Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-порталу на основі технології Typescript для резервування та продажу квитків автотранспортної компанії

Виконав(ла): студент(ка) 4 курсу, групи СП-42

спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Шобський І.В. (прізвище та ініціали)

Керівник (підпис) Коноваленко І.В. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М. Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М. Р.
(підпис) (прізвище та ініціали)
« » 2025 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Шобський Іван Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка веб-порталу на основі технології Typescript для резервування та продажу квитків автотранспортної компанії

Керівник роботи Коноваленко І.В
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 2025 року № _____

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи мова програмування TypeScript, фреймворки React, NestJS, TypeORM, середовище розробки Visual Studio Code

4. Зміст роботи (перелік питань, які потрібно розробити)
Проаналізувати предметну область, спроєктувати, розробити та протестувати вебпортал, аналіз результатів і можливості подальшого розвитку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Діаграми варіантів використання, зв'язків, діяльності та послідовності, зображення графічного інтерфейсу користувача та результатів тестування, слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Затвердження теми роботи		
2	Аналіз предметної області		
3	Проєктування загальної структури системи		
4	Проєктування структур окремих модулів		
5	Програмна реалізація вебзастосунка		
6	Тестування розробленої системи		
7	Виконання розділу з безпеки життєдіяльності та основ		
	охорони праці		
8	Оформлення пояснювальної записки		
9	Попередній захист		
10	Нормоконтроль та рецензування		
11	Захист роботи		

Студент _____ Шобський І.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Коноваленко І.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота бакалавра на тему «Розробка веб-порталу на основі технології TypeScript для резервування і продажу квитків автотранспортної компанії» виконана студентом Тернопільського національного технічного університету імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-42.

Шобський І.В Розробка веб-порталу на основі технології TypeScript для резервування і продажу квитків автотранспортної компанії: кваліфікаційна робота на здобуття освітнього ступеня бакалавра за спеціальністю «121 — інженерія програмного забезпечення» / Шобський Іван Володимирович. — Тернопіль: ТНТУ, 2025. — 59 с.

Відомості про обсяг: сторінок —59, рисунків — 20 , таблиць — 2, частин — 4, додатків — 2, посилань — 14.

Метою даної роботи є розробка повноцінного веб-порталу, що дозволяє користувачам переглядати маршрути, бронювати та купувати квитки на автобусні рейси. Для досягнення цієї мети були використані сучасні веб-технології: NestJS, React, TypeScript, PostgreSQL, а також бібліотеки для генерації PDF і надсилання квитків на електронну пошту.

Запропоноване рішення реалізує клієнтську та серверну частину, багаторівневу систему доступу (гість, користувач, адміністратор), функціонал керування маршрутами та квитками, інтеграцію з електронною поштою.

Веб-портал забезпечує зручний інтерфейс, безпечну авторизацію та надійну роботу відповідно до сучасних стандартів розробки програмного забезпечення.

ABSTRACT

Bachelor's qualification work on the topic "Development of a web portal based on TypeScript technology for booking and selling tickets for a motor transport company" was completed by a student of Ivan Pulyuy Ternopil National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, Group SP-42.

Shobsky I.V. Development of a web portal based on TypeScript technology for booking and selling tickets for a motor transport company: qualification work for obtaining a bachelor's degree in the specialty "121 — software engineering" / Shobsky Ivan Volodymyrovych. — Ternopil: TNTU, 2025. — 59 p.
Information about the volume: pages — 59, figures — 20, tables — 2, parts — 4, appendices — 2, references — 14.

The purpose of this work is to develop a full-fledged web portal that allows users to view routes, book and buy tickets for bus trips. To achieve this goal, modern web technologies were used: NestJS, React, TypeScript, PostgreSQL, as well as libraries for generating PDF and sending tickets to e-mail.

The proposed solution implements the client and server parts, a multi-level access system (guest, user, administrator), route and ticket management functionality, and integration with e-mail. The web portal provides a user-friendly interface, secure authorization, and reliable operation in accordance with modern software development standards.

ЗМІСТ

ВСТУП

Сучасні інформаційні технології відіграють вирішальну роль у процесах автоматизації бізнесу, логістики та надання послуг. Однією з важливих сфер є транспортні перевезення, де впровадження електронних систем бронювання та продажу квитків дозволяє значно підвищити ефективність обслуговування пасажирів, зменшити витрати компаній та забезпечити зручність для кінцевого користувача.

Автотранспортні компанії, які обслуговують внутрішні та міжнародні маршрути, стикаються з потребою впровадження сучасних онлайн-сервісів. В умовах жорсткої конкуренції та високих очікувань користувачів надзвичайно важливо забезпечити інтуїтивно зрозумілий веб-інтерфейс, надійність системи, гнучку систему доступу та ефективне управління даними про маршрути, пасажирів і квитки.

Актуальність теми визначається необхідністю створення веб-порталу, який дозволить автоматизувати процеси вибору маршруту, бронювання та придбання квитка з використанням сучасних технологій. Обрана мова програмування — TypeScript — у поєднанні з фреймворками React та NestJS дозволяє розробити масштабоване, безпечне та підтримуване рішення.

Метою даної роботи є розробка клієнт-серверного веб-порталу для автотранспортної компанії, який реалізує повний цикл взаємодії з користувачем: реєстрація, перегляд маршрутів, бронювання, покупка квитків, отримання підтвердження на email.

Завдання дослідження:

- проаналізувати предметну область та існуючі рішення;
- спроєктувати архітектуру системи з урахуванням ролей користувачів;
- реалізувати backend-частину з REST API, авторизацією та керуванням квитками;
- реалізувати frontend-частину з панеллю для гостей, користувачів та адміністратора;

– протестувати систему та перевірити її відповідність функціональним вимогам.

Об'єкт дослідження — процес розробки веб-порталу бронювання квитків у сфері автоперевезень.

Предмет дослідження — методи та засоби розробки клієнт-серверних інформаційних систем на основі технології TypeScript.

Для реалізації проєкту використовувалися фреймворки NestJS, React, бібліотеки TypeORM, PDFKit, а також база даних PostgreSQL. Усі компоненти системи інтегровані для забезпечення повноцінної взаємодії між клієнтом та сервером.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз наявних веб-систем

У сфері організації пасажирських перевезень веб-портали для бронювання та продажу квитків відіграють ключову роль у цифровізації транспортних послуг. Така система дозволяє не лише автоматизувати процеси обліку пасажирів та продажу квитків, а й створює комфортні умови для користувачів, забезпечуючи доступ до інформації про рейси в будь-який час.

З метою визначення функціональних особливостей і виявлення недоліків уже існуючих рішень було проведено аналіз трьох актуальних систем: Busfor.ua [1], Infobus.eu [2] та Tickets.ua [3]. Дані платформи активно використовуються як перевізниками, так і клієнтами, що дозволяє оцінити їх практичну ефективність і недоліки.

Busfor.ua — популярний сервіс в Україні, який пропонує онлайн-бронювання автобусних квитків. Система має інтеграцію з багатьма українськими та європейськими перевізниками. Її функціональність охоплює:

- пошук маршрутів за заданими параметрами (напрямок, дата, кількість пасажирів);
- перегляд вартості квитків та рейсів різних перевізників;
- онлайн-оплату квитків з автоматичною генерацією PDF-документів;
- розсилку SMS та e-mail повідомлень із деталями поїздки.

Незважаючи на широкий функціонал, Busfor не надає можливості створення власної платформи для конкретного перевізника. Власник бізнесу не має прямого доступу до управління маршрутами, а також не може змінювати інтерфейс, логіку продажу чи правила підтвердження бронювань.

Infobus.eu — міжнародна платформа, що підтримує автобусні, залізничні та деякі авіамаршрути. Платформа забезпечує мультивалютну підтримку, багатомовний інтерфейс та високу інтегрованість із закордонними перевізниками. Її сильними сторонами є:

- підтримка рейсів між сотнями міст у Європі;

- оплата картками та через електронні гаманці;
- збереження історії замовлень;
- підтримка клієнтів 24/7.

Проте, з точки зору індивідуального бізнесу, платформа не дає жодного контролю над рейсами, розкладом, тарифами чи аналітикою. Інформація про перевезення формується централізовано, а адміністрування доступне лише технічній підтримці сервісу.

Tickets.ua — багатофункціональний сервіс з підтримкою бронювання залізничних, автобусних, авіа- та туристичних послуг. Його цільова аудиторія — кінцевий споживач, а не бізнес-перевізник. Серед можливостей:

- підтримка декількох видів транспорту;
- кешбек, бонуси та інтеграція з туристичними агентствами;
- повна автоматизація покупки, включно з вибором місць.

Однак система є закритою — відсутня будь-яка кастомізація для окремих перевізників. Це фактично — маркетплейс, де автотранспортна компанія лише виставляє доступні рейси без прямого керування процесами.

Таким чином, розглянуті платформи мають важливі переваги — інтегровану оплату, мобільні додатки, автоматичні повідомлення, широкий вибір маршрутів. Але для локального перевізника вони виявляються малоефективними через відсутність повного контролю над системою, непрозору аналітику, комісії та залежність від посередника, що знижує прибуток та ускладнює ведення бізнесу.

1.2 Аналіз вимог до системи

Вимоги до розроблюваної системи визначають функціональність, архітектуру, характеристики продуктивності, безпеки та обмеження, які повинні бути враховані під час проектування і реалізації веб-порталу для резервування та продажу квитків.

Функціональні вимоги

До основних функціональних вимог системи належать:

- Пошук маршрутів за заданими параметрами (місце відправлення, пункт призначення, дата поїздки).
- Перегляд інформації про рейси: маршрут, ціна, тривалість, автобус, кількість вільних місць.
- Бронювання квитків зареєстрованими користувачами з можливістю вибору місця.
- Купівля квитків онлайн із підтвердженням на електронну пошту (у вигляді PDF).
- Особистий кабінет користувача з історією замовлень, статусами квитків та можливістю скасування броні.
- Адміністративна панель для додавання/редагування маршрутів, керування користувачами, перегляду активних бронювань.

Рольова модель доступу:

Гість — має доступ до перегляду маршрутів, без можливості купівлі;

- Користувач — може бронювати та купувати квитки;
- Адміністратор — має повний доступ до керування маршрутами, підтвердження бронювань, аналітики.
- Реєстрація/вхід користувача, збереження JWT-токена для авторизації.

Нефункціональні вимоги:

- Продуктивність: відповіді сервера не довші за 500 мс при навантаженні до 100 одночасних користувачів.
- Надійність: система має коректно обробляти помилки — недоступність бази, проблеми з мережею, неправильні запити.

Безпека:

- передача даних через HTTPS;
- використання JWT-токенів з обмеженим терміном дії;
- обмеження доступу до критичних маршрутів залежно від ролі.
- Масштабованість: можливість додавання нових міст, маршрутів, автобусів без зміни базової структури.

- Сумісність: кросбраузерна підтримка (Chrome, Firefox, Safari, Edge), адаптивність до мобільних пристроїв.
- PDF-генерація квитків має відбуватися автоматично після покупки квитка.

Технологічні вимоги:

- Мова програмування: TypeScript.
- Бекенд: NestJS [4], PostgreSQL[7], TypeORM, JWT.
- Фронтенд: React[8] з React Router, Axios.

Додаткові технології:

- nodemailer (відправлення квитків на email);
- pdfkit (генерація квитків у форматі PDF);
- Swagger [9] (автоматична документація API);
- Docker (контейнеризація сервісів);
- Docker Compose (одночасний запуск frontend + backend + база даних).

Архітектурні вимоги:

- Система повинна бути реалізована за архітектурою клієнт-сервер.
- Клієнтська частина (frontend) відповідає за інтерфейс користувача.
- Серверна частина (backend) — REST API, що обробляє запити, керує бізнес-логікою, доступом до БД.
- База даних — централізоване зберігання інформації про маршрути, користувачів, квитки.

1.3 Опис акторів та прецедентів системи

У процесі аналізу предметної області та функціональних вимог було визначено основних акторів — користувачів системи, які взаємодіють з веб-порталом для резервування і продажу квитків [5]. Кожен актор має доступ до певного набору функціональних можливостей, які формують сценарії використання системи (прецеденти).

Актори системи:

- Гість (guest) — користувач, який не має облікового запису або не виконав вхід. Має змогу переглядати маршрути, загальну інформацію про квитки та доступ до сторінок авторизації і реєстрації.
- Користувач (user) — авторизований клієнт системи. Має доступ до функцій бронювання, покупки квитків, перегляду своїх квитків, історії поїздок, а також отримує підтвердження покупки на email.
- Адміністратор (admin) — працівник автотранспортної компанії. Має розширені права: керування маршрутами, підтвердження або скасування бронювань, перегляд статистики продажу.

Для вищезгаданих акторів наведено відповідні варіанти використання в таблиці 1.1.

Таблиця 1.1 – Варіанти використання веб-порталу

Код	Актор	Назва дії	Опис дії
G1	Guest	Переглянути маршрути	Отримати список наявних рейсів без потреби авторизації
G2	Guest	Зареєструватися	Створення облікового запису з email та паролем
G3	Guest	Авторизуватися	Вхід до системи з подальшим отриманням токена доступу
U1	User	Переглянути доступні маршрути	Відображення фільтрованих рейсів за напрямком, часом, ціною
U2	User	Забронювати квиток	Створення попередньої броні із вибором місця
U3	User	Придбати квиток	Оплата й отримання квитка у форматі PDF на email
U4	User	Переглянути історію	Список придбаних квитків, їх статус та можливість скасування
U5	User	Вийти з облікового запису	Завершення сесії та видалення токена з локального сховища
A1	Administrator	Додати/редагувати маршрут	Створення та оновлення інформації про рейси
A2	Administrator	Підтвердити/скасувати бронювання	Керування статусами квитків
A3	Administrator	Переглянути всі замовлення	Доступ до повного списку бронювань та замовлень
A4	Administrator	Переглянути статистику	Аналіз продажів, звітність за обраними параметрами

Діаграма варіантів використання, зображена на рисунку 1.1, є ключовим засобом для відображення взаємодії між різними користувачами системи та основних сценаріїв їхньої поведінки. Вона створена відповідно до стандарту UML (Unified Modeling Language), який забезпечує уніфікований підхід до моделювання програмного забезпечення. На діаграмі виокремлено два головні елементи: акторів, які уособлюють ролі або зовнішні сутності, що взаємодіють із системою, та варіанти використання, які описують конкретні дії або сценарії взаємодії з системою.

Рисунок 1.1 – Діаграма варіантів використання

1.4. Опис архітектури та технологій системи

Проєктований веб-портал для резервування та продажу квитків автотранспортної компанії реалізований на основі клієнт-серверної архітектури, яка забезпечує ефективну взаємодію між користувачем та сервером, а також надає

можливість масштабування системи в майбутньому. Рішення розділене на дві основні частини — фронтенд (інтерфейс користувача) та бекенд (серверна логіка), які взаємодіють через HTTP-протокол за допомогою REST API.

Клієнтська частина реалізована з використанням React — популярної JavaScript-бібліотеки для створення інтерфейсів користувача. Всі компоненти написані з використанням TypeScript [7], що забезпечує статичну типізацію, зменшує кількість помилок на етапі компіляції, та підвищує читаність і підтримуваність коду. Серед основних компонентів веб-порталу — сторінки для входу, реєстрації, пошуку маршрутів, покупки та перегляду квитків, а також адміністративна панель.

Для забезпечення стилізації інтерфейсу використовується Tailwind CSS — утилітарний CSS-фреймворк, що дозволяє швидко створювати адаптивні й сучасні інтерфейси без необхідності написання додаткових CSS-файлів. Компонентна структура React у поєднанні з Tailwind забезпечує високу швидкість розробки і гнучкість дизайну.

Комунікація між інтерфейсом та сервером здійснюється через бібліотеку Axios, яка використовується для виконання HTTP-запитів. Усі авторизовані запити супроводжуються токеном доступу (JWT), що зберігається у localStorage користувача. Це дозволяє забезпечити контроль доступу до захищених маршрутів, наприклад, перегляд особистих квитків або доступ до панелі адміністратора.

Для маршрутизації в межах SPA (Single Page Application) використовується React Router, що дозволяє реалізувати перемикання між сторінками без перезавантаження всієї програми.

Серверна частина реалізована з використанням NestJS — сучасного серверного фреймворку для Node.js, що базується на TypeScript і використовує модульну архітектуру. NestJS підтримує MVC-підхід і надає зручні інструменти для організації контролерів, сервісів, DTO, middleware, guard’ів тощо.
Основні модулі серверної частини:

- AuthModule — обробка авторизації та реєстрації користувачів з генерацією та перевіркою JWT-токенів;

- TicketsModule — логіка створення, бронювання та обробки замовлень на квитки;
- RoutesModule — обробка доступних маршрутів, фільтрації за критеріями;
- UsersModule — зберігання даних про користувачів та їх ролі;

Для документування API використано Swagger (OpenAPI) — всі публічні ендпоінти описані з можливістю тестування прямо з браузера. Це дозволяє легко перевіряти функціональність системи без потреби додаткових засобів.

Сервер також реалізує валідацію вхідних даних за допомогою бібліотеки class-validator та ValidationPipe. Це забезпечує захист від некоректних або шкідливих запитів на етапі взаємодії з API.

У системі використовується PostgreSQL [6] — потужна реляційна СУБД, яка чудово підходить для обробки транзакцій, фільтрації даних та реалізації зв’язків між сутностями (користувачі, маршрути, квитки тощо). Для взаємодії з базою даних використовується ORM TypeORM, яка дозволяє описувати сутності у вигляді класів та виконувати CRUD-операції за допомогою репозиторіїв

Клієнтська частина взаємодіє з бекендом через REST API. Запити надсилаються на сервер, де вони обробляються відповідним контролером і передаються в сервіс, що реалізує бізнес-логіку. Сервер звертається до бази даних, отримує або змінює дані, після чого повертає відповідь у форматі JSON.

У випадку авторизованих запитів токен перевіряється guard’ами NestJS, і лише після валідації користувач отримує доступ до потрібних маршрутів.
Архітектура системи забезпечує:

- Модульність та масштабованість;
- Безпечну авторизацію через JWT;
- Зручну підтримку і тестування через Swagger;
- Гнучке управління ролями користувачів;
- Швидкий та зручний інтерфейс з TypeScript-підтримкою;

Таким чином, обрана архітектура та технології забезпечують ефективну реалізацію всіх функціональних вимог до системи, її зручність у використанні, безпечність і можливість подальшого розширення.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ

2.1 Проєктування бази даних

Ефективна база даних є ключовим елементом для функціонування веб-порталу з резервування та продажу квитків автотранспортної компанії. Вона забезпечує зберігання, обробку та зв’язки між основними сутностями системи: користувачами, маршрутами, квитками та бронюваннями. Для реалізації сховища даних у системі було обрано реляційну СУБД PostgreSQL, яка є надійною, масштабованою та безкоштовною.

У проєктованій системі визначено такі основні сутності:

1.User (Користувач)

- id: унікальний ідентифікатор користувача
- email: електронна пошта
- password: хеш пароля
- full_name: ПІБ (опціонально)
- role: роль користувача (guest, user, admin)
- created_at, updated_at: часові мітки

2. Route (Маршрут)

- id: унікальний ідентифікатор маршруту
- from: пункт відправлення
- to: пункт прибуття
- departure_time: дата й час виїзду
- arrival_time: дата й час прибуття
- bus_number: номер автобуса
- price: вартість квитка
- created_at, updated_at: часові мітки

3. Ticket (Квиток)

- id: унікальний ідентифікатор квитка
- user_id: зовнішній ключ до таблиці users
- route_id: зовнішній ключ до таблиці routes
- seat_number: номер місця
- status: статус (booked, purchased, cancelled)
- purchase_date: дата купівлі
- pdf_path: шлях до згенерованого PDF
- created_at, updated_at

Один користувач може мати багато квитків (зв'язок User → Ticket: 1→n). Один маршрут може бути пов'язаний з багатьма квитками (Route → Ticket: 1→n). Квиток належить тільки одному користувачу й маршруту (n→1). Кожне бронювання також пов'язується з користувачем і маршрутом. Роль адміністратора реалізується логічно через поле role у таблиці users.

Для збереження паролів використовується алгоритм bcrypt. Валідація у форматі DTO у бекенді на NestJS забезпечує цілісність записів. Зв'язки реалізовані за допомогою TypeORM. Для розмежування ролей (guest, user, admin) використовується авторизація по JWT;PDF-файли квитків зберігаються на сервері (локально або через S3) — у таблиці квитків фіксується шлях до файлу.

Розроблена база даних забезпечує швидкий доступ до маршрутів, замовлень і квитків, надійне управління ролями та авторизацією, масштабовану структуру, що дозволяє легко додати нові модулі, як-от історію поїздок, статистику продажу тощо.

Схема БД адаптована під потреби реальної автотранспортної компанії з урахуванням безпеки, продуктивності та зручності адміністрування.

2.2. Розробка модулів системи

Розробка модулів у рамках системи резервування та продажу квитків автотранспортної компанії здійснюється з дотриманням принципів модульності, інкапсуляції та розділення відповідальностей. Архітектура проєкту базується на фреймворку NestJS і поділена на окремі функціональні блоки — модулі, контролери та сервіси, що відповідають за обробку вхідних запитів, виконання бізнес-логіки та взаємодію з базою даних.

Архітектурний підхід

Серверна частина побудована з використанням NestJS — прогресивного фреймворку для Node.js, що застосовує TypeScript та дозволяє реалізовувати архітектуру, орієнтовану на модулі. Для зберігання даних обрано СКБД PostgreSQL, ORM-інструмент — TypeORM. Аутентифікація реалізована через JWT.

Усі функціональні блоки системи згруповано в окремі модулі, які містять:

- Контролери — обробляють вхідні HTTP-запити;
- Сервіси — реалізують бізнес-логіку та взаємодію з базою даних;
- Сутності (entity) — визначають схеми таблиць бази даних.

Основні модулі

UsersModule

- Реєстрація нових користувачів;
- Отримання інформації про користувача;
- Зміна ролі (лише для адміністратора);
- Отримання списку всіх користувачів (для адміністратора).

AuthModule

- Аутентифікація користувача (вхід за email та паролем);
- Генерація та перевірка JWT токенів;
- Захист маршрутів через Guard'и.

RoutesModule

- Створення, редагування та видалення маршрутів (доступно лише адміну);
- Перегляд маршрутів для всіх користувачів;

- Фільтрація маршрутів за напрямками.

TicketsModule

- Створення квитка (бронювання або купівля);
- Перегляд особистих або усіх квитків (залежно від ролі);
- Видалення та редагування квитків;
- Генерація PDF-файлів для підтвердження покупки.

PdfModule

- Генерація PDF-документів з деталями про маршрут, дату, час і користувача.

MailerModule

- Відправка квитків у форматі PDF на email користувача;
- Інтеграція з SMTP-сервером через nodemailer.
- Контролери (Controllers)

Кожен модуль має відповідний контролер, який визначає маршрути та методи для роботи з API. Наприклад:

- UsersController — GET/POST/PUT/DELETE для операцій з користувачами;
- AuthController — POST-запити для реєстрації та логіну;
- RoutesController — методи для перегляду та модифікації маршрутів;
- TicketsController — методи для бронювання, купівлі та перегляду квитків.

Сервіси реалізують основну бізнес-логіку:

- UsersService — робота з обліковими записами;
- AuthService — перевірка облікових даних, хешування паролів;
- RoutesService — CRUD-операції з маршрутами;
- TicketsService — логіка створення квитка, генерація PDF, валідація ролей;
- PdfService — формування PDF-документа з інформацією про маршрут і користувача;
- MailerService — надсилання квитків поштою.

Клієнтська частина

Фронтенд реалізовано з використанням React і TypeScript. Основні сторінки системи:

- HomePage — загальна інформація про сервіс;
- LoginPage / RegisterPage — сторінки входу та реєстрації;

- RoutesPage — пошук маршрутів із фільтрами "Звідки" і "Куди";
- TicketsPage — персональний кабінет користувача з переліком квитків;
- AdminRoutesPage — сторінка адміністрування маршрутів;
- BookingConfirmPage — підтвердження або відхилення бронювань;
- ProfilePage — доступ до інформації про акаунт, зміна статусу, історія покупок;

Фронтенд обмінюється даними з API за допомогою бібліотеки axios, авторизація реалізована через JWT, що зберігається у localStorage. Для маршрутизації застосовується React Router.

Система підтримує три ролі:

- Гість — перегляд маршрутів без можливості купівлі;
- Користувач — повноцінна робота з квитками, доступ до історії та профілю;
- Адміністратор — керування маршрутами, підтвердження бронювань перегляд усіх квитків та користувачів;

Розмежування реалізовано за допомогою JwtAuthGuard і RolesGuard, які фільтрують доступ до контролерів залежно від ролі користувача.

Таким чином, модульна структура дозволяє чітко розмежувати логіку за функціональними блоками, спрощує підтримку та забезпечує безпеку доступу до API. Такий підхід є сучасним стандартом у розробці корпоративних веб-систем.

2.3 Опис роботи системи та зв’язків між компонентами

Для кращого розуміння функціональності та взаємодії різних частин веб-порталу з резервування і продажу квитків автотранспортної компанії, у цьому розділі описано ключові варіанти використання системи. Ці варіанти відображають зв’язки між компонентами та послідовність їхньої взаємодії:

- реєстрація користувача;
- вхід до системи;
- перегляд маршрутів;

- бронювання або купівля квитка;
- підтвердження бронювання адміністратором.

Система побудована за архітектурним принципом клієнт-серверної взаємодії. Клієнтська частина відповідає за інтерфейс користувача, а серверна — за обробку логіки, роботу з базою даних та зовнішніми сервісами (наприклад, надсилання листів, генерація квитків). Усі компоненти взаємодіють через HTTP-запити та відповіді, з використанням REST API. Ці варіанти використання подаго нище у вигляді UML-діаграм.

Діаграму діяльності для реєстрація користувача можна побачити на рис. 2.1.

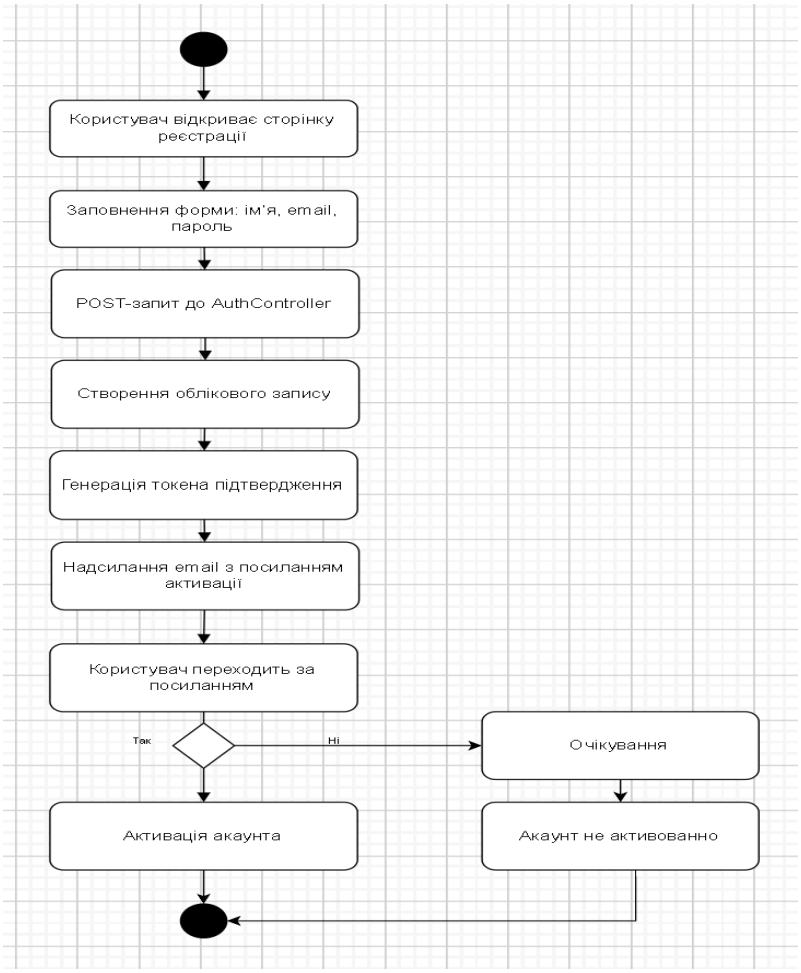

Рисунок 2.1 – Діаграма діяльності реєстрація користувача

Діаграма діяльності демонструє послідовність кроків, які відбуваються під час реєстрації нового користувача у веб-порталі для бронювання та продажу квитків. Вона охоплює як дії на клієнтській стороні (frontend), так і логіку обробки

на сервері (backend), що реалізована за допомогою NestJS та бази даних PostgreSQL.

Процес розпочинається з того, що користувач відкриває сторінку реєстрації. У відповідній формі він заповнює обов'язкові поля: ім'я, адресу електронної пошти та пароль. Після натискання кнопки «Зареєструватися» дані надсилаються HTTP POST-запитом до AuthController бекенду.

На стороні сервера в модулі автентифікації (auth.module.ts) спрацьовує AuthService, який спочатку виконує валідацію отриманих даних. Якщо дані відповідають вимогам (правильний формат email, мінімальна довжина пароля тощо), пароль хешується через bcrypt або аналогічну бібліотеку.

Далі через UsersService створюється новий запис у базі даних PostgreSQL (за допомогою TypeORM). При цьому користувач отримує статус «неактивний», а система генерує унікальний токен підтвердження. Після цього спрацьовує MailerService, який надсилає email на вказану адресу. Лист містить посилання з токеном активації. Коли користувач переходить за посиланням, бекенд обробляє GET-запит, перевіряє токен, і, у разі його дійсності, оновлює статус користувача на «активний».

Процес завершується повідомленням про успішну активацію, після чого користувач отримує змогу входити в систему та користуватися її повним функціоналом (залежно від ролі: гість, користувач або адміністратор). Ця діаграма діяльності наочно відображає узгоджену взаємодію між модулями контролю, сервісами обробки даних, надсилання листів і базою PostgreSQL, демонструючи як формується безпечний і зручний процес реєстрації з підтвердженням через email.

Вхід до системи — одна з основних функцій, що забезпечує доступ до персоналізованих можливостей користувача, таких як перегляд особистих бронювань, покупка квитків, історія поїздок та інші.

Процес починається з того, що користувач на клієнтському інтерфейсі (браузері) вводить свої облікові дані — електронну пошту та пароль. Після натискання кнопки «Увійти» формується HTTP POST-запит, який надсилається до серверного контролера AuthController. Контролер, у свою чергу, делегує логіку обробки до AuthService, який перевіряє існування користувача в базі даних, валідує

пароль (порівнюючи з хешем), і у разі успішної перевірки генерує JWT-токен (токен автентифікації).

Згенерований токен передається назад клієнту як відповідь на запит. Клієнтська частина зберігає цей токен у localStorage браузера, після чого на його основі здійснюється визначення ролі користувача (наприклад, “user”, “admin”). Залежно від ролі, динамічно оновлюється інтерфейс — з’являється особистий кабінет, опції керування квитками, а також можливості адміністратора, якщо вхід здійснено з відповідними правами доступу. Діаграму послідовностей для цього варіанту використання можна побачити на рис. 2.2.

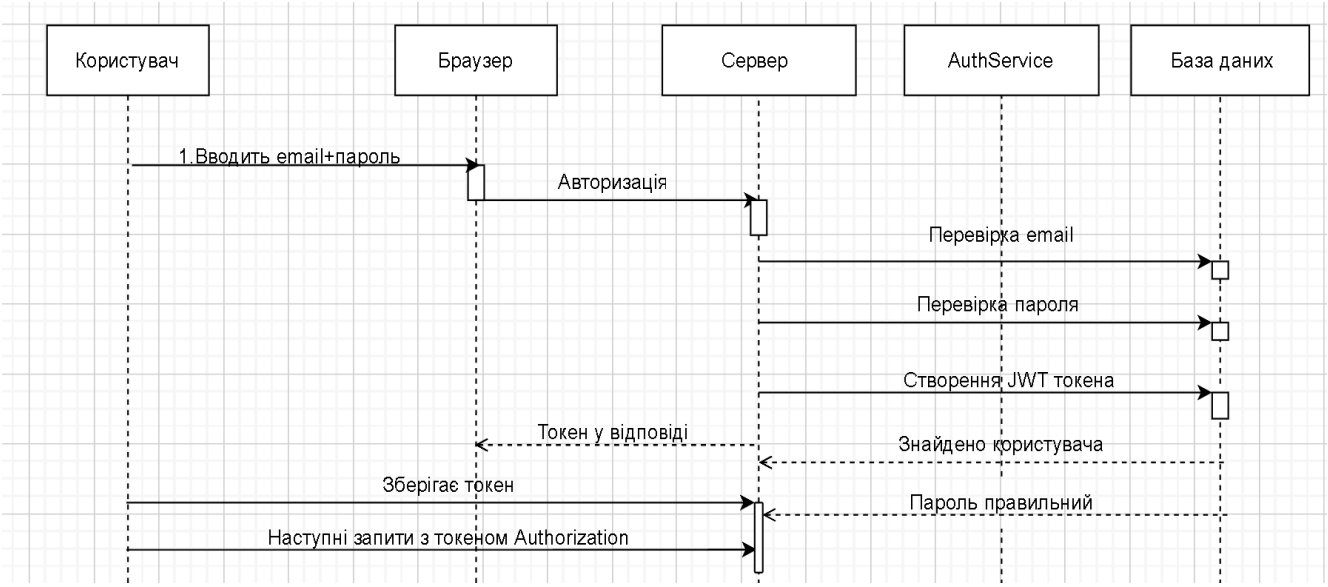

Рисунок 2.2 – Діаграма послідовностей для входу до системи

Таким чином, описаний процес демонструє чіткий ланцюг взаємодій між клієнтом, сервером і базою даних. Він є прикладом реалізації безпечної автентифікації з використанням JSON Web Token (JWT), що є стандартним підходом у сучасних веб-застосунках. Така схема дозволяє забезпечити надійний доступ до функціоналу та масштабованість у разі зростання кількості користувачів.

Діаграма демонструє послідовність взаємодії між користувачем, клієнтською частиною застосунку, сервером, сервісом і базою даних під час перегляду доступних маршрутів. Процес починається з того, що користувач відкриває сторінку «Маршрути» у браузері. Фронтенд-застосунок формує та надсилає GET-запит до сервера. Сервер передає цей запит до відповідного сервісу, який взаємодіє

з базою даних для отримання актуального списку маршрутів. Після отримання даних, сервер повертає їх фронтенду, а той відображає таблицю маршрутів користувачу у зручному вигляді. Діаграма послідовностей для входу до системи можна побачити на рис 2.3.

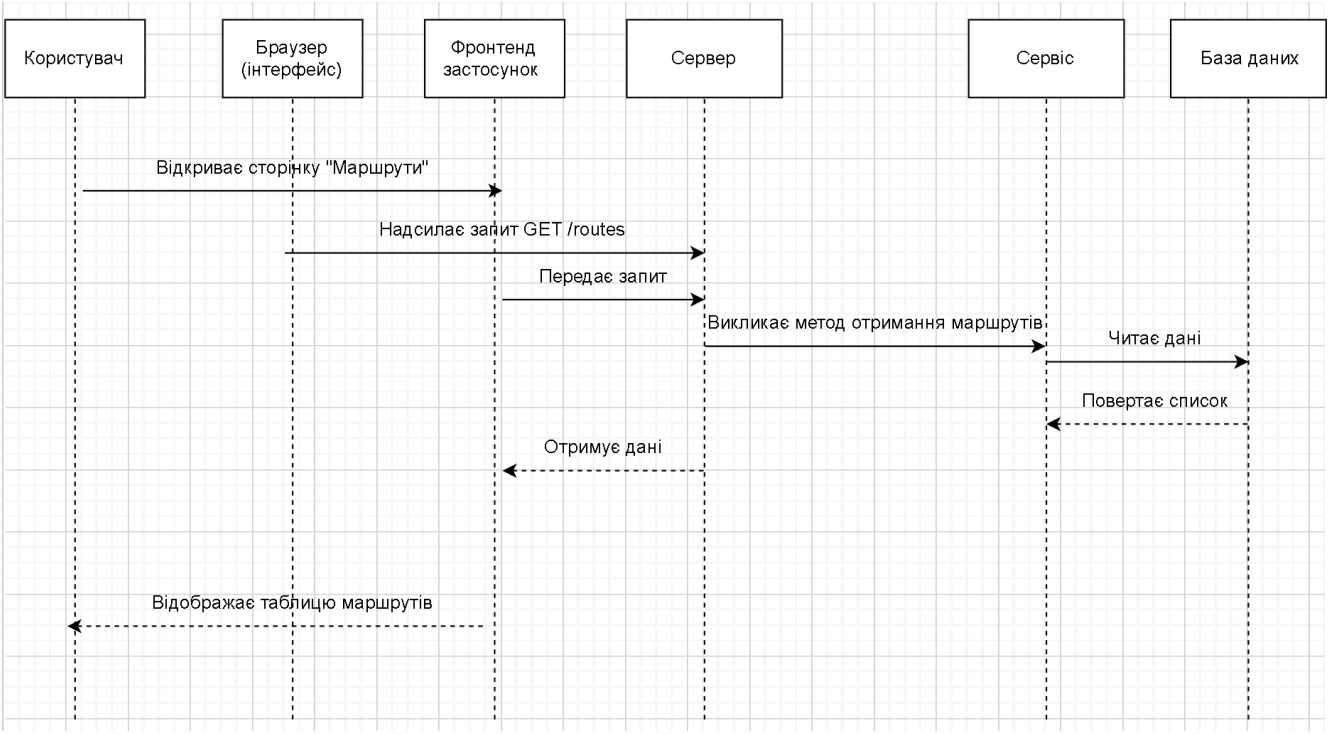

Рисунок 2.3 – Діаграма послідовностей для входу до системи

Процес бронювання або купівлі квитка починається з того, що користувач переходить на відповідну сторінку веб-порталу. Далі він обирає маршрут, дату поїздки, бажане місце та спосіб оплати. Після цього натискає кнопку «Підтвердити», що ініціює перевірку введених даних і наявності вільних місць. У випадку успішної перевірки система бронює або продає квиток, створює відповідний запис у базі даних і надсилає користувачу підтвердження (наприклад, на email). Якщо ж виникає помилка — користувач отримує повідомлення з поясненням причини і може повторити спробу. Цей процес завершується або отриманням підтвердження, або повідомленням про помилку. Діаграму діяльності для бронювання або купівля квитка можна побачити на рис. 2.4.

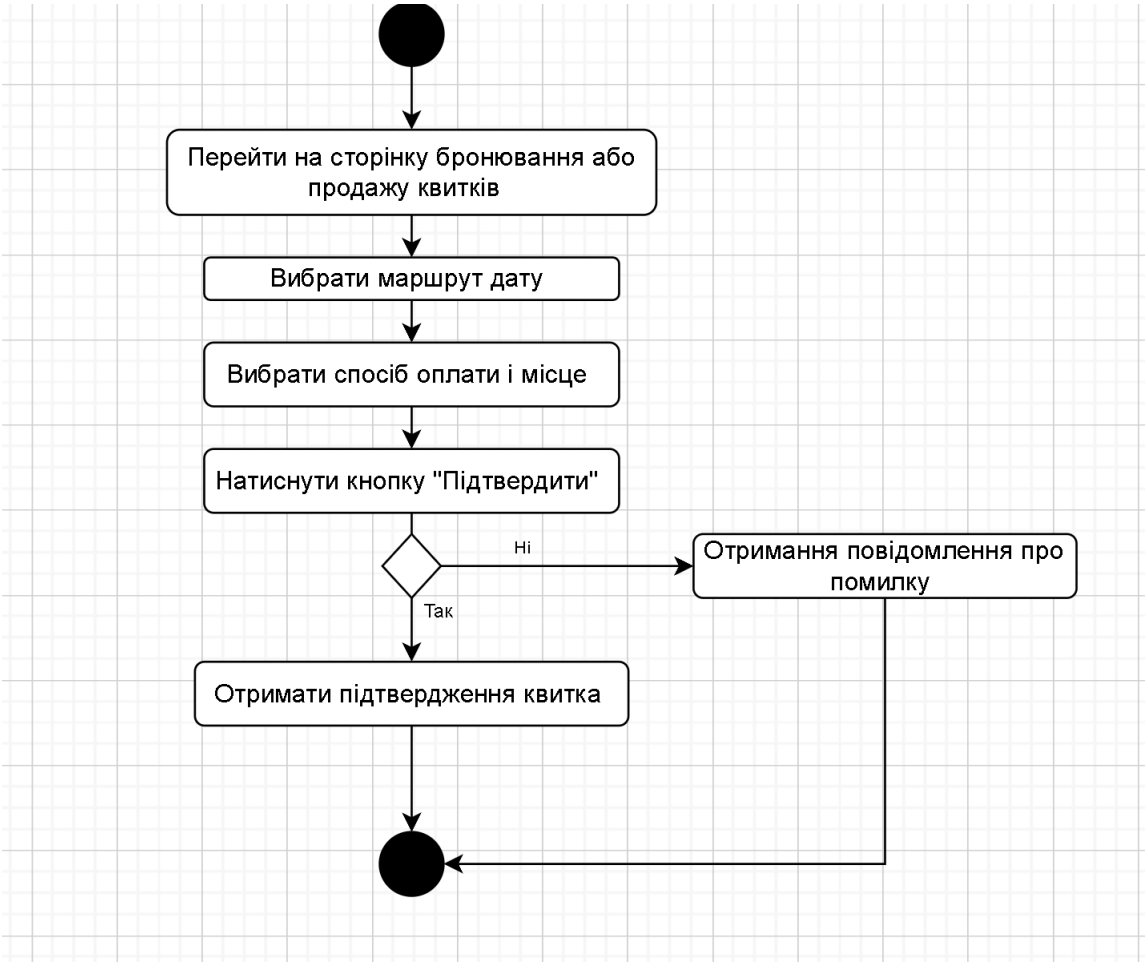

Рисунок 2.4 – Діаграма діяльності для бронювання або купівля квитка

Діаграма послідовностей підтвердження бронювання адміністратором відображає етапи взаємодії між учасниками системи: користувачем, адмін-панеллю, сервером (API), сервісною логікою та базою даних. Вона починається з того, що користувач створює запит на бронювання, який автоматично передається на сервер і зберігається у базі даних зі статусом «очікує підтвердження». Далі адміністратор відкриває адмін-панель, що ініціює GET-запит до API для отримання актуальних бронювань. API звертається до сервісу, який виконує SQL-запит до бази даних і повертає список відповідних записів.

Після перегляду адміністратор натискає кнопку «Підтвердити» біля конкретного запиту. Це генерує PUT-запит, який знову проходить через API до сервісу. Сервіс оновлює статус відповідного бронювання у базі даних. У відповідь система надсилає підтвердження до інтерфейсу адміністратора та email-повідомлення користувачу про успішне підтвердження бронювання. Діаграма

демонструє чіткий, послідовний ланцюг обробки подій, який забезпечує актуальність даних, логічну взаємодію між компонентами та прозору комунікацію між системою та її користувачами. Діаграму послідовностей для цього варіанту використання можна побачити на рис. 2.5.

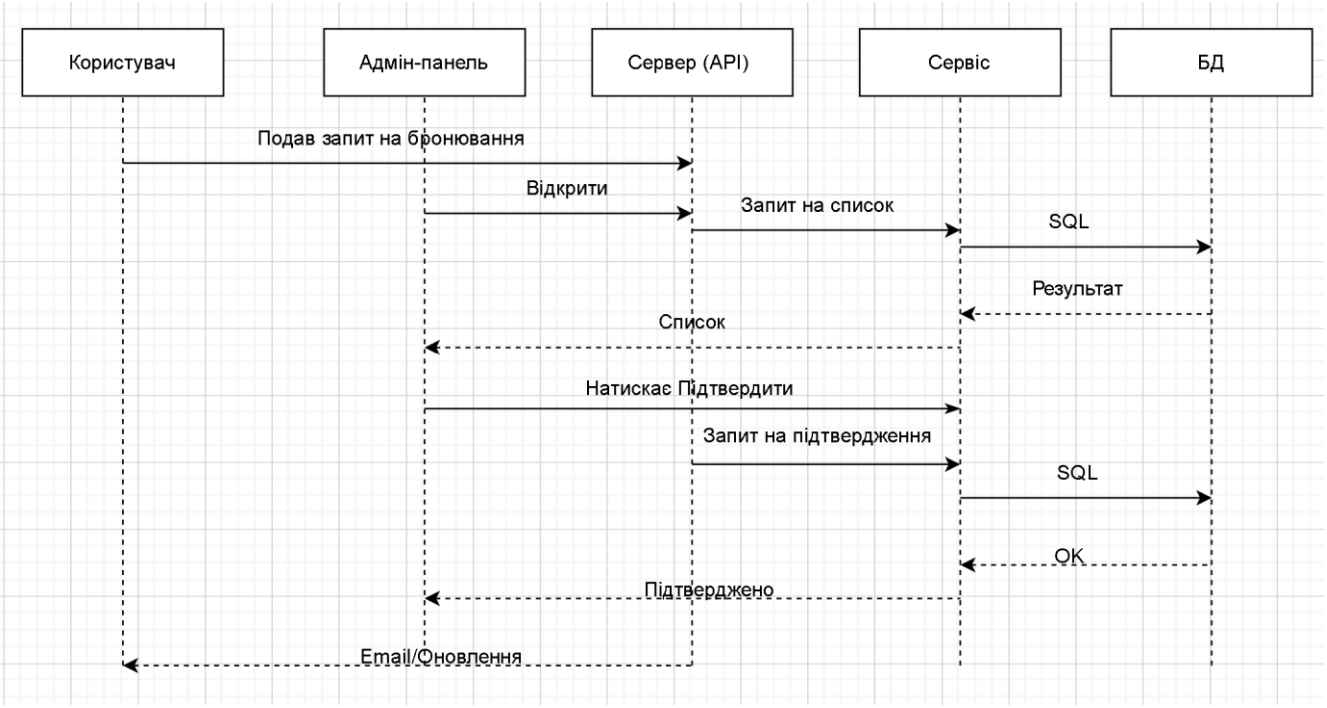

Рисунок 2.2 – Діаграма послідовностей для підтвердження бронювання адміністратором

2.4 Реалізація функціональності

У цьому підрозділі наведено приклади реалізації основних функцій веб-порталу з бронювання і продажу квитків. Це дозволяє побачити, як користувацькі дії перетворюються на логіку в межах клієнтської та серверної частин застосунку. Для реалізації використовуються технології TypeScript, NestJS, PostgreSQL на сервері та React з RTK Query на фронтенді.

Головна сторінка веб-порталу містить шапку з навігаційним меню: "Головна", "Пошук маршрутів", "Квитки", "Пошук", "Вхід", "Реєстрація". Зображення сучасного автобуса використовується як фон, що створює асоціацію з

транспортною тематикою. В центрі сторінки розташовано заголовок "Бронювання квитків" та кнопка "Пошук маршрутів". Інтерфейс адаптований під різні типи пристроїв і забезпечує зручну взаємодію для гостей. Головна сторінка показана на рис 2.6.

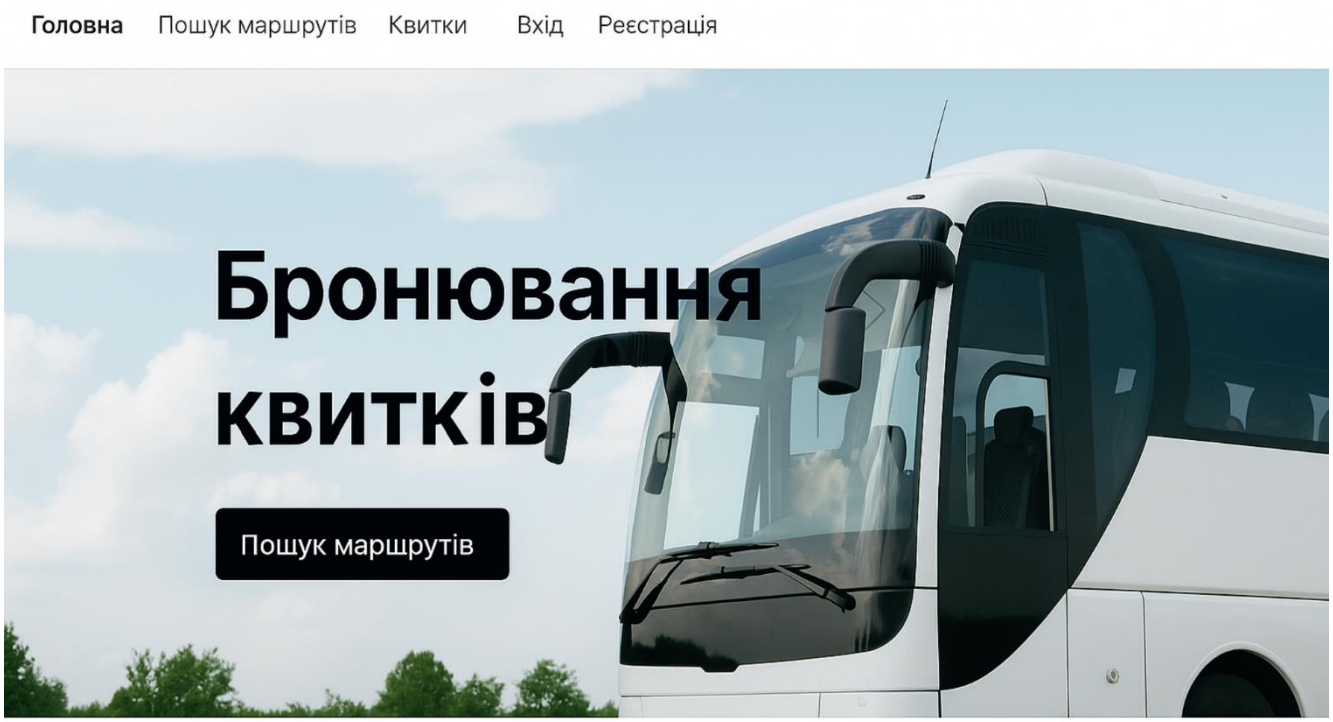

Рисунок 2.6 – Вигляд головної сторінки

Інтерфейс сторінки реєстрації реалізовано у вигляді простої форми з полями для введення імені користувача, електронної пошти та пароля. Після підтвердження форми надсилається POST-запит до API для створення нового користувача. У випадку успішної реєстрації користувач автоматично авторизується. Сторінка для реєстраціїї нового користувача показано на рис 2.7. Фрагмент коду на мові Typescripte показано на рис 2.8.

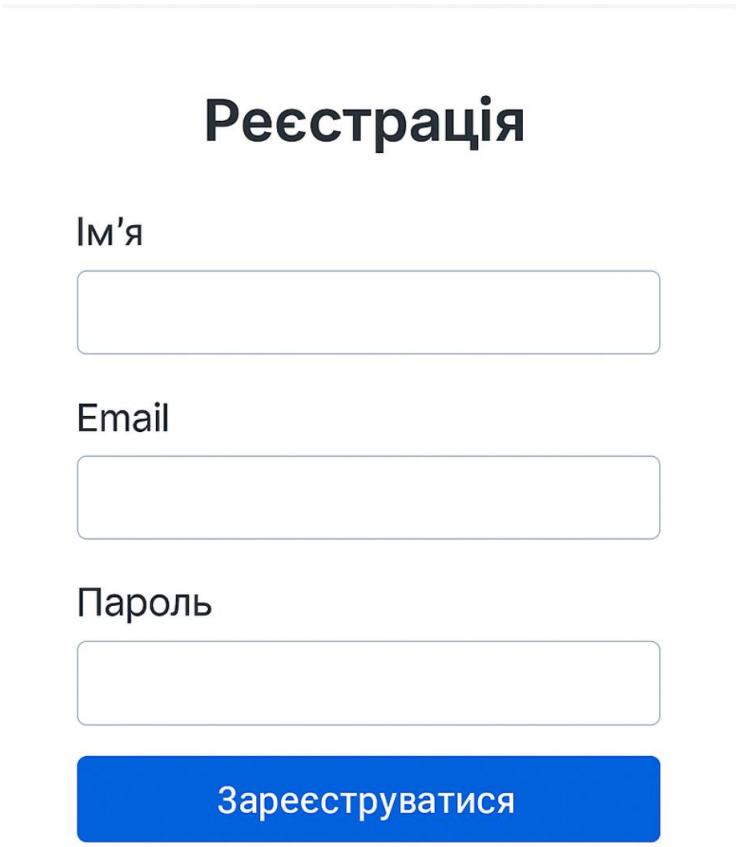

Рисунок 2.7 – Вигляд сторінки реєстрації

```
@ApiTags('auth')
@Controller('auth')
export class AuthController {
  constructor(private authService: AuthService) {}
  @Post('login')
  async login(@Body() body: LoginDto): Promise<{ access_token: string }> {
    const user = await this.authService.validateUser(body.email,
body.password);
    if (!user) {
      throw new UnauthorizedException('Invalid credentials');
    }
    return this.authService.login(user);
  }
  @Post('register')
  register(@Body() body: RegisterDto): any {
    return this.authService.register(body.email, body.password);
  }
}
```

Рисунок 2.8 – Фрагмент контролера авторизації користувача в системі

Сервіс авторизації виконує кілька важливих функцій, які забезпечують надійне управління обліковими даними користувачів. Він реалізує метод register, що хешує пароль за допомогою бібліотеки bcrypt і створює нового користувача в базі даних через сервіс користувачів. Метод login виконує перевірку введених даних за допомогою validateUser, де паролі порівнюються збереженим хешем. Якщо перевірка успішна, генерується JSON Web Token (JWT), що містить у собі ID користувача та його роль. Цей токен застосовується для автентифікації при подальших запитах до захищених маршрутів. Весь процес розділено на незалежні методи, що дозволяє повторно використовувати логіку у майбутніх розширеннях.

Такий підхід сприяє зменшенню дублювання коду та підвищенню безпеки за рахунок централізованої перевірки користувача. Також реалізована підтримка обробки винятків, яка забезпечує інформативні відповіді у випадку помилки в авторизації. Фрагмент коду на мові Typescripte показано на рис 2.9.

```
@Injectable()
export class AuthService {
  constructor(
    private usersService: UsersService,
    private jwtService: JwtService,
  ) {}
  async validateUser(
    email: string,
    password: string,
  ): Promise<Partial<User> | null> {
    const user = await this.usersService.findByEmail(email);
    if (user && (await bcrypt.compare(password, user.password))) {
      const { password, ...safeUser } = user;
      return safeUser;
    }
    return null;
  }
  login(user: Partial<User>): { access_token: string } {
    const payload = { username: user.email, sub: user.id, role: user.role
};
    return {
      access_token: this.jwtService.sign(payload),
    };
  }
  async register(email: string, password: string): Promise<User> {
    return this.usersService.create(email, password);
  }
}
```

Рисунок 2.9 – Повна реалізація AuthService з валідацією користувача, генерацією JWT та обробкою реєстрації

Користувачі мають змогу переглядати доступні маршрути у інтерфейсі з можливістю фільтрації. У бекенді реалізовано метод getRoutes, який приймає параметри запиту та передає їх до сервісного рівня. DTO відповідає за типізацію вхідних даних і забезпечує надійність запиту. Приклад сторінки перегляд маршрутів та фільтрація рис 2.10. Фрагмент коду показано на рис 2.11.

Рисунок 2.10 – Сторінка перегляду маршрутів та фільтрації

```
ApiTags('Routes')
@Controller('routes')
export class RoutesController {
  constructor(private readonly routesService: RoutesService) {}

  @Get()
  @ApiOperation({ summary: 'Отримати маршрути з фільтрами' })
  @ApiQuery({ name: 'from', required: false })
  @ApiQuery({ name: 'to', required: false })
  @ApiQuery({ name: 'minPrice', required: false })
  @ApiQuery({ name: 'maxPrice', required: false })
  async getRoutes(@Query() query: RouteFilterDto) {
    return this.routesService.findAll(query);
  }
}
```

Рисунок 2.11 – Контролер отримання маршрутів з параметрами фільтрації

Зареєстрований користувач має можливість бронювати квитки через кнопку на сторінці маршруту. Метод reserve обробляє запит і створює відповідний запис у

таблиці квитків, прив’язуючи його до ID користувача. Перед збереженням виконується базова валідація отриманих даних. Статус квитка за замовчуванням встановлюється як reserved, що означає тимчасове закріплення за користувачем. Запис зберігається у базі даних за допомогою репозиторію TypeORM. У відповідь користувач отримує JSON-об'єкт з деталями квитка. Всі операції виконуються авторизовано, з використанням JWT токена, що передається в заголовках запиту.

Такий підхід забезпечує захищене і надійне створення бронювання лише для зареєстрованих користувачів. Приклад сторінки перегляду маршруту і бронювання квитка рис 2.12.

Рисунок 2.12 – Сторінка перегляду маршруту і бронювання квитка

Адміністратор має окремий інтерфейс для створення нових маршрутів. На клієнтській частині йому доступна спеціальна форма для введення даних про новий маршрут. Дані надсилаються до захищеного ендпоінту /admin/routes, який перевіряє, чи має користувач роль admin. Метод createRoute у відповідному контролері використовує guard @Roles('admin'), що обмежує доступ лише

адміністратору. У разі успішної перевірки маршрут додається до бази даних і стає доступним для перегляду користувачами. Такий підхід забезпечує контроль над даними, які відображаються в системі, і дозволяє адміністратору оперативно оновлювати розклад або додавати нові напрямки. Крім того, архітектура реалізована таким чином, що дозволяє масштабування функцій адміністрування у майбутньому. Це забезпечує гнучкість і адаптивність системи до змін у вимогах бізнесу. Приклад сторінки інтерфейсу адміна рис 2.13. Фрагмент коду на мові Typescripte показано на рис 2.14.

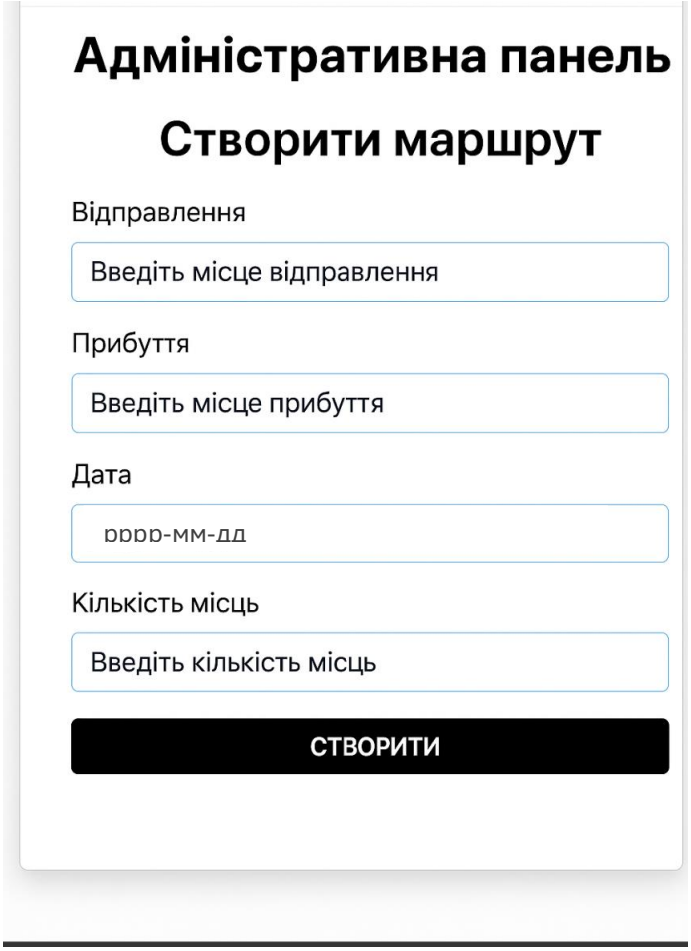

Рисунок 2.13 – Сторінка інтерфейсу адміна

```
@ApiTags('Admin')
@Controller('admin/routes')
@UseGuards(JwtAuthGuard, RolesGuard)
@Roles('admin')
export class AdminRoutesController {
  constructor(private readonly routesService: RoutesService) {}
  @Post()
  @ApiOperation({ summary: 'Створити новий маршрут' })
  async createRoute(@Body() dto: CreateRouteDto) {
    return this.routesService.create(dto);
  }
}
```

Рисунок 2.10 – Фрагмент коду панелі адміністратора для створення нового маршруту

У цьому підрозділі було представлено повну реалізацію функціональних можливостей веб-порталу з бронювання квитків. Було розглянуто всі основні компоненти — від публічних сторінок перегляду маршрутів до захищених дій, доступних лише адміністратору. Кожен фрагмент коду супроводжувався описом, прикладом застосування та візуалізацією інтерфейсу. Особливу увагу приділено архітектурі API, взаємодії фронтенду з бекендом, а також контролю доступу через механізми авторизації. Інтерфейси користувача створені з урахуванням зручності та адаптивності. Архітектура розділена на фронтенд та бекенд, що забезпечує зручність підтримки, масштабування та безпеку доступу на основі ролей користувачів. Усі приклади відповідають реальним сценаріям використання, що свідчить про практичну спрямованість розробки. Система побудована так, щоб легко адаптуватися до майбутнього розширення функціоналу.

3 ТЕСТУВАННЯ ТА ВЕРИФІКАЦІЯ ВИМОГ

3.1 Тестування серверної частини веб порталу

Тестування серверної частини є критично важливим етапом, оскільки саме сервер відповідає за обробку бізнес-логіки, управління даними в базі та взаємодію з зовнішніми сервісами, такими як платіжні шлюзи або сервіси аутентифікації. Будь-які помилки на цьому рівні можуть мати серйозні наслідки — від втрати важливої інформації до збоїв у роботі системи або порушення конфіденційності. Тестування дозволяє виявити та усунути помилки ще до запуску системи, а також забезпечити стабільність та передбачуваність її поведінки. У рамках тестування перевіряються як позитивні сценарії, так і обробка виняткових ситуацій. Зокрема, аналізується обробка запитів без авторизації, валідація полів введення, а також робота з неіснуючими маршрутами чи користувачами. Це дозволяє переконатися, що система поводиться надійно навіть у непередбачених ситуаціях. Окрему увагу було приділено перевірці механізмів контролю доступу відповідно до ролей користувачів, а також стабільності взаємодії з базою даних. Завдяки цьому серверна частина продемонструвала високу якість і відповідність заданим вимогам.

Для здійснення тестування серверної частини веб-порталу було створено низку тест-кейсів і використано інструмент Postman [10] для симуляції та перевірки запитів до API. Postman надає зручний графічний інтерфейс для надсилання HTTP-запитів і перевірки відповіді сервера. Він також дозволяє створювати колекції тестів, додавати перевірки (assertions) і тестувати негативні сценарії.

Таблиця 3.1 – Тест-кейси для тестування серверної частини веб-порталу

ID	Тип	Опис	Очікуваний результат
1	positive	Реєстрація нового користувача	Користувач успішно створений, система повертає токен авторизації
2	positive	Вхід користувача (логін)	Користувач авторизований, у відповіді присутній токен доступу
3	negative	Запит до захищеного ресурсу без токена	Система повертає повідомлення про відсутність авторизації
4	positive	Створення маршруту з роллю адміністратора	Новий маршрут успішно збережено в базі даних
5	negative	Створення маршруту без ролі admin	Система забороняє доступ і повертає повідомлення про відсутність прав

Тест-кейс 1: Було відправлено запит POST на /auth/register з валідними даними (ім’я, email, пароль). У результаті запиту було успішно створено користувача, а у відповіді повернуто JWT-токен для використання рис. 3.1.

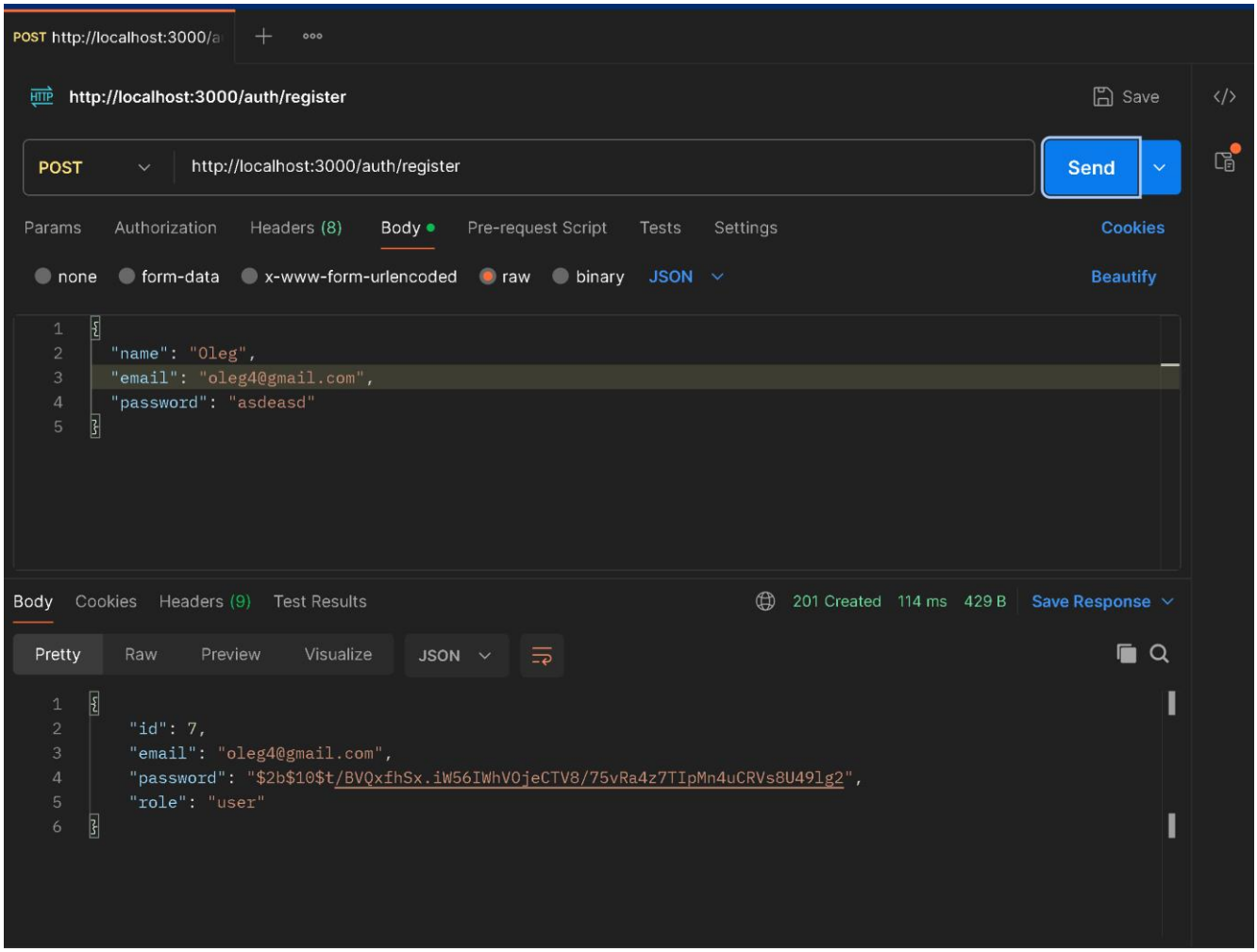

Рисунок 3.1 – Результат запит реєстрації користувача

Запит на реєстрацію користувача було успішно виконано, що підтверджується статусом 201 Created. У відповіді сервера повернуто об’єкт нового користувача з відповідними даними, включаючи роль і захешований пароль.

Тест-кейс 2: Запит на /auth/login з правильними обліковими даними повернув статус 200 та JWT у відповіді. Це дозволяє користувачу проходити авторизацію для подальших дій рис 3.2.

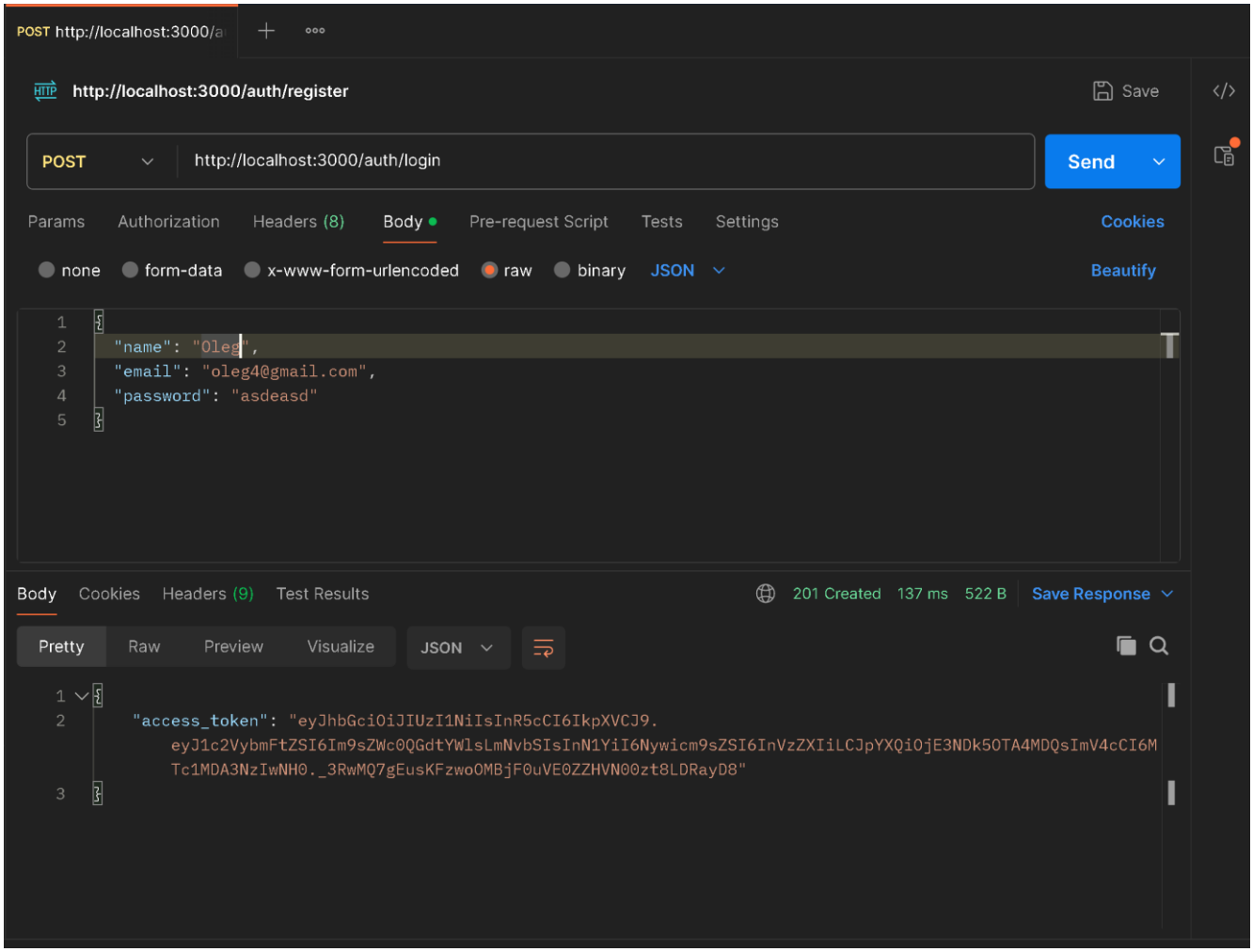

Рисунок 3.2 – Результат запит авторизації

Запит на вхід із правильними обліковими даними успішно виконано. У відповіді сервер повернув дійсний JWT-токен, який може бути використаний для доступу до захищених маршрутів API. Це свідчить про правильну реалізацію механізму авторизації у системі.

Тест-кейс 3: GET-запит до /tickets без авторизації (без токена) повернув статус 401. Це підтверджує коректну реалізацію middleware Guard у NestJS рис 3.3.

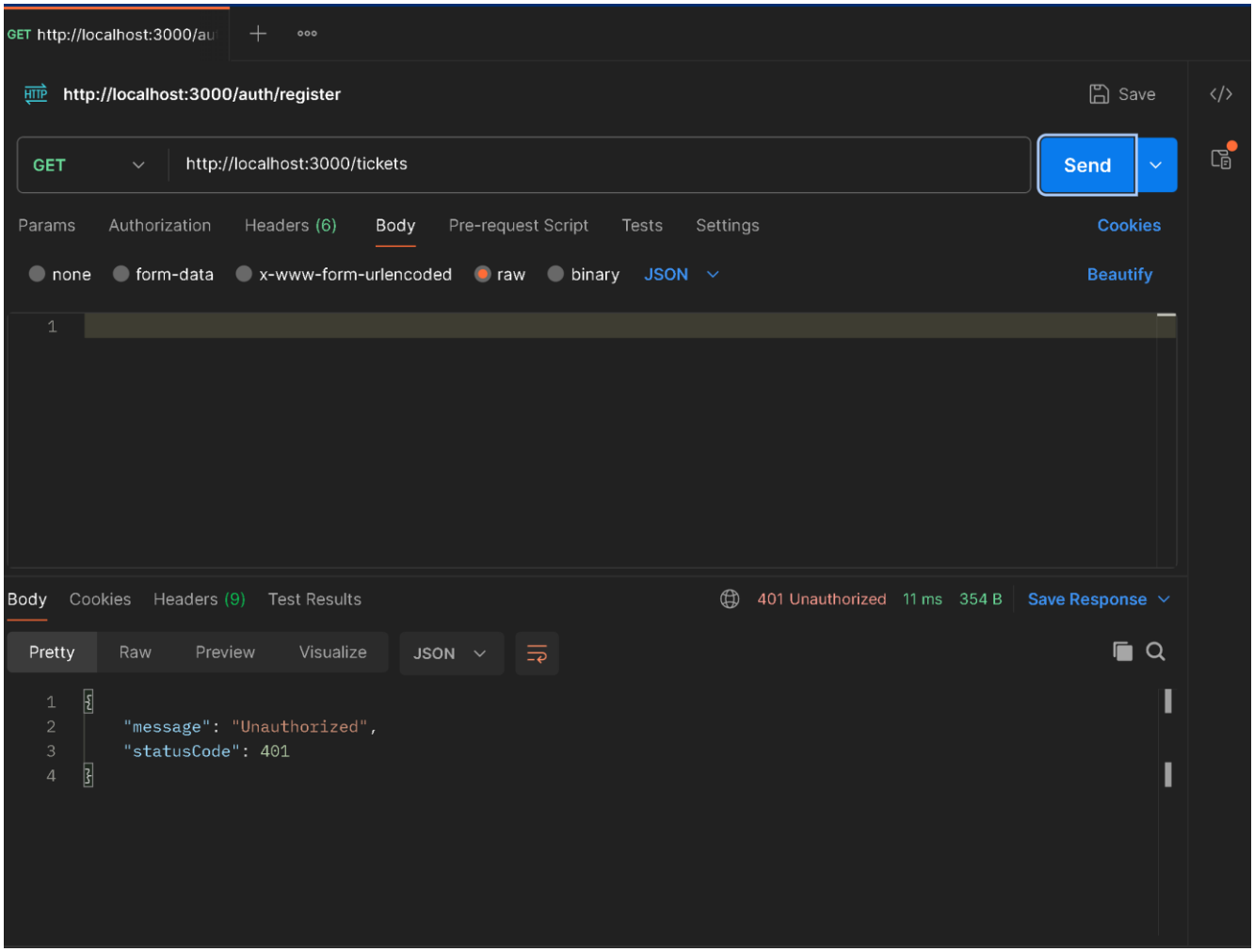

Рисунок 3.3 – Результат запит GET-запит до /tickets без авторизації

Запит до захищеного ресурсу без авторизаційного токена було відхилено з кодом 401 Unauthorized. Система чітко ідентифікує відсутність авторизації, не допускаючи неавторизованих користувачів до приватних маршрутів. Це підтверджує правильність реалізації механізму захисту маршрутів через middleware.

Тест-кейс 4: POST-запит на /admin/routes з авторизацією користувача з роллю admin створює маршрут у базі даних, відповідь — статус 201 рис 3.4.

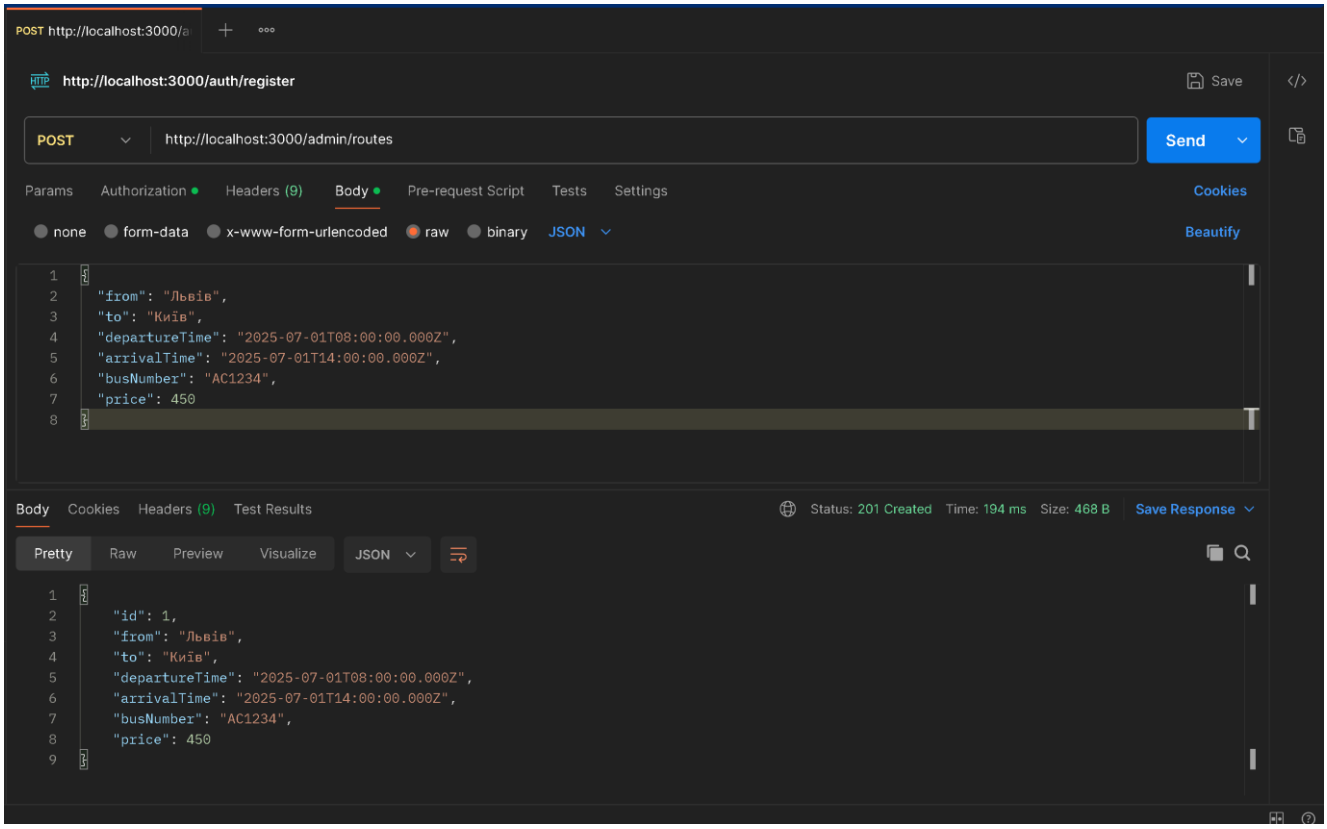

Рисунок 3.4 – Результат запит admin створює маршрут у базі даних

Запит виконано з правильними авторизаційними даними, результатом чого стало створення нового маршруту в базі даних. У відповіді повернено всі введені параметри маршруту, що підтверджує правильність роботи контролера. Таким чином, функція додавання маршруту працює згідно з вимогами до адміністратора.

Тест-кейс 5: POST-запит на той же /admin/routes з токеном користувача без адміністративних прав повертає статус 403 Forbidden, як передбачено логікою доступу в системі рис 3.5.

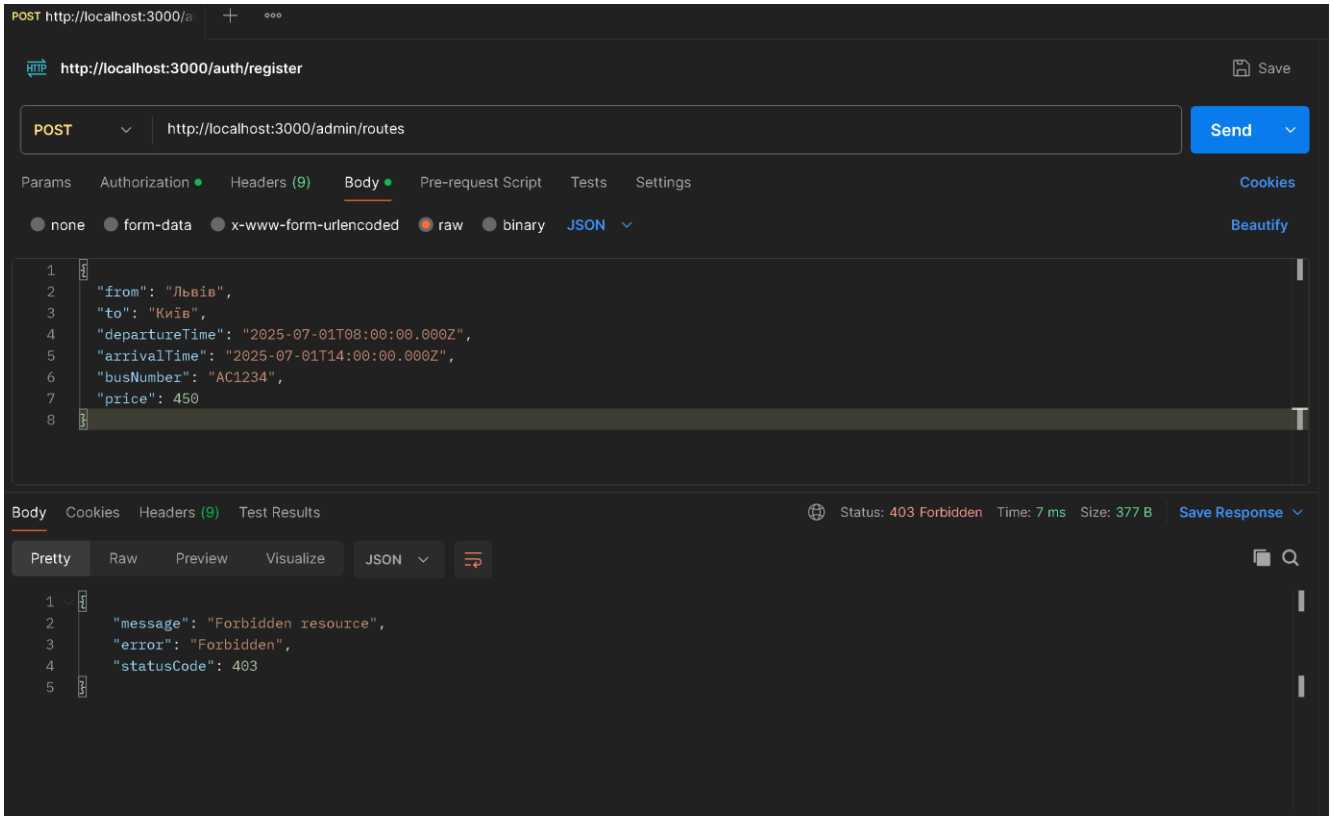

Рисунок 3.5 – Результат запит admin створює маршрут у базі даних

Система коректно обмежила доступ користувачу без ролі адміністратора, повернувши статус 403 Forbidden. Це свідчить про надійне функціонування механізму авторизації на основі ролей. Таким чином, правила доступу до адміністративних дій реалізовані відповідно до вимог безпеки.

У результаті тестування виявлено, що система виконує всі основні функції відповідно до вимог. Жодної критичної помилки не виявлено. Архітектура та логіка роботи забезпечують надійність і масштабованість системи. Це свідчить про її готовність до впровадження та подальшого використання в умовах реального середовища.

3.2 Тестування клієнтської частини веб-порталу

Тестування клієнтської частини є не менш важливим, оскільки користувацький інтерфейс є обличчям застосунку для кінцевого користувача.

Важливо забезпечити зручність використання, коректне відображення та функціонування інтерфейсу на різних пристроях та браузерах. Це включає перевірку всіх інтерактивних елементів, форм, навігації та інших компонентів інтерфейсу.

Для здійснення тестування клієнтської частини веб-системи створено алгоритм перевірки ключових сценаріїв користувача та використано інструмент Selenium IDE [11] як плагін до браузера Mozilla Firefox. Додатково було виконано перевірку адаптивного дизайну на різних розмірах екранів. Selenium IDE дозволяє автоматизувати тестування без потреби писати код вручну, забезпечує зручний запис дій користувача, підтримує команди для взаємодії з елементами DOM та може експортувати тести у форматах популярних мов програмування.

Тестування здійснювалося за наступним алгоритмом:

- Вхід до облікового запису;
- Перевірка навігації між сторінками;
- Перегляд маршрутів і фільтрація;
- Спроба бронювання квитка без входу (очікувана відмова);
- Успішне бронювання після входу;
- Перевірка адаптивності головної сторінки та форм;
- Створення маршруту з-під акаунту адміністратора.

Результати тестування клієнтської частини можна побачити на рис. 3.6. Як видно, всі етапи алгоритму виконано коректно, що свідчить про правильну роботу клієнтської логіки, валідацію та верифікацію вимог інтерфейсу.

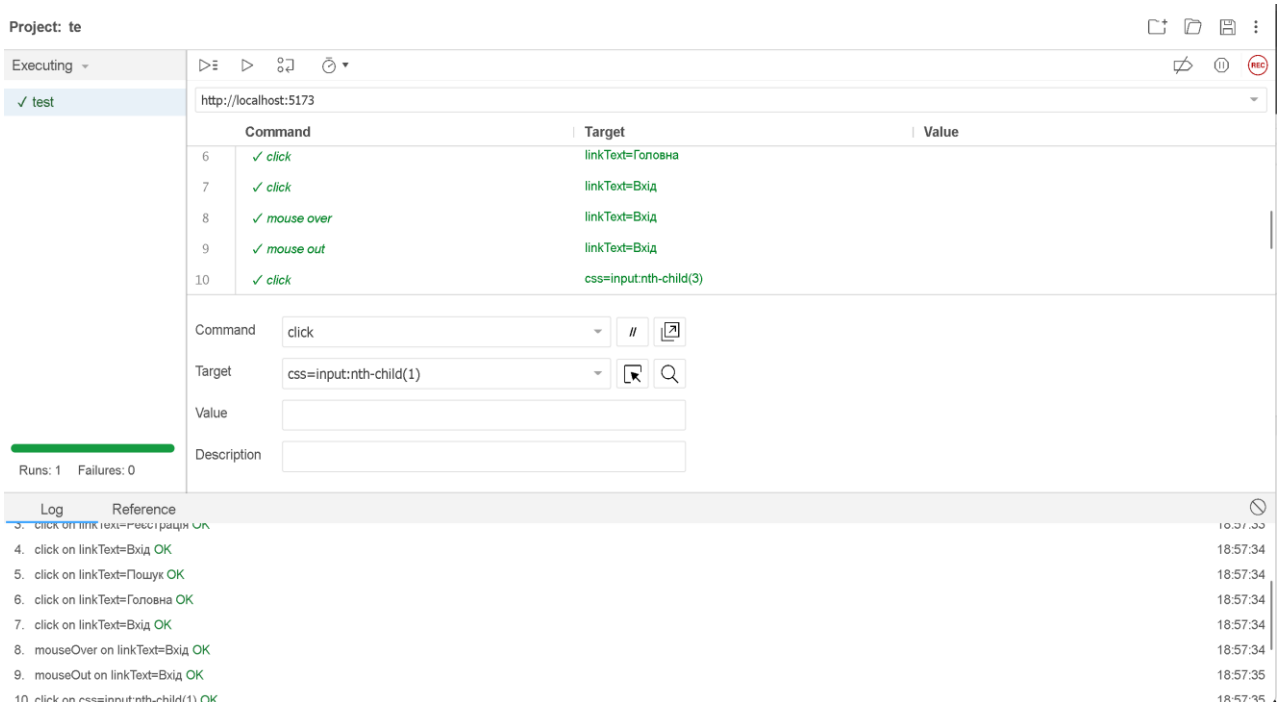

Рисунок 3.6 – Результат тестування клієнтської частини веб-порталу

3.3 Верифікація та валідація вимог

Верифікація та валідація є критично важливими процесами, які забезпечують високу якість програмного забезпечення, його відповідність вимогам замовника та зручність у використанні для кінцевого користувача. Верифікація передбачає перевірку того, чи відповідає реалізація заданим технічним специфікаціям, а валідація дозволяє переконатися, що створена система виконує свою цільову функцію та задовольняє потреби користувачів.

У межах цього проєкту верифікація здійснювалась шляхом аналізу відповідності реалізованого коду специфікації, документації API та схемам передачі даних. Було перевірено коректність відповідей серверу на різні типи запитів, обробку помилок, валідацію введених даних та правильність реалізації логіки для ролей користувачів (гості, звичайні користувачі, адміністратори).

Валідація включала ручне та автоматизоване тестування функціоналу веб-порталу з точки зору користувача. Вона дозволила виявити та усунути дрібні неузгодженості у верстці, логіці переходів між сторінками та обробці форм. Було

протестовано основні сценарії використання: реєстрація, вхід у систему, пошук маршрутів, фільтрація, бронювання квитків, керування маршрутами адміністраторами. Усі перевірки проводилися в актуальних версіях браузерів, з урахуванням адаптивності для різних розширень екранів.

Також верифікація стосувалась коректної роботи безпекових механізмів — зокрема, заборони доступу до адміністративних функцій користувачам без відповідних прав. Усі спроби несанкціонованого доступу були успішно заблоковані системою, а відповіді відповідали стандарту HTTP (401, 403).

Окрему увагу було приділено тестуванню валідації DTO об'єктів на бекенді — усі вхідні дані перевірялись за схемами, передбаченими у класах DTO, з відповідним повідомленням про помилки при порушенні правил. Це забезпечує додатковий рівень захисту даних і цілісність роботи сервісу.

Таким чином, проведені етапи верифікації та валідації підтверджують, що створена система відповідає як технічним, так і функціональним вимогам, забезпечує зручну та безпечну взаємодію з користувачем. Результати тестування дозволяють стверджувати, що веб-портал є готовим до подальшого розгортання та використання у реальному середовищі.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Надзвичайні ситуації: визначення причини, класифікація

Надзвичайна ситуація — це порушення нормальних умов життя і діяльності людей на певній території або об'єкті, що виникає внаслідок аварії, катастрофи, стихійного лиха, епідемії, терористичного акту чи інших небезпечних подій. Такі ситуації створюють загрозу для життя і здоров’я людей, можуть спричинити людські жертви, значні матеріальні збитки, руйнування інфраструктури та негативні екологічні наслідки [12].

Основними причинами виникнення надзвичайних ситуацій є як природні, так і антропогенні (тобто викликані людиною) фактори. Серед природних причин можна виділити землетруси, повені, урагани, зсуви, посухи, лавини, природні пожежі тощо. Такі явища часто називають небезпечними природними явищами, а у випадках їхнього руйнівного впливу — стихійними лихами.

До техногенних причин належать аварії на промислових об’єктах, транспорті, пожежі, вибухи, викиди небезпечних хімічних, біологічних чи радіоактивних речовин, а також руйнування життєво важливої інфраструктури, наприклад, електростанцій, водогонів або гребель.

Окремо виділяють біолого-соціальні причини, до яких належать епідемії — масові інфекційні захворювання серед людей, що охоплюють значну територію; епізоотії — інфекційні хвороби тварин, які швидко поширюються; та епіфітотії — масове захворювання або загибель рослин від хвороб чи шкідників.

Також можливі надзвичайні ситуації, спричинені протиправними діями: терактами, захопленням об’єктів, масовими заворушеннями або діями воєнного характеру. В останньому випадку НС можуть виникати внаслідок бойових дій, застосування зброї масового ураження, вибухів на складах або руйнування стратегічно важливих об'єктів.

Згідно з нормативними документами, всі надзвичайні ситуації поділяються на чотири основні класи залежно від їхнього походження:

1. надзвичайні ситуації природного характеру (стихійні лиха, епідемії тощо)

2. надзвичайні ситуації техногенного характеру (аварії, катастрофи, пожежі)
3. надзвичайні ситуації соціально-політичного характеру (терористичні акти, масові безпорядки)
4. надзвичайні ситуації воєнного характеру (викликані бойовими діями або наслідками війни)

Крім того, надзвичайні ситуації класифікуються за масштабом поширення та рівнем їхньої загрози. Встановлено чотири рівні:

- Об’єктовий рівень — ситуація не виходить за межі одного об'єкта, її можна локалізувати власними силами.
- Місцевий рівень — НС поширюється за межі об'єкта і загрожує довкіллю або населеним пунктам, але ліквідується на рівні громади чи району.
- Регіональний рівень — охоплює території кількох районів або міст, потребує допомоги з інших регіонів.
- Загальнодержавний рівень — має надзвичайно великі масштаби (дві і більше областей), може загрожувати національній безпеці, або мати транскордонні наслідки.

Урахування причин та класифікації надзвичайних ситуацій дає змогу державним органам і службам цивільного захисту своєчасно реагувати на загрози, координувати дії з ліквідації наслідків і запобігати новим небезпекам. Саме тому розробка заходів профілактики, реагування й навчання населення є одним із головних напрямів національної безпеки.

Надзвичайні ситуації становлять серйозну загрозу для життя та здоров’я людей, навколишнього середовища й економіки держави. Вони можуть виникати як унаслідок природних процесів, так і в результаті техногенних аварій, соціально-політичних конфліктів або воєнних дій.

Чітке визначення причин і класифікація надзвичайних ситуацій за характером та рівнем поширення є основою ефективної системи попередження і реагування. Розуміння цих аспектів дає змогу оперативно оцінити обстановку, правильно розподілити ресурси та мінімізувати наслідки. Тому питання

прогнозування, запобігання та ліквідації надзвичайних ситуацій набуває першочергового значення для національної безпеки України.

4.2 Пожежна профілактика на робочому місці

Пожежна профілактика — це комплекс дій і заходів, спрямованих на запобігання виникненню пожеж, зменшення шкоди від них, захист життя людей, матеріальних цінностей та навколишнього середовища [13, 14]. На робочому місці ці заходи мають систематичний характер і є частиною повсякденної діяльності працівників і керівництва підприємств.

Забезпечення пожежної безпеки є обов’язком кожного суб’єкта господарювання. Керівник підприємства зобов’язаний організувати навчання персоналу, впровадити відповідні інструкції та забезпечити технічні засоби пожежогасіння. Всі працівники, відповідно до своєї посади, зобов’язані дотримуватись правил протипожежного режиму.

Основні напрями профілактики пожеж на робочому місці поділяються на;

Організаційні заходи передбачають навчання співробітників правилам пожежної безпеки, проведення інструктажів, тренувань, лекцій, запровадження відповідальності за дотримання норм у посадових інструкціях. Також необхідно розробляти інструкції для конкретних приміщень та робочих зон.

До технічних заходів належить правильне проектування електромереж, використання надійного обладнання, встановлення пожежних бар'єрів (наприклад, брандмауерів — стін із негорючих матеріалів), герметизація трубопроводів, правильне розміщення джерел відкритого вогню (окремі приміщення для зварювальних робіт) тощо.

Експлуатаційні заходи це регулярне обслуговування та огляд обладнання, ремонт електропроводки, перевірка опалювальних та вентиляційних систем, контроль за справністю пожежної сигналізації, правильне використання засобів освітлення. Також важливо не допускати перенавантаження електромережі.

У режимних заходах забороняється проводити зварювальні та інші вогневі роботи без відповідного дозволу, палити у заборонених місцях, використовувати несправні електроприлади. Приміщення мають бути обладнані евакуаційними виходами та вогнегасниками, проходи й проїзди повинні бути вільними.

Особливу увагу потрібно приділяти приміщенням із високою концентрацією пилу, горючих матеріалів або газів, де ризик виникнення пожежі зростає. Тут не допускається використання ребристих труб, відкритого вогню чи небезпечного електрообладнання.

Загалом, пожежна профілактика є важливою складовою культури безпеки на підприємстві. Вона повинна бути системною, контрольованою та постійно оновлюваною відповідно до змін у технологіях, правилах і ризиках. Завдяки цьому можна не лише зберегти майно та запобігти нещасним випадкам, але й гарантувати безпеку працівників.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено повноцінний веб-портал для бронювання та продажу квитків автотранспортної компанії на основі технології TypeScript. Система побудована за принципами клієнт-серверної архітектури, де серверна частина реалізована з використанням фреймворку NestJS, а фронтенд — із застосуванням сучасного JavaScript-стеку. У процесі реалізації забезпечено поділ на ролі користувачів (гостьовий доступ, зареєстрований користувач, адміністратор), що дозволяє чітко структурувати функціональні можливості.

Функціонал системи охоплює всі основні сценарії користування: реєстрацію, авторизацію, перегляд маршрутів, фільтрацію за параметрами (пункт відправлення, пункт прибуття, ціна), бронювання квитків, керування маршрутами адміністраторами. Інтерфейс адаптовано до різних пристроїв, що забезпечує зручність користування як з ПК, так і з мобільних телефонів.

Окрему увагу було приділено реалізації безпечного доступу. Для авторизації застосовано JWT-токени, доступ до адміністративних функцій обмежено через Guard-механізм у NestJS. Для зберігання даних використано реляційну базу PostgreSQL, а запити до неї реалізовано з урахуванням валідації DTO-структур, що забезпечує узгодженість даних.

У процесі тестування використано інструменти Postman, Swagger UI, Selenium IDE, а також ручне тестування інтерфейсу. Тестування охопило як позитивні, так і негативні сценарії: перевірку правильності відповіді API, поведінки при несанкціонованому доступі, валідації введених даних, обробки помилок. Результати тестування підтвердили коректну роботу всіх реалізованих функцій та високу стабільність системи.

У розділі верифікації та валідації було доведено, що реалізація повністю відповідає поставленим вимогам до системи. Сформовані тест-кейси та реальні приклади сценаріїв показали, що користувачі можуть взаємодіяти з веб-порталом інтуїтивно, а всі ключові процеси працюють відповідно до очікувань.

Таким чином, поставлені цілі були досягнуті повною мірою. Результати роботи свідчать про успішну реалізацію усіх етапів розробки — від аналізу вимог до тестування. Запропонована система відповідає вимогам надійності, масштабованості, зручності використання та безпеки, що робить її готовою до впровадження у реальних умовах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Busfor — Сервіс онлайн-бронювання автобусних квитків [Електронний ресурс]. — Режим доступу: https://busfor.ua

2. Infobus — Міжнародні автобусні перевезення [Електронний ресурс]. — Режим доступу: https://infobus.eu

3. Tickets.ua — Квитки на автобуси, літаки, поїзди онлайн [Електронний ресурс]. — Режим доступу: https://tickets.ua

4. Документація NestJS [Електронний ресурс]. — Режим доступу: https://docs.nestjs.com

5. Петрик М. Р., Петрик О. Ю. Моделювання програмного забезпечення: науково-методичний посібник. — Тернопіль: Вид-во ТНТУ імені Івана Пулюя, 2015. — 200 с.

6. Документація PostgreSQL [Електронний ресурс]. — Режим доступу: https://www.postgresql.org/docs/

7. Документація TypeScript [Електронний ресурс]. — Режим доступу: https://www.typescriptlang.org/docs/

8. React documentation [Електронний ресурс]. — Режим доступу: https://react.dev/

9. Swagger UI documentation [Електронний ресурс]. — Режим доступу: https://swagger.io/tools/swagger-ui/

10. Postman Learning Center [Електронний ресурс]. — Режим доступу: https://learning.postman.com/

11. Selenium IDE documentation [Електронний ресурс]. — Режим доступу: https://www.selenium.dev/selenium-ide/

12. Про затвердження Положення про класифікацію надзвичайних ситуацій техногенного та природного характеру : постанова Кабінету Міністрів України від 17 вересня 1998 р. № 1099 [Електронний ресурс]. – Режим доступу https://zakon.rada.gov.ua/laws/show/368-2004-%D0%BF#Text – Назва з екрана.

13. Про затвердження Правил пожежної безпеки в Україні : наказ МВС України від 30 грудня 2014 р. № 1417 [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/z0252-15#Text24 – Назва з екрана.

14. Безпека життєдіяльності, основи охорони праці : електронний курс / ТНТУ ім. І. Пулюя [Електронний ресурс]. – Режим доступу: https://dl.tntu.edu.ua/content.php?cid=299415 – Назва з екрана.

15. Дистанційний курс «Кваліфікаційні роботи бакалаврів» сайту дистанційного навчання ТНТУ [Електронний ресурс]. – Режим доступу: URL: https://dl.tntu.edu.ua/bounce.php?course=5329

ДОДАТКИ

ДОДАТОК А – ЛІСТИНГ

1. APP.CONTROLLER.TS

// Фрагмент вихідного коду

```
@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get()
  getHello(): string {
    return this.appService.getHello();
  }
}
```

2. MAIN.TS

// Фрагмент вихідного коду

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.enableCors({
    origin: 'http://localhost:5173',
  });

  const config = new DocumentBuilder()
    .setTitle('Ticket Booking API')
    .setDescription('REST API для системи бронювання квитків')
    .setVersion('1.0')
    .addBearerAuth()
    .build();

  const document = SwaggerModule.createDocument(app, config);
  SwaggerModule.setup('api/docs', app, document);

  await app.listen(3000);
}
void bootstrap();
```

3. AUTH.CONTROLLER.TS

// Фрагмент вихідного коду

```
@ApiTags('auth')
@Controller('auth')
export class AuthController {
  constructor(private authService: AuthService) {}

  @Post('login')
  async login(@Body() body: LoginDto): Promise<{ access_token: string }> {
    const user = await this.authService.validateUser(body.email, body.password);
    if (!user) {
      throw new UnauthorizedException('Invalid credentials');
    }
    return this.authService.login(user);
  }

  @Post('register')
  register(@Body() body: RegisterDto): any {
    return this.authService.register(body.email, body.password);
  }
}
```

5. ADMIN.ROUTES.CONTROLLER.TS

// Фрагмент вихідного коду

```
@ApiTags('Admin')
@Controller('admin/routes')
@UseGuards(JwtAuthGuard, RolesGuard)
@Roles('admin')
export class AdminRoutesController {
  constructor(private readonly routesService: RoutesService) {}

  @Post()
  @ApiOperation({ summary: 'Створити новий маршрут' })
  async createRoute(@Body() dto: CreateRouteDto) {
    return this.routesService.create(dto);
  }
}
```

6. ROUTES.CONTROLLER.TS

// Фрагмент вихідного коду

```
@Controller('routes')
@UseGuards(JwtAuthGuard, RolesGuard)
export class RoutesController {
  constructor(private readonly routesService: RoutesService) {}

  @Get(':id')
  async findOne(@Param('id') id: number): Promise<Route> {
    const route = await this.routesService.findOne(id);
```

```
    if (!route) {
      throw new NotFoundException(`Route with ID ${id} not found`);
    }
    return route;
  }

  @Post()
  @Roles('admin')
  create(@Body() dto: CreateRouteDto) {
    return this.routesService.create(dto);
  }

  @Put(':id')
  @Roles('admin')
  update(@Param('id', ParseIntPipe) id: number, @Body() dto: Partial<Route>) {
    return this.routesService.update(id, dto);
  }

  @Delete(':id')
  @Roles('admin')
  remove(@Param('id', ParseIntPipe) id: number) {
    return this.routesService.remove(id);
  }
}
```

ДОДАТОК Б – Диск із кваліфікаційною роботою бакалавра