Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: Розробка програмного засобу для організації подорожей

Виконав: студент IV курсу
групи 5ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Бондаренко Н. О.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Бабюк Н.П.
(прізвище та ініціали)

Рецензент:. к.т.н., доц. каф ОТ Колесник І.С.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Бондаренко Наталії Олександрівні

1. Тема роботи – розробка програмного засобу для організації подорожей. Керівник роботи: Бабюк Наталія Петрівна, д.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025.

3. Вихідні дані до роботи тип програми – вебзастосунок; вхідні дані: інформація про подорожі, користувацькі профілі, публікації, маршрути, геолокаційні дані; середовище розробки – IntelliJ IDEA, мови програмування – Java, JavaScript; фреймворки – Spring Boot, React.js з Material UI; системи керування базами даних – PostgreSQL, Redis; додаткові технології – Elasticsearch, WebSockets, Docker.

4. Зміст текстової частини: вступ; аналіз стану програмних засобів для організації подорожей; порівняльний аналіз аналогів; постановка задач дослідження; аналіз вхідних даних програмного засобу; розробка моделі та алгоритмів роботи програмного засобу; обґрунтування вибору засобів для реалізації програмного забезпечення; розробка модулів програмного засобу;

тестування програмного засобу; розробка інструкції користувача; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу: титульний слайд; мета, об'єкт та предмет дослідження; задачі дослідження; новизна і практична цінність отриманих результатів; архітектура системи та діаграми компонентів; схеми інтеграції з зовнішніми API; демонстрація інтерфейсу планування подорожей; демонстрація соціальних функцій та публікацій; візуалізація картографічних можливостей; фінальний слайд.

6. Консультанти розділів роботи.

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Бабюк Н. П., к.т.н., доцент кафедри ПЗ	24.03.2025	01.06.2025

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз розвитку технологій організації подорожей з використанням вебсистем та постановка задач дослідження	25.03.25 – 06.04.25	Вик.
2	Розробка моделі і алгоритмів вебсистеми для організації подорожей	07.04.25 – 18.04.25	Вик.
3	Розробка програмного забезпечення вебсистеми для організації подорожей	19.04.25 – 04.05.25	Вик.
4	Тестування вебсистеми для організації подорожей	05.05.25 – 18.05.25	Вик.
5	Оформлення матеріалів до захисту БКР	19.05.25 – 30.05.25	Вик.

Студент (підпис) **Н. О. Бондаренко** (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи (підпис) **Н. П. Бабюк** (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.42

Бондаренко Н.О. Розробка програмного засобу для організації подорожей : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 80 с.

На укр. мові. Бібліогр.: 28 назв.; рис.: 44; табл.: 2.

У бакалаврській кваліфікаційній роботі проведено дослідження та розробку вебсистеми для організації подорожей з інтеграцією соціальних функцій. Проаналізовано сучасні технології організації подорожей, визначено недоліки існуючих застосунків та обґрунтовано необхідність створення інтегрованої платформи для планування та обміну туристичним досвідом.

Запропоновано метод структуризації даних з розбивкою по днях та удосконалено метод соціальної взаємодії в туристичних вебсервісах.

Розроблено архітектуру системи на основі монолітного підходу, що включає модулі автентифікації, організації публікацій подорожей, інтеграції з зовнішніми API, комунікації в реальному часі та захисту даних. Програмне забезпечення реалізовано з використанням технологічного стеку Java, Spring Boot, PostgreSQL, ElasticSearch, Apache Kafka, Docker, JavaScript (React.js) та Redis. Проведено тестування системи, розроблено інструкцію користувача та визначено перспективи подальшого розвитку проєкту.

Розроблені програмні модулі можуть бути використані індивідуальними мандрівниками, туристичними агенціями та організаціями для детального планування та координації подорожей.

Ключові слова: вебсистема, організація подорожей, соціальна мережа, інтерактивні карти, структуризація даних, Java, React.js, Spring Boot, WebSockets, монолітна архітектура.

ABSTRACT

UDC 004.42

Bondarenko N.O. Development of a Software Tool for Travel Organization: bachelor's thesis in specialty 121 Software Engineering, educational program - Software Engineering. Vinnytsia: VNTU, 2025. 80 p.

In Ukrainian. Bibliography: 28 references; figures: 44; tables: 2.

The bachelor's thesis conducted research and development of a web system for travel organization with the integration of social functions. Modern travel organization technologies were analyzed, the shortcomings of existing solutions were identified, and the need to create an integrated platform for planning and sharing travel experiences was justified.

An improved method of data structuring with a breakdown by days was proposed, and the method of social interaction in tourist web services was enhanced.

The system architecture was developed based on a monolithic approach, including modules for authentication, travel publication organization, integration with external APIs, real-time communication, and data protection. The software was implemented using a technology stack of Java, Spring Boot, PostgreSQL, ElasticSearch, Apache Kafka, Docker, JavaScript (React.js), and Redis. System testing was conducted, a user manual was developed, and prospects for further project development were identified.

The developed software modules can be used by individual travelers, travel agencies, and organizations for effective travel planning and coordination.

Keywords: web system, travel organization, social network, interactive maps, data structuring, Java, React.js, Spring Boot, WebSockets, monolithic architecture.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі планування подорожей стає все складнішим процесом, особливо з огляду на зростаючі обсяги туристичної інформації та кількість доступних сервісів. Традиційні методи організації поїздок часто характеризуються фрагментарністю, коли мандрівники змушені використовувати численні платформи для різних аспектів подорожі – пошуку інформації про місця призначення, бронювання транспорту, житла, планування активностей. Це призводить до відчутних витрат часу та створює ризики неузгодженості між різними елементами маршруту [1].

Інтеграція всіх етапів організації подорожі в єдину вебсистему дозволить подолати ці обмеження, а поєднання функцій планування з соціальними механізмами обміну досвідом створить передумови для більш детального планування подорожі. Система на основі структурованих публікацій з розбивкою по днях корисна для створення чіткої хронології подорожі, візуалізації маршрутів на інтерактивних картах та обміну автентичним досвідом між мандрівниками. Розробка такої системи допоможе покращити процес планування та розуміння взаємозв'язків між різними компонентами подорожі і створить можливість для отримання персоналізованих рекомендацій.

Розробка даної системи є комплексним завданням через необхідність інтеграції з різноманітними зовнішніми API, забезпечення обробки великих обсягів даних, а також створення інтуїтивного інтерфейсу для користувачів [2]. Важливим аспектом є також забезпечення безпеки персональних даних, оскільки інформація про місцеперебування та маршрути користувачів вимагає надійного захисту від несанкціонованого доступу.

Схожі системи уже існують, але в них є певні недоліки: деякі фокусуються лише на окремих аспектах організації подорожей (бронювання готелів або транспорту), інші мають обмежені можливості соціальної взаємодії, деякі не надають інструментів для детального планування з розбивкою по днях. Таким

чином, розробка програмного засобу для організації подорожей є актуальною задачею.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення зручності планування та організації подорожей за рахунок розробки комплексної вебсистеми, що дозволяє інтегрувати всі етапи підготовки та документування туристичного досвіду.

Відповідно до поставленої мети необхідно вирішити такі завдання:

- провести аналіз існуючих методів і засобів організації подорожей;
- розробити архітектуру вебсистеми, яка забезпечить високу продуктивність, масштабованість та безпеку даних;
- розробити модуль автентифікації та управління користувачами з надійною ідентифікацією та захистом персональних даних;
- створити модуль для організації публікацій подорожей з можливістю детального планування по днях;
- реалізувати модуль інтеграції з зовнішніми API для отримання актуальної інформації про туристичні об'єкти;
- розробити модуль чатів у реальному часі для комунікації між користувачами;
- здійснити тестування системи згідно поставлених задач;
- розробити інструкцію користувача.

Об'єктом дослідження є процес організації та планування подорожей з використанням інформаційних технологій.

Предметом дослідження є методи та засоби розробки вебсистем для організації подорожей з інтеграцією соціальних функцій.

Методи дослідження. В процесі виконання роботи використовувалися методи проєктування архітектури програмного забезпечення для створення

вебсистеми, методи обробки даних для забезпечення швидкої роботи з інформацією про подорожі, методи веброзробки для створення інтуїтивного інтерфейсу користувача, алгоритми пошуку для реалізації персоналізованих рекомендацій.

Новизна отриманих результатів.

1. Вдосконалено метод структуризації даних для планування подорожей, який відрізняється від існуючого можливістю розбивки контенту на блоки за хронологічним принципом, що дозволяє створити комплексну модель даних з прив'язкою до часових проміжків, географічних координат та типів діяльності.

2. Вдосконалено метод соціальної взаємодії в туристичних вебсервісах, який на відміну від існуючих рішень інтегрує механізми комунікації, що дозволяють трансформувати пасивне споживання контенту в активну колаборацію через обговорення деталей маршрутів та спільне планування подорожей.

Практична цінність отриманих результатів. Розроблено вебсистему та алгоритми, за допомогою яких користувачі зможуть детально планувати подорожі, документувати туристичний досвід та обмінюватися ним з іншими мандрівниками. Окремі програмні модулі можуть бути використані розробниками для створення інших систем у сфері туризму та соціальних комунікацій.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: використання графових алгоритмів для побудови оптимальних маршрутів [1], безпека даних у сучасних додатках [2], методи забезпечення цілісності даних в розподілених системах [3].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні результати досліджень було представлено на конференціях «Комп'ютерні ігри і мультимедіа, як інноваційний підхід до комунікації» (2024) та «Стан, досягнення

і перспективи інформаційних систем і технологій» (2025), та були опубліковані у тезах конференцій [1-3].

Публікації. Основні результати досліджень було опубліковано у матеріалах конференцій «Стан, досягнення і перспективи інформаційних систем і технологій» (2025) [1-2] та «Комп`ютерні ігри та мультимедіа як інноваційний підхід до комунікації» (2024) [3].

Аналіз. У пояснювальній записці до бакалаврської кваліфікаційної роботи було розглянуто чотири розділи та було використано 28 літературних джерел.

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ ОРГАНІЗАЦІЇ ПОДОРОЖЕЙ З ВИКОРИСТАННЯМ ВЕБСИСТЕМ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану вебсистем для організації подорожей

Сучасний етап розвитку інформаційних технологій суттєво трансформує сферу туризму та організації подорожей, пропонуючи користувачам все більш зручні та інтегровані засоби. Традиційні методи планування подорожей, хоча і залишаються актуальними для певних категорій мандрівників, проте поступово витісняються комплексними платформами.

Одним із ключових аспектів розвитку технологій у сфері туризму є консолідація різноманітних сервісів на єдиних платформах. Сучасні вебсистеми дозволяють користувачам здійснювати весь цикл підготовки до подорожі – від пошуку ідей та натхнення до бронювання житла, транспорту та розваг, а також діляться своїм досвідом після повернення.

Інтеграція з картографічними сервісами становить важливу складову сучасних систем організації подорожей. OpenStreetMap пропонує значні переваги порівняно з комерційними аналогами [5]. Ключовою перевагою OpenStreetMap є можливість глибокої кастомізації карт під потреби туристичних додатків, включаючи виділення специфічних туристичних об'єктів, пішохідних маршрутів та локальних пам'яток, що часто відсутні на комерційних картах. Також OpenStreetMap надає доступ до детальних даних без обмежень та додаткових витрат на API-запити, що критично для легко масштабованого застосунку.

Сучасні системи для бронювання житла також еволюціонували. Інтеграція з HotelBeds API відкриває доступ до глобальної бази даних з 250,000 готелів та інших варіантів розміщення по всьому світу [6]. Ця API надає детальну інформацію про наявність номерів, ціни, зручності та рейтинги в режимі реального часу, забезпечуючи користувачів актуальними даними. Важливою

перевагою HotelBeds є конкурентні ціни завдяки прямим контрактам з готелями та можливість доступу до ексклюзивних пропозицій.

Ще одним важливим аспектом планування подорожей є пошук та відвідування локальних подій та розваг. TicketMaster API надає можливість для інтеграції інформації про концерти, фестивалі, спортивні події та інші розваги [7]. Ця API дозволяє здійснювати пошук за датами, місцем проведення, жанром та іншими критеріями, забезпечуючи персоналізовані рекомендації на основі інтересів користувача. Інтеграція з TicketMaster також надає можливість прямого бронювання квитків, що покращує користувацький досвід.

Соціальний аспект організації подорожей стає все більш значущим у сучасних вебсистемах. Можливість ділитися досвідом, публікувати фотографії, створювати огляди та взаємодіяти з іншими мандрівниками формує цілісні спільноти навколо туристичних платформ. Аналіз поведінки користувачів показує, що інтеграція соціальних функцій підвищує залученість та лояльність до платформи.

Отже, аналіз стану вебсистем для організації подорожей свідчить про зростаючу потребу в інтегрованих і адаптивних застосунках, що поєднують функції соціальної взаємодії, деталізованого планування та прямого бронювання послуг. Інтеграція з відкритими картографічними сервісами OpenStreetMap, глобальною системою бронювання житла HotelBeds та платформою для пошуку подій TicketMaster створює потужну екосистему для комплексного задоволення потреб сучасних мандрівників. Розробка вебсистеми, що об'єднує всі ці компоненти в інтуїтивно зрозумілому та соціально орієнтованому інтерфейсі, відповідає актуальним тенденціям ринку та має значний потенціал для залучення активної аудиторії.

1.2 Порівняльний аналіз аналогів

Використання вебсистеми для організації подорожей стає все більш популярним в останні роки, і ця тенденція посилюється завдяки інтеграції

соціальних функцій та комплексних інструментів планування. Розробка програмного засобу для організації подорожей з елементами соціальної мережі забезпечить користувачам можливість детального планування, обміну досвідом та взаємодії з однодумцями.

До найбільш відомих аналогів на ринку відносять Skyscanner, Rome2Rio, TripAdvisor, Maps.me та Airbnb, кожен з яких фокусується на певних аспектах організації подорожей, але жоден з них не пропонує повністю інтегрованого застосунку з поєднанням соціальних та планувальних функцій. Комплексний аналіз цих сервісів демонструє потребу в єдиній платформі, яка об'єднає всі етапи організації подорожі від планування до обміну враженнями в інтуїтивно зрозумілому соціальному форматі. Сучасні мандрівники використовують цифрові інструменти під час планування подорожей, та багато з них надають перевагу багатофункціональним платформам із можливістю персоналізації. Створення інтегрованої вебсистеми також сприятиме розвитку відповідального туризму, оскільки дозволить користувачам легко ділитися інформацією про подорожі у різних регіонах світу.

Maps.me – це офлайн-картографічний сервіс [8], який дозволяє користувачам завантажувати детальні карти регіонів для подальшого використання без доступу до інтернету (див. рисунок 1.1). Застосунок базується на даних OpenStreetMap і пропонує глобальне покриття з детальною інформацією про дороги, пішохідні шляхи, пам'ятки, ресторани, готелі та інші точки інтересу [5]. Maps.me призначений для мандрівників, які потребують надійної навігації під час подорожей без постійного підключення до мережі або при обмеженому роумінгу. Функціонал включає маршрутизацію для автомобілів, пішоходів та велосипедистів, детальні карти з пошуком по назві місця або адресі, закладки для збереження важливих місць, голосову навігацію та можливість дізнатися відстань до обраних точок , що показано на рисунок 1.2. Додаток також пропонує інформацію про місцеві пам'ятки та маршрути для туристів.

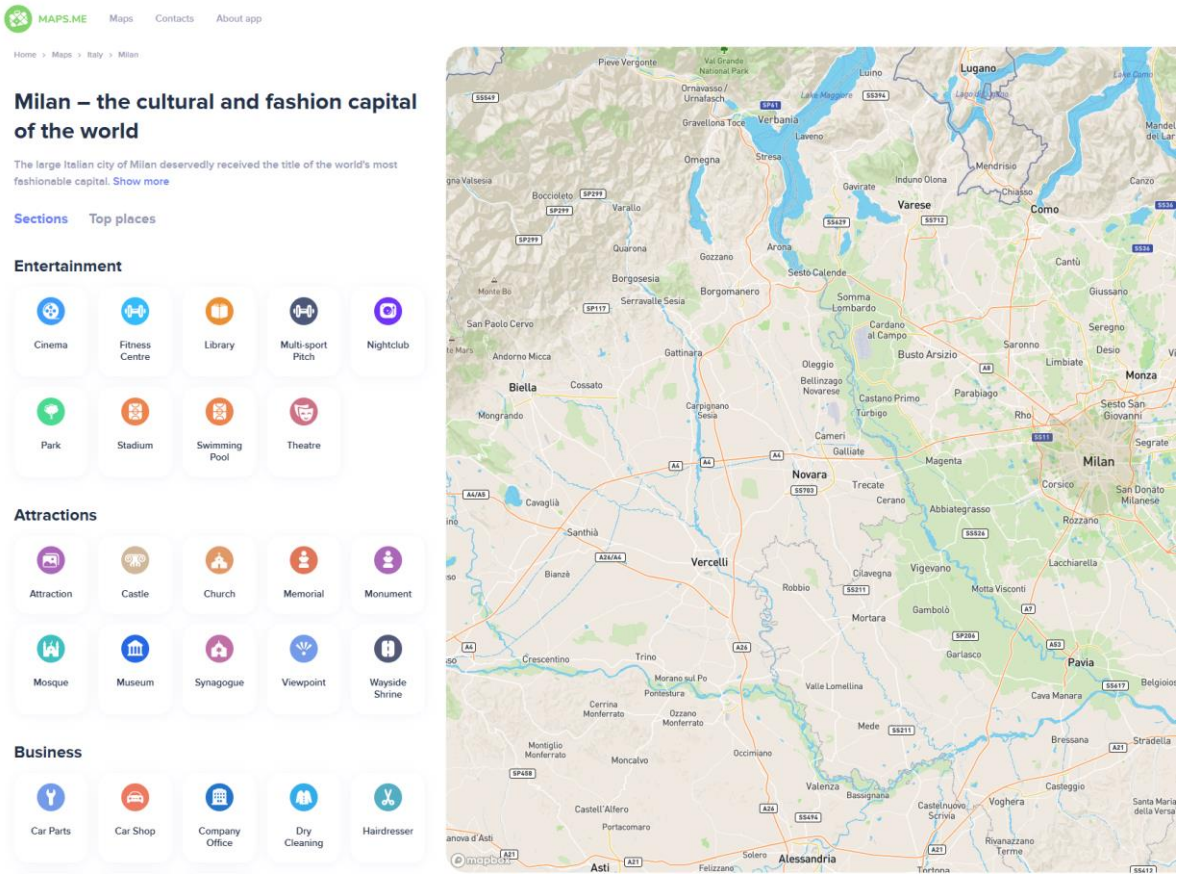

Рисунок 1.1 – Інтерфейс застосунку Maps.me

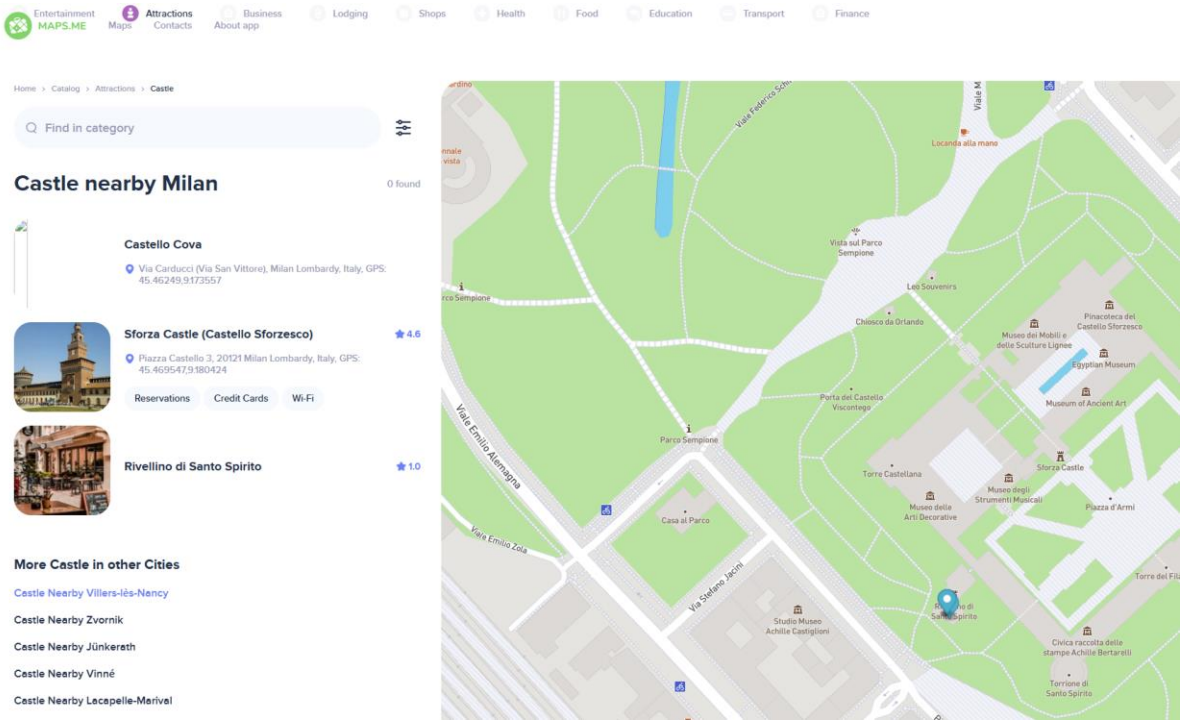

Рисунок 1.2 – Приклад роботи застосунку Maps.me

TripAdvisor – це всеохоплюючий туристичний сервіс [9], що поєднує величезну базу відгуків користувачів про готелі, ресторани, пам'ятки та розваги по всьому світу (див. рисунки 1.3 і 1.4). Додаток надає інструменти для планування подорожей, включаючи пошук та бронювання житла, ресторанів, авіаквитків і турів. TripAdvisor призначений для допомоги мандрівникам у пошуку локацій та розваг на основі реального досвіду інших людей. Функціонал включає рейтингову систему, детальні відгуки з фотографіями, форуми для обговорення подорожей, карти поблизьких місць та атракцій, можливість збереження улюблених місць у персоналізованих списках та колекціях. Сервіс також пропонує функції порівняння цін від різних постачальників туристичних послуг та можливість бронювання безпосередньо через застосунок.

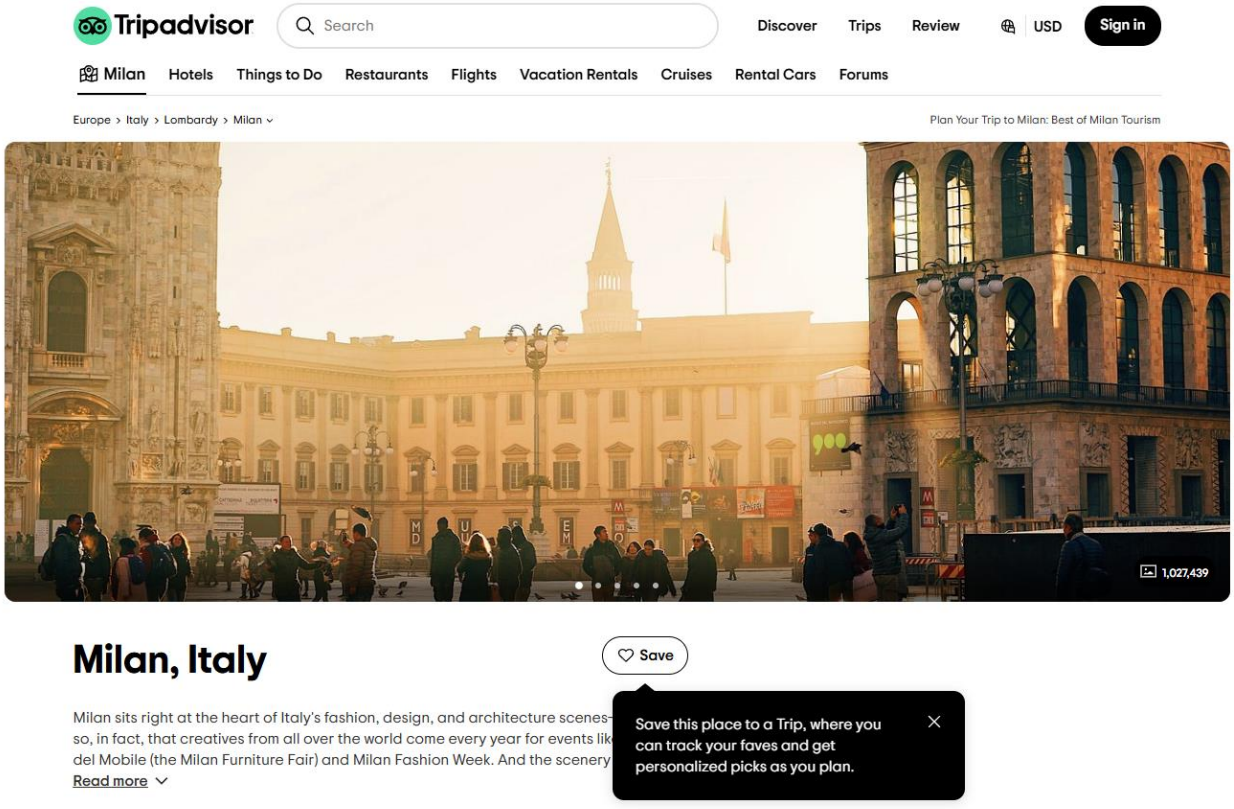

Рисунок 1.3 – Інтерфейс застосунку TripAdvisor

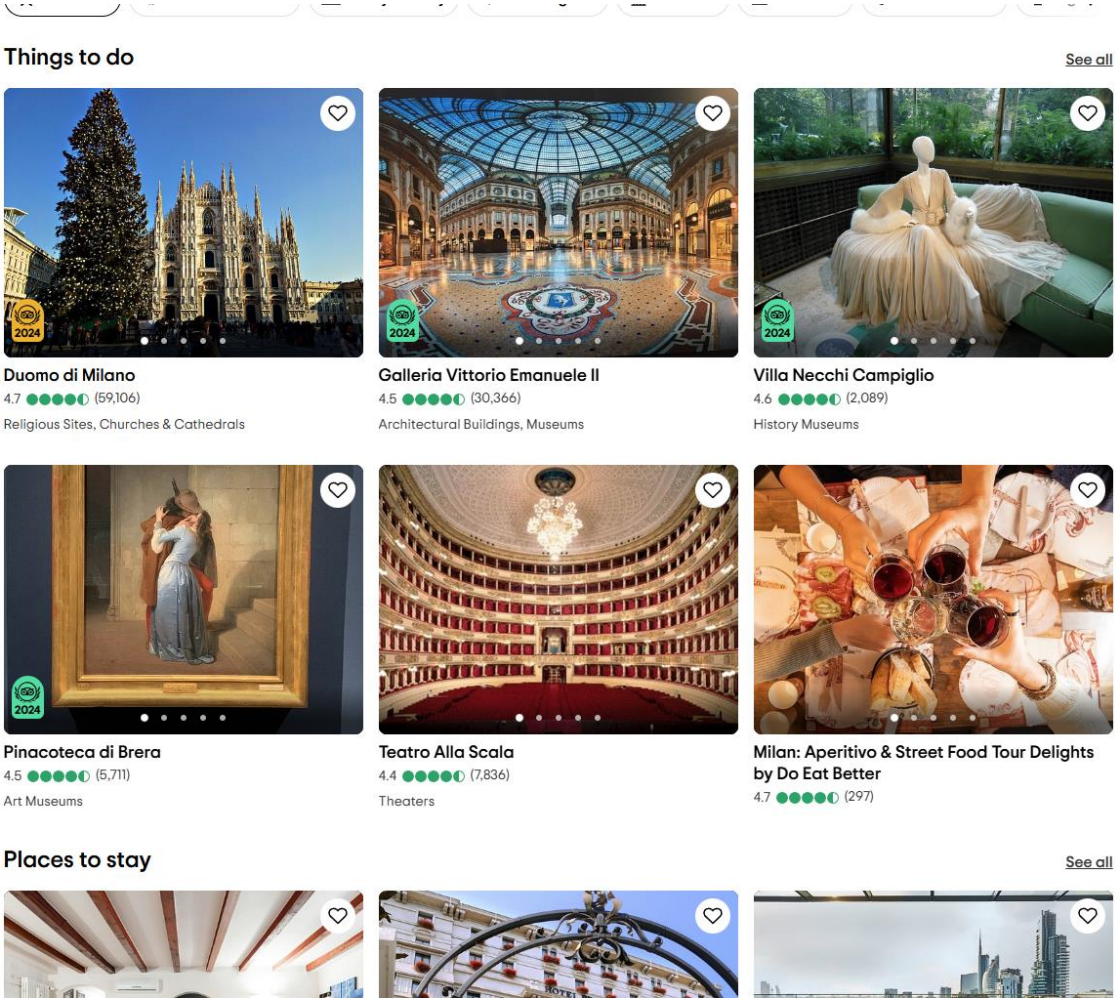

Рисунок 1.4 – Приклад роботи застосунку TripAdvisor

Rome2Rio – це сервіс планування мультимодальних подорожей [10], який дозволяє користувачам знаходити оптимальні маршрути між будь-якими двома точками світу, комбінуючи різні види транспорту (див. рисунок 1.5). Застосунок інтегрує дані про авіаперельоти, потяги, автобуси, пороми, таксі та навіть пішохідні маршрути, пропонуючи повний спектр можливих варіантів подорожі з розрахунком приблизної вартості та часу в дорозі. Rome2Rio призначений для мандрівників, які шукають найбільш швидкі способи дістатися з однієї точки в іншу, особливо при складних маршрутах. Функціонал включає детальну інформацію про кожен сегмент подорожі, карти маршрутів, порівняння альтернативних варіантів та можливість бронювання деяких видів транспорту безпосередньо через застосунок або партнерські сайти.

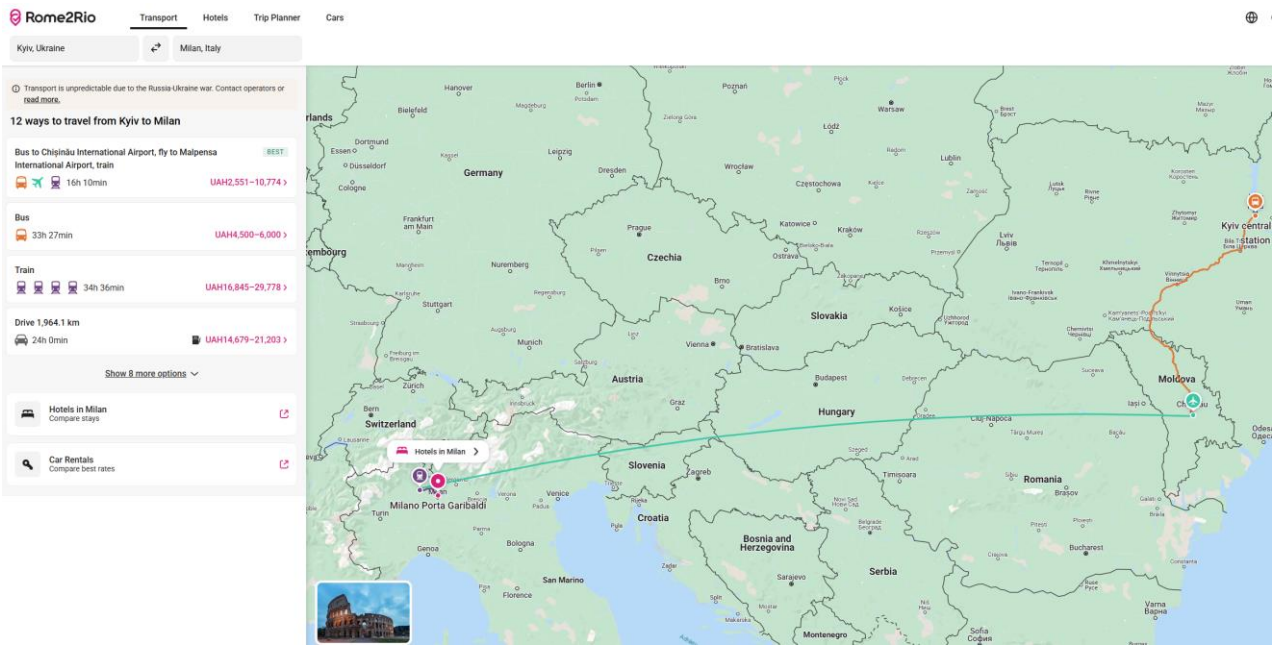

Рисунок 1.5 – Приклад роботи застосунку Rome2Rio

Skyscanner – це потужний інструмент для пошуку та порівняння цін на авіаквитки [11], готелі та оренду автомобілів (див. рисунок 1.6). Він агрегує інформацію від сотень авіакомпаній та туристичних агентств, дозволяючи користувачам знаходити найвигідніші пропозиції. Застосунок пропонує функції гнучкого пошуку за датами, можливість встановлення сповіщень про зміну цін, порівняння різних маршрутів та фільтрацію результатів за багатьма параметрами. Skyscanner призначений для економії часу та грошей при плануванні подорожей, особливо для користувачів, які шукають найвигідніші пропозиції на авіаперельоти. Функціонал включає пошук скрізь для спонтанних подорожей, багатомовний інтерфейс та перенаправлення для бронювання на офіційні сайти постачальників послуг. Варто відзначити, що застосунок має інтуїтивно зрозумілий інтерфейс та високу швидкодію навіть при обробці складних запитів з численними параметрами. Skyscanner також розробив потужну систему рекомендацій, що базується на аналізі поведінки користувачів та історії пошуку, пропонуючи персоналізовані варіанти подорожей. Особливістю застосунку є унікальний алгоритм пошуку комбінованих

маршрутів, який дозволяє економити значну суму вартості подорожі, поєднуючи рейси різних авіакомпаній.

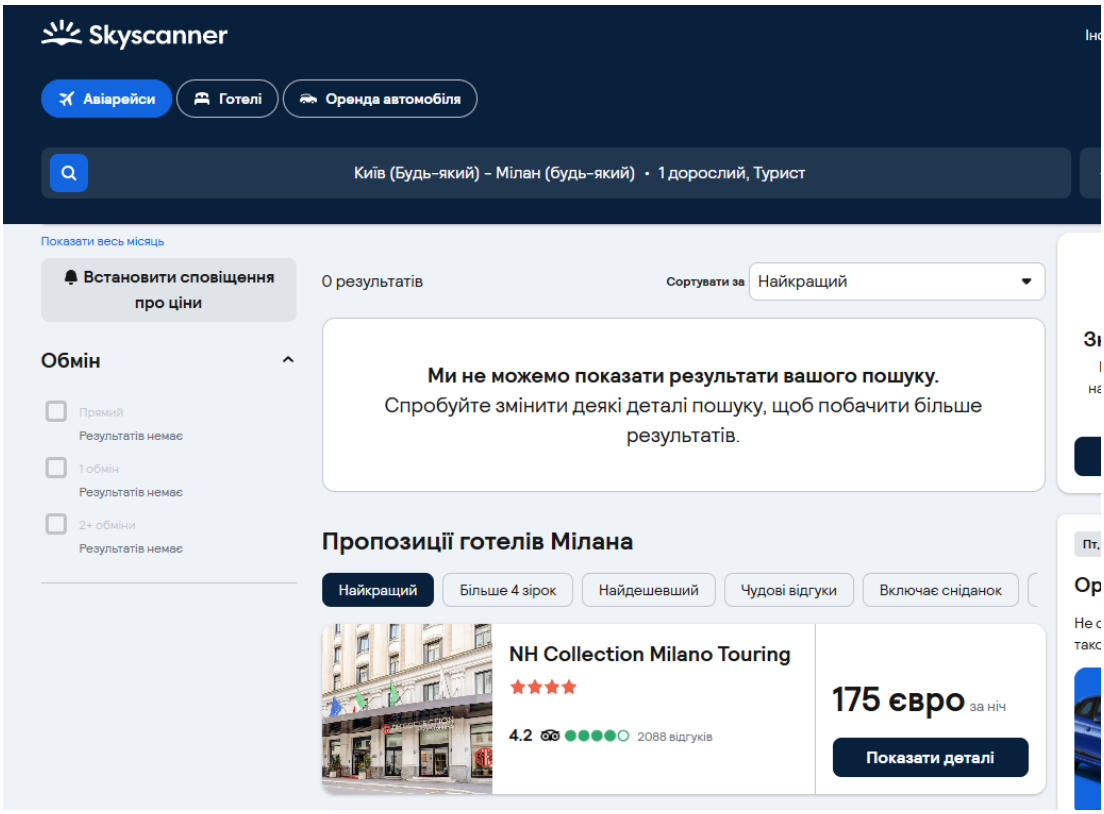

Рисунок 1.6 – Приклад роботи застосунку Skyscanner

Airbnb – це платформа для оренди житла, яка з'єднує людей [12], які хочуть здати своє житло в оренду, з тими, хто шукає місце для тимчасового проживання по всьому світу (див. рисунок 1.7). Застосунок пропонує широкий вибір варіантів розміщення – від кімнат у спільних квартирах до розкішних вілл та унікальних помешкань. Airbnb призначений для забезпечення автентичного досвіду проживання під час подорожей та економічної альтернативи традиційним готелям. Функціонал включає розширений пошук за численними фільтрами, систему бронювання з захищеними платежами, профілі господарів та гостей з відгуками та рейтингами, миттєві повідомлення для комунікації, інтерактивні карти районів та інструменти для господарів щодо управління своїми оголошеннями. Останніми роками Airbnb також додав можливість бронювання

«вражень» – місцевих активностей та екскурсій, організованих місцевими жителями.

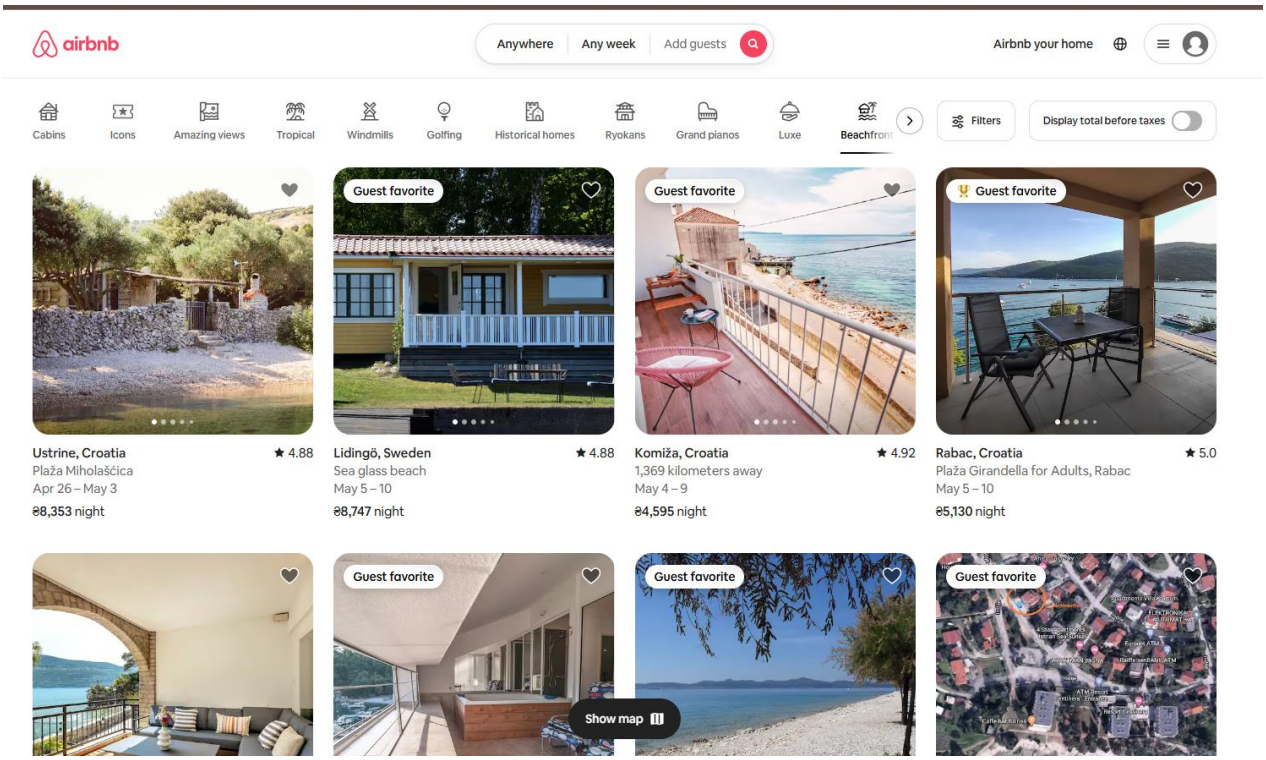

Рисунок 1.7 – Приклад роботи застосунку Airbnb

Розробка програмного засобу для організації подорожей, що поєднує функції планування маршрутів із соціальною мережею в стилі Instagram. Програмний засіб надає інструменти для створення детальних маршрутів з розбивкою по днях, інтеграцією з картами, пошуком житла, ресторанів та розваг, а також для публікації та обміну власним досвідом подорожей. Застосунок призначений для мандрівників, які хочуть не лише спланувати свою подорож, але й поділитися нею з іншими та отримати натхнення від публікацій інших користувачів. Функціонал включає стрічку публікацій у стилі Instagram, систему підписок на інших мандрівників, персональні та групові чати для обговорення деталей подорожі, інтеграцію з Google Maps для візуалізації маршрутів, прогноз погоди для планування, колаборативне редагування маршрутів та можливість бронювання житла і транспорту через інтеграції з відповідними сервісами.

Унікальність додатку полягає в поєднанні практичних інструментів планування з соціальними функціями в єдиній екосистемі.

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	Skyscanner	Rome2Rio	TripAdvisor	Maps.me	Airbnb	Власна розробка
Соціальна взаємодія (можливість створювати профілі, публікувати контент та взаємодіяти з іншими користувачам)	-	-	+	-	+	+
Стрічка публікацій	-	-	-	-	-	+
Деталізоване планування (можливість створювати подорож з розбивкою на дні з детальним описом)	-	+	-	-	-	+
Маршрутизація на мапі	-	+	+	+	-	+
Пошук житла	+	+	+	+	+	+
Пошук транспорту	+	+	+	+	-	+
Персональний та груповий чати	-	-	-	-	-	+
Загальна оцінка	28.5%	57.1%	57,1%	42.8%	28.5%	100%

Відповідно до таблиці порівняльних характеристик розробка власного програмного засобу для організації подорожей є доцільною. Отриманий продукт

зможе покрити недоліки існуючих застосунків та забезпечити розширені можливості для обговорення деталей подорожі та поширення власного досвіду.

Отже, розробка програмного засобу для організації подорожей з елементами соціальної мережі відкриває перед користувачами широкі можливості для покращення планування, документування та обміну досвідом подорожей. Завдяки інтеграції функцій соціальної взаємодії, детального планування маршрутів по днях та різноманітних API для бронювання, користувачі можуть створювати персоналізовані подорожі та поширювати їх у зручний спосіб серед друзів чи широкої аудиторії. Такий комплексний підхід дозволяє не лише покращити процес підготовки до поїздки через об'єднання всіх необхідних інструментів на одній платформі, але й створює унікальну екосистему, де натхнення від публікацій інших мандрівників безпосередньо трансформується у власні маршрути, обговорення деталей у групових чатах та врахування актуальних даних про погоду, транспорт та житло.

1.3 Аналіз методів розв’язання задачі

Розробка вебсистеми для організації подорожей з інтеграцією соціальних функцій вимагає комплексного підходу до розв’язання різноманітних технічних задачі. У цьому розділі проаналізовано методи розробки ключових компонентів системи та обґрунтовано вибір оптимальних реалізацій.

Один з основних аспектів розробки додатку для організації подорожей – це створення архітектури даних для зберігання та обробки інформації про маршрути, користувачів та їхні взаємодії. Для цього можна використовувати монолітну архітектуру з єдиною реляційною базою даних, яка забезпечує високу узгодженість даних та простоту розробки. Однак такий підхід має суттєві недоліки при масштабуванні та обробці різнорідних типів даних, характерних для соціальних мереж та систем планування подорожей. Тому монолітна архітектура не є оптимальним вибором для такого комплексного додатку.

Інший підхід полягає у використанні мікросервісної архітектури, де кожен мікросервіс використовує найбільш підходящу для своїх потреб базу даних. Наприклад, для зберігання структурованих даних про користувачів та їхні подорожі доцільно використовувати PostgreSQL, для неструктурованих даних, таких як відгуки та коментарі – MongoDB, а для швидкої обробки даних у реальному часі – Redis [13]. Така архітектура забезпечує високу гнучкість та масштабованість, проте вимагає складнішої інфраструктури та підходів до забезпечення узгодженості даних між різними сервісами.

Для реалізації геопросторових функцій, таких як відображення маршрутів та пошук найближчих місць, найбільш оптимальним є використання OpenStreetMap у поєднанні з бібліотекою Leaflet [5]. Ця платформа надає повноцінний доступ до геопросторових даних без обмежень за кількістю запитів, що є суттєвою перевагою для проектів будь-якого масштабу. На відміну від Google Maps API, OpenStreetMap спирається на спільноту користувачів, яка постійно оновлює та покращує дані, забезпечуючи високу точність у багатьох регіонах. Крім того, відкрита модель розвитку дозволяє адаптувати мапи під конкретні потреби, що робить OpenStreetMap ідеальним вибором для розробників, які цінують гнучкість та незалежність від комерційних сервісів.

Особливу увагу необхідно приділити методам забезпечення безпеки даних користувачів. Одним із ключових викликів є безпечне зберігання паролів та конфіденційної інформації. Традиційні методи зберігання паролів у відкритому вигляді або з використанням простого хешування вже не відповідають сучасним вимогам безпеки. Натомість рекомендується використовувати алгоритми хешування з впровадженням солі та перцю. Для хешування доцільно застосовувати алгоритми, стійкі до атак перебором, такі як Argon2, Bcrypt або PBKDF2. Ці алгоритми спеціально розроблені для повільного виконання, що ускладнює брутфорс-атаки [2].

Інтеграція з зовнішніми сервісами, такими як Hotelbeds API [6] або Ticketmaster API [7], вимагає надійного зберігання API-ключів та токенів

доступу. Зберігати такі токени у відкритому вигляді в базі даних небезпечно. Натомість доцільно використовувати шифрування на основі алгоритмів AES-256 або ChaCha20-Poly1305 з правильним управлінням ключами. Ключі шифрування повинні зберігатися в окремих захищених сховищах, таких як HashiCorp Vault, AWS KMS або Azure Key Vault [14].

Для реалізації функцій соціальної мережі необхідно вибрати підхід до організації стрічки публікацій. Існують два основних алгоритми: хронологічне відображення та персоналізоване ранжування. Хронологічне відображення простіше в реалізації, але менш зручне з точки зору залученості користувачів. Персоналізоване ранжування на основі машинного навчання, яке враховує інтереси користувача, історію взаємодій та популярність контенту, забезпечує вищу релевантність вмісту, але вимагає більших обчислювальних ресурсів та складнішої архітектури. На початкових етапах розробки доцільно використовувати гібридний підхід, де базове хронологічне сортування доповнюється простими факторами ранжування, такими як близькість інтересів та популярність контенту.

Для організації комунікацій у реальному часі в чатах найкращим є використання технології WebSockets [15], яка забезпечує двосторонній зв'язок між клієнтом та сервером з мінімальними затримками. Альтернативними підходами є довгі HTTP-опитування (long polling) або Server-Sent Events, але вони мають обмеження в масштабованості та використанні ресурсів. Інтеграція з протоколом MQTT може бути доцільною для додаткових функцій, таких як трекінг геолокації членів групи в реальному часі.

Транспортне шифрування даних є ще одним важливим компонентом захисту. Усі комунікації повинні відбуватися через захищені канали з використанням протоколу TLS 1.3 з сучасними налаштуваннями шифрів. Варто застосовувати технології HSTS (HTTP Strict Transport Security) для запобігання атакам зниження рівня захисту та впроваджувати пінінг сертифікатів для захисту від атак з підміною сертифікатів.

Розглянувши переваги та недоліки кожного методу, було обрано монолітну архітектуру, інтеграцію OpenStreetMap для геопросторових функцій, використання WebSockets для комунікацій у реальному часі та реалізацію комплексної системи безпеки з використанням сучасних криптографічних алгоритмів [15]. Такий підхід забезпечить оптимальний баланс між функціональністю, масштабованістю, продуктивністю та безпекою додатку для організації подорожей з елементами соціальної мережі.

1.4 Постановка задач дослідження

Проаналізувавши переваги та недоліки існуючих систем для організації подорожей та враховуючи результати аналізу технологій [16], було визначено такі завдання, які необхідно виконати для успішної розробки вебсистеми:

1. Реалізувати відображення маршрутів на OpenStreetMap.
2. Реалізувати комплексну систему безпеки даних, що включає захист особистої інформації користувачів, шифрування конфіденційних даних та безпечне зберігання API-ключів для інтеграцій з зовнішніми сервісами.
3. Розробити алгоритм та програмний модуль формування стрічки публікацій у стилі соціальної мережі.
4. Створити структуру даних та інтерфейс для створення публікацій, розбитих по днях подорожі, з можливістю додавання деталей про місця, заклади харчування, проживання та транспорт.
5. Реалізувати систему комунікацій у реальному часі з індивідуальними та груповими чатами для обговорення деталей подорожі.
6. Інтегрувати модулі пошуку та бронювання житла через HotelBeds API, забезпечуючи отримання актуальної інформації про наявність та ціни [6].
7. Розробити компонент для пошуку та відображення локальних подій та розваг з використанням TicketMaster API [7].
8. Розробити інтерфейс користувача, який буде інтуїтивно зрозумілим та адаптивним для різних пристроїв.

9. Провести тестування вебсистеми згідно поставлених задач та вимог до безпеки даних.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було розглянуто стан технологій організації подорожей із використанням вебсистем та проведено порівняльний аналіз існуючих застосунків на ринку. Були проаналізовані такі платформи як Skyscanner, Rome2Rio, TripAdvisor, Maps.me та Airbnb. Кожна з цих систем має свої переваги і недоліки у контексті планування подорожей, соціальної взаємодії, комунікаційних можливостей та інтеграцій з зовнішніми сервісами.

Особливу увагу було приділено аналізу технологій, які будуть використовуватися для розробки вебсистеми, зокрема OpenStreetMap для відображення маршрутів, HotelBeds API для пошуку житла та TicketMaster API для отримання інформації про події. Було розглянуто методи розв’язання основних технічних задач, включаючи вибір архітектури додатку, підходи до забезпечення безпеки даних, організації соціальних функцій та комунікації у реальному часі.

На основі проведеного аналізу було обґрунтовано вибір монолітної архітектури з поліглотним збереженням даних, використання WebSockets для комунікацій у реальному часі, алгоритмів операційних перетворень для колаборативного редагування та комплексної системи безпеки з використанням сучасних криптографічних методів.

Проаналізувавши переваги та недоліки існуючих систем, було сформульовано чіткий перелік завдань для розробки власної вебсистеми організації подорожей, що поєднує функції соціальної мережі з інструментами детального планування маршрутів. Таким чином, було доведено доцільність розробки власного застосунку, що заповнить існуючу нішу на ринку туристичних сервісів.

2 РОЗРОБКА МОДЕЛІ І АЛГОРИТМІВ ВЕБСИСТЕМИ ДЛЯ ОРГАНІЗАЦІЇ ПОДОРОЖЕЙ

2.1 Структура та алгоритм створення публікацій подорожей

Для забезпечення процесу створення та публікації подорожей у вебсистемі необхідно реалізувати структурований алгоритм, який координуватиме взаємодію між клієнтською частиною, серверними компонентами та базою даних. Розроблений алгоритм створення подорожі забезпечує послідовне виконання всіх необхідних етапів: від початкового введення базової інформації до фінальної публікації готового маршруту. Такий підхід дозволяє користувачам створювати детальні плани подорожей з розбивкою по днях, автоматичним розрахунком бюджету та інтеграцією з картографічними сервісами для візуалізації маршрутів. Алгоритм буде використовуватися як основний механізм для обробки користувацьких запитів на створення нових подорожей, забезпечуючи цілісність даних, валідацію введеної інформації та коректне збереження всіх компонентів подорожі в базі даних системи.

Представлений алгоритм демонструє процес створення нової подорожі у розробленій вебсистемі для організації подорожей (див. рисунок 2.1). Алгоритм реалізований у вигляді послідовної діаграми взаємодії між клієнтською частиною (React Клієнт), серверною частиною (Travel Controller та Travel Service) та базою даних (PostgreSQL Database).

Процес починається з ініціювання користувачем створення нової подорожі через клієнтський інтерфейс. Користувач надає основну інформацію про майбутню подорож, після чого клієнт формує POST-запит до ендпоінту /api/travels/basic-info з необхідними даними, такими як назва подорожі, дати початку та завершення. Контролер обробляє отриманий запит і створює новий об'єкт TravelPlanBasicInfo, який передається до сервісного шару для подальшої обробки.

На рівні сервісу відбувається валідація отриманих даних та підготовка до збереження в базі даних. Сервіс формує SQL-запит для вставки нового запису в таблицю travel з унікальним ідентифікатором travel_id: 12345.

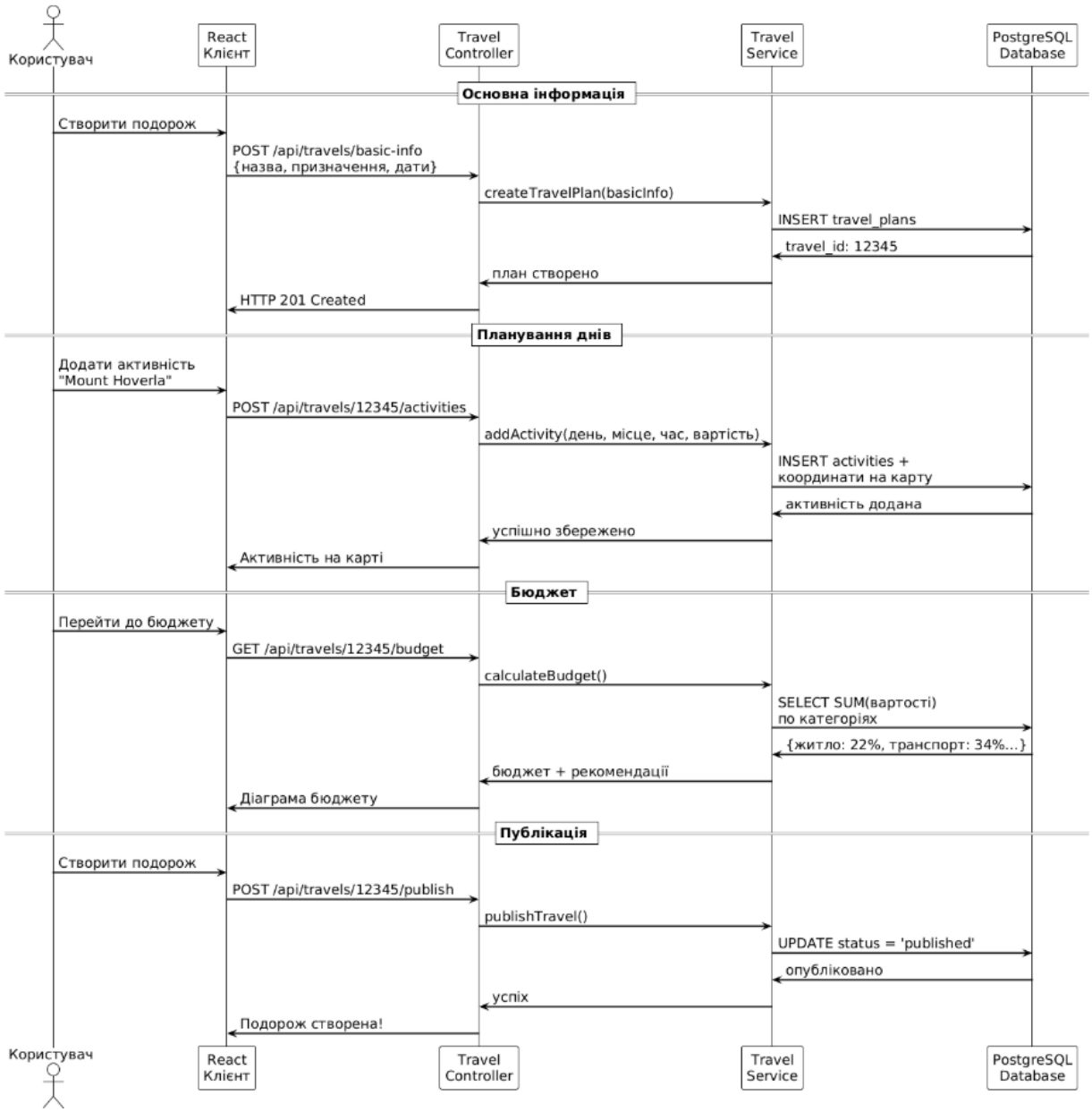

Рисунок 2.1 –UML-діаграма послідовності алгоритму створення публікацій подорожей

Після успішного збереження основної інформації, сервіс переходить до наступного етапу – планування днів подорожі. Для кожного дня, місця, часу та вартості створюються відповідні записи в базі даних через INSERT-запити до

таблиці activities, з координатами на карті та встановленням зв'язків між активностями та днями подорожі.

Наступним кроком є розрахунок бюджету подорожі. Сервіс виконує агрегуючий SELECT-запит до бази даних для підрахунку загальної суми витрат по категоріях. Ці дані повертаються користувачу для перегляду та можливого коригування. Після підтвердження користувачем усіх деталей, відбувається фінальне створення публікації подорожі.

Завершальний етап алгоритму включає публікацію створеної подорожі. Сервіс формує POST-запит /api/travels/12345/publish для публікації подорожі з вказаним ідентифікатором. База даних оновлює статус подорожі на "published" через UPDATE-запит, що робить подорож доступною для перегляду іншими користувачами системи. Після успішного завершення всіх операцій, користувач отримує підтвердження про успішне створення та публікацію подорожі, після чого може переглянути створений пост у своєму профілі.

2.2 Архітектура пошукової системи та індексація даних

Для реалізації пошуку в системі організації подорожей необхідно створити багаторівневу архітектуру пошукового сервісу на бекенді, що інтегрується з Elasticsearch [21] та зовнішніми API. Такий підхід дозволить забезпечити швидкий та релевантний пошук по різних типах даних одночасно.

Пошукова система на сервері вебзастосунку складатиметься з кількох ключових компонентів: індексатора, що наповнює Elasticsearch актуальними даними; пошукового сервісу, що обробляє запити користувачів і агрегує результати; та API-адаптерів для інтеграції із зовнішніми джерелами, що показано на рисунку 2.2.

Індексація даних здійснюватиметься як на регулярній основі для оновлення інформації із зовнішніх джерел, так і в режимі реального часу для внутрішніх даних системи [21]. Внутрішні дані про маршрути, публікації користувачів та локації індексуватимуться через механізм подій з використанням

Kafka, що забезпечить мінімальну затримку між створенням контенту та його доступністю для пошуку.

Індексація постів користувачів здійснюватиметься з розбивкою на структурні компоненти, що дозволить виконувати цільовий пошук за окремими частинами подорожі. При індексації кожен пост розкладатиметься на семантичні блоки: загальна інформація про подорож, транспортні деталі, інформація про проживання, активності та враження. Такий підхід забезпечить можливість пошуку як по всьому контенту постів, так і по їхніх конкретних розділах.

Для пошуку за транспортними деталями буде створено окремий індекс з полями, що містять інформацію про типи транспорту, маршрути, перевізників, тривалість поїздок та вартість. Це дозволить користувачам знаходити конкретні види транспорту, наприклад, нічні потяги між певними містами або бюджетні авіарейси до популярних напрямків [1].

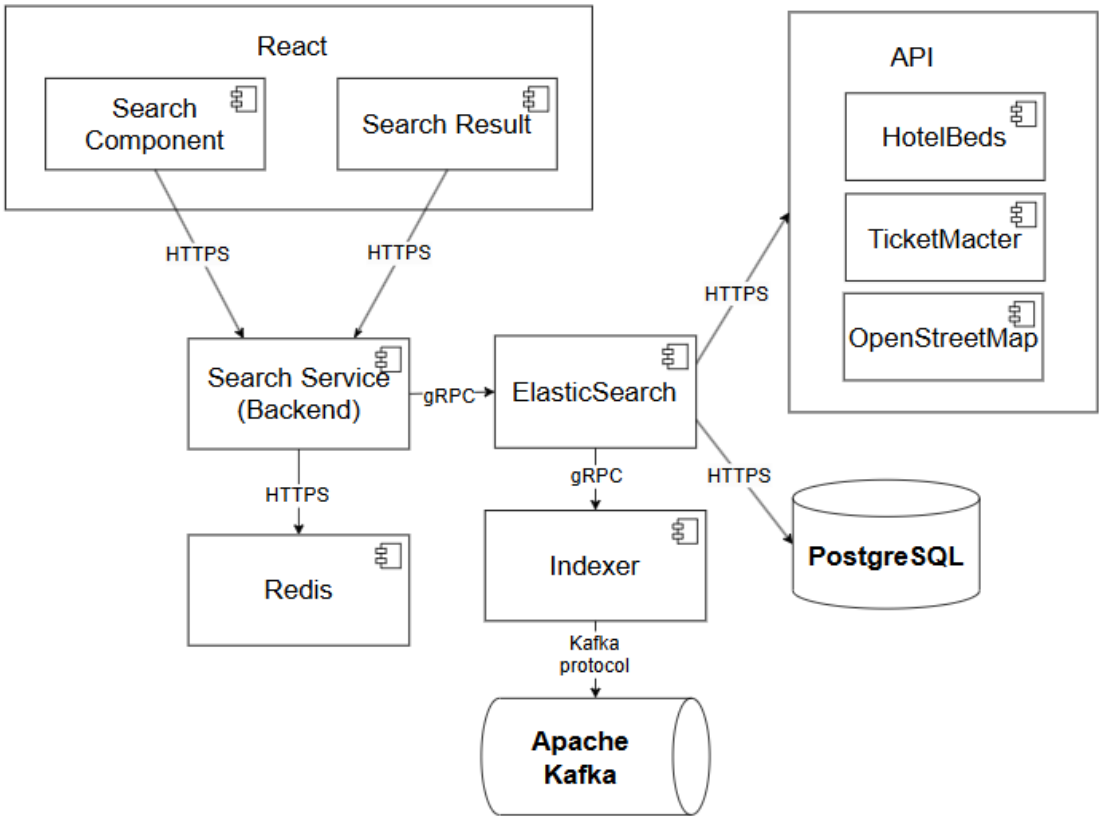

Рисунок 2.2 – UML-діаграма компонентів пошукової системи

Для даних з HotelBeds API буде реалізовано щоденне оновлення індексів з можливістю інкрементального оновлення для найпопулярніших напрямків. При індексації готелів зберігатимуться не лише базові характеристики, але й додаткова інформація, така як тип номерів, зручності, рейтинги та відгуки, що дозволить здійснювати деталізований пошук. Аналогічно, для пошуку за готелями та іншими варіантами проживання буде створено спеціалізований індекс, що включатиме також інформацію з постів користувачів про менш формальні варіанти проживання.

Дані з TicketMaster API індексуватимуться з різною частотою в залежності від типу контенту: інформація про майбутні події оновлюватиметься кілька разів на день, а дані про постійні атракції – раз на тиждень. При індексації зберігатимуться такі атрибути як дата, час, жанр, місце проведення та вартість квитків.

Для забезпечення контекстного пошуку по постах буде впроваджено технологію Named Entity Recognition, що дозволить автоматично виявляти та індексувати згадки про місця, заклади, активності та інші сутності в тексті постів. Це підвищить точність пошуку, оскільки система зможе розуміти контекст, в якому згадуються різні об'єкти.

Пошуковий сервіс прийматиме запити користувачів через RESTful API та розподілятиме їх на кілька паралельних підзапитів: до Elasticsearch для внутрішніх даних та до адаптерів зовнішніх API для актуальної інформації, не представленої в локальних індексах. Elasticsearch використовуватиметься для повнотекстового пошуку з підтримкою різних мов, фільтрацією за геолокацією та агрегацією для фасетного пошуку [21].

Пошуковий алгоритм буде розширено можливістю фільтрації за типом контенту в постах. Користувачі зможуть вказати, що вони зацікавлені лише в певних аспектах подорожі, наприклад, в ресторанах, музеях, пляжах або транспортних маршрутах. Система автоматично адаптуватиме пошуковий запит для фокусування на відповідних сегментах постів.

Кешування результатів пошуку за популярними запитами реалізовуватиметься з використанням Redis [13], що дозволить знизити навантаження на Elasticsearch та зовнішні API при повторних запитах. Час життя кешу варіюватиметься в залежності від типу даних: для статичної інформації про локації – до тижня, для подій – не більше години, для готелів – до доби.

Система обробки пошукових запитів розгортатиметься на шістьох типах серверів для забезпечення надійності та масштабованості: основні сервери програми, сервери Elasticsearch, сервери Redis для кешування, Apache Kafka для обробки повідомлень [19], сервери баз даних та сервери для API шлюзів. Таке розподілення навантаження дозволить системі швидко працювати навіть при пікових навантаженнях.

2.3 Модель системи комунікацій у реальному часі

Для реалізації функціоналу обміну повідомленнями в реальному часі у вебсистемі організації подорожей було розроблено алгоритм роботи модуля чатів на основі технології WebSocket. Даний алгоритм забезпечує миттєву доставку повідомлень між користувачами системи, що є критично важливим для координації спільних подорожей, обговорення деталей маршрутів та обміну актуальною інформацією (див. рисунок 2.3). Використання WebSocket дозволяє підтримувати постійне двостороннє з'єднання між клієнтом і сервером, що швидше за традиційні HTTP-запити для реалізації чат-функціоналу. Алгоритм інтегрується з Apache Kafka для забезпечення надійної доставки повідомлень навіть при тимчасовій недоступності одного з користувачів.

Представлений алгоритм демонструє архітектуру системи обміну повідомленнями між двома користувачами (Користувач А та Користувач Б) через WebSocket з'єднання. Кожен користувач при підключенні до системи встановлює власне WebSocket з'єднання з сервером повідомлень, який створює окрему сесію для кожного підключеного користувача. Ці сесії зберігають

інформацію про активні з'єднання та дозволяють ідентифікувати користувачів для маршрутизації повідомлень.

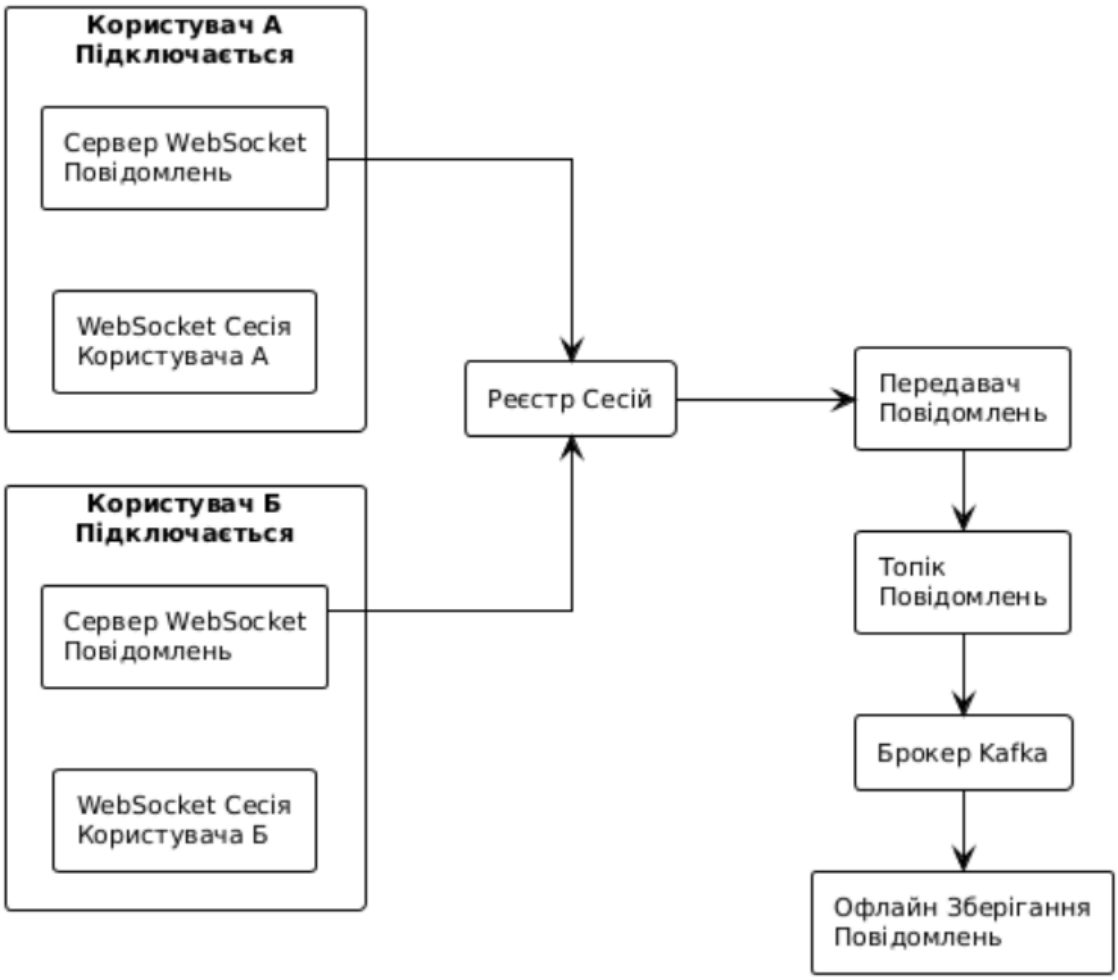

Рисунок 2.3 – UML-діаграма компонентів системи комунікацій

Центральним компонентом системи є Реєстр Сесій, який виконує функцію централізованого сховища інформації про всі активні WebSocket з'єднання. При підключенні нового користувача, сервер WebSocket автоматично реєструє його сесію в цьому реєстрі. Реєстр Сесій зберігає мапінг між ідентифікаторами користувачів та їхніми активними сесіями, що дозволяє швидко визначати, чи знаходиться користувач онлайн та до якого сервера він підключений у випадку розподіленої архітектури.

Коли Користувач А надсилає повідомлення Користувачу Б, система спочатку звертається до Реєстру Сесій для пошуку отримувача. Передавач

Повідомлень аналізує доступність адресата та визначає оптимальний спосіб доставки. Якщо отримувач онлайн, повідомлення передається безпосередньо через його активну WebSocket сесію. Паралельно повідомлення надсилається до Топіка Повідомлень для подальшої обробки та збереження.

Інтеграція з Apache Kafka через компонент Брокер Kafka забезпечує надійність системи обміну повідомленнями. Всі повідомлення проходять через Kafka топіки, що гарантує їх збереження та можливість повторної доставки у випадку збоїв. Брокер Kafka виконує роль надійної черги повідомлень, яка може обробляти високі навантаження та забезпечує горизонтальне масштабування системи при зростанні кількості користувачів.

Завершальним елементом алгоритму є компонент Офлайн Зберігання Повідомлень, який зберігає всі повідомлення в базі даних для подальшого доступу. Це дозволяє користувачам, які були офлайн, отримати всі пропущені повідомлення при наступному підключенні до системи. Така архітектура забезпечує надійну доставку повідомлень незалежно від статусу підключення користувачів та створює повноцінний досвід спілкування в рамках платформи організації подорожей.

2.4 Алгоритми забезпечення інформаційної безпеки системи

Забезпечення інформаційної безпеки є критично важливим аспектом розробки вебсистеми для організації подорожей, оскільки система обробляє персональні дані користувачів, деталі маршрутів та інші конфіденційні відомості. Розроблені алгоритми безпеки реалізують багаторівневий підхід до захисту даних, включаючи надійну автентифікацію користувачів, шифрування чутливої інформації та безпечну інтеграцію з зовнішніми API. Комплексний підхід до безпеки забезпечує відповідність системи сучасним стандартам захисту даних та створює довірче середовище для користувачів платформи.

Алгоритм реєстрації користувача з JWT-автентифікацією демонструє процес автентифікації користувача з використанням JWT-токенів, що наведено

на рисунку 2.4. При введенні облікових даних (електронної пошти та паролю), клієнт формує POST-запит до ендпоінту /api/auth/register. Spring Security Controller обробляє запит та встановлює захищене HTTPS/TLS з'єднання для передачі даних. Сервіс автентифікації звертається до User Service для верифікації наданих облікових даних. Для генерації солі використовується криптографічно стійкий генератор випадкових чисел (unique_salt_bytes = 16 bytes random), після чого пароль хешується алгоритмом Argon2id з параметрами: пам'ять 64MB, ітерації 120000 [20]. Хешований пароль зберігається в базі даних PostgreSQL у форматі argon2id.

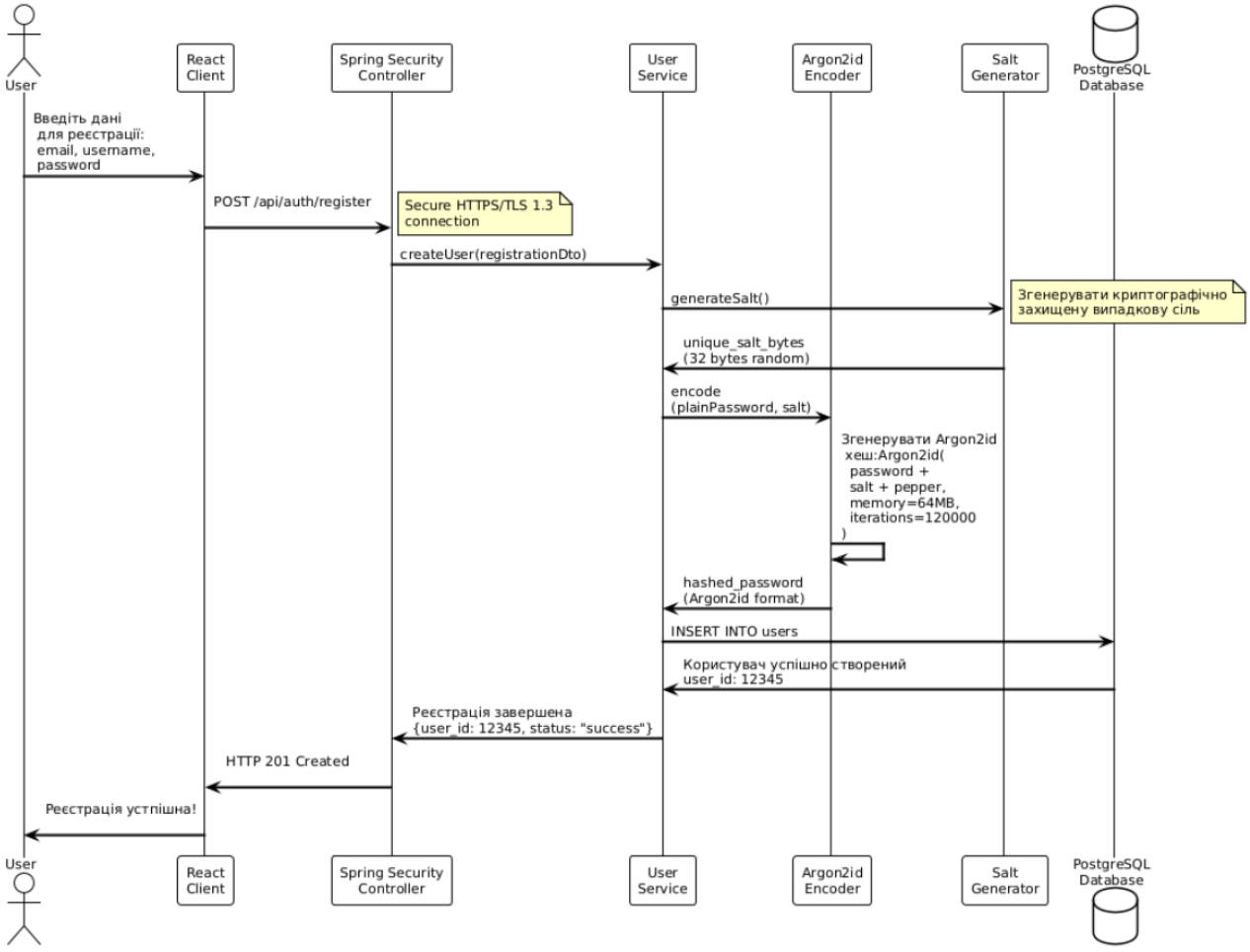

Рисунок 2.4 – UML-діаграма послідовності алгоритму реєстрації

Після успішної реєстрації відбувається генерація JWT-токена через Argon2id Encoder [20]. Токен підписується секретним ключем з використанням

алгоритму HMAC-SHA256 та містить корисне навантаження з ідентифікатором користувача (user_id: 12345) та статусом успішної автентифікації. Згенерований токен повертається клієнту через захищений канал та зберігається в криптографічно захищеному сховищі Salt Generator. Паралельно відбувається реєстрація успішної автентифікації в системі логування для подальшого аудиту безпеки. Цей алгоритм забезпечує створення безпечних облікових записів користувачів з використанням сучасних криптографічних стандартів та дозволяє уникнути атак типу rainbow table завдяки унікальним солям для кожного паролю.

Алгоритм автентифікації існуючого користувача ілюструє процес входу користувача в систему з верифікацією облікових даних, який наведено на рисунку 2.5. При спробі входу користувач надає свої облікові дані через форму входу. Клієнт формує POST-запит до /api/auth/login із зашифрованими через HTTPS/TLS даними. Spring Security Controller передає запит до User Service для пошуку користувача в базі даних PostgreSQL. Система виконує SELECT-запит для отримання хешованого паролю та солі користувача з таблиці users за вказаним username. Argon2id Encoder обчислює хеш наданого пароля з використанням збереженої солі та порівнює його зі збереженим хешем у базі даних.

При успішній верифікації паролю система генерує новий JWT-токен з терміном дії 3600 секунд. Токен містить claims з ідентифікатором користувача та часовою міткою створення. Паралельно відбувається порівняння поточного часу з часом останнього входу користувача (stored_hash = computed_hash) для виявлення потенційних аномалій. У випадку невідповідності паролів система повертає HTTP 401 Unauthorized помилку, а невдала спроба автентифікації логується для подальшого аналізу. При успішній автентифікації користувач отримує JWT-токен та перенаправляється на захищені сторінки системи. Реалізована система захисту від брутфорс-атак включає тимчасове блокування облікового запису після трьох невдалих спроб входу підряд, що підвищує рівень безпеки системи.

Алгоритм шифрування конфіденційних даних з використанням AES-256 описує процес захисту чутливих даних користувачів за допомогою симетричного шифрування (див. рисунок 2.6).

При необхідності збереження конфіденційних даних (наприклад, платіжної інформації або приватних повідомлень), Third-party authentication Service ініціює процес шифрування. Спочатку відбувається автентифікація з використанням API-ключа та секретного ключа через POST-запит /auth/login.

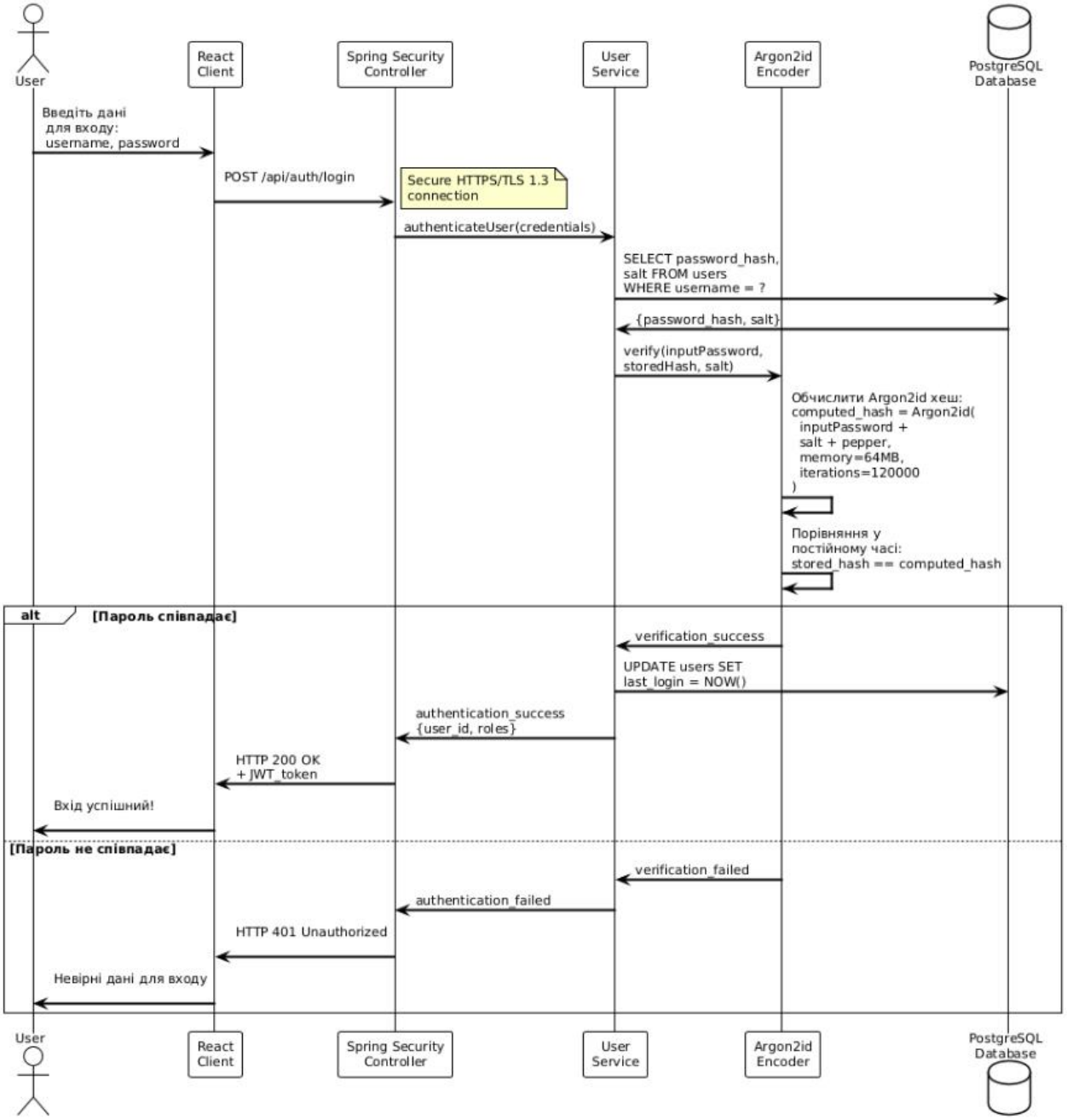

Рисунок 2.5 – UML-діаграма послідовності алгоритму автентифікації

Після успішної автентифікації система отримує майстер-ключ для шифрування з HashiCorp Vault через захищений API [21]. Для кожної операції шифрування генерується унікальний випадковий вектор ініціалізації (IV) довжиною 16 байт. AES-256 Encryption використовує режим GCM (Galois/Counter Mode) для забезпечення як конфіденційності, так і автентичності даних [22]. Зашифровані дані разом з IV та тегом автентифікації зберігаються в базі даних PostgreSQL.

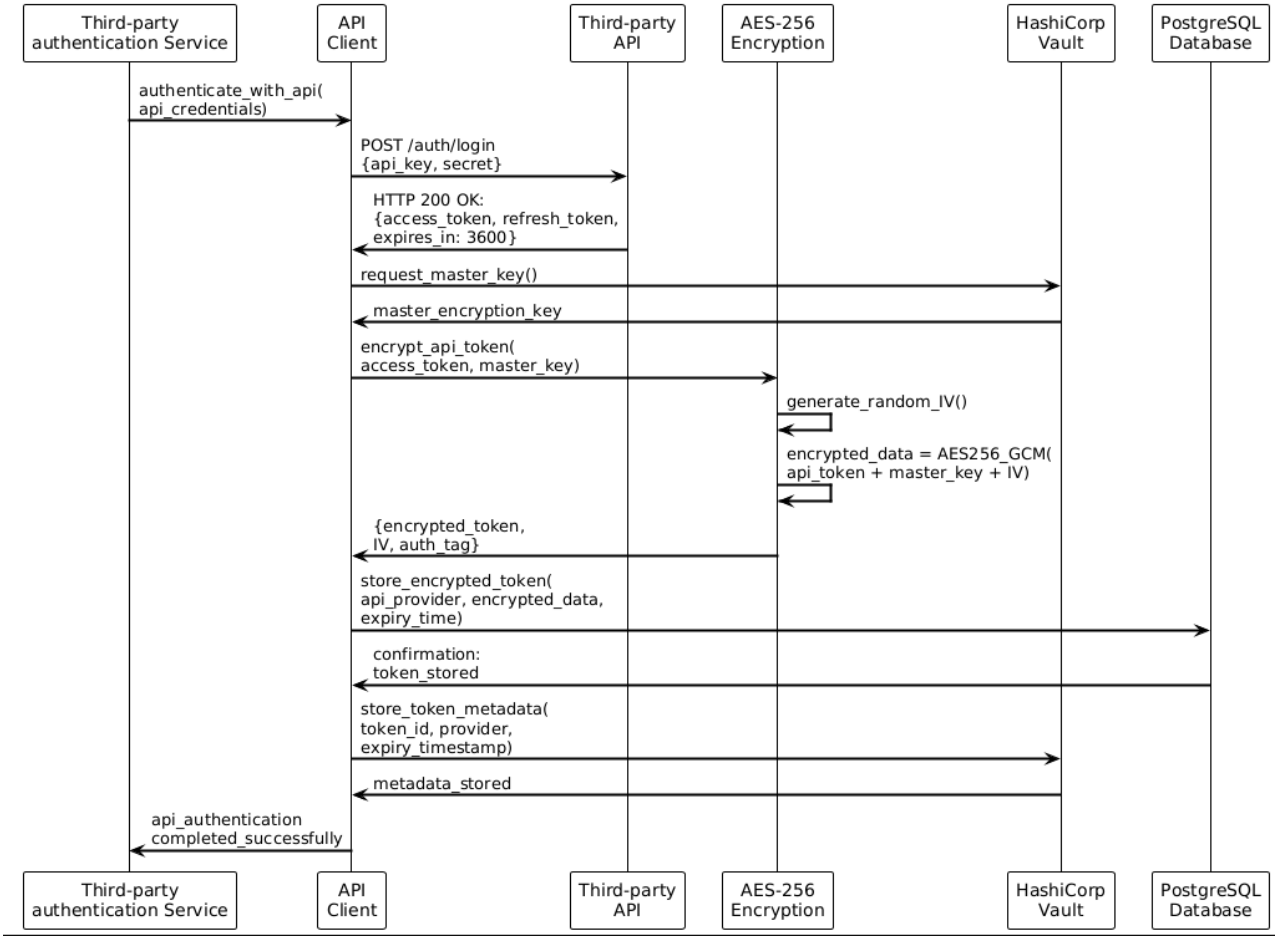

Рисунок 2.6 – UML-діаграма послідовності алгоритму шифрування даних

Метадані про операцію шифрування, включаючи ідентифікатор провайдера та часову мітку, також зберігаються для аудиту. При необхідності розшифрування даних система виконує зворотний процес, використовуючи той

самий майстер-ключ з Vault та збережений IV [21]. Такий підхід забезпечує надійний захист конфіденційних даних користувачів навіть у випадку компрометації бази даних. Інтеграція з HashiCorp Vault дозволяє централізовано управляти криптографічними ключами та забезпечує їх ротацію згідно з політиками безпеки організації, що відповідає найкращим практикам криптографічного захисту.

2.5 Висновки

Отже, було розроблено ключові алгоритми вебсистеми для організації подорожей. Для планування подорожей було реалізовано алгоритм, що координує клієнтський інтерфейс, серверну логіку на базі PostgreSQL та інтеграцію з картографічними сервісами, що забезпечило автоматизацію бюджетування та структурування даних. Пошуковий модуль було побудовано з використанням Elasticsearch, Kafka для індексації в реальному часі та Redis для кешування, що дозволило реалізувати контекстний пошук за геолокацією, типами активностей або ключовими параметрами. Для комунікацій між користувачами було використано WebSocket разом із Apache Kafka, що забезпечує надійну доставку повідомлень навіть при перервах у з’єднанні. Систему безпеки було реалізовано з JWT-автентифікацією, хешуванням паролів за допомогою Argon2id, шифруванням даних AES-256 та інтеграцією з HashiCorp Vault для керування ключами. Візуалізацію взаємодії компонентів було виконано за допомогою UML-діаграм.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВЕБСИСТЕМИ ДЛЯ ОРГАНІЗАЦІЇ ПОДОРОЖЕЙ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Розробка вебсистеми для організації подорожей потребує ретельного підходу до вибору технологічного стеку, від якого залежатиме надійність, масштабованість та зручність використання готового продукту. Архітектура системи має забезпечувати користувачам можливість планування маршрутів, бронювання житла та транспорту, а також отримання актуальної інформації про туристичні об'єкти в режимі реального часу. Інтуїтивний інтерфейс та швидкодія системи безпосередньо впливають на користувацький досвід та конкурентоспроможність продукту на ринку туристичних послуг.

Першочерговим завданням при проектуванні системи став вибір між монолітною та мікросервісною архітектурою [23]. Застосунок із монолітною архітектурою розробляється як єдиний цілісний блок, де всі компоненти тісно пов'язані та розгортаються разом. Мікросервісна архітектура розділяє систему на незалежні сервіси, кожен з яких виконує конкретну бізнес-функцію та може розроблятися, розгортатися та масштабуватися окремо. Для вебсистеми організації подорожей було обрано монолітну архітектуру через її простоту розробки на початкових етапах, менші витрати на інфраструктуру та легшу підтримку для невеликої команди розробників.

Детальне порівняння архітектурних підходів наведено в таблиці 3.1, яка демонструє ключові відмінності між монолітною та мікросервісною архітектурами. Аналіз показує, що монолітна архітектура дозволяє швидше вийти на ринок з невеликими витратами на інфраструктуру, тоді як мікросервісний підхід потребує досить довгої розробки та значно більших щомісячних витрат. Токаж для реалізації мікросервісного підходу потрібна

велика команда досвідчених розробників та тестувальників. Для стартапу фактори бюджету та швидкості виходу на ринок є критичними.

Таблиця 3.1 - Порівняння монолітної та мікросервісної архітектури

Критерій	Монолітна архітектура	Мікросервісна архітектура
Складність розробки	1-3 місяці для MVP	4-6 місяців для базової версії
Швидкість розгортання	10-15 хвилин на весь додаток	30-60 хвилин для всіх сервісів
Масштабування	Вертикальне до 32-64 ядер CPU	Горизонтальне до 100+ інстансів
Узгодженість даних	ACID транзакції в одній БД	Eventual consistency між сервісами
Тестування	70-80% покриття за 2 тижні	50-60% покриття за 4 тижні
Час виходу на ринок	3-4 місяці до першого релізу	6-9 місяців до стабільної версії

Основною мовою програмування для серверної частини було обрано Java через її стабільність, надійність та багатий набір бібліотек. Java є високорівневою об'єктно-орієнтованою мовою програмування з мінімальними залежностями від реалізації, що дозволяє писати код один раз і запускати його на будь-якій платформі, яка підтримує Java Virtual Machine [24]. Ця особливість стає критичною при розгортанні системи в хмарних середовищах різних провайдерів. Екосистема Java пропонує зрілі інструменти для роботи з вебдодатками, обробки транзакцій та інтеграції з зовнішніми сервісами.

Spring Boot було обрано як основний фреймворк для розробки серверної частини [16]. Цей фреймворк надає готові конфігурації для швидкого старту проекту, вбудовані серверні контейнери та автоматичну конфігурацію

компонентів. Spring Boot містить модулі для роботи з базами даних, безпекою, вебсервісами та багатьма іншими аспектами розробки. Використання Spring Boot дозволяє зосередитися на бізнес-логіці замість написання інфраструктурного коду.

Для зберігання структурованих даних було обрано PostgreSQL - потужну реляційну систему управління базами даних з відкритим кодом [25]. PostgreSQL виділяється підтримкою складних запитів, транзакційною цілісністю та можливістю роботи з геопросторовими даними через розширення PostGIS. Ця функціональність стає необхідною для туристичної системи, де потрібно зберігати координати об'єктів, будувати маршрути та виконувати географічні запити. База даних також підтримує JSON-типи даних, що дозволяє гнучко зберігати напівструктуровану інформацію.

Elasticsearch інтегрується в систему для реалізації повнотекстового пошуку та аналітики [19]. Ця пошукова система побудована на Apache Lucene та надає розподілений, мультитенантний пошуковий движок з RESTful API. Elasticsearch автоматично індексує дані за допомогою інвертованих індексів, що забезпечує швидкий пошук по мільйонах документів. Компоненти Logstash та Beats дозволяють збирати дані з різних джерел та трансформувати їх перед індексацією.

Apache Kafka виконує роль платформи для обробки потокових даних та забезпечує асинхронну комунікацію між компонентами системи [17]. Kafka працює як розподілений журнал подій з високою пропускною здатністю та низькою затримкою. Система використовує Kafka для обробки оновлень цін від постачальників, змін доступності номерів у готелях, повідомлень про бронювання та інших подій, які потребують негайної обробки. Архітектура публікації-підписки дозволяє легко додавати нових споживачів даних без зміни існуючої логіки.

Docker застосовується для контейнеризації всіх компонентів системи, що спрощує розробку, тестування та розгортання. Кожен компонент запаковується

у власний контейнер з усіма залежностями, що гарантує однакову поведінку в різних середовищах [18]. Docker Compose дозволяє описати всю інфраструктуру як код та запускати її однією командою. Контейнеризація також полегшує горизонтальне масштабування – при зростанні навантаження можна запустити додаткові екземпляри контейнерів.

Для клієнтської частини було обрано React.js - бібліотеку JavaScript для побудови користувацьких інтерфейсів [15]. React використовує компонентний підхід, де інтерфейс розбивається на незалежні частини, які можна повторно використовувати. Віртуальний DOM забезпечує швидке оновлення інтерфейсу при зміні даних. Односторонній потік даних спрощує відлагодження та розуміння логіки додатку. Велика екосистема бібліотек дозволяє швидко додавати функціональність без написання коду з нуля.

Redis використовується як сховище даних в оперативній пам'яті для кешування та управління сесіями [12]. Redis підтримує різні структури даних: рядки, хеші, списки, множини та впорядковані множини. Система використовує Redis для зберігання часто запитуваних даних, таких як популярні маршрути, результати пошуку та інформація про доступність. Кешування зменшує навантаження на основну базу даних та прискорює відповідь системи. Redis також застосовується для реалізації черг повідомлень та лічильників в реальному часі.

Інтеграція всіх обраних технологій створює цілісну систему, здатну обробляти складні бізнес-процеси туристичної галузі. Java та Spring Boot забезпечують надійну серверну частину, PostgreSQL зберігає критичні дані, Elasticsearch надає потужний пошук, Kafka обробляє потоки подій, Docker спрощує розгортання, React.js створює зручний інтерфейс, а Redis прискорює доступ до даних. Кожна технологія виконує свою роль, а разом вони формують сучасну вебсистему для організації подорожей.

Отже, вибір технологій для розробки вебсистеми організації подорожей, що включає Java, Spring Boot, PostgreSQL, ElasticSearch, Apache Kafka, Docker,

JavaScript (React.js) та Redis стек враховує не лише поточні потреби, але й можливості майбутнього розвитку. Поєднання цих інструментів надає комплексний підхід до розробки, що дозволяє створювати систему з високим рівнем функціональності, безпеки та зручності користування. Такий технологічний стек не лише відповідає сучасним вимогам індустрії, але й забезпечує можливості для подальшого розвитку та інтеграції нових функцій, таких як персоналізовані рекомендації на основі штучного інтелекту, віртуальні тури та розширена аналітика для бізнес-користувачів системи.

3.2 Розробка модуля автентифікації та управління користувачами

Модуль автентифікації та управління користувачами становить основу вебсистеми організації подорожей, забезпечуючи реєстрацію нових користувачів, вхід до системи, відновлення забутих паролів та керування профілями. Безпека доступу до особистих даних та персоналізованого контенту залежить від правильної реалізації цього модуля, що робить його критичним компонентом архітектури системи.

Серверна частина модуля базується на Java з використанням Spring Security - потужного фреймворку безпеки, який інтегрується зі Spring Boot. Spring Security надає готові компоненти для автентифікації та авторизації користувачів, підтримує різні методи захисту, включаючи OAuth2, LDAP та власні схеми автентифікації [16]. Для вебсистеми подорожей обрано підхід з використанням JWT-токенів, які дозволяють реалізувати безсесійну автентифікацію. Кожен токен містить закодовану інформацію про користувача та час дії, що дає можливість серверу перевіряти права доступу без звернення до бази даних при кожному запиті.

Архітектура модуля автентифікації представлена на діаграмі класів, яка демонструє взаємозв'язки між основними компонентами системи безпеки:

Діаграма класів демонструє структуру модуля автентифікації з основними компонентами (див. рисунок 3.1). Клас User представляє сутність користувача з

унікальним ідентифікатором, електронною поштою, іменем користувача та хешем паролю. Замість зберігання паролів у відкритому вигляді використовується хешування через Argon2id з випадковою сіллю для кожного користувача, що забезпечує вищий рівень безпеки порівняно з застарілими алгоритмами. Клас Authority визначає права доступу користувачів, дозволяючи створювати гнучку систему ролей. CustomUserDetailsService реалізує інтерфейс Spring Security для завантаження даних користувача під час автентифікації.

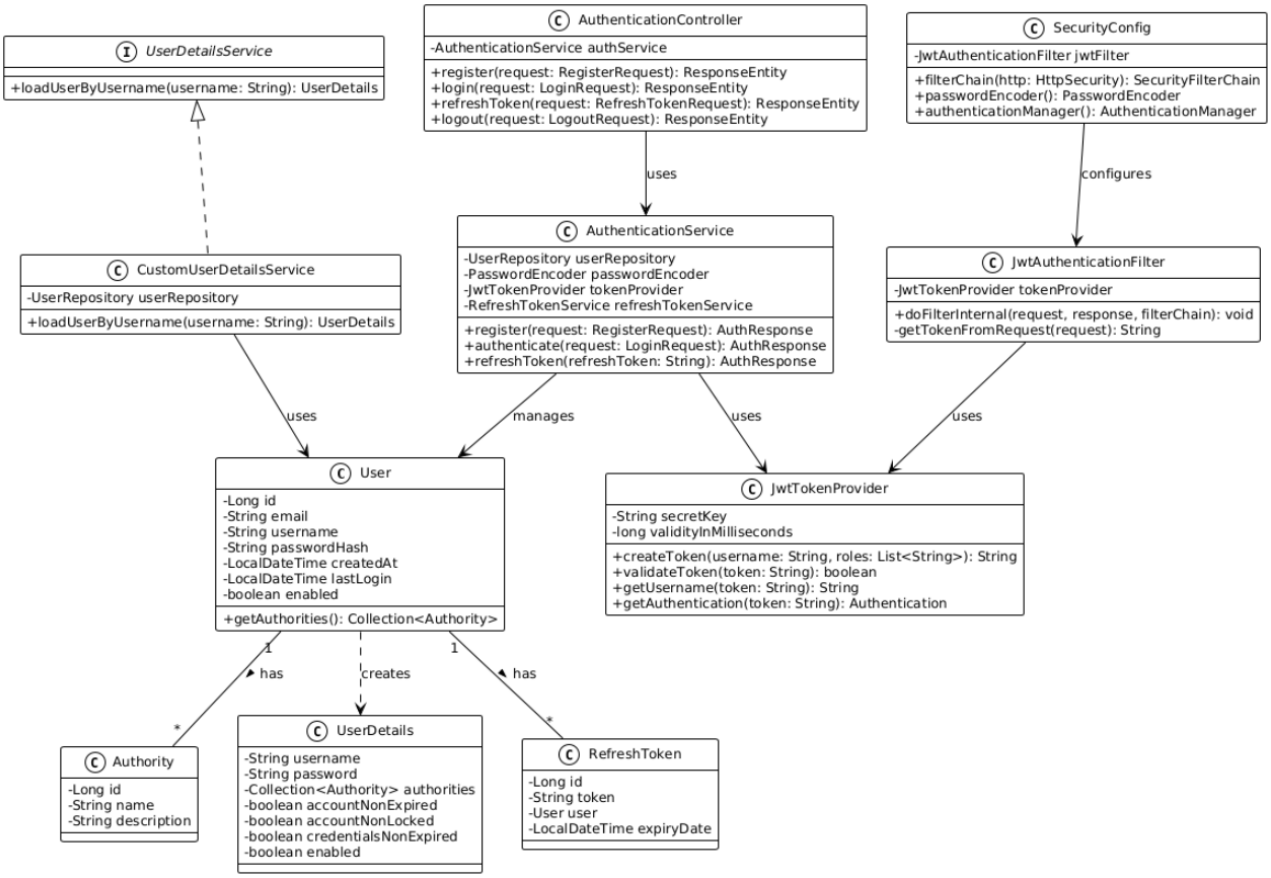

Рисунок 3.1 – UML-діаграма класів автентифікації

Процес автентифікації починається в AuthenticationController, який приймає HTTP-запити на реєстрацію, вхід та оновлення токенів. AuthenticationService містить бізнес-логіку перевірки облікових даних, створення нових користувачів та генерації токенів. JwtTokenProvider відповідає за створення та валідацію JWT-токенів з використанням секретного ключа та

алгоритму HMAC-SHA256 [2]. Кожен токен містить ідентифікатор користувача, список ролей та час закінчення дії.

Конфігурація безпеки здійснюється через клас SecurityConfig, який налаштовує ланцюжок фільтрів Spring Security. Основні налаштування включають визначення публічних ендпоінтів, які доступні без автентифікації (/api/auth/register, /api/auth/login), та захищених ресурсів, що вимагають валідного JWT-токену. JwtAuthenticationFilter перехоплює всі HTTP-запити, витягує токен з заголовка Authorization та встановлює контекст безпеки для поточного запиту.

Реалізація методу автентифікації в AuthenticationService виглядає наступним чином. Спочатку перевіряється наявність користувача в базі даних за електронною поштою або іменем користувача. Потім відбувається порівняння наданого пароля з збереженим хешем через PasswordEncoder. При успішній перевірці генерується пара токенів - основний JWT-токен з коротким терміном дії та refresh-токен з довшим терміном. Основний токен використовується для доступу до ресурсів, а refresh-токен дозволяє отримати новий основний токен без повторного введення паролю.

Для підвищення безпеки реалізовано механізм refresh-токенів через окрему сутність RefreshToken, яка зберігається в базі даних. Це дозволяє відкликати токени при виході користувача з системи або при виявленні підозрілої активності. Кожен refresh-токен прив'язаний до конкретного користувача та має унікальний ідентифікатор, що унеможливлює його підробку.

Зберігання даних користувачів відбувається в PostgreSQL з використанням таблиць users, authorities та user_authorities для реалізації зв'язку багато-до-багатьох між користувачами та ролями. Паролі зберігаються виключно в хешованому вигляді з використанням BCrypt та коефіцієнтом складності 12, що забезпечує баланс між безпекою та швидкістю обробки. Додатково в базі зберігаються метадані про час створення облікового запису, останній вхід та статус активації.

Redis використовується для кешування даних активних користувачів та зберігання чорного списку відкликаних токенів [12]. При кожній перевірці токена спочатку відбувається пошук в Redis, що дозволяє швидко відхиляти недійсні токени без звернення до основної бази даних. Також Redis зберігає тимчасові токени для відновлення паролів та активації облікових записів з автоматичним видаленням після закінчення терміну дії.

Клієнтська частина модуля реалізована на React.js з використанням хуків для управління станом автентифікації. Компонент AuthContext надає глобальний доступ до інформації про поточного користувача та методів для входу/виходу. Форми реєстрації та входу включають валідацію на стороні клієнта для миттєвого зворотного зв'язку користувачу. При успішній автентифікації токени зберігаються в localStorage для збереження сесії між перезавантаженнями сторінки, а axios interceptor автоматично додає токен до всіх запитів до API.

Контейнеризація модуля через Docker забезпечує ізольоване середовище виконання з усіма необхідними залежностями. Dockerfile включає базовий образ Java, копіювання скомпільованого JAR-файлу та налаштування змінних середовища для підключення до бази даних та Redis [18]. Apache Kafka інтегрується для асинхронної обробки подій, пов'язаних з користувачами, таких як надсилання вітальних листів після реєстрації, сповіщення про підозрілі входи з нових пристроїв та логування всіх дій для аудиту безпеки.

3.3 Розробка модуля для організації публікацій подорожей

Модуль для організації публікацій подорожей з розбивкою по днях забезпечує функціонал створення, редагування та перегляду детальних звітів про подорожі з хронологічним розподілом активностей, місць відвідування та вражень. Цей модуль є ключовим елементом соціального аспекту системи, що дозволяє користувачам ділитися своїм досвідом та надихати інших на нові маршрути. Ключовою особливістю модуля є розвинена система обробки та

зберігання медіафайлів, що дозволяє користувачам ділитися високоякісними фотографіями своїх подорожей.

Серверна частина модуля реалізована на Java з використанням Spring Boot для забезпечення RESTful API. Основними сутностями системи є Travel (подорож), TravelDay (день подорожі) та MediaFile (медіафайл) [16]. Архітектура базується на принципах багатошарової структури з контролерами, сервісами та репозиторіями для забезпечення чіткого розподілу відповідальності.

Система обробки медіафайлів становить серце модуля публікацій, забезпечуючи надійне зберігання та швидкий доступ до зображень. Архітектура системи зберігання зображень представлена на діаграмі, яка демонструє повний цикл обробки медіафайлів від завантаження до збереження (див. рисунок 3.2).

Коли користувач завантажує зображення через React клієнт, запит надходить до MediaController у Backend сервісах. Controller передає файл до MediaService, який реалізує основну бізнес-логіку обробки медіаконтенту. Ключовим компонентом системи є Thumbnailator бібліотека, яка автоматично створює мініатюри різних розмірів для адаптації відображення на різних пристроях.

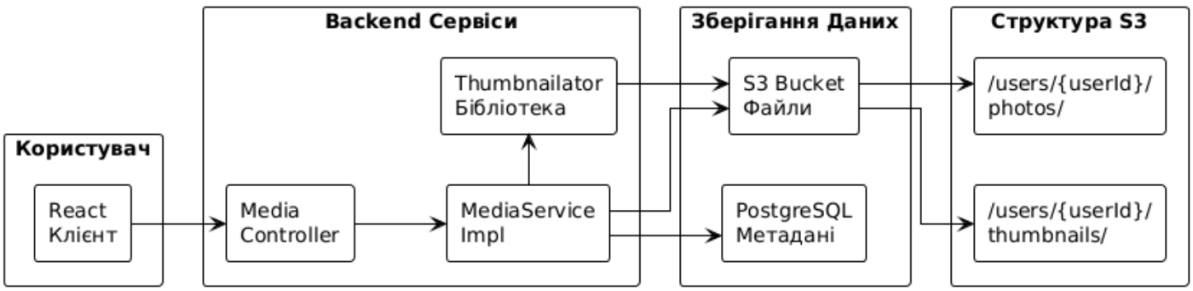

Рисунок 3.2 – UML-діаграма компонентів збереження зображення

MediaService обробляє кожне завантажене зображення, створюючи набір адаптованих версій: повнорозмірне зображення для детального перегляду, середній розмір для галерей та мініатюру для швидкого завантаження у списках

публікацій. Thumbnailator забезпечує високу якість обробки зі збереженням пропорцій та значним зменшенням розміру файлів без втрати візуальної якості.

Зберігання медіафайлів організовано у дворівневій архітектурі: метадані файлів зберігаються в PostgreSQL базі даних, а самі медіафайли розміщуються в S3 Bucket для забезпечення масштабованості та швидкості доступу [26]. Структура S3 організована ієрархічно з логічним розподілом: основні файли зберігаються у директорії /users/{userid}/photos/, а згенеровані мініатюри у відповідній піддиректорії /users/{userid}/thumbnails/.

PostgreSQL зберігає критичні метадані кожного медіафайла: унікальний ідентифікатор, оригінальну назву файлу, розмір, формат, шлях до файлу в S3 та прапорець hasThumbnails, який вказує на наявність згенерованих мініатюр. Ця інформація дозволяє frontend частині швидко визначити, який розмір зображення завантажувати для конкретного контексту відображення.

Процес завантаження медіафайлу включає декілька етапів: валідацію формату та розміру файлу, генерацію унікального імені для уникнення колізій, створення мініатюр через Thumbnailator, завантаження всіх версій до S3 та збереження метаданих в PostgreSQL. Система підтримує популярні формати зображень (JPEG, PNG, WebP) з автоматичною конвертацією у найбільш оптимальний формат для вебвідображення.

Redis використовується для кешування метаданих часто запитуваних медіафайлів та URL-посилань на зображення в S3, що значно прискорює відображення популярних публікацій. Кеш автоматично оновлюється при зміні медіаконтенту та має налаштовані TTL (Time To Live) для забезпечення актуальності даних.

3.4 Розробка модуля інтеграції з зовнішніми API

Модуль інтеграції з зовнішніми API забезпечує взаємодію вебсистеми організації подорожей з різноманітними сервісами третіх сторін, включаючи HotelBeds API для бронювання готелів, Ticketmaster API для пошуку та

бронювання квитків на події, OpenStreetMap API для картографічних сервісів та геокодування. Модуль реалізує уніфіковану архітектуру для роботи з REST API різних провайдерів, забезпечуючи надійність, масштабованість та легкість додавання нових інтеграцій.

Серверна частина модуля базується на Java з використанням Spring Boot та Spring WebClient для асинхронної взаємодії з зовнішніми сервісами. Архітектура побудована за принципом адаптерів, де для кожного зовнішнього API створюється окремий сервісний клас, що інкапсулює специфіку взаємодії з конкретним провайдером. Це дозволяє легко додавати нові інтеграції без впливу на існуючий код.

Базовий абстрактний клас ExternalApiClient містить спільну логіку для всіх інтеграцій: конфігурацію HTTP-клієнта, обробку аутентифікації, retry-механізми та стандартизовані методи логування запитів. Конкретні реалізації як HotelBedsApiClient, TicketmasterApiClient та NominatimApiClient успадковують цю функціональність та додають специфічну логіку для кожного провайдера.

HotelBedsApiClient забезпечує інтеграцію з одним з найбільших постачальників готельних послуг [5]. Клас реалізує методи пошуку готелів за географічними координатами, датами та кількістю гостей, отримання деталей конкретного готелю та ініціювання процесу бронювання. API використовує складну систему автентифікації з API ключами та підписами запитів, що реалізовано через спеціалізований HotelBedsAuthenticationProvider.

TicketmasterApiClient інтегрується з провідною платформою продажу квитків для отримання інформації про концерти, спортивні події та театральні вистави у певному регіоні [6]. Клас підтримує фільтрацію подій за категоріями, датами та географічним розташуванням, що дозволяє рекомендувати користувачам актуальні події під час планування подорожі. API повертає детальну інформацію про події, включаючи ціни, доступність місць та посилання для бронювання.

Інтеграція з OpenStreetMap демонструє роботу з картографічними API та представлена на діаграмі послідовності взаємодії (див. рисунок 3.3). Коли користувач вводить назву місця через React клієнт, MapContext компонент передає запит до Nominatim Service для геокодування [4]. Nominatim API перетворює текстову адресу у координати, після чого OpenStreetMap API надає картографічні тайли для відображення. Leaflet компонент забезпечує інтерактивне відображення карти з маркерами знайдених локацій.

Система конфігурації API ключів та параметрів підключення реалізована через Spring Boot Configuration Properties з можливістю динамічного оновлення без перезапуску додатку. Секретні дані зберігаються в HashiCorp Vault з автоматичною ротацією ключів для критичних інтеграцій.

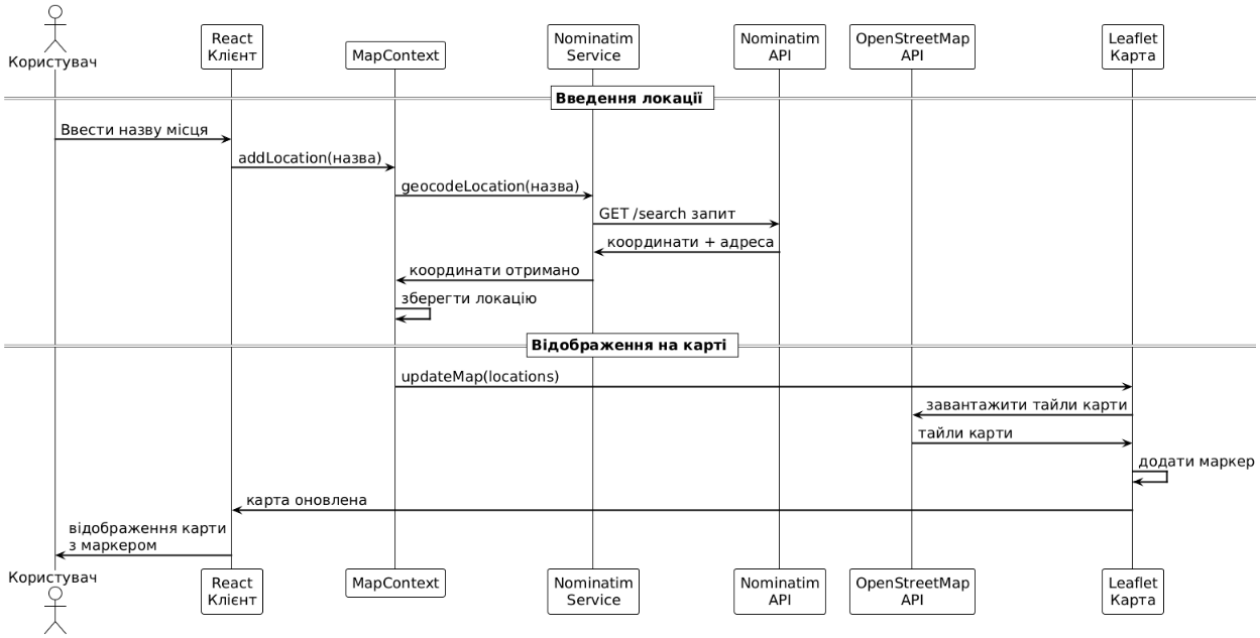

Рисунок 3.3 – UML-діаграма послідовності взаємодії з OpenstreetMap API

Для покращення продуктивності та зменшення витрат на API-виклики використовується багаторівневе кешування. Redis зберігає результати пошуку готелів та подій з TTL (Time To Live) 1-6 годин залежно від типу даних, а результати геокодування кешуються на 24 години. Це особливо важливо для HotelBeds API, де кожен запит має вартість.

Система моніторингу інтеграцій включає метрики успішності запитів, часу відповіді та витрат на API для кожного провайдера. Micrometer збирає статистику та передає її до Prometheus для аналізу трендів використання. При перевищенні лімітів запитів або високому проценті помилок автоматично активуються резервні механізми.

Circuit Breaker патерн реалізований для кожної інтеграції окремо через Resilience4j бібліотеку. За умови довгого часу недоступності HotelBeds API, система автоматично перемикається на альтернативних провайдерів готелів, а при недоступності Ticketmaster API активується кешований контент попередніх запитів.

PostgreSQL зберігає історію всіх API-запитів через таблицю api_requests_log з деталями про час запиту, тип API, параметри, статус відповіді та витрачений час. Ця інформація використовується для аналізу використання, додаткового налаштування кешування та виставлення рахунків за API-виклики.

Apache Kafka забезпечує асинхронну обробку результатів API-запитів через топіки hotels.found, events.found та locations.geocoded. Це дозволяє іншим сервісам системи реагувати на нові дані: оновлювати рекомендації, індексувати контент для пошуку та надсилати персоналізовані сповіщення користувачам [17].

Rate Limiting реалізований на рівні кожного API окремо відповідно до їх лімітів: HotelBeds дозволяє 100 запитів/хвилину, Ticketmaster – 5000 запитів/день, а OpenStreetMap має м'які обмеження. TokenBucket алгоритм забезпечує справедливий розподіл квот між паралельними запитами користувачів.

3.5 Розробка модуля чатів у реальному часі на основі WebSockets

Модуль чатів у реальному часі забезпечує миттєвий обмін повідомленнями між користувачами системи, дозволяючи обговорювати деталі подорожей, координувати спільні поїздки та отримувати консультації від експертів з

туризму. Ключовою особливістю модуля є реалізація двостороннього зв'язку через WebSocket протокол з підтримкою масштабування та high availability архітектури для обслуговування тисяч одночасних з'єднань.

Серверна частина модуля реалізована на Java з використанням Spring WebSocket та STOMP протоколу для структурованого обміну повідомленнями [14]. Архітектура базується на WebSocketConfigurer класі, який налаштовує ендпоінти, interceptors та message brokers для забезпечення маршрутизації повідомлень між клієнтами (див. рисунок 3.4).

Основною частиною реалізації є система управління WebSocket з'єднаннями та доставки повідомлень. WebSocketSessionManager відстежує всі активні з'єднання користувачів, зберігаючи маппінг між користувачами та їх сесіями в concurrent HashMap для thread-safe операцій. При встановленні з'єднання клієнт автентифікується через JWT токен, переданий як параметр WebSocket handshake.

ChatMessageController обробляє вхідні повідомлення та визначає їх тип: приватні повідомлення, групові чати або системні сповіщення. Для кожного типу повідомлення застосовується відповідна логіка маршрутизації через MessageRoutingService. Цей сервіс аналізує метадані повідомлення та визначає список цільових з'єднань для доставки.

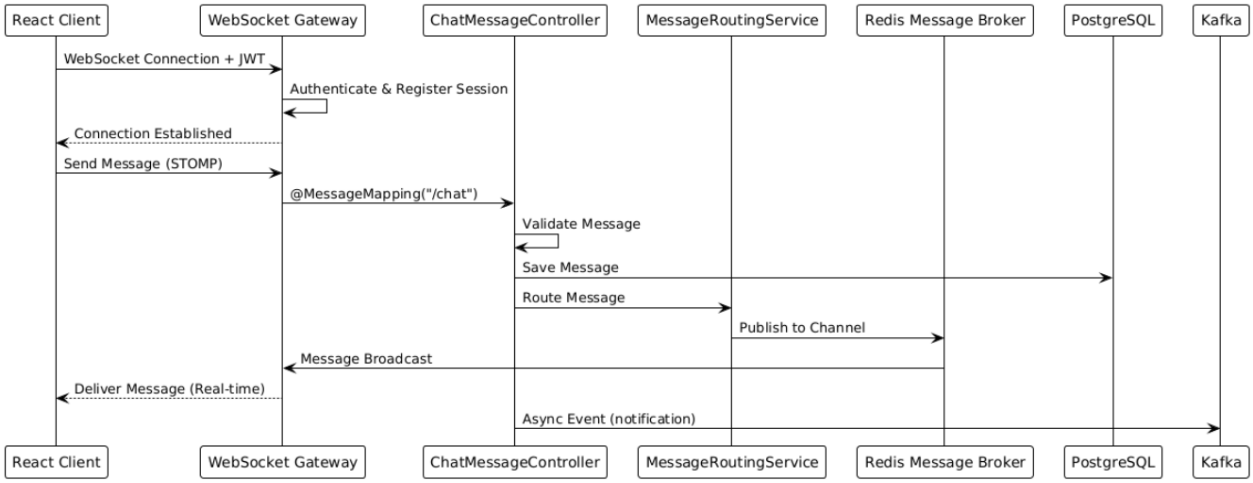

Рисунок 3.4 – UML-діаграма послідовності WebSocket чат-системи

Redis відіграє критичну роль як message broker для горизонтального масштабування чат-сервісу [12]. Коли користувач надсилає повідомлення, воно публікується в Redis channel, що дозволяє всім екземплярам серверів отримати це повідомлення та доставити відповідним клієнтам. Це забезпечує роботу чатів навіть коли відправник та отримувач підключені до різних серверів кластера.

MessagePersistenceService асинхронно зберігає всі повідомлення в PostgreSQL для збереження історії чатів. Таблиця messages партиціонується за датою створення для покращення продуктивності запитів. Додатково створюються індекси за chat_id, user_id та timestamp для швидкого пошуку повідомлень у конкретних чатах.

Система обробки повідомлень включає валідацію контенту, фільтрацію спаму та автоматичну модерацію через ContentModerationService. Кожне повідомлення проходить через ланцюжок фільтрів перед доставкою: перевірка довжини тексту, виявлення заборонених слів, аналіз частоти надсилання повідомлень користувачем для запобігання спаму.

Клієнтська частина реалізована через React компоненти з використанням SockJS та STOMP.js для встановлення WebSocket з'єднання. ChatContainer компонент управляє станом чату, включаючи список повідомлень, індикатори статусу та обробку вхідних повідомлень. MessageInput компонент забезпечує введення тексту з підтримкою emoji, файлів та геолокації.

Реалізовано оптимістичний UI для миттєвої відповіді інтерфейсу: повідомлення відображається в чаті одразу після натискання "надіслати" з індикатором "надсилається", який змінюється на "доставлено" після підтвердження сервера. При помилці доставки повідомлення позначається як "не доставлено" з можливістю повторної спроби.

WebSocket Connection Manager забезпечує стабільність з'єднання через автоматичне відновлення при втраті мережі, heartbeat механізм для виявлення розірваних з'єднань та graceful reconnection з відновленням пропущених

повідомлень [14]. Клієнт періодично надсилає ping повідомлення, а сервер відповідає pong для підтримки активності з'єднання.

Apache Kafka інтегрується для асинхронної обробки chat-подій через топіки message.sent, user.joined та user.left. Це дозволяє реалізувати push-сповіщення через окремий сервіс, аналіз активності користувачів, автоматичну модерацію контенту та інтеграцію з системами CRM без впливу на швидкість доставки повідомлень [17].

Система підтримує різні типи чатів: приватні діалоги між двома користувачами, групові чати для обговорення спільних подорожей та публічні канали для загальних тем. ChatRoomService управляє створенням, налаштуваннями та правами доступу для кожного типу чату з можливістю призначення модераторів та адміністраторів.

Контейнеризація за допомогою Docker додатково налаштована для роботи з WebSocket з'єднаннями. Було збільшено ліміти на кількість одночасних з'єднань, налаштовано keep-alive параметри та session affinity для забезпечення того, що користувач залишається підключеним до того ж сервера протягом сесії. Load balancer налаштований на sticky sessions для WebSocket трафіку.

3.6 Розробка модуля безпеки та шифрування даних

Модуль безпеки та шифрування даних забезпечує захист конфіденційної інформації користувачів, включаючи персональні дані, платіжну інформацію та дані про плани подорожей. Цей модуль є критичним компонентом системи, що гарантує дотримання законодавчих вимог щодо захисту персональних даних та запобігає несанкціонованому доступу до чутливої інформації.

Серверна частина модуля реалізована на Java з використанням Spring Security для забезпечення комплексного підходу до безпеки [13]. Архітектура базується на принципах defence in depth з множинними рівнями захисту: автентифікація, авторизація, шифрування даних та аудит безпеки, що наведено на рисунку 3.5.

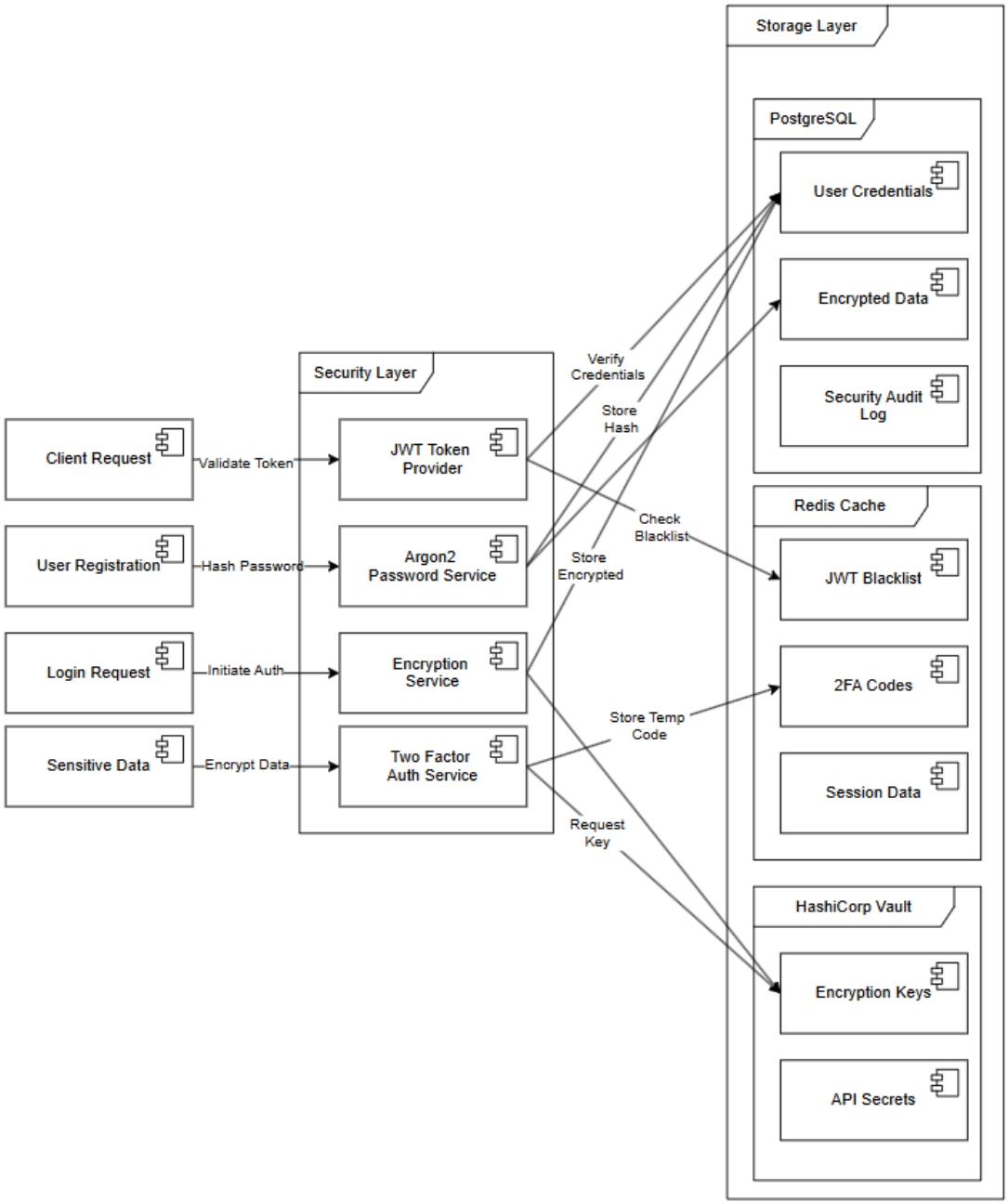

Рисунок 3.5 – UML-діаграма компонентів модуля безпеки

Основою модуля є система хешування паролів з використанням алгоритму Argon2, який є сучасним стандартом для безпечного зберігання паролів. Argon2PasswordEncoder замінює застарілий BCrypt та забезпечує кращий захист проти атак методом перебору та rainbow table атак [20]. Алгоритм Argon2 має три варіанти: Argon2d (призначений проти GPU атак), Argon2i (призначений проти side-channel атак) та Argon2id (гібридний варіант). Система використовує Argon2id як найбезпечніший.

Система шифрування даних базується на алгоритмі AES-256-GCM, який забезпечує як конфіденційність, так і автентичність даних [22]. EncryptionService реалізує envelope encryption pattern: дані шифруються унікальними DEK (Data Encryption Keys), які у свою чергу шифруються KEK (Key Encryption Key), що зберігається в HashiCorp Vault [21]. Це дозволяє зручно управляти ключами та забезпечує можливість ротації ключів без перешифрування всіх даних.

Кожна операція шифрування генерує унікальний 96-бітний nonce для GCM режиму, що гарантує семантичну безпеку навіть при шифруванні однакових даних. Результат шифрування включає зашифровані дані, nonce, authentication tag та metadata у форматі, що забезпечує цілісність та автентичність інформації.

JWT система реалізована з короткочасними access токенами та довгочасними refresh токенами для забезпечення балансу між безпекою та зручністю користування. JwtTokenProvider використовує асиметричне шифрування RS256 з приватним ключем для підпису токенів та публічним ключем для валідації, що дозволяє розподілену валідацію без доступу до приватного ключа.

Система двофакторної автентифікації реалізована через TwoFactorAuthService з підтримкою TOTP (Time-based One-Time Password) згідно RFC 6238 та SMS/email кодів [27]. TOTP генератор використовує SHA-256 hash функцію з 30-секундним часовим вікном та 6-цифровими кодами. Backup коди генеруються для відновлення доступу у випадку втрати основного 2FA пристрою.

Аудит безпеки реалізований через SecurityAuditService, який логує всі критичні операції: спроби автентифікації, зміни паролів, доступ до чутливих даних, операції шифрування/розшифрування. Логи структуровані у JSON форматі з timestamp, user ID, IP адресою, user agent та результатом операції для подальшого аналізу.

Redis використовується для зберігання blacklist відкликаних JWT токенів з TTL (Time To Live), що відповідає часу дії токенів [12]. Це забезпечує миттєве

відкликання доступу при logout або при виявленні підозрілої активності. Додатково Redis зберігає rate limiting лічильники для запобігання brute force атакам на автентифікацію.

HashiCorp Vault інтегрується як центральне сховище для всіх криптографічних ключів з автоматичною ротацією кожні 90 днів для KEK та щомісячно для DEK [21]. Dynamic secrets для API ключів генеруються on-demand з обмеженим часом життя, що мінімізує ризик компрометації.

Apache Kafka забезпечує асинхронну обробку security подій через топіки security.authentication, security.data_access та security.anomaly_detection. SIEM (Security Information and Event Management) система аналізує ці події для виявлення аномалій: множинні невдалі спроби входу, доступ з нових географічних локацій, незвичайні паттерни використання API.

Система включає AntiCSRF механізми через SameSite cookies та CSRF токени, HTTPS Strict Transport Security headers, Content Security Policy для запобігання XSS (Cross-Site Scripting) атакам та input validation на ендпоінти [13]. Rate limiting реалізований на рівні IP адрес та користувачів для запобігання Denial-of-Service (DoS) атакам.

Compliance модуль забезпечує дотримання GDPR (General Data Protection Regulation) вимог через право на забуття (cryptographic erasure шляхом видалення ключів) [2], data portability (експорт даних у структурованому форматі) та consent management для збору та обробки персональних даних.

Docker контейнеризація модуля включає окремі контейнери для різних компонентів безпеки з мінімальними правами доступу, не-root користувачами та регулярними оновленнями безпеки базових образів. Secrets передаються через зашифровані розділи та змінні середовища.

3.7 Розробка вебінтерфейсу програмного засобу

При розробці інтерфейсу вебсистеми для організації подорожей визначальними факторами були інтуїтивність навігації, зручність використання

та інтеграція соціальних функцій. Основною кольоровою гамою було обрано фіолетово-білу, що забезпечує візуальну гармонію.

Дизайн-система платформи базується на принципах Material Design з адаптацією під специфіку туристичної тематики. Використання м'яких градієнтів та сучасної типографіки створює відчуття професійності та надійності, що є критично важливим для платформи, де користувачі планують значні фінансові витрати на подорожі. Інтерфейс спроєктовано для використання на різних пристроях – від настільних комп'ютерів до мобільних телефонів, забезпечуючи цілісний досвід користувача незалежно від способу доступу до системи.

Запропонована вебсистема відкриває перед користувачами розширені можливості для планування, документування та обміну досвідом подорожей. Інтеграція соціальних елементів із функціями детального планування маршрутів дозволяє створювати персоналізовані подорожі та поширювати їх серед різних аудиторій.

Вхідний інтерфейс системи розроблено за принципом мінімалізму, що забезпечує швидкий та безперешкодний доступ до основних функцій платформи. При першому відвідуванні користувачу пропонується авторизуватися, що показано на рисунку 3.6 або створити новий обліковий запис, після чого відкривається доступ до повного функціоналу системи.

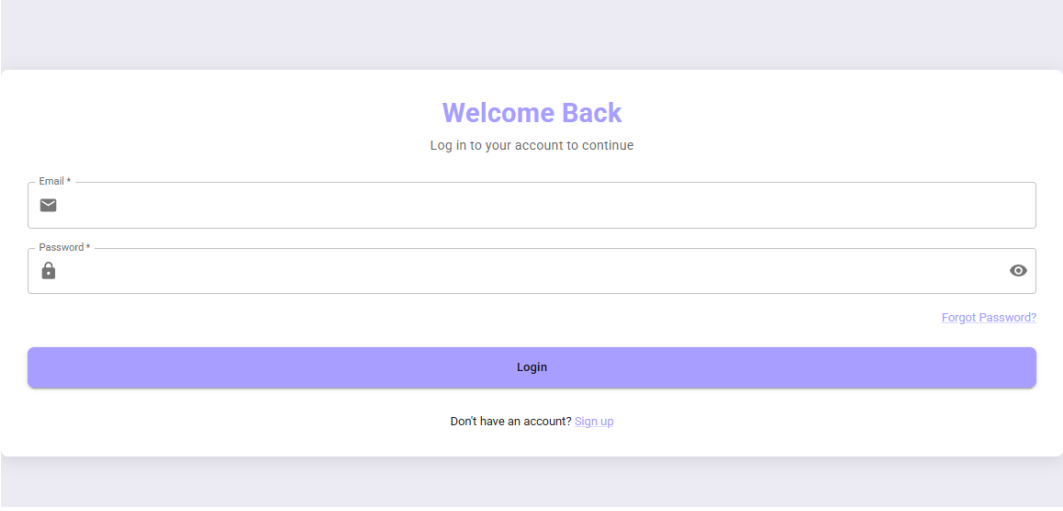

Рисунок 3.6 – Сторінка авторизації

Центральною частиною системи є функція перегляду та створення публікацій про подорожі (див. рисунок 3.7). Користувачі можуть переглядати записи інших мандрівників, які включають панорамні зображення, інтерактивні карти з відміченими пам'ятками та докладні описи вражень. Подібний обмін досвідом служить джерелом натхнення та практичної інформації для планування власних маршрутів.

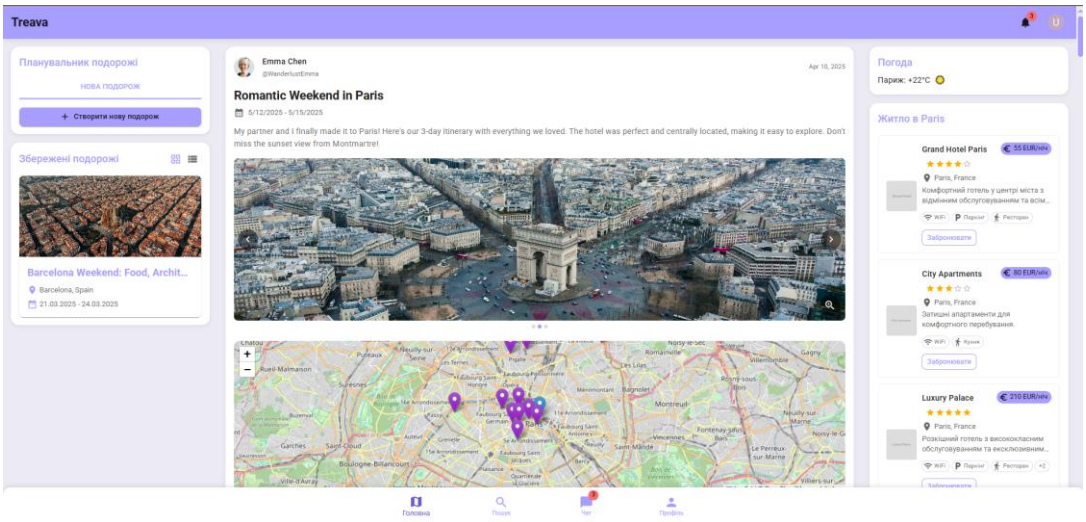

Рисунок 3.7 – Головна сторінка вебсистеми

Пошуковий механізм системи забезпечує гнучкий пошук (див. рисунок 3.8) за різними параметрами, включаючи місця призначення, типи активностей, що показано на рисунку 3.9, варіанти проживання (див. рисунок 3.10), транспорту та пости інших користувачів (див. рисунок 3.11). Результати пошуку представляються у структурованому вигляді з короткими описами та можливістю переходу до детальної інформації.

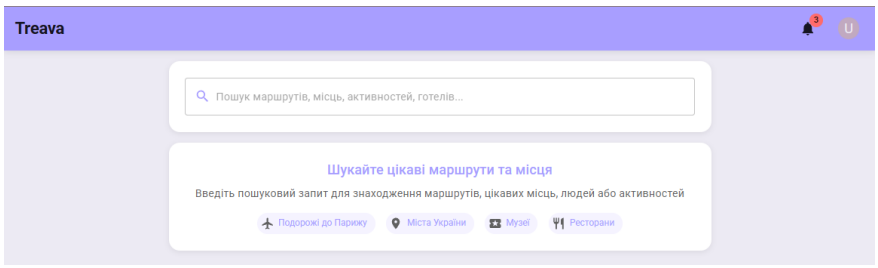

Рисунок 3.8 – Сторінка пошуку

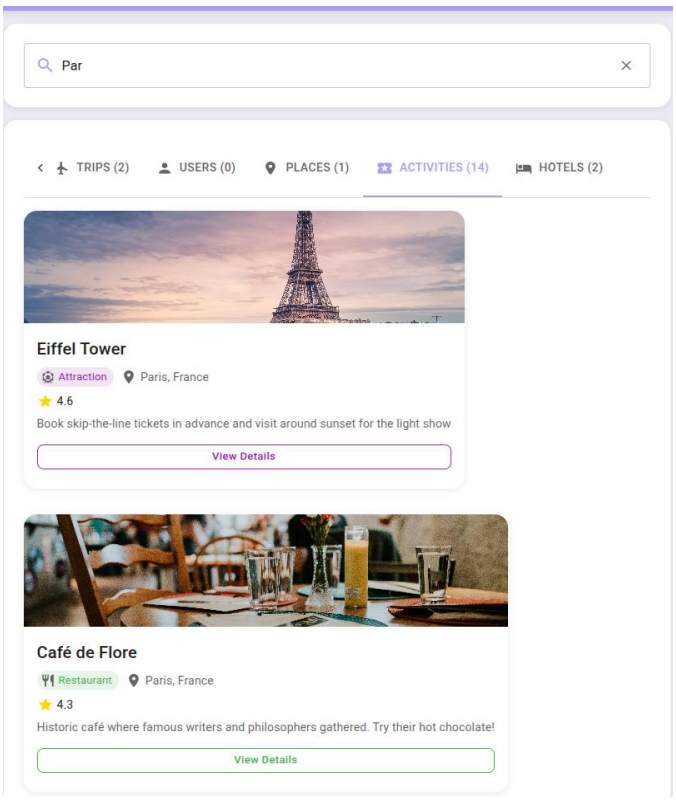

Рисунок 3.9 – Пошук активностей за запитом «Par»

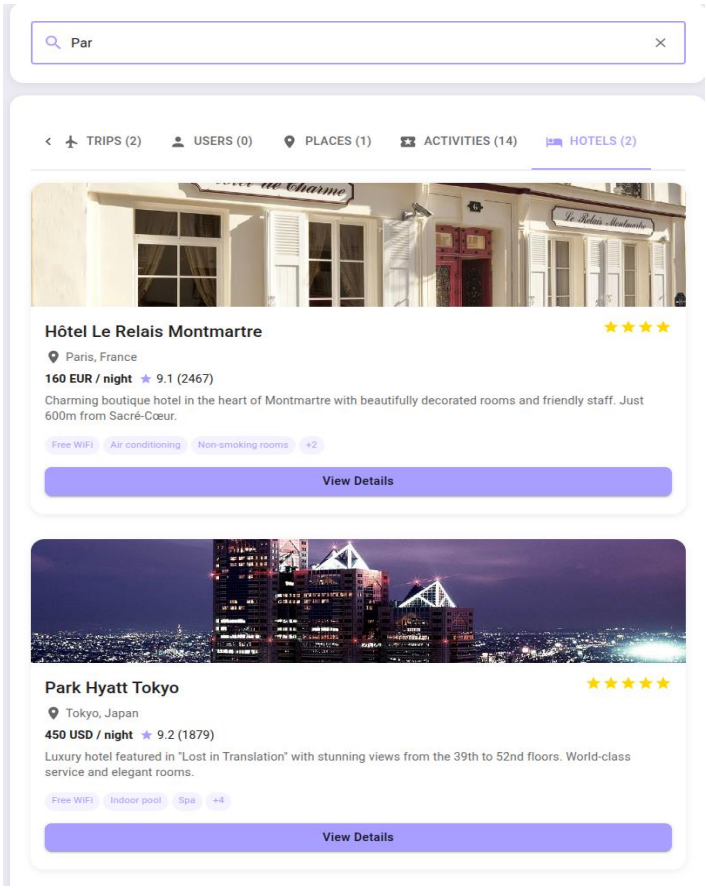

Рисунок 3.10 – Пошук готелів за запитом «Par»

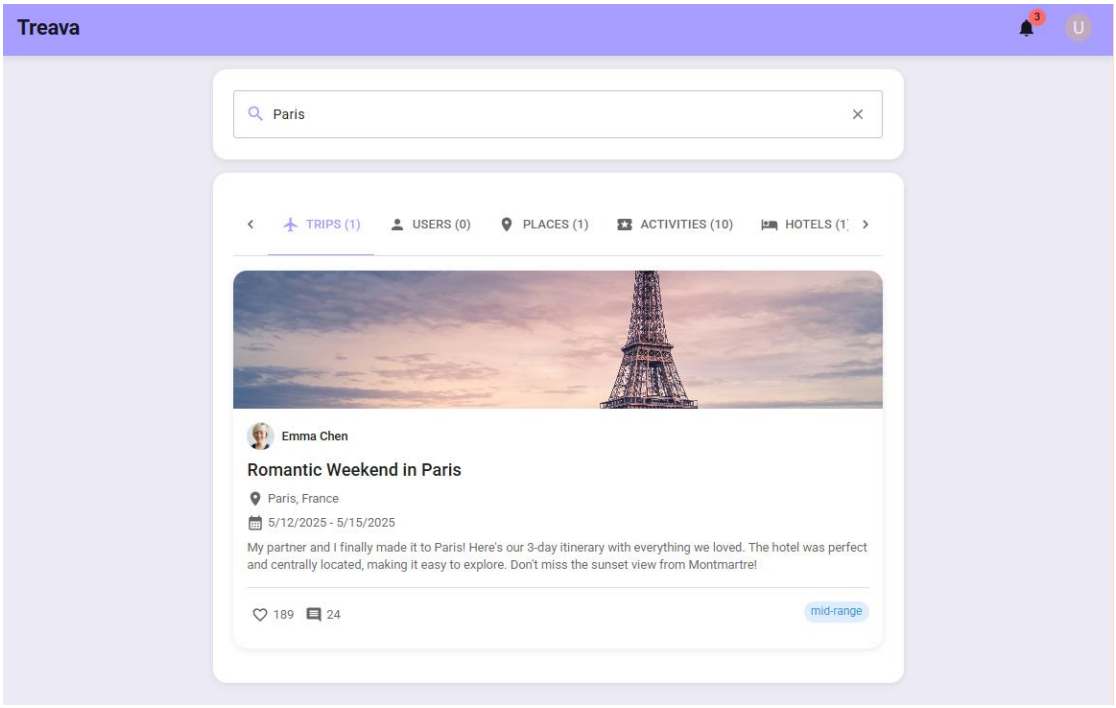

Рисунок 3.11 – Пошук постів за запитом «Paris»

Для кожного посту призначення система дозволяє внести інформацію про фото, маршрути, визначні пам'ятки, заклади харчування, транспорт, варіанти проживання та культурні події (див. рисунок 3.12).

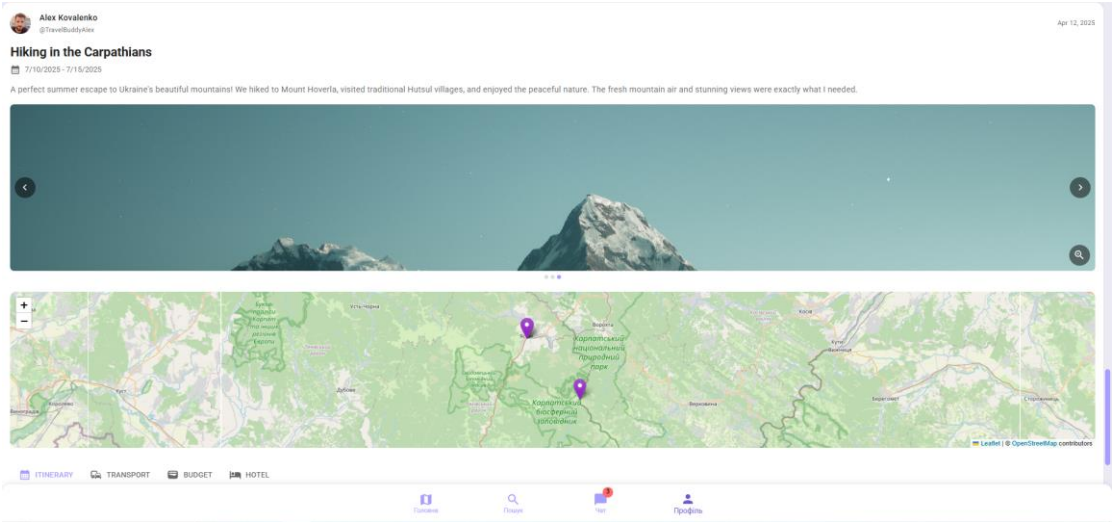

Рисунок 3.12 – Пост свореної подожі у вебсистемі

Важливою складовою системи є функція детального планування щоденного маршруту. Користувачам пропонується створювати розклад за днями з точним зазначенням часу відвідування кожного місця, вартості та додаткових приміток (див. рисунок 3.13). Такий підхід забезпечує оптимальне використання часу під час подорожі.

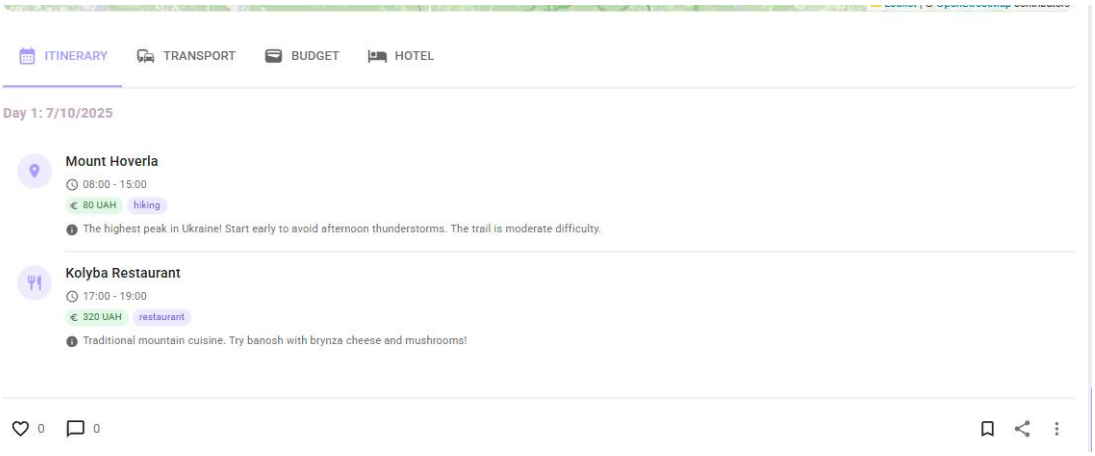

Рисунок 3.13 – Розклад подорожі по днях

Транспортний розділ системи дозволяє планувати всі аспекти переміщення, включаючи прибуття до пункту призначення, відправлення та локальні пересування (див. рисунок 3.14). Для кожного транспортного варіанту вказується час, вартість та особливості, що дозволяє обрати найзручніший спосіб пересування.

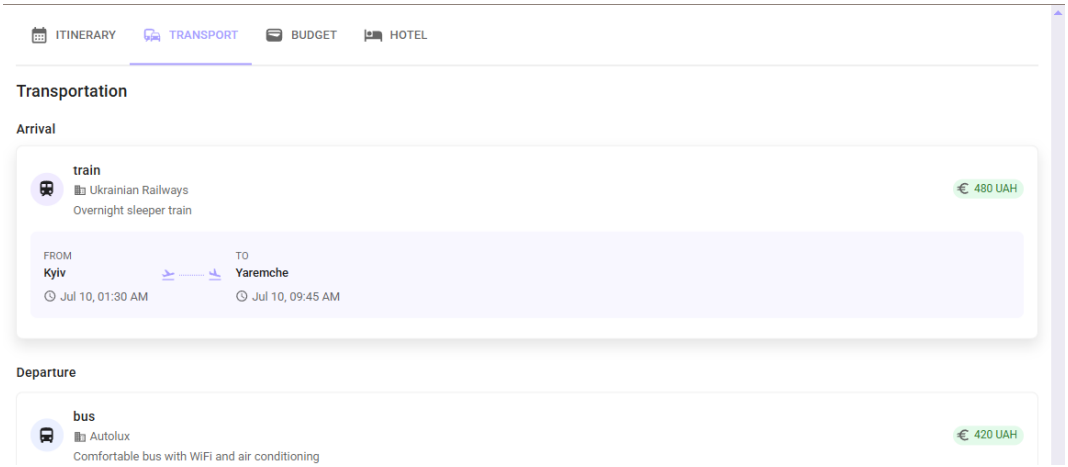

Рисунок 3.14 – Вкладка з інформацією про транспорт

Фінансове планування подорожі забезпечується через бюджетний модуль, який дозволяє створювати детальний розподіл витрат за категоріями: проживання, харчування, транспорт, активності та покупки, що автоматично розраховується з внесених даних про відповідні категорії (див. рисунок 3.15). Система візуалізує бюджет у вигляді діаграми та розраховує середні щоденні витрати.

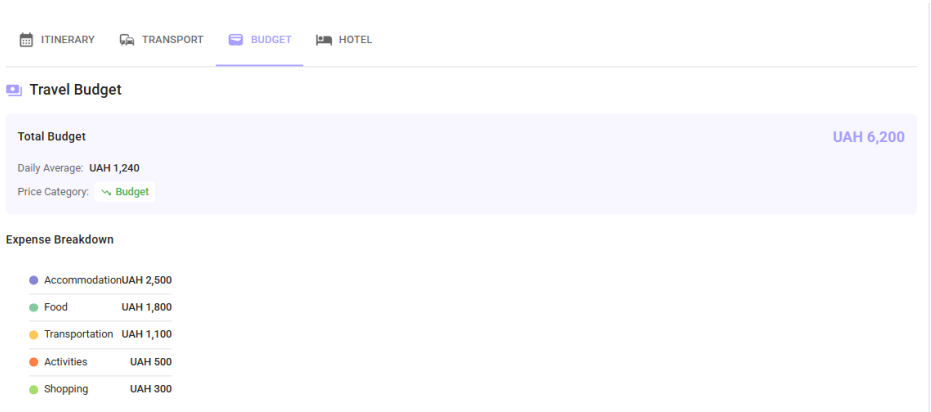

Рисунок 3.15 – Вкладка з інформацією про використання бюджету

Модуль проживання надає вичерпну інформацію про варіанти розміщення з фотографіями, описами, переліком зручностей, рейтингами та відгуками, що наведено на рисунку 3.16. Через інтеграцію з платформами бронювання користувач може легко забронювати обране житло перейшовни за актуальним посиланням на платформу бронювання.

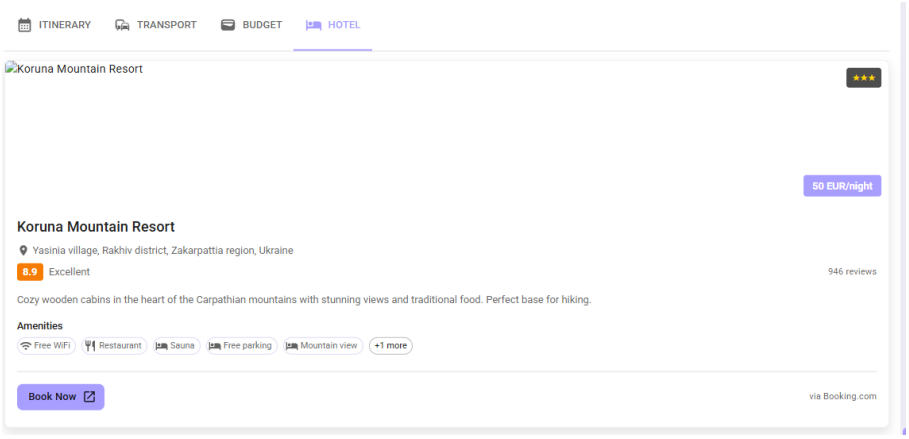

Рисунок 3.16 – Вкладка з інформаціє про готель

Культурна програма подорожі формується через модуль подій, який містить інформацію про концерти, вистави, фестивалі та інші заходи у місці призначення. Кожна подія супроводжується датою, місцем проведення, вартістю квитків та можливістю їх придбання (див. рисунок 3.17).

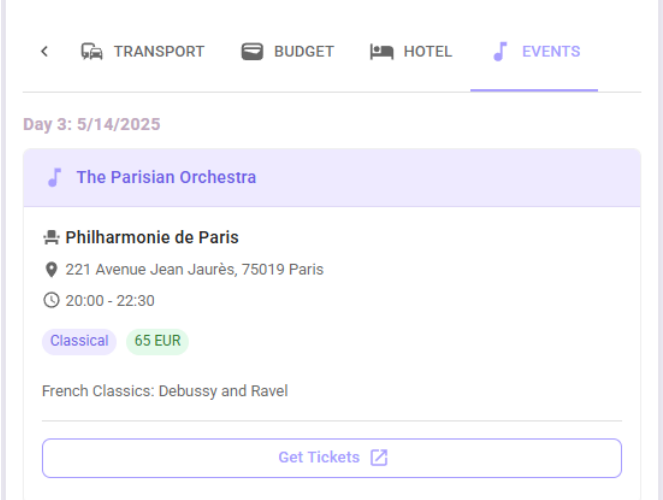

Рисунок 3.17 – Вкладка з інформаціє про події

Геопросторова складова системи реалізована через інтерактивні карти, на яких відмічаються всі включені до маршруту об'єкти (див. рисунок 3.18). Карти інтегровані з детальною інформацією про кожну локацію та маршрути між ними, що забезпечує просторову орієнтацію користувача.

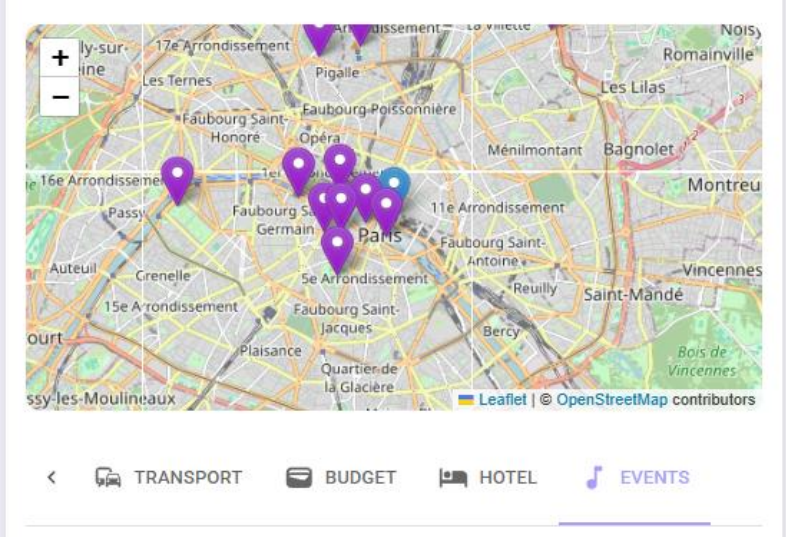

Рисунок 3.18 – Інтерактивна карта з відмітками

Комплексний підхід даної вебсистеми дозволяє не лише покращити процес підготовки до поїздки через об'єднання всіх необхідних інструментів на одній платформі, але й створює унікальну екосистему, де користувачі отримують натхнення, планують, документують та діляться власним досвідом подорожей в рамках єдиного інформаційного простору.

Розробка інтерфейсу користувача була виконана у середовищі розробки IntelliJ IDEA з використанням мови програмування JavaScript (React.js).

3.8 Висновки

Розробка вебсистеми для організації подорожей вимагає комплексного підходу до проектування та реалізації архітектури з використанням сучасних технологій. Представлений технологічний стек, що включає Java, Spring Boot, PostgreSQL, Apache Kafka, Docker, JavaScript (React.js) та Redis, забезпечує необхідний функціонал для створення надійної та масштабованої системи. Кожен з розроблених модулів – автентифікації та управління користувачами, організації публікацій подорожей, інтеграції з зовнішніми API, чатів у реальному часі, безпеки та шифрування даних, а також вебінтерфейсу програмного додатку – розв’язує специфічні задачі і разом утворює єдину екосистему для зручного планування, документування та обміну досвідом подорожей.

Особлива увага в системі приділяється інтуїтивності інтерфейсу, безпеці даних та соціальній взаємодії між користувачами, що відповідає сучасним тенденціям розвитку туристичних платформ. Реалізований функціонал дозволяє користувачам не лише створювати детальні плани подорожей , контролем бюджету та варіантами розміщення, але й ділитися своїм досвідом з іншими, комунікувати через систему чатів у реальному часі та використовувати актуальні дані зовнішніх сервісів при плануванні. Таким чином, розроблена вебсистема поєднує функціональність планувальника подорожей з елементами соціальної мережі для мандрівників.

4 ТЕСТУВАННЯ ВЕБСИСТЕМИ ДЛЯ ОРГАНІЗАЦІЇ ПОДОРОЖЕЙ

4.1 Тестування системи

Тестування програмного забезпечення є невід'ємним етапом процесу розробки, який спрямований на перевірку відповідності системи функціональним вимогам, виявлення дефектів та забезпечення належної якості продукту перед його впровадженням [28]. Основною метою тестування є виявлення невідповідностей між очікуваною та фактичною поведінкою системи, а також перевірка стабільності роботи всіх компонентів при різних сценаріях використання. Функціональне тестування перевіряє відповідність системи функціональним вимогам, описаним в технічній документації. Важливим етапом також є тестування інтерфейсу користувача, що включає перевірку зручності використання та адаптивності дизайну для різних пристроїв. Завершальним етапом є системне тестування, яке оцінює систему в цілому та її відповідність бізнес-вимогам.

Для початку тестування вебсистеми організації подорожей необхідно підготувати тестове середовище, що включає розгортання всіх компонентів системи в ізольованому оточенні з використанням Docker контейнерів, налаштування тестової бази даних PostgreSQL та підготовку тестових даних. Також розробляється план тестування, який охоплює всі функціональні модулі системи та описує тестові сценарії для кожного з них.

На представлених скріншотах відображено результати тестування різних функціональних аспектів вебсистеми. Першим етапом відбулось тестування форми реєстрації, що показало коректну роботу валідації полів, включаючи перевірку формату електронної пошти, унікальності імені користувача та відповідності паролів. Система коректно відображає помилки при введенні неправильних даних, зокрема повідомлення про невідповідність паролів (див. рисунок 4.1). Після усунення помилок система успішно завершує процес

реєстрації, про що свідчить відображення повідомлення «Registration successful!» (див. рисунок 4.2).

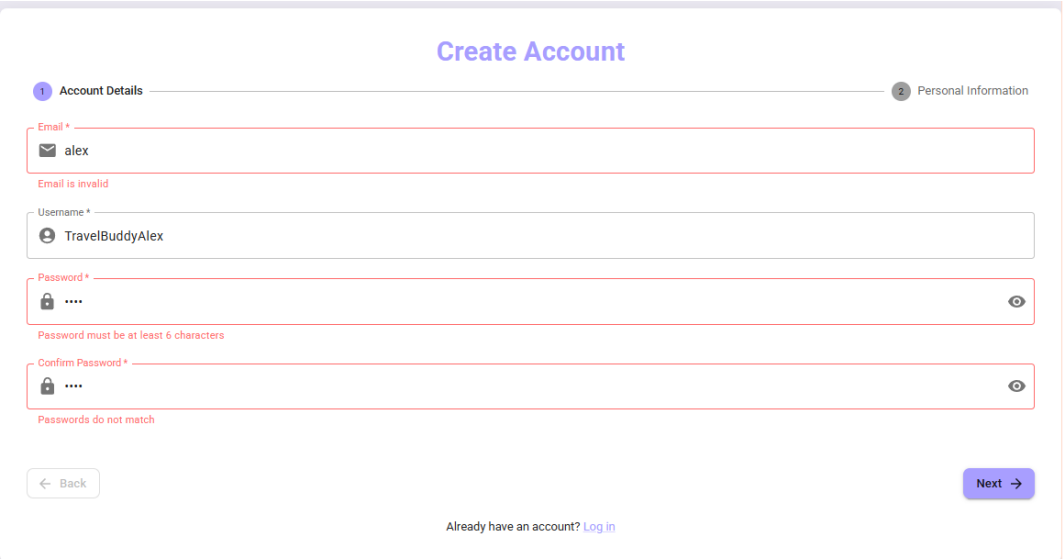

Рисунок 4.1 – Некоректно заповнена форма реєстрації

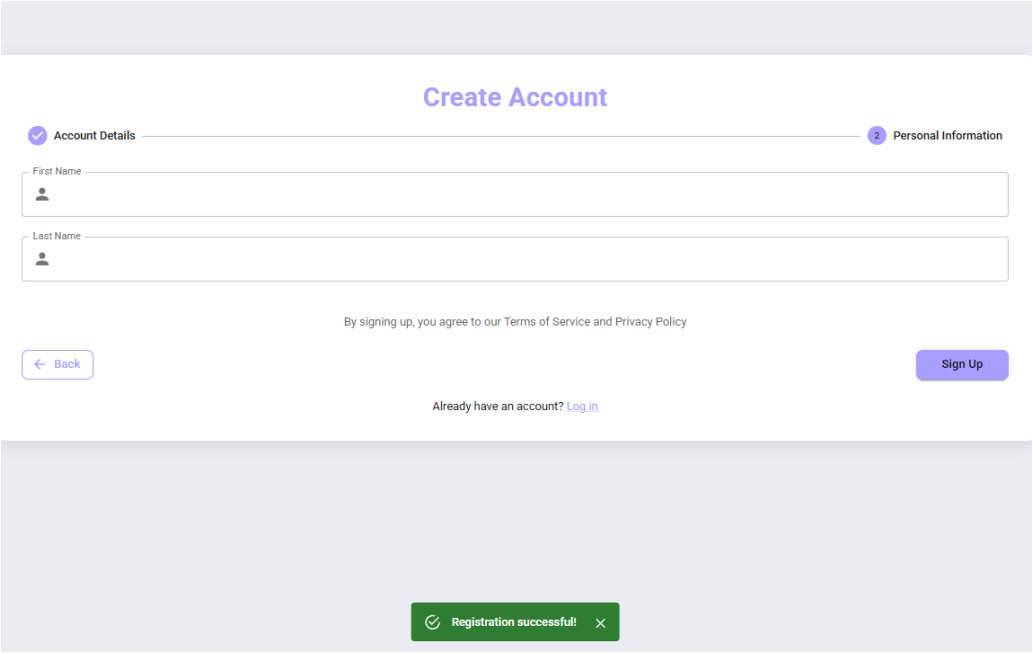

Рисунок 4.2 – Повідомлення про успішну реєстрацію

Наступним другим етапом відбулось тестування процесу реєстрації користувача, що включало перевірку всіх етапів створення облікового запису. Спочатку користувач заповнює форму з обов'язковими полями: електронна

пошта, ім'я користувача та пароль. Система виконує валідацію даних, перевіряючи унікальність електронної пошти та імені користувача в базі даних. Після підтвердження паролю користувач переходить до другого кроку, де вводить особисту інформацію – ім'я та прізвище. Після завершення реєстрації система автоматично авторизує користувача та перенаправляє його на головну сторінку та створює профіль (див. рисунок 4.3), що було успішно підтверджено під час тестування.

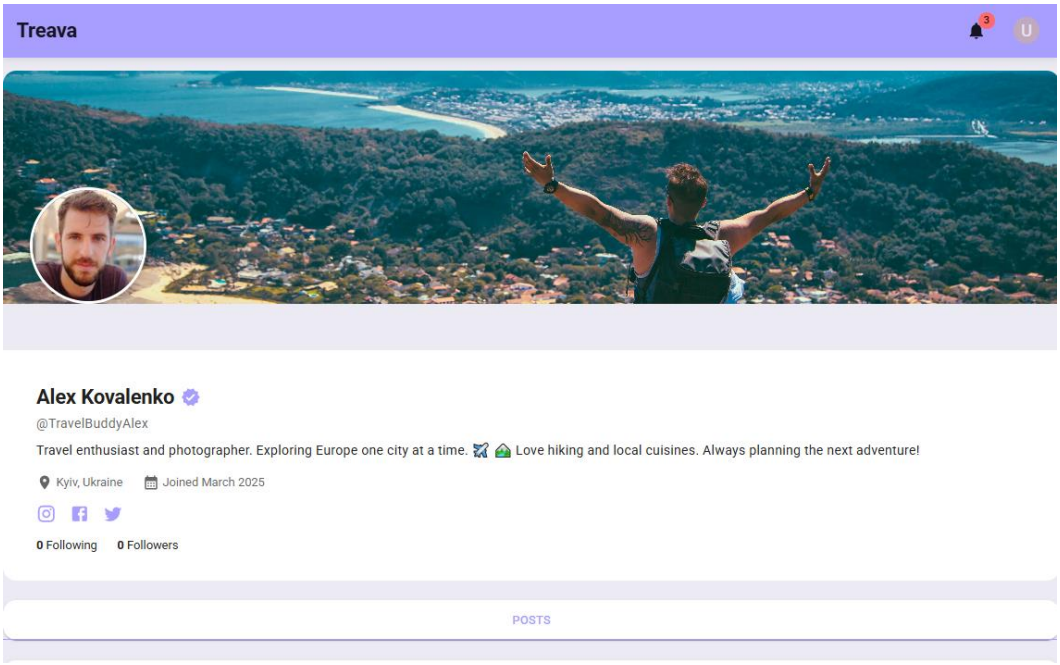

Рисунок 4.3 – Профіль користувача

Четвертим етапом стало тестування модуля комунікації, що включало перевірку функціональності як приватних, так і групових чатів. У приватному чаті перевірялася можливість надсилання текстових повідомлень між двома користувачами, коректне відображення часу повідомлень та статусу прочитання (див. рисунок 4.4). У груповому чаті тестувалася можливість додавання кількох учасників, відправка повідомлень всім учасникам групи та коректне відображення імен відправників (див. рисунок 4.5). Особлива увага приділялася тестуванню обміну посиланнями, зокрема на бронювання готелів, що є важливим функціоналом для системи організації подорожей. Результати

тестування підтвердили коректну роботу обох типів чатів, своєчасну доставку повідомлень та правильне форматування різних типів контенту.

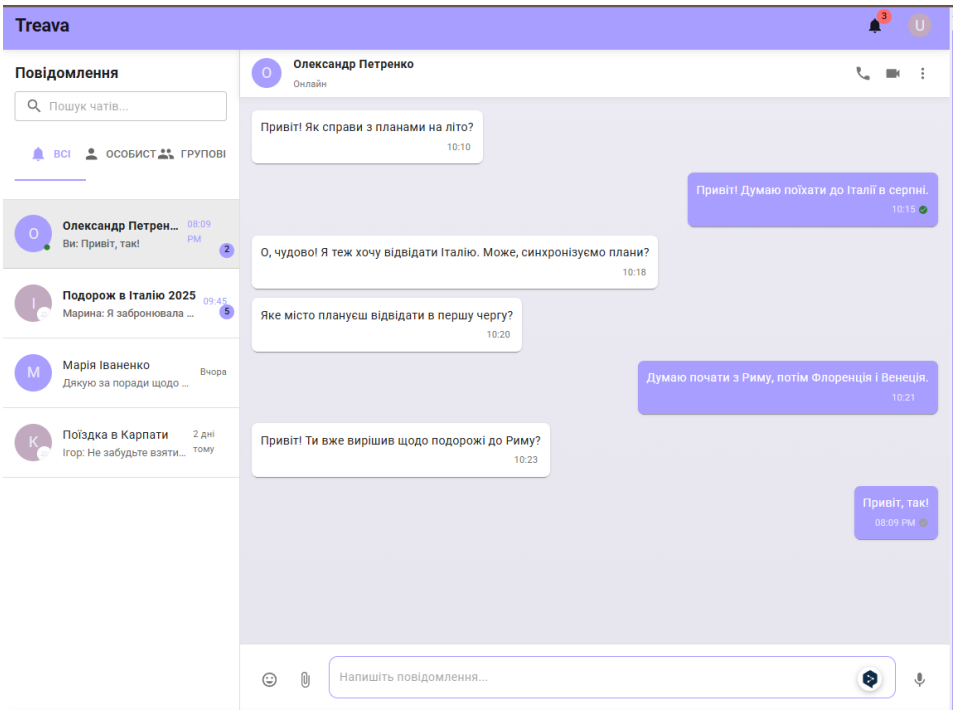

Рисунок 4.4 – Тестування приватного чату

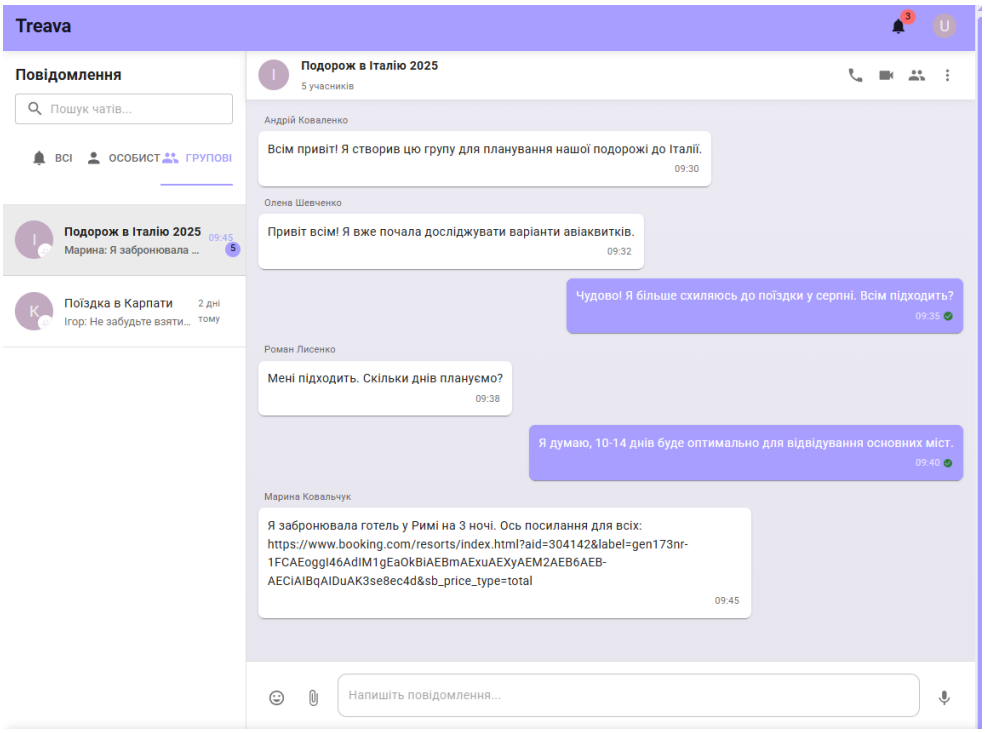

Рисунок 4.5 – Тестування групового чату

П’ятим етапом відбувся процес тестування створення подорожі, який охоплював всі етапи цього ключового функціоналу системи. На початковому екрані користувач вводить основну інформацію про подорож: назву, пункт призначення, дати початку та завершення (див. рисунок 4.6) та за потреби може додати фото подорожі. Система успішно зберігає ці дані та переходить до етапу планування днів.

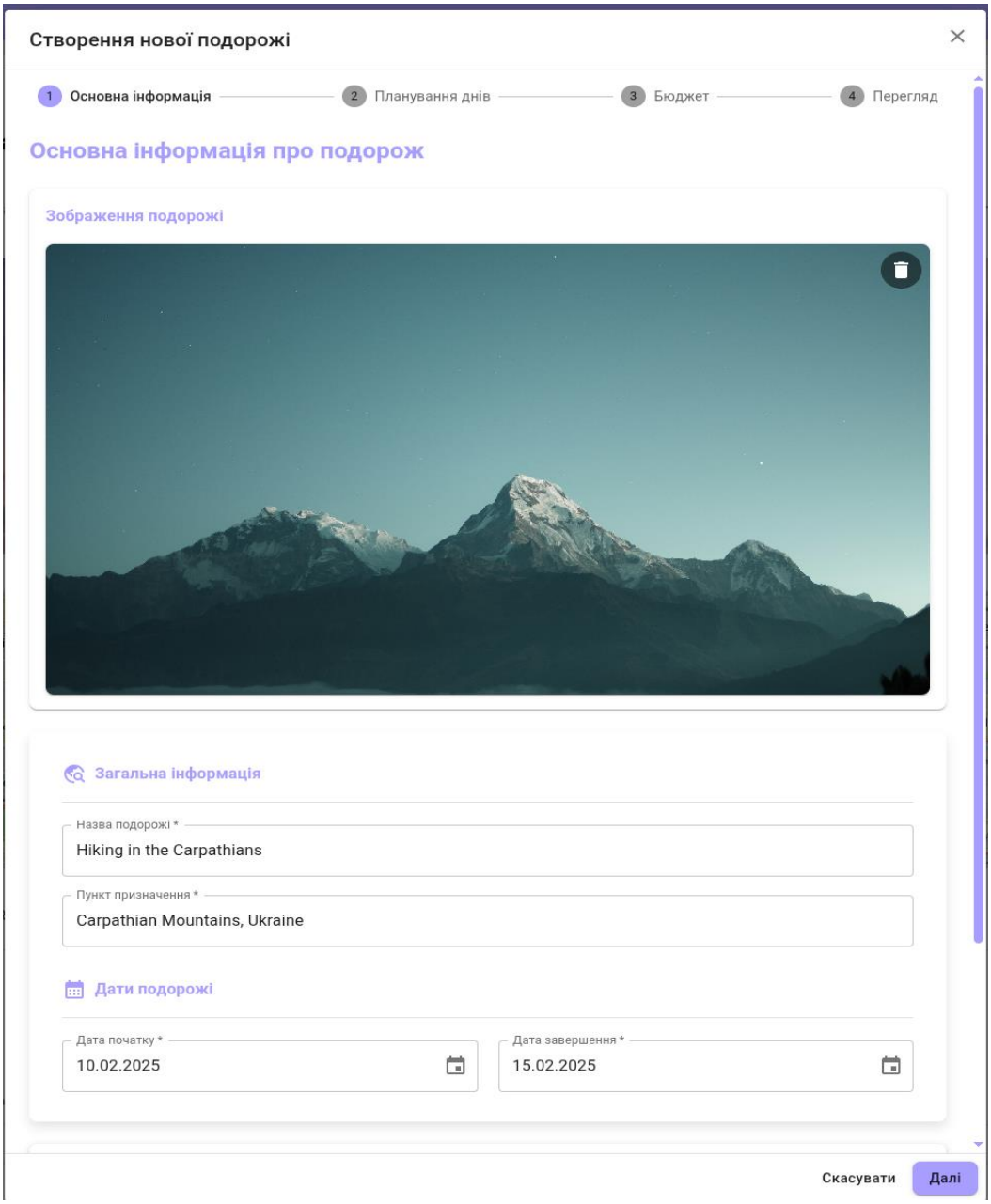

Рисунок 4.6 – Введення основної інформації про подорож

На шостому етапі тестування було перевірено планування днів подорожі, а саме можливість додавання різних типів активностей для кожного дня, вказання часу початку та завершення, вартості та приміток (див. рисунок 4.7).

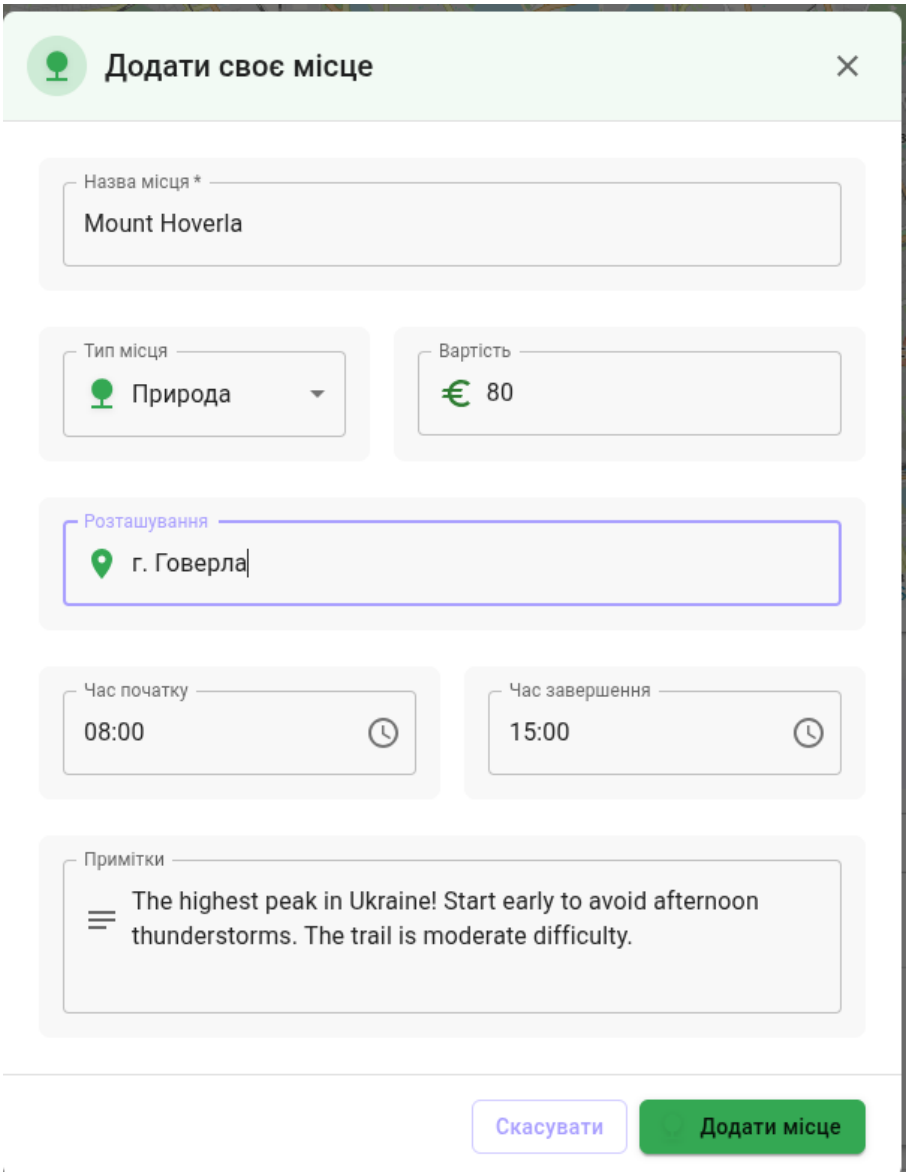

Рисунок 4.7 – Вікно створення активності

Тестування підтвердило коректну роботу інтерфейсу додавання активностей, можливість редагування та видалення вже доданих пунктів (див. рисунок 4.8), а також автоматичне сортування активностей за часом. Система правильно розраховує тривалість кожної активності та відображає загальний час, запланований на кожен день.

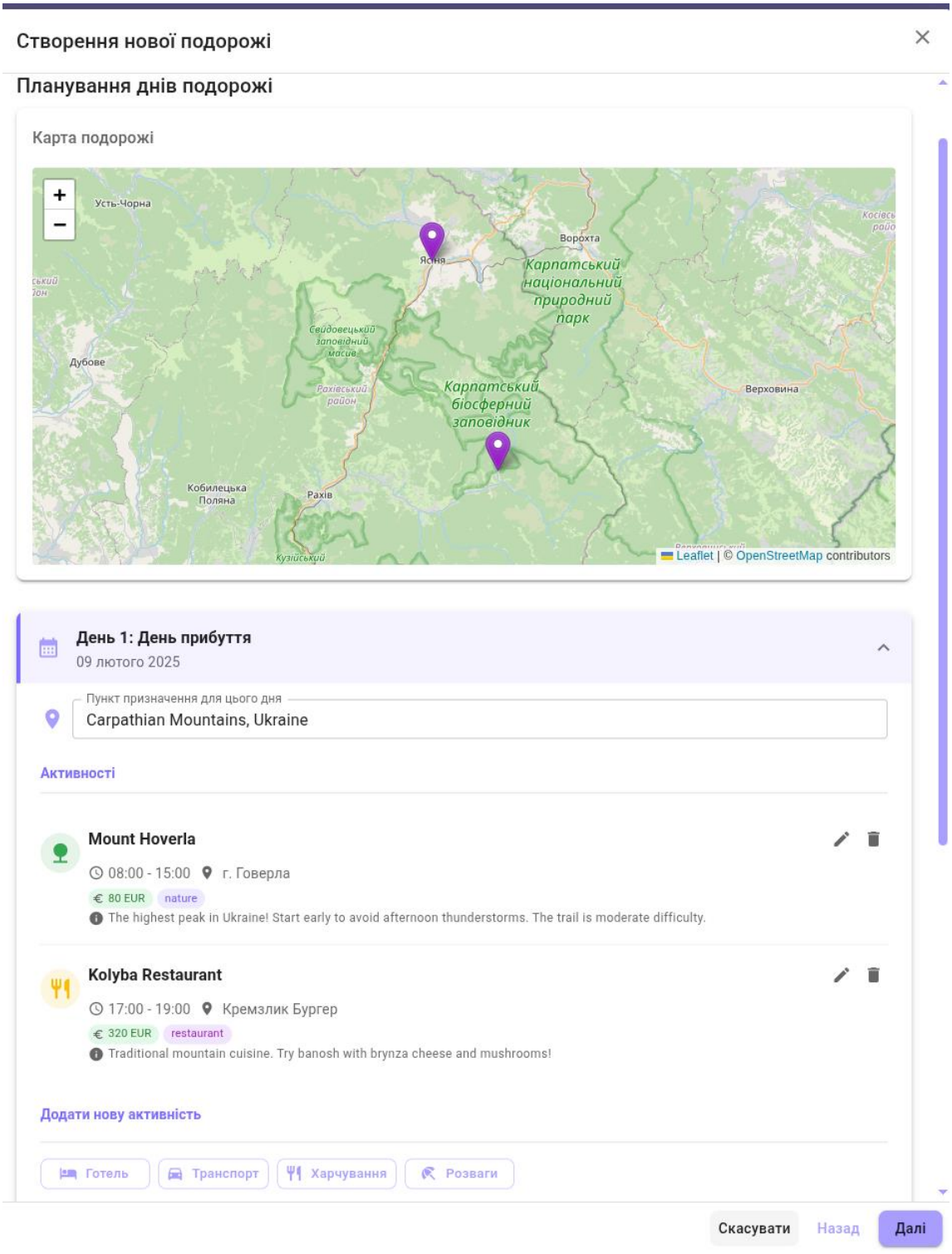

Рисунок 4.8 – Планування днів подорожі

На сьомому етапі тестування перевірялася бюджетування та коректність автоматичного розрахунку витрат за категоріями на основі вже введених даних про активності. Система успішно агрегує витрати за категоріями «Житло»,

«Транспорт», «Харчування», «Розваги та екскурсії», «Шопінг та сувеніри» та «Інше», візуалізує їх у вигляді прогрес-барів та надає рекомендації щодо оптимального розподілу бюджету (див. рисунок 4.10).

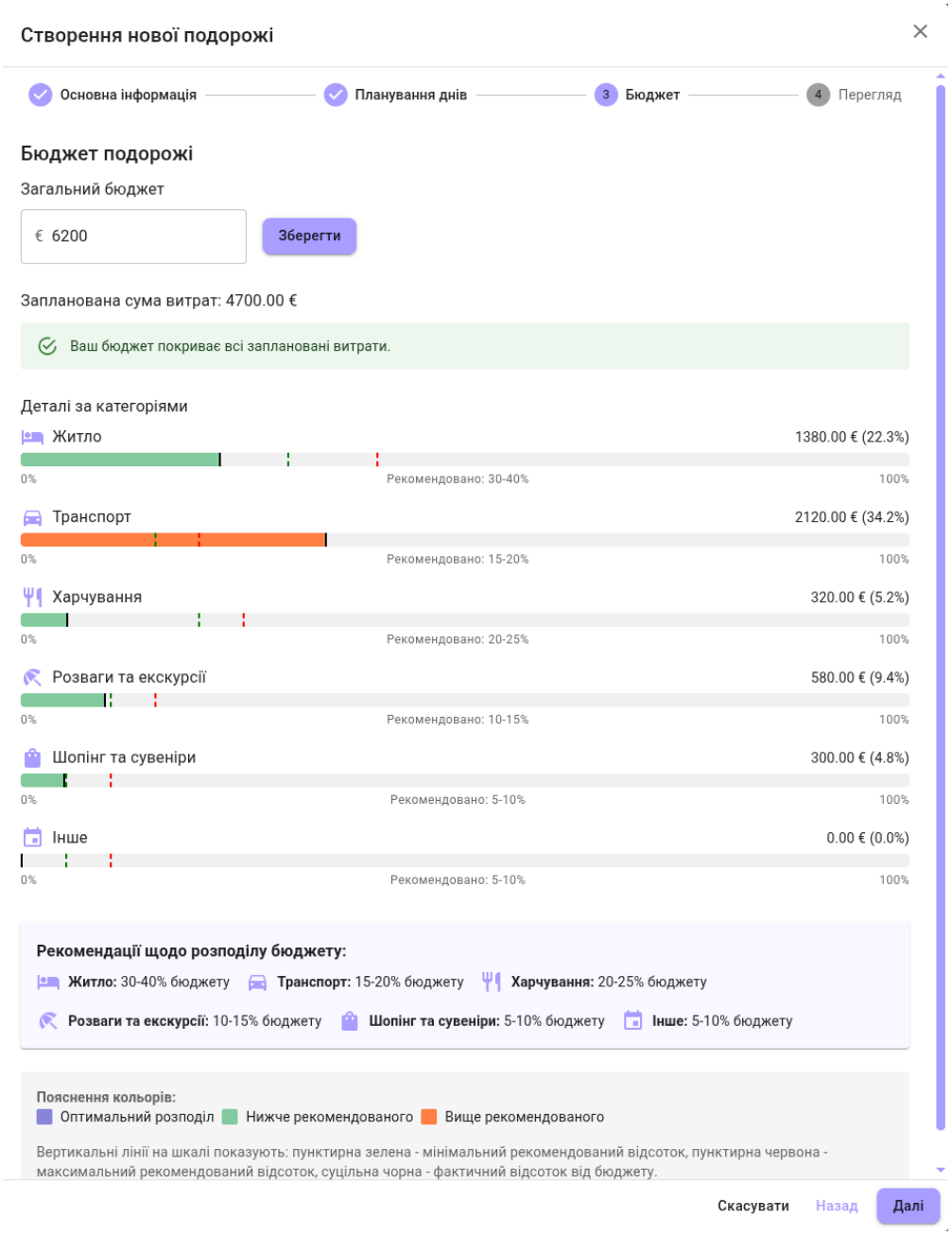

Рисунок 4.10 – Вікно планування бюджету

Тестування підтвердило коректність розрахунків відсоткового співвідношення витрат до загального бюджету та адекватність рекомендацій системи щодо розподілу витрат.

Завершальний восьмий етап тестування включав перевірку фінального перегляду створеної подорожі, де система відображає всю введену інформацію в структурованому вигляді для фінальної перевірки користувачем, що показано на рисунку 4.11.

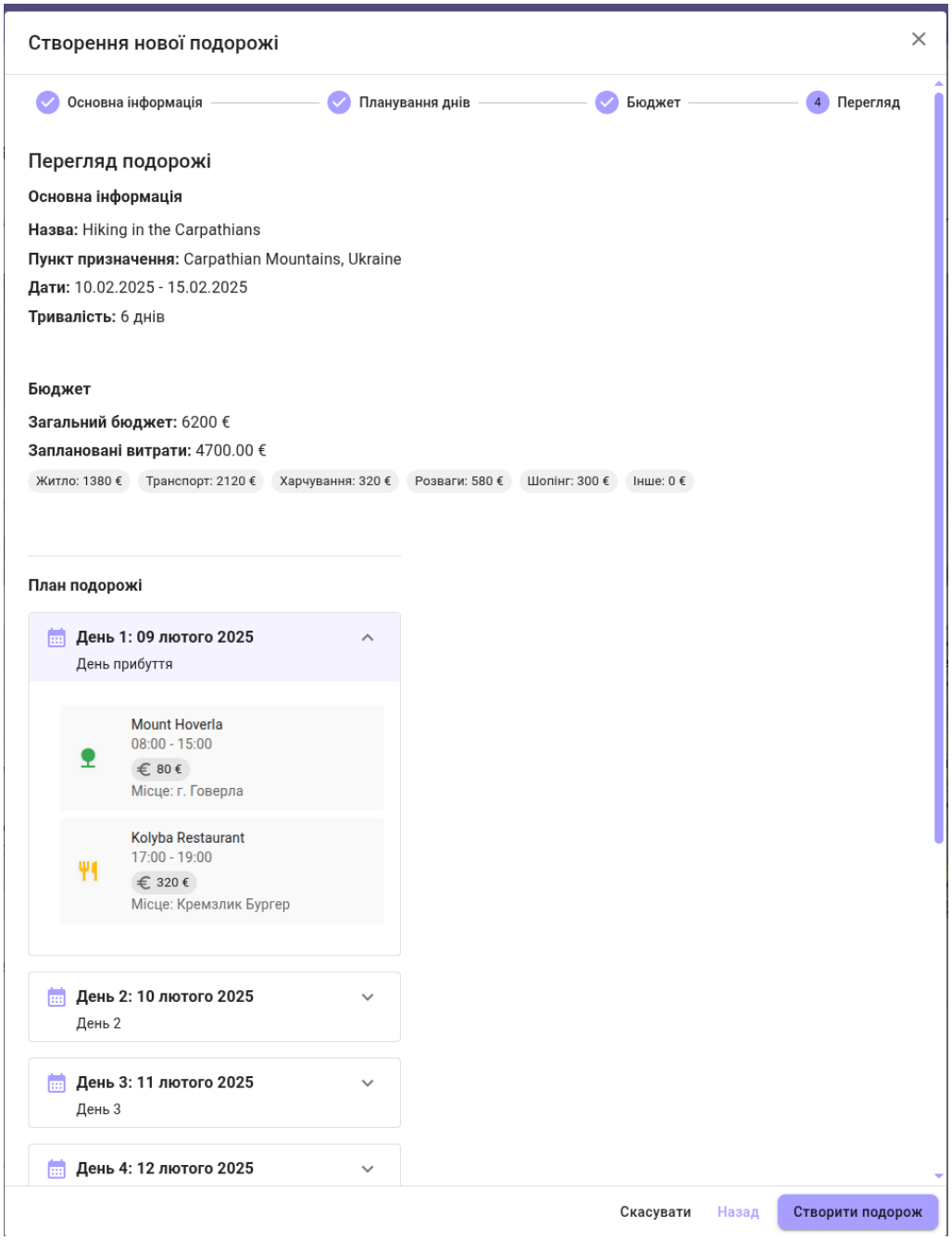

Рисунок 4.11 – Вікно фінального перегляду подорожі продовження

Після натискання кнопки «Створити подорож» система успішно зберігає всі дані в базі даних та публікує пост у профілі користувача. Тестування підтвердило, що створений пост коректно відображається у профілі користувача з усіма введеними деталями подорожі, включаючи карту з відмітками запланованих локацій та хронологічно впорядкований список активностей за днями (див. рисунок 4.12 та 4.13) .

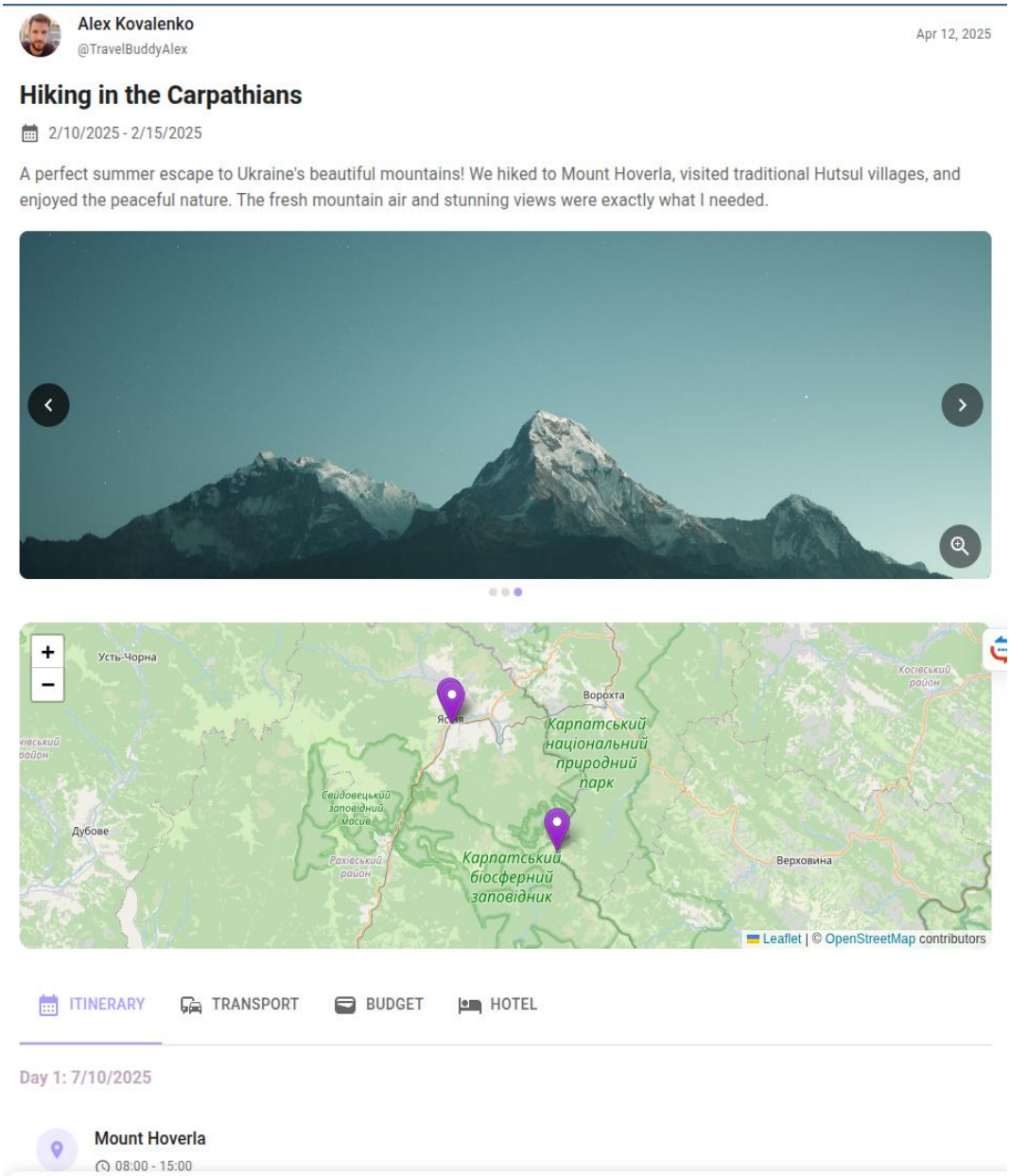

Рисунок 4.12 – Успішно створений пост

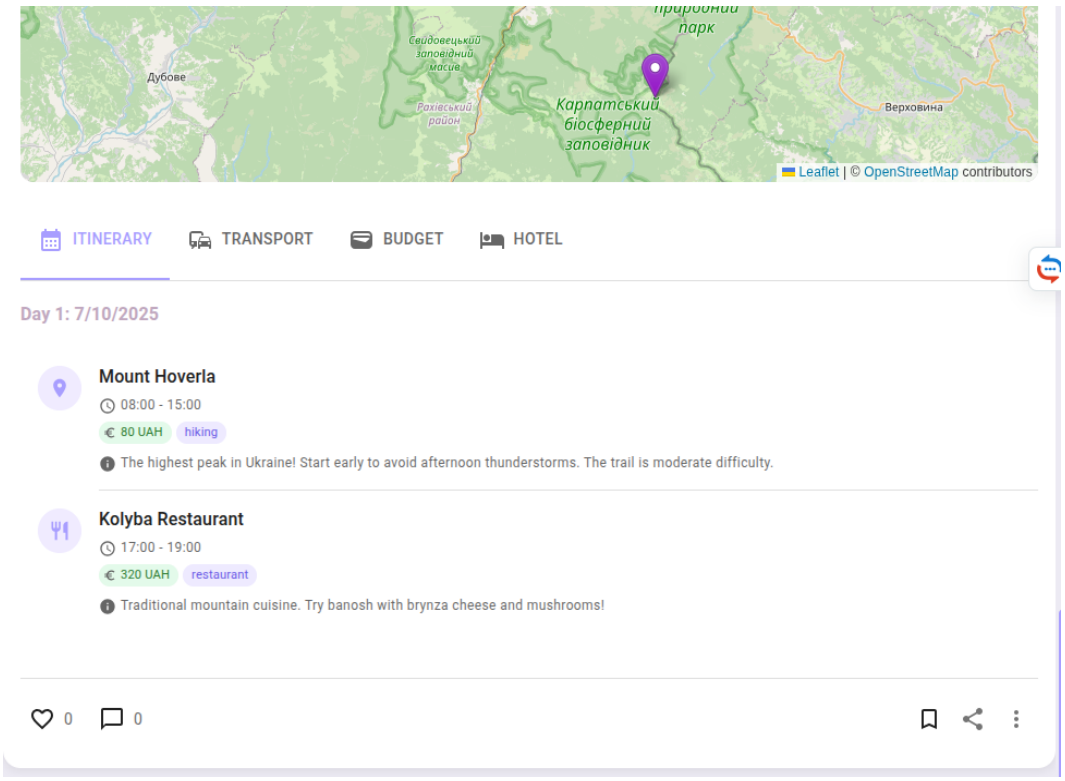

Рисунок 4.13 – Успішно створений пост розбивка по днях

Результати комплексного тестування вебсистеми організації подорожей підтвердили відповідність розробленого продукту функціональним вимогам, стабільність роботи всіх модулів та зручність користувацького інтерфейсу. Виявлені під час тестування незначні дефекти були успішно усунені, що дозволяє рекомендувати систему до впровадження в експлуатацію.

4.2 Розробка інструкції користувача

Вебсистема організації подорожей працює через інтернет-браузер без встановлення додаткових програм. Для роботи підходять браузери Google Chrome версії 90 і новіші, Mozilla Firefox версії 88 і новіші, Safari версії 14 і новіші або Microsoft Edge версії 90 і новіші. Комп'ютер або мобільний пристрій повинен мати процесор з частотою від 1.6 ГГц, мінімум 4 ГБ оперативної пам'яті та інтернет-з'єднання зі швидкістю від 5 Мбіт/с. Система автоматично адаптується під розмір екрану, забезпечуючи зручну роботу на комп'ютерах, планшетах та смартфонах. Для повноцінного використання функцій

фотографування та визначення місцезнаходження потрібно надати системі відповідні дозволи.

Реєстрація нового користувача починається з головної сторінки системи, де розташована кнопка Sign Up. Після її натискання відкривається форма реєстрації з полями для електронної пошти, імені користувача та паролю. Пароль має містити щонайменше 6 символів для забезпечення безпеки облікового запису. Система перевіряє унікальність електронної адреси та імені користувача в базі даних. На другому етапі реєстрації вводяться особисті дані - ім'я та прізвище, після чого обліковий запис створюється та користувач автоматично входить у систему. Для повторного входу використовується кнопка Log in на головній сторінці з введенням електронної пошти та паролю.

Створення подорожі розпочинається з натискання відповідної кнопки на головній сторінці. На першому етапі заповнюються поля з назвою подорожі, пунктом призначення, датами початку та завершення. Додатково можна завантажити головне фото подорожі, яке відображатиметься в публікації.

Планування днів подорожі відбувається на окремому екрані з переліком усіх днів між датами початку та завершення. Кожен день розкривається окремо для додавання активностей. Система пропонує категорії: готель для проживання, транспорт для переїздів, харчування для ресторанів та кафе, розваги для екскурсій та відвідування пам'яток, а також можливість додати власне місце. Для кожної активності заповнюються назва, час початку та завершення, вартість та додаткові примітки. Активності автоматично сортуються за часом, створюючи логічну послідовність дня.

Бюджетування подорожі відбувається автоматично на основі введених витрат для кожної активності. Система розподіляє витрати за категоріями та відображає їх у вигляді наочних діаграм. Загальний бюджет можна коригувати, і система покаже відповідність запланованих витрат наявним коштам через кольорові індикатори. Зелений колір означає достатність бюджету, жовтий попереджає про наближення до ліміту, червоний сигналізує про перевищення.

Система також надає рекомендації щодо розподілу коштів між різними категоріями на основі досвіду інших користувачів.

Фінальний етап створення подорожі включає перегляд усієї введеної інформації на одному екрані. Тут відображаються основні дані подорожі, детальний план по днях з усіма активностями та підсумковий бюджет. За потреби можна повернутися до попередніх кроків для внесення змін. Після натискання кнопки створення система зберігає подорож та публікує її в профілі користувача, де вона стає доступною для перегляду іншими учасниками спільноти.

Стрічка публікацій на головній сторінці показує подорожі інших користувачів у хронологічному порядку. Кожна публікація містить фотографії, карту маршруту, короткий опис та основну інформацію про тривалість і бюджет. Взаємодія з публікаціями включає можливість поставити вподобання, залишити коментар або зберегти пост для подальшого перегляду. Збережені публікації доступні на головній сторінці на окремій вкладці, звідки їх можна переглядати та видаляти.

Пошукова система дозволяє знаходити контент за різними критеріями. У пошуковий рядок вводиться назва міста, країни або ключові слова, після чого можна обрати категорію пошуку: всі результати, активності, готелі, ресторани або публікації користувачів. Результати відображаються з фотографіями та короткими описами. Детальна інформація про кожен об'єкт відкривається при натисканні на його назву або зображення.

Система обміну повідомленнями забезпечує спілкування між користувачами в реальному часі. Приватні розмови починаються з вибору користувача через його профіль. Групові чати створюються для обговорення спільних подорожей з можливістю додавання кількох учасників. Повідомлення відправляються натисканням кнопки або клавіші Enter. Історія переписки зберігається та доступна при кожному вході в систему.

Профіль користувача містить особисту інформацію, створені публікації. Тут можна редагувати дані профілю, переглядати власні подорожі.

4.3 Висновки

У четвертому розділі було проведено тестування функціональних модулів вебсистеми для організації подорожей. Під час тестування було перевірено працездатність усіх компонентів системи в різних сценаріях використання та з різними вхідними даними. Також у розділі було розроблено детальну інструкцію користувача, яка охоплює всі аспекти роботи з вебсистемою – від початкової реєстрації до створення детальних планів подорожей та взаємодії з іншими користувачами. Інструкція структурована у вигляді покрокових вказівок, що забезпечує зручність використання навіть для користувачів без попереднього досвіду роботи з подібними системами.

ВИСНОВКИ

У рамках бакалаврської кваліфікаційної роботи була розроблена вебсистема для організації подорожей, що забезпечує комплексний підхід до планування, документування та обміну туристичним досвідом. Для розробки використовувалися сучасні технології, включаючи Java, Spring Boot, PostgreSQL, ElasticSearch, Apache Kafka, Docker, JavaScript (React.js) та Redis, що забезпечило створення надійної, масштабованої та продуктивної системи. Реалізація бакалаврчької кваліфікаційної роботи відповідає методичним вказівкам та сучасним вимогам до розробки програмного забезпечення.

Під час виконання бакалаврської кваліфікаційної роботи було проведено аналіз сучасного стану програмних засобів організації подорожей. Були розглянуті основні аналоги вебсистем для планування подорожей, виявлені їхні недоліки та порівняно з власним розробленим продуктом. На основі цього порівняння були сформульовані основні завдання бакалаврської кваліфікаційної роботи та визначено функціональні вимоги до системи.

Під час аналізу технологій розробки було обґрунтовано вибір технологічного стеку, що забезпечує оптимальне поєднання продуктивності, безпеки та зручності розробки. Архітектура системи спроектована з урахуванням можливості масштабування та інтеграції нових функцій у майбутньому.

У бакалаврській кваліфікаційній роботі було зроблено наступне:

- розроблено архітектуру вебсистеми, яка забезпечить високу продуктивність, масштабованість та безпеку даних;
- розроблено модуль автентифікації та управління користувачами, який забезпечить надійну ідентифікацію та захист персональних даних;
- розроблено модуль для організації публікацій подорожей з можливістю детального планування по днях;
- розроблено модуль інтеграції з зовнішніми API для отримання актуальної інформації про туристичні об'єкти, транспорт та бронювання;

- розроблено модуль чатів у реальному часі для комунікації між користувачами та обміну досвідом;
- розроблено модуль безпеки та шифрування даних для захисту конфіденційної інформації;
- інтегровано різні модулі в єдину вебсистему для комплексної організації подорожей.

Було розроблено детальну архітектуру системи з використанням діаграм UML, а також схеми взаємодії компонентів та структуру бази даних. Особлива увага приділялася безпеці системи та захисту персональних даних користувачів.

Тестування вебсистеми довело повну працездатність усіх модулів та відповідність поставленому технічному завданню. Система успішно функціонує в різних браузерах та на різних пристроях, забезпечуючи зручний доступ до функціональності як з десктопних, так і з мобільних пристроїв. Також було розроблено детальну інструкцію користувача.

Розроблена вебсистема для організації подорожей має значний потенціал для практичного використання індивідуальними мандрівниками, туристичними агенціями та організаціями, що займаються плануванням корпоративних подорожей. Інтеграція соціальних функцій та комунікації у реальному часі створює унікальну вебсистему, де користувачі можуть не тільки планувати власні подорожі, але й обмінюватися досвідом та рекомендаціями.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бондаренко Н.О., Бабюк Н. П. Використання графових алгоритмів для побудовиоптимальних маршрутів. *Стан, досягнення і перспективи інформаційних систем і технологій* : матеріали XXV Всеукр. наук.-тех. конф., 17-18 квіт. 2025 р. Одеса : ОНТУ, 2025. С. 97-99.

2. Бондаренко Н.О., Бабюк Н. П. Безпека даних у сучасних додатках. *Стан, досягнення і перспективи інформаційних систем і технологій* : матеріали XXV Всеукр. наук.-тех. конф., 17-18 квіт. 2025 р. Одеса : ОНТУ, 2025. С. 34-36

3. Бондаренко Н.О., Бабюк Н.П.. Методи забезпечення цілісності даних в розподілених системах. *Комп`ютерні ігри та мультимедіа як інноваційний підхід до комунікації – 2024* : матеріали IV Всеукр. наук.-тех. конф., 26-27 вер. 2024 р. Одеса : ОНТУ, 2024. С. 216-217

4. OpenStreetMap : веб-сайт. URL: https://www.openstreetmap.org/ (дата звернення: 27.03.2025).

5. Hotelbeds API : веб-сайт. URL: https://developer.hotelbeds.com/documentation/hotels/ (дата звернення: 27.03.2025).

6. TicketMaster. Discovery API : веб-сайт. URL: https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/ (дата звернення: 27.03.2025).

7. Maps.me : веб-сайт. URL: https://maps.me/ (дата звернення: 27.03.2025).

8. Tripadvisor : веб-сайт. URL: https://www.tripadvisor.com/ (дата звернення: 27.03.2025).

9. Rome2rio : веб-сайт. URL: https://www.rome2rio.com/ (дата звернення: 27.03.2025).

10. Skyscanner : веб-сайт. URL: https://www.skyscanner.fi/ (дата звернення: 27.03.2025).

11. Airbnb : веб-сайт. URL: https://www.airbnb.com/ (дата звернення: 27.03.2025).

12. Redis for AI documentation : веб-сайт. URL: https://redis.io/docs/latest/develop/ai/ (дата звернення: 14.03.2025).

13. Secure applications identities and protect sensitive data : веб-сайт. URL: https://www.hashicorp.com/es/products/vault (дата звернення: 14.03.2025).

14. WebSocket : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (дата звернення: 14.03.2025).

15. Getting started with React: веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/React_getting_started (дата звернення: 14.03.2025).

16. Heckler M. Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications – 1st Edition. – Sebastopol: O'Reilly Media, 2021. – 310 p.

17. What is Apache Kafka? : веб-сайт. URL: https://aws.amazon.com/what-is/apache-kafka/ (дата звернення: 14.04.2025).

18. Poulton Nigel. Getting Started with Docker. Macclesfield: Nigel poulton ltd, 2023. – 138 p.

19. Elasticsearch: What it is, How it works, and what it’s used for : веб-сайт. URL: https://www.knowi.com/blog/what-is-elastic-search/ (дата звернення: 09.04.2025).

20. What Is Argon2? : веб-сайт. URL: https://jumpcloud.com/it-index/what-is-argon2 (дата звернення: 14.04.2025).

21. How Vault works : веб-сайт. URL: https://developer.hashicorp.com/vault/docs/about-vault/how-vault-works (дата звернення: 14.04.2025).

22. Understanding AES Encryption and AES/GCM Mode: An In-Depth Exploration using Java : веб-сайт. URL: https://medium.com/@pravallikayakkala123/understanding-aes-encryption-and-aes-

gcm-mode-an-in-depth-exploration-using-java-e03be85a3faa (дата звернення: 14.04.2025).

23. Monolithic vs. Microservices Architecture : веб-сайт. URL: https://www.geeksforgeeks.org/monolithic-vs-microservices-architecture/ (дата звернення: 02.05.2025).

24. Loy M. Learning Java / Marc Loy, Patrick Niemeyer, Daniel Leuck. – 5th Edition. – O'Reilly Media, 2020. – 1010 p.

25. What is PostgreSQL? : веб-сайт. URL: https://aws.amazon.com/rds/postgresql/what-is-postgresql/ (дата звернення: 02.05.2025).

26. What is Amazon S3? : веб-сайт. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html (дата звернення: 02.05.2025).

27. What Is Two-Factor Authentication (2FA)? How It Works and Example : веб-сайт. URL: https://www.investopedia.com/terms/t/twofactor-authentication-2fa.asp (дата звернення: 02.05.2025).

28. Що таке тестування ПЗ, його етапи, види, інструменти : веб-сайт. URL: https://university.sigma.software/what-is-software-testing/ (дата звернення: 10.05.2025).

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти та науки України
Вінницький національний технічний університет
факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., професор О. Н. Романюк
«24» 03 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка програмного засобу для організації подорожей» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доцент Н. П. Бабюк
«24» 03 2025 р.
Виконав:
студентка гр. 5ПІ-21б Н.О. Бондаренко
«24» 03 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка програмного засобу для організації подорожей».

Галузь застосування – туризм та подорожі.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від «20» березня 2025 р. ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою дослідження є підвищення зручності планування та організації подорожей за рахунок розробки комплексної вебсистеми, що дозволяє інтегрувати всі етапи підготовки та документування туристичного досвіду з елементами соціальної взаємодії між мандрівниками.

Призначення роботи – розробка програмного засобу для організації подорожей.

4. Вихідні дані для проведення НДР

1. Бондаренко Н.О., Бабюк Н. П. Безпека даних у сучасних додатках. *Стан, досягнення і перспективи інформаційних систем і технологій* : матеріали XXV Всеукр. наук.-тех. конф., 17-18 квіт. 2025 р. Одеса : ОНТУ, 2025. С. 34-36.

2. Heckler M. Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications – 1st Edition. – Sebastopol: O'Reilly Media, 2021. – 310 p.

3. Redis for AI documentation : веб-сайт. URL: https://redis.io/docs/latest/develop/ai/ (дата звернення: 14.04.2025).

4. Getting started with React : веб-сайт. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/React_getting_started (дата звернення: 14.04.2025).

5. Poulton Nigel. Getting Started with Docker. Macclesfield: Nigel poulton ltd, 2023. – 138 p.

6. What is Apache Kafka? : веб-сайт. URL: https://aws.amazon.com/what-is/apache-kafka/ (дата звернення: 14.04.2025).

7. What is PostgreSQL? : веб-сайт. URL: https://aws.amazon.com/rds/postgresql/what-is-postgresql/ (дата звернення: 02.05.2025).

5. Технічні вимоги

Тип програми – вебзастосунок; вхідні дані: інформація про подорожі, користувацькі профілі, публікації з розбивкою по днях, маршрути, геолокаційні дані, медіафайли; середовище розробки – IntelliJ IDEA; мови програмування – Java, JavaScript; фреймворки – Spring Boot, React.js з Material UI та Leaflet; системи керування базами даних – PostgreSQL, Redis; додаткові технології – Elasticsearch для пошуку, WebSockets для чатів, Docker для контейнеризації.

6. Конструктивні вимоги.

Вебзастосунок повинен відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до БКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз розвитку технологій організації подорожей з використанням вебсистем та постановка задач дослідження	25.03.25 – 06.04.25
2	Розробка моделі і алгоритмів вебсистеми для організації подорожей	07.04.25 – 18.04.25
3	Розробка програмного забезпечення вебсистеми для організації подорожей	19.04.25 – 04.05.25
4	Тестування вебсистеми для організації подорожей	05.05.25 – 18.05.25
5	Оформлення матеріалів до захисту БКР	19.05.25 – 30.05.25

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б – Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного засобу для організації подорожей

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

Підрозділ кафедра ПЗ, ФІТКІ, 5ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 6.7 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

_ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

_ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Бабюк Н. П., к.т.н., доц. каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Бондаренко Н. О.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програмного забезпечення

```
import React, { useCallback } from 'react';
import { Box, useMediaQuery, useTheme } from '@mui/material';
import ChatList from './ChatList';
import ChatWindow from './ChatWindow';
import MembersInfo from './MembersInfo';
import useChatActions from '../../hooks/useChatActions';

/**
 * Основний компонент чату, що об'єднує всі компоненти
 * та керує станом чату
 */
const Chat = () => {
    const theme = useTheme();
    const isSmallScreen = useMediaQuery(theme.breakpoints.down('sm'));

    const {
        chats,
        isLoading,
        selectedChat,
        isMessagesLoading,
        activeTab,
        searchQuery,
        showMembersInfo,
        groupMembers,
        filteredChats,
        selectChat,
        handleTabChange,
        setSearchQuery,
        sendMessage,
        toggleMembersInfo
    } = useChatActions();

    const handleBackClick = useCallback(() => {
        selectChat(null);
    }, [selectChat]);

    const handleSendMessage = useCallback((text) => {
        if (selectedChat) {
            sendMessage(text);
        }
    }, [selectedChat, sendMessage]);

    return (
        <Box sx={{
            height: 'calc(100vh - 136px)', // висота екрану мінус висота AppBar та BottomNavigation
            display: 'flex',
            width: '100%',
            bgcolor: '#eceaf3'
        }}>
```

```
        <ChatList
          chats={filteredChats}
          selectedChat={selectedChat}
          activeTab={activeTab}
          searchQuery={searchQuery}
          onChatSelect={selectChat}
          onTabChange={handleTabChange}
          onSearchChange={setSearchQuery}
          hideOnMobile={isSmallScreen && selectedChat}
        />


        <ChatWindow
          chat={selectedChat}
          isLoading={isMessagesLoading}
          isMobile={isSmallScreen}
          onSendMessage={handleSendMessage}
          onBackClick={handleBackClick}
          onGroupInfoClick={toggleMembersInfo}
          showGroupInfo={showMembersInfo}
          hideOnMobile={isSmallScreen && !selectedChat}
        />

        {/* Інформація про учасників групи (для групових чатів) */}
        <MembersInfo
          members={selectedChat?.type === 'group' ? selectedChat.members : []}
          show={showMembersInfo && selectedChat?.type === 'group'}
          onClose={toggleMembersInfo}
        />
      </Box>
    );
};

export default Chat;

import React from 'react';
import {
  AppBar,
  Toolbar,
  IconButton,
  Typography,
  Box,
  Menu,
  MenuItem,
  Avatar
} from '@mui/material';
import {
  ArrowBack,
  Phone,
  Videocam,
```

```
  Group,
  MoreVert
} from '@mui/icons-material';

/**
 * Компонент заголовка чату з інформацією та кнопками керування
 *
 * @param {Object} props - Властивості компонента
 * @param {Object} props.chat - Дані поточного чату
 * @param {boolean} props.isMobile - Чи мобільний режим відображення
 * @param {function} props.onBackClick - Обробник натискання кнопки "Назад"
 * @param {function} props.onGroupInfoClick - Обробник натискання кнопки інформації про групу
 * @param {boolean} props.showGroupInfo - Чи відображається інформація про групу
 * @returns {JSX.Element} React компонент
 */
const ChatHeader = ({
                      chat,
                      isMobile,
                      onBackClick,
                      onGroupInfoClick,
                      showGroupInfo
                    }) => {
  const [anchorEl, setAnchorEl] = React.useState(null);

  const handleMenuOpen = (event) => {
    setAnchorEl(event.currentTarget);
  };

  const handleMenuClose = () => {
    setAnchorEl(null);
  };

  if (!chat) return null;

  return (
    <AppBar
      position="static"
      color="default"
      elevation={0}
      sx={{
        borderBottom: '1px solid',
        borderColor: 'divider',
        bgcolor: 'background.paper'
      }}
    >
      <Toolbar>
        {/* Кнопка "Назад" на мобільних пристроях */}
        {isMobile && (
          <IconButton edge="start" onClick={onBackClick} sx={{ mr: 1 }}>
            <ArrowBack />
          </IconButton>
```

```
)}


<Avatar
  sx={{
    bgcolor: chat.type === 'personal' ? '#a89eff' : '#c1a9c0',
    mr: 2
  }}
>
  {chat.avatar}
</Avatar>

{/* Інформація про чат */}
<Box sx={{ flexGrow: 1 }}>
  <Typography variant="subtitle1" fontWeight="bold">
    {chat.name}
  </Typography>
  <Typography variant="caption" color="text.secondary">
    {chat.type === 'personal'
      ? (chat.status === 'online' ? 'Онлайн' : 'Офлайн')
      : `${chat.memberCount} учасників`
    }
  </Typography>
</Box>

<IconButton>
  <Phone />
</IconButton>
<IconButton>
  <Videocam />
</IconButton>

{chat.type === 'group' && (
  <IconButton
    onClick={onGroupInfoClick}
    color={showGroupInfo ? 'primary' : 'default'}
  >
    <Group />
  </IconButton>
)}

{/* Меню додаткових опцій */}
<IconButton onClick={handleMenuOpen}>
  <MoreVert />
</IconButton>

<Menu
  anchorEl={anchorEl}
  open={Boolean(anchorEl)}
  onClose={handleMenuClose}
>
```

```
                <MenuItem onClick={handleMenuClose}>Інформація</MenuItem>
                <MenuItem onClick={handleMenuClose}>Знайти в чаті</MenuItem>
                <MenuItem onClick={handleMenuClose}>Сповіщення</MenuItem>
                {chat.type === 'group' && (
                    <MenuItem onClick={handleMenuClose}>Вийти з групи</MenuItem>
                )}
            </Menu>
        </Toolbar>
    </AppBar>
  );
};

export default ChatHeader;
import React, { useState } from 'react';
import {
  Box,
  TextField,
  IconButton
} from '@mui/material';
import {
  Send,
  InsertEmoticon,
  AttachFile,
  Mic
} from '@mui/icons-material';

/**
 * Компонент для введення та відправки повідомлень
 *
 * @param {Object} props - Властивості компонента
 * @param {function} props.onSendMessage - Обробник відправки повідомлення
 * @returns {JSX.Element} React компонент
 */
const ChatInput = ({ onSendMessage }) => {
  const [message, setMessage] = useState('');


  const handleSend = () => {
    if (!message.trim()) return;
    onSendMessage(message);
    setMessage('');
  };

  const handleKeyPress = (e) => {
    if (e.key === 'Enter' && !e.shiftKey) {
      e.preventDefault();
      handleSend();
    }
  };

  return (
```

```
<Box
  sx={{
    p: 2,
    bgcolor: 'background.paper',
    borderTop: '1px solid',
    borderColor: 'divider'
  }}
>
  <Box
    component="form"
    sx={{
      display: 'flex',
      alignItems: 'flex-end'
    }}
    onSubmit={(e) => {
      e.preventDefault();
      handleSend();
    }}
  >

    <IconButton size="large">
      <InsertEmoticon />
    </IconButton>


    <IconButton size="large">
      <AttachFile />
    </IconButton>


    <TextField
      fullWidth
      multiline
      placeholder="Напишіть повідомлення..."
      variant="outlined"
      maxRows={4}
      value={message}
      onChange={(e) => setMessage(e.target.value)}
      onKeyPress={handleKeyPress}
      sx={{
        ml: 1,
        mr: 1,
        '& .MuiOutlinedInput-root': {
          borderRadius: 3
        }
      }}
    />


    {message.trim() ? (
      <IconButton
```

```
                color="primary"
                size="large"
                onClick={handleSend}
              >
                <Send />
              </IconButton>
            ) : (
              <IconButton size="large">
                <Mic />
              </IconButton>
            )}
          </Box>
        </Box>
    );
};

export default ChatInput;

import React from 'react';
import {
    ListItem,
    ListItemAvatar,
    ListItemText,
    Typography,
    Badge,
    Box
} from '@mui/material';
import StatusBadge from '../ui/StatusBadge';
import { formatLastMessage } from '../../utils/chatHelpers';

/**
 * Компонент для відображення превью чату в списку чатів
 *
 * @param {Object} props - Властивості компонента
 * @param {Object} props.chat - Дані чату
 * @param {boolean} props.isSelected - Чи вибраний даний чат
 * @param {function} props.onClick - Обробник кліку на елемент
 * @returns {JSX.Element} React компонент
 */
const ChatItemPreview = ({ chat, isSelected, onClick }) => {
    return (
        <ListItem
            button
            onClick={() => onClick(chat)}
            sx={{
                borderBottom: '1px solid',
                borderColor: 'divider',
                bgcolor: isSelected ? 'action.selected' : 'transparent',
                '&:hover': {
                    bgcolor: 'action.hover',
                },
```

```
        p: 2
      }}
    >
      {/* Аватар з індикатором статусу */}
      <ListItemAvatar>
        <StatusBadge
          type={chat.type}
          status={chat.status}
          avatar={chat.avatar}
          sx={{ width: 50, height: 50 }}
        />
      </ListItemAvatar>

      {/* Основна інформація */}
      <ListItemText
        primary={
          <Typography
            variant="subtitle1"
            fontWeight={chat.unread > 0 ? 'bold' : 'normal'}
            noWrap
          >
            {chat.name}
          </Typography>
        }
        secondary={
          <Typography
            variant="body2"
            color="text.secondary"
            fontWeight={chat.unread > 0 ? 'medium' : 'normal'}
            noWrap
          >
            {formatLastMessage(chat)}
          </Typography>
        }
        sx={{ ml: 1 }}
      />

      {/* Час та кількість непрочитаних */}
      <Box sx={{ display: 'flex', flexDirection: 'column', alignItems: 'flex-end', ml: 1 }}>
        <Typography
          variant="caption"
          color={chat.unread > 0 ? 'primary' : 'text.secondary'}
        >
          {chat.time}
        </Typography>

        {chat.unread > 0 && (
          <Badge
            badgeContent={chat.unread}
            color="primary"
            sx={{ mt: 0.5 }}
```

```
                />
              )}
            </Box>
        </ListItem>
    );
};

export default ChatItemPreview;

import React from 'react';
import {
    Box,
    Typography,
    TextField,
    InputAdornment,
    Tabs,
    Tab,
    List
} from '@mui/material';
import { Search, Notifications, Person, Group } from '@mui/icons-material';
import ChatItemPreview from './ChatItemPreview';

/**
 * Компонент для відображення списку чатів з можливістю фільтрації і пошуку
 *
 * @param {Object} props - Властивості компонента
 * @param {Array} props.chats - Масив чатів для відображення
 * @param {Object} props.selectedChat - Поточний вибраний чат
 * @param {string} props.activeTab - Активна вкладка (all/personal/group)
 * @param {string} props.searchQuery - Текст пошуку
 * @param {function} props.onChatSelect - Обробник вибору чату
 * @param {function} props.onTabChange - Обробник зміни вкладки
 * @param {function} props.onSearchChange - Обробник зміни пошукового запиту
 * @param {boolean} props.hideOnMobile - Приховувати на мобільних при відкритому чаті
 * @returns {JSX.Element} React компонент
 */
const ChatList = ({
                      chats,
                      selectedChat,
                      activeTab,
                      searchQuery,
                      onChatSelect,
                      onTabChange,
                      onSearchChange,
                      hideOnMobile
                  }) => {
    return (
        <Box
            sx={{
                display: hideOnMobile ? 'none' : 'flex',
                flexDirection: 'column',
```

```
        width: { xs: '100%', md: '30%', lg: '25%' },
        height: '100%',
        borderRight: '1px solid',
        borderColor: 'divider',
        bgcolor: 'background.paper'
      }}
    >
      {/* Заголовок і поле пошуку */}
      <Box sx={{ p: 2, borderBottom: '1px solid', borderColor: 'divider' }}>
        <Typography variant="h6" fontWeight="bold" sx={{ mb: 1 }}>
          Повідомлення
        </Typography>

        <TextField
          placeholder="Пошук чатів..."
          variant="outlined"
          fullWidth
          size="small"
          value={searchQuery}
          onChange={(e) => onSearchChange(e.target.value)}
          InputProps={{
            startAdornment: (
              <InputAdornment position="start">
                <Search color="action" />
              </InputAdornment>
            ),
          }}
          sx={{ mb: 1 }}
        />

        <Tabs
          value={activeTab}
          onChange={(e, newValue) => onTabChange(newValue)}
          variant="fullWidth"
          sx={{ mb: 1 }}
        >
          <Tab
            label="Всі"
            value="all"
            icon={<Notifications />}
            iconPosition="start"
            sx={{ minWidth: 0 }}
          />
          <Tab
            label="Особисті"
            value="personal"
            icon={<Person />}
            iconPosition="start"
            sx={{ minWidth: 0 }}
          />
          <Tab
```

```
                label="Групові"
                value="group"
                icon={<Group />}
                iconPosition="start"
                sx={{ minWidth: 0 }}
              />
            </Tabs>
          </Box>

          {/* Список чатів */}
          <List sx={{ flex: 1, overflow: 'auto', p: 0 }}>
            {chats.length > 0 ? (
              chats.map((chat) => (
                <ChatItemPreview
                  key={chat.id}
                  chat={chat}
                  isSelected={selectedChat && selectedChat.id === chat.id}
                  onClick={onChatSelect}
                />
              ))
            ) : (
              <Box sx={{ display: 'flex', justifyContent: 'center', p: 3 }}>
                <Typography color="text.secondary">Чати не знайдені</Typography>
              </Box>
            )}
          </List>
        </Box>
    );
};

export default ChatList;

import React from 'react';
import { Box, Typography } from '@mui/material';
import MessageStatus from '../ui/MessageStatus';

/**
 * Компонент для відображення одного повідомлення в чаті
 *
 * @param {Object} props - Властивості компонента
 * @param {Object} props.message - Дані повідомлення
 * @param {string} props.message.id - Унікальний ID повідомлення
 * @param {string} props.message.text - Текст повідомлення
 * @param {string} props.message.time - Час відправки
 * @param {string} props.message.sender - Ім'я відправника
 * @param {string} props.message.senderId - ID відправника
 * @param {string} props.message.status - Статус повідомлення
 * @param {boolean} props.isGroupChat - Чи відображається повідомлення в груповому чаті
 * @returns {JSX.Element} React компонент
 */
const ChatMessage = ({ message, isGroupChat }) => {
```

```
const isOwnMessage = message.senderId === 'currentUser';

return (
  <Box
    sx={{
      display: 'flex',
      flexDirection: 'column',
      alignSelf: isOwnMessage ? 'flex-end' : 'flex-start',
      maxWidth: '70%',
      mb: 1.5
    }}
  >
    {/* Відображаємо ім'я відправника для групового чату (крім власних повідомлень) */}
    {!isOwnMessage && isGroupChat && (
      <Typography
        variant="caption"
        color="text.secondary"
        sx={{ ml: 1, mb: 0.5 }}
      >
        {message.sender}
      </Typography>
    )}

    {/* Бульбашка повідомлення */}
    <Box
      sx={{
        bgcolor: isOwnMessage ? 'primary.main' : 'background.paper',
        color: isOwnMessage ? 'white' : 'text.primary',
        borderRadius: 2,
        p: 1.5,
        boxShadow: 1,
        position: 'relative'
      }}
    >
      {/* Текст повідомлення */}
      <Typography variant="body1">{message.text}</Typography>

      {/* Час та статус повідомлення */}
      <Box
        sx={{
          display: 'flex',
          justifyContent: 'flex-end',
          alignItems: 'center',
          mt: 0.5
        }}
      >
        <Typography
          variant="caption"
          color={isOwnMessage ? 'rgba(255,255,255,0.7)' : 'text.secondary'}
          sx={{ mr: 0.5 }}
        >
```

```
                {message.time}
              </Typography>

              {/* Відображаємо статус лише для власних повідомлень */}
              {isOwnMessage && <MessageStatus status={message.status} />}
            </Box>
          </Box>
        </Box>
    );
};

export default ChatMessage;

import React, { useRef, useEffect } from 'react';
import { Box, Typography, CircularProgress } from '@mui/material';
import { ChatBubble } from '@mui/icons-material';
import ChatHeader from './ChatHeader';
import ChatInput from './ChatInput';
import ChatMessage from './ChatMessage';

/**
 * Компонент вікна чату з повідомленнями
 *
 * @param {Object} props - Властивості компонента
 * @param {Object} props.chat - Дані поточного чату
 * @param {boolean} props.isLoading - Чи відбувається завантаження
 * @param {boolean} props.isMobile - Чи мобільний режим відображення
 * @param {function} props.onSendMessage - Обробник відправки повідомлення
 * @param {function} props.onBackClick - Обробник натискання кнопки "Назад"
 * @param {function} props.onGroupInfoClick - Обробник натискання кнопки інформації про групу
 * @param {boolean} props.showGroupInfo - Чи відображається інформація про групу
 * @param {boolean} props.hideOnMobile - Приховувати на мобільних якщо чат не обраний
 * @returns {JSX.Element} React компонент
 */
const ChatWindow = ({
                      chat,
                      isLoading,
                      isMobile,
                      onSendMessage,
                      onBackClick,
                      onGroupInfoClick,
                      showGroupInfo,
                      hideOnMobile
                    }) => {
    const messageEndRef = useRef(null);
    useEffect(() => {
        if (messageEndRef.current) {
            messageEndRef.current.scrollIntoView({ behavior: 'smooth' });
        }
    }, [chat]);
```

```
if (!chat && !isMobile) {
  return (
    <Box
      sx={{
        display: 'flex',
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'center',
        bgcolor: 'background.default',
        p: 3
      }}
    >
      <ChatBubble sx={{ fontSize: 60, color: 'primary.main', opacity: 0.5, mb: 2 }} />
      <Typography variant="h6" color="text.secondary">
        Виберіть чат для початку спілкування
      </Typography>
    </Box>
  );
}

if (isLoading) {
  return (
    <Box
      sx={{
        display: 'flex',
        flex: 1,
        flexDirection: 'column',
        justifyContent: 'center',
        alignItems: 'center',
        bgcolor: 'background.default'
      }}
    >
      <CircularProgress />
      <Typography variant="body1" sx={{ mt: 2 }}>
        Завантаження повідомлень...
      </Typography>
    </Box>
  );
}

return (
  <Box
    sx={{
      display: (chat || !isMobile) ? 'flex' : (hideOnMobile ? 'none' : 'flex'),
      flex: 1,
      flexDirection: 'column',
      height: '100%',
      bgcolor: 'background.default'
    }}
  >
```

```
        {/* Шапка чату */}
        <ChatHeader
          chat={chat}
          isMobile={isMobile}
          onBackClick={onBackClick}
          onGroupInfoClick={onGroupInfoClick}
          showGroupInfo={showGroupInfo}
        />

        {/* Повідомлення */}
        <Box
          sx={{
            flex: 1,
            overflowY: 'auto',
            p: 2,
            display: 'flex',
            flexDirection: 'column',
            backgroundImage: 'linear-gradient(to bottom, rgba(236, 234, 243, 0.5), rgba(230, 228, 237, 0.8))',
            backgroundSize: 'cover'
          }}
        >
          {chat?.messages?.map((message) => (
            <ChatMessage
              key={message.id}
              message={message}
              isGroupChat={chat.type === 'group'}
            />
          ))}
          <div ref={messageEndRef} />
        </Box>

        {/* Поле введення повідомлення */}
        <ChatInput onSendMessage={onSendMessage} />
      </Box>
    );
};

export default ChatWindow;
import React from 'react';
import {
  Box,
  Typography,
  IconButton,
  List,
  ListItem,
  ListItemAvatar,
  ListItemText,
  Badge,
  Avatar
} from '@mui/material';
import { ArrowBack } from '@mui/icons-material';
```

```
/**
 * Компонент для відображення інформації про учасників групового чату
 *
 * @param {Object} props - Властивості компонента
 * @param {Array} props.members - Масив учасників групи
 * @param {boolean} props.show - Чи відображати компонент
 * @param {function} props.onClose - Обробник закриття панелі
 * @returns {JSX.Element} React компонент
 */
const MembersInfo = ({ members, show, onClose }) => {
  if (!show) return null;

  return (
    <Box
      sx={{
        display: show ? 'flex' : 'none',
        flexDirection: 'column',
        width: { xs: '100%', sm: '30%', md: '25%', lg: '20%' },
        borderLeft: '1px solid',
        borderColor: 'divider',
        p: 2,
        bgcolor: 'background.paper'
      }}
    >
      {/* Заголовок і кнопка "Назад" */}
      <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', mb: 2 }}>
        <Typography variant="h6">Учасники</Typography>
        <IconButton onClick={onClose} size="small">
          <ArrowBack />
        </IconButton>
      </Box>

      {/* Список учасників */}
      <List>
        {members.map((member) => (
          <ListItem key={member.id} sx={{ px: 0 }}>
            <ListItemAvatar>
              <Badge
                overlap="circular"
                anchorOrigin={{ vertical: 'bottom', horizontal: 'right' }}
                variant="dot"
                color={member.status === 'online' ? 'success' : 'default'}
              >
                <Avatar sx={{ bgcolor: member.id === 'currentUser' ? 'primary.main' : 'secondary.main' }}>
                  {member.avatar}
                </Avatar>
              </Badge>
            </ListItemAvatar>
            <ListItemText
              primary={member.name}
```

```
                        secondary={member.status === 'online' ? 'Онлайн' : 'Офлайн'}
                    />
                </ListItem>
            ))}
        </List>
    </Box>
  );
};

export default MembersInfo;
import React from 'react';
import {
  Box,
  Typography,
  Avatar,
  Button,
  Divider,
  Stack,
  Paper,
  Tabs,
  Tab,
  Container
} from '@mui/material';
import {
  LocationOn,
  CalendarMonth,
  Instagram,
  Facebook,
  Twitter,
  Verified
} from '@mui/icons-material';
import PostCard from '../feed/PostCard';
import { userProfileData, userPostsData } from '../../data/profileData.js';

const ProfileContent = ({ isSmallScreen }) => {
  const [tabValue, setTabValue] = React.useState(0);

  const handleTabChange = (event, newValue) => {
    setTabValue(newValue);
  };

  return (
    <Container maxWidth="md">
      <Box
        sx={{
          position: 'relative',
          width: '100%',
          height: { xs: '180px', sm: '220px', md: '280px' },
          borderRadius: '16px 16px 0 0',
          overflow: 'hidden',
          mb: 7,
```

```
        mt: 2
      }}
    >
      <Box
        component="img"
        src={userProfileData.coverImage}
        alt="Cover"
        sx={{
          width: '100%',
          height: '100%',
          objectFit: 'cover'
        }}
      />

      {/* Profile Avatar - positioned to overlap the cover and content */}
      <Avatar
        src={userProfileData.avatar}
        alt={userProfileData.fullName}
        sx={{
          width: { xs: 90, sm: 120, md: 140 },
          height: { xs: 90, sm: 120, md: 140 },
          border: '4px solid #fff',
          position: 'absolute',
          bottom: { xs: -60, sm: -80, md: 0 },
          left: { xs: 24, sm: 32 },
          boxShadow: '0 4px 10px rgba(0,0,0,0.15)'
        }}
      />

      {/* Follow Button */}
      <Button
        variant="contained"
        sx={{
          position: 'absolute',
          bottom: { xs: -40, sm: -50 },
          right: { xs: 16, sm: 32 },
          bgcolor: '#a89eff',
          '&:hover': {
            bgcolor: '#8a7ad8',
          }
        }}
      >
        Follow
      </Button>
    </Box>

    {/* Profile Info */}
    <Paper
      elevation={0}
      sx={{
        bgcolor: '#ffffff',
```

```
        borderRadius: '0 0 16px 16px',
        p: { xs: 2, sm: 3 },
        mb: 3
      }}
    >
      <Box sx={{ pl: { xs: 0, sm: 2 }, pt: 2, pb: 1 }}>
        {/* Name and Verification */}
        <Box sx={{ display: 'flex', alignItems: 'center', mb: 0.5 }}>
          <Typography variant="h5" component="h1" sx={{ fontWeight: 'bold' }}>
            {userProfileData.fullName}
          </Typography>
          {userProfileData.verified && (
            <Verified sx={{ color: '#a89eff', ml: 1 }} />
          )}
        </Box>

        {/* Username */}
        <Typography variant="body1" color="text.secondary" sx={{ mb: 1 }}>
          @{userProfileData.username}
        </Typography>

        {/* Bio */}
        <Typography variant="body1" paragraph sx={{ mb: 2 }}>
          {userProfileData.bio}
        </Typography>

        {/* Location and Join Date */}
        <Stack direction="row" spacing={3} sx={{ mb: 2 }}>
          <Box sx={{ display: 'flex', alignItems: 'center' }}>
            <LocationOn sx={{ fontSize: 20, color: 'text.secondary', mr: 0.5 }} />
            <Typography variant="body2" color="text.secondary">
              {userProfileData.location}
            </Typography>
          </Box>
          <Box sx={{ display: 'flex', alignItems: 'center' }}>
            <CalendarMonth sx={{ fontSize: 20, color: 'text.secondary', mr: 0.5 }} />
            <Typography variant="body2" color="text.secondary">
              Joined {new Date(userProfileData.joinedDate).toLocaleDateString('en-US', { month: 'long', year: 'numeric' })}
            </Typography>
          </Box>
        </Stack>

        {/* Social Links */}
        <Stack direction="row" spacing={2} sx={{ mb: 2 }}>
          <Instagram sx={{ color: '#a89eff' }} />
          <Facebook sx={{ color: '#a89eff' }} />
          <Twitter sx={{ color: '#a89eff' }} />
        </Stack>

        {/* Followers and Following */}
        <Stack direction="row" spacing={3}>
```

```
            <Typography variant="body2">
              <strong>{userProfileData.following}</strong> Following
            </Typography>
            <Typography variant="body2">
              <strong>{userProfileData.followers}</strong> Followers
            </Typography>
          </Stack>
        </Box>
      </Paper>

      {/* Tabs for Posts, Saved, etc. */}
      <Paper sx={{ borderRadius: '16px', mb: 3 }}>
        <Tabs
          value={tabValue}
          onChange={handleTabChange}
          variant="fullWidth"
          sx={{
            '& .Mui-selected': { color: '#a89eff' },
            '& .MuiTabs-indicator': { backgroundColor: '#a89eff' }
          }}
        >
          <Tab label="Posts" />
        </Tabs>
      </Paper>

      {/* Post Content */}
      {tabValue === 0 && (
        <Box>
          {userPostsData.map(post => (
            <PostCard
              key={post.id}
              post={post}
              isSmallScreen={isSmallScreen}
            />
          ))}
        </Box>
      )}

      {/* Saved Content - Placeholder */}
      {tabValue === 1 && (
        <Paper sx={{ p: 4, borderRadius: '16px', textAlign: 'center' }}>
          <Typography variant="h6" color="text.secondary">
            Saved trips will appear here
          </Typography>
        </Paper>
      )}

      {/* Media Content - Placeholder */}
      {tabValue === 2 && (
        <Paper sx={{ p: 4, borderRadius: '16px', textAlign: 'center' }}>
          <Typography variant="h6" color="text.secondary">
```

```
                Media gallery will appear here
              </Typography>
            </Paper>
          )}
        </Container>
    );
};

const SearchContent = () => {
    const [searchQuery, setSearchQuery] = useState('');
    const [searchResults, setSearchResults] = useState([]);
    const [tabValue, setTabValue] = useState(0);
    const [loading, setLoading] = useState(false);
    const [searchType, setSearchType] = useState('all');
    const [filteredResults, setFilteredResults] = useState([]);

    const allPosts = [...posts, ...userPostsData];

    const users = […users];

    const locations = [...new Set(allPosts.map(post => post.location))].map(location => {
        const matchingPost = allPosts.find(post => post.location === location);
        return {
            location,
            imageUrl: matchingPost.images[0],
            type: 'location'
        };
    });

    const activities = [];
    allPosts.forEach(post => {
        if (post.days) {
            post.days.forEach(day => {
                if (day.places) {
                    day.places.forEach(place => {
                        activities.push({
                            name: place.name,
                            location: post.location,
                            type: 'activity',
                            activityType: place.type,
                            rating: place.rating,
                            imageUrl: place.imageUrl,
                            tips: place.tips
                        });
                    });
                }
                if (day.concert) {
                    activities.push({
                        name: day.concert.name || day.concert.artist,
                        location: post.location,
                        type: 'activity',
```

```
                    activityType: 'concert',
                    venue: day.concert.venue,
                    imageUrl: day.concert.imageUrl,
                    time: day.concert.time,
                    date: day.date
                  });
                }
              });
            }
          });

          const hotels = allPosts
            .filter(post => post.hotel)
            .map(post => ({
              ...post.hotel,
              location: post.location,
              type: 'hotel'
            }));

          const performSearch = (query) => {
            if (!query.trim()) {
              setSearchResults([]);
              setFilteredResults([]);
              return;
            }

            setLoading(true);

            const normalizedQuery = query.toLowerCase();

            const matchedPosts = allPosts.filter(post =>
              post.title.toLowerCase().includes(normalizedQuery) ||
              post.location.toLowerCase().includes(normalizedQuery) ||
              post.description.toLowerCase().includes(normalizedQuery)
            ).map(post => ({...post, type: 'post'}));

            const matchedUsers = users.filter(user =>
              user.name.toLowerCase().includes(normalizedQuery) ||
              user.username.toLowerCase().includes(normalizedQuery) ||
              user.location.toLowerCase().includes(normalizedQuery)
            );

export default ProfileContent;
```

Додаток Г – Ілюстративна частина

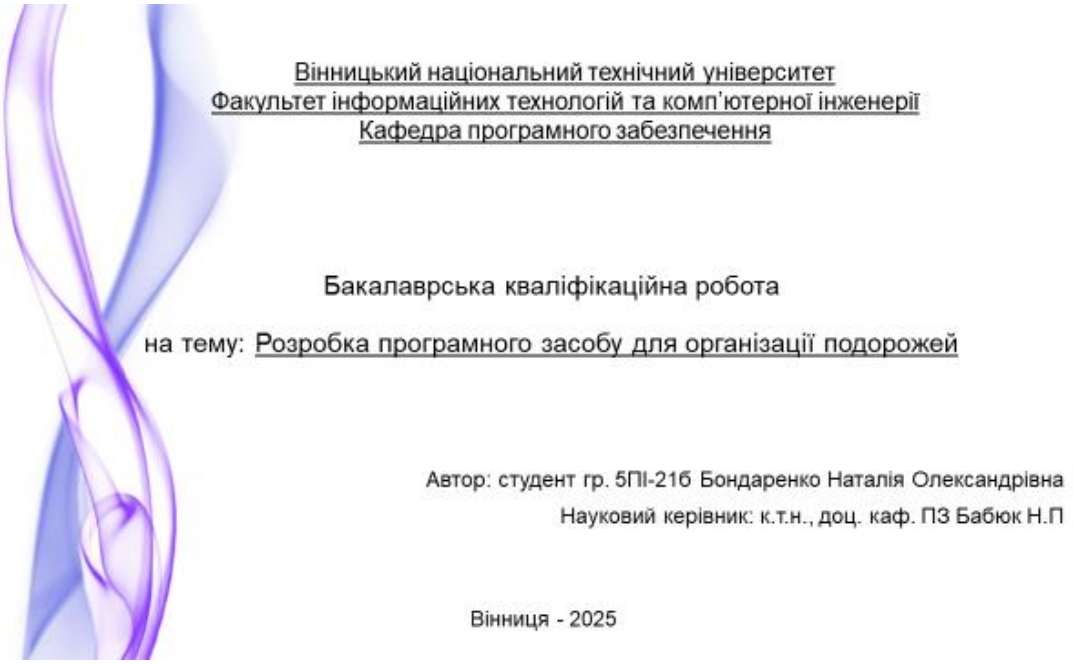

Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Актуальність

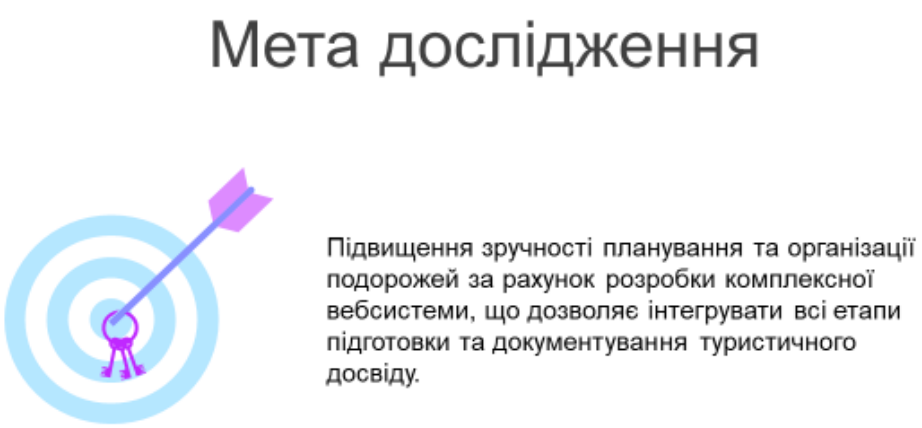

Рисунок Г.3 – Мета дослідження

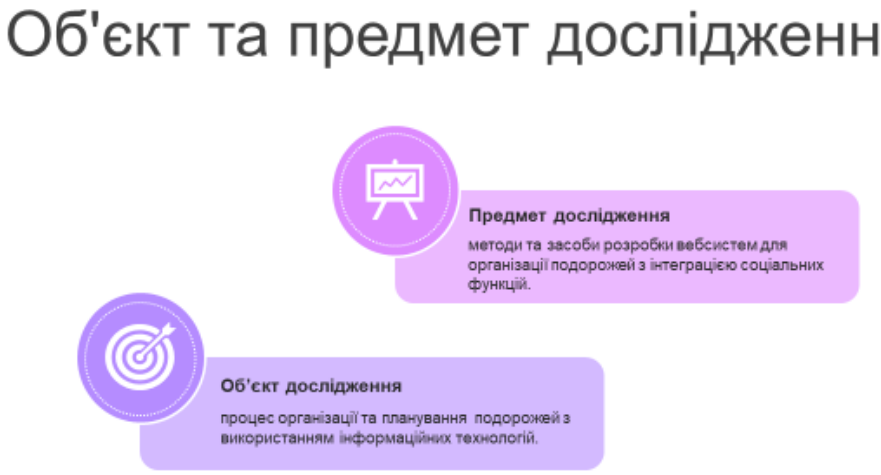

Рисунок Г.4 – Об’єкт та предмет дослідження

Рисунок Г.5 – Новизна отриманих результатів

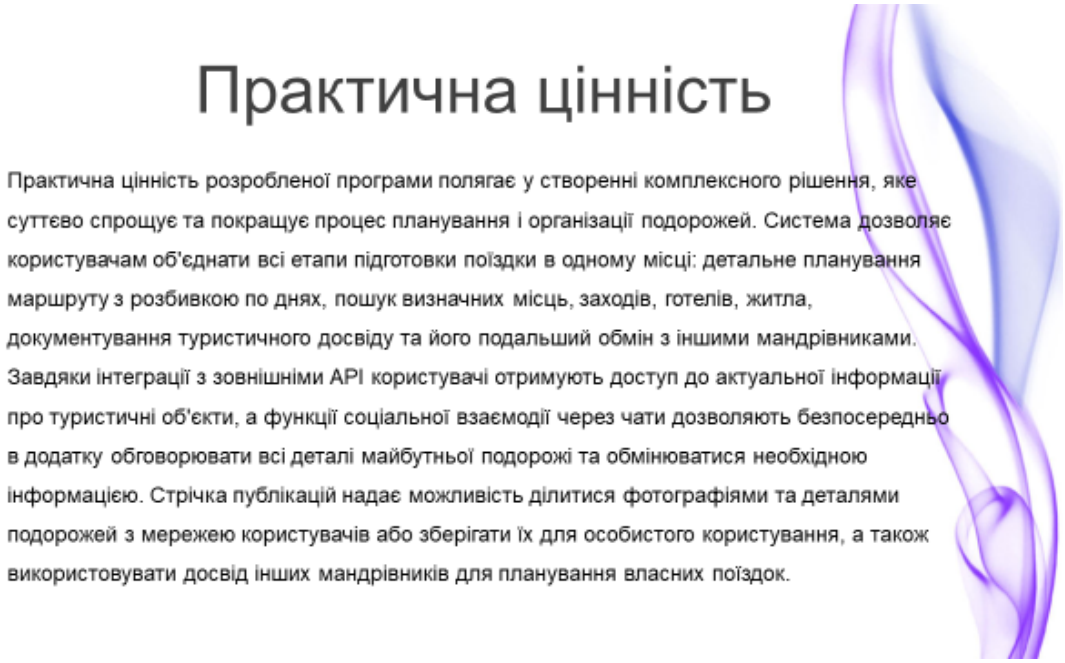

Рисунок Г.6 – Практична цінність

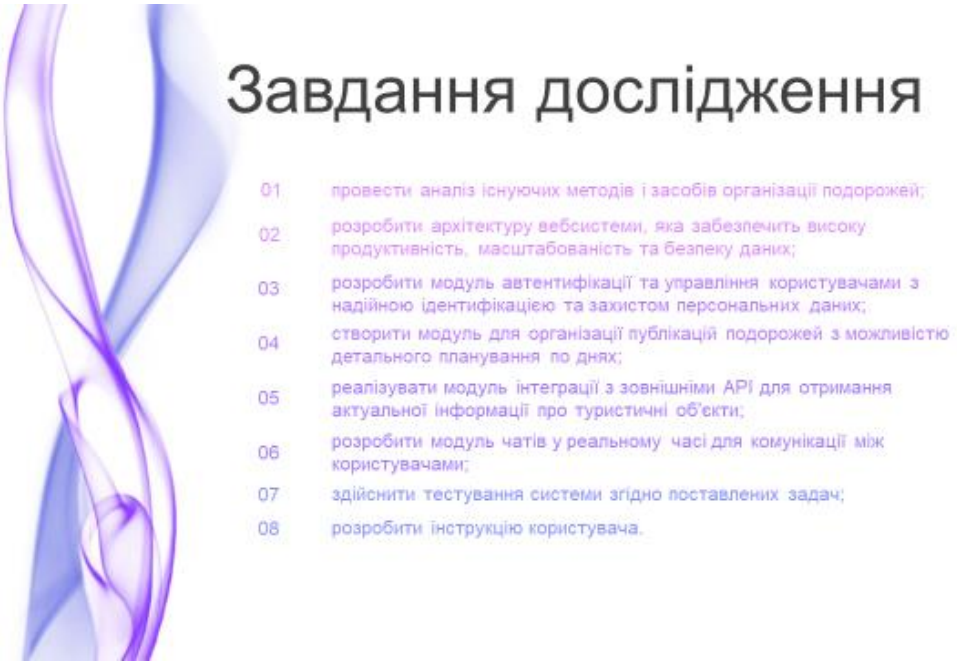

Рисунок Г.7 – Завдання дослідження

Порівняльна характеристика аналогів

Критерії	Skyscanner	Rome2Rio	TripAdvisor	Maps.me	Airbnb	Власна розробка
Соціальна взаємодія (можливість створювати профілі, публікувати контент та взаємодіяти з іншими користувачам)	-	-	+	-	+	+
Стрічка публікацій	-	-	-	-	-	+
Деталізоване планування (можливість створювати подорож з розбивкою на дні з детальним описом)	-	+	-	-	-	+
Маршрутизація на мапі	-	+	+	+	-	+
Пошук житла	+	+	+	+	+	+
Пошук транспорту	+	+	+	+	-	+
Персональний та груповий чати	-	-	-	-	-	+
Загальна оцінка	28.5%	57.1%	57,1%	42.8%	28.5%	100%

Рисунок Г.8 – Порівняльна характеристика аналогів

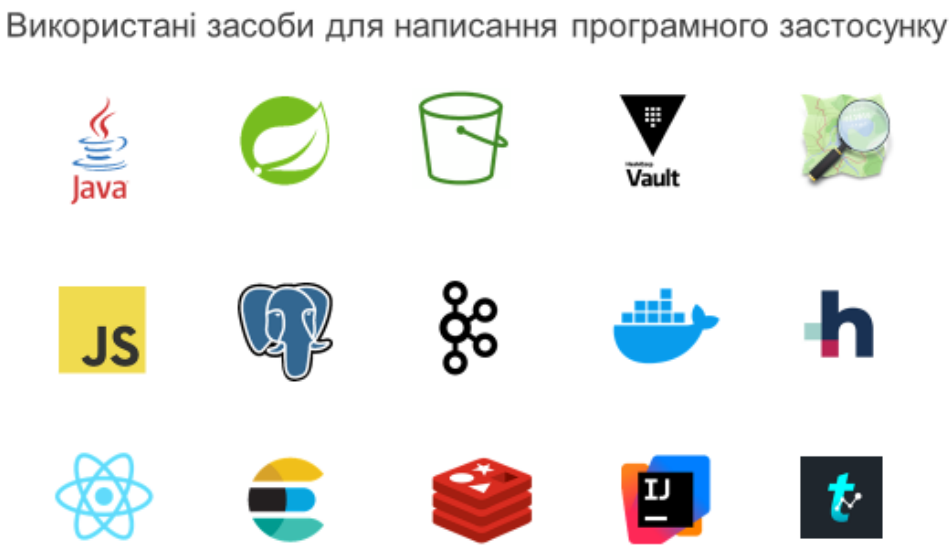

Рисунок Г.9 – Використані засоби для написання програмного застосунку

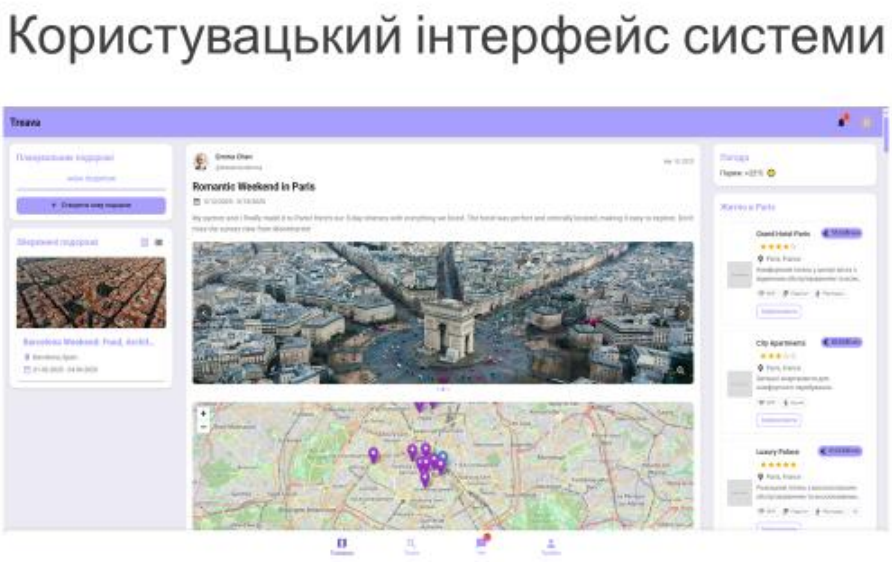

Рисунок Г.10 – Користувацький інтерфейс системи

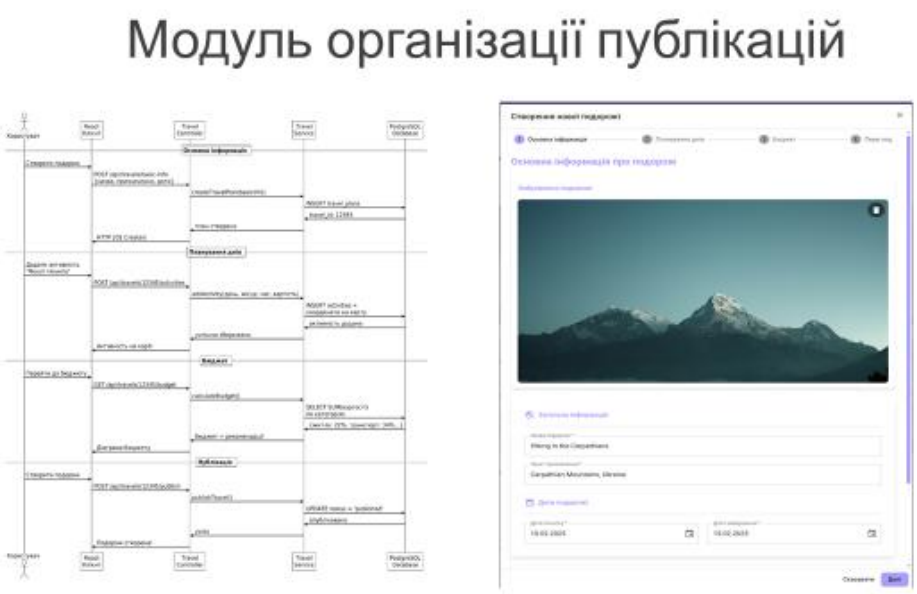

Рисунок Г.11 – Модуль організації публікацій

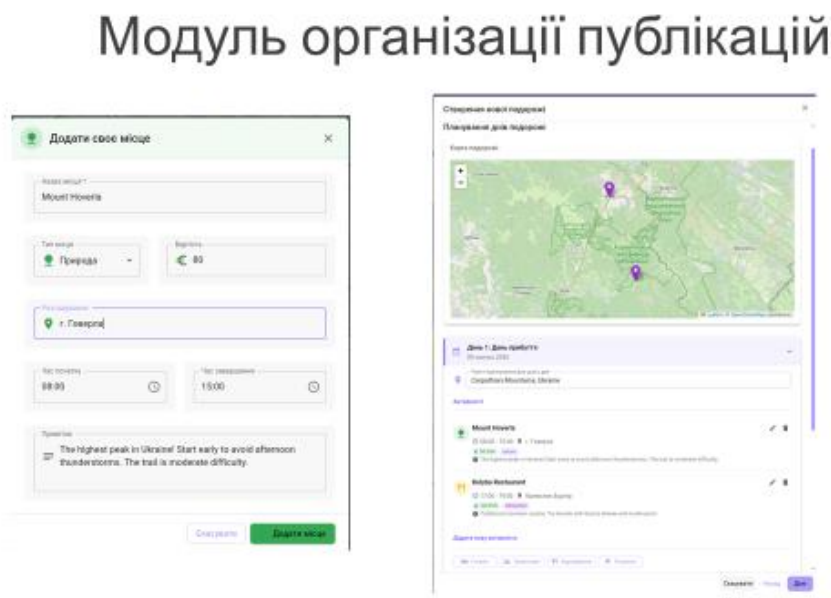

Рисунок Г.12 – Модуль організації публікацій (продовження)

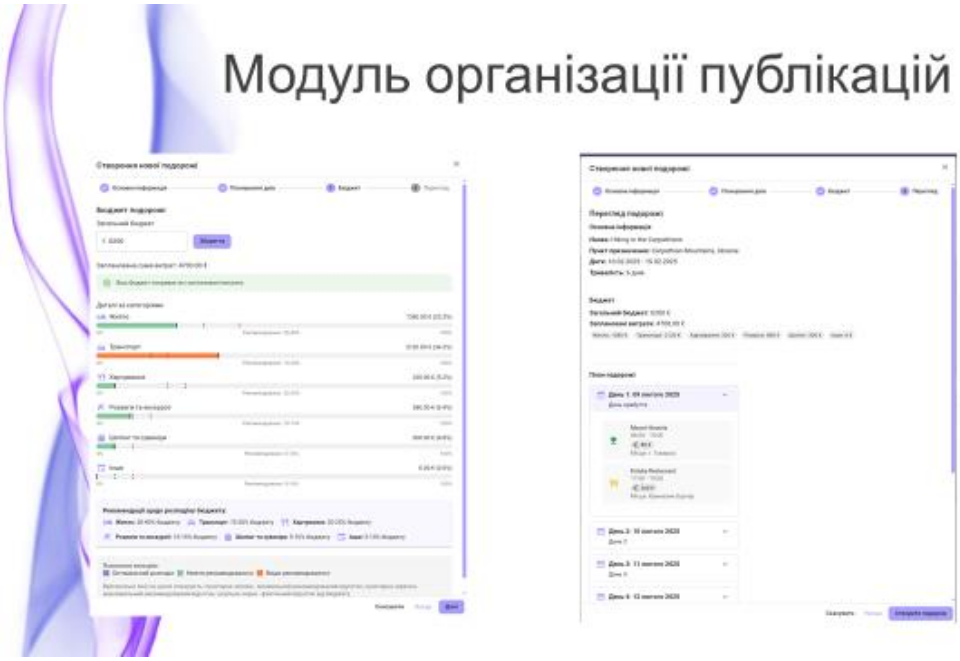

Рисунок Г.13 – Модуль організації публікацій (продовження)

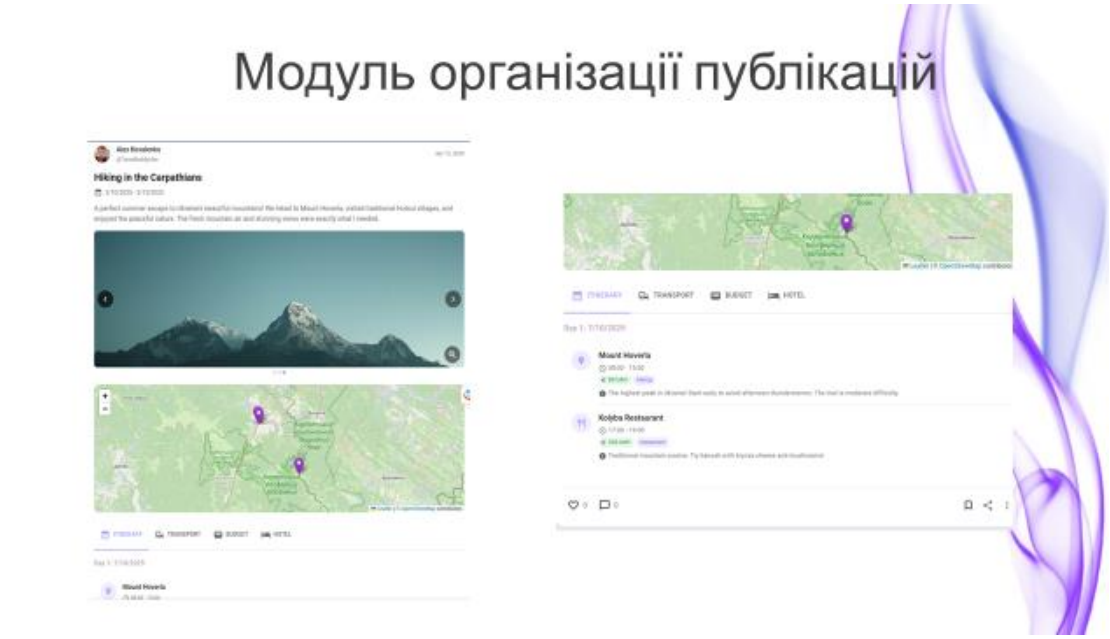

Рисунок Г.14 – Модуль організації публікацій (продовження)

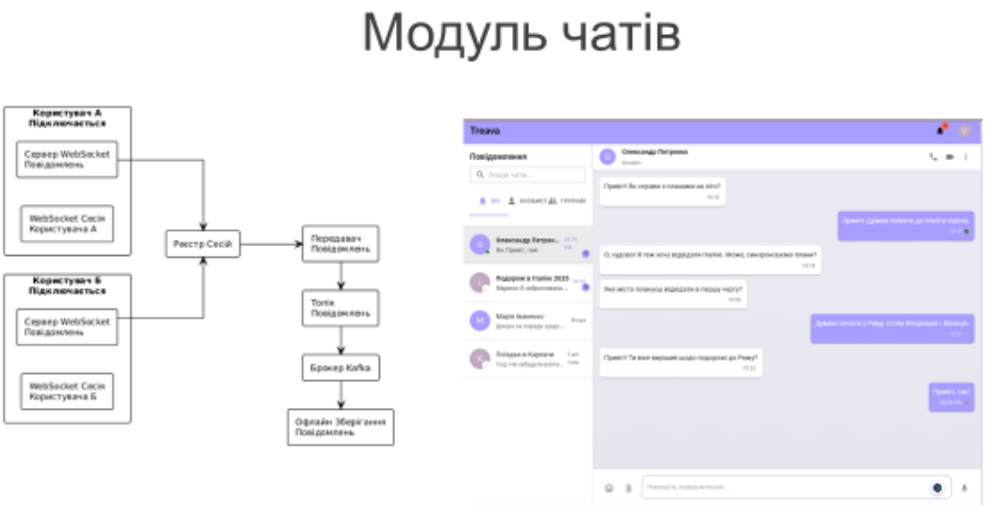

Рисунок Г.15 – Модуль чатів

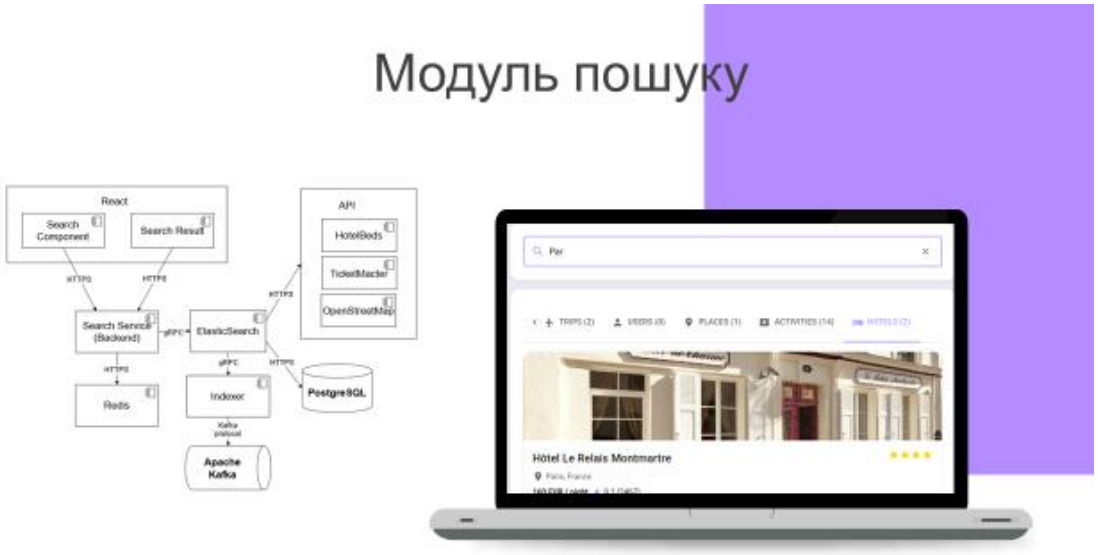

Рисунок Г.16 – Модуль пошуку

Рисунок Г.17 – Апробація

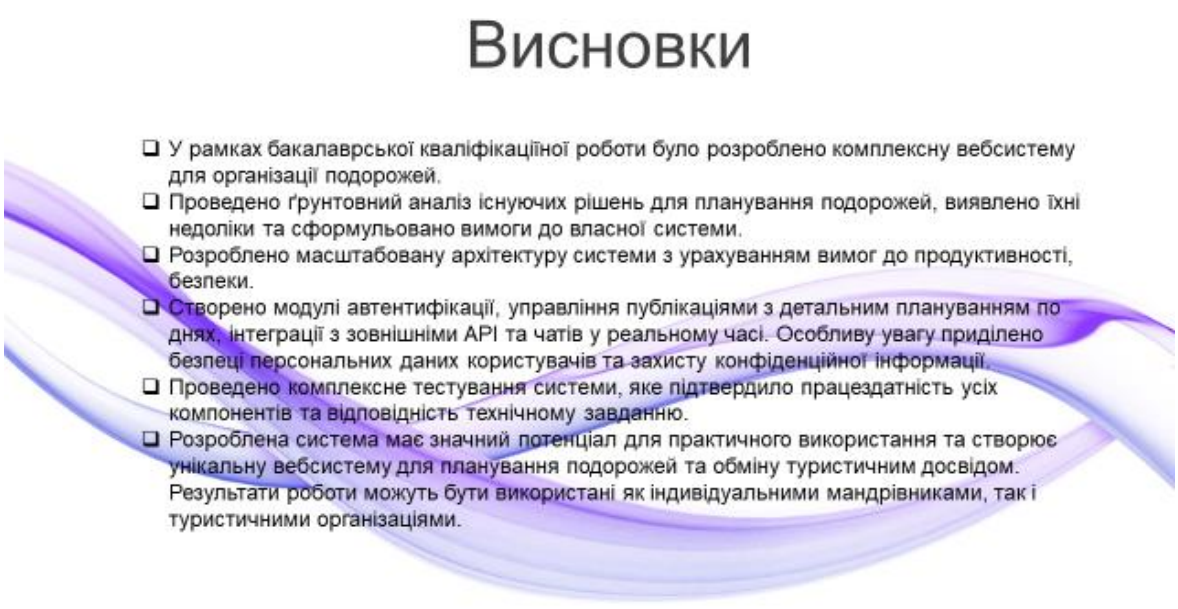

Рисунок Г.18 – Висновки