

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
ДЕРЖАВНИЙ ЗАКЛАД  
„ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ  
ІМЕНІ ТАРАСА ШЕВЧЕНКА”

Навчально-науковий інститут фізики, математики та інформаційних  
технологій

Кафедра фізико-технічних систем та інформатики

Попов Денис Дмитрович

**СИСТЕМА БРОНЮВАННЯ ЗАЛІЗНИЧНИХ КВИТКІВ**

**кваліфікаційна робота**

**здобувача вищої освіти першого (бакалаврського) рівня**

**освітньої програми «комп'ютерні науки»**

**за спеціальністю 122 комп'ютерні науки**

Особистий підпис \_\_\_\_\_



Попов Денис

Науковий керівник \_\_\_\_\_

Юрій Козуб, д.т.н., доцент

Завідувач кафедри \_\_\_\_\_

Юрій Козуб, д.т.н., доцент

Полтава – 2023

Міністерство освіти і науки України  
Державний заклад „Луганський національний університет  
імені Тараса Шевченка”

Факультет (інститут)

Навчально-науковий інститут фізики,  
математики та інформаційних технологій

Кафедра

фізико-технічних систем та інформатики

Рівень освіти

перший (бакалаврський)

Спеціальність

122 «комп'ютерні науки»

(код, назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри ФТСІ

Юрій Козуб

(підпис)

(ім'я, прізвище)

“ ” 2023 р.

## ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Попову Денису Дмитровичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) система оформлення залізничних квитків

Керівник кваліфікаційної роботи

Козуб Ю.Г.

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджена наказом по університету

« » 2023 року №

2. Строк подання студентом проекту (роботи)

3. Вихідні дані до роботи (проекту) у результаті виконання роботи

повинно бути розробити систему бронювання залізничних квитків

(визначаються кількісні або (та) якісні показники, яким повинен відповідати об'єкт розробки)

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

(визначаються назви розділів або (та) перелік питань, які повинні увійти до тексту ПЗ)

**6. Консультанти розділів проекту (роботи)**

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання „\_\_\_\_\_” \_\_\_\_\_ 2023 р.

**КАЛЕНДАРНИЙ ПЛАН**

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
	Вибір теми роботи, вивчення наукової літератури, затвердження теми та керівника.	До 1 березня	
	Аналіз літературних джерел за темою роботи. Розробка та апробація методики дослідно-експериментальної роботи. Подання структури теоретичної частини роботи та плану експериментальних досліджень.	До 20 березня	
	Робота над теоретичною частиною. Подання теоретичної частини роботи для першого читання науковим керівником.	До 1 квітня	
	Усунення зауважень, урахування рекомендацій наукового керівника. Подання теоретичної частини роботи на друге читання.	До 15 квітня	
	Проведення експериментальної роботи. Поетапний аналіз та обговорення її результатів. Перевірка стану виконання роботи.	До 30 квітня	
	Урахування рекомендацій наукового керівника, усунення недоліків, підготовка варіанта роботи до передзахисту. Розробка презентації.	До 15 травня	
	Попередній захист роботи на кафедрі	До 30 травня	
	Доопрацювання роботи з урахуванням рекомендацій після передзахисту. Подання роботи науковому керівникові та рецензентові на підготовку відгуку та рецензії	За 10 днів до державної атестації	
	Подання на кафедру остаточного варіанта роботи, переплетеного та підписаного автором, науковим керівником і рецензентом.	За 5 днів до державної атестації	

Студент

підпис

Попов Денис

Керівник проекту (роботи)

Юрій Козуб

## **АНОТАЦІЯ**

**Попов Денис**

**Тема: система оформлення залізничних квитків**

**Спеціальність: 122 "комп'ютерні науки"**

**Установа: ЛНУ імені Тараса Шевченка, 2023 р.**

**Бакалаврська робота містить: 66 с., 5 додат., 23 джерел.**

**Об'єкт дослідження: система бронювання залізничних квитків**

**Предмет дослідження: проектування системи бронювання залізничних квитків з урахуванням вимог до функціональних можливостей, зручності, безпеки та можливості масштабування.**

**Мета роботи – розробити систему бронювання залізничних квитків**

**Результати.** Основна мета роботи – це розробка та впровадження інформаційної системи бронювання залізничних квитків. Для досягнення цієї мети було розроблено клієнтський додаток на мові програмування Java в середовищі розробки IntelliJ IDEA. Програма розроблена для взаємодії з MongoDB, розподіленою нереляційною базою даних.

**Висновки.** Отриманий програмний продукт дозволяє здійснювати бронювання залізничних квитків, включаючи вибір місць для пасажирів, авторизацію у системі, контроль доступу користувачів, скасування бронювання квитків, перегляд доступних для бронювання квитків, реєстрацію користувачів і відстеження заброньованих квитків. Система розроблена таким чином, щоб бути масштабованою та підтримувати обробку великих даних.

**Ключові слова:** ІНФОРМАЦІЙНА СИСТЕМА, ОБРОБКУ ДАНИХ, ЗАЛІЗНИЧНІ КВИТКИ, НЕРЕЛЯЦІЙНА БАЗА ДАНИХ, РОЗПОДІЛЕНА БАЗА ДАНИХ, БРОНЮВАННЯ КВИТКІВ, JAVA.

## ABSTRACT

**Popov Denis**

**Subject:** system for issuing cash receipts

**Specialty:** 122 "computer science"

**Statement:** LNU named after Taras Shevchenko, 2023

**Bachelor's work:** 66 pages., 5 additional, 23 source.

**Object of follow-up:** booking system for air tickets

**Subject of study:** the design of the system for armoring the air tickets with improvement was able to achieve functional capabilities, efficiency, security and scalability.

**Meta robots** - expand the system for booking air tickets

**Results of research.** The present thesis comprises – pages and consists of – figures, – tables, and – references. Its main objective is to design and implement an information system for booking railway tickets. To achieve this goal, a client application was developed using the Java programming language within the IntelliJ IDEA development environment. The application was designed to interact with MongoDB, a distributed and non-relational database.

**Conclusions.** The resulting software product allows for the booking of railway tickets, including seat selection for passengers, system authorization, user access control, ticket reservation cancellation, viewing of available tickets for reservation, registration of users, and tracking of reserved tickets. The system is designed to be scalable, accommodating big data processing requirements.

**Key words:** INFORMATION SYSTEM, DATA PROCESSING, RAILWAY TICKETS, NON-RELATIONAL DATABASE, DISTRIBUTED DATABASE, TICKET RESERVATION, JAVA.

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

HTML –	Hypertext Markup Language – мова розмітки гіпертекстових документів
XML –	Extensible Markup Language – розширювана мова розмітки
DTO –	Data Transfer Object – шаблон оформлення для передачі даних між підсистемами програми
DAO –	Data Access Object – Об’єкт доступу до даних
SQL –	Структурована мова запитів – стандартна мова для керування реляційними базами даних
JSP –	Java Servlet Page – технологія створення динамічних HTML-сторінок
JSON –	JavaScript Object Notation – файл, який має простий формат для обміну даними
POM –	Об’єктна модель проекту – XML-файл, який зберігає всю інформацію про проект і налаштування
Sharding –	Метод розподілу даних між декількома машинами для підтримки розгортань із великими наборами даних і операцій із високим рівнем пропускнуої здатності

## ЗМІСТ

<b>ВСТУП</b>	<b>РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ ОРГАНІЗАЦІЇ СИСТЕМИ БРОНЮВАННЯ ЗАЛІЗНИЧНИХ КВИТКІВ</b>	
	121.1. Поняття системи бронювання	
	121.2. Структура мережі системи бронювання	14
	1.3. Класифікація систем бронювання	16
	1.4. Комп'ютерні мережеві технології, що використовуються в системах бронювання	18
	1.5. Особливості функціонування систем бронювання в залізничній галузі	19
<b>РОЗДІЛ 2. МЕТОДОЛОГІЯ РОЗРОБКИ ТА ПРОЕКТУВАННЯ СИСТЕМИ</b>		
	222.1 Вибір методів та технологій для розробки системи	
	222.2. Архітектура та конфігурація компонентів системи	31
<b>РОЗДІЛ 3. ТЕСТУВАННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ</b>		
	373.1. Тестування системи бронювання	
	373.2. Особливості взаємодії користувача з розробленою системою	44
	<b>ВИСНОВКИ</b>	
<b>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ</b>		52

## ВСТУП

У сучасний час підприємства стикаються з проблемою оптимізації своїх систем зберігання та керування даними, щоб краще розуміти своїх клієнтів, адаптуватися до мінливих очікувань користувачів і перевершити своїх конкурентів на ринку за допомогою нових бізнес-моделей і програм. Це призвело до значного відхилення від припущень, які лежали в основі розвитку традиційних систем бронювання.

Нові реалії цифрового ландшафту, які характеризуються вищими вимогами до продуктивності і швидким виходом на ринок, вимагають більш гнучких методологій, мікросервісів і DevOps. Розвиток нових систем бронювання є відповіддю на потребу в управлінні величезним збільшенням нових типів даних, які швидко змінюються, які генеруються новими веб-, мобільними, соціальними та IoT-додатками. Крім того, розподілені системи та хмарні обчислення дозволили розробникам скористатися перевагами обчислень на вимогу та інфраструктури зберігання, маючи можливість обслуговувати аудиторію, де б вона не була, навіть з огляду на суверенітет даних. Отже, нереляційні бази даних, прикладом яких є MongoDB, з'явилися як рішення для цих проблем, забезпечуючи більшу гнучкість для модернізації існуючих робочих навантажень і розробки нових програм.

У цьому контексті у роботі висвітлюється використання нереляційної розподіленої бази даних для роботи з великими даними на рівні бази даних при реалізації інформаційної системи бронювання залізничних квитків.

Мета дослідження полягає в розробці надійної та ефективної системи бронювання квитків, яка може обробляти великий обсяг транзакцій і забезпечувати безперебійну роботу з клієнтами.

З метою виконання поставленої мети, були визначені наступні завдання:

- розглянути поняття комп'ютерних систем бронювання, основні компоненти та особливості функціонування;
- проаналізувати стек технологій для розробки інформаційно масштабованої клієнт-серверної системи обробки даних бронювання залізничних квитків.
- розробити клієнт-серверну програму для підвищення ефективності обробки даних у системі бронювання залізничних квитків.
- деталізувати структуру та конфігурацію програмної складової системи та продемонструвати її роботу на різних рівнях доступу до кожного модуля;
- надати інструкцію щодо використання веб-інтерфейсу системи бронювання залізничних квитків.

Об'єкт дослідження – моделі систем бронювання залізничних квитків.

Предмет дослідження – конфігурація та тестування системи бронювання залізничних квитків з використанням нереляційної розподіленої бази даних.

Наукова новизна роботи полягає в застосуванні нереляційної розподіленої бази даних при розробці системи бронювання залізничних квитків. Цей підхід дозволяє керувати великим обсягом даних, що швидко змінюються, створених новими веб-додатками, мобільними додатками, соціальними додатками та додатками Інтернету речей. Використання нереляційних баз даних, прикладом яких є MongoDB, забезпечує більшу гнучкість для модернізації існуючих робочих навантажень і розробки нових програм.

Інноваційна новизна роботи полягає в інтеграції гнучких методологій, мікросервісів та DevOps у розробку системи бронювання залізничних квитків. Такий підхід дозволяє створити надійну та ефективну систему бронювання квитків, яка може обробляти великий обсяг транзакцій і забезпечувати безперебійну взаємодію з клієнтами.

Практичною новизною роботи є розробка системи бронювання залізничних квитків, яка використовує нереляційну розподілену базу даних для роботи з великими даними. Практичне застосування цієї системи має потенціал для оптимізації систем зберігання та управління даними для підприємств, дозволяючи їм краще розуміти своїх клієнтів, адаптуватися до мінливих очікувань користувачів і перевершити своїх конкурентів на ринку завдяки новим бізнес-моделям і додаткам.

## **РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ ОРГАНІЗАЦІЇ СИСТЕМИ БРОНЮВАННЯ ЗАЛІЗНИЧНИХ КВИТКІВ**

### **1.1. Поняття системи бронювання**

Поняття системи бронювання відноситься до системи, призначеної для полегшення бронювання товарів, послуг або місць клієнтами, користувачами або гостями певного підприємства, події тощо. Ця система широко використовується в різних галузях, таких як готельний бізнес, транспорт і розваги.

Концепція системи бронювання бере свій початок на початку ХХ століття, коли такі підприємства, як готелі та авіакомпанії, почали масово використовувати ручні системи для керування бронюваннями. Ці ручні системи передбачали використання паперових записів, бухгалтерських книг і телефонного зв'язку, що робило процес керування бронюваннями трудомістким і схильним до помилок [4].

У 1960-х роках поява комп'ютерних технологій привела до розвитку електронних систем бронювання, які автоматизували процес бронювання та зробили його більш ефективним. Першу електронну систему бронювання було розроблено American Airlines у 1960 році, і вона дозволяла туристичним агентам робити бронювання в електронному вигляді, а не дзвонити в офіс авіакомпанії [8].

Протягом наступних кількох десятиліть використання електронних систем бронювання стало все більш поширеним у туристичній індустрії, а інші авіакомпанії та готелі розробляли власні системи. Ці системи спочатку базувалися на мейнфреймах і були доступні лише для туристичних агентів та інших професіоналів галузі [5].

У 1990-х роках з початком поширення Інтернету, почали формуватись і системи онлайн-бронювання, які дозволяли клієнтам компаній робити бронювання безпосередньо через веб-сайти, а не

звертатися до турагента чи телефонувати в офіс бронювання [6]. Інтернет також дав можливість компаніям пропонувати в реальному часі інформацію про доступність і ціни, а також онлайн-оплату та підтвердження бронювань.

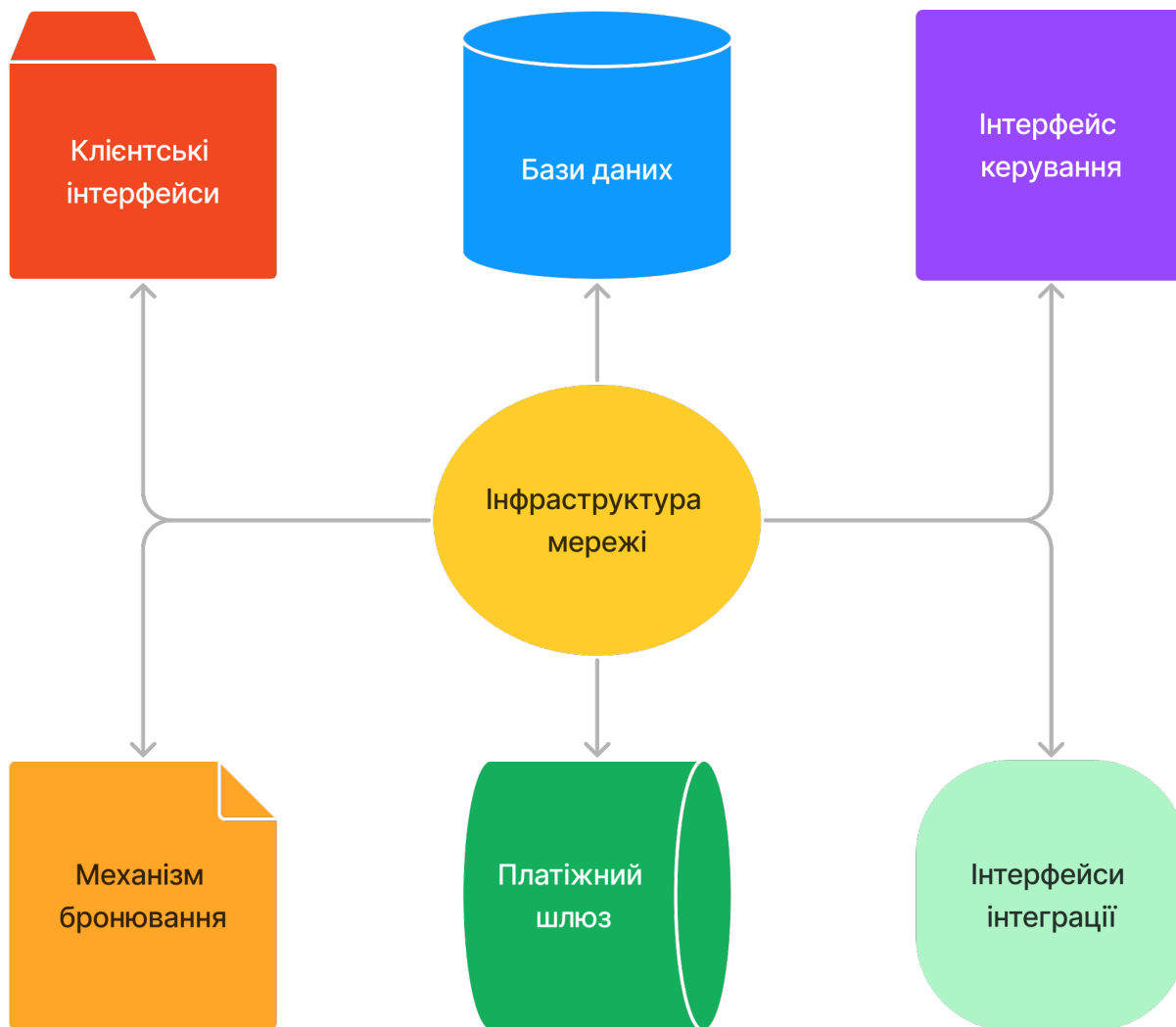
Таким чином, система бронювання – це також комп'ютеризована система, яка дозволяє клієнтам робити бронювання онлайн, по телефону або особисто. Ця система допомагає керувати бронюваннями, скорочувати час очікування та підвищувати рівень задоволеності клієнтів. Це дозволяє підприємствам ефективно розподіляти ресурси, краще планувати та оптимізувати отримання прибутку.

Традиційно система бронювання працює, дозволяючи клієнтам вибрати бажану дату та час для бронювання, кількість необхідних місць, а також будь-які додаткові послуги. Потім система обробляє запит і формує підтвердження бронювання. Система бронювання також може надавати клієнтам оновлення в режимі реального часу щодо наявності, цін і будь-яких змін або скасування бронювання.

Сучасні системи бронювання використовуються в багатьох галузях, включаючи готельний бізнес, транспорт, розваги та охорону здоров'я. Вони перетворилися на складні хмарні платформи, які пропонують такі функції, як мобільне бронювання, інтеграцію соціальних медіа та аналіз даних. Ці системи відіграють вирішальну роль у допомозі компаніям керувати бронюваннями, оптимізувати розподіл ресурсів і покращувати взаємодію з клієнтами, тим самим сприяючи успіху бізнесу та є ключовим компонентом будь-якого сучасного бізнесу.

## 1.2. Структура мережі системи бронювання

Структура мережі системи бронювання залежить від конкретної реалізації та вимог системи. Однак є деякі загальні компоненти, які зазвичай присутні в мережі системи бронювання (рис. 1.1).



**Рис. 1.1. Основні компоненти мережі системи бронювання**

Джерело: розроблено автором [1].

1. Клієнтські інтерфейси – це інтерфейси, через які кінцеві користувачі або клієнти отримують доступ до системи бронювання. Такими інтерфейсами можуть бути веб-сайт або мобільний додаток. Клієнти використовують ці інтерфейси для перевірки наявності місць, бронювання та керування бронюваннями.

2. База даних – центральний компонент мережі системи бронювання, який зберігає всю інформацію про бронювання, включаючи інформацію про клієнтів, деталі бронювання, ціни та наявність. База даних зазвичай розміщується на сервері, і до неї мають доступ різні компоненти системи бронювання.

3. Механізм бронювання – це компонент, який обробляє запити на бронювання та оновлює базу даних відповідною інформацією. Механізм бронювання перевіряє наявність ресурсів, розраховує ціни та надсилає клієнтам електронні листи з підтвердженням.

4. Платіжний шлюз – це компонент, який обробляє платежі для системи бронювання. Він взаємодіє зі сторонніми платіжними сервісами та безпечно обробляє платіжну інформацію, надану клієнтами.

5. Інтерфейс керування – це компонент, який дозволяє системним адміністраторам керувати системою бронювання. Цей інтерфейс надає такі функції, як керування ресурсами, керування цінами, звітування, управління ролями в системі тощо.

6. Інтерфейси інтеграції. Системі бронювання може знадобитися інтеграція з іншими системами, такими як системи управління майном, системи управління взаємовідносинами з клієнтами або системи бухгалтерського обліку. Інтерфейси інтеграції дозволяють цим системам обмінюватися інформацією та безперебійно працювати разом.

7. Інфраструктура мережі – апаратні та програмні компоненти, які забезпечують зв'язок між різними компонентами мережі системи бронювання: маршрутизатори, комутатори, сервери та інше мережеве обладнання.

Підсумовуючи, мережева структура системи бронювання є складною та включає різні компоненти, які працюють разом, щоб забезпечити безперебійне бронювання для клієнтів. Мережа системи

резервування повинна бути розроблена та ретельно підтримуватися, щоб забезпечити надійність, безпеку та масштабованість системи.

### 1.3. Класифікація систем бронювання

Системи бронювання можна класифікувати за різними критеріями, такими як галузь, призначення, функціональність і технологія (рис. 1.2).



**Рис. 1.2. Основні системи класифікації систем бронювання**

Джерело: розроблено автором [3].

Класифікація за галузевою ознакою: системи бронювання можна класифікувати за галуззю, у якій вони використовуються. Наприклад, системи бронювання, системи бронювання готелів, авіакомпаній, ресторанів тощо.

Класифікація за призначенням: системи бронювання можна класифікувати за призначенням. Наприклад, деякі системи бронювання призначені для керування бронюванням ресурсів, тоді як інші надають користувачам доступ до системи реєстрації на певну подію або планування онлайн-зустрічей.

Класифікація на основі функціональних можливостей: системи бронювання можна класифікувати на основі їхньої функціональності. Наприклад, деякі системи пропонують інформацію про доступність і ціни в реальному часі, тоді як інші надають розширені функції, такі як автоматична реєстрація, програми лояльності та персоналізовані рекомендації [12].

Класифікація на основі технологій: системи бронювання можна класифікувати на основі технології, яка використовується для їх впровадження. Наприклад, деякі системи використовують архітектуру клієнт-сервер, а інші використовують хмарну технологію. Деякі системи розроблено з використанням запатентованої технології, тоді як інші використовують програмне забезпечення з відкритим кодом.

Класифікація на основі доступу: системи бронювання також можна класифікувати на основі типу доступу, який вони пропонують. Наприклад, до деяких систем бронювання можна отримати доступ будь-хто через Інтернет, інші – доступні тільки для внутрішнього використання персоналом певної організації.

Таким чином, класифікація систем бронювання базується на різних критеріях, таких як галузь, призначення, функціональність, технологія та доступ. Розуміння класифікації систем бронювання є важливим для вибору правильної системи для конкретного бізнесу чи галузі.

#### **1.4. Комп'ютерні мережеві технології, що використовуються в системах бронювання**

Системи бронювання є невід'ємним компонентом багатьох галузей і використовуються компаніями для автоматизації та оптимізації процесу бронювання. З розвитком технологій системи бронювання розвинулися, щоб включити технології комп'ютерних мереж, які забезпечують безперебійний зв'язок і передачу даних між різними системами та пристроями.

Однією з основних комп'ютерних мережевих технологій, що використовуються в системах бронювання, є Інтернет. Інтернет дозволяє системам бронювання бути доступними для глобальної аудиторії, полегшуючи процес бронювання для клієнтів з будь-якої точки світу. Інтернет також дозволяє оновлювати та синхронізувати дані в режимі реального часу, забезпечуючи точність і актуальність даних бронювання на всіх системах і пристроях.

Ще одна важлива технологія, яка використовується в сучасних системах бронювання, – це хмарні обчислення. Хмарні обчислення дозволяють розміщувати системи бронювання на віддалених серверах, зменшуючи потребу у дорогому апаратному та програмному забезпеченні. Ця технологія також забезпечує швидке масштабування, дозволяючи системам бронювання обробляти великі обсяги бронювань і транзакцій у пікові періоди [7].

Окрім Інтернету та хмарних обчислень, системи бронювання також включають різні мережеві протоколи та стандарти для полегшення зв'язку

між різними системами та пристроями. Наприклад, системи резервування використовують такі протоколи, як TCP/IP, HTTP та SMTP, для передачі даних між серверами та пристроями. Такі стандарти, як XML і JSON, також використовуються для забезпечення передачі даних у стандартизованому форматі, який можна легко інтерпретувати та обробляти різними системами [11].

Нарешті, системи бронювання також використовують різні технології безпеки для захисту конфіденційних даних і запобігання несанкціонованому доступу. Ці технології безпеки включають шифрування, брандмауери та системи виявлення вторгнень, які разом забезпечують конфіденційність і безпеку даних резервування та користувачів.

Отже, комп'ютерні мережеві технології є важливим компонентом сучасних систем бронювання. Інтернет, хмарні обчислення, мережеві протоколи та технології безпеки використовуються для забезпечення безперебійного зв'язку та передачі даних між різними системами та пристроями, покращуючи ефективність і ефективність систем бронювання. Оскільки технологія продовжує розвиватися, цілком імовірно, що системи бронювання будуть включати ще більш просунуті технології комп'ютерних мереж для подальшого розширення своїх можливостей і функціональності.

### **1.5. Особливості функціонування систем бронювання в залізничній галузі**

Системи бронювання в залізничній галузі значно розвинулися протягом багатьох років, а їхня робота характеризується декількома ключовими особливостями, які забезпечують ефективні та доступні процеси бронювання та продажу квитків.

Однією з ключових особливостей систем бронювання в залізничній галузі є їх здатність обробляти великі обсяги бронювань і транзакцій. Це особливо важливо в індустрії з великим обсягом та швидкість обміну даних, де сотні або навіть тисячі пасажирів можуть бронювати квитки одночасно. Системи резервування повинні бути в змозі обробляти цей обсяг без шкоди для продуктивності та без затримок.

Іншою важливою особливістю систем бронювання в залізничній галузі є їх здатність керувати запасами квитків і ціноутворенням. Системи бронювання включають алгоритми та інструменти, які аналізують історичні дані бронювання та тенденції ринку, щоб визначити оптимальні рівні запасів і стратегії ціноутворення для різних маршрутів і класів подорожей. Це дає змогу залізницям максимізувати прибуток, одночасно гарантуючи доступність місць для клієнтів за прийнятними цінами.

Третьою особливістю систем бронювання в сучасній залізничній галузі є їх інтеграція з іншими системами та послугами. Наприклад, системи бронювання можуть бути інтегровані з платіжними шлюзами, системами управління взаємовідносинами з клієнтами (CRM) та іншими серверними системами для забезпечення безперебійного та ефективного процесу бронювання та продажу квитків. Ця інтеграція дозволяє пасажирам бронювати квитки, здійснювати платежі та отримувати підтвердження та сповіщення через єдину платформу.

Крім того, системи бронювання в залізничній галузі також повинні бути здатні виконувати складні вимоги щодо маршрутизації та планування. Залізниці працюють на складній мережі шляхів і станцій, і системи бронювання повинні мати можливість орієнтуватися в цій мережі, щоб знаходити оптимальні маршрути та розклади для пасажирів. Для цього потрібні складні алгоритми та інструменти, які можуть оптимізувати такі фактори, як час у дорозі, вартість і зручність.

Системи бронювання в залізничній галузі також повинні мати можливість обробляти спеціальні запити та розміщення (різні класи пасажирських місць, додаткові послуги, врахування потреб осіб з обмеженими можливостями тощо).

Підсумовуючи, системи бронювання в залізничній галузі характеризуються кількома ключовими особливостями, включаючи їх здатність обробляти великі обсяги бронювань, керувати запасами квитків і ціноутворенням, інтегруватися з іншими системами та послугами, обробляти складні вимоги щодо маршрутизації та планування, а також задовольняти спеціальні запити та розміщення. Ці функції дозволяють залізницям надавати ефективні послуги бронювання та продажу квитків пасажирам, сприяючи позитивному досвіду подорожі.

## **РОЗДІЛ 2. МЕТОДОЛОГІЯ РОЗРОБКИ ТА ПРОЕКТУВАННЯ СИСТЕМИ**

### **2.1 Вибір методів та технологій для розробки системи**

Для реалізації архітектурного рівня було обрано інформаційну систему бронювання залізничних квитків з можливістю масштабування сервісу. Використовуючи сучасні програмні технології, необхідно розробити систему з використанням нереляційної бази даних на основі горизонтального розподілу даних.

Розроблена система повинна мати такі критерії:

- масштабування системи при необхідності збільшення ресурсів обробки даних;
- швидка обробка запитів, оскільки розповсюдження самої системи сприяє підвищенню її продуктивності;
- клієнт-серверний шаблон ПЗ.

У роботі буде продемонстровано використання власного програмного продукту для автоматизації бронювання залізничних квитків, а також сучасної архітектурної моделі рівня бази даних для обробки великих обсягів даних.

Програмний продукт повинен мати такі функції:

- авторизація в системі;
- перегляд можливості бронювання залізничних квитків;
- можливість надання різних рівнів доступу до (залізничної) квиткової системи;
- огляд користувачів системи.

При розробці подібної системи важливим аспектом є вибір відповідних технологій для створення інформаційної системи бронювання залізничних квитків. Інструменти, зображені на рис. 2.1, були використані для розробки цієї системи.



**Рис. 2.1. Інструменти розробки системи**

Джерело: розроблено автором на основі джерела [2].

У розробці програми інформаційної системи бронювання залізничних квитків було використано низку технологій, серед яких:

- середовище розробки IntelliJ IDEA;
- мова програмування Java для сервера [16];
- мова гіпертекстової розмітки HTML з використанням технології JSP для організації веб-інтерфейсу;
- мова розмітки CSS з використанням бібліотеки Bootstrap для розробки графічного інтерфейсу користувача [23];
- технологія забезпечення автоматизації складання проекту Maven;
- Apache Tomcat для розгортання програми на локальному сервері [9];
- Git для версії розробленої системи;
- база даних MongoDB;
- реляційна база даних MySQL [21];

- Junit, призначена для написання та виконання тестів на мові програмування Java;
- PowerMock, як генератор макетів об'єктів під час тестування [15].

### **Мова програмування Java**

Мова програмування Java високого рівня була використана для написання архітектури рівня бізнес-логіки (системних процесів) завдяки своїй об'єктно-орієнтованій природі. Вибір платформи Java та її реалізації для продукту був мотивований доступністю численних технологій і фреймворків, розроблених для вирішення широкого кола практичних завдань. Однією з ключових переваг Java є її віртуальна машина (JVM), яка забезпечує кросплатформенну сумісність. Це означає, що програма, створена на Java і скомпільована в байт-код, може бути виконана на будь-якій іншій платформі, яка підтримує віртуальну машину Java, при цьому JVM виступає як рівень абстракції між кодом і обладнанням.

Крім того, мова програмування Java є кращим вибором через наявність вбудованого збирача сміття, який може ідентифікувати та видаляти об'єкти, на які більше не посилається програма. Хоча збирач сміття запобігає витоку пам'яті, розробка цього програмного продукту виявила недолік роботи з цією перевагою, а саме те, що процес вимагає додаткової оптимізації під час генерації та імпорту даних у базу даних.

Як вже згадувалось раніше, Java – це об'єктно-орієнтована мова високого рівня з сильною типізацією, що підвищує її надійність. Java надає різні зручні інструменти для програмістів, такі як Java API, який є набором команд і методів, включаючи підключення до бази даних, що було пріоритетом у розробці цієї інформаційної системи. Парадигма об'єктно-орієнтованого програмування (ООП) наділила мову такими властивостями, як масштабованість, що дозволяє багаторазово розширювати розроблену систему. Систему можна масштабувати таким

чином, щоб додавання нових компонентів не вимагало змін існуючих. Крім того, перевагою цього підходу є можливість багаторазового повторного використання написаного коду, що значно зменшує кількість написаного коду в цілому.

### **Фреймворк Apache Maven**

Apache Maven – це фреймворк, розроблений для автоматизації збирання проектів на основі опису їх структури у файлах POM (Plain Object Model). Файли POM – це XML-файли, які містять інформацію про проект, конфігурацію та залежності (додаткові структури).

Maven охоплює стандартну структуру каталогів за замовчуванням, що є дуже практичним для сумісного програмування. Основний каталог `src` складається з двох каталогів: основного та тестового (`main & test`). Maven використовує набір фаз для керування життєвим циклом проекту, що зручно під час створення проектів. Такі команди, як `clean`, яка очищає проект, і `install`, яка встановлює програмне забезпечення в локальний репозиторій Maven для доступу розробленого програмного забезпечення для інших проектів поточного користувача, часто використовувалися під час збирання цього проекту.

### **Мова розмітки HTML**

Щоб розробити зовнішній компонент цієї системи та представити його у вигляді веб-сторінки, було використано мову розмітки гіпертексту (HTML). Ця технологія є однією з найбільш широко використовуваних і фундаментальних технологій для створення будь-якої веб-сторінки. Зручна структура написання сторінки в HTML полегшує додавання сторінок, необхідних для масштабування системи, а наявність таких атрибутів, як `id` і `class`, є перевагою для рефакторингу [13].

## **Технологія JSP**

Замість статичного вмісту, який є незмінним, було представлено Java Servlet для створення динамічного веб-вмісту, який відповідає на запити користувачів, наприклад пошукові запити. Однак використання сервлету для створення презентабельної HTML-сторінки не є оптимальним рішенням, а модифікація згенерованої HTML-сторінки не вважається найкращою практикою для написання презентаційного рівня програмного продукту.

Саме тому, поява технології JSP (JavaServlet Page) спрощує поєднання динамічного та статичного веб-контенту. Подібно до популярного ASP (Microsoft Active Server Pages), JSP також забезпечує елегантний підхід до поєднання динамічного та статичного веб-вмісту. Основна сторінка написана на мові розмітки HTML, а наявність спеціальних тегів забезпечує вставку фрагментів програмного коду Java. Таким чином, JSP не є заміною JavaServlet; скоріше, це різні підходи до представлення продукту. Хоча Servlet передбачає використання мови розмітки HTML у коді Java, JSP передбачає використання Java у середині структури HTML. Іншими словами, JSP є більш зручним способом роботи з представленням веб-сторінки, ніж сервлет, але він менш потужний.

## **Мова розмітки CSS**

Мова розмітки каскадних таблиць стилів (CSS) – це спеціалізована мова, яка використовується для розробки сторінок, написаних мовами розмітки даних. CSS використовується для покращення візуальної привабливості веб-сторінок. Він працює зі шрифтами, кольорами, висотою, шириною та розташуванням елементів, таким чином надаючи структуру зовнішньому вигляду вмісту на веб-сторінці.

## **Розподілена нереляційна база даних**

Застосування технології NoSQL в контексті інформаційної системи, орієнтованої на бронювання залізничних квитків, пов'язане в першу чергу зі зберіганням великої кількості даних. Такі системи мають високий ступінь масштабованості та здатні ефективно обробляти великі обсяги даних, а також забезпечувати підтримку напівструктурованих і гнучких даних

Системи NoSQL переважно покладаються на прості операції читання-запису, що включають 27 основних операцій, продуктивність яких є критичним фактором. На даний момент існує більше ста систем NoSQL, кожна з яких характеризується різними моделями даних і API для доступу до даних.

Бази даних NoSQL можна класифікувати за чотирма основними категоріями, кожна з яких підходить для конкретних завдань, а саме:

- сховища ключ-значення (наприклад, SimpleDB);
- бази даних, орієнтовані на документи (наприклад, CouchDB, MongoDB);
- сховища широких стовпців або сімейства стовпців (наприклад, Cassandra, HBase, Big Table);
- бази даних графів (наприклад, InfoGrid, InfiniteGraph, Sones GraphDB).

Значна неоднорідність у цих категоріях породила нові виклики в області розробки даних NoSQL, причому рішення щодо моделювання значно впливають на вимоги до якості. Крім того, повністю формалізована процедура моделювання даних для систем NoSQL ще не існує, тому що у системах NoSQL розподіл даних відіграє вирішальну роль, і різні дослідницькі підходи запровадили ефективні алгоритми та методи розподілу даних.

## **База даних MongoDB**

Це дослідження має на меті представити стратегію моделювання, яка полегшує використання MongoDB, документно-орієнтованої системи керування базами даних. Як випливає з назви, документоорієнтовані бази даних були створені з метою керування та зберігання документів. Ці документи кодуються за допомогою стандартних форматів обміну даними, таких як JSON (об'єктна нотація JavaScript), XML або BSON (двійковий JSON).

У базах даних документів стовпець значення складається з пар атрибутів ім'я/значення для певних даних, які є напівструктурованими. Один стовпець може містити велику кількість таких атрибутів, причому кількість і тип записаних атрибутів потенційно змінюються від одного рядка до іншого. На відміну від простих сховищ ключів і значень, і ключі, і значення доступні для пошуку в базі даних документів.

Бази даних документів MongoDB в основному використовуються для зберігання та керування великими колекціями даних, таких як текстові документи, повідомлення електронної пошти та XML-документи, а також концептуальні «документи», як-от денормалізовані (сукупні) представлення об'єкта бази даних, наприклад продукту або клієнту. Крім того, вони також підходять для зберігання загальних даних, тобто нерегулярних (напівструктурованих) даних, які вимагають широкого використання «нульових значень» у системі керування реляційною базою даних (де нульові значення функціонують як заповнювачі для відсутніх або неіснуючих значень) [20].

Управління великими наборами даних або високопродуктивними програмами в системах баз даних може становити серйозну проблему для одного сервера. Наприклад, висока частота запитів може виснажити ресурси процесора сервера. Крім того, коли розмір робочого набору

перевищує обсяг оперативної пам'яті, доступної в системі, ємність дисків вводу-виводу стає першорядною.

Для вирішення цих проблем було запропоновано два методи

- вертикальне масштабування;
- горизонтальне масштабування.

Перший передбачає збільшення потужності апаратного чи програмного забезпечення шляхом збільшення кількості ресурсів. Однак обмеження в доступних технологіях і суворих апаратних конфігураціях хмарних провайдерів можуть обмежити потенційну потужність однієї машини для даного робочого навантаження, що призводить до практичного максимуму для вертикального масштабування.

Тому у цьому дослідженні зосередимось на горизонтальному масштабуванні, яке передбачає розподіл набору даних і навантаження системи між декількома серверами та додавання додаткових серверів за потреби для збільшення потужності.

Хоча загальна швидкість або потужність одного комп'ютера може бути невисокою, кожна машина може впоратися з частиною загального робочого навантаження, що потенційно може запропонувати кращу продуктивність, ніж один високошвидкісний сервер із високою пропускною здатністю. Крім того, розширення потужності розгортання вимагає лише додавання додаткових серверів у разі потреби, що може бути дешевшим, ніж придбання високопродуктивного обладнання для одного комп'ютера. Однак цей підхід передбачає збільшення інфраструктури та складність обслуговування для розгортання. MongoDB було обрано, тому що підтримує сегментування (або шардінг), що значно полегшує цей процес.

Підтримка MongoDB горизонтального масштабування з сегментування дозволяє розділити базу даних на окремі частини,

розташовані на різних машинах, щоб вирішити обмеження одного сервера. Процес сегментації передбачає вибір ключа з колекції та розділення даних за його значенням для розподілу документів між різними сегментами кластера.

Цей ключ, який називається сегментним ключем, може бути індексованим або індексованим складеним полем і використовується для розділення даних у колекції за допомогою різних діапазонів або блоків значень сегментного ключа. MongoDB розподіляє блоки та їхні документи між шардами в кластері, при цьому балансування сегментів і реплікація є автоматизованими процесами. Щоб оптимізувати продуктивність і можливості бази даних, важливо вибрати відповідний ключ фрагмента на основі схеми даних і операцій запиту та запису бази даних. Вбудована підтримка шардингу в MongoDB є привабливою функцією для архітекторів програмного забезпечення завдяки її гнучкій схемі та простоті розгортання. Тим не менш, розгортання процесу шардингу вимагає ретельного розгляду інфраструктури та складності обслуговування.

Отже, при розробці програмних продуктів критичним фактором є правильний вибір засобів реалізації програмного забезпечення. Вибраний стек технологій може значно вплинути на продуктивність системи, особливо з точки зору швидкості обробки запитів, надійності системи та стабільності під навантаженням.

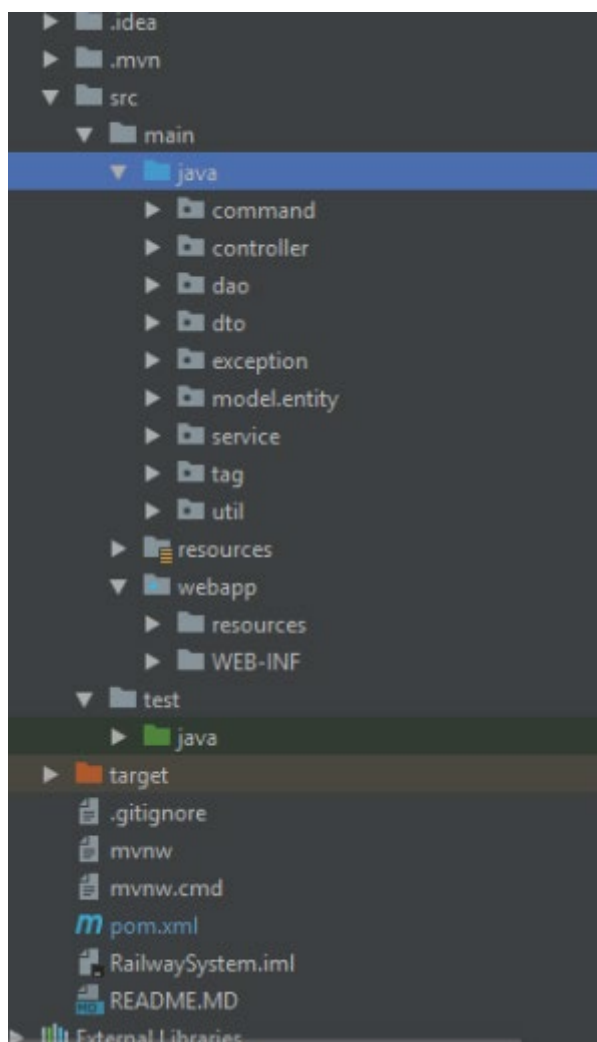
У цьому розділі був представлений аналіз технологій, придатних для розробки бази даних на архітектурному рівні, і окреслені основні принципи їх використання. Крім того, описано основні технології, які використовуються в реалізації програмного продукту, включно мова програмування Java, структура складання проекту Apache Maven і різні

методи тестування. Нарешті, також представлені основні технології фронтенд розробки.

## **2.2. Архітектура та конфігурація компонентів системи**

При розробці програмного продукту необхідним фактором є правильний вибір методів реалізації програмного забезпечення. Вибраний стек технологій може мати значний вплив на продуктивність системи, зокрема на швидкість обробки запитів, надійність системи та стабільність системи під навантаженням [10]. У цьому розділі розглядається процес конфігурації технологій, що були обрані для розробки архітектурного рівня бази даних, і викладені основні принципи їх використання.

Під час реалізації архітектури бізнес-логіки програмного продукту використовувалися різні шаблони GoF, включаючи шаблони Factory Method, Team, Single і Builder. Структура папок основних каталогів відповідає структурі Maven, як показано на рис 2.2.



**Рис. 2.2. Структура папок в програмному продукті**

Джерело: розроблено автором на основі джерела [18].

Основні каталоги включають src, target, .idea та такі файли, як .gitignore, External Libraries, RailwaySystem.iml і pom.xml. Папка src містить логіку проекту та розділена на дві основні папки:

- основну, яка містить java, ресурси та веб-додатки;
- test, яка містить усі методи з модульними тестами.

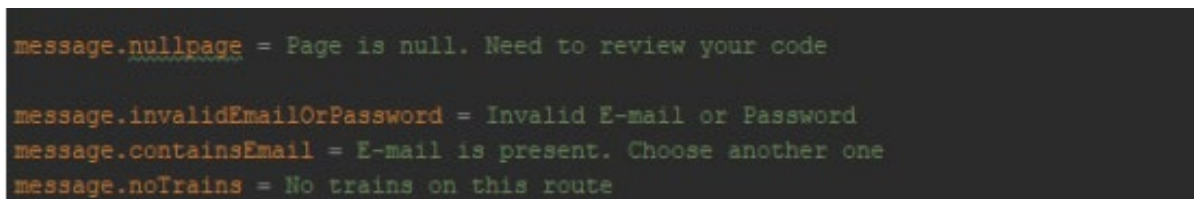
Цільова папка (target) зберігає скомпільовані файли проекту, які згодом розгортаються на сервері.

Папка .idea містить інформацію про проект і є модулем середовища розробки, створеного програмою, а зовнішні бібліотеки (External Libraries)

– це модуль, який зберігає набір підключених бібліотек, необхідних для реалізації проекту.

Файл `Railway.iml` створюється автоматично, коли проект імпортується в середовище розробки, а також зручний для використання під час повторного імпорту проекту в IntelliJ IDEA або на інший сервер. Файл `gitignore` відстежує вбудовані файли, а `pom.xml` є єдиним файлом конфігурації проекту в структурі Maven, що містить необхідну інформацію про збірку проекту та залежності.

Таким чином, основна папка підрозділяється на `java`, ресурси та `webapp`. Папка `java` містить логіку проекту, архітектуру рівня бізнес-логіки, конфігурацію підключення та взаємодії з базою даних, а також роботу з презентаційним рівнем, який охоплює інтерфейс і запити користувачів.

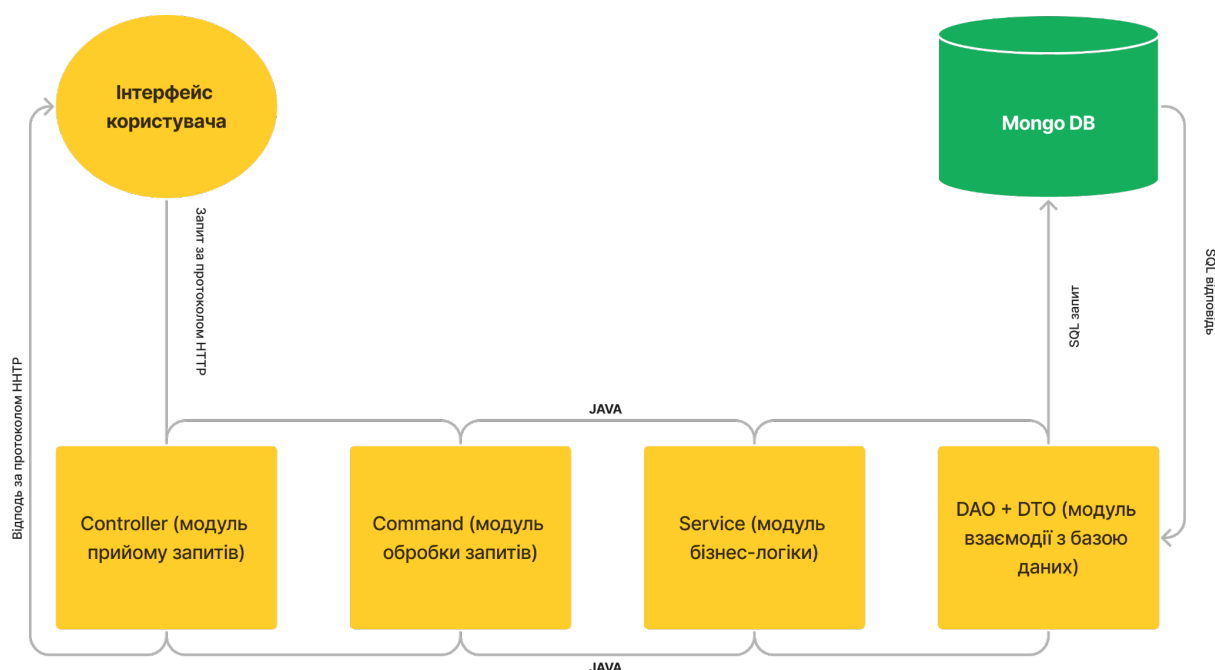


```
message.nullpage = Page is null. Need to review your code
message.invalidEmailOrPassword = Invalid E-mail or Password
message.containsEmail = E-mail is present. Choose another one
message.noTrains = No trains on this route
```

**Рис. 2.3. Папка ресурсів – повідомлення про помилку**

Джерело: розроблено автором на основі джерела [19].

Папка ресурсів містить конфігураційні файли для роботи з базами даних і для взаємодії з інтерфейсом, наприклад різні типи збережених повідомлень, як показано на рис. 2.2. Папка `webapp` містить реалізацію рівня презентації проекту.



**Рис. 2.4. Схема взаємодії модулів розробленої системи**

Джерело: розроблено автором на основі джерела [18].

Як показано на рис. 2.4 папка java містить кілька модулів:

- Controller, який відповідає за отримання та відповідь на запити (get/post) за допомогою протоколу HTTP;
- модуль Command, який обробляє запити;
- модуль Service, який є модуль бізнес-логіки;
- модулі util, dao і dto, які відповідають за взаємодію з базою даних.

Під час постановки завдання було вирішено розробити новий підхід до роботи з великими даними шляхом створення архітектурного рівня бази даних. Щоб досягти цього, була обрана нереляційна розподілена база даних MongoDB через її здатність зручно масштабувати систему, якщо це необхідно. Горизонтальне масштабування MongoDB було досягнуто за допомогою шардингу, за допомогою якого база даних була розділена на окремі частини, і кожна частина була призначена окремому серверу.

Щоб забезпечити відмовостійкість і зручне автоматичне відновлення даних, кожен сервер був запущеним процесом mongod, а для розробки

розподіленої бази даних у MongoDB було створено два сегменти та один сервер конфігурації, кожен із яких складається з трьох вузлів.

Розроблена система була налаштована на роботу з базою даних як централізованим ресурсом, а запити на дані оброблялися через центральний сервер (маршрутизатор), який розподіляв дані на різні набори реплік за допомогою ключа шардингу.

Основною перевагою цієї архітектурної моделі рівня бази даних є її горизонтальна масштабованість, яка дозволяє легко додавати додатковий набір реплік без пошкодження наявних даних у різних кластерах. Для масштабування системи необхідно просто створити новий кластер з необхідною кількістю вузлів і додати його до маршрутизатора. При цьому розроблена система може взаємодіяти як з нереляційною розподіленою базою даних, так і з реляційною базою даних MySQL.

Основною перевагою використання бази даних MongoDB в цьому проєкті є взаємодія з даними через програмний код, а не через SQL-запити, чого неможливо уникнути при роботі з розподіленою базою даних. Важливо також відзначити, що зміни в структурі бази даних повинні бути відображені тільки в програмному коді.

Як вже було згадано раніше, в цьому проєкті використовується Data Access Object (DAO) і Data Transfer Object (DTO), для роботи з даними в програмному продукті. DTO – це простий об'єкт, який зберігає дані та використовується для передачі даних між класами та модулями програми. У програмному забезпеченні, яке розглядається, DTO використовуються для зберігання таких об'єктів, як Ticket і TrainRoute, які містять інформацію про квиток і маршрут поїзда відповідно.

З іншого боку, DAO інкапсулює логіку для отримання, збереження та оновлення даних такими операціями, як створення, збереження,

читання, видалення та редагування. У програмному продукті DAO інкапсулює методи для Ticket.

Перевага використання DAO полягає в тому, що якщо виникає необхідність змінити механізм збереження, потрібно змінити лише рівень DAO, а не всі домени в логіці домену, які використовують рівень DAO. Подібним чином перевага DTO полягає в тому, що він надає велику гнучкість сервісному рівню, а згодом і дизайну всієї програми. Якщо є зміна вимоги, яка змушує перейти до іншого обсягу даних, це не впливає на рівень обслуговування чи навіть на домен. Потрібно лише перенести зміни до класу DTO, додавши нову властивість, залишивши загальний інтерфейс рівня обслуговування без змін.

Отже, у цьому розділі представлено огляд конфігурація системи та бази даних розробленої системи. Досліджено архітектуру системи, а також взаємодію її модулів між собою і з системою в цілому. Основна увага в цьому розділі зосереджена на розробці архітектури бази даних, зокрема на створенні нереляційної бази даних і розподілі системи через горизонтальне масштабування. Аналіз даних системи здійснюється як через інтерфейс, так і через термінал. Щоб полегшити розробку надійної та адаптованої структури, описано шаблони для роботи з базами даних та їхніми об'єктами, щоб гарантувати, що будь-які потенційні зміни в структурі даних не призведуть до неочікуваних проблем. Крім того, у цьому розділі розглядається сумісність системи з різними типами баз даних, включаючи реляційні та нереляційні бази даних, і аналізуються переваги та недоліки кожної з них.

## РОЗДІЛ 3. ТЕСТУВАННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

### 3.1. Тестування системи бронювання

Поточна інформаційна система бронювання залізничних квитків використовувала JUnit для цілей модульного тестування. JUnit – це платформа тестування, яка використовує анотації для визначення методів тестування та їх відповідного налаштування. Таблиця 2.1, подана нижче, пропонує вичерпний підсумок найбільш значущих анотацій у JUnit для версій 4.x і 5.x.

Таблиця 3.1

Анотації JUnit тестування

JUnit 4	Опис
@Test	Позначає метод як метод тестування. Він не має жодних атрибутів, і тестові розширення в JUnit Jupiter працюють на основі своїх спеціальних анотацій.
@Before	Вказує на те, що анотований метод слід виконувати перед кожним тестовим методом.
@After	Такі методи успадковуються, якщо вони не перевизначені або замінені.
@BeforeClass	Вказує на те, що анотований метод має бути виконано перед усіма тестовими методами в поточному класі.
@AfterClass	Вказує на те, що анотований метод має бути виконано після всіх тестових методів у поточному класі.
@Nested	Означає, що анотований клас є нестатичним вкладеним тестовим

	класом.
@Disabled	Вимикає тестовий клас або тестовий метод.

Джерело: розроблено автором на основі джерела [17].

Враховуючи те, що проект передбачав керування великою базою даних, процес тестування вимагав використання PowerMock як генератора макетів об'єктів.

PowerMock – це потужний фреймворк для тестування Java, який розширює можливості існуючих фреймворків для тестування, таких як JUnit і TestNG. Однією з ключових особливостей PowerMock є його здатність генерувати макетні об'єкти для складних залежностей, які важко перевірити ізольовано.

Фіктивні об'єкти – це фіктивні об'єкти, які контролювано імітують поведінку реальних об'єктів. Вони використовуються під час тестування для заміни залежностей, які важко створити чи маніпулювати безпосередньо. PowerMock дозволяє розробникам створювати імітаційні об'єкти для класів, які зазвичай важко імітувати, наприклад, статичні та фінальні класи.

Отже, PowerMock використовує маніпуляції байт-кодом для створення макетів об'єктів. Маніпуляції байт-кодом включають модифікацію байт-коду скомпільованих класів під час виконання, щоб змінити їх поведінку. Це дозволяє PowerMock замінювати виклики методів власною поведінкою та перехоплювати виклики конструкторів для створення макетів об'єктів.

PowerMock також надає API для створення макетів об'єктів вручну. Цей API дозволяє розробникам визначати спеціальну поведінку для методів і конструкторів, а також налаштовувати очікування для викликів методів і повернутих значень. PowerMock також може імітувати статичні

методи, фінальні методи та приватні методи, що робить його цінним інструментом для тестування складних систем.

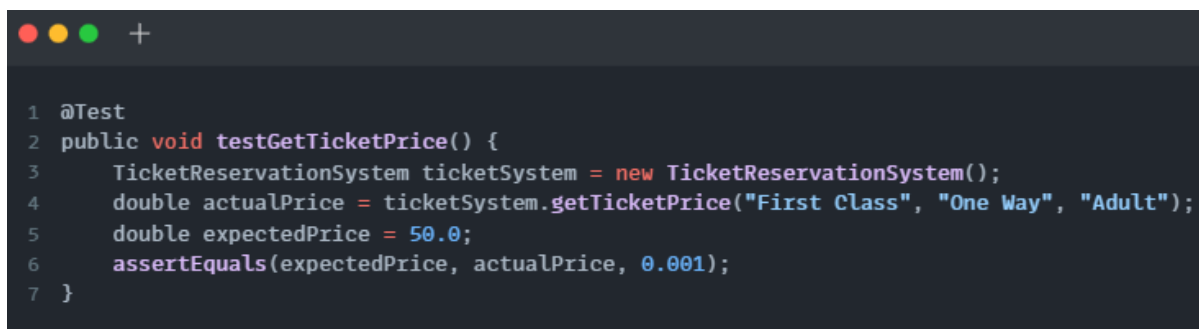
Для тестування використовується техніка модульного тестування, яке фокусується на тестуванні окремих одиниць або компонентів розробленого програмного забезпечення ізольовано від решти системи. У цьому підході кожен блок або метод тестується окремо, щоб переконатися, що він працює належним чином і відповідає вимогам і специфікаціям системи.

Модульний тест для конкретного методу зазвичай включає такі кроки:

1. Налаштувати тестове середовище: створити екземпляр класу, який містить метод, який необхідно перевірити, і ініціалізувати будь-які об'єкти або змінні, необхідні для тесту.
2. Визначити вхідні дані, які використовуватимуться для перевірки методу. Це може включати передачу аргументів у метод, налаштування тестових даних або макетів об'єктів або ініціалізацію тестових фікстур.
3. Викликати метод, використовуючи вхідні дані, визначені на попередньому кроці.
4. Перевірте вихід, щоб переконатися, що метод дав очікуваний результат. Це може включати перевірку поверненого значення, перевірку стану об'єктів або змінних або перевірку поведінки методу.

Давайте розглянемо приклад модульного тесту для конкретного методу, використовуючи як приклад метод `getTicketPrice()` класу `TicketReservationSystem` з нашого проекту. Цей метод приймає три аргументи – `ticketType`, `journeyType` і `ipassengerType` – і повинен повертати ціну квитка.

На рис. 3.1. зображено приклад модульного тесту для методу `getTicketPrice()`.



```
1 @Test
2 public void testGetTicketPrice() {
3     TicketReservationSystem ticketSystem = new TicketReservationSystem();
4     double actualPrice = ticketSystem.getTicketPrice("First Class", "One Way", "Adult");
5     double expectedPrice = 50.0;
6     assertEquals(expectedPrice, actualPrice, 0.001);
7 }
```

**Рис. 3.1. Модульний тест getTicketPrice**

Джерело: розроблено автором на основі джерела [17].

У цьому тесті спочатку створюється екземпляр класу `TicketReservationSystem`. Далі визначаються вхідні дані для тесту, які включають тип квитка, тип подорожі, тип пасажирів та очікувану вартість квитка.

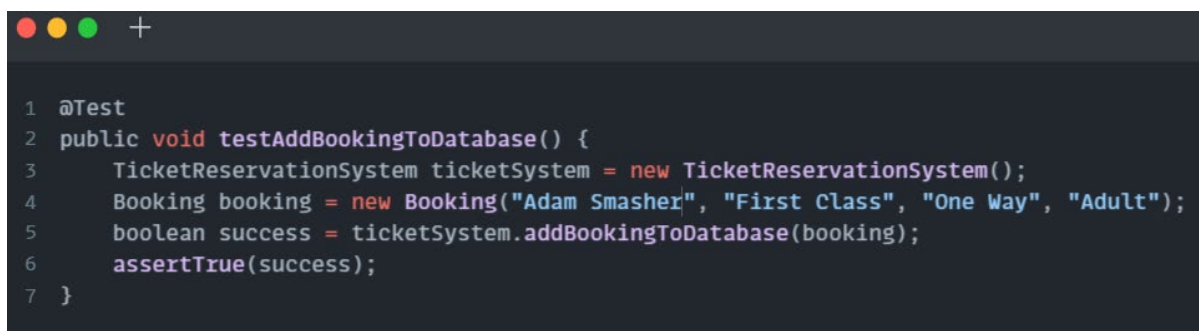
Потім із вхідними даними викликається метод `getTicketPrice()`, а результат зберігається в змінній `actualPrice`.

Метод `assertEquals()` використовується для порівняння змінних `expectedPrice` і `actualPrice` і перевірки того, що метод дав очікуваний результат. Третій аргумент для `assertEquals()` – це дельта або максимальна різниця між очікуваним і фактичним значеннями, яка вважається прийнятною. У цьому випадку дельта `0,001` використовується для врахування будь-яких помилок з плаваючою комою.

Цей тест перевіряє, чи метод `getTicketPrice()` правильно обчислює ціну квитка на основі вхідних даних і повертає очікуване значення. Таким чином, тестуючи окремі методи ізольовано за допомогою модульних тестів, ми можемо виявити помилки на ранніх стадіях розробки та переконатися, що наше програмне забезпечення відповідає вимогам і специфікаціям технічного завдання.

Ще один тип тестів, які використовуються при тестуванні цього проекту – це тести інтеграції, які призначені для перевірки взаємодії між

різними компонентами системи. У цьому випадку ми зосередимося на інтеграційному тесті для роботи з базою даних (рис. 3.2).

A screenshot of a code editor window with a dark background. The window has three colored window control buttons (red, yellow, green) and a plus sign in the top-left corner. The code is written in Java and is a JUnit test method. It is numbered 1 through 7 on the left side. The code creates a new TicketReservationSystem, adds a booking, and asserts that the operation was successful.

```
1 @Test
2 public void testAddBookingToDatabase() {
3     TicketReservationSystem ticketSystem = new TicketReservationSystem();
4     Booking booking = new Booking("Adam Smasher", "First Class", "One Way", "Adult");
5     boolean success = ticketSystem.addBookingToDatabase(booking);
6     assertTrue(success);
7 }
```

**Рис. 3.2. Інтеграційний тест для роботи з базою даних**

Джерело: розроблено автором на основі джерела [17].

Щоб виконати інтеграційний тест для операції з базою даних, нам потрібна тестова база даних, відокремлена від робочої бази даних. Ця тестова база даних повинна мати ту саму схему, що й робоча база даних, але не повинна містити конфіденційних даних.

Далі нам потрібно створити тестовий сценарій, який виконуватиме операцію з базою даних, яку ми хочемо протестувати. Цей сценарій має включати всі необхідні кроки налаштування та демонтажу, наприклад створення та видалення тестових даних.

Після цього виконуємо тестовий сценарій у тестовій базі даних і перевіряємо, чи отримано очікувані результати. Наприклад, якщо ми тестуємо операцію з базою даних, яка отримує дані з бази даних, ми повинні перевірити, чи отримані дані відповідають нашим очікуванням.

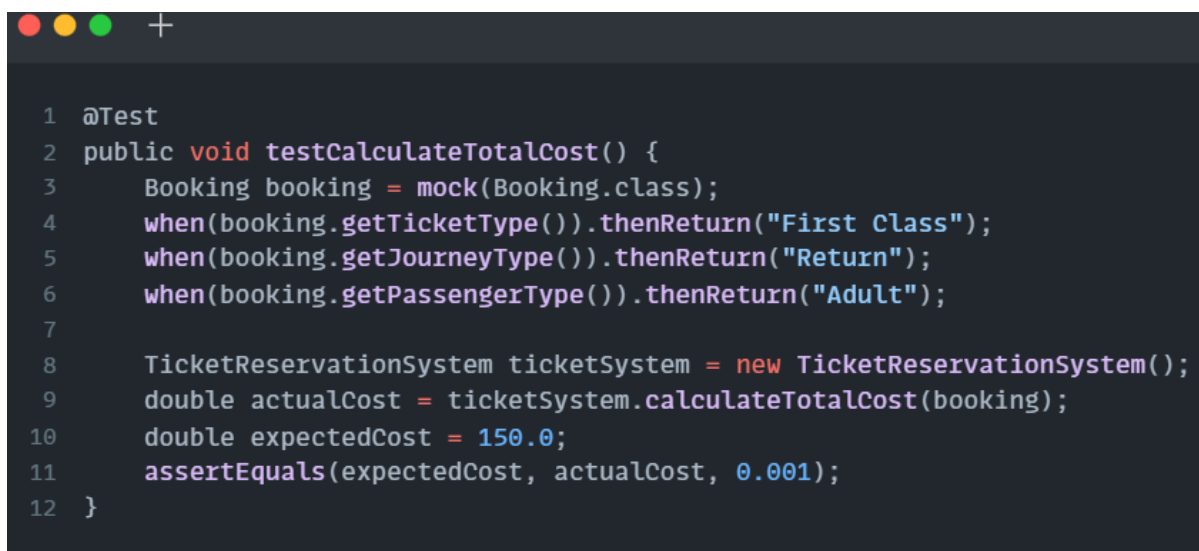
Під час інтеграційного тесту ми також повинні переконатися, що операція бази даних правильно взаємодіє з рештою програми. Це може включати перевірку того, що програма правильно обробляє помилки, які можуть виникнути під час роботи з базою даних, або що програма правильно використовує дані, отримані з бази даних.

У тесті зображеному на рис. 3.2 ми перевіряємо метод `addBookingToDatabase()` класу `TicketReservationSystem`, який додає

бронювання до бази даних. Ми створюємо екземпляр класу та створюємо новий об'єкт бронювання. Потім викликаємо метод `addBookingToDatabase()` з об'єктом `booking` і перевіряємо, чи операція була успішною.

Загалом інтеграційні тести для операцій з базою даних є важливою частиною забезпечення надійності та точності системи, яка покладається на базу даних. Тестуючи взаємодію між додатком і базою даних, ми можемо завчасно виявити помилки та переконатися, що наша система функціонує належним чином.

На завершення, ми використовуємо імітаційне тестування – це техніка, яка дозволяє нам ізолювати та тестувати окремі компоненти системи, не залежачи від інших компонентів чи зовнішніх служб. Це особливо корисно під час тестування складних операцій, які включають кілька компонентів і залежностей.

A screenshot of a code editor window with a dark background. The window has standard macOS window controls (red, yellow, green buttons and a plus sign) in the top-left corner. The code is written in Java and is a JUnit test. It uses Mockito to mock a Booking object and sets up expectations for its methods. Then, it creates a TicketReservationSystem object, calls its calculateTotalCost method with the mock booking, and asserts that the result is 150.0 with a precision of 0.001.

```
1 @Test
2 public void testCalculateTotalCost() {
3     Booking booking = mock(Booking.class);
4     when(booking.getTicketType()).thenReturn("First class");
5     when(booking.getJourneyType()).thenReturn("Return");
6     when(booking.getPassengerType()).thenReturn("Adult");
7
8     TicketReservationSystem ticketSystem = new TicketReservationSystem();
9     double actualCost = ticketSystem.calculateTotalCost(booking);
10    double expectedCost = 150.0;
11    assertEquals(expectedCost, actualCost, 0.001);
12 }
```

**Рис. 3.2. Інтеграційний тест для роботи з базою даних**

Джерело: розроблено автором на основі джерела [17].

У цьому тесті перевіряється метод `CalculateTotalCost()` класу `TicketReservationSystem`, який обчислює загальну вартість бронювання. Для цього було створено імітаційний об'єкт `Booking` за допомогою бібліотеки `PowerMock` і встановлено його атрибути за допомогою методу

when()). Потім ми створюємо екземпляр класу TicketReservationSystem і викликаємо метод calculateTotalCost() з об'єктом імітаційного бронювання. Як результат, порівнюємо фактичний результат з очікуваним і перевіряємо, чи вони збігаються в межах певної похибки.

Таким чином, щоб виконати пробний тест для складної операції, нам спочатку потрібно визначити компоненти, від яких залежить операція. Наприклад, якщо тестуємо систему обробки платежів, ми можемо залежати від системи автентифікації користувача, системи обробки кредитних карток і бази даних, яка зберігає інформацію про транзакції.

Далі нам потрібно створити макет об'єктів для кожної з цих залежностей. Макетні об'єкти – це полегшені імітовані версії реальних компонентів, які дозволяють нам контролювати їх поведінку та реакції під час тестування. Для різних мов програмування доступно кілька фреймворків імітаційних об'єктів, наприклад Mockito для Java або Mockery для PHP.

Налаштувавши макетні об'єкти, ми можемо створити тестовий сценарій, який імітує складну операцію та використовує макетні об'єкти для керування поведінкою залежних компонентів. Наприклад, ми можемо використовувати імітаційну систему аутентифікації для імітації успішних і невдалих спроб входу, і ми можемо використовувати імітаційну систему обробки кредитних карток для імітації успішних і невдалих платіжних операцій.

Під час пробного тестування перевіряємо чи складна операція поводить себе правильно за різних умов і з різними відповідями від залежних компонентів. Ми також можемо тестувати крайні випадки та сценарії помилок, які може бути важко відтворити в живому середовищі.

Імітаційне тестування дозволяє нам тестувати складні операції в контрольованому середовищі, не покладаючись на зовнішні служби чи

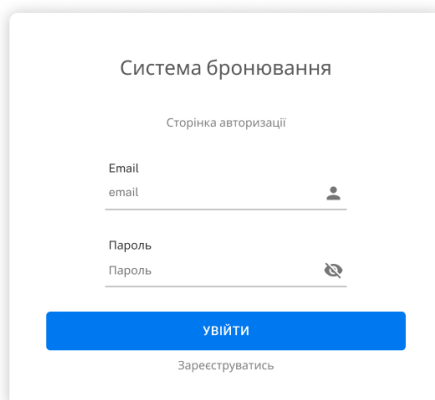
складні налаштування. Ізолюючи різні компоненти системи, ми можемо швидше виявляти та виправляти проблеми та з меншим ризиком спричинення небажаних наслідків у живому середовищі.

Загалом імітаційне тестування є цінною технікою для тестування складних операцій, які включають кілька компонентів і залежностей. Використовуючи макетні об'єкти для імітації поведінки залежних компонентів, ми можемо тестувати нашу систему більш ретельно та з більшим контролем.

### **3.2. Особливості взаємодії користувача з розробленою системою**

У цьому розділі детально описано основні вимоги до роботи з інформаційною системою, включаючи апаратні та програмні передумови, а також інтерфейс користувача.

Система передбачає підключення до Інтернету як основну вимогу для доступу з боку користувача, а сторінка авторизації, як показано на рис. 3.3, надає доступ до різних модулів на основі рівня доступу створення акаунту.



Система бронювання

Сторінка авторизації

Email  
email

Пароль  
Пароль

УВІЙТИ

Зареєструватись

**Рис. 3.3. Сторінка авторизації у систему**

Джерело: розроблено автором на основі джерела [22].

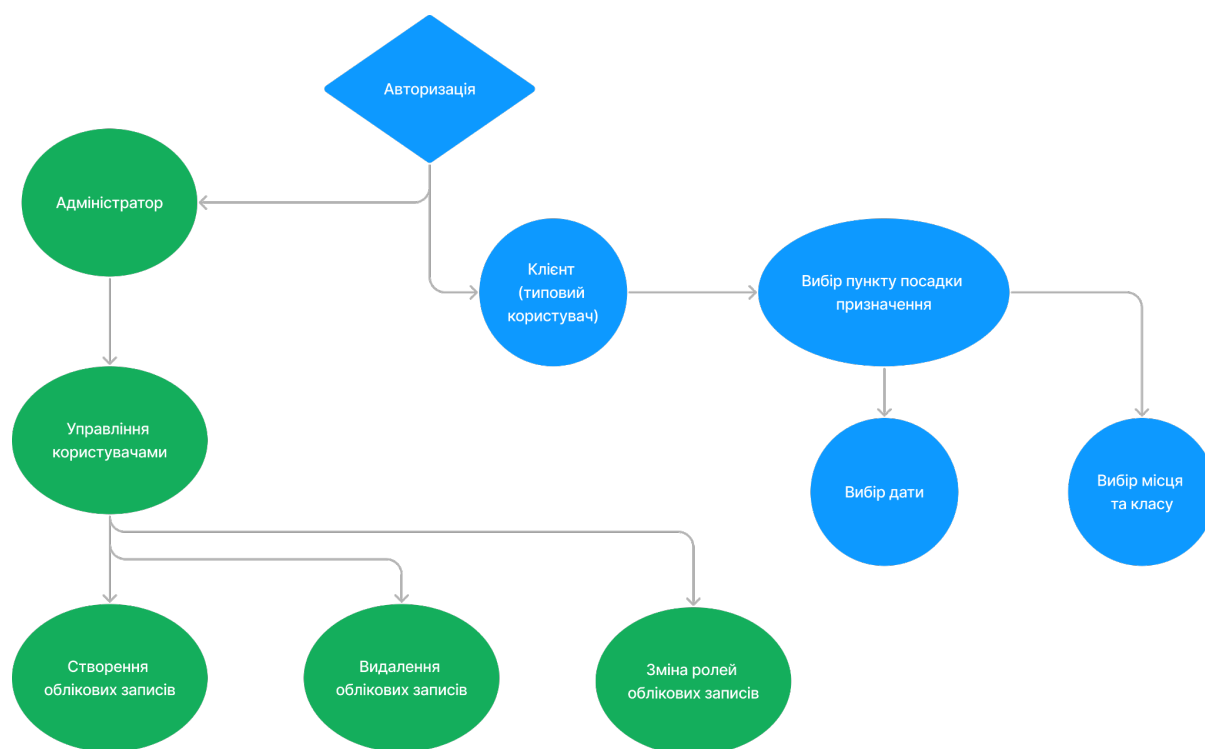
Користувачам також надається можливість вибору мови відображення сторінки: українська та англійська є двома основними мовами програми.

Крім того, інформаційна система має мінімальні вимоги до обладнання, включаючи рекомендований обсяг оперативної пам'яті не менше 1 ГБ і швидкість підключення до Інтернету не менше 128 Кбіт/с.

Розроблена інформаційна система має три ролі, важливі для її оптимального використання:

- адміністратор;
- користувач.

User story кожної з цих ролей зображена на рис. 3.4. На схемі видно, що кожен користувач системи має унікальні функції та завдання для роботи з системою.

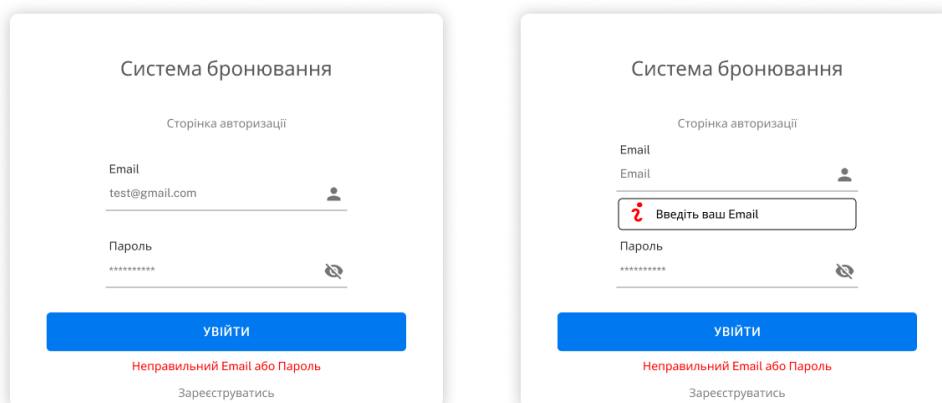


**Рис. 3.4. Ролі та функції в межах системи**

Джерело: розроблено автором на основі джерела

Адміністратор відповідає за найважливіші функції управління та передачі даних. З іншого боку, користувач системи має обмежену функціональність залежно від рівня доступу до модулів.

Щоб забронювати квитки, вхід у систему є обов'язковим, і лише авторизовані користувачі мають доступ до системи, після чого вони перенаправляються на наступну сторінку з різними функціями, залежно від ролі користувача в системі. У разі невдалої спроби входу користувач бачить сповіщення, як показано на рис. 3.5 із зазначенням недійсних даних та обов'язкових полів, необхідних для входу.



**Рис. 3.5. Проблеми з авторизації у систему**

Джерело: розроблено автором на основі джерела [22].

Якщо користувач не зареєстрований у системі, він може почати реєстраційний процес, натиснувши посилання «Реєстрація/Зареєструватися» на головній сторінці, як показано на рис. 3.5.

Після реєстрації або входу в систему користувач перенаправляється на веб-сторінку та бачить у заголовку привітання, яке підтверджує, що він увійшов до свого облікового запису під своїм іменем. Потім користувач перенаправляється на сторінку з вибором усіх необхідних пошукових даних, які необхідно ввести для пошуку конкретного рейсу, як показано на рис. 3.6.

**Система Бронювання** Моє бронювання [Вхід / Увійти](#)

**Пошук**

Київ  Львів

< Вт 15 **Ср 16** Чт 17 Пт 18 Сб 19 Нд 20 >

**Доступні потяги** 5 рейсів доступно

**12430: Дніпро - Львів**

Час відправлення:  Перегнути розклад

Трав 16 11:25 Дніпро 21 ч Трав 7 8:25 Львів

<b>КОМФОРТ</b> 046	<b>СТАНДАРТ</b> 006	<b>ЕКОНОМ</b> 036	<b>ЛЮКС</b> Недоступний
Ціна 920 ₴	Ціна 620 ₴	Ціна 216 ₴	

**12230: Київ - Львів**

Час відправлення:  Перегнути розклад

Трав 16 11:25 Київ 8 годин Трав 16 7:25 am Львів

<b>КОМФОРТ</b> 046	<b>СТАНДАРТ</b> 006	<b>ЕКОНОМ</b> 036
Ціна 920 ₴	Ціна 620 ₴	Ціна 216 ₴

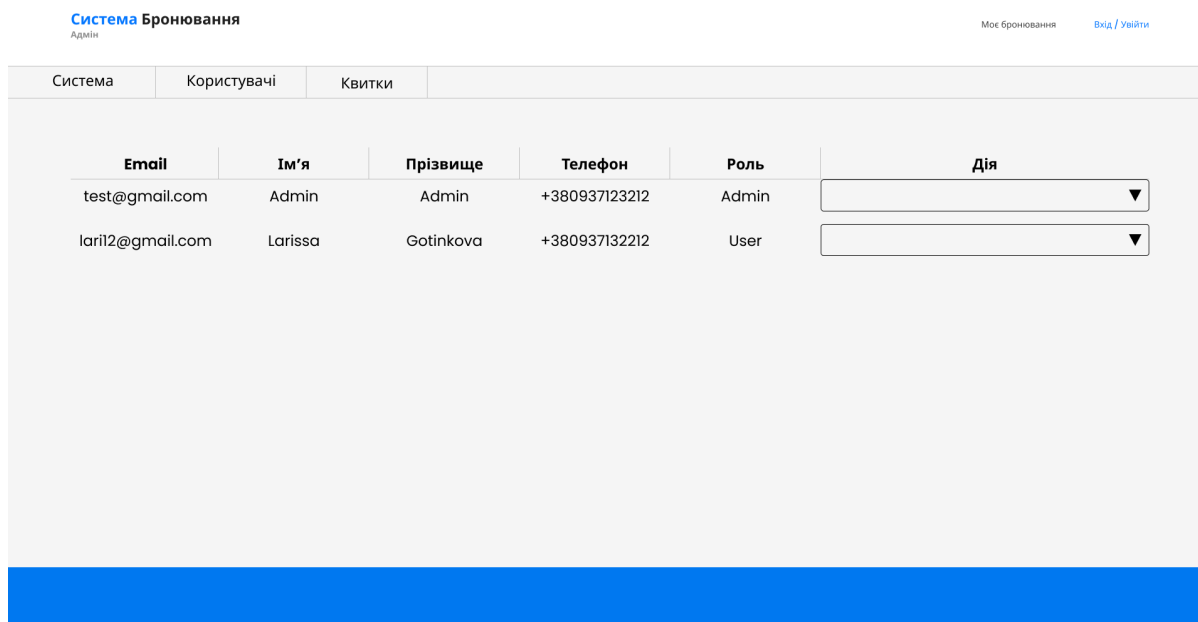
**Рис. 3.6. Пошук та вибір маршруту у системі**

Джерело: розроблено автором на основі джерела [22].

Користувач вибирає міста та дату, необхідні для пошуку безкоштовного залізничного квитка для бронювання, і, якщо можливо, вказує зручний час. Якщо користувач не вказує час, система показує всі доступні для бронювання квитки на цей конкретний день.

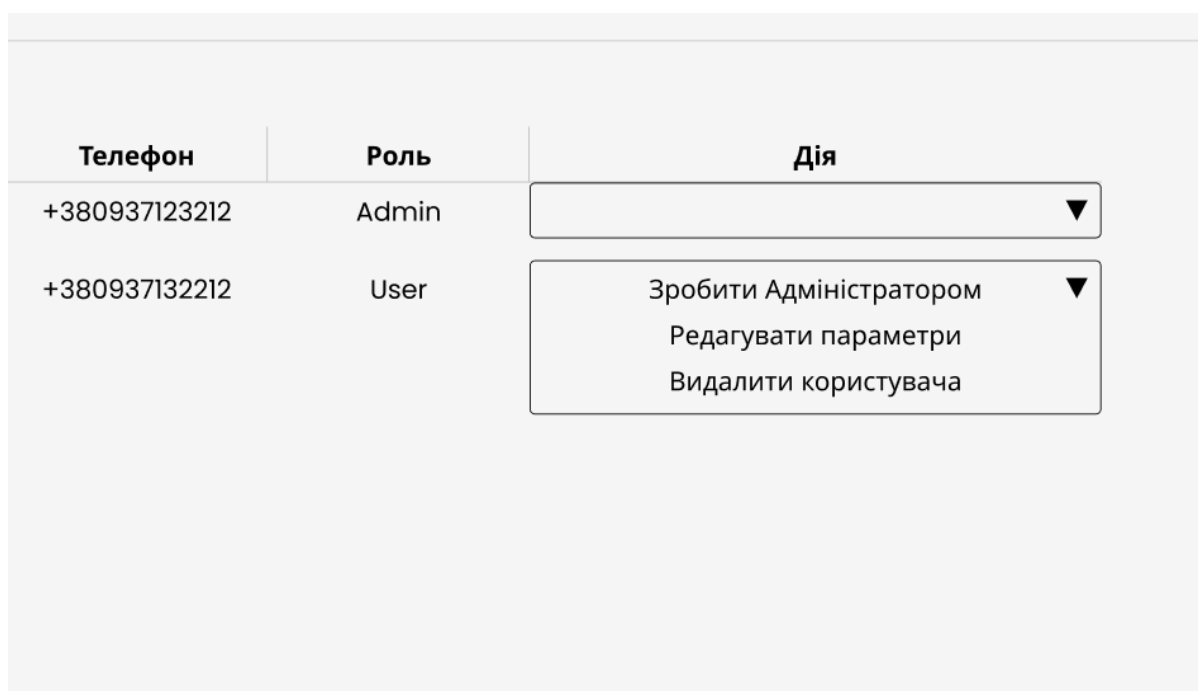
Знайшовши потрібний залізничний квиток, користувач може вибрати місце відповідно до класу пасажирського місця, а також вибрати будь-яке вільне місце як показано на рис. 3.6. Вартість квитка залежить від класу пасажирського місця, обраного користувачем. Користувач також може вибрати необхідну кількість залізничних квитків.

Щодо адміністратора, то після авторизації в системі адміністратор негайно перенаправляється на сторінку, де відображаються активні та неактивні користувачі системи, як показано на рис. 3.7.



**Рис. 3.7. Основні сторінка панелі управління адміністратора у системі**  
Джерело: розроблено автором на основі джерела [22].

Адміністратор може переглядати всіх користувачів системи та шукати користувачів у системі за адресою електронної пошти, іменем та прізвищем користувача. Адміністратор також має можливість надавати різні рівні доступу до програми кожному користувачеві, як показано на рис. 3.8.



### **Рис. 3.8. Управління ролями з боку адміністратора**

Джерело: розроблено автором на основі джерела [22].

Таким чином, адміністратор може створити іншого адміністратора, створити нового користувача та видалення вже існуючого з системи.

При переході на сторінку з інформацією про заброньовані квитки системний адміністратор бачить календар для вибору дати відправлення, за яким здійснюється пошук заброньованих квитків.

Після вибору конкретної дати на сторінці з'являється список залізничних квитків. Однією з основних функцій адміністратора є можливість видаляти квитки із системи (як усі квитки на певний день, так і окремий).

Отже, у цьому розділі було описано функціонування клієнт-серверної частини системи та наведено інструкції з експлуатації системи з боку клієнта з різними рівнями доступу.

## ВИСНОВКИ

Під час виконання цього дипломного проекту розглядалися дослідження та розробка архітектурного рівня бази даних з метою створення системи бронювання залізничних квитків. Метою цієї роботи була розробка масштабованої клієнт-серверної системи обробки даних для бронювання залізничних квитків з використанням нереляційних розподілених баз даних.

У результаті створено інформаційну клієнт-серверну систему обробки даних на основі бронювання залізничних квитків. Проведено аналіз архітектурних рішень рівня баз даних для обробки великих даних, а також оцінку процесу автоматизації бронювання квитків.

Було визначено проблеми, пов'язані з обробкою великих даних, і було прийнято рішення розробити клієнт-серверну програму з розподіленою нереляційною базою даних для підвищення ефективності обробки даних.

Проведено аналіз стеку технологій для розробки інформаційно масштабованої клієнт-серверної системи обробки даних про бронювання залізничних квитків. Було також деталізовано структуру програмної реалізації системи та продемонстровано роботу системи на різних рівнях доступу до кожного модуля з наданими інструкціями щодо використання веб-інтерфейсу.

На основі аналізу програмної реалізації архітектурного рівня бази даних було зроблено висновок, що нереляційна розподілена база даних недостатньо практична для використання в цій інформаційній системі.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Автоматизовані системи бронювання в туризмі. Все о туризме: туристическая библиотека. URL: [https://tourlib.net/statti\\_ukr/melnychenko7.htm](https://tourlib.net/statti_ukr/melnychenko7.htm) (дата звернення: 05.05.2023).
2. Альохін А., Круглик В. Проектування та розробка мікросервісної архітектури системи бронювання. Young scientist. 2020. № 12 (88). С. 95–100. URL: <https://doi.org/10.32839/2304-5809/2020-12-88-20> (дата звернення: 05.05.2023).
3. Історія виникнення комп'ютерних систем бронювання. Новости туризма. URL: <https://tourism-book.com/books/book-32/chapter-1363/> (дата звернення: 05.05.2023).
4. Неправська Н. Залізничні квитки продаємо по-французьки. Український туризм. 2016. № 4, лип. - серп. С. 64–65.
5. Основи ІМГ: Тема 6. Автоматизовані системи бронювання і резервування в туризмі. Система електронного забезпечення навчання ЗНУ. URL: <https://moodle.znu.edu.ua/mod/page/view.php?id=267462> (дата звернення: 05.05.2023).
6. Тузенко О. О., Тілінін С. В., Кличков В. О. Розробка універсальної системи бронювання для автоматизації ведення малого і середнього бізнесу. Наука та виробництво. 2020. № 22. URL: <https://doi.org/10.31498/2522-9990222020211359> (дата звернення: 05.05.2023).
7. Удосконалення залізничної пасажирської системи в умовах використання інтернету / О. М. Ходаківський та ін. Collection of scientific works of the ukrainian state university of railway transport.

2014. № 150. URL: <https://doi.org/10.18664/1994-7852.150.2014.67447> (дата звернення: 05.05.2023).
8. Airline reservations systems: a brief history – airlinegeeks.com. AirlineGeeks.com. URL: <https://airlinegeeks.com/2016/08/16/airline-reservations-systems-a-brief-history/> (date of access: 05.05.2023).
  9. Apache tomcat® - welcome!. URL: <https://tomcat.apache.org> (date of access: 05.05.2023).
  10. Based on netease train ticket booking system design and implementation. IOPscience. URL: <https://iopscience.iop.org/article/10.1088/1742-6596/1486/2/022006/meta> (date of access: 05.05.2023).
  11. Design and development of a ticket booking system using smart bot. IEEE Xplore. URL: <https://ieeexplore.ieee.org/abstract/document/9964700/> (date of access: 05.05.2023).
  12. Exploiting an event-based infrastructure to develop complex distributed systems. IEEE Xplore. URL: <https://ieeexplore.ieee.org/abstract/document/671135> (date of access: 05.05.2023).
  13. HTML: hypertext markup language | MDN. MDN Web Docs. URL: <https://developer.mozilla.org/en-US/docs/Web/HTML> (date of access: 05.05.2023).
  14. IntelliJ IDEA – the leading java and kotlin IDE. JetBrains. URL: <https://www.jetbrains.com/idea/> (date of access: 05.05.2023).
  15. Introduction to PowerMockito | Baeldung. Baeldung. URL: <https://www.baeldung.com/intro-to-powermock> (date of access: 05.05.2023).
  16. Java. URL: <https://www.java.com/en/> (date of access: 05.05.2023).

- 17.JUnit – about. JUnit. URL: <https://junit.org/junit4/> (date of access: 05.05.2023).
- 18.Kurniawan B. Java web development with servlets, jsp, and ejb. 2nd ed. Sams, 2004. 896 p.
- 19.Learn servlet tutorial - javatpoint. [www.javatpoint.com](http://www.javatpoint.com). URL: <https://www.javatpoint.com/servlet-tutorial> (date of access: 05.05.2023).
- 20.MongoDB: the developer data platform. MongoDB. URL: <https://www.mongodb.com> (date of access: 05.05.2023).
- 21.My SQL. URL: <https://www.mysql.com> (date of access: 05.05.2023).
- 22.The most popular HTML, CSS, and JS library in the world. Bootstrap. URL: <https://getbootstrap.com> (date of access: 05.05.2023).
- 23.What is CSS? - Learn web development | MDN. MDN Web Docs. URL: [https://developer.mozilla.org/en-US/docs/Learn/CSS/First\\_steps/What\\_is\\_CSS](https://developer.mozilla.org/en-US/docs/Learn/CSS/First_steps/What_is_CSS) (date of access: 05.05.2023).

## ДОДАТКИ

### Додаток А. Набір методів для обміну даними з MongoDB

```
import org.bson.Document;
import org.bson.conversions.Bson;
import org.bson.conversions.BsonToJavaConverter;
import org.bson.conversions.JavaType;
import org.bson.conversions.JavaTypeWriters;
import org.bson.conversions.BsonToJavaConverter;
import org.bson.conversions.BsonValue;
import org.bson.types.ObjectId;
import org.bson.types.ObjectIdWritable;
import org.mongodb.MongoCollection;
import org.mongodb.MongoCollection;
import org.mongodb.MongoDatabase;
import org.mongodb.MongoDocument;
import org.mongodb.MongoModel;
import org.mongodb.MongoQuery;
import org.mongodb.MongoTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.data.mongodb.repository.MongoRepositoryImpl;
import
org.springframework.data.mongodb.repository.ModifyingQueryOperation;
import
org.springframework.data.mongodb.repository.QueryCreationInformation;
import org.springframework.data.mongodb.repository.QueryOperation;
```

```

import
org.springframework.data.mongodb.repository.QueryCreationInformation;
import org.springframework.data.mongodb.repository.QueryOperation;
import
org.springframework.data.mongodb.repository.ModifyingQueryOperation;

import java.util.List;

public abstract class UserRepository extends MongoRepository<User, String> {

    @Autowired
    private MongoCollection<Document> collection;

    private MongoDB database;

    @Autowired
    private MongoTemplate mongoTemplate;

    private final String COLLECTION_NAME = "users";
    private final Query<User> findAllQuery;
    private final Query<User> findByIdQuery;
    private final Query<User> findByEmailQuery;

    @Override
    public List<User> findAll() {
        return collection.find(findAllQuery).into(new ArrayList<>());
    }
}

```

```

@Override
public User findById(String id) {
    MongoDBDocument document = collection.find(Filters.eq("_id", new
ObjectId(id)));
    return document != null ? getUser(document) : null;
}

```

```

@Override
public User findByEmail(String email) {
    MongoDBDocument document = collection.find(Filters.eq("email", email));
    return document != null ? getUser(document) : null;
}

```

```

@Override
public User create(User user) {
    return collection.insertOne(user);
}

```

```

@Override
public User update(User user) {
    UpdateResult updateResult = collection.updateOne(Filters.eq("_id", new
ObjectId(user.getId())), Updates.updatedFields("$set", new
UpdateOperations<User>("set", "set")));
    return getUser(user.getId());
}

```

```

@Override
public void delete(Long id) {

```

```
        collection.deleteOne(new ObjectId(id));  
    }  
  
    // ... other methods can be added here  
}
```

**Додаток Б. Набір методів для обміну даними з JSP-сторінками**

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.DispatcherServlet;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.StringWriter;
```

```
import javax.servlet.ServletContext;
import org.springframework.web.servlet.WebServlet;
import org.springframework.web.servlet.ServletContextEvent;
import org.springframework.web.servlet.ServletContextListener;
```

@Configuration

```
public class ServletConfiguration {
```

    @Autowired

```
    private ServletContext servletContext;
```

    @Bean

```
    public ServletContextListener servletContextListener() {
```

```
        return new ServletContextListener() {
```

            @Override

```
            public void contextInitialized(ServletContextEvent event) {
```

```

        ServletContextEvent event = (ServletContextEvent) event;
        ServletContext servletContext = event.
            .getServletContext()
            .getServletContext(ServletContext.SESSION_TYPE);
        servletContext.addServlet(
            (ServletContextEvent.getServletContext(),
                new DispatcherServlet());
    }
}

```

`@Bean`

```

public DispatcherServlet dispatcherServlet() {
    return new DispatcherServlet() {
        @Override
        protected void init() throws ServletException {
            super.init();
            Locale.setDefault(Locale.US);
        }
    };
}

```

`@Override`

```

protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
    processRequest(request, response);
}

```

```

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
    processRequest(request, response);
}

private void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
    String page = command.
        .execute(request, response);
    response.setCharacterEncoding(CHARACTER_ENCODING);
    response.setContentType(CONTENT_TYPE);
    RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher(page);
    dispatcher.forward(request, response);
}
}

```

## Додаток В. Структура MainServlet

```

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.annotation.WebServlet;
import java.io.IOException;
import java.util.Date;

@WebServlet("/")
public class MainServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        // Check user authentication
        boolean isAuthenticated = checkAuthentication(request);

        if (isAuthenticated) {
            String role = getUserRole(request);

            if ("admin".equals(role)) {
                // Redirect to admin page
                response.sendRedirect("admin.jsp");
            } else {
                // Redirect to user's home page
                response.sendRedirect("home.jsp");
            }
        }
    }
}

```

```
    }  
    } else {  
        // Redirect to login page  
        response.sendRedirect("login.jsp");  
    }  
}  
  
private boolean checkAuthentication(HttpServletRequest request) {  
    // Perform authentication logic here  
    // Return true if user is authenticated, false otherwise  
    return true;  
}  
  
private String getUserRole(HttpServletRequest request) {  
    // Retrieve user role from session or database  
    // Return the user's role (e.g., "admin" or "user")  
    return "user";  
}  
}
```

## Додаток Г. Структура Login та Register Servlet

### LoginServlet.java

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.annotation.WebServlet;
import java.io.IOException;

@WebServlet("/login")
public class LoginServlet extends HttpServlet {
    private static final String USERNAME = "admin";
    private static final String PASSWORD = "password";

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        if (USERNAME.equals(username) && PASSWORD.equals(password)) {
            // Successful login
            request.getSession().setAttribute("username", username);
            response.sendRedirect("home.jsp");
        } else {
            // Failed login

```

```

        response.sendRedirect("login.jsp?error=true");
    }
}

```

### **RegisterServlet.java**

```

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.annotation.WebServlet;
import java.io.IOException;

@WebServlet("/register")
public class RegisterServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        // Perform registration logic here

        // Redirect to login page after successful registration
        response.sendRedirect("login.jsp");
    }
}

```

## Додаток Д. Приклад структури JSP-сторінки

### Register.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Registration</title>
</head>
<body>
    <h1>Registration</h1>
    <form action="register" method="post">
        <input type="text" name="username" placeholder="Username"
required><br>
        <input type="password" name="password" placeholder="Password"
required><br>
        <input type="submit" value="Register">
    </form>
</body>
</html>
```