ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмних засобів для збору інформації для букінгу авіабілетів»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Нікіта ЛІБЕРМАН
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Нікіта ЛІБЕРМАН

Керівник: Наталя АНДРУСЕВИЧ
старший викладач

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Ірина ЗАМРІЙ
« ____» _____2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ліберману Нікіті Михайловичу

1. Тема кваліфікаційної роботи: «Розробка програмних засобів для збору інформації для букінгу авіабілетів»
керівник кваліфікаційної роботи старший викладач ІПЗ Наталя АНДРУСЕВИЧ,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки веб-застосунків, принципи побудови клієнт-серверної архітектури, опис технологій Spring Boot, PostgreSQL та Amadeus API, а також документація до фреймворків Spring та бібліотек для взаємодії з REST API.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд веб-застосунку для бронювання авіаквитків та аналіз існуючих аналогів, визначення вимог до програмного забезпечення.

2. Огляд та аналіз засобів реалізації веб-застосунку для бронювання авіаквитків.
3. Проєктування архітектури веб-застосунку для бронювання авіаквитків.
4. Програмна реалізація та тестування веб-застосунку для бронювання авіаквитків.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Архітектура застосунку
5. Діаграма класів основної логіки
6. Діаграма класів авторизації
7. Діаграма варіантів використання
8. Структура бази даних
9. Екранні форми.
10. Апробація результатів дослідження

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд та аналіз застосунку для бронювання авіаквитків	14.03-15.03.2025	
4	Огляд та аналіз засобів реалізації вебзастосунку для бронювання авіаквитків	16.03-21.03.2025	
5	Проєктування архітектури застосунку для бронювання авіаквитків	22.03-03.04.2025	
6	Програмна реалізація та тестування застосунку для бронювання авіаквитків	04.04-25.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Нікіта ЛІБЕРМАН
(підпис)

Керівник
кваліфікаційної роботи _____ Наталя АНДРУСЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 49 стор., 2 табл., 32 рис., 20 джерел.

Мета роботи – автоматизація бронювання авіаквитків за допомогою вебзастосунку.

Об'єкт дослідження – процес бронювання авіаквитків у системах онлайн-бронювання

Предмет дослідження – методи та засоби автоматизації бронювання авіаквитків за допомогою вебзастосунку.

Короткий зміст роботи: В роботі проаналізовано предметну галузь онлайн-бронювання квитків та методи реалізації відповідних програмних засобів. Проведено огляд існуючих сервісів з бронювання авіаквитків, таких як Skyscanner, Kiwi, Google Flights. Розроблено веб-застосунок, що дозволяє користувачам здійснювати реєстрацію, авторизацію, перегляд та пошук авіаквитків (у тому числі без авторизації), а також здійснювати бронювання з підтвердженим обліковим записом. Реалізовано систему підтвердження акаунта, модуль підтримки (з можливістю написання та обробки звернень), профіль користувача з функціональністю перегляду, скасування та передачі квитків іншим користувачам. Для адміністраторів реалізовано окрему панель керування з можливістю перегляду списку користувачів, блокування, зміни ролі, пошуку за електронною поштою, перегляду бронювань, а також очищення кешу запитів. Система кешування дозволяє зберігати результати однакових пошуків у базі даних для зменшення часу відповіді. Проведено тестування функціональності веб-застосунку. Для реалізації використано Spring Boot (Spring Web, Security, Data JPA), PostgreSQL для зберігання даних, та Amadeus API для отримання інформації про авіарейси.

Сферою використання застосунку є забезпечення користувачів швидким та зручним інтерфейсом для пошуку, бронювання та керування авіаквитками через інтернет.

КЛЮЧОВІ СЛОВА**:** JAVA, SPRING BOOT, POSTGRESQL, AMADEUS API, JWT, ВЕБЗАСТОСУНОК, REST API, БРОНЮВАННЯ, СИСТЕМА ПІДТРИМКИ, АДМІН-ПАНЕЛЬ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

API – Application Programming Interface

JWT – JSON Web Token

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

SQL – Structured Query Language

ORM – Object-Relational Mapping

IDE – Integrated Development Environment

IATA – International Air Transport Association

SPA – Single Page Application

UI – User Interface

DB – Database

CRUD – Create, Read, Update, Delete

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

REST – Representational State Transfer

ВСТУП

Актуальність: у сучасному світі онлайн-бронювання авіаквитків стало не лише популярним трендом, а й невіддільною складовою повсякденного життя багатьох користувачів. Зі стрімким розвитком цифрових технологій, зокрема вебплатформ і мобільних застосунків, зростає попит на автоматизовані системи, що дозволяють швидко, зручно та безпечно здійснювати пошук і придбання авіаквитків у режимі реального часу. Такий підхід дає змогу уникнути черг, економить час і ресурси, а також надає змогу легко порівнювати доступні варіанти перельотів за вартістю, тривалістю та умовами подорожі.

Сучасне суспільство характеризується підвищеною мобільністю населення, активною діловою комунікацією та інтенсивним розвитком туризму. Усе це сприяє зростанню попиту на ефективні онлайн- рішення, здатні забезпечити своєчасний і зручний доступ до актуальної інформації про авіарейси. Глобалізація економіки, розширення бізнес-діяльності та збільшення кількості міжнародних зв’язків ще більше актуалізують потребу в сучасних платформах для бронювання.

Попри наявність великої кількості сервісів з бронювання, більшість із них зосереджені переважно на базових функціях пошуку та купівлі квитків, залишаючи поза увагою такі важливі аспекти, як гнучке керування бронюванням, можливість скасування чи передачі квитків іншим користувачам, а також відсутність зручного функціоналу для адміністраторів системи. Значна частина сервісів має обмежений або закритий доступ до API, що ускладнює розробку власних рішень на їхній основі.

Крім того, відсутність механізмів кешування запитів та обмеження на кількість звернень до сторонніх API-сервісів створюють технічні виклики для забезпечення стабільної роботи застосунку при високих навантаженнях.

Ураховуючи вищезазначене, створення повнофункціонального вебзастосунку для автоматизації процесів пошуку, бронювання, скасування та передачі авіаквитків з використанням сучасних технологій (зокрема Java, Spring Boot, REST API, PostgreSQL, JWT, зовнішніх API для даних про рейси) є не лише актуальним, але й практично значущим завданням. Такий застосунок дозволяє врахувати як потреби звичайних користувачів, так і адміністраторів системи, забезпечуючи високий рівень функціональності, надійності та зручності в користуванні.

Об’єктом дослідження є системи онлайн-бронювання авіаквитків, їх функціональність та взаємодія з користувачами.

Предметом дослідження є вебзастосунок, що забезпечує зручний інтерфейс для бронювання, управління та передачі авіаквитків.

Метою роботи є розробка програмного забезпечення для автоматизації та оптимізації процесу бронювання авіаквитків через вебзастосунок, що дозволяє зменшити час виконання користувачем основних дій, мінімізувати кількість помилок і покращити загальний користувацький досвід.

Методи дослідження. На першому етапі було проаналізовано предметну галузь онлайн-бронювання та виявлено ключові проблеми і функціональні потреби. Далі проведено огляд існуючих сервісів (Skyscanner, Kiwi, Google Flights) та вивчено особливості роботи з Amadeus API. Для реалізації вебзастосунок обрано технології: Java та Spring Boot для серверної частини, PostgreSQL як систему управління базами даних, HTML/CSS/JavaScript для фронтенду, JWT для авторизації, REST API для взаємодії компонентів, а також реалізовано

кешування результатів запитів до API. Створено адаптивний інтерфейс для користувача, а також окрему панель керування для адміністратора.

Наукова новизна роботи полягає в реалізації поєднання функціоналу пошуку, бронювання, скасування, передачі авіаквитків та керування користувачами в єдиному вебзастосунок з інтеграцією до стороннього API (Amadeus).

Окремою особливістю є впровадження кешування даних пошуку та рольового доступу з безпековими обмеженнями.

Практична значущість полягає в можливості використання створеного програмного забезпечення в реальних умовах для ефективного онлайн-бронювання квитків. Система може бути адаптована як для індивідуального, так і для корпоративного використання у сфері туристичних послуг або внутрішніх службових перевезень.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Ліберман Н.М., Андрусевич Н.В. Розробка програмних засобів для збору інформації для букінгу авіабілетів

// Матеріали VI Всеукраїнської науково-технічної конференції

«Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.25, ДУКІТ, Київ, Україна, С.141-143.

2. Ліберман Н.М. Захист інформації у web-застосунку для бронювання авіаквитків // Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, С.292-293.

1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ БРОНЮВАННЯ АВІАКВИТКІВ

1.1 Вебзастосунок для бронювання авіаквитків

Вебзастосунок для бронювання авіаквитків — це сучасне онлайн-рішення, яке дозволяє автоматизувати процеси пошуку, вибору та бронювання квитків на авіарейси, а також подальшого управління бронюваннями. У зв’язку з розвитком цифрових сервісів та зростаючою потребою в швидкому доступі до транспортних послуг, такі рішення стали важливим інструментом як для кінцевих користувачів, так і для туристичних компаній.

На відміну від традиційного способу купівлі авіаквитків через каси або телефоном, вебзастосунки надають користувачам можливість самостійно переглядати доступні рейси, фільтрувати результати за необхідними параметрами (час, ціна, авіакомпанія), бронювати квитки, а також керувати вже зробленими замовленнями. Це забезпечує значне підвищення зручності, економію часу та персоналізований підхід до обслуговування.

Основна мета вебзастосунку — забезпечити швидкий, доступний та захищений механізм для бронювання квитків, що працює в режимі 24/7 та адаптується до індивідуальних потреб користувача.

Основні функціональні компоненти вебзастосунку включають:

— модуль пошуку рейсів за напрямком, датою, кількістю пасажирів;

— інтерфейс відображення знайдених пропозицій;

— механізм бронювання з перевіркою підтвердженого акаунта;

— особистий кабінет користувача з історією бронювань та можливістю їх скасування або передачі іншим користувачам;

— система підтвердження акаунта через код на email;

— модуль підтримки для зворотного зв’язку з адміністрацією сервісу;

— панель адміністратора для моніторингу дій користувачів, перегляду та модерації бронювань, а також керування правами доступу;

— вбудована система кешування запитів для зменшення навантаження на зовнішній API (Amadeus) та прискорення роботи.

Переваги:

— економія часу на оформлення квитків;

— повний контроль користувача над бронюванням;

— безпечна авторизація (JWT);

— вбудована підтримка та адаптивний інтерфейс;

— можливість передачі квитків іншим особам без звернення до служби підтримки.

Виклики та обмеження:

— необхідність обробки великої кількості даних у режимі реального часу;

— залежність від стороннього API (ліміти, плата за використання);

— потреба у високому рівні захисту персональних даних згідно з вимогами GDPR;

— забезпечення стабільної роботи кешування, оскільки повторні запити повинні оброблятися без затримок.

1.2 Специфіка автоматизації бронювання авіаквитків

Автоматизація процесу бронювання авіаквитків передбачає використання програмних інструментів для повного або часткового виконання рутинних дій, які раніше здійснювалися вручну. Такий підхід дозволяє значно пришвидшити обробку запитів, мінімізувати людський фактор та покращити точність та своєчасність надання інформації.

Основні етапи автоматизованого бронювання:

1. Формування запиту користувача — обираються міста відправлення й прибуття, дата, кількість пасажирів.
2. Використання зовнішнього API (Amadeus) — на підставі запиту система надсилає HTTP-запит до API й отримує відповідь у форматі JSON.
3. Обробка відповіді на сервері — парсинг даних, перевірка валідності, збереження у базу даних та кеш.
4. Візуалізація результатів — на клієнтському інтерфейсі відображаються рейси, які можна забронювати.
5. Оформлення бронювання — створення нового запису у базі даних, прив’язка до користувача, перевірка підтвердження акаунта.
6. Передача або скасування — через функціонал особистого кабінету користувач може змінювати статуси своїх квитків.
7. Адміністрування системи — відбувається з панелі адміністратора, де є доступ до журналу дій, профілів користувачів та системної статистики.

1.3 Цільова аудиторія

Цільовою аудиторією розробленого вебзастосунку є широке коло користувачів, які мають потребу в самостійному та зручному бронюванні авіаквитків:

— приватні особи — студенти, мандрівники, працівники, що часто подорожують і хочуть бронювати квитки без посередників;

— бізнес-користувачі — працівники компаній, що організовують службові поїздки;

— менеджери туристичних агентств — можуть використовувати платформу для бронювання для клієнтів;

— адміністратори які контролюють правильність використання застосунку, відповідають на запити підтримки, проводять технічний аудит.

Очікування користувачів від системи:

— простий та інтуїтивний інтерфейс, що не потребує застосунку навчання;

— швидка реєстрація та підтвердження облікового запису;

— можливість отримати детальну інформацію про рейси;

— безпечне зберігання та обробка персональних даних;

— підтримка української та англійської мови;

— швидке оформлення бронювання в кілька кліків;

— зручна система скасування та передачі квитків.

Таким чином, система повинна задовольняти потреби як окремих користувачів, так і малих організацій, пропонуючи багатофункціональний, безпечний та масштабований інструмент для онлайн-бронювання.

1.4 Дослідження існуючих аналогів

У сучасному цифровому середовищі, яке стрімко розвивається та змінюється, онлайн-сервіси для пошуку, бронювання та порівняння авіарейсів стали невід'ємною частиною щоденного життя мільйонів користувачів по всьому світу. Зростання мобільності населення, активізація туристичних потоків, розвиток бізнес-подорожей, а також загальна цифровізація послуг призвели до того, що зручний і швидкий доступ до інформації про авіарейси став однією з ключових потреб сучасної людини.

Серед великої кількості онлайн-платформ, які надають подібні послуги, особливої уваги заслуговують такі гіганти, як Skyscanner, Kiwi.com, Google Flights та Amadeus API. Кожен з цих сервісів має свої особливості, сильні сторони та обмеження, що робить їх цікавими об’єктами для детального дослідження та аналізу.

Наприклад, Skyscanner вирізняється широким охопленням ринку та можливістю порівнювати ціни між різними авіакомпаніями та агентствами. Kiwi.com впроваджує інноваційні алгоритми побудови складних маршрутів із пересадками, які часто є вигіднішими за прямі рейси. Google Flights забезпечує

надзвичайно швидкий пошук, зручний інтерфейс і потужні аналітичні інструменти для прогнозу змін цін. Amadeus API, у свою чергу, орієнтований на розробників та корпоративні рішення, забезпечуючи гнучкий і масштабований доступ до реальних даних з глобальної системи бронювання.

Проведення порівняльного аналізу зазначених сервісів є необхідним кроком на шляху до створення власного вебзастосунку для бронювання авіаквитків. Такий аналіз дозволяє чітко визначити, які функції є найбільш затребуваними серед користувачів, які можливості варто реалізувати у власному рішенні, а які аспекти потребують удосконалення чи оптимізації. Оцінка переваг і недоліків існуючих платформ дає змогу сформулювати чіткі технічні та функціональні вимоги до системи, що розробляється, і побудувати її таким чином, щоб вона була конкурентоспроможною, зручною для користувача та технологічно ефективною.

Таким чином, вивчення сучасних рішень на ринку онлайн-бронювання авіаквитків дозволяє закласти надійне підґрунтя для подальшого проєктування та реалізації інноваційного вебзастосунку, орієнтованого на потреби кінцевого користувача та здатного вирішувати реальні задачі у сфері авіаційних перевезень.

1.4.1 Skyscanner

Skyscanner є одним із найвідоміших метапошукових сервісів у галузі подорожей. Його головна функція — агрегування даних з численних сайтів авіакомпаній, агентств і партнерських систем для надання користувачу найвигідніших пропозицій.

Основні переваги:

— велика база авіакомпаній;

— зручний фільтр пошуку за датами, пересадками, ціною;

— можливість шукати дешеві місяці;

— підтримка декількох валют та мов.

Недоліки:

— бронювання здійснюється на сторонніх сайтах (перенаправлення);

— обмежений контроль за процесом оформлення замовлення;

— відсутність функцій адміністрування та кастомізації.

Skyscanner ідеально підходить для пошуку, але не охоплює повного циклу взаємодії з користувачем після бронювання.

1.4.2 Kiwi.com

Kiwi.com — це сервіс, який надає більш розширену функціональність порівняно зі Skyscanner. Однією з головних його переваг є можливість створення складних маршрутів із пересадками між різними авіакомпаніями (у тому числі лоукостерами), навіть якщо між ними немає офіційної співпраці.

Переваги:

— функція Smart Booking — побудова маршрутів на основі непов’язаних авіакомпаній;

— гарантія Kiwi у разі затримок/скасувань;

— зручний інтерфейс та інтегрована оплата;

— часткова підтримка мобільного застосунку для керування бронюваннями.

Недоліки:

— закритість API (доступ лише партнерам);

— відсутність кастомізації інтерфейсу та логіки для розробників;

— неможливість передачі квитка іншій особі.

Kiwi.com робить акцент на зручність користувача під час пошуку, але не є відкритою платформою для розробки власних рішень.

1.4.3 Google Flights

Google Flights — продукт від компанії Google, що дозволяє шукати авіарейси через зручний інтерфейс з інтеграцією карт Google. Його особливістю є швидкість роботи та прогнозування змін цін на рейси.

Переваги:

— візуалізація маршрутів на карті;

— підказки щодо кращих цін та дат;

— швидка відповідь і простота інтерфейсу.

Недоліки:

— відсутність власної системи бронювання (використовується перенаправлення);

— немає відкритого API для сторонніх розробників;

— мінімальні можливості персоналізації або подальшого керування бронюванням.

Цей сервіс більше підходить для одноразового пошуку, ніж для повноцінної взаємодії з користувачем після бронювання.

1.4.4 Amadeus API

На відміну від попередніх, Amadeus API — це не інтерфейс для кінцевого користувача, а інструмент для розробників. Він надає доступ до глобальної бази авіарейсів, готелів, прокату авто, дозволяючи реалізовувати повнофункціональні туристичні платформи.

Переваги:

— потужне API з докладною документацією;

— підтримка пошуку, бронювання, перевірки наявності, багажу тощо;

— можливість роботи у тестовому режимі (sandbox);

— доступ до великої кількості авіаперевізників.

Недоліки:

— потребує високої технічної кваліфікації;

— частина функціональності платна;

— без належної реалізації можуть виникати затримки у відповідях.

Amadeus API є найкращим варіантом для створення кастомного вебзастосунку, оскільки дає повний контроль над логікою пошуку, обробки й подальшого керування квитками.

Таблиця 1.1

Аналіз аналогів

Характеристика	Skyscanner	Kiwi.com	Google Flights	Amadeus API	Aviasales	Booking .com
Відкрите API	Так	Ні	Ні	Так	Так	Так
Можливість бронювання	Ні	Так	Ні	Так	Частково	Так
Підтримка передачі квитків	Ні	Ні	Ні	Ні	Ні	Так
Підтримка кешування	Частково	Так	Так	Так	Ні	Так
Адмін-панель	Ні	Ні	Ні	Ні	Ні	Так
Гнучкість інтеграції	Середня	Низька	Низька	Висока	Висока	Середня

1.5 Визначення вимог до вебзастосунку для бронювання авіаквитків

На основі аналізу предметної галузі, вивчення функціоналу існуючих сервісів (зокрема Skyscanner, Kiwi.com, Google Flights), а також з урахуванням очікувань користувачів і цілей кваліфікаційної роботи, було сформовано перелік функціональних і нефункціональних вимог до розроблюваного вебзастосунку.

Функціональні вимоги:

— реєстрація та авторизація користувачів – з перевіркою пошти та можливістю підтвердження облікового запису;

— доступ до особистого кабінету – після авторизації користувач повинен мати змогу переглядати свої бронювання, скасовувати або передавати квитки;

— пошук авіарейсів – за заданими параметрами (пункти відправлення та прибуття, дата, кількість пасажирів);

— бронювання квитків – із можливістю підтвердження через модальне вікно;

— передача квитків іншим користувачам – за допомогою введення електронної адреси зареєстрованого користувача;

— адміністративна панель – для керування обліковими записами, перегляду бронювань та адміністрування запитів.

Нефункціональні вимоги:

— захист даних – використання JWT (JSON Web Token) для безпечної аутентифікації та авторизації користувачів;

— інтеграція з API – підключення до Amadeus API для отримання актуальної інформації про рейси;

— висока швидкодія – результати пошуку повинні відображатися протягом однієї секунди;

— резервне копіювання даних – реалізація механізму резервного збереження інформації для запобігання її втраті.

2 ЗАСОБИ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ

2.1 Середовище розробки та система контролю версій

Для розробки вебзастосунку було використано середовище IntelliJ IDEA, яке на сьогоднішній день є одним із найпотужніших та найпопулярніших інтегрованих середовищ розробки (IDE) для мови програмування Java. Воно забезпечує високу продуктивність праці завдяки широкому набору функцій, інтуїтивно зрозумілому інтерфейсу та розширеній підтримці сучасних фреймворків і технологій. IntelliJ IDEA дозволяє ефективно працювати з такими інструментами, як Spring Boot, Hibernate, Maven, Gradle, а також має вбудовану підтримку для REST API, SQL- запитів, юніт-тестування та дебагінгу.

Однією з ключових переваг IntelliJ IDEA є розумне автодоповнення коду (smart code completion), статичний аналіз, підсвічування помилок у реальному часі, генерація шаблонного коду, підтримка рефакторингу, а також можливість одночасної роботи з фронтендом (HTML/CSS/JS) та бекендом у єдиному середовищі. Завдяки цьому середовище розробки значно пришвидшує процес створення програмного забезпечення, зменшуючи кількість синтаксичних помилок та підвищуючи якість коду. Для організації проєкту використовувалась структура з розділенням на шари (controller, service, repository), яку зручно підтримувати завдяки навігації та автоматизованому керуванню залежностями.

У якості системи контролю версій було обрано Git — розподілену систему керування версіями, яка дозволяє зберігати всі зміни, відслідковувати історію розробки, повертатись до попередніх версій проєкту та забезпечує надійний механізм інтеграції змін у командній роботі. Проєктний репозиторій було розміщено на платформі GitHub, що забезпечує застосункові інструменти співпраці — такі як pull requests, issue tracking, branch protection, та можливість публічного або приватного зберігання коду.

2.2 Вибір мови програмування та фреймворків

Основною мовою програмування для реалізації серверної частини обрано Java — надійну, об'єктно-орієнтовану мову, яка широко використовується для створення вебзастосунків. Для побудови бекенду використано фреймворк Spring Boot, який забезпечує швидкий старт проєкту, автоматичне налаштування компонентів та легку інтеграцію з базами даних і REST-сервісами.

Для фронтенду використано стандартні вебтехнології: HTML, CSS та JavaScript, що дозволяють створити адаптивний, інтерактивний та зручний інтерфейс користувача.

2.3 Огляд Spring Boot як основного бекенд-інструменту

Spring Boot забезпечує спрощений підхід до створення Java-застосунків, дозволяючи зменшити кількість конфігурацій. Його головними перевагами є підтримка REST-контролерів, автоматичне керування залежностями через Spring Initializr, а також інтеграція з Spring Security, JPA та іншими модулями.

У проєкті Spring Boot використовується для:

— побудови REST API для обробки запитів користувачів;

— реалізації сервісної логіки через анотовані сервіси;

— доступу до бази даних через Spring Data JPA;

— керування конфігураціями та властивостями застосунку.

2.4 Amadeus API для отримання інформації про рейси

Для пошуку та бронювання авіаквитків у проєкті інтегровано Amadeus API — професійний API для отримання даних про авіарейси. Він дозволяє отримувати найактуальніші дані щодо маршрутів, розкладу, цін, кількості доступних місць та авіакомпаній. Взаємодія з API реалізується через окремий сервіс, який формує

запити, обробляє JSON-відповіді та адаптує їх до формату, зручного для використання у вебінтерфейсі.

2.5 PostgreSQL як система управління базами даних

Для зберігання даних застосовано PostgreSQL — надійну, масштабовану та безпечну СУБД з відкритим кодом. Вона підтримує транзакції, зовнішні ключі, індекси та запити з високим рівнем складності. PostgreSQL забезпечує ефективне зберігання даних користувачів, історії бронювань, підтвердження акаунтів та взаємодію з Amadeus API.

Зв'язок із базою реалізовано за допомогою Spring Data JPA, що дозволяє працювати з даними через репозиторії без необхідності написання SQL-запитів вручну.

Інструменти безпеки: Spring Security, JWT

Для забезпечення безпеки застосунку реалізовано аутентифікацію та авторизацію за допомогою Spring Security та JWT (JSON Web Token). Після входу користувач отримує токен, який зберігається в localStorage та додається до кожного запиту.

JWT-токени перевіряються спеціальним фільтром JwtAuthenticationFilter, який перевіряє підпис, термін дії та ролі користувача. Це дозволяє надійно обмежити доступ до функцій, призначених для авторизованих або адміністраторських облікових записів.

2.6 Інструменти для тестування та API-клієнти

Для перевірки API до інтеграції з фронтендом використовувалися інструмент Postman. Вони дозволяють надсилати запити до серверної частини, тестувати всі кінцеві точки, перевіряти авторизацію, статуси та відповіді.

Це значно спростило налагодження взаємодії між клієнтом і сервером до завершення верстки фронтенду.

Усі ці компоненти разом утворюють сучасну архітектуру, здатну обслуговувати реального користувача та масштабуватись для більшої кількості одночасних запитів у майбутньому.

3 ПРОЄКТУВАННЯ

3.1 Проєктування архітектури застосунку

Вебзастосунок для бронювання авіаквитків реалізовано за моделлю клієнт–сервер, що дозволяє розмежувати функціональність між інтерфейсом користувача (frontend) та серверною логікою (backend). Такий підхід забезпечує кращу масштабованість, безпеку та можливість підтримки системи в майбутньому.

Клієнтська частина розроблена з використанням HTML, CSS і JavaScript. Вона реалізована у вигляді статичних сторінок index.html, profile.html та admin.html. Ці сторінки відповідають за взаємодію з користувачем, зокрема пошук рейсів, перегляд і керування бронюваннями, реєстрацію, підтвердження акаунта, авторизацію, скасування або передачу квитків іншим користувачам. Всі дії на клієнті здійснюються через асинхронні HTTP-запити до сервера.

Серверна частина застосунку реалізована з використанням фреймворку Spring Boot. Вона відповідає за обробку запитів, виконання бізнес-логіки, збереження і отримання даних з бази даних PostgreSQL. Для захисту сервісу використано Spring Security з JWT-аутентифікацією. Це дозволяє реалізувати обмеження доступу до функцій на основі ролей користувачів (див.рисунок 3.1).

Сервер також здійснює інтеграцію з зовнішнім сервісом — Amadeus API. Через нього відбувається запит до бази авіарейсів та отримання актуальних пропозицій перельотів. Отримані дані кешуються на стороні сервера для оптимізації швидкості відповіді та зниження навантаження на зовнішнє API.

Загальна архітектура побудована відповідно до принципів REST, що забезпечує простоту масштабування, повторного використання коду, гнучкість та ефективну інтеграцію з іншими сервісами в майбутньому (див. рисунок 3.2).

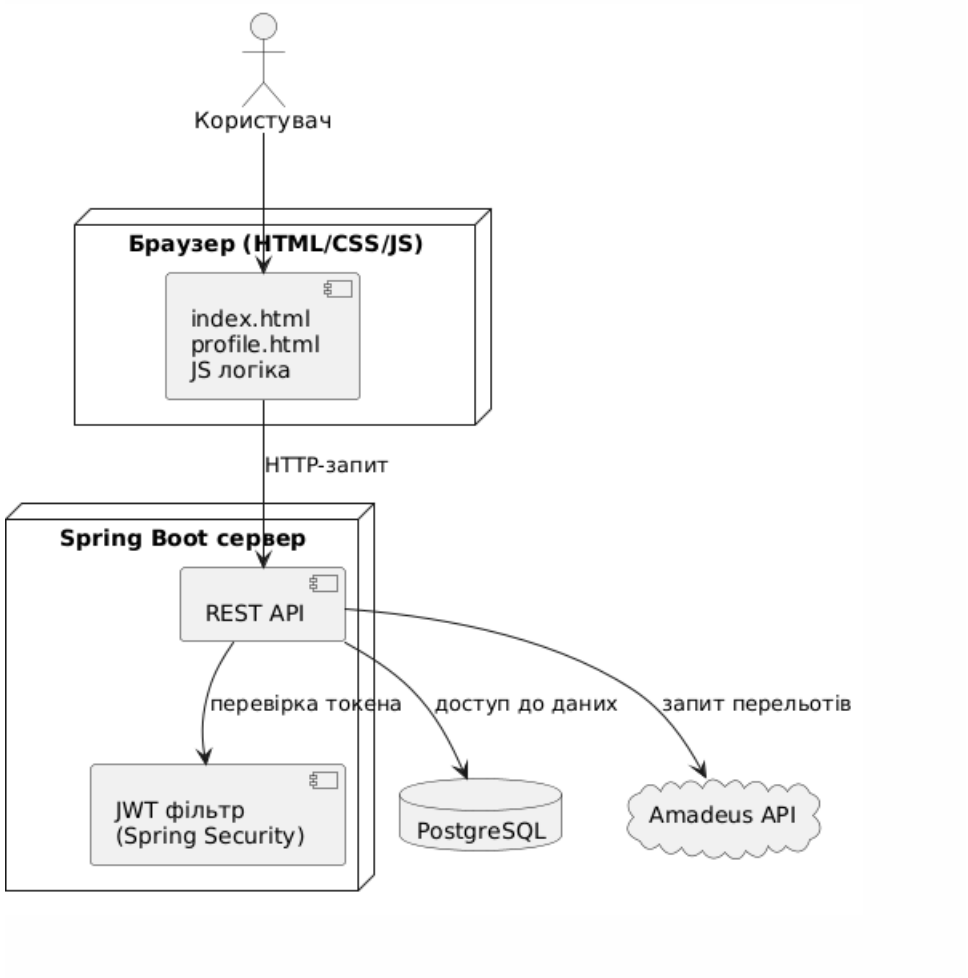


Рис. 3.1 Діаграма розгортання клієнт серверної архітектури

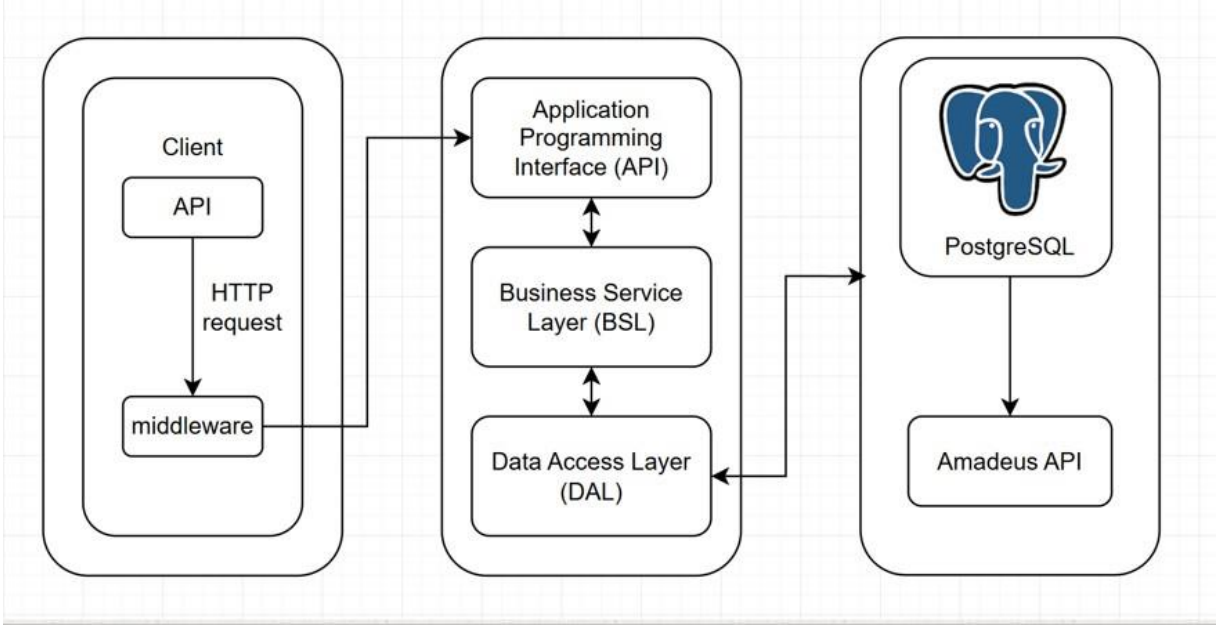


Рис. 3.2 Діаграма архітектура застосунку

3.2 Архітектура серверної частини (REST API, безпека)

Серверна частина вебзастосунку для бронювання авіаквитків реалізована на основі фреймворку Spring Boot, який дозволяє швидко створювати масштабовані, безпечні REST-сервіси з розділенням відповідальностей між різними компонентами.

Основу архітектури становить трирівнева структура:

— контролери (Controllers) — приймають HTTP-запити від клієнта, обробляють маршрути, викликають відповідні сервіси;

— сервіси (Services) — містять бізнес-логіку, обробляють запити та координують взаємодію з базою даних;

— репозиторії (Repositories) — виконують доступ до бази даних через Spring Data JPA, з підтримкою транзакцій та запитів.

Безпека

Для забезпечення безпеки реалізовано JWT-аутентифікацію, що дозволяє працювати з токенами доступу:

— після авторизації користувач отримує JWT-токен;

— всі подальші запити містять цей токен у заголовку Authorization;

— доступ до контролерів обмежено відповідно до ролей користувача (USER, ADMIN);

— токен перевіряється фільтром JwtAuthenticationFilter.

Конфігурація безпеки реалізована у класі SecurityConfig і включає:

— дозволи на реєстрацію, логін, підтвердження акаунта — без авторизації;

— обмеження доступу до адміністративних функцій — лише для ролі ADMIN;

— обробку помилок авторизації та несанкціонованих запитів.

REST API

Вебсервіс підтримує такі основні маршрути:

— post /api/auth/register — реєстрація користувача;

— post /api/auth/login — аутентифікація та видача JWT;
— get /api/flights/search — пошук рейсів за параметрами;
— post /api/bookings — бронювання квитків;
— get /api/profile/bookings — перегляд бронювань;
— post /api/tickets/transfer — передача квитка іншому користувачеві;
— delete /api/bookings/{id} — скасування бронювання;
— get /api/admin/users — перегляд користувачів (ADMIN);
— get /api/admin/bookings — перегляд усіх бронювань (ADMIN).

Також реалізовано систему кешування результатів пошуку через окремий шар сервісів, що зменшує кількість запитів до Amadeus API (див. рисунок 3.3).

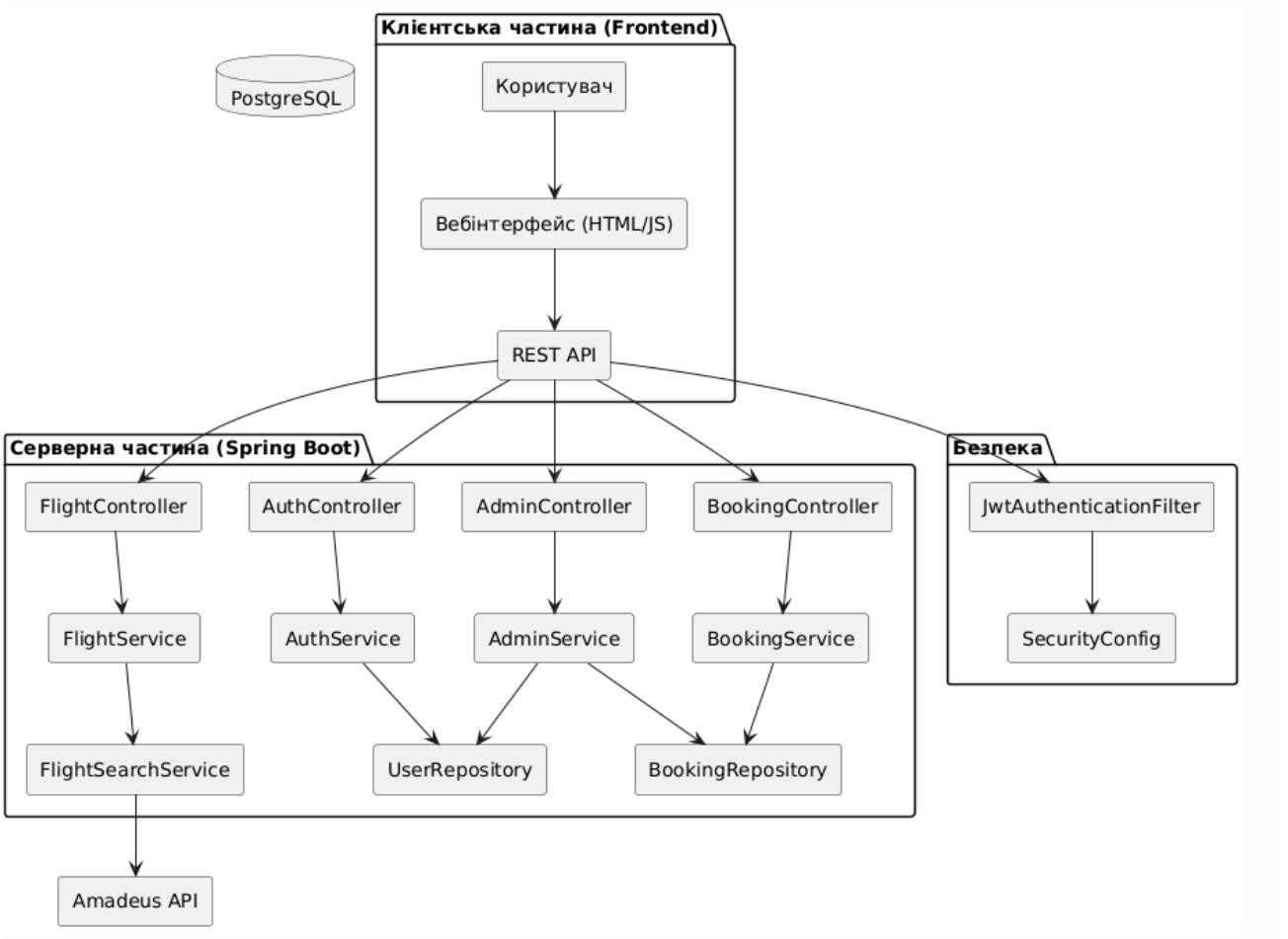


Рис. 3.3 Діаграма компонентів серверної частини

3.3 Архітектура клієнтської частини (HTML/CSS/JS)

Клієнтська частина вебзастосунку для бронювання авіаквитків реалізована з використанням стандартних вебтехнологій: HTML, CSS та JavaScript**.** Така архітектура забезпечує гнучкість, швидкодію та максимальну сумісність із сучасними браузерами.

Проєкт клієнтської частини розділено на кілька ключових файлів, кожен з яких виконує окрему функцію в інтерфейсі користувача:

HTML (index.html, profile.html, admin.html)

HTML-файли відповідають за структурну розмітку сторінок:

— index.html — головна сторінка пошуку рейсів, з формою для вибору напрямку, дати та кількості пасажирів;

— profile.html — профіль користувача з відображенням заброньованих квитків, кнопками «Скасувати» та «Передати»;

— admin.html — панель адміністратора для перегляду користувачів, бронювань та керування ролями.

CSS (style.css)

Файл style.css визначає візуальне оформлення інтерфейсу**:**

— адаптивний дизайн під десктопи та мобільні пристрої;

— підтримка темної/світлої теми;

— стилізація модальних вікон**,** карток квитків**,** повідомлень та ін.

JavaScript (main.js, profile.js, admin.js)

JavaScript відповідає за динаміку інтерфейсу та інтеграцію з бекендом:

— main.js — логіка для головної сторінки: пошук рейсів, бронювання, перевірка авторизації, теми;

— profile.js — логіка профілю: перегляд, скасування, передача квитків, підтвердження email;

— admin.js — адміністрування: блокування користувачів, підвищення прав, пошук, перегляд кешу.

Всі файли спілкуються з сервером через HTTP-запити (fetch)**,** що дозволяє реалізувати асинхронну взаємодію з REST API (див. рисунок 3.4). Зокрема:

— при пошуку рейсів викликається API Amadeus;

— при бронюванні — бекенд формує новий запис у БД;

— при передачі квитка — сервер перевіряє нового користувача та змінює власника бронювання.

Переваги обраної архітектури:

— модульність: розділення JS-коду на логічні блоки спрощує супровід і розширення функціональності;

— простота оновлення інтерфейсу: легко додавати нові сторінки або оновлювати окремі елементи без переробки всього застосунку;

— кросбраузерність**:** завдяки використанню стандартних технологій HTML/CSS/JS застосунок працює у всіх сучасних браузерах; діаграми компонентів клієнтської частини.

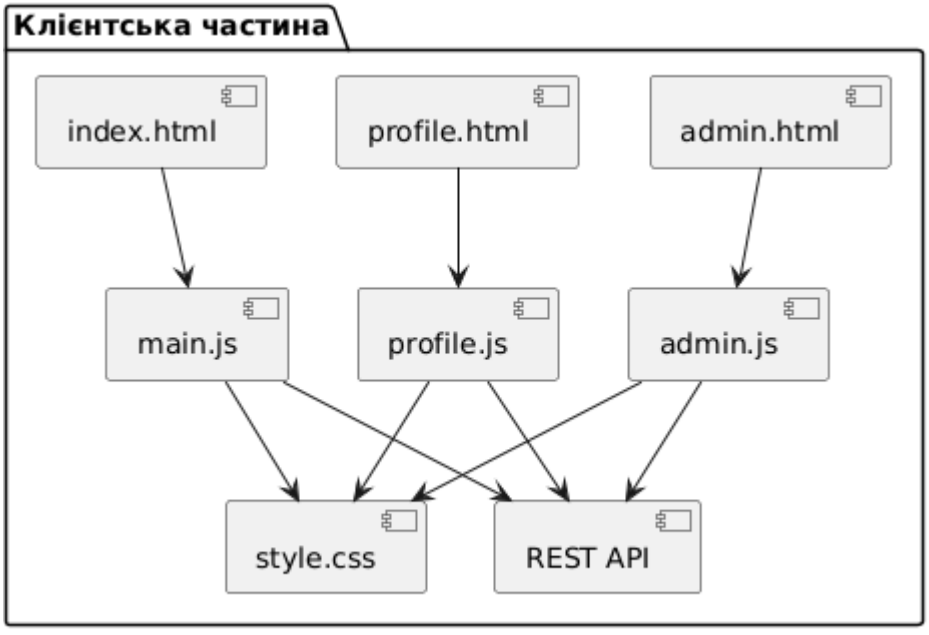


Рис. 3.4 Діаграма компонентів клієнтської частини

Для зображення взаємодії користувача з системою створимо діаграму варіантів використання (див. рисунок 3.5).

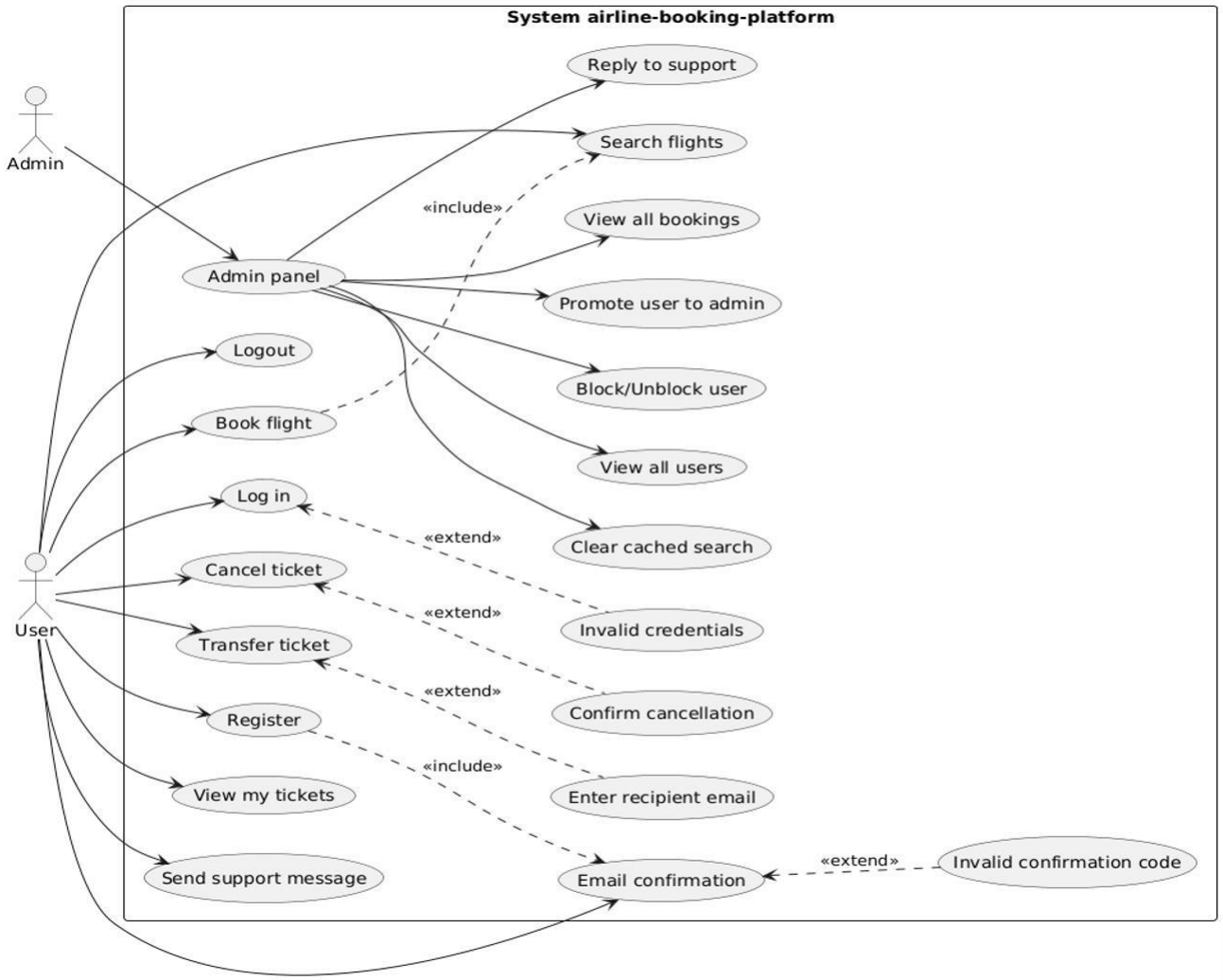


Рис. 3.5 Діаграма варіантів використання

3.4 Структура бази даних та кешування результатів пошуку

У вебзастосунку для бронювання авіаквитків реалізована реляційна база даних, побудована на основі PostgreSQL. Архітектура бази забезпечує логічне поділ на сутності, які відображають ключові бізнес-об’єкти системи: користувачів, квитки, пропозиції рейсів, звернення до служби підтримки тощо, (див. рисунок 3.6) .

Основні сутності:

— user — зберігає основну інформацію про користувача: email, пароль, роль, стан підтвердження акаунта, а також confirmationCode для верифікації;

— ticketBooking — представляє бронювання користувачем авіаквитка. Містить посилання на конкретного користувача і обрану FlightOffer, має статус активності та ознаку, чи було квиток передано іншому користувачеві;

— flightOffer — описує параметри перельоту, отримані з Amadeus API. Містить код авіакомпанії, міста, дати, кількість місць, ціни;

— price — окрема сутність для відображення вартості квитка (загальна, базова та валюта);

— supportRequest — звернення до технічної підтримки, з email, повідомленням користувача, відповіддю адміністратора та міткою часу;

— supportMessage — використовується для внутрішньої взаємодії адміністратора з користувачем через профіль.

Уся система побудована з урахуванням зв’язків типу "один до багатьох"**:**

— один користувач може мати багато бронювань (User ↔ TicketBooking);

— одне бронювання прив’язане до однієї пропозиції перельоту (TicketBooking ↔ FlightOffer);

— одне звернення відповідає одному користувачу (User ↔ SupportRequest).

Кешування результатів пошуку

Інформація про пропозиції перельотів (FlightOffer) зберігається у базі після першого запиту. Наступні запити з такими ж параметрами (міста, дати, кількість пасажирів) використовують кеш, що дозволяє:

— уникнути надмірних запитів до Amadeus API;

— скоротити час відповіді до 1 секунди;

— знизити навантаження на систему.

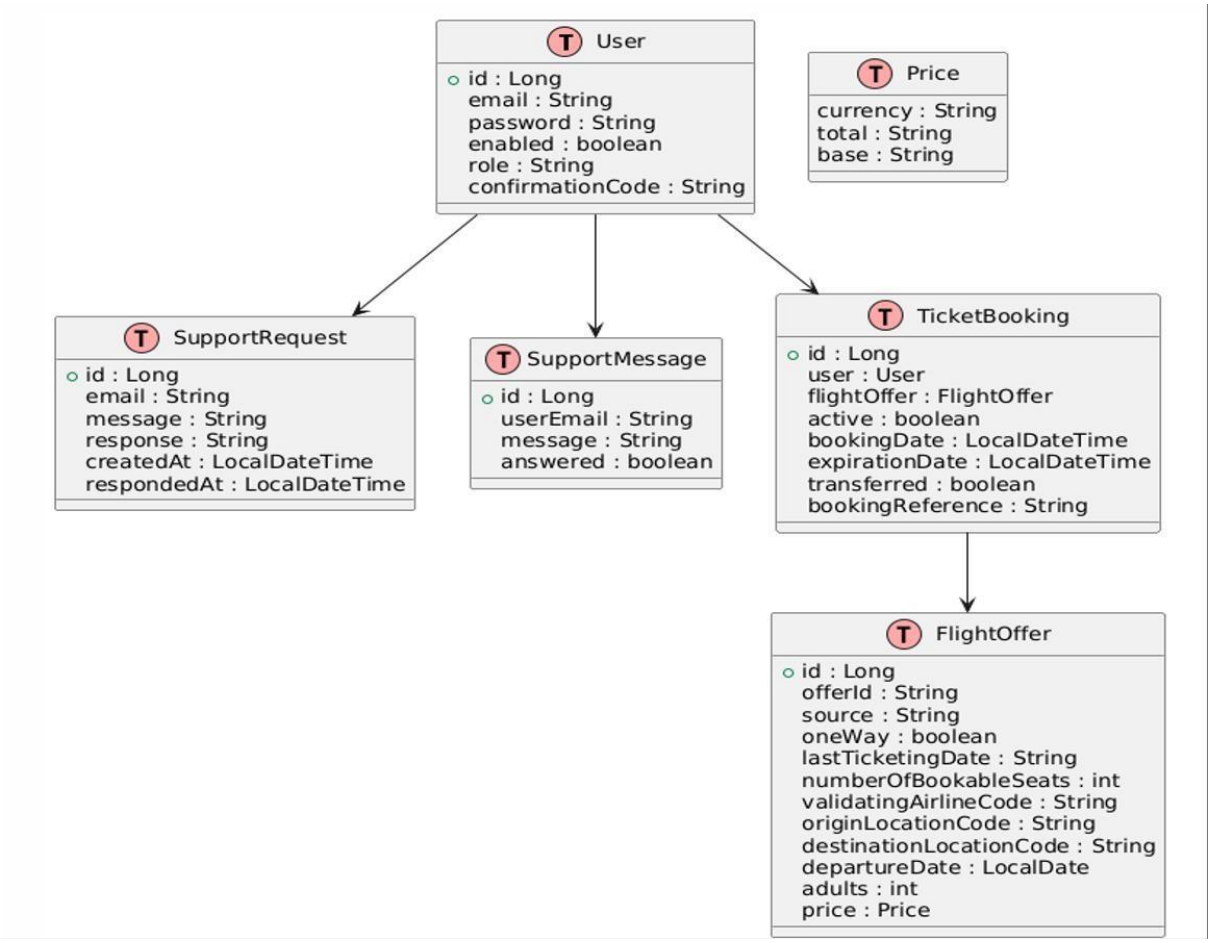


Рис. 3.6 Діаграма структури бази даних

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ

4.1 Реєстрація, авторизація, підтвердження акаунта

На першому етапі реалізації вебзастосунку було створено систему управління користувачами, що включає функціонал реєстрації, авторизації та підтвердження акаунта через електронну пошту. Основна мета — забезпечити безпеку, персоналізацію доступу до функціоналу та захист даних користувачів.

Для цього була реалізована форма реєстрації (див. рисунок 4.1), яка перевіряє правильність введених даних (email, пароль) на клієнтському рівні. Після натискання кнопки «Зареєструватися» дані передаються на сервер, де відбувається створення облікового запису, а також генерація унікального коду підтвердження.

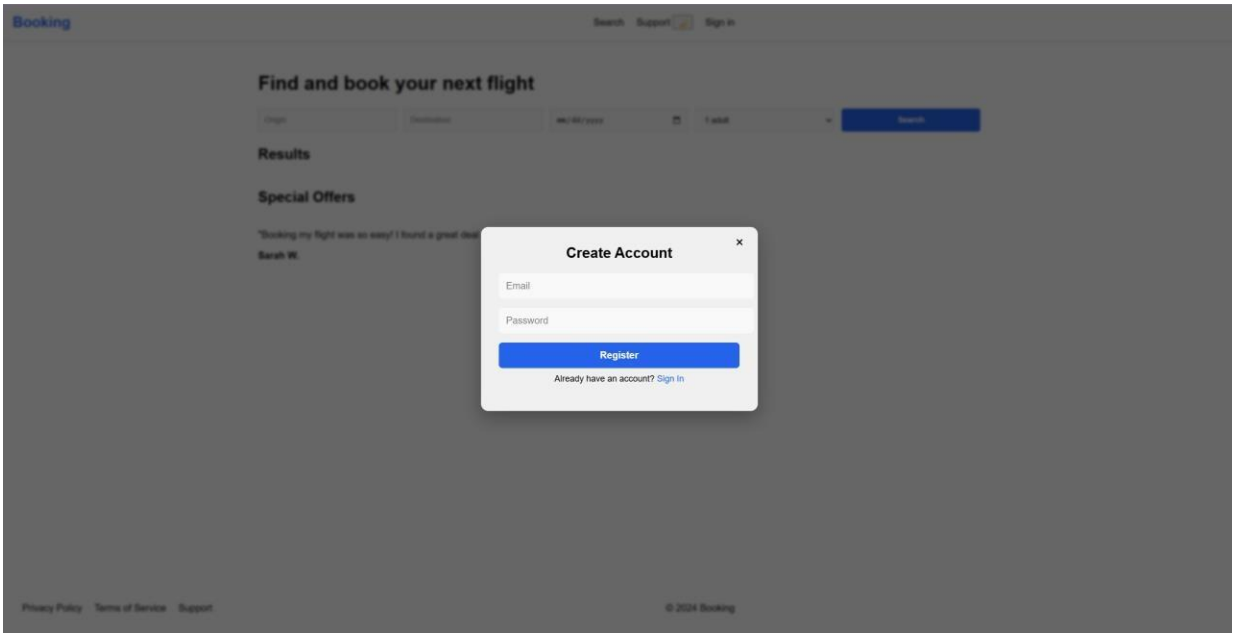


Рис. 4.1 Форма реєстрації користувача

Одразу після реєстрації система надсилає email на вказану адресу з кодом підтвердження, (див. рисунок 4.2). До активації акаунта користувач не має змоги здійснювати бронювання квитків. Це підвищує рівень довіри до сервісу та запобігає використанню фейкових облікових записів.

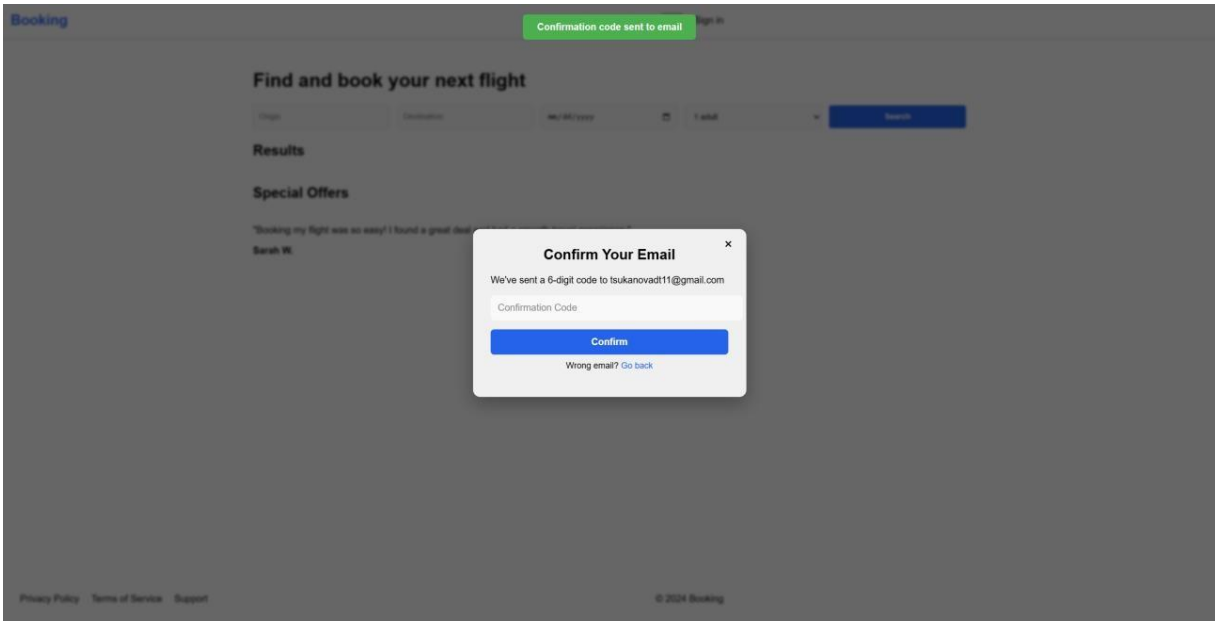


Рис. 4.2 Повідомлення про необхідність підтвердження акаунта

Після переходу на сторінку підтвердження користувач вводить код із листа, (див. рисунок 4.3). У разі правильного введення акаунт активується, і користувач отримує доступ до всіх функцій системи.

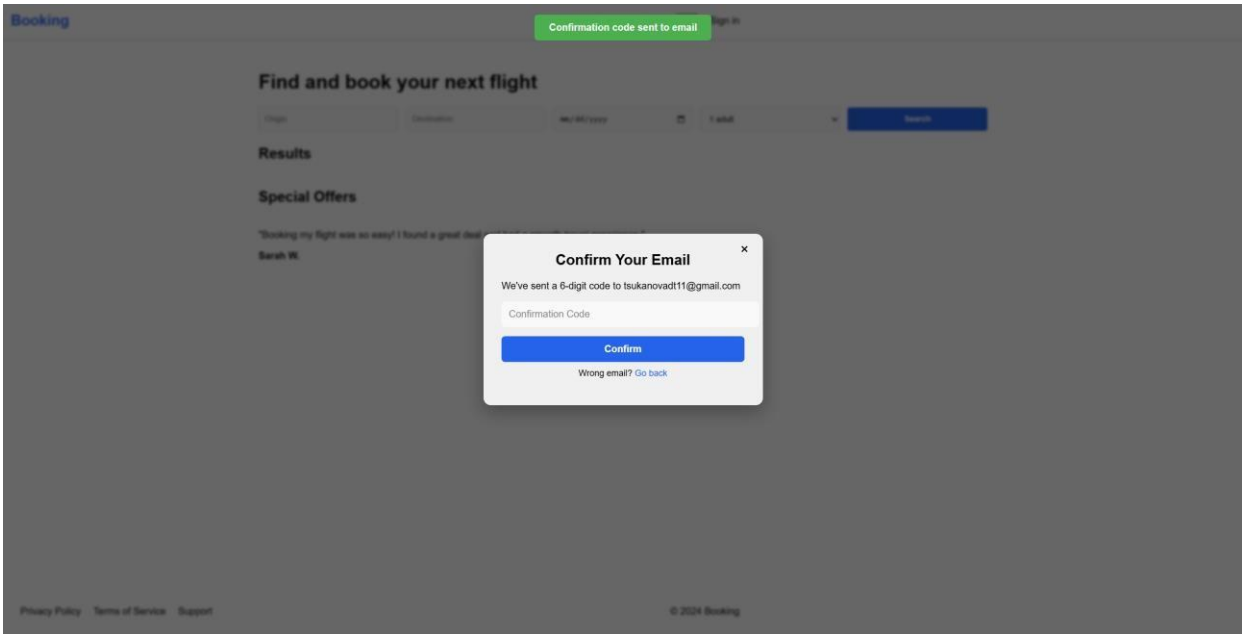


Рис. 4.3 Форма підтвердження акаунта

Авторизація реалізована через введення email і пароля. При успішному вході користувач отримує JWT-токен, який зберігається в localStorage. Це забезпечує збереження сесії навіть після перезавантаження сторінки, (див. рисунок 4.4). При кожному запиті до захищених ресурсів токен додається до заголовку Authorization.

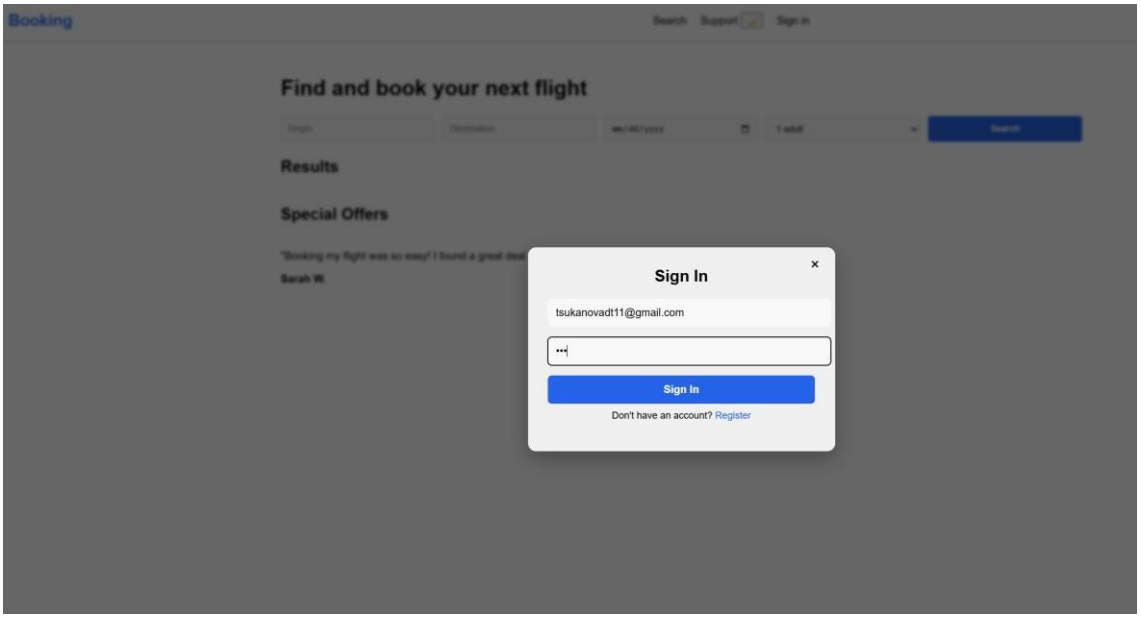


Рис. 4.4 Форма авторизації з перевіркою помилок

Для управління станом сесії реалізовано автоматичну перевірку токена при завантаженні сторінки, (див. рисунок 4.5). У разі його відсутності або прострочення користувач перенаправляється на сторінку входу.

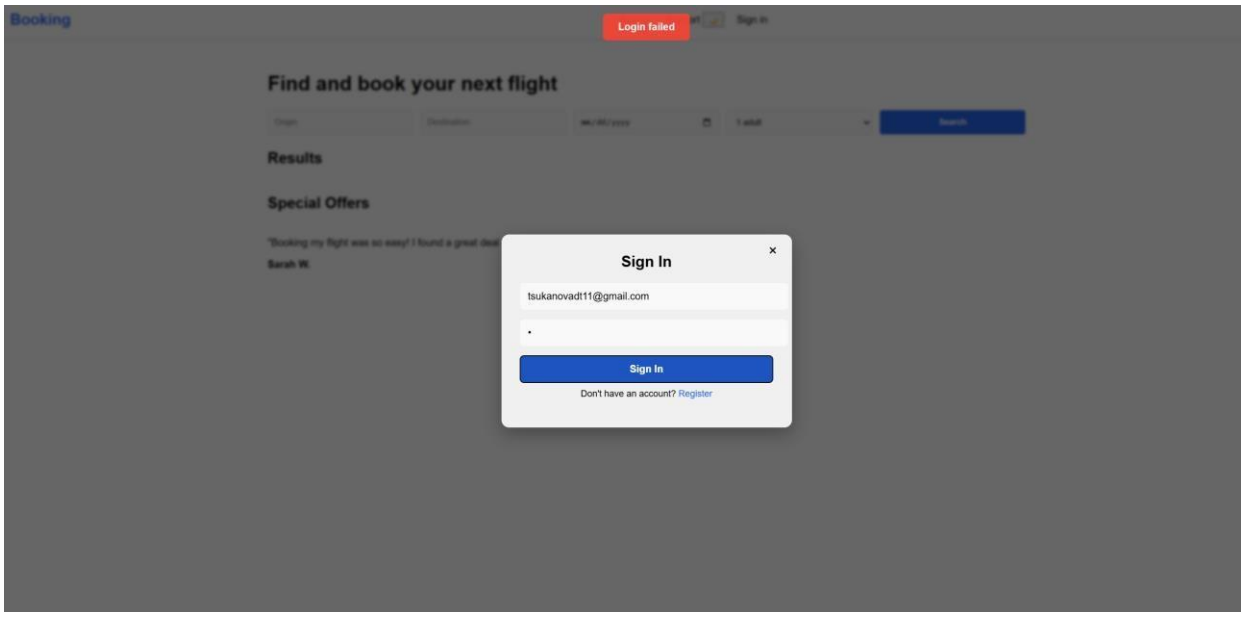


Рис. 4.5 Інтерфейс повідомлення про помилку входу або неправильний токен

Загальну логіку авторизації можно подивитись (див. рисунок 4.6).

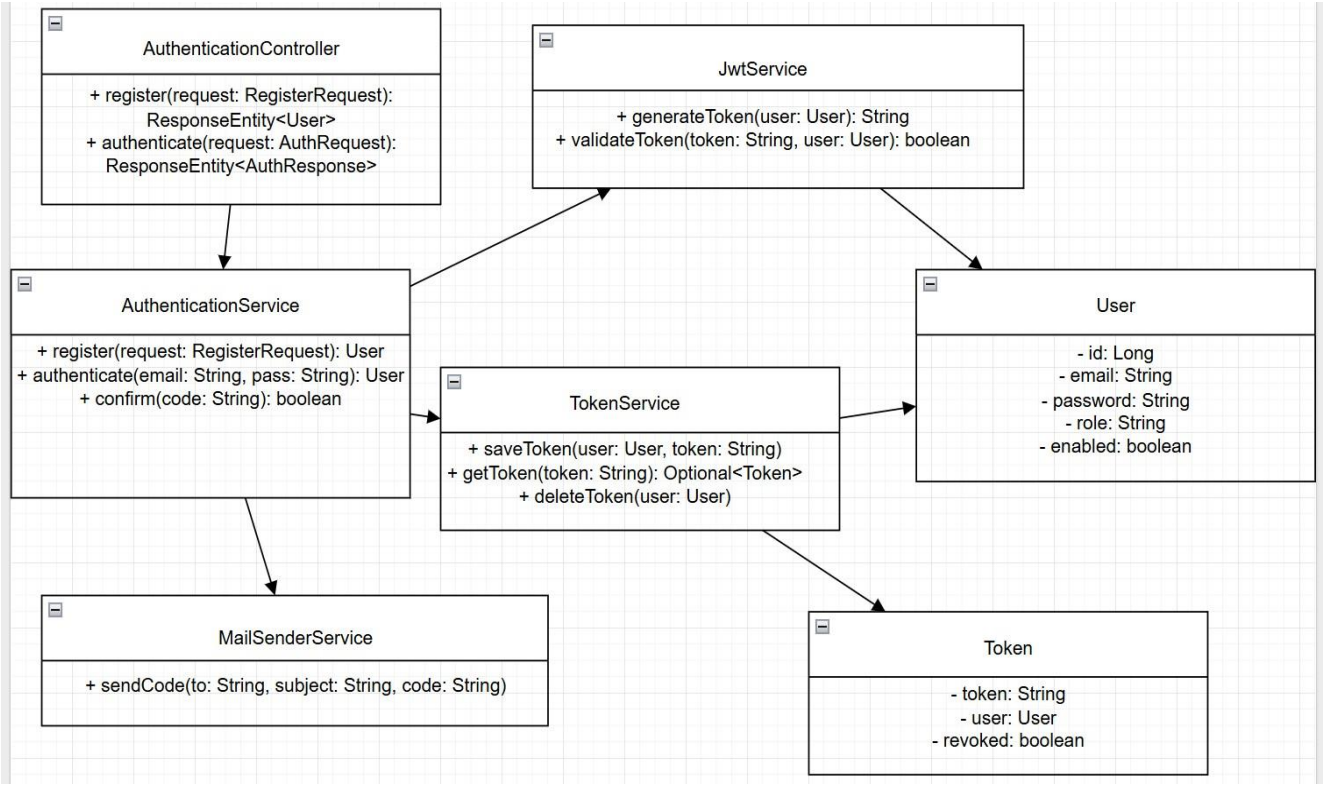


Рис. 4.6 Діаграма класів авторизації

4.2 Пошук квитків через Amadeus API з кешуванням результатів

Пошук авіарейсів — ключова функція вебзастосунку. Реалізація базується на інтеграції з Amadeus API, що дозволяє отримувати актуальну інформацію про авіарейси за вказаними параметрами: місто відправлення, місто прибуття, дата подорожі, кількість пасажирів.

Користувач має доступ до зручної форми пошуку, де значення міст вибираються зі списку доступних IATA-кодів, (див. рисунок 4.7). Це знижує ймовірність помилок під час формування запиту.

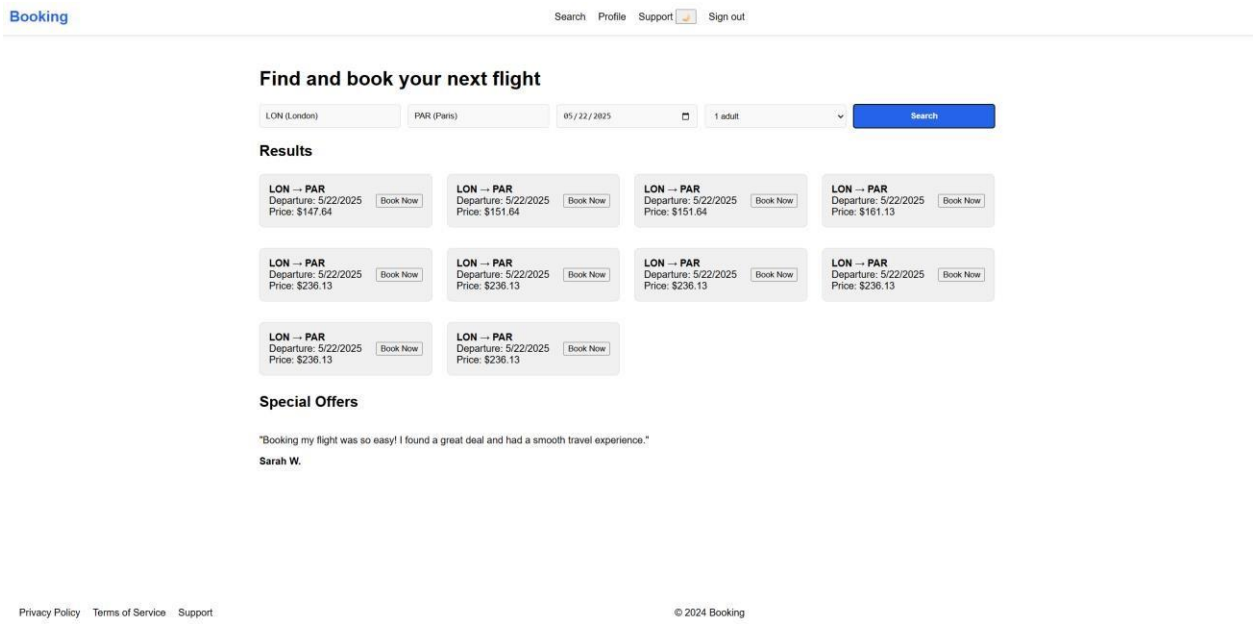


Рис. 4.7 Форма пошуку квитків

Після надсилання запиту дані перевіряються на наявність у кеші (PostgreSQL), (див. рисунок 4.8). Якщо відповідь уже збережена в системі, результат пошуку повертається миттєво, без звернення до Amadeus API.

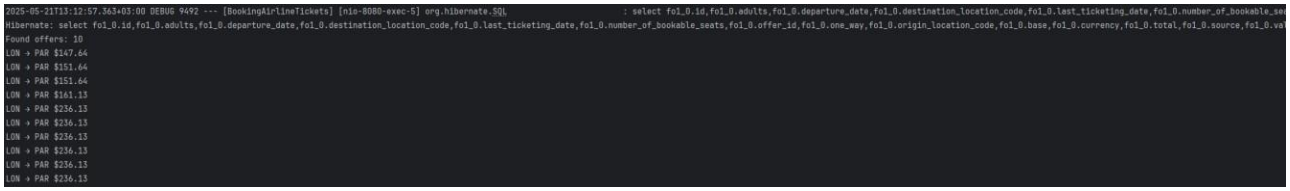


Рис. 4.8 Повідомлення про пошук у кеші

У разі відсутності результату в кеші бекенд надсилає запит до Amadeus API, отримує дані та записує їх у базу даних для подальшого повторного використання, (див. рисунок 4.9).

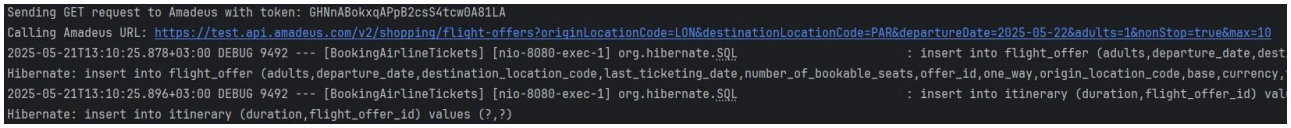


Рис. 4.9 Відповідь із Amadeus API, оброблена бекендом

Результати відображаються у вигляді карток із ціною, деталями рейсу та кнопкою «Забронювати». Це дозволяє користувачу швидко оцінити найкращі варіанти, (див. рисунок 4.10).

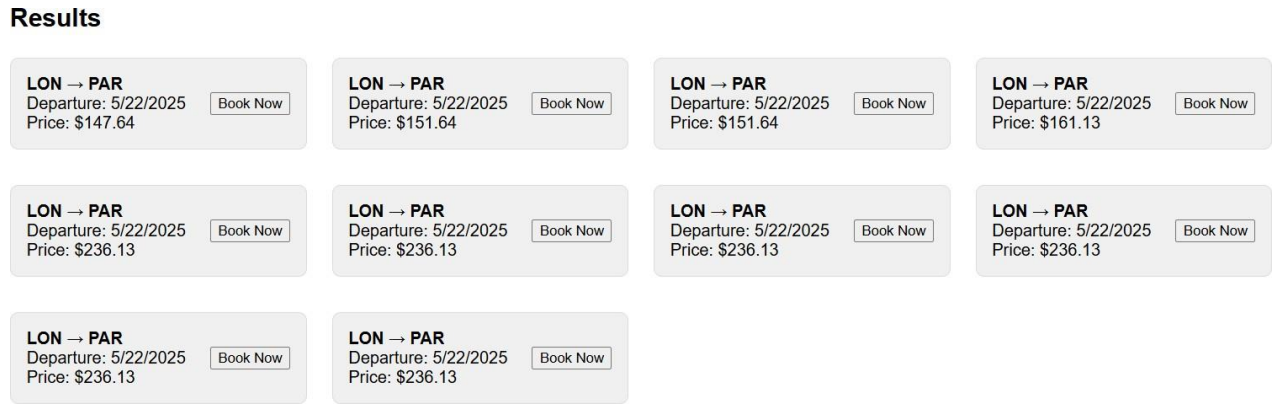


Рис. 4.10 Картки результатів пошуку з кнопками бронювання

Кешування реалізовано на рівні бази даних через сутність CachedFlightOffer. Для унікальності кешу застосовується комбінація параметрів origin-destination-date.

Технічна перевага кешування**:**

— зменшення кількості звернень до зовнішнього API;

— підвищення швидкодії (відповідь у <1 секунди з кешу);

— можливість аналітики найбільш популярних запитів.

4.3 Бронювання, перегляд, скасування та передача квитків

Після вибору авіарейсу користувач має змогу здійснити бронювання безпосередньо зі сторінки результатів пошуку. Кнопка **«**Забронювати**»** стає активною лише для авторизованих та підтверджених користувачів. Це дозволяє обмежити доступ до критично важливого функціоналу.

При натисканні кнопки відкривається модальне вікно з коротким підтвердженням дії, (див. рисунок 4.11). Користувач може натиснути **«**Так, підтвердити бронювання» або скасувати дію.

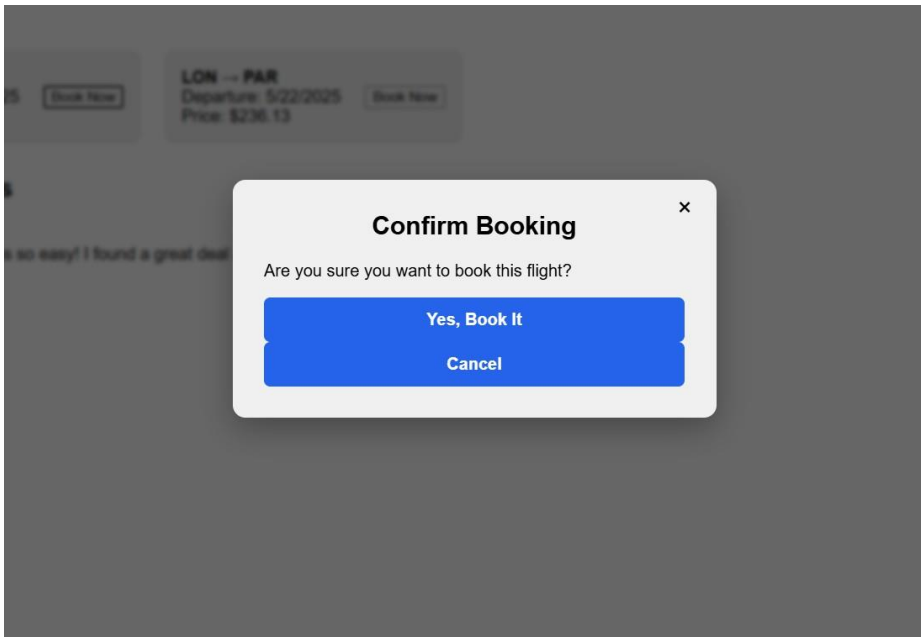


Рис. 4.11 Модальне вікно підтвердження бронювання квитка

Після підтвердження запит передається до бекенду, де створюється об’єкт бронювання (TicketBooking) і зв’язується з конкретною пропозицією рейсу (FlightOffer) та обліковим записом користувача (User). У базі фіксується час бронювання, час дії, стан квитка (активний/переданий/скасований).

Успішне бронювання підтверджується модальним повідомленням із кнопкою «Перейти до профілю», (див. рисунок 4.12).

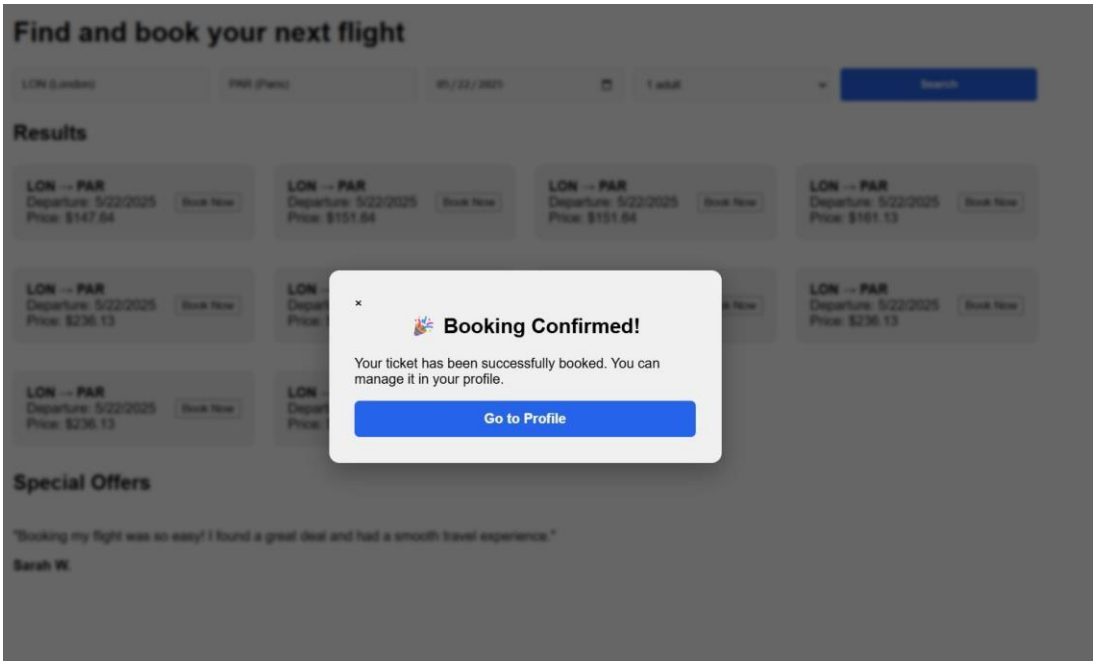


Рис. 4.12 Повідомлення про успішне бронювання з переходом до профілю

У профілі користувача доступний повний перелік його бронювань із зазначенням:

— пунктів вильоту та прибуття;

— дати рейсу;

— стану квитка (активний, переданий, скасований).

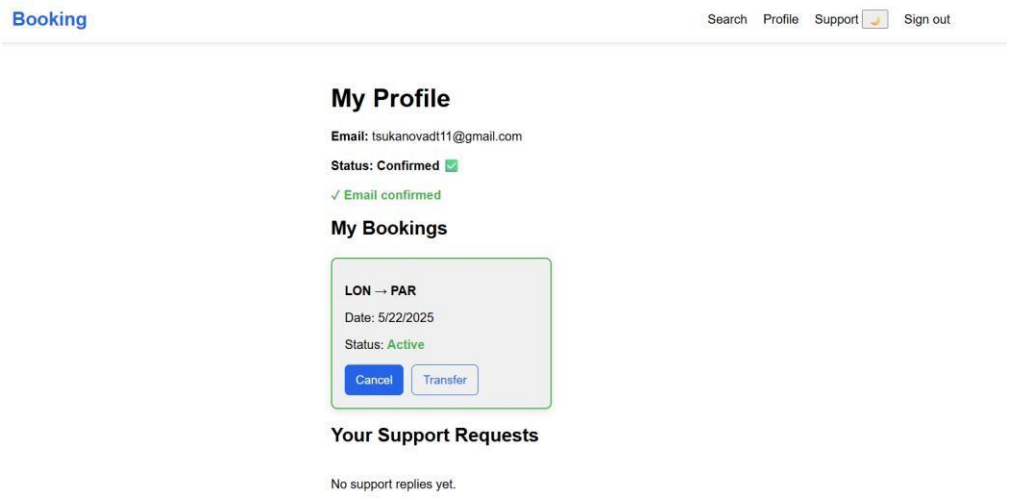


Рис. 4.13 Таблиця з активними квитками користувача

Кожен квиток має дві кнопки: «Скасувати» та «Передати». Вони неактивні, якщо дата рейсу вже минула або квиток вже не активний.

При натисканні «Скасувати» відкривається модальне вікно з підтвердженням, (див. рисунок 4.14). У разі підтвердження статус квитка змінюється на canceled, а система додає запис до журналу дій.

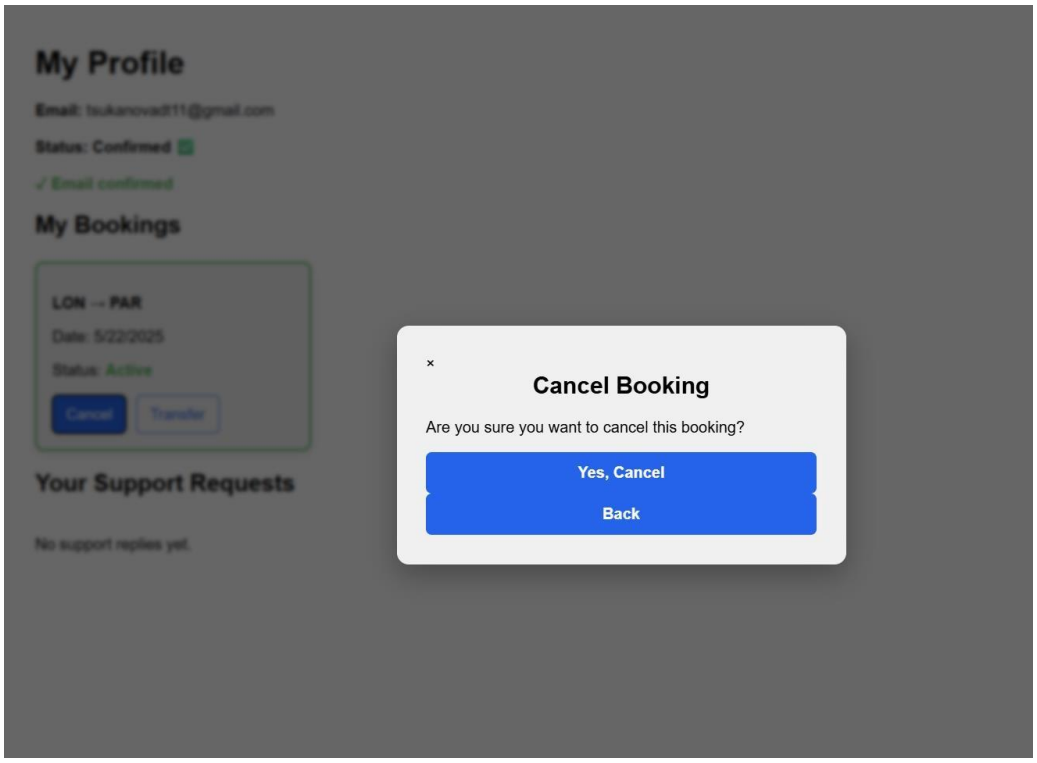


Рис. 4.14 Скасування квитка

Передача квитка реалізована через введення email іншого зареєстрованого користувача, (див. рисунок 4.15). Валідація перевіряє, чи існує такий користувач у базі.

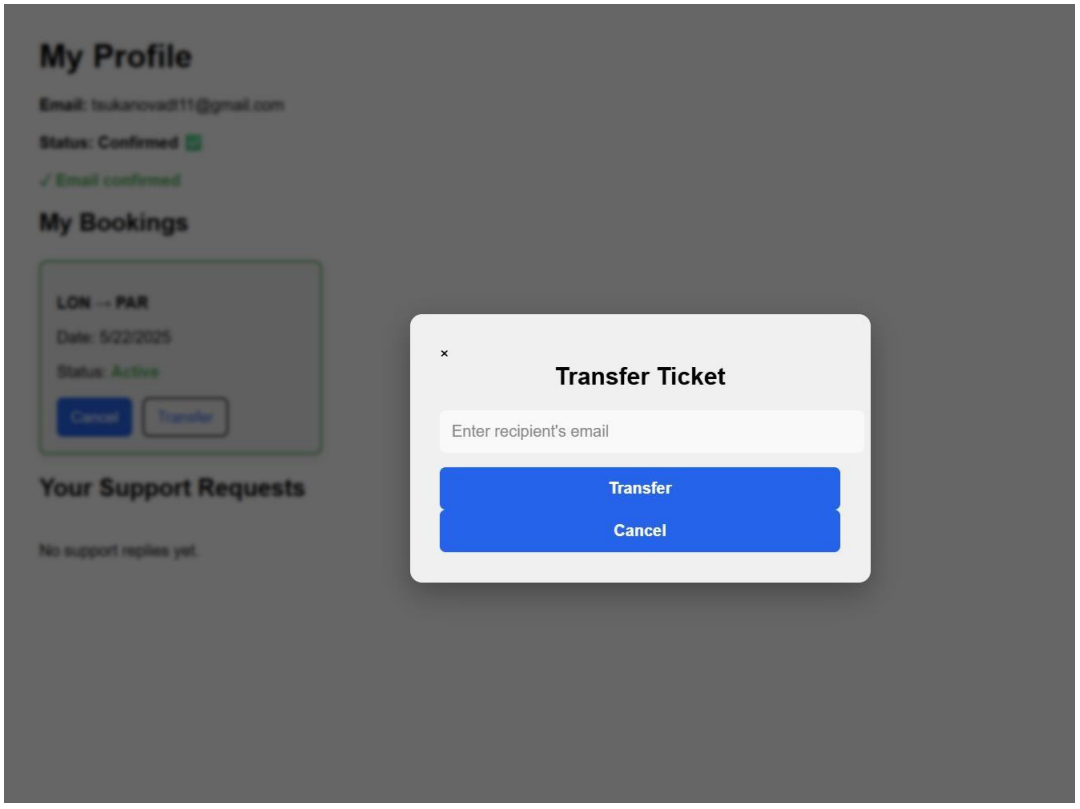


Рис. 4.15 Модальне вікно передачі квитка з полем для введення email

У разі успішної передачі квиток прив’язується до нового користувача, а колишній власник втрачає доступ до нього, (див. рисунок 4.16). У системі змінюється статус на transferred.

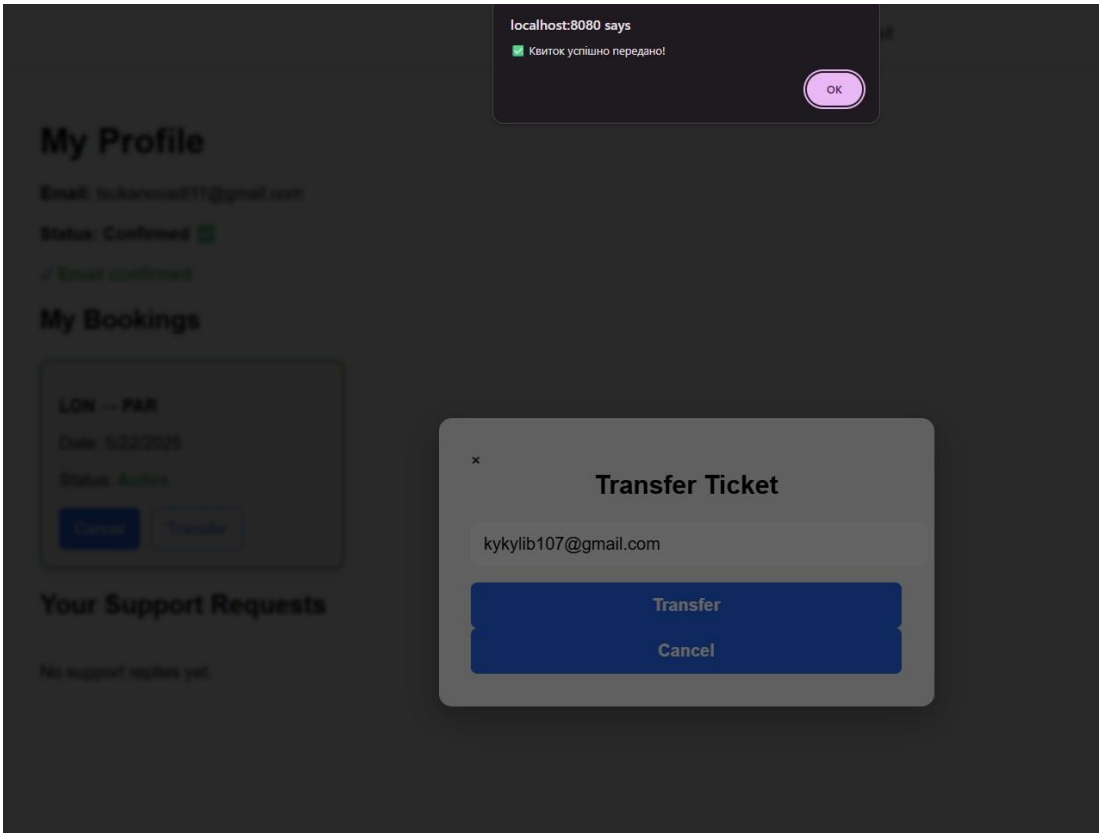


Рис. 4.16 Повідомлення про успішну передачу квитка

Такий підхід дає змогу користувачам гнучко керувати своїми квитками — включаючи зміну рішень щодо поїздки, передачу квитка родичу або колезі тощо.

Загальну послідовність дій ми можемо переглянути на (див. рисунок 4.17)

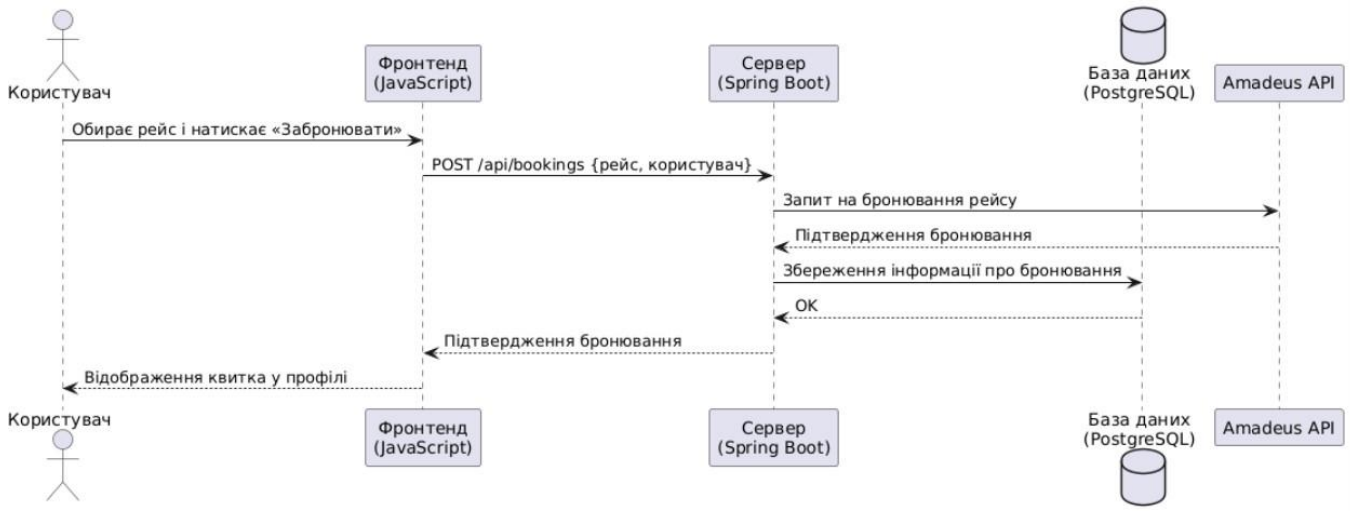


Рис. 4.17 Sequence diagram

4.4 Реалізація сторінки профілю користувача

Сторінка профілю (profile.html) — це центр керування персональним кабінетом користувача, (див. рисунок 4.18). Вона стає доступною після авторизації та забезпечує наступні функції:

– підтвердження акаунта (при першому вході);

– перегляд усіх активних, скасованих або переданих квитків;

– взаємодія з підтримкою (надсилання та перегляд звернень).

Після переходу на сторінку профілю, система автоматично підтягує JWT-токен із localStorage, ідентифікує користувача та запитує список його квитків через /api/tickets/my.

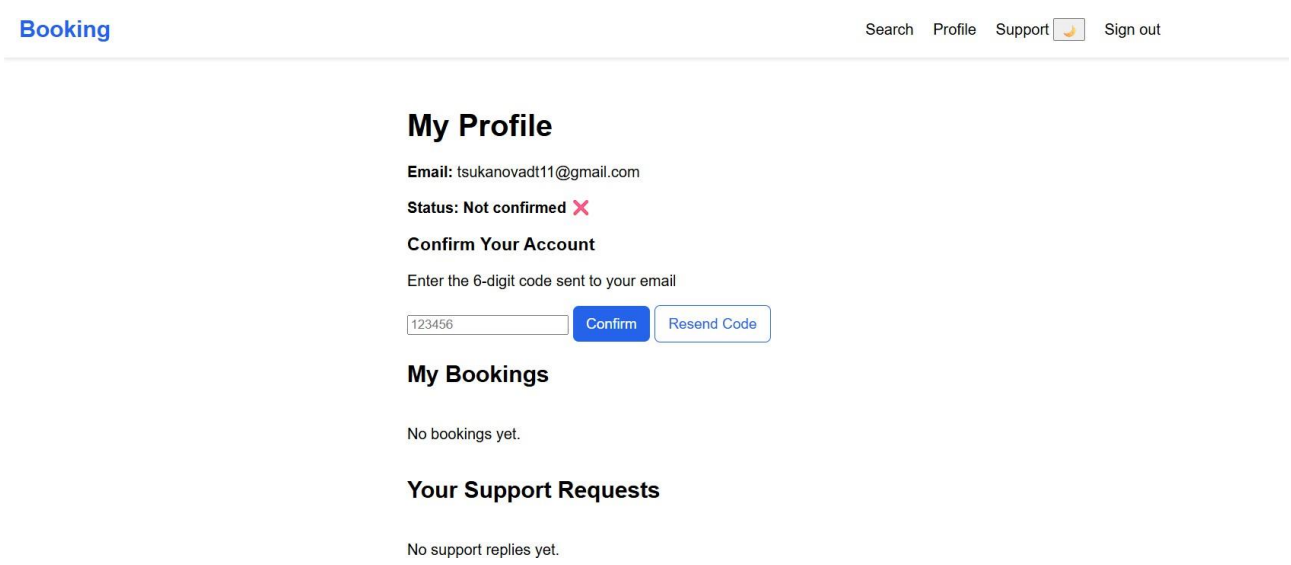


Рис. 4.18 Форма підтвердження email у профілі

Якщо підтвердження вже пройдено, цей блок приховується, і користувач має доступ до функцій управління квитками, (див. рисунок 4.19). Кожен квиток у таблиці має чіткий інтерфейс:

— інформація про рейс (напрямок, дата, авіакомпанія);

— кнопки дій (якщо дозволено) — скасування або передача;

— статус виконання (активний, переданий, прострочений).

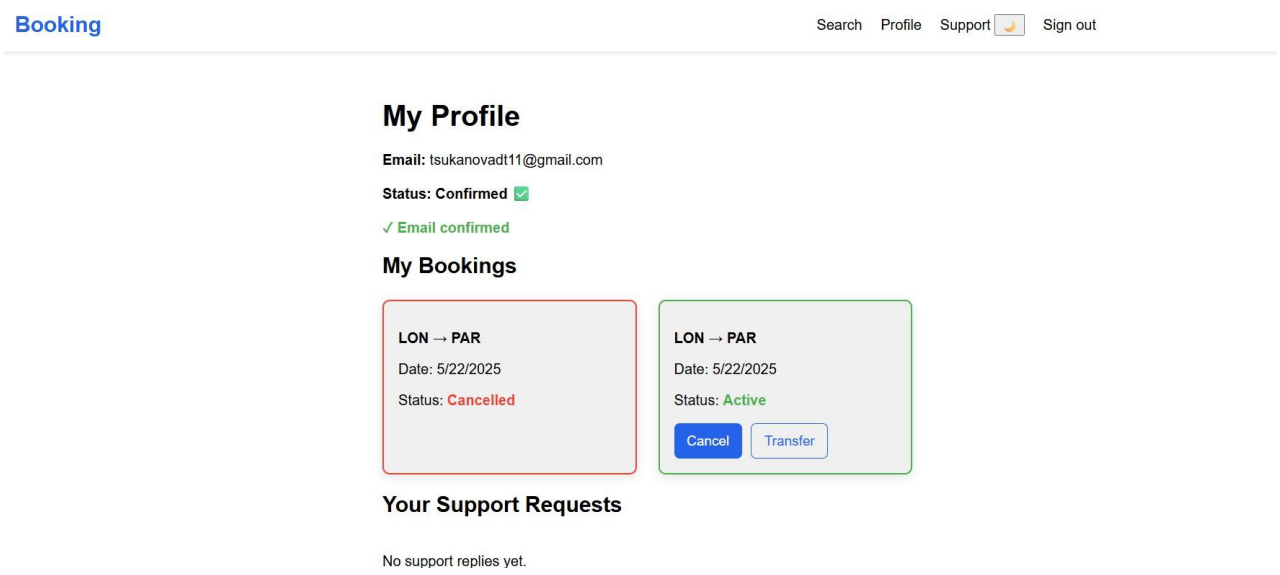


Рис. 4.19 Таблиця квитків із діями та статусами

На тій же сторінці реалізовано інтерфейс взаємодії з підтримкою: форма надсилання звернення, а також список відповідей адміністратора (якщо вони є), (див. рисунок 4.20).

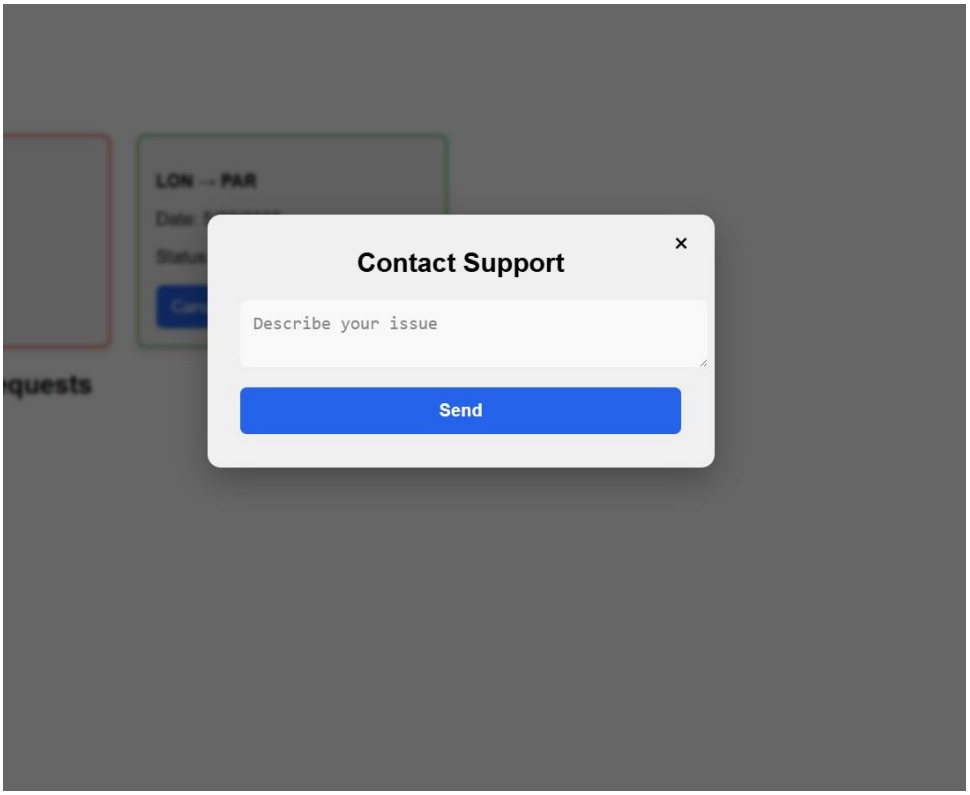


Рис. 4.20 Форма звернення до технічної підтримки

Також реалізовано механізм повідомлень: якщо дію виконано успішно (наприклад, передача квитка або скасування), система показує відповідне повідомлення вгорі сторінки, (див. рисунок 4.21).

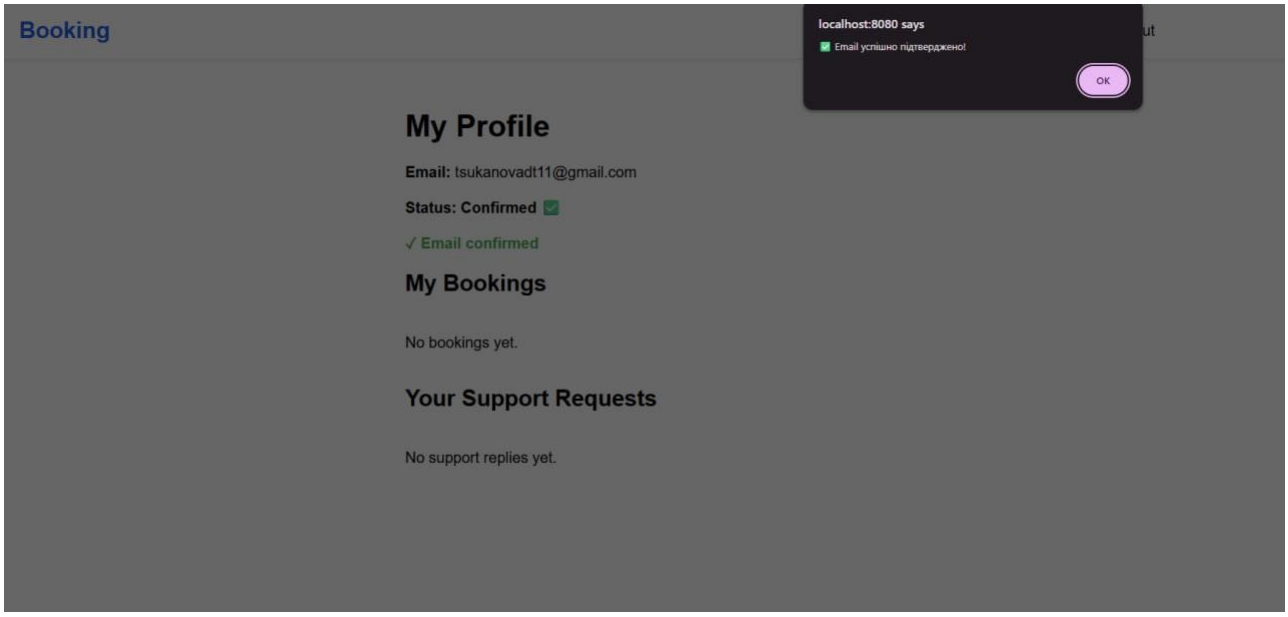


Рис. 4.21 Повідомлення про успішну дію у профілі користувача

Таким чином, сторінка профілю виступає як особистий хаб для користувача, де він може керувати бронюваннями, підтримкою та безпекою облікового запису.

4.5 Панель адміністратора: функції, права доступу, керування

Панель адміністратора реалізована на окремій сторінці admin.html, яка доступна лише користувачам із роллю ADMIN, (див. рисунок 4.22). Захист доступу реалізовано як на фронтенді (через перевірку ролі у JWT-токені), так і на бекенді (через Spring Security з авторитетом "ADMIN").

Після успішного входу адміністратор отримує доступ до таких функцій:

— перегляд списку всіх зареєстрованих користувачів;

— блокування та розблокування акаунтів;

— зміна ролі користувача (USER/ADMIN);

— пошук користувача за email;

— перегляд усіх бронювань;

— очищення кешу запитів до Amadeus API;

— перегляд і відповіді на звернення від користувачів.

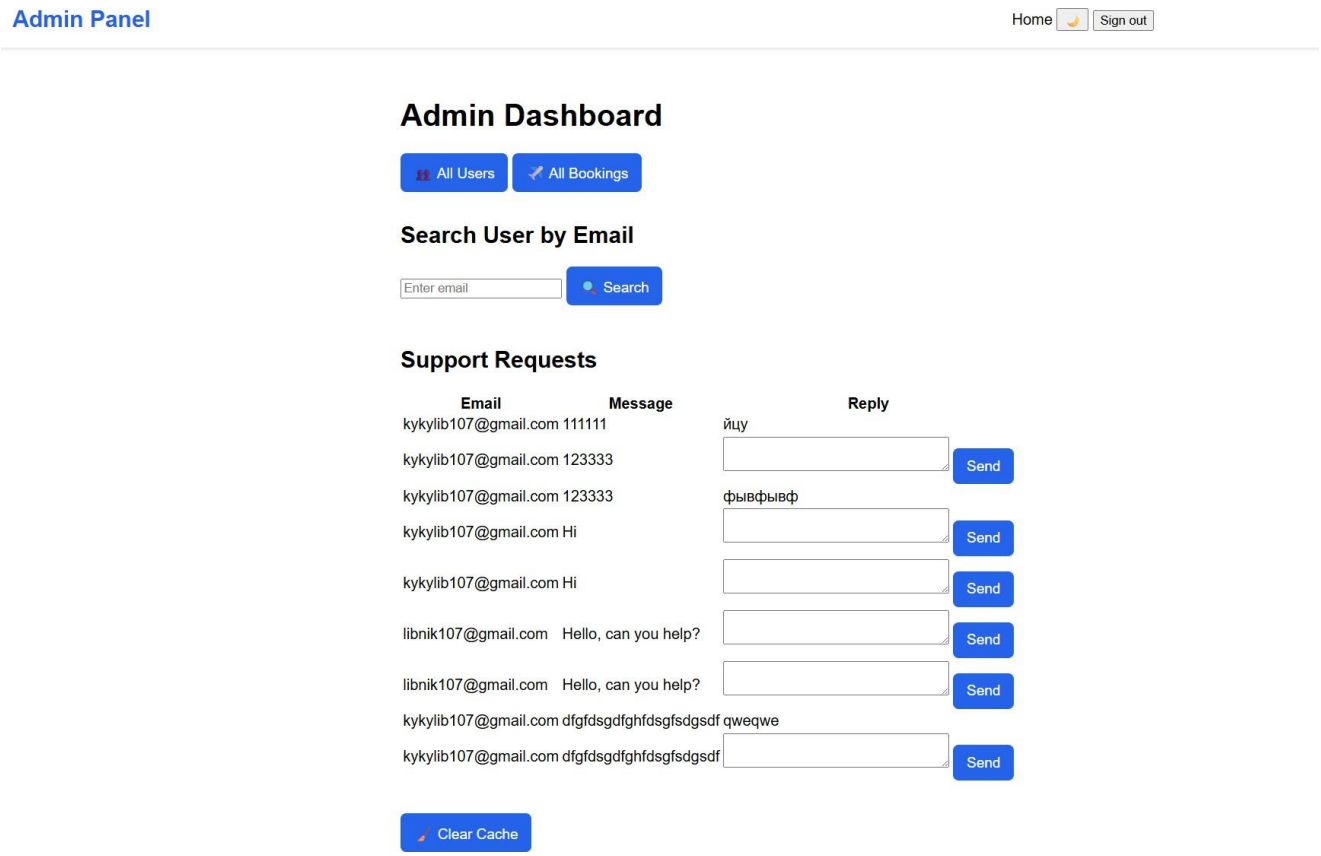


Рис. 4.22 Інтерфейс панелі адміністратора після входу

У таблиці користувачів доступна базова інформація, (див. рисунок 4.23) (ID, email, статус підтвердження, роль) і кнопки дій:

— блокування/розблокування (зміна поля enabled);

— зміна ролі (перемикання між USER та ADMIN).

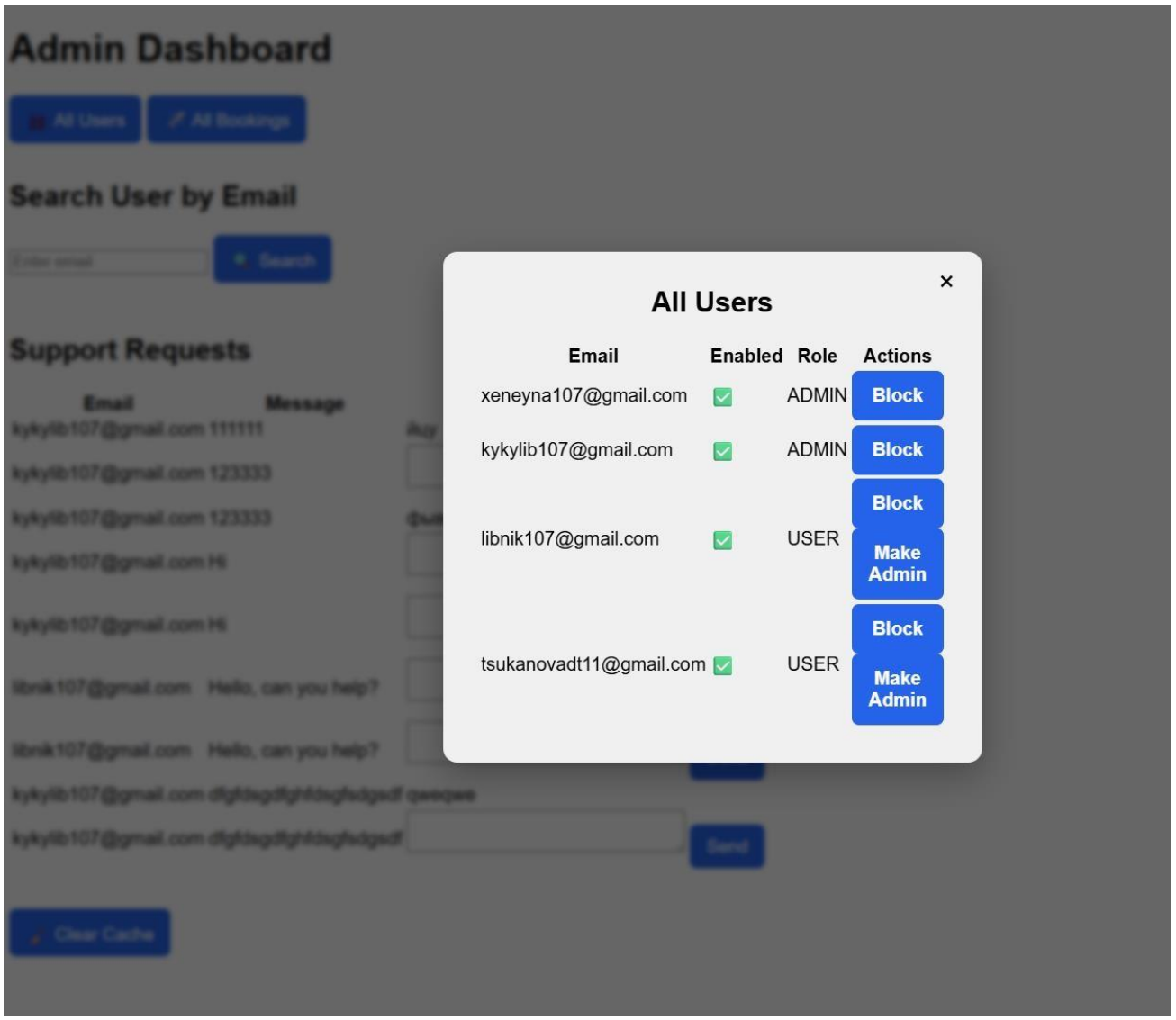


Рис. 4.23 Таблиця користувачів із керуванням

Перегляд бронювань дає змогу адміністратору отримати повну картину використання системи: скільки квитків активні, які скасовано, ким і коли було здійснено бронювання, (див. рисунок 4.24).

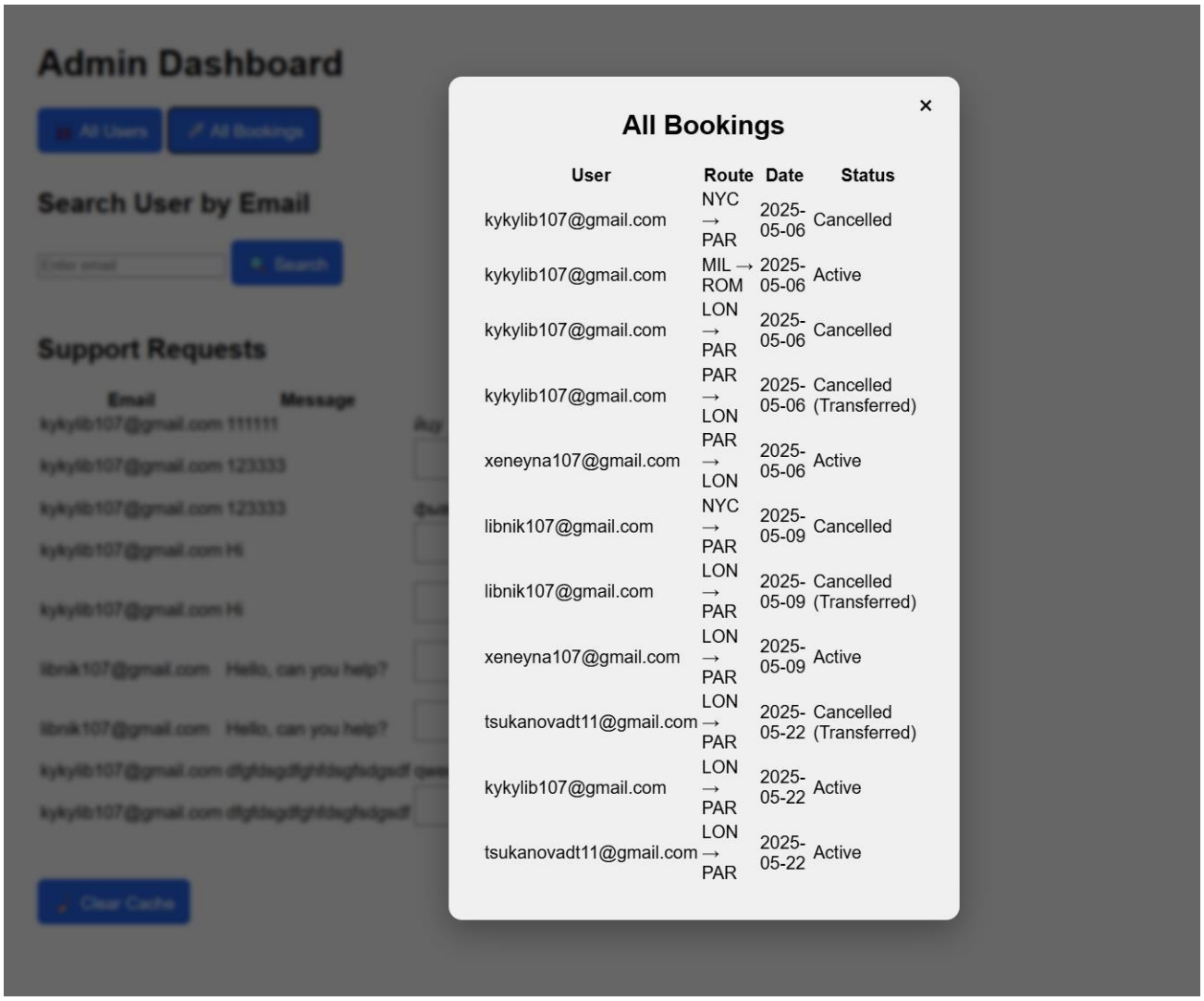


Рис. 4.24 Таблиця всіх бронювань у панелі адміністратора

Додатково реалізовано кнопку «Очистити кеш запитів», що викликає backend-метод для видалення застарілих збережених відповідей на запити пошуку, які зберігалися у базі з метою пришвидшення, (див. рисунок 4.25).

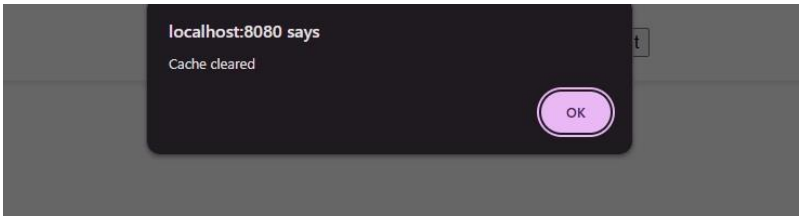


Рис. 4.25 Повідомлення про успішне очищення кешу

Це дозволяє адміністраторам підтримувати актуальність інформації у системі та швидко реагувати на зміни в даних авіакомпаній.

Також реалізована система пошука по email, тобто адміністратор має можливість шукати користувачів за email (див. рисунок 4.26).

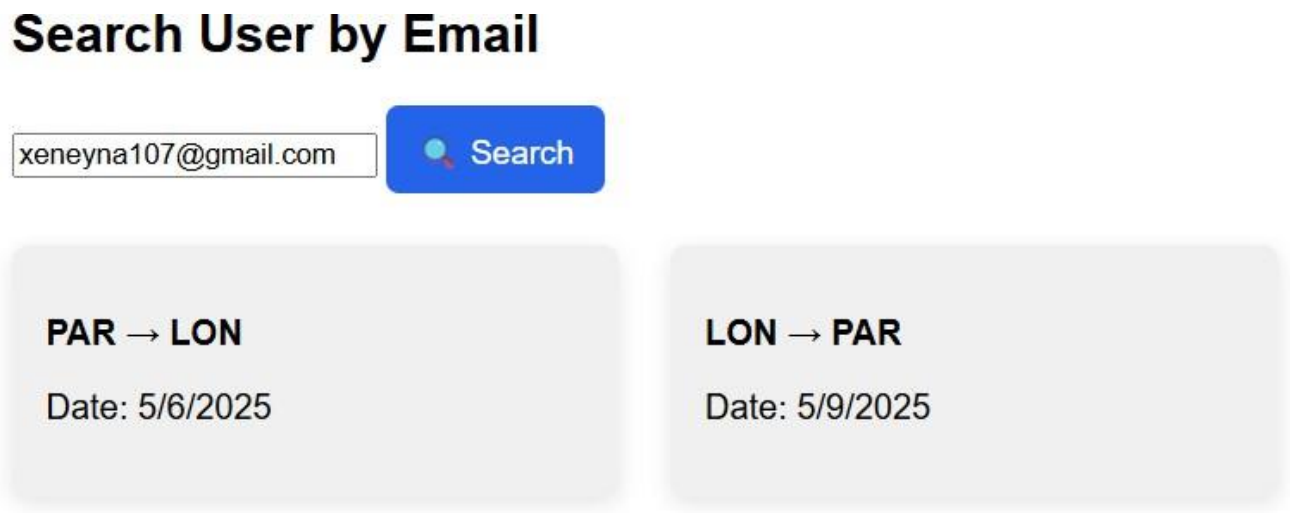


Рис. 4.26 Інтерфейс пошуку

4.6 Підтримка користувачів: обробка звернень

Модуль підтримки реалізовано як просту систему обміну повідомленнями між користувачем і адміністратором. З боку користувача (у profile.html) передбачено форму подання звернення, де вказується тема та зміст повідомлення.

Звернення зберігаються у таблиці support_requests і містять такі поля:

— id звернення;
— email користувача;
— тема звернення;
— текст;
— статус (оброблено/не оброблено);
— відповідь адміністратора.

На стороні адміністратора всі звернення виводяться у вигляді таблиці з можливістю відповіді, (див. рисунок 4.27).

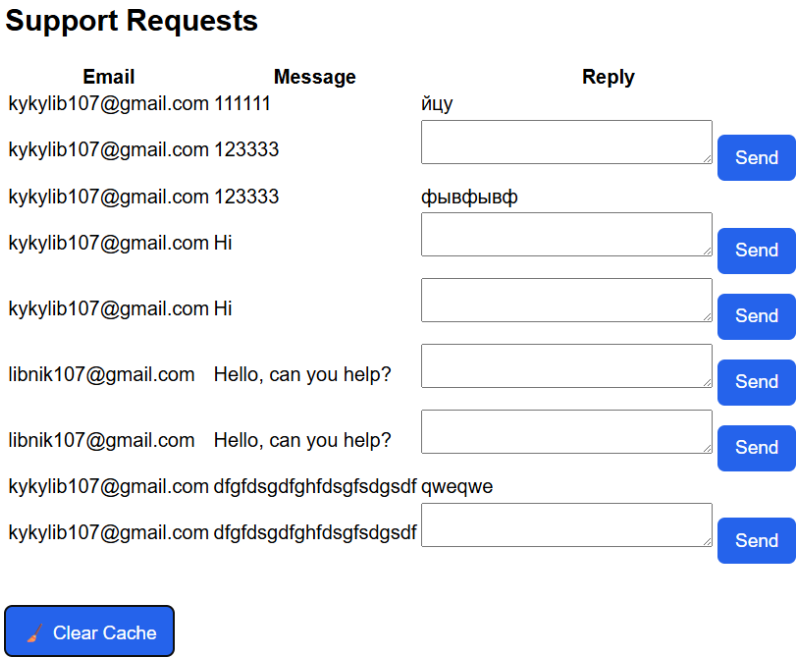


Рис. 4.27 Список звернень користувачів у панелі адміністратора

Після натискання кнопки «Відповісти» відкривається модальне вікно, де можна ввести відповідь. Вона буде показана користувачеві в його профілі.

Завдяки цій системі підтримка стала інтегрованою частиною вебзастосунку, а користувачі можуть отримати допомогу без потреби звертатись через сторонні сервіси.

4.7 Тестування функціональних модулів застосунку

Тестування є критично важливою частиною розробки, оскільки дозволяє переконатися у правильності роботи всіх компонентів системи. У рамках практики було проведено ручне та частково автоматизоване тестування.

Ручне тестування

Тестування проводилось із боку користувача через браузер на реальному інтерфейсі застосунку:

— реєстрація нового користувача;

— вхід та підтвердження акаунта;

— пошук рейсів;
— бронювання квитків;
— передача квитка іншому користувачу;
— скасування квитків;
— надсилання звернень у підтримку;
— доступ до адмінпанелі з перевіркою обмежень.

Таблиця 4.1

Тест-кейси форми підтвердження акаунта

№ тест-кейсу	OK/NOK	Дії	Очікуваний результат
1	OK	1. Відкрити сторінку реєстрації. 2. Ввести електронну пошту, пароль. 3. Натиснути кнопку «Зареєструватися».	1. Відображається повідомлення про успішну реєстрацію. 2. На пошту надходить лист із кодом підтвердження.
2	OK	1. Ввести email та пароль на сторінці входу. 2. Натиснути «Увійти». 3. Ввести код підтвердження з пошти.	1. Успішний вхід до профілю. 2. Обліковий запис підтверджено.
3	OK	1. Вибрати міста та дату на головній сторінці. 2. Натиснути «Пошук рейсів».	1. Відображаються знайдені рейси відповідно до введених параметрів.
4	OK	1. Натиснути кнопку «Забронювати» на картці рейсу. 2. Підтвердити дію у модальному вікні.	1. З’являється повідомлення про успішне бронювання. 2. Квиток зберігається у профілі.

Продовження таблиці 4.1

Тест-кейси форми підтвердження акаунта

№ тест-кейсу	OK/NOK	Дії	Очікуваний результат
5	OK	1. Перейти до профілю. 2. Натиснути «Передати» біля квитка. 3. Ввести email іншого користувача та підтвердити.	1. Квиток передано іншому користувачу. 2. Власник змінено.
6	OK	1. Перейти до профілю. 2. Натиснути «Скасувати» біля квитка. 3. Підтвердити дію.	1. Квиток позначено як неактивний. 2. Видалено з активного списку.
7	OK	1. Перейти до вкладки «Підтримка». 2. Ввести повідомлення та надіслати.	1. Повідомлення збережено. 2. Відображається у списку запитів.
8	OK	1. Авторизуватися як адміністратор. 2. Перейти на сторінку admin.html	1. Відображається адмінпанель. 2. Доступні дії лише для користувачів з роллю ADMIN.

Тестування API

Для перевірки бекенду використовувалися Postman та Insomnia. Запити тестували такі сценарії:

— отримання переліку рейсів (GET /api/flights/search);

— створення бронювання (POST /api/tickets/book);

— скасування та передача квитків;

— робота з JWT-токеном (перевірка обмеження доступу).

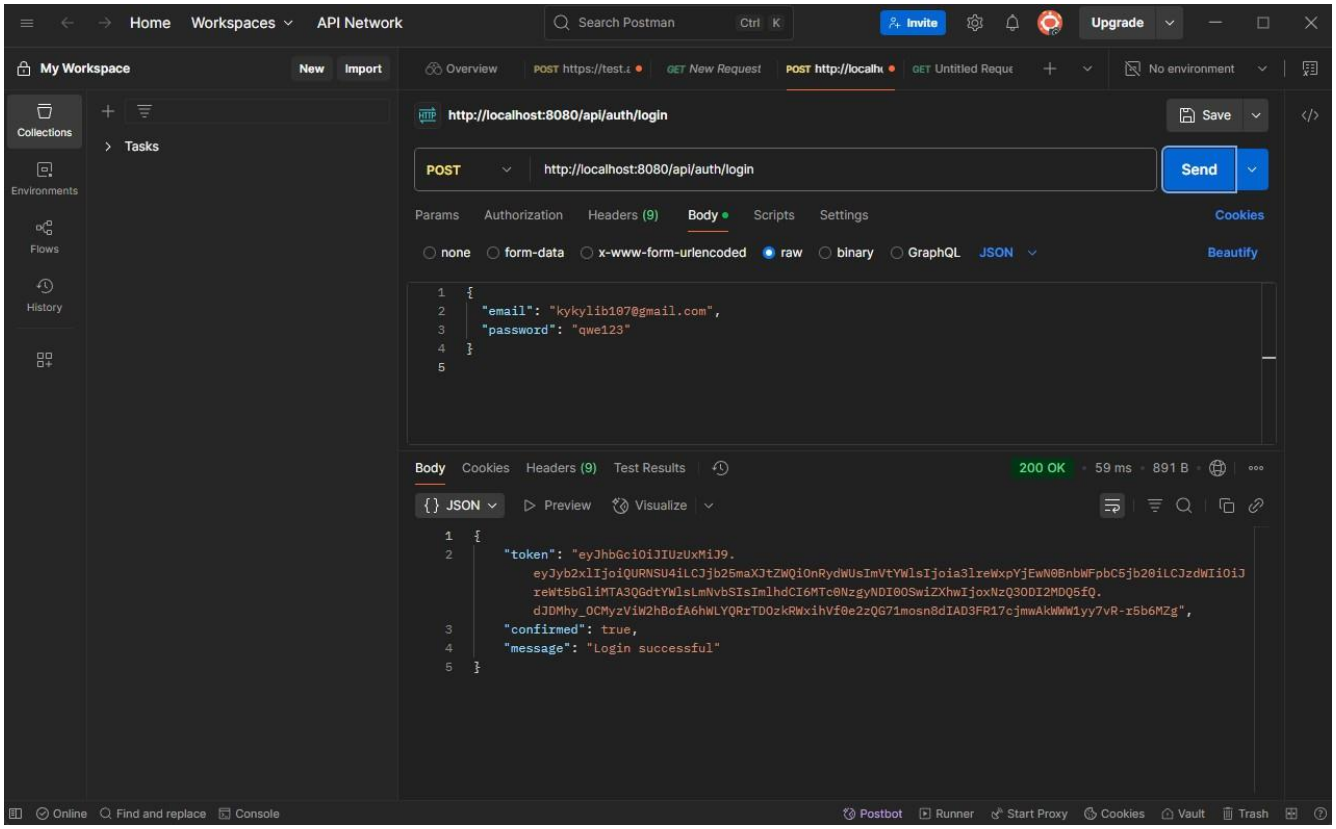


Рис. 4.28 Приклад тестового API-запиту в Postman

Валідація безпеки

Було перевірено:

— неможливість доступу до панелі адміністратора з роллю USER;

— відсутність уразливості типу SQL Injection (через параметризовані запити JPA);

— перевірку токена на бекенді (JwtAuthenticationFilter);

— захист endpoint'ів через .hasAuthority("ADMIN") або .authenticated().

Загальну картину основної логіки можно побачити на (див. рисунок 4.29)

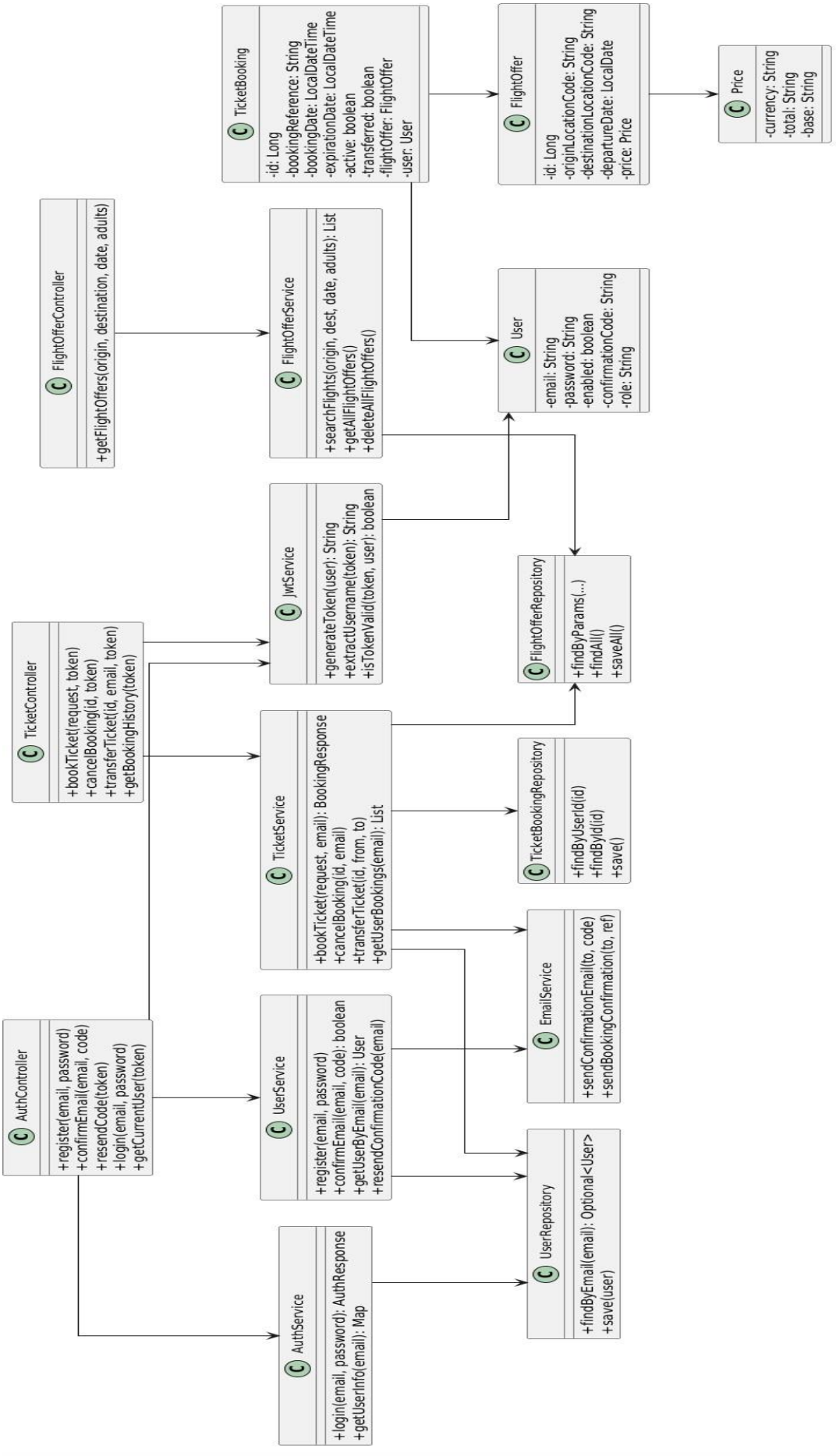


Рис. 4.29 Діаграма класів основної логіки

ВИСНОВКИ

У рамках кваліфікаційної роботи було виконано повний цикл розробки вебзастосунку для бронювання авіаквитків, що включав аналіз предметної галузі, дослідження аналогів, проєктування архітектури, реалізацію серверної та клієнтської частини, а також тестування функціональних модулів.

На початковому етапі було визначено актуальність задачі автоматизації процесу бронювання квитків через вебінтерфейс, з урахуванням сучасних потреб користувачів у гнучкості, зручності та безпеці. Проведено огляд існуючих сервісів, таких як Skyscanner, Kiwi.com, Google Flights, що дозволило виявити недоліки типових рішень — відсутність можливості передачі квитків, обмеженість адміністрування, відсутність кешування даних тощо.

З урахуванням результатів аналізу сформульовано чіткі функціональні та нефункціональні вимоги до власного вебзастосунку: підтримка реєстрації та підтвердження акаунта, пошук і бронювання квитків, передача або скасування бронювань, реалізація особистого кабінету та адміністрування. Окремо було передбачено механізм зворотного зв’язку для підтримки користувачів.

Для реалізації використано сучасний технологічний стек: Java + Spring Boot для бекенду, PostgreSQL для зберігання даних, HTML/CSS/JS для клієнтської частини, Amadeus API для отримання актуальних авіарейсів, JWT для авторизації, а також механізм кешування запитів у базі даних.

Серверна частина побудована на REST-архітектурі, з чітким поділом логіки на контролери, сервіси, репозиторії та DTO. Усі маршрути захищено авторизацією через JWT-токени та обмежено за ролями користувачів. Додано механізми безпечного підтвердження акаунта через код підтвердження.

Клієнтська частина представлена двома основними сторінками — index.html (пошук і бронювання) та profile.html (керування акаунтом), а також окремою admin.html для адміністраторів. Застосунок підтримує темну та світлу тему,

адаптивну верстку, модальні вікна, перевірку авторизації, інтерактивні повідомлення.

Реалізовано повний набір функціональності:

— бронювання з підтвердженням;

— перегляд списку квитків;

— скасування та передача квитка за email;

— панель адміністрування з блокуванням користувачів;

— кешування результатів пошуку для зменшення навантаження;

Здійснено комплексне тестування: ручне тестування інтерфейсів, API-запити через Postman, перевірка безпеки, доступності функцій за ролями. Вебзастосунок успішно пройшов перевірку всіх ключових сценаріїв і продемонстрував стабільність та відповідність вимогам.

Таким чином, розроблений вебзастосунок повністю відповідає поставленій меті — автоматизації процесу бронювання авіаквитків через зручний, захищений та функціонально багатий вебінтерфейс. Він має практичну цінність як для користувачів, так і для адміністраторів системи, і може бути основою для подальшого розширення (інтеграція оплат, багатомовність, мобільна адаптація.

ПЕРЕЛІК ПОСИЛАНЬ

1. Amadeus for Developers. Flight Offers Search API. URL: https://developers.amadeus.com/self-service/category/air/api-doc/flight-offers-search
2. Skyscanner for Business. Travel APIs. URL: https://partners.skyscanner.net/apis
3. Google Flights – How it works. URL: https://www.google.com/flights
4. Степаненко О. В. Розробка вебзастосунків з використанням Java та Spring Boot. Наукові праці НТУУ «КПІ». 2022. №1. С. 88–92.
5. Kiwi.com API Documentation. URL: https://docs.kiwi.com
6. Nguyen P., Nguyen T. Designing scalable flight booking systems: architecture and performance overview. *International Journal of Web Information Systems.* 2021. Vol. 17(4). P. 321–336.
7. PostgreSQL Documentation: Introduction and Concepts. URL: https://www.postgresql.org/docs/
8. Spring Security Reference Documentation. URL: https://docs.spring.io/spring-security/reference/index.html
9. JWT.io: JSON Web Token Introduction. URL: https://jwt.io/introduction
10. OpenAPI Specification. Swagger Documentation. URL: https://swagger.io/specification/
11. Fowler M. *Patterns of Enterprise Application Architecture.* Addison-Wesley, 2002. ISBN 978-0321127426.
12. Baeldung. Introduction to Spring Boot REST APIs. URL: https://www.baeldung.com/rest-with-spring-series
13. Mozilla Developer Network (MDN). Web security basics. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Web_applications/Security_basics
14. Gavin D. *RESTful Java with JAX-RS 2.0.* O'Reilly Media, 2013. ISBN: 9781449379791.

15. Spring Boot Reference Guide. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/

16. Amadeus Blog. How Amadeus APIs help build modern travel apps. URL: https://developers.amadeus.com/blog

17. OWASP Foundation. Top 10 Web Application Security Risks. URL: https://owasp.org/www-project-top-ten/

18. W3C. HTML & CSS Standards. URL: https://www.w3.org/standards/webdesign/htmlcss

19. JetBrains. IntelliJ IDEA для Spring Boot розробки. URL: https://www.jetbrains.com/idea/features/spring.html

20. GitHub Docs. REST API development workflow and security. URL: https://docs.github.com/en/rest

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка програмних засобів для збору інформації для букінгу авіабілетів

Виконав студент 4 курсу
групи ПД-41
Ліберман Нікіта Михайлович
Керівник роботи
старший викладач кафедри ІПЗ Андрусевич Наталя Володимирівна

Київ – 2025

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – Автоматизація бронювання авіаквитків за допомогою вебзастосунку.

Об'єкт дослідження – Процес бронювання авіаквитків у системах онлайн-бронювання.

Предмет дослідження - Методи та засоби автоматизації бронювання авіаквитків за допомогою вебзастосунку.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз літературних джерел та сучасних вебрішень у сфері онлайн-бронювання авіаквитків, виявити основні тенденції та потреби користувачів у таких системах.

2. Здійснити огляд і аналіз популярних сервісів для бронювання авіаквитків (наприклад, Skyscanner, Kiwi), визначити їх переваги, недоліки та функціональні особливості.

3. Провести огляд сучасних засобів і технологій для розробки вебзастосунків, зокрема фреймворку Spring Boot, бази даних PostgreSQL, Amadeus API, інструментів для безпеки та авторизації.
4. Сформувати функціональні та нефункціональні вимоги до вебзастосунку для бронювання авіаквитків.

5. Спроєктувати архітектуру клієнт-серверного вебзастосунку з урахуванням логіки взаємодії з користувачем, адміністратором та стороннім API (Amadeus).

6. Розробити вебзастосунок для бронювання авіаквитків, реалізувавши такі функції: реєстрація, авторизація, підтвердження акаунта, пошук рейсів, передача квитків, бронювання квитків, управління бронюваннями, адміністративна панель.

7. Провести модульне та інтеграційне тестування вебзастосунку для перевірки коректності його роботи в різних сценаріях використання.

АНАЛІЗ АНАЛОГІВ

Характеристика	Skyscanner	Kiwi.com	Google Flights	Amadeus API	Aviasales	Booking.com
Відкрите API	Так	Ні	Ні	Так	Так	Так
Можливість бронювання	Ні	Так	Ні	Так	Частково	Так
Підтримка передачі квитків	Ні	Ні	Ні	Ні	Ні	Так
Підтримка кешування	Частково	Так	Так	Так	Ні	Так
Адмін-панель	Ні	Ні	Ні	Ні	Ні	Так
Гнучкість інтеграції	Середня	Низька	Низька	Висока	Висока	Середня

ВИМОГИ ДО ЗАСТОСУНКУ

Функціональні вимоги:

1. Реєстрація та авторизація користувачів.
2. Можливість заходити в особистий кабінет.
3. Пошук авіарейсів.
4. Бронювання квитків.
5. Можливість передачі квитків іншим користувачам.
6. Адмінпанель.

Нефункціональні вимоги:

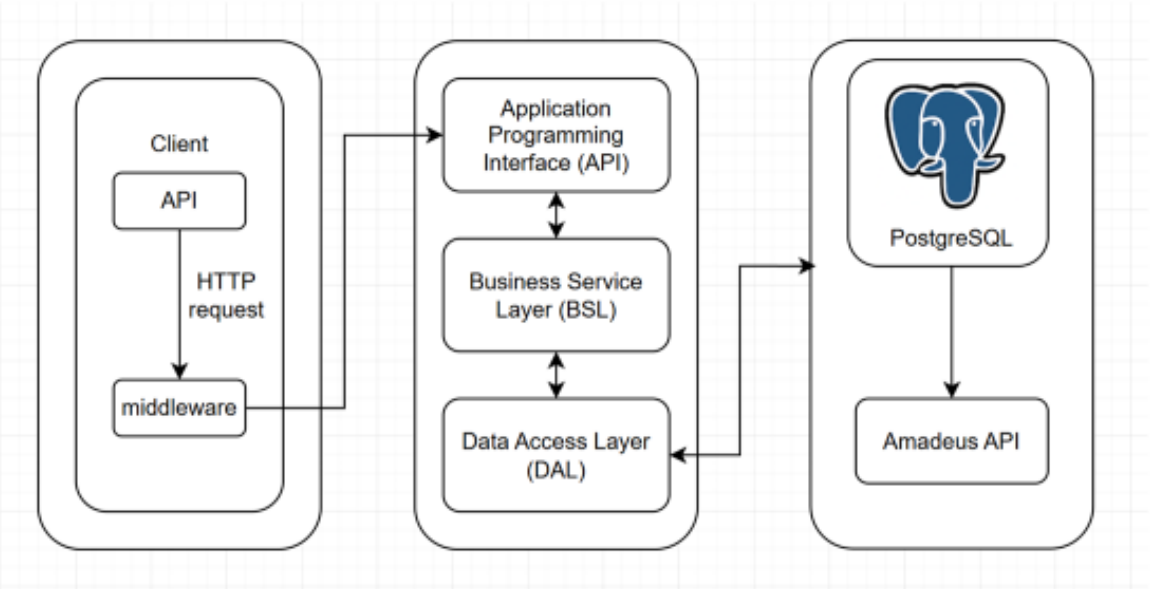
1. Захист даних.
2. Інтеграція з API для актуальних даних про рейси.
3. Швидкий пошук рейсів (~ 1сек).
4. Резервне копіювання даних.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

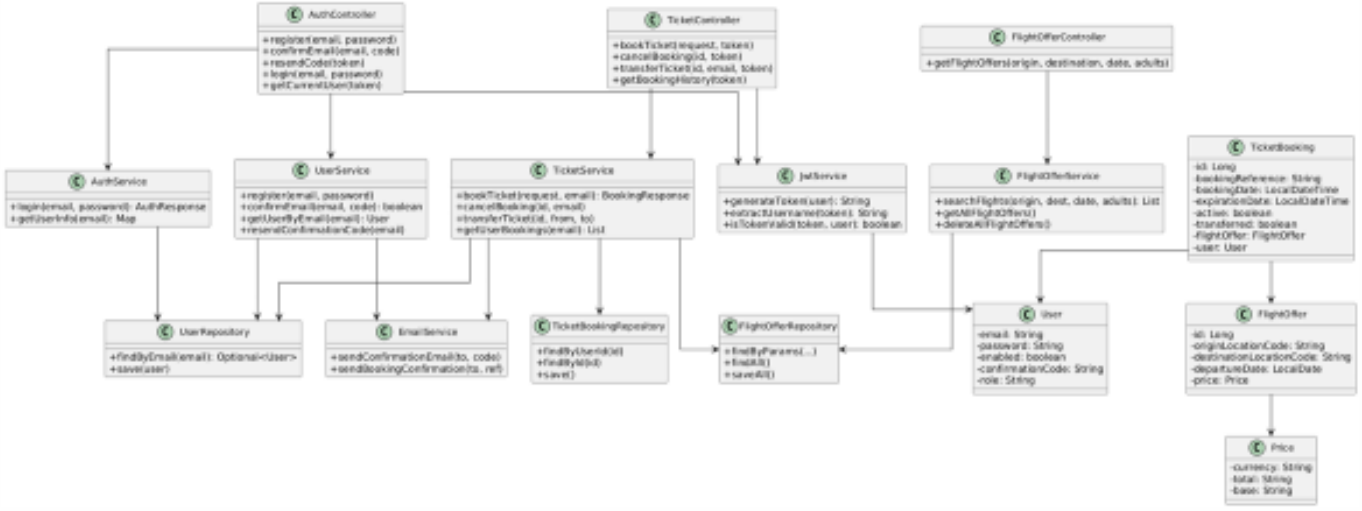
6

АРХІТЕКТУРА ЗАСТОСУНКУ



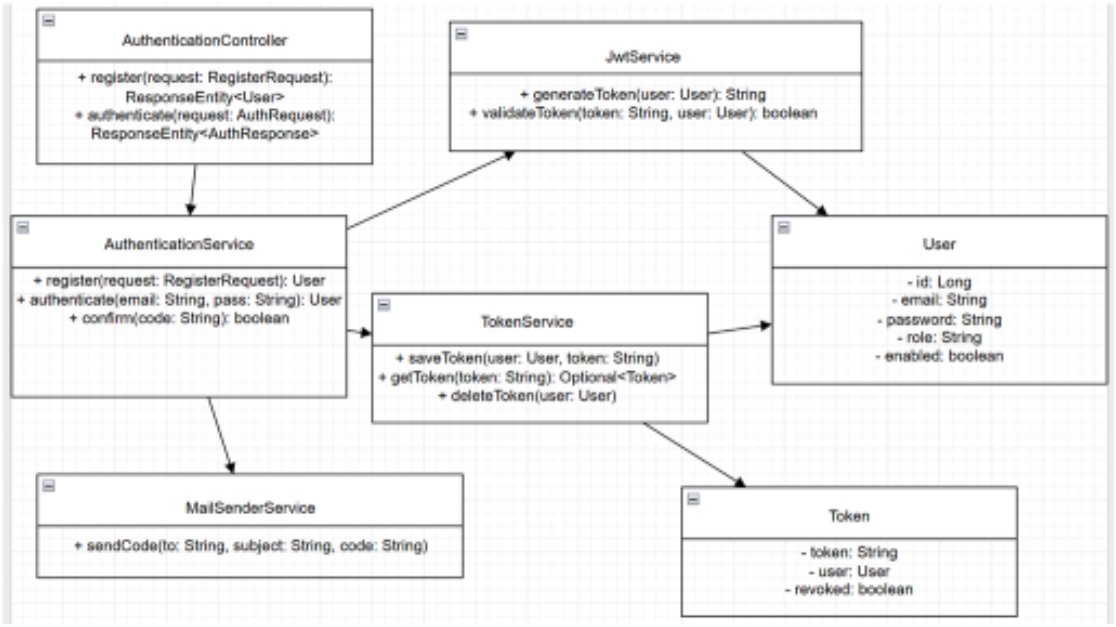
7

ДІАГРАМА КЛАСІВ ОСНОВНОЇ ЛОГІКИ



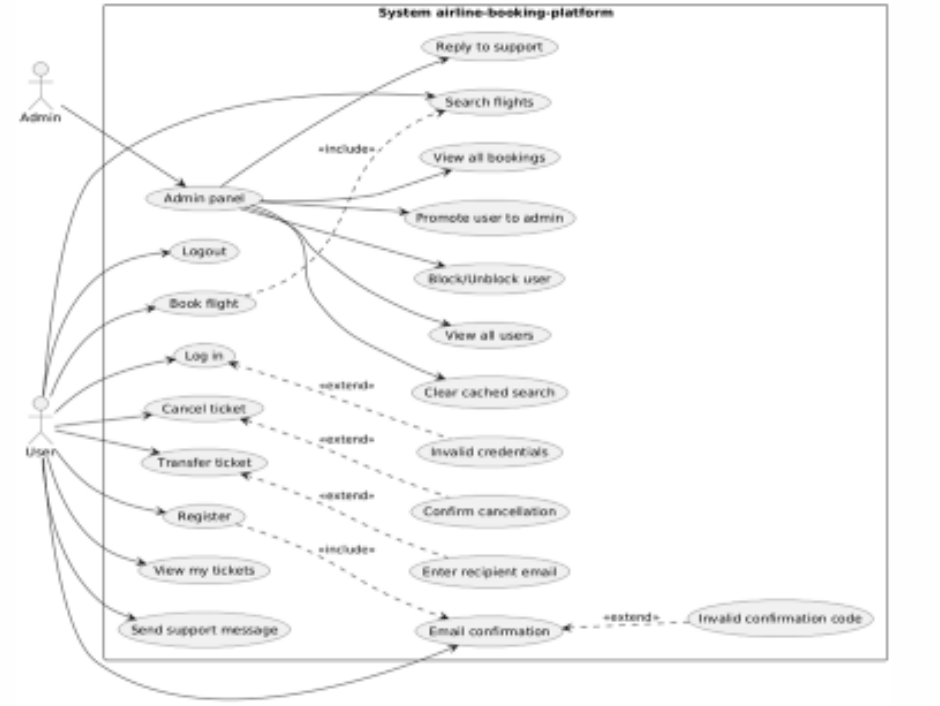
8

ДІАГРАМА КЛАСІВ АВТОРИЗАЦІЇ



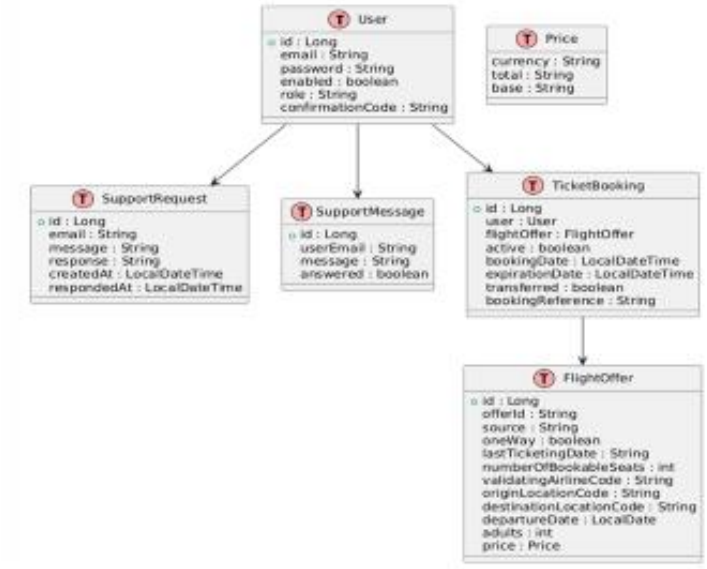
9

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



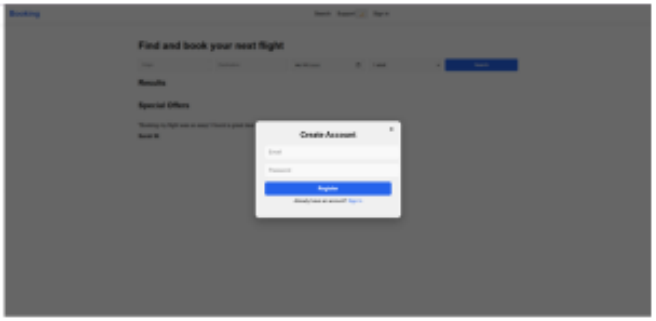
10

СТРУКТУРА БАЗИ ДАННИХ



11

ЕКРАННІ ФОРМИ



Головна сторінка

Сторінка реєстрації

12

ЕКРАННІ ФОРМИ

Сторінка користувача

Сторінка адміністратора

13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1.Ліберман Н.М., Андрусевич Н.В. Розробка програмних засобів для збору інформації для букінгу авіабілетів // Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.25, ДУКІТ, Київ, Україна, с.…(подано до друку)

2.Ліберман Н.М. Захист інформації у web-застосунку для бронювання авіаквитків // Матеріали XII Всеукраїнської науково-практичної конференції молодих учених «Інформаційні технології – 2025», 15.05.25, КСУ імені Бориса Грінченка, Київ, Україна, с.…(подано до друку)

14

ВИСНОВКИ

1. Проведено аналіз літературних джерел та сучасних рішень у сфері онлайн-бронювання авіаквитків, що дозволило виявити ключові тенденції, потреби користувачів та напрямки розвитку подібних систем.

2. Здійснено порівняльний огляд популярних сервісів бронювання авіаквитків (зокрема Skyscanner, Kiwi), визначено їх переваги, недоліки та функціональні особливості, що стало підґрунтям для формування вимог до власного застосунку.

3. Проведено аналіз сучасних технологій розробки вебзастосунків, таких як Spring Boot, PostgreSQL, Amadeus API, а також засобів для забезпечення безпеки та авторизації користувачів, що дозволило обґрунтовано обрати технологічний стек проєкту.

4. Сформовано повний перелік функціональних та нефункціональних вимог до вебзастосунку, орієнтуючись на зручність користувача, швидкодію системи, безпеку та масштабованість.

5. Спроєктовано архітектуру клієнт-серверного вебзастосунку з урахуванням логіки взаємодії з користувачами, адміністраторами та сторонніми API Amadeus для отримання актуальних даних про рейси.

6. Реалізовано вебзастосунок для бронювання авіаквитків, який включає основні функціональні модулі: реєстрація, авторизація, підтвердження акаунта, пошук та бронювання квитків, передача квитків між користувачами, адміністрування.

7. Проведено модульне та інтеграційне тестування вебзастосунку для перевірки його працездатності в різних сценаріях, що підтвердило відповідність реалізації сформульованим вимогам.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Код основних контролерів

```
@Slf4j @RestController
@RequestMapping("/api/tickets")
@RequiredArgsConstructor public class
TicketController {
    private final TicketService
ticketService;
    private final JwtService jwtService;


    @PostMapping("/book") public
ResponseEntity<BookingResponse> bookTicket(
        @RequestBody BookingRequest
request,


@RequestHeader("Authorization") String token) {
    public ResponseEntity<String> cancelBooking(
        @PathVariable Long bookingId,


@RequestHeader("Authorization") String token) {
        String email =
jwtService.extractUsername(token.substring(7)
);


ticketService.cancelBooking(bookingId, email);
        return
ResponseEntity.ok("Booking cancelled
successfully");
    }
        String email =
jwtService.extractUsername(token.substring(7)
);
        BookingResponse response =
ticketService.bookTicket(request, email);
        return ResponseEntity.ok(response);
    }



@PostMapping("/cancel/{bookingId}")
@PostMapping("/transfer/{bookingId}") public
    ResponseEntity<String>
transferTicket(
        @PathVariable Long bookingId,
        @RequestBody(required =
false) TransferRequest request,


@RequestHeader("Authorization") String token) {
        System.out.println(" Incoming transfer
request:");
        String recipientEmail = (request != null) ?
request.getEmail() : null;
        System.out.println(" Booking ID: "
+ bookingId);
        System.out.println(" Request email: " +
recipientEmail);

        if (recipientEmail == null ||
recipientEmail.isBlank()) {
            return
ResponseEntity.badRequest().body("Recipient email is
required");
        }


            String senderEmail =
jwtService.extractUsername(token.substring(7)
);
            log.warn("Transfer: bookingId={}, sender={},
recipient={}", bookingId, senderEmail, recipientEmail);


ticketService.transferTicket(bookingId, senderEmail,
recipientEmail);
            return ResponseEntity.ok("Ticket transferred
successfully");
        }



    @GetMapping("/history")
    public
ResponseEntity<List<TicketBookingDto>>
getBookingHistory(

@RequestHeader("Authorization") String
token) {
        String email =
jwtService.extractUsername(token.substring(7)
);
        List<TicketBookingDto> bookings
= ticketService.getUserBookings(email);
        return
ResponseEntity.ok(bookings);
    }
}


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

        private final UserService userService; private
        final AuthService
authService;
        private final JwtService jwtService;
        private final UserRepository
userRepository;


        @PostMapping("/register")
        public ResponseEntity<String>
register(@RequestBody RegisterRequest
request) {
```

```
userService.register(request.getEmail(),
request.getPassword());
                return ResponseEntity.ok("Confirmation code
sent to your email");
            }


            @PostMapping("/confirm")        public
            ResponseEntity<?>
confirmEmail(@RequestHeader(value                    =
"Authorization", required = false) String authHeader,
                    if
                        (succes
                        s) {
                        User
                        user =
userService.getUserByEmail(email);
                        String newToken
= jwtService.generateToken(user);
                        return
ResponseEntity.ok(Map.of(
                                "message", "Email
confirmed successfully",
                                "confirmed", true,
                                "token", newToken
@RequestBody  Map<String,  String>  request)  {
                String email = request.get("email");
                String code = request.get("code");


                if ((email == null || email.isBlank()) &&
authHeader != null && authHeader.startsWith("Bearer ")) {
                    email =
jwtService.extractUsername(authHeader.substri  ng(7));
                }


                if (email == null || code == null || code.length()
!= 6) {
                    return
ResponseEntity.badRequest().body(Map.of("m  essage",
"Email and 6-digit code are required"));
                }


                try {
                    boolean success =
userService.confirmEmail(email, code);
                    ));
                    } else {
                        return
ResponseEntity.badRequest().body(Map.of("m  essage",
"Invalid confirmation code"));
                    }
                } catch (Exception e) {
                    return
ResponseEntity.status(500).body(Map.of("mes  sage",
"Confirmation error"));
                }
            }



            @PostMapping("/resend-code") public
            ResponseEntity<String>
resendCode(@RequestHeader("Authorization")  String
authHeader) {
                try {
                    String
token =
authHeader.substring(7);
                    String email =
jwtService.extractUsername(token);
userService.resendConfirmationCode(email);
                    return ResponseEntity.ok("New
confirmation code sent");
                } catch (Exception e) { return
ResponseEntity.status(500).body("Failed to resend code");
                }
            }



            @PostMapping("/login") public
            ResponseEntity<?>
login(@RequestBody LoginRequest request) { try {
                AuthService.AuthResponse response =
authService.login(request.getEmail(), request.getPassword());
                return  ResponseEntity.ok()
                        .header("Authorization", "Bearer " +
response.getToken())
                        .body(response);
                } catch (Exception e) { return
ResponseEntity.status(401).body(Map.of("mes  sage", "Login
failed"));
                }
            }



            @GetMapping("/me") public
            ResponseEntity<?>
getCurrentUser(@RequestHeader("Authorizati  on") String
token) {
                try {
                    String email =
jwtService.extractUsername(token.substring(7)
);
                    return
ResponseEntity.ok(authService.getUserInfo(em  ail));
                } catch (Exception e) {
                    return
ResponseEntity.status(401).body("Invalid or expired
token");
                }
            }



            @Data
            public static class ConfirmRequest {
                private String code;
            }
        }
```

Код основних сервісів

```
        @Slf4j
        @Service
        @RequiredArgsConstructor
        public class UserService {
            private final UserRepository
userRepository;
            private final PasswordEncoder
passwordEncoder;
            private final EmailService
emailService;
            private final Random random = new Random();
            public void register(String email, String
password) {
                if (userRepository.findByEmail(email).isPresent(
)) {
                    throw new
RuntimeException("User with this email already
exists");
                }
```

```
                String confirmationCode =
    generateConfirmationCode();
                User user = new User(); user.setEmail(email);
                if (user.isEnabled()) {
                    return true; // Уже подтвержден
                }


                if (code == null ||
    !code.equals(user.getConfirmationCode())) {
    user.setPassword(passwordEncoder.encode(pas sword));

    user.setConfirmationCode(confirmationCode);
    + email);



                                                        }

return true;
                user.setEnabled(false); user.setRole("USER");


                                                    email) {
 public User getUserByEmail(String


    return
                userRepository.save(user);


    emailService.sendConfirmationEmail(email, confirmationCode);
                }


                public boolean confirmEmail(String email, String
    code) {
                    User user = userRepository.findByEmail(email)
                        .orElseThrow(() -> new
    RuntimeException("User not found"));
    userRepository.findByEmail(email)
                        .orElseThrow(() -> new
    RuntimeException("User not found"));
                }


                public void
    resendConfirmationCode(String email) {
                    User user =
    userRepository.findByEmail(email)
                        .orElseThrow(() -> new
    RuntimeException("User not found"));

                    if (user.isEnabled()) {
                        throw new RuntimeException("Email
    already confirmed");
                    }


                    String newCode =
    generateConfirmationCode();

                 user.setConfirmationCode(newCode);
                            userRepository.save(user);


    emailService.sendConfirmationEmail(email, newCode);
                }


                private String generateConfirmationCode() {
                    return String.format("%06d",
    random.nextInt(999999));
                }
            }
```

```
                    log.warn("Confirmation code
mismatch for user: " + email);
                    return false;
                }


                user.setEnabled(true);
                user.setConfirmationCode(null);
                userRepository.save(user); log.info("User
                email confirmed: "


        @Slf4j @Service
        @RequiredArgsConstructor public class
        TicketService {
            private final TicketBookingRepository
bookingRepository;
            private final UserRepository
userRepository;
            private final FlightOfferRepository
offerRepository;
            private final EmailService
emailService;
            public BookingResponse
bookTicket(BookingRequest request, String email)
{
                User user =
userRepository.findByEmail(email)
                    .orElseThrow(() -> new
ResponseStatusException(HttpStatus.UNAUT HORIZED,
"User not found"));

                if (!user.isEnabled()) {
                    throw new
ResponseStatusException(HttpStatus.FORBID DEN,
"Confirm your email first");
                }


                FlightOffer offer =
offerRepository.findById(request.getFlightOffe rId())
                    .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FO UND,
"Flight not found"));

                TicketBooking booking =
TicketBooking.builder()
                    .user(user)
                    .flightOffer(offer)


.bookingReference(UUID.randomUUID().toStr ing())
                    .active(true)
                    .transferred(false)


.bookingDate(LocalDateTime.now())


.expirationDate(offer.getDepartureDate().atStar tOfDay())
                    .build();


                TicketBooking savedBooking =
bookingRepository.save(booking);

emailService.sendBookingConfirmation(email,
savedBooking.getBookingReference());

                    return BookingResponse.builder()


.bookingReference(savedBooking.getBooking Reference())
```

```
.ticketId(savedBooking.getId())
                        .status("CONFIRMED")

.createdAt(savedBooking.getBookingDate())

.flightDetails(buildFlightDetails(offer))
                        .build();
        }
        public void cancelBooking(Long bookingId,
String email) {
            TicketBooking booking =
bookingRepository.findById(bookingId)
                        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FO UND, "Booking
not found"));

            if (!booking.getUser().getEmail().equals(email))
{
                throw new
ResponseStatusException(HttpStatus.FORBID DEN,
"Unauthorized");
            }

            booking.setActive(false);
            bookingRepository.save(booking);
        }

        public void transferTicket(Long bookingId,
String fromEmail, String toEmail) {
            log.info("🎟 START
transferTicket: from={}, to={}, bookingId={}",
fromEmail, toEmail, bookingId);

            User sender =
userRepository.findByEmail(fromEmail)
                        .orElseThrow(() -> {
                            log.error("❌ Sender not
found: {}", fromEmail);
                            return new
ResponseStatusException(HttpStatus.NOT_FO UND,
"Sender not found");
                        });

            User recipient =
userRepository.findByEmail(toEmail)
                        .orElseThrow(() -> { log.error("❌
                            Recipient not
found: {}", toEmail);
                            return new
ResponseStatusException(HttpStatus.NOT_FO UND,
"Recipient not found");
                        });

            TicketBooking original =
bookingRepository.findById(bookingId)
                        .orElseThrow(() -> { log.error("❌
                            Booking not
found: {}", bookingId);
                            return new
ResponseStatusException(HttpStatus.NOT_FO UND, "Booking
not found");
                        });

            log.info("➕ Creating new
booking for recipient...");
            TicketBooking newBooking =
TicketBooking.builder()
                        .user(recipient)

.flightOffer(original.getFlightOffer())

.bookingReference(UUID.randomUUID().toStr ing())

            log.info("☑ Booking found. active={},
transferred={}, expDate={}",
                        original.isActive(),
original.isTransferred(), original.getExpirationDate());

            if (!original.getUser().getId().equals(sender.getId(
))) {
                        .active(true)
                        .transferred(false)

.bookingDate(LocalDateTime.now())

.expirationDate(original.getExpirationDate())
                        .build();

bookingRepository.save(newBooking);
                log.warn("⚠ Sender does not match owner of
booking");
                throw new
ResponseStatusException(HttpStatus.FORBID DEN,
"Unauthorized");
            }

            log.info("🔁 Deactivating old
ticket...");
            original.setActive(false);
            original.setTransferred(true);
            bookingRepository.save(original);
            log.info("📧 Sending confirmation to:
{}", toEmail);

emailService.sendBookingConfirmation(toEma il,
newBooking.getBookingReference());

            log.info("☑ Transfer complete: new
bookingId={}, reference={}",
                        newBooking.getId(),
newBooking.getBookingReference());
        }
                                                }

        public List<TicketBookingDto>
getUserBookings(String email) {
            User user =
userRepository.findByEmail(email)
                        .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FO UND, "User
not found"));

            return
bookingRepository.findByUserId(user.getId()).
stream().map(booking -> {
                FlightOffer offer =
booking.getFlightOffer();
                return TicketBookingDto.builder()
                        .id(booking.getId())
                        .active(booking.isActive())
```

```
.transferred(booking.isTransferred())

.bookingDate(booking.getBookingDate())

.expirationDate(booking.getExpirationDate())

.origin(offer.getOriginLocationCode())

.destination(offer.getDestinationLocationCode(
))

.departureDate(offer.getDepartureDate().toStri  ng())
            private String
buildFlightDetails(FlightOffer offer) {
                return String.format("%s → %s |
%s",

offer.getOriginLocationCode(),

offer.getDestinationLocationCode(),
                        offer.getDepartureDate());
        }



    }
                    .build();
            }).toList();
```