МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБСАЙТУ КОВОРКІНГ-ЦЕНТРУ ІЗ ЗАСТОСУВАННЯМ БІБЛІОТЕКИ REACT»**

Виконав: студент 3 курсу, групи 6.1211-пі-с
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія (зі скороченим терміном навчання)
(назва освітньої програми)
М.І. Сафонов
(ініціали та прізвище)
Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Лимаренко Ю.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент професор кафедри комп'ютерних наук, доцент, д.т.н. Шило Г.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія (зі скороченим терміном навчання)

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Сафонову Микиті Ігоровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебсайту коворкінг-центру із застосуванням бібліотеки React

керівник роботи Лимаренко Юлія Олексіївна, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та реалізація програмного забезпечення.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	26.01.2024	
3.	Обробка методичних та теоретичних	16.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	12.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	21.06.2024	

Студент _____ М.І. Сафонов
(підпис) (ініціали та прізвище)

Керівник роботи _____ Ю.О. Лимаренко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка вебсайту коворкінг-центру із застосуванням бібліотеки React»: 65 с., 40 рис., 11 джерел, 1 додаток.

АВТЕНТИФІКАЦІЯ, АВТОРИЗАЦІЯ, БАЗА ДАНИХ, БЕКЕНД, ВЕБДОДАТОК, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ТОКЕН, ФРОНТЕНД.

Об’єкт дослідження – основні методи та засоби розробки вебдодатків.

Мета роботи: розробка вебдодатка коворкінг-центру.

Метод дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання та проєктування програмного забезпечення.

У кваліфікаційній роботі було наведено аналіз предметної області, огляд існуючих систем з предметної області та описано засоби для створення вебдодатків. На основі даних теоретичних відомостей розроблено проєкт вебдодатка коворкінг-центру. Результатом виконання кваліфікаційної роботи є вебдодаток з клієнт-серверною архітектурою, написаний, використовуючи стек MERN (MongoDB, Express.js, React, Node.js).

SUMMARY

Bachelor’s qualifying paper “Development of a Coworking Center Website Using the React Library”: 65 pages, 40 figures, 11 references, 1 supplement.

AUTHENTICATION, AUTHORIZATION, BACKEND, CLIENT-SERVER ARCHITECTURE, DATABASE, FRONTEND, TOKEN, WEB APPLICATION.

The object of the study is main methods and tools of developing web applications.

The aim of the study is development of a co-working center web-application.

The methods of research are collecting and analyzing software requirements, methods of modeling and designing software.

The qualification paper provided an analysis of the subject area, an overview of existing systems in the subject area, and described tools for creating web applications. On the basis of these theoretical information, a web application project of the co-working center was developed. The result of the qualification work is a web application with a client-server architecture, written using the MERN stack (MongoDB, Express.js, React, Node.js).

ЗМІСТ

ВСТУП

У сучасному світі коворкінг-центри стають все більш популярними, пропонуючи зручні робочі простори для фрілансерів, стартапів та малих бізнесів. Створення вебдодатка для управління коворкінг-центром є актуальним і корисним завданням.

Додаток для управління коворкінг-центром може залучати користувачів і координувати їх діяльність, надаючи зручний спосіб бронювання робочих місць і ресурсів. Користувачі можуть легко знайти вільні робочі місця, забронювати конференц-зали та користуватися іншими послугами коворкінгу. Додаток також може полегшити процес управління коворкінг-центром. Організатори можуть створювати та оновлювати бронювання, встановлювати терміни використання, відстежувати доступність ресурсів, надавати звіти тощо. Додаток може забезпечувати зручні інструменти для комунікації.

Отже, у якості виконання кваліфікаційної роботи було вирішено створити вебдодаток для управління коворкінг-центром. Додаток був розроблений за допомогою стеку технологій MERN (MongoDB, Express.js, React, Node.js).

Перший розділ кваліфікаційної роботи містить характеристику предметної області. Наведені три приклади додатків для управління коворкінг-центрами, описаний їхній функціонал. Описаний стек технологій, за допомогою якого можна розробити вебдодаток: MongoDB, Express.js, React, Node.js.

У другому розділі описано технічне завдання, призначення розробки та технічні вимоги до програмного продукту. Наведено діаграму варіантів використання, ER-діаграму бази даних. Описані етапи створення вебдодатка та наведені макети застосунку.

Третій розділ містить особливості реалізації додатка. Описано ініціалізацію та налаштування проєктів серверної та клієнтської частин.

Наведений опис процесу створення бази даних MongoDB та роботи з нею за допомогою Mongoose. Описано процес розробки серверної частини, використовуючи платформу Node.js, фреймворк Express.js та супутні зовнішні пакети. Описано процес розробки клієнтської частини, використовуючи фреймворк React та супутні йому бібліотеки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБ'ЄКТУ ДОСЛІДЖЕННЯ

1.1 Аналіз предметної області

Коворкінг-центри – це сучасні робочі простори, які надають фрілансерам, стартапам, малим та середнім підприємствам, а також великим компаніям можливість орендувати робочі місця на гнучких умовах. Основна мета коворкінг-центрів полягає у створенні комфортного та продуктивного середовища для роботи, що допомагає забезпечити ефективну діяльність та розвиток бізнесу клієнтів.

Ринок коворкінг-центрів активно розвивається як на глобальному, так і на локальному рівні. Цей розвиток обумовлений зростанням кількості фрілансерів, стартапів та компаній, що переходять на гнучкі графіки роботи. Коворкінг-центри стають популярними в мегаполісах і великих містах, де висока вартість оренди традиційних офісів та потреба в гнучкості сприяють попиту на коворкінги.

Основними користувачами коворкінг-центрів є фрілансери, IT-спеціалісти, дизайнери, копірайтери, маркетологи та інші професіонали, які шукають комфортне робоче місце поза домом. Також до цільової аудиторії належать стартапи, які потребують доступного офісного простору для своїх команд, малі та середні підприємства, що шукають тимчасові офісні приміщення або прагнуть знизити витрати на оренду, і великі компанії, що використовують коворкінг-центри для тимчасових проєктів, зустрічей або тренінгів.

Конкуренція на ринку коворкінг-центрів висока. Основними конкурентами є інші коворкінг-центри – локальні та міжнародні мережі, що пропонують різні формати робочих просторів, традиційні офіси, що здаються в оренду на тривалі строки, та альтернативні робочі місця, такі як кафе, бібліотеки та домашні офіси.

Коворкінг-центри пропонують широкий спектр послуг та зручностей, що роблять їх привабливими для різних груп користувачів. До них належать гарячі столи, закріплені місця, приватні офіси, високошвидкісний інтернет, офісна техніка (принтери, сканери), конференц-зали, кухні, лаунж-зони, переговорні кімнати, тренажерні зали та організація заходів. Крім того, коворкінг-центри сприяють комунікації та співпраці, організовуючи події, семінари та тренінги, що допомагають створити професійну спільноту.

Економічні аспекти включають високі початкові витрати на облаштування простору, але при правильному управлінні коворкінг-центри можуть стати прибутковими. Важливо інвестувати в сучасні технології для управління бронюванням, доступом, безпекою та обслуговуванням клієнтів.

Сучасні тенденції включають зростання популярності гібридних моделей роботи, підвищення уваги до здоров'я та безпеки на робочому місці, а також зростання попиту на спеціалізовані коворкінги для окремих галузей (наприклад, для IT-фахівців чи креативних професій). Коворкінг-центри, що зможуть адаптувати свої послуги до цих змін, матимуть конкурентні переваги.

Аналіз предметної області показує, що коворкінг-центри мають великий потенціал для зростання завдяки змінюваним робочим практикам та потребам сучасних професіоналів. Успішне функціонування коворкінг-центру залежить від здатності пропонувати зручні, безпечні та інноваційні рішення, що відповідають потребам різних категорій користувачів.

1.2 Огляд існуючих коворкінг-центрів

У цьому підрозділі буде проведено аналіз існуючих аналогів за наступними пунктами:

- послуги, що надаються сервісом;
- зручність використання (зручність, адаптивність інтерфейсу);
- особливості сервісу.

1.2.1 Коворкінг-центр “Creative State”

Першим аналогом є коворкінг-центр “Creative State”, який надає великий спектр послуг. Опираючись на опис сервісу на вебсайті цього центру, надаються наступні послуги:

- можливість бронювання робочого місця;
- велика кількість локацій та типів робочих місць, що дозволяє будь-якому користувачу знайти те, що він шукає;
- багато зручних властивостей у робочих місць (наприклад, pet friendly, що дозволяє брати з собою тварин);
- велика кількість різних тарифів, для гнучкої оренди робочого місця;
- наявні менеджери, які можуть виконати багато різних функцій (наприклад, якщо користувач планує відпустку, менеджер може забронювати готель, тощо);
- партнерські програми;
- щотижневі події;
- наявний телеграм бот, за допомогою якого можна виконувати багато функцій.

Щодо зручності використання, інтерфейс вебдодатка гарно структурований та стилізований, адаптивний, зручний.

У цього сервісу також наявні особливості, серед них можна виділити наступні:

- зручний перемикач теми (світла, темна);
- перемикач мови (українська, англійська);
- загальна якість сервісу (наявні багато послуг, якість реалізації вебдодатку).

1.2.2 Коворкінг-центр “Platforma Leonardo”

Другим аналогом є коворкінг-центр “Platforma Leonardo”, який надає менший спектр послуг ніж попередник, але все одно залишається непоганим сервісом. Надаються наступні послуги:

– основна функція – бронювання робочого місця;
– наявні декілька локацій для оренди робочих місць;
– наявні декілька типів робочих місць, кожний тип має свої тарифи;
– наявний блог, на якому можна переглянути різні статті та інформацію;
– наявні власні події сервісу (тобто, не користувачі відповідають за ці події, а саме сервіс).

Щодо зручності використання, інтерфейс вебдодатка непогано структурований та стилізований, адаптивний, зручний. Але все ж таки є недоліки у стилізації, через це можуть виникати певні незручності у використанні.

У цього сервісу також наявні особливості, серед них можна виділити наступні:

– перемикач мови (українська, англійська, російська);
– перемикач локації (вибір міста).

1.2.3 Коворкінг-центр “_AND WORK”

Третім аналогом є коворкінг-центр “_AND WORK”, який надає не надає додаткових послуг, окрім бронювання робочого місця. Сервіс надає наступні послуги:

– одна локація з різними типами робочих місць;
– наявні декілька тарифів, які надають різні можливості та переваги.

Щодо зручності використання, інтерфейс вебдодатка має досить багато

недоліків, наявні проблеми з кольорами (деяка інформація погано зчитується, через неправильно підібрані кольори), проблеми з адаптивністю та загалом зі структурою.

У цього сервісу відсутні особливості (раніше був наявний перемикач мови, наразі він відсутній).

1.3 Засоби створення вебдодатка

На початку розробки проєкту вебдодатка коворкінг центру постало питання вибору стеку технологій для його створення.

Було вирішено розробити SPA. Single Page Application (SPA) – це тип web-додатку, в якому вся інтерактивність і відображення даних відбувається на одній web-сторінці, без необхідності завантаження додаткових сторінок або перезавантаження браузера [1].

Однією з головних переваг SPA є покращений користувацький досвід (UX) завдяки миттєвому завантаженню даних і відображенню змін на сторінці. SPA також забезпечує швидку роботу на мобільних пристроях і дає змогу створювати крос-платформні застосунки [1].

У SPA всі компоненти, такі як кнопки, форми, меню та інші елементи, зазвичай керуються через JavaScript-фреймворки та бібліотеки, як React, Angular, Vue.js та інші. Ці фреймворки забезпечують модульну архітектуру і можливість створення компонентів, які можна використовувати повторно [1].

На противагу SPA, існує також MPA. Multi Page Application – це багатосторінкові додатки, які працюють за традиційною схемою. Це означає, що при кожній незначній зміні даних або завантаженні нової інформації сторінка оновлюється. Такі додатки важче, ніж односторінкові, тому їх використання доцільно тільки в тих випадках, коли потрібно відобразити велику кількість контенту [2].

Кожна архітектура має свої переваги та недоліки і добре підходить для

певного типу проєктів. SPA відрізняється швидкістю і можливістю розробити мобільний додаток на основі готового коду. Але водночас SPA має погану SEO-оптимізацію. Таким чином, ця архітектура є відмінним підходом для SaaS-платформ, соціальних мереж, закритих спільнот, де пошукова оптимізація не має значення. MPA більше підходить для інтернет-магазинів, бізнес-сайтів, каталогів, маркетплейсів на кшталт Etsy тощо. Добре оптимізований MPA має високу продуктивність, але все ще не дозволяє легко розробити мобільний додаток. MPA і SPA з правильною архітектурою добре підходять для розробки масштабованих вебдодатків [3].

Отже, розробку проєкту було вирішено провести використовуючи наступний стек технологій:

– документо-орієнтована система керування базами даних MongoDB;
– платформу Node.js, фреймворк Express.js та допоміжні зовнішні Node.js пакети;
– бібліотеку React з супутнім йому стеком.

1.3.1 Огляд фреймворку Express.js

Express – це мінімалістичний та гнучкий фреймворк для вебзастосунків, побудованих на Node.js, що надає широкий набір функціональності. Маючи в своєму розпорядженні безліч допоміжних HTTP-методів та проміжних обробників, створювати надійні API можна легко і швидко. Express забезпечує тонкий прошарок базової функціональності для вебзастосунків, що не спотворює звичну та зручну функціональність Node.js [4].

Стосовно переваг фреймворку, серед них можна виділити наступні:

– простота використання (простий та зрозумілий синтаксис);
– гнучкість (можна обирати лише ті компоненти, які необхідні для вашого проєкту, і легко розширювати функціональність за допомогою сторонніх модулів);

– багата екосистема (є велика кількість модулів та middleware для Express.js, що дозволяють реалізовувати різноманітні функції, наприклад, авторизація, автентифікація, обробка запитів, тощо).

Також є певні недоліки:

– брак стандартизації (оскільки Express.js надає велику гнучкість, можуть виникнути проблеми з різноманітністю підходів до розробки між різними проєктами, тобто, код може дуже відрізнятись);
– потенційні проблеми з безпекою (при неправильному використанні, може призвести до проблем з безпекою, таких як вразливості або різноманітні атаки).

1.3.2 Огляд бібліотеки React

React – це декларативна, ефективна і гнучка JavaScript-бібліотека для створення інтерфейсів користувача. Вона дозволяє розробникам будувати компоненти, які забезпечують відображення і взаємодію з даними на вебсторінці [5].

React базується на компонентній архітектурі, що дозволяє розбивати інтерфейс на незалежні компоненти. Це спрощує розробку, тестування та підтримку коду. Компоненти можуть повторно використовуватися в різних частинах додатку [5, 6].

Основні переваги React:

– модульність коду, який можна використовувати повторно;
– ефективне оновлення і повторний рендеринг компонентів при зміні даних;
– широкі можливості для розробки вебдодатків і мобільних застосунків будь-якого масштабу;
– велика і активна спільнота розробників.

React обробляє тільки користувацький інтерфейс в застосунках, надаючи

розробникам свободу вибору інших інструментів для вирішення інших завдань. Він безпосередньо спрямований на побудову інтерфейсів, а не на надання повної «схеми додатків».

React добре підходить для розробки проєктів будь-якого масштабу. Він надає можливості для легкого розширення та перевикористання компонентів, інтеграції з іншими бібліотеками та фреймворками. Також підтримує серверний рендеринг, що дозволяє поліпшити швидкість завантаження сторінок та оптимізувати пошукову оптимізацію [6].

Таким чином, React – це потужна і гнучка бібліотека для створення сучасних інтерактивних інтерфейсів, яка дозволяє писати модульний, ефективний і масштабований код.

1.3.3 Огляд СКБД MongoDB

MongoDB – це відкрите документно-орієнтоване сховище даних, яке призначене для обробки великих обсягів даних і забезпечення високої продуктивності. Воно використовує гнучку схему, яка дозволяє зберігати структуровані, напівструктуровані та неструктуровані дані [7].

MongoDB зберігає дані у вигляді документів у форматі JSON (JavaScript Object Notation) або BSON (Binary JSON), що є аналогом рядків у реляційних базах даних. Документи можуть мати різну структуру і містити вкладені дані, що робить їх зручними для представлення складних ієрархічних відношень [7, 8].

MongoDB підтримує широкий спектр операцій запитів, індексування, агрегації та обробки транзакцій. Це дозволяє ефективно обробляти дані в реальному часі та проводити аналітику великих обсягів даних.

Основні переваги MongoDB [7, 8]:

– гнучка схема даних, яка дозволяє зберігати різнотипні дані в одному документі;

– висока швидкість обробки великих обсягів даних;
– горизонтальна масштабованість за допомогою шардингу (розподілу даних по серверах);
– висока доступність завдяки вбудованій реплікації та відмовостійкості;
– підтримка багатьох мов програмування через драйвери.

Недоліки MongoDB [8]:

– обмеження розміру документа до 16 МБ;
– відсутність підтримки транзакцій між документами;
– складність налаштування реплікації та шардингу.

MongoDB широко використовується в додатках, які потребують гнучкого зберігання даних, високої доступності та масштабованості, таких як вебдодатки, мобільні додатки, системи аналітики та управління контентом.

MongoDB має кілька переваг порівняно з реляційними базами даних [9]:

– гнучка схема даних – MongoDB дозволяє зберігати дані у вигляді документів в форматі JSON, що дає можливість працювати з неструктурованими та напівструктурованими даними (це контрастує з жорсткою схемою реляційних баз даних);
– висока швидкість обробки великих обсягів даних – MongoDB оптимізована для роботи з великими обсягами даних, забезпечуючи високу продуктивність (це робить її ефективною для додатків, що працюють з великими наборами даних);
– горизонтальна масштабованість – MongoDB підтримує шардинг, який дозволяє розподіляти дані по декількох серверах, забезпечуючи горизонтальне масштабування (це дає можливість обробляти великі навантаження);
– висока доступність – MongoDB має вбудовану реплікацію, що забезпечує відмовостійкість та доступність даних (це важливо для критичних додатків);
– підтримка багатьох мов програмування – MongoDB має драйвери для

більшості популярних мов програмування, що спрощує інтеграцію з додатками.

Таким чином, MongoDB виграє у гнучкості, масштабованості та продуктивності порівняно з реляційними базами даних, що робить її привабливим вибором для сучасних додатків, які працюють з великими обсягами різнотипних даних.

2 РОЗРОБКА ПРОЄКТУ

2.1 Технічне завдання

Перед початком створення програмної реалізації вебдодатка коворкінг-центру потрібно сформувати технічне завдання, технічні вимоги до додатка та етапи його розробки.

2.1.1 Найменування та область застосування

Програмний продукт, що розробляється, має найменування “Коворкінг-центр”. Додаток призначений для покращення функціонування та управління коворкінг-центром, а також полегшення взаємодії з резидентами та клієнтами.

2.1.2 Призначення розробки

Даний проєкт призначений для вирішення наступних завдань:

– авторизація (реєстрація, вхід, вихід);
– можливість редагування власного профілю;
– можливість бронювання робочого місця;
– можливість перегляду стрічки новин та оголошень;
– можливість перегляду інформації про доступні робочі місця;
– можливість перегляду інформації про доступні тарифи.

2.1.3 Технічні вимоги до програмного продукту

На початку роботи були сформовані наступні технічні вимоги:

– навігацію та взаємодію з сервером потрібно реалізувати за допомогою AJAX (Asynchronous JavaScript And XML) без перезавантаження сторінки;
– модуль аутентифікації має бути реалізований за допомогою JWT (JSON Web Token), використовувати систему з access- і refresh-токенів;
– сервер повинен повертати потрібні коди відповідей (у випадках успіху та при помилках);
– певний функціонал має бути захищеним від неавторизованого користувача (не потрібно захищати сторінки цілком, так як на них присутній функціонал, який доступний для неавторизованого користувача також);
– має бути присутня валідація даних для всіх форм, має відображатись відповідний інтерфейс;
– всі сторінки повинні мати єдиний стиль;
– інтерфейс має бути зручним та адаптивним;
– потрібно реалізувати модуль налаштувань для зміни мови та кольору (для зручності).

2.2 Етапи створення вебдодатка

Процес розробки вебдодатка коворкінг-центру можна розділити на декілька етапів:

– складання технічного завдання та вимог до системи;
– вибір стеку для реалізації;
– створення макетів сторінок;

– реалізація серверної частини;
– реалізація клієнтської частини;
– тестування.

Перший етап є фундаментальним, оскільки визначає, що саме потрібно від системи. Без чіткого розуміння вимог розробка може піти в неправильному напрямку, що призведе до марнотратства ресурсів та часу. Включає в себе збирання інформації про функціональні потреби, як бронювання робочих місць та управління резидентами, а також нефункціональні вимоги, такі як продуктивність, безпека та масштабованість. Це дозволяє сформувати чітке бачення проєкту та встановити критерії успіху.

Вибір відповідних технологій для реалізації проєкту впливає на ефективність розробки, продуктивність та масштабованість кінцевого продукту. Вибір стеку базується на вимогах проєкту, доступності ресурсів та експертизі команди. Правильно обраний стек забезпечує стабільність, підтримку та розвиток проєкту у майбутньому.

Цей етап включає проєктування інтерфейсу користувача (UI) і користувацького досвіду (UX). Макети допомагають візуалізувати, як виглядатиме кінцевий продукт, та забезпечують зручність використання системи для користувачів. Чіткі та зручні макети допомагають уникнути непорозумінь між замовником та командою розробки, забезпечують послідовність у дизайні та допомагають швидко отримати зворотний зв’язок від користувачів ще до початку розробки.

Серверна частина обробляє запити від клієнтської частини, управляє даними та забезпечує безпеку системи. Це «мозок» додатка, який відповідає за логіку роботи, обробку даних та їх збереження. Надійна серверна частина гарантує стабільність, безпеку та ефективність роботи всієї системи.

Клієнтська частина є тим, з чим взаємодіють користувачі. Вона повинна бути зручною, швидкою та інтуїтивно зрозумілою. Якісно реалізована клієнтська частина забезпечує позитивний користувацький досвід, що є ключовим фактором для залучення та утримання клієнтів. Вона також повинна бути адаптивною, щоб коректно працювати на різних пристроях.

Тестування є критично важливим для виявлення та виправлення помилок, забезпечення стабільності та безпеки системи перед її запуском. Воно включає перевірку всіх аспектів додатка, від окремих компонентів до системи в цілому. Якісне тестування знижує ризики збоїв, забезпечує відповідність вимогам та покращує користувацький досвід.

2.3 Діаграма прецедентів

Діаграма варіантів використання (діаграма прецедентів, сценарій використання, use case) — дозволяє уявити типи ролей та їх взаємодію із системою. Проте не показує порядок виконання кроків. Зображує функціональні вимоги (те, що система може зробити) з точки зору користувача. Може описуватись текстом або у вигляді діаграми [11].

Діаграми використання розробляються на ранній стадії проєктування системи та призначені для: 1) простого пояснення роботи системи, створення та погодження ТЗ; 2) формування функціональних (що має робити) вимог до системи; 3) створення основи для документації та тестування [11].

У діаграмі прецедентів зображуються актори (користувачі системи) та прецеденти (функціональність системи). Актори представляють зовнішніх користувачів, а прецеденти описують конкретні дії, які можуть бути виконані акторами.

Далі будуть приведені описи прецедентів для акторів – адміністратора та користувача, а також сама діаграма (рис. 2.1), в котрій однотипні прецеденти були об’єднані для покращення зчитування.

Прецеденти актора “Адміністратор”:

– зареєструватися у системі;
– виконати вхід у систему;
– вийти з системи;
– переглянути оголошення;

– створити оголошення;
– відредагувати існуюче оголошення;
– видалити оголошення;
– переглянути новини;
– створити новину;
– відредагувати існуючу новину;
– видалити новину;
– переглянути робочі місця;
– створити нове робоче місце;
– відредагувати існуюче робоче місце;
– видалити робоче місце;
– переглянути тарифи;
– створити новий тариф;
– відредагувати існуючий тариф;
– видалити тариф;
– створити бронювання робочого місця;
– скасувати бронювання робочого місця;
– переглянути історію всіх бронювань;
– відредагувати інформацію про бронювання;
– редагувати власний профіль;
– відредагувати певного користувача;
– зберегти зміни профілю.

Прецеденти актора “ Користувач”:

– зареєструватися у системі;
– виконати вхід у систему;
– вийти з системи;
– переглянути оголошення;
– переглянути новини;
– переглянути робочі місця;
– переглянути тарифи;

– створити бронювання робочого місця;
– скасувати бронювання робочого місця;
– переглянути історію всіх бронювань;
– редагувати власний профіль;
– зберегти зміни профілю.

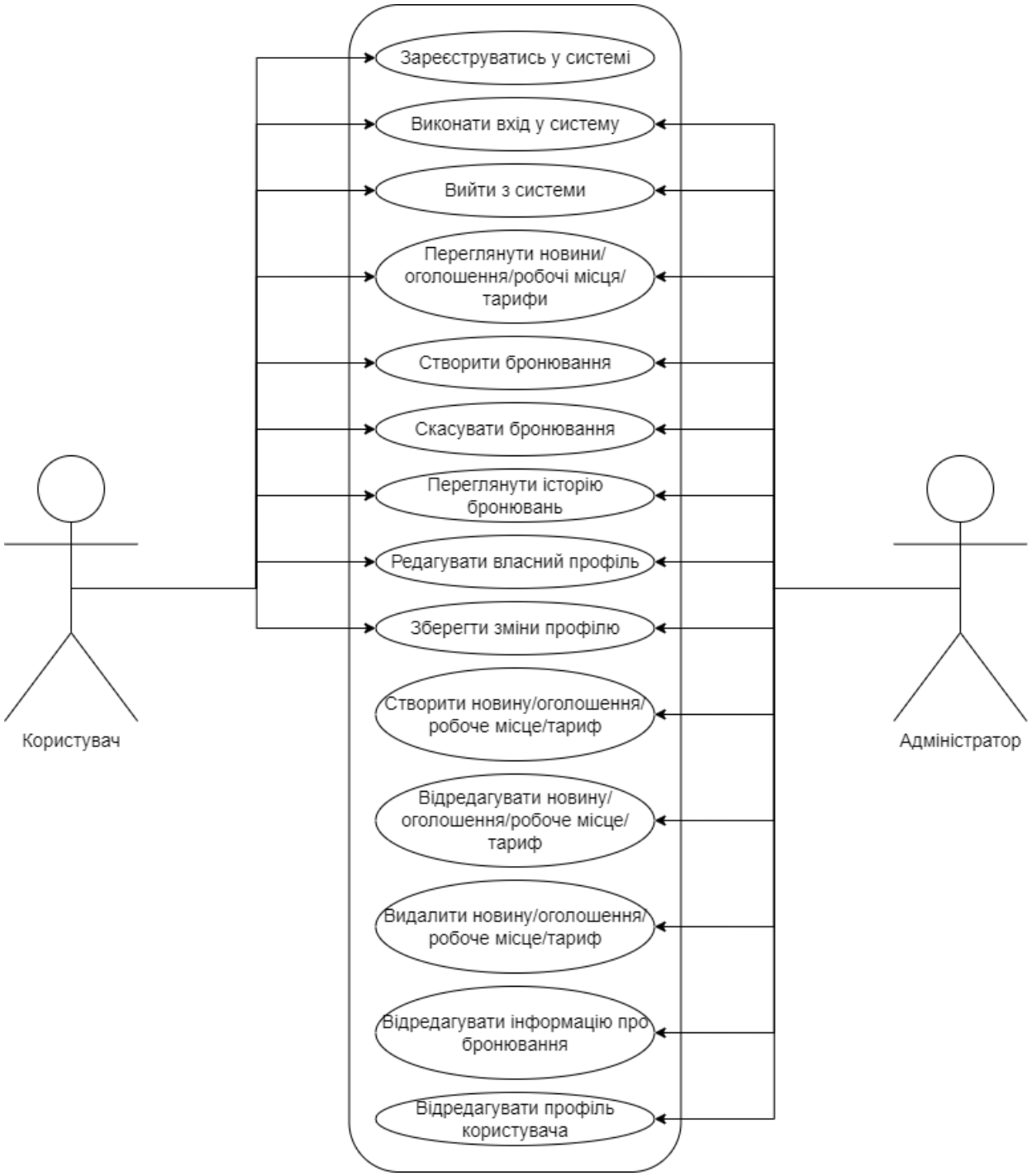

Рисунок 2.1 – Діаграма прецедентів

2.4 ER-модель бази даних

Була створена ER-модель бази даних, відповідно до вимог проєкту. Визначені сутності та їх атрибути, а також встановлені зв’язки між ними. Вона приведена нижче (рис. 2.2).

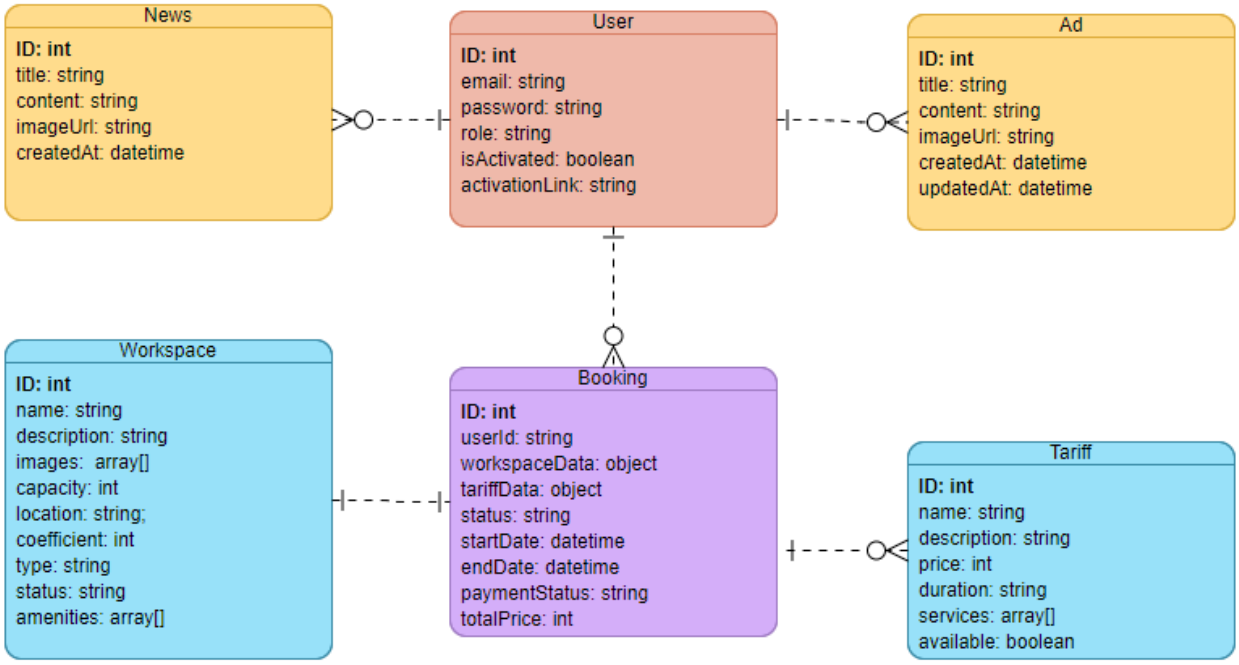

Рисунок 2.2 – ER-модель бази даних

2.5 Проєктування інтерфейсу

Опираючись на технічне завдання, був спроєктований інтерфейс додатку. Всього за планом, потрібно створити 6 сторінок, а також декілька модальних вікон (для авторизації, налаштувань). Нижче будуть приведені макети усіх необхідних сторінок.

Перед цим, слід зауважити, що усі сторінки мають спільні елементи, такі як, статичний сайдбар (який містить навігацію та кнопку налаштувань) та кнопки для авторизації, отже описані будуть лише елементи сторінки (без спільних елементів).

Перший макет головної сторінки, яка містить: хіро-секцію з привітанням та кнопку для бронювання робочого місця (рис. 2.3).

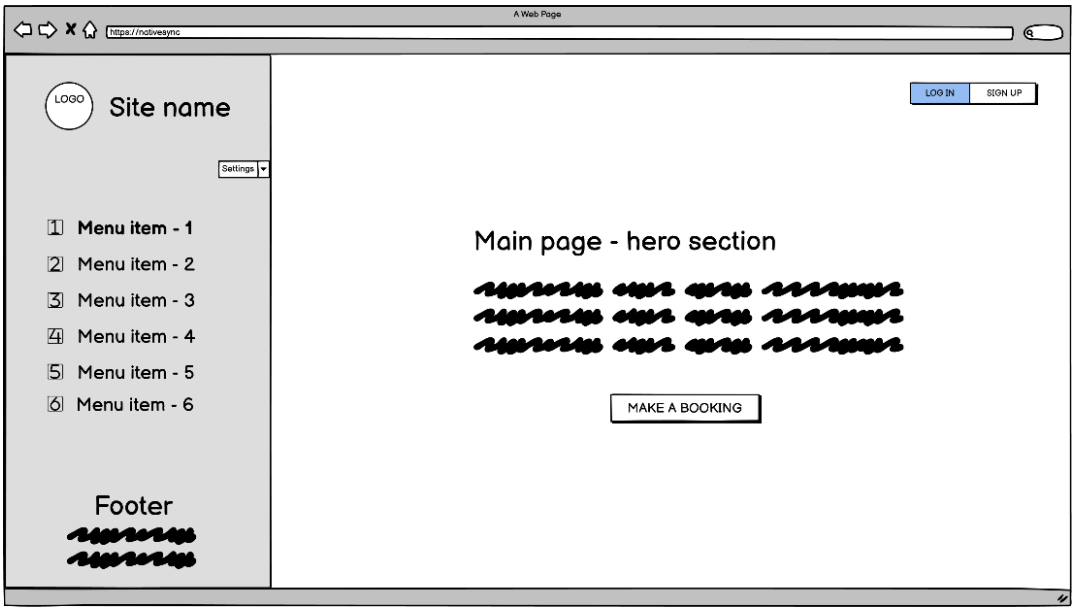

Рисунок 2.3 – Макет головної сторінки

Другий макет сторінки новин та оголошень, яка містить: заголовок сторінки, невеликий опис та секцію з картками новин та оголошень (в залежності від обраної вкладки), також наявний фільтр відображення по даті публікації (рис. 2.4).

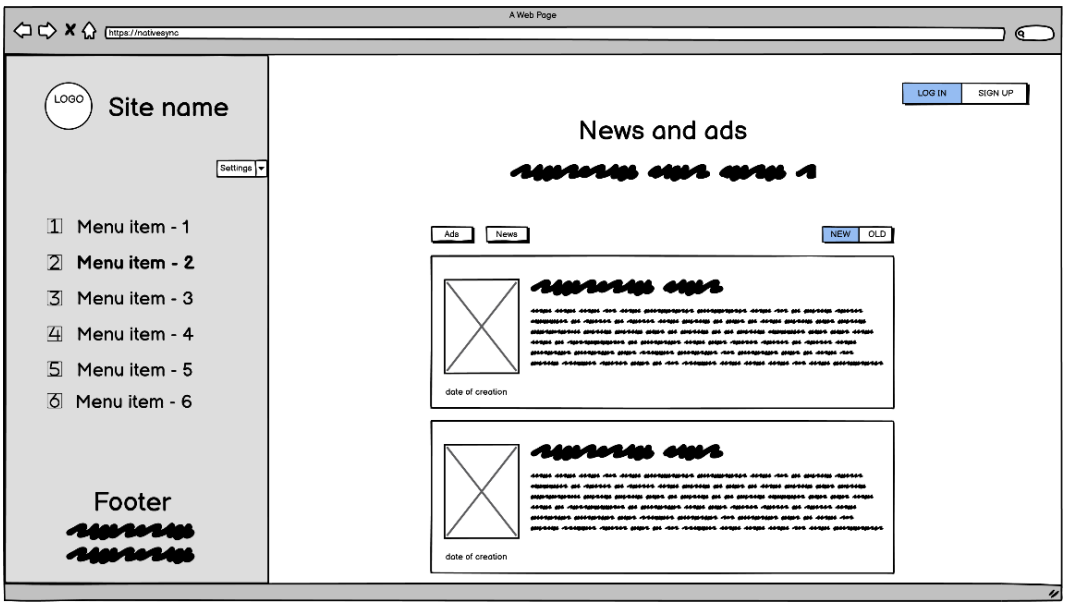

Рисунок 2.4 – Макет сторінки з новинами та оголошеннями

Третій макет сторінки з усіма локаціями (робочими місцями), яка містить: заголовок сторінки, невеликий опис та секцію, у якій відображаються робочі місця, в залежності від обраного типу робочого місця, робоче місце відображається у вигляді картки, яка містить усю необхідну інформацію про робоче місце (рис. 2.5).

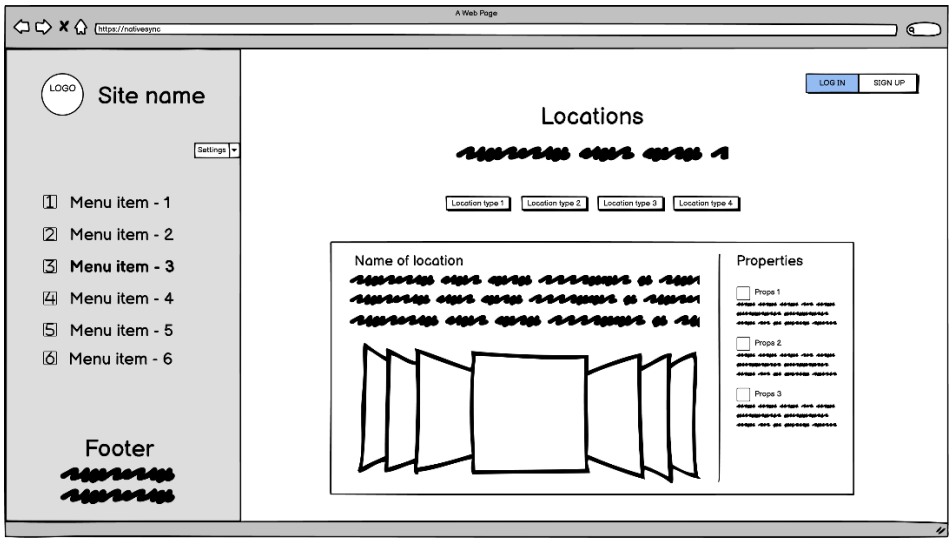

Рисунок 2.5 – Макет сторінки з робочими місцями

Четвертий макет сторінки з тарифами, яка містить: заголовок сторінки, невеликий опис та секцію з картками, на яких відображено опис кожного з тарифів (рис. 2.6).

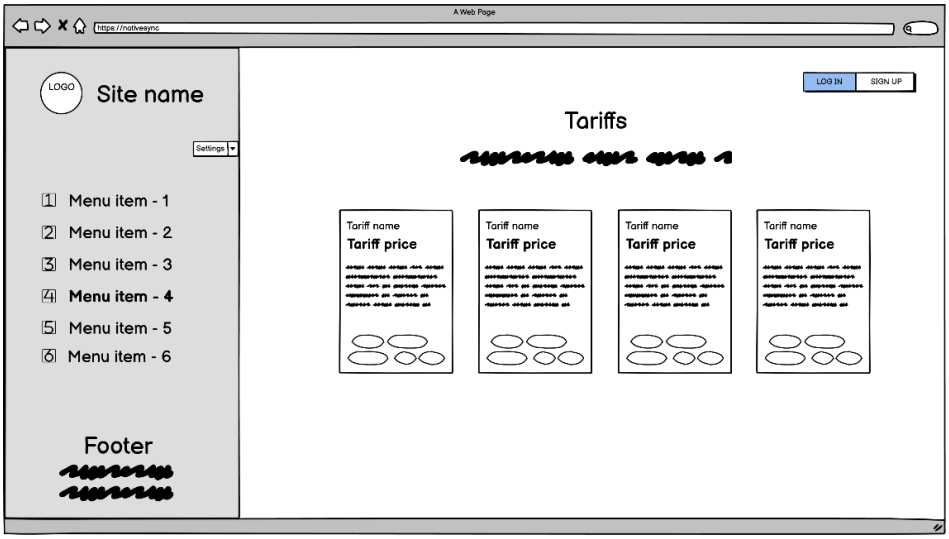

Рисунок 2.6 – Макет сторінки з тарифами

П’ятий макет сторінки з інформацією про коворкінг-центр, яка містить: опис сервісу, його переваги та функції (рис. 2.7).

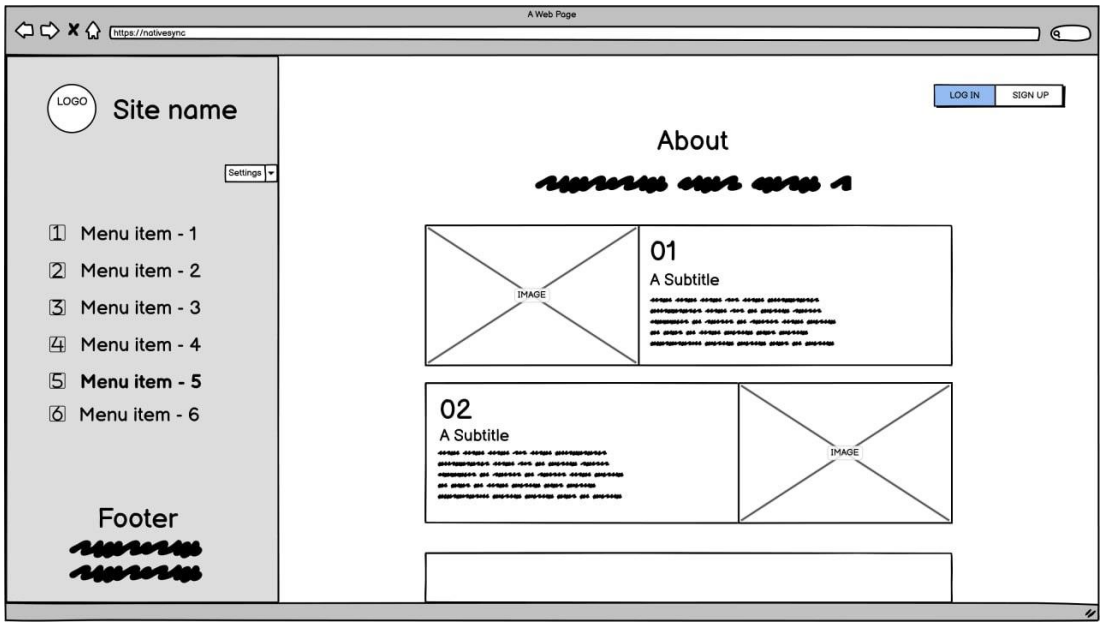

Рисунок 2.7 – Макет сторінки з описом коворкінг-центру

Шостий макет сторінки з контактними даними для зворотного зв’язку, яка містить: заголовок сторінки, блок з контактними даними, форму для зворотного зв’язку, мапу з місцеположенням центру (рис. 2.8).

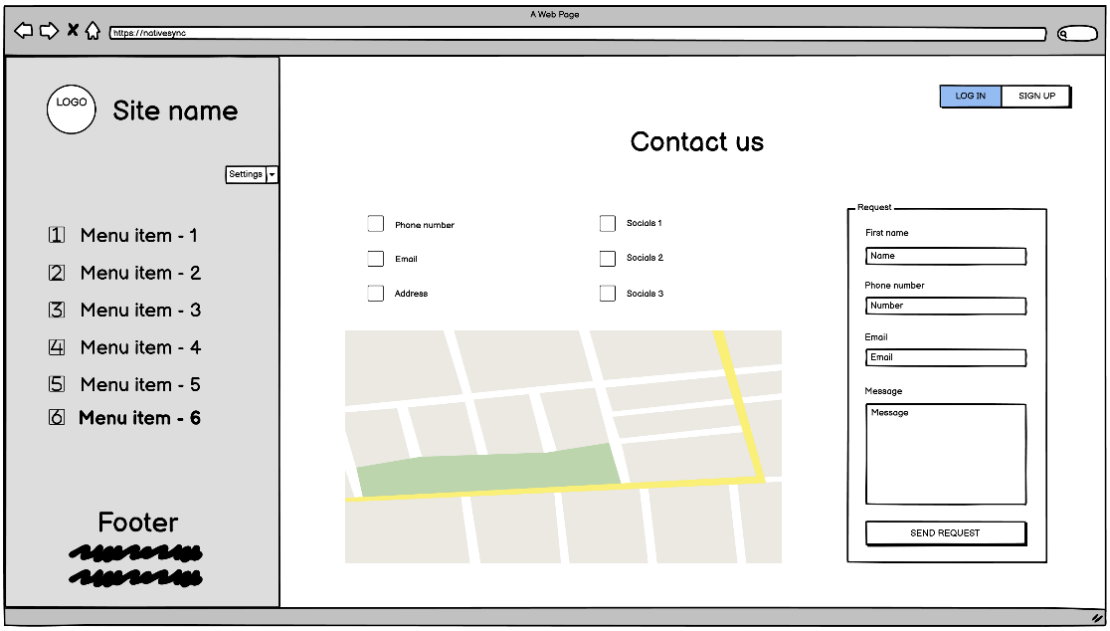

Рисунок 2.8 – Макет сторінки з контактними даними

Додатково будуть приведені макети модальних вікон, які відповідають за авторизацію та налаштування додатку (рис. 2.9 – 2.11).

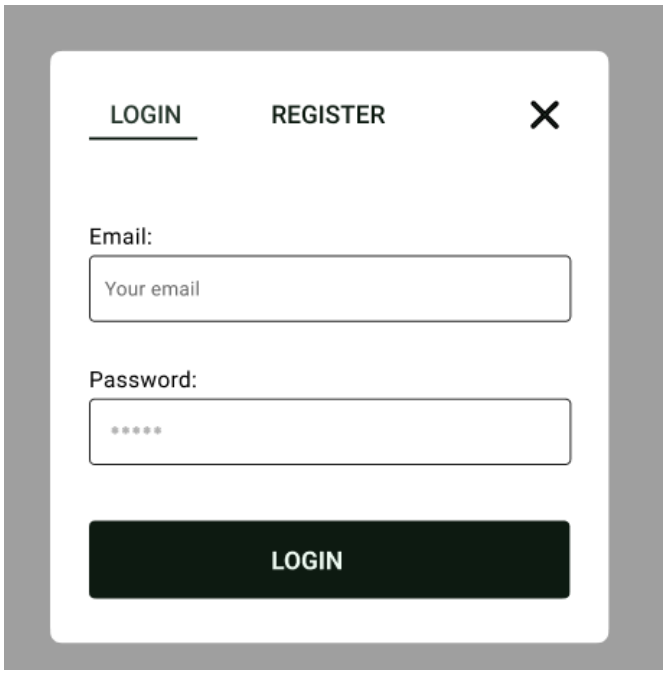

Рисунок 2.9 – Макет модального вікна авторизації

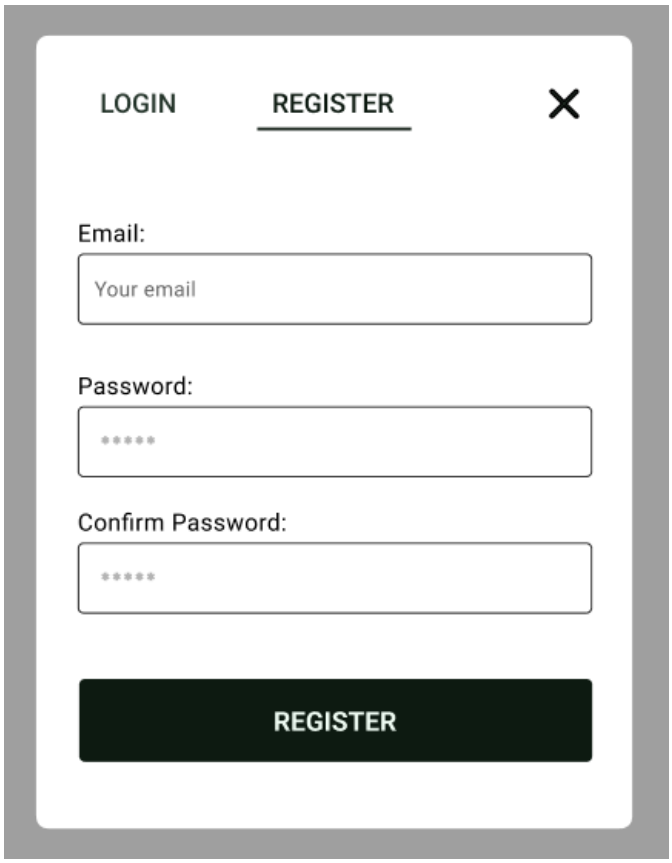

Рисунок 2.10 – Макет модального вікна реєстрації

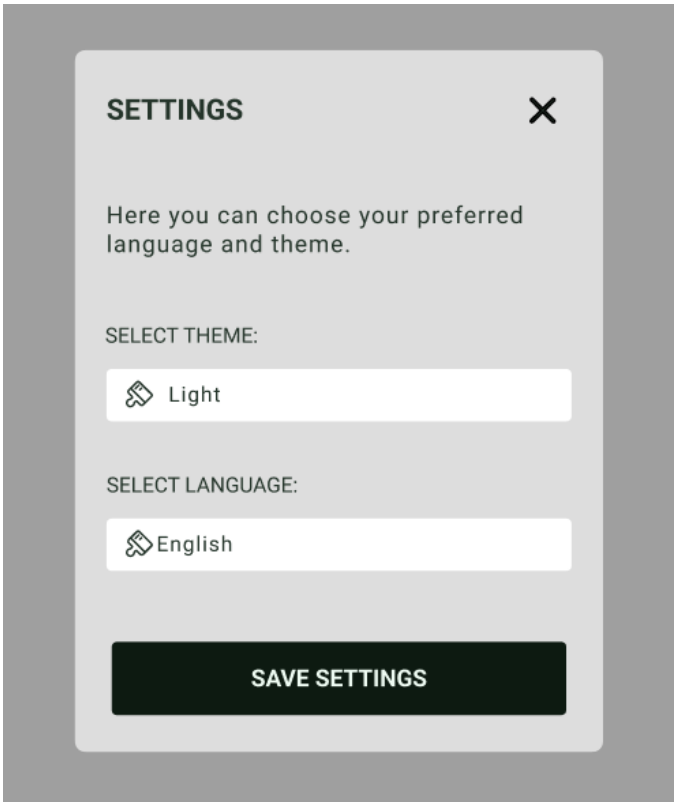

Рисунок 2.11 – Макет модального вікна налаштувань

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

На етапі проєктування додатку було вирішено створити SPA засобами MongoDB, Express.js, React, Node.js та супутніми бібліотеками. Під час розробки використовувалася система контролю версій Git. У цьому випадку, був створений монорепозиторій для клієнтської та серверної частин.

3.1 Розробка серверної частини

Перший етап розробки додатку – розробка серверної частини. Результатом цього етапу очікується – API, що розроблений на платформі Node.js з використанням фреймворку Express.js, працює з форматом даних JSON та реалізує весь функціонал, описаний на етапі проєктування.

3.1.1 Ініціалізація та налаштування проєкту

Ініціалізація Node.js проєкту була виконана за допомогою команди “init” пакетного менеджеру npm. Результат роботи команди – створений файл “package.json”, що є основною конфігурацією Node.js проєкту, головна інформація якого – необхідні пакети, потрібні для роботи додатка, пакети, необхідні при його розробці, та описання команд.

Було встановлено зовнішній пакет “nodemon” – інструмент розробки для Node.js, який дозволяє автоматично перезапускати серверний додаток при зміні файлів в проєкті. Також були встановлені необхідні для роботи додатка такі зовнішні пакети, як “express”, “jsonwebtoken”, “dotenv”, “cors”, “uuid”.

Був створений та налаштований екземпляр класу застосунку з бібліотеки Express.js та оброблений GET запит за маршрутом за замовчуванням.

Так як деплой додатку не планується, CORS policy був налаштований таким чином, щоб тільки клієнт з певним портом міг взаємодіяти з сервером. Далі буде приведено код налаштованого серверу (рис. 3.1).

```
require('dotenv').config()
const express = require('express')
const cors = require('cors')
const cookieParser = require('cookie-parser')
const mongoose = require('mongoose')
const errorHandler = require('./middlewares/errorHandler.middleware')
const Router = require('./routes/index.routes')

const PORT = process.env.PORT || 5000
const app = express()

app.use(express.json())
app.use(cookieParser())
app.use(
	cors({
		origin: 'http://localhost:3000',
		credentials: true
	})
)
app.use('/api', Router)
app.use(errorHandler)

const startServer = async () => {
	try {
		await mongoose.connect(process.env.DB_URL)
		app.listen(PORT, () => console.log(`Server started on port ${PORT}`))
	} catch (error) {
		console.log(error)
	}
}

startServer()
```

Рисунок 3.1 – Ініціалізація Node.js проєкту

До екземпляру застосунку було додане проміжне програмне забезпечення для обробки запитів з використанням JSON формату. Код “app.use(express.json())” вказує Express.js використовувати вбудоване проміжне програмне забезпечення “express.json”, яке аналізує тіло запиту в форматі JSON та перетворює його в об’єкт – тип даних мови JavaScript. Це дозволяє легко отримувати доступ до даних, надісланих у вигляді JSON, в контролерах, які обробляють запити до серверного додатка.

В кореневій директорії проєкту було створено файл “.env”, який дозволяє зберігати конфіденційні змінні середовища. Зовнішній модуль “dotenv” дозволяє завантажити змінні середовища з файлу змінних оточення “.env” і використовувати їх у коді.

Було підключено Mongoose – це бібліотека для роботи з MongoDB у середовищі Node.js. Вона надає простий спосіб моделювання даних та взаємодію з базою даних MongoDB.

Наступним кроком відбувається підключення “errorHandler” – це кастомний middleware (функція, яка виконується до чи після обробників маршрутів та дозволяє виконати додаткові операції при обробці запитів) для обробки помилок

У функції яка запускає сервер, окрім прослухування запитів за допомогою методу екземпляру застосунку “listen” на певному порті, вказаному у файлі змінних оточення, також виконується підключення до бази даних MongoDB по URL вказаному у змінній оточення “DB_URL”.

3.1.2 Створення бази даних

Так як у якості бази даних використовується MongoDB, нею можна користуватися двома шляхами: перший – встановити локально, другий – за допомогою MongoDB Atlas у браузері.

Першим кроком є встановлення сервера бази даних MongoDB на вашому комп’ютері або на сервері. Тож встановлюємо MongoDB на сервер.

Після встановлення потрібно запустити сервер MongoDB. Це може відбуватися автоматично під час встановлення або можна запустити його вручну через командний рядок.

MongoDB не потребує явного створення бази даних перед її використанням. База даних буде створена автоматично, якщо почати використовувати колекції в цій базі даних.

Під час написання цього проєкту, використовувалася бібліотека Mongoose для Node.js, для підключення до бази даних MongoDB через URL. Ви використовуєте URL з'єднання з вашим сервером MongoDB, який включає ім'я користувача, пароль, IP-адресу та порт, якщо це необхідно.

Також були створені моделі для кожної сутності, з використанням Mongoose, щоб структурувати дані, що будуть використовуватися додатком для взаємодії з БД.

Далі будуть приведені моделі створених сутностей. Модель оголошення (рис. 3.2), модель новин (рис. 3.3), модель робочого місця (рис. 3.4), модель тарифу (рис. 3.5), модель користувача (рис. 3.6) та модель бронювання (рис. 3.7).

```
const mongoose = require('mongoose')

const adSchema = new mongoose.Schema({
	title: { type: String, required: true },
	content: { type: String, required: true },
	imageUrl: { type: String },
	createdAt: { type: Date, default: Date.now },
	updatedAt: { type: Date, default: Date.now }
})

adSchema.pre('findOneAndUpdate', function (next) {
	this.set({ updatedAt: new Date() })
	next()
})

const Ad = mongoose.model('Ad', adSchema)

module.exports = Ad
```

Рисунок 3.2 – Модель оголошення

```
const mongoose = require('mongoose')

const newsSchema = new mongoose.Schema({
	title: { type: String, required: true },
	content: { type: String, required: true },
	imageUrl: { type: String },
	createdAt: { type: Date, default: Date.now },
	updatedAt: { type: Date }
})

newsSchema.pre('findOneAndUpdate', function (next) {
	this.set({ updatedAt: new Date() })
	next()
})

const News = mongoose.model('News', newsSchema)

module.exports = News
```

Рисунок 3.3 – Модель новин

```
const mongoose = require('mongoose')

const workspaceSchema = new mongoose.Schema({
	name: { type: String, required: true },
	description: { type: String, required: true },
	images: [String],
	capacity: { type: Number, required: true },
	location: { type: String, required: true },
	coefficient: { type: Number, required: true },
	type: {
		type: String,
		enum: ['Office', 'Co-Working', 'Meeting Room', 'Event Space'],
		default: 'Office',
		required: true
	},
	status: {
		type: String,
		enum: ['occupied', 'available'],
		default: 'available'
	},
	amenities: {
		type: [String],
		enum: ['Wi-Fi', 'Coffee', 'Printer', 'Parking', 'Kitchen'],
		default: []
	}
})

const Workspace = mongoose.model('Workspace', workspaceSchema)
module.exports = Workspace
```

Рисунок 3.4 – Модель робочого місця

```
const mongoose = require('mongoose')

const tariffSchema = new mongoose.Schema({
	name: {
		type: String,
		required: true,
		unique: true
	},
	description: {
		type: String,
		required: true
	},
	price: {
		type: Number,
		required: true
	},
	duration: {
		type: String,
		enum: ['day', 'week', 'month', 'annual'],
		required: true
	},
	services: [String],
	available: {
		type: Boolean,
		default: true
	}
})

const Tariff = mongoose.model('Tariff', tariffSchema)
module.exports = Tariff
```

Рисунок 3.5 – Модель тарифу

```
const mongoose = require('mongoose')

const userSchema = new mongoose.Schema({
	email: { type: String, required: true, unique: true },
	password: { type: String, required: true },
	role: { type: String, enum: ['user', 'admin', 'creator'], default: 'user' },
	isActivated: { type: Boolean, default: false },
	activationLink: { type: String },
	firstName: { type: String },
	lastName: { type: String },
	gender: { type: String, enum: ['male', 'female', 'other'] },
	dateOfBirth: { type: Date },
	phoneNumber: { type: String },
	profilePicture: { type: String }
})

const User = mongoose.model('User', userSchema)
module.exports = User
```

Рисунок 3.6 – Модель користувача

```
const mongoose = require('mongoose')

const bookingSchema = new mongoose.Schema({
	userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true
},
	workspaceData: {
		type: mongoose.Schema.Types.ObjectId,
		ref: 'Workspace',
		required: true
	},
	tariffData: {
		type: mongoose.Schema.Types.ObjectId,
		ref: 'Tariff',
		required: true
	},
	status: {
		type: String,
		enum: ['active', 'completed', 'cancelled', 'pending'],
		default: 'pending'
	},
	startDate: { type: Date },
	endDate: { type: Date },
	paymentStatus: { type: String, enum: ['paid', 'unpaid'], default: 'unpaid' },
	totalPrice: { type: Number }
})

bookingSchema.pre('findOne', function (next) {
	this.populate('workspaceData')
	this.populate('tariffData')
	next()
})

const Booking = mongoose.model('Booking', bookingSchema)
module.exports = Booking
```

Рисунок 3.7 – Модель бронювання

3.1.3 Розробка модуля авторизації

Метою цього етапу є реалізація автентифікації, авторизації на серверній частині додатку, а також створити механізм для захисту маршрутів (middleware). Для реалізації автентифікації було використано JWT (Json Web Token) з парою токенів (access, refresh) та зчитуванням access-токена у заголовку запита “Authorization” для захисту запитів від неавтентифікованих користувачів. Для додаткової безпеки, використовується модуль “cookie-parser”, за допомогою якого виконується збереження та керування refresh-токеном користувача, також, встановлені флаг “httpOnly: true” для запобігання доступу до кукі через JavaScript, що зменшує ризик XSS-атаки.

Було створено user-service, у якому створено клас, який надає сервіси для управління користувачами в системі. Він включає функціонал для реєстрації, входу, виходу, активації облікового запису, оновлення токенів та призначення ролей (див. Додаток А.1).

Для реалізації JWT аутентифікації в Node.js було встановлено пакет “jsonwebtoken”, який надає можливість створювати та перевіряти JWT токени. Була написана функція для генерування JWT токену, яка задає вказане корисне навантаження, час дії токену та потребує ключа для шифрування підпису. Також була написана функція, яка за допомогою ключа перевіряє валідність підпису токена та повертає корисне навантаження, наявне у токені. Робота з JWT токеном була винесена у окремий сервіс.

Був створений та налаштований екземпляр класу Route з бібліотеки Express.js. Даний клас групує ендпоінти, що відносяться до модулю авторизації, та задає їм базовий початок шляху. Модуль авторизації містить 9 ендпоінтів: “register”, “login”, “logout”, “activate”, “getAllUsers”, “getUserById”, “updateUserById”, “deleteUserById”, “assignRoleUserById”. Перші чотири ендпоінти обробляються відповідним методом контролера авторизації, який в свою чергу звертається до сервісу авторизації, інші 5 методів призначені для керування користувачем або перегляду інформації та призначені для адміністратора.

Ендпоінт типу POST та шляхом «register» очікує в тілі запиту поля, що описують користувача, на основі яких формується та зберігається в базі даних новий об’єкт користувача (для реєстрації достатньо двох полів: пошта та пароль). Пароль користувача зберігається у базі даних у зашифрованому вигляді, шифрування було виконане за допомогою бібліотеки “bcrypt”.

Ендпоінт типу POST та шляхом «login» очікує в тілі запиту два поля – пошту та пароль користувача, які читає з тіла запиту контролер та передає у сервіс авторизації. Відповідний метод сервісу знаходить в базі даних користувача з вказаною поштою та за допомогою бібліотеки Bcrypt перевіряє пароль, отриманий з клієнта, з паролем, збереженим у базі даних. Якщо запис користувача знайдено у базі даних та надано правильний пароль, відбувається генерування JWT токена. Для цього було використано метод “sign” з пакету “jsonwebtoken”. У корисне навантаження токену був доданий ідентифікатор користувача. Access-токен було створено з часом життя в 30 хвилин. Також було згенеровано refresh-токен для оновлення access-токену після закінчення часу його дії. Пару токенів було відправлено у форматі JSON як відповідь на даний HTTP-запит.

Вихід користувача реалізований таким чином, що при виході з системи кукі з refreshToken очищуються, що завершує сесію користувача і запобігає подальшому використанню токена.

Ендпоінт типу GET та шляхом «refresh» очікує в тілі запиту виданий клієнту раніше refresh-токен та повертає у відповіді новий access-токен.

Було реалізовано перевірку параметрів запитів на валідність та відповідність вимогам. Якщо параметри не відповідають вимогам, запит повертає відповідний статус код та відповідь містить повідомлення з помилкою.

Для захисту ендпоінтів API від неавторизованих користувачів було створено middleware для перевірки авторизації користувача, а також у цьому ж middleware реалізовано перевірку на роль користувача (тому що деякі ендпоінти потребують певну роль користувача). Було захищено всі ендпоінти,

що відповідають за бізнес-логіку. Доступними без токена залишаються тільки роути, що відповідають за реєстрацію та логін. Захищені ендпоінти очікують access-токен з валідним підписом та актуальним часом життя у заголовку «Authorization».

3.1.4 Розробка API бізнес-логіки

На даному етапі створений серверний Node.js додаток був розширений реалізацією ендпоінтів, що реалізують бізнес-логіку системи.

При проєктуванні API було дотримано правил архітектурного стилю REST. Основні вимоги, які були реалізовані:

– додаток має клієнт-серверну архітектуру: система поділена на серверну частину та клієнтську;
– API є уніфікованим: були використані потрібні стандартні HTTP методи (GET, POST, PUT, DELETE), запити повертають потрібні статус коди та використовується єдиний формат даних.

У цьому проєкті були прописані routes файли з ендпоінтами окремо для кожної сутності. Всього було прописано моделі для шести сутностей, таких як, “ad” – оголошення, “news” – новини, “workspace” – робоче місце, “tariff” – тариф, “user” – користувач, “booking” – бронювання.

Перед тим як описувати всі ендпоінти, слід зауважити наступне, так як ендпоінти груповані за сутністю, а не за рівнем доступу, у файлі можуть бути ендпоінти, як для звичайного користувача, так і для адміністратора, у випадку з ендпоінтами адміністратора, до них додатково додані middleware для перевірки на авторизацію та роль.

Перший файл з ендпоінтами реалізований для сутності оголошення, він містить наступні ендпоінти: “createAd” – POST запит для створення нового оголошення, “getAllAds” – GET запит для отримання інформації про всі наявні оголошення (для їх відображення та перегляду), “getAdById” – GET Запит для

отримання інформації про певне оголошення по айді, “updateAdById” – PUT запит для редагування/оновлення даних оголошення, “deleteAdById” – DELETE запит для видалення оголошення по айді.

Другий файл з ендпоінтами реалізований для сутності новин, він дуже схожий на попередній та майже нічим не відрізняється та містить наступні ендпоінти: “createNews” – POST запит для створення новини, “getAllNews” – GET запит для отримання інформації про всі наявні новини (для їх відображення та перегляду), “getNewsById” – GET запит для отримання інформації про певну новину по айді, “updateNewsById” – PUT запит для редагування/оновлення даних новини, “deleteNewsById” – DELETE запит для видалення новини по айді.

Третій файл з ендпоінтами реалізований для сутності робочого місця, він містить наступні ендпоінти: “createWorkspace” – POST запит для створення нового робочого місця, “getAllWorkspaces” – GET запит для отримання інформації про всі наявні робочі місця (для їх відображення та перегляду), “getWorkspaceById” – GET запит для отримання інформації про певне робоче місце по айді, “updateWorkspaceById” – PUT запит для для редагування/оновлення даних робочого місця, “deleteWorkspaceById” – DELETE запит для видалення робочого місця по айді.

Четвертий файл з ендпоінтами реалізований для сутності тарифу, він містить наступні ендпоінти: “createTariff” – POST запит для створення нового тарифу, “getAllTariff” – GET запит для отримання інформації про всі наявні тарифи (для їх відображення та перегляду), “getTariffById” – GET запит для отримання інформації про певний тариф по айді, “updateTariffById” – PUT запит для редагування/оновлення даних тарифу, “deleteTariffById” – DELETE запит для видалення робочого місця по айді.

П’ятий файл з ендпоінтами є одним з найважливіших та реалізований для сутності користувача, він містить наступні ендпоінти: “register” – POST запит для реєстрації користувача, “login” – POST запит для авторизації користувача, “logout” – POST запит для виходу з облікового запису,

“activateUser” – GET запит для активації облікового запису користувача, “getAllUsers” – GET запит для отримання інформації про усіх користувачів, “getUserById” – GET запит для отримання інформації про певного користувача, “updateUserById” – PUT запит для редагування/оновлення інформації користувача (якщо опиратись на модель, то не всі поля можна редагувати), “deleteUserById” – DELETE запит для видалення користувача по айді та останній “assignRoleUserById” – PUT запит для призначення ролі користувача (підвищення ролі до адміністратора, заниження ролі до користувача), цей ендпоінт призначений тільки для людини, яка є головною у цій системи, вона має роль “creator” та має доступ до всіх ендпоінтів проєкту.

Фінальним шостим файлом є файл з ендпоінтами реалізованими для бронювання, він також є одним з найважливіших та має наступні ендпоінти: “createBooking” – POST запит для створення бронювання робочого місця, “getAllBookings” – GET запит для отримання інформації про всі бронювання, “viewBookingHistory” – GET запит для отримання історії бронювань користувача, “getBookingById” – GET запит для отримання певного бронювання по айді, “updateBookingById” – PUT запит для редагування даних бронювання, “cancelBooking” – PUT запит для скасування бронювання, “deleteBookingById – DELETE запит для видалення бронювання по айді.

При розробці API було використано Postman – інструмент, який дозволяє тестувати, документувати та взаємодіяти з API. Він надає корисні функції для створення та виконання запитів до API, тестування їх, перевірки відповідей та розробки API-документації. Для проєкту була створена колекція в Postman, куди було додано розроблені запити з їх очікуваними вхідними даними та прописані такі налаштування, як глобальні змінні і середовища.

3.2 Розробка клієнтської частини

Наступний етап розробки проєкту – розробка клієнтської частини. Очікуваний результат цього етапу – клієнт для вебплатформи, що є SPA

додатком, розробленим, використовуючи бібліотеку React та супутні їй пакети та інструменти, написаний, дотримуючись основних кращих практик написання інтерфейсів, реалізує функціонал описаної на етапі проєктування бізнес-логіки та має інтерфейс, що відповідає створеним макетам.

3.2.1 Ініціалізація та налаштування проєкту

Для початку розробки проєкту, потрібно виконати ініціалізацію та налаштування. Для цього спочатку слід переконатися, що на пристрої встановлено Node.js.

Так як збірка проєкту буде виконуватися засобами Vite, наступним кроком, потрібно відкрити термінал та виконати команду “npm create vite@latest” та обрати шаблон “react-ts”.

Далі, слід перейти до директорії проєкту та встановити необхідні залежності. У цьому випадку наступні залежності (рис. 3.8).

```
"dependencies": {
    "axios": "^1.6.8",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.23.1",
    "styled-components": "^6.1.11"
  },
  "devDependencies": {
    "@types/axios": "^0.14.0",
    "@types/react": "^18.2.66",
    "@types/react-dom": "^18.2.22",
    "@types/sass": "^1.45.0",
    "@typescript-eslint/eslint-plugin": "^7.2.0",
    "@typescript-eslint/parser": "^7.2.0",
    "@vitejs/plugin-react": "^4.2.1",
    "eslint": "^8.57.0",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.4.6",
    "sass": "^1.77.1",
    "typescript": "^5.2.2",
    "vite": "^5.2.0"
  }
```

Рисунок 3.8 – Перелік залежностей клієнту

Також можна виконати налаштування у конфігураційних файлах, якщо це необхідно.

3.2.2 Інтеграція з серверною частиною

Виконані налаштування axios для здійснення HTTP-запитів до API за базовою URL адресою “http://localhost:5000/api” (локальна адреса клієнту). Запити будуть включати кукі та заголовок Authorization з токеном з localStorage. Це забезпечує автоматичне додавання токену до кожного запиту для автентифікації користувача (рис. 3.9).

```
import axios from 'axios'

export const API_URL = 'http://localhost:5000/api'

const $api = axios.create({
	withCredentials: true,
	baseURL: API_URL
})

$api.interceptors.request.use(config => {
	config.headers.Authorization = `Bearer ${localStorage.getItem('token')}`
	return config
})

export default $api
```

Рисунок 3.9 – Налаштування axios

Далі були створені сервіси, щоб можна було надсилати запити по ендпоінтам. Як і у випадку з групуванням файлів з ендпоінтами на сервері, тут так само буде 6 сервісів, відповідно до кожної моделі сутності. А також для кожної сутності були створені typescript інтерфейси, для того щоб структурувати дані.

Перший сервіс – “AdService” для виконання запитів, які стосуються оголошень (див. Додаток А.3).

Другий сервіс – “NewsService” він майже не відрізняється від попереднього, та призначений для виконання запитів, що стосуються новин (див. Додаток А.4).

Третій сервіс – “WorkspaceService” для виконання запитів, які стосуються робочих місць (див. Додаток А.5).

Четвертий сервіс – “TariffService” для виконання запитів, які стосуються тарифів (див. Додаток А.6).

П’ятий сервіс – “UserService” для виконання запитів, які відносяться до користувача (див. Додаток А.2).

Та останній шостий сервіс – “BookingService” для виконання запитів, які відносяться до бронювання робочого місця (див. Додаток А.7).

3.2.3 Розробка модуля авторизації

На цьому етапі розробки клієнту було розроблено модуль авторизації клієнтського додатка: створені шаблони інтерфейсу, виконана інтеграція з серверною частиною, реалізований захист функцій, які цього потребують, від неавторизованих користувачів.

Коли користувач хоче увійти або зареєструватись, відкривається модальне вікно у якому можна це зробити (рис. 3.10, рис. 3.11), після натискання кнопки “зареєструватися” або “увійти” збираються усі введені дані та надсилається запит до API, який повертає згенерований JWT токен, у випадку валідності введених даних. Отриманий токен зберігається у кукі.

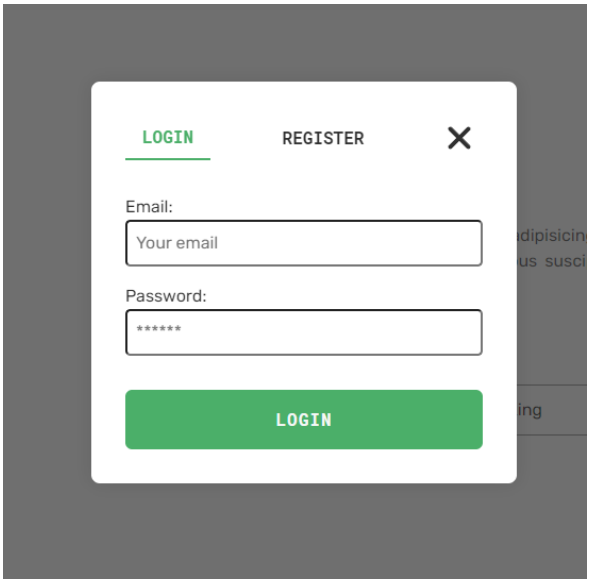

Рисунок 3.10 – Форма авторизації

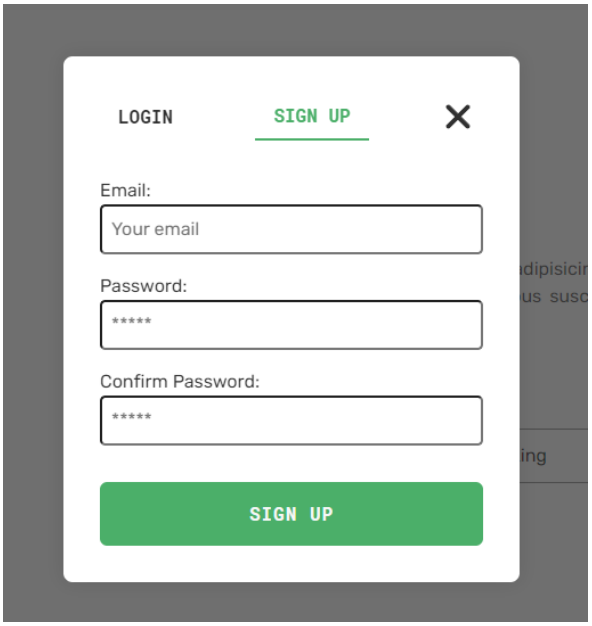

Рисунок 3.11 – Форма реєстрації

При першому завантаженні вебдодатка зчитується токен, та у разі його наявності, він зберігається в стан, що забезпечує неперервність сесії користувача при закритті вкладки браузера.

При намірі користувача вийти з системи, токен видаляється зі стейту додатка та з локального сховища браузера.

Всі запити до розробленого API, крім запитів на реєстрацію та логін у систему, очікують JWT токен для аутентифікації клієнта. Клієнт має передавати виданий сервером токен у заголовку Authorization – це HTTP заголовок, який використовується для передачі інформації про авторизацію користувача при виконанні HTTP-запитів.

Для додавання токену в заголовок Authorization запитів було розроблено інтерцептор за допомогою axios. При виконанні HTTP-запитів, які потребують аутентифікації клієнта, токен, збережений у сторі додатка, автоматично буде доданий заголовок Authorization з типом авторизації Bearer та токеном (рис. 3.12).

Всі маршрути клієнтського додатка, крім маршрутів модулю авторизації та деяких маршрутів, які дозволяють просто переглядати інформацію, були захищені від неавторизованих користувачів.

```
import axios from 'axios'

export const API_URL = 'http://localhost:5000/api'

const $api = axios.create({
  withCredentials: true,
  baseURL: API_URL
})

$api.interceptors.request.use(config => {
  config.headers.Authorization = `Bearer ${localStorage.getItem('token')}`
  return config
})

export default $api
```

Рисунок 3.12 – Інтерцептор для прикріплення заголовку авторизації

3.2.4 Розробка користувацького інтерфейсу

Після ініціалізації, налаштування проєкту та розробки модуля авторизації було виконано розробку основного функціоналу додатка.

Було розроблено користувацький інтерфейс відповідно до макетів та усіх інших вимог. Додаток має шість сторінок, чотири модальних вікна та два загальних елементи.

Першою чергою буде виконано опис загальних елементів, ними є: сайдбар з навігацією та налаштуваннями, невелике меню авторизації. Сайдбар містить у собі меню навігації, кнопку налаштувань, яка відкриває модальне вікно з налаштуваннями, логотип та назву проєкту, та підвал. Знаходиться сайдбар у лівій частині екрану та виглядає наступним чином (рис. 3.13).

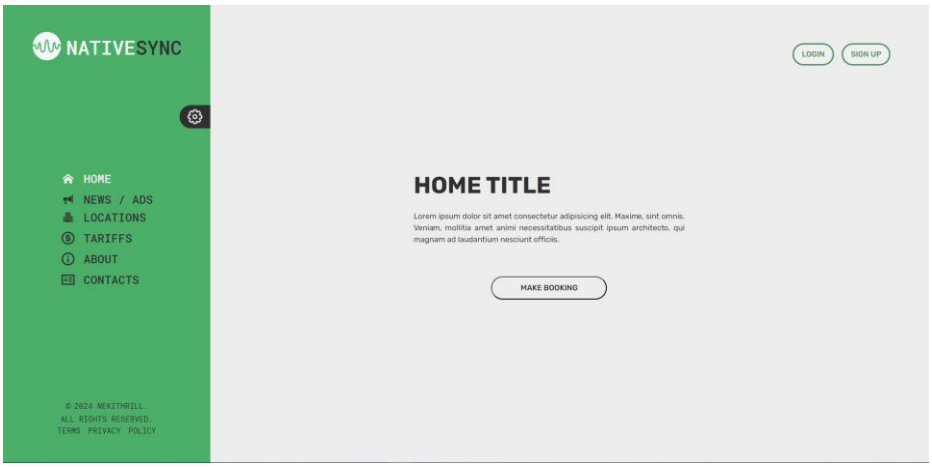

Рисунок 3.13 – Спільний елемент – сайдбар

Кнопка налаштувань знаходиться між елементами навігації та логотипом додатку, взаємодія з цією кнопкою призводить до відкриття модального вікна, у якому можна обрати тему та мову додатку та виглядає воно наступним чином (рис. 3.14).

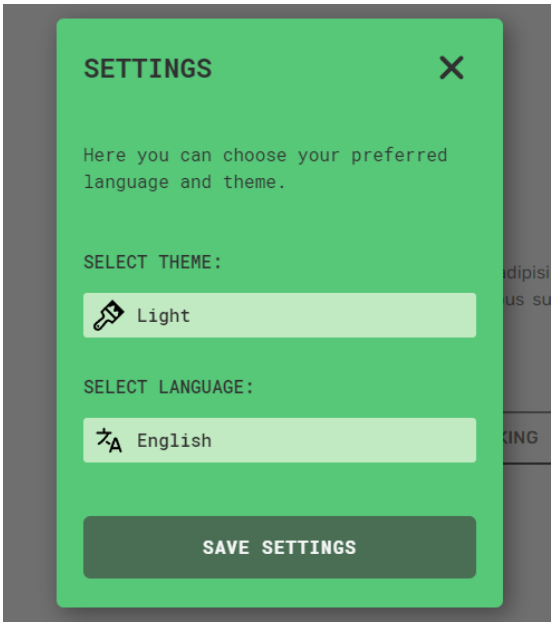

Рисунок 3.14 – Модальне вікно налаштувань

Наступним спільним елементом є невелике меню авторизації, яке складається з двох кнопок “Login” та “Sign Up”, взаємодія з якими приведе до спільного модального вікна реєстрації/авторизації (рис. 3.15, рис. 3.16). Кнопки знаходяться у верхньому правому куті та виглядають наступним чином (рис. 3.17).

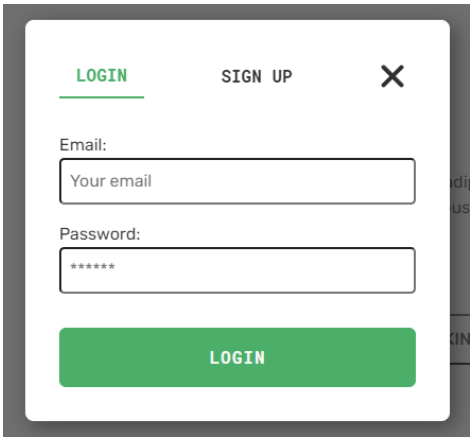

Рисунок 3.15 – Модальне вікно авторизації

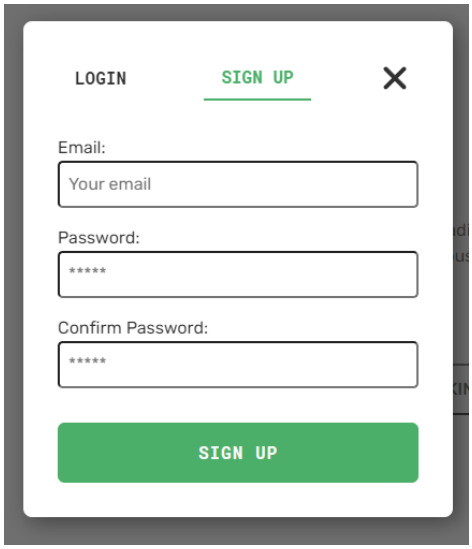

Рисунок 3.16 – Модальне вікно реєстрації

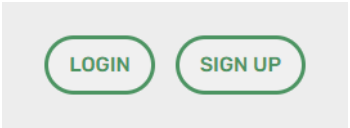

Рисунок 3.17 – Меню авторизації

Переходимо до опису сторінок, першою сторінкою є головна сторінка, яка містить заголовок, гасло та кнопку, яка дозволяє забронювати робоче місце (рис. 3.18).

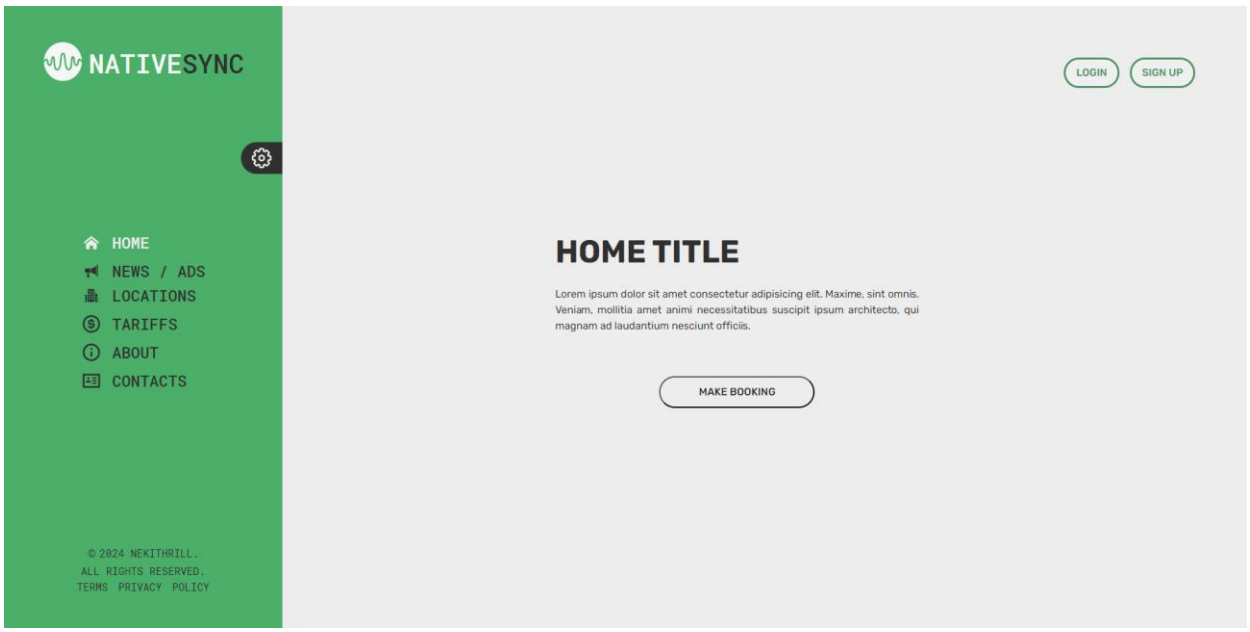

Рисунок 3.18

Модальне вікно бронювання робочого місця виглядає наступним чином (рис. 3.19).

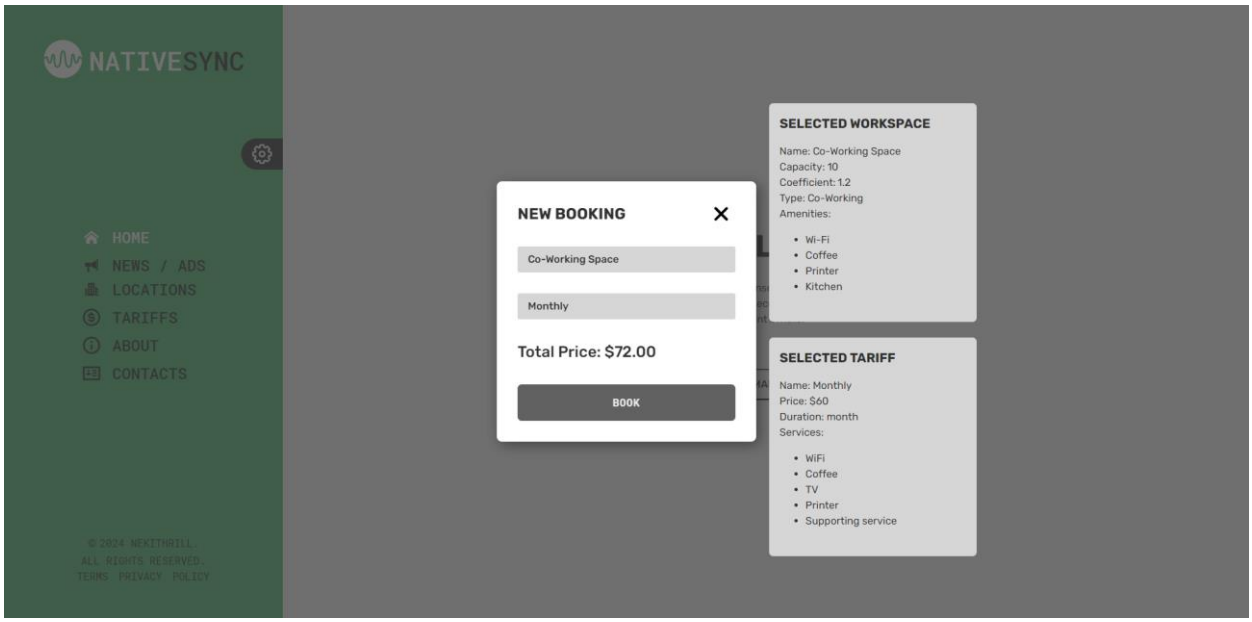

Рисунок 3.19 – Модальне вікно бронювання

Наступною сторінкою є сторінка новин та оголошень, вона містить дві вкладки “News” та “Ads”, які дозволяють перемикатися між потрібною інформацією, а також є фільтр за датою “New” та “Old” (рис. 3.20, рис. 3.21).

Рисунок 3.20 – Стрічка новин

Рисунок 3.21 – Стрічка оголошень

Наступною сторінкою є сторінка з локаціями (робочими місцями), вона містить стрічку з робочими місцями, а також перемикачем з наявними типами робочих місць, що дозволяє обрати потрібний тип робочого місця та отримати всі доступні робочі місця певного типу (рис. 3.22).

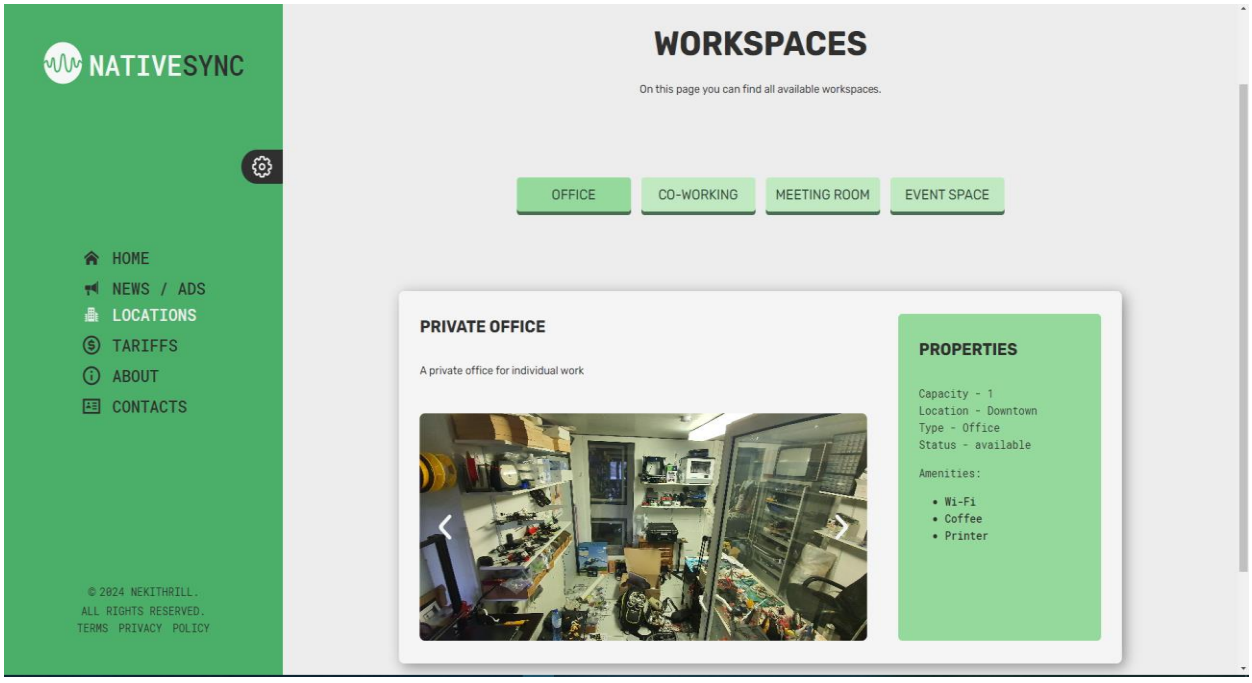

Рисунок 3.22 – Сторінка з робочими місцями

Стрічка містить картки з інформацією про робоче місце, сама картка містить наступну інформацію: назва, опис, галерея зображень, характеристики та зручності (рис. 3.23).

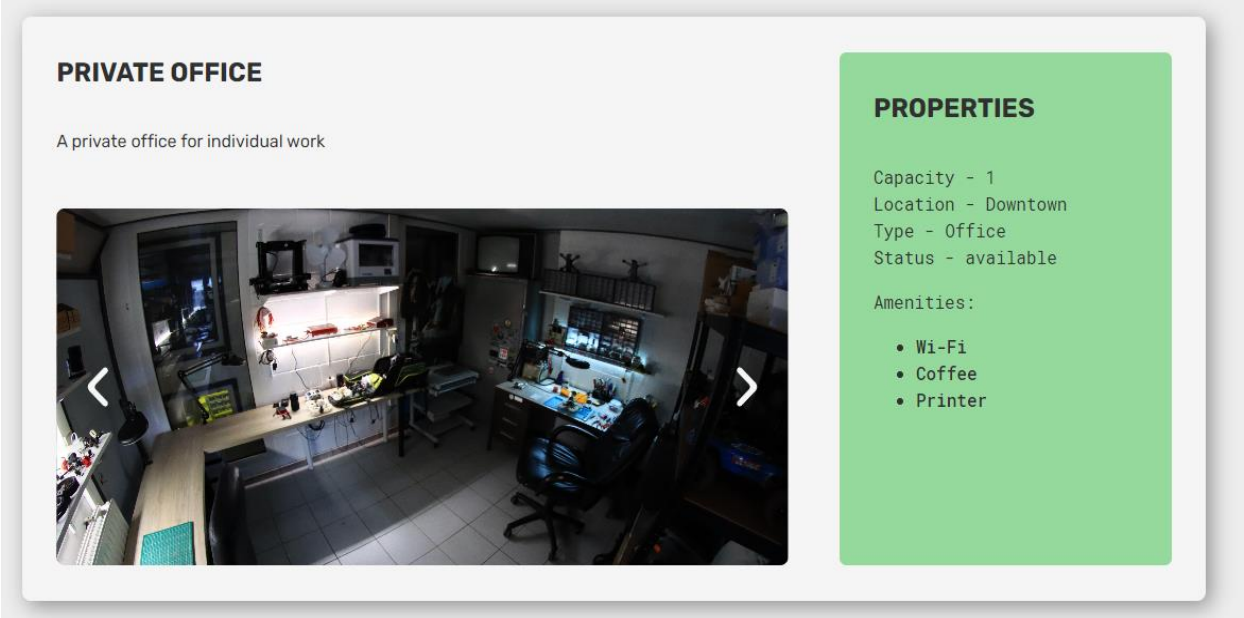

Рисунок 3.23 – Картка робочого місця

Далі йде сторінка з тарифами, яка містить картки з описом наявних тарифів (рис. 3.24, рис. 3.25).

Рисунок 3.24 – Сторінка з тарифами

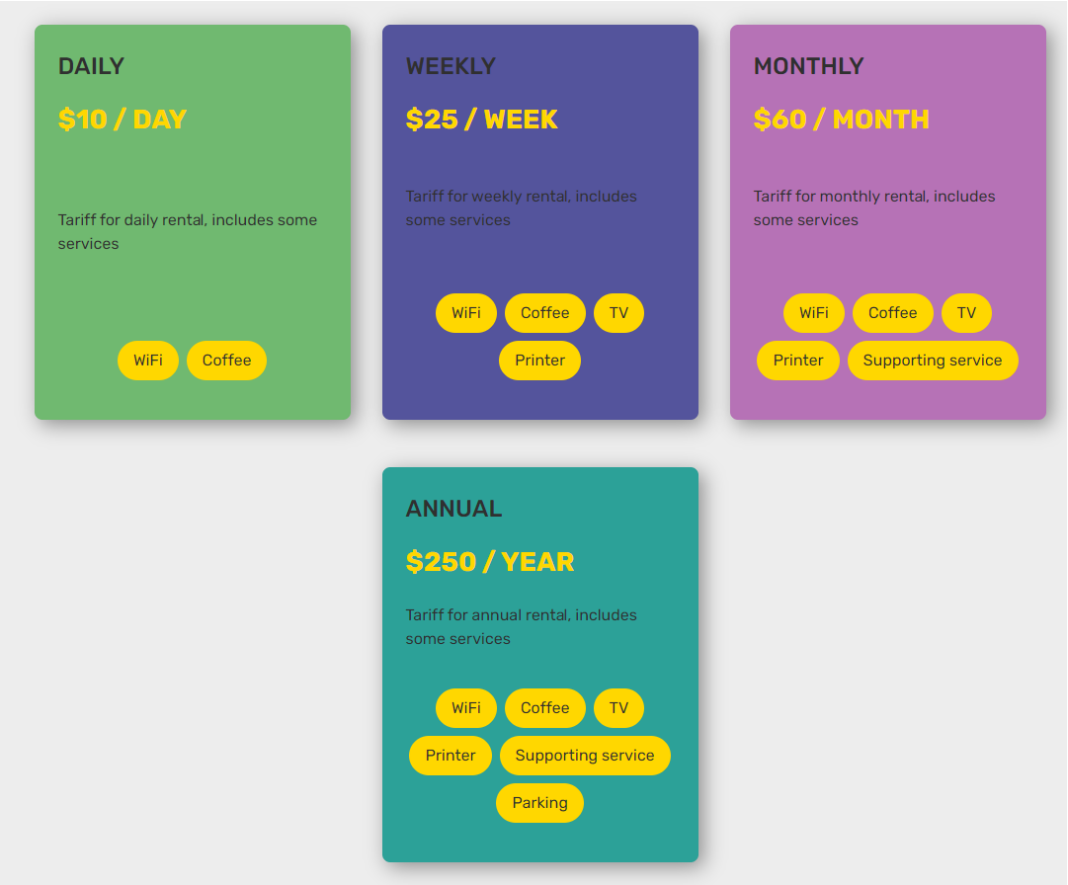

Рисунок 3.25 – Картки тарифів

Передостанньою сторінкою є сторінка з описом додатку та його перевагами, вона містить стрічку з картками (рис. 3.26, рис. 3.27).

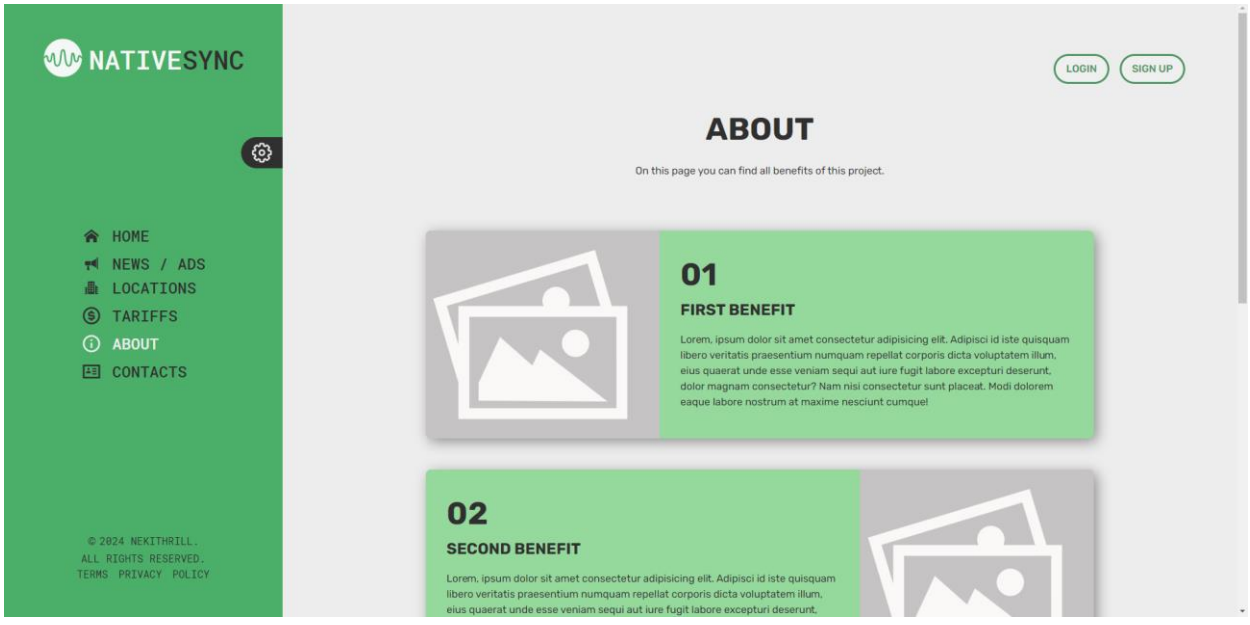

Рисунок 3.26 – Сторінка з описом та перевагами

Рисунок 3.27 – Продовження сторінки з описом та перевагами

Остання сторінка додатку – сторінка контактів, яка містить три блоки, перший містить контактні дані, другий містить форму для зворотного зв’язку та третій містить мапу з місцеположенням (рис. 3.28, рис. 3.29).

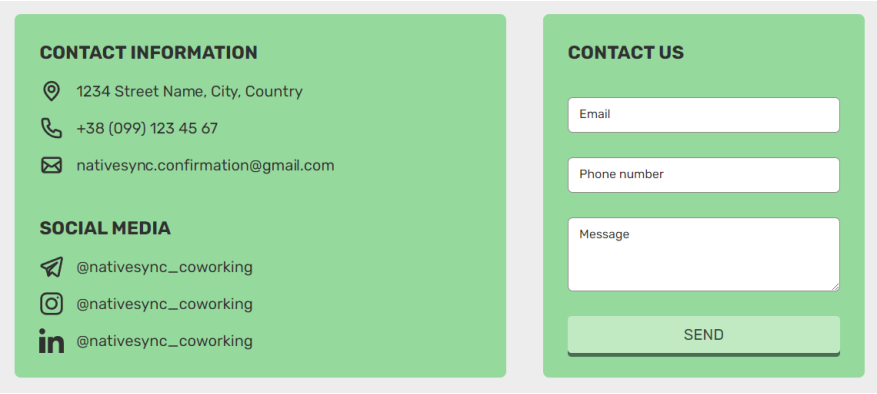

Рисунок 3.28 – Блок з контактними даними та формою зворотного зв’язку

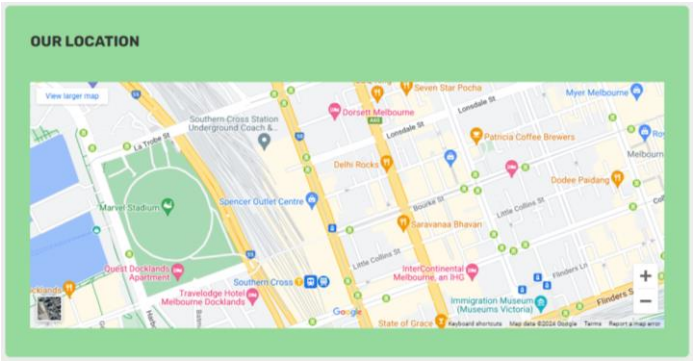

Рисунок 3.29 – Блок з мапою

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було створено програмну реалізацію вебдодатка для розміщення оголошень з пошуку та надання волонтерської допомоги.

Для розробки вебдодатка використовувався наступний стек технологій:

– платформа Node.js, мова JavaScript, фреймворк Express.js;
– мова TypeScript, бібліотека React та бандлер Vite;
– препроцесор SCSS;
– СКБД MongoDB.

У ході роботи були виконані такі завдання:

– проведення аналізу предметної області, визначення функціональності аналогічних систем;
– визначення мети розробки системи, складання технічного завдання та вимог до системи;
– розробка схеми API та створення макетів інтерфейсу;
– проєктування та створення бази даних, використовуючи СКБД MongoDB;
– розробка RESTful API, використовуючи платформу Node.js та фреймворк Express.js;
– розробка клієнтського додатка, використовуючи бібліотеку React;
– тестування додатка.

Розроблений додаток відповідає сформованому технічному завданню та технічним вимогам.

ПЕРЕЛІК ПОСИЛАНЬ

1. SPA у програмуванні. *Foxminded*. URL: https://foxminded.ua/spa-u-prohramuvanni/ (дата звернення: 03.03.2024).
2. Односторінкові (SPA) та багатосторінкові (PWA) додатки. *EMBO Studio*. URL: https://embo.com.ua/uk/blog/odnostorinkovi-spa-i-bagatostorinkovi-pwa/ (дата звернення: 03.03.2024).
3. Односторінковий додаток (SPA) vs багатосторінковий додаток (MPA): переваги та недоліки. *Merehead*. URL: https://merehead.com/ua/blog/spa-vs-mpa-application/ (дата звернення: 03.03.2024).
4. Express.js documentation. URL: https://expressjs.com/uk/ (дата звернення: 03.03.2024).
5. React documentation. URL: https://uk.legacy.reactjs.org/tutorial/tutorial.html (дата звернення: 07.03.2024).
6. Що таке React JS? Як почати вити React? *Cases media*. URL: https://cases.media/article/sho-take-react-js-yak-pochati-vivchati-reakt-navichki-dlya-react-developer (дата звернення: 12.03.2024).
7. What Is MongoDB? Working, Architecture, Features, and Use Cases. *Spiceworks*. URL: https://www.spiceworks.com/tech/cloud/articles/what-is-mongodb/ (дата звернення: 14.03.2024).
8. What is MongoDB? *Techtarget*. URL: https://www.techtarget.com/searchdatamanagement/definition/MongoDB (дата звернення: 17.03.2024).
9. Все, що потрібно знати про бази даних для початківців: MySQL, PostgreSQL, MongoDB. *Dan-it*. URL: https://dan-it.com.ua/uk/blog/vse-shho-potribno-znati-pro-bazi-danih-dlja-pochatkivciv-mysql-postgresql-mongodb/ (дата звернення: 20.03.2024).
10. MongoDB. *Brander*. URL: https://brander.ua/technologies/mongo-db (дата звернення: 23.03.2024).

11. Як будувати UML-діаграми? *Dou*. URL: https://dou.ua/forums/topic/40575/ (дата звернення: 25.03.2024).

ДОДАТОК А

Сервіси вебдодатку

А.1 Сервіс user-service для керування користувачем (на сервері)

```
const UserModel = require('../models/user.model')
const bcrypt = require('bcrypt')
const uuid = require('uuid')
const mailService = require('./mail-service')
const tokenService = require('./token-service')
const UserDto = require('../dtos/user.dto')
const ApiError = require('../errors/api-error')

class UserService {
	async registration(email, password) {
		const candidate = await UserModel.findOne({ email })
		if (candidate) {
			throw ApiError.BadRequestError(
				`User with email - ${email} already exists.`
			)
		}

		const role = 'user'
		const hashPassword = await bcrypt.hash(password, 3)
		const activationLink = uuid.v4()

		const user = await UserModel.create({
			email,
			password: hashPassword,
			role,
			activationLink
		})
		await mailService.sendActivationMail(
			email,
			`${process.env.API_URL}/api/user/activate/${activationLink}`
		)

		const userDto = new UserDto(user)
		const tokens = tokenService.generateTokens({ ...userDto })
		await tokenService.saveToken(userDto.id, tokens.refreshToken)

		return {
			...tokens,
			user: userDto
		}
	}

	async login(email, password) {
		const user = await UserModel.findOne({ email })
		if (!user) {
			throw ApiError.BadRequestError(`User with email - ${email} doesnt
exist.`)
		}
		const isPassEquals = await bcrypt.compare(password, user.password)
```

```
            if (!isPassEquals) {
                throw ApiError.badRequestError('Invalid password.')
            }
            const userDto = new UserDto(user)
            const tokens = tokenService.generateTokens({ ...userDto })

            await tokenService.saveToken(userDto.id, tokens.refreshToken)

            return {
                ...tokens,
                user: userDto
            }
        }

        async logout(refreshToken) {
            const token = await tokenService.removeToken(refreshToken)
            return token
        }

        async activate(activationLink) {
            const user = await UserModel.findOne({ activationLink })
            if (!user) {
                throw ApiError.BadRequestError('Wrong activation link')
            }
            user.isActivated = true
            await user.save()
        }

        async refresh(refreshToken) {
            if (!refreshToken) {
                throw ApiError.UnauthorizedError()
            }
            const userData = tokenService.validateRefreshToken(refreshToken)
            const tokenFromDb = await tokenService.findToken(refreshToken)
            if (!userData || !tokenFromDb) {
                throw ApiError.UnauthorizedError()
            }
            const user = await UserModel.findById(userData.id)
            const userDto = new UserDto(user)
            const tokens = tokenService.generateTokens({ ...userDto })

            await tokenService.saveToken(userDto.id, tokens.refreshToken)
            return { ...tokens, user: userDto }
        }

        async assignRole(userId, newRole) {
            const user = await UserModel.findById(userId)

            if (!user) {
                throw ApiError.NotFoundError('User not found')
            }

            user.role = newRole
            await user.save()

            return user
        }
}

module.exports = new UserService()
```

A.2 Сервіс UserService для керування користувачем (на клієнті)

```
import { AxiosResponse } from 'axios'
import $api from '../http'
import { IUser } from '../models/entities/IUser'
import { AuthResponse } from '../models/responses/UserResponse'

export default class UserService {
	static async register(
		email: string,
		password: string
	): Promise<AxiosResponse<AuthResponse>> {
		return $api.post<AuthResponse>('/user/registration', { email, password
})
	}

	static async login(
		email: string,
		password: string
	): Promise<AxiosResponse<AuthResponse>> {
		return $api.post<AuthResponse>('/user/login', { email, password })
	}

	static async logout(): Promise<void> {
		return $api.post('/user/logout')
	}

	static getAllUsers(): Promise<AxiosResponse<IUser[]>> {
		return $api.get<IUser[]>('/user/all')
	}

	static getUserById(id: string): Promise<AxiosResponse<IUser>> {
		return $api.get<IUser>(`/user/${id}`)
	}

	static activateUser(activationLink: string): Promise<AxiosResponse<void>> {
		return $api.get<void>(`/user/activate/${activationLink}`)
	}

	static updateUserById(
		id: string,
		userData: Partial<IUser>
	): Promise<AxiosResponse<IUser>> {
		return $api.put<IUser>(`/user/edit/${id}`, userData)
	}

	static assingRoleUserById(
		id: string,
		role: string
	): Promise<AxiosResponse<IUser>> {
		return $api.put<IUser>(`/user/role/${id}/${role}`)
	}

	static deleteUserById(id: string): Promise<AxiosResponse<void>> {
		return $api.delete<void>(`/user/delete/${id}`)
	}
}
```

А.3 Сервіс AdService для керування оголошеннями

```
import { AxiosResponse } from 'axios'
import $api from '../http'
import { INewsAndAds } from '../models/entities/INewsAndAds'

export default class AdService {
	static async createAd(
		title: string,
		content: string,
		adData: Partial<INewsAndAds>
	): Promise<AxiosResponse<INewsAndAds>> {
		const adPayload = {
			title,
			content,
			...adData
		}
		return $api.post<INewsAndAds>(`/ad/create`, adPayload)
	}

	static async getAllAds(): Promise<AxiosResponse<INewsAndAds[]>> {
		return $api.get<INewsAndAds[]>(`/ad/all`)
	}

	static async getAdById(adId: string): Promise<AxiosResponse<INewsAndAds>> {
		return $api.get<INewsAndAds>(`/ad/${adId}`)
	}

	static async updateAdById(
		adId: string,
		adData: Partial<INewsAndAds>
	): Promise<AxiosResponse<INewsAndAds>> {
		return $api.put<INewsAndAds>(`/ad/edit/${adId}`, adData)
	}

	static async deleteAdById(adId: string): Promise<AxiosResponse<void>> {
		return $api.delete<void>(`/ad/delete/${adId}`)
	}
}
```

А.4 Сервіс NewsService для керування новинами

```
import { AxiosResponse } from 'axios'
import $api from '../http'
import { INewsAndAds } from '../models/entities/INewsAndAds'

export default class NewsService {
	static async createnews(
		title: string,
		content: string,
		newsData: Partial<INewsAndAds>
	): Promise<AxiosResponse<INewsAndAds>> {
		const newsPaylonews = {
			title,
```

```
                content,
                ...newsData
        }
        return $api.post<INewsAndAds>(`/news/create`, newsPaylonews)
    }

    static async getAllNews(): Promise<AxiosResponse<INewsAndAds[]>> {
        return $api.get<INewsAndAds[]>(`/news/all`)
    }

    static async getnewsById(
        newsId: string
    ): Promise<AxiosResponse<INewsAndAds>> {
        return $api.get<INewsAndAds>(`/news/${newsId}`)
    }

    static async updatenewsById(
        newsId: string,
        newsData: Partial<INewsAndAds>
    ): Promise<AxiosResponse<INewsAndAds>> {
        return $api.put<INewsAndAds>(`/news/edit/${newsId}`, newsData)
    }

    static async deletenewsById(newsId: string): Promise<AxiosResponse<void>> {
        return $api.delete<void>(`/news/delete/${newsId}`)
    }
}
```

A.5 Сервіс WorkspaceService для керування робочими місцями

```
import { AxiosResponse } from 'axios'
import $api from '../http'
import { IWorkspace } from '../models/entities/IWorkspace'

export default class WorkspaceService {
    static async createWorkspace(
        name: string,
        description: string,
        capacity: number,
        location: string,
        coefficient: number,
        type: 'Office' | 'Co-Working' | 'Meeting Room' | 'Event Space',
        workspaceData: Partial<IWorkspace> = {}
    ): Promise<AxiosResponse<IWorkspace>> {
        const workspacePayload: Partial<IWorkspace> = {
            name,
            description,
            capacity,
            location,
            coefficient,
            type,
            ...workspaceData
        }

        return $api.post<IWorkspace>('/workspace/create', workspacePayload)
    }
```

```
	static async getAllWorkspaces(): Promise<AxiosResponse<IWorkspace[]>> {
		return $api.get<IWorkspace[]>('/workspace/all')
	}

	static async getWorkspaceById(
		workspaceId: string
	): Promise<AxiosResponse<IWorkspace>> {
		return $api.get<IWorkspace>(`/workspace/${workspaceId}`)
	}

	static async updateWorkspaceById(
		workspaceId: string,
		workspaceData: Partial<IWorkspace>
	): Promise<AxiosResponse<IWorkspace>> {
		return $api.put<IWorkspace>(`/workspace/edit/${workspaceId}`,
workspaceData)
	}

	static async deleteWorkspaceById(
		workspaceId: string
	): Promise<AxiosResponse<void>> {
		return $api.delete<void>(`/workspace/delete/${workspaceId}`)
	}
}
```

A.6 Сервіс TariffService для керування тарифами

```
import { AxiosResponse } from 'axios'
import $api from '../http'
import { ITariff } from '../models/entities/ITariff'

export default class TariffService {
	static async createTariff(
		name: string,
		description: string,
		price: number,
		duration: 'daily' | 'weekly' | 'monthly',
		tariffData?: Partial<ITariff>
	): Promise<AxiosResponse<ITariff>> {
		const tariffPayload = {
			name,
			description,
			price,
			duration,
			...tariffData
		}
		return $api.post<ITariff>('/tariff/create', tariffPayload)
	}

	static async getAllTariffs(): Promise<AxiosResponse<ITariff[]>> {
		return $api.get<ITariff[]>('/tariff/all')
	}

	static async getTariffById(
		tariffId: string
```

```
	): Promise<AxiosResponse<ITariff>> {
		return $api.get<ITariff>(`/tariff/${tariffId}`)
	}

	static async updateTariffById(
		tariffId: string,
		tariffData: Partial<ITariff>
	): Promise<AxiosResponse<ITariff>> {
		return $api.put<ITariff>(`/tariff/edit/${tariffId}`, tariffData)
	}

	static async deleteTariffById(
		tariffId: string
	): Promise<AxiosResponse<void>> {
		return $api.delete<void>(`/tariff/delete/${tariffId}`)
	}
}
```

A.7 Сервіс BookingService для керування бронюваннями

```
import { AxiosResponse } from 'axios'
import $api from '../http'
import { IBooking } from '../models/entities/IBooking'

export default class BookingService {
	static async createBooking(
		userId: string,
		workspaceId: string,
		tariffId: string,
		bookingData?: Partial<IBooking>
	): Promise<AxiosResponse<IBooking>> {
		return $api.post<IBooking>('/booking/create', {
			userId,
			workspaceId,
			tariffId,
			...bookingData
		})
	}

	static async getAllBookings(): Promise<AxiosResponse<IBooking[]>> {
		return $api.get<IBooking[]>('/booking/all')
	}

	static async viewBookingHistory(): Promise<AxiosResponse<IBooking[]>> {
		return $api.get<IBooking[]>('/booking/history')
	}

	static async getBookingById(
		bookingId: string
	): Promise<AxiosResponse<IBooking>> {
		return $api.get<IBooking>(`/booking/${bookingId}`)
	}

	static async updateBookingById(
		bookingId: string,
		bookingData: Partial<IBooking>
```

```
        ): Promise<AxiosResponse<IBooking>> {
                return $api.put<IBooking>(`/booking/edit/${bookingId}`, bookingData)
        }

        static async cancelBooking(
                bookingId: string
        ): Promise<AxiosResponse<{ message: string; booking?: IBooking }>> {
                return $api.put<{ message: string; booking?: IBooking }>(
                        `/booking/cancel/${bookingId}`
                )
        }

        static async deleteBookingById(
                bookingId: string
        ): Promise<AxiosResponse<{ message: string }>> {
                return $api.delete<{ message: string }>(`/booking/delete/${bookingId}`)
        }
}
```