

**Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення**

**КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»**

**РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ БРОНЮВАННЯ ТА КУПІВЛІ
ТУРИСТИЧНИХ КВИТКІВ**

**DEVELOPMENT OF A WEB PLATFORM FOR BOOKING AND
PURCHASING TOURIST TICKETS**

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-41
Михайловський Вадим Вікторович

Керівник:
Гордєєв Олександр Олександрович

Кваліфікаційну роботу
допущено до захисту
«__» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

«__» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Михайловському Вадиму Вікторовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка вебплатформи для бронювання та купівлі туристичних квитків»

Керівник роботи: професор Гордєєв О. О.

затверджені наказом закладу вищої освіти від «31» грудня 2025 р. № 541/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра «29» травня 2026 р.

3. Вихідні дані до роботи: предметна область бронювання та купівлі туристичних квитків онлайн, технології розробки веб-застосунків (JavaScript, React, Node.js, Express), засоби роботи з базами даних (SQLite, Prisma ORM), технології онлайн-платежів (Stripe API), вимоги до веб-інформаційних систем та забезпечення інформаційної безпеки користувачів.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): аналіз сучасного стану проблем автоматизації обліку персоналу та управління задачами, постановка завдання, формування вимог до розроблюваної інформаційної системи, обґрунтування вибору технологій та засобів розробки, практична реалізація об'єкта проєктування, розробка структури бази даних, обґрунтування принципів захисту інформаційно-комп'ютерної системи, тестування та налагодження програмного забезпечення.

5. Перелік графічного матеріалу: 14 рисунків, 7 лістингів, 1 таблиця.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	<i>Гордєєв О. О.</i>		
Специфікація вимог до розробленої системи	<i>Гордєєв О. О.</i>		
Розробка об'єкта проектування	<i>Гордєєв О. О.</i>		
Нормоконтроль	<i>Суринович О. М.</i>		
Гарант ОП	<i>Ліщина Н. М.</i>		
Показник запозичень тексту	<u> </u> %		
Академічна доброчесність	<i>Гордєєв О. О.</i>		

7. Дата видачі завдання «3» лютого 2026 р.

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 14.02.2026 р.	
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 03.03.2026 р.	
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 14.03.2026 р.	
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 14.04.2026 р.	
5	Практична реалізація об'єкта проектування та розробка бази даних	до 20.04.2026 р.	
6	Тестування та налагодження об'єкта проектування	до 04.05.2026 р.	
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі	до 29.05.2026 р.	

Здобувач вищої освіти _____

Вадим МИХАЙЛОВСЬКИЙ

Керівник кваліфікаційної роботи _____

Олександр ГОРДЄЄВ

АНОТАЦІЯ

Михайловський В. В. Розробка вебплатформи для бронювання та купівлі туристичних квитків. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі досліджено предметну область онлайн-бронювання туристичних квитків, розглянуто наявні сервіси та виявлено їхні переваги й недоліки, а також обґрунтовано потребу у розробці власного веб-рішення. У другому розділі визначено вимоги до веб-застосунку, виконано проектування системи із застосуванням UML-нотації та обґрунтовано вибір технологій і засобів розробки. У третьому розділі описано процес реалізації вебплатформи, розроблено структуру бази даних, розглянуто питання захисту користувацьких даних та автентифікації, інтегровано платіжний сервіс, а також наведено результати тестування програмного забезпечення. У висновках підсумовано отримані результати, підтверджено досягнення мети роботи та виконання поставлених завдань.

Ключові слова: веб-застосунок, бронювання квитків, туристичний сервіс, REST API, база даних, JavaScript, React, Node.js, Express, SQLite, Prisma ORM, Stripe API, JWT-автентифікація.

ABSTRACT

Mykhailovskyi V. V. Development of a web platform for booking and purchasing tourist tickets. Manuscript. Qualification work for bachelor's degree in Software Engineering, specialty "Software Engineering". Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification work consists of an introduction, three chapters, conclusions, a list of references.

The first chapter examines the subject area of online tourist ticket booking, reviews existing services and identifies their strengths and weaknesses, and justifies the need to develop a custom web solution. The second chapter defines the requirements for the web application, covers system design using UML notation, and justifies the choice of technologies and development tools. The third chapter describes the implementation of the website, the database structure, user data protection and authentication, payment service integration, and software testing results. The conclusions summarize the results obtained and confirm that the goal of the thesis and all assigned tasks were achieved.

Keywords: web application, ticket booking, tourist service, REST API, database, JavaScript, React, Node.js, Express, SQLite, Prisma ORM, Stripe API, JWT authentication.

ЗМІСТ

ВСТУП	7
РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ РОЗРОБКИ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ	9
1.1 Аналіз сучасного стану проблеми	9
1.2 Постановка завдання на кваліфікаційну роботу бакалавра	13
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ ВЕБПЛАТФОРМИ.....	15
2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення.....	15
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання	20
РОЗДІЛ 3 РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ БРОНЮВАННЯ ТА КУПІВЛІ ТУРИСТИЧНИХ КВИТКІВ.....	23
3.1 Практична реалізація об'єкта проєктування	23
3.2 Розробка бази даних.....	31
3.3 Захист інформаційно-комп'ютерної системи	34
3.4 Тестування та налагодження інформаційно-комп'ютерної системи.....	35
ВИСНОВКИ.....	40
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	42

ВСТУП

Актуальність теми. Онлайн-бронювання квитків давно стало звичною справою – люди просто не бачать сенсу йти до каси, коли все робиться з телефону за кілька хвилин. За цією зручністю стоїть складна система: пошук рейсів, захист платіжних даних, особистий кабінет, синхронізація бронювань. Ринок росте, і більшість мандрівників вже давно обирають онлайн-сервіси замість традиційних агентств [1].

Проблема в тому, що більшість таких платформ – закриті комерційні продукти, заточені під масштаб великих авіакомпаній чи залізниць. Для навчальних або регіональних задач вони або недоступні, або занадто громіздкі. А от сучасний веб-стек цілком дозволяє зібрати такий сервіс з нуля. Node.js з Express дає легкий серверний каркас з REST API, де маршрути, middleware і логіка бази даних нормально розділені [2].

Окремо стоїть питання безпеки. Дослідження MDPI 2024 року проаналізувало інциденти в онлайн-агентствах за 20 років і показало: витоки даних і шкідливе ПЗ – це не виняток, а постійна загроза для таких сервісів. Тобто автентифікація і захист даних – не опція, а обов’язкова частина будь-якого сервісу бронювання [3].

Тому і виникає потреба розробити власний веб-застосунок, який покриває весь цикл: пошук рейсів, бронювання, оплату і особистий кабінет.

Метою кваліфікаційної роботи бакалавра є розробка веб-застосунку для пошуку, бронювання та купівлі туристичних квитків із реалізацією автентифікації користувачів та інтеграцією онлайн-платежів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

- проаналізувати наявні сервіси бронювання квитків та визначити їхні переваги й недоліки;
- визначити вимоги до веб-застосунку;

- обґрунтувати вибір технологій розробки;
- спроектувати структуру системи та базу даних;
- реалізувати клієнтську та серверну частини застосунку
- інтегрувати платіжний сервіс та забезпечити захист користувацьких даних;
- провести тестування програмного забезпечення.

Об'єктом дослідження є процес онлайн-бронювання та купівлі туристичних квитків.

Предметом дослідження є методи та засоби розробки клієнт-серверного веб-застосунку з використанням технологій React, Node.js та інтеграцією платіжних систем.

Практичне значення роботи полягає у створенні веб-застосунку, що охоплює повний цикл розробки сучасного сервісу бронювання – від проєктування бази даних до інтеграції онлайн-платежів.

РОЗДІЛ 1

АНАЛІЗ МЕТОДІВ РОЗРОБКИ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Онлайн-продаж квитків існує вже понад двадцять років. Те, що починалось як експеримент кількох авіакомпаній, поступово стало стандартом – зараз більшість перевізників просто не мають касового продажу як основного каналу. Купити квиток на літак, автобус чи потяг можна за кілька хвилин не виходячи з дому, і ніхто вже не вважає це чимось особливим.

Але ринок нерівномірний. Великі агрегатори охоплюють десятки тисяч маршрутів і мільйони користувачів, тоді як менші перевізники або взагалі продають квитки тільки в касі, або підключені до агрегаторів на умовах, які їм не дуже вигідні. Це означає, що потреба у власних, незалежних платформах нікуди не зникла [4].

Щоб зрозуміти, що вже стало нормою в таких системах, розглянемо три сервіси з українського ринку: Tickets.ua, Busfor і Booking.com.

Tickets.ua почали у 2009 році як звичайну агенцію з продажу авіаквитків. Через два роки перевели все в онлайн і поступово розширили асортимент – зараз тут можна купити авіа, залізничні та автобусні квитки, забронювати готель, замовити трансфер або страховку. Щоденна відвідуваність близько 60 000 користувачів, більше 2 000 квитків. Серед партнерів – Amadeus, Booking.com, TripAdvisor [5].

Флоу простий і прямолінійний, це пошукова форма, список рейсів, деталі, дані пасажира, оплата, квиток на пошту. Без зайвих кроків. Саме це і зробило сервіс одним із основних гравців на ринку – люди не хочуть розбиратись у складних інтерфейсах, коли просто треба купити квиток. на добу. Зовнішній вигляд головної сторінки сервісу наведено на рисунку 1.1.

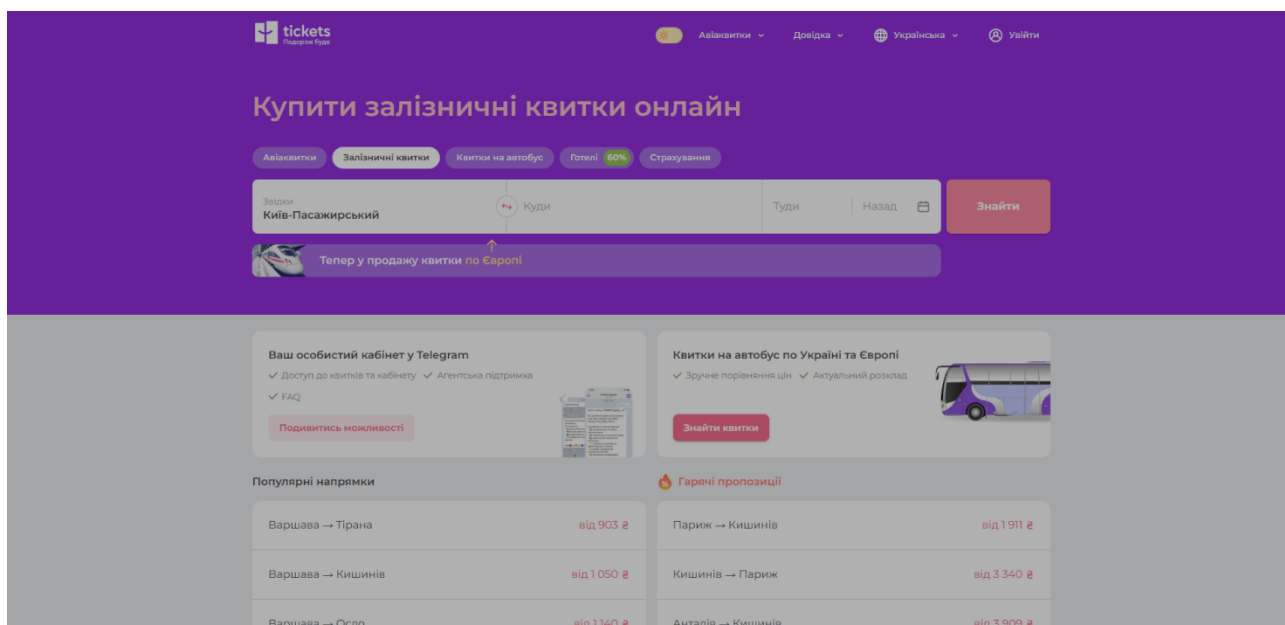


Рисунок 1.1 – Головна сторінка сервісу Tickets.ua [6]

Busfor вужчий за профілем, але детальніший у своїй ніші. Тільки автобуси – понад 20 000 маршрутів від більш ніж 200 перевізників. Тут можна не просто вибрати рейс, а й подивитись схему салону, обрати конкретне місце, відфільтрувати за типом автобуса або перевізником. Для когось це зайво, але хто регулярно їздить автобусом – знає, що різниця між місцем біля вікна і місцем над колесом може зіпсувати всю поїздку.

Додаток є для iOS і Android, оплата онлайн, квиток на пошту. За межі автобусних квитків сервіс не виходить – і це свідоме рішення. Вузька спеціалізація дає змогу зробити деталі, які широкі агрегатори зазвичай не чіпають [7].

Інтерфейс сервісу побудований так, щоб користувач витрачав мінімум часу на вибір – від введення маршруту до підтвердження місця все відбувається в кілька кроків. Саме тому Busfor залишається популярним серед тих, хто регулярно користується автобусним сполученням. Досвід Busfor показує, що навіть простий інтерфейс може був ефективним, якщо продумана логіка вибору рейсу та оформлення бронювання. Сторінку пошуку рейсів сервісу Busfor наведено на рисунку 1.2.

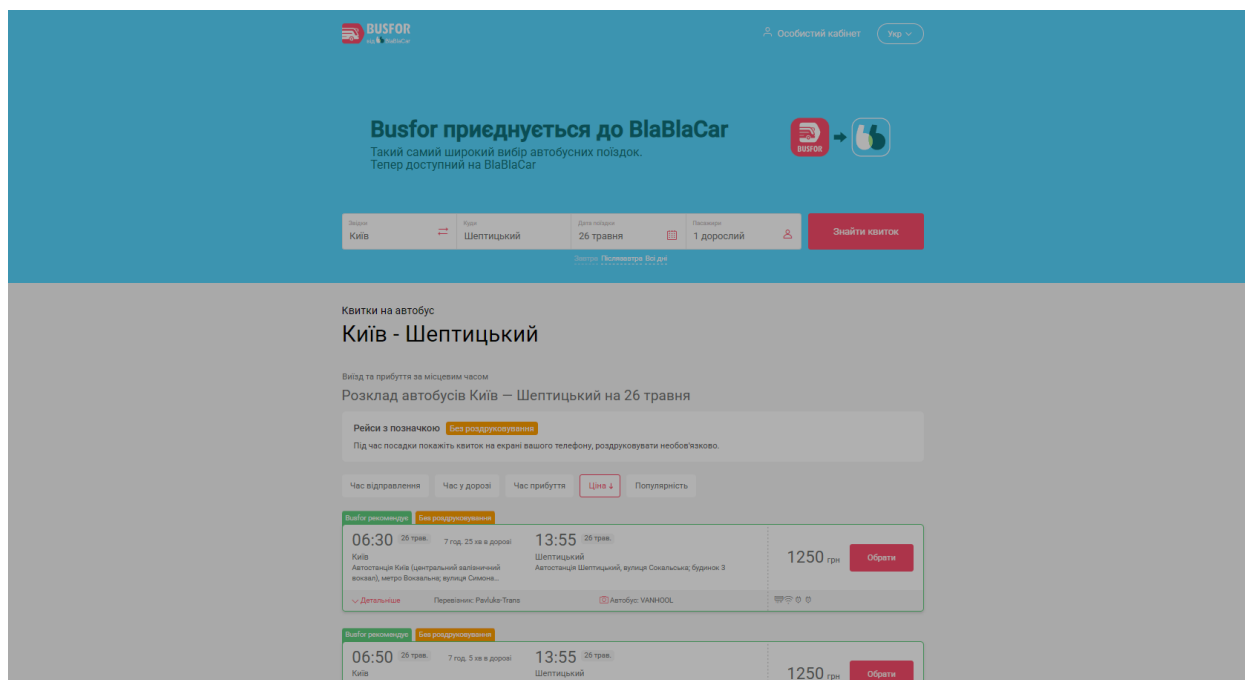


Рисунок 1.2 – Сторінка пошуку рейсів на сервісі Busfor [7]

Booking.com – це вже інша категорія і порівнювати його з двома попередніми не зовсім коректно. 3,4 млн об’єктів розміщення, понад 220 країн, десятки мов. Основний продукт – готелі та апартаменти, але за останні роки платформа додала авіаквитки, оренду авто, атракції. Масштаб такий, що це вже не агрегатор, а окрема інфраструктура.

Інтерфейс відточувався під мільйони різних сценаріїв, хтось шукає найдешевше, хтось читає кожен відгук, хтось бронює за п’ять хвилин не дивлячись. Система намагається закрити все це одночасно. Не як зразок для копіювання, але як орієнтир – корисно.

Платформа підтримує десятки мов інтерфейсу і працює однаково для користувача з України та з Японії. Такий рівень універсальності досягається роками технічної роботи, яку неможливо відтворити в рамках одного проєкту.

Водночас деякі рішення Booking.com можна адаптувати на рівні архітектури: розділення пошуку та бронювання на окремі кроки, збереження улюблених напрямків та персоналізовані алерти на ціну.

Результати пошуку на платформі Booking.com наведено на рисунку 1.3.

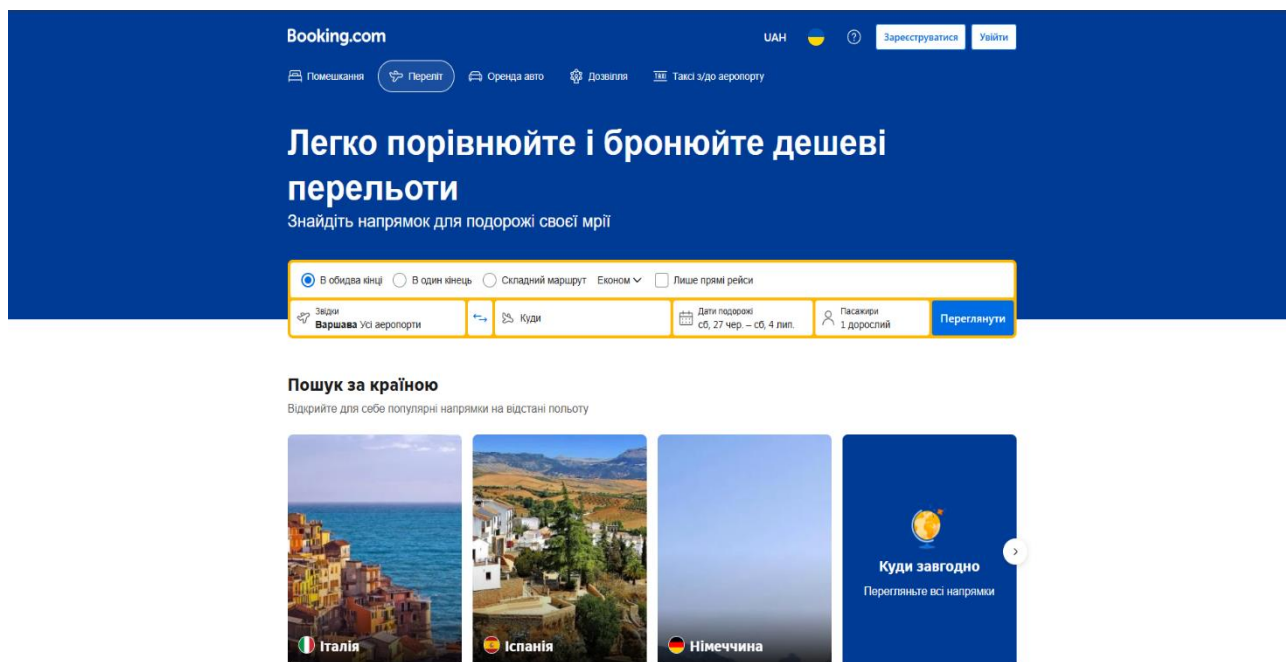


Рисунок 1.3 – Результати пошуку на платформі Booking.com [8]

З цих трьох сервісів видно, що є функції, без яких система бронювання не існує:

- форма пошуку з вибором напрямку, дати та кількості пасажирів;
- список результатів з фільтрацією та сортуванням;
- сторінка деталей рейсу з ціною та умовами;
- оформлення бронювання з даними пасажира;
- онлайн-оплата та підтвердження на email.

Порівняльний аналіз розглянутих сервісів за основними критеріями наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих сервісів бронювання

Критерій	Tickets.ua	Busfor	Booking.com
Тип квитків	Авіа, ЗД, автобус	Автобус	Готелі, авіа
Мова інтерфейсу	Українська	Українська	43 мови
Мобільний додаток	Так	Так	Так
Особистий кабінет	Так	Так	Так
Онлайн-оплата	Так	Так	Так
Відкритий вихідний код	Ні	Ні	Ні
Орієнтація	Європа	Європа	Весь світ

Базовий набір функцій є у всіх трьох. Але жоден із них не має відкритого коду – це закриті комерційні продукти. Розробити власне рішення в такому контексті означає не конкурувати з ними, а отримати реальний досвід побудови такої системи з нуля.

Якщо подивитись на ці сервіси разом, помітна одна спільна риса – всі вони розвивались роками, обростаючи функціями поступово. Жоден не з'явився одразу в тому вигляді, в якому існує зараз. Це важливо розуміти при проєктуванні власної системи: не намагатись відтворити все й одразу, а почати з ядра – пошук, бронювання, оплата – і будувати далі.

Ринок онлайн-бронювання загалом рухається у бік мобільних пристроїв і спрощення інтерфейсів. Користувачі дедалі рідше сидять за комп'ютером, коли планують поїздки – більшість перевіряють рейси з телефону, там само порівнюють ціни, і нерідко одразу платять. Системи, які не враховують цього, просто втрачають частину аудиторії.

Українські реалії підтверджують ту саму тенденцію. Онлайн-квитки давно перестали бути чимось новим – люди звикли купувати їх так само природно, як замовляти таксі або їжу. Але більшість наявних сервісів заточені під конкретний вид транспорту або великі маршрути. Туристичний сегмент – рейси між містами з повноцінним вибором місця, онлайн-оплатою і особистим кабінетом – залишається менш охопленим, ніж можна було б очікувати.

Саме в цьому просторі і розташовується розроблюваний веб-застосунок. Не як спроба замінити Tickets.ua чи Busfor, а як самостійна система з чітко визначеним функціональним ядром і реальною технічною реалізацією.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою кваліфікаційної роботи є розробка веб-застосунку для пошуку, бронювання та купівлі туристичних квитків із реалізацією автентифікації користувачів та інтеграцією онлайн-платежів

Для досягнення поставленої мети необхідно вирішити такі завдання:

- здійснити аналіз сучасного стану ринку онлайн-бронювання квитків, розглянути наявні сервіси та визначити їхні переваги й недоліки;
- визначити функціональні та нефункціональні вимоги до розроблюваного веб-застосунку;
- обґрунтувати вибір технологій, інструментальних засобів та архітектури програмного забезпечення;
- спроектувати структуру системи, включаючи логічну модель даних та взаємодію клієнтської і серверної частин;
- розробити базу даних для зберігання інформації про рейси, користувачів та бронювання;
- реалізувати клієнтську частину застосунку з використанням бібліотеки React та серверну частину на основі Node.js і Express з підключенням Prisma ORM;
- забезпечити реалізацію механізмів автентифікації користувачів на основі JWT та інтеграцію платіжного сервісу Stripe;
- провести тестування та налагодження розробленого веб-застосунку з метою перевірки його працездатності та відповідності поставленим вимогам.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ ВЕБПЛАТФОРМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

Перш ніж писати код, треба зрозуміти що саме система має робити. Для сервісу бронювання квитків це не абстрактне питання – тут є кілька типів користувачів, кілька сценаріїв використання і реальні гроші в процесі. Нечітко визначені вимоги на цьому етапі потім обертаються переробками, які коштують значно більше часу ніж сама розробка.

Вимоги прийнято ділити на два типи – це функціональні, тобто, що система робить, і нефункціональні, як вона це робить. Перші описують конкретні дії користувача, а другі швидкість, безпеку, зручність і надійність.

В системі є один тип користувача з двома станами – до авторизації і після. Незареєстрований відвідувач може переглядати сайт і шукати рейси, але бронювати не може. Авторизований користувач має доступ до повного циклу. Виходячи з цього, система має забезпечувати:

- реєстрацію та авторизацію з перевіркою введених даних;
- пошук рейсів за напрямком і датою відправлення;
- перегляд списку доступних рейсів з основною інформацією – маршрут, час, ціна;
- перегляд детальної інформації про конкретний рейс;
- оформлення бронювання з введенням даних пасажирів;
- онлайн-оплату через платіжний сервіс;
- перегляд історії бронювань в особистому кабінеті;
- вихід із системи.

Це мінімальні критерії. Без будь-якого з цих пунктів сервіс просто не працює як сервіс бронювання.

Окрім функціональних вимог, важливо визначити і нефункціональні. Паролі зберігаються у хешованому вигляді, автентифікація реалізована через

JWT-токени з обмеженим терміном дії. Номер картки та платіжна інформація не зберігаються на сервері – вся обробка відбувається на стороні Stripe. Застосунок коректно відображається на екранах різної ширини – від мобільного до десктопу. Усі поля введення перевіряються на стороні клієнта до відправки запиту на сервер. Користувач завжди розуміє на якому кроці він знаходиться і що відбудеться після наступної дії.

Для наочного представлення взаємодії користувача із системою побудовано діаграму варіантів використання. Вона охоплює два актори – незареєстрованого відвідувача і авторизованого користувача. Незареєстрований відвідувач може переглядати головну сторінку, шукати рейси і переходити до реєстрації або авторизації. Авторизований користувач має доступ до повного циклу: пошук, деталі рейсу, бронювання, оплата, особистий кабінет. Діаграму варіантів використання наведено на рисунку 2.1.

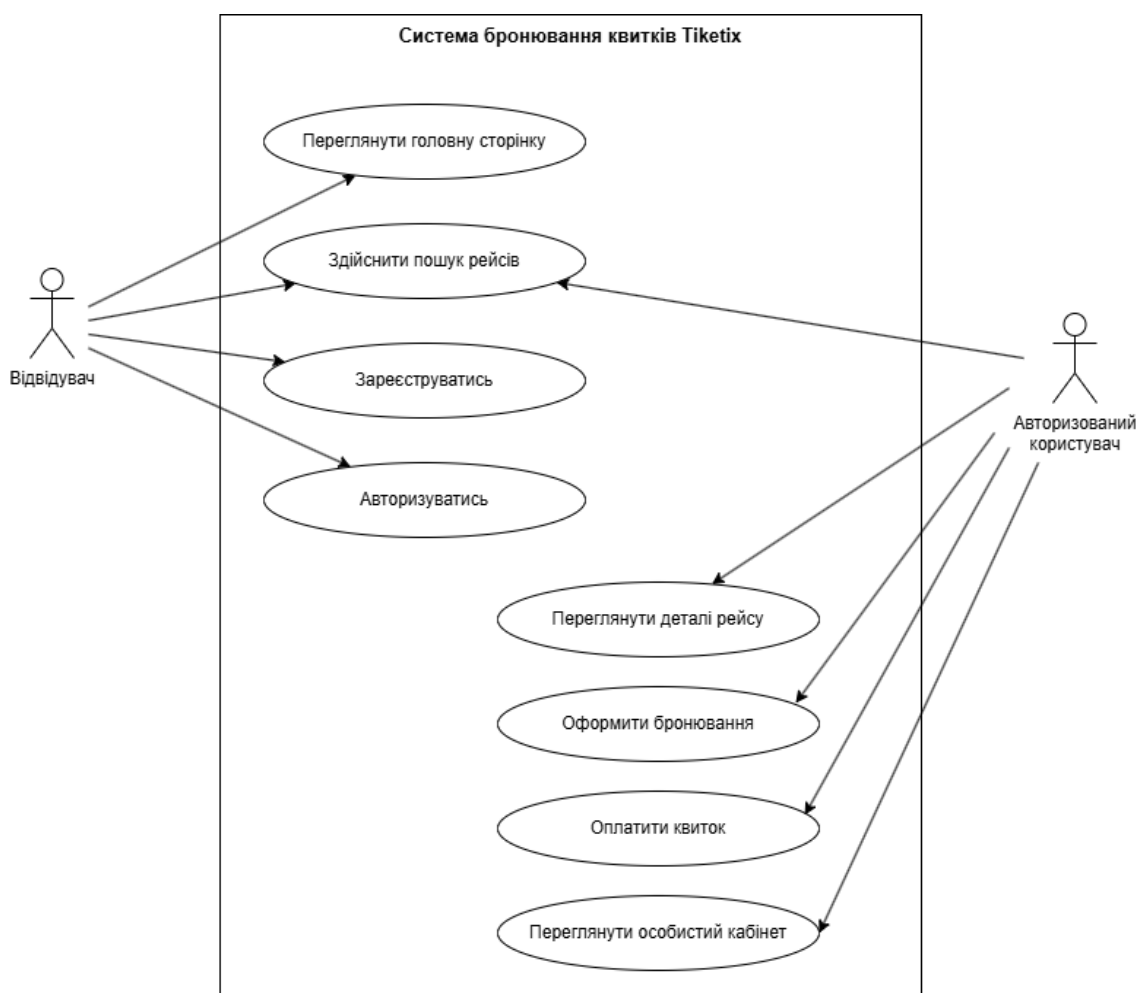


Рисунок 2.1 – Діаграма варіантів використання системи

Пошук рейсів доступний усім – і незареєстрованим відвідувачам, і авторизованим користувачам. Вводиш місто відправлення, місто прибуття і дату – отримуєш список рейсів. Якщо нічого не знайдено, система про це повідомляє. А от перейти до деталей рейсу або щось забронювати без входу в акаунт не вийде – система перенаправить на сторінку авторизації.

Реєстрація проста: ім'я, email, пароль. Поля перевіряються одразу на стороні клієнта – email на формат, пароль на довжину. Якщо такий email вже є в базі, сервер поверне помилку і користувач побачить повідомлення. Якщо все добре – система сама авторизує і зберігає токен, окремий вхід після реєстрації не потрібен.

Бронювання – це кілька послідовних кроків: вибір рейсу, дані пасажирів, оплата. Пропустити крок не можна – без вибраного рейсу форма пасажирів просто недоступна, без заповнених даних до оплати не дістанешся. Після того як платіж пройшов, бронювання з'являється в особистому кабінеті.

Наступним я хочу докладніше розглянути базу даних. База даних складається з трьох таблиць: User, Flight і Booking. User зберігає ідентифікатор, ім'я, email, хешований пароль і дату реєстрації. Flight містить усю інформацію про рейс: маршрут, дату, час відправлення і прибуття, тривалість, авіакомпанію, ціну і кількість місць. Booking пов'язує користувача і рейс, зберігає дані пасажирів, статус і дату створення.

Між таблицями два зв'язки типу «один до багатьох»: один користувач може мати багато бронювань, один рейс також може мати багато бронювань. Саме бронювання завжди належить рівно одному користувачу і одному рейсу. Окрім основних таблиць, база даних містить додаткові таблиці для зберігання пасажирів, способів оплати, налаштувань користувача та збережених напрямків. Ці таблиці пов'язані з користувачем через зовнішні ключі та використовуються для розширення функціональності особистого кабінету. Уся структура бази даних побудована на ORM Prisma, що дозволяє працювати з таблицями через JavaScript-об'єкти без написання SQL-запитів напряду. ER-діаграму бази даних наведено на рисунку 2.2.

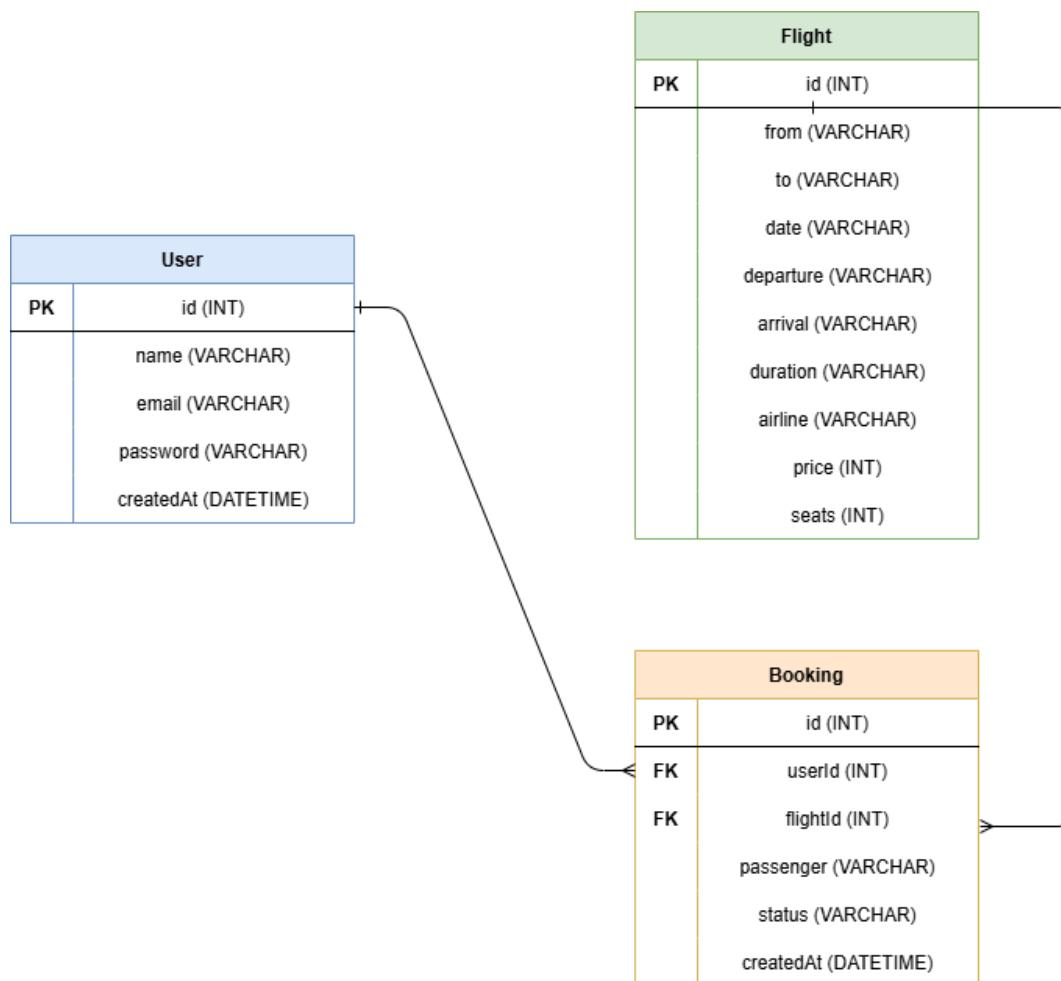


Рисунок 2.2 – ER-діаграма бази даних

Таблиця **User** зберігає базову інформацію про зареєстрованого користувача – ідентифікатор, ім'я, email і хешований пароль. Email має обмеження унікальності, тобто два акаунти з однаковою поштою створити не вийде. Дата реєстрації фіксується автоматично.

Таблиця **Flight** містить усе що потрібно знати про рейс: звідки і куди летить, коли відправляється і прибуває, скільки триває переліт, яка авіакомпанія і скільки коштує квиток. Поле **seats** показує кількість доступних місць – за замовчуванням 50 на кожен рейс.

Таблиця **Booking** – це зв'язок між користувачем і рейсом. Вона зберігає хто купив, який рейс, дані пасажира і статус бронювання. Статус за замовчуванням – «confirmed», одразу після успішної оплати. Зовнішні ключі **userId** і **flightId** забезпечують цілісність даних – бронювання без існуючого користувача або рейсу створити неможливо.

Застосунок побудований за клієнт-серверною архітектурою. Браузер завантажує React-застосунок, який спілкується з сервером через REST API. Сервер на Node.js і Express приймає запити, обробляє логіку, звертається до бази даних через Prisma і повертає відповідь. Окремо стоїть Stripe – зовнішній платіжний сервіс, з яким взаємодіє і клієнт, і сервер на різних етапах оплати.

Клієнтська частина – набір сторінок і компонентів. Кожна сторінка відповідає за окремий крок: головна з пошуком, результати, деталі рейсу, оформлення замовлення, особистий кабінет. Спільні елементи – шапка, підвал, форма пошуку – винесені в окремі компоненти і використовуються на кількох сторінках одночасно.

Серверна частина організована за маршрутами: окремі роутери обробляють запити до авторизації, рейсів, бронювань і платежів. Запити до захищених ендпоінтів проходять через middleware автентифікації, який перевіряє JWT-токен перед тим як пустити запит далі. Діаграму компонентів системи наведено на рисунку 2.3.

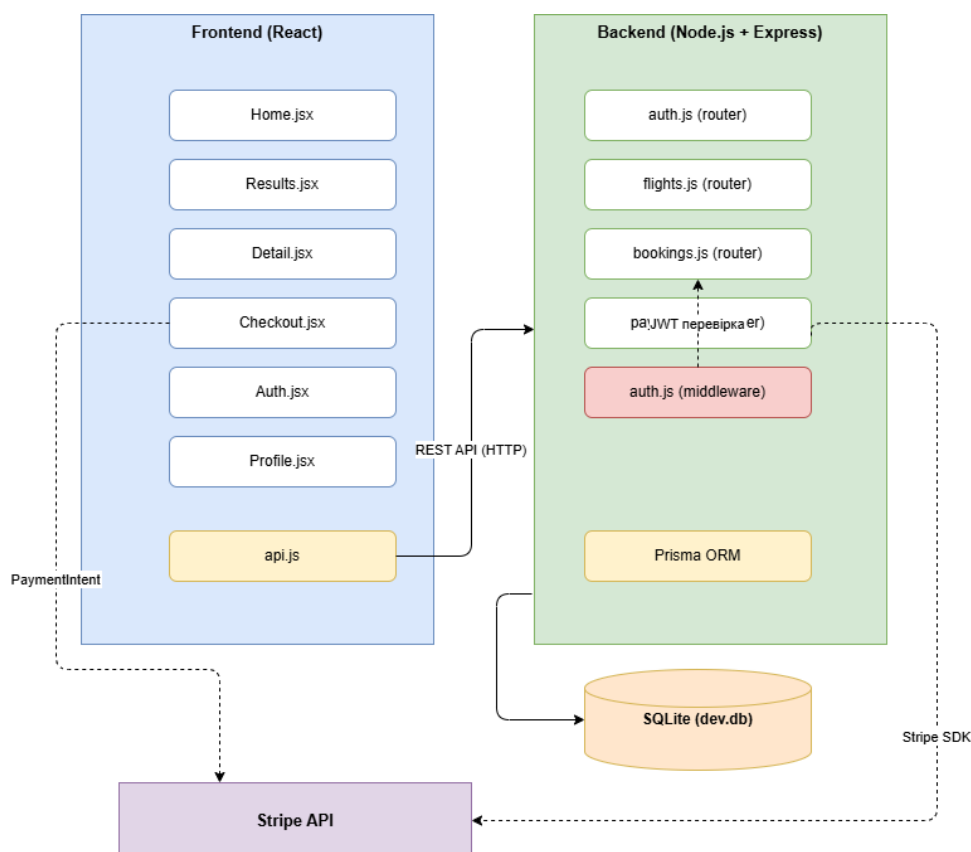


Рисунок 2.3 – Діаграма компонентів системи

Така архітектура чітко розділяє відповідальність між частинами системи – клієнт відповідає за відображення, сервер за логіку і дані, Stripe за платежі, і кожна з них може змінюватись незалежно від інших.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Вибір технологій для веб-застосунку – це не просто список інструментів. Кожне рішення має причину: або воно краще підходить для конкретної задачі, або добре інтегрується з рештою стеку, або є галузевим стандартом для таких систем.

React – бібліотека для побудови інтерфейсів, розроблена Meta. Основна ідея – компонентний підхід: інтерфейс складається з незалежних блоків, кожен з яких відповідає за свою частину сторінки. Для сервісу бронювання це природно – форма пошуку, список рейсів, картка рейсу, форма пасажира – все це окремі компоненти, які розробляються незалежно один від одного. Якщо треба змінити вигляд картки рейсу – чіпаєш тільки один файл, решта не зачіпається. React має величезну екосистему і залишається одним із найпопулярніших інструментів фронтенд-розробки [9].

Для маршрутизації між сторінками використано React Router DOM. Це стандартне рішення для односторінкових React-застосунків – перехід між сторінками відбувається без перезавантаження браузера, що дає відчуття швидкого і плавного інтерфейсу. Користувач не бачить білого екрану між переходами, все відбувається миттєво.

Для збірки проєкту обрано Vite 6. Порівняно з традиційним Webpack він запускає локальний сервер майже миттєво і оновлює сторінку при змінах без повної перезбірки. На практиці це означає, що між збереженням файлу і результатом у браузері проходить менше секунди. Для активної розробки це суттєва різниця [10].

На серверній стороні використано Node.js – середовище виконання JavaScript поза браузером. Головна перевага в контексті цього проєкту – єдина мова на клієнті і сервері. Не треба перемикатись між різними синтаксисами і підходами, весь код написаний однією мовою. Node.js добре підходить для REST API з багатьма паралельними запитами завдяки асинхронній моделі виконання – сервер не блокується на очікуванні відповіді від бази даних, а обробляє інші запити паралельно [11].

Express – мінімалістичний фреймворк для Node.js. Він не нав'язує жорсткої структури проєкту, але дає зручні інструменти для визначення маршрутів, роботи з middleware і обробки запитів. Альтернативи – NestJS або Fastify – або надмірно складні для проєкту такого масштабу, або потребують більшого часу на налаштування. Express дозволяє почати працювати за лічені хвилини.

Prisma це ORM для роботи з базою даних. Замість написання SQL-запитів вручну розробник описує схему даних у спеціальному файлі, і Prisma генерує клієнт для роботи з базою. Це зменшує кількість помилок, спрощує зміни структури таблиць через міграції і робить код читабельнішим. Якщо треба додати нове поле до таблиці – змінюєш схему, запускаєш міграцію, і клієнт оновлюється автоматично [12].

SQLite обрано як базу даних. Це файлова база – вся інформація зберігається в одному файлі, не потребує окремого сервера і жодного налаштування. Для навчального проєкту це ідеально: просто розгортається, легко переноситься між машинами і не вимагає адміністрування. Важливо, що Prisma абстрагує різницю між базами даних – при потребі SQLite можна замінити на PostgreSQL без змін у коді застосунку, просто змінивши рядок підключення.

Для автентифікації використано JSON Web Token. Після входу сервер генерує токен з ідентифікатором користувача і терміном дії – 7 днів. Клієнт зберігає його в localStorage і передає з кожним запитом до захищених ендпоінтів у заголовку Authorization. Сервер перевіряє підпис і, якщо токен дійсний, виконує запит. Паролі перед збереженням хешуються через bcryptjs – навіть якщо база даних потрапить до чужих рук, оригінальні паролі залишаться

невідомими. Такий підхід є stateless: сервер не зберігає сесії і не звертається до бази для кожної перевірки автентифікації [13].

Stripe обрано для обробки платежів. Це один із найпоширеніших платіжних сервісів – він надає SDK для клієнта, API для сервера і безпечно обробляє дані картки на своїй стороні. Схема така, що клієнт відправляє дані картки напряму до Stripe, отримує токен підтвердження і передає його на сервер. Сервер використовує цей токен для завершення транзакції через Stripe API. Номер картки жодного разу не проходить через власний сервер застосунку – це знімає з розробника відповідальність за зберігання платіжних даних і відповідність стандартам безпеки PCI DSS [14].

РОЗДІЛ 3

РОЗРОБКА ВЕБПЛАТФОРМИ ДЛЯ БРОНЮВАННЯ ТА КУПІВЛІ ТУРИСТИЧНИХ КВИТКІВ

3.1 Практична реалізація об'єкта проєктування

Застосунок складається з двох незалежних частин – клієнтської і серверної. Вони запускаються окремо і спілкуються через HTTP: фронтенд на порту 5173, бекенд на порту 3001. Такий поділ дозволяє розробляти і тестувати кожну частину незалежно одна від одної.

Точка входу клієнтської частини – файл `App.jsx`, який визначає маршрутизацію. Кожен URL відповідає окремій сторінці: головна, результати пошуку, деталі рейсу, оформлення бронювання, авторизація, особистий кабінет. Дані між сторінками передаються через стан навігації – без зайвих запитів до сервера. Наприклад, після вибору рейсу на сторінці результатів об'єкт рейсу передається напряму на сторінку деталей, не завантажуючись повторно з API.

Усі звернення до API зосереджені в одному файлі – `api.js`. Він містить функції для роботи з токеном у `localStorage`, базову функцію `apiFetch` і об'єкт `api` з методами для кожного ендпоінту. Якщо змінюється адреса сервера або спосіб авторизації – правиться один файл, а не десятки місць по всьому проєкту. Реалізацію модуля наведено у лістингу 3.1.

Лістинг 3.1 – Модуль `api.js`

```
const BASE = 'http://localhost:3001'

export function getAuth() {
  const raw = localStorage.getItem('ticketix_auth')
  return raw ? JSON.parse(raw) : null
}

function authHeaders() {
  const auth = getAuth()
  return auth ? { Authorization: `Bearer ${auth.token}` } : {}
}

export function setAuth(data) {
```

```

    localStorage.setItem('tiketix_auth', JSON.stringify(data))
  }

  export function clearAuth() {
    localStorage.removeItem('tiketix_auth')
  }

  export async function apiFetch(path, options = {}) {
    const res = await fetch(`${BASE}${path}`, {
      headers: { 'Content-Type': 'application/json', ...authHeaders(),
    ...options.headers },
      ...options,
    })
    const data = await res.json()
    if (!res.ok) throw new Error(data.error || 'Помилка сервера')
    return data
  }

  export const api = {
    login: (email, password) => apiFetch('/api/auth/login', { method:
    'POST', body: JSON.stringify({ email, password }) }),
    register: (email, password, name) => apiFetch('/api/auth/register', {
    method: 'POST', body: JSON.stringify({ email, password, name }) }),
    flights: (from, to, date) =>
    apiFetch(`/api/flights?from=${encodeURIComponent(from)}&to=${encodeURIComponent(to)}${date ? `&date=${date}` : ''}`),
    myBookings: () => apiFetch('/api/bookings/my'),
    createBooking: (flightId, passenger) => apiFetch('/api/bookings', {
    method: 'POST', body: JSON.stringify({ flightId, passenger }) }),
    createPaymentIntent: (amount) => apiFetch('/api/payments/create-intent', { method: 'POST', body: JSON.stringify({ amount }) }),
  }

```

кінець лістингу 3.1

apiFetch – це центральна функція для всіх HTTP-запитів у застосунку. Вона сама додає Content-Type і токен авторизації до кожного запиту, і сама обробляє помилки – якщо сервер повернув збій, летить виключення з текстом помилки. Об'єкт api поверх неї – просто зручна обгортка, замість того щоб щоразу формувати URL і тіло запиту вручну, компонент викликає api.login(email, password) або api.createBooking(flightId, passenger).

Серверна частина побудована на Express. Головний файл index.js підключає middleware, реєструє роутери і запускає сервер. Кожен роутер відповідає за свій домен це авторизація, рейси, бронювання, платежі. Якщо

потрібна логіка бронювань – одразу йдеш у routes/bookings.js, не шукаючи по всьому проєкту [15].

Наступним розглянемо роутер авторизації, що реалізує два ендпоінти реєстрацію і вхід. Обидва використовують спільну функцію authResponse, яка генерує JWT-токен і повертає його разом з даними користувача. Це дозволяє уникнути дублювання коду – логіка формування відповіді написана один раз. Реалізацію наведено у лістингу 3.2.

Лістинг 3.2 – Роутер авторизації routes/auth.js

```
const router = require('express').Router();
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const prisma = require('../lib/prisma');

function authResponse(user) {
  const token = jwt.sign({ userId: user.id }, process.env.JWT_SECRET, {
    expiresIn: '7d' });
  return { token, user: { id: user.id, email: user.email, name: user.name
  } }};

  router.post('/register', async (req, res) => {
    const { email, password, name } = req.body;
    if (!email || !password || !name) return res.status(400).json({ error:
    'Всі поля обов'язкові' });
    const existing = await prisma.user.findUnique({ where: { email } });
    if (existing) return res.status(400).json({ error: 'Email вже
    використовується' });
    const hashed = await bcrypt.hash(password, 10);
    const user = await prisma.user.create({ data: { email, password:
    hashed, name } });
    res.json(authResponse(user));
  });

  router.post('/login', async (req, res) => {
    const { email, password } = req.body;
    const user = await prisma.user.findUnique({ where: { email } });
    if (!user) return res.status(401).json({ error: 'Невірний email або
    пароль' });
    const ok = await bcrypt.compare(password, user.password);
    if (!ok) return res.status(401).json({ error: 'Невірний email або
    пароль' });
    res.json(authResponse(user));
  });
}
```

кінець лістингу 3.2

При реєстрації сервер спочатку перевіряє чи всі поля заповнені, потім шукає в базі користувача з таким email. Знайшов – повертає помилку. Не знайшов – хешує пароль і створює запис. При вході простіше: знайти за email, порівняти пароль з хешем через `bcrypt.compare`. В обох випадках успішна відповідь формується через одну функцію `authResponse` – токен з терміном дії 7 днів і базові дані користувача.

Далі розглянемо роутер бронювань, який містить два захищені ендпоінти – список бронювань користувача і створення нового. Обидва проходять через `middleware` автентифікації перед виконанням. Без дійсного токена запит навіть не доходить до логіки – повертається 401 одразу на рівні `middleware`. Реалізацію наведено у лістингу 3.3.

Лістинг 3.3 – Роутер бронювань `routes/bookings.js`

```
const router = require('express').Router();
const auth = require('../middleware/auth');
const prisma = require('../lib/prisma');

router.get('/my', auth, async (req, res) => {
  const bookings = await prisma.booking.findMany({
    where: { userId: req.user.userId },
    include: { flight: true },
    orderBy: { createdAt: 'desc' },
  });
  res.json(bookings);
});

router.post('/', auth, async (req, res) => {
  const { flightId, passenger } = req.body;
  if (!flightId || !passenger) return res.status(400).json({ error:
'flightId та passenger обов'язкові' });

  const booking = await prisma.booking.create({
    data: { userId: req.user.userId, flightId: Number(flightId),
passenger },
    include: { flight: true },
  });
  res.json(booking);
});
```

кінець лістингу 3.3

Є одна деталь яку варто відзначити: при отриманні бронювань роутер бере `userId` з розкодованого токена, а не з тіла запиту. Це не випадково – тіло запиту користувач може підмінити, токен – ні. Тобто навіть якщо хтось надішле чужий `userId` у запиті, система поверне бронювання того, чий токен використовується. При створенні бронювання `flightId` явно конвертується в число через `Number()` – без цього Prisma може отримати рядок замість цілого і впасти з помилкою типізації.

Ще одним важливим модулем є роутер платежів, який реалізує один ендпоінт – створення `PaymentIntent` через Stripe API. Сервер отримує суму від клієнта, перевіряє її і створює намір оплати. У відповідь клієнт отримує `clientSecret` – ключ для завершення транзакції безпосередньо на стороні Stripe. Реалізацію наведено у лістингу 3.4.

Лістинг 3.4 – Роутер платежів `routes/payments.js`

```
const router = require('express').Router();
const Stripe = require('stripe');
const stripe = Stripe(process.env.STRIPE_SECRET_KEY);

router.post('/create-intent', async (req, res) => {
  const { amount } = req.body;
  if (!amount || amount < 50) return res.status(400).json({ error: 'Сума некоректна' });

  const intent = await stripe.paymentIntents.create({
    amount,
    currency: 'uah',
    automatic_payment_methods: { enabled: true },
  });

  res.json({ clientSecret: intent.client_secret });
});
```

кінець лістингу 3.4

Сума передається у копійках – це стандарт Stripe для більшості валют, включно з гривнею. Перевірка `amount < 50` відсіює некоректні запити ще до того як вони дійдуть до Stripe. `automatic_payment_methods: { enabled: true }` означає

що Stripe сам визначає доступні методи оплати залежно від країни і типу картки – не треба прописувати їх вручну.

Middleware автентифікації – невеликий але критичний файл. Він перехоплює кожен запит до захищених маршрутів, витягує токен з заголовка Authorization, перевіряє підпис і або пускає запит далі, або повертає помилку. Якщо заголовок відсутній взагалі – відповідь 401 без зайвих перевірок. Реалізацію наведено у лістингу 3.5.

Лістинг 3.5 – Middleware автентифікації middleware/auth.js

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  const header = req.headers.authorization;
  if (!header) return res.status(401).json({ error: 'Не авторизовано' });

  const token = header.replace('Bearer ', '');
  try {
    req.user = jwt.verify(token, process.env.JWT_SECRET);
    next();
  } catch {
    res.status(401).json({ error: 'Невалідний токен' });
  }
};
```

кінець лістингу 3.5

Головна сторінка – перше що бачить користувач. У центрі – форма пошуку з трьома полями: місто відправлення, місто прибуття і дата. Під формою розміщено секцію з популярними напрямками – картки міст, які ведуть на сторінку конкретного напрямку. Шапка містить логотип і навігацію: якщо користувач авторизований – відображається його ім'я і посилання на особистий кабінет, якщо ні – кнопка входу. Нижче розміщено табло вильотів з актуальною інформацією про рейси та секцію переваг сервісу. Також на сторінці присутній блок підписки на алерти ціни, де користувач може залишити email для отримання сповіщень про здешевлення квитків. Зовнішній вигляд головної сторінки наведено на рисунку 3.1.

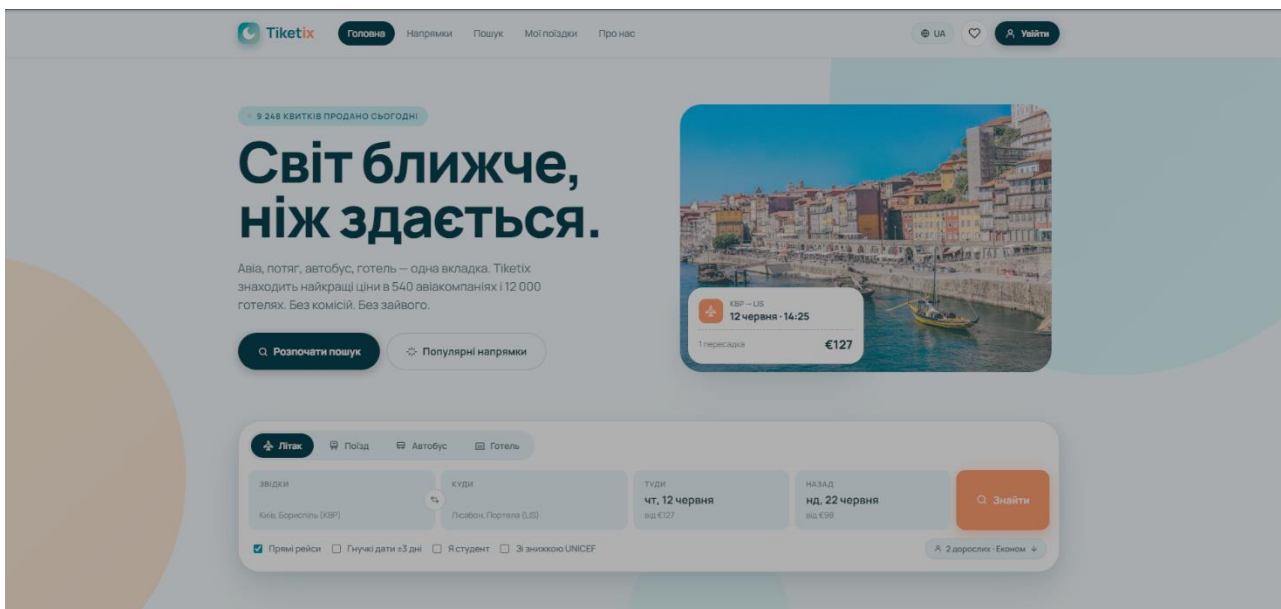


Рисунок 3.1 – Головна сторінка застосунку Tiktix

Сторінка результатів відображає список знайдених рейсів – маршрут, час відправлення і прибуття, тривалість перельоту, авіакомпанія і ціна. Кожна картка рейсу має кнопку переходу до деталей. Якщо за вказаним маршрутом рейсів немає – відображається відповідне повідомлення. Зовнішній вигляд сторінки результатів наведено на рисунку 3.2.

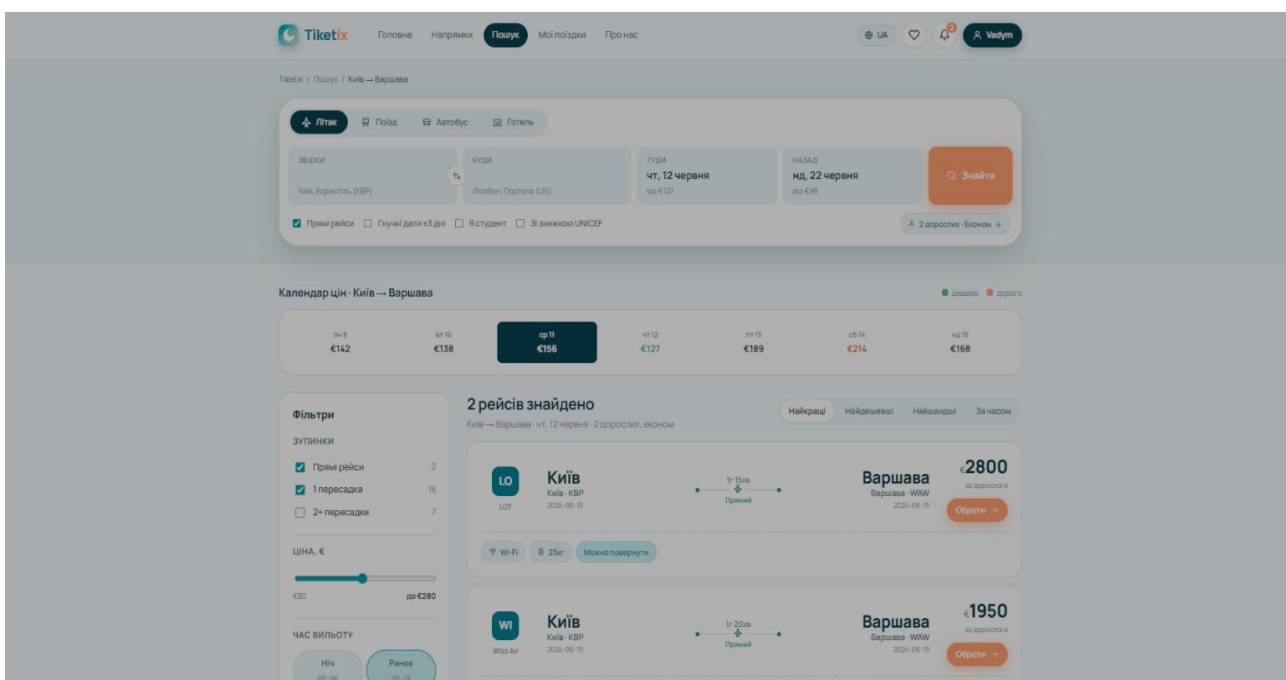


Рисунок 3.2 – Сторінка результатів пошуку

Сторінка оформлення бронювання реалізована як покроковий процес з трьох етапів. На першому користувач вводить дані пасажирів – ім'я і прізвище. На другому – вводить платіжні дані через форму Stripe CardElement, номер картки ніколи не потрапляє на власний сервер. На третьому – отримує підтвердження бронювання з деталями рейсу. Зовнішній вигляд наведено на рисунку 3.3.

The screenshot displays the Ticketix booking interface. At the top, there's a navigation bar with links like 'Головна', 'Напрямки', 'Політи', 'Мої польоти', and 'Про нас'. Below this, a progress indicator shows three steps: '1. Пасажирів', '2. Оплата', and '3. Готово'. The main form is divided into sections for 'Пасажир 1: Дорослий' and 'Пасажир 2: Дорослий', each with fields for 'Ім'я (вк у паспорті)', 'Прізвище', 'Дата народження', and 'Номер паспорта'. A 'Контактна інформація' section includes fields for 'Email' and 'Телефон'. On the right, a 'Твоє бронювання' summary box shows flight details: '08:15' to '12:05', 'Тариф: 1 пасажир', and 'Разом: \$248'. At the bottom, there are buttons for 'Назад до рейсу' and 'Перейти до оплати'.

Рисунок 3.3 – Сторінка оформлення бронювання

Після успішної оплати бронювання одразу з'являється в базі і користувач бачить щось на зразок посадкового талона – маршрут, дата, ім'я пасажирів, номер бронювання. Не треба нікуди переходити чи чекати листа, все відображається прямо на сторінці. Звідти можна одразу перейти до особистого кабінету. Якщо оплата не пройшла – форма не зникає, просто з'являється повідомлення про помилку від Stripe і можна спробувати ще раз.

Особистий кабінет показує всі бронювання користувача у вигляді карток. Кожна картка містить маршрут, дату, авіакомпанію, ім'я пасажирів і статус бронювання. Внизу сторінки – кнопка виходу з акаунту. Зовнішній вигляд наведено на рисунку 3.4.

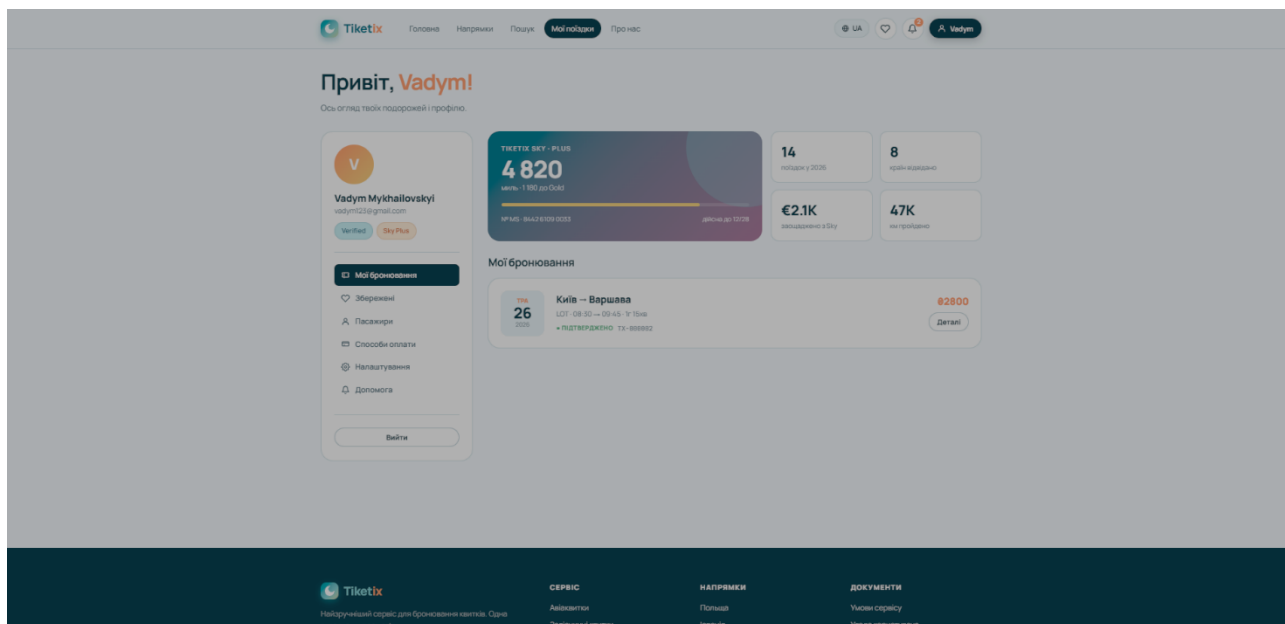


Рисунок 3.4 – Особистий кабінет користувача

3.2 Розробка бази даних

База даних застосунку побудована на SQLite і керується через Prisma ORM. Уся структура описана в одному файлі – `schema.prisma`. Це єдине місце де визначаються таблиці, поля і зв'язки між ними. Треба змінити структуру – міняєш схему, запускаєш міграцію, Prisma сама генерує SQL і застосовує зміни. Руками писати ALTER TABLE не треба.

Схема містить три моделі: User, Flight і Booking. Їхню реалізацію наведено у лістингу 3.6.

Лістинг 3.6 – Схема бази даних

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}

model User {
  id      Int      @id @default(autoincrement())
  email   String   @unique
```

```

    password String
    name      String
    createdAt DateTime @default(now())
    bookings Booking[]
}

model Flight {
  id          Int          @id @default(autoincrement())
  from        String
  to          String
  date        String
  departure   String
  arrival     String
  duration    String
  airline     String
  price       Int
  seats       Int          @default(50)
  bookings    Booking[]
}

model Booking {
  id          Int          @id @default(autoincrement())
  userId      Int
  flightId    Int
  passenger   String
  status      String      @default("confirmed")
  createdAt   DateTime    @default(now())
  user        User        @relation(fields: [userId], references: [id])
  flight      Flight      @relation(fields: [flightId], references: [id])
}

```

кінець лістингу 3.6

User зберігає email з обмеженням унікальності – два акаунти з однаковою поштою не створити, це перевіряється на рівні бази. Пароль зберігається вже хешованим, відкритого тексту в базі немає взагалі. createdAt заповнюється автоматично.

Flight містить усе що потрібно знати про рейс: маршрут, дату, час відправлення і прибуття, тривалість, авіакомпанію, ціну і кількість місць. Ціна зберігається у копійках як ціле число – стандартний підхід при роботі з грошима, який уникає проблем з округленням. Кількість місць за замовчуванням – 50.

Booking пов'язує користувача і рейс через зовнішні ключі `userId` і `flightId`. Поле `passenger` зберігає ім'я пасажирів, те що ввів користувач під час оформлення. Статус одразу після створення – `confirmed`.

Для початкового наповнення бази написано `seed.js`. Він містить 10 тестових рейсів між українськими та європейськими містами: Київ, Варшава, Берлін, Відень, Прага, Барселона, Рим, Амстердам. Різні напрямки, різні авіакомпанії, ціни від 1950 до 5800 гривень – достатньо щоб одразу перевірити пошук і фільтрацію без ручного введення даних. Реалізацію `seed`-файлу наведено у лістингу 3.7.

Лістинг 3.7 – Файл початкового наповнення бази даних

```
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

const flights = [
  { from: 'Київ', to: 'Варшава', date: '2024-06-15', departure: '08:30',
    arrival: '09:45', duration: '1г 15хв', airline: 'LOT', price: 2800 },
  { from: 'Київ', до: 'Варшава', date: '2024-06-15', departure: '14:00',
    arrival: '15:20', duration: '1г 20хв', airline: 'Wizz Air', price: 1950 },
  { from: 'Київ', to: 'Берлін', date: '2024-06-15', departure: '10:15',
    arrival: '12:30', duration: '2г 15хв', airline: 'Lufthansa', price: 4200 },
  { from: 'Київ', to: 'Відень', date: '2024-06-15', departure: '07:00',
    arrival: '08:45', duration: '1г 45хв', airline: 'Austrian', price: 3600 },
];

async function main() {
  await prisma.flight.deleteMany();
  await prisma.flight.createMany({ data: flights });
  console.log(`Створено ${flights.length} рейсів`);
}

main().catch(console.error).finally(() => prisma.$disconnect());
```

кінець лістингу 3.7

Перед додаванням `seed` спочатку видаляє всі наявні рейси через `deleteMany()`. Це гарантує що після повторного запуску дублікатів не буде. Після завершення клієнт явно відключається через `$disconnect()`.

3.3 Захист інформаційно-комп'ютерної системи

Безпека у веб-застосунку – це не одна функція, а кілька рівнів захисту, які працюють разом. У Tiketix реалізовано чотири основних механізми: хешування паролів, JWT-автентифікація, захист платіжних даних через Stripe і валідація вхідних даних.

Паролі користувачів ніколи не зберігаються у відкритому вигляді. Перед записом у базу кожен пароль хешується через бібліотеку bcryptjs з коефіцієнтом складності 10. Це означає що навіть якщо хтось отримає доступ до файлу бази даних – побачить лише набір символів без можливості відновити оригінальний пароль. При вході система порівнює введений пароль з хешем через bcrypt.compare – оригінал при цьому ніде не фігурує.

Після успішної авторизації сервер генерує JWT-токен. Він містить userId і термін дії – 7 днів. Токен підписується секретним ключем з .env файлу, тобто підробити його без знання ключа неможливо. Клієнт зберігає токен у localStorage і передає його з кожним запитом до захищених ендпоінтів у заголовок Authorization: Bearer <token>. Middleware перехоплює цей заголовок, перевіряє підпис і або пускає запит далі, або повертає 401.

Термін дії токена – важливий момент. Через 7 днів токен стає недійсним і користувач автоматично виходить із системи. Це обмежує вікно можливого зловживання якщо токен потрапить до чужих рук.

Платіжні дані захищені архітектурно. Номер картки, CVV і термін дії ніколи не проходять через власний сервер застосунку – це фізично неможливо за схемою інтеграції Stripe. Клієнт відправляє дані картки напряму до серверів Stripe через їхній SDK, отримує clientSecret і вже з ним завершує транзакцію. Власний сервер бачить тільки суму і результат – підтверджено чи відхилено. Така схема знімає з розробника відповідальність за відповідність стандартам безпеки платіжних даних PCI DSS.

Валідація вхідних даних реалізована на двох рівнях. На стороні клієнта – форми перевіряють поля до відправки запиту: email на коректний формат, пароль

на мінімальну довжину, обов'язкові поля на заповненість. Це дає миттєвий зворотний зв'язок без зайвих запитів до сервера. На стороні сервера – кожен роутер перевіряє наявність обов'язкових полів і повертає помилку 400 якщо щось відсутнє. Подвійна перевірка захищає від ситуації коли запит надходить не через браузер, а напряму до API.

Змінні середовища – `JWT_SECRET`, `STRIPE_SECRET_KEY`, `DATABASE_URL` – зберігаються у файлі `.env` і ніколи не потрапляють у вихідний код. Файл `.env` додано до `.gitignore`, тобто він не публікується у репозиторії. Це стандартна практика для будь-якого застосунку що працює з секретними ключами.

3.4 Тестування та налагодження інформаційно-комп'ютерної системи

Тестування застосунку проводилось вручну – перевірялись усі основні сценарії використання від реєстрації до підтвердження оплати. Автоматизовані тести для проєкту такого масштабу надлишкові, тому основний підхід – функціональне тестування через браузер і перевірка API через інструменти розробника.

Тестування охоплювало чотири основних напрямки: авторизація, пошук рейсів, процес бронювання і захищені маршрути.

Перевірка авторизації включала кілька сценаріїв. Реєстрація з коректними даними – система створює акаунт і одразу авторизує. Реєстрація з email що вже існує – сервер повертає помилку, форма показує повідомлення. Вхід з неправильним паролем – помилка 401, користувач залишається на сторінці. Вхід з коректними даними – токен зберігається в `localStorage`, шапка показує ім'я користувача. Перевірка пошуку рейсів підтвердила коректну фільтрацію результатів за напрямком, діапазоном дат та обраними параметрами. Захищені маршрути (бронювання, профіль, збережені) повертали помилку 401 при відсутності токена, що підтверджує працездатність JWT-авторизації. Часткові результати перевірки авторизації наведено на рисунку 3.5.

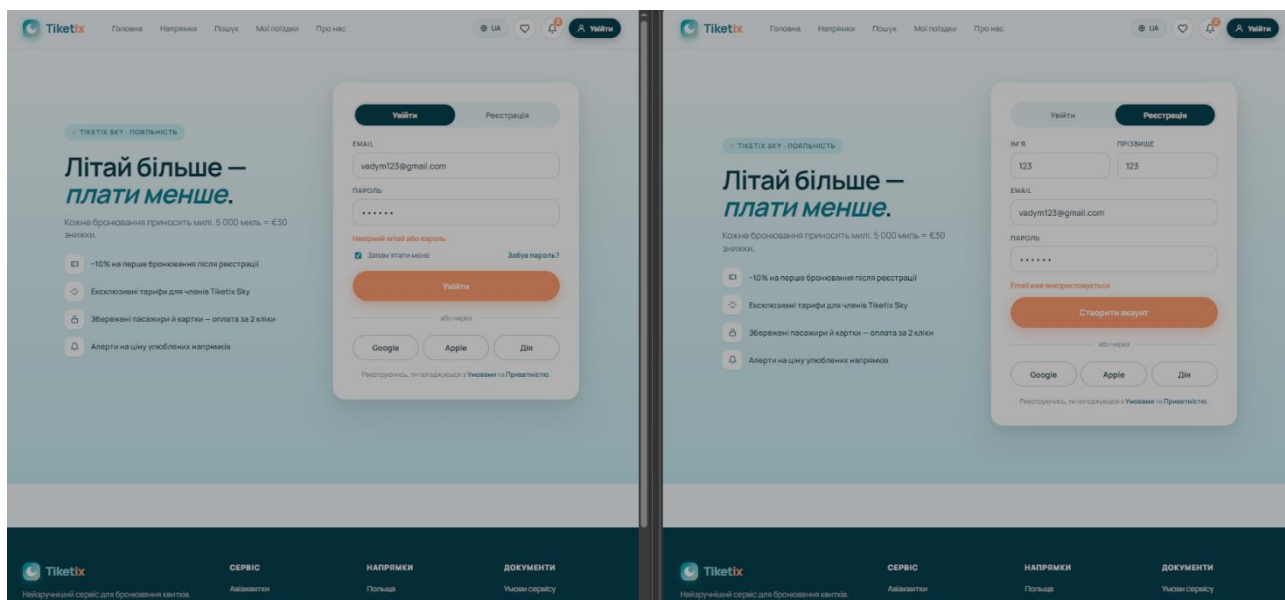


Рисунок 3.5 – Перевірка форми авторизації

Пошук рейсів тестувався з різними комбінаціями маршрутів. Існуючий маршрут – список рейсів відображається коректно з усіма полями. Неіснуючий маршрут – відображається повідомлення про відсутність рейсів. Пошук без заповнених полів – форма не відправляється, підсвічуються обов’язкові поля. Результати пошуку наведено на рисунку 3.6.

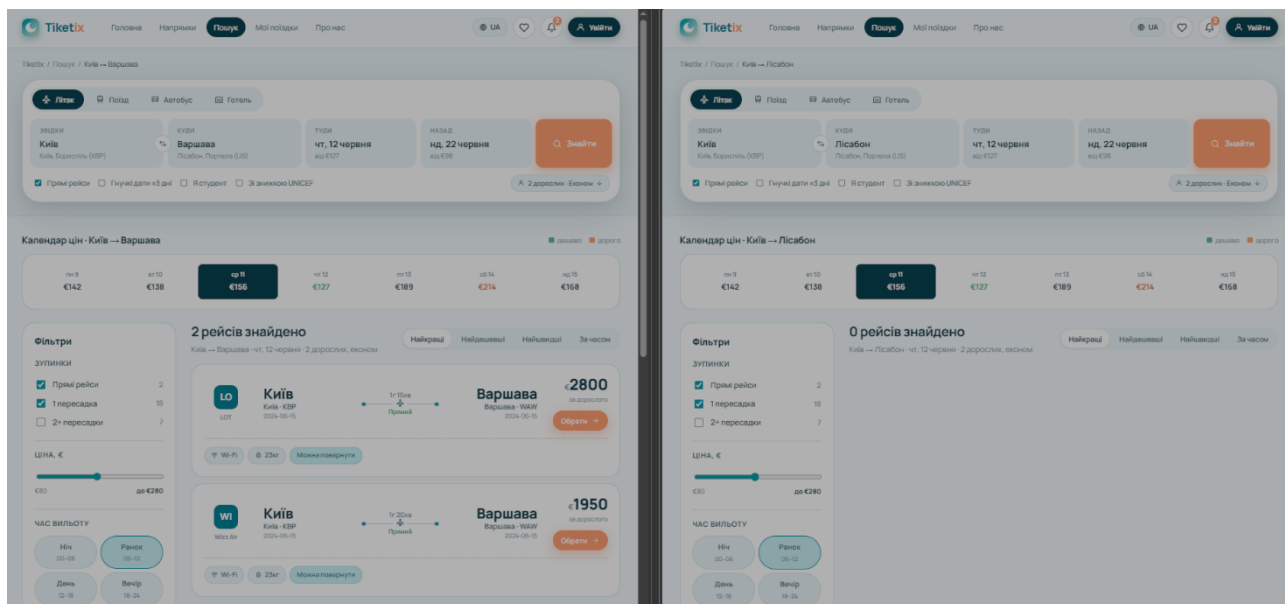


Рисунок 3.6 – Результат пошуку рейсів при тестуванні

Процес бронювання перевірявся як повний цикл: вибір рейсу, введення даних пасажирів, оплата тестовою картою Stripe 4242 4242 4242 4242. Після успішної оплати бронювання з'являлось в особистому кабінеті з коректними даними. Також перевірявся негативний сценарій – картка 4000 0000 0000 0002 яка завжди відхиляється Stripe. Форма показувала помилку без перезавантаження сторінки і залишалась активною для повторної спроби. Результати тестування оплати наведено на рисунку 3.7.

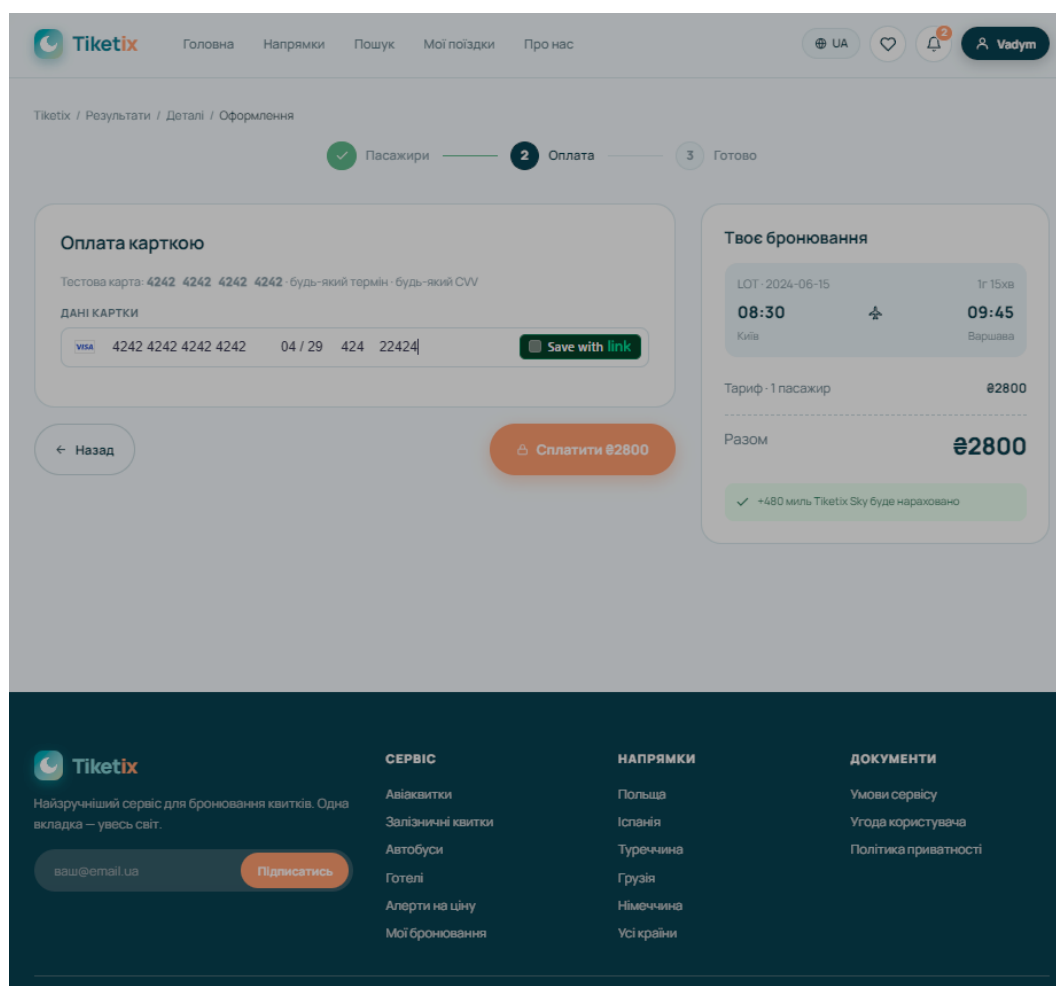


Рисунок 3.7 – Тестування оплати через Stripe

Захищені маршрути перевірялись через пряме звернення до API без токена. Запити до `/api/bookings/my` і `POST /api/bookings` без заголовка `Authorization` повертали 401 – доступ заблоковано на рівні middleware. Спроба передати

прострочений або підроблений токен – аналогічний результат. Це підтверджує, що middleware працює коректно і захищені ендпоінти недоступні без авторизації.

Окремо перевірялась адаптивність інтерфейсу – застосунок тестувався на різних розмірах екрану через інструменти розробника Chrome: мобільний (375px), планшет (768px) і десктоп (1280px). На всіх розмірах інтерфейс відображався коректно без горизонтального скролу і перекриття елементів. Результати перевірки адаптивності на планшеті наведено на рисунку 3.8.

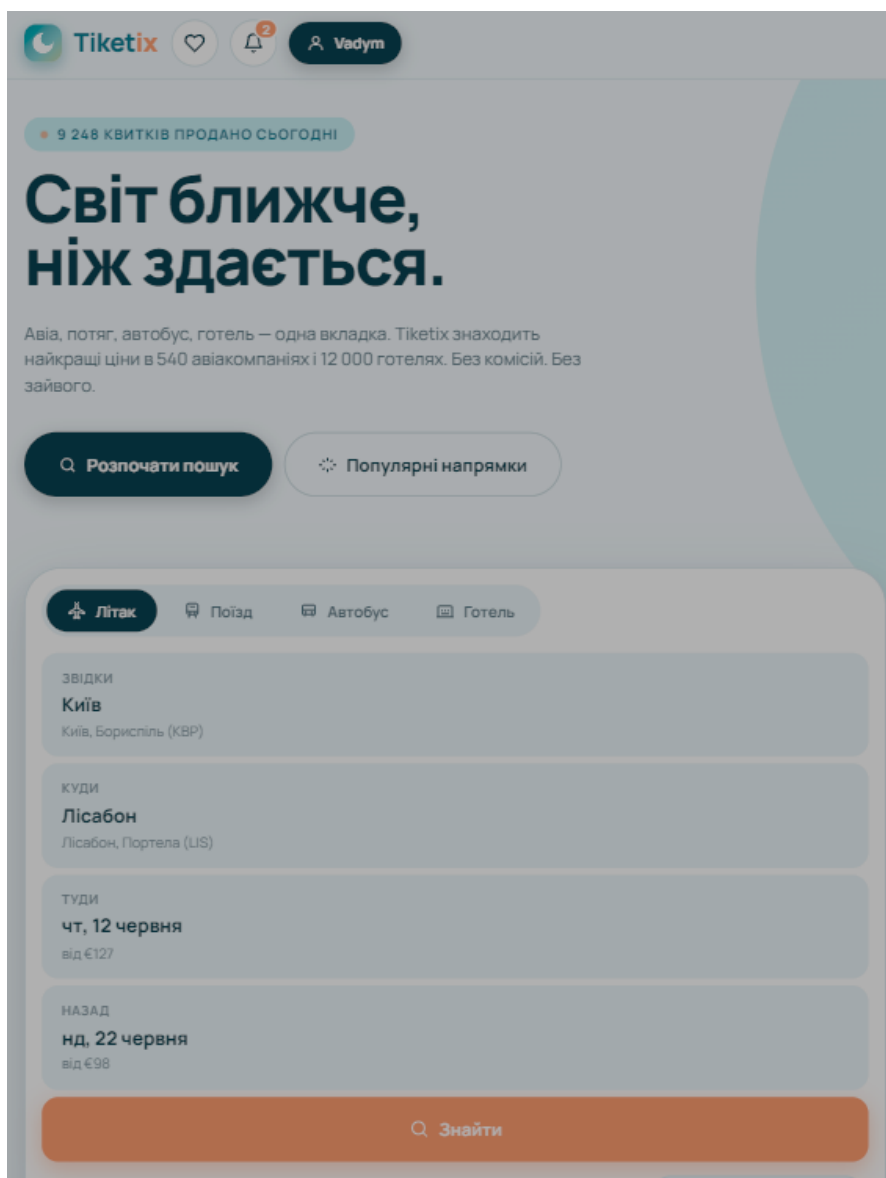


Рисунок 3.8 – Перевірка адаптивності інтерфейсу для планшету

У процесі тестування виявлено і виправлено кілька проблем. При швидкому переході між сторінками стан форми пошуку іноді не очищався –

виправлено скиданням стану при розмонтуванні компонента. При невдалій оплаті кнопка підтвердження залишалась заблокованою – виправлено скиданням стану кнопки після отримання помилки від Stripe. Ці правки зроблено безпосередньо під час тестування, до фінальної версії застосунку вони вже включені.

ВИСНОВКИ

У кваліфікаційній роботі досягнуто поставлену мету – розроблено веб-застосунок для пошуку, бронювання та купівлі туристичних квитків із реалізацією автентифікації користувачів та інтеграцією онлайн-платежів.

Розглянуто три діючих сервіси – Tickets.ua, Busfor і Booking.com – визначено їхні функціональні можливості та недоліки. Встановлено що всі наявні платформи є закритими комерційними продуктами, що обмежує можливість їх вивчення та адаптації. На основі цього обґрунтовано доцільність розробки власного рішення з відкритою архітектурою.

Сформовано перелік основних функцій: реєстрація і авторизація, пошук рейсів, перегляд деталей, оформлення бронювання, онлайн-оплата і особистий кабінет. Окремо визначено вимоги до безпеки, адаптивності інтерфейсу і валідації даних.

Для клієнтської частини обрано React і Vite, для серверної – Node.js і Express, для роботи з базою даних – Prisma ORM і SQLite, для обробки платежів – Stripe API. Такий стек забезпечив єдину мову розробки на клієнті і сервері та спростив інтеграцію між компонентами системи.

Побудовано діаграму варіантів використання, ER-діаграму бази даних і діаграму компонентів. Це дозволило формалізувати архітектуру застосунку і визначити взаємозв'язки між його частинами до початку реалізації. Створено три таблиці – User, Flight і Booking – з відповідними зв'язками між ними. Реалізовано обмеження унікальності email, зовнішні ключі для забезпечення цілісності даних і автоматичне заповнення часових полів. Базу наповнено десятьма тестовими рейсами між українськими та європейськими містами.

Розроблено дев'ять сторінок інтерфейсу, реалізовано REST API з чотирма роутерами, налаштовано маршрутизацію і передачу даних між сторінками. У роботі наведено фрагменти програмного коду що підтверджують практичну реалізацію системи.

Реалізовано хешування паролів через bcryptjs, JWT-автентифікацію з терміном дії токена 7 днів, захист платіжних даних через архітектуру Stripe і валідацію вхідних даних на стороні клієнта і сервера. Секретні ключі винесено у змінні середовища і не включено до репозиторію.

Перевірено всі основні сценарії використання – авторизація, пошук рейсів, процес бронювання і захищені маршрути. Окремо протестовано адаптивність інтерфейсу на різних розмірах екрану. Виявлені в процесі тестування помилки усунуено до фінальної версії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SNS Insider pvt ltd. Online Travel Market Size to Surpass USD 1457.52 Billion by 2032, Owing to Rising Digital Adoption and Mobile-Based Booking Trends. Report by SNS Insider. Yahoo Finance. URL: <https://finance.yahoo.com/news/online-travel-market-size-surpass-130000589.html> (дата звернення: 01.03.2026).
2. Express/Node introduction – Learn web development. MDN. MDN Web Docs. URL: https://developer.mozilla.org/docs/Learn_web_development/extensions/Server-side/express_nodejs/introduction (дата звернення: 05.03.2026).
3. Florido-Benitez L. The Cybersecurity Applied by Online Travel Agencies and Hotels to Protect Users' Private Data in Smart Cities. MDPI. URL: <https://www.mdpi.com/2624-6511/7/1/19> (дата звернення: 07.03.2026).
4. Online Travel Booking Statistics in 2026. OysterLink. URL: <https://oysterlink.com/spotlight/online-travel-booking-statistics/> (дата звернення: 09.03.2026).
5. Учасники проєктів Вікімедіа. Tickets.ua – Вікіпедія. Вікіпедія. URL: <https://uk.wikipedia.org/wiki/Tickets.ua> (дата звернення: 11.03.2026).
6. Дешеві квитки на літак онлайн. Пошук і порівняння цін на авіаквитки / Tickets.ua. Tickets flight. URL: <https://tickets.ua/> (дата звернення: 15.03.2026).
7. Busfor Україна – офіційний сайт. – URL: <https://busfor.ua> (дата звернення: 23.03.2026).
8. Booking.com: офіційний сайт. URL: booking.com (дата звернення: 27.03.2026).
9. React. React. URL: <https://react.dev/> (дата звернення: 02.04.2026).
10. Vite. vitejs. URL: <https://vite.dev/> (дата звернення: 06.04.2026).
11. Index. Node.js v26.2.0 Documentation. Node.js – Run JavaScript Everywhere. URL: <https://nodejs.org/docs/latest/api/> (дата звернення: 12.04.2026).
12. Prisma Documentation. Prisma ORM. URL: <https://www.prisma.io/docs/orm> (дата звернення: 17.04.2026).

13. JSON Web Token Documentation. URL: <https://jwt.io/> (дата звернення: 23.04.2026).

14. Documentation. Online payment service and payment service provider. Stripe. URL: <https://stripe.com/docs> (дата звернення: 28.04.2026).

15. Express Documentation. URL: <https://expressjs.com/> (дата звернення: 01.05.2026).