Вінницький національний технічний університет

(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматики
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська кваліфікаційна робота на тему:

«Програмний модуль бронювання залізничних квитків»

Виконав: студент 4 курсу, групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)

Павленко О. П.
(прізвище та ініціали)

Керівник: асистент кафедри КН
Малініч І. П.
(прізвище та ініціали)

«04» червня 2025 р.

Рецензент: к.т.н., проф., доц каф АІІТ

Паламарчук Є.А.
(прізвище та ініціали)

«05» червня 2025 р.

Допущено до захисту
Зав. кафедри КН Яровий А.А.
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
«05» травня 2025 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Павленку Олександру Павловичу

1. Тема роботи: Програмний модуль бронювання залізничних квитків
керівник роботи: Малініч І. П., асистент
затверджені наказом вищого навчального закладу "20" березня 2025 року №97
2. Строк подання студентом роботи 30 травня 2025.р
3. Вихідні дані до роботи : мова програмування C# та React, бібліотека Entity Framework, СКБД MSSQL, середовище розробки Visual Studio,
4. Зміст текстової частини (перелік питань, які потрібно розробити):
описати завдання розробки системи, аналіз існуючих систем, зробити опис засобів розробки програмного забезпечення, розробка програмного забезпечення описати процес використання системи користувачами
5. Орієнтований перелік ілюстративного матеріалу рисунки, що демонструють роботу алгоритмів інтерфейсу та візуалізують теоретичний матеріал, архітектуру системи , взаємодію користувачів із системою, приклади роботи з програмним продуктом

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Малініч І. П., асистент каф. КН		

7. Дата видачі завдання 05. травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз предметної області	05.05.25 – 07.05.25	
2	Проєктування архітектури системи, модульної структури та бази даних	08.05.23 – 15.05.25	
3	Реалізація клієнтської та серверної частин системи	16.05.25 26.05.25	
4	Тестування, налагодження та інтеграція модулів	25.05.25 01.06.25	
5	Оформлення пояснювальної записки та підготовка до захисту	02.06.25 – 04.06.25	

Студент _____ Павленко О.П.

(підпис) *(прізвище та ініціали)*

Керівник роботи _____ Малініч І.П.

(підпис) *(прізвище та ініціали*

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 89 сторінок формату А4, містить 14 рисунків, список використаних джерел налічує 20 найменування.

Ця дипломна робота присвячена створенню та впровадженню інформаційної системи для онлайн-бронювання залізничних квитків. Основна мета проєкту — розробити зручний, швидкий та безпечний сервіс, який дозволить користувачам ефективно здійснювати пошук і бронювання квитків, забезпечуючи при цьому комфортний рівень обслуговування.

На початковому етапі було проведено аналіз існуючих аналогічних систем, вивчено їх сильні сторони та виявлено типові недоліки. На основі цього сформульовано перелік функціональних і технічних вимог до майбутньої системи.

У ході реалізації використано сучасні інструменти та технології: C# як основна мова для бекенду, бібліотека React — для побудови інтуїтивного й динамічного інтерфейсу користувача. Платформа .NET Core була обрана як основа серверної частини через свою стабільність та хорошу масштабованість. Для збереження даних використовувався Microsoft SQL Server, а доступ до бази реалізовано через ORM Entity Framework. Розробку та відлагодження коду здійснено в середовищі Visual Studio.

Архітектура системи включає основні модулі: реєстрація, вхід до системи, пошук рейсів, вибір місць, оформлення замовлень, оплата та управління обліковим записом користувача. Проведено комплексне тестування, підготовлено документацію та виконано доопрацювання на основі виявлених помилок.

Структурно робота складається з вступу, теоретичного огляду, опису реалізації, тестування та підсумкових висновків. Усі етапи проєкту реалізовано з використанням сучасних технологій, що дозволило досягти високої надійності та зручності у користуванні розробленою системою.

ABSTRACT

The bachelor’s thesis consists of 89 A4 pages, contains 14 figures, and the list of references includes 20 sources.

This bachelor's thesis focuses on the development and implementation of an online railway ticket booking system. The main goal of the project is to create a convenient, fast, and secure service that enables users to efficiently search for and book tickets while ensuring a high level of user experience.

At the initial stage, an analysis of existing booking systems was conducted to identify their strengths and typical weaknesses. Based on this analysis, a list of functional and technical requirements for the future system was defined.

Modern technologies and tools were used during implementation: C# was chosen for backend development, and the React library was used to build a dynamic and user-friendly interface. The .NET Core platform served as the backend foundation due to its reliability and scalability. Microsoft SQL Server was used for data storage, with Entity Framework applied for database access. Microsoft Visual Studio was used as the main development environment.

The system architecture includes key modules such as user registration and authentication, route search, seat selection, booking processing, payment, and profile management. The system underwent comprehensive testing, documentation was prepared, and fixes were made based on identified issues.

The structure of the thesis includes an introduction, theoretical review, implementation section, system testing, and final conclusions. Each stage was completed using modern development practices, resulting in a reliable and user-friendly booking system.

ЗМІСТ

ВСТУП

Актуальність дослідження. У сучасному швидкому темпі життя користувачі надають перевагу онлайн-бронюванню залізничних квитків завдяки зручності та оперативності цього процесу. Втім, наявні сервіси часто не відповідають очікуванням через недостатню функціональність, заплутаний інтерфейс або проблеми з інтеграцією платіжних та бонусних систем. Через це користувачі можуть відмовлятися від таких сервісів, а компанії втрачають потенційних клієнтів. Саме тому є потреба у вдосконаленні існуючих систем бронювання шляхом розширення їх функціональних можливостей, підвищення швидкодії та зручності використання, особливо на мобільних пристроях.

Мета дослідження — покращення ефективності та зручності процесу онлайн-бронювання залізничних квитків шляхом розширення функціональних можливостей системи, спрощення її інтерфейсу та підвищення рівня інтеграції з платіжними сервісами.

Об’єкт дослідження — процеси автоматизації онлайн-сервісів пасажирських перевезень, зокрема бронювання залізничних квитків.

Предмет дослідження — інформаційна система бронювання квитків, розроблена у рамках дипломного проєкту, зокрема її архітектура, функціональні модулі та інтерфейс користувача.

Для досягнення мети визначені наступні завдання:

1. провести аналіз сучасних методів і підходів до автоматизації процесів бронювання квитків;
2. оцінити можливості й обрати відповідні програмні засоби для реалізації поставлених задач;
3. розробити інформаційну систему з розширеними функціями бронювання;
4. провести тестування розробленого продукту та оцінити його ефективність за визначеними критеріями.

Попит на сучасні та ефективні онлайн-сервіси бронювання постійно зростає, що робить актуальним створення вдосконалених систем. Розроблена інформаційна

система дозволить суттєво покращити взаємодію користувачів із сервісом, підвищити конкурентоспроможність компаній, а також створити основу для подальшого розвитку і масштабування функціоналу.

Дипломна робота включає вступ, основну частину з аналізом, проєктуванням та реалізацією системи, а також висновки щодо досягнення поставлених цілей. У процесі виконання використовувалися сучасні технології, що дозволили створити стабільну та ефективну інформаційну систему бронювання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВІДОМИХ РЕАЛІЗАЦІЙ СИСТЕМ БРОНЮВАННЯ КВИТКІВ

1.1 Формулювання основної проблеми

У наш час зручне бронювання квитків на потяг є необхідною потребою, а не бонусом. Відповідно, основне завдання створення цієї інформаційної системи полягає в тому, щоб надати людям простий і надійний спосіб швидкого придбання квитків без ризиків і труднощів. Основна мета — зробити сервіс доступним для кожного, незалежно від того, чи користувач «на ти» з комп'ютером чи Інтернетом.

Зручний пошук квитків є важливим моментом, оскільки він дозволяє задати дату, час, тип поїзда та вагон і швидко побачити, що є в наявності. Це дозволить людям вибирати те, що їм найкраще, не витрачаючи часу.

Можливість миттєвої онлайн-оплати квитка також є важливою деталлю. Просто в кількох кліках, прямо з дому чи на телефоні, без черг і друку. Для цього система повинна мати можливість інтегруватися з відомими платіжними сервісами.

Безпека також важлива, особливо щодо особистих даних і платежів. Щоб користувачі могли спокійно користуватися сервісом, безпека від зловмисників, шифрування даних і загальна надійність мають бути на першому місці.

Крім того, система повинна відображати поточну інформацію про доступність, а також дозволяти переглядати розклад і вибрати місце у вагоні. Це допоможе клієнтам краще планувати свої подорожі та мати впевненість, що вони все контролюють.

Збереження історії бронювань — ще одна корисна функція. У кілька кліків можна повторити замовлення, якщо хтось часто використовує той самий маршрут. Це ще більше полегшує користування системою та економить час.

Таким чином, мета цієї системи полягала в тому, щоб забезпечити зручність, безпеку та простоту для звичайних користувачів. Вона повинна

відповідати сучасним стандартам і справді допомагати людям у щоденних справах.

1.2 Головні принципи та інструменти

Для реалізації функціональної, стабільної та масштабованої системи бронювання квитків було обрано сучасні технології й архітектурні підходи, що відповідають вимогам надійності, безпеки та зручності. Основу всієї структури становить принцип поділу відповідальностей між клієнтською і серверною частинами, що дозволяє досягти високої продуктивності та легкості підтримки системи в довгостроковій перспективі.

Згідно з цим принципом, серверна частина відповідає за обробку бізнес-логіки, управління даними та безпеку, тоді як клієнтська частина зосереджена на взаємодії з користувачем, забезпечуючи інтуїтивно зрозумілий та швидкий інтерфейс. Такий розподіл дозволяє кожному компоненту системи функціонувати максимально ефективно у своїй зоні відповідальності.

Для досягнення чіткої організації внутрішньої логіки, система поділена на окремі функціональні модулі: модуль обробки бронювання, модуль обробки платежів, модуль управління користувачами, модуль формування розкладів та перегляду історії, модуль аналітики та звітів.

Кожен із цих блоків реалізований незалежно, що дозволяє легко вносити зміни або додавати нові функції без ризику порушення цілісності роботи всієї системи. Такий підхід відповідає принципам гнучкої архітектури (modular loosely-coupled design), яка сьогодні є стандартом у проектуванні складних веб-застосунків.

Особливу увагу було приділено питанням інформаційної безпеки. Захист персональних даних користувачів, а також конфіденційність платіжних реквізитів, досягаються завдяки використанню сучасних криптографічних алгоритмів, протоколу TLS 1.3 для шифрування трафіку та автентифікації на базі JWT-токенів, які генеруються після входу користувача.

Фронтенд-частина реалізована за допомогою бібліотеки React, яка завдяки компонентному підходу дозволяє швидко створювати й підтримувати інтерактивний інтерфейс. Користувачі отримують миттєву реакцію системи на дії, з мінімальними затримками, що особливо важливо при бронюванні квитків у режимі реального часу. Сторінки не перезавантажуються повністю, а оновлюються частково, що значно підвищує швидкодію й комфорт використання.

Для зберігання всієї критичної інформації — користувачів, поїздок, транзакцій, історії бронювань — використано реляційну базу даних Microsoft SQL Server, яка зарекомендувала себе як потужний і надійний інструмент. У тандемі з ORM-технологією Entity Framework Core база даних інтегрується з логікою додатку, забезпечуючи швидкий доступ до інформації, збереження цілісності записів та мінімізацію помилок при виконанні запитів.

У підсумку, грамотне поєднання архітектурних принципів, перевірених технологій і сучасних засобів безпеки дозволило створити систему, що повністю відповідає очікуванням користувачів — вона працює швидко, стабільно, легко масштабовується і відповідає вимогам сучасного ринку онлайн-сервісів. Таким чином, реалізована модель може слугувати основою для подальшого розширення або адаптації під інші типи транспортних перевезень та суміжні сервіси.

1.3 Огляд програмних засобів

Щоб розробити систему бронювання, були обрані перевірені й сучасні інструменти, які дозволяють реалізувати потрібний функціонал надійно та зручно.

Фронтенд реалізовано за допомогою бібліотеки React. Вона дозволяє створювати живий, інтерактивний інтерфейс, який миттєво реагує на дії користувача без перезавантаження сторінки. Завдяки підходу з компонентами, додавати нові функції або змінювати окремі частини сайту дуже просто, що значно спрощує підтримку проєкту.

Серверну частину написано мовою C# із використанням .NET Core. Це потужна кросплатформна платформа від Microsoft, яка дає змогу створювати стабільні та масштабовані веб-додатки. У поєднанні з C#, .NET Core забезпечує високу продуктивність і надійність навіть при великому навантаженні [1].

Для зберігання інформації використовується Microsoft SQL Server — класична реляційна база даних, яка добре підходить для роботи з великими обсягами даних і дозволяє забезпечити їхню швидку обробку. Працювати з базою ми вирішили через Entity Framework — це ORM, який дозволяє взаємодіяти з таблицями як зі звичайними об'єктами, без необхідності писати складні SQL-запити вручну.

Щоб захистити особисті дані та забезпечити безпечний вхід до системи, ми використали Identity та JWT (JSON Web Tokens). Завдяки цим технологіям, кожен користувач отримує унікальний токен після входу, і цей токен перевіряється на кожному запиті до сервера. Це дозволяє уникнути постійного зберігання сесій і робить систему гнучкішою та захищеною.

Також у систему інтегровані платіжні шлюзи, які дають змогу оплачувати квитки онлайн просто й безпечно. Користувачі можуть обирати зручний спосіб оплати — це підвищує комфорт і довіру до сервісу.

Загалом, усі вибрані технології добре між собою поєднуються і дозволяють реалізувати систему, яка буде працювати стабільно, безпечно й зручно як для користувача, так і для розробника

Мова програмування C# — це мова програмування, створена компанією Microsoft у межах платформи .NET. Вона з’явилася у 2000 році й досить швидко стала популярною завдяки своїй зрозумілості, зручності у використанні та широким можливостям — як для створення звичайних програм на комп’ютер, так і для веб-додатків.

Мову спроєктували з урахуванням недоліків попередніх мов, як-от C++ чи Java, тому C# вийшла більш безпечною й простою. Вона повністю підтримує об’єктно-орієнтований підхід — тут є все: наслідування, інкапсуляція,

поліморфізм, абстракція. Завдяки цьому можна створювати зручний, структурований і добре підтримуваний код [2].

Ще одна важлива річ — це автоматичне керування пам’яттю. У C# не потрібно вручну звільняти ресурси: за це відповідає вбудований збирач сміття. Це суттєво знижує кількість помилок, пов’язаних із витоками пам’яті чи зависаннями.

Загалом, C# — це сучасний інструмент, який чудово підходить як для новачків, так і для досвідчених розробників, які хочуть писати чистий, безпечний і надійний код.

.NET Core — це потужна і гнучка платформа від Microsoft, яка дозволяє створювати різноманітні додатки: від звичайних серверних до хмарних і веб-рішень. Вона з’явилася як відповідь на потребу в сучасному, універсальному інструменті, який добре працює в будь-якому середовищі.

Одна з найбільших переваг .NET Core — це те, що вона кросплатформенна. Тобто можна розробити додаток один раз — і він буде запускатися на Windows, macOS або Linux без необхідності щось переробляти. Це дуже зручно, особливо якщо проєкт планується використовувати на різних типах серверів або середовищах.

Ще один плюс — висока продуктивність. Завдяки продуманому середовищу виконання, використанню JIT-компіляції, ефективному керуванню пам’яттю та підтримці багатопотоковості, програми на .NET Core працюють швидко й стабільно навіть при високому навантаженні. Це дуже важливо для тих, хто розробляє комерційні чи масштабні сервіси.

Модульна архітектура — ще один момент, який значно спрощує роботу. Розробник може підключити лише ті бібліотеки, які справді потрібні, і не тягнути за собою зайве. Завдяки NuGet, можна легко додавати нові функції або сторонні інструменти — і це буквально в кілька кліків.

.NET Core також ідеально підходить для роботи з хмарою. Платформа добре інтегрується з Azure, AWS, Google Cloud, що дозволяє без проблем розгортати додатки, масштабувати їх, налаштовувати безперервну інтеграцію та

оновлення (CI/CD). Це дає розробникам справді гнучкий і зручний процес доставки продукту [3].

Не забули й про безпеку. Тут є все необхідне для захисту: аутентифікація, авторизація, шифрування, захист від SQL-ін'єкцій і XSS. Тобто платформа дає всі базові засоби, щоб не «вигадувати велосипед» і бути впевненим у надійності системи.

Ще один великий плюс — підтримка популярних мов програмування, зокрема C#, який на цій платформі розкривається на повну. Завдяки постійним оновленням і розвитку, розробники мають доступ до нових можливостей і не відстають від трендів.

У результаті, .NET Core — це платформа, яка підходить майже для будь-якого проєкту: вона сучасна, швидка, зручна, безпечна й постійно оновлюється. Саме тому її обирають як індивідуальні розробники, так і великі компанії по всьому світу.

React — це популярна JavaScript-бібліотека, створена Facebook ще у 2013 році. Вона одразу стала улюбленцем серед розробників завдяки тому, що значно спрощує створення сучасних веб-інтерфейсів: швидких, динамічних і зручних у користуванні.

Сьогодні, коли більшість сервісів переходить в онлайн і користувачі очікують швидкої реакції на будь-які дії, важливо мати інструмент, який дозволяє реалізувати це без зайвого клопоту. React саме таким і є.

Одна з головних фішок React — це компонентний підхід. Тобто інтерфейс збирається з окремих незалежних блоків (компонентів), кожен з яких відповідає за якусь конкретну частину сторінки. Кожен компонент має власну логіку й стан, і це дозволяє легко створювати складні й масштабовані додатки, де все чітко структуровано.

React також використовує односторонній потік даних, тобто дані передаються згори вниз — від "батьківського" компонента до "дочірнього". Це робить логіку додатка зрозумілішою й легшою для відлагодження, а значить —

і для підтримки. Завдяки цьому, навіть великі проєкти залишаються керованими [4].

До того ж, React має величезну спільноту та велику кількість бібліотек-доповнень, що дозволяє розширювати можливості майже без обмежень. Усе це робить бібліотеку практичним вибором як для новачків, так і для професіоналів.

Entity Framework — це справжній порятунок для .NET-розробника, коли мова заходить про роботу з базами даних. Його створила Microsoft саме для того, щоб звільнити розробників від необхідності вручну писати купу SQL-запитів. Замість цього можна просто працювати з об’єктами, як зазвичай, — і все працює.

Суть у тому, що EF виступає посередником між додатком і базою даних. Розробник пише звичайний C#-код, працює з класами, які представляють таблиці, а EF уже сам перетворює ці дії у відповідні SQL-запити. Це справді зручно — особливо коли проєкт великий і має багато таблиць.

Для прикладу, замість того, щоб писати складний SQL для вибірки даних із фільтрацією, можна просто написати звичний метод у C#, використовуючи LINQ — мову запитів, вбудовану в .NET. Виглядає зрозуміло, читається легко, і що головне — менше шансів зробити помилку.

Ще один важливий момент — гнучкість у підходах. Хочеш повний контроль — використовуй Code First і моделюй базу прямо з коду. Маєш уже готову базу — підійде Database First. А якщо зручно спочатку намалювати модель — є варіант з Model First. У кожного підходу є свої плюси, і EF дає можливість обрати той, що підходить саме тобі [5].

Коли структура бази змінюється (а це буває часто), в пригоді стане система міграцій. Вона дозволяє плавно оновлювати базу, не боячись, що щось зламається або зітреться. До того ж, можна контролювати історію змін, як у Git. Що до безпеки, то тут усе продумано: EF автоматично захищає від SQL-ін’єкцій, використовуючи параметризовані запити. Це один із найважливіших моментів, особливо коли працюєш із персональними даними користувачів чи фінансовою інформацією.

Ще один плюс — продуктивність. EF має механізми кешування та оптимізації запитів, які значно пришвидшують роботу, навіть якщо даних багато. Це дозволяє не турбуватись, що система почне «гальмувати» при зростанні навантаження.

І що приємно — підтримується робота не лише з Microsoft SQL Server, а й з іншими СУБД: MySQL, PostgreSQL, SQLite, навіть Oracle. Тобто EF — це не тільки зручно, а ще й універсально.

Загалом, Entity Framework — це потужний інструмент, який бере на себе багато «нудної» роботи й дозволяє розробникам сконцентруватися на справжній логіці додатка. І це саме той випадок, коли автоматизація справді спрощує життя.

Identity та JWT. У сучасному вебі безпека — це вже не просто бажаний плюс, а обов'язкова вимога. Саме тому в розробці веб-застосунків усе частіше використовують Microsoft Identity разом із JSON Web Tokens (JWT). Обидві технології прекрасно взаємодіють і дають змогу реалізувати надійну аутентифікацію й авторизацію без зайвого клопоту.

Microsoft Identity — це зручна система, яку створила Microsoft для .NET-платформи. Вона покриває всі базові сценарії, які можуть знадобитися розробнику: реєстрація, вхід, керування паролями, ролями, обмеження доступу тощо. Причому працює як зі звичайними логінами й паролями, так і з соцмережами (Google, Facebook), або з зовнішніми сервісами, що використовують OAuth2 чи OpenID Connect.

Одна з сильних сторін Identity — гнучкість. Тобто ти можеш налаштувати політики доступу, вимоги до складності пароля, затримки при неправильних спробах входу, двофакторну перевірку — загалом усе, що тільки може знадобитися для надійного захисту. До того ж, все це добре інтегрується з іншими елементами .NET, тож налаштування не виглядає як окрема система — усе працює в комплексі [6].

А що стосується JWT, то це спосіб передавати дані про користувача у вигляді спеціального токена. Коли користувач проходить авторизацію, сервер видає йому токен, який той зберігає на клієнті (наприклад, у локальному сховищі

браузера). Надалі, коли клієнт звертається до сервера, він просто прикріплює цей токен до запиту — і сервер перевіряє, чи все з ним гаразд.

JWT складається з трьох частин — заголовка, навантаження і підпису. У навантаженні можуть бути, наприклад, дані про ім’я користувача, його роль, час дії токена та інша службова інформація. Все це підписується секретом, щоб ніхто не міг підробити токен.

Цей підхід має низку переваг: по-перше, не потрібно зберігати сесії на сервері, що зменшує навантаження. По-друге, система працює швидко й зручно — підходить навіть для мікросервісних архітектур, де сервіси можуть самостійно перевіряти дійсність токена, не звертаючись до спільного сховища.

Разом Microsoft Identity та JWT утворюють надійну пару. Один компонент займається управлінням користувачами, другий — передачею прав доступу. Такий підхід дозволяє легко налаштувати рольову модель, політики доступу або ж навіть авторизацію на основі claims (тобто конкретних властивостей користувача, як-от група, країна, статус тощо).

Загалом, якщо стоїть завдання зробити безпечний .NET-додаток — то цей дует можна вважати перевіреною основою. Він добре масштабується, адаптується під конкретні вимоги і дозволяє зробити додаток максимально надійним.

1.4 Висновок

У цьому розділі було з’ясовано, що сучасні сервіси для бронювання квитків повинні бути не лише зручними у використанні, а й швидкими, зрозумілими та безпечними. Люди хочуть мати можливість знайти квиток, вибрати місце, оплатити замовлення онлайн і зробити все це без зайвих складнощів. Саме на такі потреби й орієнтована система, яку планується створити.

Було визначено основні завдання, які має вирішувати ця система: надання актуальної інформації про рейси, зручний пошук, вибір місця, підтримка онлайн-оплати та захист особистих даних користувача. Крім цього, підкреслюється

важливість збереження історії бронювань, що робить сервіс ще зручнішим для тих, хто подорожує часто.

У процесі аналізу також обґрунтовано вибір технологій, на яких буде побудована система. Вирішено реалізувати рішення на основі клієнт-серверної моделі з використанням React, .NET Core, Microsoft SQL Server та Entity Framework. Для захисту даних та безпечного входу будуть використані технології JWT та Identity.

Загалом, у цьому розділі закладено чітке розуміння того, що саме має робити система, якою вона повинна бути зсередини й ззовні, та які інструменти потрібні для її реалізації. Усе це стане основою для подальшої розробки.

2 ПРОЕКТУВАННЯ ТА ТЕХНОЛОГІЇ РОЗРОБКИ СИСТЕМИ

2.1 Організація обміну даними між клієнтом і сервером

У сучасному світі цифрових технологій, де сервіси працюють 24/7, і користувач очікує швидкої відповіді буквально за секунду, архітектура програмного забезпечення має надзвичайно велике значення. Від того, як побудована система зсередини, залежить дуже багато — швидкість, надійність, безпека, масштабованість, і навіть зручність для розробника. Саме тому для реалізації системи бронювання квитків було обрано клієнт-серверну архітектуру — рішення, яке вже давно стало стандартом де-факто в розробці сучасних веб-додатків.

Цей підхід передбачає чіткий поділ на дві частини: клієнтську (frontend) і серверну (backend). Уявімо це як команду — у якій кожен виконує свою роль. Клієнтська частина — це все, з чим безпосередньо працює користувач: кнопки, форми, списки, фільтри, підказки, тобто все те, що він бачить на екрані. У нашому випадку — це React-додаток, який працює в браузері або на смартфоні. Цей інтерфейс дозволяє взаємодіяти з системою, але сам по собі не обробляє дані, а лише формує запити.

У той час як клієнт “питає” — сервер “відповідає”. Серверна частина працює у фоновому режимі й бере на себе обробку всіх запитів, перевірку прав доступу, запис і читання з бази даних, обчислення, логіку, перевірку бізнес-правил. Якщо клієнт — це вітрина магазину, то сервер — це склад, бухгалтерія, логістика й охорона одночасно.

Один із головних плюсів такого підходу — розподіл навантаження. Клієнт не перевантажується обробкою великої кількості даних. Замість цього, він просто передає дані на сервер, отримує відповідь і оновлює інтерфейс. Це дозволяє навіть слабким пристроям — старим ноутбукам, бюджетним телефонам — працювати з системою без проблем. Водночас серверна частина може масштабуватись незалежно: наприклад, у разі зростання кількості користувачів

просто додаються додаткові сервери або розподіляється навантаження між кількома інстансами.

Клієнт-серверна архітектура також дозволяє досягти високого рівня безпеки. Дані користувача, наприклад — інформація про квитки, особисті дані або платіжні реквізити — ніколи не зберігаються на клієнтському боці. Вони весь час перебувають на захищеному сервері, доступ до якого має лише авторизований користувач, і лише в рамках своїх прав. Серверна частина відповідає за перевірку токенів, логіку доступу, перевірку ролей, шифрування та інші аспекти безпеки.

Ще один важливий аспект — масштабованість і адаптивність. Якщо з часом з’явиться потреба додати нові функції — наприклад, бронювання автобусних квитків чи авіаквитків — це можна реалізувати без повної перебудови системи. Часто достатньо внести зміни в окремий модуль. Це можливо завдяки модульному принципу клієнт-серверної архітектури, яка дозволяє оновлювати одну частину системи, не зачіпаючи інші.

З точки зору розробки, цей підхід також має великий плюс — можливість паралельної роботи над різними частинами. Наприклад, одна команда може займатись дизайном та верстkою інтерфейсу, інша — створювати API та обробку даних, а третя — адмініструвати базу. Це прискорює весь процес і робить його більш організованим.

Не менш важливо й те, що клієнт-серверна архітектура добре працює з хмарними технологіями. Сервери можна розміщувати в AWS, Azure чи будь-якому іншому хмарному провайдері, отримуючи доступ до сучасних засобів балансування навантаження, моніторингу, CI/CD, автоматичного масштабування тощо.

Загалом, клієнт-серверна модель — це не просто вибір “за звичкою”, а добре продумане архітектурне рішення, яке дає можливість досягти балансу між швидкістю, безпекою, зручністю та гнучкістю. Для системи бронювання квитків, де важливо одночасно обробляти запити від багатьох користувачів, гарантувати

точність даних, забезпечити зручність інтерфейсу та захист особистої інформації — це рішення виглядає більш ніж виправданим.

Клієнтська частина. Коли ми говоримо про будь-яку сучасну веб-систему, перше, з чим стикається користувач, — це інтерфейс. Саме з нього починається знайомство з додатком, і від того, наскільки швидко й зручно усе працює, часто залежить загальне враження від усього проєкту. У нашому випадку клієнтська частина системи реалізована за допомогою бібліотеки React — одного з найпопулярніших інструментів у світі фронтенд-розробки. Вона не просто задає “обличчя” системи, а фактично є її представником у руках користувача.

React став вибором номер один не просто так — він дозволяє створювати живі, інтерактивні сторінки, які оновлюються прямо під час роботи, без необхідності перезавантаження браузера. Коли користувач вводить дані, натискає кнопку чи обирає щось із випадаючого списку — сторінка реагує миттєво. Усе це забезпечується завдяки віртуальному DOM, який оновлюється дуже ефективно, не перезавантажуючи всю сторінку кожного разу.

Структурно клієнтська частина складається з кількох основних рівнів:

Інтерфейс користувача (UI), який включає всі візуальні елементи: кнопки, поля вводу, меню, таблиці, форми, повідомлення, індикатори — усе, що бачить і чим користується людина.

Логіка інтерфейсу, що обробляє події: кліки, вводи, перемикання вкладок тощо.

Засоби обміну даними із сервером, які дозволяють взаємодіяти з бекендом.

Завдяки компонентному підходу React, кожен шматочок інтерфейсу — це окремий компонент, який можна розробити, протестувати й при потребі повторно використати в інших місцях системи. Це дуже зручно: один раз створив, скажімо, кнопку з певною логікою — і потім просто вставляєш її в інші частини проєкту.

Щодо обміну даними — тут використовуються сучасні підходи: Fetch API, Axios, асинхронні запити (AJAX). Це дозволяє отримувати чи відправляти

інформацію на сервер без перезавантаження сторінки. Наприклад, користувач натискає “Пошук квитків”, і вже за мить бачить результат — і все це відбувається фоново, без жодних "білих екранів".

Такий підхід значно підвищує швидкодію та робить користувацький досвід більш плавним, “реальним” — як ніби ти працюєш з повноцінною програмою, а не просто з сайтом. React чудово справляється із завданням підтримки стану додатка, коли потрібно синхронізувати дії користувача, відображення інтерфейсу та відповіді сервера.

Ще одна величезна перевага — гнучкість у підтримці й масштабуванні. Якщо треба змінити дизайн, додати новий модуль чи внести оновлення — це можна зробити швидко, не переписуючи весь інтерфейс із нуля. Особливо це цінно в командній розробці, коли різні частини додатку створюються різними людьми.

Крім того, React має активну спільноту, багатий набір бібліотек і безліч готових рішень для будь-якої задачі — від маршрутизації до управління глобальним станом.

Отже, клієнтська частина нашої системи — це не просто набір кнопок і форм, а повноцінна, гнучка та сучасна оболонка, яка забезпечує швидку й комфортну взаємодію з користувачем. І саме React дозволяє зробити це легко, ефективно та красиво.

Серверна частина. Серверна частина відіграє ключову роль у роботі всієї системи. Саме тут обробляються всі запити, виконується основна логіка, зберігаються й оновлюються дані. Для реалізації цієї частини було використано платформу .NET Core і мову програмування C#. Таке поєднання обране через його стабільність, швидкодію та широкі можливості для побудови веб-сервісів.

Запити, які надсилає клієнт, обробляються сервером за допомогою фреймворку ASP.NET Core. Він дозволяє ефективно приймати запити, опрацьовувати їх, звертатися до бази даних та формувати відповіді. Завдяки своїй гнучкій структурі та здатності витримувати велике навантаження, ASP.NET Core підходить для систем, де одночасно працює багато користувачів.

На рисунку 2.1 показано сценарій взаємодії користувача з серверною частиною під час пошуку та бронювання квитків.

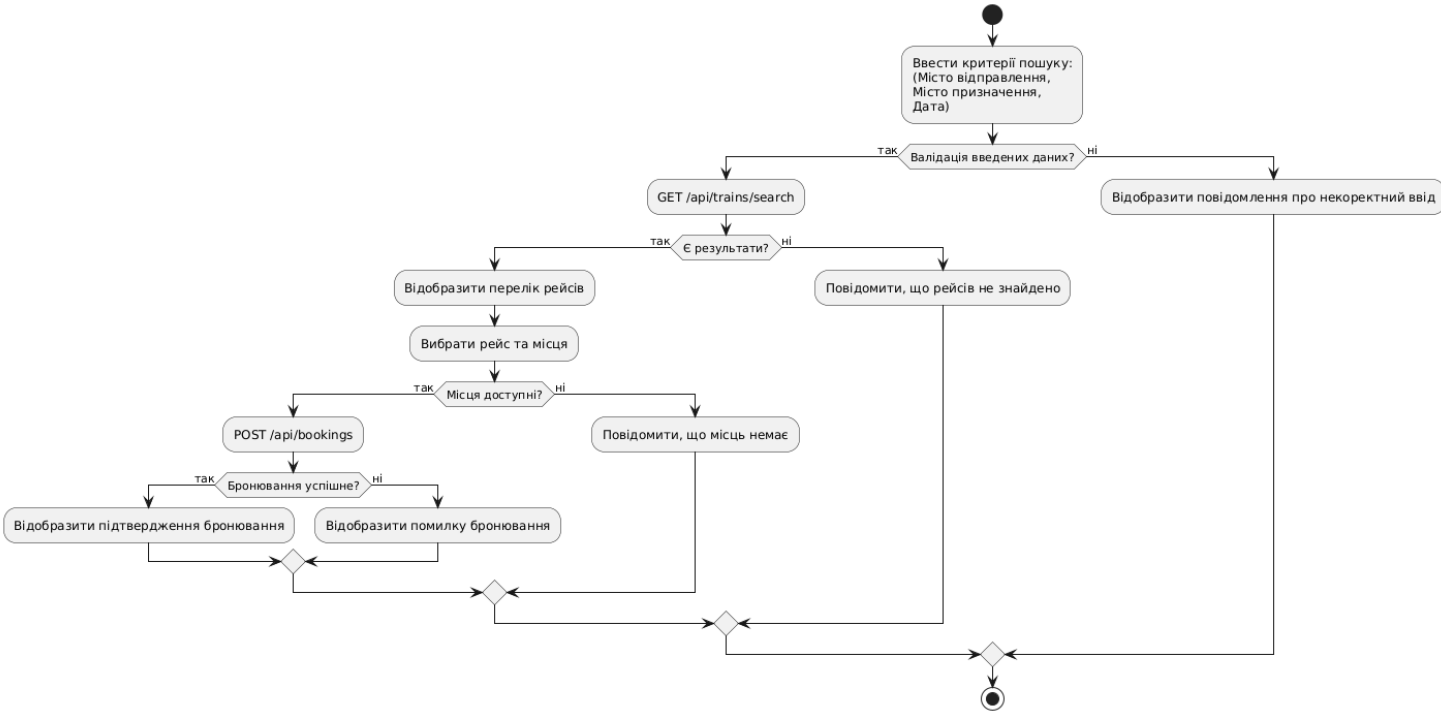


Рисунок 2.1 – Діаграма діяльності для пошуку рейсу та бронювання

Діаграма демонструє процес взаємодії, починаючи з введення параметрів пошуку до успішного бронювання або виведення помилок, що допомагає виявити потенційні проблемні точки та оптимізувати взаємодію.

Вся логіка, яка описує, як саме система повинна поводитися в тій чи іншій ситуації, зосереджена в коді, написаному на C#. Тут обробляється все — від створення бронювання до перевірки прав доступу користувача. Важливо, що цю частину можна легко змінювати й доповнювати в майбутньому без необхідності повністю переписувати систему.

Для зберігання та роботи з даними використовується Entity Framework — технологія, яка дозволяє працювати з базою не через складні SQL-запити, а у звичному для C# середовищі. Це значно спрощує розробку й зменшує кількість

помилок. Окрім цього, EF підтримує так звані “міграції”, завдяки яким можна змінювати структуру бази без втрати вже збережених даних.

Таким чином, серверна частина забезпечує зв’язок між інтерфейсом користувача і даними, що зберігаються у системі. Вона відповідає за правильність роботи всієї логіки, стабільність взаємодії і цілісність даних. Завдяки сучасному технічному рішенню, розробка системи стала гнучкою, а сама система — надійною і готовою до подальшого розвитку. [7].

2.2 Модульна організація системи

Однією з ключових вимог до створення інформаційної системи бронювання квитків є чітка і зрозуміла модульна архітектура. Такий підхід дозволяє структурувати програму на незалежні компоненти, кожен з яких виконує окреме функціональне завдання. Це забезпечує легкість масштабування, спрощує процеси супроводу та оновлення коду, дозволяючи швидко адаптуватися до нових бізнес-вимог та технологічних змін. Завдяки модульності також стає значно простіше розподіляти задачі між командами розробників, що оптимізує роботу і скорочує терміни реалізації проекту.

Модуль бронювання є одним із центральних компонентів системи. Він відповідає за обробку заявок користувачів на резервування місць у транспортних засобах. Процес роботи цього модуля включає кілька основних етапів. Спочатку виконується перевірка наявності вільних місць у вибраному користувачем транспорті. Після цього здійснюється розрахунок вартості бронювання, де враховуються такі фактори, як клас вагона, відстань поїздки та додаткові послуги (наприклад, постіль, харчування або додатковий багаж). Потім дані про бронювання надійно зберігаються у базі даних, забезпечуючи можливість подальшого перегляду та редагування інформації про замовлення. Завдяки використанню оптимізованих алгоритмів та продуманій логіці, цей модуль дозволяє уникнути дублювання записів і забезпечує швидку обробку навіть у періоди пікових навантажень.

Важливим є також модуль оплати, який інтегрований з провідними платіжними системами (PayPal, Stripe, LiqPay тощо). Цей компонент відповідає за виконання фінансових транзакцій, включаючи валідацію реквізитів та ініціювання списання коштів. Для гарантії безпеки фінансових операцій реалізовано шифрування та токенізацію даних, які відповідають сучасним стандартам PCI DSS, що захищає користувачів від можливих витоків інформації.

Наступний важливий компонент – модуль управління користувачами, який відповідає за реєстрацію нових акаунтів, авторизацію існуючих користувачів, відновлення паролів та оновлення персональних даних. У системі реалізована гнучка модель призначення ролей (пасажир, оператор, адміністратор), що дозволяє ефективно управляти рівнями доступу та правами користувачів. Це забезпечує додатковий рівень безпеки і зручність використання системи.

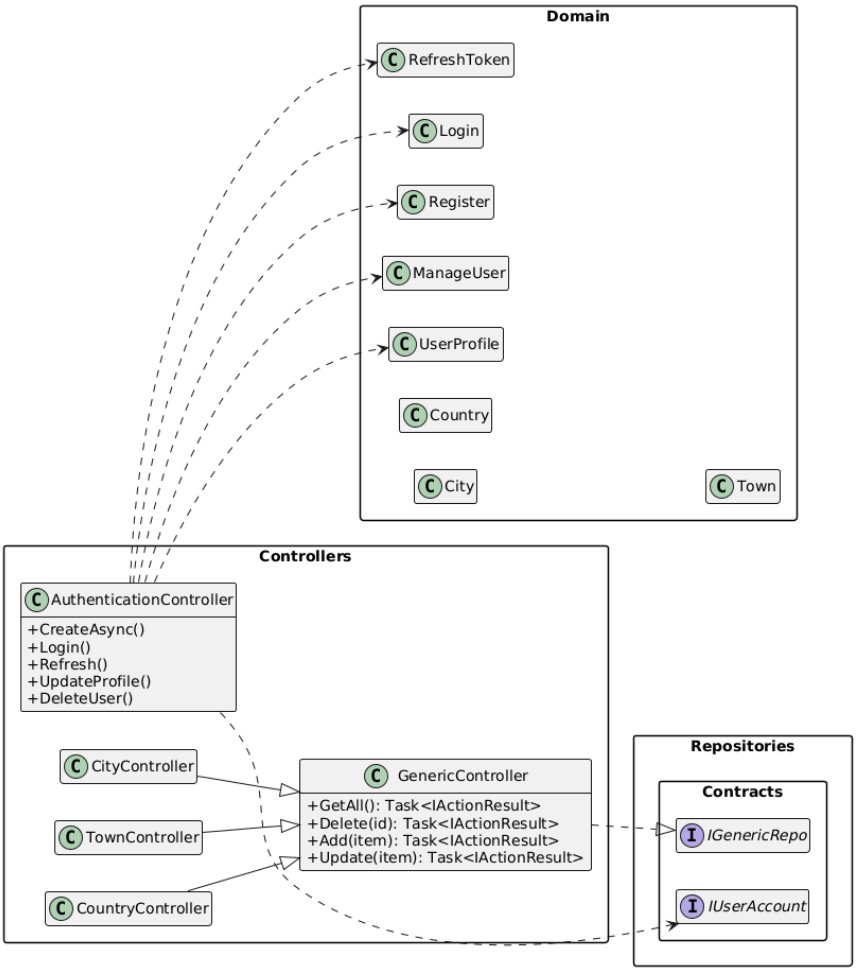


Рисунок 2.2 – Класова діаграма модуля бронювання квитків

На рисунку 2.2 наведено класову діаграму модуля бронювання, яка наочно демонструє структуру ключових класів, таких як AuthenticationController, GenericController, а також зв'язок з доменними класами (UserProfile, Login, Register) та репозиторіями. Ця діаграма полегшує розуміння архітектури, робить взаємозв'язки між компонентами очевидними для розробників, сприяючи ефективній командній взаємодії та легкості майбутніх оновлень.

Модуль розкладів відіграє важливу роль у забезпеченні актуальності інформації про графіки руху потягів та автобусів. Оператори системи можуть оперативно додавати спеціальні рейси, змінювати час відправлення або скасовувати рейси з технічних причин. Всі зміни негайно синхронізуються з модулем бронювання, що дозволяє користувачам бачити тільки актуальні дані та уникати ситуацій з продажем неіснуючих місць.

Окремо варто зазначити важливість модуля звітності, який формує аналітичні звіти про ефективність роботи системи. Цей модуль агрегує інформацію про обсяг продажів, завантаженість маршрутів, ефективність рекламних кампаній та зміну попиту у різні періоди року. Створені звіти можуть бути експортовані в популярні формати (наприклад, PDF, Excel), що дозволяє керівництву компанії оперативно приймати управлінські рішення на основі чітких і зрозумілих даних.

Додатково до функціональних модулів також розроблені детальні сценарії роботи користувача з системою, зокрема процес реєстрації нового користувача та авторизації вже існуючого. Ці сценарії описують послідовність кроків, які виконує користувач, та логіку обробки інформації системою, забезпечуючи інтуїтивну зрозумілість процесів. На рисунку 2.3 наведена діаграма реєстрації користувача, яка демонструє перевірку введених даних, унікальність електронної пошти та подальше створення акаунта.

Рисунок 2.3 – діаграма реєстрації

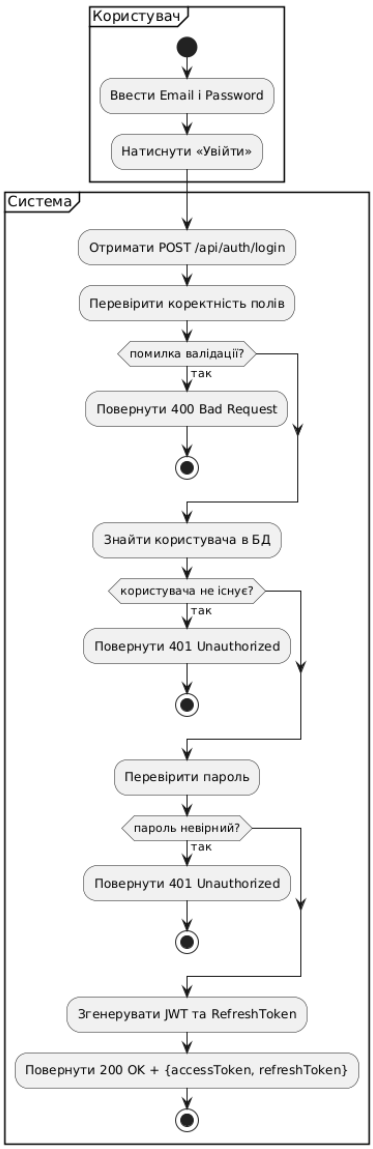
На рисунку 2.4 зображено послідовність дій під час авторизації користувача в системі. Процес починається з того, що користувач вводить свою електронну адресу та пароль, після чого натискає кнопку "Увійти". Система приймає POST-запит на /api/auth/login.

Далі відбувається перевірка коректності введених даних. Якщо дані не відповідають вимогам валідації (наприклад, порожні поля або неправильний формат email), система повертає відповідь 400 Bad Request, що сигналізує про помилку в запиті.

У разі успішної валідації система намагається знайти користувача в базі даних. Якщо запис не знайдено — повертається відповідь 401 Unauthorized, що означає відмову в доступі через неправильні облікові дані.

Далі перевіряється відповідність пароля. У разі невірного пароля знову повертається 401 Unauthorized. Якщо ж пароль правильний, система переходить до генерації JWT та RefreshToken, які дозволяють користувачу отримати авторизований доступ до захищених ресурсів системи.

Завершується процес поверненням відповіді 200 OK, у тілі якої передаються accessToken та refreshToken — ключові компоненти механізму авторизації.



рисунку 2.4 – діаграма авторизації

Таким чином, детально продумана і реалізована модульна організація системи дозволяє команді розробників ефективно підтримувати та розширювати функціональні можливості сервісу бронювання квитків. Вона забезпечує високий рівень стабільності, зручності і безпеки користувачів, а також значно спрощує процеси підтримки та супроводу системи у майбутньому.

2.3 Об'єктно-орієнтоване програмування

Об’єктно-орієнтований підхід (ООП) був обраний як основа програмної реалізації системи бронювання через його ефективність, перевірену десятиліттями досвіду розробки складних та масштабованих інформаційних систем. Завдяки своїм базовим принципам — інкапсуляції, наслідуванню, поліморфізму та абстракції, — ООП дозволяє створювати код, що легко читається, змінюється, підтримується та масштабовується.

Інкапсуляція, як перший і один з найважливіших принципів ООП, дозволяє приховати внутрішню реалізацію об'єктів від зовнішнього впливу. У рамках цього проєкту вона забезпечує контроль над тим, які саме поля та методи доступні ззовні класу, що, у свою чергу, захищає дані від некоректного використання. Наприклад, усі змінні, які відповідають за стан квитка, пасажира чи транзакції, ізольовано в межах класів, і доступ до них надається тільки через визначені методи. Це значно знижує ризик помилок та полегшує діагностику під час тестування.

Наслідування використано для усунення дублювання коду і створення гнучкої ієрархії об'єктів. Так, було створено базовий клас Ticket, який містить спільні характеристики для всіх типів квитків, зокрема ідентифікатор, дату випуску та базову вартість. Від нього успадковуються класи TrainTicket та BusTicket, що додають специфічні поля — наприклад, номер поїзда, вагон або автобусний маршрут. Завдяки цьому вдалося уникнути повторення логіки в кількох місцях і зосередити загальну поведінку в одному класі.

Поліморфізм, у свою чергу, дозволив реалізувати універсальний підхід до взаємодії з об’єктами, які реалізують однакові інтерфейси. Наприклад, сервіс друку квитків може працювати з будь-яким типом квитка, викликаючи метод Print() незалежно від того, чи це квиток на поїзд, автобус або інший вид транспорту. Це значно спрощує додавання нових типів квитків у майбутньому, адже не потрібно змінювати вже існуючий код — достатньо реалізувати потрібний інтерфейс у новому класі.

Абстракція забезпечила можливість проєктувати код навколо логічних сутностей, таких як Квиток, Користувач, Рейс, не зосереджуючись на деталях реалізації. Це дозволило побудувати систему, де об’єкти відображають реальні бізнес-процеси, що, своєю чергою, полегшує комунікацію між технічними фахівцями та замовниками, а також спрощує підтримку та масштабування проєкту.

Крім того, в розробці дотримувалися принципів S.O.L.I.D., що є основою якісного об'єктно-орієнтованого дизайну. Зокрема, реалізовано розподіл відповідальностей між класами, використано інтерфейси для досягнення слабкого зв’язування та забезпечено легкість внесення змін без порушення цілісності коду. Наприклад, доступ до бази даних реалізовано через шаблон Repository, що дозволяє легко замінити джерело даних без зміни бізнес-логіки. Також застосовано шаблон Factory, який спрощує створення об’єктів з урахуванням їхніх залежностей.

Таким чином, застосування ООП стало основою для створення системи, що легко підтримується, адаптується до нових вимог та забезпечує чисту архітектуру коду. Це дозволяє команді розробників впевнено розвивати продукт у майбутньому, зберігаючи високу якість і стабільність програмного коду навіть при зміні бізнес-логіки.

2.4 Методи забезпечення безпеки

У контексті розробки сучасної веб-системи бронювання особливої уваги потребує питання захисту інформації, адже система працює з персональними,

контактними та платіжними даними користувачів. Надійність збереження таких даних є критично важливою для забезпечення довіри до сервісу, а також для дотримання чинного законодавства в галузі інформаційної безпеки.

Захист інформації реалізовано багаторівнево, із використанням перевірених криптографічних алгоритмів, сучасних методів аутентифікації та авторизації, а також технік превентивного захисту від типових загроз, описаних у рейтингу OWASP Top Ten.

По-перше, усі дані шифруються як під час передавання, так і в режимі збереження («у дорозі» та «у спокої»). Передача інформації між клієнтом і сервером здійснюється по захищеному каналу, за допомогою протоколу TLS версії 1.3, що забезпечує високий рівень стійкості до перехоплення. Крім того, у самій базі даних критичні поля — такі як паролі, токени або платіжні ідентифікатори — шифруються за допомогою симетричних алгоритмів, ключі до яких зберігаються у зашифрованому сховищі на сервері. Це дозволяє навіть у випадку несанкціонованого доступу до БД запобігти розшифруванню даних зловмисником.

По-друге, реалізовано сучасну систему автентифікації та авторизації. Для цього використовується комбінація ASP.NET Core Identity (яка забезпечує базову аутентифікацію користувача, хешування паролів та контроль політик безпеки) та JWT (JSON Web Tokens), яка дає змогу працювати з сесіями без збереження стану на сервері. Токени містять claims — службові дані про користувача, підписані на сервері, — і передаються у заголовку Authorization [8]. Усі запити до захищених маршрутів перевіряються middleware-компонентом, який гарантує, що користувач має відповідні права доступу. Такий підхід є сучасним стандартом в архітектурах безсерверних та масштабованих вебсервісів.

По-третє, система реалізує повний набір заходів для протидії найпоширенішим вразливостям. Зокрема:

Всі SQL-запити виконуються через ORM Entity Framework Core, що гарантує параметризацію запитів і виключає можливість SQL-ін’єкцій. Усі вхідні дані проходять попередню HTML-екранізацію, що запобігає XSS-атакам

(вставлення шкідливих скриптів у форму або поле). У політиках безпеки використовується заголовок Content-Security-Policy, який забороняє завантаження або виконання скриптів із сторонніх джерел. У формах застосовується механізм CSRF-токенів, що захищає від підроблених POST-запитів при активній сесії користувача. Крім цього, адміністративна частина системи має обмежений доступ лише для авторизованих осіб, і всі критичні дії супроводжуються повторною перевіркою ролі або токена доступу.

У результаті, розроблена модель безпеки базується на принципах CIA-трикутника — конфіденційності, цілісності та доступності. Це дозволяє мінімізувати площу атаки, забезпечити високий рівень стійкості до загроз, а також підтримувати відповідність галузевим стандартам і вимогам GDPR/ISO/OWASP. Такий підхід робить систему безпечною як для користувачів, так і для власників платформи, захищаючи її від як зовнішніх, так і внутрішніх ризиків.

2.5 Реактивне програмування

Для того щоб інтерфейс системи реагував на дії користувача буквально «миттєво», у фронтенді застосовано підхід reactive programming. Його сутність полягає у тому, що будь-які зміни стану -— чи то нові дані з сервера, чи дії користувача на формі — розглядаються як безперервні потоки подій. Замість періодично опитувати сервер або вручну «проштовхувати» оновлення крізь ланцюжок компонентів, програма просто «підписується» на потрібні потоки, і коли в них з’являється нове значення, усе, що пов’язане з цим значенням, оновлюється автоматично. Завдяки такій моделі дані і UI залишаються синхронізованими без надлишкового коду, а затримка між появою події та її відображенням у браузері не відчувається.

У реалізації використано бібліотеку RxJS разом із React. RxJS надає абстракцію Observable, тобто спостережуваного потоку, а також багатий набір операторів для фільтрації, перетворення та комбінування цих потоків. Наприклад, ланцюжок from(fetch(...)).pipe(map(res=>res.json()),

debounceTime(300)) дозволяє за один рядок отримати відповідь сервера, перетворити її у потрібний формат і приглушити «зайві» події, якщо користувач дуже швидко змінює параметри пошуку. Компоненти React же, використовуючи хук useObservable або кастомні прив’язки на кшталт useEffect(() => subscription, []), підписуються на потрібні потоки й автоматично перемальовуються, коли приходять нові значення [9].

Практичний ефект від реактивного підходу відчутний одразу: сторінка не «зависає» під час мережевих викликів, автодоповнення маршрутів спрацьовує без видимих затримок, а індикатори завантаження зникають саме в ту мить, коли дані реально готові. До того ж значно спрощується підтримка коду: логіка асинхронного обміну зосереджена в одному місці, немає розгалужених «дерев» з then/catch чи складних схем зі спільними змінними стану.

Під кутом зору архітектури важливо, що реактивна модель добре масштабується. Коли кількість джерел подій зростає — наприклад, додаються push-повідомлення або веб-сокети для оновлення розкладу в реальному часі — треба лише створити ще один Observable і «підміксувати» його через оператор merge, не переписуючи решту логіки. Саме тому сучасні фронтенд-фреймворки (Angular, React у зв’язці з RxJS, Vue 3 з Vue-Rx) дедалі частіше рекомендують такий спосіб роботи з асинхронщиною.

2.6 Проектуваннія бази даних

При проектуванні бази даних було враховано потребу в ефективності, гнучкості та масштабованості системи бронювання квитків. Для цього базу поділено на три основні групи таблиць: користувацьку (User Tables), автобусну (Bus Tables) та залізничну (Train Tables).

На рисунку 3.1 представлена логічна структура бази даних, яка відповідає за зберігання інформації про користувачів, їхню рольову приналежність, взаємодію з системою, історію покупок та залишені відгуки. Цей фрагмент моделі є центральним для всієї інформаційної системи, адже саме він забезпечує основну взаємодію користувача з програмним продуктом. Грамотно організовані

зв’язки між таблицями дозволяють не лише зберігати ключову інформацію, а й забезпечувати безперебійну обробку запитів, аутентифікацію, авторизацію та управління правами доступу.

Центральне місце займає таблиця AspNetUsers, яка виконує роль головного сховища особистих і службових відомостей користувача. Вона містить базову інформацію: ім’я, прізвище, електронну пошту, номер телефону, а також технічні поля, зокрема: гешований пароль, статус блокування, лічильник помилок при вході, параметри двофакторної автентифікації, маркери безпеки тощо. Такий рівень деталізації дозволяє не лише здійснювати авторизацію, а й підтримувати високий рівень захисту даних.

Для реалізації системи ролей у проєкті використовується зв’язка таблиць AspNetRoles та AspNetUserRoles. Перша з них містить перелік доступних ролей (наприклад: «Користувач», «Оператор», «Адміністратор»), а друга — є проміжною таблицею, яка реалізує зв’язок типу «багато-до-багатьох» між користувачами і їхніми ролями. Це дозволяє гнучко призначати права доступу до функціоналу системи, а також створювати кастомні політики безпеки.

Розширення функціоналу забезпечується за допомогою таблиць AspNetUserClaims і AspNetRoleClaims, у яких можна задавати додаткові атрибути для конкретного користувача або цілої ролі. Наприклад, можна визначити, чи має користувач право на перегляд аналітики, або доступ до модулів адміністрування.

Особливу увагу приділено можливості входу через сторонні сервіси, такі як Google чи Facebook. Для цього передбачено таблицю AspNetUserLogins, яка фіксує джерело входу, ключі зовнішніх провайдерів і пов’язує ці дані з відповідним обліковим записом у системі. Таким чином, користувач може входити як через стандартну форму авторизації, так і через OAuth2-автентифікацію.

Для безпечного керування життєвим циклом сесій використовується таблиця AspNetUserTokens, в якій зберігаються токени доступу — зокрема, JWT

або refresh-токени. Це дозволяє зберігати користувача в системі навіть після закриття браузера та забезпечити безпечне відновлення сеансу.

Історія взаємодії користувача з сервісом фіксується через таблиці TrainPurchaseHistory та BusPurchaseHistory. У цих таблицях зберігається інформація про всі здійснені бронювання, включаючи номер замовлення, дату покупки, ідентифікатор рейсу та його вартість. Це дозволяє користувачеві переглядати свої минулі замовлення, повторно їх оформлювати, а також формувати особисту історію подорожей.

Нарешті, таблиця Feedbacks забезпечує механізм зворотного зв’язку між користувачами та системою. Вона дозволяє залишати коментарі, виставляти рейтинг та надавати відгуки про поїздки, що є важливою складовою покращення сервісу та побудови довіри між компанією й пасажирами.

Таким чином, структура таблиць, представлена на рисунку 3.1, утворює повноцінний модуль керування користувачами. Її гнучкість, розширюваність і відповідність вимогам безпеки робить її надійним фундаментом для масштабованого сервісу онлайн-бронювання..

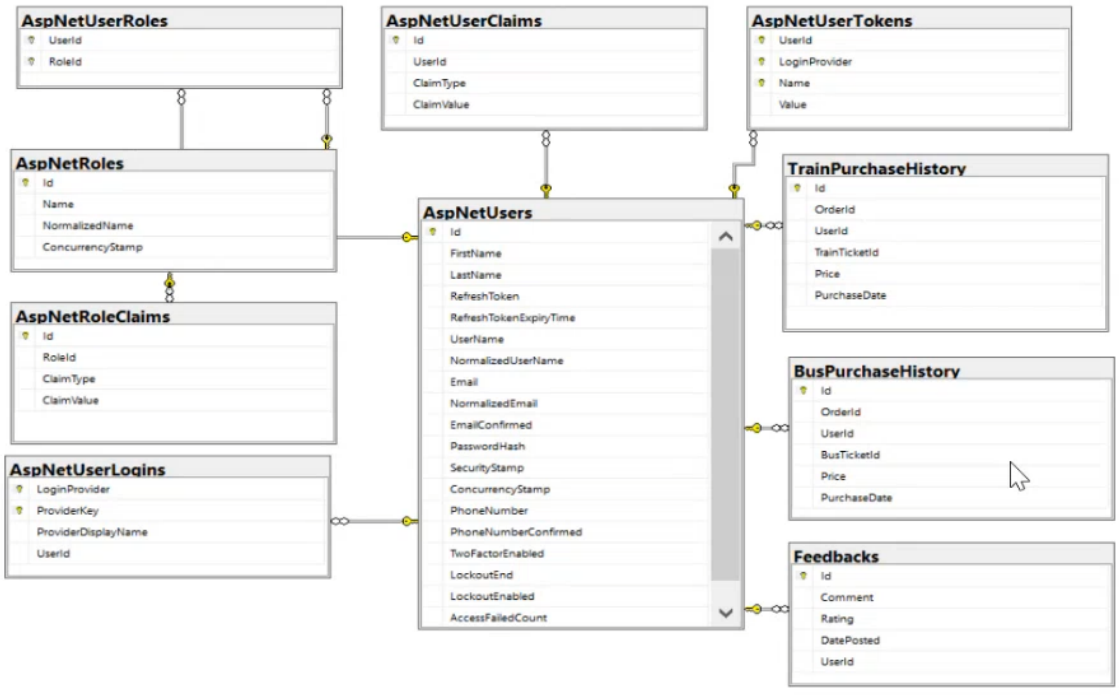


Рисунок 2.5 – User Tables

Автобусна складова бази даних представлена на рисунку 3.2 та відображає логічну структуру, що забезпечує повноцінне управління інформацією про автобусні рейси, транспортні засоби, маршрути, зупинки, а також квитки пасажирів. Дана частина бази є критично важливою для реалізації функціоналу бронювання квитків на автобуси та супутнього сервісу, оскільки охоплює всі етапи — від формування рейсу до фіксації транзакцій покупки.

Центральну роль у цій підсистемі відіграє таблиця BusLines — вона акумулює дані про існуючі маршрути, визначаючи початкову та кінцеву зупинку, а також базовий опис лінії. Саме з цієї таблиці починається логіка побудови маршруту та формування пошукових запитів користувачів. У ній міститься інформація, яка є статичною і не змінюється впродовж часу — наприклад, назви міст, код перевізника тощо.

Таблиця BusMovements реалізує більш динамічну частину логіки — вона відстежує конкретні рейси, пов’язані з маршрутами. Тут зберігаються дата, час відправлення, час прибуття, а також зупинки, на яких планується зупинка. Це дозволяє диспетчерам моніторити виконання рейсів, планувати графіки та реагувати на відхилення в реальному часі. Саме завдяки цій таблиці формується актуальна інформація про доступні рейси.

Дані про автобуси зберігаються в таблиці Buses, яка містить технічні характеристики транспортного засобу: модель, рік випуску, кількість місць, поточний технічний стан та належність до маршруту. Це дозволяє системі перевіряти відповідність автобуса певному рейсу, а також здійснювати контроль за зношуванням парку.

Таблиця BusSeats деталізує план салону автобуса: кожен рядок відповідає окремому місцю, із зазначенням його номера, типу (звичайне, комфорт, економ) і статусу — вільне або заброньоване. Такий підхід дозволяє реалізувати візуальну схему вагону в інтерфейсі користувача і здійснювати точне бронювання.

Інформація про географію руху зберігається в таблиці BusStations, яка включає назви станцій, координати та зв’язок із таблицею Locations, де зафіксовано поштові індекси та країну. Така структура забезпечує однозначність станцій навіть при однакових назвах у різних містах.

Комерційну сторону сервісу реалізує таблиця BusTickets, яка фіксує факт оформлення квитка користувачем. Тут зберігається вся необхідна інформація для перевірки: email пасажира, дата купівлі, ціна, тип квитка (звичайний, студентський, дитячий), часи відправлення й прибуття, а також поточний статус — оплачено, заброньовано, скасовано тощо. Така деталізація дозволяє контролювати фінансову складову системи та забезпечує можливість інтеграції з CRM або бухгалтерськими сервісами.

Загалом, автобусна частина бази даних проектувалась з урахуванням вимог зручності, надійності та масштабованості. Вона дозволяє швидко обробляти запити користувачів, точно відображати наявність місць, гнучке керування графіками та ефективно обслуговувати процес продажу квитків на автобуси.

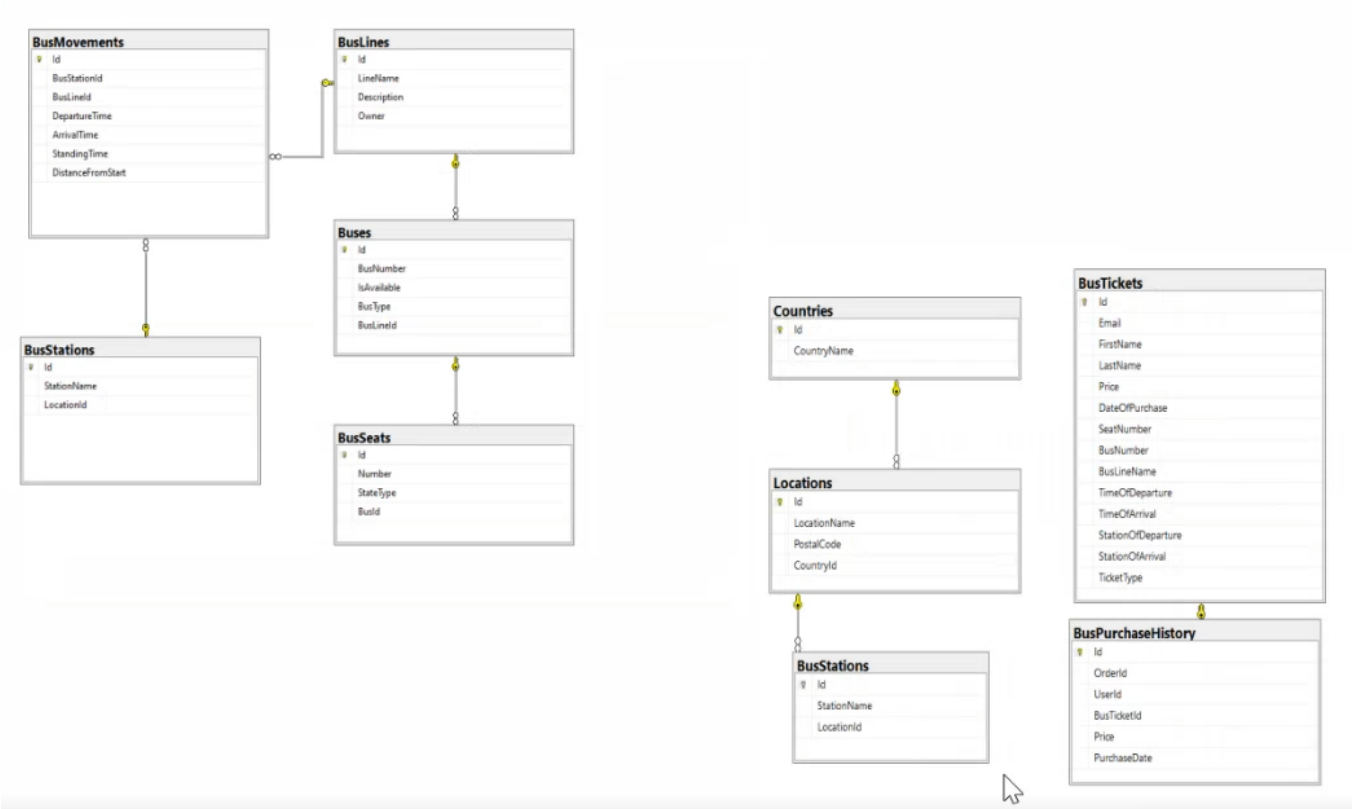


Рисунок 2.6 – Bus Tables

Залізнична частина інформаційної системи займає важливе місце в загальній структурі бази даних, оскільки саме вона забезпечує збереження та обробку ключових даних, необхідних для організації перевезень, управління маршрутами та бронювання квитків. На рисунку 3.3 представлено логічну схему таблиць, які реалізують функціональність залізничного підмодуля, а також відображають взаємозв’язки між окремими компонентами даних.

Центральною таблицею цього сегменту є TrainLines, яка відіграє роль основного сховища для маршрутної інформації. У ній міститься унікальний ідентифікатор маршруту, а також його назва, що дозволяє ідентифікувати напрямок руху. Саме ця таблиця слугує базою для побудови всіх інших операцій — як із розкладом, так і з квитками.

Для детального опису руху потягів використовується таблиця TrainMovements. У ній фіксуються такі параметри, як час прибуття та відправлення на конкретних зупинках, час стоянки та відстань від початку маршруту. Це дозволяє не тільки відображати точний розклад, а й виконувати аналітичні обчислення, зокрема для оптимізації маршрутів або оцінки дотримання графіків.

Інформація про фізичні транспортні засоби організована через таблицю Trains, де вказано номер потяга, назву, тип (пасажирський, експрес тощо) та посилання на маршрут. Додаткову деталізацію забезпечують таблиці Wagons та TrainSeats, які описують, відповідно, типи вагонів, їхню нумерацію та кількість місць, а також конкретні місця з позначенням їх статусу (вільне, заброньоване, технічно недоступне). Це дозволяє реалізувати детальний візуальний вибір місця для користувача в інтерфейсі системи.

Планування маршруту та його реалізацію у часі підтримує таблиця TrainSchedules. Вона фіксує зв’язок між потягами та конкретними рейсами, дозволяє формувати гнучкий розклад і синхронізується з реальними даними з таблиці TrainMovements, що створює єдину інформаційну модель руху.

Географічна складова реалізована за допомогою таблиць Countries, Locations та TrainStations. Така ієрархічна модель дозволяє зберігати точну

інформацію про розташування станцій, запобігає дублюванню назв та забезпечує можливість формування міжнародних маршрутів. Кожна станція через TrainStations пов’язана з конкретною локацією, яка, у свою чергу, належить певній країні.

Фінансова та транзакційна частина моделі реалізована через таблиці TrainTickets та TrainPurchaseHistory. У TrainTickets міститься повна інформація про оформлений квиток: персональні дані пасажира, номер вагона та місця, маршрут, дати та часи відправлення й прибуття. У таблиці TrainPurchaseHistory фіксуються дані про здійснення покупки: ідентифікатор замовлення, вартість, дата операції та прив’язка до конкретного користувача. Така структура дозволяє реалізувати відображення історії замовлень, формування електронного квитка, а також зберігання важливих фінансових даних.

Загалом, залізнична частина бази даних сформована з урахуванням вимог до продуктивності, логічної цілісності та зручності масштабування. Вона охоплює як статичну (маршрути, станції), так і динамічну (розклади, квитки) інформацію, забезпечуючи безперервну роботу системи бронювання в умовах змінних навантажень та високих вимог до точності даних.

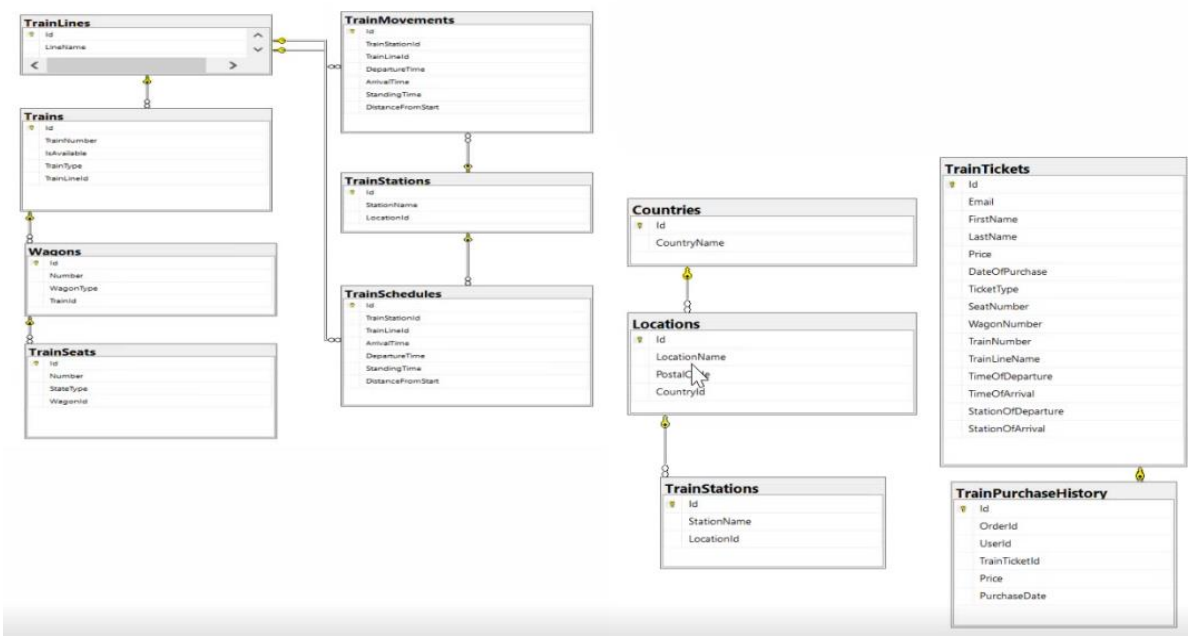


Рисунок 2.7 – Train Tables

Використання ORM-фреймворка Entity Framework Core забезпечує автоматичне формування параметризованих SQL-запитів, що підвищує безпеку даних та продуктивність системи. Завдяки продуманій структурі бази даних і чіткій сегментації таблиць, система демонструє ефективну роботу навіть при великому навантаженні [10].

2.7 Інтеграція з платіжними системами

У системі передбачено підтримку онлайн-оплати шляхом підключення зовнішніх платіжних шлюзів. Насамперед було обрано провайдери, що мають стійку репутацію, сертифіковані за PCI DSS і дають змогу працювати з великою кількістю валют та методів розрахунку — зокрема Stripe, PayPal і LiqPay. Наприклад, офіційна документація Stripe підкреслює, що сервіс охоплює 135+ валют і понад сотню способів оплати, що істотно полегшує вихід на міжнародний ринок.

Інтеграція відбувається через REST-API кожного шлюзу: застосунок ініціює створення платіжного запиту, отримує токен-ідентифікатор транзакції, перевіряє її статус і, за потреби, формує запити на повернення коштів. Такий механізм мінімізує участь користувача: фінансова операція завершується без переходу на сторонні сторінки, а дані платежу зберігаються лише у платіжному провайдера, що знижує ризики для сервера додатка.

Захист платіжної інформації базується на кількох рівнях. Під час передачі інформації між клієнтом і сервером застосовується шифрування TLS 1.3, а зберігати самі реквізити картки система не має права — це прямо забороняє стандарт PCI DSS, що визначає технічні та організаційні вимоги для всіх компаній, які обробляють платежі. Повний номер PAN та інші чутливі відомості не потрапляють у логи, а для повторних транзакцій замість реальних даних використовується токен [11]. Таким чином, поєднання перевірених платіжних шлюзів, API-інтеграції та дотримання вимог PCI DSS гарантує швидкі й безпечні транзакції, а користувачам забезпечує звичний і довірчий досвід оплати.

2.8 Тестування та налагодження

У системі бронювання якість коду перевіряли ретельно й послідовно, тому процес тестування та налагодження охопив кілька рівнів. Насамперед виконувалося юніт-тестування: кожен окремий метод чи клас ізолювали від решти програми й проганяли через набір автотестів. Такий підхід дозволив одразу виявляти помилки на ранній стадії, коли їх дешевше виправити, і гарантував, що базова логіка працює коректно незалежно від зовнішніх залежностей.

Коли окремі модулі показали стабільні результати, переходили до інтеграційних перевірок. На цьому етапі оцінювали, як компоненти взаємодіють між собою: чи збігаються формати даних, чи коректно відпрацьовують спільні сценарії, чи немає «розривів» на межі між сервісами. Саме інтеграційне тестування допомагало виявити приховані неузгодженості, які не було видно під час ізольованих запусків [12].

Далі проводилося системне тестування. Тут програму запускали як цілісний продукт, перевіряючи відповідність усім функціональним вимогам, стійкість під навантаженням і дотримання вимог безпеки. Стрес-тести моделювали пікову активність, а захисні сценарії імітували потенційні атаки, щоб переконатися у відсутності критичних вразливостей.

Після завершення технічних перевірок до процесу залучали реальних користувачів. Вони тестували інтерфейс у звичних для себе умовах, і саме на цьому етапі виявлялися нефункціональні недоліки: непередбачувана поведінка елементів, незручне розташування кнопок чи неочевидні повідомлення про помилки. Зібрані відгуки лягали в основу фінальних доопрацювань, завдяки чому система стала не лише стабільною, а й справді зручною у щоденному використанні [13].

Налаштування програмних засобів

Коректне налаштування інструментів — основа стабільної роботи всієї системи, тому підготовку середовища розпочинають із встановлення ключового софту. Для серверної частини обирають Visual Studio, а JavaScript-інфраструктура ґрунтується на Node.js разом із менеджером пакетів npm, що дає змогу оперативно додавати залежності та оновлювати їх. Коли середовище розробника готове, переходять до конфігурації веб-сервера: зазвичай це IIS або інший аналог, де налаштовують параметри безпеки, вірний шлях до застосунку та потрібні модулі, аби забезпечити безперервну роботу сервісу у production-умовах.

Далі створюють базу даних (наприклад, на платформі Microsoft SQL Server), встановлюють підключення з застосунком і ініціалізують схему — таблиці, зв’язки, а також політику резервного копіювання й відновлення. Це критично для збереження цілісності інформації та швидкого відновлення у разі збою. Після цього налаштовують фронтенд: через npm встановлюють необхідні бібліотеки, а Webpack і Babel збирають та оптимізують вихідний код, що забезпечує швидке завантаження й коректну роботу інтерфейсу у браузері.

Фінальний крок — інтеграція платіжних сервісів (PayPal, Stripe, LiqPay тощо). Тут конфігурують API-ключі, перевіряють захищеність з’єднання та виконують тестові транзакції, переконуючись, що шифрування та інші механізми захисту конфіденційних даних працюють коректно.

Таким чином, послідовно налаштоване середовище для розробки, тестування й розгортання гарантує надійну й безпечну роботу системи, задовольняючи вимоги користувачів і підтримуючи високий комфорт її подальшої експлуатації.

2.9 Висновок

У другому розділі розглянуто ключові технічні рішення, на яких побудована система бронювання квитків. Обрана клієнт-серверна архітектура

забезпечує ефективний розподіл обов’язків між інтерфейсом і сервером, що сприяє гнучкості, масштабованості та продуктивності.

Модульна структура дозволяє легко підтримувати та розширювати систему, а сучасні технології — .NET Core, React, Entity Framework Core, JWT і Identity — забезпечують надійність, швидкодію та безпеку. Додатково були представлені UML-діаграми, що ілюструють сценарії роботи користувачів із системою, проєктування бази даних і модуль бронювання.

Таким чином, комплексний підхід до архітектурного та технічного проєктування забезпечує створення якісного та ефективного програмного продукту.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ

3.1 Фронтенд розробка

Для реалізації клієнтської частини системи бронювання було свідомо обрано React — популярну JavaScript-бібліотеку, розроблену компанією Meta. З моменту свого запуску вона впевнено зайняла лідируючі позиції серед засобів побудови користувацьких інтерфейсів, і згідно з опитуванням State of JS 2023, React залишається найбільш використовуваним інструментом, суттєво випереджаючи такі альтернативи, як Angular і Vue. Така популярність не є випадковою, а зумовлена низкою важливих архітектурних переваг, які роблять бібліотеку особливо привабливою для розробників сучасних масштабованих веб-додатків.

Однією з основних концепцій React є компонентна модель, що дозволяє розбивати інтерфейс на невеликі ізольовані блоки, кожен з яких має свою власну логіку, стан і відповідальність. Це не лише спрощує процес тестування та повторного використання коду, а й істотно полегшує підтримку системи у майбутньому. Наприклад, внесення змін у конкретний компонент не призводить до небажаних побічних ефектів в інших частинах інтерфейсу, що особливо важливо при роботі у великих командах або у тривалих проєктах з поступовим розвитком.

Не менш важливою є й продуктивність React, яка досягається завдяки впровадженню віртуального DOM (Virtual DOM). Цей підхід дозволяє React оптимізувати процес оновлення сторінки, зводячи кількість прямих змін у реальному DOM до мінімуму. Як результат — зростає швидкість взаємодії з інтерфейсом, що особливо помітно на складних сторінках із великою кількістю динамічних елементів. Незалежні тестування підтверджують, що така стратегія дозволяє досягати вищої швидкодії в порівнянні з традиційними методами взаємодії з DOM API [14].

Окрім технічних характеристик, React відзначається винятковою гнучкістю та широкою екосистемою. Він легко інтегрується з іншими

бібліотеками та фреймворками: наприклад, для управління станом часто використовують Redux або Zustand, для обміну даними з сервером — Axios або GraphQL-клієнти (зокрема, Apollo). Це робить можливим адаптацію React до будь-якого стеку технологій і дозволяє поступово впроваджувати нові функціональні можливості без повної перебудови коду. У книзі Pro React 18 А. Фрімана окремо підкреслюється здатність React до безболісної інтеграції в уже існуючі архітектурні рішення, що лише підтверджує універсальність і життєздатність цієї бібліотеки.

Таким чином, застосування React у проєкті бронювання квитків є цілком обґрунтованим. Компонентна структура сприяє масштабованості та повторному використанню, virtual DOM — покращенню продуктивності, а гнучкість екосистеми — легкій адаптації до вимог користувачів та змін у проєкті. У сукупності ці властивості створюють потужне середовище для побудови надійного, динамічного та зручного у використанні інтерфейсу.

3.2 Бекенд розробка

Розробка серверної частини системи бронювання здійснювалась із фокусом на стабільність, продуктивність і масштабованість. Як основну платформу було обрано .NET Core — сучасне кросплатформне рішення від компанії Microsoft, що поєднує в собі високу продуктивність, гнучкість архітектури та потужну інтеграцію з інструментами хмарної інфраструктури. Основною мовою програмування стала C#, яка є одним із найбільш універсальних інструментів у сфері веб-розробки, особливо для побудови серверних API та сервісів.

C# підтримує об’єктно-орієнтовану парадигму, що дозволяє створювати модульні та масштабовані програмні рішення. Високий рівень суворої типізації у C# значно зменшує ймовірність помилок під час компіляції, а автоматичне керування пам’яттю забезпечує стабільність навіть при роботі з великими обсягами запитів і даних. Застосування вбудованих механізмів, як-от збирач

сміття (GC), дає змогу уникати витоків пам’яті та аварійних завершень процесів, що є критично важливим для цілодобового веб-сервісу.

Платформа .NET Core, у свою чергу, є повноцінним інструментарієм для створення RESTful API, що дозволяє ефективно реалізувати архітектуру клієнт-сервер. Значна перевага .NET Core полягає в її кросплатформенності: один і той самий додаток можна запускати під Windows, Linux або macOS без потреби в зміні коду. Це відкриває широкі можливості для розгортання проєкту на будь-яких серверних платформах, включаючи Docker-контейнери та хмарні сервіси.

Особливу увагу приділено інтеграції з Microsoft Azure — завдяки нативній підтримці хмарних сервісів, система легко масштабується в умовах зростання навантаження. Azure DevOps забезпечує налаштування CI/CD-конвеєра, що дозволяє реалізувати автоматизоване розгортання та оновлення програмного забезпечення. Також доступна підтримка логування, моніторингу продуктивності та інструменти для аварійного відновлення, що гарантує високу надійність роботи системи [15].

Доступ до бази даних реалізовано за допомогою ORM-технології Entity Framework Core, яка абстрагує низькорівневу SQL-логіку у вигляді зручного об’єктного доступу. Це дозволяє зменшити кількість помилок, пов’язаних із некоректними SQL-запитами, та суттєво прискорити процес розробки. ORM також підтримує міграції, що полегшує оновлення структури бази даних без втрати даних.

У результаті, зв’язка C# + .NET Core + EF Core дозволила реалізувати серверну логіку, яка відповідає сучасним стандартам розробки: вона є продуктивною, безпечною, масштабованою та легко супроводжуваною. Такий вибір технологій обґрунтований потребою у створенні довготривалого, стабільного та гнучкого рішення, яке можна буде легко адаптувати під майбутні вимоги або розширити у відповідь на нові виклики.

3.3 Система баз даних

Ефективне зберігання, обробка та захист даних є однією з найважливіших складових будь-якої сучасної інформаційної системи, особливо якщо йдеться про систему бронювання, де оперують великою кількістю чутливої інформації. Для реалізації серверної частини цієї системи було обрано Microsoft SQL Server — одну з найпотужніших реляційних систем управління базами даних (СУБД), яка поєднує в собі високий рівень продуктивності, масштабованість та сучасні засоби безпеки.

Однією з ключових переваг Microsoft SQL Server є оптимізований рушій запитів, що дозволяє досягати значної швидкодії навіть при обробці великих обсягів інформації. Завдяки підтримці технології in-memory OLTP, частина таблиць може зберігатися в оперативній пам’яті, що значно зменшує час доступу до даних та зменшує затримки при високому навантаженні.

У сфері безпеки SQL Server також демонструє високий рівень надійності. Зокрема, підтримується Transparent Data Encryption (TDE) — вбудований механізм, який шифрує дані "на льоту", забезпечуючи їх захист навіть у разі несанкціонованого доступу до файлової системи. Крім того, система дозволяє реалізувати гнучке керування ролями та правами доступу, що критично важливо у багатокористувацькому середовищі: кожен користувач має доступ лише до тієї частини даних, яка відповідає його рівню авторизації.

Не менш важливою є інтеграція SQL Server у ширшу екосистему Microsoft. Це дозволяє використовувати такі сервіси, як Azure SQL Database для організації хмарного зберігання, Azure Backup для резервного копіювання, а також Power BI для візуалізації аналітичної інформації у зручному форматі. Така сумісність значно полегшує адміністрування, масштабування та моніторинг стану бази даних у режимі реального часу.

Завдяки таким характеристикам, SQL Server став надійною основою для розгортання системи бронювання квитків. Його використання дозволяє впевнено говорити про довготривалу стабільність, адаптивність до зростаючого

навантаження та можливість гнучкого розвитку у разі зміни бізнес-вимог. У підсумку, вибір цієї СУБД забезпечив надійне функціонування всієї системи з урахуванням як поточних, так і майбутніх потреб.

3.4 Вимоги до користувачів

Службовий веб-хостинг робить систему онлайн-бронювання цілком незалежною від конкретної платформи: користувач заходить через будь-який браузер і фактично з тієї самої миті отримує повний набір можливостей — від пошуку рейсу до оплати квитка. Головним тут є те, що сам інтерфейс розроблено за принципами responsive web design і device-independence: макет та стилі автоматично підлаштовуються під розмір екрана й можливості пристрою, тож сторінка коректно відображається і на 6-дюймовому смартфоні, і на великому десктопному моніторі.

На практиці це означає, що власнику сучасного Android- або iOS-смартфона досить відкрити браузер з активним підключенням до мережі — система сама оптимізує шрифти, кнопки й масиви даних, щоб ними було зручно користуватися пальцями навіть на невеликому дисплеї. Планшет дає ще більше «повітря» для елементів керування, тому перегляд розкладу, вибір вагона чи керування вже придбаними квитками відбувається без потреби масштабувати сторінку чи гортати горизонтально. Користувачі ноутбуків та стаціонарних ПК, своєю чергою, одержують повноцінний настільний інтерфейс із розширеною панеллю фільтрів і «гарячими» клавішами, що скорочують час на повторні операції.

Незалежно від типу клієнтського пристрою ключову роль відіграє стабільне інтернет-з’єднання: від цього залежить, наскільки швидко сервер поверне актуальний стан місць і підтвердить платіж. Сам веб-сайт протестовано у сучасних браузерах Chrome, Firefox, Edge та Safari; підтримка стандартів HTML5, CSS Media Queries і Service Worker-кеша гарантує, що навіть за тимчасових мережевих затримок інтерфейс залишиться відгуковим, а вже завантажені дані не зникнуть до повного поновлення сеансу.

Таким чином, завдяки адаптивній верстці й широкому браузерному охопленню система бронювання забезпечує однаково комфортний досвід як для мобільних пасажирів, що купують квиток «на ходу», так і для користувачів, котрі планують подорож із робочого ноутбука чи домашнього ПК [16].

3.4 Опис програмної реалізації

Програмна реалізація системи бронювання квитків базується на сучасному, перевіреному в індустрії стеку технологій, що забезпечує високу продуктивність, масштабованість і зручність супроводу. Кожен з обраних інструментів виконує чітко визначену функцію у загальній архітектурі системи, що дозволяє забезпечити стабільну роботу та можливість подальшого розширення функціоналу.

Клієнтська частина, яка відповідає за взаємодію з користувачем, реалізована з використанням бібліотеки React мовою JavaScript. Завдяки компонентній структурі інтерфейс поділено на окремі незалежні блоки (компоненти), що дозволяє ефективно керувати логікою UI, повторно використовувати код та динамічно оновлювати контент без повного перезавантаження сторінки. Механізм Virtual DOM значно покращує швидкість рендерингу, що позитивно позначається на відгуку системи [17].

Серверна логіка розроблена мовою C# із використанням платформи .NET Core, яка забезпечує кросплатформеність, високу швидкодію, а також надає гнучкі інструменти для обробки запитів, контролю доступу та підключення до зовнішніх API. Саме ця частина відповідає за обробку бізнес-логіки: обчислення тарифів, перевірку даних, взаємодію з базою та безпечну авторизацію [18].

Для зберігання даних та організації доступу до них використовується ORM-фреймворк Entity Framework Core, який виконує автоматичне перетворення об'єктів у таблиці SQL Server і навпаки. Це дозволяє розробникам працювати із сутностями як зі звичайними класами C#, не опускаючись до низькорівневого SQL-коду. Крім того, EF Core підтримує міграції, що значно спрощує внесення змін у структуру БД без втрати існуючих даних [19].

З метою безпечної обробки персональних даних реалізовано зв'язку ASP.NET Core Identity (для управління акаунтами) та JWT (JSON Web Tokens) для авторизації. Користувач після успішного входу отримує токен доступу, який автоматично перевіряється сервером при кожному запиті. Це забезпечує високий рівень безпеки та дозволяє уникнути необхідності зберігання сесій на сервері.

В окремих модулях реалізовано математичну частину, зокрема алгоритми відбору та сортування маршрутів. Вони дозволяють швидко обрати найвигідніший варіант поїздки на основі введених параметрів: відстань, тип вагона, наявність додаткових послуг (наприклад, постіль чи харчування). Це покращує якість обслуговування клієнтів та автоматизує складні розрахунки вартості квитків у реальному часі.

Таким чином, реалізована система є прикладом комплексного веб-рішення, в якому гармонійно поєднуються зручність для користувача, безпека, ефективність та гнучкість. Обрані інструменти підтвердили свою ефективність під час тестування та продемонстрували стабільну роботу у різних сценаріях використання.

3.5 Системні вимоги

Щоб система бронювання залізничних і автобусних квитків працювала стабільно та без збоїв, потрібно дотримуватися певних технічних вимог. Вони охоплюють обидві частини — клієнтську (тобто все, що бачить користувач у браузері) і серверну (де обробляються запити, зберігаються дані й виконується логіка системи).

Клієнтські вимоги стосуються обладнання та програмного забезпечення, з якого користувач звертається до сайту.

Серверна частина потребує більш потужного середовища — зокрема, встановленої платформи .NET Core, SQL-сервера, підтримки API, а також конфігурації, що дозволяє обробляти велику кількість запитів одночасно. Розділення вимог на дві зони дає змогу точно визначити, де саме можуть виникати проблеми, і своєчасно їх усунути.

Для клієнтської частини. Щоб клієнтська частина системи бронювання працювала без збоїв, пристрій користувача має відповідати базовим технічним вимогам. Насамперед — це операційна система: рекомендується використовувати сучасні версії Windows (не нижче Windows 7), macOS (від 10.12 і вище), або ж актуальні дистрибуції Linux. Це забезпечує сумісність з технологіями, на яких побудовано інтерфейс системи.

Ключову роль відіграє також вибір веб-браузера. Найкращу стабільність та швидкодію гарантує Google Chrome, хоча підтримуються й інші популярні варіанти — Mozilla Firefox, Microsoft Edge та Safari. Головне, щоб браузер був оновленим і підтримував сучасні веб-стандарти, такі як HTML5, CSS3 та JavaScript ES6.

Щодо апаратного забезпечення, для комфортної роботи з системою необхідно, щоб пристрій мав щонайменше 2 ГБ оперативної пам’яті. Цього достатньо для стабільного функціонування навіть у разі відкриття кількох вкладок одночасно. Рекомендований процесор — з двома ядрами та тактовою частотою не менше 2 ГГц, що дозволить без затримок обробляти запити, динамічні елементи інтерфейсу та обмін із сервером.

Загалом ці параметри відповідають середньому рівню сучасних користувацьких пристроїв, тому доступ до системи є можливим як з ноутбуків і настільних ПК, так і зі смартфонів та планшетів, за умови стабільного інтернет-з’єднання.

Для серверної частини: Щоб серверна частина системи бронювання працювала стабільно та витримувала навантаження, потрібно дотримуватись певних апаратних і програмних вимог. У першу чергу це стосується операційної системи: рекомендовано використовувати Windows Server (версія 2016 або новіша) або одну з актуальних дистрибуцій Linux. Обидва варіанти забезпечують достатній рівень надійності, стабільності й підтримку усіх необхідних інструментів для розгортання серверного ПЗ

За обсягом ресурсів сервер має бути оснащений щонайменше 4 ГБ оперативної пам’яті — цього вистачає для обробки одночасних запитів,

виконання бізнес-логіки, обслуговування бази даних та підтримки API без затримок. Щодо процесора, то він повинен мати щонайменше 4 фізичних ядра з тактовою частотою від 2.5 ГГц. Така конфігурація дозволяє ефективно обробляти складні обчислення, великі обсяги даних і паралельні запити від клієнтів.

Для збереження логів, файлів конфігурації та службових даних необхідно мати хоча б 1 ГБ вільного простору на диску. При цьому важливо, щоб диск працював на швидкому SSD — це покращує загальну продуктивність системи при зверненні до файлів журналів і даних.

У частині СУБД використовується Microsoft SQL Server версії 2017 або новішої. Для його роботи потрібно щонайменше 3 ГБ вільного простору на диску для збереження таблиць із інформацією про бронювання, користувачів, поїздки та транзакції. Ця СУБД також підтримує функції резервного копіювання, захисту доступу, індексації та оптимізації запитів, що суттєво впливає на загальну швидкодію та надійність системи в цілому.

3.6 Завантаження та налаштування системи

Щоб розпочати роботу з системою бронювання, потрібно спочатку отримати її архівний пакет. Його можна завантажити з офіційного сайту проєкту або з відкритого репозиторію, наприклад, GitHub. Після завантаження архіву, слід розпакувати вміст у зручне місце на сервері або локальному комп’ютері — наприклад, у каталог C:\BookingSystem або іншу обрану директорію.

Далі необхідно перейти до налаштування бази даних. Якщо на сервері ще не інстальовано Microsoft SQL Server, його слід завантажити з офіційного сайту Microsoft і виконати стандартну установку. Після цього відкрийте SQL Server Management Studio (SSMS) або інший зручний інструмент роботи з БД, та запустіть SQL-скрипти, що входять у комплект поставки системи. Ці скрипти створюють таблиці, зв’язки, індекси, а також початкові записи, необхідні для роботи (наприклад, адміністратора за замовчуванням або базові довідники).

Після виконання скриптів база даних буде повністю готова до взаємодії з бекендом системи.

Налаштування серверної та клієнтської частин. Щоб система запрацювала повноцінно, потрібно поетапно налаштувати і серверну, і клієнтську частини. Серверна частина починається з установки .NET Core SDK — бажано останньої стабільної версії. Далі слід відкрити бекенд-проєкт у середовищі розробки, наприклад, Visual Studio або Visual Studio Code. Перед запуском необхідно налаштувати файл конфігурації appsettings.json: у ньому вказуються параметри з'єднання з базою даних, зокрема рядок підключення (ConnectionString), а також додаткові опції — як-от логування або режим міграцій. Після цього серверну частину можна запустити через термінал командою dotnet run. Якщо всі залежності встановлені правильно, сервіс почне працювати, про що свідчитиме повідомлення у консолі (див. рисунок 3.1).

Після того як сервер піднято й він відповідає на запити, можна переходити до клієнтської частини, яка взаємодіє з API і відображає інтерфейс користувача.

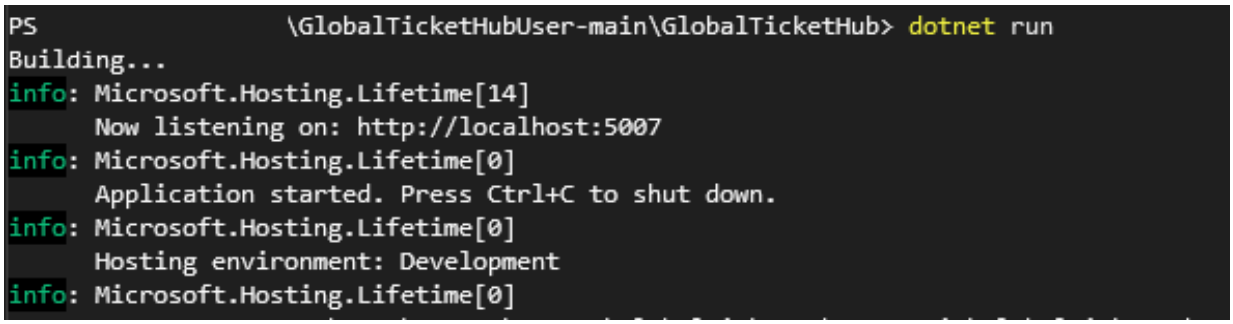


Рисунок 3.1 – Запуск сервера

Щоб запустити клієнтську частину, спочатку встановіть Node.js і npm. Далі відкрийте проєкт у терміналі та виконайте команду npm install, щоб підтягнути всі залежності. Після цього запустіть додаток за допомогою npm start. Якщо все налаштовано правильно, інтерфейс відкриється у браузері (зазвичай на http://localhost:3000), як показано на рисунку 4.2. Система буде готова до використання.

```
Compiled successfully!

You can now view globalticket.client in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.151:3000
```

Рисунок 3.2 – Запуск клієнтської частини

3.7 Сценарій роботи користувача з системою

Після авторизації користувач потрапляє на головну сторінку з простим і зручним інтерфейсом, як показано на рисунку 4.3. У верхній частині розміщено логотип компанії "Pass", який виконує функцію впізнаваного візуального елемента та допомагає швидко зорієнтуватися на сайті. Інтерфейс розроблений таким чином, щоб навіть новий користувач зміг легко розпочати бронювання квитків.

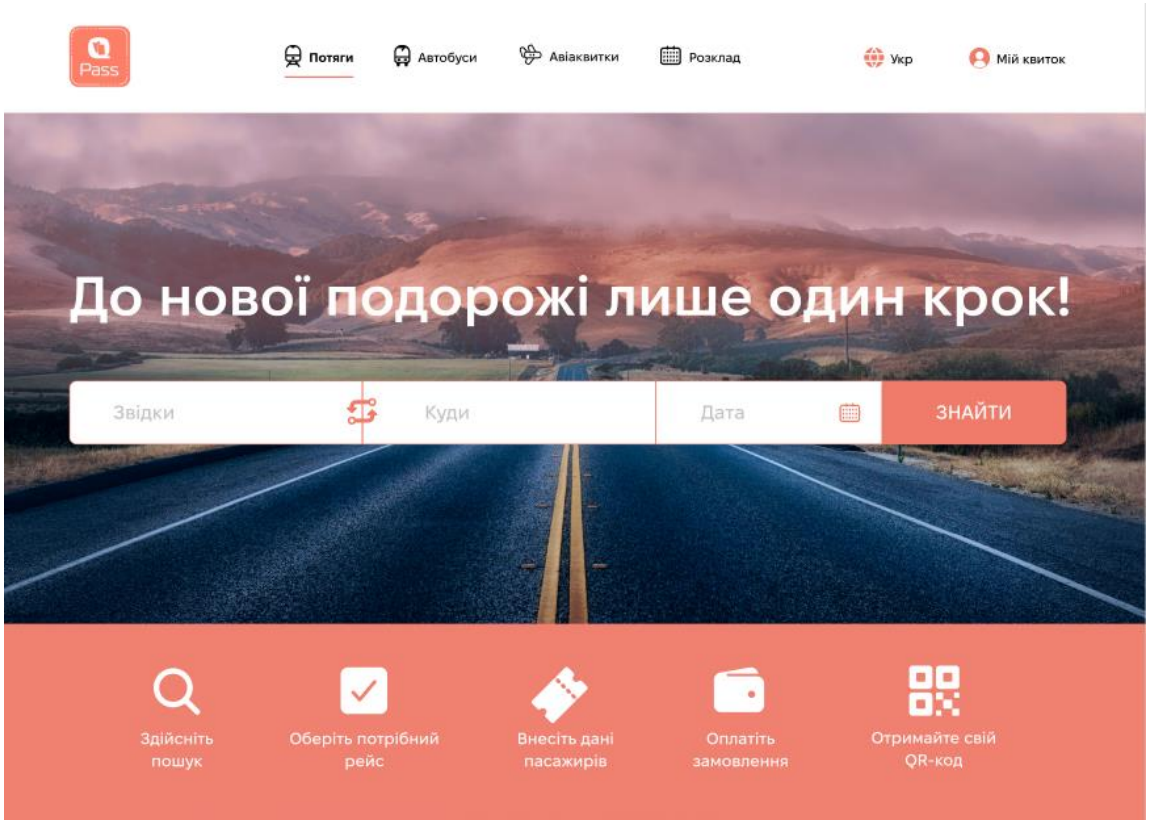


Рисунок 3.3 – Головний екран

У центральній частині головної сторінки знаходиться панель пошуку маршруту. Тут користувач вводить станцію відправлення, пункт призначення та дату поїздки. Після цього достатньо натиснути кнопку "Знайти", щоб система автоматично сформувала список відповідних рейсів.

На рисунку 4.4 зображено приклад виведення результатів пошуку після введення користувачем параметрів маршруту. Інтерфейс реалізовано таким чином, щоб навіть за великої кількості доступних рейсів пасажир міг легко й швидко зорієнтуватися серед варіантів. У таблиці чітко структуровано інформацію: зазначено час відправлення й прибуття, номер потяга, маршрут, загальну тривалість у дорозі, а також передбачено кнопку для переходу до бронювання.

Кожен блок представлений у форматі окремої картки, що дозволяє візуально розділяти рейси між собою. Це значно полегшує навігацію, особливо на мобільних пристроях. Крім того, позначки часу та візуальні маркери роблять інтерфейс більш доступним для користувачів з різними рівнями цифрової грамотності.

Додатково передбачена можливість сортування результатів за кількома критеріями — за часом відправлення, тривалістю подорожі або ціною. Це дозволяє швидко налаштувати відображення даних під індивідуальні потреби: наприклад, обрати найшвидший варіант або той, що прибуває в зручний час.

Таким чином, система не лише надає базову інформацію про рейси, а й допомагає приймати обґрунтовані рішення за допомогою простого й ефективного візуального представлення даних.

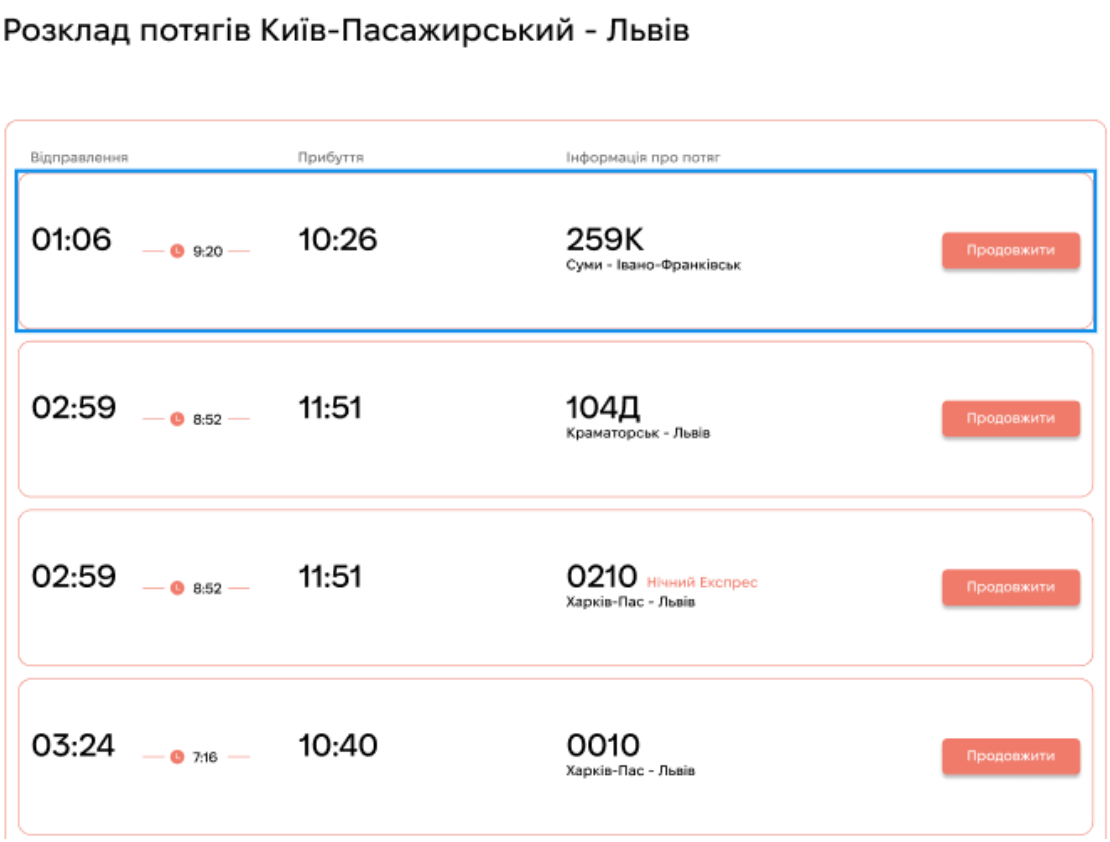


Рисунок 3.4 – Результати пошуку

Після того як користувач ознайомився з наявними рейсами та обрав відповідний маршрут, система перенаправляє його до наступного етапу — вибору місця. Це один із ключових кроків у процесі бронювання, адже саме на цьому етапі користувач приймає остаточне рішення щодо умов подорожі.

На рисунку 3.5 зображено інтерфейс із візуальною схемою вагона, де чітко розділено вільні та зайняті місця. Кожне місце позначено кольором відповідно до його статусу: вільні місця виділено одним кольором зайняті місця іншим, обране місце — ще одним. Цей підхід забезпечує максимальну зручність: користувач бачить повну картину розміщення пасажирів у вагоні, що дає змогу інтуїтивно обрати бажане місце. Особливо це зручно для пасажирів, які подорожують у групі або надають перевагу певному розташуванню — наприклад, біля вікна чи ближче до виходу.

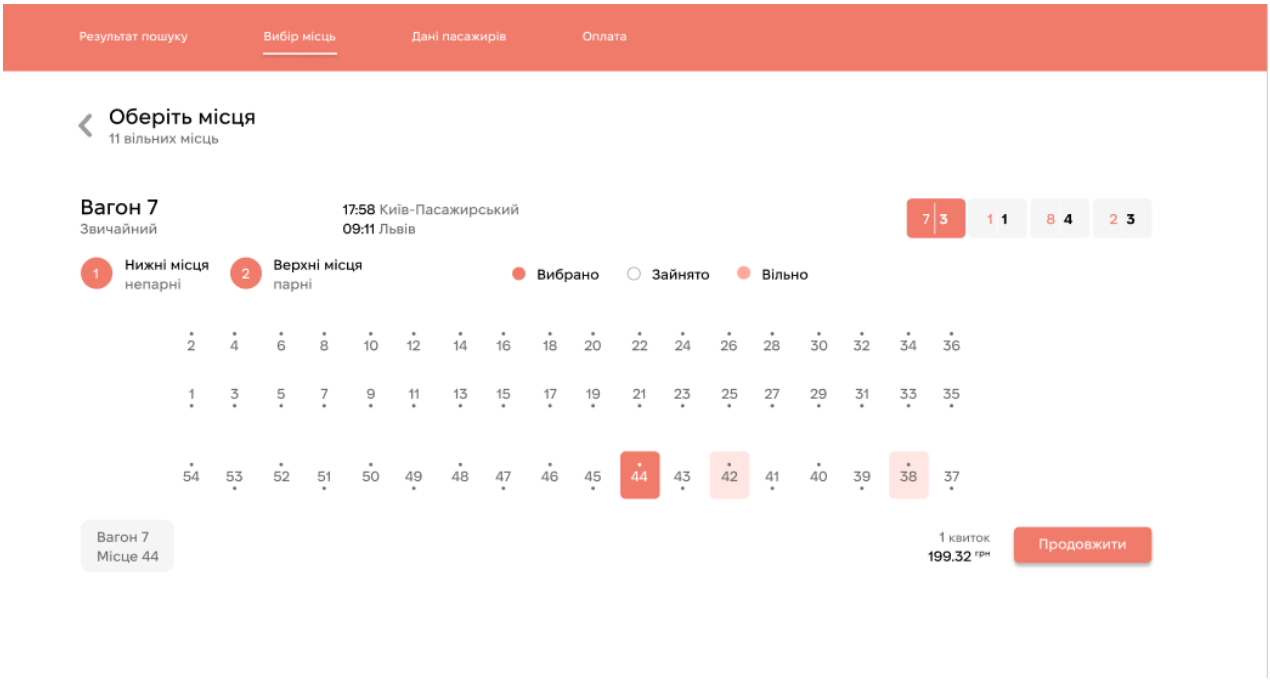


Рисунок 3.5 – Вибір місця

Після того як користувач вибрав рейс та конкретне місце у вагоні, система автоматично переходить до етапу введення персональних даних пасажира. Цей етап є критичним, оскільки від правильності введеної інформації залежить як коректність формування квитка, так і можливість повернення або зміни замовлення в майбутньому.

На інтерфейсі, зображеному на рисунку 4.6, користувачу пропонується обрати тип пасажира (дорослий, дитячий, студентський), ввести ім’я та прізвище, а також зазначити додаткові параметри: вік, наявність пільг, обрати додаткові послуги (наприклад, постіль або напої) тощо. У правій частині екрана в режимі реального часу оновлюється загальна інформація про замовлення — маршрут, час відправлення та прибуття, а також фінальна вартість квитка з урахуванням обраних параметрів.

Кожне введене значення одразу перевіряється на валідність, а в разі помилки система підсвічує поле та надає підказку. Завдяки зручній структурі форми, користувач швидко орієнтується у необхідних полях і легко завершує етап заповнення. Таким чином, реалізований інтерфейс не тільки спрощує

процес взаємодії з системою, а й знижує ймовірність помилок при оформленні квитка.

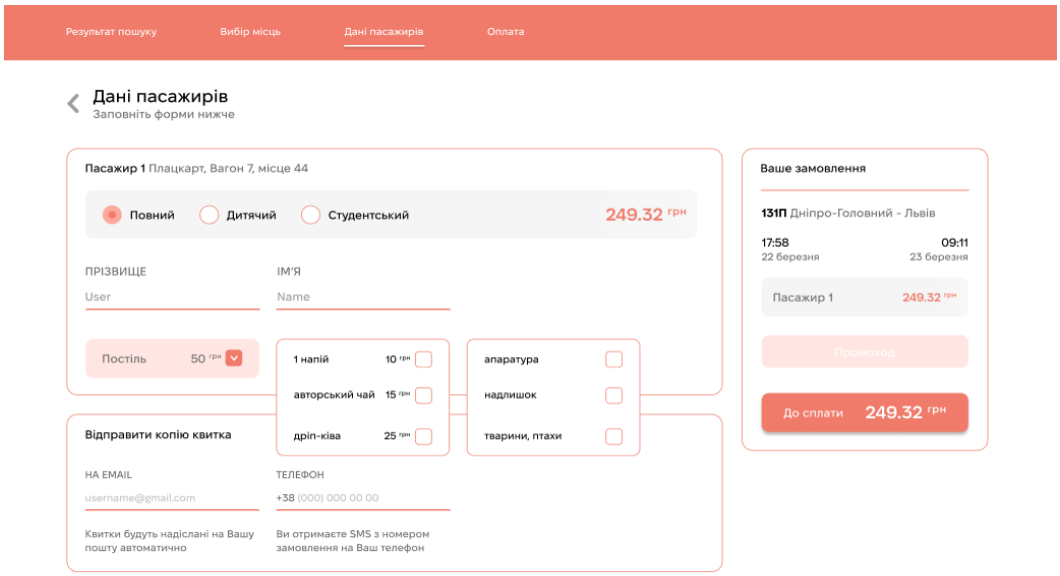


Рисунок 3.6 – Дані пасажира

На етапі бронювання квитка користувач вводить свої основні дані — ім'я, прізвище та тип пасажира (дитина, студент або дорослий). Залежно від обраного статусу система автоматично застосовує відповідну знижку: для студентів — за наявності студентського квитка, для дітей — 50% від вартості.

Також можна обрати спосіб доставки квитка: SMS на номер телефону або електронною поштою. У відповідне поле вводиться номер або e-mail залежно від обраного варіанту.

Крім того, доступні додаткові послуги — наприклад, напої, постіль, аудіообладнання, або перевезення тварин. Кожна з них додається до ціни замовлення.

На завершальному етапі користувач обирає зручний спосіб оплати та натискає кнопку "Оплатити". Після успішного проведення транзакції система показує підтвердження бронювання з усіма деталями квитка.

Зареєстровані користувачі мають доступ до функцій редагування своїх персональних даних: вони можуть у будь-який момент оновити ім’я, прізвище, email чи номер телефону — це зручно для підтримання актуальної інформації

Також у профілі доступна зміна пароля. Це важлива функція безпеки, яка дозволяє захищати обліковий запис від стороннього доступу та гарантує конфіденційність особистих даних.

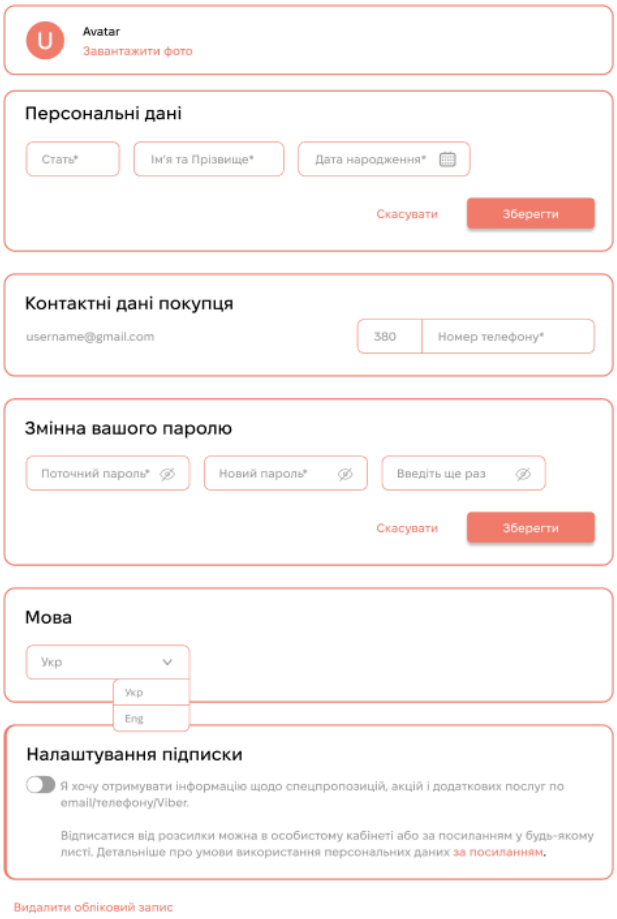


Рисунок 3.7 – Налаштування акаунта

Система дає змогу користувачам не лише оновлювати особисту інформацію й змінювати пароль, а й повністю керувати своїм обліковим записом. За потреби можна видалити профіль — це допомагає зберегти конфіденційність і деактивувати доступ, якщо сервіс більше не використовується.

Зареєстровані користувачі після входу до особистого кабінету отримують зручний інструмент для контролю своїх поїздок. На сторінці перегляду

замовлень відображається вся інформація про бронювання: дати, напрямки, типи квитків, їхній поточний статус — наприклад, чи оплата пройшла, чи квиток активний, чи поїздка вже відбулася.

Можливість мати доступ до всіх деталей у кілька кліків — це саме те, що робить сервіс зручним і сучасним. Така функція — це не просто зручність, а відчуття контролю та впевненості, що вся потрібна інформація завжди під рукою.

Згідно з результатами тестування, розроблена система бронювання квитків успішно виконує всі функції, що зазначалися в проектувані програмного додатку [20]

3.8 Висновок

У цьому розділі детально розглянуто процес взаємодії користувача із системою бронювання залізничних квитків. Починаючи від встановлення і налаштування, і закінчуючи завершенням покупки — система забезпечує чітку, зручну й логічну послідовність дій. Усі ключові функції, як-от пошук маршруту, вибір місця, оплата та перегляд історії замовлень, реалізовано інтуїтивно зрозуміло. Завдяки адаптивному інтерфейсу платформа підходить як для комп’ютерів, так і для мобільних пристроїв, що забезпечує гнучкість використання.

Крім того, система дає можливість користувачам управляти особистим профілем, редагувати контактні дані та контролювати статус квитків. Окрему увагу приділено безпеці доступу до акаунта та захисту персональних даних. Загалом, реалізована функціональність повністю відповідає потребам сучасного користувача та забезпечує зручний досвід онлайн-бронювання.

ВИСНОВКИ

У рамках дипломної роботи було розроблено, протестовано та вдосконалено інформаційну систему бронювання залізничних та автобусних квитків, яка відповідає сучасним вимогам безпеки, продуктивності та зручності використання.

Метою дослідження було підвищення функціональних можливостей системи бронювання квитків шляхом розробки нових модулів та вдосконалення взаємодії з користувачем, що дозволяє покращити якість сервісу, зменшити час на оформлення замовлення та підвищити рівень автоматизації. Цю мету було досягнуто.

В процесі реалізації було виконано всі поставлені завдання:

1. Розглянуто й проаналізовано сучасні підходи до реалізації систем бронювання. У першому розділі представлено огляд архітектурних рішень, засобів авторизації, взаємодії з БД, а також приклади успішних реалізацій подібних систем.
2. Проведено вибір технологій та інструментів. Було обґрунтовано використання таких стеків: для клієнтської частини — React, для серверної — .NET Core та C#, для роботи з базою даних — SQL Server і Entity Framework. Для безпеки — JWT та ASP.NET Core Identity.
3. Розроблено повнофункціональний програмний продукт. Створено модулі реєстрації, авторизації, бронювання, перегляду розкладів, оплати та перегляду історії замовлень. Передбачено обробку помилок, перевірку доступності місць та зручну навігацію.
4. Здійснено тестування роботи системи. Проведено юніт- та інтеграційні тести, а також функціональне тестування з боку користувача. Система продемонструвала стабільну роботу під час запитів та коректну обробку типових сценаріїв.

У процесі виконання бакалаврської кваліфікаційної роботи було послідовно реалізовано всі ключові етапи, передбачені поставленими завданнями.

У першому розділі проведено глибокий аналіз предметної області — ринку онлайн-бронювання квитків. Розглянуто наявні сервіси, їхні сильні та слабкі сторони, що дозволило визначити чіткі вимоги до нової системи. Було обґрунтовано доцільність створення власного рішення з акцентом на зручність інтерфейсу, безпечну інтеграцію з платіжними сервісами та можливість збереження історії замовлень.

Другий розділ був присвячений архітектурному та технічному проєктуванню системи. Обрана клієнт-серверна модель із чітким розподілом між фронтендом (React) і бекендом (.NET Core + C#). Детально описано модульну структуру, методи забезпечення безпеки, підхід до побудови бази даних, інтеграцію з платіжними системами, а також застосування реактивного програмування для підвищення продуктивності інтерфейсу. Представлені UML-діаграми та обґрунтовано використання ООП та SOLID-принципів.

У третьому розділі здійснено безпосередню реалізацію програмного модуля, описано розгортання як фронтенд-, так і бекенд-частин, а також конфігурацію бази даних у середовищі Microsoft SQL Server. Проведено тестування, описано системні вимоги, налаштування середовища, взаємодію з API платіжних сервісів та покрокову інструкцію завантаження системи. Реалізований функціонал відповідає очікуванням користувача та технічним вимогам.

Загалом, виконання цих трьох етапів дозволило сформувати повноцінну, стабільну та безпечну систему онлайн-бронювання квитків, готову до впровадження та подальшого масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Albahari J., Albahari B. C# 8.0 and .NET Core 3.0: Modern Cross-Platform Development. Packt Publishing, 2019.
Посилання: https://www.packtpub.com/product/c-80-and-net-core-30-modern-cross-platform-development/9781788478120
2. Richter J. CLR via C#. Microsoft Press, 2018.
URL:https://www.microsoftpressstore.com/store/clr-via-c-sharp-9780735686558
3. Price M. Programming .NET Core. O'Reilly Media, 2020.
URL:https://www.oreilly.com/library/view/programmingnetcore/97814920/
4.Freeman A. Pro React 16. Apress, 2019.
URL: https://link.springer.com/book/10.1007/978-1-4842-4229-4
5. Bishop J. Entity Framework Core in Action. Manning Publications, 2020.
URL: https://www.manning.com/books/entity-framework-core-in-action
6. Allen J. ASP.NET Core Security: Securing modern web apps and APIs. Manning Publications, 2021.
URL:https://www.manning.com/books/asp-net-core-security
7. Price M. Programming .NET Core. O'Reilly Media, 2020.
https://www.oreilly.com/library/view/programming-net-core/9781492056812
8. Microsoft Docs. Secure ASP.NET Core Applications, 2023.
https://learn.microsoft.com/aspnet/core/security/
9.https://objectcomputing.com/resources/publications/mnb/2014/01/15/reactive-opendds-part-i?
10. Microsoft Learn. "Saving Data – EF Core." 2022.
https://learn.microsoft.com/en-us/ef/core/saving/
11. Sparks W. Understanding TLS 1.3 Encryption and Its Role in PCI DSS Compliance. Schellman & Company, LLC, 2023.
URL: https://www.schellman.com/blog/pci-compliance/tls-1.3-encryption-and-pci-dss-compliance

12. Microsoft Learn. Unit testing C# in .NET using dotnet test and xUnit. 2024. URL: https://learn.microsoft.com/en-us/dotnet/core/testing/unit-testing-csharp-with-xunit

13. Testing101. Які існують різні рівні тестування? 2022. URL-адреса: https://www.testing101.net/post/what-are-the-different-test-levels

14. Документація React. Знайомство з JSX та компонентами . URL-адреса: https://react.dev/l

15. Microsoft. Introduction to memory-optimized tables. Microsoft Learn, 2024. URL: https://learn.microsoft.com/sql/relational-databases/in-memory-oltp/introduction-to-memory-optimized-tables

16. Google Developers. Responsive Web Design Fundamentals. Google, 2024. URL: https://developers.google.com/web/fundamentals/design-and-ux/responsive

17.React Documentation. Introducing JSX and Virtual DOM. React Official Docs, 2024. URL: https://reactjs.org/docs/introducing-jsx.html

18.Esposito D. Programming ASP.NET Core, 7th Edition. Microsoft Press, 2023. URL: https://www.microsoftpressstore.com/store/programming-asp.net-core-9780137929335

19.Price J. Entity Framework Core in Action. Manning, 2022. URL: https://www.manning.com/books/entity-framework-core-in-action

20.А. А. Яровий, О. В. Сілагін, І. В. Богач. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп’ютерні науки» Вінниця : ВНТУ, 2023. 54 с.

ДОДАТОК А

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Програмний модуль бронювання залізничних квитків

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

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 1,03 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

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

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

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

Експертна комісія:

Яровий А.А., зав. каф. КН _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Малініч І.П.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Павленко О.П.

ДОДАТОК Б

ІНСТРУКЦІЯ КОРИСТУВАЧА

1. Відкрийте веб-додаток у браузері за вказаним посиланням.

2. На головній сторінці натисніть кнопку «Увійти» або «Зареєструватися», якщо ще не маєте облікового запису.

3. Після входу відкриється головна панель, де доступний пошук рейсів.

4. Введіть пункт відправлення, пункт призначення та оберіть дату поїздки, потім натисніть «Знайти».

5. Ознайомтеся з результатами пошуку та натисніть «Обрати» біля потрібного рейсу.

6. Оберіть місце у вагоні, натиснувши на нього. Натисніть кнопку «Вибрати», щоб підтвердити вибір.

7. Уведіть персональні дані пасажира та перейдіть до розділу «Оплата».

8. Оберіть спосіб оплати, введіть необхідні реквізити та натисніть «Сплатити».

9. Після успішної оплати система згенерує квиток, який можна переглянути в розділі «Мої замовлення».

10. Для виходу з облікового запису натисніть на аватар або меню користувача та виберіть пункт «Вийти».

ДОДАТОК В

ЛІСТИНГ ПРОГРАМНОГО КОДУ

Вміст файлу globalticket.client/App.js

```
import { useEffect, useState } from 'react';
import './App.css';
import { AuthProvider } from './providers/AuthProvider';
import { TicketProvider } from './providers/TicketProvider';
import Home from './components/home/Home'
import { ToastContainer } from 'react-toastify';
import MainProfile from './components/profile/main_profile/MainProfile';
import { Routes, Route} from 'react-router-dom';
import HomeMain from './components/home/HomeMain';
import Layout from './components/Layout';
import TicketsBlock from './components/ticket/TicketsBlock';
import HomeProfileBody from './components/profile/body/HomeProfileBody';
import Settings from './components/profile/body/settings/Settings';
import './i18n'
import SeatPicker from './components/ticket/seat-picker/SeatPicker';
import ClientDataBlock from './components/ticket/passagire-data/ClientDataBlock';
import ClientDataBlockBus from './components/ticket/passagire-data/ClientDataBlockBus';
import PaymentBlock from './components/payment/payment-block/PaymentBlock';
import TicketList from './components/profile/body/ticket-list/TicketList'
import TransportLayout from './components/transport/TransportLayout';
import i18next from 'i18next';
import moment from 'moment';
import HomeBus from './components/home/bus/HomeBus';
import HomeBusMain from './components/home/bus/HomeBusMain';
import BusSeatPicker from './components/ticket/seat-picker/BusSeatPicker';
import HomeAirline from './components/home/airline/HomeAirline';
import HomeAirlineMain from './components/home/airline/HomeAirlineMain';
import AuthGuard from './components/guards/auth-guard/AuthGuard';
import RouteGuard from './components/guards/route-guard/RouteGuard';
import { PayProvider } from './providers/PayProvider';
function App() {
```

```
const [startPoint, setStartPoint] = useState('')
const [endPoint, setEndPoint] = useState('')
 const setPoints = (start,end)=>{
    setStartPoint(start)
    setEndPoint(end)
 }
 useEffect(()=>{moment.locale(i18next.language)},[i18next.language])
return (
 <>
  <AuthProvider>
   <TicketProvider>
    <PayProvider>
     <Routes>
      <Route path='/' element={<Layout />}>
       <Route path='/' element={<Home startPoint={startPoint} endPoint={endPoint}/>}>
        <Route path='/' element={<HomeMain setPoints={setPoints}/>}/>
        <Route path='/train' element={<RouteGuard endpoint={'/'}><TransportLayout type={'train'}/></RouteGuard>}>
         <Route path='search' element={<TicketsBlock type={"TRAIN"}/>}/>
         <Route path='seat' element={<SeatPicker/>}/>
         <Route path='client' element={<ClientDataBlock/>}/>
        </Route>
       </Route>
       <Route path='/profile' element={<AuthGuard><MainProfile /></AuthGuard>}>
        <Route path='*' element={<HomeProfileBody/>}/>
        <Route path='ticket/train' element={<TicketList/>}/>
        <Route path='ticket/bus' element={<TicketList/>}/>
        <Route path='ticket/airplane' element={<TicketList/>}/>
        <Route path='settings' element={<Settings/>}/>
       </Route>
       <Route path='/home-bus' element={<HomeBus startPoint={startPoint} endPoint={endPoint}/>}>
        <Route path='' element={<HomeBusMain setPoints={setPoints}/>}/>
        <Route path='bus' element={<RouteGuard endpoint={'/home-bus'}><TransportLayout type="bus"/></RouteGuard>}>
         <Route path='search' element={<TicketsBlock type={"BUS"}/>}/>
         <Route path='seat' element={<BusSeatPicker/>}/>
         <Route path='client' element={<ClientDataBlockBus/>}/>
        </Route>
       </Route>
       <Route path='/home-airline' element={<HomeAirline/>}>
```

```
            <Route path='' element={<HomeAirlineMain/>}/>
          </Route>
          <Route path='/schedule' element={<><h1>!</h1></>}/>
          <Route path='/payment' element={<PaymentBlock/>}/>
        </Route>
        <Route path='*' elemnt={<h1>Not found</h1>}/>
      </Routes>
      <ToastContainer/>
    </PayProvider>
  </TicketProvider>
 </AuthProvider>
</>)
}

export default App;
```

Вміст файлу App.js

```
const express = require('express')
const cors = require('cors')
const crypto = require('crypto')
const sha1 = crypto.createHash('sha1')
const app = express();
const PORT=5007
app.use(express.json())
app.use(express.urlencoded({extended: true}))
app.use(cors({
  origin:['http://localhost:3000','http://localhost:3005'],
  methods:["GET","POST","PUT","DELETE"]
}))

var tickets = [
  {
    id:1,
    tarinId: 1,
    startTime: '10:30',
    startDate: '12.05.2024',
    endTime: '18:30',
    endDate: '1.05.2024',
```

```
  duration: '08:00',
  transportName: '090У',
  route: 'Київ-Львів',
  type:'',
  places: [
    {
      placeName: 'Плацкарт Економ',
      price: 199,
      numberOfPlaces: 200
    },
    {
      placeName: 'Сидаячий 1-й клас',
      price: 583,
      numberOfPlaces: 137
    },
  ]
},
{
  id:2,
  tarinId: 2,
  startTime: '11:30',
  startDate: '12.05.2024',
  endTime: '13:00',
  endDate: '12.05.2024',
  duration: '01:30',
  transportName: '125Е',
  route: 'Хмельницький-Рівне',
  type:'Експрес',
  places: [
    {
      placeName: 'Плацкарт Економ',
      price: 1827.78,
      numberOfPlaces:105
    }
  ]
},
{
  id:3,
```

```
    tarinId: 3,
    startTime: '09:00',
    startDate: '12.05.2024',
    endTime: '18:30',
    endDate: '12.05.2024',
    duration: '09:30',
    transportName: '180Т',
    route: 'Луцьк-Одеса',
    type:'',
    places: [
      {
        placeName: 'Сидаячий 1-й клас',
        price: 326,
        numberOfPlaces:238
      },
      {
        placeName: 'Сидаячий 2-й клас',
        price: 587,
        numberOfPlaces:178
      },
    ]
  },
]
var busTickets = [
  {
    id:1,
    tarinId: 1,
    startTime: '10:30',
    startDate: '12.05.2024',
    endTime: '18:30',
    endDate: '1.05.2024',
    duration: '08:00',
    transportName: 'ДАЛЕКОГО СПОЛУЧЕННЯ',
    route: '1631-Київ АС (центральний Зал. вокзал) - Ужгород АС-1',
    type:'Павлюк М.І.',
    places: [
      {
        placeName: '',
```

```
        price: 199,
        numberOfPlaces: 200
      },
    ]
  },
  {
    id:2,
    tarinId: 2,
    startTime: '11:30',
    startDate: '12.05.2024',
    endTime: '13:00',
    endDate: '12.05.2024',
    duration: '01:30',
    transportName: 'МЕРСЕДЕС-(25)',
    route: '503 1630 КИЇВ УЖГОРОД Львів - Ужгород',
    type:'Павлюк В.І.',
    places: [
      {
        placeName: '',
        price: 1827.78,
        numberOfPlaces:105
      }
    ]
  },
  {
    id:3,
    tarinId: 3,
    startTime: '09:00',
    startDate: '12.05.2024',
    endTime: '18:30',
    endDate: '12.05.2024',
    duration: '09:30',
    transportName: 'Mersedes Sprinter',
    route: '138-Харків-Ужгород - Львів-Ужгород',
    type:'Юхно М.О.',
    places: [
      {
        placeName: '',
```

```
            price: 326,
            numberOfPlaces:238
          },
        ]
    },
]
const ticketTest = [
    {
      "id": 3,
      "trainId":4,
      "departureStation": "Київ-Пасажирський",
      "arrivalStation": "Дніпро-Головний",
      "departureTime": "07:30:00",
      "arrivalTime": "17:00:00",
      "departureDate": "2024-05-14T00:00:00",
      "arrivalDate": "2024-05-14T00:00:00",
      "availablePlaces": 75,
      "totalPlaces": 75,
      "duration": "09:30:00",
      "trainType": 2,
      "trainLineName": "702К",
      "trainLineDescription": "Київ-Пасажирський - Дніпро-Головний",
      "wagonTypes": [2, 1, 0],
      "wagons": [
        {
          "number": 4,
          "wagonType": 2,
          "seats": null,
          "availableSeats": 36,
        },
        {
          "number": 5,
          "wagonType": 1,
          "seats": null,
          "availableSeats": 36
        },
        {
          "number": 6,
```

```
            "wagonType": 0,
            "seats": null,
            "availableSeats": 36
          }
        ],
        "firstPrices":[
            500.26,
            200.00,
            125.50
        ]
      }
]
const token = '63rUqczHuwFzdJ7c1d8OKvuRhlvG9HEl'
app.post('/api/Users/login',(req,res)=>{
   console.log("login-success!")
   res.status(200).json(
      {
         message: 'Welcome!',
         token: token
      }
   )
});
app.post('/api/Users/log-in-fail',(req,res)=>{
   console.log("login-fail!")
   res.status(201).json({
      message:'Error, incorrect password or email!'
   });
});
app.post('/api/Users/register',(req,res)=>{
   console.log("signup-success!")
   res.status(200).json(
      {
         message: 'Welcome!',
         token: token
      }
   )
});
app.post('/api/Users/sign-up-fail',(req,res)=>{
```

```
    console.log("signup-fail!")
    res.status(201).json({
        message:'Error, incorrect password or email!'
    });
});
app.get('/api/Users/detail',(req,res)=>{
    console.log("detail!")
    res.status(200).json({
        userId: 1,
        firstName:'Test',
        lastName:'Test',
        email:'test@test.net',
        phoneNumber:'1111111'
    });
});
app.get('/api/Home/post-feedback',(req,res)=>{
    console.log("comment!")
    res.status(200).json({
        message: 'Comment!'
    })
})
app.post('/api/Users/logout',(req,res)=>{
    console.log("logout!")
    res.status(200).json({
        message: 'Logout!'
    })
})
app.post('/api/Users/change-password-success',(req,res)=>{
    console.log("changePassword!")
    res.status(200).json({
        message: 'Change!'
    })
})
app.post('/api/Users/change-password-fail',()=>{
    console.log("changePassword!")
    res.status(403).json({
        message: 'Fail change!'
    })
```

```
})
app.post('/api/Users/detail-set-success',(req,res)=>{
   console.log("setDEATIL!")
   res.status(200).json({
      message: 'DETAIL!'
   })
})
app.post('/api/Users/detail-set-fail',(req,res)=>{
   console.log("setDEATIL!")
   res.status(403).json({
      message: 'ERROR DETAIL!'
   })
})
app.get('/api/Home/find-appropriate-lines',(req,res)=>{
   console.log("tickets Train")
   res.status(200).json(ticketTest)
});
app.get('/api/Home/bus',(req,res)=>{
   console.log("tickets BUS")
   res.status(200).json(busTickets)
})
app.get('/api/tickets/train/carriage/placesE',(req,res)=>{

})
app.get('/api/tickets/train/carriage/placesNE',(req,res)=>{

})
app.get('/api/tickets/buss',(req,res)=>{

});
app.get('/api/tickets/airplane',(req,res)=>{

});
const routes = [
   {id: 1, route: 'Київ-Львів'},
   {id: 2, route: 'Київ-Харків'},
   {id: 3, route: 'Львів-Одеса'},
   {id: 4, route: 'Харків-Дніпро'},
```

```
  {id: 5, route: 'Одеса-Київ'},
  {id: 6, route: 'Дніпро-Львів'},
  {id: 7, route: 'Львів-Хмельницький'},
  {id: 8, route: 'Хмельницький-Запоріжжя'},
  {id: 9, route: 'Запоріжжя-Херсон'},
  {id: 10, route: 'Херсон-Суми'}];
app.get('/api/Admin/get-routes',(req,res)=>{
  console.log("ADMIN ROUTES")
  res.status(200).json(routes)
})
const route = [
  {
    point: "Хмельницький",
    arrivalTime: "09:00",
    departureTime: "10:30",
    duration: "01:30"
  },
  {
    point: "Вінниця",
    arrivalTime: "11:30",
    departureTime: "12:45",
    duration: "01:15"
  },
  {
    point: "Київ",
    arrivalTime: "14:00",
    departureTime: "15:30",
    duration: "01:30"
  },
  {
    point: "Львів",
    arrivalTime: "16:45",
    departureTime: "18:00",
    duration: "01:15"
  },
  {
    point: "Одеса",
    arrivalTime: "20:30",
```

```
      departureTime: "22:00",
      duration: "01:30"
    }
]
app.get('/api/Admin/get-route-details',(req,res)=>{
   console.log("ADMIN ROUTE DETAILS")
   res.status(200).json(route)
})
//Транспорт
const trains = [
   {id: 1, name: '123A', route: 'Київ-Львів'},
   {id: 2, name: '456B', route: 'Київ-Харків'},
   {id: 3, name: '789C', route: 'Львів-Одеса'},
   {id: 4, name: '321D', route: 'Харків-Дніпро'},
   {id: 5, name: '654E', route: 'Одеса-Київ'},
   {id: 6, name: '987F', route: 'Дніпро-Львів'},
   {id: 7, name: '210G', route: 'Львів-Хмельницький'},
   {id: 8, name: '543H', route: 'Хмельницький-Запоріжжя'},
   {id: 9, name: '876I', route: 'Запоріжжя-Херсон'},
   {id: 10, name: '109J', route: 'Херсон-Суми'}
];
app.get('/api/Admin/get-trains',(req,res)=>{
   console.log("ADMIN TRAINS")
   res.status(200).json(trains)
})

const buses = [
   {id: 1, name: 'ПреміумЕкспрес', route: 'Київ-Львів'},
   {id: 2, name: 'Український тур', route: 'Київ-Харків'},
   {id: 3, name: 'Львівський експрес', route: 'Львів-Одеса'},
   {id: 4, name: 'Дніпровський лайнер', route: 'Харків-Дніпро'},
   {id: 5, name: 'Одеський комфорт', route: 'Одеса-Київ'},
   {id: 6, name: 'Дельта експрес', route: 'Дніпро-Львів'},
   {id: 7, name: 'Львівський швидкий', route: 'Львів-Хмельницький'},
   {id: 8, name: 'Хмельницька зірка', route: 'Хмельницький-Запоріжжя'},
   {id: 9, name: 'Запорізький кур\'єр', route: 'Запоріжжя-Херсон'},
   {id: 10, name: 'Сумський комфорт', route: 'Херсон-Суми'}
];
```

```
app.get('/api/Admin/get-buses',(req,res)=>{
   console.log("ADMIN BUSES")
   res.status(200).json(buses)
})

const bus = {
   name: 'ПреміумЕкспрес',
   price: 200,
   seats: 28,
   route: 'Київ-Львів'
}
app.get('/api/Admin/get-bus-details',(req,res)=>{
   console.log("ADMIN BUS DETAILS")
   res.status(200).json(bus)
})
const train = {
   name: '123A',
   type: '1',
   route: 'Київ-Львів',
   wagons: [
      {
         type:1,
         seatsCount: 36,
         price: 500.56
      },
      {
         type:1,
         seatsCount: 36,
         price: 500.56
      },
      {
         type:1,
         seatsCount: 36,
         price: 500.56
      },
      {
         type:3,
```

```
            seatsCount: 18,
            price: 990.56
        },
        {
            type:3,
            seatsCount: 18,
            price: 990.56
        },
    ]
}
app.get('/api/Admin/get-train-details',(req,res)=>{
    console.log("ADMIN TRAIN DETAILS")
    res.status(200).json(train)
})
```

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

«Програмний модуль бронювання залізничних квитків»

Виконав: студент 4 курсу, групи 1КН-21б
Спеціальності 122 – Комп'ютерні науки
(шифр і назва напрямку підготовки, спеціальності)
Павленко О.П.
(прізвище та ініціали)
Керівник: асистент каф. КН
Малініч І. П.
(прізвище та ініціали)

«3» червня 2025 р.

Вінниця ВНТУ – 2025 рік

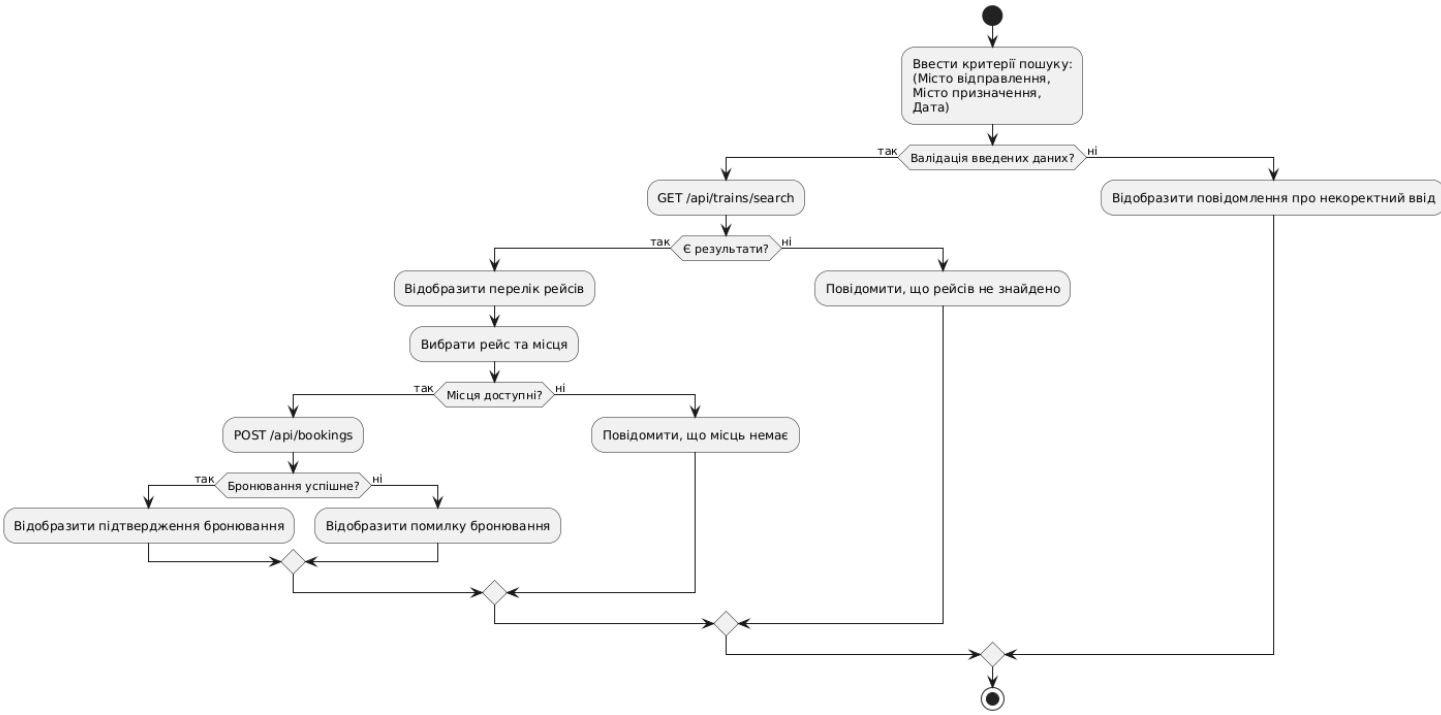


Рисунок Г.1 – Діаграма діяльності для пошуку рейсу та бронювання

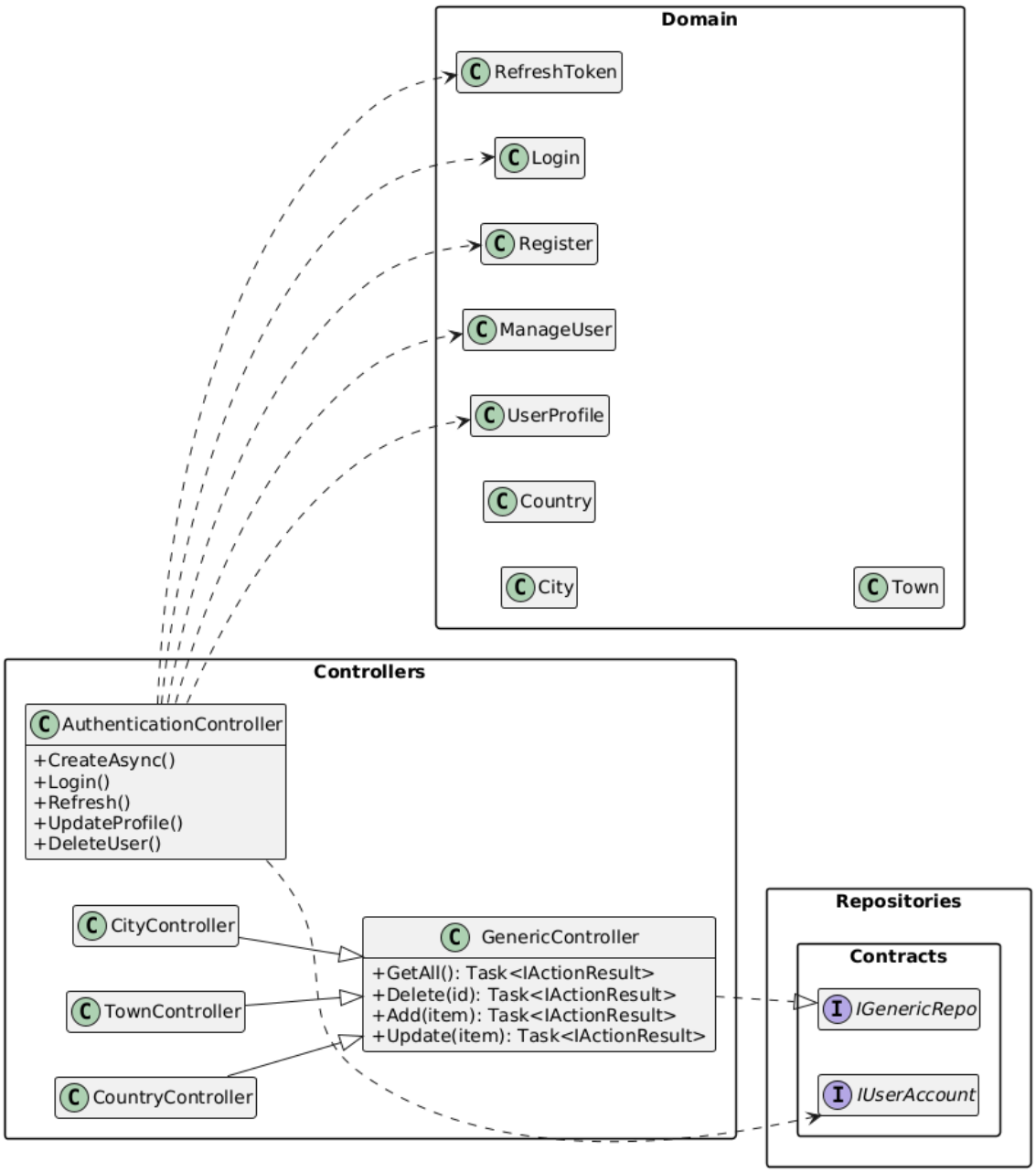


Рисунок Г.2 – Класова діаграма модуля бронювання квитків

Рисунок Г.3 – діаграма реєстрації

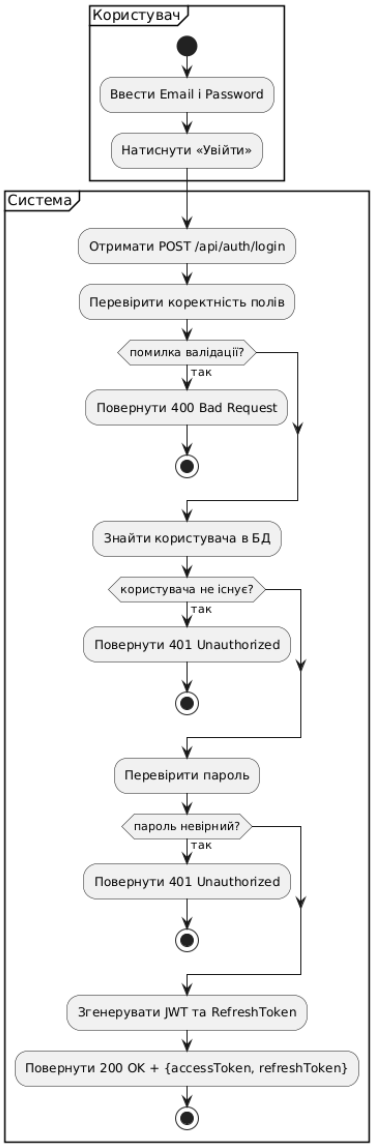


Рисунок Г.4 – діаграма авторизації

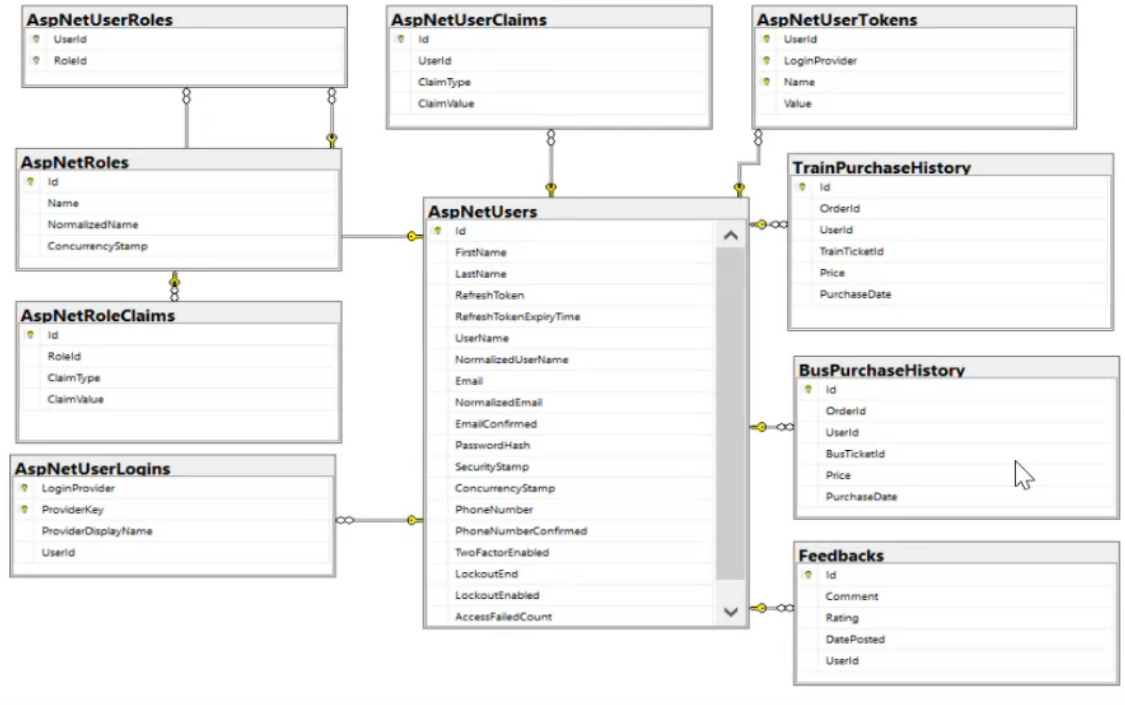


Рисунок Г.5 – User Tables

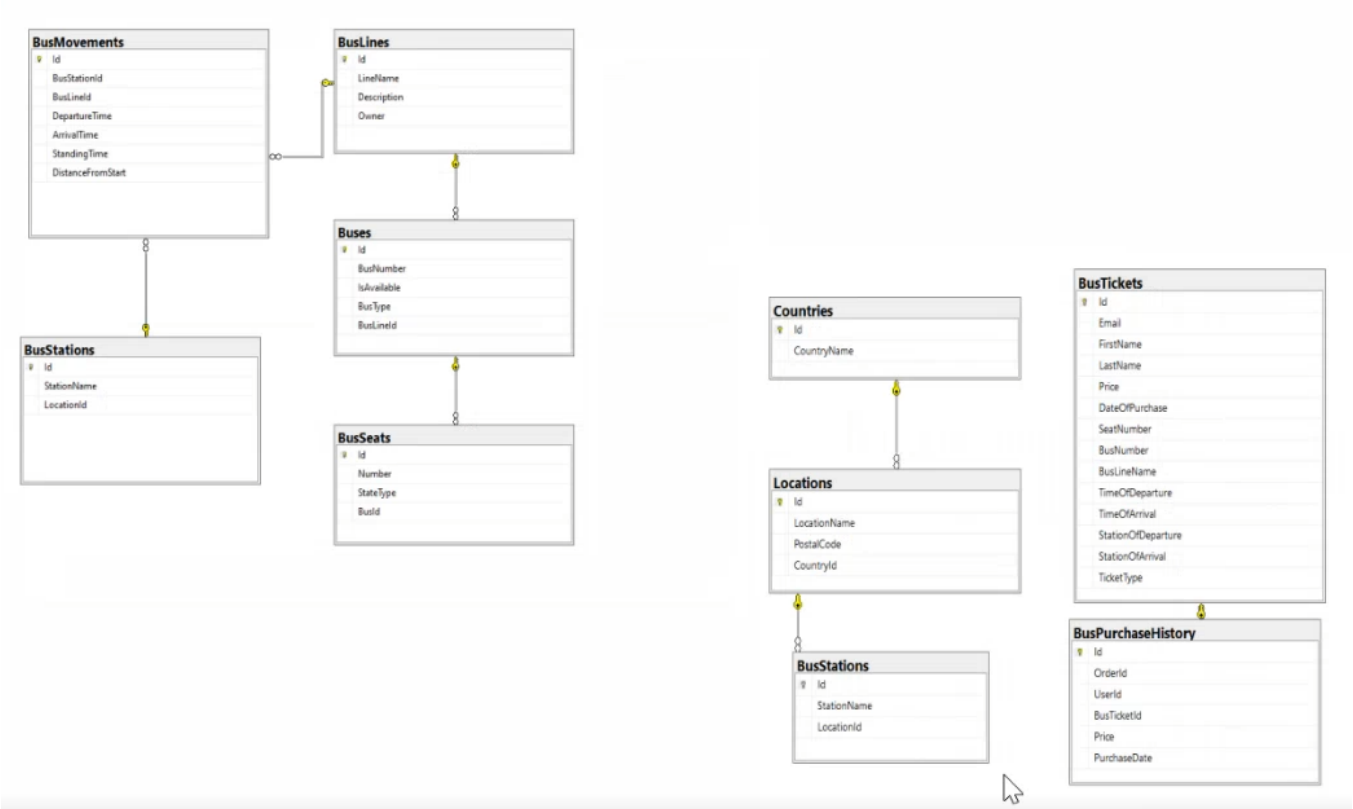


Рисунок Г.6 – Bus Tables

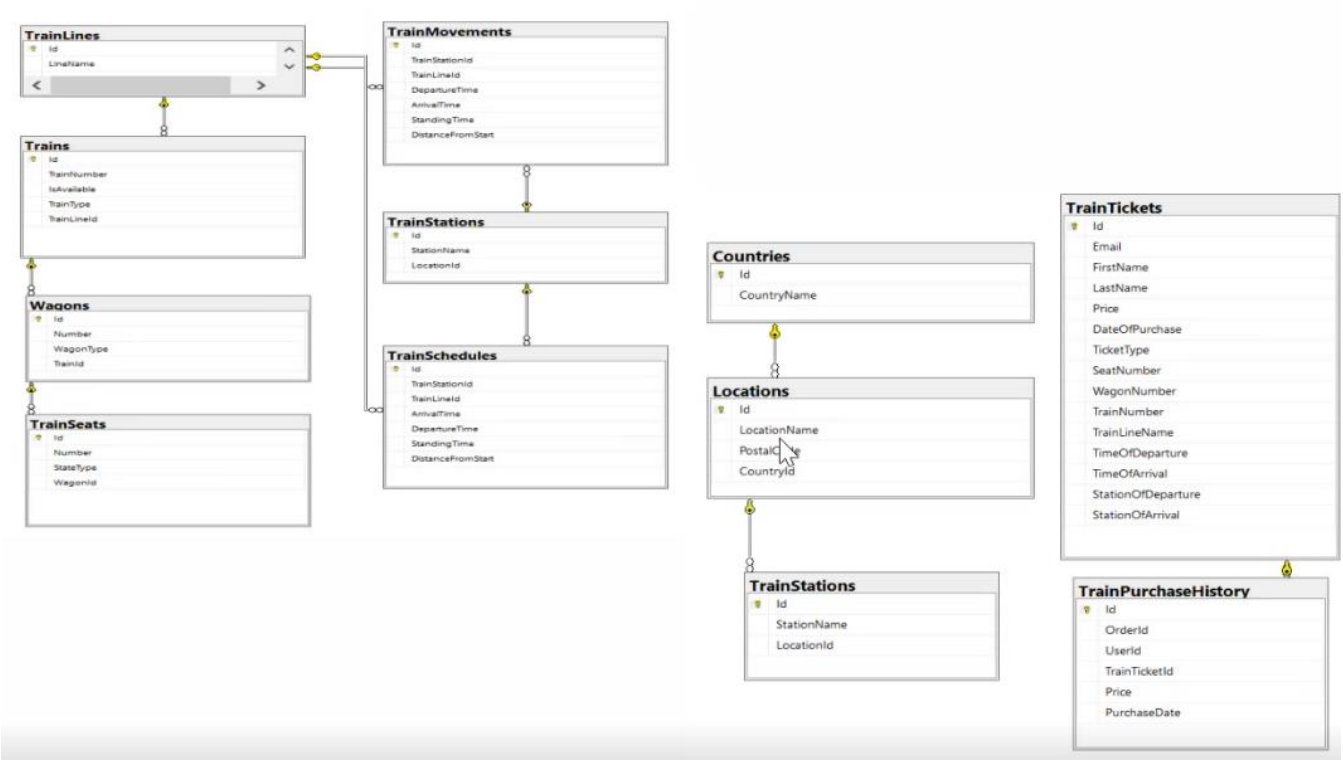


Рисунок Г.7 – Train Tables

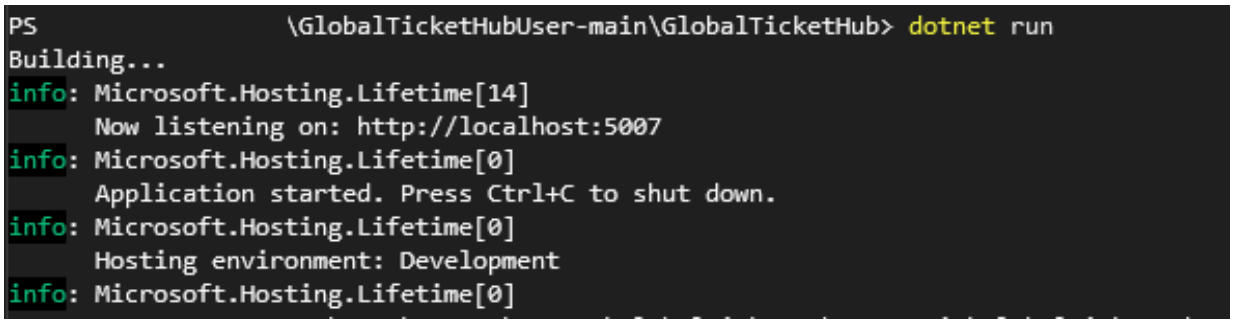


Рисунок Г.8 – Запуск сервера

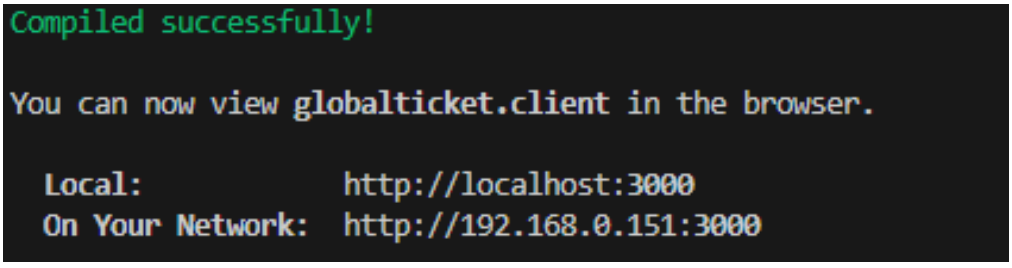


Рисунок Г.9 – Запуск клієнтської частини

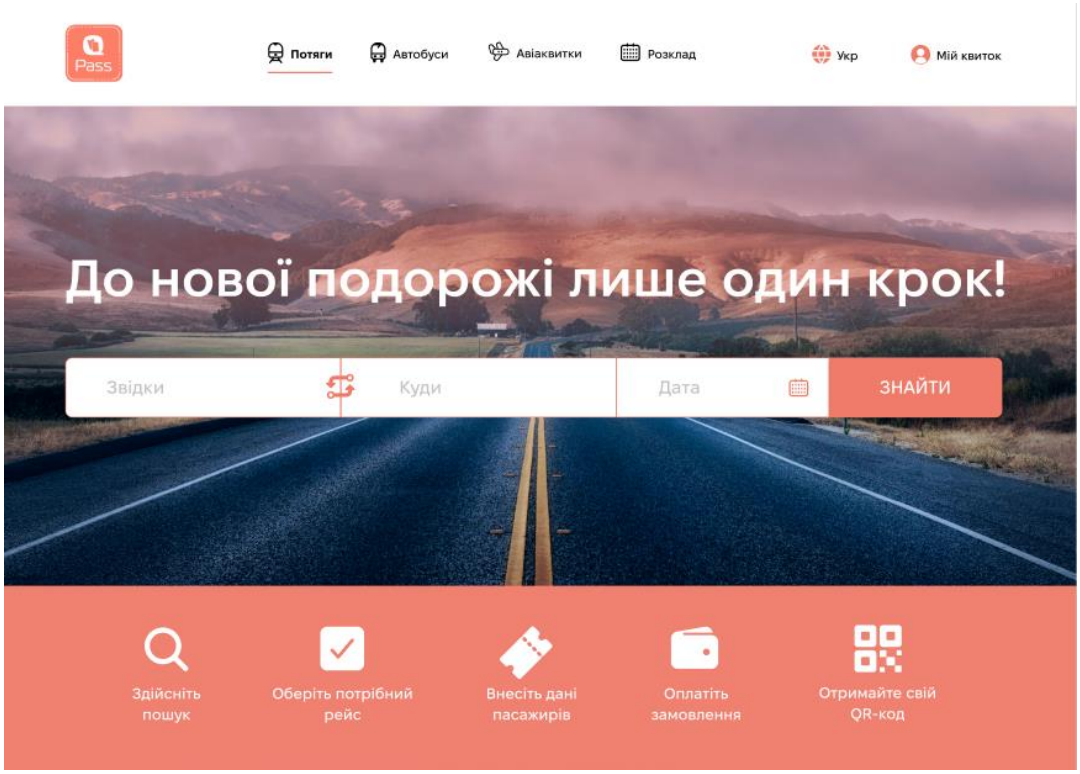


Рисунок Г.10 – Головний екран

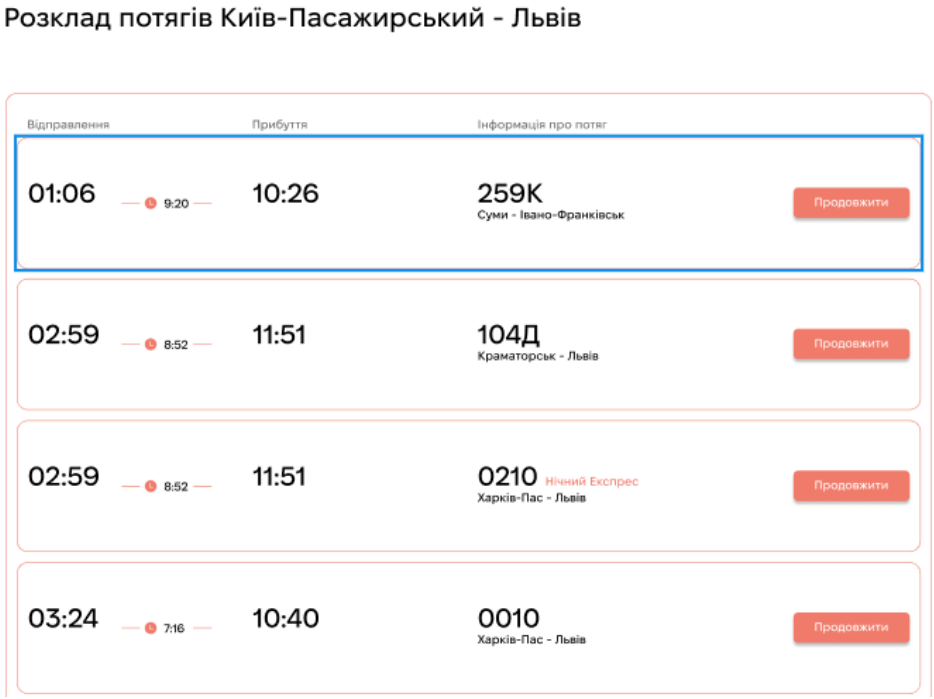


Рисунок Г.11 – Результати пошуку

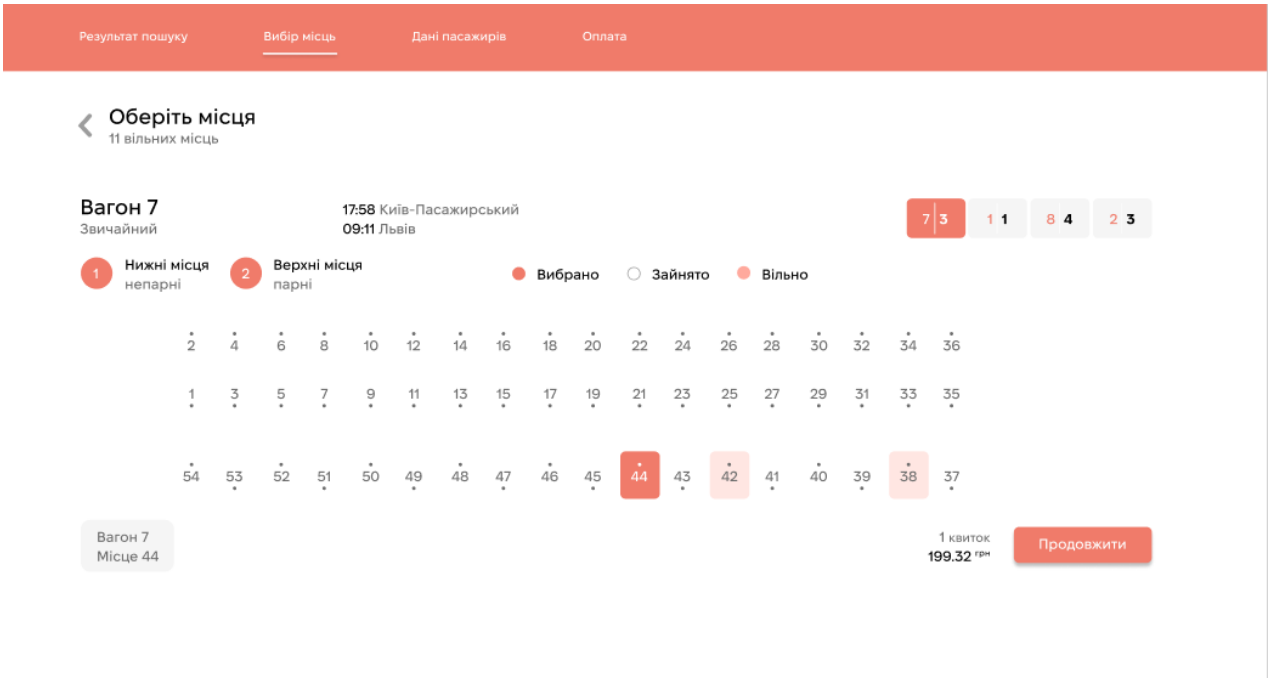


Рисунок Г.12 – Вибір місця

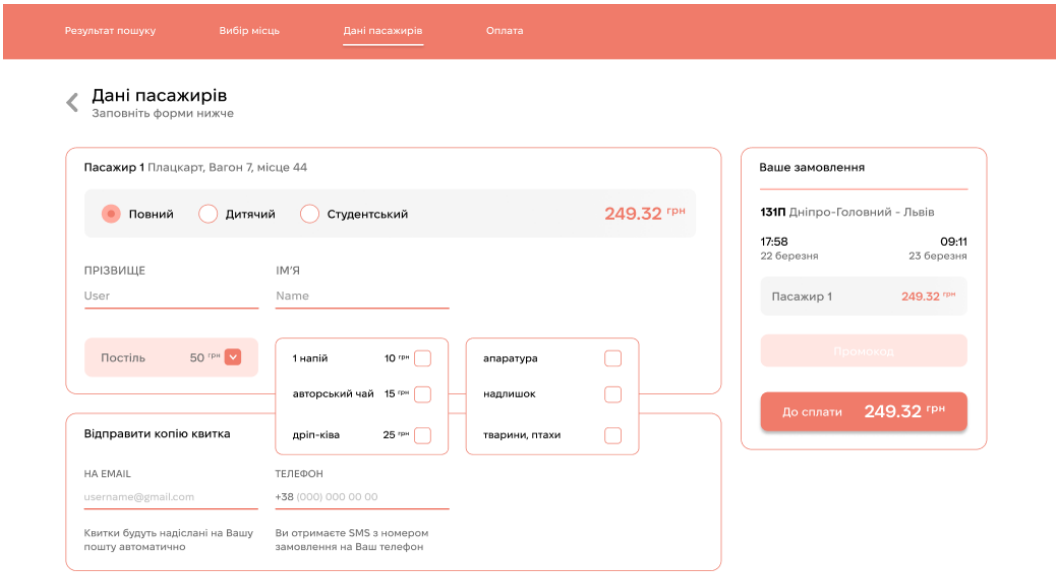


Рисунок Г.13 – Дані пасажира

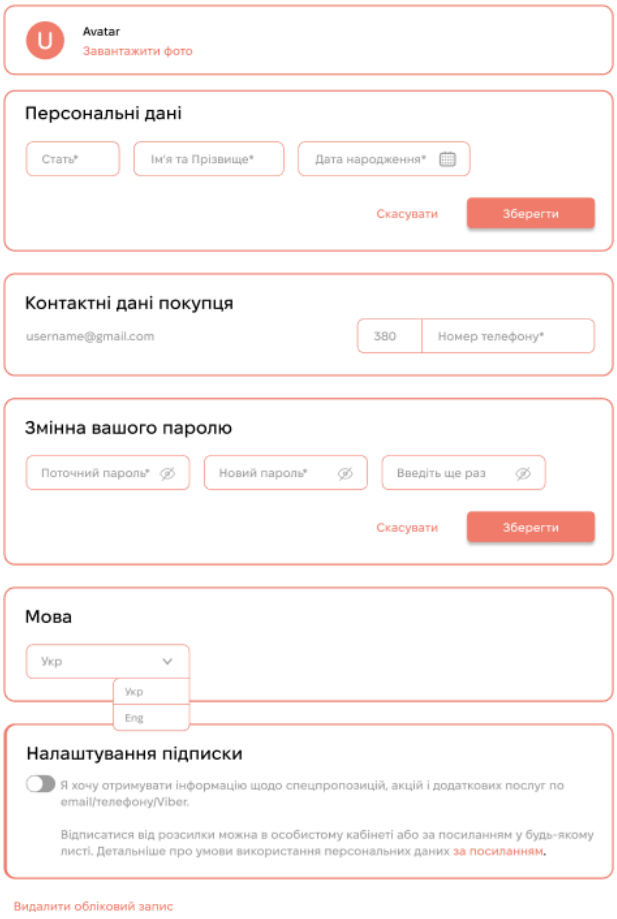


Рисунок Г.14 – Налаштування акаунта