Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії і управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти перший (бакалаврський)

Онлайн-платформа для бронювання автобусних квитків

(тема)

Виконав:
здобувач 4 року навчання,
групи КІУКІ-21-4
Іван ЛЕБІДЬ
(власне ім'я, прізвище)

Спеціальність
123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма
Комп'ютерна інженерія
(повна назва освітньої програми)

Керівник: ас. Віталій СІТНІКОВ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри ЕОМ _____ Андрій КОВАЛЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Лебідю Івану Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Онлайн-платформа для бронювання автобусних квитків

затверджена наказом по університету від “ 26 ” травня 2025 р. № 424 Ст

2. Термін подання здобувачем роботи до екзаменаційної комісії 17 червня 2025 р.

3. Вхідні дані до роботи Java
Spring Framework
Spring MVC
JSP
JSTL
MySQL

4. Перелік питань, що потрібно опрацювати у роботі
Обґрунтування вибору технологічного стеку для веб-застосунку
Проєктування структури бази даних та моделювання сутностей
Розробка REST API для взаємодії з клієнтською частиною
Створення та налаштування реляційної бази даних у MySQL
Розробка адаптивного користувацького інтерфейсу (UI) засобами JSP та Bootstrap
Формування супровідної технічної та пояснювальної документації

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій 16 ілюстрацій

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	27.05.2025-30.05.2025	
2	Вибір технології розробки та інструментальних	31.05.2025-02.06.2025	
	засобів		
3	Розробка алгоритмічного забезпечення	03.06.2025-05.06.2025	
4	Розробка та відлагодження програмного	03.06.2025-05.06.2025	
	забезпечення	06.06.2025-09.06.2025	
5	Оформлення матеріалів кваліфікаційної роботи		
6	Подання кваліфікаційної роботи керівникові та	10.06.2025-11.06.2025	
	її попередній захист	12.06.2025-13.06.2025	
7	Подання кваліфікаційної роботи на рецензу-		
	вання	14.06.2025-16.06.2025	

Дата видачі завдання " 26 " травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Віталій СІТНІКОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 71 с., 32 рис., 0 табл., 2 дод., 17 джерел.

ОНЛАЙН-БРОНЮВАННЯ, ВЕБ-ЗАСТОСУНОК, SPRING BOOT, SPRING SECURITY, JSP/JSTL, MYSQL, REST API.

Метою кваліфікаційної роботи є розробка сучасної онлайн-платформи для бронювання автобусних квитків, яка забезпечує автоматизований процес пошуку, бронювання та оплати квитків на міжміські рейси. Веб-застосунок реалізовано на основі технологій Java Spring Boot із використанням Spring Security для захисту, JSP/JSTL та Bootstrap для формування адаптивного інтерфейсу, а також реляційної бази даних MySQL для зберігання інформації про користувачів, рейси та замовлення. У роботі розглянуто проектування архітектури застосунку, структуру бази даних, розробку REST API. Система передбачає окремі функціональні модулі для адміністраторів та користувачів, інтеграцію механізмів автентифікації, ведення історії замовлень і безпечну обробку персональних даних. Запропоноване рішення відзначається гнучкістю, масштабованістю та відповідає сучасним вимогам до безпеки та зручності використання. Отриманий веб-застосунок може використовуватись як основа для подальшого розвитку та впровадження в реальні умови перевізників.

ABSTRACT

Bachelor’s thesis: 71 pages, 32 figures, 0 tables, 2 appendices, 17 sources.

ONLINE BOOKING, WEB APPLICATION, SPRING BOOT, SPRING SECURITY, JSP/JSTL, MYSQL, REST API.

The aim of this bachelor’s thesis is to develop a modern online platform for booking bus tickets, which provides an automated process for searching, booking, and paying for intercity bus tickets. The web application is implemented using Java Spring Boot technologies, with Spring Security for protection, JSP/JSTL and Bootstrap for an adaptive interface, and a MySQL relational database for storing information about users, routes, and bookings. The thesis covers the design of the application architecture, the structure of the database, and the development of the REST API. The system includes separate functional modules for administrators and users, integration of authentication mechanisms, order history tracking, and secure processing of personal data. The proposed solution is characterized by flexibility, scalability, and compliance with modern requirements for security and ease of use. The resulting web application can serve as a foundation for further development and implementation in real-world transportation companies.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

AOP – Aspect Oriented Programming (аспектно-орієнтоване програмування)

ACID – Atomicity, Consistency, Isolation, Durability (властивості транзакцій у базах даних)

CRM – Customer Relationship Management (система управління взаємовідносинами з клієнтами)

HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту)

HTML – HyperText Markup Language (мова розмітки гіпертексту)

IoC – Inversion of Control (інверсія управління)

JPA – Java Persistence API (інтерфейс збереження Java-об’єктів)

JPQL – Java Persistence Query Language (мова запитів JPA)

JSP – JavaServer Pages (технологія для створення динамічних веб-сторінок)

JSTL – JavaServer Pages Standard Tag Library (стандартна бібліотека тегів для JSP)

JSON – JavaScript Object Notation (формат обміну даними)

MVC – Model-View-Controller (архітектурний шаблон проєктування)

REST API – Representational State Transfer Application Programming Interface (інтерфейс прикладного програмування REST)

SPA – Single Page Application (односторінковий застосунок)

UX/UI – User Experience/User Interface (досвід користувача/інтерфейс користувача)

ВСТУП

Сьогоднішній розвиток цифрових технологій кардинально трансформує підходи до організації повсякденних процесів, зокрема в сфері транспортних перевезень. Однією з найбільш актуальних задач є забезпечення швидкого, надійного та зручного доступу до послуг пасажирських перевезень для широкого кола користувачів. З огляду на активний розвиток інформаційних систем, онлайн-сервіси бронювання квитків стають невід'ємною складовою інфраструктури сучасного транспорту, дозволяючи пасажирам ефективно планувати свої подорожі та уникати черг у касах.

Проте, незважаючи на появу численних рішень, більшість наявних онлайн-платформ для бронювання автобусних квитків мають певні обмеження, пов'язані з функціональністю, зручністю використання, відсутністю адаптивного інтерфейсу чи сучасної інтеграції з платіжними системами. Це створює передумови для подальшого вдосконалення подібних систем, використовуючи інноваційні технології та сучасні підходи до архітектури веб-застосунків.

Дана кваліфійкаційна робота присвячена аналізу, проектуванню та розробці інформаційної системи для бронювання автобусних квитків із застосуванням сучасних засобів веб-розробки. Особлива увага приділяється вибору оптимального технологічного стеку – від інтерфейсних рішень до бекенд-технологій і засобів забезпечення безпеки. Робота містить ґрунтовний аналіз ринку онлайн-сервісів, огляд ключових проблем автоматизації пасажирських перевезень, а також поетапний опис проектування, реалізації та впровадження веб-застосунку. Запропонована система дозволяє автоматизувати процеси бронювання, підвищити рівень зручності для користувачів та впровадити сучасні методи оплати й управління даними.

Результати роботи можуть бути корисними для підприємств

транспортної галузі, розробників веб-систем, а також для всіх, хто цікавиться питаннями цифровізації сервісів та впровадження новітніх технологій у реальний сектор економіки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз ринку онлайн-систем бронювання квитків

Упродовж останнього десятиліття цифровізація транспортної галузі стала невід’ємною частиною інфраструктурного розвитку багатьох країн. Онлайн-системи бронювання квитків поступово витісняють традиційні каси, надаючи пасажирам можливість швидко, зручно та безконтактно планувати поїздки. Особливо це актуально для автобусних перевезень, що відіграють ключову роль у внутрішньому сполученні як між містами, так і в межах окремих регіонів.

В Україні за останні роки спостерігається зростання кількості онлайн-платформ, що надають послуги з бронювання квитків, зокрема такі сервіси, як Busfor, Tickets.ua, GoFo і приватні інтерфейси автоперевізників. Незважаючи на це, більшість із них мають обмежений функціонал або не інтегруються з локальними маршрутами менш масштабних операторів. Деякі сервіси орієнтовані лише на продаж квитків, не надаючи інструментів для управління рейсами або збору статистики для перевізників.

У процесі розробки системи бронювання автобусних квитків було проведено аналіз ринку, зокрема, враховано досвід таких сервісів, як Busfor, Tickets.ua, GoFo, а також приватних інтерфейсів автоперевізників. Більшість із них мають обмежений функціонал або не інтегруються з локальними маршрутами менш масштабних операторів [1, 2].

У контексті цифрової трансформації транспортної системи існує потреба у створенні доступної, масштабованої та функціонально повноцінної веб-платформи, що забезпечить не лише пошук і купівлю квитків, але й підтримку динамічного управління рейсами, верифікацію пасажирів, а також можливість інтеграції з платіжними сервісами та іншими IT-рішеннями. У якості ілюстрації поточного стану ринку онлайн-сервісів для бронювання

автобусних квитків доцільно навести приклад інтерфейсу популярної платформи (рис. 1.1).

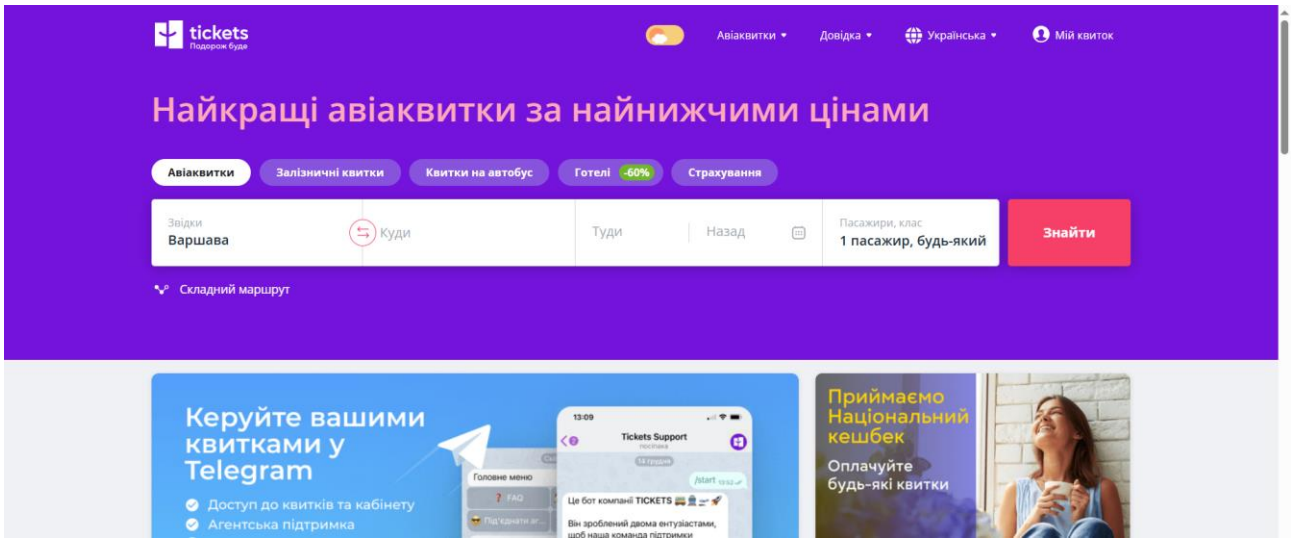

Рисунок 1.1 – Головна сторінка популярного онлайн-сервісу для бронювання автобусних квитків

Незважаючи на активний розвиток цифрових сервісів у сфері залізничних та авіаційних перевезень, автобусні маршрути й досі часто залишаються поза увагою високотехнологічних інновацій. У багатьох регіонах процес бронювання автобусних квитків здійснюється застарілими способами – телефоном або безпосередньо в автостанціях, що значно ускладнює доступ пасажирів до актуальної інформації про розклад та наявність місць, а також створює ризик помилок і дублювання бронювань.

Однією з суттєвих проблем є відсутність централізованих електронних систем для управління автобусними рейсами та контролю їхньої заповнюваності. Це призводить до неефективного використання ресурсів перевізника, втрати доходу через неповне завантаження транспорту і унеможливлює своєчасне інформування пасажирів про зміни у розкладі чи появу додаткових рейсів. Водночас пасажири залишаються без зручних інструментів для перегляду розкладу, бронювання та оплати квитків онлайн.

Крім того, більшість існуючих платформ або зовсім не інтегруються з

сучасними платіжними сервісами, або ж мають обмежені можливості для підключення до CRM та аналітичних систем. Це гальмує розвиток повноцінної цифрової екосистеми перевізника, ускладнює автоматизацію обліку та прогнозування попиту. Окремо стоять питання безпеки обробки даних, автентифікації користувачів та забезпечення стійкої роботи системи при великій кількості одночасних запитів.

Усі ці виклики підкреслюють необхідність створення сучасної веб-платформи для бронювання автобусних квитків, яка відповідатиме вимогам безпеки, автоматизації та інтеграції, а також забезпечуватиме високий рівень сервісу для як пасажирів, так і перевізників.

1.2 Проблематика автоматизації автобусних перевезень

Автоматизація автобусних перевезень в Україні залишається одним із найактуальніших викликів для транспортної галузі. Попри поступовий перехід авіаційних і залізничних перевізників до сучасних цифрових сервісів, автобусний сегмент досі часто використовує застарілі схеми організації роботи, що суттєво обмежує ефективність системи. У багатьох перевізників відсутні централізовані електронні платформи для управління маршрутами, продажем квитків та обліком пасажиропотоку, а процес бронювання найчастіше відбувається телефоном або безпосередньо на автостанціях.

Одна з основних проблем – це фрагментованість інформаційного простору. Пасажири змушені шукати розклади на різних сайтах, у соціальних мережах або через телефонні довідки, що ускладнює швидкий доступ до актуальної інформації про маршрути, вартість квитків та наявність місць. Через це зростає ризик дублювання рейсів, переповнення автобусів чи, навпаки, недозавантаження транспортних засобів, що призводить до фінансових втрат перевізників.

Ще однією проблемою є відсутність уніфікованої системи бронювання та продажу квитків. Це не дозволяє пасажирам зручно планувати свої

поїздки, а перевізникам – оперативно управляти завантаженням, аналізувати попит і формувати оптимальні графіки руху. Окремо слід відзначити й низький рівень інтеграції із сучасними платіжними системами: у багатьох випадках онлайн-оплата або недоступна, або реалізована у вигляді сторонніх сервісів, що знижує довіру користувачів та ускладнює процес обробки транзакцій.

Безпека даних і захист персональної інформації також є недостатньо вирішеними питаннями. Відсутність стандартів автентифікації, застарілі методи зберігання даних і нестача інструментів для моніторингу операцій створюють додаткові ризики для користувачів та перевізників.

У підсумку, проблематика автоматизації автобусних перевезень охоплює питання фрагментованості інформації, низької цифрової зрілості перевізників, недостатньої інтеграції із сучасними платіжними та аналітичними інструментами, а також відсутність зручних і безпечних платформ для управління бронюванням квитків і взаємодії з клієнтами. Вирішення цих проблем є передумовою для підвищення якості транспортних послуг і створення сучасної цифрової екосистеми пасажирських перевезень.

1.3 Мета і завдання кваліфікаційної роботи

Головною метою даної кваліфікаційної роботи є створення сучасного веб-застосунку, який забезпечить автоматизований процес бронювання автобусних квитків у зручній для користувача формі. Реалізація такого інструменту дозволить мінімізувати втрати часу при купівлі квитків, спростити доступ до інформації про рейси, а також покращити взаємодію між пасажирами та перевізниками. Особлива увага приділяється побудові інтуїтивно зрозумілого інтерфейсу, розробці гнучкої архітектури програмного забезпечення, що передбачає розмежування прав доступу між адміністративною частиною та користувацьким функціоналом, а також забезпеченню безпечної обробки особистих та платіжних даних. У межах

роботи передбачається вивчення існуючих аналогів, проєктування технічної структури застосунку, реалізація основних модулів для пошуку рейсів, бронювання місць та обробки замовлень, а також впровадження механізмів автентифікації, REST API і подальше тестування системи на предмет її працездатності, зручності та ефективності.

2 АНАЛІЗ І ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЙ

2.1 Аналіз технологій Front-end розробки

Ефективна реалізація клієнтської частини веб-застосунку є критично важливою складовою загального користувацького досвіду. У рамках розробки системи бронювання автобусних квитків було здійснено аналіз доступних засобів для створення інтерфейсу, що відповідає сучасним вимогам адаптивності, інтерактивності та зручності навігації. Під час вибору технологій перевагу надано зв’язці JSP та JSTL як перевіреному підходу у Java-орієнтованих рішеннях, а також CSS-фреймворку Bootstrap для забезпечення адаптивного дизайну сторінок.

Технологія JavaServer Pages (JSP) забезпечує динамічне формування HTML-контенту на стороні сервера, що є особливо доцільним у випадках, коли веб-застосунок тісно інтегрований із Java-бекендом. Її поєднання з бібліотекою тегів JSTL дозволяє реалізовувати логіку виводу даних без надмірного змішування Java-коду з HTML-розміткою. На відміну від сучасних SPA-фреймворків (React, Angular або Vue.js), JSP не потребує складної клієнтської збірки або налаштування окремого API-клієнта, що спрощує структуру проєкту та прискорює розгортання, особливо у випадку монолітної архітектури[3]. Водночас варто зазначити, що хоча JSP дещо поступається за динамічністю новим JavaScript-фреймворкам, для інформаційних систем з переважно табличними представленнями даних і формами введення вона залишається ефективним і стабільним рішенням.

Ще одним ключовим компонентом клієнтської частини є фреймворк Bootstrap, який дозволив сформувати однорідний візуальний стиль та забезпечити адаптивність інтерфейсу до різних типів пристроїв без необхідності написання великого обсягу CSS-коду вручну. Завдяки використанню готових класів та компонентів Bootstrap, таких як сіткова

система, форми, кнопки, модальні вікна та навігаційні елементи, вдалося суттєво скоротити час розробки. Адаптивність платформи є надзвичайно важливою, оскільки значна частина цільової аудиторії користується мобільними пристроями, і саме на цих екранах зручність навігації часто визначає бажання скористатися послугою повторно.

Таким чином, аналіз технологій для реалізації інтерфейсу користувача засвідчив доцільність використання JSP та JSTL у поєднанні з Bootstrap, що дозволяє забезпечити ефективну візуалізацію даних, високу швидкість розгортання та відповідність сучасним вимогам до зручності й доступності веб-сервісів. Це рішення є оптимальним з точки зору підтримки, масштабування та технічної сумісності з обраним серверним стеком.

Наведена на рис. 2.1 схема ілюструє, як поєднання JSP, JSTL та Bootstrap дозволяє організувати ефективний процес формування та відображення інтерфейсу користувача, забезпечуючи динамічну генерацію сторінок, адаптивний дизайн і сучасний користувацький досвід.

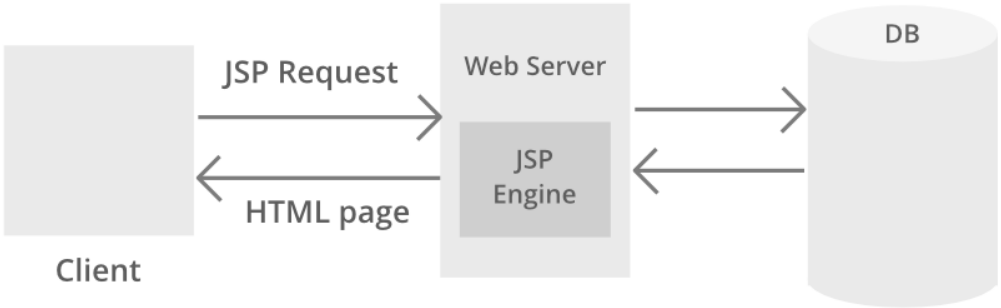

Рисунок 2.1 – Схема архітектури JSP, JSTL і Bootstrap

2.1.1 Порівняльний аналіз JSP/JSTL з альтернативами

У процесі вибору інструментів для реалізації клієнтської частини веб-застосунку важливим кроком стало обґрунтування доцільності використання

JavaServer Pages (JSP) у поєднанні з бібліотекою JSTL у порівнянні з іншими доступними технологіями відображення даних. Такий аналіз дозволив не лише оцінити відповідність обраного підходу функціональним вимогам системи, але й забезпечити відповідний рівень сумісності із серверною

Серед основних альтернатив варто розглянути такі технології, як Thymeleaf, Freemarker та сучасні JavaScript-фреймворки на зразок React чи Angular. Thymeleaf позиціонується як більш сучасний механізм шаблонізації, який дозволяє попередній перегляд сторінок без запуску сервера, а також має розширені можливості роботи з HTML5. Проте в умовах побудови застосунку з акцентом на класичну модель Spring MVC, JSP демонструє стабільні результати, забезпечує високу продуктивність та має широку підтримку серед веб-серверів і IDE, що робить її практичним вибором, зокрема для освітніх і прикладних проєктів [5].

Що стосується Freemarker, він також має багатий функціонал, але є складнішим у використанні, а його інтеграція з Spring потребує додаткових налаштувань. React і Angular як бібліотеки та фреймворки для створення односторінкових застосунків надають потужні можливості інтерактивності та керування станом, але значно ускладнюють архітектуру проєкту, вимагають налаштування окремих клієнтських середовищ, збірки фронтенду, а також реалізації повноцінного REST API навіть для простих операцій відображення.

З урахуванням специфіки проєкту – створення цілісного монолітного застосунку з переважанням серверної логіки і контрольованого відображення даних – вибір на користь JSP і JSTL є технічно обґрунтованим. Він забезпечує необхідний баланс між функціональністю, простотою реалізації та інтеграційною прозорістю в межах Java-екосистеми. Більше того, з точки зору безпеки та продуктивності, така зв’язка дає змогу уникнути надлишкового коду на клієнті та зберегти повний контроль над потоком даних з боку сервера.

2.1.2 Використання Bootstrap для адаптивного дизайну

Сучасні веб-застосунки повинні відповідати високим вимогам до адаптивності інтерфейсу, оскільки користувачі дедалі частіше звертаються до сервісів через мобільні пристрої. У зв’язку з цим при розробці клієнтської частини системи бронювання автобусних квитків особливу увагу було приділено побудові зручного і гнучкого інтерфейсу, який би однаково коректно функціонував на різних типах екранів. Для реалізації таких вимог було обрано фреймворк Bootstrap – один із найпопулярніших засобів швидкої розробки інтерфейсів з адаптивним дизайном.

Bootstrap є open source CSS-фреймворком, що включає набір готових стилів і компонентів для побудови візуально привабливих та однорідних інтерфейсів. Його використання дозволило уникнути надмірного написання власного стилізованого коду, завдяки чому розробка була суттєво прискорена, а візуальна складова проєкту – уніфікована. За допомогою вбудованої сіткової системи Bootstrap було досягнуто динамічного масштабування елементів сторінки відповідно до розміру екрану користувача, що значно підвищує загальну зручність роботи з застосунком[6].

Окрім адаптивності, Bootstrap також надає розробникам доступ до широкого спектру інтерактивних компонентів, реалізованих за допомогою JavaScript – таких як модальні вікна, навігаційні панелі, випадаючі списки, каруселі тощо. Ці компоненти були успішно використані у процесі реалізації функціоналу перегляду рейсів, форм бронювання та підтвердження замовлення, де потрібна як швидкодія, так і інтуїтивно зрозуміле оформлення.

Особливої уваги заслуговує можливість швидкої кастомізації Bootstrap-компонентів без необхідності втручання в базовий код фреймворку. Це дозволяє зберігати індивідуальність інтерфейсу при дотриманні єдиної стилістики. У межах даного проєкту застосовувались переважно базові класи

компонування та колірні схеми, що дозволило уникнути перевантаження інтерфейсу зайвими візуальними елементами й зосередитись на функціональності системи.

Таким чином, використання Bootstrap забезпечило не лише адаптивність веб-інтерфейсу, але й сприяло зниженню витрат часу на стилізацію та підтримку коду. Крім того, завдяки дотриманню сучасних стандартів верстки, платформа стала доступною для користувачів із різних пристроїв та браузерів без втрати якості візуального відображення чи функціоналу. Такий підхід відповідає принципам UX/UI-дизайну та дозволяє задовольнити очікування сучасних користувачів.

2.2 Аналіз технологій Back-end розробки

Архітектура серверної частини є основою функціональності будь-якого веб-застосунку. У випадку платформи онлайн-бронювання автобусних квитків особливо важливою є надійність обробки запитів, швидкий доступ до даних, а також можливість масштабування системи при зростанні навантаження. Для реалізації бекенд-частини цього проєкту було обрано стек технологій на основі Java, що включає Spring Framework, модуль Spring Boot, шаблон MVC, а також бібліотеки для роботи з базами даних через JPA[7,8].

Модель “контролер-сервіс-репозиторій”, яка реалізується через Spring MVC, дозволяє розділити обробку HTTP-запитів, бізнес-логіку та взаємодію з базою даних у межах окремих логічних шарів. Такий підхід позитивно впливає на супровідність і тестованість коду, а також спрощує командну розробку. Застосування шаблону MVC також дає змогу більш гнучко керувати форматами відповіді (HTML, JSON тощо), що є важливим при одночасній підтримці веб-інтерфейсу та API.

Для взаємодії з базою даних обрано Spring Data JPA, який абстрагує розробника від прямого написання SQL-запитів, натомість дозволяючи будувати запити на основі Java-інтерфейсів і анотацій. Ця технологія

автоматично генерує запити до бази даних відповідно до імен методів, що суттєво зменшує обсяг рутинного коду. Такий підхід особливо ефективний у випадках, коли система працює з великою кількістю сутностей, таких як користувачі, рейси, квитки, замовлення тощо.

Серед інших функціональних переваг обраного бекенд-стеку слід відзначити розширену підтримку безпеки, кешування, логування, а також наявність готових рішень для інтеграції з зовнішніми сервісами, такими як платіжні системи чи API геолокації. Крім того, Spring має потужну документацію та активну спільноту, що суттєво полегшує вирішення технічних проблем у процесі розробки.

Таким чином, вибір технологій для серверної частини ґрунтувався на принципах стабільності, масштабованості та простоти розгортання. Обраний стек дозволяє швидко реалізувати як стандартні бізнес-процеси (реєстрація, бронювання, підтвердження замовлення), так і розширити функціонал у разі майбутніх змін або інтеграцій.

2.2.1 Переваги Spring Framework і Spring Boot

Spring Framework вже тривалий час вважається одним із найпотужніших і найгнучкіших фреймворків для розробки Java-застосунків, зокрема у сфері веб-програмування. Його архітектурна модульність, підтримка інверсії керування (IoC) та аспектно-орієнтованого програмування (AOP) дозволяє будувати масштабовані, добре структуровані й легко підтримувані системи. Однією з ключових переваг Spring є його здатність об’єднувати в собі численні технології та стандарти, забезпечуючи при цьому єдиний спосіб керування залежностями, транзакціями та безпекою.

Особливо значущу роль у проєкті відіграє модуль Spring Boot, який є надбудовою над базовим фреймворком і орієнтований на швидку розробку production-ready застосунків. Його головна ідея полягає у філософії “конвенція понад конфігурацію”: більшість стандартних параметрів задано за

замовчуванням, що дозволяє зосередитися на логіці системи, а не на технічному налаштуванні середовища. Spring Boot також постачається з вбудованим веб-сервером (наприклад, Apache Tomcat), що суттєво спрощує процес запуску і тестування додатку на локальному комп'ютері або в хмарному середовищі.

Ще однією важливою перевагою використання Spring Boot є сумісність із різними середовищами виконання, що дає можливість швидко масштабувати проєкт на нові сервери, без суттєвих змін у кодовій базі. Враховуючи потенційне розширення системи – наприклад, за рахунок додавання нових функцій адміністрування, підтримки різних мов інтерфейсу чи інтеграції зі сторонніми API – Spring Boot забезпечує необхідний рівень гнучкості та технічної готовності до еволюції. Взаємодію основних компонентів серверної частини проєкту ілюструє рис. 2.2.

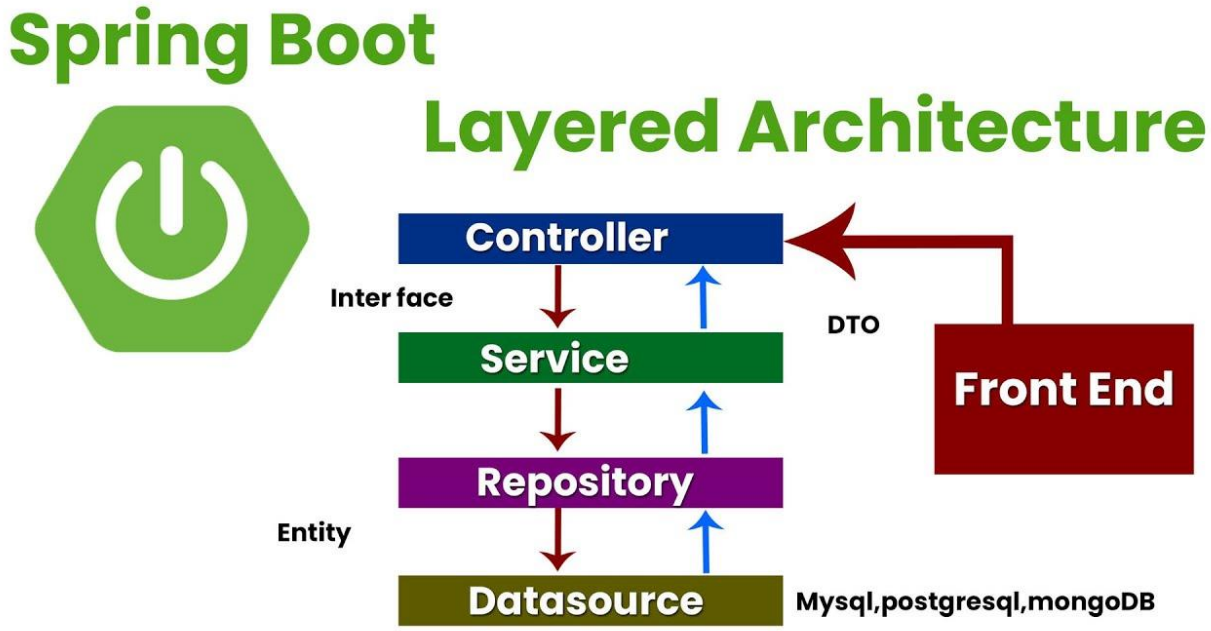

Рисунок 2.2 – Шарова архітектура веб-застосунку на основі Spring Boot

Таким чином, використання Spring Framework у поєднанні з Spring Boot дозволило не лише реалізувати надійну серверну логіку, але й побудувати основу для подальшого зростання проєкту без потреби у

кардинальній перебудові архітектури.

2.2.2 Особливості реалізації MVC-архітектури

Одним із центральних архітектурних підходів, що застосовується у межах реалізації веб-застосунку для бронювання автобусних квитків, є модель Model-View-Controller (MVC). Вибір цієї архітектурної парадигми зумовлений її здатністю забезпечити логічне розділення відповідальностей між різними компонентами системи, що значно спрощує її підтримку, масштабування і розширення функціоналу.

У рамках MVC-архітектури компоненти взаємодіють таким чином: модель (Model) відповідає за роботу з даними та бізнес-логікою, представлення (View) – за відображення даних користувачеві, а контролер (Controller) – за обробку вхідних запитів та керування потоком даних між моделлю і представленням. У даному проєкті ця концепція була реалізована за допомогою Spring MVC, що надає зручні засоби для створення контролерів, обробки запитів HTTP, валідації форм, а також маршрутизації.

Контролери в системі виконують функцію логічних точок входу до функціоналу: саме вони приймають вхідні запити, звертаються до відповідних сервісів, а потім передають отримані дані до шаблонів JSP для формування HTML-сторінки. Такий підхід дозволяє Чітко організувати код, уникаючи дублювання логіки і створюючи зрозумілу структуру проєкту. Моделі представлені класами, що описують сутності предметної області (користувачі, рейси, квитки, замовлення), і безпосередньо взаємодіють із базою даних за допомогою JPA.

Суттєвою перевагою Spring MVC є також підтримка двостороннього зв’язку між формами на клієнтському боці і моделями, що дозволяє ефективно працювати з введенням даних, забезпечуючи автоматичну передачу значень із форм до об’єктів у Java. Завдяки цьому стало можливим реалізувати динамічні форми бронювання, редагування профілю, зміни

пароля та інші сервіси з мінімальними зусиллями.

Отже, використання шаблону MVC у проєкті дозволило організувати логіку веб-застосунку у зрозумілий і масштабований спосіб. Це сприяло зменшенню складності коду, підвищенню його читабельності та забезпечило основу для надійної й довготривалої підтримки системи.

2.2.3 Spring Data JPA та ORM-технології

Одним із ключових завдань у побудові сучасного веб-застосунку є ефективна і безпечна взаємодія з базою даних. У рамках даного проєкту цю функцію було реалізовано за допомогою ORM-технологій (Object-Relational Mapping), що дозволяють працювати з таблицями бази даних через об’єкти мови програмування. Конкретно, для реалізації цієї взаємодії було використано Spring Data JPA – підсистему фреймворку Spring, яка базується на Java Persistence API.

У рамках реалізації веб-платформи для бронювання автобусних квитків, ORM-підхід також дозволив забезпечити високий рівень узгодженості між логікою застосунку та його структурою даних. Визначення зв’язків між сутностями (наприклад, відношення один-до-багатьох між рейсом і квитками, або багато-до-одного між користувачем і замовленням) було реалізовано за допомогою відповідних анотацій JPA, що надало змогу уникнути дублювання інформації та забезпечити цілісність даних на рівні доменної моделі.

Крім того, підтримка кастомізованих запитів за допомогою JPQL (Java Persistence Query Language) дала змогу створити складні вибірки – зокрема, пошук рейсів за параметрами, відображення історії бронювання або фільтрація доступних місць. Такий підхід є гнучким, масштабованим і, що важливо, інтегрується безпосередньо в екосистему Spring без потреби у сторонніх інструментах.

Таким чином, використання Spring Data JPA та ORM-технологій у

даному проєкті забезпечило високий рівень абстракції при роботі з базою даних, зменшило кількість шаблонного коду і зробило систему більш гнучкою, прозорою та зручною для подальшої підтримки й розширення.

2.3 Обґрунтування вибору СУБД MySQL

У процесі проєктування серверної частини системи бронювання автобусних квитків важливим етапом стало визначення оптимальної системи управління базами даних (СУБД), яка б задовольняла вимоги до продуктивності, надійності, масштабованості та простоти інтеграції з вибраним технологічним стеком. З огляду на ці критерії, а також беручи до уваги досвід використання подібних систем у веб-розробці, було прийнято рішення на користь MySQL – однієї з найпоширеніших у світі реляційних СУБД.

MySQL має відкритий вихідний код, що дозволяє використовувати її без додаткових ліцензійних витрат. Завдяки цьому вона широко застосовується в освітніх, стартапових і комерційних проєктах різного масштабу. Крім того, система має багаторічну історію розвитку, добре документована та підтримується потужною спільнотою, що забезпечує стабільність, високу надійність та оперативне оновлення безпекових механізмів.

З технічної точки зору, MySQL забезпечує достатньо високу швидкодію при обробці запитів, особливо в умовах інтенсивного читання даних, що є характерним для інформаційно-пошукових систем, подібних до даної платформи. Завдяки вбудованій підтримці транзакційного механізму з дотриманням принципів ACID (атомарність, цілісність, ізольованість, довговічність), ця СУБД забезпечує коректність виконання взаємозалежних операцій, таких як бронювання квитка і оновлення кількості доступних місць[9,10].

MySQL також добре поєднується зі Spring Data JPA – фреймворком, що

був використаний у межах даного проєкту для взаємодії з базою даних. Це дозволяє уникнути зайвих труднощів при налаштуванні з'єднання, виконанні запитів і роботі з транзакціями. Крім того, MySQL підтримує використання зовнішніх ключів, обмежень цілісності та індексування, що дає змогу ефективно реалізовувати зв'язки між сутностями (наприклад, між користувачами, замовленнями та рейсами), підвищуючи швидкість виконання запитів і зменшуючи навантаження на сервер.

Важливим аргументом на користь вибору саме MySQL стала також її стабільна робота у поєднанні з вбудованими веб-серверами (зокрема, Apache Tomcat) та безпроблемне розгортання як на локальних, так і на хмарних платформах. Завдяки простій конфігурації та зручному інтерфейсу адміністрування, MySQL дозволяє легко супроводжувати систему навіть у випадках обмеженого доступу до технічної підтримки.

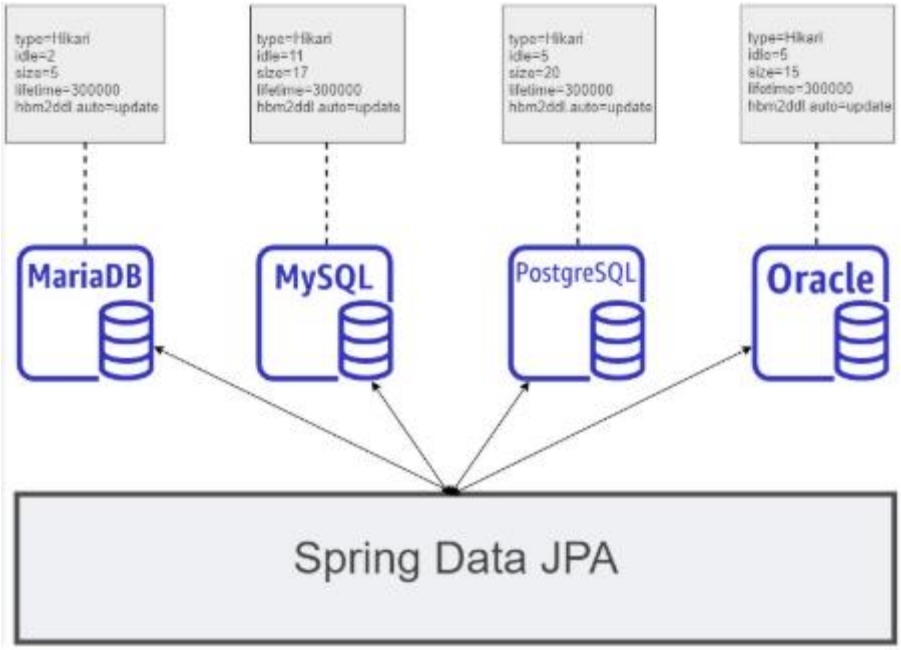

Рисунок 2.3 – Схема інтеграції Spring Data JPA з реляційними базами даних

Таким чином, вибір MySQL як основної СУБД для веб-застосунку базувався на її сумісності з обраним серверним фреймворком, високій продуктивності, стабільності, а також гнучкості при реалізації складних запитів і підтримці структури даних, яка відповідає потребам

автоматизованої системи бронювання.

2.3.1 Створення архітектури бази даних

Для забезпечення зберігання та ефективної обробки інформації в межах веб-застосунку було розроблено реляційну базу даних, структура якої відображає основні бізнес-процеси системи онлайн-бронювання квитків на потяги. Проєктування бази даних здійснювалося з урахуванням вимог нормалізації, цілісності даних та забезпечення гнучкості для майбутнього розширення функціоналу. На рисунку 2.4 представлено логічну схему реляційної бази даних для системи бронювання квитків (див. рис. нижче).

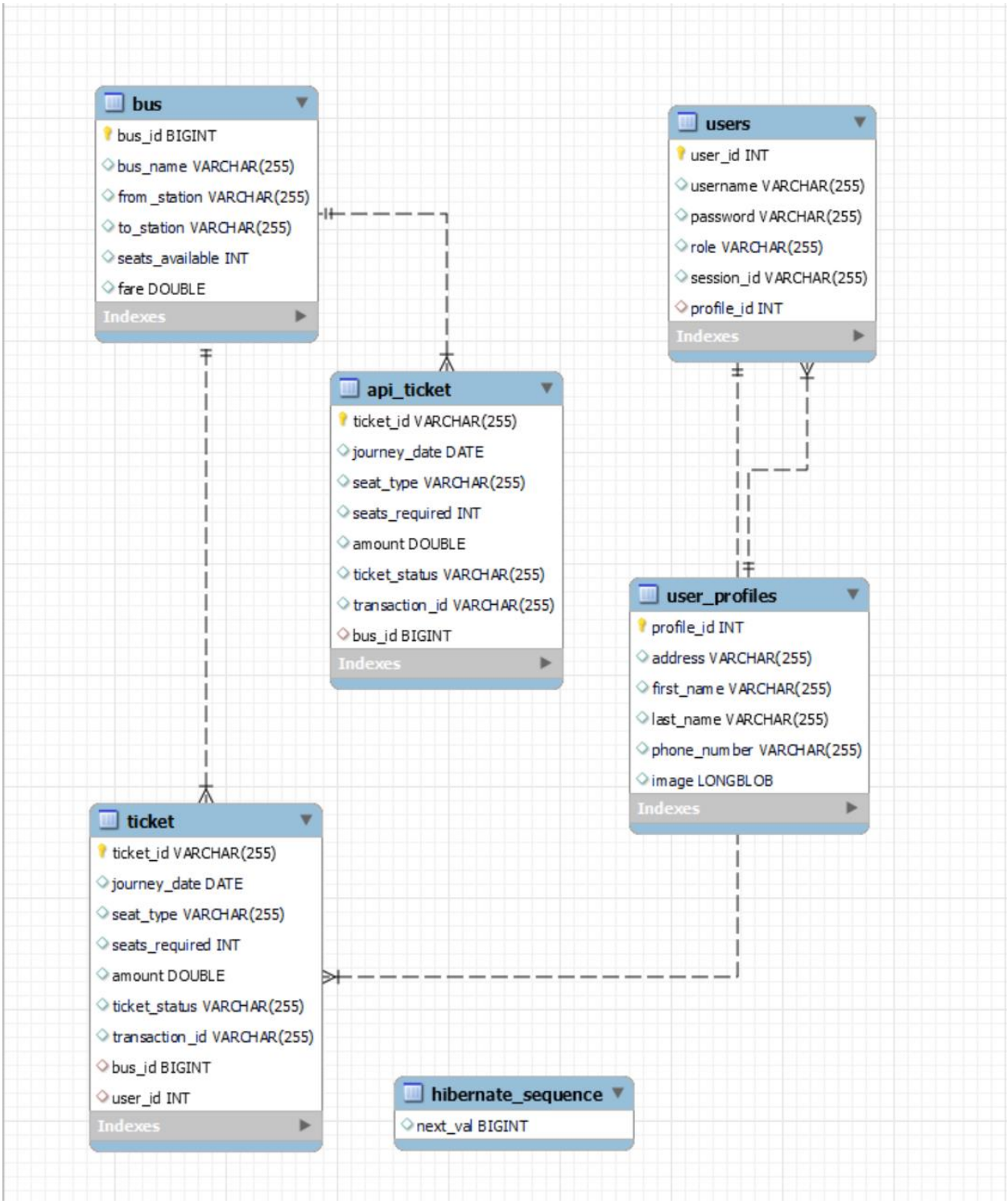

Рисунок 2.4 – Логічна схема бази даних для системи онлайн-бронювання квитків

Структура бази даних охоплює такі основні таблиці:

- users – містить інформацію про зареєстрованих користувачів системи. Основні поля: user_id (унікальний ідентифікатор), username (логін), password (хешований пароль), role (роль користувача), session_id

(ідентифікатор сесії), profile_id (зовнішній ключ до таблиці профілів).

- user_profiles – таблиця, що зберігає детальну інформацію про профіль користувача: адресу, ім'я, прізвище, номер телефону, а також фото (image). Вона пов'язана з таблицею users відношенням «один до одного» через поле profile_id.

- bus – зберігає інформацію про автобуси. Ключові поля: bus_no (унікальний номер потяга), bus_name (назва), fare (вартість), from_station та to_station (станції відправлення і призначення), seats_available (кількість доступних місць).

- ticket – основна таблиця для зберігання даних про заброньовані квитки. Містить поля: ticket_id (унікальний ідентифікатор квитка), journey_date (дата поїздки), seat_type (тип місця), seats_required (кількість місць), amount (вартість), ticket_status (статус квитка), transaction_id (ідентифікатор транзакції), bus_no (зовнішній ключ до таблиці bus), user_id (зовнішній ключ до таблиці users). Таким чином, квиток завжди пов'язаний із конкретним користувачем і конкретним потягом.

- api_ticket – додаткова таблиця, призначена для зберігання квитків, оформлених через API. Її структура подібна до таблиці ticket, але не містить зовнішнього ключа на користувача. Наявність окремої таблиці дає змогу легко інтегрувати зовнішні сервіси чи автоматизовані процеси оформлення квитків.

- hibernate_sequence – службова таблиця, яку використовує ORM-фреймворк Hibernate для генерації унікальних ідентифікаторів. Вона містить лише одне поле next_val, що відстежує останнє використане значення.

- Зв'язки між таблицями реалізовані таким чином, щоб забезпечити цілісність і мінімізувати дублювання даних. Зокрема, кожен користувач має лише один профіль, але може мати декілька квитків. Кожен квиток належить до одного потяга та одного користувача. Таблиця потягів (train) виступає як центральний вузол для зберігання актуальної інформації про доступні поїзди.

Така структура бази даних дозволяє легко реалізовувати основні

операції: реєстрацію та автентифікацію користувачів, оформлення квитків, відображення інформації про потяги, перевірку доступності місць, а також обробку історії замовлень. Дотримання принципів реляційного моделювання забезпечує надійність зберігання даних, їхню узгодженість і ефективність виконання складних запитів при масштабуванні системи.

2.4 Інструменти та середовища розробки

Ефективна реалізація веб-застосунку передбачає не лише правильний вибір архітектури та мов програмування, але й використання відповідних інструментів і середовищ, що забезпечують організований процес розробки, зручність деплойменту та контроль над якістю програмного коду. У межах реалізації системи онлайн-бронювання автобусних квитків було використано низку перевірених засобів, які забезпечили стабільну і продуктивну роботу на всіх етапах життєвого циклу програмного продукту.

Проєкт було організовано з використанням системи управління залежностями Maven. Цей інструмент дозволив автоматизувати процес збирання застосунку, керувати зовнішніми бібліотеками та забезпечити сумісність версій підключених модулів. Maven має гнучку структуру конфігураційного файлу pom.xml, у якому визначаються всі необхідні залежності, плагіни та параметри збірки. Такий підхід дозволив уникнути конфліктів між бібліотеками та спростив інтеграцію з іншими середовищами. Крім того, за допомогою Maven було реалізовано стандартизовану структуру директорій, що полегшило орієнтацію в коді та супровід проєкту.

Завдяки використанню Maven вдалося оптимізувати процес збирання проєкту, уніфікувати підключення зовнішніх бібліотек та забезпечити контроль за версіями залежностей. Конфігурація застосунку здійснюється у файлі pom.xml, де вказуються основні залежності та плагіни, необхідні для коректної роботи веб-застосунку (лістинг 2.1). Це дозволило підтримувати чисту структуру проєкту, уникнути конфліктів і підвищити якість розробки.

Лістинг 2.1 – Фрагмент pom.xml із основними залежностями проєкту

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
    <dependency>
        <groupId>org.thymeleaf</groupId>
        <artifactId>thymeleaf-spring5</artifactId>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <scope>provided</scope>
    </dependency>
    <!-- Додаткові залежності за потреби -->
</dependencies>
```

Для ведення журналу подій у системі використовувалась бібліотека Log4j2p[11]. Її використання дозволило фіксувати важливі події під час роботи застосунку: зокрема, спроби авторизації, помилки обробки запитів, збої при взаємодії з базою даних тощо. Наявність логування відіграє суттєву роль при тестуванні, налагодженні та подальшій експлуатації веб-застосунку, адже дозволяє швидко виявляти і локалізувати проблемні ділянки коду або конфігурації.

Для організації логування у веб-застосунку було використано Log4j2, який забезпечує гнучке налаштування рівнів логування (info, warn, error, debug) та зручний формат виведення повідомлень у консоль або файл. Це дозволяє не лише відслідковувати роботу системи у режимі реального часу, але й зберігати історію подій для подальшого аналізу. Приклад налаштування

логування через конфігураційний файл log4j2.xml наведено нижче (лістинг 2.2).

Лістинг 2.2 – Фрагмент налаштування log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Appenders>
        <Console name="Console" target="SYSTEM_OUT">
            <PatternLayout
                pattern="%d{HH:mm:ss.SSS} [%t] %-5level
%logger{36} - %msg%n" />
        </Console>
        <File name="FileLogger" fileName="logs/app.log">
            <PatternLayout
                pattern="%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L -
%m%n" />
        </File>
    </Appenders>
    <Loggers>
        <Root level="info">
            <AppenderRef ref="Console" />
            <AppenderRef ref="FileLogger" />
        </Root>
    </Loggers>
</Configuration>
```

У якості веб-сервера для розгортання застосунку було обрано Apache Tomcat – легкий та широко використовуваний сервер для Java-додатків. Його підтримка сервлетів, JSP і простота налаштування ідеально відповідали обраному стеку технологій. Завдяки цьому середовищу застосунок міг бути запущений як локально, так і на хостингових платформах без змін у кодовій базі або конфігураційних файлах.

Розробка здійснювалась у середовищі IntelliJ IDEA, яке забезпечило розширене автодоповнення, зручну навігацію по коду, засоби рефакторингу, інтеграцію з системами контролю версій та зручну підтримку Spring Framework. Завдяки використанню цього IDE вдалося значно підвищити продуктивність розробників і зменшити кількість помилок на ранніх етапах створення застосунку[13].

У результаті поєднання зазначених інструментів було сформовано

узгоджене та ефективне середовище розробки, яке дозволило реалізувати веб-систему із сучасним функціоналом, гнучкою архітектурою і можливістю її подальшого розширення без кардинальної перебудови інфраструктури.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Архітектура веб-застосунку

Проєктування архітектури веб-застосунку для системи онлайн-бронювання автобусних квитків базується на принципах модульності, масштабованості та підтримуваності. Архітектура побудована у відповідності до шаблону Model-View-Controller (MVC), що дозволяє логічно розмежувати відповідальність між різними рівнями системи: представленням, логікою обробки даних та доступом до джерел інформації. Такий підхід забезпечує чітку організацію коду, полегшує підтримку та розвиток програмного забезпечення у майбутньому. Схематичний опис шаблону MVC можна побачити на рисунку 3.1.

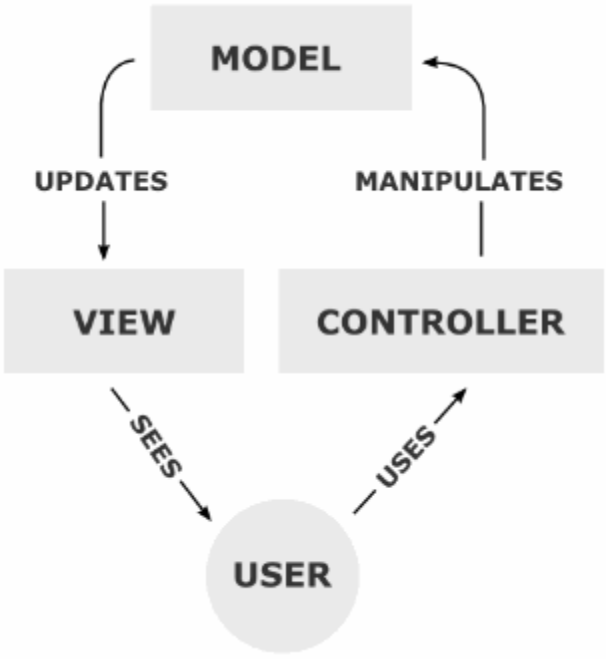

Рисунок 3.1 – Архітектурна діаграма системи Model-View-Controller

Зображення ілюструє класичну модель архітектурного шаблону Model-View-Controller (MVC). У цій структурі всі взаємодії розділені на три

ключові компоненти:

- Model – відповідає за зберігання, обробку та оновлення даних. Усі зміни стану програми або бізнес-логіки відбуваються саме на цьому рівні.
- View – відповідає за відображення даних користувачу. Вигляд отримує оновлення від моделі та представляє інформацію у зручному для користувача форматі.
- Controller – посередник між користувачем та системою. Він приймає команди від користувача, передає їх моделі для обробки, а потім оновлює вигляд.

На рисунку 3.1 відображено основні зв’язки між компонентами:

- Користувач використовує Controller для взаємодії з системою.
- Controller маніпулює Model, змінюючи стан даних.
- Model оновлює View, надсилаючи зміни для візуалізації.
- Користувач бачить View, сприймаючи результати дій.

Слід додати, що застосунок було реалізовано як багаторівневу систему, у якій фронтенд-інтерфейс взаємодіє з серверною логікою через HTTP-протокол. Представлення (View) побудоване за допомогою JSP-шаблонів із використанням JSTL для динамічного наповнення сторінок. Це забезпечує гнучкість у відображенні інформації та підтримку локалізації, що є актуальним для мультикористувацьких систем.

На серверному боці головну роль відіграє фреймворк Spring Boot, який координує взаємодію між компонентами системи. Контролери обробляють запити користувачів, викликають відповідні сервіси, де реалізована бізнес-логіка, і передають результати до рівня представлення. Використання Spring MVC дозволило структурувати логіку таким чином, щоб кожен компонент системи був максимально незалежним від інших, що спрощує подальше тестування та розширення функціоналу.

Робота з даними реалізована за допомогою ORM-рішення на основі Spring Data JPA. База даних MySQL зберігає інформацію про користувачів, рейси, замовлення, квитки та інші сутності, важливі для роботи системи.

Кожна сутність у додатку відповідає окремому класу у доменній моделі, що забезпечує повну відповідність між логікою застосунку та структурою бази даних.

3.1.1 Структура бази даних та моделювання сутностей (ER-модель, JPA Entity)

Розробка системи бронювання квитків передбачає зберігання та обробку великого обсягу даних, пов’язаних з користувачами, рейсами, замовленнями, квитками та адміністративними діями. В основі проєкту лежить реляційна модель бази даних, реалізована за допомогою СУБД MySQL. Структура бази даних була спроєктована з урахуванням принципів нормалізації, що забезпечує мінімізацію надлишковості та збереження цілісності даних[15,16].

Концептуальна модель бази даних ґрунтується на п’яти основних сутностях: User, Bus, Booking, Ticket та Route. Сутність User містить інформацію про облікові записи користувачів і включає такі поля, як ідентифікатор, ім’я, електронну адресу, роль, пароль тощо. Кожному користувачу може відповідати декілька замовлень, що формує зв’язок один-до-багатьох між таблицями User і Booking.

Сутність Bus відображає параметри конкретного транспортного засобу: номер автобуса, кількість місць, модель, дата реєстрації. Вона пов’язана з Route – таблицею, що зберігає маршрути, включаючи станції відправлення, призначення, дату, час відправлення і прибуття, а також ціну. Один автобус може обслуговувати багато маршрутів.

Таблиця Booking виконує роль зв’язуючої ланки між користувачем і маршрутом. Кожне бронювання містить інформацію про дату замовлення, кількість місць, статус, та зв’язується з Ticket, який описує конкретне місце, зарезервоване в певному рейсі.

Уся модель реалізована у вигляді класів JPA (Java Persistence API), що

анотуються згідно з ORM-підходом. Наприклад, клас User містить відповідні позначення @Entity, @Table, а також @OneToMany, що демонструє зв’язок з бронюваннями (лістинг 3.1).

Лістинг 3.1 – Опис сутності User з використанням JPA-анотацій для зв’язків з таблицею бронювань

```
@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    private String email;

    private String password;

    private String role;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private List<Booking> bookings = new ArrayList<>();

    // геттери і сеттери
}
```

Аналогічно, клас Booking містить зв’язки до користувача, маршруту та квитків. Таким чином, структура сутностей на рівні JPA повністю відповідає структурі таблиць у базі даних, забезпечуючи прозору інтеграцію між логікою застосунку і сховищем даних.

Побудована таким чином модель даних дозволяє ефективно реалізовувати складні вибірки (наприклад, перегляд усіх замовлень певного користувача, перевірка наявності вільних місць на маршруті, облік статистики бронювань), а також підтримує масштабованість системи. За потреби можуть бути легко додані нові сутності – наприклад, Payment для збереження транзакцій, без порушення існуючої логіки або структури.

3.2 Розробка клієнтської частини

Клієнтська частина веб-застосунку виступає ключовим елементом у взаємодії користувача з функціоналом системи бронювання квитків. Саме через неї здійснюється доступ до основних сервісів: пошуку рейсів, перегляду розкладу, бронювання місць, авторизації та перегляду історії замовлень. У процесі проєктування клієнтської частини основний акцент було зроблено на забезпеченні інтуїтивно зрозумілого інтерфейсу, що відповідає принципам доступності, адаптивності та зручності використання.

Розмітка сторінок реалізована на основі технології JavaServer Pages. Це дозволило здійснювати формування вмісту безпосередньо на сервері з урахуванням логіки, закладеної у контролерах, та передавати користувачеві вже підготовлений HTML-код. Застосування JSTL як доповнення до JSP забезпечило гнучке виведення даних, реалізацію умовних блоків та ітерацій над колекціями без залучення складного JavaScript-коду, що підвищило стабільність інтерфейсу та зменшило вірогідність помилок при виведенні динамічного вмісту.

Візуальна складова клієнтського інтерфейсу була реалізована з використанням фреймворку Bootstrap. Завдяки цьому вдалось досягти сучасного вигляду сторінок та їхньої адаптивності до різних розмірів екранів – від десктопних моніторів до мобільних пристроїв. Bootstrap значно прискорив розробку за рахунок широкого набору готових стилізованих компонентів, таких як форми, кнопки, панелі навігації, таблиці та модальні вікна. Це дозволило зосередити зусилля на логіці інтерфейсу замість детальної ручної верстки.

Серед ключових функцій клієнтської частини слід відзначити реалізацію форм реєстрації та входу в систему, які включають перевірку введених даних як на клієнтському рівні (через валідаційні класи Bootstrap), так і на серверному (через механізми Spring MVC). Крім того, реалізовано сторінки пошуку рейсів, де користувач вводить пункти відправлення і

призначення, дату поїздки та кількість пасажирів, після чого система виводить відповідні результати у зручному табличному вигляді.

Особистий кабінет користувача дозволяє переглядати історію замовлень, переглядати статус оплати, оновлювати профільну інформацію та змінювати пароль. Інтерфейс адміністратора, що також реалізований на основі тих самих технологій, містить додаткові функції, пов’язані з керуванням розкладом, додаванням нових рейсів та переглядом статистичних даних.

Таким чином, клієнтська частина веб-застосунку реалізована як логічне і функціональне розширення серверної частини, що забезпечує повноцінну взаємодію з користувачем через структуровану, адаптивну та візуально привабливу систему інтерфейсів. Її побудова на основі перевірених технологій JSP, JSTL і Bootstrap дозволила створити надійний, швидкий у роботі і зручний для подальшої підтримки клієнтський рівень програмного забезпечення.

3.2.1 Структура веб-інтерфейсу та його реалізація засобами JSP і Bootstrap

Проєктування інтерфейсу користувача веб-застосунку вимагало врахування як функціональних потреб майбутніх користувачів, так і сучасних принципів UI/UX-дизайну. Основною метою на цьому етапі було створення інтуїтивно зрозумілого, логічно структурованого та естетично привабливого інтерфейсу, який би забезпечував швидкий доступ до ключових функцій системи – пошуку рейсів, перегляду доступних квитків, здійснення бронювання, керування профілем та історією замовлень.

Розробка інтерфейсу розпочиналась із побудови макетів основних сторінок, зокрема головної, сторінки входу, реєстрації, результатів пошуку, особистого кабінету та адмін-панелі. На етапі макетування було визначено логіку розміщення елементів, шляхи навігації та типові сценарії взаємодії

користувача з системою. Структура кожної сторінки була сформована відповідно до завдань, які вона мала вирішувати, що дозволило мінімізувати кількість дій, необхідних для досягнення цілей користувача.

Технічна реалізація інтерфейсу була виконана за допомогою JSP у поєднанні з фреймворком Bootstrap. Таке поєднання дозволило сформувати динамічні сторінки, які адаптуються до контенту в реальному часі, а також гарантує їхню коректну роботу на різних пристроях – від смартфонів до настільних комп’ютерів. Компоненти Bootstrap, зокрема сіткова система, вкладки, навігаційні панелі, алерти, модальні вікна та кнопки, використовувались для створення єдиного візуального стилю та забезпечення однакової поведінки інтерфейсу у різних браузерах.

Особливу увагу приділено кольоровій палітрі та типографіці. Перевагу було надано спокійним і нейтральним відтінкам, які не відволікають користувача від основного вмісту. Шрифти обирались з урахуванням зчитуваності, а розміри елементів оптимізовано для роботи як мишкою, так і на сенсорних екранах. Усі активні елементи мають візуальні підказки, наведення та ефекти, що підвищують зрозумілість взаємодії.

На рівні зручності користування інтерфейс реалізує механізми повідомлень про помилки, підтвердження дій, повідомлень про успішне бронювання чи помилки валідації форми. Це дозволяє користувачеві бути впевненим у тому, що його дії в системі є очікуваними, і вчасно отримувати відповідний зворотний зв’язок.

Для кращої ілюстрації принципів побудови інтерфейсу на рисунку нижче наведено приклад макету сторінки веб-застосунку з основними елементами дизайну (рисунок 3.7). На схемі чітко відображені заголовки, поля для введення даних, кнопки підтвердження та повідомлення, що забезпечує наочність структури користувацького інтерфейсу.

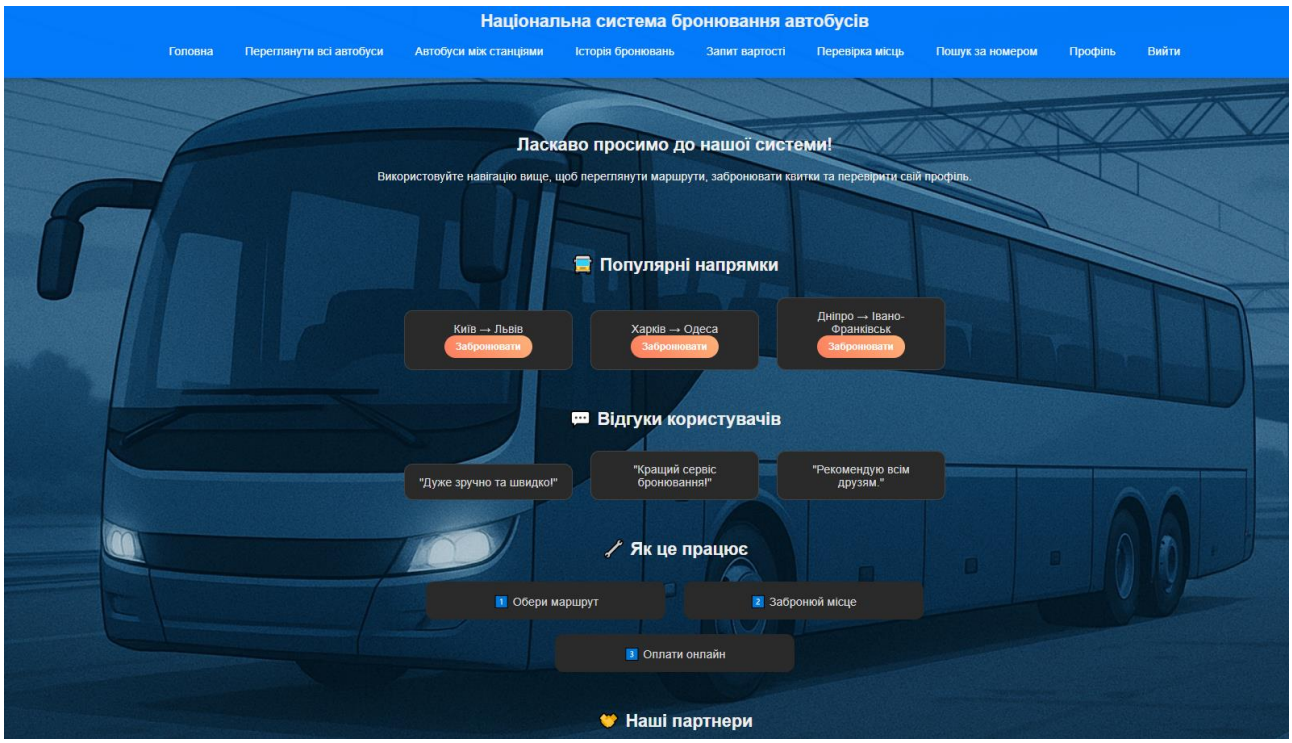

Рисунок 3.2 – Макет інтерфейсу користувача веб-застосунку для онлайн-бронювання квитків

Таким чином, інтерфейс користувача в цьому веб-застосунку не лише реалізує весь передбачений функціонал, а й дотримується сучасних стандартів зручності та доступності, що є критично важливим для взаємодії з широким колом користувачів, які можуть мати різний технічний досвід і використовувати різноманітні пристрої.

3.2.2 Механізм реєстрації та автентифікації користувачів (controller + interceptor)

Функціонал реєстрації та автентифікації користувачів є критично важливим компонентом будь-якої системи з обмеженим доступом до персоналізованих функцій. У реалізованому веб-застосунку для бронювання автобусних квитків цей механізм забезпечує розмежування доступу між незареєстрованими користувачами, зареєстрованими пасажирами та адміністраторами. Технічно система побудована на основі класичного

підходу Spring MVC, з обробкою запитів через контролери та захистом маршрутів за допомогою інтерцепторів.

Реєстрація нових користувачів реалізована через форму, що передає дані на відповідний метод контролера. У класі RegisterController обробляється як відображення форми (GET-запит), так і її обробка (POST-запит). У разі надходження валідних даних створюється новий об’єкт користувача, якому присвоюється роль ROLE_USER, після чого він зберігається в базі даних через відповідний сервіс (лістинг 3.2).

Лістинг 3.2 – Реалізація контролера для реєстрації нового користувача

```
@PostMapping("/register")
public String registerUser(@ModelAttribute("user") User user,
Model model) {
    user.setRole("ROLE_USER");
    boolean isRegistered = userService.registerUser(user);
    if (isRegistered) {
        return "redirect:/login?success";
    } else {
        model.addAttribute("error", "Користувач із такою
електронною поштою вже існує.");
        return "register";
    }
}
```

Аутентифікація користувача реалізована окремо – після введення логіна і пароля система звертається до сервісу перевірки облікових даних. У разі успішного входу сесія користувача зберігається, а сам користувач перенаправляється до особистого кабінету. Якщо дані некоректні – користувач отримує повідомлення про помилку.

Особливістю даного застосунку є використання спеціального перехоплювача (LoginInterceptor), що перевіряє статус авторизації користувача перед обробкою більшості запитів. Інтерцептор реалізує інтерфейс HandlerInterceptor та виконує перевірку сесії на наявність авторизованого об’єкта користувача, що продемонстровано (лістинг 3.3).

Лістинг 3.3 – Перевірка авторизації користувача у HTTP-запиті через Inteceptor

```
@Override
public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) throws Exception {
    HttpSession session = request.getSession(false);
    if (session != null && session.getAttribute("user") != null)
{
        return true;
    } else {
        response.sendRedirect("/login");
        return false;
    }
}
```

Цей механізм забезпечує базовий рівень безпеки, запобігаючи доступу до захищених маршрутів без попереднього входу в систему. Водночас його реалізація є легкою і не вимагає використання складних конфігурацій Spring Security, що дозволяє краще контролювати логіку входу, зокрема в навчальних чи малих проєктах.

Інтерфейс взаємодії користувача з системою побудований у вигляді простих і зрозумілих форм авторизації та реєстрації, реалізованих через JSP-шаблони. Сторінки містять підказки, перевірку обов’язкових полів і повідомлення про помилки, що покращує користувацький досвід. Реалізацію зовнішнього вигляду можна побачити на рисунку 3.3.

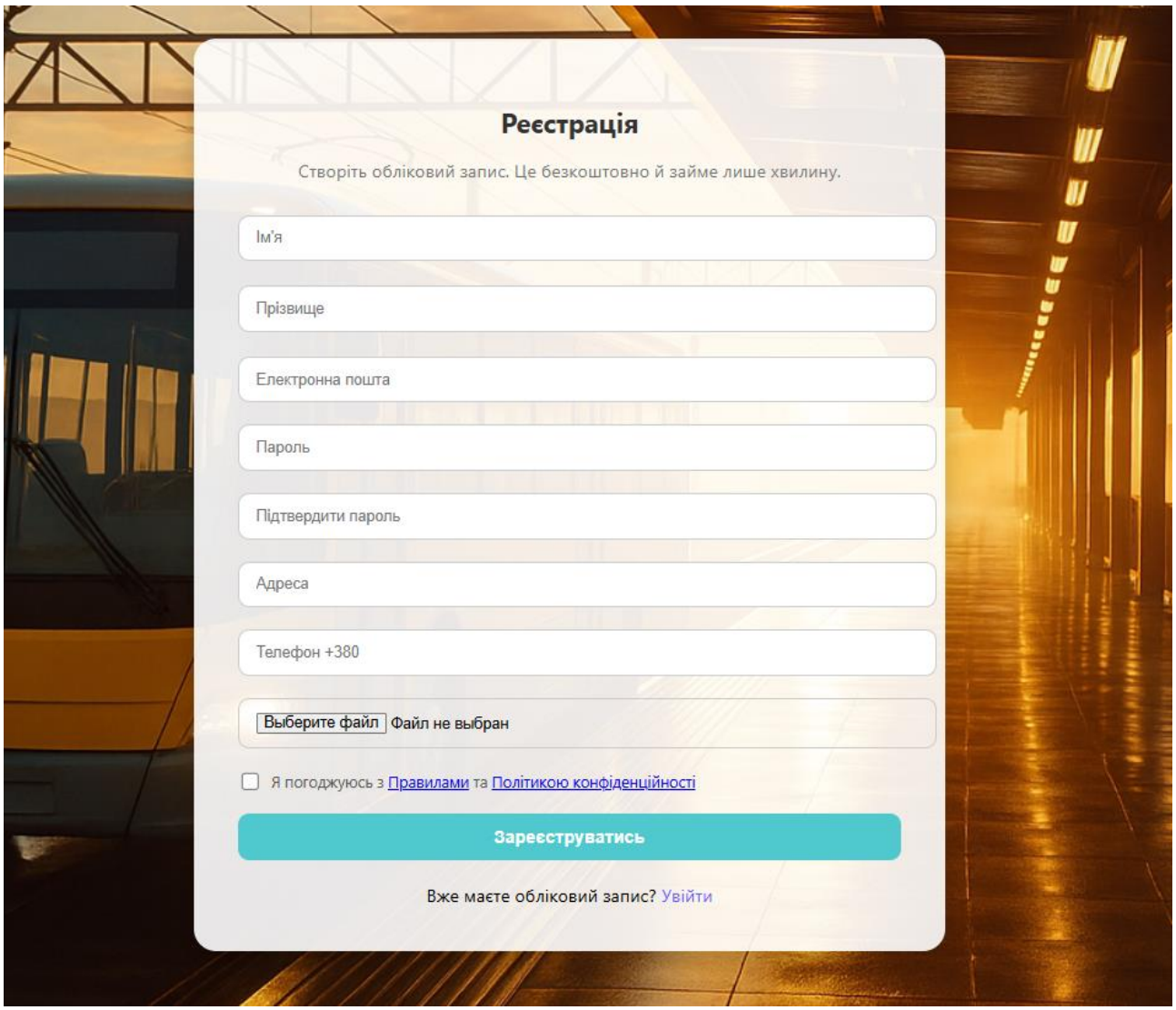

Рисунок 3.3 – Форма, яка використовується для реєстрації нових користувачів

Таким чином, система автентифікації забезпечує надійний контроль доступу до персоналізованого функціоналу застосунку, зберігаючи при цьому простоту реалізації та зручність для кінцевого користувача.

3.2.3 Реалізація перегляду розкладу і логіки бронювання квитків (controller + JSP)

Однією з ключових функцій веб-застосунку для бронювання автобусних квитків є забезпечення користувачеві простого та інтуїтивного інтерфейсу для пошуку, перегляду й бронювання місць на необхідний рейс.

Для цього на клієнтській частині реалізовано кілька базових сценаріїв взаємодії.

На головній сторінці користувач бачить зручну навігаційну панель і форму для додавання нового автобуса (для адміністратора), що дозволяє оперативно оновлювати перелік маршрутів (рис. 3.4). Введення даних здійснюється через інтуїтивно зрозумілі поля – назва автобуса, станції, кількість місць, вартість тощо.

Після цього відображається список усіх доступних автобусів із детальною інформацією про напрямок, час відправлення й прибуття, кількість доступних місць та вартість квитка. Для кожного маршруту передбачена окрема кнопка для переходу до процесу бронювання (рис. 3.5). Завдяки чіткій таблиці та кольоровому маркуванню, користувач може легко порівняти варіанти й обрати потрібний рейс.

Для пошуку міжміських рейсів реалізовано окрему форму, де користувач може обрати пункти відправлення й призначення (рис. 3.6). Після натискання кнопки “Знайти” система фільтрує список доступних маршрутів і відображає лише релевантні результати (рис. 3.7). Це значно підвищує зручність користування платформою, особливо при великій кількості маршрутів.

Логіка пошуку та бронювання реалізована на основі MVC-підходу. Контролери Spring обробляють введені користувачем дані, звертаються до відповідних сервісів для отримання актуальної інформації з бази даних, після чого формують динамічні JSP-сторінки з результатами. Завдяки використанню Bootstrap, інтерфейс зберігає адаптивність і зручний вигляд на різних пристроях.

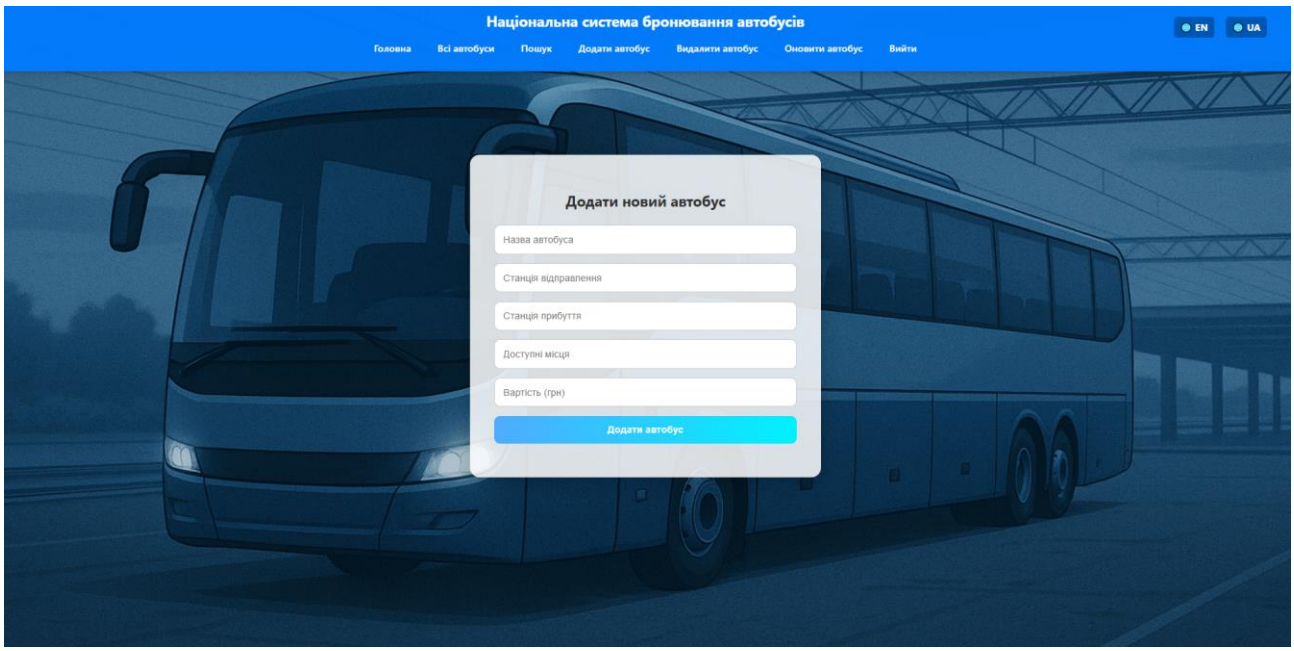

Рисунок 3.4 – Форма, яка використовується для реєстрації нових користувачів

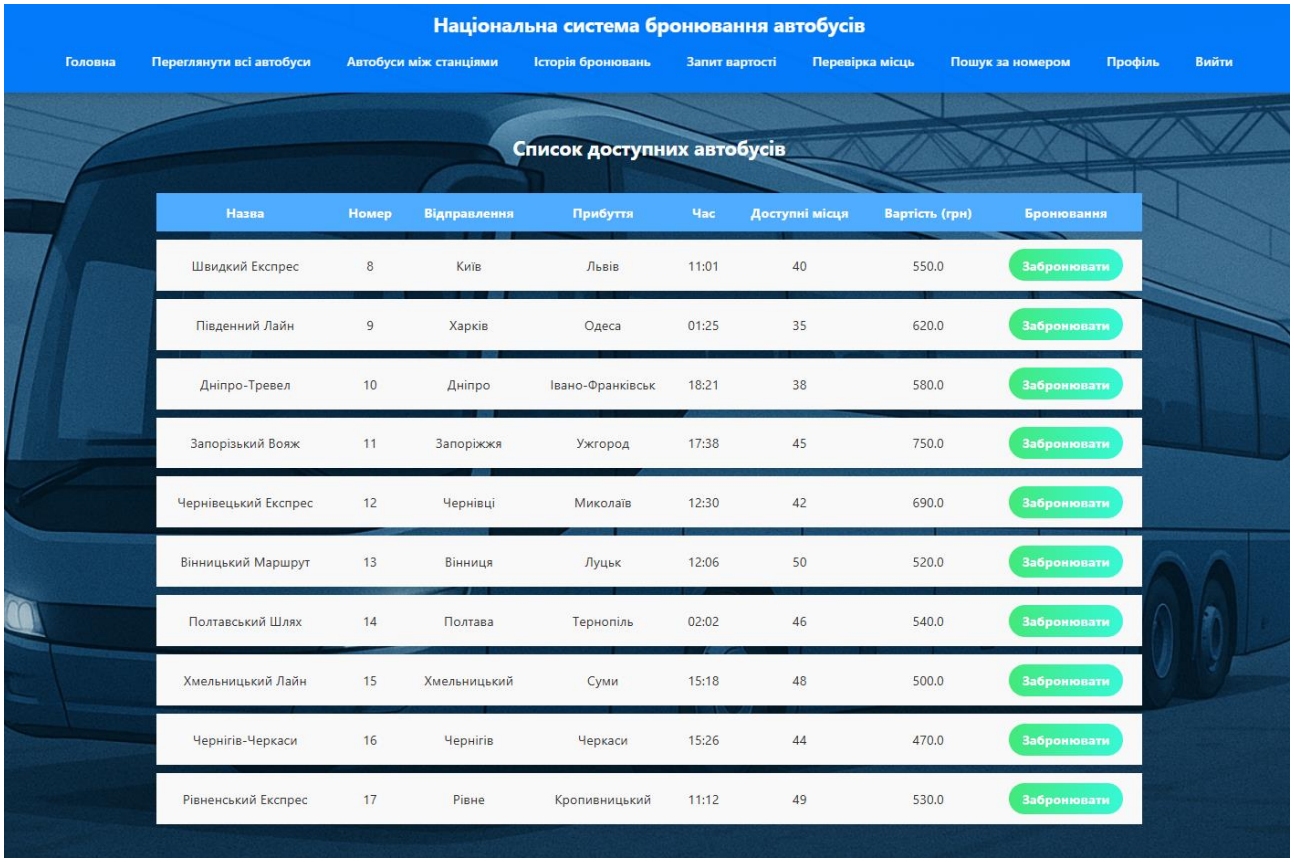

Рисунок 3.5 – Форма, яка використовується для реєстрації нових користувачів

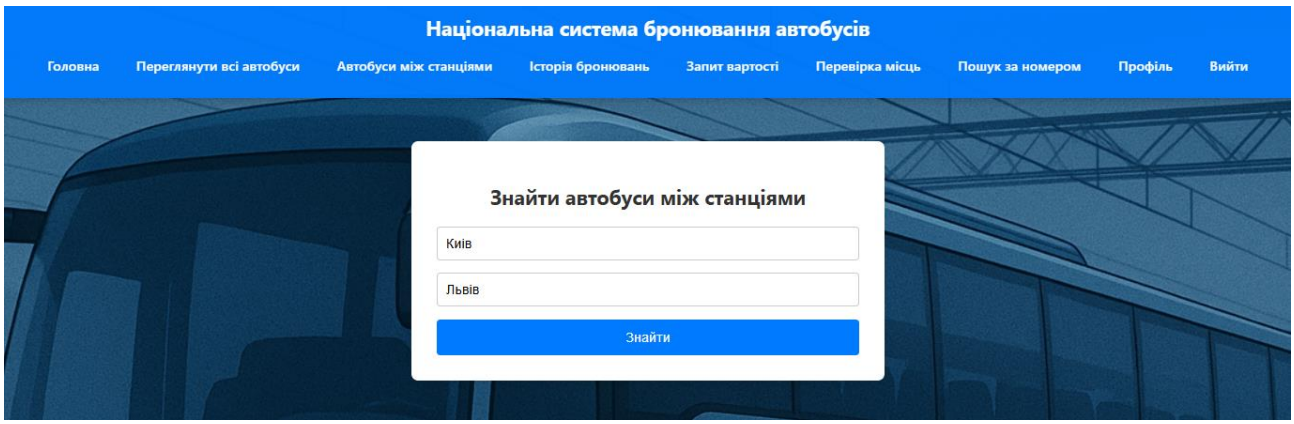

Рисунок 3.6 – Форма, яка використовується для реєстрації нових користувачів

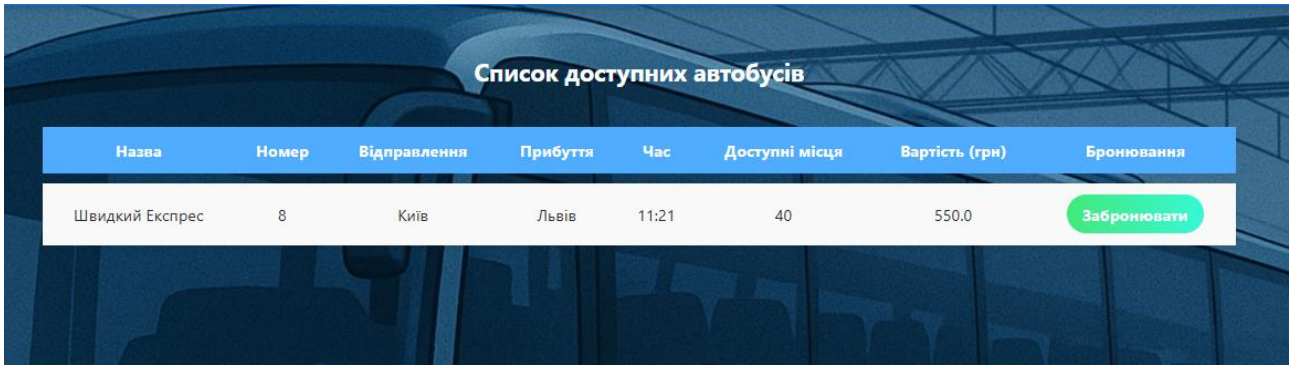

Рисунок 3.7 – Форма, яка використовується для реєстрації нових користувачів

3.3 Серверна логіка і приклад REST-архітектури

Архітектура серверної частини веб-застосунку реалізована відповідно до принципів REST (Representational State Transfer), що дозволяє забезпечити чітке розмежування між представленням і логікою обробки запитів, а також полегшує масштабування й потенційну інтеграцію з мобільними або сторонніми клієнтами. REST-орієнтована серверна логіка побудована на використанні Spring Boot та Spring Web MVC, що надає зручні засоби для створення контролерів, які відповідають на HTTP-запити у форматі JSON.

Основна частина REST-функціональності зосереджена у спеціалізованих класах, анотованих як @RestController. Вони обробляють

запити, що надходять за визначеними маршрутами, та повертають структуровані дані без використання JSP або інших візуальних шаблонів. Замість традиційного рендерингу сторінок, REST-контролери надають клієнтам доступ до об’єктів і ресурсів у вигляді JSON-відповідей, що ідеально підходить для використання з frontend-фреймворками або мобільними застосунками.

REST-архітектура проєкту охоплює такі основні групи функціоналу: керування користувачами, отримання даних про рейси, перегляд історії бронювань, створення й скасування замовлень. Для кожного з типів сутностей реалізовані стандартні CRUD-операції – створення, читання, оновлення, видалення – згідно з методами HTTP (POST, GET, PUT, DELETE). Завдяки цьому досягнуто відповідність архітектурним принципам REST і забезпечено уніфікованість обробки запитів.

3.3.1 REST API для роботи з рейсами, квитками та користувачами (restcontroller)

У межах реалізації серверної логіки важливу роль відіграє REST API, який надає структурований доступ до основних ресурсів системи: маршрутів, користувачів, бронювань та квитків. Реалізація REST API побудована відповідно до принципів REST-архітектури, а всі REST-контролери оформлені за допомогою анотацій @RestController, @RequestMapping, @GetMapping, @PostMapping тощо.

Клас BookingRestController є одним із центральних компонентів API і відповідає за створення бронювань, перегляд історії замовлень і скасування квитків. Приклад методу, який дозволяє отримати всі бронювання певного користувача продемонстровано нище (лістинг 3.4).

Лістинг 3.4 – REST-метод для отримання історії бронювань користувача

```
@GetMapping("/booking-history/{userId}")
public ResponseEntity<List<BookingHistoryResponse>>
getBookingHistory(@PathVariable Long userId) {
    List<BookingHistoryResponse> history =
bookingService.getBookingHistory(userId);
    return ResponseEntity.ok(history);
}
```

Цей метод повертає список бронювань у форматі JSON, що дозволяє легко обробляти його на стороні мобільного застосунку або у frontend-фреймворках. Відповідь формується на основі спеціального DTO-класу BookingHistoryResponse, який містить лише ті поля, що є необхідними для відображення: назва маршруту, дата, статус, кількість місць, сума тощо.

3.3.2 Репозиторії та доступ до даних через Spring Data JPA (repository)

Для організації взаємодії між серверною логікою та базою даних у системі бронювання квитків використано фреймворк Spring Data JPA. Цей модуль надає абстрактний рівень над JPA та Hibernate і дозволяє ефективно реалізовувати CRUD-операції без написання явного SQL-коду. Завдяки цьому доступ до сутностей, збереження, оновлення, видалення та запит даних реалізовано за допомогою інтерфейсів репозиторіїв, які автоматично реалізуються фреймворком під час запуску застосунку.

Таким чином, Spring Data JPA дозволяє гнучко реалізовувати доступ до бази даних, забезпечуючи високу продуктивність, масштабованість і зниження ризику помилок за рахунок автоматизації. Репозиторії чітко розділяють відповідальність між рівнями архітектури, ізолюють доступ до даних та дають змогу зосередитися на бізнес-логіці, а не на деталях реалізації запитів.

3.3.3 Адміністративний функціонал і обробка запитів адміністратором (admin controller)

Веб-застосунок для бронювання автобусних квитків містить окремий блок функціональності, доступний лише для адміністратора. Цей функціонал реалізовано у вигляді спеціального контролера AdminController, який забезпечує доступ до критично важливих дій – додавання, редагування, видалення рейсів, перегляду списків користувачів, бронювань та управління маршрутами.

Адміністративна логіка відокремлена як на рівні контролера, так і через механізм авторизації, що перевіряє роль користувача. Доступ до URL-адрес на кшталт /admin/add-bus чи /admin/view-users надається лише авторизованим користувачам із роллю ROLE_ADMIN. Це перевіряється або через сесію, або через перехоплювач (Interceptor), залежно від обраної конфігурації.

Наприклад, метод додавання нового рейсу (лістинг 3.5).

Лістинг 3.5 – Додавання нового автобуса через адміністративний контролер

```
@PostMapping("/admin/add-bus")
public String addBus(@ModelAttribute("bus") Bus bus,
RedirectAttributes attributes) {
    busService.saveBus(bus);
    attributes.addFlashAttribute("success", "Новий автобус
успішно додано.");
    return "redirect:/admin/view-buses";
}
```

Цей метод отримує модель Bus, яка заповнюється на основі форми на JSP-сторінці. Після збереження об’єкта через сервіс відбувається редирект до сторінки перегляду всіх автобусів. Аналогічно реалізовані методи для редагування і видалення рейсів, які ідентифікуються за busId.

Щоб відобразити адміністративні дані, у контролері передбачені методи перегляду списків користувачів (лістинг 3.6).

Лістинг 3.6 – Відображення списку користувачів у адміністративній панелі

```
@GetMapping("/admin/view-users")
public String viewUsers(Model model) {
    List<User> users = userService.getAllUsers();
    model.addAttribute("users", users);
    return "admin/view-users";
}
```

3.3.4 Оплата, підтвердження бронювання та історія замовлень

Однією з ключових функцій веб-застосунку є зручний пошук, перегляд і бронювання квитків на доступні автобусні рейси. Уся логіка побудована на поєднанні контролерів Spring MVC та динамічних JSP-сторінок, що забезпечують інтерактивний, адаптивний інтерфейс для користувача.

Для підтвердження бронювання користувач переходить на спеціальну сторінку платіжного шлюзу (рис. 3.8). Тут потрібно ввести реквізити банківської картки або скористатися альтернативними методами оплати, такими як LiqPay. Після успішного проведення платежу всі дані щодо бронювання автоматично фіксуються у базі даних.

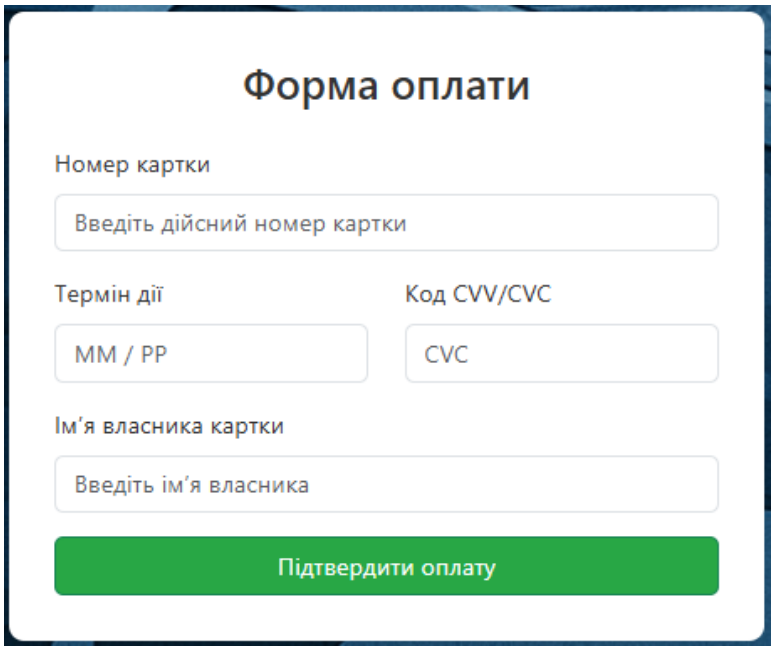

Рисунок 3.8 – Форма виконная оплати

Після завершення процедури оплати та підтвердження бронювання система автоматично зберігає інформацію про куплений квиток у профілі користувача. На окремій сторінці особистого кабінету реалізовано розділ “Історія бронювань квитків”, де у вигляді зручної таблиці відображаються всі попередні покупки користувача (рис. 3.9, 3.10).

У таблиці для кожного квитка вказані такі основні параметри:

- унікальний ID квитка;
- назва автобуса та маршрут (пункти відправлення й прибуття);
- дата поїздки;
- кількість місць;
- вартість;
- клас квитка;
- статус (наприклад, BOOKED — успішно заброньовано).

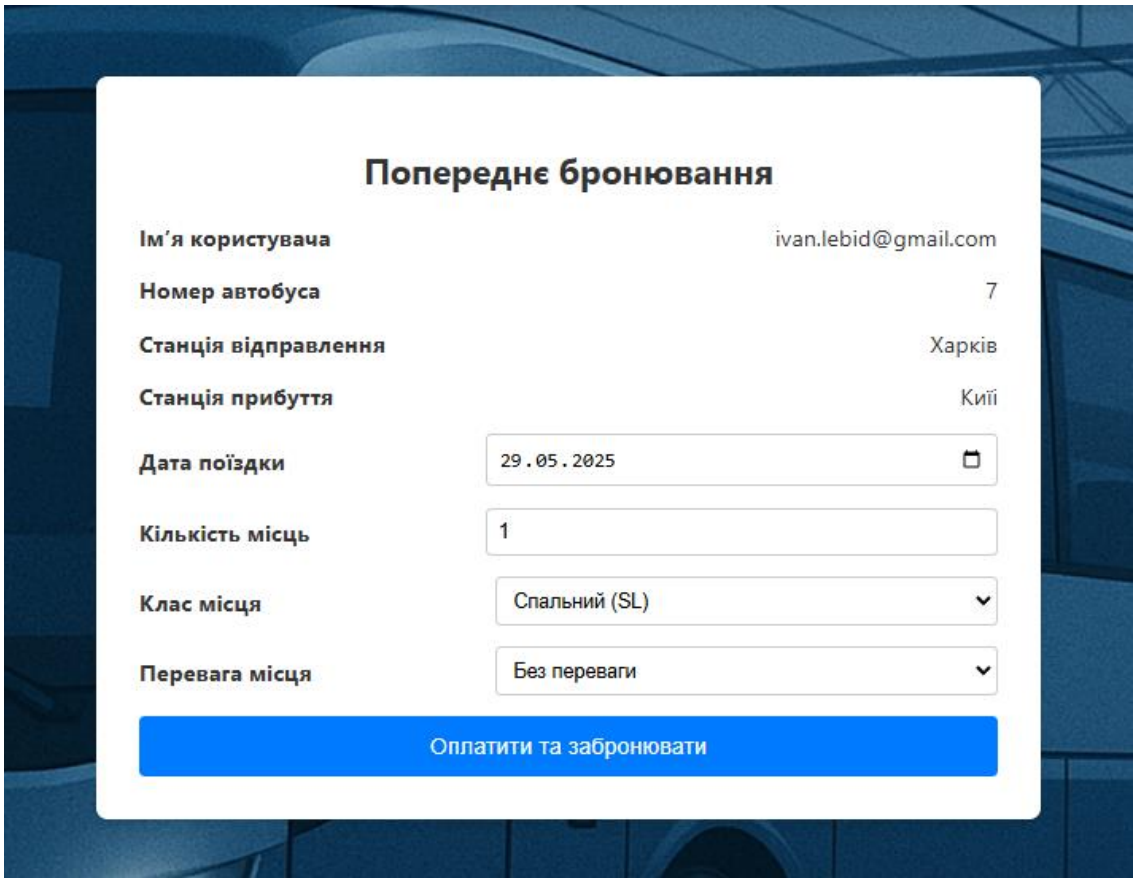

Рисунок 3.9 – Сторінка з історією бронювань користувача

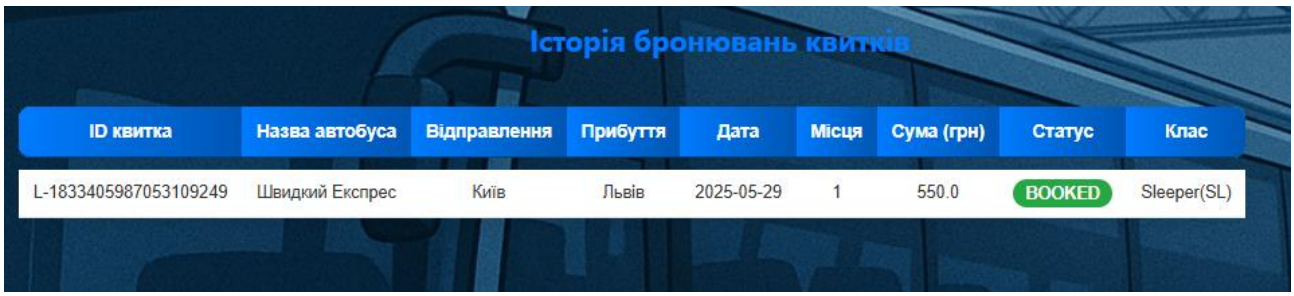

Рисунок 3.10 – Історія бронювання квитків

Дана функціональність дає змогу користувачеві швидко знаходити необхідну інформацію для подорожей, контролювати витрати й відстежувати історію замовлень. У разі потреби (наприклад, для відшкодування чи доказу покупки) дані з таблиці можуть бути використані як підтвердження здійсненого бронювання.

Після завершення платежу система додатково показує сторінку з підтвердженням бронювання, де відображаються всі деталі замовлення, статус операції та спеціальне повідомлення про успішне бронювання (рис. 3.11). Це підвищує прозорість сервісу й формує довіру користувачів до платформи.

Рисунок 3.11 – Підтвердження успішного бронювання квитка

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

Для початку роботи з веб-застосунком відкрийте його у будь-якому сучасному браузері, ввівши URL-адресу в адресному рядку. Після завантаження сайту ви потрапите на головну сторінку, з якої можна отримати доступ до розкладу рейсів, пошуку квитків, особистого кабінету та, за потреби, адміністративної панелі.

Нові користувачі можуть зареєструватися, заповнивши коротку форму з основними даними. Після цього доступ до системи здійснюється за допомогою електронної пошти та пароля. Пошук квитків відбувається через відповідний розділ або головну сторінку, де можна обрати напрямок, дату та кількість місць. Після вибору рейсу користувач переходить до оформлення та оплати замовлення через підключену платіжну систему. Оплачений квиток зберігається в особистому кабінеті.

В розділі «Мої квитки» можна переглядати історію замовлень і деталі поїздок. Адміністратори мають доступ до окремого інтерфейсу, де можуть керувати рейсами, користувачами та переглядати статистику. Для оновлення особистої інформації чи зміни пароля передбачено розділ «Профіль», а в разі питань доступна форма зворотного зв’язку.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було створено сучасний веб-застосунок для продажу автобусних квитків із використанням таких технологій, як Spring Boot, JSP/JSTL, Bootstrap, MySQL та інтеграція з платіжними системами. Усі ключові етапи – від аналізу ринку та вибору оптимального технологічного стеку до проєктування архітектури, розробки клієнтської та серверної частин, налаштування системи управління рейсами, користувачами й квитками, а також реалізації адміністративної панелі – були послідовно виконані.

Розроблений застосунок забезпечує зручну взаємодію користувачів із системою, включаючи перегляд розкладу рейсів, пошук і бронювання квитків, швидку авторизацію та можливість онлайн-оплати. Адміністративна частина платформи дозволяє ефективно керувати перевезеннями та обслуговувати пасажирів. Завдяки адаптивному інтерфейсу та надійній системі зберігання даних, додаток може бути зручно використаний на різних пристроях і масштабований для реального комерційного використання.

Окрему увагу варто звернути на потенційні шляхи розвитку проєкту, зокрема інтеграцію елементів машинного навчання для подальшої оптимізації роботи системи. Застосування методів ML у сфері управління інформаційними потоками, зокрема для аналізу поведінки користувачів, прогнозування попиту на маршрути, виявлення аномалій у трафіку або навантаженні на систему, може суттєво підвищити рівень адаптивності та безпеки. Такі підходи вже активно застосовуються в IT-інфраструктурах, як-от у дата-центрах та транспортних системах, і можуть стати наступним кроком у розвитку веб-застосунку.

Таким чином, реалізація проєкту не лише продемонструвала ефективність сучасних веб-технологій у транспортній галузі, а й відкрила перспективи для впровадження інтелектуальних рішень на базі машинного

навчання. Це підтверджує готовність автора до подальшої професійної діяльності у сфері інформаційних систем та цифрової трансформації транспортної інфраструктури.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Busfor. [Електронний ресурс] // Режим доступу: https://busfor.ua/ (дата звернення: 21.05.2025).

2. Tickets.ua. [Електронний ресурс] // Режим доступу: https://tickets.ua/ (дата звернення: 21.05.2025).

3. GoFo. [Електронний ресурс] // Режим доступу: https://gofo.com.ua/ (дата звернення: 21.05.2025).

4. JavaServer Pages Technology. Oracle Documentation. [Електронний ресурс] // Режим доступу: https://docs.oracle.com/javaee/7/tutorial/jsf-intro.htm (дата звернення: 21.05.2025).

5. Thymeleaf. Офіційна документація. [Електронний ресурс] // Режим доступу: https://www.thymeleaf.org/ (дата звернення: 21.05.2025).

6. React – офіційний сайт. [Електронний ресурс] // Режим доступу: https://react.dev/ (дата звернення: 21.05.2025).

7. Bootstrap – Офіційна документація. [Електронний ресурс] // Режим доступу: https://getbootstrap.com/ (дата звернення: 21.05.2025).

8. Spring Framework – Documentation. [Електронний ресурс] // Режим доступу: https://spring.io/projects/spring-framework (дата звернення: 21.05.2025).

9. Spring Boot – Documentation. [Електронний ресурс] // Режим доступу: https://spring.io/projects/spring-boot (дата звернення: 21.05.2025).

10. MySQL – Офіційний сайт. [Електронний ресурс] // Режим доступу: https://www.mysql.com/ (дата звернення: 21.05.2025).

11. Hibernate ORM – Documentation. [Електронний ресурс] // Режим доступу: https://hibernate.org/orm/documentation/ (дата звернення: 21.05.2025).

12. Log4j2 – Documentation. [Електронний ресурс] // Режим доступу: https://logging.apache.org/log4j/2.x/manual/configuration.html (дата звернення: 21.05.2025).

13. Apache Tomcat – Documentation. [Електронний ресурс] // Режим доступу: https://tomcat.apache.org/ (дата звернення: 21.05.2025).

14. IntelliJ IDEA – Documentation. [Електронний ресурс] // Режим доступу: https://www.jetbrains.com/idea/ (дата звернення: 21.05.2025).

15. Fowler M. Patterns of Enterprise Application Architecture. – Addison-Wesley, 2002.

16. Freeman E., Robson E. Head First Design Patterns. – O'Reilly Media, 2020.

17. Лебідь І.С., Сітніков В.І Мультиагентні системи та їх застосування в логістиці та економіці Сучасні напрями розвитку інформаційнокомунікаційних технологій та засобів управління: тези доповідей п’ятнадцятої міжнародної науково-технічної конференції. Т.2: секція 2. Баку: ІСУ АР; Харків: НТУ «ХПІ»;. Харків: ХНУРЕ; Харків: НАУ «ХАІ»; Жиліна: УмЖ. 2025. С. 49