МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет імені В.Н.Каразіна

Факультет математики і інформатики

Кафедра теоретичної та прикладної інформатики

Кваліфікаційна робота

бакалавр

на тему:

«Використання паттерну MVVM для побудови архітектури системи онлайн продажу квитків»

Виконав: студент 4 курсу, групи МФ-42

спеціальність 122 Комп'ютерні науки

освітньо-професійна програма

«Інформатика»

Чантурія Георгій Нукрійович

Керівник ____ Меняйлов Є.С._____

Рецензент_____

(прізвище та ініціали)

Харків – 2023 року

ЗМІСТ

1. ВСТУП
   1.1. Формулювання мети роботи, задач та обґрунтування актуальності теми
   1.2. Стислий огляд відомих результатів
   1.3. Відомості про одержані результати та їх новизна
   1.4. Теоретичне та практичне значення результатів, можливі області використання результати
2. ОСНОВНА ЧАСТИНА
   2.1. Постановка задачі
   2.2. Технічні вимоги
   2.3. Розвинутий огляд сучасного стану справ в області
   2.4. Опис моделі вимог до інформаційної системи
   2.5. Опис архітектури інформаційної системи
   2.6. Опис функціональності системи
   2.7. Опис та обґрунтування алгоритмів
   2.8. Використані технології та обґрунтування вибраного інструментарію
   2.9. Результати тестування
   2.10. Результати впровадження
3. ВИСНОВКИ
4. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1.Вступ

1.1 Формулювання мети роботи, задач та обґрунтування актуальності теми

Мета роботи - оновити застосунок для працівників УЗД. Зробити застосунок, який буде більш інтуїтивно зрозумілим та з меншими помилками при входженні. Також планується використання більш сучасних патернів програмування. Таким чином, це дозволить зменшити зв'язність компонентів системи та покращення її масштабованості та підтримки. Це дозволить додати нові фічі та можливості з найменшими зусиллями.

Основна мета - оновити саме той застосунок, який використовується саме для продажу квитків на вокзалі, тому що консольний вигляд - це вже не є актуальним. Та в подальшому застосунок можна буде масштабувати для продажу квитків на інші види транспорту та в перспективі зробити єдину базу для продажу квитків.

Наприклад, при запуску додатку користувачеві буде відображено список доступних поїздів за часом відправлення та прибуття, маршрутом, типом поїзда, вартістю та кількістю вільних місць в кожному вагоні. Користувач зможе вибрати конкретний поїзд, побачити всі доступні вагони та обрати вільне місце для купівлі квитка.

Щоб забезпечити зручність користувача, буде додана можливість фільтрації за типом вагона (наприклад, спальний, купейний, плацкартний) та кількістю вільних місць. Крім того, забезпечиться користувача можливістю переглянути всі деталі поїзда, включаючи додаткову інформацію про маршрут та планований час прибуття.

Також плануэться використати патерн MVVM для розробки застосунку, щоб забезпечити його масштабованість та підтримку в майбутньому. Цей патерн дозволить виділити компоненти додатку, що дозволить легше модифікувати окремі частини програми, додавати нові функції та зменшувати зв'язність між компонентами. Ще можна додати перегляд вагона, в якому планується купити

місце. Це допоможе користувачеві краще уявити умови та обрати оптимальне місце.

Оновлення застосунку для працівників УЗД може бути корисним для багатьох людей.

1.2Стислий огляд відомих результатів

Мобільний додаток "Train Tickets Ukraine":

Розробник: Mobile Dimension LLC

Стек технологій: Java, Kotlin, Swift, Objective-C, React Native

Функціонал: пошук рейсів, купівля квитків онлайн, збереження квитків у додатку, перегляд графіка руху потягів, маршрутів та цін на квитки.

Сайт "booking.uz.gov.ua":

Розробник: Укрзалізниця

Стек технологій: ASP.NET, C#, MS SQL

Функціонал: пошук рейсів, купівля квитків онлайн, збереження квитків у вигляді QR-коду, перегляд графіка руху потягів та маршрутів, інформація про стан руху потягів.

Додаток "Railway Tickets":

Розробник: iTicket

Стек технологій: Java

Функціонал: пошук рейсів, купівля квитків онлайн, збереження квитків у додатку, перегляд графіка руху потягів та маршрутів, інформація про стан руху потягів, замовлення трансферу.

Сайт "uz.gov.ua":

Розробник: Укрзалізниця

Стек технологій: PHP, Drupal

Функціонал: пошук рейсів, купівля квитків онлайн, збереження квитків у вигляді QR-коду, перегляд графіка руху потягів та маршрутів, інформація про стан руху потягів та акції.

Щодо консольного застосунку УЗД, немає точних відомостей про використані технології. Проте, можна зробити припущення, що він може бути написаний на мові програмування C++ (версія C++98 або C++03) та використовувати систему контролю версій, таку як CVS, Subversion (SVN) або Visual SourceSafe (VSS), і бібліотеки, такі як Standard Template Library (STL), Boost, Windows API або інші доступні на той момент.

Щодо розробки додатку, ймовірно, використовувалися такі патерни програмування:

Модель-Вид-Контролер (MVC) - для відокремлення логіки бізнес-логіки від візуального інтерфейсу. Інверсія залежностей (Dependency Inversion) - для розриву залежностей між компонентами програми. Фасад (Facade) - для спрощення складних систем і приховування деталей роботи. Адаптер (Adapter) - для використання об'єктів з різних систем. Одиночка (Singleton) - для створення лише одного екземпляра класу. Фабрика (Factory) - для створення об'єктів без деталей їх створення. Паттерн Strategy (Стратегія) - для використання сімейства алгоритмів та їх замінюваності.

Ці патерни допомагають полегшити розробку, зробити програму більш гнучкою та підтримуваною.

1.3Відомості про одержані результати та їх новизна

Результати звіту надають огляд найпопулярніших додатків та веб-сайтів, використовуваних для купівлі квитків на Укрзалізниці (УЗ). Звіт також містить припущення щодо використаних технологій та патернів програмування для кожного з цих продуктів.

Що стосується новизни, важливо відзначити, що використання мобільних додатків та веб-сайтів для купівлі квитків у галузі транспорту є сучасним підходом, який дозволяє користувачам зручно та швидко здійснювати покупки без необхідності особистого відвідування кас або вокзалів. Це сприяє покращенню досвіду користувача та оптимізації процесу продажу квитків.

У звіті також згадано використання різних технологій та патернів програмування, таких як Java, Kotlin, Swift, Objective-C, React Native, ASP.NET, C#, MS SQL, PHP, Drupal, C++ та інші. Ці технології та патерни є добре відомими та широко використовуються у розробці програмного забезпечення.

Незважаючи на те, що ці технології та патерни не є новими у світі програмування, їх використання в контексті розробки додатків та веб-сайтів для Укрзалізниці відображає актуальні підходи та стандарти в галузі транспортних послуг.

Отже, результати звіту надають інформацію про застосування сучасних технологій та патернів програмування у сфері купівлі квитків на Укрзалізниці, що сприяє покращенню зручності та швидкості обслуговування пасажирів.

1.4Теоретичне та практичне значення результатів, можливі області використання результати

Результати, пов'язані з використанням патерну MVVM (Model-View-ViewModel), мають велике теоретичне значення для побудови архітектури системи онлайн продажу квитків. Основні аспекти, що варто відзначити, включають: розподіл обов'язків: Застосування патерну MVVM дозволяє чітко розділити обов'язки між компонентами системи. Модель (Model) відповідає за обробку даних та бізнес-логіку, Представлення (View) відповідає за візуалізацію даних та взаємодію з користувачем, а ViewModel виконує роль посередника між Моделлю та Представленням, надаючи дані та команди для їх обробки.

Реактивний підхід, MVVM працює на основі зв'язку даних та обробки подій, що дозволяє створювати реактивні інтерфейси користувача. Зміни в даних автоматично відображаються у відповідних представленнях, що спрощує синхронізацію та оновлення інтерфейсу при зміні даних. Тестування, завдяки розділенню логіки та інтерфейсу, MVVM сприяє полегшенню тестування. Модель та ViewModel можуть бути покриті автоматизованими тестами для перевірки правильності обробки даних та бізнес-логіки без прив'язки до конкретного представлення. Використання патерну MVVM для побудови архітектури системи онлайн продажу квитків має практичне значення в ряді областей:

Розширюваність - застосування MVVM дозволяє створити гнучку та легко розширювану систему. Компоненти можуть бути додані або модифіковані з мінімальним впливом на решту системи, що спрощує впровадження нових функціональностей та змін.

Повторне використання - чіткий розподіл обов'язків у MVVM дозволяє повторно використовувати компоненти. ViewModel може бути використаний для різних

представлень, що забезпечує ефективне використання коду та зменшує дублювання.

Розвиток паралельних команд - MVVM сприяє паралельному розвитку різних команд у проекті. Розробники, відповідальні за модель та ViewModel, можуть працювати незалежно один від одного, що спрощує процес розробки та зменшує залежності.

Оптимізація користувацького досвіду - зв'язок даних та реактивний підхід в MVVM дозволяють створювати багатофункціональні та зручні інтерфейси користувача. Відображення змін даних в реальному часі та ефективна обробка подій сприяють поліпшенню користувацького досвіду.

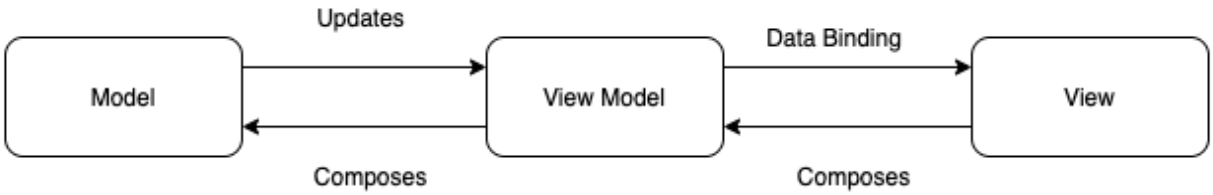


Рис. 1 Високорівнева архітектура MVVM

Області використання результатів:

1 Розробка мобільних додатків для онлайн продажу квитків.
2 Створення веб-порталів для бронювання та купівлі квитків.
3 Розробка систем управління продажами та аналізу даних для залізничних компаній.
4 Впровадження електронних квитків та систем електронного квиткування.
5 Розробка додатків та платформи для управління лояльністю та програмами знижок для пасажирів.

Ці результати можуть бути використані розробниками та архітекторами систем для покращення архітектури та функціональності системи онлайн продажу

квитків, забезпечуючи ефективне використання ресурсів та забезпечуючи задоволення користувачів.

2 ОСНОВНА ЧАСТИНА

2.1 Постановка задачі

Мета даної роботи полягає в оновленні застосунку для працівників УЗД (Українська залізниця) з метою поліпшення його функціональності, інтуїтивного сприйняття та зменшення кількості помилок при використанні. Також планується використання більш сучасних патернів програмування з метою зменшення зв'язності компонентів системи та поліпшення її масштабованості та підтримки. Оновлений застосунок буде спрямований саме на продаж квитків на вокзалі, замінюючи застарілий консольний інтерфейс. Проте, планується зробити цей застосунок масштабованим, щоб у майбутньому він міг бути використаний для продажу квитків на інші види транспорту. Віддалена перспектива полягає в створенні єдиної бази для продажу квитків на різні види транспорту.

Технології, які будуть використані в проекті, включають Java Spring, Hibernate, MySQL, ApiService, HTML та CSS. Задачі проекту включають: розробка інтуїтивного та привабливого веб-інтерфейсу для продажу квитків, що замінить консольний вигляд застосунку. Впровадження патерну MVVM (Model-View-ViewModel) для побудови архітектури системи. Цей патерн дозволить розділити логіку бізнес-логіки (модель), відображення даних (представлення) та управління даними (ViewModel), забезпечуючи зв'язність та розширюваність компонентів. Використання Java Spring для розробки серверної частини застосунку, забезпечуючи роботу з HTTP-запитами, обробку бізнес-логіки та взаємодію з базою даних. Використання Hibernate для забезпечення об'єктно-реляційного відображення та управління персистентними об'єктами в базі даних MySQL.
Розробка ApiService для взаємодії з зовнішніми сервісами та отримання необхідних даних для продажу квитків.

Реалізація функцій продажу квитків, оплати, перегляду розкладу, вибору місць тощо з використанням вищезгаданих технологій. Таким чином, результати даної роботи матимуть наступні теоретичне та практичне значення:

Теоретичне значення - виявлення і застосування патерну MVVM для побудови архітектури системи онлайн продажу квитків. Дослідження та аналіз сучасних патернів програмування та їх впровадження в розробку веб-додатків. Вивчення можливостей та переваг використання Java Spring, Hibernate, MySQL, ApiService, HTML та CSS у контексті розробки систем продажу квитків.

Практичне значення - оновлений застосунок з поліпшеною функціональністю та інтуїтивно зрозумілим веб-інтерфейсом для продажу квитків на вокзалі. Зменшення помилок та покращення взаємодії працівників УЗД з застосунком, що призведе до підвищення продуктивності та ефективності роботи. Можливість масштабування застосунку для продажу квитків на інші види транспорту. Підвищення зручності та задоволення користувачів під час використання застосунку.

Можливі області використання результатів проекту:

Транспортні компанії - результати проекту можуть бути використані іншими транспортними компаніями для оновлення своїх систем продажу квитків, поліпшення взаємодії з клієнтами та забезпечення більш зручного та ефективного процесу продажу.

Інші сфери - застосунок може бути адаптований та використаний у інших галузях, які потребують продажу квитків, наприклад, виставки, конференції, кіно, спортивні події тощо.

Розробка програмного забезпечення - результати проекту можуть бути використані розробниками програмного забезпечення як приклад використання

патерну MVVM та сучасних технологій для побудови масштабованих та підтримуваних систем.

Академічна галузь - результати проекту можуть бути використані в навчальних цілях для вивчення патернів програмування, архітектурних рішень та розробки веб-додатків з використанням популярних технологій.

Загалом, оновлення застосунку для продажу квитків з використанням патерну MVVM та сучасних технологій має великий потенціал у поліпшенні функціональності, зручності використання та масштабованості системи продажу квитків. Додаток можна масштабувати до більш детального, як для звичайних користувачів, так і навпаки для працівників, зокрема для касових працівників вокзалів.

2.2 Технічні вимоги

Мова програмування, яку обирано для використання в архітектурі системи онлайн продажу квитків з використанням патерну MVVM, - це Java. Використання Java є відповідним в даному випадку, оскільки вже зазначино використання Java Spring та Hibernate. Ці технології забезпечують зручне та ефективне програмування на мові Java, що відповідає вимогам патерну MVVM. Обирана Java як мову програмування, оскільки вона має широкий спектр інструментів та бібліотек для побудови архітектурних шаблонів, включаючи підтримку MVVM. З Java Spring та Hibernate можливо легко реалізувати складні бізнес-логіки та здійснювати доступ до бази даних.

Таким чином, використання мови програмування Java у поєднанні з Java Spring та Hibernate дозволяє побудувати потужну та масштабовану систему продажу квитків з використанням патерну MVVM. Обирая використання фреймворку, який сприяє реалізації MVVM, у проекті з побудови архітектури системи онлайн продажу квитків. Java Spring обрано як фреймворк для проекту, оскільки він надає потужні інструменти для розробки веб-додатків і підтримує патерн MVVM. За допомогою Spring Framework можливо легко створювати та управляти компонентами системи, такими як контролери, сервіси та репозиторії .Використання Java Spring дозволить мені ефективно розділити логіку додатку на модель, представлення та відображення. Spring Framework надає можливості для зв'язування даних, реалізації бізнес-логіки та управління переходами між різними екранами. Завдяки використанню Java Spring, я зможу легко керувати залежностями, забезпечити швидку розробку та тестування, а також забезпечити масштабованість мого додатку для майбутніх розширень.

Отже, використання фреймворку Java Spring допоможе ефективно реалізувати архітектуру системи продажу квитків, забезпечити зручну роботу з моделлю,

представленням та відображенням даних та забезпечити високу продуктивність та розширюваність додатку.

У розробці проекту, будуть створені модель даних, яка відображатимуть основні сутності системи продажу квитків, такі як квитки, рейси, користувачі та інші. Ця модель даних буде незалежною від візуального представлення і міститиме бізнес-логіку. За допомогою Spring Framework, можливо визначити класи, які представлятимуть сутності системи, та встановити між ними зв'язки, які відображатимуть їх взаємозв'язок в реальному світі. У моделі даних також включатиметься бізнес-логіку, яка буде відповідати за правила обробки даних та виконання різних операцій. Це дозволить забезпечити централізовану та контрольовану обробку даних в системі продажу квитків. Завдяки незалежності моделі даних від візуального представлення, можливо легко змінювати або розширювати інтерфейс користувача, не впливаючи на основну структуру даних та бізнес-логіку системи. Це забезпечить гнучкість та масштабованість додатку.

Отже, розробка моделі даних з використанням Java Spring дозволить створити стійку до змін та потужну основу для системи продажу квитків, яка буде містити необхідні сутності та бізнес-логіку для ефективної роботи з даними. У проект і візуальне представлення або інтерфейс користувача, який відображатиме дані з моделі та забезпечуватиме можливість взаємодії з користувачем на HTML, CSS та JavaScript для реалізації цього. За допомогою HTML, буде створена структура сторінок та елементів інтерфейсу. Це дозволить створити розмітку, відображати текстові елементи, зображення та інші візуальні компоненти. CSS використовуватиметься для стилізації інтерфейсу. З можливістю визначити кольори, шрифти, розміри та розташування елементів, щоб створити привабливий та зручний дизайн. JavaScript буде використовуватися для додавання динамічності до інтерфейсу. За допомогою JavaScript, можливо створювати взаємодію з користувачем, обробляти події, валідувати введені дані

та забезпечувати асинхронну взаємодію з сервером. Це візуальне представлення дозволить користувачам бачити та маніпулювати даними з моделі у зручний та інтуїтивно зрозумілий спосіб. Вони зможуть виконувати операції, такі як перегляд рейсів, вибір квитків та оформлення замовлення.

Візуальну представлення сприяє задоволенню користувачів та полегшить їх взаємодію з системою продажу квитків. Прив'язки даних між моделлю та представленням - це важлива частина архітектури системи продажу квитків, яка забезпечує синхронізацію даних із їх візуальним представленням. Для досягнення прив'язки даних, використовується спеціальні засоби, які надаються фреймворком Java Spring або розробляю власноручно механізми прив'язки даних. Ці засоби дозволяють встановлювати зв'язок між моделью даних та візуальними компонентами, що відображають ці дані.

Наприклад, можливо використовувати анотації Spring для прив'язки полів моделі до елементів у веб-інтерфейсі. Це дозволяє автоматично оновлювати значення відповідних елементів при зміні даних у моделі, а також передавати зміни в модель при взаємодії користувача з візуальними компонентами. Також можливо використовувати розроблені механізми прив'язки даних, наприклад, шляхом встановлення слухачів на зміни даних у моделі та відповідне оновлення візуальних елементів. Забезпечення прив'язки даних дозволяє забезпечити консистентність даних у всій системі та забезпечує їх автоматичне оновлення та синхронізацію з візуальним представленням. Це робить взаємодію користувачів з системою продажу квитків зручною та ефективною, оскільки вони завжди бачать актуальні дані та можуть безперешкодно взаємодіяти з ними.

ViewModel виступає проміжним звеном між моделлю даних та представленням. ViewModel відіграє важливу роль у системі продажу квитків, оскільки він відповідає за обробку бізнес-логіки та постачання даних для візуального

представлення. У коді розробляю ViewModel згідно з паттерном MVVM та його баченням, де він включає в себе необхідні поля та методи для обробки логіки та управління даними. ViewModel отримує дані від моделі та опрацьовує їх з урахуванням бізнес-правил та вимог системи.

Наприклад, в ViewModel реалізуются методи для отримання списку доступних рейсів, купівлі квитків, обробки платежів та багато іншого. Ці методи виконуються на основі вхідних даних з моделі та здійснюють необхідні перетворення та обчислення, щоб забезпечити потрібні результати для візуального представлення. Крім того, ViewModel забезпечує розробку та використання спеціальних об'єктів, які дозволяють зберігати стан візуальних елементів та відстежувати їх зміни. Це дозволяє підтримувати біндінг даних між ViewModel та представленням, що забезпечує автоматичне оновлення відображення при зміні даних у ViewModel.

Загалом, розробка ViewModel дозволяє відокремити бізнес-логіку від візуального представлення та забезпечити ефективну комунікацію між ними. Це сприяє покращенню розширюваності, тестирування та обслуговування системи продажу квитків.

2.3 Розвинутий огляд сучасного стану справ в області

На сьогоднішній день розробка додатків для продажу квитків перебуває на високому рівні розвитку, завдяки постійному зростанню популярності онлайн-купівлі квитків і зручності, яку це надає користувачам. Ось деякі ключові аспекти та сучасні тенденції в цій галузі:
з мобільними пристроями стає все популярнішими, додатки для продажу квитків стали дуже зручними для користувачів. Розробни зосереджуються на створенні додатків для різних платформ, таких як iOS та Android, щоб користувачі могли з легкістю купувати квитки через свої смартфони. Надається можливість отримувати електронні квитки, які зберігаються безпосередньо на їх пристроях через пошту, що робить процес отримання та використання квитків набагато зручнішим.

Інтеграція різних платіжних систем у додатку для продажу квитків дозволяє користувачам зручно та безпечно оплачувати квитки. Зосереджуючись на використанні безпечних платіжних шлюзів та шифруванні даних, щоб забезпечити високий рівень безпеки та захисту особистої інформації користувачів. Ретельно обираючи надійні платіжні системи, що дозволяють нашим користувачам здійснювати платежі з впевненістю і спокоєм. Додатки забезпечують зручний та безпечний процес онлайн-платежів, що є важливим аспектом сучасних додатків для продажу квитків. Додатки для продажу квитків розроблюється функціонал, який підтримує різні види квитків. Користувачі можуть зручно купувати квитки не лише на події, такі як концерти, спортивні змагання чи кіносеанси, але й на різні види транспорту, такі як авіація, залізниця, автобуси та багато інших. Це дозволяє створити більш універсальні та зручні додатки для користувачів, які можуть задовольнити їх різноманітні потреби у придбанні квитків. Працючи над тим, щоб забезпечити широкий спектр варіантів

та можливостей для користувачів при покупці квитків на різні види подій та транспорту.

У стеку технологій для розробки додатків для продажу квитків, також використовуються технології штучного інтелекту. Завдяки цим технологіям, можливо надати користувачам персоналізовані рекомендації, аналізувати їх поведінку та прогнозувати попит на квитки.

Один з підходів полягає в застосуванні алгоритмів машинного навчання для аналізу великого обсягу даних про користувачів, їх вибірки квитків, а також факторів, що впливають на їх вибір. Використовуя такі техніки, як колаборативний фільтринг та аналіз змісту, щоб надати користувачам рекомендації на основі їхніх інтересів та попередніх вибірок. Крім того, Використовуя технології обробки природної мови (Natural Language Processing, NLP) для аналізу коментарів та відгуків користувачів щодо подій та квитків. Це дозволяє отримати інформацію про задоволення користувачів, їхні побажання та погляди, що може використовуватися для вдосконалення послуг та покращення користувацького досвіду.

Загалом, використання технологій штучного інтелекту в додатках дозволяє надати більш персоналізований та кращий сервіс нашим користувачам, а також покращити процес продажу квитків шляхом аналізу та прогнозування попиту.

2.4 Опис моделі вимог до інформаційної системи

Функціональні вимоги

- Користувачі повинні мати змогу шукати доступні поїзди та маршрути
- Користувачі повинні мати змогу бронювати квитки на певний поїзд та маршрут
- Користувачі повинні мати змогу переглядати історію своїх бронювань та скасовувати бронювання
- Адміністратори повинні мати змогу керувати інформацією про поїзд та маршрут
- Адміністратори повинні мати змогу переглядати звіти та статистику щодо бронювань

Нефункціональні вимоги

- Додаток повинен бути масштабованим та продуктивним для обробки великої кількості користувачів та бронювань
- Додаток повинен бути безпечним, з реалізованою аутентифікацією та авторизацією користувачів
- Додаток повинен бути адаптивним та доступним для користувачів на різних пристроях та платформах

Архітектура

Високорівнева архітектура

Додаток буде побудований з використанням Spring Framework та використовувати архітектурний патерн Model-View-ViewModel (MVVM). Додаток буде складатися з трьох рівнів: рівня моделі, рівня представлення та рівня контролера.

Рівень моделі

Рівень моделі буде відповідати за управління даними та бізнес-логікою додатка. Він буде включати наступні компоненти:

Сутності: класи Java, що представляють сутності в системі, такі як користувачі, поїзди, маршрути та бронювання.

Репозиторії: репозиторії Spring Data для управління постійністю сутностей в базі даних.

2.5 Опис архітектури інформаційної системи

Архітектура системи онлайн продажу квитків базується на використанні патерну MVVM (Model-View-ViewModel) та використанні таких технологій як Java Spring, HTML, CSS та Hibernate. Модель в системі продажу квитків представляє основні сутності, які використовуються в процесі продажу квитків, такі як квитки, рейси, користувачі та інші. Вона виступає незалежною від візуального представлення частиною системи і містить бізнес-логіку, що відповідає за обробку та маніпуляції даними.

Модель виконує кілька основних функцій:

Управління даними: модель забезпечує доступ до даних, які використовуються в системі продажу квитків. Вона взаємодіє з базою даних за допомогою Hibernate, що дозволяє зручно взаємодіяти з об'єктами бази даних та виконувати операції збереження, оновлення, видалення та отримання даних.

Бізнес-логіка: модель містить бізнес-логіку, яка визначає правила та операції, що пов'язані з продажем квитків. Наприклад, вона може включати логіку перевірки доступності квитків, розрахунок вартості, обробку замовлень та інші важливі аспекти продажу квитків.

Валідація даних: модель відповідає за перевірку та валідацію даних, що вводяться користувачами. Вона перевіряє правильність формату даних, обмеження щодо введених значень та забезпечує консистентність та точність даних, що зберігаються в системі.

Обробка подій: модель може включати обробку подій, які відбуваються в системі, наприклад, підтвердження замовлення, відміна або зміна квитків. Вона відповідає за виконання необхідних дій у відповідь на ці події та забезпечення консистентності даних.

Використання Hibernate дозволяє зручно працювати з базою даних, використовуючи об'єктно-реляційне відображення. Він забезпечує автоматичне

перетворення об'єктів моделі в записи бази даних та навпаки, спрощуючи роботу з даними та забезпечуючи їх цілісність та безпеку.

Загалом, модель виконує ключову роль у системі продажу квитків, забезпечуючи правильну обробку даних та реалізацію бізнес-логіки, незалежно від візуального представлення.

View (Представлення) в системі продажу квитків відповідає за створення візуального інтерфейсу, який користувачі можуть використовувати для взаємодії з системою. Для створення цього інтерфейсу використовуються HTML та CSS, що дозволяє створювати зручний та естетичний дизайн. У представленні відображаються дані, які надаються ViewModel. ViewModel виконує підготовчу роботу та постачає необхідні дані для відображення, а представлення бере ці дані та створює візуальну репрезентацію. У системі продажу квитків представлення може містити різні елементи, такі як форми для пошуку квитків, списки подій, деталі квитків та інші. Крім того, можуть бути використані різні елементи інтерфейсу, такі як кнопки, посилання, картинки та інші, для поліпшення користувацького досвіду.

Важливо, щоб представлення було інтуїтивно зрозумілим та привабливим для користувачів. Добре спроектоване та естетичне представлення допомагає залучити користувачів та забезпечити позитивний враження від використання системи продажу квитків. ViewModel в архітектурі MVVM є важливим компонентом, який відповідає за зв'язок між моделлю та представленням. Вона виконує функції обробки бізнес-логіки та постачання даних для відображення у представленні. За допомогою Java Spring, ViewModel отримує доступ до моделі та її даних. Вона може виконувати різні операції та обробку даних залежно від потреб системи продажу квитків. Наприклад, це може включати фільтрацію або

сортування списку квитків, перевірку доступності квитків, розрахунок вартості та інші операції, необхідні для правильного відображення та роботи з даними. ViewModel також може включати механізми прив'язки даних, які дозволяють автоматично оновлювати представлення при зміні даних моделі. Це забезпечує синхронізацію між моделлю та представленням, що дозволяє відображати актуальні дані та реагувати на їх зміни в режимі реального часу.

ViewModel також може містити додаткові методи та функції, які виконуються при взаємодії користувача з представленням. Наприклад, це можуть бути функції обробки подій кнопок або форм для збереження даних або виконання певних операцій.

Використання ViewModel допомагає розділити логіку та обробку даних від візуального представлення, що полегшує розробку, тестування та підтримку системи продажу квитків. Вона дозволяє ефективно керувати даними та забезпечує простий та чистий спосіб взаємодії між моделлю та представленням.

Для збереження даних про квитки, рейси та користувачів використовується база даних. Hibernate, який використовується у системі продажу квитків, надає підтримку об'єктно-реляційного відображення (ORM). Це означає, що реляційні дані з бази даних можуть бути представлені у вигляді об'єктів Java, що дозволяє розробникам зручно працювати з даними без необхідності безпосередньо взаємодіяти з базою даних.

Hibernate використовує анотації або XML-файли для відображення об'єктів Java на таблиці бази даних і встановлює зв'язок між ними. Він автоматично генерує та виконує SQL-запити для доступу до даних, що дозволяє розробникам працювати на вищому рівні абстракції та уникнути написання складних SQL-запитів вручну. Hibernate також забезпечує кешування даних, що сприяє поліпшенню продуктивності системи. Він може кешувати часто використовувані об'єкти та результати запитів, щоб уникнути зайвих запитів до бази даних і забезпечити

швидкий доступ до даних. Hibernate спрощує розробку та підтримку системи, оскільки розробнику не потрібно безпосередньо взаємодіяти з SQL-запитами та таблицями бази даних. Замість цього, розробник може працювати з об'єктами Java, які автоматично відображаються на відповідні таблиці бази даних. Всі ці компоненти взаємодіють між собою за допомогою визначених інтерфейсів та механізмів обміну даними. Використання патерну MVVM дозволяє розділити бізнес-логіку, візуальне представлення та збереження даних у незалежні компоненти, що спрощує розробку, тестування та підтримку системи. Технології Java Spring, HTML, CSS та Hibernate надають потужний інструментарій для реалізації цієї архітектури та забезпечують ефективну та надійну роботу системи онлайн продажу квитків.

2.6 Опис функціональності системи

Для реєстрації використовують настпуні поля: ім’я, прізвище, електронна пошта, номер телефону, пароль, підтвердження паролю. Після чого використовується комбінація аутентифікації на основі електронної пошти та пароля. Це найпоширеніший та зручний спосіб для веб-додатка. Користувач повинен ввести свою електронну пошту та пароль відповідно до поля введення на сторінці реєстрації. Валідація - приймається введена інформація та перевіряється на відповідність заданим вимогам, наприклад, довжині пароля або правильності формату електронної пошти. Зареєстровані дані, такі як електронна пошта та хешований пароль, зберігаються в базі даних для подальшого використання. Підтвердження реєстрації, на електронну пошту, вказану користувачем під час реєстрації, відправляється лист з посиланням для підтвердження реєстрації. Користувач повинен перейти за посиланням для активації облікового запису. Для забезпечення безпеки, рекомендується зберігати хешований пароль замість самого пароля в базі даних. Під час аутентифікації, введений пароль з хешованим паролем порівнюється для перевірки. Додаткові функції можуть включати відправку листа підтвердження, відновлення паролю, перевірку сили паролю та інші механізми безпеки.

На домашній сторінці можно побачити найпопулярніші маршрути, натиснувши на копку “Переглянути” користувача перекидує на сторінку покупки квитку з полями пунк відправлення та прибуття, дата відправлення та прибуття. Полям пунк відправлення та прибуття будуть заповнені значеннями які були на домашній сторінці, обираєш дату та натискаєш “ Далі”. На наступній сторінці обираєшь номер вагону та місце в ньому, після натискаешь кнопку “Купити” і обираєш способ оплати. Деталі стосовно квитків, вагонів та іншого будуть відображатись впродовж процесу, якщо користувач авторизован то він може преглянути своє замовлення в особистом кабінеті разом з іншими замовленнями

і історією покупок. Можливо повернути квиток якщо користувач авторизован, для кожного квитка різні умови. В разі якщо користувач не авторизован то білет надсилається на почту з контактами підтримки.

Система оглядів та рейтингу не була реалізована але ось як бачення цього. Система дозволить користувачам залишати огляди та оцінювати різні аспекти процесу покупки та використання квитків. Вона надасть корисну інформацію для інших користувачів та допоможе покращити якість обслуговування. Користувачі повинні мати обліковий запис, який дозволяє їм залишати огляди та оцінювати послуги. Користувачі можуть залишати огляди про процес покупки квитків, якість обслуговування, комфорт поїздки тощо. Для кожного огляду зберігається текстовий коментар, оцінка та дата написання. Користувачі можуть надавати оцінку (наприклад, за шкалою від 1 до 5 зірок) для різних аспектів, таких як комфорт, пунктуальність, якість обслуговування тощо. Рейтинги обчислюються як середнє значення оцінок, отриманих від користувачів. Користувачі можуть сортувати огляди за датою, рейтингом та іншими параметрами. Фільтри можуть допомагати користувачам швидко знайти огляди, які їх цікавлять (наприклад, за місцем походження або пунктом призначення).

Адміністратор повинен мати спеціальні привілеї, щоб увійти в систему та мати доступ до функцій адміністрування. Аутентифікація може здійснюватися за допомогою унікальних облікових записів адміністраторів. Адміністратор може створювати, редагувати та видаляти облікові записи користувачів. Він також може керувати ролями та привілеями користувачів (наприклад, надавати доступ до певних функцій або обмежувати їх права). Адміністратор може надсилати повідомлення користувачам щодо важливих оголошень, змін у системі, акцій тощо. Користувачі можуть також звертатися до адміністратора через систему повідомлень.

Адміністратор може отримувати доступ до статистики, звітів та аналітики щодо продажів квитків, активності користувачів тощо. Це допомагає адміністратору

виявляти потенційні проблеми та оптимізувати процеси в системі. Адміністратор може відповідати на огляди користувачів, вирішувати проблеми та надавати пояснення щодо питань, які виникли. Для запобігання спаму та небажаних повідомлень можуть бути встановлені механізми модерації, наприклад, перевірка належності повідомлення до певної мови або використання CAPTCHA.

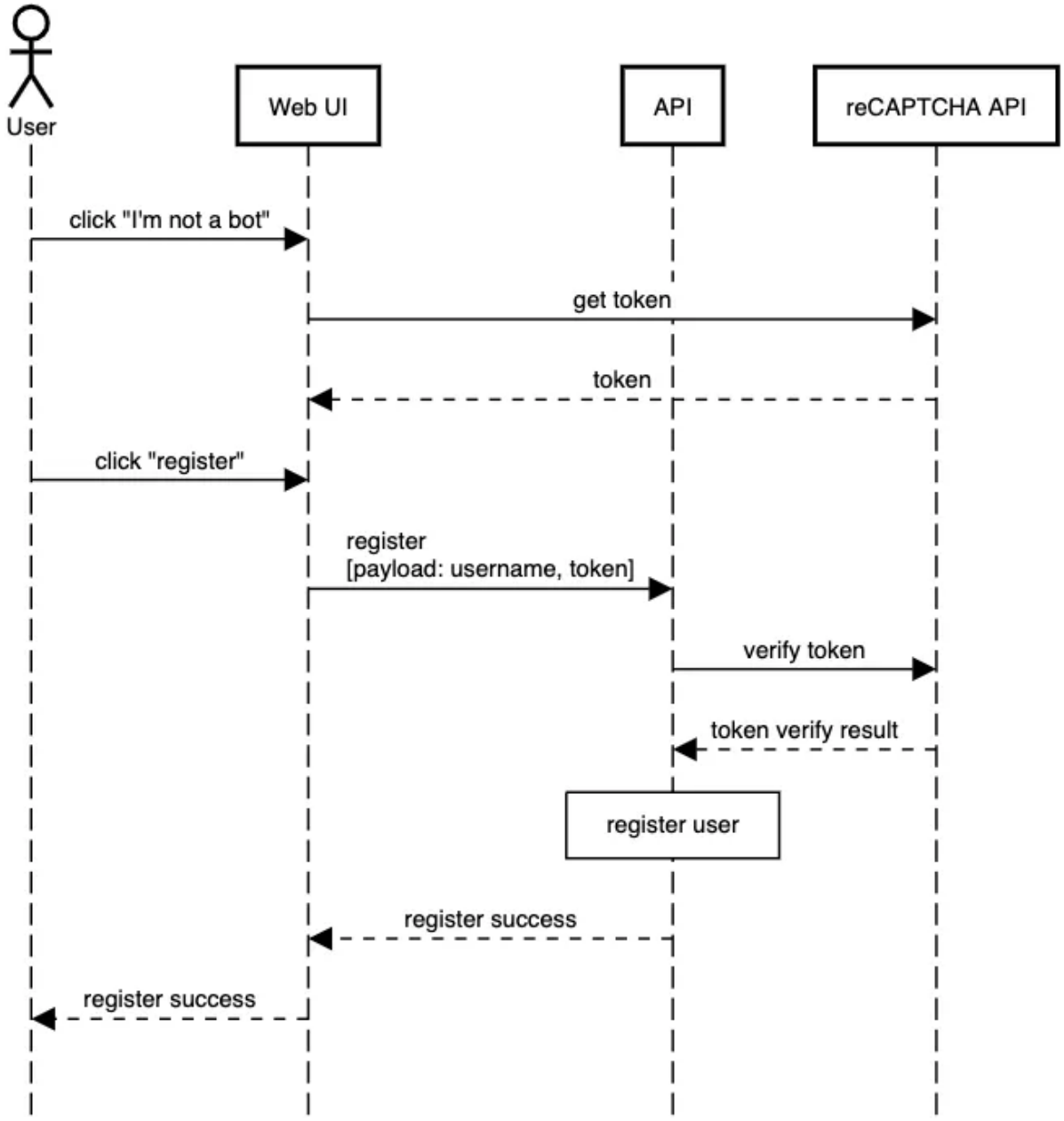


Рис. 2 Діаграм послідовності коли немає проблеми із зображенням

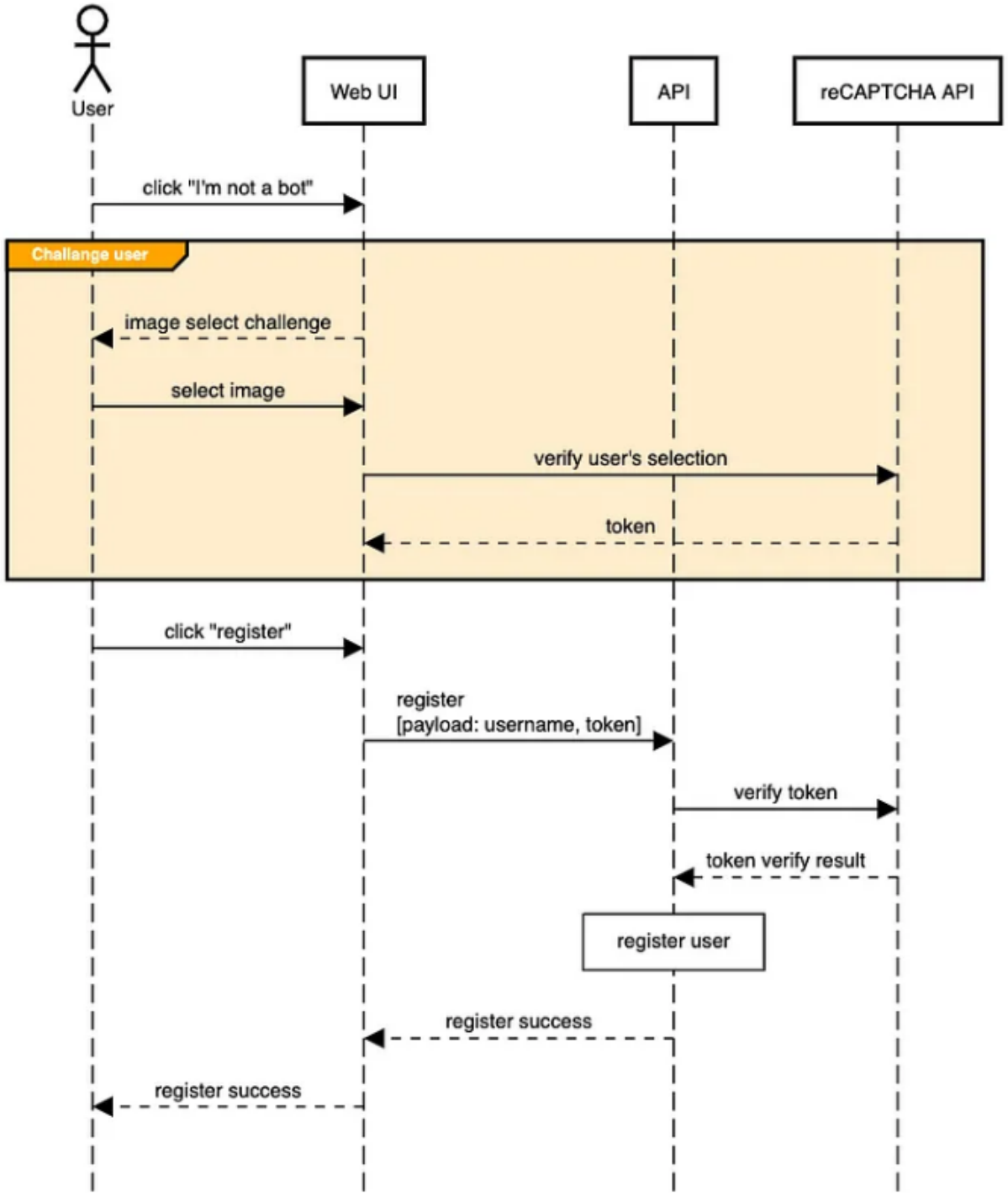


Рис. 3 Діаграм послідовності коли є проблема із зображенням

Система повинна відстежувати та збирати різні дані, які стосуються продажів квитків. Це повинно включати інформацію про кількість проданих квитків, типи

квитків, ціни, час та дату продажу, інформацію про користувачів. Дані будуть зберігатись в базі даних. Система може використовувати алгоритми та методи аналізу даних для обробки зібраних даних.

Наприклад, регресійний аналіз для встановлення зв'язку між різними змінними та прогнозування продажів квитків. Наприклад, ви можете вивчити вплив ціни квитків, дня тижня, часу відправлення тощо на кількість проданих квитків. Аналіз сентименту, методи аналізу сентименту для оцінки настрою або емоцій, виражених користувачами у відгуках, коментарях або соціальних медіа. Це може допомогти вам зрозуміти, як сприймаються події та якість обслуговування, що впливає на продажі. Аналіз часових рядів, методи аналізу часових рядів, такі як ARIMA (авторегресійна інтегрована ковзна середня) або LSTM (довга краткострокова пам'ять) для прогнозування майбутніх продажів квитків на основі історичних даних про продажі. На основі оброблених даних система може створювати звіти, які містять важливі аналітичні дані. Наприклад статистику продажів - це включає загальну кількість проданих квитків, дохід, середній чек, динаміку продажів (наприклад, щомісячні або щорічні зміни), розподіл продажів за різними маршрутами або класами обслуговування. Аналіз популярності подій: Які події або рейси є найбільш популярними серед користувачів - це допоможе вам виявити популярні маршрути та події, які варто враховувати при плануванні пропозицій та маркетингових акцій. Дослідіть вашу базу клієнтів та визначте їхні характеристики, такі як вік, стать, місце проживання, типи подій або рейсів, які вони найчастіше купують. Це допоможе вам створити цільові групи та налаштувати спеціальні пропозиції для різних сегментів клієнтів. Аналіз відгуків та рейтингів, залишені користувачами після подорожей або подій. Це допоможе вам виявити потенційні проблеми, збільшити задоволеність клієнтів та вдосконалити якість обслуговування. Вивчення поведінки користувачів на веб-застосунку, таку як перегляди подій, додавання до кошика, купівлі та відміни. Це

допоможе розуміти, як користувачі взаємодіють з вашим застосунком та виявити можливі моменти, які можна поліпшити для збільшення конверсії та задоволеності користувачів.

Аналіз результатів своїх маркетингових кампаній, ефективність різних каналів маркетингу. Це допоможе вам визначити найбільш успішні канали та стратегії маркетингу. Система повинна мати можливості планування та автоматизації генерації звітів. Наприклад, звіти можуть бути автоматично створювані та надсилатися на електронну пошту адміністраторам або генеруватися за запитом користувача. Загалом, система надання звітів та аналітичних даних допомагає адміністраторам та іншим користувачам отримувати цінну інформацію про продажі, популярність та ефективність системи продажу залізничних квитків. Це дозволяє приймати обґрунтовані рішення, вдосконалювати процеси та досягати кращих результатів.

2.7 Опис та обґрунтування алгоритмів

Алгоритм обробки даних, метод home() відбувається обробка даних про квитки і додавання їх в модель. Це включає перебір списку поїздів та додавання їх властивостей (назва, рейтинг, категорію і тд.) у модель. У методі submitLoginForm() відбувається автентифікація користувача з використанням переданих імені користувача та пароля.

Алгоритми роботи з колекціями, у методі home() ітерується за списком квитків(List<Ticket > films) з використанням циклу for, щоб отримати інформацію про кожен фільм і додати її до моделі. Методи show…() витягують списки даних з бази даних і ці дані додаються до моделі. Алгоритми керування потоком виконання, відбувається перевірка на наявність користувача за допомогою методу loggedIn(). Залежно від результату відбувається повернення різних уявлень. У методі submitLoginForm() відбувається перевірка успішності аутентифікації користувача, і залежно від результату відбувається повернення різних уявлень. Алгоритми роботи з мережею, у методах get”Entity”List() повертаються об'єкт ResponseEntity з інформацією про фільм та відповідним HTTP-статусом. Алгоритми роботи з даними, методи login() та createUser() класу ApiService використовуються для автентифікації користувача та створення нового користувача відповідно. Метод loggedIn() перевіряє, чи увійшов користувач у систему, шляхом перевірки, чи є у об'єкта user значення, відмінне від null. Метод login() здійснює аутентифікацію користувача, перевіряючи відповідність введеного логіну та пароля з даними зі списку користувачів (List<User> temp). Для цього відбувається ітерація за списком користувачів та порівняння значень. Спосіб createUser() створює нового користувача, зберігає його в базі даних за допомогою HibernateHelper і встановлює поточного користувача (this.user) у створеного користувача.

Метод login() перевіряє результати аутентифікації користувача і, залежно від результату, встановлює значення user та повертає відповідний булевий результат. Метод buyTicket() купує квиток за допомогою HibernateHelper та передає ідентифікатор квитка та ідентифікатор поточного користувача для купівлі квитка. Методи використовують HibernateHelper для виконання операцій з базою даних, таких як вилучення списків маршрутыв, користувачів, квитків і т.д.

Веб-застосунок використовується для обробки HTTP-запитів і управління потоком виконання програми. Контролер взаємодіє з класом ApiService надає інтерфейс для взаємодії з базою даних через HibernateHelper. Контролер містить бізнес-логіку програми та взаємодіє з базою даних.

2.8 Використані технології та обґрунтування вибраного інструментарію

Java Spring надає широкий набір інструментів та бібліотек для побудови ефективних і масштабованих додатків. Spring дозволяє створювати компоненти, управляти залежностями, обробляти запити та багато іншого.

Hibernate - Hibernate є ORM (Object-Relational Mapping) фреймворком для роботи з базами даних у Java-додатках. Він забезпечує спрощене управління об'єктами, мапування об'єктів на таблиці бази даних, а також виконання запитів і зв'язків між об'єктами та таблицями. MySQL є однією з найпоширеніших відкритих реляційних баз даних. Використовуючи MySQL, зберігаються та керуються дані про події, квитки, користувачів тощо. MySQL пропонує широкий набір можливостей для оптимізації та управління даними.

ApiService: ApiService буде використовуватись для взаємодії з зовнішніми сервісами, які можуть надавати інформацію про події, розклади поїздів, наявність квитків та іншу додаткову інформацію. Цей сервіс дозволить отримувати актуальні дані та інтегрувати їх у систему продажу квитків.

HTML та CSS: HTML (HyperText Markup Language) використовується для створення структури веб-сторінок, а CSS (Cascading Style Sheets) - для надання їм зовнішнього вигляду та стилізації. Вони будуть використовуватись для розробки користувацького інтерфейсу, відображення інформації про події, форм покупки квитків та інших елементів інтерфейсу.

Ці технології використовуються для побудови потужної та функціональної системи онлайн продажу квитків, яка дозволить користувачам зручно та ефективно шукати, вибирати та придбавати квитки на залізничні події.

Діаграма розгортання UML ілюструє архітектуру розгортання веб-додатку для продавця залізничних квитків. Основним артефактом програми є файл WAR під назвою "ticket_seller.war", який представляє запаковану веб-програму. Артефакт

"ticket_seller.war" розгортається в контейнері JSP 2.0 версії 2.4, який є частиною веб-сервера Apache Tomcat 5.5. Контейнер JSP 2.0 забезпечує необхідне середовище виконання для виконання JavaServer Pages (JSP) і Java Servlets. В артефакті "ticket_seller.war" сберігаються компоненти. Вони обслуговує функції, пов’язані з керуванням та обробкою онлайн-замовлень. Цей компонент проявляється артефактом "ticket_seller.war", який вказує на те, що він міститься в розгорнутій веб-програмі. Крім того, артефакт включає інші артефакти, які сприяють його функціональності. Середовище розгортання включає сервер додатків, представлений як «пристрій» (комп’ютерний сервер). Цей сервер розміщує та виконує розгорнуту веб-програму, обробляючи вхідні запити клієнтів і керуючи середовищем виконання програми. Крім того, існує окремий сервер бази даних, представлений як інший «пристрій» (сервер), який зберігає постійні дані, які використовуються веб-програмою. Сервер додатків має шлях зв’язку з сервером бази даних, що вказує на те, що він взаємодіє з базою даних для виконання операцій з даними та отримання інформації.

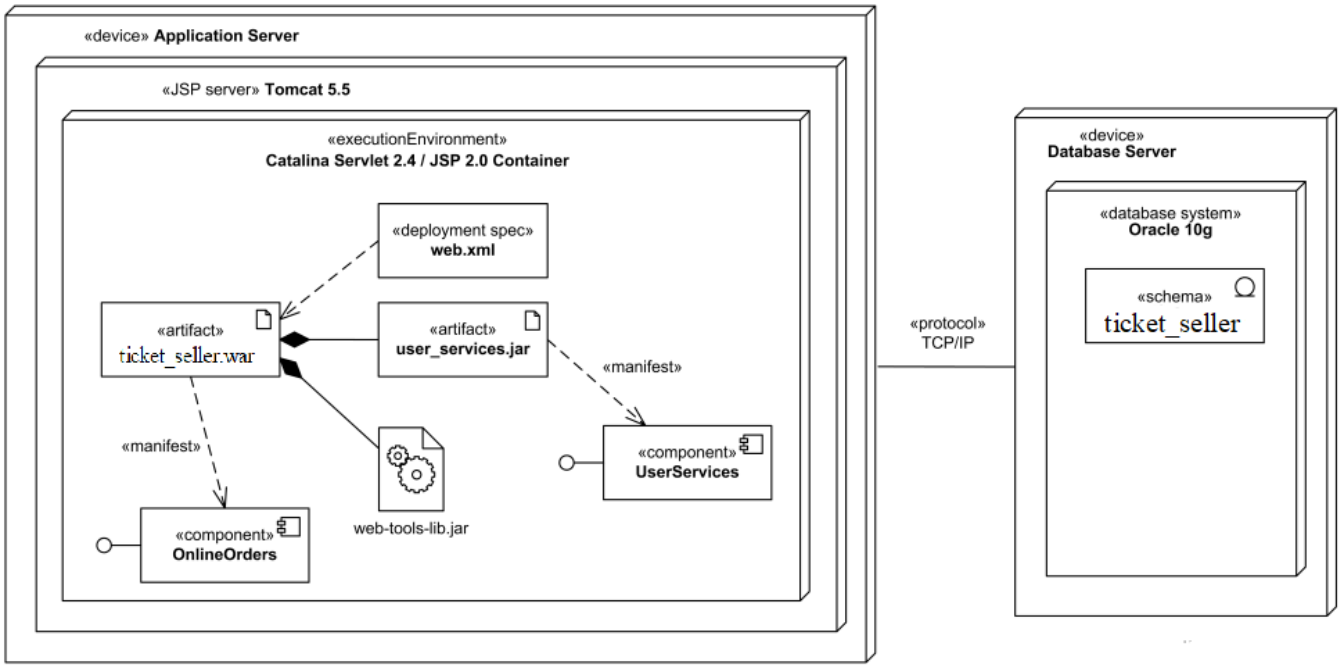


Рис. 4 Веб-додаток

Приклад діаграми розгортання UML

Діаграма послідовності UML представляє потік взаємодій і подій між різними компонентами, задіяними в управлінні транзакціями в програмі Spring, яка інтегрує Hibernate. Діаграма демонструє типовий сценарій, коли викликається бізнес-метод і як обробляється керування транзакціями разом із обробкою винятків. Клієнт ініціює виклик бізнес-методу, викликаючи службу або компонент у програмі Spring. Spring Transaction Interceptor перехоплює виклик бізнес-методу. Цей перехоплювач є частиною фреймворку Spring і відповідає за керування транзакціями. Перехоплювач створює сеанс Hibernate і запускає транзакцію Hibernate JDBC. Цей крок гарантує, що бізнес-метод виконується в контексті нової транзакції. Бізнес-метод виконується в контексті транзакцій. Він виконує різні операції, такі як запити до бази даних, оновлення або будь-яка інша бізнес-логіка.

Під час виконання бізнес-методу може виникнути кілька можливих сценаріїв: Якщо виконання бізнес-методу завершується успішно без винятків, транзакція продовжується.

Якщо бізнес-метод стикається з (неперевіреним) винятком середовища виконання Java, наприклад NullPointerException або IllegalArgumentException, його буде створено бізнес-методом. Якщо бізнес-метод стикається зі специфічною для бізнесу (перевіреною) винятковою ситуацією, такою як помилка перевірки, пов’язана з певним додатком, або користувальницький виняток, вона буде викинута з бізнес-методу. Якщо під час виконання бізнес-методу виникає виняток, Spring Transaction Interceptor перехоплює виняток і виконує необхідну обробку транзакції.

У разі виняткової ситуації перехоплювач ініціює відкат транзакції Hibernate JDBC. Це гарантує, що будь-які зміни, внесені в рамках транзакції, будуть скасовані, а дані залишаться в узгодженому стані. Виняток потім поширюється на відповідний обробник або перехоплюється розв’язувачем винятків у програмі Spring. Обробка винятку може передбачати ведення журналу, звітування про помилки або ініціювання подальших дій на основі конкретного типу винятку. Нарешті клієнту повертається відповідь із зазначенням результату виконання бізнес-методу та статусу транзакції.

Загалом ця діаграма послідовності UML демонструє процес керування транзакціями в додатку Spring, що інтегрує Hibernate. У ньому підкреслюється роль Spring Transaction Interceptor у створенні та управлінні транзакціями, виконанні бізнес-методу в контексті транзакції та обробці винятків для забезпечення узгодженості даних і відповідної обробки помилок.

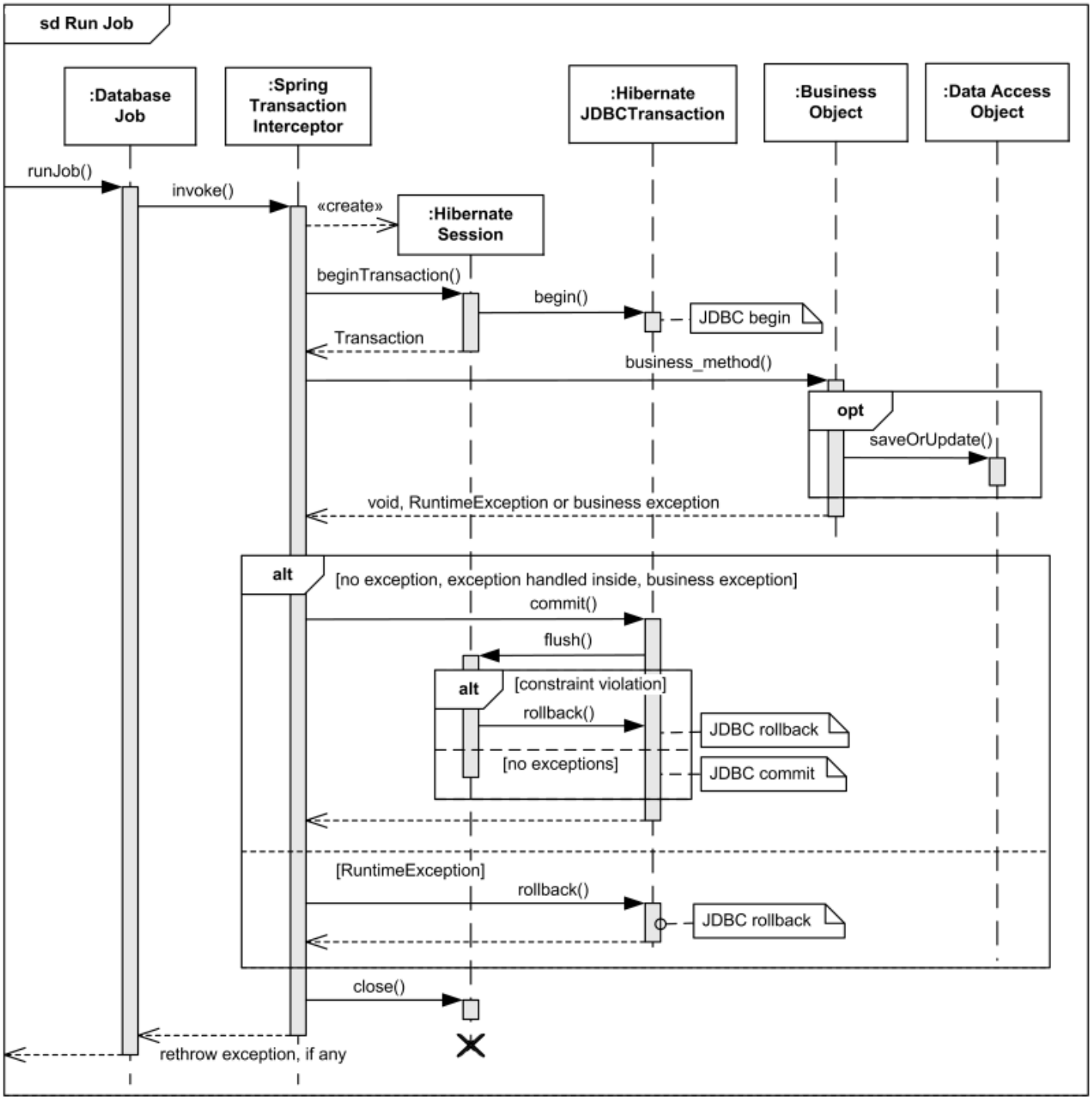


Рис. 5 Транзакції Spring і Hibernate Приклад діаграми послідовності UML

Діаграма пакету UML демонструє організацію та зв'язки між різними пакетами та класами, пов'язаними з доступом до даних у контексті інтеграції Spring і Hibernate. Діаграма складається з кількох пакетів, що представляють різні технології та фреймворки, задіяні в доступі до даних, включаючи Spring

Framework і Hibernate. Пакет «org.springframework» представляє основний пакет Spring Framework, який забезпечує підтримку різних функцій розробки корпоративних додатків, включаючи доступ до даних. У пакеті «org.springframework» є підпакети, пов’язані з доступом до даних, наприклад «org.springframework.jdbc» і «org.springframework.orm». Ці пакети містять класи та інтерфейси, які забезпечують інтеграцію з різними технологіями доступу до даних. Пакет "org.springframework.orm" спеціально зосереджений на інтеграції об'єктно-реляційного відображення (ORM). Він забезпечує підтримку для інтеграції різних фреймворків ORM, включаючи Hibernate, JDO, Oracle TopLink, iBATIS SQL Maps і JPA (Java Persistence API).
Ієрархії визначають стандартизовані класи винятків, які можна використовувати в різних технологіях доступу до даних, забезпечуючи узгоджену обробку помилок і розповсюдження винятків у екосистемі Spring.

Загалом ця діаграма пакетів UML ілюструє організацію та інтеграцію різних пакетів і класів, пов’язаних із доступом до даних у контексті Spring і Hibernate. У ньому підкреслюється підтримка Spring Framework для різних технологій ORM, з особливим акцентом на Hibernate, і наголошується на дотриманні стандартизованих транзакцій та ієрархії винятків DAO у пакетах доступу до даних Spring.

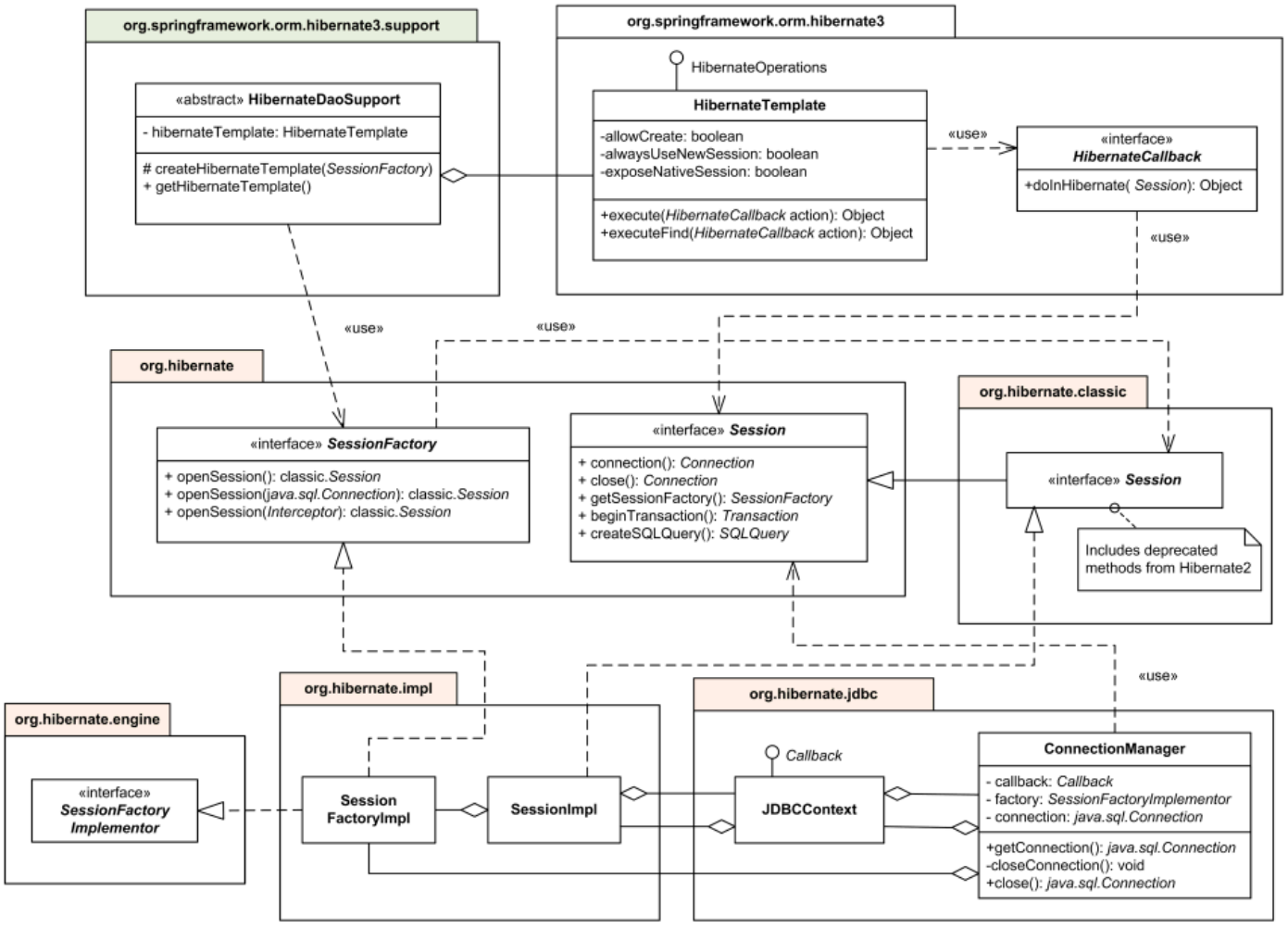


Рис. 6 Spring і Hibernate класи Приклад діаграми пакета UML

Діаграму варіантів використання для веб-програми з інтеграцією Spring і Hibernate описати так:

Актори:

Користувач: представляє користувачів веб-додатку, включаючи клієнтів і адміністраторів.

Випадки використання: Реєстрація користувача: дозволяє користувачеві створити новий обліковий запис у системі, надавши свої дані, такі як ім’я, адреса електронної пошти та пароль.

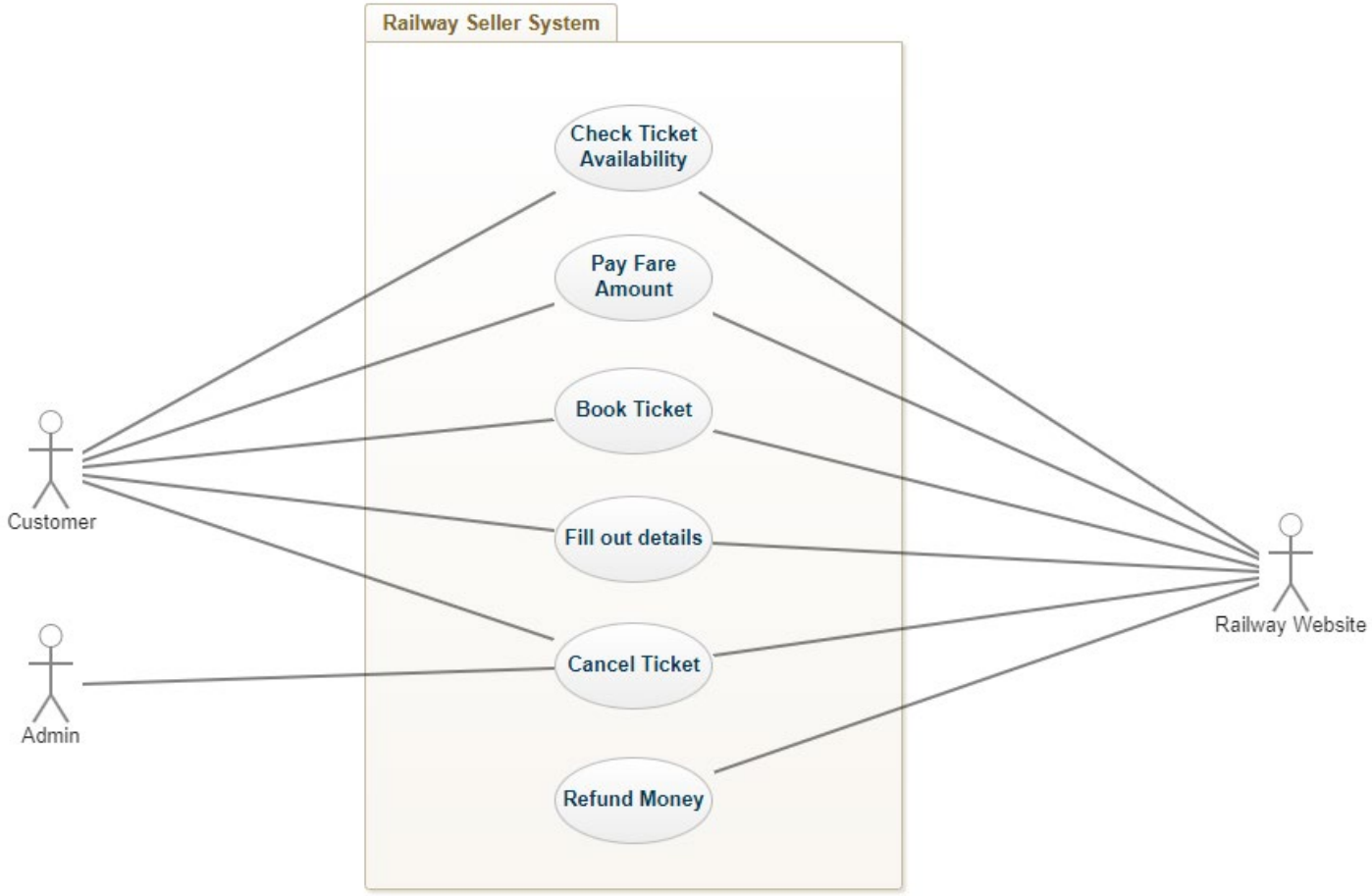
Вхід користувача: дозволяє користувачеві автентифікувати себе, ввівши свої облікові дані (ім’я користувача та пароль) для доступу до свого облікового запису.

Огляд квитків: дозволяє користувачам шукати та переглядати доступні залізничні квитки на різні напрямки.

Придбати квиток: дає змогу користувачам вибрати квиток, надати необхідну інформацію та завершити процес покупки.

Керування квитками: надає адміністративні функції для керування квитками, включаючи додавання нових квитків, оновлення даних квитків і видалення квитків із системи.

Створення звітів: дозволяє адміністраторам створювати різноманітні звіти щодо продажу квитків, доходу, популярних маршрутів тощо.



мал. 7 Use Case Діаграма

Проект був розгорнутий на локальному комп'ютері, але буде оптимальним варіантом використання вертикального масштабування та оптимізація бази даних. Збільшуючи ресурсів на одному сервері, можливо використовувати більш потужне апаратне забезпечення, таке як сервер із швидшим процесором, більшою оперативною пам'яттю або SSD-накопичувачем, щоб збільшити продуктивність системи. Наприклад, можливо розгорнути програму на сервері з більш високою ємністю та продуктивністю для обслуговування більшої кількості користувачів або обробки більшого обсягу даних. Віртуалізація також можливо використовувати віртуалізацію, щоб ефективно використовувати ресурси і масштабувати програму. Віртуалізація дозволяє запускати кілька віртуальних машин чи контейнерів одному сервері, кожен із яких може обробляти частина навантаження. Це дозволяє гнучкіше масштабувати систему залежно від необхідного обсягу роботи. Створення правильних індексів на таблицях бази даних може значно покращити продуктивність запитів. Аналізуйте запити, що часто виконуються, і визначте, які стовпці вимагають індексації для прискорення пошуку та сортування даних.

Оптимізація запитів: можливість оптимізації запитів, такі як використання підзапитів, об'єднань та правильного використання інструкцій JOIN для мінімізації кількості запитів та обсягу даних, що передаються між базою даних та програмою. Використання механізмів кешування, такі як Spring Cache або зовнішні інструменти кешування, щоб зберігати результати запитів, що часто виконуються в пам'яті. Це може суттєво знизити навантаження на базу даних та покращити відгук системи. Розділіть базу даних на окремі сервери або кластери та використовувати механізми реплікації даних, щоб забезпечити високу доступність та розподіл навантаження. Наприклад, можливо використовувати майстер-слейв реплікацію, де записи виконуються одному сервері (майстрі), а

читання виконується інших серверах (слейвах), що дозволяє обробляти більше запитів паралельно.

Ось приклад оцінки характеристик і пропускної здатності сервера для додатка:
Очікувана кількість користувачів: додаток планується обслуговувати 10 000 активних користувачів на день.
Завантаження на сервер: кожен користувач виконує в середньому 3 запити до сервера в хвилину. Це означає, що потрібно обробляти приблизно 30 000 запитів за хвилину або приблизно 500 запитів за секунду.
Процессор: використовуватиметься сервер з кількома високопродуктивними ядрами. Наприклад, сервер з 8 ядрами Intel Xeon з тактової частотою 2,4 ГГц.
Оперативна пам'ять: мати достатньо оперативної пам'яті для обробки запитів і зберігання даних у пам'яті. Наприклад, сервер з 16 ГБ оперативної пам'яті.
Сетевой інтерфейс: Рекомендується мати високошвидке мережеве підключення для обробки великої кількості запитів. Наприклад, 1 Гбіт/с мережевого інтерфейсу.
База даних: MySQL для зберігання даних, рекомендується використовувати сервер баз даних із високою продуктивністю та достатнім об’ємом пам’яті для кешування даних.

У додатку не была реалізована система онлайн платежів, але ось як це можно зробити. В якості платіжного провайдера LiqPay. LiqPay надає набір API та SDK для інтеграції з додатком, а також підтримує різні способи оплати, включаючи платіжні картки Visa та Mastercard, електронні гаманці, банківські перекази та інші. Після чого реєструється та створюється обліковий запис у платіжного провайдера. Після створення облікового запису платіжний провайдер надасть ключі API, такі як ключі API, токени доступу або секретні ключі. Ці ключі дозволять програмі безпечно спілкуватися з серверами платіжного провайдера.

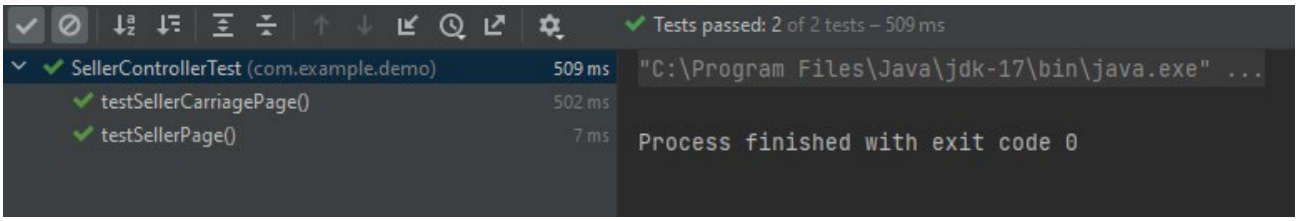
Інтегруэться клієнтська бібліотека або SDK платіжного провайдера у веб-додаток. Безпечна передача данних реалызуэтсья через протокол HTTPS, щоб захистити їх від перехоплення та несанкціонованого доступу. Після чого реалізація серверної частини інтеграції це включає обробку запитів на оплату, перевірку транзакцій та обробку відповідей від серверів платіжного провайдера. Безпечно зберігається API-ключі на сервері та уникає їх розкриття у клієнтському коді. Налаштовується програма для отримання та обробки повідомлень про платіж або вебхуків від платіжного провайдера. Ці повідомлення інформують статус транзакцій, дозволяючи відповідним чином оновлювати систему. Виконується тестування та налагодження після чого відбувається запуск у продакшн.

2.9 Результати тестування

Для тестування було бран фреймворк `MockitoFramework ось кілька прикладів.`

```
class SellerControllerTest {


 @Mock
 private ApiService apiService;


 @Mock
 private Model model;


 @InjectMocks
 private SellerController sellerController;


 @BeforeEach
 void setUp() {
       MockitoAnnotations.openMocks(this);
 }


 @Test
 void testSellerPage() {
       // Mock data
       List<DepartureStation>     departureStations     =     Arrays.asList(new
DepartureStation("Dep1"), new DepartureStation("Dep2"));
       List<ArrivalStation>      arrivalStations      =      Arrays.asList(new
ArrivalStation("Arr1"), new ArrivalStation("Arr2"));


       // Mock method calls
       when(apiService.showDepartureStation()).thenReturn(departureStations);
       when(apiService.showArrivalStation()).thenReturn(arrivalStations);


       // Invoke the method
       String result = sellerController.sellerPage(model);
```

```
        // Verify the interactions
        verify(apiService).showDepartureStation();
        verify(apiService).showArrivalStation();
        verify(model).addAttribute("title", "Ticket selling");
        verify(model).addAttribute("departureStations", departureStations);
        verify(model).addAttribute("arrivalStations", arrivalStations);

        // Perform assertions
        assertEquals("Seller Page", result);
    }

    @Test
    void testSellerCarriagePage() {
        // Invoke the method
        String result = sellerController.sellerCarriagePage(model);

        // Verify the interactions
        verify(model).addAttribute("title", "Ticket selling");

        // Perform assertions
        assertEquals("Seller Carriage Page", result);
    }
}
```



class MainControllerTest {

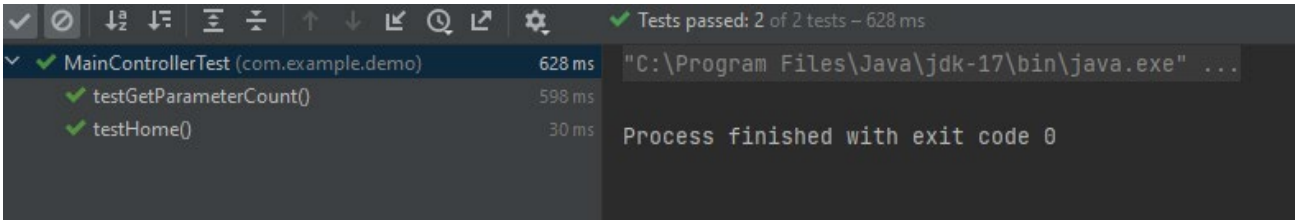
@Mock

private RouteRepository routeRepository;

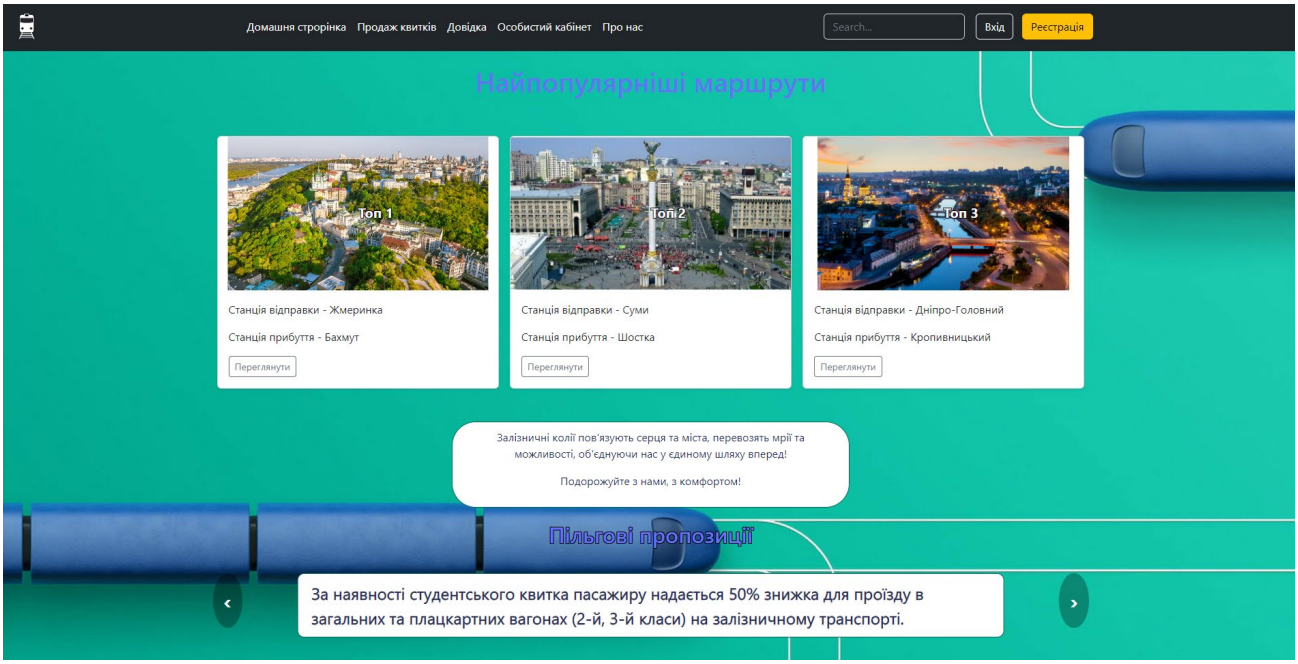
```
    @Mock
    private ApiService apiService;

    @Mock
    private Model model;

    private MainController mainController;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
        mainController = new MainController(routeRepository, apiService);
    }

    @Test
    void testHome() {
        // Mock data
        List<Object[]> topPairs = Arrays.asList(new Object[]{1, 2}, new Object[]{3, 4});
        List<DepartureStation> departureStations = Arrays.asList(new
DepartureStation("Dep1"), new DepartureStation("Dep2"));
        List<ArrivalStation> arrivalStations = Arrays.asList(new ArrivalStation("Arr1"),
new ArrivalStation("Arr2"));

        // Mock method calls
        when(routeRepository.findTopDepartureArrivalPairs()).thenReturn(topPairs);
        when(apiService.showDepartureStation()).thenReturn(departureStations);
```

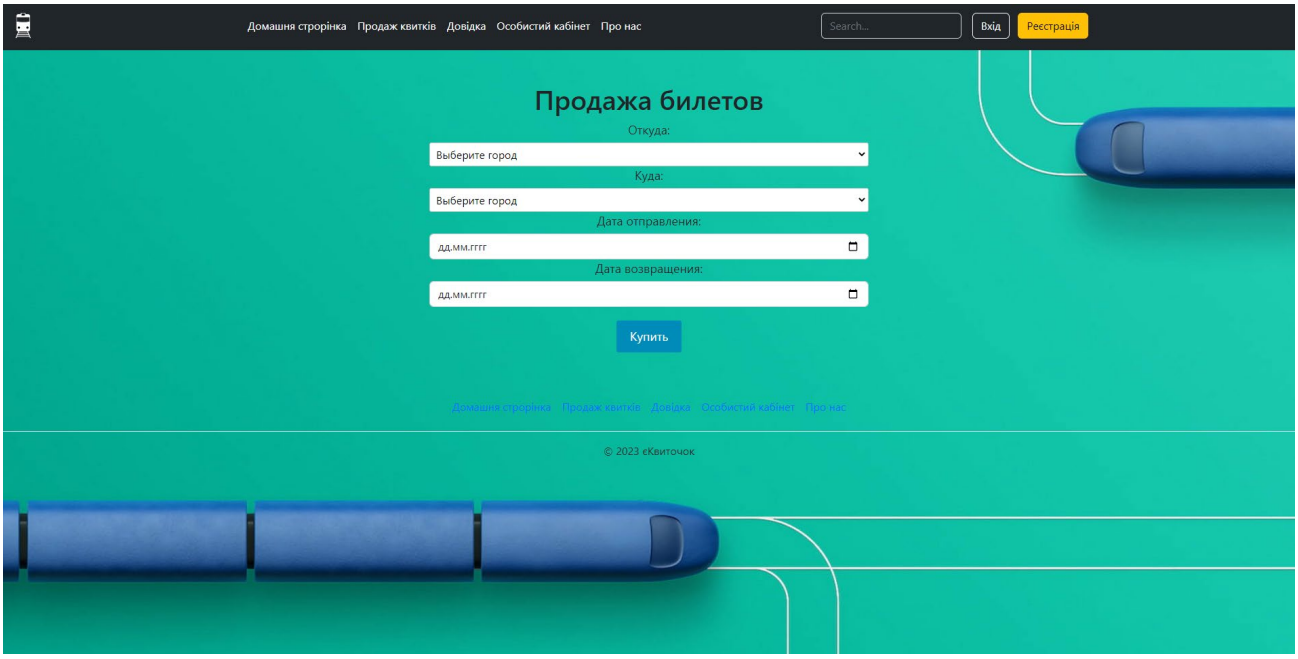
```
    when(apiService.showArrivalStation()).thenReturn(arrivalStations);

    // Invoke the method
    String result = mainController.home(model);

    // Verify the interactions
    verify(model).addAttribute("title", "Home page");
    verify(routeRepository).findTopDepartureArrivalPairs();
    verify(apiService).showDepartureStation();
    verify(apiService).showArrivalStation();

    // Verify that addAttribute is called twice
    verify(model, times(1)).addAttribute(anyString(), anyString());

    // Perform assertions
    assertEquals("Home page", result);
}


@Test
void testGetParameterCount() {
    // Mock data
    class TestObject {
        private String field1;
        private int field2;
    }
```

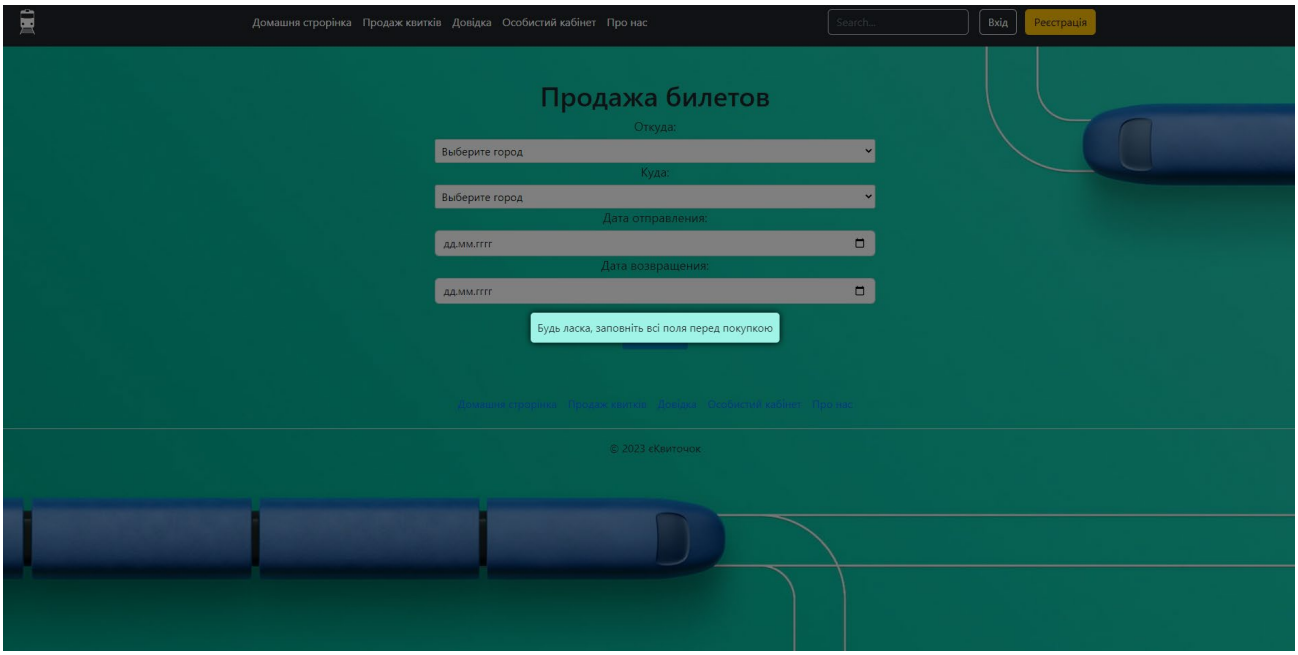
```
        TestObject testObject = new TestObject();

        // Invoke the method
        int result = MainController.getParameterCount(testObject);

        // Perform assertions
        assertEquals(3, result);
    }
}
```

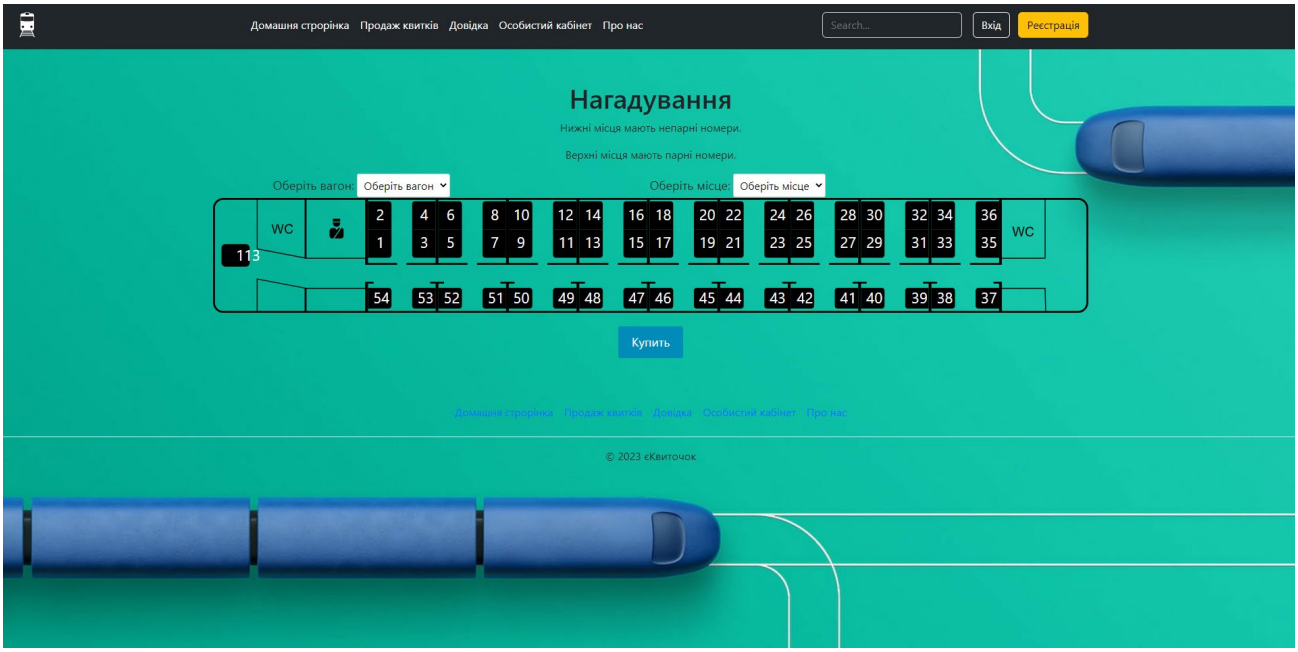


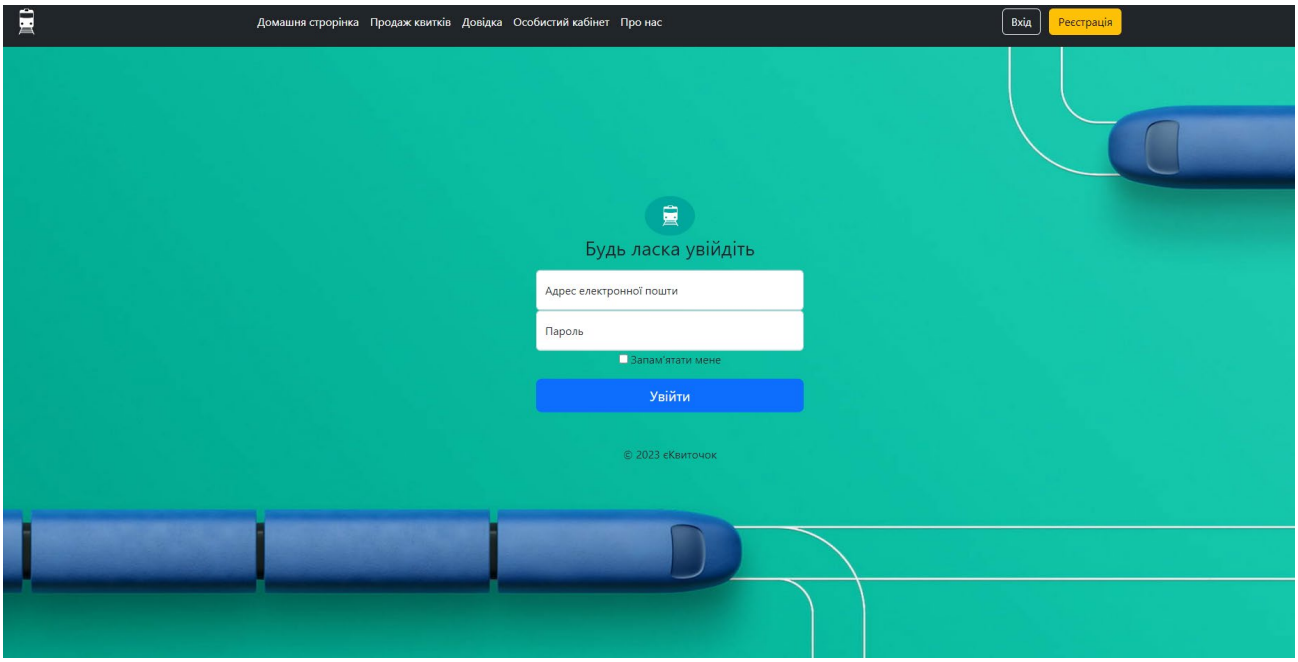
2.10 Результати впровадження

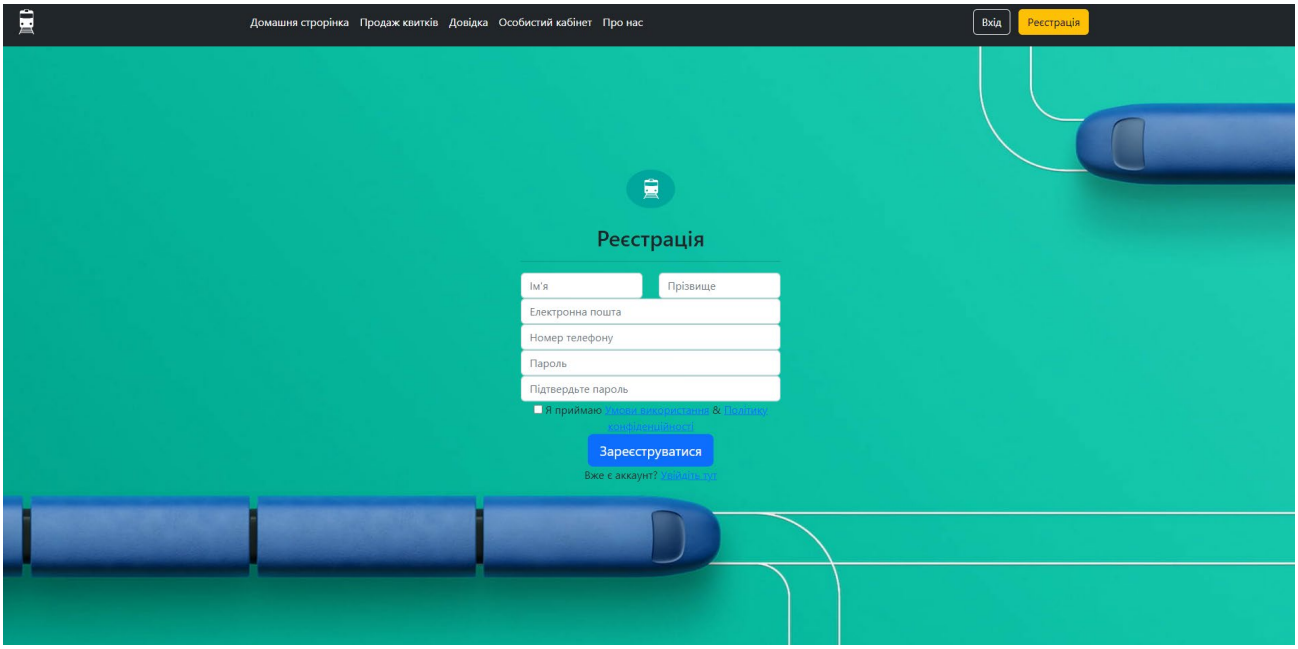





Домашня сторінка
Продаж квитків
Довідка
Особистий кабінет
Про нас
Search...
Вхід
Реєстрація
Продажа билетов
Откуда:
Выберите город
Куда:
Выберите город
Дата отправления:
дд.мм.гггг
Дата возвращения:
дд.мм.гггг
Будь ласка, заповніть всі поля перед покупкою
© 2023 єКвиточок


Домашня сторінка
Продаж квитків
Довідка
Особистий кабінет
Про нас
Search...
Вхід
Реєстрація
Нагадування
Нижні місця мають непарні номери.
Верхні місця мають парні номери.
Оберіть вагон:
Оберіть вагон
Оберіть місце:
Оберіть місце
WC
WC
113
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35
54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37
Купить
© 2023 єКвиточок


Домашня сторінка
Продаж квитків
Довідка
Особистий кабінет
Про нас
Вхід
Реєстрація
Будь ласка увійдіть
Адрес електронної пошти
Пароль
Запам'ятати мене
Увійти
© 2023 єКвиточок


Домашня сторінка
Продаж квитків
Довідка
Особистий кабінет
Про нас
Вхід
Реєстрація
Реєстрація
Ім'я
Прізвище
Електронна пошта
Номер телефону
Пароль
Підтвердьте пароль
Я приймаю Умови використання & Політику конфіденційності
Зареєструватися
Вже є аккаунт? Увійдіть тут

3. ВИСНОВКИ

У ході розробки дипломної роботи на тему "Використання патерну MVVM для побудови архітектури системи онлайн продаж квитків" було проведено докладну аналітичну роботу та оцінку вимог проекту. Використовуючи надану інформацію, було запропоновано архітектуру системи, засновану на патерні MVVM. При розробці системи онлайн продаж квитків були використані такі технології: Java Spring, Hibernate, MySQL, ApiService, HTML та CSS. Java Spring був обраний як основний фреймворк розробки, що забезпечує управління залежностями, інверсію управління, обробку запитів та інші функціональності. Hibernate був інтегрований для управління транзакціями та доступом до даних у базі даних MySQL.

Архітектура системи була побудована на основі патерну MVVM, який розділяє компоненти системи на три основні частини: Model, View та ViewModel. Model представляє бізнес-логіку і дані програми, View відповідає за відображення інтерфейсу користувача, а ViewModel забезпечує зв'язок між Model і View, обробляє дії користувача і оновлює стан Model і View. Переваги використання патерну MVVM в архітектурі системи онлайн продажів квитків полягають у наступному:

Поділ відповідальності: Паттерн MVVM дозволяє явно поділити відповідальності між різними компонентами системи. Model відповідає за бізнес-логіку і доступ до даних, View відповідає за відображення інтерфейсу користувача, а ViewModel служить як посередник між Model і View.

Покращена тестованість: Завдяки розділенню компонентів та явному визначенню інтерфейсів патерн MVVM полегшує тестування кожної частини системи незалежно. Модульні тести можуть бути написані для перевірки функціональності Model, ViewModel та View окремо.

Гнучкість та масштабованість: Паттерн MVVM дозволяє легко змінювати та розширювати функціональність системи. Зміни в Model або ViewModel не потребують модифікації View, що спрощує підтримку та масштабування програми.

Покращена чуйність інтерфейсу користувача: Паттерн MVVM дозволяє реагувати на зміни даних у Model і автоматично оновлювати View. Це забезпечує швидку чуйність інтерфейсу користувача і більш плавна взаємодія з користувачем.

Таким чином, використання патерну MVVM в архітектурі системи онлайн продаж квитків на базі Java Spring і Hibernate дозволяє досягти високої гнучкості, модульності, тестованості та чуйності користувальницького інтерфейсу. Це сприяє розробці ефективної та масштабованої системи для продажу залізничних квитків.

4. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація Spring Framework https://spring.io/guides
2. Офіційна документація Hibernate https://hibernate.org/
3. Patterns of Enterprise Application Architecture 1st Edition by Martin Fowler
4. Spring Boot in Action First Edition by Craig Walls
5. Пільги на проїзд залізничним транспортом - Укрзалізниця https://uz.gov.ua/passengers/ticket_discounts/