Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка автоматизованої системи управління бронюванням авіаквитків»

Виконав: студент IV курсу
групи ЗПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Туренко В. Р.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Романюк О. В.
(прізвище та ініціали)

Рецензент:. к.т.н., доц. каф. ЗІ Войтович О.П.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ
д.т.н., проф. О. Н. Романюк
«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 СТУДЕНТУ

Туренку Віталію Романовичу

1. Тема роботи – розробка автоматизованої системи управління бронюванням авіаквитків.

Керівник роботи: Романюк Оксана Володимирівна, к.т.н., доц. 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 програмного забезпечення – вебзастосунок; тип архітектури – клієнт-серверна; система керування базами даних – PostgreSQL; середовище розробки – IntelliJ IDEA, WebStorm; мови програмування – Java, TypeScript; фреймворки та бібліотеки – Spring Boot, Angular; використані API – Google Flights API, Google Geocoding API; метод обчислення відстаней – формула гаверсинуса; вхідні дані – запити користувача, географічні координати, типи транспорту, JSON-файл аеропортів; вихідні дані – згенеровані мультимодальні маршрути, оцінка вуглецевого сліду, екологічний рейтинг акаунту, персоналізовані рекомендації; архітектурний підхід – Clean Architecture; операційна система – Windows 10/11.

4. Зміст текстової частини: вступ, обгрунтування вибору методу розробки та постановка задач дослідження, розробка методів та алгоритмів роботи програмного продукту, розробка програмних модулів реалізації системи управління бронюванням авіаквитків, тестування програми, висновки, список використаних джерел, додатки, ілюстративна частина.

5. Перелік ілюстративного матеріалу: титульний слайд – автор, науковий керівник бакалаврської кваліфікаційної роботи, тема; актуальність розробки; мета, об'єкт та предмет дослідження; постановка задачі; наукова новизна; практична

цінність; огляд та аналіз аналогів; архітектура системи, діаграма варіантів використання, загальний алгоритм роботи системи; метод пошуку мультимодальних маршрутів; метод оцінки екологічного рейтингу акаунту; використані технології; функціонал розробленої системи; апробація та публікація результатів роботи; фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Романюк О. В. к.т.н., доцент кафедри ПЗ	25.03.2025	30.05.2025

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку систем бронювання авіаквитків та постановка задач розробки	25.03.25 – 31.03.25	виконано
2	Розробка методу побудови мультимодального маршруту з урахуванням вуглецевого сліду	01.04.25 – 07.04.25	виконано
3	Розробка методу оцінки екологічного рейтингу акаунту користувача	08.04.25 – 14.04.25	виконано
4	Розробка програмного забезпечення	15.04.25 – 12.05.25	виконано
5	Тестування програми	13.05.25 – 15.05.25	виконано
6	Оформлення матеріалів до захисту БКР	16.05.25 – 30.05.25	виконано

Студент _____ **Туренко В. Р.**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **Романюк О. В.**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

УДК 004.42

Туренко В. Р. Розробка автоматизованої системи управління бронюванням авіаквитків: бакалаврська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – Інженерія програмного забезпечення. – Вінниця: ВНТУ, 2025. – 104 с.

На укр. мові. Бібліогр.: 22 назв; рис.: 17; табл.: 3.

У бакалаврській кваліфікаційній роботі проаналізовано сучасний стан систем цифрового бронювання авіаквитків і тенденції до мультимодальності, персоналізації та врахування екологічних чинників. Обґрунтовано потребу у створенні вебзастосунку, який дозволяє здійснювати пошук маршрутів із урахуванням пересадок, альтернативного транспорту та викидів CO_2.

Розроблено метод побудови мультимодального маршруту на основі зовнішніх API, що включає етапи геокодування, пошуку рейсів і формування комбінованих варіантів. Запропоновано метод оцінки екологічного рейтингу акаунту, особливістю якого є поєднання об’єктивних параметрів поїздок та результатів опитування з використанням вагових коефіцієнтів і коригуючих множників. Розроблено модулі персоналізованих рекомендацій і візуалізації екорейтингу. Архітектура реалізована за принципами Clean Architecture з використанням Angular, Spring Boot і PostgreSQL.

Отримані результати можуть бути використані для підвищення екологічної поінформованості користувачів при плануванні подорожей та подальшого розвитку сервісів бронювання.

Ключові слова: мультимодальні маршрути, бронювання авіаквитків, екологічний слід, гейміфікація, Angular, Spring Boot, Typescript, Java.

ABSTRACT

UDC 004.42

Turenko V. R. Development of an automated airline ticket booking management system: bachelor's thesis in specialty 121 – Software Engineering, educational program – Software Engineering. – VNTU, 2025. – 104 p.

In Ukrainian speech. Bibliogr.: 22 titles; Figs.: 17; tables: 3.

The bachelor’s thesis analyzes the current state of digital airline booking systems and the growing trends toward multimodality, personalization, and environmental awareness. The need for a web-based application enabling route search with transfers, alternative transportation, and CO_2 emissions estimation is substantiated.

A method for constructing multimodal routes is implemented using external APIs, involving geocoding, flight search, and combined itinerary generation. A method for evaluating the ecological rating of a user account is proposed, distinguished by combining objective travel data with survey responses using weighted coefficients and adjustment factors. Modules for personalized recommendations and eco-rating visualization are developed. The software architecture follows Clean Architecture principles and is implemented with Angular, Spring Boot, and PostgreSQL.

The results may be applied to enhance user environmental awareness in travel planning and serve as a basis for future development of booking services.

Keywords: multimodal routes, airline ticket booking, carbon footprint, gamification, Angular, Spring Boot, TypeScript, Java.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Авіаперельоти є важливою складовою глобальної транспортної інфраструктури. Вони забезпечують швидке сполучення між містами та країнами, сприяючи міжнародній торгівлі, туризму, освіті та бізнесу. Щороку мільярди людей користуються послугами авіаперевізників для подорожей з особистих або професійних причин. Згідно з даними Міжнародної асоціації повітряного транспорту (IATA), лише у 2023 році послугами авіаційного транспорту скористалися понад 4,5 мільярда пасажирів [1].

Бронювання авіаквитків є процесом попереднього резервування місця на певному рейсі авіакомпанії. Воно дає змогу користувачеві гарантувати свою присутність на обраному маршруті та спланувати подорож заздалегідь. Сьогодні бронювання здійснюється здебільшого в цифровому середовищі – через сайти авіакомпаній, мобільні застосунки або онлайн-агрегатори. Це значно спростило доступ до послуг, однак водночас відкрило нові виклики.

До основних способів бронювання належать: пряме звернення до авіаперевізника (через вебсайт або застосунок), використання платформ-агрегаторів (таких як Skyscanner чи Google Flights), а також послуги туристичних агентств. У кожного способу є свої переваги, але спільною проблемою залишається обмежена підтримка мультимодальних маршрутів – коли подорож передбачає кілька етапів із використанням різних видів транспорту. Часто користувачі змушені самостійно поєднувати рейси, підбираючи стикування вручну.

Окрему увагу привертає питання екологічності авіаперевезень. За інформацією Всесвітньої туристичної організації (UNWTO), транспорт, зокрема авіаційний, є значним джерелом викидів парникових газів. Організація підкреслює важливість впровадження рішень, що сприяють зниженню вуглецевого сліду, та заохочує використання інструментів для інформування споживачів про вплив їхніх виборів на довкілля. Водночас меншість сервісів відображають інформацію про екологічність маршруту, і далеко не всі надають її у зручному для користувача вигляді.

Ще одним сучасним трендом є персоналізація користувацького досвіду. Системи бронювання, що враховують історію вибору, попередні бронювання, а також типові вподобання користувача, дозволяють запропонувати кращі варіанти маршрутів та підвищити ймовірність задоволення від послуги. Однак реалізація такої персоналізації потребує надійної архітектури, збирання аналітики та розробки відповідних рекомендаційних механізмів.

Також варто зазначити, що інтерфейси сучасних сервісів часто ускладнюють навігацію через велику кількість параметрів і відсутність інтуїтивного фільтрування. Це створює запит на розробку систем, що поєднують зручність, швидкодію та візуальну привабливість. Додатковим способом покращити досвід користувача може стати використання гейміфікації – впровадження елементів гри для стимулювання екологічного вибору або активної участі у сервісі.

Таким чином, тема розробки системи управління бронюванням авіаквитків є актуальною з технічного, соціального та екологічного погляду. Вона передбачає інтеграцію з API авіаперевізників, обробку великого обсягу даних, підтримку мультимодальних маршрутів, персоналізовані рекомендації та візуалізацію екологічного впливу. Створення такої системи дозволить зробити планування подорожей більш ефективним, інформативним та орієнтованим на потреби сучасного користувача.

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

Мета та завдання дослідження. Метою роботи є підвищення ефективності процесу пошуку та бронювання авіаквитків шляхом розробки нових методів та автоматизованої системи управління, яка підтримує мультимодальні маршрути та враховує екологічний слід кожного варіанту подорожі.

Відповідно до поставленої мети потрібно вирішити такі **завдання:**

– здійснити аналіз сучасних підходів до організації бронювання транспортних маршрутів;

– спроєктувати архітектуру інформаційної системи з урахуванням мультимодальних перевезень;

– розробити метод побудови мультимодального маршруту з урахуванням пересадок та вуглецевого сліду;

– розробити метод оцінки екологічного рейтингу акаунту користувача на основі історії замовлень і відповідей на опитування;

– реалізувати механізм персоналізованих рекомендацій на основі історії вибору користувача;

– реалізувати механізм візуалізації екологічного рейтингу акаунту;

– реалізувати модуль пошуку маршрутів із використанням зовнішніх API авіаперевізників;

– здійснити інтеграцію з API геокодування для автоматичного пошуку найближчих аеропортів;

– провести тестування застосунку та перевірку відповідності функціональним вимогам;

– розробити інструкцію користувача та описати системні вимоги до програмного забезпечення.

Об’єктом дослідження є процес автоматизованого бронювання авіаквитків з урахуванням мультимодальних маршрутів і екологічних характеристик.

Предметом дослідження є методи та засоби розробки програмного забезпечення веб-системи управління бронюванням авіаквитків.

Методи дослідження. У процесі досліджень використовувались такі методи: методи порівняльного аналізу для оцінки функціональних характеристик існуючих систем бронювання та формування вимог до розробки; теорія алгоритмів для розробки та побудови блок-схем алгоритмів для побудови маршрутів з урахуванням доступних рейсів і логіки пересадок; емпіричні методи дослідження для оцінювання вуглецевого сліду з орієнтацією на наближені значення; метод інтегральної оцінки для обчислення екологічного рейтингу користувача на основі історії замовлень і результатів опитування; методи тестування для перевірки працездатності окремих модулів системи та оцінки зручності її використання;

комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Уперше запропоновано метод побудови мультимодальних маршрутів, особливістю якого є включення оцінки вуглецевого сліду, що дало можливість підвищити гнучкість планування поїздок і точність оцінки їхнього екологічного впливу.

2. Уперше запропоновано метод оцінки екологічного рейтингу акаунту на основі аналізу історії замовлень та відповідей користувача на спеціалізоване опитування, особливістю якого є поєднання кількісних показників активності з якісними даними екологічних уподобань, що дало змогу обчислювати узагальнений екорейтинг і персоналізувати маршрутні рекомендації.

Практична цінність отриманих результатів. Практичне значення отриманих результатів полягає в тому, що на основі запропонованих теоретичних положень були розроблені алгоритми та програмні компоненти для системи управління бронюванням авіаквитків.

Особистий внесок здобувача. Усі наукові результати, що викладені в бакалаврській кваліфікаційній роботі, отримано автором самостійно. У публікаціях, виконаних у співавторстві, автору належать такі результати: аналіз сучасних систем мультимодального бронювання і розробка алгоритму пошуку мультимодальних маршрутів [2], розробка методу і алгоритму оцінки екологічного рейтингу акаунту [3].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Результати роботи доповідалися на:

− LIV Науково-технічній конференції підрозділів Вінницького національного технічного університету (Вінниця, 2025 р.);

− Міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи» (Вінниця, 2025 р.).

Публікації. За тематикою дослідження опубліковано 2 наукові праці у збірниках матеріалів конференцій.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану технологій організації процесу бронювання авіаквитків

У сучасних умовах авіаперевезення стали невід'ємною частиною як ділового життя, так і туризму. Швидкість переміщення, зручність планування маршрутів та доступність різноманітних напрямків роблять авіаподорожі важливою складовою глобалізованого світу. У зв'язку з цим процес бронювання авіаквитків набуває особливого значення. Від правильності й зручності бронювання залежить ефективність планування подорожей, економія коштів і часу пасажирів.

Існує кілька способів організації процесу бронювання авіаквитків. Серед них можна виділити традиційне бронювання через туристичні агентства, бронювання через кол-центри авіакомпаній, онлайн-бронювання через офіційні сайти перевізників, використання агрегаторів авіаквитків, мобільні додатки, а також спеціалізовані корпоративні системи для бізнес-клієнтів [4]. Розглянемо детально кожен із них.

Традиційне бронювання через туристичні агентства є одним із найстаріших способів організації придбання авіаквитків. У цьому випадку пасажир звертається до спеціалізованої компанії, де менеджер допомагає підібрати оптимальний маршрут, знайти відповідні рейси та оформити бронювання. Агенти часто мають доступ до спеціальних тарифів, можуть надати консультацію щодо пересадок, візових вимог та інших нюансів подорожі.

Перевагою такого способу є можливість отримати професійну допомогу та зекономити час на пошук варіантів самостійно. Також агент може врахувати особливі побажання клієнта або запропонувати додаткові послуги. Проте недоліками традиційного бронювання є необхідність особистого відвідування агентства або тривала комунікація через телефон чи електронну пошту, що може займати багато часу. Крім того, вибір варіантів іноді обмежується лише партнерами агентства, а вартість квитків може включати додаткову комісію.

Бронювання через кол-центри авіакомпаній передбачає оформлення квитків шляхом телефонного дзвінка до служби підтримки перевізника. Пасажир повідомляє оператору свої побажання щодо маршруту, дати подорожі та інших деталей, а оператор підбирає варіанти рейсів і допомагає оформити бронювання.

Перевагою такого способу є можливість швидко отримати інформацію без необхідності особистого відвідування офісу або самостійного пошуку в інтернеті. Оператор може надати актуальні дані про доступні місця, тарифи та умови змін або повернення квитків. Однак недоліком є обмеженість функціоналу – через телефон складніше опрацьовувати складні маршрути з пересадками або порівнювати велику кількість варіантів. Також можливі затримки через черги на лінії або тривалість консультації.

Онлайн-бронювання через агрегатори та агентські платформи стало одним із найпопулярніших способів придбання авіаквитків. Агрегатори, такі як Skyscanner, Expedia чи Kayak, збирають інформацію про рейси від різних авіакомпаній і агентств, дозволяючи користувачам швидко порівнювати ціни, умови та тривалість подорожі.

Серед плюсів такого способу – можливість легко знайти найвигідніші пропозиції завдяки широкому вибору варіантів, зручним фільтрам і системам сортування. Також часто доступні спеціальні акційні пропозиції або додаткові бонуси для користувачів. З недоліків може бути наявність прихованих комісій, які додаються вже на етапі оформлення покупки, а також ризик отримання застарілої або неточної інформації через затримки в оновленні даних з боку авіакомпаній.

Мобільні додатки для бронювання авіаквитків є сучасним і зручним способом організації подорожей. Такі додатки пропонують як самі авіакомпанії, так і агрегатори. Завдяки їм користувачі можуть шукати рейси, бронювати квитки, отримувати сповіщення про зміни в розкладі та зберігати посадкові талони безпосередньо на своїх смартфонах.

Основною перевагою, яку надають мобільні додатки, є можливість оперативно бронювати квитки у будь-який час і в будь-якому місці. Багато додатків пропонують персоналізовані пропозиції, програми лояльності та функціонал для

зручного керування бронюванням. Недоліком є можливі обмеження функціоналу порівняно з веб-версіями сайтів, особливо при пошуку складних маршрутів або багатоквиткових комбінацій.

Також варто відзначити бронювання через корпоративні системи, що є популярним методом для організації подорожей співробітників компаній [5]. Такі системи дозволяють автоматизувати процес бронювання авіаквитків, готелів, оренди автомобілів і інших послуг для бізнес-поїздок. Вони часто інтегровані з внутрішніми фінансовими системами компанії та пропонують зручний інтерфейс для управління витратами. Такі системи є дуже зручними для співробітників, але є обмеженими щодо використання зовнішніми користувачами.

Отже, проаналізувавши різні способи було обрано бронювання з використанням агрегаторів через їхню зручність та можливість порівнювати пропозиції від різних авіакомпаній і агентств в реальному часі. Незважаючи на незначні та рідкісні проблеми із синхронізацією даних, агрегатори забезпечують швидкий доступ до різноманітних варіантів рейсів, що дозволяє знайти найбільш вигідні тарифи та відповідні умови подорожі. Крім того, ці платформи часто пропонують додаткові функції, такі як фільтри для підбору рейсів, повідомлення про зміни та акційні пропозиції. Загалом, цей спосіб є ефективним для тих, хто самостійно шукає оптимальні варіанти подорожей через інтернет.

1.2 Порівняльний аналіз існуючих систем бронювання авіаквитків

З плином останніх десятиліть системи бронювання авіаквитків відчутно трансформувалися, пристосовуючись до технологічного прогресу та змін у запитах споживачів. Спочатку ключову роль у процесі бронювання відігравали традиційні туристичні агентства та авіакомпанії, через які користувачі могли замовляти лише квитки на окремі рейси. Однак з розвитком інтернет-технологій з’явилися онлайн-агрегатори, які дозволили порівнювати ціни на різні рейси від різних авіакомпаній, значно полегшуючи процес вибору та бронювання квитків. З часом ці системи стали більш комплексними, пропонуючи додаткові опції, як-от бронювання готелів, прокат автомобілів та створення цілісних туристичних пакетів.

Сучасні тренди в індустрії бронювання авіаквитків демонструють значний акцент на мультимодальних подорожах – маршрутах, що комбінують авіаперельоти з іншими видами транспорту, наприклад, поїздами, автобусами чи таксі. Це дозволяє користувачам формувати зручні та економічно вигідні маршрути з використанням різних видів транспорту, що особливо актуально для мандрівників, які прагнуть знизити викиди CO_2 та забезпечити більш екологічні подорожі, оскільки авіаперельоти є найбільшим забрудником атмосфери на рівні з автотранспортом [6].

У зв’язку з цими змінами виникає необхідність порівняння існуючих систем бронювання авіаквитків, оскільки нові вимоги, такі як екологічні маршрути та впровадження елементів гейміфікації, вимагають значно складнішої інтеграції різних видів транспорту та нових функцій. Вибір екологічних маршрутів, що передбачають мінімальний вуглецевий слід, стає важливою складовою в плануванні подорожей. Крім того, гейміфікація, як інструмент мотивації та залучення користувачів, може сприяти вибору більш сталих варіантів подорожей, пропонуючи бонуси за екологічно чисті маршрути чи участь у змаганнях за зменшення викидів. Таким чином, порівняння існуючих систем дозволить визначити, які з них найбільше відповідають сучасним вимогам сталого розвитку та інноваційних технологій.

Розглянемо 3 системи, що користуються високим попитом на ринках глобальних подорожей та аввіаперельотів: Kayak, Scyscanner і Google Flights.

Kayak – це туристична метапошукова система, яка спеціалізується на пошуку авіаквитків, бронюванні готелів, оренді автомобілів, а також пошуку турпакетів і круїзів [7]. Цей онлайн-сервіс належить компанії Booking Holdings. Kayak працює як метапошуковик, порівнюючи пропозиції від різних партнерів на основі обраних фільтрів та критеріїв сортування, таких як ціна, тривалість подорожі, тип транспорту тощо (див. рисунок 1.1). Система дозволяє зберігати вибрані варіанти, переглядати історію пошуків і підписуватись на сповіщення про зміну цін, а також шукати дешевші перельоти за допомогою функції «Гнучкі дати». Kayak доступний у вебверсії та як мобільний застосунок.

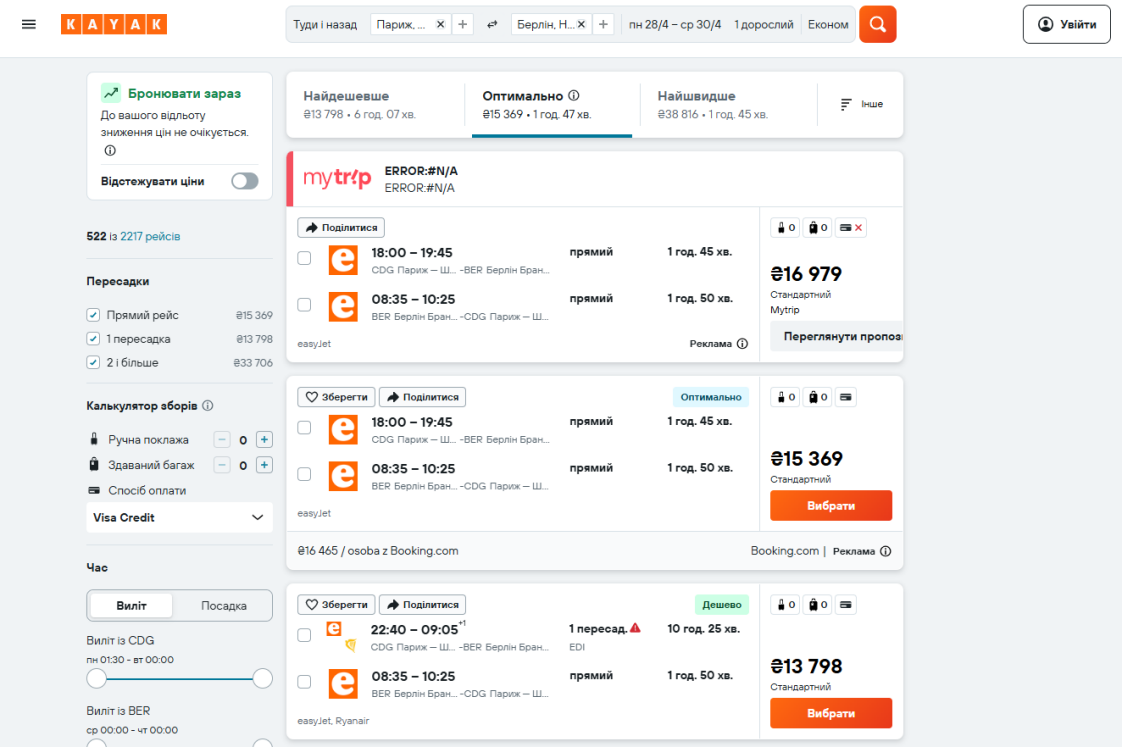

Рисунок 1.1 – Фільтр пошуку сервісу Kayak

Важливо зазначити, що Kayak не є платформою для безпосереднього бронювання або покупки послуг. Користувачі здійснюють вибір варіанту перельоту, готелю чи оренди автомобіля безпосередньо на сайті метапошуковика, однак фінальна оплата проводиться на сайті партнерів, таких як авіакомпанії, готелі чи прокатні компанії. Це дає можливість Kayak обійти онлайн-турагентства, які можуть відстежувати користувачів і пропонувати різні ціни в залежності від їхнього пристрою (наприклад, для користувачів Mac чи iOS, ціна може бути вищою).

Завдяки такому підходу Kayak забезпечує користувачів зручним інтерфейсом для пошуку і порівняння різних варіантів подорожей, але без здійснення безпосередніх транзакцій.

Skyscanner – це відомий метапошуковик для планування подорожей, штаб-квартира якого розташована в Единбурзі, Шотландія [8]. Сервіс дозволяє мандрівникам знайти та порівняти ціни на авіаквитки, готелі та оренду автомобілів. Skyscanner працює шляхом пошуку пропозицій у базах даних численних постачальників і агентств, але сам не здійснює безпосередньо бронювання.

Користувачі можуть вибрати з численних варіантів, а бронювання здійснюється безпосередньо на сайтах партнерів.

Skyscanner надає можливість будувати складні маршрути, комбінуючи різні авіарейси для здійснення перельотів, що дозволяє користувачам планувати довгі подорожі з кількома пересадками. Окрім цього, система дає можливість окремо забронювати готелі та орендувати автомобілі на місці, що забезпечує мандрівникам зручність у плануванні всієї подорожі.

Хоча в Skyscanner є функції, що вказують на екологічність варіантів, сервіс не пропонує можливості сортування маршрутів за рівнем вуглецевого сліду. Окрім того, сервіс має мобільний додаток, що дозволяє зручно планувати подорожі за допомогою смартфона. Skyscanner також надає інструменти для моніторингу цін, що дозволяють відстежувати зміни в тарифах і купувати квитки в найвигідніший момент (див. рисунок 1.2).

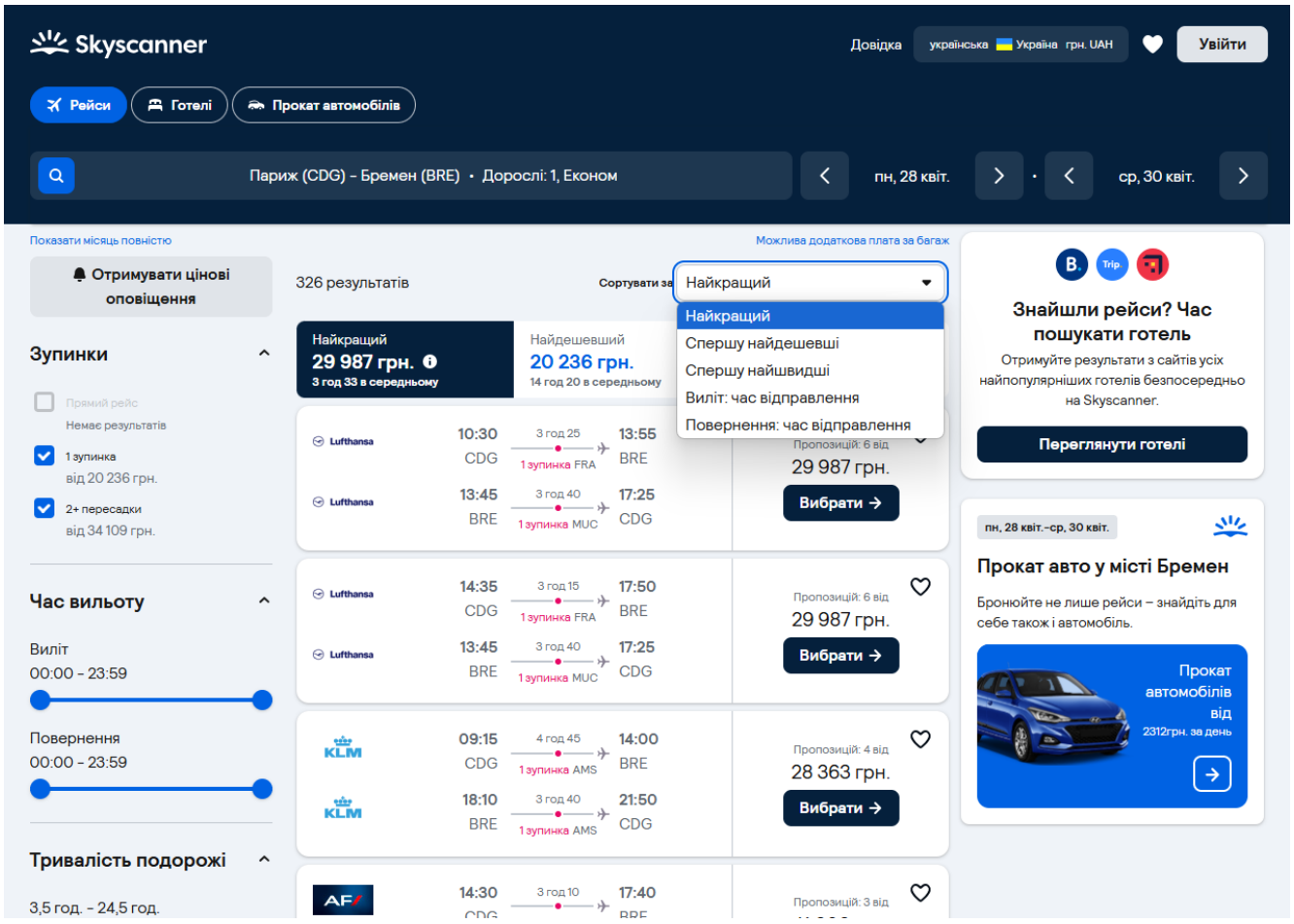

Рисунок 1.2 – Фільтр пошуку сервісу Scyscanner

Google Flights – це потужний онлайн-інструмент для пошуку та бронювання авіаквитків, який надає користувачам можливість порівнювати ціни на рейси від різних авіакомпаній, а також бронювати авіаквитки через сайти партнерів. Інтерфейс платформи дозволяє зручно фільтрувати результати пошуку за різними критеріями, такими як ціна, час вильоту, кількість пересадок та інші параметри [9]. Google Flights також пропонує інструменти для моніторингу цін, що дозволяє користувачам стежити за коливаннями цін на рейси і отримувати сповіщення, коли ціна на вибраний рейс змінюється (див. рисунок 1.3).

Особливістю Google Flights є інтеграція з іншими сервісами Google, що дозволяє користувачам, які використовують інші продукти компанії, зручно планувати свої поїздки. Платформа також надає варіанти вибору альтернативних маршрутів і перельотів, що можуть бути дешевшими або зручнішими. Крім того, Google Flights пропонує опції для пошуку авіаквитків в умовах, коли користувач не має конкретної дати подорожі, даючи можливість переглядати варіанти за кілька тижнів або місяців вперед.

Щодо екологічності, Google Flights дає можливість користувачам порівнювати викиди вуглецю для кожного маршруту, надаючи інформацію про рівень вуглецевого сліду конкретного рейсу. Також існує опція сортування за рівнем вуглецевого сліду, що надає можливості для вибору екологічних варіантів.

Google Flights може пропонувати альтернативи у вигляді залізничних перевезень для деяких маршрутів, що дозволяє користувачам обирати між авіаперельотами та потягами залежно від ціни, часу, зручності та викидів CO_2. Однак, платформа не має функції побудови мультимодальних подорожей, що обмежує можливості для планування подорожей з урахуванням різних видів транспорту, що важливо для користувачів, які шукають більш гнучкі та екологічні варіанти подорожей.

Крім основної функціональності, Google Flights дозволяє швидко аналізувати найпопулярніші напрямки та інтерактивно обирати їх на віртуальному глобусі, що може бути корисним для мандрівників із гнучкими планами. Платформа підтримує багатомовний інтерфейс і адаптована до мобільних пристроїв.

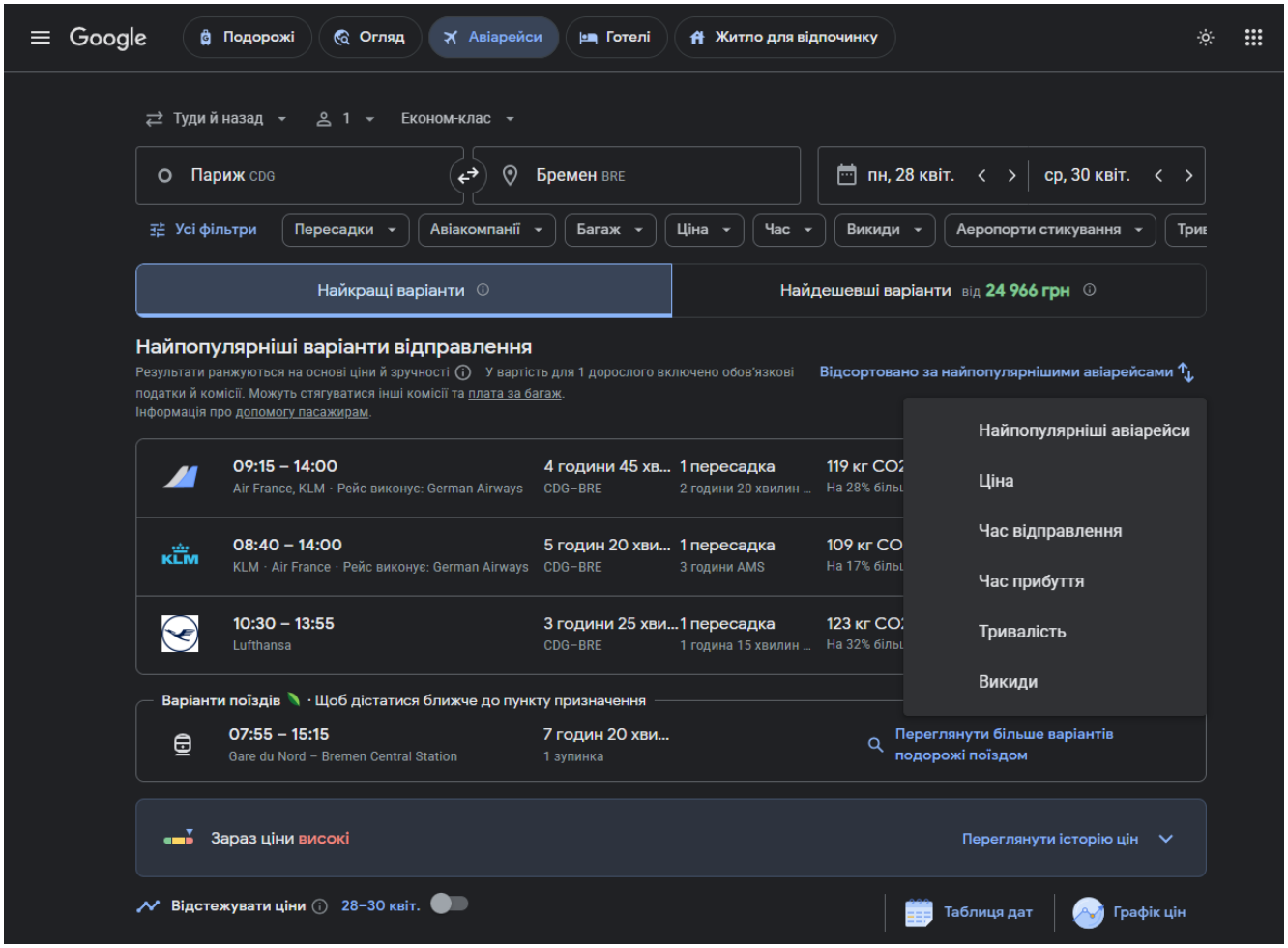

Рисунок 1.3 – Фільтр пошуку сервісу Google Flights

Нижче наведено таблицю 1.1, що узагальнює аналіз аналогів.

Таблиця 1.1 – Порівняльна характеристика аналогів

Функція	Kayak	Scyscanner	Google Flights	Власна реалізація
Мультимодальні маршрути	+	-	-	+
Альтернативні види транспорту	+	-	+	+
Гейміфікація	-	-	-	+
Врахування CO_2	-	+	+	+
Персоналізація рекомендацій	+	-	+	+
Сумарний коефіцієнт	3	1	3	5

Пояснення критеріїв оцінки:

1. Підтримка мультимодальних маршрутів: оцінка здатності системи пропонувати маршрути, що поєднують різні види транспорту (наприклад, авіапереліт, потяг, автобус, оренда велосипеда тощо). Це дозволяє користувачам будувати більш гнучкі і ефективні маршрути, що відповідають їхнім уподобанням щодо економії часу, коштів або екологічності. Система може враховувати пересадки та оптимізувати маршрут з урахуванням зручності і вартості.

2. Наявність альтернативних видів транспорту: оцінка можливості пропонування варіантів подорожі, окрім традиційних авіаперельотів (наприклад, автобуси, поїзди, велосипеди або навіть пішохідні маршрути). Цей критерій важливий для користувачів, які прагнуть знизити свій екологічний слід або мають обмеження в бюджеті. Чим більше альтернативних варіантів пропонує система, тим зручніше користувачеві планувати подорожі, враховуючи різні аспекти комфорту та екологічної свідомості.

3. Гейміфікація: оцінка того, наскільки система впроваджує елементи гейміфікації (ігрові елементи, нагороди, досягнення, рейтинги). Наприклад, за вибір екологічно чистих маршрутів користувач може отримувати бали або досягнення, що мотивують користувача робити більш усвідомлені екологічні вибори. Гейміфікація дозволяє підвищити залученість користувачів і створити емоційно насичений досвід, що позитивно впливає на їхню лояльність до системи.

4. Врахування вуглецевого сліду: оцінка здатності системи надавати інформацію про вуглецевий слід кожного маршруту, а також пропонувати маршрути, які мінімізують цей слід. Цей критерій важливий для користувачів, які піклуються про екологію та бажають зменшити свій вплив на навколишнє середовище. Користувач може порівнювати різні маршрути не тільки за вартістю та зручністю, а й за екологічним впливом.

5. Рівень персоналізації рекомендацій: оцінка того, наскільки система адаптується до інтересів та уподобань користувача, зокрема в контексті збереження екологічних покупок і відображення цього в акаунті користувача (наприклад, екологічний рейтинг або історія екологічно чистих маршрутів). Це дозволяє

створити більш персоналізований досвід, зокрема через рекомендації на основі попередніх виборів.

1.3 Аналіз методів розв'язання задачі автоматизації бронювання

Одним із традиційних методів автоматизації процесу бронювання авіаквитків є використання глобальних дистрибутивних систем (Global Distribution Systems, GDS). Ці системи були створені ще у другій половині XX століття великими авіакомпаніями для централізованого управління даними про рейси, ціни та доступність місць. До найвідоміших GDS належать Sabre, Amadeus, Galileo та Worldspan.

GDS виконують роль посередників між авіакомпаніями, туристичними агентствами та кінцевими споживачами. Вони об'єднують у єдиній базі даних інформацію про рейси різних авіакомпаній, що дозволяє агентствам і системам бронювання швидко знаходити потрібні варіанти для клієнтів. Завдяки GDS забезпечується оперативне оновлення розкладів, тарифів, наявності місць та миттєве підтвердження бронювання. Крім цього, системи GDS підтримують стандартизовані протоколи обміну даними, що спрощує інтеграцію з іншими сервісами й дозволяє автоматизувати обробку великої кількості запитів у реальному часі.

Основними перевагами використання GDS є [4]:

- широкий доступ до пропозицій від різних авіакомпаній у реальному часі;
- можливість миттєвого бронювання та оплати;
- надійність і масштабованість рішення, перевірена десятиліттями експлуатації.

Однак використання GDS має і певні обмеження. Вартість підключення та користування сервісом зазвичай досить висока, що може бути недоцільним для невеликих компаній. Крім того, традиційні GDS орієнтовані переважно на авіаперевезення й готелі та лише нещодавно почали інтегрувати інші види транспорту, що ускладнює реалізацію повноцінних мультимодальних маршрутів.

Також у більшості GDS відсутні вбудовані можливості врахування екологічного впливу або застосування елементів гейміфікації для користувачів.

Таким чином, хоча GDS залишаються важливим елементом екосистеми бронювання авіаквитків, для розробки сучасної мультимодальної системи з акцентом на екологічність і персоналізацію вони не є оптимальним рішенням.

Одним із поширених підходів до автоматизації процесу бронювання авіаквитків є використання API (Application Programming Interface) авіакомпаній та агрегаторів. API надають стандартизований спосіб взаємодії з базами даних перевізників або платформ бронювання в режимі реального часу. Через API можна отримувати актуальні дані про наявність місць, розклади рейсів, тарифи, спеціальні пропозиції, а також здійснювати бронювання та оплачувати квитки.

API авіакомпаній зазвичай дозволяють працювати безпосередньо з інформацією перевізника, що забезпечує точність і оперативність даних. Водночас використання API агрегаторів, таких як Amadeus, Skyscanner, Kiwi тощо, дає змогу отримувати доступ до зведеної інформації відразу з багатьох джерел, що спрощує пошук оптимальних варіантів для користувача.

Використання API значно спрощує інтеграцію з існуючими системами бронювання та дозволяє реалізувати розширені функціональні можливості, зокрема підтримку мультимодальних маршрутів, автоматичну перевірку вуглецевого сліду рейсів та гейміфікаційні механіки для стимулювання екологічного вибору.

Особливу увагу при використанні API слід приділяти питанням безпеки даних, обмеженням ліцензійних умов, а також моніторингу стабільності доступу до сервісів постачальників.

Можливим шляхом автоматизації процесу бронювання є створення власних інтегрованих рішень. Такий підхід передбачає розробку програмного забезпечення, яке самостійно об'єднує дані з кількох джерел, оптимізує маршрути та забезпечує додаткову функціональність відповідно до специфічних потреб користувачів.

Власне рішення дає змогу краще контролювати логіку пошуку та бронювання, адаптувати сервіс під особливі вимоги – наприклад, підвищену увагу

до екологічності маршрутів, підтримку мультимодальних поїздок (авіа, залізниця, автобус), реалізацію елементів гейміфікації чи персоналізованих рекомендацій.

Така архітектура зазвичай передбачає створення спеціального бекенду, що виконує агрегування даних, інтелектуальну обробку запитів, взаємодію з кількома зовнішніми API, а також використання додаткових модулів для аналітики, кешування результатів і обробки платежів [10]. Це складніший шлях порівняно з інтеграцією окремого API, але він забезпечує вищу гнучкість, масштабованість і відповідність унікальним вимогам проєкту.

Розробка власного інтегрованого рішення також дозволяє уникнути обмежень, пов’язаних із політиками сторонніх API, такими як ліміти на кількість запитів, вимоги до брендування або високі ліцензійні платежі.

Отже, при розробці буде використовуватися власне рішення, що буде інтегрувати дані з декількох джерел та збирати їх в зручному для користувача вигляді.

1.4 Постановка задач для розробки системи управління бронюванням авіаквитків

Проаналізувавши стан розвитку систем бронювання авіаквитків та мультимодальних подорожей, а також виявивши потребу у врахуванні екологічних критеріїв і елементів гейміфікації, було визначено наступні задачі для розробки веб-застосунку:

– здійснити аналіз сучасних підходів до організації бронювання транспортних маршрутів;

– спроєктувати архітектуру інформаційної системи з урахуванням мультимодальних перевезень;

– розробити метод побудови мультимодального маршруту з урахуванням пересадок та вуглецевого сліду;

– розробити метод оцінки екологічного рейтингу акаунту користувача на основі історії замовлень і відповідей на опитування;

– реалізувати механізм персоналізованих рекомендацій на основі історії вибору користувача;

– реалізувати механізм візуалізації екологічного рейтингу акаунту;

– реалізувати модуль пошуку маршрутів із використанням зовнішніх API авіаперевізників;

– здійснити інтеграцію з API геокодування для автоматичного пошуку найближчих аеропортів;

– провести тестування застосунку та перевірку відповідності функціональним вимогам;

– розробити інструкцію користувача та описати системні вимоги до програмного забезпечення.

Технічне завдання на розробку наведено у додатку А.

1.5 Висновки

У першому розділі було проведено аналіз сучасних систем бронювання авіаквитків, розглянуто тенденції розвитку індустрії подорожей, зокрема мультимодальність, екологічність та гейміфікацію. В результаті порівняльного аналізу існуючих рішень (Kayak, Skyscanner, Google Flights) виявлено їхні сильні сторони та обмеження в контексті нових вимог користувачів.

Особливу увагу приділено аналізу методів автоматизації бронювання, серед яких розглянуто використання глобальних систем дистрибуції (GDS), інтеграцію через API та можливості створення власних програмних рішень.

На основі проведеного дослідження сформульовано основні задачі для розробки веб-сервісу, що орієнтується на побудову мультимодальних екологічних маршрутів із залученням елементів гейміфікації та персоналізованих рекомендацій.

У наступних розділах буде деталізовано процес проєктування, реалізації та тестування запропонованого рішення.

2 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних та вихідних даних системи

Вхідні та вихідні дані є основою роботи мультимодального застосунку для пошуку та бронювання маршрутів. Від правильності обробки вхідних даних залежить точність і релевантність результатів, а якість вихідних даних визначає користувацький досвід. Основними джерелами вхідних даних у програмному продукті є сам користувач, який формулює запит через інтерфейс застосунку, а також зовнішні API-сервіси, які забезпечують актуальну інформацію про доступні маршрути.

Користувач взаємодіє із системою через введення параметрів подорожі, де зазначає пункт відправлення, пункт призначення, бажану дату та час подорожі, а також додаткові фільтри – наприклад, обрати дешевший, швидший або екологічніший маршрут. Ці дані є важливими для коректної побудови персоналізованих варіантів поїздок. Крім параметрів пошуку, вхідними даними є реєстраційна інформація користувача, яка включає адресу електронної пошти, пароль та, за потреби, інші персональні дані. Всі дані реєстрації та входу обов’язково проходять валідацію для забезпечення безпеки облікового запису.

Після первинного запиту застосунок звертається до зовнішніх сервісів. Через інтеграцію з Serp API (Google Flights) система отримує актуальний список авіарейсів із детальною інформацією про маршрути, ціни та тривалість перельоту [11]. Одночасно через Google Cloud Platform використовується сервіс геокодування для перетворення введених користувачем адрес у координати, що необхідно для побудови маршрутів та їхньої подальшої візуалізації. Таким чином, зовнішні API стають важливим джерелом вхідних даних, які комбінуються із даними користувача для формування результатів.

Під час взаємодії з системою відбувається накопичення даних про історію поїздок користувача, його екологічний рейтинг, обрані маршрути та історію бронювань. Ці внутрішні дані профілю також слугують вхідними для

персоналізації подальших рекомендацій. Наприклад, аналізуючи попередні вибори користувача, система може пропонувати маршрути, які краще відповідають його вподобанням щодо вартості або екологічності.

Вихідні дані застосунку включають результати пошуку маршрутів, що задовольняють введені параметри. Користувач отримує список варіантів подорожі, кожен із яких містить детальну інформацію про види транспорту, тривалість, вартість, кількість пересадок та прогнозований рівень викидів вуглецю. Крім основної інформації, результати пошуку можуть бути доповнені персоналізованими рекомендаціями на основі історії активності. Для покращення користувацького досвіду система візуалізує маршрут на карті, чітко виділяючи сегменти за видами транспорту.

Після вибору маршруту користувач має можливість переглянути деталі бронювання, провести оплату та отримати підтвердження, що також належить до вихідних даних програми. Інформація про успішно заброньовані маршрути додається до історії поїздок профілю користувача. Застосунок також автоматично розраховує оновлений екологічний рейтинг облікового запису, що відображається у профілі.

Особливістю вихідних даних є можливість їхнього динамічного оновлення. Наприклад, при зміні статусу рейсу користувач отримує сповіщення про затримку або скасування, що підвищує рівень сервісу. Додатково система генерує бонусні бали за екологічні вибори, які також виводяться у профіль і впливають на персоналізовані пропозиції у майбутньому.

Таким чином, визначено вхідні та вихідні дані, їхнє правильне опрацювання та інтеграція із зовнішніми джерелами дозволяють забезпечити високу якість функціонування застосунку, підвищуючи рівень користувацького досвіду.

2.2 Розробка архітектури програмного застосунку

Архітектура програмної системи визначає структуру і взаємодію компонентів, а також принципи проєктування, що забезпечують гнучкість, масштабованість і підтримку в майбутньому. Враховуючи принципи Clean

Architecture, архітектура застосунку розроблена з фокусом на ізоляцію компонентів, чітке розмежування відповідальностей і мінімізацію залежностей.

Згідно з цими принципами, застосунок складається з кількох шарів, кожен з яких відповідає за окрему функціональність [12]:

- Core Layer (Доменний шар) містить бізнес-логіку і моделі даних, які не залежать від зовнішніх систем, таких як API.

- Application Layer (Шар застосунку) включає сервіси для обробки запитів на бронювання, взаємодію з API та роботу з історією поїздок, не залежачи від зовнішніх систем.

- Interface Layer (Шар інтерфейсу) забезпечує взаємодію з користувачем та зовнішніми сервісами через інтерфейси для відображення даних і отримання запитів.

- Infrastructure Layer (Інфраструктурний шар) реалізує зовнішні залежності, зокрема інтеграцію з базою даних і зовнішніми API.

Взаємодія між шарами здійснюється через чітко визначені інтерфейси. Доменна логіка з Core Layer визначає роботу функціональності, яка обробляється в Application Layer. Шар застосунку передає дані до Interface Layer, який, у свою чергу, взаємодіє з зовнішніми системами через Infrastructure Layer.

Переваги такої архітектури [13]:

- Зміна одного компонента не впливає на інші частини системи, наприклад, можна змінювати API без порушення бізнес-логіки.

- Архітектура дозволяє тестувати кожен шар окремо, що покращує якість коду та знижує ймовірність помилок.

- Система легка для підтримки і оновлення, оскільки зміни в інтеграціях чи базі даних не порушують основну логіку застосунку.

- Інтеграція з зовнішніми API здійснюється через інтерфейси, що знижує залежність від конкретних реалізацій. Це дозволяє заміняти чи оновлювати API без змін в основній логіці.

У рамках розробки застосунку для бронювання авіаперельотів та побудови мультимодальних маршрутів, було визначено чітку структуру програмної системи,

яка складається з кількох ключових шарів, кожен з яких має свою функціональність. Кожен із цих шарів виконує свою роль, забезпечуючи взаємодію між бізнес-логікою, користувачем та зовнішніми сервісами, що дозволяє побудувати масштабовану та підтримувану систему.

1. Core Layer (Доменний шар):

- моделі даних для маршрутів, користувачів, історії поїздок і екологічного рейтингу;

- алгоритм розрахунку оптимальних маршрутів, який враховує не лише час, але й вуглецевий слід, що дає змогу знаходити найбільш екологічні варіанти;

- обробка запитів користувачів для пошуку та бронювання маршрутів;

- логіка персоналізованих рекомендацій на основі історії поїздок;

- оцінка екологічного рейтингу для маршруту.

2. Application Layer (Шар застосунку):

- сервіси для взаємодії з API, такими як Google Flights API для отримання інформації про рейси та Google Cloud API для геокодування та візуалізації маршрутів;

- обробка запитів на бронювання маршрутів та взаємодія з користувачем для вибору найкращого маршруту;

- управління історією поїздок користувача, що дає змогу зберігати й обробляти попередні маршрути;

- формування персоналізованих рекомендацій на основі профілю користувача;

- операції фільтрації результатів пошуку за різними критеріями, такими як ціна, час у дорозі або екологічний рейтинг.

3. Interface Layer (Шар інтерфейсу):

- графічний інтерфейс (UI), який відображає результати пошуку, надає можливість бронювання маршрутів і відображає інформацію про користувача;

- інтерфейси для взаємодії з зовнішніми API (наприклад, Google Flights API для отримання актуальних даних про рейси);

- обробка вхідних даних від користувача, таких як вибір параметрів подорожі або налаштування фільтрів пошуку.

4. Infrastructure Layer (Інфраструктурний шар):

- зберігання даних про маршрути та користувачів у базі даних, що забезпечує збереження важливої інформації для подальшого аналізу та використання;

- інтеграція з зовнішніми сервісами (Google Flights API, Google Cloud API) для отримання актуальної інформації та візуалізації маршрутів;

- управління файлами, наприклад, завантаження та збереження зображень профілю користувача або візуалізацій маршруту.

Багатошарову архітектуру застосунку показано на рисунку 2.1.

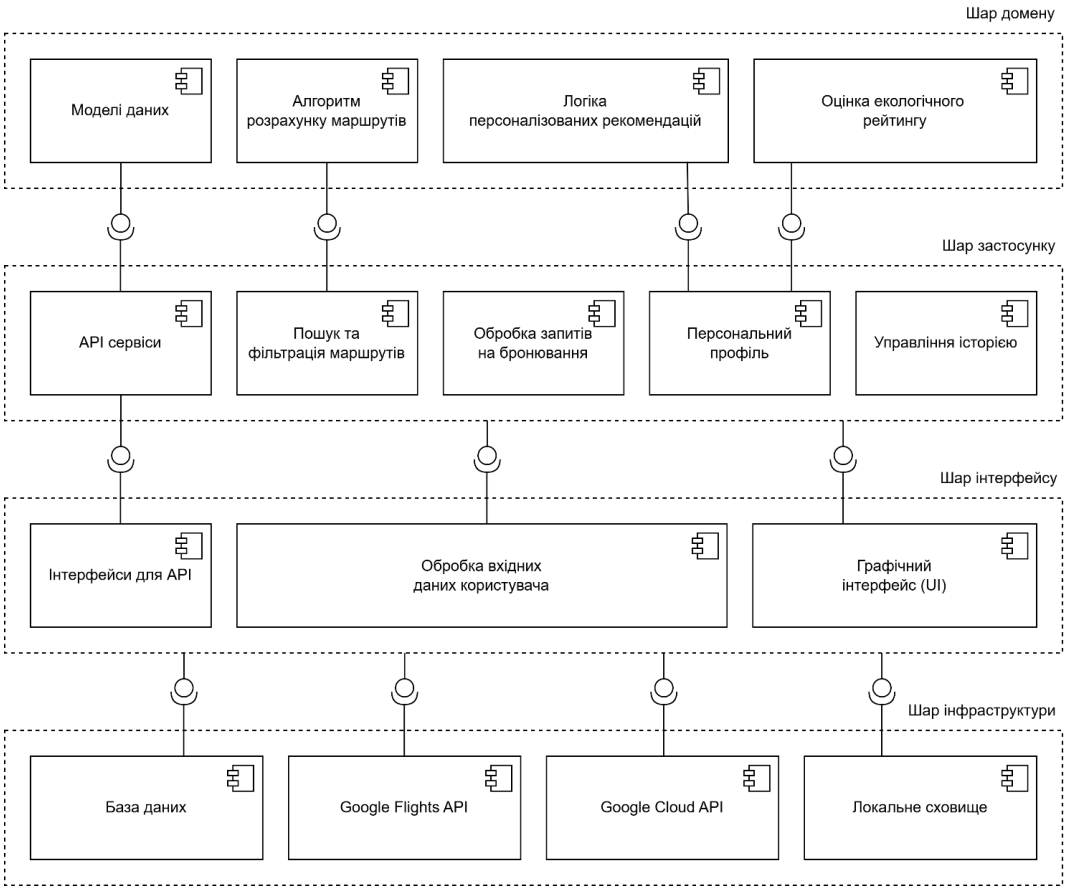

Рисунок 2.1 – Багатошарова архітектура застосунку бронювання авіаквитків

Загалом, архітектура за принципами Clean Architecture гарантує високу гнучкість, масштабованість та простоту в підтримці системи. Чітке розмежування шарів і залежностей сприяє адаптації до змін у вимогах та технологіях, а також забезпечує зручність тестування та інтеграції з новими сервісами.

2.3 Розробка структури програмного застосунку

Use Case діаграма (діаграма варіантів використання) є одним з основних інструментів уніфікованого моделювання (UML) для фіксації функціональних вимог до системи. Діаграма варіантів використання описує зовнішню поведінку системи з точки зору її користувачів (акторів) і показує, які сценарії (use cases) вони можуть ініціювати або в яких можуть брати участь [14]. Основна мета таких діаграм – допомогти виявити вимоги до системи і забезпечити чітке розуміння взаємодії між користувачами та функціональністю програмного продукту.

В рамках розробки застосунку для бронювання мультимодальних екологічних маршрутів було побудовано діаграму варіантів використання, яка відображає усі ключові функціональні можливості системи та основних акторів.

Основними акторами системи є:

1) Користувач – фізична особа, яка взаємодіє із системою для реєстрації, авторизації, пошуку маршрутів, бронювання поїздок, перегляду історії подорожей, отримання рекомендацій та перегляду екологічного рейтингу акаунту.

2) Serp API (Google Flights) – сторонній API, для отримання даних про рейси;

3) Google Cloud APIs – картографічні сервіси (для геокодування та візуалізації маршрутів).

Користувач може виконувати наступні варіанти використання:

- реєстрація акаунту з подальшою авторизацією в системі;
- авторизація за наявності облікового запису;
- пошук маршрутів за заданими параметрами;
- перегляд списку доступних маршрутів;
- перегляд детальної інформації про вибраний маршрут;
- бронювання обраного маршруту;
- перегляд статусу заброньованих маршрутів;
- перегляд історії минулих бронювань;
- перегляд профілю та екологічного рейтингу;
- отримання персоналізованих рекомендацій;

- отримання гейміфікаційних нагород за вибір маршрутів із мінімальним вуглецевим слідом.

Взаємодія із зовнішніми сервісами охоплює:

- отримання актуальної інформації про рейси та альтернативні види транспорту;

- отримання картографічних даних для візуалізації маршрутів на мапі.

Діаграма варіантів використання подана на рисунку 2.2.

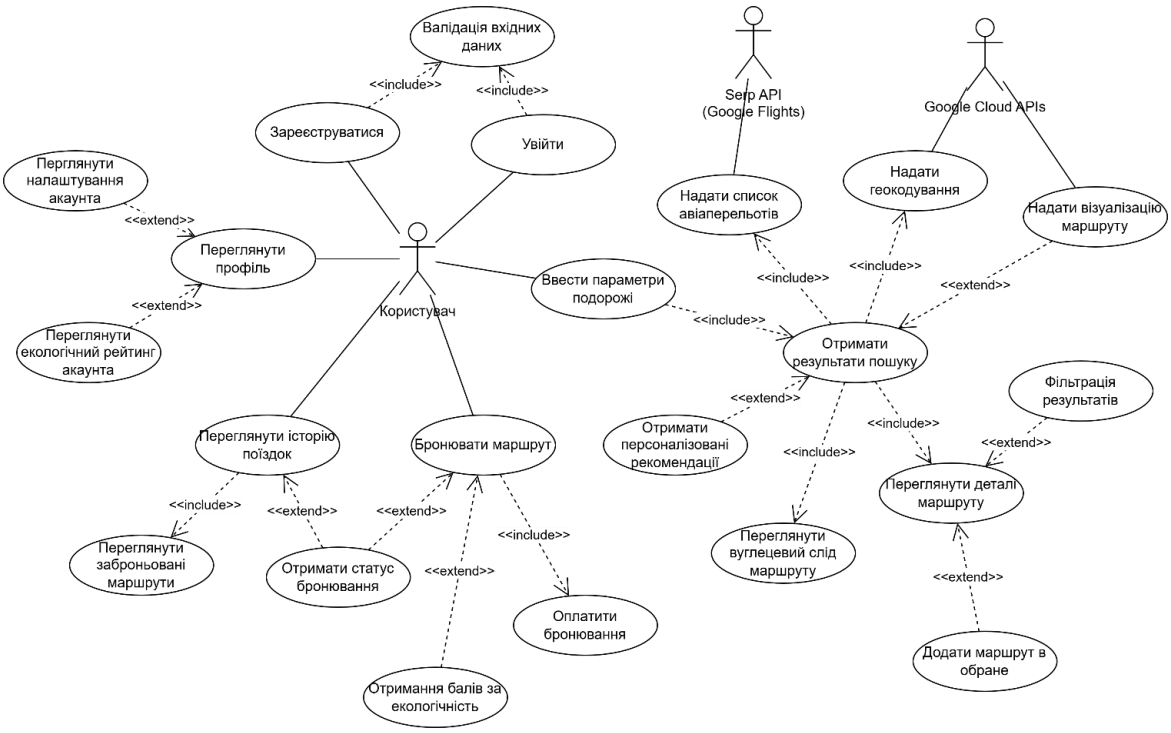

Рисунок 2.2 – Діаграма варіантів використання

Для відображення залежностей між варіантами використання застосовано зв’язки типу:

<<include>> – обов’язкове включення одного сценарію до іншого (наприклад, процес бронювання маршруту включає оплату бронювання).

<<extend>> – умовне розширення поведінки (наприклад, отримання нагороди за вибір екологічного маршруту є розширенням основного сценарію бронювання).

Таким чином, побудована діаграма варіантів використання дає змогу систематизувати основні функціональні вимоги до застосунку та забезпечує основу для подальшої деталізації сценаріїв роботи користувача.

2.4 Розробка загального алгоритму програми

Діаграма станів (State Machine Diagram) описує різні стани об’єкта та переходи між ними під дією подій. Вона є особливо корисною для опису поведінки реактивних систем, де на зміну станів впливають зовнішні або внутрішні тригери [14].

У розроблюваному застосунку перехід між станами розпочинається з початкового стану запуску програми, що переводить користувача на головну сторінку. Головна сторінка виступає як центральний вузол, з якого доступні різні напрямки роботи із застосунком. Один з основних процесів – це авторизація користувача. З головної сторінки можливий перехід до сторінки авторизації. На екрані авторизації користувач може або ввести облікові дані для входу, або обрати створення нового акаунту. При спробі авторизації система перевіряє валідність введених даних. Якщо дані некоректні, користувач повертається до форми авторизації для повторного введення; у разі успішної авторизації відбувається повернення на головну сторінку. Якщо користувач обирає реєстрацію нового акаунту, він заповнює відповідну форму і після її успішного заповнення також перенаправляється на головну сторінку.

Іншим важливим процесом є перегляд профілю. З головної сторінки користувач має змогу перейти до профілю. У цьому випадку система перевіряє, чи користувач авторизований. Якщо так, відкривається екран профілю. У профілі користувач може налаштувати власні вподобання для пошуку маршрутів, змінити ім’я або фото профілю, а також переглянути свій екологічний рейтинг. Після завершення роботи із профілем користувач може вийти, що переводить застосунок у кінцевий стан.

Процес пошуку маршрутів починається з переходу з головної сторінки на форму пошуку. Користувач вводить параметри пошуку та ініціює запит маршрутів. Результати відображаються у вигляді списку маршрутів. Після вибору певного маршруту користувач переглядає його деталі. Зі сторінки деталей маршруту можна ініціювати бронювання. У момент бронювання з’являється умовний блок: користувач може підтвердити або відхилити бронювання. Якщо маршрут

підтверджено, користувач переходить до перегляду статусу маршруту. З цього стану можна або переглянути історію попередніх маршрутів, або завершити роботу із застосунком, перейшовши у кінцевий стан.

Перегляд історії маршрутів також є окремим процесом. Із головної сторінки користувач може перейти безпосередньо до розділу історії маршрутів, де він має змогу переглядати всі свої попередні бронювання. Крім перегляду, користувач також може додавати маршрути в уподобані для швидкого доступу у майбутньому. Завершення перегляду історії маршрутів здійснюється через вихід, що знову переводить систему у кінцевий стан.

Таким чином, загальний алгоритм роботи застосунку охоплює послідовність станів, що охоплюють реєстрацію та авторизацію користувачів, пошук і бронювання маршрутів, перегляд статусу та історії бронювань, а також роботу з персональним профілем. Кожен стан чітко визначений і залежить від дій користувача, що забезпечує логічну та послідовну поведінку системи.

Розроблену діаграму станів наведено на рисунку 2.3.

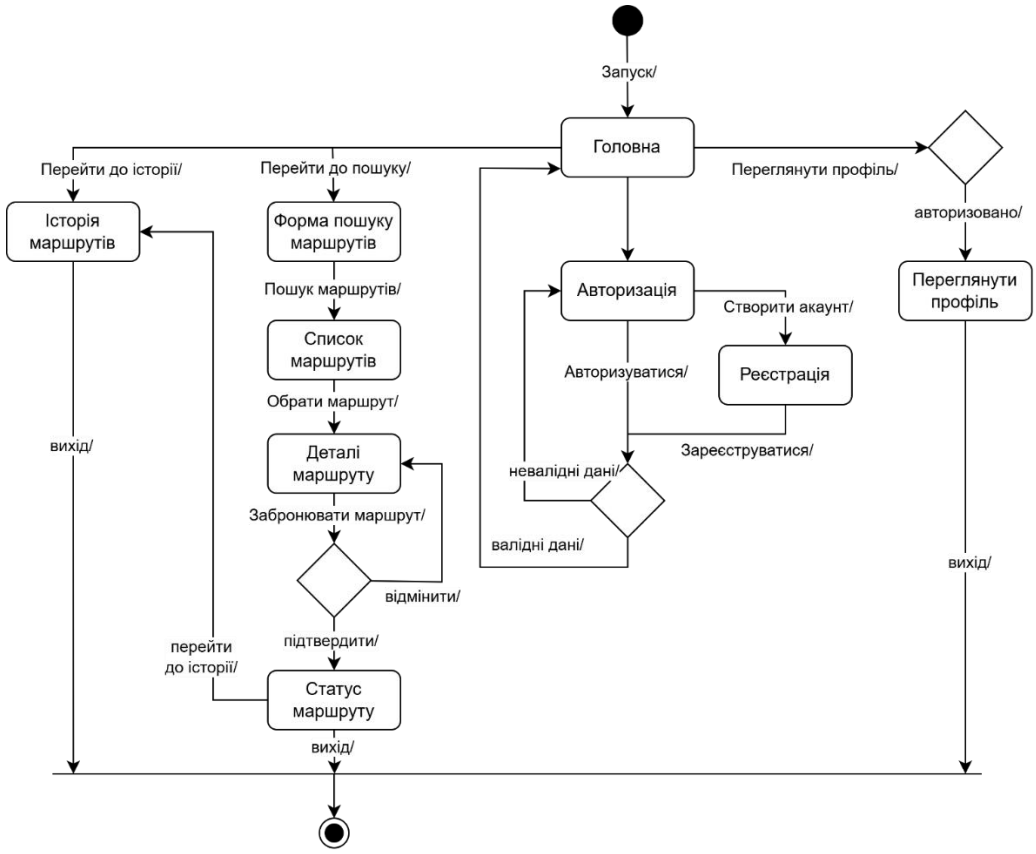

Рисунок 2.3 – Загальний алгоритм роботи програми у вигляді діаграми станів

2.5 Розробка методу побудови мультимодальних маршрутів з урахуванням вуглецевого сліду

Великі аеропорти, такі як міжнародні хаби, зазвичай обслуговують далекомагістральні рейси, де використовуються сучасні літаки, наприклад, Airbus A380 чи Boeing 787. Ці літаки перевозять сотні пасажирів за один рейс, що забезпечує високий коефіцієнт завантаження (load factor). Завдяки цьому викиди CO_2, розподілені на одного пасажира, значно нижчі. Довгі рейси також ефективніші, оскільки більшу частину часу літак проводить у крейсерському польоті, який потребує менше палива порівняно з енергоємними фазами зльоту та посадки. Наприклад, за даними досліджень, викиди на пасажиро-кілометр для далекомагістральних рейсів можуть бути вдвічі нижчими, ніж для коротких перельотів [15].

На противагу цьому, маленькі аеропорти здебільшого обслуговують регіональні маршрути, де використовуються менші літаки, такі як Bombardier CRJ чи ATR 72. Ці літаки мають нижчу енергоефективність і часто літають із меншою кількістю пасажирів на борту. Короткі рейси, характерні для таких аеропортів, додатково підвищують викиди на пасажиро-кілометр, оскільки значна частка палива витрачається на зліт і посадку. Дослідження показують, що регіональні літаки можуть генерувати до 150-200 г CO_2 на пасажиро-кілометр, тоді як великі літаки на довгих маршрутах – близько 80-100 г [15]. Отже, великі аеропорти мають перевагу в зниженні викидів CO_2 на пасажира завдяки економії масштабу, сучасним технологіям і довшим маршрутам.

Електричні поїзди є однією з найкращих альтернатив авіаційному транспорту з точки зору викидів CO_2 на пасажира, особливо на коротких і середніх маршрутах до 1000 км. Електричні поїзди в середньому генерують лише 14 г CO_2 на пасажиро-кілометр, тоді як авіарейси на короткі відстані можуть продукувати 150-250 г CO_2 на пасажиро-кілометр [16]. Особливо низькі викиди спостерігаються в країнах із низьковуглецевою енергетикою, таких як Франція чи Швеція, де електроенергія виробляється з ядерних або відновлювальних джерел. Наприклад, швидкісний

поїзд Eurostar, який сполучає Лондон і Париж, має мінімальний вуглецевий слід порівняно з авіарейсами на цьому маршруті.

Електропоїзди також мають додаткові переваги: вони не продукують локальних викидів NOx чи дрібних частинок (PM2.5), які характерні для авіаційних двигунів, і сприяють меншій руйнації екосистем порівняно з розширенням аеропортів. На маршрутах до 1000 км поїзди можуть конкурувати з авіацією за часом, особливо якщо врахувати час на реєстрацію та трансфер в аеропортах. Однак їхня ефективність залежить від джерела електроенергії: у країнах із високою часткою вугільної енергетики викиди можуть бути вищими, але все одно значно нижчими, ніж у авіації. Таким чином, електричні поїзди є ключовим елементом декарбонізації транспорту, особливо для заміни коротких авіарейсів.

Таким чином великі аеропорти демонструють нижчі викиди CO_2 на пасажира порівняно з маленькими завдяки економії масштабу, використанню сучасних літаків і довшим маршрутам, які зменшують частку енергоємних фаз польоту. Водночас електричні поїзди є значно екологічнішою альтернативою авіації на коротких і середніх маршрутах, генеруючи в рази нижчі викиди. Це вказує на можливість оптимізації транспортних систем шляхом спрямування пасажирів через великі авіаційні хаби для далекомагістральних перельотів і заміни коротких авіарейсів швидкісними електропоїздами. Такий підхід не лише зменшує вуглецевий слід, але й сприяє сталому розвитку транспортної інфраструктури. На основі цих висновків можна розробити алгоритм для вибору оптимальних маршрутів із урахуванням екологічних і практичних факторів.

Алгоритм призначений для вибору оптимального транспортного маршруту між двома точками, враховуючи викиди CO_2 на пасажира, час у дорозі та зручність. Він використовує базу даних із ~4000 комерційних аеропортів і додатково ~3000 регіональних аеродромів (загалом 7000 об’єктів), а також інтегрує альтернативні види транспорту, зокрема електричні поїзди. Алгоритм оптимізує маршрути через великі аеропорти (з пасажиропотоком від 2 млн осіб), які є енергоефективнішими завдяки економії масштабу та сучасним літакам [17].

Опис вхідних даних алгоритму:

1. Початкова та кінцева точки: адреси у природньому вигляді (наприклад, «Київ, Україна» – «Париж, Франція»).

2. База аеропортів: JSON-файл із ~7000 аеропортів, що містить відповідність між 3-значним IATA-кодом і географічними координатами.

3. Вподобання користувача: сортування за мінімальними викидами, за найшвидшим маршрутом, за найдешевшим маршрутом.

Кроки алгоритму:

1. Геокодинг точок. Використовуємо Google Geocoding API для перетворення адрес у географічні координати (широта, довгота).

2. Зберігаємо координати для подальшого пошуку найближчих аеропортів.

3. Пошук найближчих аеропортів.

Етап 1: Первинний пошук серед 300 найбільших аеропортів.

Обчислюємо відстань між координатами точки та аеропорту за формулою гаверсинуса (більш точна для сферичної Землі, ніж формула Піфагора) [18]:

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{2.1}$$

де a – безрозмірна змінна ($0 \le a \le 1$),

$\Delta\varphi = \varphi_2 - \varphi_1$ – різниця широт у радіанах,

λ_1, λ_2 – довгота першої та другої точки відповідно, у радіанах,

$\Delta\lambda = \lambda_2 - \lambda_1$ – різниця довгот в радіанах,

$$c = 2 \cdot atan2\left(\sqrt{a}, \sqrt{1-a}\right) \tag{2.2}$$

де c – центральний кут між двома точками на сфері в радіанах,

a – безрозмірна змінна, $0 \le a \le 1$.

$$d = R \cdot c \tag{2.3}$$

де d – відстань між двома точками на сфері по поверхні, в кілометрах (км),

R – радіус Землі у кілометрах (км). Значення R ≈ 6371 км – середній радіус Землі,

c – центральний кут, обчислений у попередньому рівнянні (у радіанах).

Вибираємо три найближчі аеропорти для кожної точки (початкової та кінцевої).

Етап 2: Вторинний пошук серед 7000 комерційних аеропортів

Якщо в радіусі 200 км від точки немає великих аеропортів або потрібні альтернативні маршрути, розширюємо пошук до бази 7000 комерційних аеропортів.

Для прискорення використовуємо алгоритм кластеризації (наприклад, k-means або DBSCAN) для групування аеропортів за географічними координатами, що зменшує кількість обчислень відстаней.

4. Конструювання маршрутів. Формуємо комбінації авіаційних маршрутів з використанням Google Flights API, а також прямі рейси електропоїздів. Також враховуємо комбінації маршрутів наземного і авіатранспорту.

5. Оцінюємо орієнтовні рівні викидів з використанням зведених значень:

14 г CO_2/пасажиро-км для поїздів проти 150–250 г для авіарейсів) [16].

На рисунку 2.4 зображено формалізований алгоритм пошуку мультимодальних маршрутів у вигляді блок-схеми.

Перед видачею, отримані маршрути додатково фільтруємо та сортуємо з використанням фітнес-функції. Оцінюємо кожен маршрут за комбінацією критеріїв:

- вуглецевий слід (emissions);
- тривалість поїздки (travel_time);
- вартість поїздки (cost);
- зручність (кількість пересадок, n).

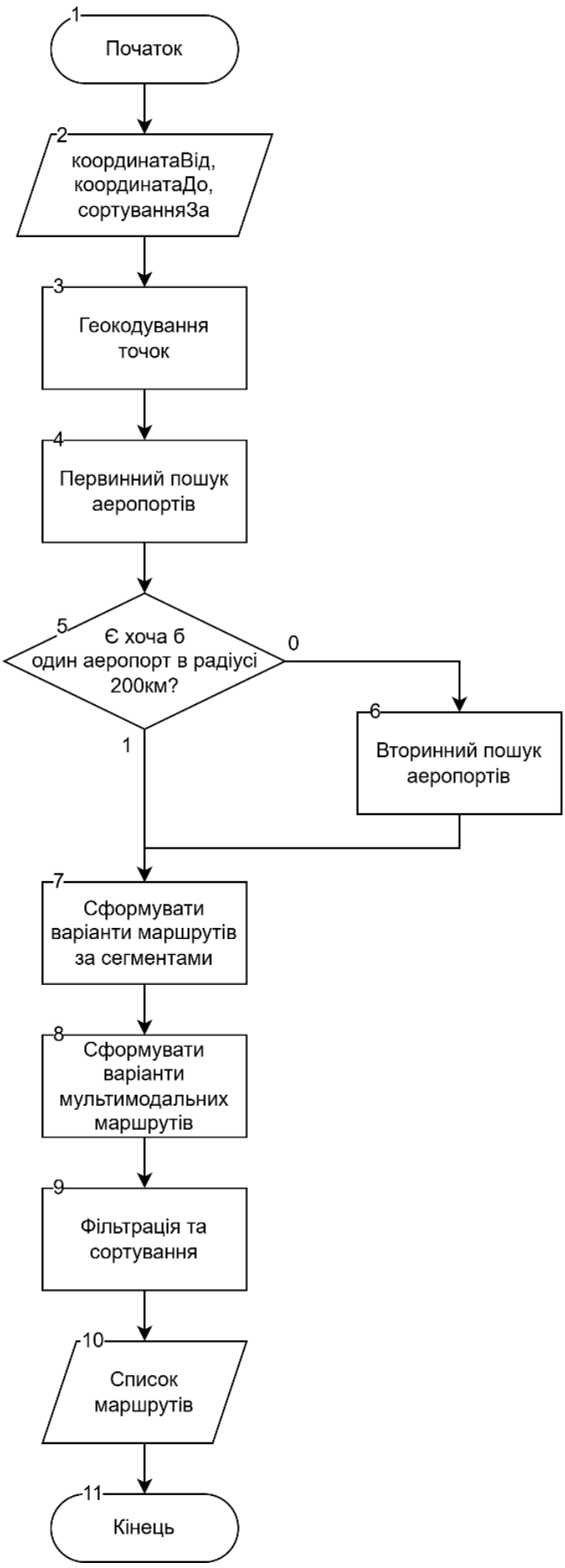

Рисунок 2.4 – Блок-схема алгоритму пошуку мультимодальних маршрутів

Формула фітнес-функції:

$$Score = w_1 \cdot emissions + w_2 \cdot travel\_time + w_3 \cdot cost + w_4 \cdot n \quad (2.4)$$

де w1, w2, w3, w4 – вагові коефіцієнти, які налаштовуються залежно від пріоритетів користувача та обраних параметрів.

Маршрути, що сильно відхиляються від середньої оцінки в гіршу сторону відсіюємо як нерелевантні (можна переглянути шляхом натискання «Показати більше»).

Теоретичні основи розробки автоматизованої системи бронювання авіаквитків базуються на оптимізації транспортних маршрутів із урахуванням екологічних і мультимодальних аспектів. Авіаційні перевезення забезпечують швидке сполучення, сприяючи туризму, бізнесу та торгівлі, але зростання обсягів підвищує викиди парникових газів, що вимагає інноваційних методів оцінки й зниження екологічного впливу. Системи, які інтегрують дані про різні види транспорту, надають користувачам інструменти для усвідомленого вибору маршрутів.

Основним принципом є обробка даних про поїздки: тривалість, викиди CO_2, тип транспорту та вартість. Авіарейси мають високий вуглецевий слід (80–250 г CO_2/пасажиро-кілометр [15]), на відміну від електропоїздів (14 г CO_2/пасажиро-кілометр [16]), що слугує базою для алгоритмів оптимізації. Великі аеропорти (понад 2 млн пасажирів на рік) знижують викиди до 80–100 г CO_2/пасажиро-кілометр завдяки сучасним літакам [17].

Важливим є залучення суб’єктивних даних через анкети, де оцінки (0–10) відображають екологічну свідомість і мотивують до сталого вибору, наприклад, перевагу поїздам над авіарейсами. Аналіз дисперсії допомагає виявити неконсистентність у поведінці для коригування рейтингу.

Теоретична база використовує гейміфікацію, стимулюючи екологічну поведінку через бали чи значки за маршрути з низьким вуглецевим слідом, що вже

працює в транспортних додатках. Інтеграція з API (Google Geocoding, SerpAPI) забезпечує актуальні геодані та розклади для мультимодальних маршрутів.

Отже, такий підхід поєднує аналіз транспортних даних, оцінку екологічного впливу та суб'єктивні фактори, формуючи метод із мотиваційним потенціалом гейміфікації. Це дозволяє оптимізувати маршрути й підтримувати сталий розвиток транспортної інфраструктури.

2.6 Розробка методу оцінки екологічного рейтингу акаунту користувача на основі історії замовлень та відповідей на опитування

Для реалізації оцінки екологічного рейтингу акаунту було запропоновано наступний алгоритм:

1. Збір і валідація даних: На першому етапі здійснюється збирання інформації про поїздки (тривалість, викиди CO_2, тип транспорту) та анкетних відповідей. Дані проходять валідацію для усунення помилок, таких як некоректні значення чи пропуски.

2. Ініціалізація вагових коефіцієнтів: Встановлюються початкові ваги $w_{trip} = 0.5$ та $w_{survey} = 0.5$, що визначають внесок об'єктивних і суб'єктивних даних у загальну оцінку.

3. Обчислення комбінованого показника: Розраховується початковий показник $CS = (w_{trip} \times TS) + (w_{survey} \times SS)$, де TS – агрегований показник поїздок, SS – нормалізована сума анкетних оцінок.

4. Ітеративна корекція: Застосовуються коригуючі множники: бонус +10% при $TS < 20$ та $SS > 40$, штраф -15% при $TS > 40$ та $SS < 20$, а також додаткове коригування -5% при $|TS - SS| > 30$, що відображає невідповідність між поведінкою та намірами.

5. Нормалізація рейтингу: Кінцевий рейтинг R приводиться до діапазону [0, 100] за формулою $R = 100 \times CS/(\max(TS, SS) + \min(TS, SS))$, забезпечуючи узгодженість результатів.

На рисунку 2.5 наведено блок-схему алгоритму обчислення екологічного рейтингу акаунту. Ця схема ілюструє послідовність етапів, починаючи від збору

даних і закінчуючи нормалізацією, що дозволяє візуально оцінити логіку процесу. Алгоритм забезпечує гнучкість завдяки можливості адаптації вагових коефіцієнтів і коректив до конкретних умов використання, а також сприяє мотивації користувачів до екологічно орієнтованих рішень через інтеграцію гейміфікаційних елементів. Такий підхід не лише оптимізує транспортні маршрути, але й підтримує глобальні цілі сталого розвитку, зокрема зменшення вуглецевого сліду авіаційної галузі.

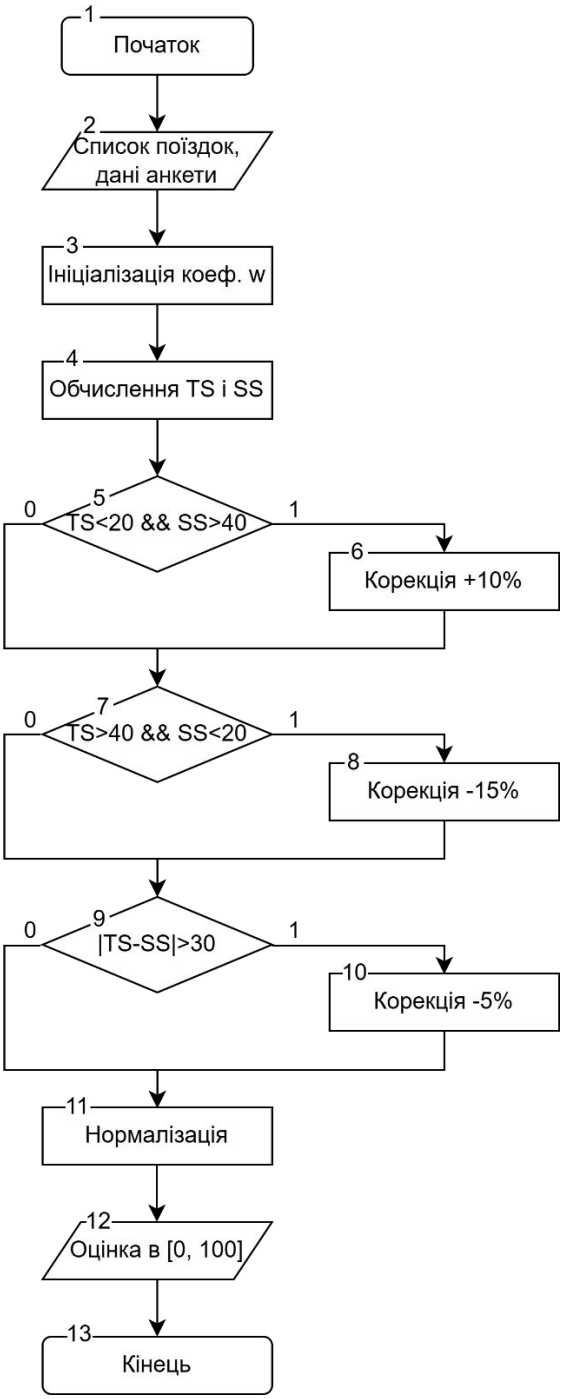

Рисунок 2.5 – Блок-схема алгоритму обчислення екологічного рейтингу акаунту

Отже, було розроблено основні алгоритми, що описують розроблювану систему.

2.7 Висновки

У розділі 2 проведено комплексний аналіз вхідних та вихідних даних системи, що забезпечує точне визначення вимог до обробки інформації про аеропорти, маршрути та транспортні альтернативи. Розроблена тришарова архітектура системи та use case діаграма чітко окреслюють взаємодію користувачів із системою, забезпечують її масштабованість, простоту реалізації та супроводу. Запропонований метод пошуку мультимодальних маршрутів забезпечує побудову оптимальних варіантів подорожі з урахуванням викидів CO_2, часу в дорозі та вартості, тоді як метод оцінки екологічного рейтингу акаунту дозволяє персоналізувати маршрутні рекомендації, інтегруючи дані про фактичну поведінку користувача та його екологічні вподобання. Ці розробки створюють надійну основу для реалізації екологічно орієнтованої транспортної системи.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ РЕАЛІЗАЦІЇ СИСТЕМИ УПРАВЛІННЯ БРОНЮВАННЯМ АВІАКВИТКІВ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Розробка сучасного веб-застосунку вимагає ретельного вибору відповідних технологічних засобів. Успішна реалізація системи бронювання мультимодальних маршрутів із врахуванням екологічних критеріїв та інтерактивної взаємодії з користувачем безпосередньо залежить від правильно підібраного інструментарію. Основними вимогами до програмної системи є забезпечення високої швидкодії, гнучкості інтерфейсу, підтримки масштабування, можливості інтеграції зі сторонніми API-сервісами, а також здатність до ефективної роботи з великими об’ємами географічних даних.

Вибір засобів для розробки повинен гарантувати не лише технічну придатність для виконання поставлених завдань, але й відповідати сучасним трендам у сфері веб-розробки, забезпечуючи при цьому комфорт розробки, надійність і підтримуваність коду в довгостроковій перспективі. Особлива увага приділяється інтеграції з зовнішніми API, такими як сервіси геокодування, пошуку рейсів та даних про транспортні вузли, що суттєво розширюють функціональні можливості застосунку.

У цьому підрозділі буде наведено аналіз варіантів вибору інструментів розробки та обґрунтування вибору оптимальних технологічних рішень для реалізації поставлених задач.

Одним із ключових рішень у процесі розробки веб-застосунку є вибір фреймворку для побудови інтерфейсу користувача. Серед популярних технологій сучасної веб-розробки можна виділити такі фреймворки як Next.js, Vue.js та Angular. Кожен із них має свої переваги та недоліки залежно від характеру завдань, які необхідно реалізувати.

Next.js є фреймворком для React, орієнтованим на серверний рендеринг, генерацію статичних сайтів та оптимізацію продуктивності [19]. Завдяки своїй

архітектурі Next.js дозволяє легко створювати SEO-оптимізовані застосунки з високою швидкістю завантаження сторінок. Він ідеально підходить для проєктів, де важливо мінімізувати час завантаження і підвищити індексацію пошуковими системами. Однак для створення складних інтерактивних клієнтських застосунків із великою кількістю взаємодій може знадобитися додаткове конфігурування.

Vue.js – це прогресивний JavaScript-фреймворк, який відомий своєю простотою у вивченні та використанні. Vue надає гнучку структуру, яка дозволяє як поступово інтегрувати фреймворк у існуючі проєкти, так і будувати повноцінні односторінкові застосунки. Vue має розвинену екосистему (Vue Router, Vuex) і є дуже популярним серед невеликих та середніх команд [20]. Його головними перевагами є легкість налаштування, хороша документація та можливість швидкої розробки прототипів. Водночас для великих корпоративних рішень Vue може вимагати ретельнішого контролю архітектури проєкту.

Angular – це комплексний фронтенд-фреймворк від Google, який пропонує рішення «все в одному» для створення масштабованих веб-застосунків. Angular базується на TypeScript, що забезпечує сильну типізацію та полегшує супровід великих проєктів. Фреймворк включає в себе модулі для маршрутизації, роботи з формами, HTTP-запитами, валідації, має потужну систему компонентів, сервісів та інверсії залежностей (Dependency Injection – DI) [21]. Angular пропонує сувору структуру коду, що робить його особливо придатним для розробки великих застосунків із багатьма функціональними модулями. Основним недоліком Angular вважається його складність на початкових етапах навчання та більший об’єм початкової конфігурації в порівнянні з Next.js або Vue.js.

Нижче наведено порівняльну таблицю 3.1 характеристик кожного фреймворку.

Отже, за результатами варіантного аналізу було визначено, що саме Angular найкраще відповідає вимогам проєкту. Завдяки суворій структурі, зручному керуванню станом та високій декларативності, цей фреймворк забезпечує надійну основу для розробки масштабованого та підтримуваного застосунку. Враховуючи також власний досвід розробки, Angular було обрано як оптимальне рішення.

Таблиця 3.1 – Порівняльна характеристика фронтенд-фреймворків

Характеристика	Next	Vue	Angular
Керування станом застосунку	+	+	+
Досвід розробки	+	-	+
Строга структура застосунку	-	-	+
Декларативність	-	+	+
Масштабованість	+/-	+/-	+
Сумарний коефіцієнт	2,5	1,5	5

В якості серверної мови програмування буде використовуватися Java та фреймворк Spring завдяки їх надійності та продуктивності. Для зберігання даних буде використовуватися популярне та функціональне рішення – база даних PostgreSQL.

3.2 Розробка модуля пошуку мультимодальних маршрутів

Модуль пошуку мультимодальних маршрутів є центральним компонентом системи управління бронюванням, що відповідає за побудову зручних, екологічно доцільних і персоналізованих маршрутів для користувача. Його архітектура реалізована відповідно до принципів чистої архітектури (Clean Architecture) та включає розмежування на окремі підсистеми, що відповідають за геокодування, пошук аеропортів, запити до зовнішніх API (наприклад, Serp API, Google Cloud APIs), побудову маршрутів і їх оцінку.

На логічному рівні модуль складається з кількох основних сервісів:

1) GeoCodingService – виконує перетворення введених користувачем назв міст у географічні координати;

2) AirportSearchService – визначає найближчі аеропорти до початкової і кінцевої точки подорожі (спочатку серед найбільших, потім серед повного набору);

3) FlightSearchService – виконує пошук прямих та непрямих авіарейсів між визначеними парами аеропортів;

4) MultimodalRouteBuilder – формує повний маршрут з додаванням наземного транспорту на початку і в кінці подорожі (наприклад, авто або поїзд до/від аеропорту);

5) RouteEvaluator – оцінює кожен варіант за кількома критеріями (ціна, час у дорозі, вуглецевий слід) та виконує ранжування результатів.

Компоненти тісно пов’язані між собою через інтерфейси сервісів у Application Layer, що дозволяє забезпечити гнучкість і простоту тестування. Також реалізовано взаємодію з Infrastructure Layer, де знаходяться реалізації зчитування баз аеропортів та доступу до зовнішніх API. Основний контролер приймає запит від фронтенду з параметрами початкового та кінцевого пункту, ініціює обробку в кілька етапів і повертає список рекомендованих маршрутів у форматі JSON.

Такий підхід дозволяє досягти високого ступеня розширюваності – наприклад, у майбутньому до модуля можна додати нові види транспорту (електробуси, водне сполучення), альтернативні API або більш складну логіку кешування результатів.

Модуль геокодування є першим етапом у побудові маршруту. Користувач вводить зрозумілі людині назви населених пунктів (наприклад, «Варшава» або «Бухарест»), які необхідно перетворити у точні географічні координати. Для цього використовується зовнішній API – зокрема, Google Geocoding API, що забезпечує високу точність та багатомовну підтримку. Отримані координати (широта та довгота) далі використовуються для просторових обчислень при пошуку найближчих транспортних вузлів.

Модуль геокодування повністю ізольований від основної логіки і реалізується як окремий сервіс GeocodingService, що дозволяє зручно замінити його або адаптувати до іншого API без зміни основної бізнес-логіки.

Реалізація модуля геокодування. Метод getCoordinates(String place) виконує завдання отримання географічних координат (широти та довготи) для заданої назви місця (наприклад, міста, адреси тощо) шляхом надсилання HTTP-запиту до сервісу Google Maps Geocoding API та обробки відповіді.

Алгоритм роботи методу можна описати поетапно:

1. На першому кроці здійснюється кодування рядка place у формат URL за допомогою стандартного кодування UTF-8. Це необхідно для коректної передачі адреси в URL-запиті. Наприклад, пробіли замінюються на %20.

2. Формується повна адреса запиту до API геокодування. Вона містить базову адресу https://maps.googleapis.com/maps/api/geocode/json, до якої додається параметр address з кодуванням місця і параметр key, який містить API-ключ, що зберігається у полі apiKey.

3. Створюється HTTP-з'єднання на основі сформованого URL. Після цього з'єднанню задається метод запиту GET, який означає отримання даних.

4. Відповідь сервера зчитується через потік введення, що обгорнутий у BufferedReader. Для зручності обробки відповідь накопичується в об'єкті StringBuilder построково. Кожна прочитана лінія додається до кінцевого рядка відповіді.

5. Для обробки отриманого JSON використовується бібліотека Jackson. Рядок відповіді перетворюється на об'єкт JsonNode. Далі з цього дерева обирається перший елемент масиву results, з нього – вкладений об'єкт geometry, а з нього – об'єкт location.

6. Із об'єкта location вилучаються значення полів lat (latitude – широта) та lng (longitude – довгота). Обидва значення перетворюються на числа з плаваючою комою.

7. На завершальному етапі створюється новий об'єкт типу Coordinates з отриманими широтою та довготою, який повертається як результат виконання методу.

У випадку виникнення будь-якої помилки (наприклад, під час мережевого запиту або парсингу JSON), буде згенерований RuntimeException з відповідним повідомленням та інформацією про місце, яке не вдалося обробити.

Після отримання координат користувача, ініціюється робота сервісу AirportNearestService, який відповідає за просторовий пошук найближчого аеропорту. В основі цього сервісу лежить стратегія двоетапного пошуку з

пріоритетом на великі міжнародні вузли, що дозволяє поєднати точність і ефективність.

Спочатку завантажується два окремих списки: перший – це підмножина найбільших аеропортів світу (приблизно 300–400 об’єктів), яка формується з файлу big-airports.json; другий – повна база з більш ніж 7000 аеропортів, що береться з airports.json. Обидва ці набори доступні через спеціалізований провайдер AirportDataProvider, що абстрагує джерела даних від логіки пошуку.

Після цього запускається метод findNearestAirport, який спершу викликає допоміжну функцію findNearestWithinRadius. Вона приймає список великих аеропортів і координати користувача та намагається знайти аеропорт у межах радіусу 200 км. Усередині цієї функції здійснюється ітерація по всіх аеропортах списку. Для кожного з них обчислюється відстань до заданих координат за допомогою формули.

У процесі обходу всіх аеропортів зберігається мінімальна знайдена відстань (змінна minDistance) і відповідний об’єкт аеропорту (nearest). Якщо жоден із великих аеропортів не виявляється в межах допустимого радіусу, відбувається другий етап – пошук повторюється, але вже серед повної бази аеропортів. Таким чином, гарантується, що буде знайдено хоч один аеропорт навіть у віддалених регіонах.

Цей підхід дозволяє досягти двох цілей: по-перше, забезпечити високу швидкодію в більшості типових випадків, коли поблизу користувача є великий вузол; по-друге, забезпечити надійність і повноту пошуку за рахунок резервної перевірки повної бази. Крім того, пріоритет великим аеропортам не випадковий – саме вони найчастіше є частиною міжнародних хабів, мають кращу інфраструктуру, часті рейси, нижчу вартість перельоту та більшу ймовірність стикувань. Таким чином, користувач отримує результат, що одночасно є як зручним, так і оптимальним з точки зору вартості та логістики.

У результаті виконання алгоритму метод повертає об’єкт аеропорту, який є найближчим до заданої точки. Цей аеропорт буде використаний як вихідна або кінцева точка при побудові мультимодального маршруту.

Обчислення геодезичної відстані реалізовано в методі calculateHaversineDistance. Вхідними даними є дві географічні точки з координатами – широтою (lat1, lat2) та довготою (lon1, lon2). Усі розрахунки виконуються у кілометрах, із урахуванням сферичної форми Землі.

На початку обчислюються дельти – різниця між широтами та довготами точок, і ці значення одразу переводяться з градусів у радіани, оскільки математичні функції sin, cos, atan2 у Java працюють саме з радіанами.

Далі обчислюється проміжна змінна a, яка представляє синусну частину формули. У неї входять квадрат синуса половини дельта-широти, а також добуток косинусів обох широт і квадрат синуса половини дельта-довготи. Весь вираз розраховується як один вираз для оптимізації продуктивності та уникнення зайвих проміжних змінних.

Після цього розраховується змінна c, яка обчислюється через функцію atan2 – вона приймає два аргументи: корінь квадратний із a та корінь квадратний з 1 – a. Це дозволяє стабільно обчислити центральний кут між точками на сфері.

На завершення отримане значення c множиться на EARTH_RADIUS_KM, яка є сталою і дорівнює середньому радіусу Землі в кілометрах (6371 км). Результатом є відстань між двома точками по поверхні Землі, яку повертає метод у вигляді double.

Модуль FlightSearchService (у поточній реалізації – клас FindFlights) відповідає за пошук доступних авіарейсів між двома заданими аеропортами. Він побудований на основі взаємодії з зовнішнім API – SerpAPI, що опосередковано надає дані з Google Flights. Такий підхід дозволяє отримати зведену інформацію про рейси, навіть за відсутності прямого доступу до офіційних API авіаперевізників, що часто вимагають окремих угод або мають обмеження на використання.

Метод findFlights приймає два параметри – об'єкти типу Airport, які містять необхідні ідентифікатори IATA. Також, як додатковий параметр до запиту може додаватися дата подорожі. Зібрана інформація включає код аеропорту відправлення та прибуття, а також службові параметри: тип пошукової системи

(engine=google_flights), мову відповіді (hl=en) і ключ доступу до API (api_key), який зберігається у змінній serpApiKey.

Для збирання повного URL використовується зручний механізм UriComponentsBuilder. Це дозволяє гнучко й безпечно формувати URL-запит із параметрами, уникаючи ручного складання рядків, що часто є джерелом помилок. Отриманий рядок запиту перетворюється у повну URL-адресу, яка готова для надсилання HTTP-запиту типу GET.

Для взаємодії з зовнішнім API використовується об'єкт RestTemplate. За допомогою методу getForObject надсилається запит і одразу очікується масив об’єктів типу FlightOption, які відповідають структурі відповіді від SerpAPI. Цей масив конвертується в список Java за допомогою Arrays.asList.

У випадку, якщо відповідь API виявилася порожньою або null, повертається порожній список (List.of()), що унеможливлює винятки при обробці результату. Це дозволяє підтримувати стабільність роботи системи навіть у разі тимчасової недоступності стороннього сервісу або неправильного запиту.

З точки зору архітектури, реалізація є незалежною від конкретного API – лише базова URL-адреса та ключ передаються через конфігурацію, що відкриває можливість легкої заміни SerpAPI на інші джерела без зміни внутрішньої логіки. Це особливо важливо в довгостроковій перспективі, коли буде можливість інтегруватися напряму з системами бронювання авіаперевізників або глобальними GDS-платформами.

З огляду на нестабільність сторонніх API, у подальшій розробці може бути реалізовано кешування відповідей – наприклад, зберігання результатів запитів у базі даних або у пам’яті (наприклад, через Redis) з обмеженим часом життя. Це дозволить не лише знизити кількість зовнішніх запитів, а й покращити загальну швидкодію системи. Такий підхід особливо доцільний у сценаріях, коли багато користувачів здійснюють пошук за одними й тими самими маршрутами в межах короткого проміжку часу.

У результаті виконання findFlights система отримує список можливих варіантів перельотів, які далі використовуються для побудови мультимодального

маршруту, зокрема у поєднанні з іншими видами транспорту та модулями оцінки екологічного впливу.

Після того, як система визначила список релевантних авіарейсів між найближчими до заданих координат аеропортами, запускається модуль побудови мультимодальних маршрутів – MultimodalRouteBuilder. Його завдання полягає у формуванні повного шляху, що поєднує авіасегмент із двома допоміжними наземними ділянками: від точки відправлення до аеропорту вильоту та від аеропорту прибуття до точки призначення.

На вхід метод buildRoutes отримує список авіарейсів (flights) та координати початкової (userStart) і кінцевої (userEnd) точок користувача. Для кожного окремого авіарейсу виконується окрема побудова повного маршруту.

Першим етапом є генерація сегмента наземного транспорту від місця перебування користувача до аеропорту вильоту. Це здійснюється через виклик groundTransportService.buildGroundSegment, який, використовує Google Maps API або збережені шаблони для оцінки маршруту (наприклад, тривалості, вартості, типу транспорту). Аналогічна операція виконується для ділянки від аеропорту прибуття до фінального пункту подорожі – обидва сегменти створюються як окремі об’єкти типу GroundSegment.

Після цього обчислюються сукупні характеристики всього маршруту:

- загальний час у дорозі (totalDuration) розраховується як сума тривалості трьох сегментів – наземного до, авіаперельоту та наземного після;

- загальна вартість подорожі (totalPrice) обчислюється шляхом додавання цін кожного з трьох сегментів;

- загальний вуглецевий слід (totalCarbon) визначається як сума викидів CO_2 від кожного етапу.

На основі цих обчислень створюється новий об’єкт RouteOption, який агрегує всю інформацію про маршрут: список сегментів, тривалість, ціну та викиди. Кожен такий маршрут додається до загального списку routeOptions.

У фіналі метод повертає повний список альтернативних мультимодальних маршрутів, які можна подати користувачеві для вибору. Цей підхід дає змогу

порівнювати не лише тривалість подорожі, а й економічну та екологічну доцільність кожного варіанту. Оскільки джерелом даних для наземних ділянок може виступати як Google Cloud Platform, так і локально збережені шаблони, система є гнучкою до варіантів реалізації, і при бажанні може працювати навіть у режимі офлайн або в умовах обмеженого API-доступу.

Завдяки цьому модулю система стає повноцінним планувальником подорожей, що враховує не лише авіасполучення, а й добирання «від дверей до дверей» – з урахуванням вартості, часу в дорозі та впливу на довкілля.

Отже, було розроблено модуль пошуку мультимодальних маршрутів, який дозволяє ефективно комбінувати різні види транспорту для оптимального планування подорожей. Модуль інтегрується з API транспортних операторів, що забезпечує актуальність даних, і включає алгоритми для порівняння маршрутів за різними критеріями, такими як час, вартість і екологічний вплив.

3.3 Висновки

Отже, у результаті для реалізації обрано сучасний технічний стек, включаючи Java з використанням Spring Boot для бекенду та PostgreSQL як базу даних, що забезпечує надійність і масштабованість системи. Angular був вибраний для фронтенду, що дозволяє створити зручний і ефективний інтерфейс користувача. Було успішно реалізовано ключові модулі системи, такі як пошук маршрутів, візуалізація екологічного рейтингу та елементи гейміфікації, що покращують досвід користувачів. Бекенд реалізує інтеграцію з різними API для отримання актуальної інформації про аеропорти та рейси, а також ефективну обробку мультимодальних маршрутів через модуль побудови маршрутів та фільтрації результатів.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування програмного забезпечення є невід'ємною частиною життєвого циклу розробки, яка забезпечує виявлення помилок, дефектів та недоліків у функціональності системи до її впровадження. У межах цього проєкту застосовуються як функціональні, так і нефункціональні техніки тестування. Основна мета – перевірити коректність роботи модулів, API-запитів, обробки даних та взаємодії користувача з інтерфейсом.

Розглянемо функціональне і нефункціональне тестування.

Функціональне тестування – це вид тестування програмного забезпечення, спрямований на перевірку того, як система виконує свої функції відповідно до заданих вимог і специфікацій [22]. Воно охоплює різні рівні тестування, включаючи модульне, інтеграційне, системне та приймальне тестування. Основна мета полягає в тому, щоб переконатися, що окремі компоненти і система в цілому працюють коректно, виконують визначені задачі та забезпечують очікувану поведінку. Застосування функціонального тестування дозволяє виявити відхилення від заданих вимог, своєчасно виявити дефекти та підвищити якість програмного продукту. Такий підхід особливо важливий у розробці складних інтегрованих систем, де навіть незначні помилки можуть впливати на кінцеву функціональність застосунку.

Функціональні техніки тестування:

Модульне тестування (Unit testing) – полягає у перевірці окремих компонентів або сервісів на ізольованість і правильність логіки. Наприклад, у бекенд-частині застосовуються тести для перевірки роботи сервісів геокодування, маршрутизації та авторизації (на рівні Java-класів). У фронтенді – тести компонентів Angular.

Інтеграційне тестування (Integration testing) – використовується для перевірки взаємодії між окремими модулями (наприклад, API сервіс пошуку

маршрутів – база даних – зовнішні API). Це дозволяє переконатися, що модулі правильно обмінюються даними й не порушують бізнес-логіку.

Системне тестування (System testing) – охоплює перевірку повної функціональності застосунку у реальних умовах. У цьому проєкті системне тестування передбачає перевірку маршруту з геокодуванням, побудовою мультимодального маршруту, збереженням результату та візуалізацією для користувача.

Тестування «чорної скриньки» (Black-box testing) – у проєкті застосовується для перевірки відповідей REST API без заглиблення в реалізацію. Наприклад, тестується правильність відповіді на запит /api/routes/search при різних вхідних параметрах.

Нефункціональне тестування зосереджене на перевірці таких характеристик системи, як продуктивність, стабільність, масштабованість, зручність у користуванні та надійність зберігання даних. Його мета – оцінити, наскільки програмне забезпечення відповідає вимогам реального використання, забезпечуючи комфортну взаємодію та стабільну роботу навіть за високих навантажень [22].

Одним із ключових напрямів є тестування продуктивності, що дозволяє визначити швидкодію системи за різних сценаріїв використання. У межах даного проєкту особлива увага приділяється перевірці часу відповіді при побудові маршрутів та зверненні до зовнішніх API (Google, SerpAPI), що дає змогу виявити потенційні вузькі місця в архітектурі.

Надійність збереження даних перевіряється шляхом тестування стабільності роботи з базою даних PostgreSQL. Зокрема, аналізується коректність запису та читання профілів користувачів, маршрутів і результатів пошуку в умовах нормального та підвищеного навантаження.

Юзабіліті-тестування охоплює оцінку зрозумілості інтерфейсу, логічності навігації та доступності основних функцій.

Проведемо перевірку стабільності та коректності роботи маршрутизуючого API /api/routes/search, який реалізує основну бізнес-логіку побудови мульти-

модальних маршрутів. У тестах використовувалися як реальні географічні пари, так і сценарії з помилками або відсутніми даними.

Першим запитом стало популярне європейське сполучення – Париж до Бухареста. Сервіс швидко геокодував обидва міста та знайшов кілька авіарейсів, деякі з них доповнювалися короткими наземними трансферами. Маршрути мали позначку рівня викидів CO_2, що дозволило наочно оцінити екологічну ефективність кожного варіанту. Пошук завершився успішно, інтерфейс правильно візуалізував усі варіанти. Результати пошуку на рисунку 4.1.

Рисунок 4.1 – Результати пошуку для маршруту Париж – Бухарест

Наступним кроком було тестування міжконтинентального маршруту. Було введено: Істанбул (Туреччина) → Маніла (Філіпіни). Очікувалась довша побудова через складність маршруту. Після кількох секунд опрацювання система запропонувала переліт із пересадкою у Дубаї або Дохі. Цікаво, що окремі маршрути включали трансфер наземним транспортом з іншого міста, де виліт був дешевший або екологічніший. Це підтвердило, що система коректно працює з мультимодальністю (див. рисунок 4.2).

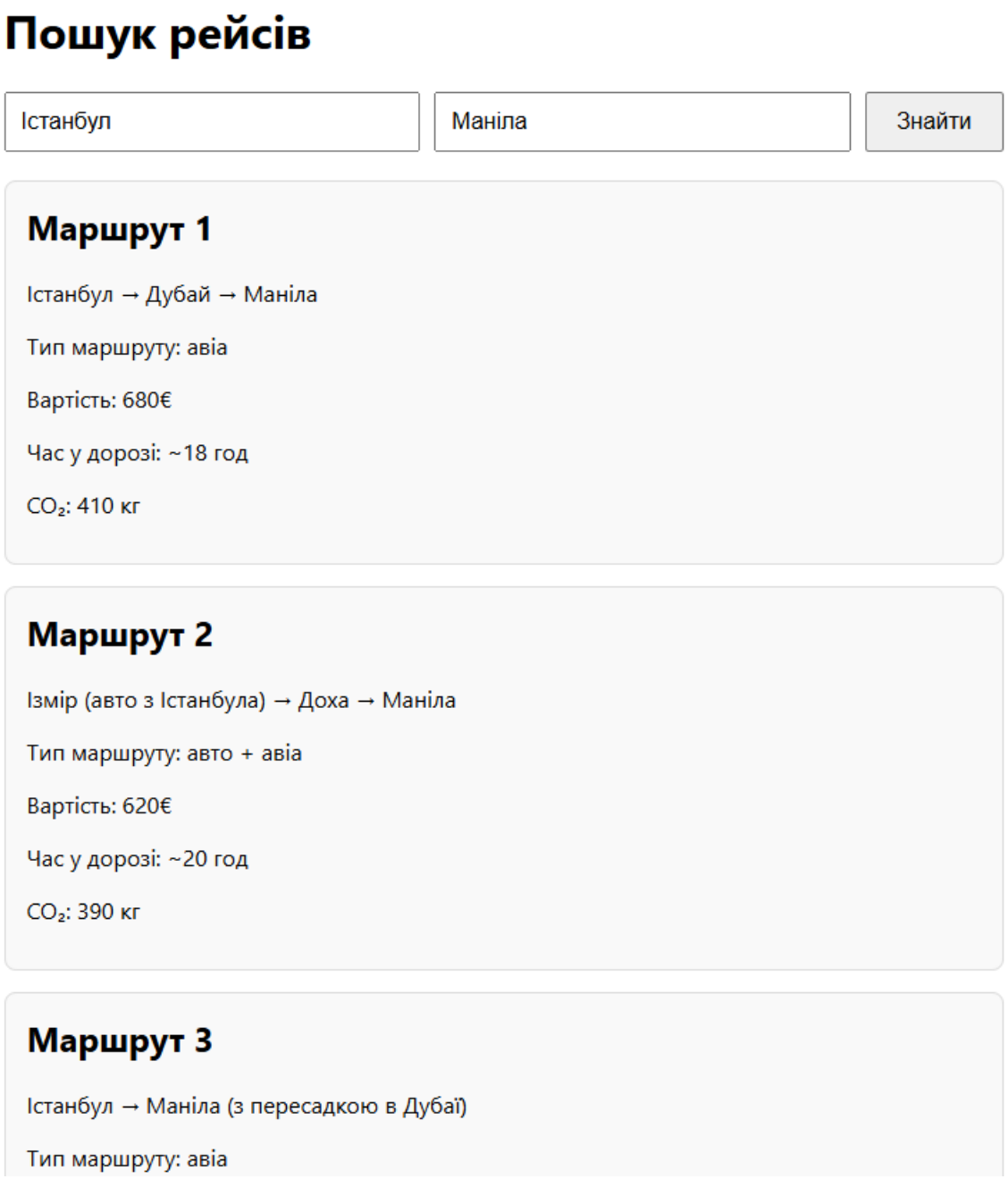

Рисунок 4.2 – Результати пошуку для маршруту Істанбул – Маніла

Аби перевірити стійкість до відсутніх даних, введено запит: Острог (Україна) → Грютвікен (Південна Джорджія і Південні Сандвічеві острови). Очевидно, що прямих маршрутів між цими точками не існує. Система спробувала знайти найближчі аеропорти, але результатом стало логічне повідомлення про те, що жодного релевантного маршруту побудувати не вдалося (див. рисунок 4.3). Жодних збоїв не виникло.

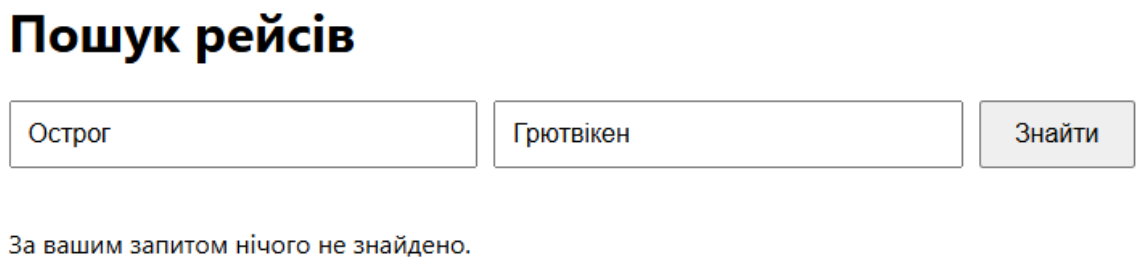

Рисунок 4.3 – Результати пошуку для маршруту Острог – Грютвікен

У реальних умовах зовнішні API можуть бути тимчасово недоступними через перевищення ліміту запитів, збої на стороні провайдера або проблеми з мережею. Щоб перевірити, наскільки система готова до таких сценаріїв, було проведено два окремих тести.

Спершу змодельовано ситуацію з недоступністю сервісу геокодування Google. Для цього штучно вимкнено API-ключ, після чого ініційовано запит на побудову маршруту. Система не зависла й не згенерувала фатальну помилку. Натомість повернулося повідомлення про недоступність сервісу.

Наступний тест стосувався відмови SerpAPI, який використовується для отримання інформації про авіарейси. При недоступності цього джерела, система також повернула повідомлення про проблему з підключенням до джерела рейсів.

Отримане спливаюче повідомлення про помилку зображено на рисунку 4.4.

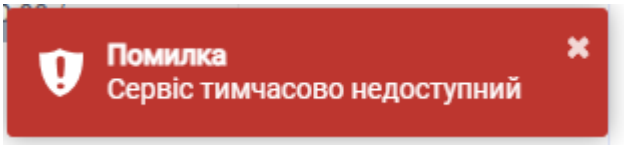

Рисунок 4.4 – Повідомлення про недоступність API-сервісів

Виконаємо перевірку механізмів автентифікації користувачів та персоналізованого відображення даних у профілі. Основна увага приділялася процесам реєстрації, входу, обробки помилок авторизації, а також завантаженню індивідуальних даних – зокрема, історії маршрутів та екологічного рейтингу.

Реєстрація нового користувача. Для початку виконано перехід на сторінку реєстрації (див. рисунок 4.5). Заповнено три стандартні поля – ім'я, email та пароль. У прикладі: ім'я «Марія», email maria@example.com, пароль – довільний, але з мінімумом валідації. Натискання кнопки «Зареєструватися» викликало POST-запит до REST API /api/auth/register. Сервер створив новий акаунт і автоматично повернув JWT-токен у відповідь, що вказує на негайний вхід у систему після реєстрації.

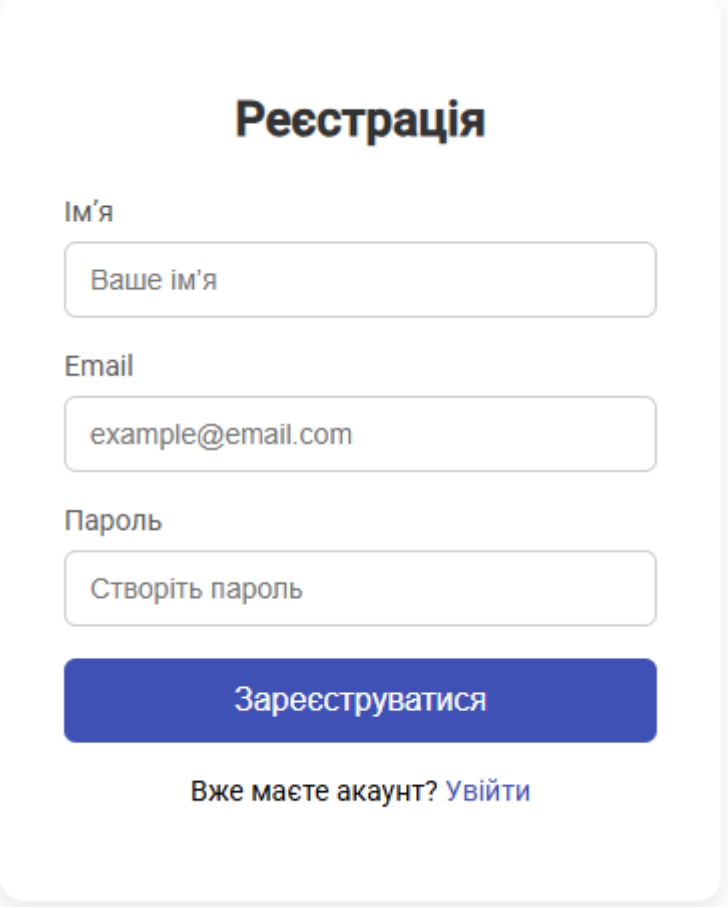

Рисунок 4.5 – Форма реєстрації

Наступним кроком була перевірка звичайного входу. Використано ті самі дані: maria@example.com + пароль. Відправлений запит /api/auth/login повернув

валідний токен, після чого відбувся редирект на сторінку профілю. У заголовку інтерфейсу з’явилося ім’я користувача та рейтинг його акаунту (див. рисунок 4.6).

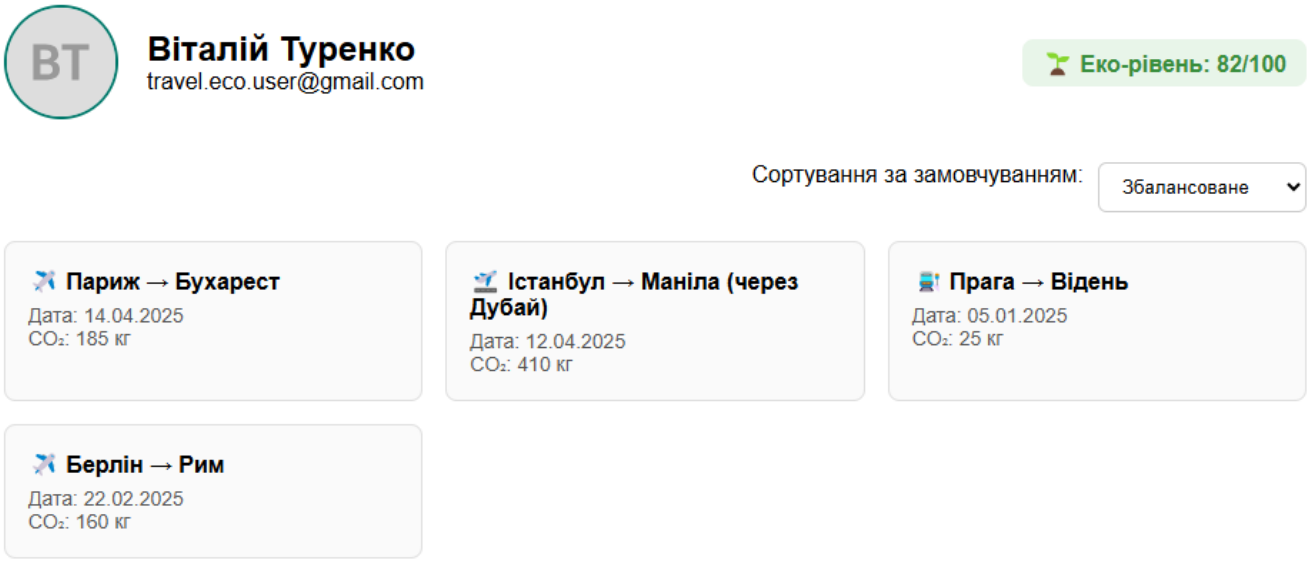

Рисунок 4.6 – Профіль після успішного входу

Для перевірки валідації помилок введено неправильний пароль до вже зареєстрованого email. У відповідь система повернула статус 401 і повідомлення: «Неправильний email або пароль». Важливо, що UI не розкрив подробиць, наприклад, який саме параметр некоректний, – що відповідає хорошій практиці безпеки (див. рисунок 4.7).

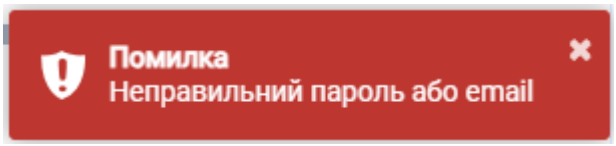

Рисунок 4.7 – Повідомлення про помилку входу

Після входу було перевірено профіль. Один з розділів – «Історія подорожей» – містив список останніх маршрутів, які були створені під час попереднього тестування (див. рисунок 4.6).

Кожен маршрут супроводжувався позначкою дати, типу транспорту та кількістю CO_2. Дані збігаються з тими, що зберігаються в базі (див. рисунок 4.8).

id [PK] inte	user_email character varying (255)	origin_city character vary	destination_c character var	transport_type character varying (50)	travel_date date	co2_em integer
1	travel.eco.user@gmail.com	Париж	Бухарест	авіа	2025-04-14	185
2	travel.eco.user@gmail.com	Істанбул	Маніла	мультимодальний	2025-04-12	410
3	travel.eco.user@gmail.com	Прага	Відень	залізниця	2025-01-05	25
4	travel.eco.user@gmail.com	Берлін	Рим	авіа	2025-02-22	160

Рисунок 4.8 – Вміст бази даних

Вгорі сторінки профілю було відображено «Екорейтинг» – у вигляді числового значення, кількості зібраних балів та рівня. Поруч розміщено візуальні значки (badge), що символізують досягнення – наприклад, «Перший зелений маршрут» або «Уникнення CO_2 понад 500 кг». Ця інформація динамічно оновлюється залежно від історії вибраних маршрутів. Відображення екологічного рейтингу показано на рисунку 4.9.

Рисунок 4.9 – Відображення екологічного рейтингу

Усі тести на автентифікацію й персоналізацію пройдено успішно. Система коректно обробляє як валідні сценарії входу, так і помилкові запити, зберігає історію маршрутів, а також надає користувачеві зрозумілу візуалізацію його екологічного внеску.

4.2 Розробка інструкції користувача

Для зручного та ефективного використання розробленого вебзастосунку EcoFlights користувачеві необхідно дотримуватись наступної послідовності дій.

Після відкриття головної сторінки користувач бачить інтерфейс пошуку маршрутів. У верхній частині сторінки розташовано два основні поля введення – «Звідки» та «Куди», у які користувач вводить назви населених пунктів початкової та кінцевої точки подорожі. Після введення даних необхідно натиснути кнопку «Шукати». Система виконує обробку запиту, визначає координати вказаних точок, знаходить найближчі аеропорти та формує список мультимодальних маршрутів.

Після завершення пошуку користувачу буде представлено список маршрутів. Кожен маршрут містить коротку інформацію про типи транспорту, приблизну тривалість, вартість і обсяг викидів CO_2. Для перегляду детальнішої інформації слід натиснути на відповідний маршрут. Це відкриє сторінку деталей маршруту, де подано сегменти поїздки (наземний транспорт, авіапереліт тощо), детальну оцінку екологічного впливу та кнопку «Забронювати».

Для того щоб забронювати маршрут, користувач повинен пройти авторизацію. Якщо обліковий запис уже існує, необхідно ввести електронну пошту та пароль на сторінці входу. У випадку відсутності акаунта – перейти до сторінки реєстрації, заповнити необхідні поля (ім’я, email, пароль) та завершити реєстрацію. Після цього користувач буде автоматично перенаправлений до головної сторінки.

У розділі профілю користувач може переглядати особисті дані, історію поїздок, змінювати налаштування сортування маршрутів (наприклад, за екологічністю, швидкістю чи вартістю), а також слідкувати за своїм екологічним рейтингом. Цей рейтинг формується на основі раніше заброньованих маршрутів із урахуванням їхнього вуглецевого сліду.

Для мотивації екологічних рішень реалізовано елементи гейміфікації – користувач отримує бали за кожен маршрут з низьким рівнем CO_2, може підвищувати рівень, заробляти значки та відкривати нові досягнення.

В ході тестування було виділено основні системні вимоги до програмної системи (див. таблицю 4.1).

Таблиця 4.1 – Основні системні вимоги до програмної системи

Параметр	Значення
Операційна система	Windows 10 / 11
Середовище виконання клієнта	Сучасний браузер (Chrome, Edge, Firefox), підтримка JavaScript ES6+
Середовище виконання сервера	Java 17+, Spring Boot
База даних	PostgreSQL 13+
Об’єм оперативної пам’яті	Не менше 4 ГБ
Процесор	2 ядра, частота не менше 2.0 ГГц
Роздільна здатність екрана	Мінімум 1280×720
Інтернет-з’єднання	Стабільне, з пропускною здатністю не менше 2 Мбіт/с

Таким чином, розроблена система є інтуїтивно зрозумілою для користувача і забезпечує повний цикл взаємодії – від пошуку маршруту до його бронювання та аналізу впливу на довкілля.

4.3 Висновки

Проведене тестування підтвердило коректність роботи ключових функцій застосунку, включно з побудовою маршрутів та обробкою помилок сторонніх сервісів. Система стабільно реагує на виняткові ситуації й забезпечує збереження даних. Розроблена текстова інструкція полегшує роботу з інтерфейсом, що покращує загальну зручність використання. Отже, застосунок готовий до подальшого використання.

ВИСНОВКИ

У межах бакалаврської кваліфікаційної роботи було розроблено систему управління бронюванням авіаквитків із підтримкою мультимодальних маршрутів та екологічного рейтингу користувача. Вебсайт було реалізовано з використанням сучасного технологічного стеку технологій: Java, Spring Boot для серверної логіки, Angular для побудови інтерфейсу користувача та PostgreSQL як системи керування базами даних.

На підготовчому етапі було проведено аналіз аналогічних систем та сервісів з бронювання, зокрема з погляду екологічності та мультимодальності. Отримані результати дозволили визначити функціональні обмеження наявних платформ і сформувати вимоги до власного програмного рішення.

У процесі реалізації обґрунтовано вибір архітектури (Clean Architecture), яка забезпечує розділення логіки застосунку на незалежні рівні, полегшуючи масштабування та тестування. Для побудови маршруту реалізовано ланцюжок модулів: геокодування початкової та кінцевої точки, пошук найближчих аеропортів, отримання даних про рейси, побудова мультимодального маршруту, фільтрація та ранжування варіантів за критеріями вартості, часу і викидів CO_2. Для візуалізації та взаємодії з користувачем створено зручний інтерфейс на Angular, що підтримує профіль користувача, історію поїздок та гейміфікаційні елементи.

У межах бакалаврської роботи було реалізовано систему бронювання авіаквитків з підтримкою мультимодальних маршрутів і врахуванням екологічних показників. Проведено аналіз існуючих рішень, спроєктовано архітектуру застосунку, розроблено метод побудови маршрутів із використанням зовнішніх API та метод оцінки екологічного рейтингу акаунту. Реалізовано персоналізовані рекомендації, систему реєстрації, перегляд історії подорожей і візуалізацію екологічної статистики.

Проведене функціональне і нефункціональне тестування показало, що система стабільно працює у більшості ключових сценаріїв. Особливу увагу було приділено перевірці пошуку маршрутів у складних напрямках, правильній обробці

помилок у разі недоступності зовнішніх сервісів (Google Geocoding, SerpAPI), роботі автентифікації та відображенню історії користувача в інтерфейсі. Усі перевірені дані збіглися з інформацією, збереженою в базі даних PostgreSQL.

Підсумки тестування підтвердили працездатність розробленого програмного забезпечення, відповідність функціональних модулів поставленим цілям, а також наявність перспектив для подальшого розширення системи – зокрема, за рахунок інтеграції нових видів транспорту, розширення механізмів персоналізації та покращення візуалізації впливу користувача на довкілля.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. IATA – Annual Review [Електронний ресурс] – Режим доступу до ресурсу: https://www.iata.org/en/publications/annual-review/ (дата звернення 11.05.2025).

2. Туренко В.Р., Романюк О.В. «Алгоритм підбору авіаквитків з урахуванням наземного транспорту та екологічного впливу» / Матеріали LIV Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ–2025): збірник доповідей [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/paper/view/24024/19897.

3. Туренко В.Р., Романюк О.В. «Метод оцінки екологічного рейтингу акаунту» / Матеріали Міжнародної науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)». [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25611.

4. Mastering the Travel Intermediaries: Origins and Future of Global Distribution Systems, Travel Management Companies, and Online Travel Agencies / Springer Nature. – 2024. – 478 с.

5. Travel Booking Software | Concur [Електронний ресурс] – Режим доступу до ресурсу: https://www.concur.com/en-us/travel-booking (дата звернення: 10.04.2025).

6. Aviation – IEA [Електронний ресурс] – Режим доступу до ресурсу: https://www.iea.org/energy-system/transport/aviation (дата звернення: 10.04.2025).

7. KAYAK – Пошук авіаквитків, готелів і авто [Електронний ресурс] – Режим доступу до ресурсу: https://www.ua.kayak.com/ (дата звернення: 10.04.2025).

8. Skyscanner – Вікіпедія [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Skyscanner (дата звернення: 10.04.2025).

9. Google Flights [Електронний ресурс] – Режим доступу до ресурсу: https://www.google.com/travel/flights (дата звернення: 10.04.2025).

10. Mastering API Architecture: Design, Operate, and Evolve API-Based Systems / James Gough, Daniel Bryant, Matthew Auburn. – 1st Edition. – 2022.

11. SerpAPI – Real-Time Google Search API [Електронний ресурс] – Режим доступу до ресурсу: https://serpapi.com/ (дата звернення: 10.04.2025).

12. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin. – 1st Edition. – Kindle Edition.

13. Multi-layered architecture – LinkedIn Pulse [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/pulse/multi-layered-architecture-milad-rezaeighale/ (дата звернення: 10.04.2025).

14. UML Summarized: Key Concepts and Diagrams for Software Engineers, Architects, and Designers / Phillip Mrzyglocki. – PDF Edition. – 2023.

15. Fact Sheet: The Growth in Greenhouse Gas Emissions from Commercial Aviation – EESI [Електронний ресурс] – Режим доступу до ресурсу: https://www.eesi.org/papers/view/fact-sheet-the-growth-in-greenhouse-gas-emissions-from-commercial-aviation (дата звернення: 10.04.2025).

16. Climate change: Aviation's growing problem – BBC News [Електронний ресурс] – Режим доступу до ресурсу: https://www.bbc.com/news/science-environment-49349566 (дата звернення: 10.04.2025).

17. Airports of Thailand – Annual Report 2022 [Електронний ресурс] – Режим доступу до ресурсу: https://www.airportthai.co.th/wp-content/uploads/2023/04/ANNUAL-REPORT-2022.pdf (дата звернення: 10.04.2025).

18. Haversine formula – Wikipedia [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Haversine_formula (дата звернення: 10.04.2025).

19. Next.js – Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://nextjs.org/ (дата звернення: 10.04.2025).

20. Vue.js – The Progressive JavaScript Framework [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/ (дата звернення: 10.04.2025).

21. Angular – Вебсайт фреймворку Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.dev/ (дата звернення: 10.04.2025).

22. Software Testing Strategies: A Testing Guide for the 2020s. – Практичне керівництво з сучасного тестування програмного забезпечення.

Додатки

Додаток А. Технічне завдання

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

ЗАТВЕРДЖУЮ
Завідувач кафедрою ПЗ
д.т.н., проф.
_____ О. Н. Романюк
«25» березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка автоматизованої системи управління бронюванням авіаквитків» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
_____ к.т.н., доцент О. В. Романюк
«24» березня 2025 р.
Виконав:
_____ студент гр. ЗПІ-21б В. Р. Туренко
«24» березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка автоматизованої системи управління бронюванням авіаквитків».

Галузь застосування – веб-системи для організації мультимодальних подорожей з урахуванням екологічних показників, використовується у сфері електронної комерції, туристичних онлайн-сервісів, транспортної логістики.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року про закріплення тем БКР.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності процесу пошуку та бронювання авіаквитків шляхом розробки нових методів та автоматизованої системи управління, яка підтримує мультимодальні маршрути та враховує екологічний слід кожного варіанту подорожі.

Призначення роботи – забезпечити зручне та екологічно орієнтоване планування подорожей для користувачів. Система дозволяє оцінювати вплив різних маршрутів на довкілля та обирати більш сталий варіант. Впроваджено елементи гейміфікації для мотивації екологічно відповідальної поведінки.

4. Вихідні дані для проведення НДР

1. Mastering the Travel Intermediaries: Origins and Future of Global Distribution Systems, Travel Management Companies, and Online Travel Agencies / Springer Nature. – 2024. – 478 с.

2. Mastering API Architecture: Design, Operate, and Evolve API-Based Systems / James Gough, Daniel Bryant, Matthew Auburn. – 1st Edition. – 2022.

3. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin. – 1st Edition. – Kindle Edition.

5. Технічні вимоги

Тип програмного забезпечення – вебзастосунок; тип архітектури – клієнт-серверна; система керування базами даних – PostgreSQL; середовище розробки –

IntelliJ IDEA, WebStorm; мови програмування – Java, TypeScript; фреймворки та бібліотеки – Spring Boot, Angular; використані API – Google Flights API, Google Geocoding API; метод обчислення відстаней – формула гаверсинуса; вхідні дані – запити користувача, географічні координати, типи транспорту, JSON-файл аеропортів; вихідні дані – згенеровані мультимодальні маршрути, оцінка вуглецевого сліду, екологічний рейтинг акаунту, персоналізовані рекомендації; архітектурний підхід – Clean Architecture; операційна система – Windows 10/11.

6. Конструктивні вимоги.

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

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

– пояснювальна записка до бакалаврської кваліфікаційної роботи;

– 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 стану розвитку систем бронювання авіаквитків та постановка задач розробки	25.03.25 – 31.03.25
2	Розробка методу побудови мультимодального маршруту з урахуванням вуглецевого сліду	01.04.25 – 07.04.25
3	Розробка методу оцінки екологічного рейтингу акаунту користувача	08.04.25 – 14.04.25
4	Розробка програмного забезпечення	15.04.25 – 12.05.25
5	Тестування програми	13.05.25 – 15.05.25
6	Оформлення матеріалів до захисту БКР	16.05.25 – 30.05.25

10. Порядок контролю та прийняття.

Всі етапи бакалаврської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б. Протокол перевірки бакалаврської кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка автоматизованої системи управління бронюванням авіаквитків.

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ: кафедра програмного забезпечення, ФІТКІ, ЗПІ-21б

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 8,7 %

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 _____ Романюк О. В., к.т.н доцент
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Туренко В. Р.
(підпис) (прізвище, ініціали)

Додаток В. Лістинг програми

Лістинг модуля AirportController.java

```
package com.vitalytyrenko.ecoflights.airport.controller;

import com.vitalytyrenko.ecoflights._models.Airport;
import com.vitalytyrenko.ecoflights.airport.service.AirportService;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/airports")
public class AirportController {

    private final AirportService service;

    public AirportController(AirportService service) {
        this.service = service;
    }

    @GetMapping("/all")
    public List<Airport> getAllAirports() {
        return service.getAllAirports();
    }

    @GetMapping("/big")
    public List<Airport> getBigAirports() {
        return service.getBigAirports();
    }
}
```

Лістинг модуля AirportService.java

```
package com.vitalytyrenko.ecoflights.airport.service;

import com.vitalytyrenko.ecoflights._models.Airport;
import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import jakarta.annotation.PostConstruct;
import lombok.Getter;
import org.springframework.stereotype.Service;
```

```
import java.io.InputStream;
import java.util.*;

@Getter
@Service
public class AirportService {

    private final List<Airport> allAirports = new ArrayList<>();
    private final List<Airport> bigAirports = new ArrayList<>();

    @PostConstruct
    public void init() {
        ObjectMapper mapper = new ObjectMapper();
        try {
            InputStream                          allInput                          =
getClass().getResourceAsStream("/static/airports.json");
            InputStream  bigInput  =  getClass().getResourceAsStream("/static/big-
airports.json");

            if (allInput != null) {
                allAirports.addAll(mapper.readValue(allInput,                    new
TypeReference<List<Airport>>() {}));
            }

            if (bigInput != null) {
                bigAirports.addAll(mapper.readValue(bigInput,                    new
TypeReference<List<Airport>>() {}));
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Лістинг модуля AirportDataProvider.java

```
package com.vitalytyrenko.ecoflights.airportdataprovider.service;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.vitalytyrenko.ecoflights._models.Airport;
```

```
import lombok.Getter;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.core.io.ClassPathResource;
import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class AirportDataProvider {

    private final ObjectMapper objectMapper;

    @Getter
    private List<Airport> allAirports;

    @Getter
    private List<Airport> bigAirports;

    @PostConstruct
    public void loadAirports() {
        try {
            InputStream allInputStream = new
ClassPathResource("static/airports.json").getInputStream();
            allAirports = objectMapper.readValue(allInputStream, new
TypeReference<>() {});

            InputStream bigInputStream = new ClassPathResource("static/big-
airports.json").getInputStream();
            bigAirports = objectMapper.readValue(bigInputStream, new
TypeReference<>() {});

            log.info("Loaded {} airports and {} big airports", allAirports.size(),
bigAirports.size());

        } catch (IOException e) {
            log.error("Failed to load airport data", e);
            throw new RuntimeException("Failed to load airport data", e);
```

```
        }
    }
}
```

Лістинг модуля AirportNearestService.java

```
package com.vitalytyrenko.ecoflights.airportnearest.service;

import com.vitalytyrenko.ecoflights._models.Coordinates;
import com.vitalytyrenko.ecoflights._models.Airport;
import
com.vitalytyrenko.ecoflights.airportdataprovider.service.AirportDataProvider;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class AirportNearestService {

    private final AirportDataProvider airportDataProvider; // Клас для завантаження
аеропортів

    private static final double EARTH_RADIUS_KM = 6371.0;

    public Airport findNearestAirport(Coordinates userCoords) {
        List<Airport> bigAirports = airportDataProvider.getBigAirports();
        List<Airport> allAirports = airportDataProvider.getAllAirports();

        Airport nearest = findNearestWithinRadius(bigAirports, userCoords);

        if (nearest == null) {
            nearest = findNearest(allAirports, userCoords);
        }

        return nearest;
    }

    private  Airport  findNearestWithinRadius(List<Airport>  airports,  Coordinates
coords) {
        final double radiusKm = 200.0;
```

```
        Airport nearest = null;
        double minDistance = Double.MAX_VALUE;

        for (Airport airport : airports) {
            double distance = calculateHaversineDistance(
                    coords.getLatitude(), coords.getLongitude(),
                    airport.getLatitude(), airport.getLongitude()
            );
            if (distance <= radiusKm && distance < minDistance) {
                minDistance = distance;
                nearest = airport;
            }
        }
        return nearest;
    }

    private Airport findNearest(List<Airport> airports, Coordinates coords) {
        Airport nearest = null;
        double minDistance = Double.MAX_VALUE;

        for (Airport airport : airports) {
            double distance = calculateHaversineDistance(
                    coords.getLatitude(), coords.getLongitude(),
                    airport.getLatitude(), airport.getLongitude()
            );
            if (distance < minDistance) {
                minDistance = distance;
                nearest = airport;
            }
        }
        return nearest;
    }

    private double calculateHaversineDistance(double lat1, double lon1, double lat2,
double lon2) {
        double dLat = Math.toRadians(lat2 - lat1);
        double dLon = Math.toRadians(lon2 - lon1);

        double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
                Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2)) *
                        Math.sin(dLon/2) * Math.sin(dLon/2);

        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
```

```
        return EARTH_RADIUS_KM * c;
    }
}
```

Лістинг модуля GeocodingService.java

```
package com.vitalytyrenko.ecoflights.geocoding.service;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.vitalytyrenko.ecoflights._models.Coordinates;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;

@Service
public class GeocodingService {

    @Value("${google.api.key}")
    private String apiKey;

    private static final String GEOCODING_API_URL =
"https://maps.googleapis.com/maps/api/geocode/json";

    public Coordinates getCoordinates(String place) {
        try {
            String encodedPlace = URLEncoder.encode(place, StandardCharsets.UTF_8);
            String url = GEOCODING_API_URL + "?address=" + encodedPlace + "&key=" +
apiKey;

            HttpURLConnection connection = (HttpURLConnection) new
URL(url).openConnection();
            connection.setRequestMethod("GET");

            StringBuilder response;
```

```
            try (BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()))) {
                response = new StringBuilder();
                String line;
                while ((line = reader.readLine()) != null) {
                    response.append(line);
                }
            }

            ObjectMapper objectMapper = new ObjectMapper();
            JsonNode root = objectMapper.readTree(response.toString());

            JsonNode location =
root.path("results").get(0).path("geometry").path("location");
            double lat = location.path("lat").asDouble();
            double lng = location.path("lng").asDouble();

            return new Coordinates(lat, lng);

        } catch (Exception e) {
            throw new RuntimeException("Failed to get coordinates for: " + place,
e);
        }
    }
}
```

Лістинг модуля RouteService.java

```
package com.vitalytyrenko.ecoflights.routes.service;

import com.vitalytyrenko.ecoflights._models.Airport;
import com.vitalytyrenko.ecoflights._models.Coordinates;
import com.vitalytyrenko.ecoflights._models.FlightOption;
import com.vitalytyrenko.ecoflights.airportnearest.service.AirportNearestService;
import com.vitalytyrenko.ecoflights.flightsearch.service.FlightSearchService;
import com.vitalytyrenko.ecoflights.geocoding.service.GeocodingService;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
```

```
public class RouteService {

    private final GeocodingService geocodingService;
    private final AirportNearestService airportNearestService;
    private final FlightSearchService flightSearchService;

    public List<FlightOption> searchRoutes(String fromPlace, String toPlace) {

        // Геокодування
        Coordinates fromCoords = geocodingService.getCoordinates(fromPlace);
        Coordinates toCoords = geocodingService.getCoordinates(toPlace);

        System.out.println("From: " + fromCoords + " To: " + toCoords);

        // Пошук найближчих аеропортів
        Airport                          nearestFromAirport                          =
airportNearestService.findNearestAirport(fromCoords);
        Airport                           nearestToAirport                           =
airportNearestService.findNearestAirport(toCoords);

        System.out.println("nearestFromAirport: " + nearestFromAirport);
        System.out.println("nearestToAirport" + nearestToAirport);

        // Пошук доступних авіарейсів між аеропортами
        List<FlightOption>                       flightOptions                       =
flightSearchService.findFlights(nearestFromAirport, nearestToAirport);

        // Побудова мультимодальних маршрутів
        List<RouteOption> multimodalRoutes = multimodalBuilder.buildRoutes(
                flightOptions, fromCoords, toCoords
        );

        // Оцінка та сортування маршрутів за критеріями
        List<RouteOption>                       rankedRoutes                        =
routeEvaluator.rankRoutes(multimodalRoutes);

        // Повертаємо відсортований список найкращих маршрутів
        return rankedRoutes;
    }
}
```

Лістинг модуля UserController.java

```
package com.vitalytyrenko.ecoflights.user;

import com.vitalytyrenko.ecoflights.auth.JwtService;
import com.vitalytyrenko.ecoflights.auth.UserRepository;
import com.vitalytyrenko.ecoflights.auth._models.Preference;
import com.vitalytyrenko.ecoflights.user._models.UserInfoResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Map;

@RestController
@RequestMapping("/user")
@RequiredArgsConstructor
public class UserController {

    private final UserRepository userRepository;
    private final JwtService jwtService;

    @GetMapping("/me")
    public                                          ResponseEntity<UserInfoResponse>
getCurrentUser(@RequestHeader("Authorization") String tokenHeader) {
        String email = jwtService.extractEmailFromHeader(tokenHeader);

        return userRepository.findByEmail(email)
                .map(user -> {
                    String preference = user.getPreference() != null
                            ? user.getPreference().toString()
                            : "OPTIMAL";
                    return                                      ResponseEntity.ok(new
UserInfoResponse(user.getFullName(), user.getEmail(), preference));
                })
                .orElse(ResponseEntity.status(HttpStatus.NOT_FOUND).build());
    }

    @PostMapping("/preference")
    public ResponseEntity<?> updatePreference(
            @RequestHeader("Authorization") String tokenHeader,
            @RequestBody Map<String, String> body
```

```
    ) {
        String email = jwtService.extractEmailFromHeader(tokenHeader);
        String pref = body.get("preference");

        return userRepository.findByEmail(email).map(user -> {
            user.setPreference(Preference.valueOf(pref.toUpperCase()));
            userRepository.save(user);
            return ResponseEntity.ok("Preference updated");
        }).orElse(ResponseEntity.status(HttpStatus.NOT_FOUND).body("User not found"));
    }
}
```

Лістинг модуля SecurityConfig.java

```
package com.vitalytyrenko.ecoflights;

import jakarta.servlet.http.HttpServletResponse;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers("**").permitAll()
                        .anyRequest().authenticated()
                )
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                )
                .exceptionHandling(e -> e
                        .authenticationEntryPoint((req, res, ex) -> res.setStatus(HttpServletResponse.SC_UNAUTHORIZED))
                )
                .csrf(csrf -> csrf.disable())
```

```
                .formLogin(config -> config.disable())
                .httpBasic(config -> config.disable())
                .build();
    }
}
```

Лістинг модуля SecurityConfig.java

```
package com.vitalytyrenko.ecoflights.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class CorsConfig {
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**") // усі шляхи
                        .allowedOrigins("*") // дозволити з будь-якого джерела
                        .allowedMethods("*") // дозволити всі методи (GET, POST,
PUT...)
                        .allowedHeaders("*"); // дозволити всі заголовки
            }
        };
    }
}
```

Лістинг модуля AuthController.java

```
package com.vitalytyrenko.ecoflights.auth;

import com.vitalytyrenko.ecoflights.auth._models.LoginRequest;
import com.vitalytyrenko.ecoflights.auth._models.LoginResponse;
import com.vitalytyrenko.ecoflights.auth._models.RegisterRequest;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/auth")
@RequiredArgsConstructor
public class AuthController {

    private final AuthService authService;

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody RegisterRequest request) {
        try {
            authService.register(request);
            return ResponseEntity.ok("Користувача зареєстровано успішно");
        } catch (Exception e) {
            return ResponseEntity.badRequest().body("Помилка реєстрації: " +
e.getMessage());
        }
    }

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest request) {
        try {
            String token = authService.login(request);
            return ResponseEntity.ok(new LoginResponse(token));
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Невірний
логін або пароль");
        }
    }
}
```

Лістинг модуля AuthSecurityConfig.java

```
package com.vitalytyrenko.ecoflights.auth;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
```

```
public class AuthSecurityConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Лістинг модуля app-routing.module.ts

```
import {NgModule} from '@angular/core';
import {RouterModule, Routes} from '@angular/router';
import {RouteSearchComponent} from './route-search/route-search.component';
import {LandingComponent} from './landing/landing.component';
import {RouteDetailsComponent} from './mock/route-details/route-details.component';
import {ProfileComponent} from './mock/profile/profile.component';
import {AchievementsComponent} from './mock/achievements/achievements.component';
import {LoginComponent} from './mock/login/login.component';
import {RegisterComponent} from './mock/register/register.component';

const routes: Routes = [
    {path: '', component: LandingComponent},
    {path: 'search-routes', component: RouteSearchComponent},
    {path: 'landing', component: LandingComponent},
    {path: 'routes-details', component: RouteDetailsComponent},
    {path: 'profile', component: ProfileComponent},
    {path: 'achievements', component: AchievementsComponent},
    {path: 'login', component: LoginComponent},
    {path: 'register', component: RegisterComponent}
];

@NgModule({
    imports: [RouterModule.forRoot(routes)],
    exports: [RouterModule]
})
export class AppRoutingModule {
}
```

Лістинг модуля route-search.component.ts

```
import {Component} from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {environment} from '../../environments/environment';
```

```
import {RouteDetailsDialogComponent} from '../route-details-dialog/route-details-dialog.component';
import {MatDialog} from '@angular/material/dialog';

@Component({
    selector: 'app-route-search',
    standalone: false,
    templateUrl: './route-search.component.html',
    styleUrl: './route-search.component.css'
})
export class RouteSearchComponent {
    from = '';
    to = '';
    loading = false;
    routes: any[] = [];

    constructor(
        private http: HttpClient,
        private dialog: MatDialog
    ) {
    }

    getRouteSearch() {
        return this.routes.length;
    }

    search() {
        this.loading = true;
        this.routes = [];

        this.http.get<any[]>(`${environment.backendUrl}/api/routes/search`, {
            params: {
                from: this.from,
                to: this.to
            }
        }).subscribe({
            next: (res: any[]) => {
                this.routes = res;
                this.loading = false;
            },
            error: () => this.loading = false
        });
    }
```

```
    isLoggedIn(): boolean {
        return !!localStorage.getItem('accessToken');
    }

    onRouteClick(route: any) {
        this.dialog.open(RouteDetailsDialogComponent, {
            data: route,
            width: '500px',
            panelClass: 'custom-dialog'
        });
    }
}
```

Лістинг модуля route-search.component.html

```
<div class="page-container">
    <div class="header">
        <h1>Пошук рейсів</h1>
        <div class="user-access">
            <button    *ngIf="!isLoggedIn()"    routerLink="/login"    class="login-
btn">Увійти</button>
            <div    *ngIf="isLoggedIn()"    routerLink="/profile"    class="avatar-
circle">ВР</div>
        </div>
    </div>

    <div class="search-form">
        <input type="text" placeholder="Звідки" [(ngModel)]="from" />
        <input type="text" placeholder="Куди" [(ngModel)]="to" />
        <button    (click)="search()"    [disabled]="loading    ||    !from    ||
!to">Знайти</button>
    </div>

    <div class="routes-list" *ngIf="getRouteSearch() > 0">
        <div *ngFor="let route of routes" class="route-row">
            <div class="route-info">
                <h3>{{ route.fromAirport.city }} → {{ route.toAirport.city }}</h3>
                <p><strong>Вартість:</strong> {{ route.price }} €</p>
                <p><strong>Тривалість:</strong> {{ route.durationMinutes }} хв</p>
                <p><strong>CO₂:</strong> {{ route.carbonFootprintKg }} кг</p>
            </div>
```

```
            <button                                                    class="view-button"
(click)="onRouteClick(route)">Переглянути</button>
        </div>
    </div>
</div>
```

Лістинг модуля route-search.component.html

```
.page-container {
    max-width: 800px;
    margin: 0 auto;
    padding: 24px;
    font-family: 'Segoe UI', sans-serif;
}

.header {
    display: flex;
    justify-content: space-between;
    align-items: center;
    margin-bottom: 24px;
}

.user-access {
    display: flex;
    align-items: center;
}

.login-btn {
    padding: 8px 16px;
    background-color: #3f51b5;
    color: white;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    font-weight: 500;
}

.login-btn:hover {
    background-color: #303f9f;
}

.avatar-circle {
    cursor: pointer;
```

```
    width: 40px;
    height: 40px;
    background-color: #2196f3;
    color: white;
    border-radius: 50%;
    display: flex;
    align-items: center;
    justify-content: center;
    font-weight: bold;
    font-size: 16px;
}

.search-form {
    display: flex;
    gap: 12px;
    margin-bottom: 24px;
}

.search-form input {
    flex: 1;
    padding: 10px 12px;
    border: 1px solid #ccc;
    border-radius: 6px;
    font-size: 16px;
}

.search-form button {
    padding: 10px 16px;
    background-color: #4caf50;
    color: white;
    border: none;
    border-radius: 6px;
    font-weight: 500;
    cursor: pointer;
}

.search-form button[disabled] {
    background-color: #a5d6a7;
    cursor: not-allowed;
}

.routes-list {
    display: flex;
```

```
    flex-direction: column;
    gap: 12px;
    margin-top: 24px;
}

.route-row {
    display: flex;
    justify-content: space-between;
    align-items: center;
    background: #f9f9f9;
    border-radius: 12px;
    padding: 16px 24px;
    transition: background-color 0.2s, box-shadow 0.2s;
}

.route-row:hover {
    background-color: #eef8ff;
    box-shadow: 0 4px 12px rgba(0,0,0,0.1);
}

.route-info h3 {
    margin: 0 0 4px 0;
    font-size: 20px;
}

.route-info p {
    margin: 2px 0;
    font-size: 14px;
}

.view-button {
    cursor: pointer;
    padding: 8px 16px;
    background-color: #1976d2;
    color: white;
    border: none;
    border-radius: 8px;
    font-size: 14px;
    transition: background-color 0.2s;
}

.view-button:hover {
    background-color: #125aa3;
```

```
}
```

Лістинг модуля login.component.ts

```
import {Component, OnInit} from '@angular/core';
import {FormBuilder, FormGroup, Validators} from '@angular/forms';
import {AuthService} from '../../../services/auth.service';
import {Router} from '@angular/router';

@Component({
    selector: 'app-login',
    standalone: false,
    templateUrl: './login.component.html',
    styleUrl: './auth.component.css'
})
export class LoginComponent implements OnInit {
    form: FormGroup | undefined;

    constructor(
        private formBuilder: FormBuilder,
        private authService: AuthService,
        private router: Router
    ) {
    }

    ngOnInit() {
        this.form = this.formBuilder.group({
            email: ['', Validators.required],
            password: ['', Validators.required]
        });
    }

    submitForm(): void {
        this.authService.login(this.form!.getRawValue()).subscribe(() => {
            this.router.navigate(['search-routes']).then();
        });
    }
}
```

Лістинг модуля login.component.html

```
<div class="auth-root">
    <div class="auth-container" [formGroup]="form!">
        <h1>Вхід до системи</h1>
```

```
        <form class="auth-form" [formGroup]="form!">
            <label for="login-email">Email</label>
            <input type="email" id="login-email" placeholder="example@email.com"
formControlName="email" />

            <label for="login-password">Пароль</label>
            <input type="password" id="login-password" placeholder="Ваш пароль"
formControlName="password" />

            <button type="submit" (click)="form?.valid ? submitForm() :
form?.markAllAsTouched()">Увійти</button>
        </form>

        <p class="redirect-text">
            Ще не маєте акаунта? <a href="/register">Зареєструватися</a>
        </p>
    </div>

</div>
```

Лістинг модуля auth.component.css

```
body {
    font-family: 'Segoe UI', sans-serif;
    background-color: #f5f7fa;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.auth-root {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    width: 100%;
}

.auth-container {
```

```
    background-color: #ffffff;
    padding: 32px;
    border-radius: 12px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
    width: 100%;
    max-width: 360px;
    box-sizing: border-box;
}

.auth-container h1 {
    text-align: center;
    margin-bottom: 24px;
    font-size: 24px;
    color: #333333;
}

.auth-form {
    display: flex;
    flex-direction: column;
}

.auth-form label {
    margin-bottom: 6px;
    font-size: 14px;
    color: #555;
}

.auth-form input {
    padding: 10px 12px;
    margin-bottom: 16px;
    border: 1px solid #ccc;
    border-radius: 6px;
    font-size: 14px;
    transition: border-color 0.3s;
}

.auth-form input:focus {
    border-color: #3f51b5;
    outline: none;
}

.auth-form button {
    background-color: #3f51b5;
```

```
    color: white;
    padding: 12px;
    border: none;
    border-radius: 6px;
    font-size: 16px;
    cursor: pointer;
    transition: background-color 0.3s;
}

.auth-form button:hover {
    background-color: #303f9f;
}

.redirect-text {
    margin-top: 16px;
    text-align: center;
    font-size: 14px;
}

.redirect-text a {
    color: #3f51b5;
    text-decoration: none;
}

.redirect-text a:hover {
    text-decoration: underline;
}
```

Лістинг модуля auth.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { BehaviorSubject, Observable, tap } from 'rxjs';

interface LoginRequest {
    email: string;
    password: string;
}

interface RegisterRequest {
    email: string;
    password: string;
    fullName: string;
```

```
}

interface LoginResponse {
    token: string;
}

@Injectable({
    providedIn: 'root'
})
export class AuthService {

    private readonly API_URL = '/api/auth';
    private loggedIn$ = new BehaviorSubject<boolean>(this.hasToken());

    constructor(private http: HttpClient) {}

    login(data: LoginRequest): Observable<LoginResponse> {
        return this.http.post<LoginResponse>(`${this.API_URL}/login`, data, {
            headers: { 'Content-Type': 'application/json' }
        }).pipe(
            tap(response => {
                localStorage.setItem('accessToken', response.token);
                this.loggedIn$.next(true);
            })
        );
    }

    register(data: RegisterRequest): Observable<any> {
        return this.http.post(`${this.API_URL}/register`, data, {
            headers: { 'Content-Type': 'application/json' }
        });
    }

    logout(): void {
        localStorage.removeItem('accessToken');
        this.loggedIn$.next(false);
    }

    isLoggedIn(): Observable<boolean> {
        return this.loggedIn$.asObservable();
    }

    getToken(): string | null {
```

```
        return localStorage.getItem('accessToken');
    }

    private hasToken(): boolean {
        return !!localStorage.getItem('accessToken');
    }
}
```

Лістинг модуля auth.service.ts

```
import {Injectable} from '@angular/core';
import {HttpErrorResponse, HttpEvent, HttpHandler, HttpInterceptor, HttpRequest}
from '@angular/common/http';
import {Observable, throwError} from 'rxjs';
import {catchError, switchMap} from 'rxjs/operators';
import {AmadeusService} from '../services/amadeus.service';

@Injectable({
    providedIn: 'root'
})
export class TokenInterceptor implements HttpInterceptor {

    constructor(private amadeusService: AmadeusService) {}

    intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
        const accessToken: string | null = localStorage.getItem('accessToken');
        const isPublicApi = request.headers.keys()?.includes('Public');

        console.log(accessToken);
        console.log(isPublicApi)
        if (isPublicApi) {
            return next.handle(request);
        }

        if (accessToken) {
            request = this.addAuthorizationHeader(request, accessToken);
            return next.handle(request).pipe(
                catchError(err => this.handleAuthError(err, request, next))
            );
        } else {
            return this.amadeusService.authorize().pipe(
                switchMap(token => {
```

```
                    localStorage.setItem('accessToken', token);
                    const authReq = this.addAuthorizationHeader(request, token);
                    return next.handle(authReq);
                })
            );
        }
    }

    private addAuthorizationHeader(request: HttpRequest<any>, token: string): HttpRequest<any> {
        return request.clone({
            setHeaders: {
                Authorization: `Bearer ${token}`
            }
        });
    }

    private handleAuthError(error: any, request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        if (error instanceof HttpErrorResponse && error.status === 401) {
            return this.amadeusService.authorize().pipe(
                switchMap(newToken => {
                    localStorage.setItem('accessToken', newToken);
                    const authReq = this.addAuthorizationHeader(request, newToken);
                    return next.handle(authReq);
                })
            );
        }
        return throwError(() => error);
    }
}
```

Додаток Г. Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА АВТОМАТИЗОВАНОЇ СИСТЕМИ УПРАВЛІННЯ БРОНЮВАННЯМ АВІАКВИТКІВ

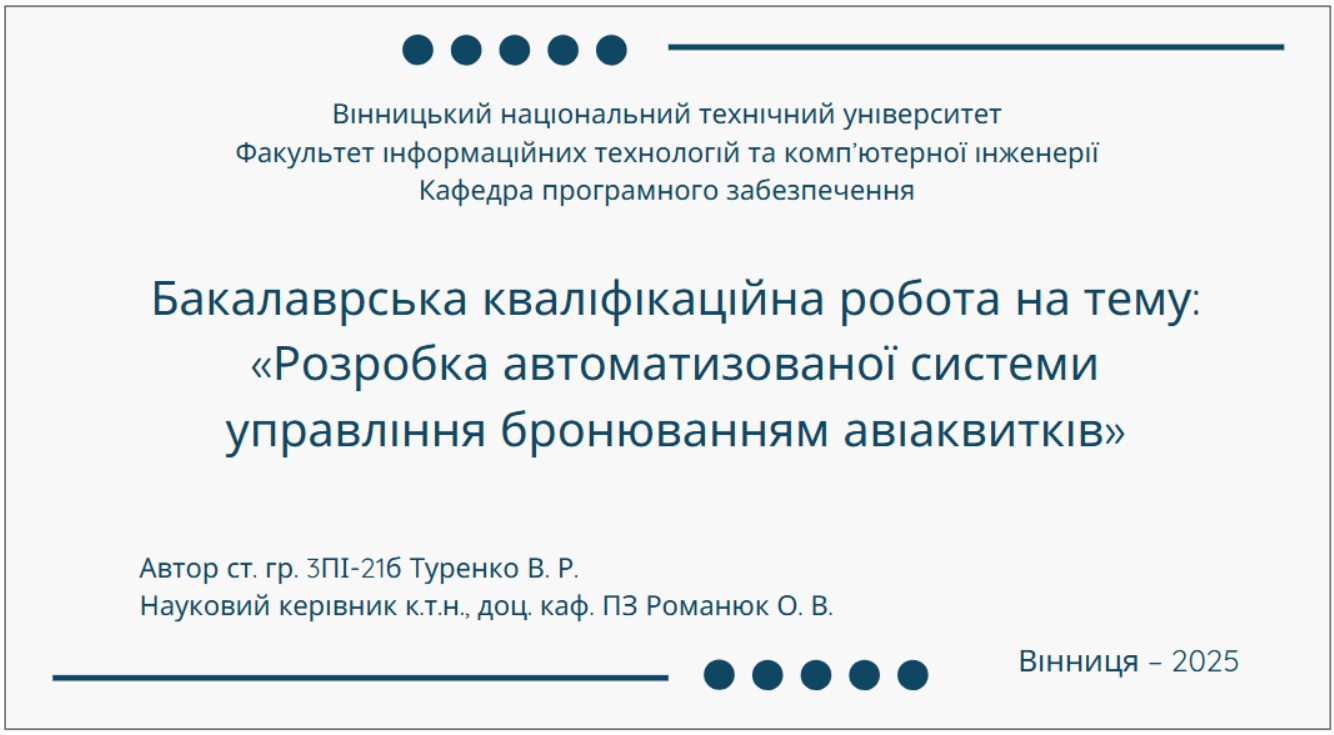

Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Актуальність розробки

Рисунок Г.3 – Мета дослідження

Рисунок Г.4 – Об’єкт та предмет дослідження

Рисунок Г.5 – Постановка задачі

Рисунок Г.6 – Новизна

Рисунок Г.7 – Практична цінність

Рисунок Г.8 – Існуючі аналоги

Аналіз аналогів

Функція	Kayak	Scyscanner	Google Flights	Власна реалізація
Мультимодальні маршрути	+	-	-	+
Альтернативні види транспорту	+	-	+	+
Гейміфікація	-	-	-	+
Врахування CO_2	-	+	+	+
Персоналізація рекомендацій	+	-	+	+
Сумарний коефіцієнт	3	1	3	5

Рисунок Г.9 – Аналіз аналогів

Чотиришарова архітектура

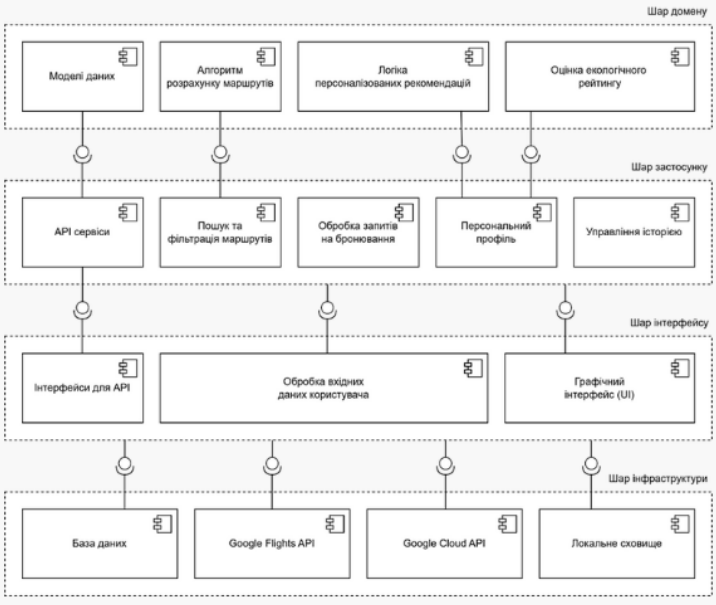

Рисунок Г.10 – Чотиришарова архітектура

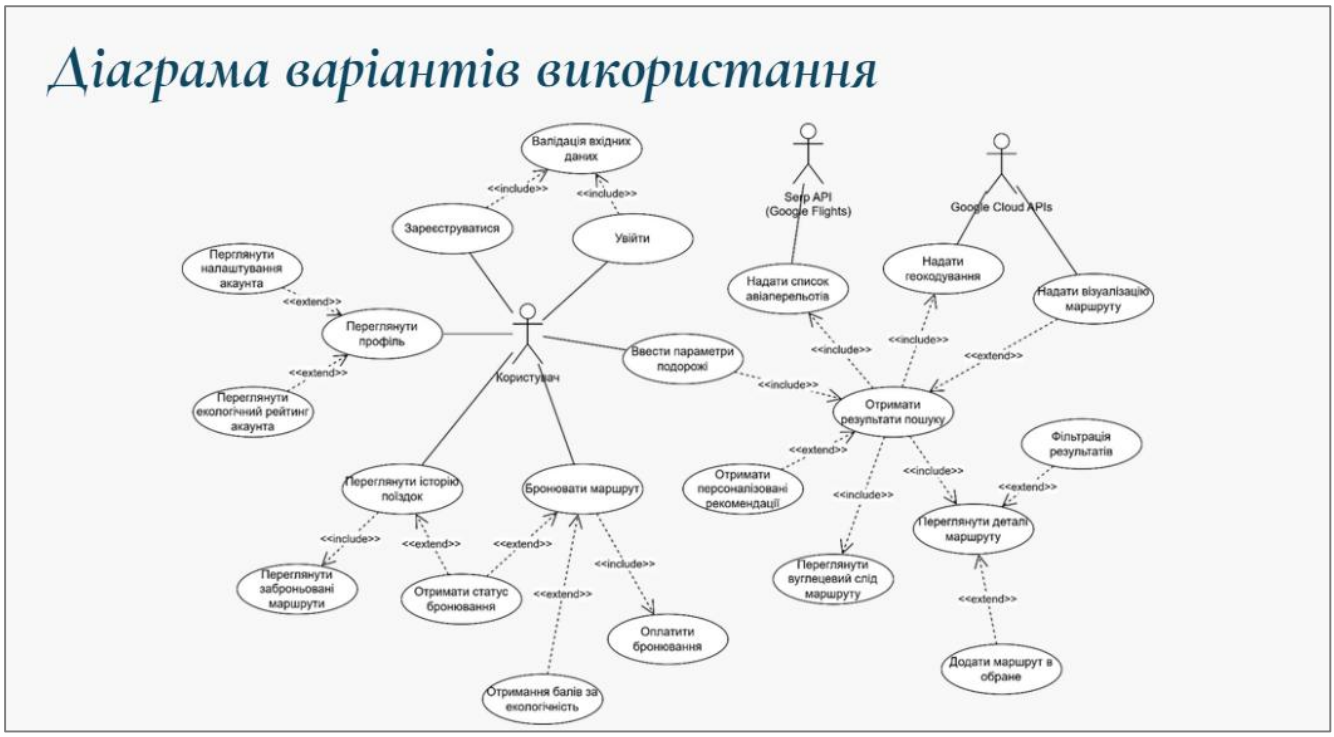

Рисунок Г.11 – Діаграма варіантів використання

Рисунок Г.12 – Загальний алгоритм роботи програми

Метод пошуку мультимодальних маршрутів

Використовується формула гаверсинуса для обчислення відстані між двома географічними точками з урахуванням сферичної форми Землі

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2 \cdot atan2\left(\sqrt{a}, \sqrt{1-a}\right) \qquad d = R \cdot c$$

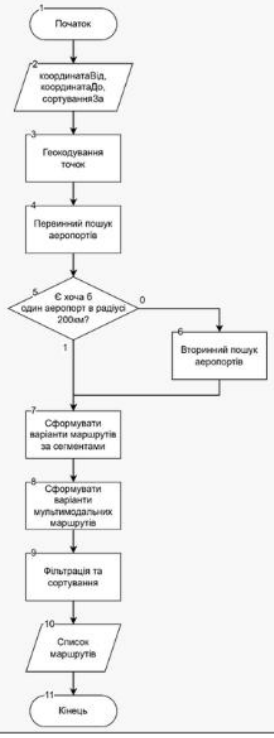

Рисунок Г.13 – Метод пошуку мультимодальних маршрутів

Метод оцінки екологічного рейтингу акаунту

1. Обчислюємо два параметра:
   - TS - об'єктивний коефіцієнт
   - SS - суб'єктивний коефіцієнт
2. Отримуємо загальну оцінку складанням TS і SS
3. Корегуємо отриману оцінку:
   - якщо TS < 20 && SS > 40 - корекція +10%
   - якщо TS > 40 && SS < 20 - корекція -15%
   - якщо |TS-SS| > 30 - корекція -5%
4. Виконуємо нормалізацію до 100%

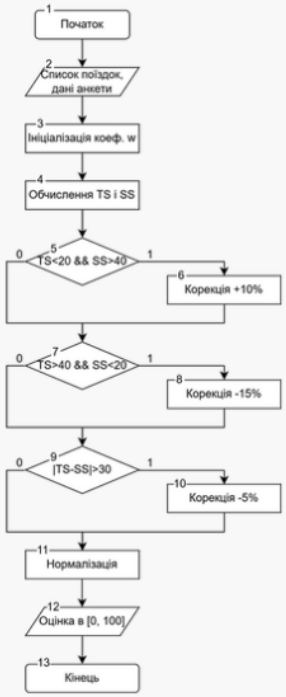

Рисунок Г.14 – Метод оцінки екологічного рейтингу акаунту

Рисунок Г.15 – Використані технології

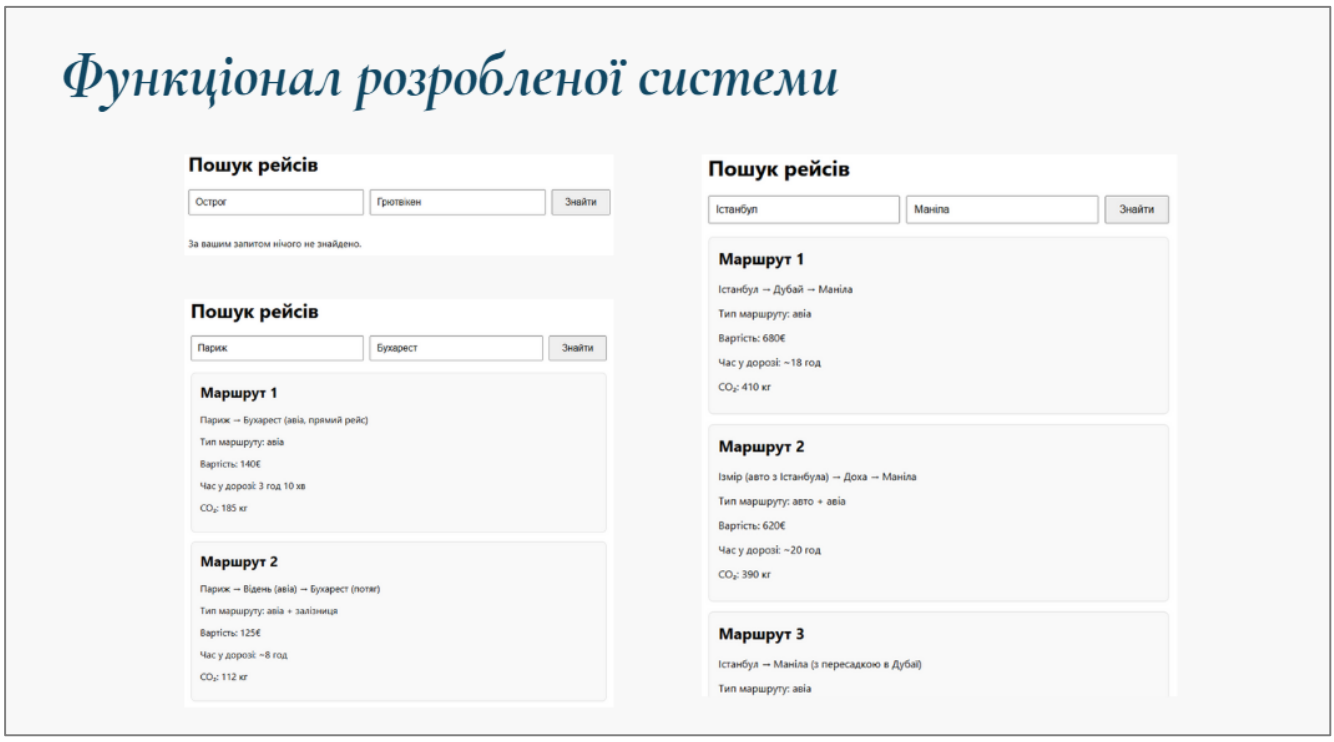

Рисунок Г.16 – Функціонал розробленої системи

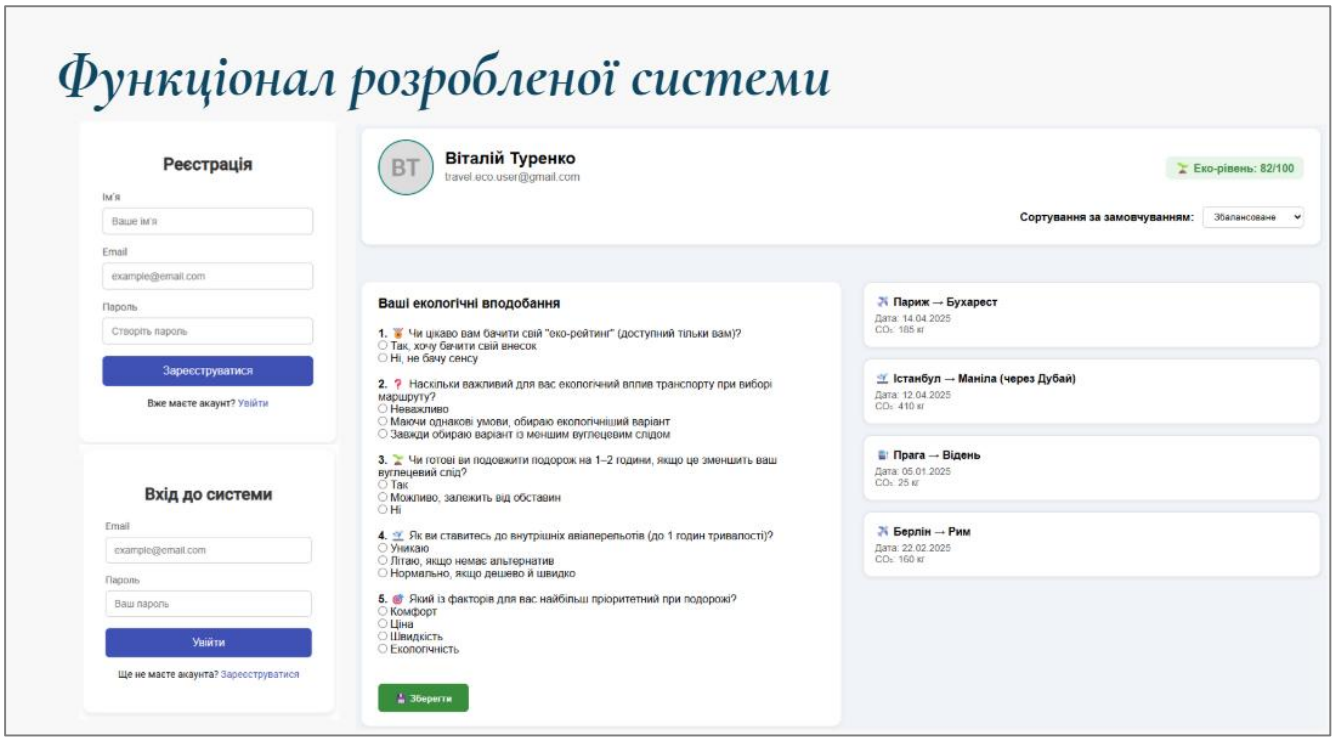

Рисунок Г.17 – Функціонал розробленої системи

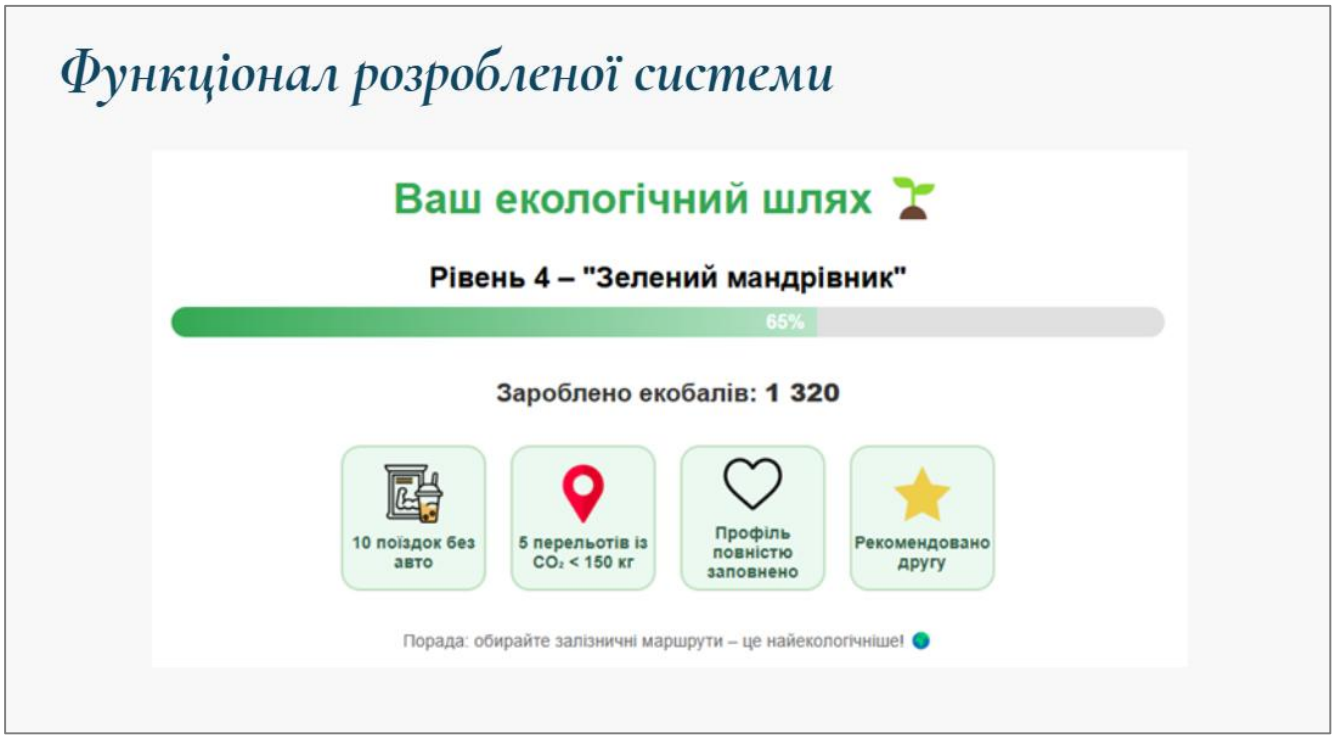

Рисунок Г.18 – Функціонал розробленої системи

Рисунок Г.19 – Апробація та публікація результатів роботи

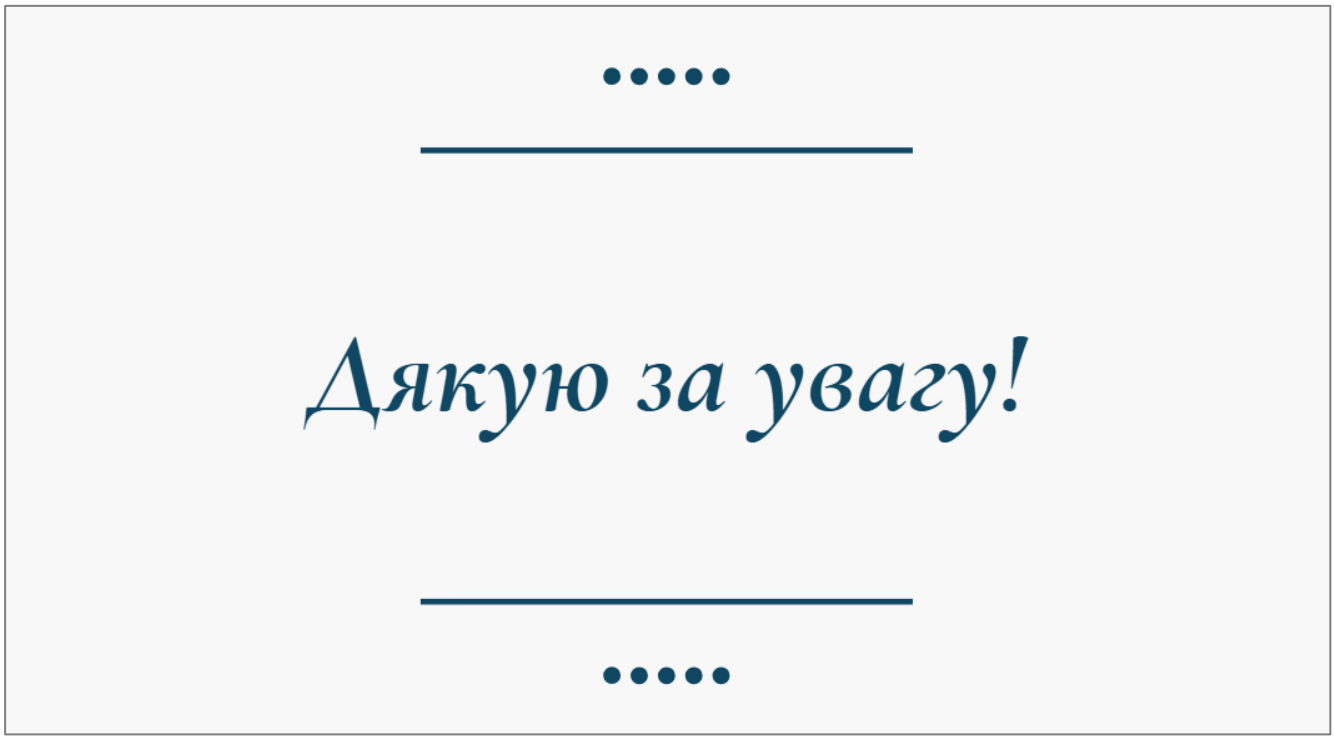

Рисунок Г.20 – Фінальний слайд