МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА САЙТУ ДЛЯ ОРГАНІЗАЦІЇ ТА ПРОВЕДЕННЯ ЕКСКУРСІЙ»**

Виконала: студентка 4 курсу, групи 6.1211-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Л.О. Алабіна
(ініціали та прізвище)

Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

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

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Алабіній Лілії Олександрівні
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка сайту для організації та проведення екскурсій

керівник роботи Лісняк Андрій Олександрович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 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. Проєктування архітектури вебзастосунку.
3. Реалізація програмного забезпечення.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 26.12.2024 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.	20.01.2025	
2.	Збір вихідних даних.	10.02.2025	
3.	Обробка методичних та теоретичних	14.03.2025	
	джерел.		
4.	Розробка першого та другого розділу.	18.04.2025	
5.	Розробка третього розділу.	19.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	16.06.2025	

Студент _____ Л.О. Алабіна
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка сайту для організації та проведення екскурсій»: 56 с., 19 рис., 2 табл., 16 джерел.

БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ЕКСКУРСІЇ, ОРГАНІЗАТОРИ ТА УЧАСНИКИ, ПРОГРАМНА ІНЖЕНЕРІЯ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, СЕРВІС.

Об’єкт дослідження – процес взаємодії між організаторами екскурсій та учасниками в рамках туристичних і культурних заходів.

Предмет дослідження – комплекс теоретичних, методичних та практичних рекомендацій щодо розробки програмного забезпечення для організації та координації екскурсій.

Мета роботи – розробити вебсайт для організації та проведення екскурсій: систему, яка дозволить організаторам публікувати заходи, а користувачам – реєструватися, отримувати інформацію та брати участь в екскурсіях.

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

Під час розробки системи було проведено аналіз предметної області. Розроблено план дій за допомогою діаграми Ганта. Обрано та спроєктовано архітектуру вебзастосунку із використанням UML: діаграм прецедентів, класів, активностей, а також ER-діаграми та діаграм eEPC і BPMN. Серверну частину реалізовано з використанням Spring. Клієнтську частину створено за допомогою HTML, CSS, JavaScript і шаблонного рушія Thymeleaf.

У результаті виконаної роботи створено програмне забезпечення для ефективної організації, публікації та управління екскурсіями між організаторами та відвідувачами.

SUMMARY

Bachelor’s qualifying paper “Development of a Website for Organizing and Conducting Excursions”: 56 pages, 19 figures, 2 tables, 16 references.

DATABASE, EXCURSIONS, ORGANIZERS AND PARTICIPANTS, WEB APPLICATION, SOFTWARE ENGINEERING, SOFTWARE DEVELOPMENT, SERVICE.

The object of the study is the interaction between tour organizers and participants during travel and cultural events.

The subject of the study is a set of theoretical, methodological, and practical recommendations for developing software to organize and coordinate tours.

The aim of the study is to develop a website for organizing and conducting tours a system that allows organizers to publish events and users to register, receive information, and take part in tours.

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

As part of the development process, the subject area was analyzed, and a plan was created using a Gantt chart. The architecture of the web application was designed using UML diagrams (use case, class, activity), as well as ER, eEPC, and BPMN diagrams. The backend was implemented using Spring, and the frontend was built with HTML, CSS, JavaScript, and the Thymeleaf template engine.

As a result, software was created to help organizers and visitors efficiently plan, share, and manage tours.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 7
1 Огляд теоретичних властивостей та планування проєкту 9
1.1 Огляд існуючих вебсервісів для організації екскурсій 9
1.2 Аналіз та обґрунтування вибору технологій фронтенду 11
1.3 Аналіз та обґрунтування вибору технологій бекенду 14
1.4 Аналіз функціональних та нефункціональних вимог 16
1.5 Висновки до розділу 1 19
2 Проєктування системи організації та проведення екскурсій 21
2.1 Концептуальне проєктування інтерфейсу 21
2.2 Проєктування фронтендної частини 26
2.3 Проєктування серверної частини 33
2.4 Висновки до розділу 2 34
3 Реалізація та тестування 36
3.1 Реалізація фронтендної частини 36
3.2 Реалізація серверної частини 41
3.3 Налаштування бази даних 45
3.4 Тестування вебзастосунку 46
3.5 Висновки до розділу 3 52
Висновки 53
Перелік посилань..... 55

ВСТУП

У сучасному світі цифрові технології відіграють ключову роль у трансформації способів організації дозвілля, подорожей і подій. Однією з актуальних потреб є створення зручних онлайн-інструментів для організації та управління екскурсіями.

Відсутність єдиної універсальної платформи для замовлення, планування та керування екскурсіями створює труднощі як для організаторів, так і для потенційних учасників заходів. У цьому контексті створення вебсайту для організації та проведення екскурсій є актуальним і практично значущим завданням, яке дозволить автоматизувати багато процесів, пов’язаних із реєстрацією, комунікацією та керуванням подіями.

Мета роботи – розробити вебзастосунок для організації екскурсій, який дозволяє створювати екскурсії, керувати розкладом, реєструвати учасників і забезпечувати зручний зворотний зв’язок.

Досягнення поставленої мети передбачає виконання наступних завдань:

– дослідити існуючі рішення для управління екскурсіями та подіями;
– визначити основні функціональні вимоги до системи;
– обґрунтувати вибір технологій: React – для створення зручного інтерфейсу, Node.js – для побудови API, SQLite – для зберігання даних;
– спроєктувати архітектуру вебзастосунку та структуру бази даних;
– реалізувати клієнтську та серверну частини системи;
– протестувати функціональність системи на відповідність вимогам.

Об’єкт дослідження – процес організації та управління екскурсіями в онлайн-середовищі.

Предмет дослідження – методи, засоби та принципи створення вебсайту для організації та проведення екскурсій, що забезпечує інтерактивність, зручність та ефективність взаємодії користувачів.

Методи дослідження – методи аналізу вимог, моделювання систем, проєктування архітектури програмного забезпечення, розробки та тестування вебзастосунків.

Для реалізації проєкту було обрано стек технологій, який поєднує гнучкість і простоту. React дозволяє створювати динамічний клієнтський інтерфейс зі зручною навігацією та обробкою подій. Node.js у поєднанні з Express забезпечує швидку й ефективну обробку запитів на сервері та реалізацію REST API. Для зберігання даних про екскурсії, користувачів та заявки використовується SQLite – легка реляційна база даних, що не потребує окремого сервера.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та переліку посилань. Загальний обсяг роботи становить 56 сторінок. У переліку літератури використано 16 джерел.

У першому розділі подано огляд існуючих рішень для організації заходів, а також детальний аналіз вибраних технологій для реалізації клієнтської та серверної частин застосунку.

Другий розділ присвячений проєктуванню системи: формалізації вимог, побудові архітектури, моделюванню бази даних, створенню UML-діаграм, а також опису основного функціоналу для користувачів (організаторів та учасників).

У третьому розділі описано реалізацію застосунку – розробку REST API, побудову інтерфейсів у React, взаємодію з базою даних, авторизацію користувачів, а також механізм перегляду майбутніх і завершених екскурсій. Особливу увагу приділено створенню сторінки «Історія екскурсій», яка дозволяє зберігати архів подій та переглядати їх учасникам і організаторам.

1 ОГЛЯД ТЕОРЕТИЧНИХ ВЛАСТИВОСТЕЙ ТА ПЛАНУВАННЯ ПРОЄКТУ

1.1 Огляд існуючих вебсервісів для організації екскурсій

У сучасному туристичному середовищі онлайн-сервіси для бронювання та організації екскурсій набули широкого поширення. Вони надають зручні інструменти як для мандрівників, так і для організаторів, дозволяючи планувати маршрути, здійснювати оплату та комунікувати з гідами або туристичними компаніями. Давайте зробимо короткий огляд найбільш популярних платформ, що працюють у цій галузі.

GetYourGuide – провідна глобальна онлайн-платформа для бронювання туристичних екскурсій, заходів і атракцій, яка функціонує з 2009 року. Сервіс об’єднує широкий спектр пропозицій від локальних організаторів по всьому світу, охоплюючи понад 100 країн і тисячі напрямків. Основна мета платформи – зробити процес планування подорожей та організації дозвілля максимально простим, швидким і зручним як для мандрівників, так і для постачальників послуг [15].

GetYourGuide – це потужна та зручна платформа для туристів, які шукають екскурсії та розваги у різних куточках світу. Однак, для організаторів екскурсій, особливо локальних, потреба у більш гнучких і повноцінних інструментах управління залишається відкритою. Розробка власного вебзастосунку з функціоналом, адаптованим до специфіки організації екскурсій, дозволить створити ефективний сервіс, який буде враховувати усі особливості процесу від створення події до збору відгуків і аналізу статистики.

Tripster – це українська онлайн-платформа для організації та бронювання екскурсій, турів і подорожей. Заснована з метою популяризації локального туризму, вона пропонує широкий вибір авторських екскурсій від місцевих гідів по Україні та за її межами. Tripster акцентує увагу на унікальних, нестандартних маршрутах і культурних враженнях [16].

Tripster використовує сучасний технологічний стек, який забезпечує високу продуктивність, надійність та зручність використання сервісу. Для фронтенд-розробки застосовується React, що дозволяє створювати інтерактивний та адаптивний інтерфейс користувача. Серверна частина реалізована на Node.js, що забезпечує ефективну обробку запитів і масштабованість системи. Для зберігання даних використовується реляційна база даних PostgreSQL, що гарантує цілісність і надійність інформації.

Для хостингу та обробки даних застосовуються хмарні сервіси, зокрема Google Cloud Platform (GCP), що дозволяє забезпечити безперебійну роботу сервісу при великих навантаженнях. Дані користувачів та інформація про бронювання захищені відповідно до сучасних стандартів безпеки, з використанням шифрування і багаторівневих механізмів аутентифікації [15].

Tiqets – це міжнародна онлайн-платформа для миттєвого бронювання квитків до музеїв, культурних та розважальних закладів. Заснована в 2014 році в Амстердамі, компанія прагне зробити культуру доступнішою, надаючи туристам можливість швидко та зручно бронювати квитки через мобільні додатки або вебсайт.

Tiqets використовує сучасний технологічний стек, який забезпечує високу продуктивність, надійність та зручність користування сервісом. Для фронтенд-розробки застосовується AngularJS, що дозволяє створювати швидкий та інтерактивний інтерфейс користувача.

Серверна частина реалізована з використанням сучасних технологій, що забезпечують ефективну обробку запитів та масштабованість системи. Для зберігання даних використовується реляційна база даних, яка гарантує цілісність та безпеку інформації.

Для хостингу та обробки даних Tiqets використовує хмарні сервіси, що забезпечують стабільну роботу платформи навіть при великих навантаженнях.

Tiqets є потужною платформою для миттєвого бронювання квитків до музеїв та атракцій, що забезпечує зручність та доступність для туристів. Однак для організації та проведення екскурсій, особливо локального масштабу, може

бути доцільно розглянути розробку спеціалізованого сервісу, який би краще відповідав специфічним потребам організаторів та учасників [16].

1.2 Аналіз та обґрунтування вибору технологій фронтенду

При розробці фронтенд частини програмного забезпечення для платформи з організації та проведення екскурсій, вибір бібліотеки чи фреймворку має вирішальне значення для забезпечення зручності використання, масштабованості та стабільної роботи системи. Враховуючи вимоги проєкту до інтерфейсу – динамічність, інтерактивність, адаптивність – було обрано React як основну технологію розробки клієнтської частини вебзастосунку [1].

React має низку суттєвих переваг, які роблять його особливо зручним для розробки сучасних вебзастосунків. Однією з ключових особливостей є компонентно-орієнтований підхід. Це означає, що інтерфейс можна розбити на незалежні логічні частини – компоненти, кожен з яких відповідає за певну функцію або елемент. Такий підхід забезпечує зручність повторного використання коду, спрощує тестування та полегшує підтримку і масштабування системи.

Ще однією важливою перевагою є використання віртуального DOM. Це спеціальний механізм, який дозволяє оновлювати лише ті частини інтерфейсу, що зазнали змін, не перезавантажуючи повністю сторінку. Завдяки цьому досягається висока продуктивність додатку, що особливо важливо під час роботи з великими масивами інформації, як-от списки екскурсій або історія бронювань.

React також забезпечує однаковий підхід до розробки як веб, так і мобільних застосунків. Завдяки бібліотеці React Native можна розробляти мобільні версії системи, використовуючи ті самі концепції та навіть частково повторно використовуючи код. Це дає змогу скоротити час і витрати на розробку.

Гнучкість React полягає в тому, що це не фреймворк, а саме бібліотека для створення інтерфейсу. Розробник самостійно обирає інструменти, які доповнюють роботу з React – наприклад, для маршрутизації, керування станом або обробки форм. Це дозволяє адаптувати рішення під конкретні потреби проєкту.

Окрему увагу заслуговує активна спільнота React. Бібліотека постійно оновлюється завдяки підтримці з боку компанії Meta (Facebook) і великої кількості розробників по всьому світу. Це гарантує стабільність, актуальність технології та наявність великої кількості готових рішень і документації.

Незважаючи на численні переваги, React має й певні недоліки, які варто враховувати під час розробки якісного вебзастосунку. Насамперед слід зазначити, що React не є повноцінним фреймворком. Це лише бібліотека для створення інтерфейсу користувача, тому для реалізації повноцінної клієнтської частини доводиться додатково підключати сторонні бібліотеки. Зокрема, для організації маршрутизації зазвичай використовують React Router, для керування глобальним станом – Redux, Zustand або інші рішення, а для роботи з формами – Formik або React Hook Form. Це збільшує складність конфігурації проєкту та вимагає від розробника більш глибоких знань екосистеми.

Ще одним недоліком є високий поріг входу при створенні масштабних застосунків. Хоча базові принципи React доступні й зрозумілі, розробка великого вебзастосунку потребує знання архітектурних шаблонів, ефективного управління станом, розуміння життєвого циклу компонентів, асинхронної взаємодії з API тощо. Недосвідченим розробникам може бути складно впровадити ці елементи правильно, що впливає на підтримуваність і якість коду.

Також варто враховувати швидкий темп розвитку React та його екосистеми. Інструменти та підходи, які використовувались кілька років тому, можуть бути вже застарілими або заміненими на нові. Це змушує команду постійно оновлювати свої знання, адаптуватися до змін і оновлювати кодову

базу, щоб відповідати сучасним стандартам та забезпечувати стабільну роботу застосунку.

У підсумку, React є сучасною, гнучкою та продуктивною технологією, яка ідеально підходить для реалізації клієнтської частини платформи організації екскурсій, де важлива інтерактивність, масштабованість і зручність користування [2].

У процесі розробки важливу роль відіграє робота з формами – це форма зворотного зв’язку, запису на екскурсію, створення облікового запису тощо. React дозволяє реалізувати ці форми з високим рівнем інтерактивності, валідацією введених даних у режимі реального часу та зручним для користувача способом взаємодії [3].

Також, враховуючи відкритість ресурсу для широкого кола користувачів, важливо реалізувати безпечну автентифікацію та захист персональних даних. У цьому контексті React дозволяє впровадити сучасні механізми аутентифікації та авторизації, зокрема використання токенів доступу та перевірку ролей користувача (турист, гід, адміністратор).

При виборі технологій для розробки вебсайту, який забезпечує взаємодію між користувачами, екскурсоводами та адміністраторами, важливо провести порівняльний аналіз сучасних JavaScript-фреймворків, здатних забезпечити ефективну роботу платформи. На 2025 рік до найпопулярніших рішень для створення інтерфейсів користувача належать:

– Vue.js;
– Angular;
– Svelte;
– Solid.js.

Для реалізації вебплатформи з пошуку, бронювання та управління екскурсіями важливо обрати рішення, яке поєднує в собі гнучкість, масштабованість, легкість підтримки та добру інтеграцію з backend API.

Angular, натомість, є повноцінним фреймворком зі строгою архітектурою, вбудованими засобами для форм, безпеки та маршрутизації.

Але через високу складність і строгу структуру Angular часто є менш зручним для гнучкої розробки UI, особливо у проєктах, де потрібна динамічність і часті зміни у вимогах до інтерфейсу [3].

Таким чином, вибір React як основного інструменту для фронтенд-розробки платформи екскурсій є обґрунтованим.

1.3 Аналіз та обґрунтування вибору технологій бекенду

Для реалізації серверної частини вебсайту, призначеного для організації та проведення екскурсій, було обрано технологію Node.js як основну платформу для створення API та обробки запитів користувачів. Node.js забезпечує високу продуктивність, неблокуюче виконання коду та зручну інтеграцію з базами даних і сторонніми сервісами [4].

У процесі розробки були використані такі інструменти та бібліотеки. Node.js виступає як середовище виконання серверного JavaScript-коду, що дозволяє ефективно обробляти велику кількість одночасних запитів. Express.js – це легкий і гнучкий фреймворк для побудови RESTful API, який значно спрощує створення маршрутів, обробку запитів та помилок. SQLite – проста та швидка реляційна база даних, що зберігається у вигляді одного локального файлу; вона ідеально підходить для невеликих проєктів або MVP і використовується для зберігання інформації про екскурсії, гідів, користувачів та бронювання.

JWT (JSON Web Tokens) застосовується для реалізації системи авторизації та автентифікації користувачів [6]. Бібліотека dotenv використовується для зберігання конфіденційних даних, таких як ключі доступу, у спеціальному .env файлі. Крім того, Sequelize – це ORM-бібліотека, яка спрощує взаємодію з базою даних, дозволяє працювати з моделями та абстрагує складні SQL-запити, роблячи розробку більш ефективною та структурованою.

Реалізація серверної частини сайту для організації та проведення екскурсій передбачає вибір надійних і масштабованих технологій, які забезпечують стабільність роботи, швидку обробку запитів та безпеку даних користувачів. Нижче наведено короткий огляд ключових компонентів, що використовуються в бекенд-архітектурі.

Перевагами Node.js є велика екосистема пакетів, легкість масштабування та можливість використання єдиної мови програмування як на сервері, так і на клієнті [5].

Express.js – популярний фреймворк для Node.js, який спрощує розробку вебсерверів і REST API. Він забезпечує гнучке маршрутизування запитів, зручну обробку помилок і підтримку проміжного програмного забезпечення (middleware), що дає змогу реалізувати складну бізнес-логіку, наприклад, реєстрацію та бронювання екскурсій, а також автентифікацію користувачів.

Для зберігання даних про екскурсії, користувачів, бронювання та гідів використовується реляційна база даних PostgreSQL. Вона забезпечує надійність, цілісність даних та підтримку складних запитів. Завдяки потужним інструментам керування, PostgreSQL ідеально підходить для систем із великою кількістю взаємопов’язаних сутностей.

Для реалізації системи безпечної автентифікації та авторизації користувачів використовується JWT. Токени зберігають інформацію про права доступу користувачів, що дозволяє захистити ресурси сайту та контролювати доступ до функцій, таких як бронювання екскурсій або управління профілем.

Для збереження конфіденційних налаштувань (ключів доступу до бази даних, секретів JWT тощо) застосовується пакет dotenv, який дозволяє зберігати ці дані у файлі .env, виключаючи їх зі сховища коду та підвищуючи безпеку застосунку.

Для реалізації функціоналу реального часу, наприклад, оновлення статусу екскурсії чи повідомлень між організаторами і учасниками,

використовується Socket.IO – бібліотека для вебсокетів, яка забезпечує двонапрямлену комунікацію між сервером і клієнтом.

Таким чином, інтеграція Node.js із Express.js, PostgreSQL, Sequelize, JWT та іншими інструментами забезпечує створення масштабованої, безпечної та ефективної серверної архітектури для платформи організації та проведення екскурсій. Це дозволяє гнучко розвивати проєкт, забезпечувати швидку обробку користувацьких запитів та підтримувати високу надійність роботи системи [5].

1.4 Аналіз функціональних та нефункціональних вимог

Перед початком розробки вебсайту для організації та проведення екскурсій, виконавши аналіз переваг та недоліків існуючих аналогів, необхідно провести детальний аналіз вимог до системи. Це дозволить чітко визначити функціональні межі майбутнього продукту, його можливості та сформувати критерії успішності проєкту. Такий підхід забезпечить створення рішення, яке максимально відповідатиме потребам користувачів, мінімізує ризики помилок і затримок у розробці, а також стане основою для подальшого проєктування та впровадження [14].

Функціональні вимоги до сайту визначають конкретні можливості та сервіси, які має надавати кінцевий продукт, описуючи його поведінку у різних сценаріях використання. Вони охоплюють дії користувачів, адміністраторів і взаємодію із зовнішніми сервісами. У таблиці 1.1 наведено основні функціональні можливості, які повинна реалізовувати система для забезпечення повноцінної та зручної взаємодії з користувачами.

Сайт для організації та проведення екскурсій повинен забезпечувати повний життєвий цикл екскурсій – від їх створення до проведення та завершення – підтримуючи всі необхідні функції для ефективного управління заходами. Особливу увагу слід приділити зручному та інтуїтивно зрозумілому

інтерфейсу для створення та редагування екскурсій, який дозволяє швидко вносити зміни, задавати різні параметри (тип екскурсії, дату, тривалість, місце збору).

Таблиця 1.1 – Функціональні вимоги

№	**Функціональна можливість**
1	Створення, редагування та видалення екскурсій з можливістю встановлення типів і категорій (історичні, тощо).
2	Керування розкладом проведення екскурсій із зазначенням дати, часу та місця збору.
3	Реєстрація учасників на екскурсії з автоматичним формуванням списків.
4	Надсилання сповіщень та нагадувань учасникам щодо подій і змін у розкладі.
5	Відображення інформації про екскурсії з фільтрацією за датою, типом, тривалістю, вартістю.
6	Інтеграція з картографічними сервісами для маршрутів і точок збору.
7	Можливість залишати відгуки та оцінки щодо екскурсій.
8	Обмін повідомленнями між організаторами та учасниками.
9	Генерація звітів про екскурсії, кількість учасників і відгуки.
10	Налаштування прав доступу для ролей: адміністратор, організатор, учасник.
11	Реалізація пошуку та сортування екскурсій за різними критеріями.

Важливим аспектом є наявність системи сповіщень, яка інформуватиме учасників про майбутні екскурсії, зміни в розкладі або важливі новини. Механізми відстеження статусу екскурсії мають відображати загальну інформацію про майбутні та проведені заходи, а також надавати детальні дані про кількість зареєстрованих учасників і відгуки. Для зручності організаторів та користувачів варто передбачити візуалізацію даних у вигляді календаря або списку екскурсій із фільтрами за різними параметрами [12].

Нефункціональні вимоги визначають якісні характеристики вебсайту для організації та проведення екскурсій, які не стосуються безпосередньо його функціоналу, проте є критично важливими для забезпечення зручності

використання, надійності та стабільної роботи платформи. У таблиці 1.2 наведено основні нефункціональні вимоги до системи, які охоплюють аспекти продуктивності, безпеки, адаптивності, масштабованості та відновлюваності.

Таблиця 1.2 – Нефункціональні вимоги

№	Характеристика	Опис вимоги
1	Продуктивність	Сайт повинен швидко завантажуватися та підтримувати стабільну роботу при одночасному доступі до 100 користувачів, включаючи операції перегляду та оплати.
2	Надійність	Рівень доступності платформи має становити не менше 99,5% часу, що забезпечує безперервне функціонування сервісу.
3	Масштабованість	Архітектура повинна підтримувати збільшення кількості екскурсій, користувачів і замовлень без зниження продуктивності.
4	Безпека	Конфіденційна інформація має бути захищена шифруванням, передача даних – лише через HTTPS. У майбутньому – підтримка 2FA для організаторів.
5	Зручність використання	Інтерфейс повинен бути інтуїтивно зрозумілим і зручним як для користувачів, так і для адміністраторів платформи.
6	Адаптивність	Вебсайт має коректно працювати на ПК, ноутбуках, планшетах і мобільних телефонах.
7	Локалізація	Платформа повинна підтримувати багатомовність: українську, англійську, німецьку та мати можливість додавання нових мов.
8	Відновлюваність	Система має забезпечувати регулярне автоматичне резервне копіювання та швидке відновлення даних у разі збою.

Забезпечення високої продуктивності та надійності вебсайту для організації та проведення екскурсій є ключовими чинниками його успішного впровадження, особливо в умовах сезонного зростання попиту та одночасної активності великої кількості користувачів. Архітектурні рішення повинні

дозволяти горизонтальне масштабування, що забезпечить гнучке розширення ресурсів у разі збільшення навантаження без потреби повного оновлення інфраструктури.

Особливу увагу потрібно приділити безпеці – зокрема, впровадженню захисту від поширених загроз, таких як SQL-ін’єкції, XSS та CSRF-атаки, а також забезпеченню регулярного резервного копіювання даних користувачів, екскурсій та бронювань. Важливим елементом є впровадження системи журналювання подій та аудиту дій користувачів для виявлення підозрілої активності. Крім того, захист персональних даних повинен відповідати вимогам чинного законодавства [9].

Інтерфейс користувача сайту має бути адаптивним, сучасним та зручним для взаємодії як з боку туристів, так і з боку організаторів екскурсій. Таким чином, сформульовані функціональні та нефункціональні вимоги слугують міцною основою для наступних етапів проєктування, реалізації та впровадження системи.

1.5 Висновки до розділу 1

У першому розділі було проведено аналіз існуючих платформ для організації екскурсій та технологій для їх розробки. Виконано детальний огляд популярних сервісів, таких як GetYourGuide, Tripster та Tiqets, з визначенням їхніх ключових функцій, переваг та обмежень, що дозволило сформувати уявлення про сучасні вимоги до таких систем. Обґрунтовано вибір Node.js як основної технології для серверної частини через її високу продуктивність, масштабованість та можливість ефективної обробки великої кількості одночасних запитів. Для роботи з базою даних обрано PostgreSQL із застосуванням ORM-бібліотеки Sequelize, що забезпечує надійне зберігання і цілісність даних. Також проведено дослідження доступних інструментів та бібліотек для забезпечення безпеки, автентифікації та управління

конфіденційною інформацією, що дозволило спроєктувати безпечну і стабільну серверну архітектуру для платформи організації екскурсій.

2 ПРОЄКТУВАННЯ СИСТЕМИ ОРГАНІЗАЦІЇ ТА ПРОВЕДЕННЯ ЕКСКУРСІЙ

2.1 Концептуальне проєктування інтерфейсу

Концептуальне проєктування є важливим етапом розробки вебсайту, що дозволяє визначити основні принципи взаємодії користувачів із системою. На цьому етапі формується загальне уявлення про функціональність ресурсу, що суттєво знижує ризики створення рішення, яке не відповідає очікуванням кінцевих користувачів, та задає напрямок для подальших етапів реалізації.

На рисунку 2.1 представлено діаграму варіантів використання (прецедентів) сайту для організації та проведення екскурсій, яка ілюструє ключові функціональні можливості платформи, а також взаємодію між основними типами користувачів, зокрема туристами, адміністраторами та екскурсоводами.

Представлена діаграма прецедентів відображає ключові варіанти використання системи «Сайт екскурсій» та взаємодію трьох основних типів користувачів: туриста, гіда та адміністратора. Кожна з ролей має доступ до певного підмножини функціональностей системи відповідно до своїх повноважень та потреб [8].

Турист – це кінцевий користувач, який відвідує сайт з метою ознайомлення з доступними екскурсіями та їх бронювання. Основні варіанти використання для цієї ролі включають перегляд списку екскурсій, фільтрацію екскурсій за визначеними критеріями (тематика, дата, місто тощо), бронювання вибраної екскурсії, а також залишення відгуку після участі в екскурсії.

Гід – користувач, що пропонує свої екскурсійні послуги через платформу. Його функціональні можливості охоплюють створення нових екскурсій із зазначенням маршруту, тривалості, ціни та іншої важливої

інформації, редагування вже створених екскурсій, а також відповідь на відгуки туристів з метою підтримання високого рівня обслуговування.

Адміністратор є відповідальним за управління системою в цілому. До його прецедентів належить модерація створених екскурсій, зокрема активація або деактивація пропозицій, перегляд статистики системи (наприклад, кількість активних екскурсій, відвідуваність тощо), керування обліковими записами користувачів – включно з призначенням ролей (турист, гід) та блокуванням порушників. Крім того, адміністратор відповідає за управління контентом на сайті, таким як новини, оголошення та інші інформаційні матеріали. Таким чином, діаграма відображає повну взаємодію між користувачами та функціональними можливостями системи, забезпечуючи основу для її подальшого технічного проєктування.

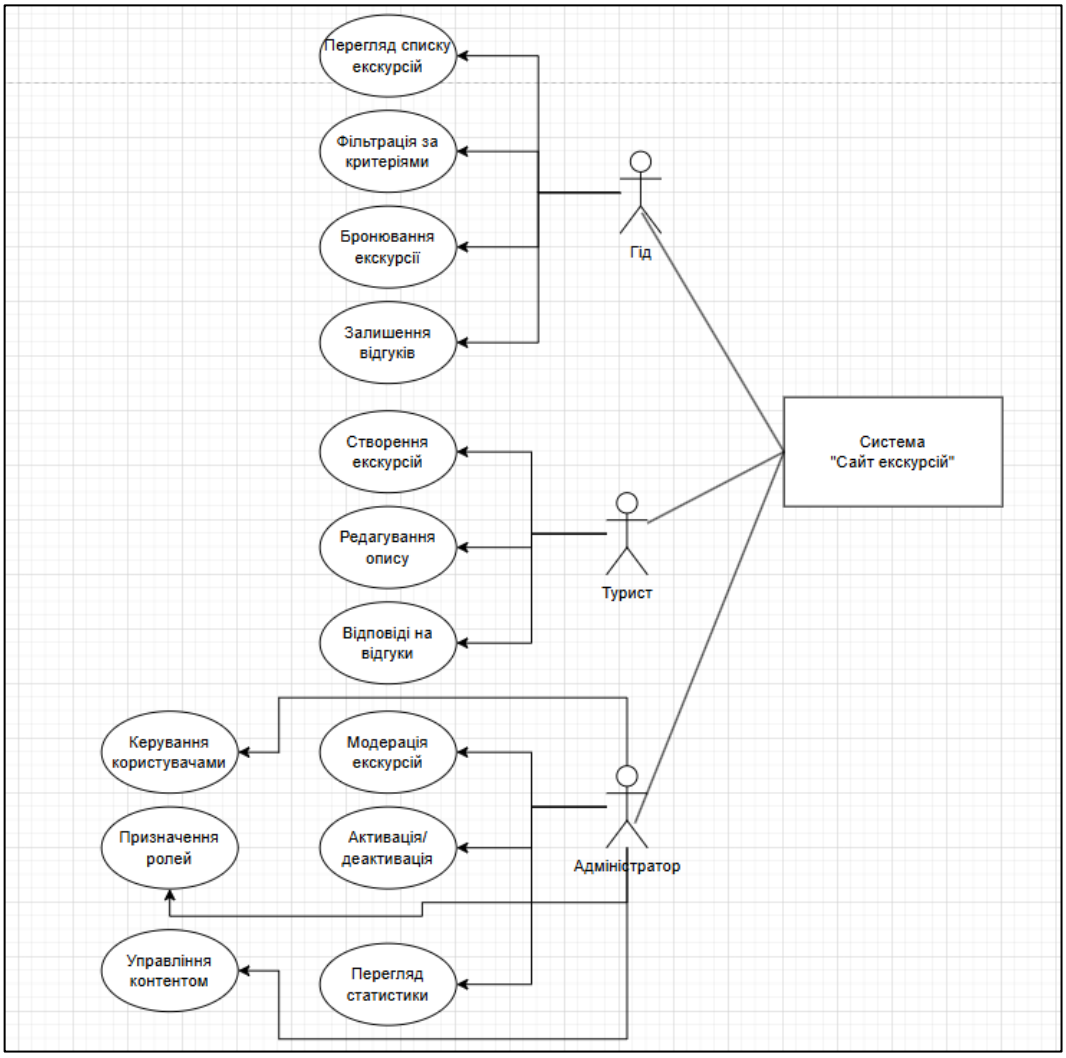


Рисунок 2.1 – Діаграма прецедентів вебзастосунку

При проєктуванні програмного забезпечення критичну роль відіграє прототипування (створення макетів) основних компонентів майбутнього вебсайту. Це дозволяє на ранніх етапах візуалізувати логіку взаємодії користувачів із системою, підвищити зручність інтерфейсу та уникнути помилок у реалізації.

Для розробки сайту для організації та проведення екскурсій було створено кілька ключових макетів інтерфейсу – головна сторінка, сторінка списку екскурсій із фільтрами, а також сторінка з деталями обраної екскурсії та функцією бронювання. На рисунку 2.2 представлено макет головної сторінки сайту, яка є центральним елементом взаємодії з користувачем. Вона включає панель навігації, пошук екскурсій за ключовими критеріями, банери із популярними маршрутами, а також доступ до особистого кабінету для зареєстрованих користувачів [8].

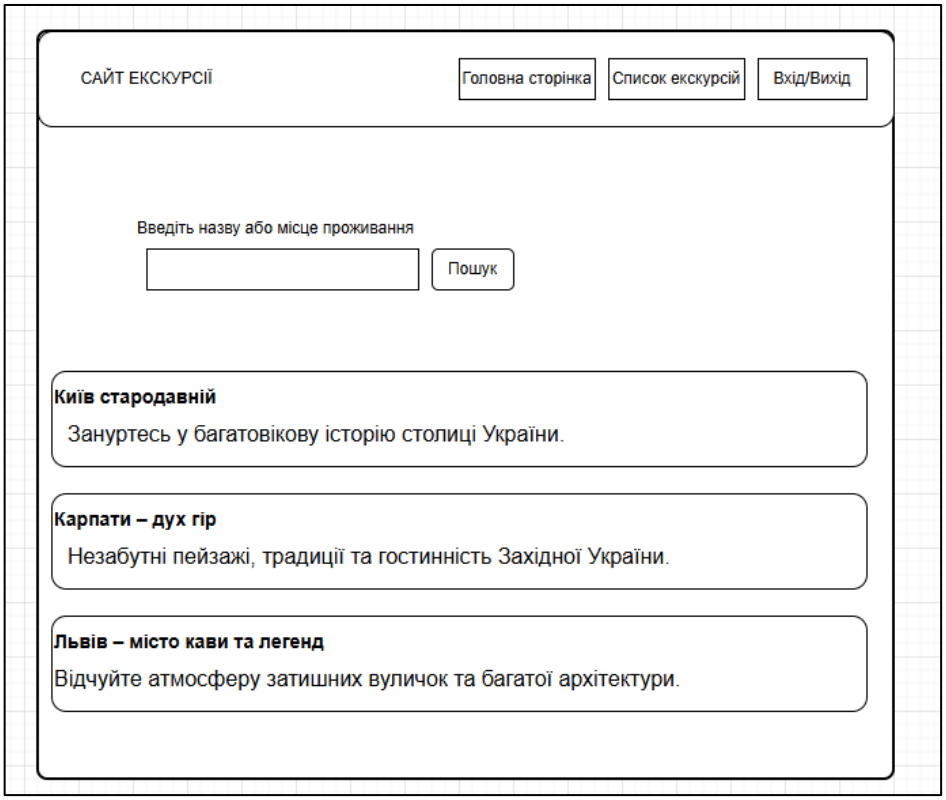


Рисунок 2.2 – Макет головної сторінки

Відповідно до наведеного макету, користувача зустрічає привітальне повідомлення та основні інформаційні блоки. У верхній частині сторінки

розташована стандартна горизонтальна навігаційна панель із кнопками: «Головна сторінка», «Список екскурсій», «Вхід/Вихід», «Пошук». Нижче розташовані банери з популярними екскурсіями, які приваблюють увагу та допомагають швидко обрати цікавий маршрут. Такий дизайн забезпечує зручну та інтуїтивно зрозумілу навігацію для користувачів сайту [7].

Основним компонентом сайту для організації та проведення екскурсій є список екскурсій, макет якого наведено на рисунку 2.3.

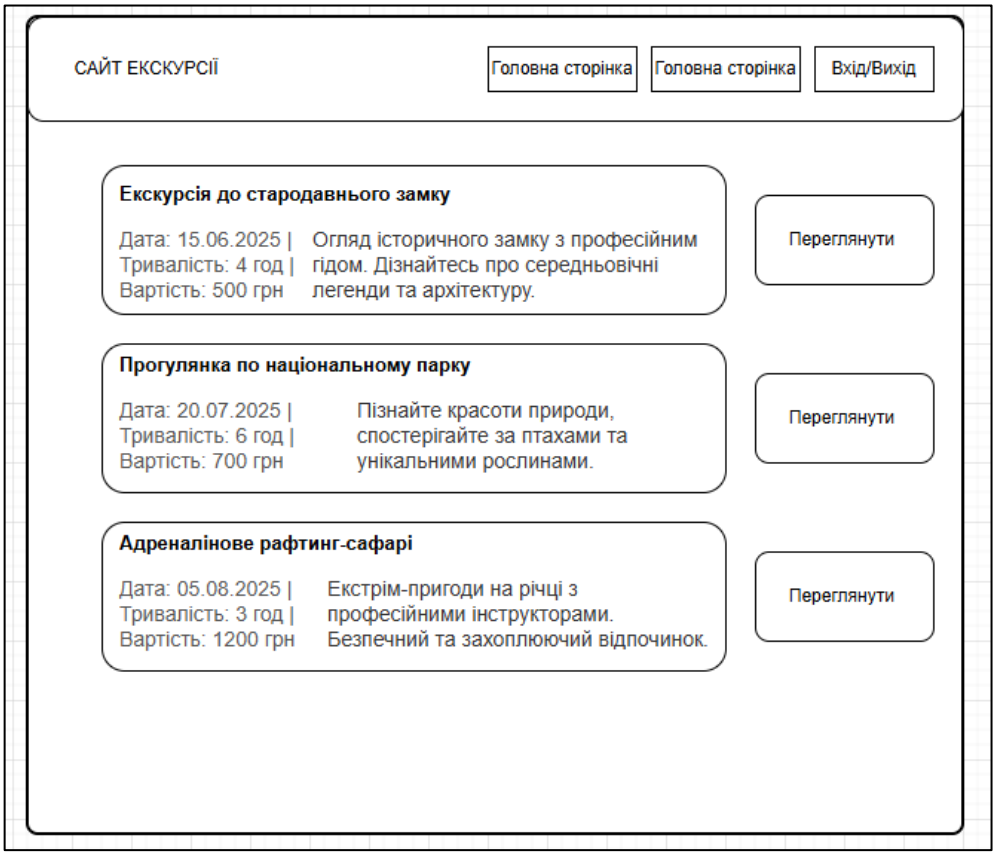


Рисунок 2.3 – Макет списку екскурсій

Компонентами інтерфейсу сторінки реєстрації сайту є логотип системи та повна назва сайту, які розміщені у верхній частині сторінки для ідентифікації бренду. Основна частина сторінки займає форма реєстрації нового користувача, що містить поля для введення необхідних даних: ім’я, прізвище, електронна пошта, номер телефону, пароль та підтвердження паролю. Нижче полів розташована кнопка «Зареєструватися», яка активує процес створення нового акаунта.

Для зручності користувача передбачена посилання на сторінку входу для тих, хто вже має обліковий запис. Інтерфейс виконано у чистому та мінімалістичному стилі, без зайвих елементів, що дозволяє зосередитися на процесі реєстрації.

Колірна палітра макету світла з акцентними елементами для кнопок та важливих повідомлень. Всі поля введення мають підказки (плейсхолдери) та повідомлення про помилки, що допомагають користувачу коректно заповнити форму.

Відповідно до рисунку 2.4, наведені матеки блоків логотипу та повної назви сайту залишаються незмінними, розташованими у верхній частині сторінки для забезпечення впізнаваності бренду. На відміну від інших сторінок, навігаційне меню у згорнутому вигляді на сторінці реєстрації відсутнє, щоб не відволікати користувача від заповнення форми реєстрації.

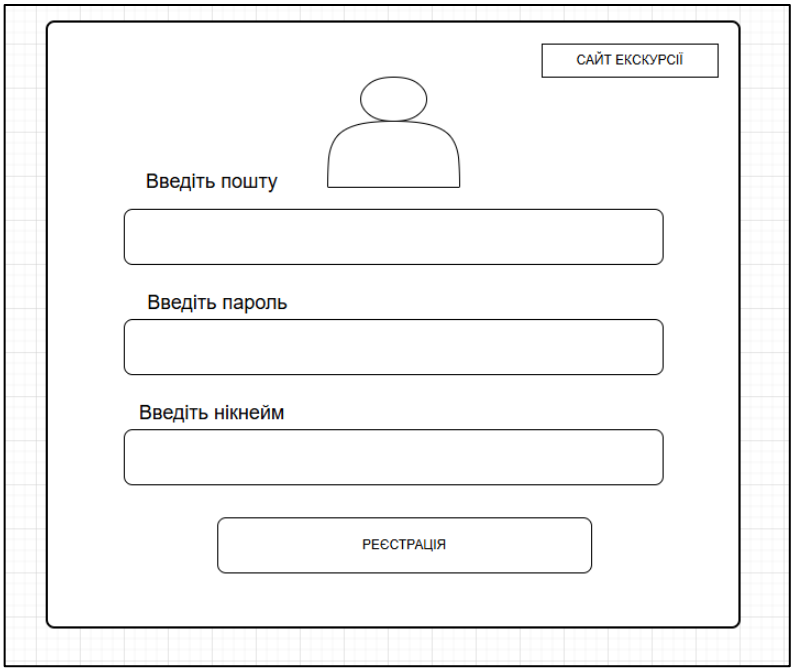


Рисунок 2.4 – Макет реєстрації

На основі проведеного концептуального проєктування сайту для організації та проведення екскурсій можна зробити висновок, що розроблені діаграми прецедентів та макети інтерфейсу забезпечують базове розуміння функціональних можливостей майбутньої системи та особливостей взаємодії

користувачів із нею. Сайт передбачає три основні типи користувачів – туриста, гіда та адміністратора – з чітким розмежуванням прав доступу. Передбачено набір функцій, зокрема перегляд екскурсій, їх фільтрація, бронювання, залишення відгуків, створення нових маршрутів, їх редагування та модерація.

Представлені макети інтерфейсу головної сторінки, списку екскурсій та форми реєстрації демонструють логічну та зручну структуру сайту, що враховує потреби різних категорій користувачів. Це створює міцну основу для наступних етапів розробки, мінімізуючи ризики невідповідності функціоналу очікуванням кінцевих користувачів.

2.2 Проєктування фронтендної частини

Логічне проєктування в розробці вебзастосунку – це процес створення абстрактної моделі системи, яка описує її функціональну структуру, організацію даних та логіку взаємодії між компонентами інтерфейсу незалежно від технологій реалізації. Основна увага зосереджується на узгодженій роботі елементів інтерфейсу та зв’язках між даними.

На рисунку 2.5 представлено діаграму класів фронтенд-частини сайту для організації та проведення екскурсій. Ця діаграма відображає логічну структуру основних класів, що відповідають за взаємодію користувача з екскурсіями, профілем та іншими функціональними модулями.

Ключовими класами системи є кілька основних моделей, кожна з яких виконує специфічну роль у структурі вебзастосунку для організації екскурсій.

Клас Tour представляє модель екскурсії. Він містить такі властивості, як назва (title), опис (description), ціна (price), дата проведення (date), місце (location) та ідентифікатор гіда (guideId). Між класами Tour та Guide реалізовано зв’язок типу «багато до одного», тобто один гід може проводити багато екскурсій.

Клас User відповідає за модель користувача системи. Користувачі можуть мати різні ролі – турист або гід, що визначає набір функцій, доступних

у їхньому інтерфейсі. Ролі впливають на можливості перегляду, бронювання або створення екскурсій.

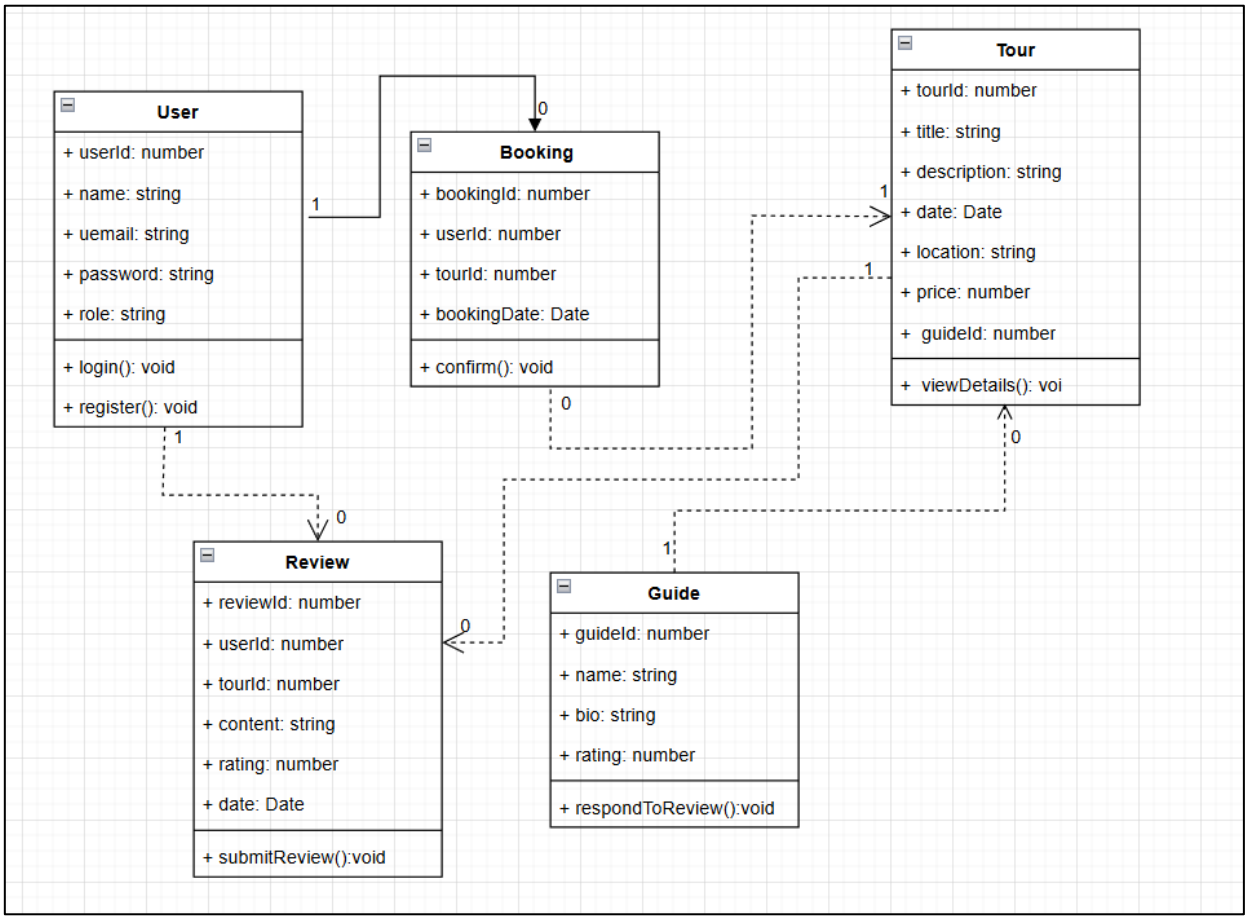


Рисунок 2.5 – Діаграма класів фронтенд-частини вебзастосунку

Booking – це клас, що реалізує механізм бронювання. Він має зв’язки типу «багато до одного» з класами User (як турист) та Tour. Це дає змогу точно визначати, які користувачі забронювали конкретні екскурсії.

Клас Review призначений для зберігання відгуків. Він має зв’язки з класом User, що визначає автора відгуку, а також з класом Tour, до якого належить відповідний коментар.

Окрему роль у системі виконує клас Guide. Цей клас деталізує профіль гіда: він може редагувати власну інформацію, відповідати на відгуки користувачів, а також створювати та редагувати екскурсії.

Клас User є центральним елементом системи й забезпечує автентифікацію користувачів за допомогою методів login() та register(). Кожен

користувач може здійснювати кілька бронювань (Booking) та залишати відгуки (Review) про різні екскурсії, що показує зв’язок типу «один до багатьох».

Бронювання (Booking) містить інформацію про користувача, обрану екскурсію та дату реєстрації. Відповідно, кожна екскурсія (Tour) може мати багато бронювань і багато відгуків, що відображає активну взаємодію між відвідувачами та турами.

Клас Tour включає основні характеристики екскурсії: назву, опис, місце проведення, дату, ціну та ідентифікатор гіда. Кожен тур асоційований з одним гідом (Guide), який відповідає за проведення екскурсії. Це зв’язок типу «багато до одного».

Клас Review моделює функціональність зворотного зв’язку – кожен користувач може залишити відгук про конкретний тур, оцінити його та додати текстовий коментар. Це сприяє підвищенню якості сервісу через відкриті оцінки.

Таким чином, структура класів підтримує основні функції системи: реєстрацію користувачів, перегляд екскурсій, здійснення бронювання, залишення відгуків і керування турами зі сторони гідів. Всі компоненти інтерфейсу взаємодіють із цими класами опосередковано через відповідні сервіси, що забезпечує модульність та масштабованість архітектури фронтенду [11].

Для демонстрації фізичної організації системи на рисунку 2.6 наведено діаграму компонентів фронтенд-частини проєкту системи бронювання турів.

Діаграма компонентів фронтенду системи бронювання, реалізованої з використанням бібліотеки React, ілюструє модульну компонентно-орієнтовану архітектуру, що відповідає вимогам, сформульованим у функціональній специфікації. Центральними елементами є компоненти, пов’язані з управлінням турами (TourComponent) та бронюваннями (BookingComponent), які організовані через React Router для забезпечення зручної навігації між розділами застосунку.

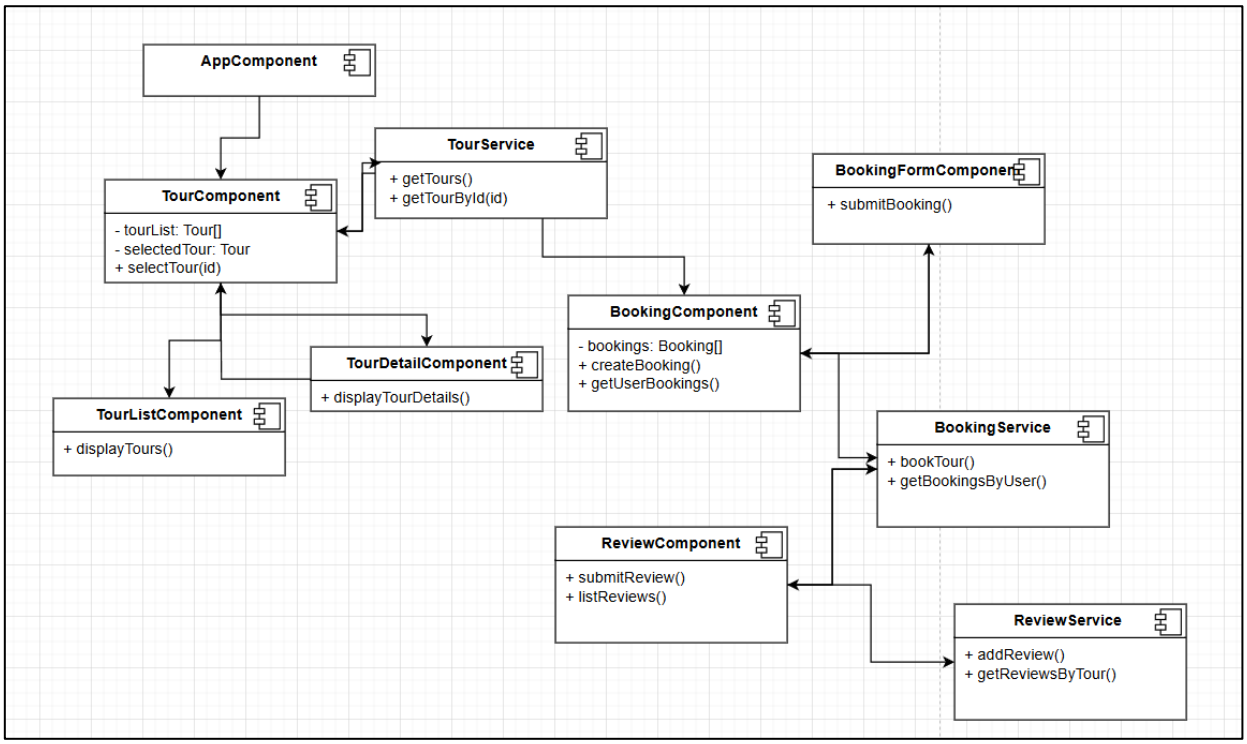


Рисунок 2.6 – Діаграма компонентів

Компонент TourComponent відповідає за відображення списку турів та деталей окремого туру через підкомпоненти TourList і TourDetail. Він взаємодіє з TourService (або кастомним хуком useTourService), який забезпечує доступ до даних про тури з бекенду. Аналогічно, компонент BookingComponent обробляє процес бронювання, використовуючи BookingForm для створення заявки та BookingList для перегляду поточних бронювань користувача. Всі ці підкомпоненти обмінюються інформацією через пропси та callback-функції [14].

Компоненти TourComponent і BookingComponent мають залежність від відповідних сервісів (TourService, BookingService), які взаємодіють з бекендом через HTTP-запити (наприклад, через fetch або axios), абстрагуючи REST API. Для структурування даних використовуються моделі TourModel, BookingModel та UserModel, що спрощує передачу й обробку інформації на стороні клієнта.

Також на діаграмі відображено компонент ReviewComponent, що відповідає за виведення й додавання відгуків до туру. Він залежить від ReviewService і тісно інтегрується з TourDetailComponent.

На рисунку 2.7 зображено діаграму розгортання вебзастосунку, розробленого для організації та проведення екскурсій. Діаграма ілюструє основні апаратно-програмні компоненти системи та взаємозв’язки між ними в середовищі розгортання.

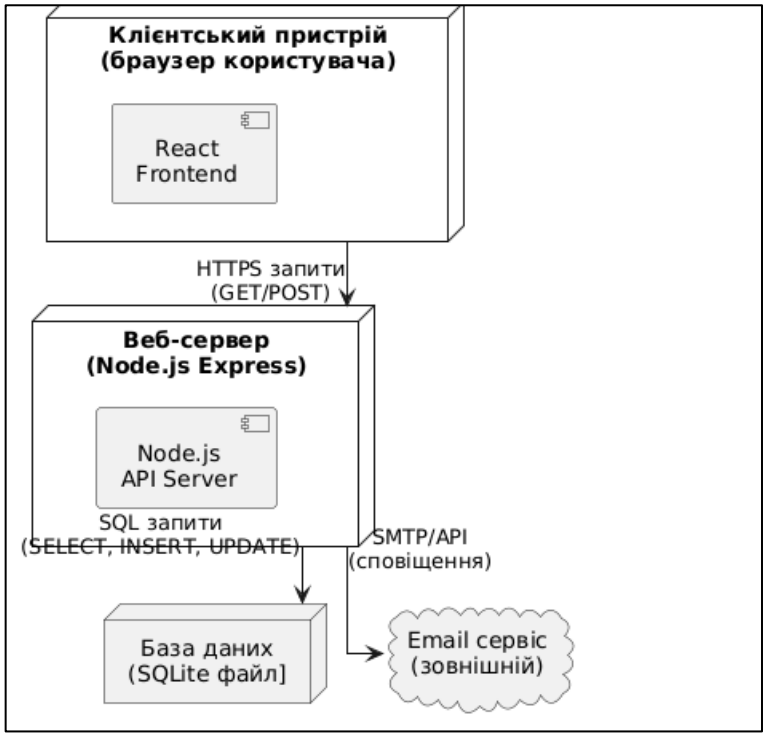


Рисунок 2.7 – Діаграма розгортання

Система базується на сучасному технологічному стеку, до якого входять React для реалізації клієнтської частини, Node.js для створення серверної логіки та SQLite як вбудована реляційна база даних для зберігання інформації про користувачів, екскурсії, бронювання тощо.

Клієнтська частина (Frontend), реалізована з використанням React, працює у веббраузері користувача. Вона відповідає за відображення інтерфейсу, приймання введених даних та взаємодію з користувачем. Усі дії користувача – наприклад, реєстрація, перегляд екскурсій, запис на подію – надсилаються у вигляді HTTP-запитів до серверної частини.

Серверна частина (Backend), реалізована на платформі Node.js, розгорнута на вебсервері. Вона приймає запити з клієнта, виконує бізнес-логіку, обробляє дані, зберігає або читає їх із бази даних SQLite та формує

відповіді для клієнта. У разі потреби сервер також може взаємодіяти з зовнішніми сервісами, наприклад, API для надсилання листів, SMS або геолокації.

База даних SQLite використовується для локального зберігання структурованої інформації. Вона підходить для невеликих або середніх проєктів завдяки простоті налаштування, високій продуктивності на малих об’ємах даних і відсутності необхідності в окремому сервері бази даних.

Користувачі взаємодіють із системою через браузери з будь-якого пристрою, що має доступ до Інтернету: комп’ютери, ноутбуки, планшети або смартфони. Вся взаємодія відбувається через безпечне HTTPS-з’єднання, що гарантує захист переданих даних.

На рисунку 2.8 зображено діаграму структури бази даних системи для організації та проведення екскурсій. У цій моделі визначено ключові сутності, зокрема користувачів, екскурсії, бронювання, відгуки та гідів, а також встановлено між ними логічні зв’язки.

Кожен користувач може мати одну з кількох ролей (наприклад, турист або гід), що визначає доступні функціональні можливості. Сутність Tour представляє екскурсію, яку проводить певний гід, що задається через зв’язок з таблицею Guide. Таблиця Booking використовується для зберігання інформації про реєстрацію користувачів на екскурсії, включаючи статус бронювання.

Діаграма ілюструє, що один користувач може створити багато бронювань та залишити декілька відгуків. Це дозволяє користувачам неодноразово брати участь в різних екскурсіях і ділитися своїми враженнями після кожного заходу, що підвищує залученість та активність у системі.

Один гід може проводити декілька екскурсій. Це відображає реальний сценарій, коли один фахівець обслуговує кілька маршрутів або подій у різні дні, що забезпечує гнучкість у плануванні та організації екскурсійної діяльності. Кожна екскурсія може мати багато учасників і отримати кілька відгуків. Це забезпечує можливість масового обслуговування туристів у

рамках однієї події та дозволяє системі накопичувати оцінки й коментарі, що можуть впливати на репутацію туру та організатора.

Сутність Guide є розширенням User, що містить додаткову інформацію про гіда. Такий підхід дозволяє ефективно реалізувати рольову модель у базі даних, не дублюючи основну інформацію про користувачів, а лише додаючи необхідні розширення для певних ролей.

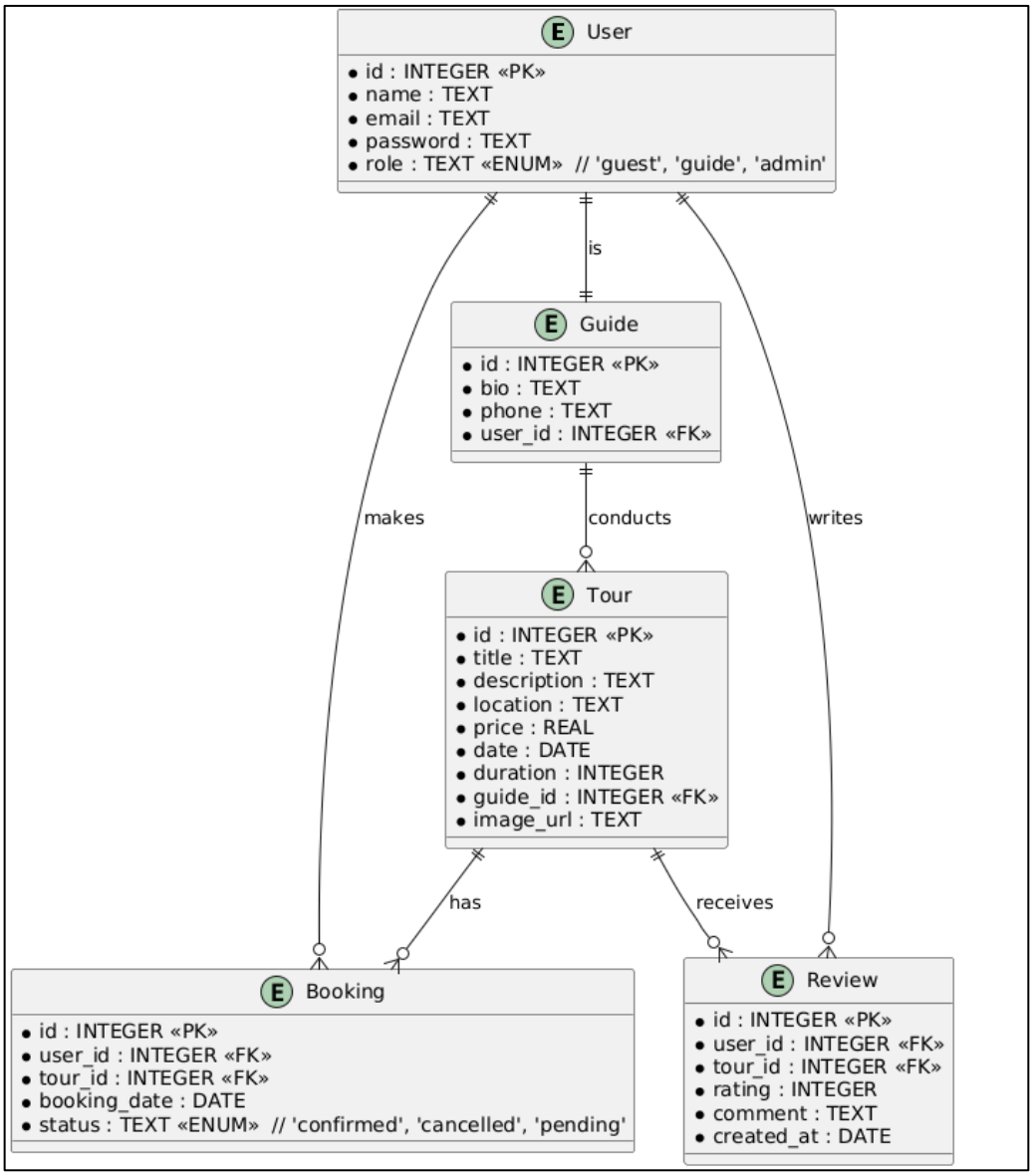


Рисунок 2.8 – Діаграма структури бази даних

Таким чином, побудована архітектура дозволяє досягти високого рівня модульності, повторного використання коду та спрощує розширення функціоналу застосунку.

2.3 Проєктування серверної частини

Проєктування серверної частини системи бронювання турів зосереджено на забезпеченні надійної, масштабованої та безпечної взаємодії між клієнтським застосунком та базою даних. Серверна частина реалізована з використанням Node.js та Express.js, що дозволяє створити RESTful API для обробки запитів від фронтенду [5].

Основними компонентами серверної архітектури є контролери (Controllers), які обробляють HTTP-запити, виконують валідацію даних, викликають бізнес-логіку та повертають відповіді клієнту. Сервіси (Services) містять основну бізнес-логіку, відповідають за обробку даних і взаємодіють з моделями.

Моделі (Models) відображають структуру даних і забезпечують взаємодію з базою даних PostgreSQL через ORM, наприклад, Sequelize або TypeORM. Маршрути (Routes) визначають кінцеві точки API для різних ресурсів системи, таких як користувачі, тури, бронювання, відгуки тощо.

Мідлвари (Middleware) реалізують додаткові функції – зокрема, аутентифікацію (наприклад, за допомогою JWT), авторизацію, обробку помилок та логування.

Архітектура забезпечує розділення відповідальностей між компонентами, що сприяє гнучкості та розширюваності системи. Використання RESTful API дозволяє легко інтегрувати сервер з різними клієнтськими застосунками, а застосування сучасних стандартів безпеки гарантує захист даних користувачів [13].

Express.js виступає як вебфреймворк для організації маршрутів та обробки HTTP-запитів, що дає змогу створювати REST API, яке забезпечує чітку та логічну взаємодію між фронтендом та бекендом системи. REST архітектура підтримує стандартизовані методи HTTP (GET, POST, PUT, DELETE), що відповідає основним операціям над ресурсами системи: користувачами, турами, бронюваннями та відгуками [6].

Кожен ресурс системи має власний набір ендпоінтів. Зокрема, для користувачів передбачено ендпоінти реєстрації, аутентифікації та оновлення профілю. Ресурс турів включає можливість отримання списку турів, перегляду детальної інформації про тур, а також додавання чи редагування туру (останні функції доступні лише адміністраторам). Для бронювання реалізовано ендпоінти створення нового бронювання, перегляду поточних бронювань користувача та скасування бронювання. Нарешті, для відгуків передбачено додавання відгуку на тур, а також отримання вже наявних відгуків.

Для безпеки взаємодії використовується механізм аутентифікації та авторизації на основі JWT (JSON Web Token), що гарантує захист персональних даних і доступ лише авторизованих користувачів до певних функцій системи. Для збереження даних застосовується реляційна база даних (наприклад, PostgreSQL), доступ до якої здійснюється через ORM (Object-Relational Mapping), що спрощує маніпуляції з даними та підтримує цілісність інформації.

Такий підхід до розробки серверної частини забезпечує надійний, швидкий та гнучкий бекенд, який легко інтегрується з фронтендом на React та дозволяє масштабувати систему відповідно до потреб користувачів [10].

2.4 Висновки до розділу 2

У розділі проєктування системи бронювання турів було проведено аналіз функціональних та нефункціональних вимог, що дозволило визначити ключові можливості системи та критерії її ефективності. Розроблено концептуальну модель за допомогою діаграми прецедентів та макетів інтерфейсу, які ілюструють взаємодію користувачів із системою. Створено логічну архітектуру через діаграму класів, що описує основні сутності, їх атрибути та взаємозв’язки, а також структуру даних, необхідних для роботи системи.

Реалізовано фізичну модель через діаграму компонентів фронтенд-частини, розробленої на React, та опис серверної частини із використанням REST API на базі Node.js і Express. Така архітектурна організація забезпечує модульність, масштабованість та надійність системи, що відповідає визначеним функціональним і нефункціональним вимогам, і створює міцну основу для подальшої розробки та впровадження проєкту.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Реалізація фронтендної частини

Фронтендна частина вебзастосунку для організації та проведення екскурсій була реалізована з використанням фреймворку React, який забезпечує побудову динамічного та інтерактивного інтерфейсу користувача. Основною метою створення клієнтської частини є надання користувачам інтуїтивно зрозумілого та зручного вебінтерфейсу для перегляду доступних екскурсій, реєстрації на них, ознайомлення з деталями маршрутів, датою та вартістю. Візуальна структура і навігація були розроблені з урахуванням потреб як туристів, так і організаторів подій.

У процесі розробки було створено такі ключові компоненти. Головна сторінка містить загальну інформацію про сервіс, а також короткий список популярних екскурсій з можливістю переходу до детального перегляду. Вона слугує відправною точкою для нового користувача.

Сторінка авторизації та реєстрації дозволяє створити новий обліковий запис або увійти до вже наявного профілю. Після успішної авторизації користувач отримує доступ до особистого кабінету, що розширює функціональність системи.

Сторінка екскурсій призначена для відображення повного списку доступних турів. Вона містить інструменти фільтрації за містом, датою проведення або категорією екскурсії, що полегшує пошук відповідної події.

Детальна сторінка екскурсії показує повну інформацію про конкретний тур: маршрут, ім’я гіда, тривалість, вартість, зображення та форму для запису. Це дозволяє користувачу ознайомитися з усіма деталями перед бронюванням.

Особистий кабінет користувача забезпечує доступ до історії реєстрацій, дозволяє редагувати персональні дані, а також скасовувати участь у забронюваних екскурсіях, що забезпечує зручність та контроль з боку

користувача. Компоненти було реалізовано з використанням бібліотеки React Router для організації маршрутизації, а також Axios для обміну даними з бекендом. Для адаптивності інтерфейсу застосовано бібліотеку Tailwind CSS, що забезпечує сучасний зовнішній вигляд сайту на різних пристроях.

На рисунку 3.1 зображено компонент TourCard.jsx, який відповідає за візуалізацію окремої картки екскурсії на сайті. Цей компонент використовується для відображення ключової інформації про тур – назви, міста проведення, тривалості, короткого опису, вартості та зображення.

```jsx
const TourCard = ({ tour }) => {
  return (
    <div className="max-w-sm rounded-2xl overflow-hidden shadow-md bg-white p-5 hover:shadow-xl transition duration-300">
      <img
        src={tour.image}
        alt={tour.title}
        className="w-full h-48 object-cover rounded-xl"
      />

      <div className="mt-4">
        <h2 className="text-xl font-semibold text-gray-800">{tour.title}</h2>

        <p className="text-sm text-gray-500 mt-1">
          {tour.city} &bull; {tour.duration} год
        </p>

        <p className="text-gray-700 mt-2 text-sm">
          {tour.description.length > 120
            ? tour.description.substring(0, 120) + '...'
            : tour.description}
        </p>

        <div className="mt-4 flex justify-between items-center">
          <span className="text-lg font-bold text-green-600">
            {tour.price} грн
          </span>

          <a
            href={`/tours/${tour.id}`}
            className="text-sm bg-blue-600 text-white px-4 py-2 rounded-xl hover:bg-blue-700 transition"
          >
            Детальніше
```

Рисунок 3.1 – Компонент TourCard

Компонент призначений для відображення інформації про окрему екскурсію та приймає необхідні дані через пропси (властивості), які передаються від батьківського компонента. Це дозволяє зробити компонент максимально універсальним і придатним для повторного використання з різними даними.

Серед основних пропсів, які очікує компонент, є назва туру (title), що відображає заголовок екскурсії, та короткий опис (description), який дає змогу ознайомитися з основною ідеєю заходу.

Також передається місто проведення (location), яке інформує користувача про місце збору або проведення екскурсії, та вартість (price), що є важливою характеристикою для ухвалення рішення про бронювання.

Крім того, компонент отримує тривалість туру (duration), яка дозволяє зрозуміти, скільки часу триватиме подія, URL зображення (image) – для візуального представлення туру, а також унікальний ідентифікатор (id), який необхідний для реалізації навігації до детальної сторінки екскурсії.

Весь цей набір даних формується у вигляді картки, яка оформлена за допомогою стилів TailwindCSS: з тінями, закругленням, hover-ефектами та структурованим макетом. Зображення розташовується у верхній частині картки, нижче йде назва екскурсії, коротка інформація, а в самому низу – кнопка «Детальніше», яка веде на окрему сторінку з розгорнутою інформацією про обрану екскурсію. Ця кнопка реалізована через Link із React Router, що забезпечує плавну маршрутизацію без перезавантаження сторінки.

Таким чином, компонент TourCard виконує одразу декілька важливих функцій. Він слугує узагальненою візуальною одиницею для виведення інформації про тур, полегшує взаємодію з контентом завдяки зрозумілій навігації, а також дозволяє швидко оцінити та порівняти кілька турів завдяки уніфікованому зовнішньому вигляду.

Таке компонування сприяє покращенню користувацького досвіду та допомагає забезпечити адаптивність і масштабованість інтерфейсу при додаванні нових турів.

На рисунку 3.2 зображено використання компонента TourCard, який демонструє, як конкретна екскурсія може бути передана до компонента у вигляді об’єкта з параметрами.

```
<TourCard
  tour={{
    id: '1',
    title: 'Пішохідна екскурсія по Львову',
    description: 'Цікава подорож старовинними вулицями міста...',
    image: '/images/lviv-tour.jpg',
    city: 'Львів',
    duration: 2,
    price: 250
  }}
/>
```

Рисунок 3.2 – Об’єкти для передачі даних

Такий підхід використовується для динамічного рендерингу інформації про тури, отриманих, наприклад, з бази даних або API.

У даному випадку компонент отримує проп tour, який містить повний опис однієї екскурсії:

- id: '1' – унікальний ідентифікатор екскурсії, що може використовуватись для побудови маршруту або ідентифікації в БД;
- title: 'Пішохідна екскурсія по Львову' – назва туру, яка буде відображена на картці;
- description: 'Цікава подорож старовинними вулицями міста...' – короткий опис, що мотивує користувача ознайомитись з деталями;
- image: '/images/lviv-tour.jpg' – шлях до зображення, яке візуально доповнює опис туру;
- city: 'Львів' – місто проведення, корисне для фільтрації або категоризації;
- duration: 2 – тривалість екскурсії у годинах або днях (залежно від контексту сайту);
- price: 250 – вартість участі в турі (у гривнях).

Цей ілюструє, як легко можна масштабувати застосунок: досить передати інший об’єкт з відповідними даними, і компонент автоматично відобразить нову екскурсію без потреби змінювати сам шаблон.

Крім того, використання компонента у вигляді тегу <TourCard /> сприяє підтримці чистого, модульного коду у React-проєкті, де кожен окремий тур представлений у вигляді незалежної, повторно використовуваної візуальної одиниці.

На рисунку 3.3 зображено компонент TourList, який виконує функцію відображення списку екскурсій на клієнтській частині вебсайту. Він є центральною частиною інтерфейсу користувача, адже забезпечує візуальне представлення турів у зручному та інформативному форматі. Кожна екскурсія представлена у вигляді окремої картки (використовується компонент TourCard), яка включає зображення туру, його назву, короткий опис, місто

проведення, тривалість та вартість. Це дозволяє користувачеві отримати загальне уявлення про тур перед переходом до детальнішої інформації.

TourList.jsx > ...

```jsx
import React, { useState } from 'react';
import TourCard from './TourCard';
import CityFilter from './CityFilter';

const TourList = ({ tours }) => {
  const [searchTerm, setSearchTerm] = useState('');
  const [selectedCity, setSelectedCity] = useState('');
  const [sortOrder, setSortOrder] = useState('asc');

  const handleCityChange = (city) => {
    setSelectedCity(city);
  };

  const handleSearchChange = (e) => {
    setSearchTerm(e.target.value);
  };

  const handleSortChange = (e) => {
    setSortOrder(e.target.value);
  };

  const filteredTours = tours
    .filter((tour) => {
      const matchesCity = selectedCity ? tour.city === selectedCity : true;
      const matchesSearch = tour.title.toLowerCase().includes(searchTerm.toLowerCase());
      return matchesCity && matchesSearch;
    })
    .sort((a, b) => {
      return sortOrder === 'asc' ? a.price - b.price : b.price - a.price;
    });

  const cities = Array.from(new Set(tours.map((t) => t.city)));

  return (
    <div className="p-4">
      <div className="mb-4 flex flex-col md:flex-row items-center gap-4">
```

Рисунок 3.3 – Компонент TourList

Другий важливий елемент функціоналу компонента – пошук по назві туру. За допомогою текстового поля користувач може ввести ключове слово, і компонент динамічно оновлює список турів відповідно до введеного запиту. Це значно спрощує навігацію особливо у випадку великої кількості доступних екскурсій.

Третім функціональним блоком є фільтрація за містом, яка реалізована через випадаючий список. Користувач має змогу обрати конкретне місто, після чого список оновлюється та відображає лише ті екскурсії, що проходять у вибраному регіоні. Це робить взаємодію з інтерфейсом логічною та гнучкою.

Окрім цього, реалізовано сортування за ціною, що дозволяє користувачеві впорядкувати екскурсії у зростаючому або спадному порядку за вартістю. Ця функція є особливо корисною для тих, хто планує бюджет подорожі.

Загалом компонент TourList має адаптивну верстку та побудований таким чином, щоб забезпечити приємний досвід користування з різних типів пристроїв – як настільних комп'ютерів, так і смартфонів або планшетів. Це робить його важливою частиною вебзастосунку, що підтримує гнучке та ефективне управління інформацією про доступні тури.

3.2 Реалізація серверної частини

Серверна частина вебзастосунку для організації та проведення екскурсій реалізована з використанням середовища виконання Node.js, що забезпечує неблокуючу подієво-орієнтовану архітектуру, ідеальну для вебсервісів з великою кількістю одночасних запитів. У якості вебфреймворку використано Express.js, який спрощує створення REST API, забезпечуючи ефективну маршрутизацію та обробку HTTP-запитів.

На рисунку 3.4 зображено приклад конфігурації серверної частини вебзастосунку, реалізованого на Node.js, який взаємодіє з базою даних SQLite.

```
const express = require('express');
const cors = require('cors');
const sqlite3 = require('sqlite3').verbose();
const path = require('path');

const app = express();
app.use(cors());
app.use(express.json());

// Ініціалізація підключення до бази SQLite
const dbPath = path.resolve(__dirname, 'database.sqlite');
const db = new sqlite3.Database(dbPath, (err) => {
  if (err) {
    console.error('Помилка при підключенні до SQLite:', err.message);
  } else {
    console.log('Підключено до SQLite бази даних.');
  }
});

db.serialize(() => {
  db.run(`
    CREATE TABLE IF NOT EXISTS tours (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      title TEXT NOT NULL,
      description TEXT,
      image TEXT,
      city TEXT,
      duration INTEGER,
      price REAL
    )
  `);
});
```

Рисунок 3.4 – Компонент server

У цьому фрагменті коду відбувається ініціалізація серверу, налаштування middleware та підключення маршрутизаторів, що обробляють HTTP-запити, пов’язані з екскурсіями.

Зокрема, використовуються такі основні компоненти:

- пакет express для створення вебсерверу і організації маршрутів;
- Middleware cors для забезпечення безперешкодної взаємодії між клієнтською частиною (наприклад, React-додатком) та сервером, який працює на різних доменах або портах;
- вбудований middleware express.json() для автоматичного розбору JSON у тілі запитів;
- підключення маршрутизаторів (tourRoutes), які визначають кінцеві точки API для роботи з екскурсіями;
- ініціалізація з’єднання з локальною базою даних SQLite, що відрізняється від MongoDB використанням файлової СУБД, з простою конфігурацією та невеликими вимогами до середовища.

Сервер запускається на порту 5000 (або вказаному у змінній середовища PORT) і готовий приймати запити від клієнтів, виконувати логіку обробки даних та взаємодіяти із базою.

Таким чином, цей код становить основу для побудови масштабованої серверної частини системи організації та проведення екскурсій із використанням Node.js та SQLite. Він забезпечує стабільне підключення до бази, обробку запитів та налаштування кросдоменної взаємодії.

Сервер запускається на порту 5000 (або вказаному у змінній середовища PORT) і готовий приймати запити від клієнтів, виконувати логіку обробки даних та взаємодіяти із базою.

Таким чином, цей код становить основу для побудови масштабованої серверної частини системи організації та проведення екскурсій із використанням Node.js та SQLite. Він забезпечує стабільне підключення до бази, обробку запитів та налаштування кросдоменної взаємодії.

На рисунку 3.5 зображено реалізацію моделі даних для екскурсійної системи з використанням бази даних SQLite у середовищі Node.js. Для роботи

з SQLite застосовано пакет sqlite3, який забезпечує створення таблиці tours з необхідними полями: id, title, description, image, city, duration та price.

```
1
2   const sqlite3 = require('sqlite3').verbose();
3   const path = require('path');
4
5   const dbPath = path.resolve(__dirname, '..', 'database.sqlite');
6   const db = new sqlite3.Database(dbPath);
7
8   // Створення таблиці
9   db.serialize(() => {
10    db.run(`
11      CREATE TABLE IF NOT EXISTS tours (
12        id INTEGER PRIMARY KEY AUTOINCREMENT,
13        title TEXT NOT NULL,
14        description TEXT,
15        image TEXT,
16        city TEXT,
17        duration INTEGER,
18        price REAL
19      )
20    `);
21  });
22
23  // Функції для роботи з таблицею tours
24
25  function getAllTours(callback) {
26    db.all('SELECT * FROM tours', [], (err, rows) => {
27      callback(err, rows);
28    });
29  }
30
31  function getTourById(id, callback) {
32    db.get('SELECT * FROM tours WHERE id = ?', [id], (err, row) => {
33      callback(err, row);
34    });
35  }
```

Рисунок 3.5 – Компонент tours

У коді продемонстровано основні функції для взаємодії з таблицею екскурсій: отримання списку всіх екскурсій, отримання екскурсії за ідентифікатором, створення нової екскурсії, оновлення існуючої та видалення. Кожна функція використовує відповідні SQL-запити для виконання CRUD-операцій.

Такий підхід дозволяє організувати зручний і простий у використанні інтерфейс для роботи з реляційною базою даних SQLite, що є оптимальним рішенням для невеликих або середніх за масштабом проєктів із збереженням структурованих даних.

На рисунку 3.6 зображено реалізацію маршрутизатора (router) для серверної частини застосунку, що працює з базою даних SQLite. У цьому коді показано обробку основних HTTP-запитів для роботи з екскурсіями:

отримання списку всіх турів, додавання нового туру та видалення туру за його ідентифікатором.

tourRoutes.js > ...

```
1   const express = require('express');
2   const router = express.Router();
3   const sqlite3 = require('sqlite3').verbose();
4   const db = new sqlite3.Database('./tours.db');
5
6   // Обгортка для роботи з базою через проміси
7   const dbAll = (sql, params=[]) => new Promise((resolve, reject) => {
8     db.all(sql, params, (err, rows) => {
9       if (err) reject(err);
10      else resolve(rows);
11    });
12  });
13
14  const dbGet = (sql, params=[]) => new Promise((resolve, reject) => {
15    db.get(sql, params, (err, row) => {
16      if (err) reject(err);
17      else resolve(row);
18    });
19  });
20
21  const dbRun = (sql, params=[]) => new Promise((resolve, reject) => {
22    db.run(sql, params, function(err) {
23      if (err) reject(err);
24      else resolve(this);
25    });
26  });
27
28  // GET all tours
29  router.get('/', async (req, res) => {
30    try {
31      const tours = await dbAll('SELECT * FROM tours');
32      res.json(tours);
33    } catch (err) {
34      res.status(500).json({ error: err.message });
35    }
36  });
```

Рисунок 3.6 – Компонент tourRoutes

Для взаємодії з SQLite використовується модуль sqlite3, а операції з базою даних обгорнуті у проміси для зручного застосування асинхронного синтаксису async/await. Такий підхід забезпечує читабельність коду та надійне оброблення результатів і помилок.

Важливою частиною є структура таблиці tours у базі даних, яка містить такі поля як: id (унікальний ідентифікатор), title (назва екскурсії), description (опис), image (посилання на зображення), city (місто проведення), duration (тривалість) та price (ціна).

Завдяки реалізації цього маршрутизатора сервер може обслуговувати запити клієнтської частини, що відображає інформацію про тури, дозволяє додавати нові екскурсії та видаляти застарілі, що є ключовим для функціоналу сайту з організації та проведення екскурсій.

3.3 Налаштування бази даних

Для ефективної роботи сайту з організації та проведення екскурсій та збереження інформації про екскурсії, користувачів, бронювання та відгуки було обрано легку вбудовану базу даних SQLite. Структура бази даних реалізує основні таблиці для зберігання даних про екскурсії, міста, користувачів, їхні ролі, бронювання та відгуки, що забезпечує надійну організацію та швидкий доступ до інформації для обробки запитів серверною частиною.

Особливу роль у системі відіграє таблиця bookings, яка зберігає інформацію про бронювання екскурсій користувачами. Детальне збереження даних про ідентифікатори користувачів, екскурсій, дати бронювання та статус дозволяє сайту підтримувати актуальність інформації, відстежувати активність користувачів і забезпечувати зручність у плануванні екскурсій.

Накопичення даних про популярні маршрути, активність користувачів та їхні відгуки відкриває можливості для аналізу вподобань і покращення якості послуг, що є важливим для подальшого розвитку проєкту. Збереження історії бронювань та взаємодій користувачів також забезпечує можливість аудиту і контролю діяльності, що підвищує безпеку та довіру до платформи.

Підключення вебзастосунку до бази даних здійснюється через популярні бібліотеки для роботи з SQLite, які забезпечують просте та швидке встановлення з’єднання, підтримку виконання SQL-запитів і транзакцій у вбудованому режимі.

Оскільки SQLite зберігає дані у вигляді файлу на файловій системі, це забезпечує кілька важливих переваг. По-перше, спрощується розгортання проєкту, оскільки немає потреби налаштовувати окремий сервер бази даних. По-друге, забезпечується висока портативність і можливість швидкого перенесення проєкту між різними середовищами. І нарешті, використання SQLite є зручним для тестування та локальної розробки, що робить його ідеальним рішенням для невеликих або початкових проєктів.

Конфігурація підключення до бази даних здійснюється шляхом вказання шляху до файлу бази в налаштуваннях застосунку, що підвищує гнучкість та безпеку системи, дозволяє легко масштабувати рішення і підтримувати його розвиток.

3.4 Тестування вебзастосунку

Тестування є важливим етапом життєвого циклу програмного забезпечення, що дозволяє перевірити правильність, надійність і відповідність реалізованої системи поставленим вимогам. У межах проєкту розробки вебзастосунку для організації та проведення екскурсій було застосовано декілька видів тестування, які охоплюють як окремі модулі, так і систему в цілому.

Модульне тестування полягало у перевірці окремих функціональних одиниць системи – компонентів інтерфейсу користувача (фронтенд) та обробників запитів на сервері (бекенд). Це дало змогу виявити локальні помилки та забезпечити правильну роботу кожного елементу окремо.

Інтеграційне тестування дозволило перевірити, як різні модулі взаємодіють між собою – зокрема, чи правильно передаються дані між клієнтською та серверною частинами, чи коректно працює логіка збереження та отримання інформації з бази даних.

Функціональне тестування було спрямоване на перевірку відповідності системи початковим вимогам. Було протестовано основні сценарії використання, як-от перегляд екскурсій, фільтрація за містами, оформлення бронювання тощо.

Тестування користувацького інтерфейсу (UI) дало змогу перевірити зручність використання застосунку, правильність розмітки, відповідність стилів та поведінку елементів при взаємодії.

Тестування надійності та стійкості до помилок включало перевірку роботи системи в нештатних ситуаціях – наприклад, при відсутності

підключення до бази даних, введенні некоректних даних користувачем або спробах несанкціонованого доступу.

На рисунку 3.7 зображено модульне тестування серверної частини вебзастосунку для організації та проведення екскурсій, що реалізований із використанням популярних бібліотек Jest та Supertest. Цей приклад ілюструє основні підходи до автоматичного тестування REST API, що забезпечує якість та стабільність роботи серверного коду.

```
5   describe('GET /api/tours', () => {
6     it('повертає список екскурсій у форматі JSON', async () => {
7       const response = await request(app).get('/api/tours');
8       expect(response.statusCode).toBe(200);
9       expect(response.headers['content-type']).toMatch(/json/);
10      expect(Array.isArray(response.body)).toBe(true);
11    });
12  });
13
14  describe('POST /api/tours', () => {
15    it('додає нову екскурсію', async () => {
16      const newTour = {
17        title: 'Тестова екскурсія',
18        description: 'Опис тестової екскурсії',
19        image: '/images/test.jpg',
20        city: 'Київ',
21        duration: 3,
22        price: 300
23      };
```

Рисунок 3.7 – Модульне тестування

У представленому тесті здійснюється перевірка двох ключових функціональних можливостей: отримання списку екскурсій (GET-запит) та додавання нової екскурсії до бази даних (POST-запит). Для цього тестовий код відправляє HTTP-запити до відповідних маршрутів сервера, а потім перевіряє коректність отриманих відповідей. Зокрема, перевіряються такі параметри, як HTTP-статус відповіді, формат і структура даних, а також відповідність фактичних даних очікуваним значенням.

Таке тестування дозволяє виявити помилки у логіці роботи серверу, неправильну обробку запитів, проблеми з підключенням до бази даних чи невідповідність формату відповіді. Регулярне автоматичне тестування покращує якість коду, пришвидшує процес розробки і знижує ризик появи багів у продуктивному середовищі.

Крім того, модульні тести є важливим інструментом для підтримки масштабованості проєкту: при додаванні нових функцій чи зміні архітектури розробники можуть швидко перевірити, що основний функціонал не було порушено. Усе це робить тестування невід'ємною частиною сучасної практики розробки вебзастосунків, зокрема в проєктах, пов'язаних з організацією та проведенням екскурсій.

На рисунку 3.8 зображено реалізацію автоматизованого тестування API вебзастосунку для управління екскурсіями, реалізованого на основі Node.js та фреймворку Express. Тестування здійснюється із застосуванням бібліотек Supertest та Jest, які широко використовуються для перевірки REST API та серверної логіки.

```
describe('GET /api/tours/:id', () => {
  it('повинен повернути екскурсію за id', async () => {
    const tourId = '1';
    const response = await request(app).get(`/api/tours/${tourId}`);

    expect(response.statusCode).toBe(200);
    expect(response.body).toHaveProperty('id', tourId);
    expect(response.body).toHaveProperty('title');
    expect(response.body).toHaveProperty('description');
  });

  it('повинен повернути 404, якщо екскурсія не знайдена', async () => {
    const invalidId = '9999';
    const response = await request(app).get(`/api/tours/${invalidId}`);

    expect(response.statusCode).toBe(404);
  });
```

Рисунок 3.8 – Тестування для отримання конкретної екскурсії

Перший тест перевіряє коректну відповідь сервера у випадку, коли екскурсія з заданим id існує в базі даних. Тест відправляє GET-запит на маршрут /api/tours/:id і очікує, що сервер поверне статус відповіді HTTP 200 (OK). Крім того, у тілі відповіді перевіряється наявність необхідних полів: id, title, description. Ці перевірки гарантують, що сервер коректно знаходить екскурсію та повертає повну інформацію про неї.

Другий тест імітує ситуацію, коли запитаний id екскурсії відсутній у базі. У цьому випадку сервер має повернути статус 404 (Not Found), що сигналізує клієнту про відсутність відповідного ресурсу. Цей тест перевіряє

стійкість системи до некоректних запитів і правильне опрацювання помилкових ситуацій.

Виконання таких тестів є важливим етапом у процесі розробки, оскільки вони дозволяють автоматизовано перевіряти працездатність функціоналу без необхідності ручного тестування. Це підвищує якість коду, допомагає своєчасно виявляти та виправляти помилки, а також забезпечує стабільність і надійність роботи вебзастосунку у подальшій експлуатації.

На рисунку 3.9 зображено реалізацію автоматизованого тестування оновлення даних екскурсії через REST API вебзастосунку. Тест реалізовано з використанням бібліотек Jest і Supertest, які дозволяють перевірити коректність роботи серверної частини, побудованої на Node.js та Express.

```
describe('PUT /api/tours/:id', () => {
  it('повинен оновити екскурсію і повернути оновлені дані', async () => {
    const tourId = '1';
    const updatedData = {
      title: 'Оновлена пішохідна екскурсія по Львову',
      price: 300
    };

    const response = await request(app)
      .put(`/api/tours/${tourId}`)
      .send(updatedData);

    expect(response.statusCode).toBe(200);
    expect(response.body).toHaveProperty('title', updatedData.title);
    expect(response.body).toHaveProperty('price', updatedData.price);
  });
});
```

Рисунок 3.9 – Тестування для оновлення екскурсій

У наведеній реалізації тестування виконує HTTP PUT-запит на маршрут /api/tours/:id, що відповідає за оновлення інформації про конкретну екскурсію за її унікальним ідентифікатором. В тілі запиту передаються нові значення полів, які необхідно змінити – у цьому випадку назва екскурсії (title) та її ціна (price).

Тест перевіряє, що відповідь сервера має статус 200, що свідчить про успішне оновлення запису. Крім того, у тілі відповіді підтверджується, що значення полів title і price дійсно були змінені на передані в запиті. Таким чином, цей тест гарантує коректність роботи функції редагування даних екскурсій у системі.

Застосування подібних тестів дозволяє автоматизувати контроль за змінами у функціоналі додатку, запобігти появі регресій та забезпечити стабільність роботи вебсервісу при подальшому розвитку. Це особливо важливо для підтримки якості програмного продукту, адже тестування допомагає своєчасно виявляти помилки та невідповідності, знижуючи ризики виходу в продуктивне середовище з багами.

Отже, на рисунку 3.9 представлено важливий етап перевірки сервісу оновлення екскурсій, який є невід'ємною частиною загальної системи тестування вебзастосунку для організації та проведення екскурсій.

На рисунку 3.10 зображено інтерфейс вебзастосунку, розробленого в рамках нашого проєкту для організації екскурсій територією України. Головна сторінка застосунку оформлена в приємній кольоровій гамі та містить ключові елементи взаємодії з користувачем: поле пошуку за назвою туру, випадаючі списки для фільтрації за містами та сортування за обраними критеріями.

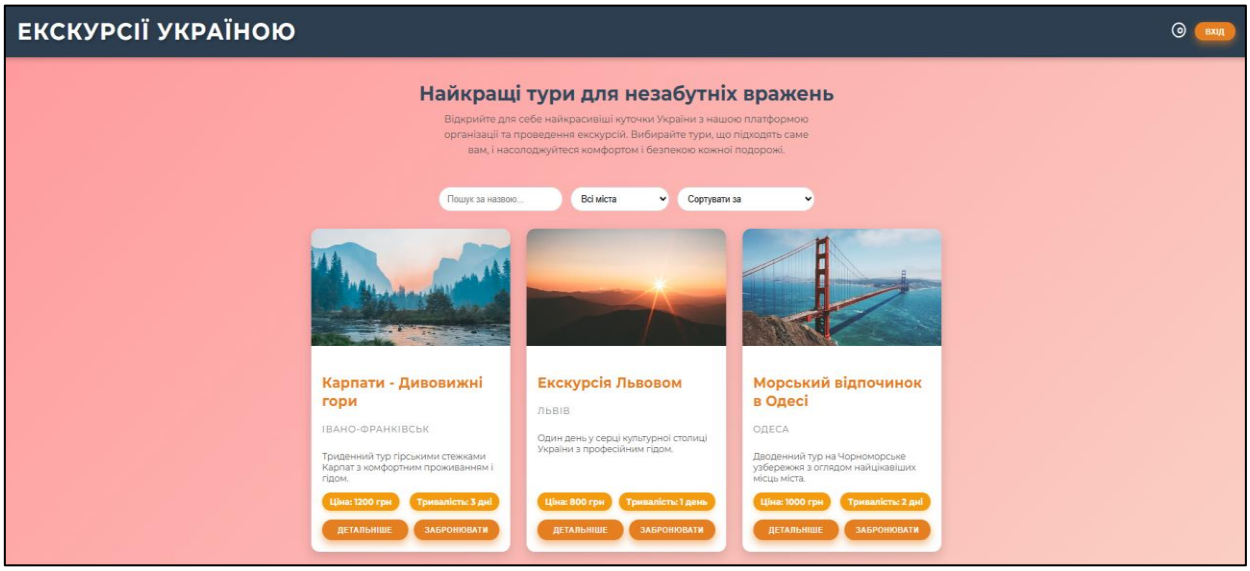


Рисунок 3.10 – Голована сторінка

Центральну частину займають інформаційні картки з турами, які автоматично формуються на основі бази даних. Кожна картка включає зображення, назву подорожі, місто проведення, короткий опис, тривалість туру, вартість, а також кнопки «Детальніше» та «Записатися» для переходу до відповідного функціоналу.

На рисунку 3.11 зображено інтерфейс сторінки реєстрації користувача вебсайту «Екскурсії Україною». Ця сторінка призначена для створення нового облікового запису, що дає змогу користувачам отримати повний доступ до функціоналу платформи, зокрема замовляти тури та користуватися персоналізованими сервісами.

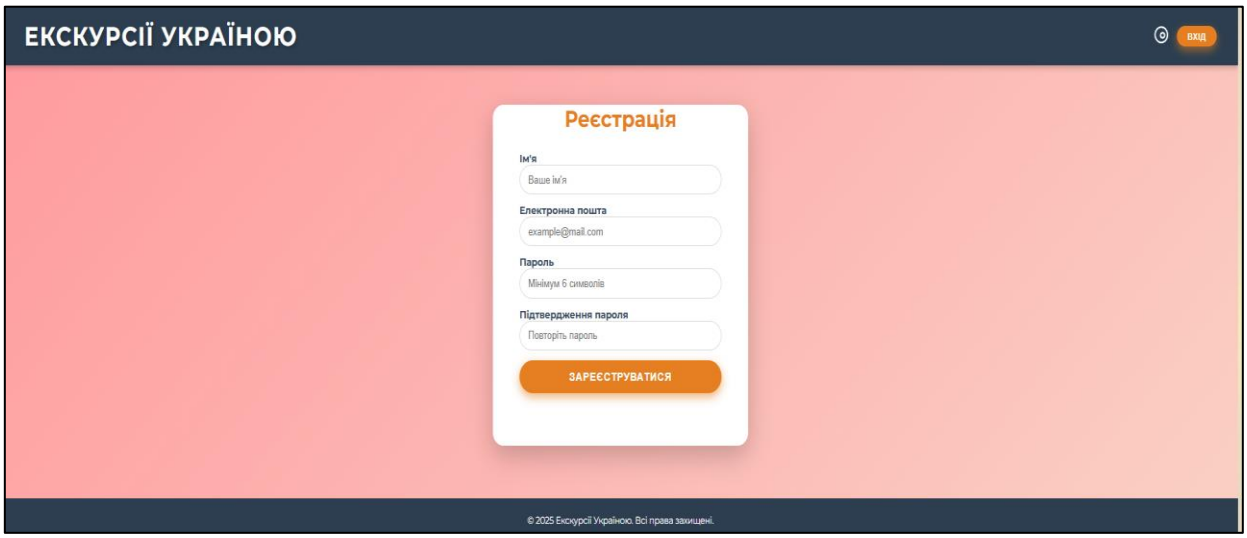


Рисунок 3.11 – Сторінка реєстрації

Верхня частина сторінки містить заголовок із назвою сайту, виконану у сучасному стилі з використанням шрифту Montserrat, що додає професійності та впізнаваності бренду. Поруч із заголовком розташована кнопка входу або виходу з облікового запису, а також іконка користувача, яка слугує швидким візуальним маркером для автентифікації. Це забезпечує зручність навігації та інтуїтивне керування профілем.

Основна частина сторінки представлена формою реєстрації, яка структурована у вигляді кількох полів введення: ім’я, електронна пошта, пароль і підтвердження пароля. Кожне поле має зрозумілі плейсхолдери та лейбли, що полегшує сприйняття та заповнення форми користувачем. Для покращення користувацького досвіду передбачено валідацію введених даних у режимі реального часу – у разі помилки під відповідним полем з’являється повідомлення з підказкою, наприклад, якщо пароль недостатньо складний або електронна пошта введена неправильно.

Після заповнення форми користувач може натиснути кнопку «Зареєструватися», яка стилізована у фірмових кольорах сайту з плавними переходами та ефектом наведенні, що стимулює до взаємодії. У разі успішної реєстрації система відображає підтверджувальне повідомлення або автоматично перенаправляє користувача на сторінку входу.

3.5 Висновки до розділу 3

У третьому розділі було розглянуто процес реалізації та тестування основних компонентів вебсайту для організації та проведення екскурсій, розробленого з використанням React для клієнтської частини, Node.js (Express) для серверної логіки та SQLite як системи керування базою даних. Основну увагу приділено інтеграції клієнтської та серверної частин, налаштуванню безпечної взаємодії з базою даних, а також реалізації механізмів керування екскурсіями, їх додавання, редагування та видалення.

У розділі також описано різні типи тестування, зокрема модульне, інтеграційне та функціональне, які були застосовані для перевірки коректності роботи основних компонентів системи. Було продемонстровано, як за допомогою інструментів Jest та Supertest забезпечується надійність, стабільність і правильність функціонування API та клієнтського інтерфейсу.

Особлива увага приділялась тестуванню API-методів, що відповідають за роботу з даними екскурсій – створення, отримання, оновлення та видалення. Завдяки комплексним тестам було підтверджено, що серверна частина коректно обробляє запити, а клієнтська – правильно відображає інформацію користувачу, що забезпечує стабільну та зручну роботу системи.

Таким чином, проведені розробка та тестування свідчать про те, що реалізований вебзастосунок для організації екскурсій є ефективним, масштабованим і готовим до подальшого впровадження у реальних умовах.

ВИСНОВКИ

У кваліфікаційній роботі розроблено вебзастосунок для організації та проведення екскурсій із використанням сучасного технологічного стеку – React для клієнтської частини, Node.js для серверної логіки та SQLite як системи керування базами даних. Основною метою розробки є забезпечення зручного інструменту для користувачів, який дозволяє переглядати доступні екскурсії, бронювати участь, отримувати інформацію про гідів, а також взаємодіяти з адміністрацією сайту через зворотний зв’язок.

У роботі проаналізовано функціональні та нефункціональні вимоги, на основі яких визначено основні цілі системи: доступність, зручність використання, швидкодія та безпека. На основі цих вимог було сформовано логічну та фізичну архітектуру системи, включаючи структуру бази даних та класи об’єктів.

Клієнтську частину реалізовано з використанням React, що дозволило створити адаптивний та інтуїтивно зрозумілий інтерфейс, включаючи динамічні компоненти для перегляду екскурсій, сторінок бронювання, особистого кабінету користувача, форм зворотного зв’язку та повідомлень. Інтерфейс взаємодіє з сервером через API-запити, забезпечуючи оновлення даних у реальному часі без перезавантаження сторінки.

Серверна частина побудована на основі Node.js, де реалізовано REST API, обробку запитів, маршрутизацію, валідацію введених даних та механізми автентифікації користувачів на основі токенів. Для зберігання даних використовується SQLite – легка, вбудована база даних, що ідеально підходить для проєктів невеликого та середнього масштабу. Також реалізовано механізми захисту персональних даних, обробку помилок та базове логування.

Було проведено тестування ключових компонентів системи за допомогою Jest для клієнтської частини та Mocha/Chai для серверної, що підтвердило працездатність і стабільність усіх модулів перед розгортанням.

Результатом роботи став функціональний вебсайт для організації екскурсій, який відповідає сучасним вимогам до подібних систем, має зручний інтерфейс, надійний бекенд та гнучку архітектуру, що дозволяє легко розширювати функціональність у майбутньому.

ПЕРЕЛІК ПОСИЛАНЬ

1. React Documentation – Офіційна документація React. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 01.04.2025).

2. TypeScript Documentation – Документація TypeScript. URL: https://www.typescriptlang.org/docs/ (дата звернення: 03.04.2025).

3. React Router Documentation – Документація React Router. URL: https://reactrouter.com/ (дата звернення: 03.04.2025).

4. Axios Documentation – Документація Axios. URL: https://axios-http.com/docs/intro (дата звернення: 10.04.2025).

5. Node.js Documentation – Офіційна документація Node.js. URL: https://nodejs.org/en/docs/ (дата звернення: 03.04.2025).

6. Express.js Documentation – Документація Express. URL: https://expressjs.com/ (дата звернення: 10.04.2025).

7. Sequelize Documentation – Документація Sequelize. URL: https://sequelize.org/master/ (дата звернення: 10.04.2025).

8. JWT.io Documentation – Документація JSON Web Token. URL: https://jwt.io/introduction (дата звернення: 15.04.2025).

9. Docker Documentation – Документація Docker. URL: https://docs.docker.com/ (дата звернення: 17.04.2025).

10. React Bootstrap Documentation – Документація React Bootstrap. URL: https://react-bootstrap.github.io/ (дата звернення: 20.04.2025).

11. OAuth 2.0 Documentation – Документація протоколу OAuth 2.0. URL: https://oauth.net/2/ (дата звернення: 23.04.2025).

12. ESLint Documentation – Документація ESLint. URL: https://eslint.org/docs/user-guide/getting-started (дата звернення: 23.04.2025).

13. GitHub Actions Documentation – Документація GitHub Actions. URL: https://docs.github.com/en/actions (дата звернення: 23.04.2025).

14. WebSocket API Documentation – Документація WebSocket API. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (дата звернення: 23.04.2025).

15. World Tourism Organization (UNWTO). International Tourism Highlights : The Impact of COVID-19 on Tourism (2020-2022). Madrid : UNWTO, 2023. 32 p. URL: https://voyagesafriq.com/wp-content/uploads/2023/09/Tourism_Highlights_2023.pdf (дата звернення: 23.04.2025).

16. UN Tourism Data Dashboard. *UN Tourism*. URL: https://www.unwto.org/tourism-data/un-tourism-tourism-dashboard (дата звернення: 23.04.2025).