереження та обробки даних, логіку бронювання, а також підтримку інтерфейсу відповідно до статусу користувача.

На основі виділених актантів було визначено основні варіанти використання, які охоплюють базову функціональність веб-застосунку:

1. Реєстрація користувача – введення логіна, пароля та імені для створення нового облікового запису.
2. Авторизація – вхід до системи із використанням логіна і пароля.
3. Перегляд доступних коворкінгів – користувач може переглянути вільні місця на конкретну дату та обрати відповідне.
4. Фільтрація робочих місць – пошук місць за параметрами: тип (відкритий простір, кімната), кількість місць, наявність обладнання тощо.
5. Бронювання місця – обрання робочого місця, дати, часу початку і завершення бронювання.
6. Перегляд особистих бронювань – користувач може переглядати перелік майбутніх і завершених бронювань.
7. Скасування бронювання – скасування активного бронювання за умови дотримання встановлених обмежень.
8. Редагування профілю користувача – зміна особистих даних, контактної інформації, пароля.

9. Вихід із системи – завершення сеансу користувача.

10. Керування робочими місцями (для адміністратора) – додавання, редагування або видалення інформації про робочі місця.

11. Перегляд усіх бронювань (для адміністратора) – перегляд історії бронювання усіх користувачів.

12. Керування користувачами (для адміністратора) – можливість блокування акаунтів або зміни ролей користувачів.

Для наочного представлення взаємодії актантів із системою було побудовано діаграму варіантів використання [13] за допомогою додатку draw.io [14] – рисунок 2.3.

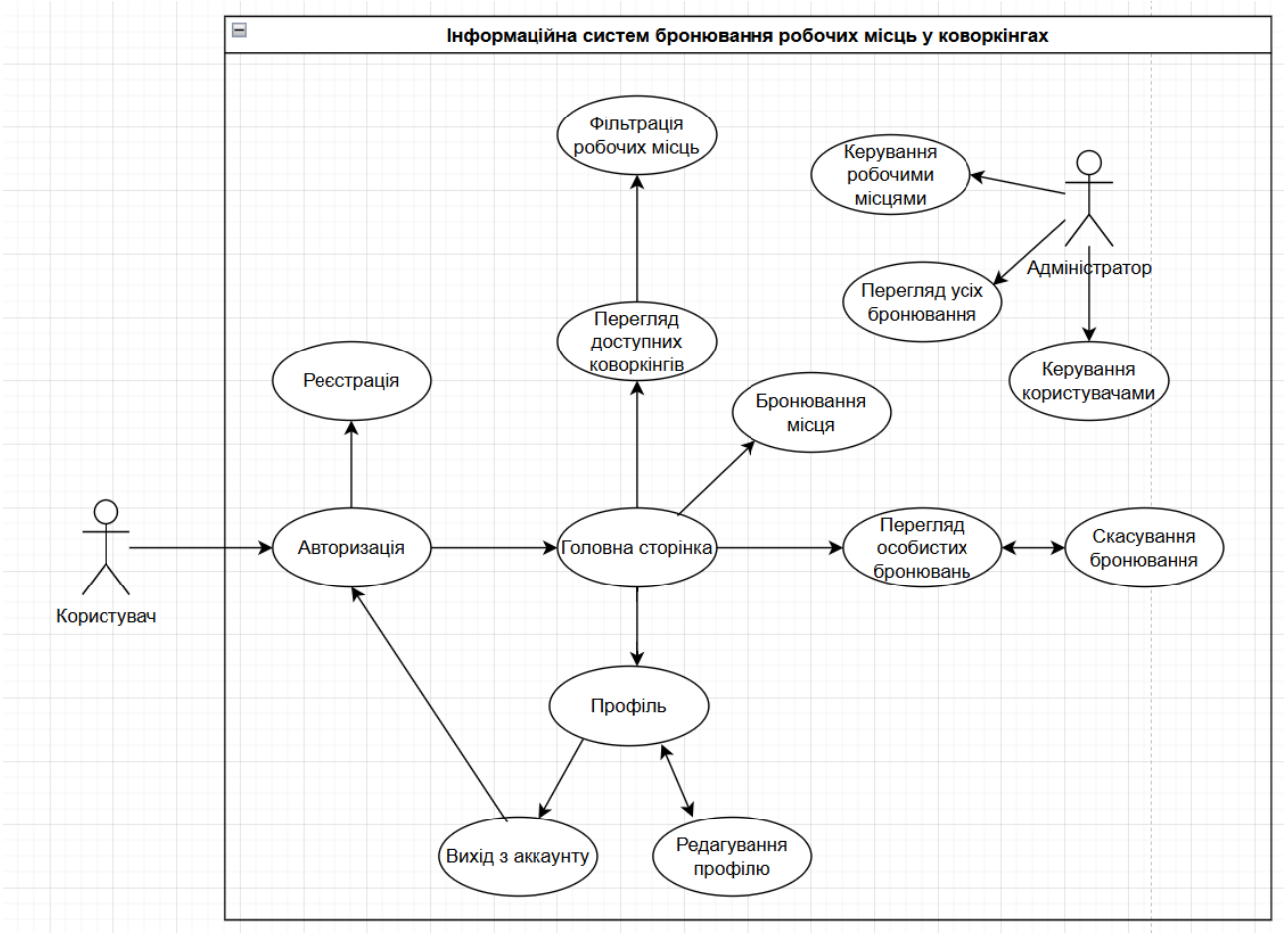

Рисунок 2.3 – Діаграма варіантів використання інформаційонї системи бронювання робочих місць у коворкінгах

Ця діаграма демонструє, як користувач взаємодіє із системою через набір функціональних сценаріїв. Вона дозволяє побачити загальну архітектуру дій та зрозуміти, як реалізована логіка роботи інформаційної системи на основі веб-застосунку.

2.3 Проектування та реалізація екранів користувацького інтерфейсу

Користувацький інтерфейс є невід'ємною частиною інформаційної системи, оскільки саме через нього користувачі взаємодіють із функціональними можливостями веб-застосунку [15]. У межах розробки інформаційної системи для бронювання робочих місць у коворкінгах було створено набір інтерфейсних екранів, кожен з яких виконує певну роль у процесі взаємодії користувача із системою [31].

Розробка інтерфейсу здійснювалася із використанням бібліотеки React, що надає широкі можливості для створення динамічних, реактивних та масштабованих інтерфейсів. Для організації маршрутизації застосовано бібліотеку React Router, яка забезпечує зручне перемикання між екранами без перезавантаження сторінки [16]. Були відокремлені наступні екрани:

- екран авторизації (LoginPage), за допомогою якого користувач проходить авторизацію;
- екран реєстрації (RegisterPage), за допомогою якого користувач проходить реєстрацію;
- головна сторінка (HomePage), яка виконує роль дашборду та відображає загальну інформацію про доступні локації, активні бронювання користувача;
- екран вибору локації (LocationSelectionPage), за допомогою якого користувач обирає коворкінг для подальшого перегляду вільних робочих місць;
- екран перегляду місць (WorkspaceViewPage), надає візуальне представлення робочих місць на обраній локації, їхній статус (вільне або заброньоване), а також доступ до функції бронювання;

- екран бронювання (BookingPage), який дозволяє користувачу вибрати дату, час, тривалість бронювання та підтвердження заявки;
- екран перегляду бронювань (MyBookingsPage), за допомогою якого можемо переглядати історію та активні бронювання, а також скасувати або змінити бронювання;
- екран профілю користувача (ProfilePage), який дозволяє редагувати персональні дані, змінювати пароль, а також вийти з аккаунту;
- адміністративна панель (AdminDashboard), яка доступна лише користувачам із правами адміністратора, дозволяє керувати локаціями, додавати нові робочі місця, змінювати розклад доступності та переглядати статистику використання системи.

Для переміщення між екранами у веб-застосунку використовується система маршрутизації React Router, яка забезпечує декларативну навігацію між сторінками. У додатку використовується єдиний (Routes) компонент, у межах якого визначено маршрути (Route path) для кожної сторінки. Переміщення між екранами реалізується за допомогою хука useNavigate, який дозволяє здійснювати перехід на інші маршрути (наприклад, navigate("/home")) або повертатися назад (navigate(-1)). Це дає змогу гнучко керувати переходами та зберігати контекст, забезпечуючи зручну та зрозумілу навігацію для користувача.

Домашнім (першим при запуску) екраном є сторінка авторизації (LoginPage), або головна сторінка (HomePage) – залежно від того, чи користувач уже авторизувався. Якщо в облікових даних збережено userId у localStorage, його буде автоматично перенаправлено на головну сторінку. Інакше користувач потрапить на сторінку входу, де може також перейти на сторінку реєстрації (RegisterPage), якщо не має облікового запису.

Після успішної авторизації користувач переходить на головну сторінку (HomePage), яка є центральним вузлом навігації. З неї користувач може перейти до сторінки локацій (LocationsPage), де відображаються доступні коворкінги з фільтрами та зображеннями. Сторінки бронювання (BookPage) для вибору

локації, дати та часу бронювання робочого місця. Сторінки перегляду бронювань (MyBookingsPage), де користувач бачить список активних та завершених бронювань і може скасувати поточне. Профілю користувача (ProfilePage), де можна редагувати особисті дані, змінити пароль або вийти з облікового запису.

Для зручності навігації у верхній частині сайту присутнє навігаційне меню, яке доступне на всіх сторінках після авторизації. Адміністратор має доступ до окремих сторінок для керування локаціями, користувачами та перегляду загальної статистики, які доступні через пункт меню "Admin".

2.4 Проектування бази даних

У процесі розробки інформаційної системи бронювання робочих місць у коворкінгах було обрано MongoDB як основну систему керування базами даних [17]. MongoDB — це документоорієнтована нереляційна база даних, яка забезпечує гнучкий підхід до зберігання даних у форматі BSON (бінарна форма JSON). Завдяки цьому розробникам не потрібно чітко визначати структуру таблиць наперед, що особливо зручно для динамічних веб-застосунків, структура яких може змінюватися у процесі розробки.

Однією з ключових переваг використання MongoDB є простота інтеграції з JavaScript/Node.js середовищем, що було вирішальним фактором при побудові серверної частини застосунку. Усі операції над базою даних (створення, читання, оновлення та видалення записів) реалізовані за допомогою бібліотеки Mongoose, яка надає зручний і типобезпечний інтерфейс для взаємодії з MongoDB. Mongoose дозволяє створювати моделі, які описують структуру документів у колекціях, що значно спрощує розробку та сприяє зменшенню кількості помилок.

Завдяки гнучкості MongoDB, система підтримує зберігання інформації про робочі простори, користувачів, бронювання, а також дозволяє ефективно реалізовувати фільтрацію, пошук і агрегацію даних. Наприклад, при виборі вільного місця користувачем застосунок формує запит до бази даних із

фільтрацією за датою, часом та ідентифікатором локації, що дозволяє оперативно визначити доступні слоти для бронювання.

Загалом, вибір MongoDB як СКБД забезпечив масштабованість, високу швидкодію при роботі з великими обсягами даних та спростив реалізацію гнучкої схеми зберігання, що є важливим у контексті веб-застосунків із постійно змінними функціональними вимогами. На рисунку 2.4 наведено принцип роботи бібліотеки MongoDB.

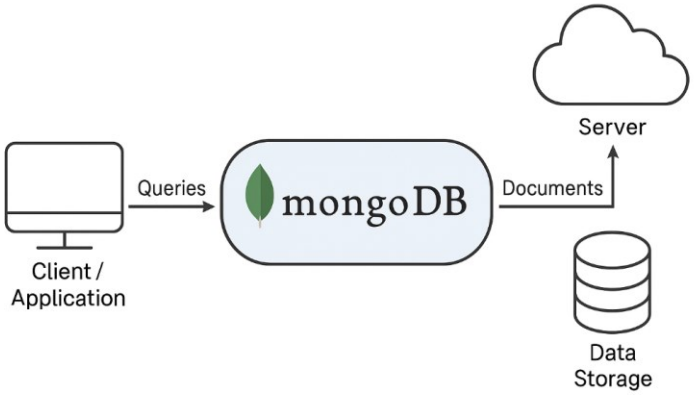

Рисунок 2.4 – Принцип роботи бібліотеки MongoDB

У розробленій інформаційній системі для бронювання робочих місць у коворкінгах було реалізовано такі сутності колекцій у базі даних MongoDB:

- User – зберігає облікову інформацію про користувачів: ім’я, електронну пошту, пароль, а також роль (користувач або адміністратор).
- Workspace – описує робочі місця: назва, локація, тип (відкритий простір, кімната), кількість місць, доступність обладнання та інші характеристики.
- Booking – зберігає інформацію про бронювання: ID користувача, ID робочого місця, дата та час початку і завершення бронювання, статус (активне, завершене, скасоване).
- Location – містить дані про фізичні локації коворкінгів, включаючи назву, адресу, фото та опис.

- Admin – колекція для зберігання дій адміністратора (наприклад, зміна доступності місця, блокування акаунтів).

Усі взаємозв’язки між сутностями реалізовані через зберігання відповідних ObjectId у документах MongoDB (наприклад, у колекції Booking зберігаються посилання на документи User та Workspace).

MongoDB обрана як гнучке NoSQL-рішення, що дозволяє легко масштабувати систему та працювати з нереляційними структурами. Створення колекцій не потребує явного визначення схеми, однак для забезпечення цілісності та перевірки даних використовуються Mongoose-схеми на рівні серверної частини застосунку.

Для автентифікації та авторизації реалізовано механізм збереження токенів доступу в localStorage, а також ідентифікацію користувача за допомогою JWT. При вході користувача відповідний токен передається у заголовку запиту для перевірки його прав доступу до ресурсів.

Загальну структуру системи та зв’язки між сутностями можна побачити на рисунку 2.5 – ER-діаграмі бази даних, створеній за допомогою сервісу Mermaid [18].

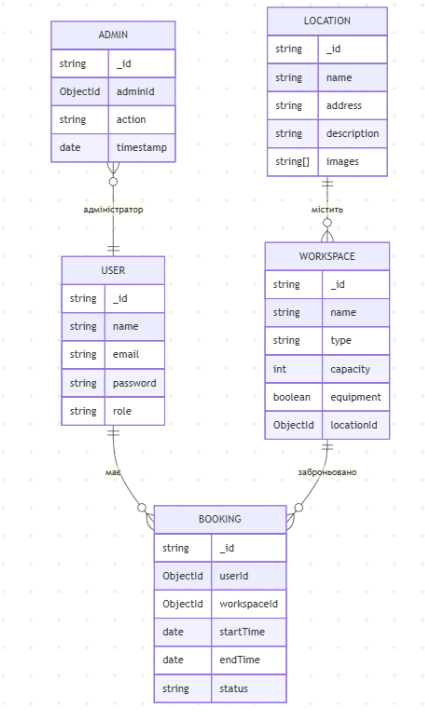

Рисунок 2.5 – ER-діаграма бази даних

Зв’язки між сутностями:

- User → Booking: Один до багатьох (1:N). Один користувач може мати декілька бронювань. У колекції Booking зберігається поле userId, яке посилається на відповідний запис у колекції User
- Workspace → Booking: Один до багатьох (1:N). Кожне робоче місце може бути заброньоване кілька разів у різні дні або години. У полі workspaceId сутності Booking зберігається ідентифікатор відповідного робочого місця.
- Location → Workspace: Один до багатьох (1:N). Одна локація може містити декілька робочих місць. У Workspace зберігається посилання locationId, що вказує на відповідну локацію.
- Admin → User: Один до багатьох (1:N). Кожен запис у колекції Admin зберігає інформацію про дії адміністратора через поле adminId, яке посилається на користувача з відповідною роллю.

Використання MongoDB у даному вебзастосунку дозволило реалізувати гнучку та масштабовану структуру зберігання даних. Завдяки документно-орієнтованому підходу було спрощено процес зберігання зв’язаних даних (через ObjectId), що забезпечує просту та ефективну взаємодію з колекціями Booking, User, Workspace, Location. Такий підхід ідеально підходить для динамічного середовища, де облік бронювань та управління просторами мають бути максимально швидкими та надійними.

2.5 Реалізація логіки веб-застосунку

Під час розробки інформаційної системи бронювання робочих місць у коворкінгах було застосовано компонентно-орієнтований підхід (Component-Based Architecture), що передбачає модульну побудову застосунку із чітко розмежованими функціональними блоками [10]. Кожен компонент відповідає за окрему частину інтерфейсу користувача або логіки взаємодії з сервером, що забезпечує високу гнучкість, повторне використання коду та масштабованість системи.

Фронтенд реалізовано за допомогою бібліотеки React, яка дозволяє створювати реактивні інтерфейси користувача з використанням функціональних компонентів [19]. Основні компоненти, такі як Login, Register, Book, MyBookings, Locations тощо, реалізують ключові сценарії користувача: авторизацію, перегляд доступних робочих місць, бронювання та перегляд створених бронювань. Стан усередині компонентів керується за допомогою хуків useState та useEffect, що забезпечує динамічне оновлення інтерфейсу у відповідь на зміни даних.

Для навігації між екранами використано React Router, що дозволяє визначити маршрути (routes) між сторінками додатку, забезпечуючи логічну структуру переходів між компонентами без перезавантаження сторінки [16].

Комунікація клієнта з сервером реалізується через окремий модуль api.js (див. додаток А), у якому визначено функції для HTTP-запитів (наприклад, loginUser, createBooking, getWorkspaces, getAvailableSlots, getUserBookings). Це дозволяє централізувати логіку взаємодії з бекендом, підвищуючи зручність та підтримуваність коду.

Бекенд реалізовано за допомогою Node.js і фреймворку Express. Сервер надає RESTful API, через яке здійснюється обробка запитів клієнта, авторизація користувачів, управління робочими місцями, обробка бронювань тощо. Усі дані зберігаються у MongoDB – документно-орієнтованій базі даних, яка забезпечує гнучкість структури збереження завдяки використанню форматів JSON (BSON).

Моделі сутностей (User, Workspace, Booking) описані за допомогою бібліотеки Mongoose, яка надає зручний інтерфейс для взаємодії з MongoDB та реалізації схем із валідацією. Перед створенням нового бронювання система перевіряє, чи вже не зайнятий відповідний слот у певній локації на обрану дату та час. У випадку конфлікту повертається відповідь з кодом 409 (Conflict), що обробляється на клієнті з відповідним повідомленням.

Сторінка бронювання Book.jsx (див. додаток А), реалізовує перевірку доступних часових слотів у реальному часі. При виборі локації та дати

компонент динамічно завантажує вільні часові інтервали, надаючи користувачеві лише ті опції, які ще не були зарезервовані.

Для збереження стану користувача (ідентифікатора після входу) використовується localStorage, що дозволяє зберігати активну сесію між оновленнями сторінки.

Таким чином, використання Component-Based Architecture у поєднанні з MongoDB та сучасними технологіями фронтенд- і бекенд-розробки дозволило створити масштабовану, реактивну та зручну в користуванні інформаційну систему для бронювання місць у коворкінгах. Такий підхід сприяє ефективному управлінню даними, спрощує підтримку та відкриває перспективи для подальшого розвитку системи.

2.6 Висновки до другого розділу

У цьому розділі було детально розглянуто процес проєктування та реалізації інформаційної системи для бронювання робочих місць у коворкінгах. На основі аналізу сучасних підходів до архітектури веб-застосунків було обґрунтовано вибір компонентно-орієнтованої архітектури, яка забезпечує гнучке розділення функціональності на незалежні частини інтерфейсу. Такий підхід сприяє високій модульності, повторному використанню коду та спрощенню розширення проєкту.

У процесі проєктування були визначені основні ролі користувачів (звичайний користувач та адміністратор) та перелік функціональних вимог до системи. Відповідно до них сформовано діаграму варіантів використання, яка наочно демонструє взаємодію між користувачами та системою.

Користувацький інтерфейс було реалізовано за допомогою бібліотек React та TailwindCSS, що забезпечило створення адаптивного, сучасного і зручного для користувача дизайну. Основні екрани системи охоплюють авторизацію, реєстрацію, перегляд та фільтрацію доступних локацій, створення бронювань, керування бронюваннями, а також функціонал адміністрування.

У якості бази даних було обрано MongoDB, яка є документо-орієнтованою СУБД і дозволяє ефективно зберігати та обробляти дані у форматі JSON. Для взаємодії з базою даних реалізовано відповідні моделі сутностей (User, Workspace, Booking), які забезпечують виконання CRUD-операцій.

Загалом, у другому розділі було здійснено повний цикл реалізації інформаційної системи: від проєктування структури та вибору технологій до впровадження логіки користувацької взаємодії та забезпечення збереження даних. Отриманий результат є зручним у використанні, функціональним та придатним для подальшого масштабування.

РОЗДІЛ 3. ПРАКТИЧНА ЧАСТИНА. ДЕМОНСТРАЦІЯ, ТЕСТУВАННЯ ТА УЗАГАЛЬНЕННЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1 Демонстрація роботи інформаційної системи

Розроблений веб-застосунок для бронювання робочих місць у коворкінгах реалізовано з використанням фреймворку React для побудови клієнтської частини та серверного середовища Node.js з Express, що забезпечує обробку запитів [20]. База даних реалізована на основі MongoDB, яка зберігає інформацію про користувачів, локації, бронювання тощо.

Для демонстрації роботи застосунку було обрано веббраузер у середовищі розробника, що дозволяє емулювати різні розміри екранів і перевірити адаптивність інтерфейсу. Завдяки використанню бібліотеки TailwindCSS інтерфейс є сучасним, адаптивним і підтримує легку стилізацію компонентів.

Після запуску веб-застосунку користувач потрапляє на сторінку авторизації (див. додаток А). У разі відсутності облікового запису він має змогу перейти на сторінку реєстрації, де вводить ім’я, електронну пошту та пароль. Дані користувача зберігаються в базі даних MongoDB, а його сесія зберігається за допомогою JWT-токена у localStorage, що забезпечує подальшу аутентифікацію при навігації між сторінками [21]. На рисунку 3.1 подано приклад екранів авторизації та реєстрації, що ілюструють початковий етап взаємодії користувача з системою.

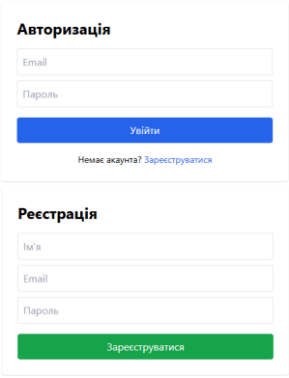

Рисунок 3.1 – Екрани авторизації та реєстрації

Після завершення авторизації або реєстрації користувач потрапляє на головний екран веб-застосунку. Центральний екран містить основну інформацію про доступні коворкінги в різних містах України. Головна мета цього екрану забезпечити зручний і візуально привабливий доступ до функції бронювання робочих місць. Користувач бачить:

1. Картки коворкінгів — кожна картка включає:

- назву коворкінгу;
- розташування (місто);
- кількість доступних місць;
- відповідне зображення локації для кращої візуалізації;
- кнопку «Забронювати зараз!», яка перенаправляє на форму бронювання.

2. Пошук — зверху розташоване поле пошуку, що дозволяє швидко фільтрувати коворкінги за назвою або місцем розташування.

3. Верхня навігація — дає змогу перейти до інших функціональних сторінок застосунку, таких як:

- список усіх коворкінгів;
- перегляд власних бронювань;
- екран налаштувань профілю користувача.

Такий підхід до побудови головного екрану забезпечує інтуїтивно зрозумілу навігацію, привабливу візуальну подачу даних та простоту здійснення бронювання. На рисунку 3.2 зображено головний екран.

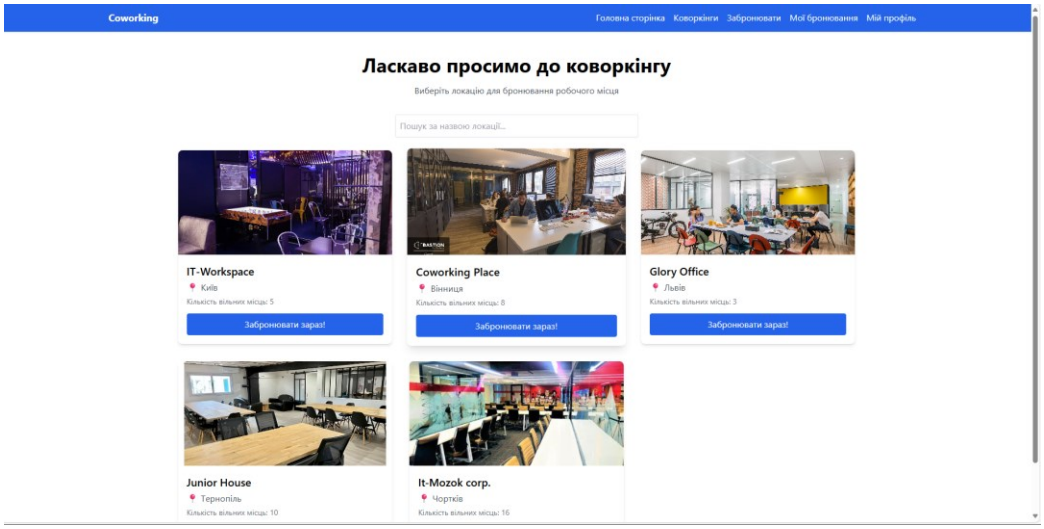

Рисунок 3.2 – Головний екран веб-застосунку

Після натискання на кнопку «Забронювати зараз!» користувач переходить на екран оформлення бронювання. Цей екран є ключовим компонентом інформаційної системи, що забезпечує процес резервування обраного робочого місця у вказаній локації. Інтерфейс побудований таким чином, щоб бути максимально простим та інтуїтивно зрозумілим.

У верхній частині екрана відображається інформація про обраний коворкінг, зокрема його назва, місто та актуальна кількість доступних місць на момент відкриття сторінки. Нижче розміщується елемент вибору дати, що реалізований у вигляді інтерактивного календаря. Користувач має змогу обрати лише ті дні, для яких ще доступні вільні місця.

Після вибору дати автоматично підвантажується список доступних часових слотів. Слоти, які вже були заброньовані іншими користувачами, автоматично приховуються з інтерфейсу, що унеможливлює помилковий вибір. Далі користувач може обрати кількість робочих місць, які бажає зарезервувати, а також зазначити тип робочого місця.

Після вибору необхідних параметрів користувач натискає кнопку «Підтвердити бронювання». У цей момент дані надсилаються на сервер, де виконується перевірка на наявність конфліктів. Якщо зазначені параметри не перетинаються з іншими активними бронюваннями, запис успішно зберігається

у базі даних, а користувач отримує повідомлення про успішне оформлення бронювання. У випадку виявлення конфлікту система повідомляє користувача про помилку та пропонує обрати інший час або дату.

Такий підхід дозволяє зробити процес бронювання зручним, швидким та надійним, а також забезпечує захист від дублювання записів, гарантуючи цілісність і послідовність у роботі системи. На рисунку 3.3 показано екран оформлення бронювання.

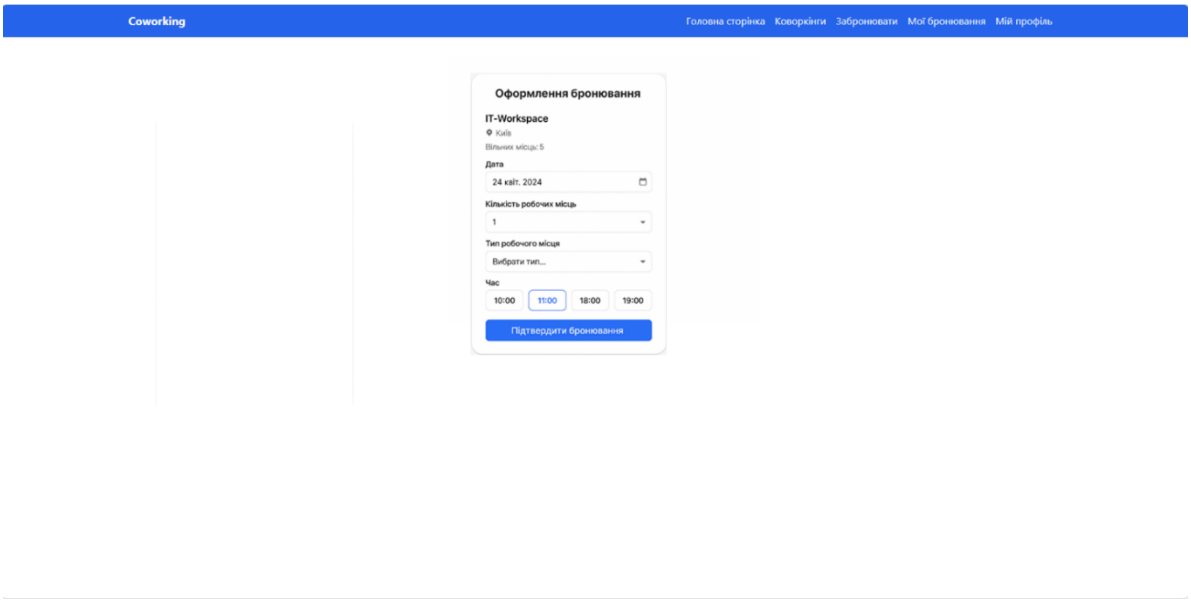

Рисунок 3.3 – Екран оформлення бронювання

Також, на рисунку 3.4 можемо побачити повідомлення про підтвердження бронювання.

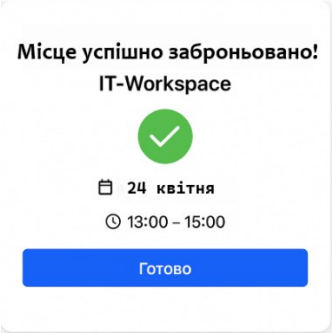

Рисунок 3.4 – Повідомлення підтвердження бронювання

Після оформлення бронювання користувач має змогу переглядати свої активні та минулі бронювання на окремому екрані «Мої бронювання». Цей екран реалізовано для забезпечення зручного доступу до всієї історії взаємодії користувача з системою. На рисунку 3.5 зображено екран бронювань.

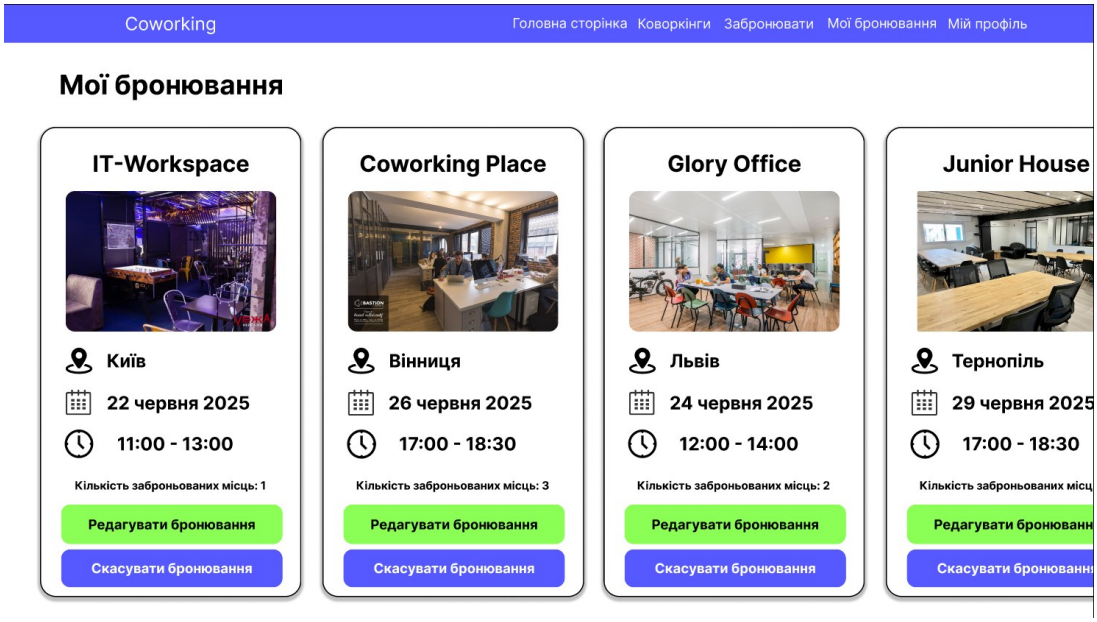

Рисунок 3.5 – Екран бронювань

У верхній частині екрана відображається заголовок, який чітко ідентифікує сторінку як розділ перегляду особистих бронювань. Нижче розміщується список усіх створених користувачем бронювань, які витягуються з бази даних шляхом фільтрації за його унікальним ідентифікатором. Кожен запис містить інформацію про назву коворкінгу, його розташування, дату, час та кількість заброньованих місць.

Для зручності сприйняття кожне бронювання візуально оформлено у вигляді окремої карточки або блоку з чітко структурованою інформацією. У разі потреби, біля кожного активного бронювання розміщено кнопку «Скасувати бронювання», що дозволяє користувачу видалити його. При натисканні на цю кнопку надсилається запит до серверу, після чого відповідний запис видаляється з бази даних.

Крім того, реалізовано механізм автоматичного оновлення списку бронювань після здійснення змін, що забезпечує динамічний та актуальний інтерфейс користувача. У випадку відсутності жодного активного бронювання, на екрані з'являється відповідне інформативне повідомлення.

Наступним важливим елементом є профіль користувача інформаційної системи бронювання робочих місць у коворкінгах, який забезпечує персоналізовану взаємодію із системою. У межах функціоналу профілю реалізовано можливість перегляду та редагування особистих даних, таких як ім’я користувача та адреса електронної пошти. Така функціональність дозволяє підтримувати актуальність інформації про користувача, що є необхідним для коректного відображення даних у процесі бронювання. Користувач також має змогу змінити пароль для доступу до свого облікового запису, що підвищує рівень безпеки системи. Окрім того, з профілю можна вийти із системи натисканням відповідної кнопки, що призводить до очищення збереженої сесії та повернення на екран авторизації (див. додаток Б). На рисунку 3.6 зображено екран профілю користувача.

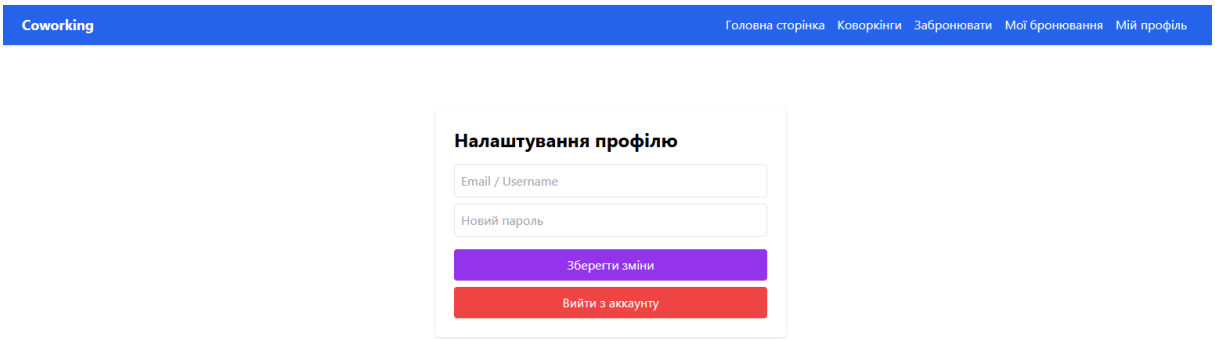

Рисунок 3.6 – Екран профілю користувача

Адміністративна панель є окремим модулем інформаційної системи, доступ до якого мають лише користувачі з відповідними правами (адміністратори). Її основне призначення — управління ключовими об’єктами

системи: робочими місцями, користувачами та бронюваннями. Таким чином, адмін-панель виконує функції контролю, модерації та підтримки актуальності даних у системі (див. додаток А). На рисунку 3.7 зображено екран адміністративної панелі.

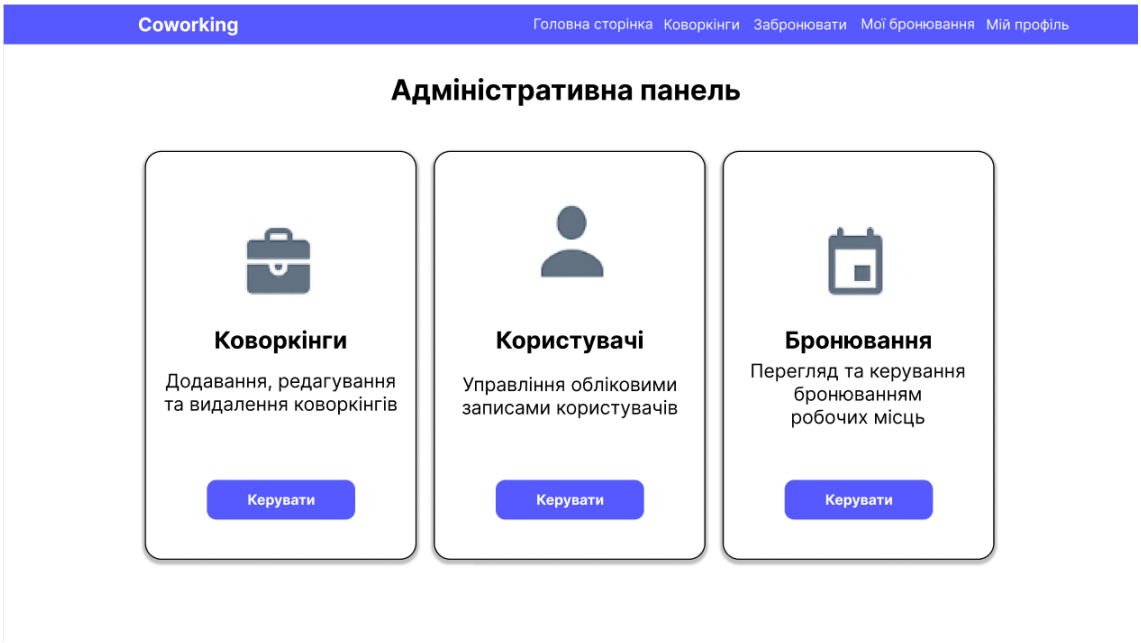

Рисунок 3.7 – Екран адміністративної панелі

Демонстрація роботи веб-застосунку в браузері засвідчила стабільність функціонування всіх компонентів, швидке реагування інтерфейсу та коректне збереження і обробку даних на стороні сервера.

3.2 Тестування та валідація інформаційної системи

Після завершення етапів розробки веб-застосунку та огляду функціональності було проведено ручне тестування з метою перевірки правильності реалізації основного функціоналу, виявлення можливих помилок, а також оцінки зручності користувацького інтерфейсу [22]. Тестування здійснювалося без використання спеціалізованих інструментів автоматизації, шляхом проходження типових сценаріїв взаємодії з веб-застосунком через браузер [23]. Кожен із функціональних модулів був перевірений на відповідність

вимогам, зазначеним у технічному завданні. Результати тестування наведено в таблиці 3.1.

Таблиця 3.1 – Результати ручного тестування веб-застосунку

№	Функція	Очікувана поведінка	Фактичний результат	Статус	Коментар
1	Реєстрація	Користувач створює обліковий запис	Обліковий запис створено успішно	Успішно	
2	Авторизація	Вхід із коректними даними	Вхід виконано	Успішно	
3	Перегляд доступних локацій	Відображається список доступних коворкінгів	Список локацій відображено	Виправлено	Усі локації відображались з однаковим зображенням
4	Бронювання місця	Користувач обирає локацію, дату та час	Бронювання створено	Успішно	
5	Перевірка доступності слота	Недоступні слоти не мають з’являтись у списку вибору	Недоступні слоти не відображаються	Виправлено	Бекенд повертав помилку 400 через неправильні параметри запиту
6	Перегляд власних бронювань	Виводиться список бронювань користувача	Бронювання відображаються	Успішно	
7	Скасування бронювання	Користувач може видалити бронювання	Бронювання видалено	Успішно	
8	Профіль користувача	Відображається ім’я, email, можливість вийти з акаунта	Дані профілю коректно завантажено	Успішно	
9	Адмін-панель	Адмін може переглядати, редагувати та видаляти список локацій і користувачів	Інтерфейс відображає всі потрібні дані	Успішно	Доступ лише після автентифікації адміністратора

Під час тестування було виявлено декілька помилок:

- Помилка при перевірці доступності слотів: бекенд повертав помилку 400 через неправильні параметри запиту (workspaceId і date не передавались). Було оновлено логіку формування запиту та перевірки параметрів на бекенді.

- Помилка при перегляді доступних локацій: усі локації відображались з однаковим зображенням. Було додано в модель Workspace поле images, оновили бекенд і фронтенд для підтримки унікальних зображень.

Ці ситуації підтверджують важливість ручного тестування навіть для проєктів з відносно невеликим обсягом функціоналу. Усі ці виявлені проблеми були проаналізовані та усунуті.

3.3 Висновки до третього розділу

У третьому розділі було здійснено огляд реалізованої функціональності вебзастосунку для бронювання робочих місць у коворкінгах та проведено ручне тестування його основних модулів. Детально описано послідовність взаємодії користувача із системою — від реєстрації та входу до оформлення бронювання й перегляду особистого профілю.

Результати тестування підтвердили працездатність ключового функціоналу, зокрема: створення облікового запису, додавання та перегляд бронювань, вибір локації та часових слотів, формування аналітики. Під час перевірки було виявлено низку технічних недоліків, які були усунуті в процесі вдосконалення системи.

Проведене тестування дозволило переконатися у відповідності веб-застосунку функціональним вимогам, а також у стабільності та коректності його роботи в межах типової моделі використання.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при ураженні електричним струмом

У процесі розробки інформаційних систем, особливо на етапах, що передбачають роботу з апаратною частиною або обслуговування серверного обладнання, неминуче виникає взаємодія з різними електричними приладами та установками. У зв’язку з цим актуальним постає питання дотримання правил електробезпеки, зокрема надання першої допомоги при ураженні електричним струмом. Відсутність належної обізнаності та навичок у цій сфері може спричинити тяжкі наслідки як для розробників, так і для обслуговуючого персоналу.

Однією з найбільш небезпечних ситуацій є дотик до струмопровідних частин обладнання, що перебуває під напругою. У більшості випадків такий контакт призводить до рефлекторного скорочення м’язів, внаслідок чого потерпілий втрачає можливість самостійно відірватися від джерела струму. Це ускладнює надання допомоги та вимагає негайного втручання інших осіб з дотриманням усіх правил безпеки [24].

Відповідно до Правил безпечної експлуатації електроустановок споживачів, затверджених наказом Міністерства енергетики та електрифікації України від 09.01.1998 № 4 (НПАОП 40.1-1.21-98), під час виконання робіт, пов’язаних з експлуатацією електрообладнання, працівники повинні дотримуватися вимог електробезпеки та вміти надавати першу допомогу у разі ураження електричним струмом [25]. Ці вимоги є обов’язковими для всіх працівників, які під час своєї діяльності взаємодіють з електричними установками напругою до 1000 В. Вкрай важливо пам’ятати про техніку безпеки під час рятування потерпілого: діяти слід однією рукою, уникаючи одночасного дотику до металевих частин іншими частинами тіла або другою рукою. Для уникнення ураження рятувальника самим електричним струмом доцільно

використовувати діелектричні підстилки або стати на суху дерев'яну поверхню [28].

Залежно від ступеня ураження електричним струмом, виділяють три можливі стани потерпілого, що потребують різного підходу до надання першої допомоги:

- Перший стан – потерпілий перебуває у свідомості, скаржиться на загальну слабкість, можливі серцебиття або головний біль. У такому випадку необхідно забезпечити людині повний фізичний та психологічний спокій, спостерігати за її станом упродовж щонайменше 2–3 годин, а також викликати медичного працівника для огляду.
- Другий стан – потерпілий перебуває без свідомості, однак зберігається самостійне дихання. Постраждалого слід покласти в горизонтальне положення, розстебнути комір та пасок для полегшення дихання, забезпечити приплив свіжого повітря, дати понюхати нашатирний спирт або оббризкати обличчя холодною водою. Негайно слід викликати швидку медичну допомогу.
- Третій стан – потерпілий не дихає або дихання спостерігається з перервами, уривчасто, як у процесі агонії. У такому випадку необхідно негайно розпочати проведення серцево-легеневої реанімації: штучне дихання за методом «рот у рот» або «рот у ніс» у поєднанні з непрямим масажем серця до моменту відновлення самостійного дихання або прибуття медиків.

Таким чином, надання долікарської допомоги при ураженні електричним струмом є критично важливим під час виконання завдань з проєктування, розробки та супроводу інформаційних систем. Це дозволяє мінімізувати ризики нещасних випадків та забезпечити збереження життя і здоров'я працівників, залучених до технічної реалізації проєкту.

4.2 Організація праці на робочому місці

Відповідно до положень ДСТУ 8604:2015 «Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи», організація праці на робочому

місці є невід’ємною складовою ефективної та безпечної трудової діяльності [26]. Стандарт визначає вимоги до ергономічного проектування робочого простору, що забезпечує комфорт працівника, знижує рівень фізичного і психоемоційного навантаження, а також сприяє підвищенню продуктивності [32].

Організація праці охоплює раціональне розміщення обладнання, інструментів і засобів керування в межах досяжності працівника, а також врахування фізіологічних параметрів користувача під час розробки робочого місця. У ДСТУ 8604:2015 передбачено основні вимоги до робочої пози, зокрема при сидячій роботі, яка є типовою для фахівців у сфері інформаційних технологій. Така поза забезпечує стабільність положення тіла, зменшує загальну стомлюваність і дозволяє здійснювати точні рухи руками.

Таким чином, дотримання положень національного стандарту ДСТУ 8604:2015 при організації робочого місця є важливою умовою забезпечення ергономічної безпеки, ефективності праці та збереження здоров’я працівників, що особливо актуально в умовах розробки програмного забезпечення та роботи з комп’ютерною технікою.

Робоче місце визначається як простір, організований і технічно обладнаний для виконання певного виду трудової діяльності однією особою або колективом. Зазвичай воно включає меблі, технічні засоби, інструменти, матеріали, а також програмне забезпечення, необхідне для виконання поставлених завдань [27].

Розумне планування робочого простору передбачає:

- вибір адекватної робочої пози залежно від характеру діяльності;
- організацію системи робочих рухів для зменшення зайвих зусиль;
- визначення розмірів і меж робочої зони;
- раціональне розміщення засобів керування, пристроїв, матеріалів, документації та інструментів таким чином, щоб забезпечити зручність, безпечність та ефективність роботи;

- визначення оптимального співвідношення часу активної праці та періодів відпочинку, що дозволяє знизити рівень фізичного та психоемоційного навантаження.

У випадку розробки програмного забезпечення, зокрема інформаційної системи, основна частина робочого часу проводиться в положенні сидячи перед екраном комп’ютера. Робоча поза сидячи є найбільш поширеною для фахівців ІТ-сфери, оскільки вона забезпечує відносно низький рівень статичної втоми, стабільне положення тулуба та рук, що сприяє точному виконанню дрібних моторних дій, зокрема – набору тексту, роботи з клавіатурою, мишею, сенсорною панеллю чи графічним планшетом.

Проте тривале перебування у сидячому положенні може призводити до низки негативних наслідків для здоров’я, таких як порушення постави, зниження кровообігу, підвищене навантаження на хребет. Тому важливо враховувати ергономічні вимоги до організації робочого місця, зокрема: використовувати стілець з регульованою висотою і підтримкою попереку, встановлювати монітор на рівні очей, дотримуватись правильної відстані до екрана (приблизно 50–70 см), а також робити регулярні перерви для фізичної розрядки й профілактики зорового напруження [26].

Застосування таких підходів до організації праці дозволяє підвищити продуктивність праці, зменшити ризики професійного вигорання та хронічної втоми, а також забезпечити комфортні та безпечні умови для здійснення інтелектуальної діяльності, пов’язаної з розробкою програмного забезпечення.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було реалізовано повноцінний цикл створення інформаційної системи бронювання робочих місць у коворкінгах. На етапі аналізу предметної області досліджено існуючі рішення у сфері онлайн-бронювання, визначено їх сильні та слабкі сторони, що дозволило сформулювати функціональні та нефункціональні вимоги до розроблюваного веб-застосунку. Обґрунтовано вибір компонентно-орієнтованої архітектури та технологій реалізації, зокрема використання Node.js та Express для серверної частини, MongoDB для зберігання даних, а також React і TailwindCSS для побудови сучасного адаптивного інтерфейсу.

У процесі проєктування системи створено модель бази даних, реалізовано основні сутності, маршрути та API-запити для авторизації, перегляду та створення бронювань, керування коворкінгами й обліку користувачів. Інтерфейс побудовано з урахуванням зручності для кінцевого користувача та забезпечення інтуїтивної навігації між екранами.

У третьому розділі проведено демонстрацію роботи веб-застосунку та ручне тестування основного функціоналу, включаючи авторизацію, бронювання місць, перегляд активних бронювань, роботу з календарем та фільтрацією. У результаті тестування було виявлено та виправлено ряд недоліків, що позитивно вплинуло на стабільність системи.

Загалом, розроблена система відповідає заявленим вимогам, є надійною, функціонально повною та зручною у використанні. Вона має потенціал для подальшого розширення — зокрема, інтеграції платіжних систем, системи повідомлень, адміністрування бронювань, аналітики використання тощо. Отримані результати свідчать про доцільність і ефективність обраних технічних рішень при проєктуванні сучасних інформаційних систем для сфери коворкінг-послуг.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Програма онлайн-бронювання та CRM управління бізнесом EasyWeek [Електронний ресурс]. Режим доступу до ресурсу: https://easyweek.com.ua

2. Коворкінг Carpe Diem Coworking [Електронний ресурс]. Режим доступу до ресурсу: https://coworking.carpediem.agency

3. ITEAHub – найсучашніший коворкінг у Львові [Електронний ресурс]. Режим доступу до ресурсу: https://iteahub.com

4. JavaScript [Електронний ресурс]. Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JavaScript

5. Фреймворк React [Електронний ресурс]. Режим доступу до ресурсу: https://react.dev

6. Марусенко І. М. Веб-програмування: HTML, CSS, JavaScript, PHP. Навч.посіб. / – Тернопіль: ТНТУ, 2023. – 256 с.

7. TypeScript [Електронний ресурс]. Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/TypeScript

8. Visual Studio Code [Електронний ресурс]. Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Visual_Studio_Code

9. Маслов В. І. Основи розробки інформаційних систем. Навч.посіб. / – К.: Ліра-К, 2020. – 296 с.

10. Архітектурна модель Component-Based [Електронний ресурс]. Режим доступу до ресурсу: https://medium.com/clean-code-channel/component-based-architecture-2cb38ea1b247

11. Що таке REST API? [Електронний ресурс]. Режим доступу до ресурсу: https://foxminded.ua/shcho-take-rest-api/

12. Express.js [Електронний ресурс]. Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Express.js

13. Діаграма прецедентів [Електронний ресурс]. Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Діаграма_прецедентів

14. Draw.io [Електронний ресурс]. Режим доступу до ресурсу: https://www.drawio.com

15. User interface design and usability in information systems [Електронний ресурс]. Режим доступу до ресурсу: https://www.researchgate.net/publication/383578205_User_Interface_Design_and_Usability_in_Information_Systems

16. Основи React Router [Електронний ресурс]. Режим доступу до ресурсу: http://fpm.dnu.dp.ua/2019/12/04/react-router/

17. Основи MongoDB [Електронний ресурс]. Режим доступу до ресурсу: https://devzone.org.ua/post/osnovy-mongodb

18. Mermaid Live Editor [Електронний ресурс]. Режим доступу до ресурсу: https://mermaid.live/

19. React [Електронний ресурс]. Режим доступу до ресурсу: https://uk.legacy.reactjs.org/

20. Аспекти використання Node.js та MongoDB для створення веб-застосунків [Електронний ресурс]. Режим доступу до ресурсу: http://infotech-soccult.knukim.edu.ua/article/view/283958

21. JWT-токен [Електронний ресурс]. Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/JSON_Web_Token

22. В чому різниця між автоматизованим і ручним тестуванням? [Електронний ресурс]. Режим доступу до ресурсу: https://university.sigma.software/manual-testing-vs-automation-testing/

23. Тестування програмного забезпечення. [Електронний ресурс]. Режим доступу до ресурсу: https://eprints.cdu.edu.ua/1482/1/testyvan.pdf

24. Про затвердження Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці (НПАОП 40.1-1.21-98) [Електронний ресурс]. Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0231-05#Text

25. Долікарська допомога при ураженні електричним струмом [Електронний ресурс]. Режим доступу до ресурсу: https://dl.tntu.edu.ua/content.php?cid=299863

26. Ергономіка робочого місця [Електронний ресурс]. Режим доступу до ресурсу: https://medicasano.com.ua/blog/erhonomika-robochoho-mistsia-chomu-vona-vazhlyva-dlia-zdorov-ia-khrebta-ta-suhlobiv/

27. Організація праці на робочому місці [Електронний ресурс]. Режим доступу до ресурсу: https://cpo.stu.cn.ua/Oksana/posibnik/1000.html?utm_source=chatgpt.com

28. Зеркалов Д. В. Безпека життєдіяльності. Навч.посіб. / – Київ: Основа, 2016. – 267 с. – ISBN 978-966-699-866-1.

29. Мобільна доступність і перевага онлайн-сервісів [Електронний ресурс]. Режим доступу до ресурсу: https://sennalabs.com/blog/web-application-development-for-enhanced-user-experience

30. Етапи розробки веб-застосунку [Електронний ресурс]. Режим доступу до ресурсу: https://terentevdesignstudio.com/blog/etapi-rozrobki-veb-dodatkiv/

31. Липова О. М., Люта М. В. Особливості розробки інтерфейсу веб додатків. Сучасні електромеханічні та інформаційні системи. 2021. С. 112–117. [Електронний ресурс]. Режим доступу до ресурсу: https://er.knutd.edu.ua/handle/123456789/19938

32. ДСТУ EN 8604:2015 Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. На заміну ДСТУ EN 8604:2015; чинний від 2022–08–01. Вид. офіц. Київ : УкрНДНЦ, 2022. 126 с. [Електронний ресурс]. Режим доступу до ресурсу: https://zakon.isu.net.ua/sites/default/files/normdocs/dstu_en_623053_2021_bliskavko zakhist._chastina_3._fizichni_p.pdf

33. Leshchyshyn, Y., Scherbak, L., Nazarevych, O., Gotovych, V., Tymkiv, P., & Shymchuk, G. (2019, May). Multicomponent Model of the Heart Rate Variability Change-point. In 2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 110-113). IEEE.

34. Lytvynenko, I., Lupenko, S., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, September). Mathematical model of gas consumption process in

the form of cyclic random process. In 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 1, pp. 232-235). IEEE.

35. Kozlovskyi, V., Balanyuk, Y., Martyniuk, H., Nazarevych, O., Scherbak, L., & Shymchuk, G. (2022, April). Information Technology for Estimating City Gas Consumption During the Year. In 2022 International Conference on Smart Information Systems and Technologies (SIST) (pp. 1-4). IEEE.

36. Lytvynenko, I., Lupenko, S., Kunanets, N., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, November). Simulation of gas consumption process based on the mathematical model in the form of cyclic random process considering the scale factors. In 1st International Workshop on Information Technologies: Theoretical and Applied Problems, ITTAP 2021.

37. Kunanets, N., Pasichnyk, V., Bodnarchuk, I., Martsenko, S., Matsiuk, O., Matsiuk, A., ... & Shymchuk, H. (2019). Information system for visual analyzer disease diagnostics. In CEUR Workshop Proceedings (pp. 43-56).

38. Lupenko, S., Lytvynenko, I., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, December). Approach to gas consumption process forecasting on the basis of a mathematical model in the form of a random cyclic process. In Proceedings of the International Conference „Advanced applied energy and information technologies 2021", 2021 (pp. 213-219). TNTU, Zhytomyr «Publishing house „Book-Druk "» LLC.

39. Lytvynenko, I., Lupenko, S., Nazarevych, O., Shymchuk, H., & Hotovych, V. (2022). Additive mathematical model of gas consumption process. Вісник Тернопільського національного технічного університету, 104(4), 87-97.

40. ШИМЧУК, Г., ШЕВЧЕНКО, Н., ШВИРЛО, К., & ГАРМАТЮК, Н. (2025). СИСТЕМА ВІДНОВЛЕННЯ ДАНИХ У БЕЗДРОТОВИХ СЕНСОРНИХ МЕРЕЖАХ НА ОСНОВІ МАШИННОГО НАВЧАННЯ. Herald of Khmelnytskyi National University. Technical sciences, 353(3.2), 246-250.

ДОДАТКИ

Додаток А

Код пов’язаний з логікою веб-застосунку

Лістинг модуля api.js

```
import axios from "axios";
const API = "http://localhost:5000/api";
const BASE_URL = "http://localhost:5000/api";
const API_BASE = "http://localhost:5000/api";

export async function registerUser(userData) {
  const res = await fetch(${BASE_URL}/register, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(userData),
  });
  const data = await res.json();
  if (!res.ok) {
    throw new Error(data?.error || "Failed to register");
  }
  return data;
}


export const loginUser = async (email, password) => {
  const res = await fetch(${API}/login, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ email, password }),
  });

  if (!res.ok) {
    throw new Error("Login failed");
  }

  return await res.json();
};


export const getWorkspaces = async () => {
  return axios.get("http://localhost:5000/api/workspaces");
};

export const createBooking = async (bookingData) => {
  const res = await fetch(${API}/bookings, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(bookingData),
  });
```

```
  return await res.json();
};

export const getUserBookings = (userId) => {
  return axios.get(${BASE_URL}/bookings/${userId});
};

export const deleteBooking = (id) => axios.delete(${API}/bookings/${id});

export const getAvailableSlots = (workspaceId, date) => {
  return axios.get(/api/bookings/slots, {
    params: {
      workspaceId,
      date
    }
  });
};
```

Лістинг модуля бронювання (book.jsx)

```
import React, { useEffect, useState } from "react";
import { createBooking, getWorkspaces, getAvailableSlots } from
"../api.js";

export default function Book() {
  const [workspaces, setWorkspaces] = useState([]);
  const [workspaceId, setWorkspaceId] = useState("");
  const [date, setDate] = useState("");
  const [timeSlot, setTimeSlot] = useState("");
  const [availableSlots, setAvailableSlots] = useState([]);
  const userId = localStorage.getItem("userId");

  // Завантаження робочих просторів
  useEffect(() => {
    const fetchWorkspaces = async () => {
      try {
        const res = await getWorkspaces();
        setWorkspaces(res.data || []);
      } catch (err) {
        console.error("Failed to fetch workspaces", err);
        setWorkspaces([]);
      }
    };

    fetchWorkspaces();
  }, []);

  // Отримання доступних слотів
  useEffect(() => {
    const fetchSlots = async () => {
      if (workspaceId && date) {
        try {
          const res = await getAvailableSlots(workspaceId, date);
          setAvailableSlots(res.data || []);
        } catch (err) {
```

```
          console.error("Failed to fetch slots", err);
          setAvailableSlots([]);
        }
      } else {
        setAvailableSlots([]);
      }
    };

    fetchSlots();
  }, [workspaceId, date]);

  // Обробник бронювання
  const handleBooking = async () => {
    if (!workspaceId  !date  !timeSlot) {
      alert("Please fill in all fields");
      return;
    }

    try {
      await createBooking({ userId, workspaceId, date, timeSlot });
      alert("Booking successful");
    } catch (err) {
      console.error("Booking failed", err);
      if (err.response?.status === 409) {
        alert("Це місце вже заброньоване на вибраний час.");
      } else {
        alert("Помилка при бронюванні.");
      }
    }
  };

  return (
    <div className="max-w-md mx-auto mt-10 bg-white p-6 rounded
shadow">
      <h2 className="text-2xl font-bold mb-4">Забронювати робоче
місце</h2>

      {Array.isArray(workspaces) && workspaces.length === 0 ? (
        <div className="text-red-600 font-semibold">
          Немає доступних коворкінгів, напишіть адміністратору.
        </div>
      ) : (
        <>
          <label className="block mb-1 font-semibold">Виберіть
локацію</label>
          <select
            value={workspaceId}
            onChange={(e) => setWorkspaceId(e.target.value)}
            className="w-full border p-2 mb-4 rounded"
          >
            <option value="">Виберіть коворкінг</option>
            {workspaces.map((w) => (
              <option key={w._id} value={w._id}>
                {w.name} — {w.location}
              </option>
            ))}
          </select>
```

```
            <label className="block mb-1 font-semibold">Виберіть
    дату</label>
            <input
              type="date"
              value={date}
              onChange={(e) => setDate(e.target.value)}
              className="w-full border p-2 mb-4 rounded"
            />

            <label className="block mb-1 font-semibold">Select
    Time</label>
            <select
              value={timeSlot}
              onChange={(e) => setTimeSlot(e.target.value)}
              className="w-full border p-2 mb-4 rounded"
            >
              <option value="">--:--</option>
              {availableSlots.length > 0 ? (
                availableSlots.map((slot, index) => (
                  <option key={index} value={slot}>
                    {slot}
                  </option>
                ))
              ) : (
                <option disabled>Немає вільних місць!</option>
              )}
            </select>

            <button
              onClick={handleBooking}
              className="w-full bg-blue-600 text-white py-2 rounded
    hover:bg-blue-700"
            >
              Book
            </button>
          </>
        )}
      </div>
    );
}
```

Лістинг модуля авторизації (Login.jsx)

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import { loginUser } from "../api.js";
export default function Login() {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const navigate = useNavigate();
  const handleLogin = async () => {
    if (!email || !password) {
      alert("Please enter both email and password");
      return;
    }
    try {
      const user = await loginUser(email, password);
```

```
        localStorage.setItem("userId", user._id);
        navigate("/");
      } catch (err) {
        console.error(err);
        alert("Login failed");
      }
    };
    return (
      <div className="max-w-md mx-auto mt-20 bg-white p-6 rounded
shadow">
        <h2 className="text-2xl font-bold mb-4">Авторизація</h2>
        <input
          type="email"
          placeholder="Email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          className="w-full border p-2 mb-2 rounded"
        />
        <input
          type="password"
          placeholder="Пароль"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          className="w-full border p-2 mb-4 rounded"
        />
        <button
          onClick={handleLogin}
          className="w-full bg-blue-600 text-white py-2 rounded
hover:bg-blue-700"
        >
          Увійти
        </button>
        {/* Блок для переходу на реєстрацію */}
        <div className="mt-4 text-center text-sm">
          Немає акаунта?{" "}
          <button
            onClick={() => navigate("/register")}
            className="text-blue-600 hover:underline"
          >
            Зареєструватися
          </button>
        </div>
      </div>
    );
}
```

Лістинг модуля реєстрації (Register.jsx)

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import { registerUser } from "../api.js";
export default function Register() {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const navigate = useNavigate();
  const handleRegister = async () => {
```

```
    if (!name  !email  !password) {
      alert("Please fill in all fields");
      return;
    }
    try {
      const user = await registerUser({ name, email, password });
      if (user && user._id) {
        localStorage.setItem("userId", user._id);
        navigate("/");
      } else {
        alert("Invalid response from server");
      }
    } catch (err) {
      console.error(err);
      alert("Registration failed");
    }
  };
  return (
    <div className="max-w-md mx-auto mt-20 bg-white p-6 rounded
shadow">
      <h2 className="text-2xl font-bold mb-4">Реєстрація</h2>
      <input
        type="text"
        placeholder="Ім'я"
        value={name}
        onChange={(e) => setName(e.target.value)}
        className="w-full border p-2 mb-2 rounded"
      />
      <input
        type="email"
        placeholder="Email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        className="w-full border p-2 mb-2 rounded"
      />
      <input
        type="password"
        placeholder="Пароль"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        className="w-full border p-2 mb-4 rounded"
      />
      <button
        onClick={handleRegister}
        className="w-full bg-green-600 text-white py-2 rounded
hover:bg-green-700"
      >
        Зареєструватися
      </button>
    </div>
  );
```

Лістинг модуля налаштування профілю (Profile.jsx)

```
import React, { useState, useEffect } from "react";
```

```
export default function Profile() {
  const [name, setName] = useState("");
  const [password, setPassword] = useState("");

  useEffect(() => {
    // Симуляція: отримання email з localStorage
    const storedEmail = localStorage.getItem("userId");
    setName(storedEmail || "");
  }, []);

  const handleSave = () => {
    alert("Changes saved (mock only).");
  };

  const handleLogout = () => {
    localStorage.removeItem("userId");
    window.location.href = "/login";
  };

  return (
    <div className="max-w-md mx-auto mt-20 bg-white p-6 rounded
shadow">
      <h2 className="text-2xl font-bold mb-4">Налаштування
профілю</h2>
      <input
        type="text"
        placeholder="Email / Username"
        value={name}
        onChange={(e) => setName(e.target.value)}
        className="w-full border p-2 mb-2 rounded"
      />
      <input
        type="password"
        placeholder="Новий пароль"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        className="w-full border p-2 mb-4 rounded"
      />
      <button
        onClick={handleSave}
        className="w-full bg-purple-600 text-white py-2 rounded
hover:bg-purple-700 mb-2"
      >
       Зберегти зміни
      </button>
      <button
        onClick={handleLogout}
        className="w-full bg-red-500 text-white py-2 rounded
hover:bg-red-600"
      >
        Вийти з аккаунту
      </button>
    </div>
  );
}
```

Лістинг модулю адміністративної панелі (Admin.jsx)

```
import React, { useState } from "react";
import axios from "axios";

export default function Admin() {
  const [name, setName] = useState("");
  const [location, setLocation] = useState("");
  const [capacity, setCapacity] = useState(10);
  const [amenities, setAmenities] = useState("");
  const [images, setImages] = useState("");


  const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const workspace = {
        name,
        location,
        capacity: parseInt(capacity),
        amenities: amenities.split(",").map((item) => item.trim()),
      images: images.split(',').map(i => i.trim())

      };

      await axios.post("http://localhost:5000/api/workspaces",
workspace);
      alert("Робоча локація була створена успішно!");
      setName("");
      setLocation("");
      setCapacity(10);
      setAmenities("");
    } catch (err) {
      console.error(err);
      alert("Failed to create workspace");
    }
  };

  return (
    <div className="max-w-md mx-auto mt-10 p-6 bg-white shadow-md
rounded">
      <h2 className="text-2xl font-bold mb-4">Добавити новий
коворкінг</h2>
      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Назва коворкінгу"
          value={name}
          onChange={(e) => setName(e.target.value)}
          className="w-full mb-2 p-2 border rounded"
        />
        <input
          type="text"
          placeholder="Локація"
          value={location}
          onChange={(e) => setLocation(e.target.value)}
          className="w-full mb-2 p-2 border rounded"
```

```
        />
        <input
          type="number"
          placeholder="Кількість робочих місць"
          value={capacity}
          onChange={(e) => setCapacity(e.target.value)}
          className="w-full mb-2 p-2 border rounded"
        />
        <input
          type="text"
          placeholder="Зручності (через кому)"
          value={amenities}
          onChange={(e) => setAmenities(e.target.value)}
          className="w-full mb-4 p-2 border rounded"
        />
        <input
  type="text"
  placeholder="Посилання на зображення"
  value={images}
  onChange={(e) => setImages(e.target.value)}
  className="w-full border p-2 mb-2 rounded"
/>

        <button
          type="submit"
          className="w-full bg-blue-600 text-white py-2 rounded
hover:bg-blue-700"
        >
          Створити новий коворкінг
        </button>
      </form>
    </div>
  );
}
```

Лістинг модуля головної сторінки (Home.jsx)

```
import React, { useEffect, useState } from "react";
import { getWorkspaces } from "../api";
import { useNavigate } from "react-router-dom";
import { motion } from "framer-motion";

export default function Home() {
  const [workspaces, setWorkspaces] = useState([]);
  const [search, setSearch] = useState("");
  const navigate = useNavigate();

  useEffect(() => {
    const fetchWorkspaces = async () => {
      try {
        const res = await getWorkspaces();
        setWorkspaces(res.data || []);
      } catch (err) {
        console.error("Не вдалось загрузити робочі місця, зверність
до адміністратора.", err);
      }
    };
```

```
    fetchWorkspaces();
  }, []);

  const filteredWorkspaces = workspaces.filter((w) =>
    w.name.toLowerCase().includes(search.toLowerCase()) ||
    w.location.toLowerCase().includes(search.toLowerCase())
  );

  return (
    <div className="max-w-7xl mx-auto px-4 py-10">
      <motion.h1
        className="text-4xl font-bold text-center mb-4"
        initial={{ opacity: 0, y: -30 }}
        animate={{ opacity: 1, y: 0 }}
        transition={{ duration: 0.6 }}
      >
        Ласкаво просимо до коворкінгу
      </motion.h1>

      <motion.p
        className="text-center text-gray-600 mb-8"
        initial={{ opacity: 0 }}
        animate={{ opacity: 1 }}
        transition={{ delay: 0.3, duration: 0.6 }}
      >
        Виберіть локацію для бронювання робочого місця
      </motion.p>

      <div className="mb-6 max-w-md mx-auto">
        <input
          type="text"
          placeholder="Пошук за назвою локації..."
          value={search}
          onChange={(e) => setSearch(e.target.value)}
          className="w-full border p-2 rounded shadow-sm"
        />
      </div>

      <motion.div
        className="grid grid-cols-1 sm:grid-cols-2 lg:grid-cols-3
gap-8"
        initial="hidden"
        animate="visible"
        variants={{
          visible: {
            transition: {
              staggerChildren: 0.1,
            },
          },
        }}
      >
        {filteredWorkspaces.map((w) => (
          <motion.div
            key={w._id}
            className="bg-white rounded-lg shadow-md overflow-hidden
hover:shadow-lg transition-shadow duration-300"
            whileHover={{ scale: 1.02 }}
```

```
            variants={{
              hidden: { opacity: 0, y: 20 },
              visible: { opacity: 1, y: 0 },
            }}
          >
            <img
  src={
    w.image
      ? w.image
      : /images/${w.location.toLowerCase().replace(/\s/g, "")}.jpg
  }
  alt={w.name}
  className="w-full h-48 object-cover"
/>

            <div className="p-4">
              <h3 className="text-xl font-semibold mb-
1">{w.name}</h3>
              <p className="text-gray-600 mb-1">🔑 {w.location}</p>
              <p className="text-sm text-gray-500 mb-3">
                Кількість вільних місць: {w.capacity || "Unknown"}
              </p>
              <button
                onClick={() => navigate("/book")}
                className="bg-blue-600 text-white px-4 py-2 rounded
hover:bg-blue-700 w-full"
              >
                Забронювати зараз!
              </button>
            </div>
          </motion.div>
        ))}
      </motion.div>

      {filteredWorkspaces.length === 0 && (
        <p className="text-center text-gray-500 mt-8">Не знайдено
робочих місць.</p>
      )}
    </div>
  );
}
```

Додаток Б

Код серверної сторони

Лістинг коду бази даних та комунікацій з клієнтом

```
const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');
const app = express();
const path = require('path');

// Middleware
app.use(cors());
app.use(express.json());
app.use('/images', express.static(path.join(__dirname,
'public/images'))); // Serve images statically

// MongoDB connection
mongoose.connect('mongodb://localhost:27017/cowork_booking', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
}).then(() => {
  console.log('MongoDB connected');
}).catch((err) => {
  console.error('MongoDB connection error:', err);
});

app.get('/', (req, res) => {
  res.send('Backend is running ✅');
});

// --- Models ---

const UserSchema = new mongoose.Schema({
  name: String,
  email: String,
  password: String,
});
const User = mongoose.model('User', UserSchema);

const WorkspaceSchema = new mongoose.Schema({
  name: String,
  location: String,
  capacity: Number,
  amenities: [String],
  image: String, // New field for image path
});
const Workspace = mongoose.model('Workspace', WorkspaceSchema);

const BookingSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },
  workspaceId: { type: mongoose.Schema.Types.ObjectId, ref:
'Workspace' },
  date: String,
  timeSlot: String,
```

```
});
const Booking = mongoose.model('Booking', BookingSchema);

// --- Routes ---

// Login user
app.post('/api/login', async (req, res) => {
  const { email, password } = req.body;
  const user = await User.findOne({ email, password });

  if (!user) {
    return res.status(401).json({ error: 'Invalid credentials' });
  }

  res.status(200).json(user);
});

// Register user
app.post('/api/register', async (req, res) => {
  const { name, email, password } = req.body;
  if (!name || !email || !password)
    return res.status(400).json({ error: 'Missing fields' });

  const newUser = new User({ name, email, password });
  await newUser.save();
  res.status(201).json(newUser);
});

// Get all workspaces
app.get('/api/workspaces', async (req, res) => {
  const workspaces = await Workspace.find();
  res.json(workspaces);
});

app.post('/api/workspaces', async (req, res) => {
  const { name, location, capacity, amenities, image } = req.body;

  if (!name || !location || !capacity) {
    return res.status(400).json({ error: 'Missing required fields'
});
  }

  try {
    const newWorkspace = new Workspace({
      name,
      location,
      capacity,
      amenities: amenities
        ? Array.isArray(amenities)
          ? amenities
          : amenities.split(',').map(item => item.trim())
        : [],
      image: image || `/images/default.jpg`,
    });

    await newWorkspace.save();
    res.status(201).json(newWorkspace);
  } catch (err) {
```

```
    console.error('Error creating workspace:', err);
    res.status(500).json({ error: 'Server error' });
  }
});

// Create a booking with availability check
app.post('/api/bookings', async (req, res) => {
  const { userId, workspaceId, date, timeSlot } = req.body;

  if (!userId || !workspaceId || !date || !timeSlot) {
    return res.status(400).json({ error: 'Missing fields' });
  }

  try {
    const existing = await Booking.findOne({ workspaceId, date,
timeSlot });

    if (existing) {
      return res.status(409).json({ error: 'This timeslot is already
booked' });
    }

    const newBooking = new Booking({ userId, workspaceId, date,
timeSlot });
    await newBooking.save();
    res.status(201).json(newBooking);
  } catch (err) {
    console.error("Booking creation failed:", err);
    res.status(500).json({ error: "Server error" });
  }
});

// bookings
app.get('/api/bookings/:userId', async (req, res) => {
  try {
    const userObjectId = new
mongoose.Types.ObjectId(req.params.userId);
    const bookings = await Booking.find({ userId: userObjectId
}).populate('workspaceId');
    res.json(bookings);
  } catch (err) {
    console.error("Error fetching bookings:", err);
    res.status(400).json({ error: 'Invalid userId' });
  }
});

// Delete booking by ID
app.delete('/api/bookings/:id', async (req, res) => {
  try {
    const { id } = req.params;
    await Booking.findByIdAndDelete(id);
    res.status(200).json({ message: 'Booking deleted' });
  } catch (err) {
    console.error('Failed to delete booking:', err);
    res.status(500).json({ error: 'Server error' });
  }
});
```

```
// Отримати доступні часові слоти
app.get('/api/bookings/slots', async (req, res) => {
  const { workspaceId, date } = req.query;

  if (!workspaceId || !date) {
    return res.status(400).json({ error: 'Missing workspaceId or
date' });
  }

  const allSlots = [
    "09:00", "10:00", "11:00", "12:00",
    "13:00", "14:00", "15:00", "16:00",
    "17:00", "18:00", "19:00"
  ];

  try {
    const bookings = await Booking.find({ workspaceId, date });
    const bookedSlots = bookings.map(b => b.timeSlot);

    const availableSlots = allSlots.filter(slot =>
!bookedSlots.includes(slot));

    res.json(availableSlots);
  } catch (err) {
    console.error("Error fetching slots:", err);
    res.status(500).json({ error: "Server error" });
  }
});

// Start server
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```