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

Факультет інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка веб-додатку круїзної компанії на Java з використанням Spring Framework

Виконав(ла): студент(ка) 4 курсу, групи СП-41
спеціальності 121
«Інженерія програмного забезпечення»
(шифр і назва спеціальності)

(підпис) Марків К. А. (прізвище та ініціали)

Керівник (підпис) Петрик М. Р. (прізвище та ініціали)

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

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

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2024

АНОТАЦІЯ

Кваліфікаційна робота бакалавра. Тернопільський національний технічний університет імені Івана Пулюя, кафедра програмної інженерії, спеціальність 121 «Інженерія програмного забезпечення». ТНТУ, 2024. Сторінок 95, таблиць 2, рисунків 26, додатків 3, презентація.

Тема: Розробка веб-додатку круїзної компанії на Java з використанням Spring Framework.

Основною метою роботи було розробити надійну, масштабовану та зручну у використанні систему, що забезпечує повний цикл обслуговування клієнтів, включаючи управління круїзами, бронювання квитків, керування маршрутами та лайнерами, а також автентифікацію та авторизацію користувачів. Для досягнення цієї мети було обрано технологію Spring Framework, яка дозволяє створювати масштабовані веб-додатки з чіткою архітектурою та високою продуктивністю.

У ході роботи було розроблено багатошарову архітектуру додатку, яка включає контролери для обробки HTTP-запитів, сервіси для реалізації бізнес-логіки та репозиторії для взаємодії з базою даних. Користувацький інтерфейс було створено з урахуванням принципів зручності та інтуїтивності, що забезпечує легкість використання системи.

Результати роботи показали, що розроблений веб-додаток є конкурентоспроможним продуктом, який може бути успішно використаний у сфері круїзних перевезень. Додаток забезпечує високу якість обслуговування клієнтів, має розширені функціональні можливості та простий у використанні інтерфейс. Подальший розвиток системи може включати додавання нових функцій та інтеграцію з іншими сервісами, що дозволить ще більше покращити якість обслуговування клієнтів та підвищити ефективність роботи компанії.

Ключові слова: веб-додаток, круїзна компанія, Java, Spring Framework, багатошарова архітектура, аналіз, розробка, інтерфейс користувача.

ANNOTATION

Bachelor's certification work. Ternopil Ivan Puluj National Technical University, Department of Software Engineering, specialty 121 "Software Engineering". TNTU, 2024. Pages 95, tables 2, figures 26, appendices 3, presentation. Topic: Development of a cruise line web application in Java using the Spring Framework.

The main goal of the work was to develop a reliable, scalable, and user-friendly system that provides a complete cycle of customer service, including cruise management, ticket booking, route and liner management, as well as user authentication and authorization. The Spring Framework technology was chosen to achieve this goal, as it allows the creation of scalable web applications with clear architecture and high performance.

During the work, a multi-layered architecture of the application was developed, which includes controllers for processing HTTP requests, services for implementing business logic, and repositories for interacting with the database. The user interface was designed considering ease of use and intuitiveness, ensuring the system's usability.

The results of the work demonstrated that the developed web application is a competitive product that can be successfully used in the cruise industry. The application ensures high-quality customer service, has extensive functional capabilities, and a simple user interface. Further development of the system may include adding new features and integrating with other services, which will further improve customer service quality and increase the company's operational efficiency.

Keywords: web application, cruise company, Java, Spring Framework, multi-layered architecture, analysis, development, user interface.

ЗМІСТ

ВСТУП

В сучасному світі інформаційні технології відіграють ключову роль в різних галузях бізнесу, забезпечуючи ефективність, зручність та доступність послуг для користувачів. Однією з таких галузей є круїзний бізнес, де веб-додатки стають невід'ємною частиною обслуговування клієнтів, планування подорожей та управління бізнес-процесами. В цьому контексті розробка веб-додатку для круїзної компанії є важливим завданням, що дозволяє забезпечити інтерактивний доступ до інформації про круїзи, бронювання квитків, отримання консультацій та інших послуг.

Ця дипломна робота присвячена розробці веб-додатку для круїзної компанії з використанням мови програмування Java та Spring Framework. Вибір Java як основної мови обумовлений її високою продуктивністю, масштабованістю та надійністю, а Spring Framework надає потужний інструментарій для побудови сучасних веб-додатків з високим рівнем абстракції, що значно спрощує процес розробки та підтримки коду. Сучасні підходи до розробки програмного забезпечення дозволяють створювати масштабовані рішення, здатні задовольнити зростаючі вимоги бізнесу та кінцевих користувачів.

Метою цієї роботи є створення веб-додатку, який дозволить користувачам легко знаходити та бронювати круїзи, отримувати актуальну інформацію про маршрути, судна, послуги на борту та багато іншого. Розроблений додаток повинен бути інтуїтивно зрозумілим, безпечним, продуктивним та масштабованим, щоб задовольнити вимоги сучасних користувачів і забезпечити високий рівень обслуговування. Особливу увагу буде приділено забезпеченню високого рівня безпеки даних, оскільки це є критично важливим аспектом для захисту конфіденційної інформації клієнтів та фінансових транзакцій.

Розробка веб-додатку включає кілька ключових етапів. Перш за все, необхідно провести аналіз вимог до системи та визначити функціональні і нефункціональні вимоги, що будуть визначати основні параметри роботи додатку.

На основі цього аналізу буде розроблена архітектура додатку з використанням Spring Framework, що дозволить забезпечити гнучкість та розширюваність системи.

Далі буде реалізовано основні функціональні можливості, такі як пошук та бронювання круїзів, управління обліковими записами користувачів, інтеграція з платіжними системами та іншими сервісами. Особлива увага буде приділена користувацькому інтерфейсу, який повинен бути зручним та інтуїтивно зрозумілим. Важливим аспектом є також забезпечення безпеки даних та користувацької інформації, що досягається шляхом впровадження сучасних методів аутентифікації та авторизації, шифрування даних та інших заходів безпеки.

Тестування та впровадження системи є завершальними етапами розробки. Тестування проводитиметься на різних рівнях, включаючи модульне тестування, інтеграційне тестування та тестування користувацького інтерфейсу, щоб забезпечити високу якість програмного забезпечення та відповідність його функціональних характеристик вимогам. Впровадження системи включає підготовку інфраструктури для її розгортання, налаштування серверного середовища та запуск у промислову експлуатацію.

Робота над веб-додатком включає не тільки технічну реалізацію, але й вивчення потреб ринку та кінцевих користувачів, що дозволить створити продукт, який максимально відповідає очікуванням та вимогам сучасного круїзного бізнесу. Застосування сучасних підходів до розробки та впровадження веб-додатків на основі Java та Spring Framework дозволить створити надійну, масштабовану та продуктивну систему, яка сприятиме розвитку круїзного бізнесу та покращенню обслуговування клієнтів.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд конкурентів

У теперішній час круїзний бізнес є однією з найбільш динамічних і конкурентних галузей. Популярність круїзних подорожей постійно зростає, що стимулює розвиток компаній, які надають такі послуги. Водночас, постійно зростають вимоги клієнтів до якості обслуговування, зручності бронювання та різноманіття пропонованих маршрутів. Для ефективного функціонування та розвитку круїзної компанії необхідно ретельно аналізувати діяльність основних конкурентів. Це дозволяє визначити сильні та слабкі сторони їх підходів до бізнесу, виявити ринкові тенденції та запозичити найкращі практики. Серед численних гравців на цьому ринку виділяється компанія 4 Gates Group, яка зарекомендувала себе як один з провідних постачальників круїзних послуг.

4 Gates Group є відомою круїзною компанією, яка пропонує широкий спектр послуг, включаючи організацію круїзів, продаж квитків, бронювання додаткових послуг та надання консультацій. Компанія має розвинену інфраструктуру та використовує сучасні технології для обслуговування клієнтів.

Даного конкурента було проаналізовано за наступними критеріями:

— Зручність користування

— Функціональність

— Безпека

— Продуктивність

— Підтримка користувачів

— Інноваційність

— Маркетинг та промоції

— Відгуки та рейтинг

Рисунок 1.1.1 – Логотип компанії 4 Gates Group

Веб-додаток відзначається простим та легко зрозумілим інтерфейсом, а його візуальний дизайн є ергономічним та не надто навантаженим з логічно розташованими елементами. Проте кольорова гамма не є гармонічною, що значною мірою впливає на враження клієнта. Сторінки додатку завантажуються швидко, без помітних затримок. Веб-додаток має оптимізацію під мобільні пристрої, що в рухомому режимі забезпечує зручність використання [2].

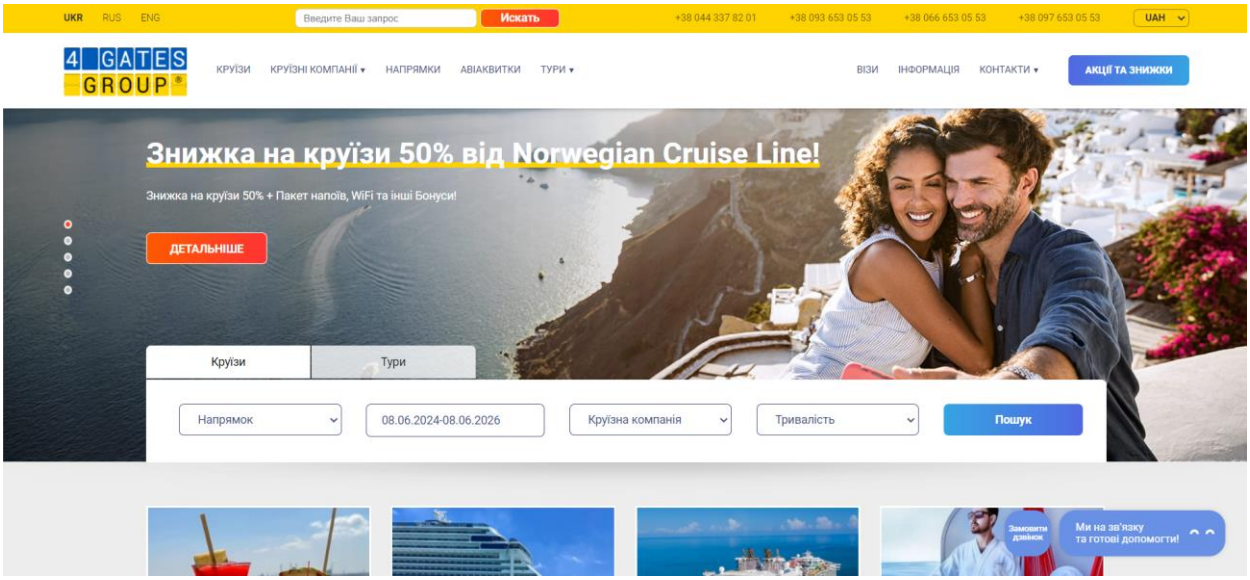


Рисунок 1.1.2 – Інтерфейс веб-додатку компанії 4 Gates Group

Дана круїзна компанія пропонує ефективний та зручний механізм пошуку та бронювання круїзів за різними параметрами. Компанія пропонує численні маршрути по всьому світу, охоплюючи популярні туристичні напрямки та екзотичні місця. Це забезпечує широкий вибір для клієнтів з різними інтересами та бюджетами. Завдяки цьому, клієнти можуть знайти круїз, що відповідає їхнім

очікуванням та потребам. Користувачі можуть легко переглядати, змінювати або скасовувати бронювання через даний додаток. Також наявна детальна інформація про доступні маршрути, судна та послуги на борту. Бронювання круїзів не є єдиною можливістю 4 Gates Group: присутня інтеграція з іншими послугами, такими як екскурсії та трансфери [2].

Для безпеки даних своїх користувачів компанія використовує шифрування для захисту персональних даних та забезпечення безпеки транзакцій. Додаток містить надійні механізми аутентифікації користувачів та контролю доступу до особистих даних. Клієнти компанії можуть відчувати себе захищеними, адже додаток відповідає сучасним стандартам безпеки. Додаток підтримує різноманітні платіжні системи, що забезпечує зручність та безпеку фінансових операцій [2].

При аналізі роботи додатку не помічено збоїв та помилок у роботі, що демонструє стабільність системи. Також додаток виявляє себе стабільно при зростанні обсягів користувацької активності, проте при аналізі сайт мав вищий індекс швидкості ніж рекомендовано.

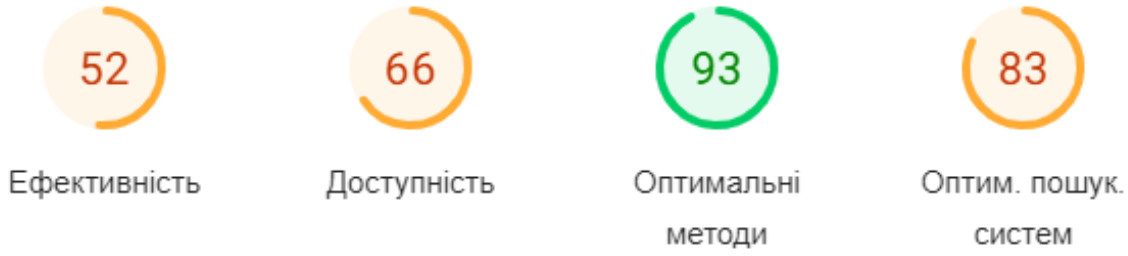


Рисунок 1.1.3 – Результат перевірки веб-додатку компанії 4 Gates Group

4 Gates Group пропонує користувачу різні канали підтримки, такі як телефонна лінія, електронна пошта та онлайн-чат, щоб надати можливість вирішити проблеми та надати відповіді на усі питання у найбільш зручний спосіб. Також веб-додаток містить докладну документацію, FAQ-розділ та відео-уроки, що допомагає вирішенню питань без додаткової допомоги [2].

Компанія активно використовує сучасні для покращення користувацького досвіду та надання інноваційних послуг, таких як штучний інтелект для персоналізованих рекомендацій та машинне навчання для аналізу поведінки користувачів. Додаток має інтерактивні функції, які забезпечують зручність використання та сприяють взаємодії користувачів [2].

4 Gates Group регулярно пропонує різноманітні рекламні акції та знижки для залучення нових клієнтів та збереження старих. Компанія має програми лояльності, які надають знижки та бонуси постійним клієнтам.

4 Gates Group має систему оцінювання, яка дозволяє користувачам залишати відгуки та оцінювати свої круїзні подорожі. Компанія активно аналізує відгуки користувачів для постійного покращення якості послуг [2].

Для чіткого розуміння далі побудовано таблицю з оцінками згідно критеріїв, що були описані на початку даного підрозділу у десятибальній системі:

Таблиця 1.1 – Результат оцінювання за вказаними критеріями

Критерій	Результат за десятибальною шкалою
Зручність користування	9
Функціональність	9
Безпека	8
Продуктивність	6
Підтримка користувачів	9
Інноваційність	8
Маркетинг та промоції	7
Відгуки та рейтинг	9

1.2 Обґрунтування вибору напрямку дослідження

Зараз люди все частіше шукають інформацію виключно у інтернет-джерелах. Це також стосується пошуку зручного відпочинку, в тому числі вибір комфортного круїзу. Тож для круїзної компанії веб-застосунок буде корисним інструментом. Веб-додатки доступні з будь-якого пристрою з підключенням до Інтернету, такого як комп'ютери, смартфони, планшети тощо. Це робить їх дуже зручними для використання в будь-якому місці і в будь-який час. Саме це дозволяє залучити максимальну кількість різноманітних клієнтів.

По при це веб-додатки можуть автоматизувати багато рутинних процесів, що дозволяє підприємствам та користувачам ефективно використовувати свій час і ресурси. Ці можливості будуть неодмінно корисними навіть для роботи центрів обслуговування клієнтів компанією. Також завдяки розвитку технологій, веб-додатки можуть виконувати різноманітні завдання, які дозволятимуть чітко бачити загальну картину результатів роботи бізнесу та зручний для побудови статистик. Даний інструмент також може збирати дані про користувачів та їхню взаємодію з додатком, що дозволяє підприємствам аналізувати ці дані для покращення продукту та підтримки користувачів.

Пріоритетом вибору саме веб-додатку дозволяє підтримувати його в актуальному стані та постійно розвивати, оскільки це уникає необхідності. завантаження та встановлення нових версій. Крім того, вони можуть бути сумісні з різними операційними системами та пристроями.

З розвитком соціальних мереж та розширенням співпраці країн та держав у світі туризм активно набирає популярності. Не зважаючи на різкий спад у 2020 році через пандемію COVID-19, за прогнозами та результатами статистик у 2024 році показник туризму досягне того ж рівня, що і у 2019 році і ймовірно його перевищить [3].

Таблиця 1.2 – Світова туристична статистика

Рік	Прибуття міжнародних туристів	Внесок у ВВП	Щорічне зростання кількості туристів	Щорічне зростання доходу (доларів США)
2019 рік	1,5 (мільярда)	10,4%	4% порівняно з минулим роком	9,3 мільярда
2020 рік	381 (мільйон)	1,7%	-74,6%	2,3 мільярда
2021 рік	415 (мільйонів)	2,15%	8,92%	2,5 мільярда
2022 рік	965 (мільйонів)	7,6%	132,53%	5,9 мільярдів
2023 рік	975 (мільйонів)	9,1%	35,23%	8,8 мільярда
2024 рік	Очікується 1,53 (мільярда)	10,6%	17,24%	9,4 мільярда

Круїзний туризм також стає все більш популярним серед подорожуючих. Це може пов'язано зі зростанням інтересу до альтернативних видів відпочинку та ширшим вибором круїзів та маршрутів. Вони також пропонують комфорт та розкішне відпочинок, об'єднаний з можливістю відвідати різноманітні місця у вигляді портів призначення. Крім того, круїзні лайнери часто мають широкий спектр розваг та активностей для пасажирів, що робить їх привабливими для різних категорій людей.

Даний вид туризму є важливою галуззю економіки для багатьох країн, особливо для тих, які розташовані біля морських або річкових маршрутів. Круїзні лайнери забезпечують робочі місця для тисяч людей, включаючи екіпаж та персонал, а також сприяють розвитку місцевої інфраструктури у портах призначення.

Також круїзні подорожі мають потенціал для розвитку сталого туризму, зокрема шляхом впровадження екологічно чистих технологій у круїзні судна та підтримки екологічних ініціатив у портах призначення.

Сумарно всі ці фактори роблять дану предметну область актуальною для розгляду.

Для написання веб-додатку обрано мову Java з використанням Spring Framework. Такий вибір ґрунтується на тому, що Java є однією з найпопулярніших мов програмування у світі, що підтверджується її використанням у різних галузях, таких як фінанси, телекомунікації, банківська справа, а також у розробці корпоративного програмного забезпечення. Популярність Java забезпечує широку спільноту розробників, велику кількість бібліотек та фреймворків, а також багато навчальних ресурсів [4].

Java має високий рівень продуктивності завдяки сучасним оптимізуючим компіляторам та вдосконаленій віртуальній машині. Вона добре підходить для розробки масштабованих корпоративних додатків, що можуть обробляти великі обсяги даних і забезпечувати високу продуктивність. Завдяки підтримці багатопотоковості, Java дозволяє ефективно використовувати ресурси сучасних багатоядерних процесорів [4].

Java підтримує об'єктно-орієнтований підхід, що сприяє розробці модульного та розширюваного програмного забезпечення. Це дозволяє створювати великі додатки, розбиваючи їх на дрібніші модулі, які можуть бути легко обслуговувані, тестовані та повторно використовувані. Крім того, сучасні версії Java (починаючи з Java 9) підтримують модульну систему, що дозволяє ще більше підвищити розширюваність і організованість коду [4].

Spring Framework є одним з найпоширеніших інструментів для розробки веб-додатків на Java. Він надає гнучку архітектуру та широкий набір потужних інструментів і бібліотек, які значно полегшують розробку складних корпоративних додатків, що полегшують розробку та підтримку програмного забезпечення. Spring включає модулі для різних аспектів розробки, такі як управління залежностями (Spring Core), веб-розробка (Spring MVC), обробка даних (Spring Data), безпека (Spring Security) та багато інших. Використання Spring дозволяє створювати масштабовані, гнучкі та надійні додатки. Spring Framework підтримує сучасні архітектурні підходи, такі як мікросервіси, RESTful веб-сервіси та реактивне програмування [5].

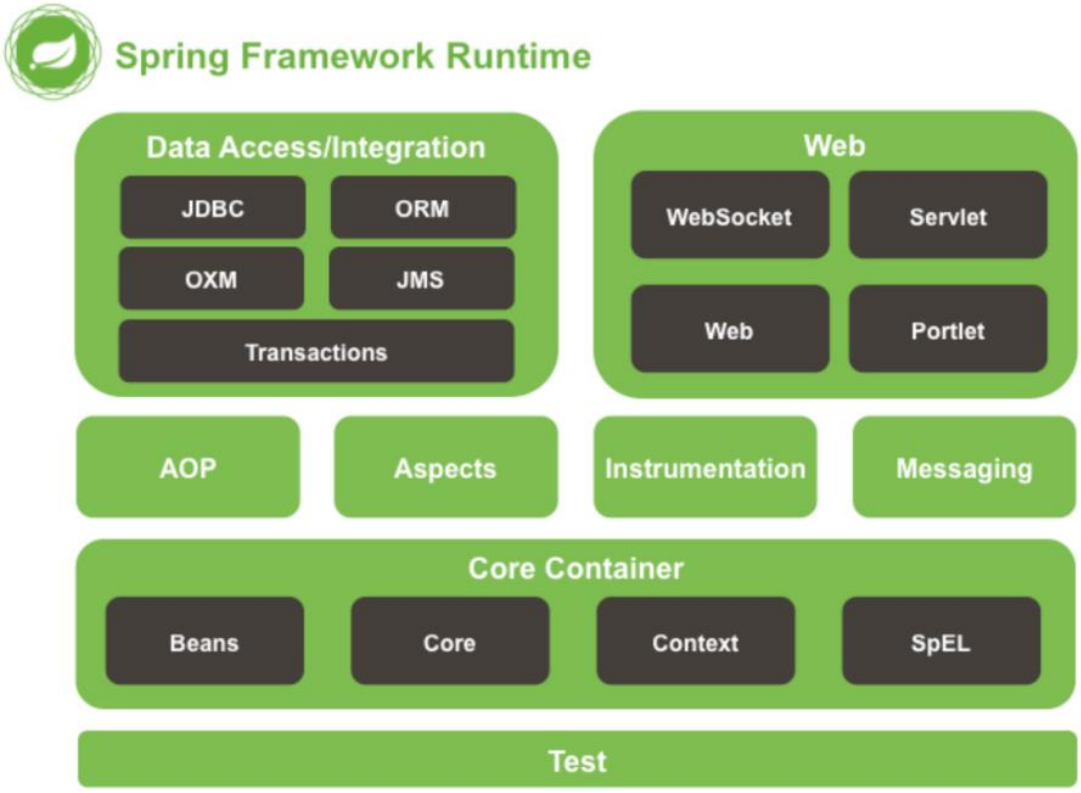


Рисунок 1.2.1 – Архітектура Spring Framework

Фреймворк для ORM в Java Hibernate є одним з найпопулярніших і найпотужніших фреймворків для об'єктно-реляційного відображення (ORM) в Java. Він дозволяє розробникам працювати з реляційними базами даних, використовуючи об'єктно-орієнтований підхід. Це спрощує процес збереження, отримання, оновлення та видалення даних з бази, значно скорочуючи обсяг написання SQL-коду. Hibernate автоматично відображає класи Java на таблиці бази даних. Hibernate підтримує різні рівні кешування, що значно підвищує продуктивність додатків. Hibernate може автоматично управляти зв'язками між об'єктами, такими як один-до-одного, один-до-багатьох, багато-до-одного та багато-до-багатьох. Hibernate підтримує різні реляційні бази даних, такі як MySQL, PostgreSQL, Oracle, SQL Server та інші [6].

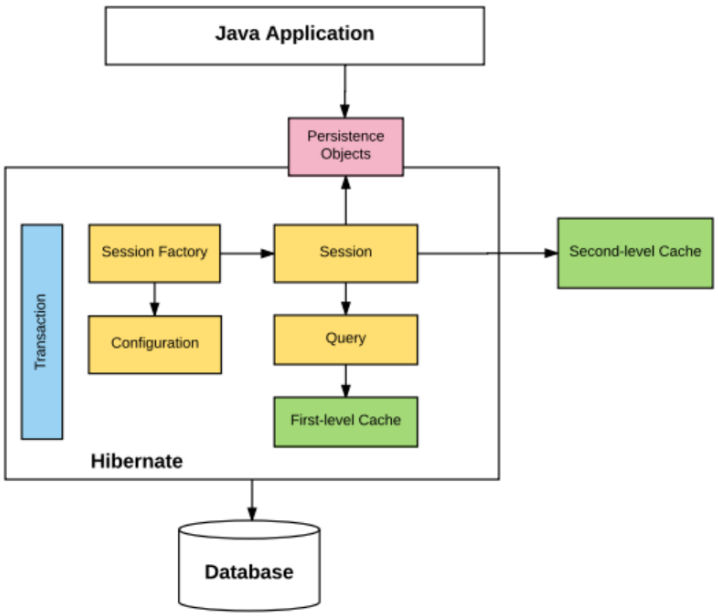


Рисунок 1.2.2 – Схема роботи Hibernate

Також в проекті було використано Maven, який є інструментом управління проєктами та системою автоматизації збірки для Java-проєктів, розробленим Apache Software Foundation. Використовуючи концепцію Project Object Model (POM), Maven дозволяє описувати структуру проєкту, його залежності, плагіни та цілі збірки в XML-форматі. Основні функції Maven включають автоматичне завантаження та управління бібліотеками, розширення функціональності за допомогою плагінів, підтримку багатомодульних проєктів і профілів збірки для різних середовищ. Maven широко підтримується спільнотою та інтегрується з багатьма іншими інструментами розробки, забезпечуючи стандартизацію процесу збірки та управління залежностями [7].

Для збереження даних було обрано PostgreSQL. Це потужна система керування реляційними базами даних з відкритим вихідним кодом. PostgreSQL підтримує широкий спектр функцій, включаючи транзакції, які гарантують атомарність, цілісність, ізоляцію та довговічність (ACID). База даних підтримує складні запити SQL, що дозволяє виконувати аналітичні запити, агрегації та інші складні операції з даними. Має вбудовану підтримку для багатьох типів даних, таких як числові, текстові, географічні та інші спеціалізовані типи [8].

PostgreSQL забезпечує повну підтримку операцій з транзакціями, включаючи багатоверсійне управління одночасним доступом (MVCC), що дозволяє виконувати одночасні операції без блокування читачів і записувачів. PostgreSQL підтримує розширення, що дозволяє додавати нові типи даних, індекси, функції та інші об'єкти бази даних [8].

Дана система має потужну систему розмежування прав доступу, дозволяючи точно налаштовувати права доступу для користувачів та ролей. PostgreSQL підтримує реплікацію даних для забезпечення високої доступності та відмовостійкості. Також присутні можливості горизонтального масштабування, що дозволяє обробляти великі обсяги даних і підвищувати продуктивність [8].

Рисунок 1.2.3 – Логотип бази даних PostgreSQL

Як середовище розробки для програмування на мові Java було обрано програмне забезпечення від JetBrains – Intelij IDEA. Даний продукт надає широкий набір інструментів для розробки програмного забезпечення на Java, включаючи підтримку автоматичного доповнення коду, рефакторингу, аналізу коду, налагодження та інші функції, що полегшують розробку. Intelij IDEA має інтегровану систему керування версіями, що дозволяє розробникам ефективно працювати з репозиторіями, такими як Git, SVN та Mercurial. Середовище підтримує різні фреймворки і технології, такі як Spring, Hibernate, Maven, Gradle та інші, що робить його відмінним інструментом для розробки різноманітних проектів на Java. Крім того, IntelliJ IDEA має дружній інтерфейс користувача, що дозволяє швидко освоїти його та ефективно працювати над проектами. Вона також має велику кількість плагінів, що розширюють її функціональність та дозволяють

налаштувати робочий процес під потреби розробника. IntelliJ IDEA підтримує різні версії мови Java, включаючи Java 8, Java 11 та більш нові версії, що дозволяє розробникам використовувати останні функції мови та бібліотеки [9].

Рисунок 1.2.4 – Особливості середовища розробки Intelij IDEA

Розробка розмітки і стилів для веб-додатку круїзної компанії на HTML і CSS є важливою складовою, яка визначає зовнішній вигляд та взаємодію користувача з системою. Основною метою є створення привабливого, зручного та інтуїтивно зрозумілого інтерфейсу, що забезпечує позитивний користувацький досвід.

Для додатку зручно використовувати шаблони HTML з адаптивним змістом. Для реалізації обрано шаблонний двигун Thymeleaf для розробки веб-додатків на Java, який дозволяє легко і ефективно вбудовувати динамічний контент у HTML-сторінки. Він працює на стороні сервера, генеруючи HTML-код на основі даних, що надходять з сервера або з інших джерел, і відправляє цей HTML-код клієнтам.

Thymeleaf забезпечує простий та ефективний спосіб вбудовування динамічного контенту у HTML-сторінки, що робить його популярним вибором для розробки веб-додатків на Java.

Завдяки цьому у додатку додана мовна локалізація для забезпечення того, щоб контент відображався мовою користувача, забезпечуючи краще розуміння та зручність взаємодії. Такий підхід є ефективним способом забезпечення коректного відображення контенту для різних мовних аудиторій без необхідності змінювати сам код сторінки.

Також для зручності на сторінках з великою кількістю елементів додано автоматичну пагінацію. Щоб веб-додаток відображався в одному стилі додано футери до сторінок.

CSS використовується для стилізації HTML-елементів, надаючи їм вигляду та поведінки. CSS дозволяє задавати кольори, шрифти, відступи, вирівнювання, анімації та інші стилі, щоб зробити веб-сторінку привабливою та зручною для користувача.

1.3 Технічний аспект проблеми

Для розробки даного веб-додатку обрано клієнт-серверну архітектуру. Ця модель дозволяє розділити обчислювальні завдання між декількома машинами та забезпечує масштабованість, ефективність і гнучкість. Дана модель працює таким чином: клієнт надсилає запити на сервер, сервер їх обробляє та повертає відповідь. Для даного додатку обрано двохрівневу архітекуру: клієнт та сервер [10].

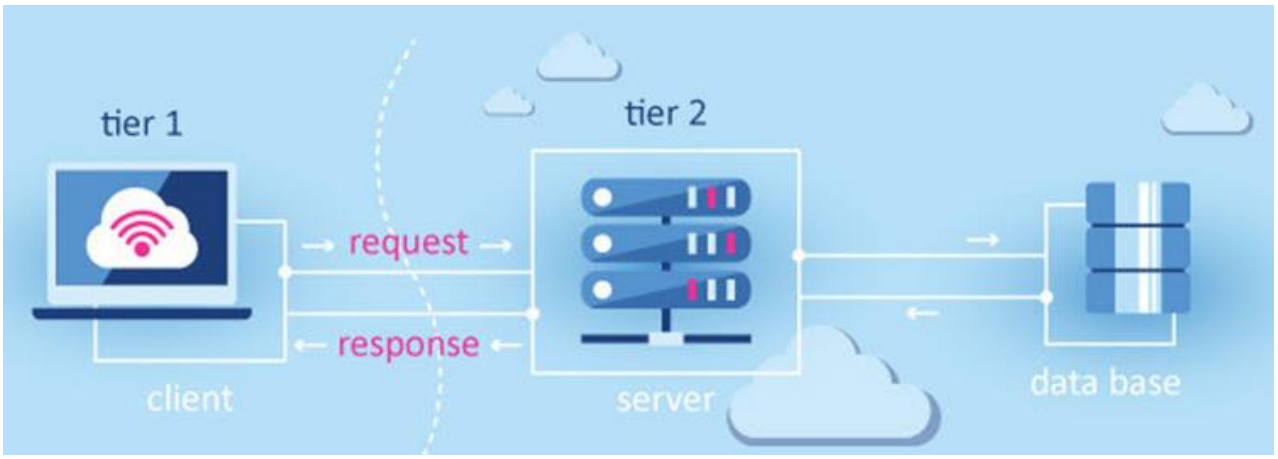


Рисунок 1.3.1 – Двохрівнева клієнт-серверна архітектура

Клієнтська частина відповідає за взаємодію з користувачами. Вона надсилає запити до сервера та відображає отримані дані. Клієнт може бути реалізований у вигляді веб-додатку, мобільного додатку або десктопного додатку. Дана робота передбачає саме веб-додаток.

Веб-клієнт є однією з найпоширеніших форм клієнтської частини і працює в браузері користувача. Зазвичай він включає HTML, CSS та JavaScript для відображення інтерфейсу користувача та взаємодії з сервером. У контексті розробки на Java з використанням Spring Framework, веб-клієнт реалізується для обробки запитів, отримання даних з сервера, відображення інформації та забезпечення інтерактивного користувацького досвіду. Використання Spring Framework дозволяє спростити реалізацію веб-клієнта завдяки вбудованим засобам для обробки HTTP-запитів та управління сесіями.

Spring Boot є зручним інструментом для швидкого створення готового до використання веб-додатка. Spring MVC забезпечує структуру для обробки HTTP-запитів та відправки відповідей. Створення контролерів дозволяє визначити маршрути для обробки запитів від клієнтів. Для взаємодії з бекендом, веб-клієнт використовує HTTP-запити RESTful API, відправляючи дані у форматі JSON та отримуючи відповіді від сервера. Наприклад, при завантаженні списку круїзів, веб-клієнт надсилає GET-запит до відповідного кінцевого пункту API і отримує JSON-дані, які потім відображаються у вигляді таблиці або карток на веб-сторінці.

Важливим аспектом є обробка станів та подій на стороні клієнта. Наприклад, при натисканні кнопки бронювання, веб-клієнт може відправити POST-запит з даними про користувача та обраний круїз, а потім обробити відповідь від сервера, відобразивши повідомлення про успішну бронь або помилку.

З точки зору безпеки, веб-клієнт повинен забезпечувати захист даних користувачів. Використання HTTPS забезпечує шифрування даних між клієнтом і сервером. Крім того, для захисту від CSRF-атак (Cross-Site Request Forgery), Spring Security пропонує вбудовані механізми, такі як токени CSRF, які додаються до кожного запиту, що змінює стан на сервері.

Серверна частина веб-додатку на Java з використанням Spring Framework є центральною складовою, що забезпечує обробку логіки додатку, управління базою даних, безпеку та взаємодію з клієнтською частиною через API. В основі серверної архітектури лежить Spring Boot, який спрощує налаштування та розгортання додатків завдяки вбудованим конфігураціям і широкій екосистемі підтримуваних компонентів.

Серверний додаток на Spring Boot включає ряд основних компонентів, таких як контролери, сервіси, репозиторії та моделі. Контролери обробляють HTTP-запити від клієнта, визначають кінцеві точки API та повертають відповіді у форматі JSON. Наприклад, контролер може обробляти запити на отримання списку круїзів, деталей конкретного круїзу або створення нового бронювання.

Сервіси реалізують бізнес-логіку додатку, забезпечуючи виконання операцій, таких як розрахунок вартості круїзу, перевірка доступності кают або управління даними користувачів. Вони виступають посередниками між контролерами та репозиторіями, інкапсулюючи логіку, яка не повинна бути безпосередньо в контролерах.

Spring Security забезпечує безпеку додатку, реалізуючи механізми аутентифікації та авторизації. Це включає управління користувачами, ролями та правами доступу до різних частин додатку. Наприклад, тільки авторизовані користувачі можуть створювати бронювання або переглядати особисту

інформацію. Spring Security також підтримує різні способи аутентифікації, включаючи базову, формову та OAuth2.

Для взаємодії з базою даних використовується PostgreSQL, яка є потужною та надійною реляційною базою даних. Spring Boot дозволяє легко інтегруватися з PostgreSQL, забезпечуючи автоматичне створення схем бази даних та управління транзакціями. Це робить процес зберігання та обробки даних більш ефективним та безпечним.

При побудові веб-додатку важливо передбачити логування щоб відстежувати та фіксувати події, помилки і іншу важливу інформацію під час виконання програми. Для цього було використано бібліотеку Log4j 2. Дана бібліотека передбачає декілька рівнів логування, кожен з яких має своє призначення і дозволяє фільтрувати, які повідомлення будуть записані. Апендери в Log4j 2 відповідають за те, куди будуть виводитися логи. Логери визначатимуться для різних компонентів програми та відповідатимуть за логування в цих компонентах [11].

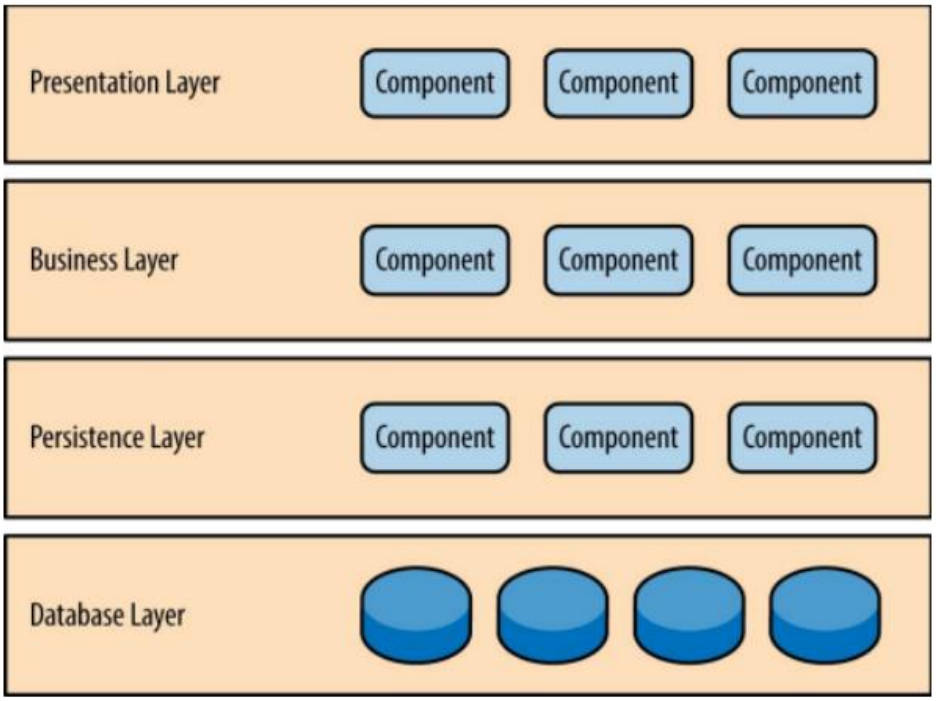


Рисунок 1.3.2 – Схема багатошарової архітектури

Архітектура Spring Framework дозволяє реалізувати багатошаровість, що є досить популярним рішенням для веб-додатків. Багатошаровість у Spring допомагає розділити відповідальності між компонентами додатку, що сприяє полегшенню розробки, тестування та підтримки програмного забезпечення. Кожен шар може бути розглянутий як окремий модуль, що дозволяє використовувати принципи інкапсуляції та модульності для підтримки чистого та організованого коду [12].

В шарі представлення розміщаються компоненти, які стосуються інтерфейсу користувача. У веб-додатках це можуть бути контролери, які обробляють HTTP-запити, та представлення, які відображають дані для користувача. Spring MVC (Model-View-Controller) є ключовим компонентом Spring для реалізації цього шару.

Шар бізнес-логіки визначає, які операції та обробки повинні виконуватися з даними додатку. Він відокремлюється від представлення та забезпечується сервісами або компонентами, які обробляють логіку додатку. Spring дозволяє використовувати компоненти, такі як сервіси, для цієї мети.

У шарі доступу до даних реалізовані компоненти, які взаємодіють з базою даних або з іншими джерелами даних. Spring надає можливості для реалізації доступу до даних через JPA (Java Persistence API), JDBC (Java Database Connectivity) або ORM (Object-Relational Mapping) інструменти, такі як Hibernate який розглядається в даній роботі.

Шар інфраструктури забезпечує виконання низькорівневих завдань, таких як керування транзакціями, безпека, логування та кешування. Spring надає різні модулі для цих потреб, такі як Spring Transaction, Spring Security, Spring AOP (Aspect-Oriented Programming) та інші.

2 РОЗРОБКА МОДЕЛІ ТА ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Проєктування

Проєктування програмного забезпечення — це критичний етап у процесі розробки, який полягає у визначенні структури, компонентів та їхньої взаємодії. Воно включає розробку архітектури системи, що забезпечує ефективне, масштабоване та зручне в обслуговуванні рішення. Під час проєктування визначаються основні елементи системи, їх функції та способи взаємодії між ними. Успішне проєктування враховує вимоги користувачів, технічні обмеження та найкращі практики індустрії. Це дозволяє створити основу, яка забезпечить надійність, продуктивність та зручність у використанні програмного забезпечення.

2.1.1. Розробка моделі предметної області

Розробка моделі предметної області є важливою складовою етапу проектування програмного забезпечення. Цей процес включає аналіз і моделювання реальних об'єктів та їхніх взаємозв'язків, які мають бути відображені в системі. Модель предметної області дозволяє мати єдине уявлення про структуру даних та логіку бізнес-процесів.

Перший крок у розробці моделі предметної області — це збір вимог, де визначаються ключові елементи і функції системи. Далі проводиться аналіз цих вимог для ідентифікації основних об'єктів, атрибутів та їхніх взаємозв'язків. Цей аналіз може включати діаграми класів, які наочно показують структуру об'єктів і зв'язки між ними.

Після аналізу вимог розробляється концептуальна модель, яка є абстрактним поданням предметної області і не залежить від конкретних технічних реалізацій.

На основі цієї моделі створюється логічна модель, яка деталізує структуру даних та зв'язки між об'єктами, враховуючи можливі обмеження та правила бізнесу.

Заключним етапом є розробка фізичної моделі, яка відображає логічну модель у конкретній реалізації, враховуючи обрану технологію баз даних та інші технічні аспекти. Це включає визначення таблиць, колонок, ключів та індексів, які забезпечують ефективне зберігання і доступ до даних.

Розробка моделі предметної області дозволяє створити чітку і структуровану основу для подальшої реалізації програмного забезпечення, забезпечуючи зрозумілу і логічну структуру системи, що відповідає вимогам користувачів та бізнесу.

Для кращого розуміння функціонування програми побудовано діаграми послідовностей, що зображена на рисунку 2.1.1. Побудова діаграми послідовностей починається з ідентифікації основних акторів, таких як користувачі та система, і їх взаємодії на кожному етапі. Потім визначаються важливі події та операції, що відбуваються між акторами на кожному кроці. Діаграма послідовностей відображає ці кроки у логічній послідовності, допомагаючи розуміти взаємодію між користувачем і системою на кожному етапі.

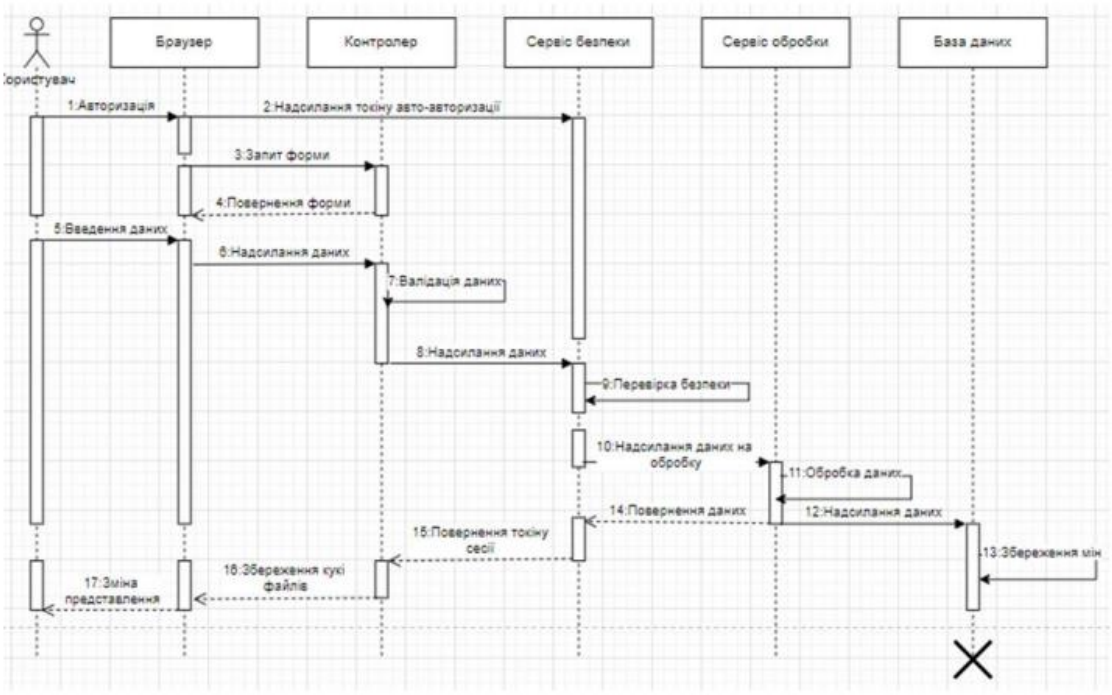


Рисунок 2.1.1 – Діаграма послідовності у програмі

Діаграма послідовностей відображає процес взаємодії між різними компонентами системи, такими як користувач, браузер, контролер, сервіси та база даних. Ці дані відображені у діаграмі вертикальним лініями. Горизонтальні стрілки між цими лініями показують повідомлення або взаємодії, які відбуваються між об’єктами. Ці повідомлення можуть включати запити, відповіді, виклики методів та інші види комунікації. Кожна взаємодія на діаграмі має свій номер, що вказує на послідовність кроків у процесі.

Аналізуючи діаграму з рисунка 2.1.1, можна побачити, що при вході користувача на платформу, якщо його дані доступні, вони передаються на сервер авторизації для перевірки. В іншому випадку, система пропонує користувачу форму авторизації, яку створює контролер. Після введення користувачем даних, вони обробляються та перевіряються з базою даних. Сервіси на діаграмі включають сервіс безпеки на етапах виконання запитів, який перевіряє дані на валідність, та сервіс обробки, який виконує необхідні операції, такі як оновлення бази даних. Після обробки запиту сервіси повертають відповіді до контролера, який, у свою чергу, повертає результат користувачеві через браузер. У кінцевому результаті, учасник процесу отримує доступ до своїх особистих даних та можливість провадити подальші дії.

Ця діаграма важлива для розуміння послідовності кроків у процесах системи та для виявлення потенційних точок збою або неефективності. Вона також допомагає визначити, як оптимізувати взаємодію між компонентами для покращення продуктивності та безпеки.

2.1.2. Розробка бізнес моделі

Для візуалізації моделі предметної області часто використовуються діаграми класів UML (Unified Modeling Language), які дозволяють чітко представити сутності, їх атрибути та зв'язки між ними. Це полегшує розуміння та обговорення

моделі серед команди розробників і з замовником, а також служить основою для подальшої реалізації програмного забезпечення.

Для проектованого веб-додатку основними варіантами використання є бронювання круїзів, управління бронюваннями та перегляд інформації. Діаграму варіантів використання продемонстровано на рисунку 2.1.2

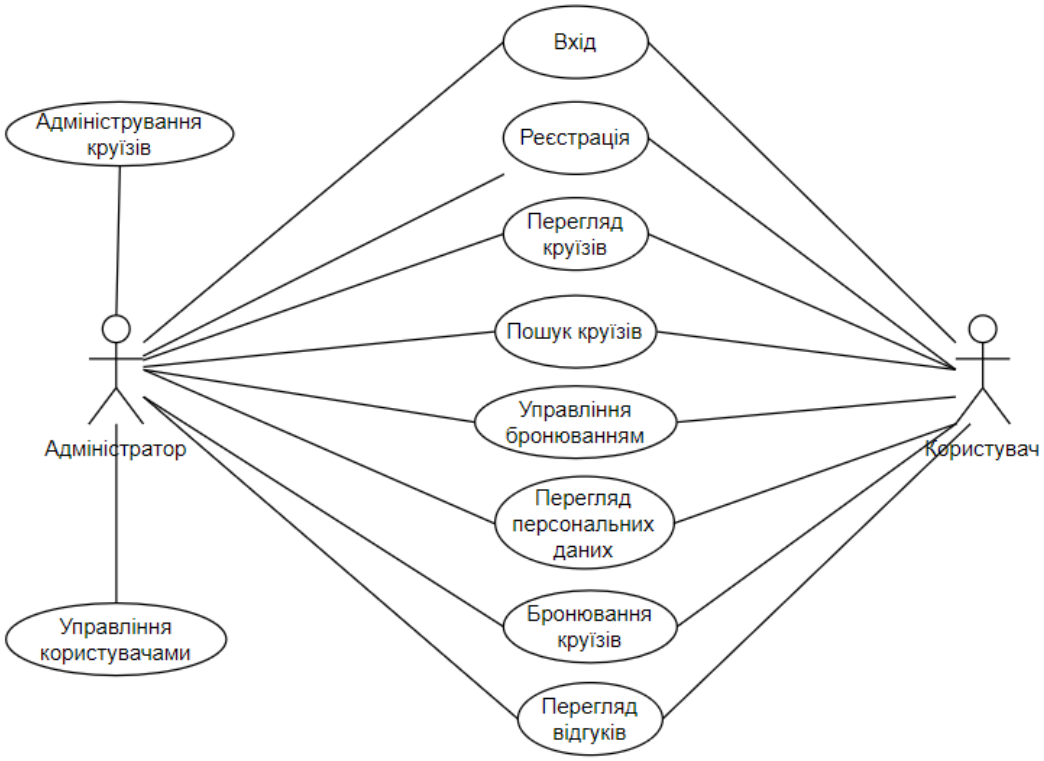


Рисунок 2.1.2 – Діаграма варіантів використання

Діаграма Use Case для веб-застосунку круїзної компанії демонструє різні сценарії взаємодії між користувачами системи та самою системою. На діаграмі показано двох основних акторів: Адміністратор і Користувач.

Користувач може ввійти в систему, використовуючи свої облікові дані, а якщо обліковий запис користувача недоступний або раніше не створювався він може виконати реєстрацію. Користувач може переглядати список доступних круїзів, включаючи їх деталі, такі як маршрути, дати, тривалість і ціни. Також користувач може здійснювати пошук круїзів за різними критеріями, такими як дати, тривалість, напрямки тощо. Користувач може бронювати круїзи, обираючи відповідні варіанти та заповнюючи необхідну інформацію. Користувач може

переглядати і змінювати свої бронювання, включаючи скасування або оновлення деталей. Користувач може переглядати та оновлювати свою персональну інформацію, таку як контактні дані та паролі. Користувач може переглядати відгуки інших користувачів про круїзи і залишати власні відгуки.

Адміністратор має розширені можливості порівняно з користувачем. Крім доступу до всіх функцій користувача, адміністратор може виконувати адміністрування круїзів та управління користувачами. Адміністратор може додавати нові круїзи, оновлювати існуючі круїзи або видаляти їх з системи. Це включає управління деталями круїзів, такими як маршрути, лайнери, дати і ціни. Адміністратор може керувати обліковими записами користувачів, включаючи створення нових користувачів, оновлення їхньої інформації та видалення облікових записів, якщо це необхідно.

На діаграмі також показано, що всі функції адміністрування круїзів і управління користувачами доступні тільки адміністратору, тоді як функції входу, реєстрації, перегляду круїзів, пошуку круїзів, управління бронюванням, перегляду персональних даних, бронювання круїзів і перегляду відгуків спільні для обох акторів, але адміністратор має додаткові можливості для управління системою.

Ця діаграма Use Case забезпечує ясну візуалізацію того, як різні ролі взаємодіють з системою, і які основні функції доступні для кожного типу користувача. Вона допомагає в розумінні функціональних вимог до веб-застосунку круїзної компанії та полегшує процес проектування та розробки системи, визначаючи ключові області взаємодії та можливості користувачів.

2.1.3. Проєктування архітектури

У розділі проєктування архітектури програмного забезпечення розглядаються концепції та методи, що лежать в основі створення структури системи. На цьому етапі визначаються класи системи, їх взаємозв'язки та інші

аспекти, що визначають її структуру. Одним з ключових інструментів при проєктуванні є діаграма класів, яка надає візуальне представлення структури системи та її компонентів.

Для візуального моделювання веб-додатку використано UML-діаграми. UML-діаграми використовуються для відображення різних аспектів системи, включаючи її структуру, функціональність, динаміку та інші характеристики. Вони дозволяють візуалізувати складні системи та їх взаємодію, що сприяє кращому розумінню та аналізу програмного забезпечення [13].

Для розуміння структури класів у цьому розділі розглянемо діаграми класів. Даний вид діаграм є однім з найбільш поширених та важливих UML-діаграм для моделювання програмних систем. Діаграма класів відображає структуру системи, представляючи класи, їх атрибути, методи та взаємозв'язки між ними. Діаграма класів допомагає визначити основні компоненти системи та їх взаємодію, спрощує процес проєктування та реалізації програмного забезпечення. Вона є потужним інструментом для аналізу та проєктування об'єктно-орієнтованих систем, а також забезпечує чітку основу для подальшого розвитку та підтримки програмного забезпечення [14].

Класи є основним концептом в об'єктно-орієнтованому програмуванні (ООП), що дозволяє структурувати програмний код для ефективного використання та повторного використання. Кожен клас визначає шаблон для створення об'єктів, які мають спільні характеристики та поведінку. Він включає в себе дані (атрибути або властивості), які представляють стан або характеристики об'єкта, а також методи (функції або процедури), які визначають операції чи дії, які можуть бути виконані над об'єктом.

Класи дозволяють моделювати реальні або абстрактні сутності в програмному коді, що робить їх потужним інструментом для створення складних систем. Вони допомагають організувати код у логічні блоки, що спрощує розробку, тестування та підтримку програмного забезпечення. Кожен клас може мати власний інтерфейс, що визначає доступні методи та їх параметри, а також специфікації, що вказують, які операції він може виконувати [14].

Класи є основними будівельними блоками в ООП, і вони дозволяють розбити складні задачі на менші та керовані одиниці, що полегшує розробку та підтримку програмного забезпечення. Вони є ключовим елементом у створенні ефективних та модульних програм, що можуть бути легко розширені та модифіковані з часом. Використання класів дозволяє розробникам створювати реалістичні та легко зрозумілі моделі систем, що сприяє ефективній комунікації та співпраці між членами команди розробки.

Багатошарова архітектура, обрана для серверної частини проекту, відображає організацію програмного забезпечення у вертикальні шари, кожен з яких відповідає за певний аспект функціональності додатка. Цей підхід є стандартом у багатьох серверних додатках через свою ефективність та зручність у розробці та підтримці.

На верхньому рівні розташовується шар представлення, який відповідає за відображення інформації користувачам та обробку їх взаємодії з системою через інтерфейс користувача. Цей шар включає в себе веб-інтерфейс, графічний інтерфейс користувача або інші методи комунікації з користувачем.

Нижче знаходиться шар бізнес-логіки, який визначає логіку та правила, які керують обробкою даних та виконанням операцій в системі. В цьому шарі реалізовані функціональність та операції, які відповідають за бізнес-процеси та вимоги додатка.

Ще нижче розташовується шар доступу до даних, який забезпечує взаємодію з базою даних та забезпечує доступ до неї з інших шарів. Цей шар містить логіку, яка відповідає за зберігання, витягування та оновлення даних у базі даних.

Найнижчим рівнем є шар бази даних, де зберігаються всі дані, необхідні для роботи системи. Цей шар відповідає за забезпечення надійності, ефективності та безпеки зберігання даних.

Кожен шар має свою власну функціональність та взаємодіє з іншими шарами через визначені інтерфейси. Ця структура дозволяє розділити відповідальності та спростити розробку, тестування та підтримку додатка. Якщо необхідно, можуть додаватися додаткові шари для реалізації конкретних функціональних вимог або для забезпечення більшої масштабованості та гнучкості системи.

У додатку А продемонстровано загальний вигляд даної системи, розподілений по рівнях.

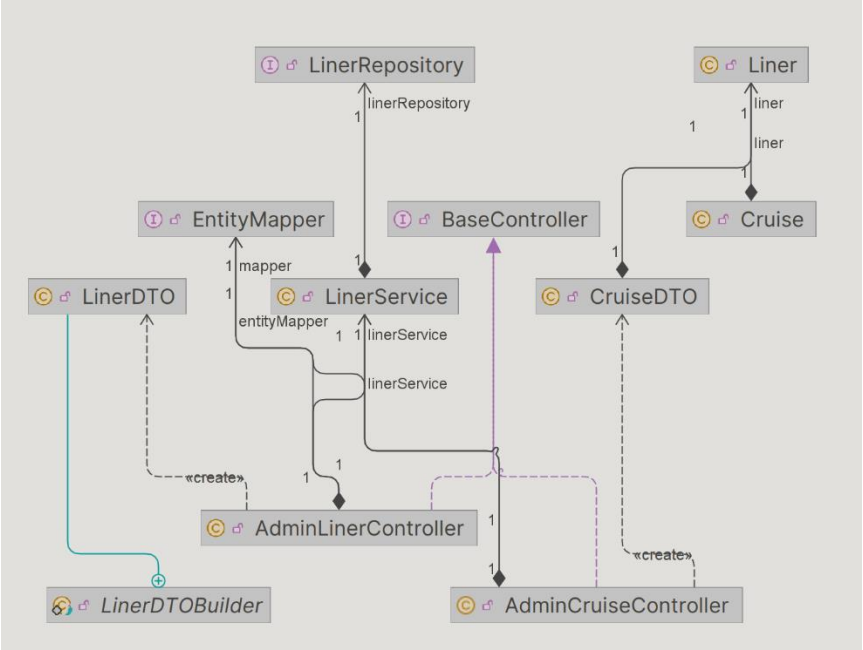


Рисунок 2.1.3 – Діаграма класів сутності Liner

Діаграма класів для сутності Liner зображена на рисунку 2.1.3 включає кілька основних класів і сервісів, які забезпечують управління лайнерами та їх взаємодію з іншими компонентами системи. Центральним елементом є клас Liner, який представляє конкретний лайнер (корабель). Для передачі даних про лайнери використовується клас LinerDTO, який має ті ж атрибути, що і Liner. Клас LinerDTOBuilder використовується для створення об'єктів LinerDTO за допомогою патерну Builder і має методи для встановлення атрибутів та створення об'єкта LinerDTO. Клас LinerService забезпечує управління лайнерами. Репозиторій LinerRepository забезпечує доступ до даних лайнерів. Клас AdminLinerController забезпечує керування лайнерами на рівні контролера. Клас EntityMapper містить методи для перетворення об'єктів різних типів, таких як Liner, Cruise, Route, Ticket, User, у відповідні об'єкти DTO (Data Transfer Object) і навпаки.

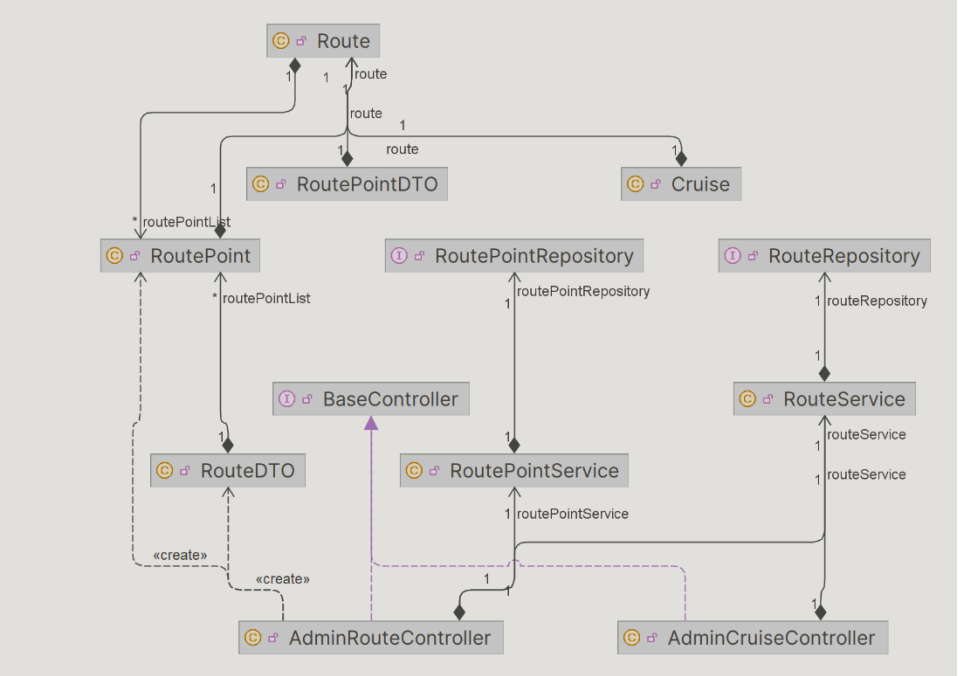


Рисунок 2.1.4 – Діаграма класів сутності Route

Діаграма класів для сутності Route зображена на рисунку 2.1.4 включає кілька основних класів і їх взаємозв'язки, які забезпечують управління маршрутами та точками маршруту. Центральним елементом є клас Route , який представляє конкретний шлях. Для передачі даних існує клас RouteDTO, який має ті ж атрибути, що і Route, плюс атрибут builder, який посилається на об'єкт RouteDTOBuilder. Клас RouteDTOBuilder використовується для створення об'єктів RouteDTO з допомогою паттерну Builder. Клас RoutePoint представляє точки маршруту. Для передачі даних про точки маршруту використовується клас RoutePointDTO. Клас RoutePointDTOBuilder використовується для створення об'єктів RoutePointDTO з допомогою паттерну Builder. Для роботи з маршрутами та точками маршруту існують сервіси RouteService та RoutePointService. Репозиторії RouteRepository та RoutePointRepository забезпечують доступ до даних маршрутів і точок маршруту відповідно. Для управління маршрутами і точками маршруту існує клас AdminRouteController. Допоміжний клас BaseController перевіряє змінені об'єкти в конфігурації. Крім того, існує клас AdminCruiseController, який відповідає за управління круїзами та відповідно отримує інформацію про сутності Route, що приєднані до круїзів.

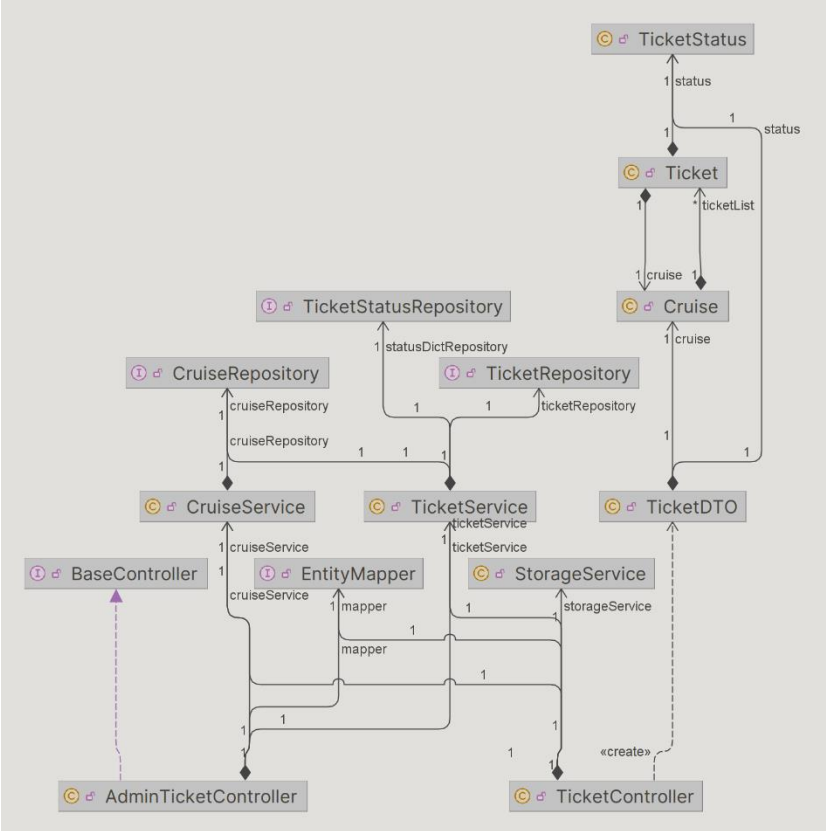


Рисунок 2.1.5 – Діаграма класів сутності Ticket

Діаграма класів для сутності Ticket зображена на рисунку 2.1.5 включає декілька основних класів, які забезпечують управління квитками на круїзи, їх статусами та іншими пов'язаними даними. Центральним елементом є клас Ticket, який представляє квиток. Для передачі даних про квитки використовується клас TicketDTO, який має ті ж атрибути, що і Ticket, а також атрибут builder. Клас TicketDTOBuilder використовується для створення об'єктів TicketDTO з допомогою паттерну Builder. Клас TicketStatus представляє статус квитка. Для роботи з квитками існує сервіс TicketService. Цей сервіс використовує репозиторії TicketRepository та TicketStatusRepository для доступу до даних квитків та їх статусів. Репозиторії TicketRepository та TicketStatusRepository забезпечують доступ до даних квитків та статусів відповідно. Клас TicketController забезпечує керування квитками на рівні контролера.

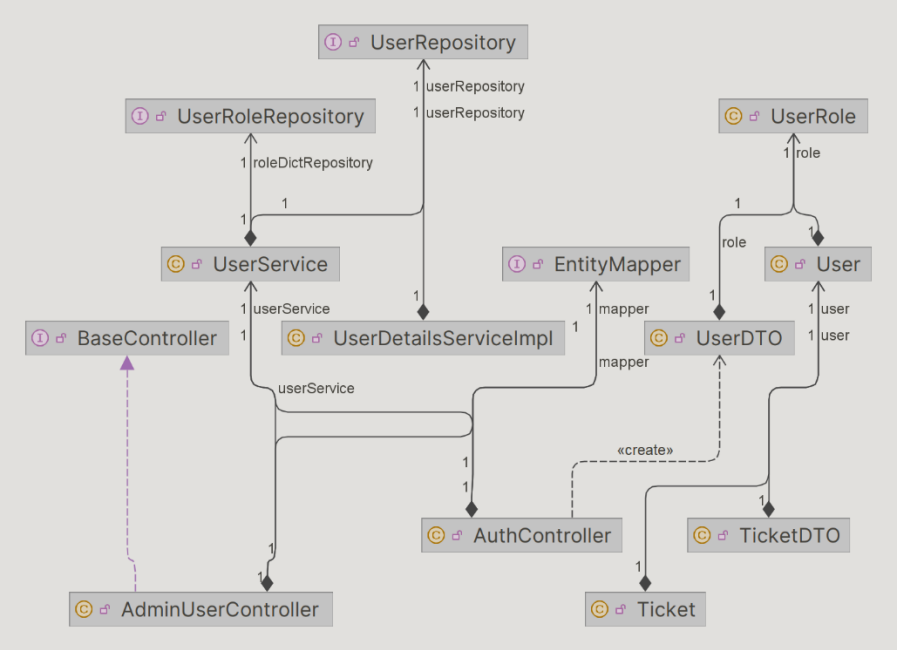


Рисунок 2.1.6 – Діаграма класів сутності User

Діаграма класів для сутності User зображена на рисунку 2.1.6 включає кілька основних класів і їх взаємозв'язки, які забезпечують управління користувачами, їх ролями та квитками. Центральним елементом є клас User, який собою представляє користувача веб-додатку. Для передачі даних про користувачів існує клас UserDTO. Клас UserDTOBuilder використовується для створення об'єктів UserDTO з допомогою паттерну Builder. Клас UserRole представляє ролі користувачів. Для роботи з користувачами існує сервіс UserService. Цей сервіс використовує UserRepository для доступу до даних користувачів та UserRoleRepository для доступу до даних ролей користувачів. Репозиторії UserRepository та UserRoleRepository забезпечують доступ до даних користувачів і ролей користувачів відповідно. Для управління користувачами існує клас AdminUserController. Клас UserDetailsServiceImpl реалізує інтерфейс UserDetailsService та завантажує деталі користувача за його логіном. Цей клас використовує UserRepository для доступу до даних користувачів.

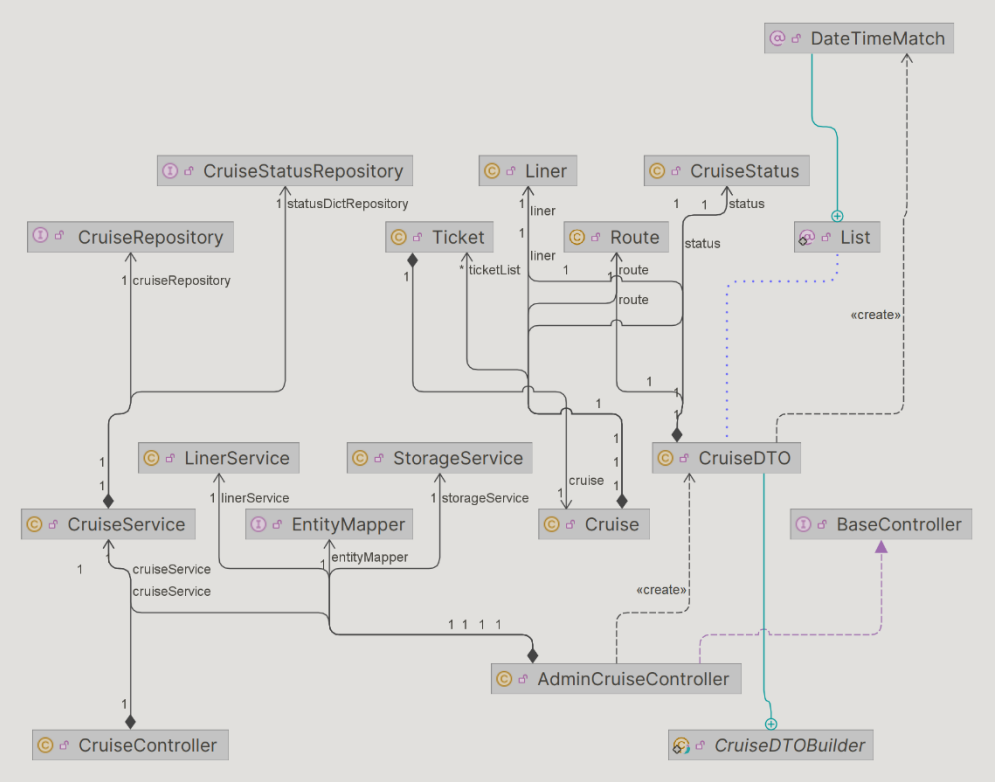


Рисунок 2.1.7 – Діаграма класів сутності Cruise

Діаграма класів для сутності Cruise зображена на рисунку 2.1.7 включає декілька основних класів, які забезпечують управління круїзами, їх статусами, маршрутами, лайнерами та іншими пов'язаними даними. Центральним елементом є клас Cruise, який представляє круїз. Клас CruiseStatus містить інформацію про статус круїзу. Для передачі даних про круїзи використовується клас CruiseDTO. Клас CruiseDTOBuilder забезпечує методи для побудови об'єктів CruiseDTO з допомогою паттерну Builder. CruiseRepository є інтерфейсом, що визначає методи для взаємодії з базою даних. Аналогічно, CruiseStatusRepository визначає методи для роботи зі статусами круїзів. Клас StorageService використовується для роботи з файловою системою, зокрема для збереження та видалення файлів. Клас AdminCruiseController містить контролери для адміністративного управління круїзами. Клас CruiseController містить контролери для загального управління круїзами, включаючи методи для перегляду списку круїзів та окремих круїзів.

Сутності Cruise, Liner, Ticket, Route та User взаємопов'язані, створюючи цілісну систему управління круїзами.

Сутність Cruise представляє конкретний круїз, який містить посилання на сутність Liner. Кожен круїз пов'язаний з певним лайнером (кораблем), на якому він здійснюється. Зв'язок між Cruise і Liner можна описати як "багато до одного", оскільки один лайнер може бути використаний для багатьох круїзів, але кожен конкретний круїз пов'язаний з одним лайнером.

Сутність Cruise також містить посилання на сутність Route. Кожен круїз має визначений маршрут, по якому він буде проходити. Зв'язок між Cruise і Route також є "багато до одного", оскільки один маршрут може бути використаний для багатьох круїзів, але кожен конкретний круїз має лише один маршрут.

Сутність Cruise містить список посилань на всі квитки, що належать до цього круїзу. Зв'язок між Cruise і Ticket можна описати як "один до багатьох", оскільки один круїз може мати багато квитків, але кожен квиток належить до одного конкретного круїзу.

Сутність Ticket містить посилання на користувача, який є його власником. Зв'язок між Ticket і User можна описати як "багато до одного", оскільки один користувач може мати багато квитків, але кожен квиток належить одному користувачу. Сутність User має зв’язок лише з сутністю Ticket.

Безпосереднього прямого зв'язку між Liner і Ticket в схемі немає, зв'язок здійснюється через сутність Cruise. Оскільки квиток пов'язаний з круїзом, а круїз пов'язаний з лайнером, можна сказати, що квиток опосередковано пов'язаний з лайнером.

Аналогічно, як і з Liner і Ticket, безпосереднього прямого зв'язку між Route і Ticket немає, але зв'язок здійснюється через сутність Cruise. Оскільки квиток належить до круїзу, а круїз має маршрут, квиток опосередковано пов'язаний з маршрутом.

2.2 Конструювання

Конструювання програмного забезпечення є ключовим етапом розробки, що включає створення та інтеграцію компонентів системи згідно з проектною документацією. На цьому етапі розробники реалізують функціональність, визначену на стадії проектування, використовуючи програмні мови та інструменти. Конструювання включає написання коду, розробку алгоритмів, тестування та інтеграцію окремих модулів у єдину систему. Важливими аспектами цього процесу є забезпечення якості коду, відповідність стандартам програмування, а також проведення модульного та інтеграційного тестування для виявлення та усунення помилок. Конструювання є критичним для створення надійного, масштабованого та ефективного програмного забезпечення, яке відповідає вимогам користувачів та специфікаціям проекту.

2.2.1. Реалізація ключових класів

Після визначення всіх класів додатка та їх опису за допомогою діаграм, стає зрозуміло, які функції та методи необхідно реалізувати. Для обробки даних класів використовується CRUD-підхід, що включає основні операції над даними, а також принципи RESTful для реалізації запитів. Структура системи розподілена на окремі модулі: конфігураційні класи, контролери, класи для перенесення даних, моделі (сутності), репозиторії, сервіси, допоміжні класи, класи валідації даних та ресурси На рисунку 2.2.1 зображено створені модулі.

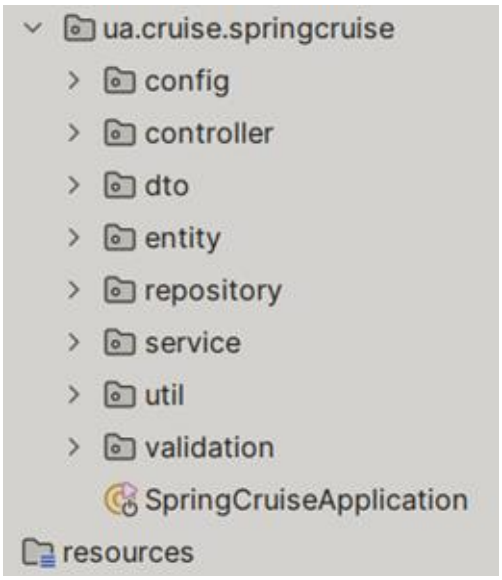


Рисунок 2.2.1 – Директорії системи

По зазначених директоріях погруповано класи, які відповідають певним вимогам та виконують специфічні функції з обробки даних. Класи контролери обробляють запити користувачів, направляючи їх до відповідних сервісів та формуючи структуру вихідних даних. Сервіси відповідають за перевірку отриманих даних, формування відповідей та виклик методів з репозиторіїв. Репозиторії є інтерфейсами, які виконують запити до бази даних, включаючи як персоналізовані запити, написані мовою SQL, так і автоматично згенеровані запити Пакет конфігурацій містить налаштування, необхідні для забезпечення безпечного доступу до системи. DTO-класи використовуються для перенесення даних між різними шарами додатка. Вони допомагають структурувати дані, що передаються від сервісів до контролерів і навпаки, з метою зменшення кількості переданої інформації та забезпечення безпеки. Утилітарні класи містять допоміжні функції та методи, які можуть використовуватися в різних частинах системи для полегшення та оптимізації коду. Модуль валідації містить класи та методи для перевірки коректності даних, що надходять у систему.

Клас SpringCruiseApplication є основним класом запуску додатка на базі Spring Boot і він не належить до жодного модуля системи. Він містить анотацію @SpringBootApplication, яка вказує Spring Boot, що це клас конфігурації, а також включає @EnableScheduling для дозволу планування завдань у додатку. Метод

main даного класу використовує SpringApplication.run для запуску всього додатка, ініціалізуючи контекст Spring та всі налаштування, необхідні для роботи додатка. Цей клас служить точкою входу в додаток, забезпечуючи запуск і налаштування всіх компонентів і служб, визначених у Spring контексті.

Рисунок 2.2.2 – Метод налаштування ланцюжка фільтрів безпеки

Клас SecurityConfig є конфігураційним класом, який налаштовує параметри безпеки для веб-додатка, використовуючи Spring Security. Цей клас визначає, як обробляються HTTP-запити щодо безпеки, а також містить налаштування для шифрування паролів. Метод filterChain(HttpSecurity http), який продемонстровано на рисунку 2.2.2, відповідає за налаштування ланцюжка фільтрів безпеки. Він вимикає захист від CSRF, визначає правила авторизації для різних маршрутів та налаштовує сторінку входу і виходу з системи. Зокрема, маршрути, що починаються на "/admin", доступні лише користувачам з ролями "OWNER" або "ADMIN", тоді як маршрути "/ticket/**" доступні для "OWNER", "ADMIN" та "USER". Інші загальнодоступні маршрути дозволяють доступ всім користувачам. Також налаштовується сторінка входу на "/auth/signIn" та сторінка виходу на "/auth/signOut". Метод bCryptPasswordEncoder() створює і повертає об'єкт BCryptPasswordEncoder, який використовується для шифрування паролів

користувачів. Це забезпечує збереження паролів у захищеному вигляді в базі даних, що є критично важливим для безпеки додатка.

Інтерфейс `EntityMapper` представляє собою контракт для конвертації об'єктів DTO в сутності і навпаки, що дозволяє забезпечити чітку відокремленість бізнес-логіки від представлення даних, підтримувати структурну цілісність та спрощувати взаємодію з базою даних або іншими джерелами даних у програмі. Він використовує анотацію `@Mapper` з параметром `componentModel = "spring"`, що дозволяє інтегрувати його з веб-додатком на основі Spring. Інтерфейс містить методи для перетворення об'єктів типу `Cruise`, `Liner`, `Route`, `RoutePoint`, `Ticket` і `User` у відповідні об'єкти DTO (`CruiseDTO`, `LinerDTO`, `RouteDTO`, `RoutePointDTO`, `TicketDTO`, `UserDTO`) і навпаки.

Інтерфейс BaseController, який відображено на рисунку 2.2.3, є базовим для контролерів і містить метод з типовою реалізацією, що використовується іншими контролерами. Метод checkModifiedObjectsConflict перевіряє конфлікти під час модифікації об'єктів. Він отримує ідентифікатор користувача, який виконує операцію, і порівнює його з користувачем, який вже модифікує цей об'єкт, якщо такий є.

```
public interface BaseController {  5 implementations
    default void checkModifiedObjectsConflict(Map<Long, String> map, Long id) throws ResponseStatusException {
        Object principal = SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        String login;
        if (principal instanceof UserDetails) {
            login = ((UserDetails) principal).getUsername();
        } else {
            login = principal.toString();
        }

        if (map.containsKey(id)) {
            if (!map.get(id).equals(login))
                throw new ResponseStatusException(HttpStatus.CONFLICT, "Object modifing conflict [object_id=" + id + ", login=" + login + "]");
        } else {
            map.put(id, login);
        }
    }
}
```

Рисунок 2.2.3 – Інтерфейс BaseController

Клас CruiseController є контролером у Spring, який обробляє запити користувачів, пов'язані з круїзами, і керує представленням відповідних даних. Він містить два основних методи: readAll і readById. Метод readAll обробляє GET-

запити для перегляду списку круїзів із можливістю фільтрації за датою, тривалістю та сторінками, використовуючи параметри запиту. Після обробки запиту, метод додає відповідні дані до моделі і повертає ім'я представлення для відображення списку круїзів. Метод readById обробляє GET-запити для перегляду конкретного круїзу за його ідентифікатором, знаходить круїз через сервісний клас CruiseService та додає його до моделі для відображення на відповідній сторінці.

Клас TicketController керує запитами користувачів, пов'язаними з квитками, забезпечуючи можливість створення, перегляду та зміни статусу квитків. Метод readByUserId відображає всі квитки користувача, витягуючи інформацію про користувача з контексту безпеки та додаючи її до моделі. Метод createForm відображає форму для створення нового квитка для певного круїзу, додаючи до моделі інформацію про круїз та список актуальних квитків. Метод create обробляє POST-запити для створення нового квитка, зберігаючи завантажене зображення квитка та перевіряючи, чи позиція не зайнята, а також оновлюючи статус круїзу за необхідності. Метод changeStatus змінює статус квитка на сплачений або скасований та відповідно оновлює статус круїзу залежно від заповненості.

Клас AuthController керує запитами незареєстрованих користувачів, пов'язаними з авторизацією, зокрема, реєстрацією нових користувачів. Метод signUpForm відображає форму реєстрації, додаючи до моделі об'єкт UserDTO. Метод signUp обробляє POST-запити для створення нового користувача, перевіряючи валідність введених даних і чи існує вже користувач з таким логіном. Якщо все в порядку, він кодує пароль користувача, встановлює роль за замовчуванням і зберігає нового користувача в базі даних. У разі помилки, метод повертає відповідний статус і повідомлення про помилку.

Клас AdminCruiseController є контролером веб-додатку, який обробляє запити користувачів, авторизованих як адміністратори, щодо адміністрування круїзів. Він забезпечує можливість виконання операцій CRUD для круїзів, а також управління статусами круїзів, які розпочалися чи закінчилися. Клас використовує анотації Spring для обробки HTTP-запитів, включаючи @GetMapping, @PatchMapping, @PostMapping, та @DeleteMapping, і мапує їх на відповідні

методи, які взаємодіють з сервісами (CruiseService, LinerService, RouteService, StorageService) для виконання бізнес-логіки додатку. Контролер також використовує клас EntityMapper для конвертації між сутностями і DTO. Крім того, використовується валідація об'єктів за допомогою анотації @Valid, а помилки обробляються за допомогою ResponseStatusException, щоб повернути відповідні статуси HTTP при невдалих спробах створення або оновлення лайнера. Аналогічно до цього класу побудовані класи AdminLinerController, AdminTicketController, AdminUserController, AdminRouteController

Клас AdminLinerController є контролером, який обробляє запити користувачів, авторизованих як адміністратори, щодо адміністрування лайнерів. Він забезпечує можливість виконання операцій CRUD (створення, читання, оновлення, видалення) для об'єктів типу Liner і перенаправляє їх на відповідні сервіси. Контролер використовує анотації Spring для мапінгу HTTP-запитів, такі як @GetMapping, @PatchMapping, @PostMapping та @DeleteMapping, на методи, що виконують відповідні операції з об'єктами Liner. Крім того, використовується валідація об'єктів за допомогою анотації @Valid, а помилки обробляються за допомогою ResponseStatusException, щоб повернути відповідні статуси HTTP при невдалих спробах створення або оновлення лайнера.

Клас AdminTicketController є контролером, який обробляє запити користувачів, авторизованих як адміністратори, щодо адміністрування квитків. Він надає можливість виконання операцій CRUD (створення, читання, оновлення, видалення) для об'єктів типу Ticket і перенаправляє їх на відповідні сервіси. Контролер використовує анотації Spring для мапінгу HTTP-запитів, такі як @GetMapping, @PatchMapping і @GetMapping, на методи, які забезпечують необхідні операції. Крім того, він використовує EntityMapper для конвертації об'єктів типу Ticket у відповідні об'єкти DTO (TicketDTO) і навпаки. Контролер також включає механізм валідації об'єктів за допомогою анотації @Valid і обробляє помилки за допомогою ResponseStatusException, щоб повертати відповідні статуси HTTP при виникненні проблем при оновленні статусів квитків або при спробах змінити вже зайнятий квиток.

Класи DTO (Data Transfer Object) використовуються для передачі даних між різними шарами програмного забезпечення, зокрема між рівнями контролера і сервісів. Вони представляють собою спрощені моделі даних, які відображають структуру сутностей (наприклад, круїзів, квитків, користувачів тощо) і містять анотації валідації, що дозволяють забезпечити правильність інформації. Класи DTO зазвичай містять поля, які відповідають полям сутностей, з якими вони пов'язані, а також додаткові анотації для валідації даних (такі як @NotNull, @NotBlank, @PositiveOrZero тощо). Ці об'єкти не мають логіки та є лише носіями даних, що спрощує їх маніпулювання і передачу між різними частинами програми. Приклад реалізації даних класів у системі зображено на рисунку 2.2.4.

```
@NoArgsConstructor  24 usages
@AllArgsConstructor
@Getter
@Setter
@Builder
public class TicketDTO {

    private Long id;

    @NotNull
    private Cruise cruise;

    @NotNull
    private User user;

    @NotEmpty
    private int position;

    private String imageName;

    @PositiveOrZero
    private BigDecimal price;

    private TicketStatus status;
}
```

Рисунок 2.2.4 – Приклад реалізації DTO класів додатка

Класи сутностей використовуються для моделювання об'єктів доменної моделі в базі даних через ORM (Object-Relational Mapping) технології. Вони відображають структуру даних, які зберігаються в базі даних, і містять поля, які відповідають колонкам таблиць бази даних. Приклад реалізації даних класів у системі зображено на рисунку 2.2.5. Класи можуть містити анотації (наприклад, @Entity, @Table, @Id, @GeneratedValue), які вказують на їх приналежність до

ORM-системи та налаштування зв'язків між сутностями. Вони також можуть реалізовувати інші інтерфейси для додаткової функціональності, такої як робота з даними користувачів, або мати методи, які обробляють бізнес-логіку об'єктів. Класи сутностей дозволяють зручно та ефективно працювати з даними бази даних через об'єктно-орієнтований підхід, а ORM технології дозволяють автоматизувати багато операцій з реляційною базою даних. У даній системі цими класами виступають: User, Ticket, Route, RoutePoint, Liner, Cruise, UserRole, TicketStatus, CruiseStatus.

```java
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@ToString
@Entity
@Table(name = "ticket", schema = "cruise")
public class Ticket {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    @Column(name = "id", nullable = false)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY)
    @OnDelete(action = OnDeleteAction.CASCADE)
    @JoinColumn(name = "cruise_id")
    @ToString.Exclude
    private Cruise cruise;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;

    private int position;

    private String imageName;

    private BigDecimal price;

    @ManyToOne
    @JoinColumn(name = "status_id")
    private TicketStatus status;
```

Рисунок 2.2.5 – Приклад реалізації класів сутностей

Класи-репозиторії використовуються для доступу до даних з бази даних через Spring Data JPA. Приклад реалізації даних класів у системі зображено на рисунку 2.2.6. Вони розширюють інтерфейси JpaRepository та JpaSpecificationExecutor, що надають стандартні методи для операцій CRUD та

специфікаційних запитів відповідно. Кожен репозиторій визначає власні методи, які відображають запити до бази даних або користувацькі запити через анотації @Query. Методи можуть бути анотовані з @Transactional та @Modifying для виконання транзакційних та модифікаційних запитів. Репозиторії дозволяють зручно взаємодіяти з даними в базі через об'єктно-орієнтовану модель і використовуються для інкапсуляції логіки доступу до даних у веб-додатках на основі Spring Framework. У даній системі цими класами виступають: UserRepository, TicketRepository, RouteRepository, RoutePointRepository, LinerRepository, CruiseRepository, UserRoleRepository, TicketStatusRepository, CruiseStatusRepository.

```
public interface CruiseRepository extends JpaRepository<Cruise, Long>, JpaSpecificationExecutor<Cruise> {  10 usages

    List<Cruise> findByOrderByIdAsc();

    Optional<Cruise> findByName(String name);

    boolean existsByName(String name);

    @Transactional  1 usage
    @Modifying
    @Query("update Cruise c set c.status = ?1 where c.startDateTime < ?2")
    int updateStartedCruises(CruiseStatus status, LocalDateTime startDateTime);

    @Transactional  1 usage
    @Modifying
    @Query("update Cruise c set c.status = ?1 where c.endDateTime < ?2")
    int updateEndedCruises(CruiseStatus status, LocalDateTime endDateTime);

    List<Cruise> findByStartDateTimeBefore(LocalDateTime startDateTime);  5 usages

    long countByIdAndTicketList_Status_IdLessThan(Long cruiseId, Long statusId);  3 usages

}
```

Рисунок 2.2.6 – Приклад реалізації класів-репозиторіїв

Класи-сервіси для сутностей виконують проміжну логіку між контролерами та репозиторіями для обробки бізнес-логіки та управління операціями з даними. Вони включають методи для отримання, створення, оновлення та видалення об'єктів з бази даних, валідацію вхідних даних, обробку виключень та реалізацію розкладених задач, таких як автоматичне оновлення статусів круїзів на основі їхньої актуальності та вільних місць на лайнерах. Технічні особливості таких

сервісів включають використання вбудованих можливостей Spring, таких як інжектування залежностей через анотації (@Autowired), використання репозиторіїв для доступу до даних через Spring Data JPA, організацію транзакцій для забезпечення консистентності даних, а також використання планувальника завдань (@Scheduled) для автоматичного оновлення статусів круїзів. Такі сервіси ізолюють логіку бізнес-процесів від інфраструктурних аспектів, що робить код більш зрозумілим, підтримуваним та легше тестованим. У системі даними класами є: UserService, TicketService, RouteService, RoutePointService, LinerService, CruiseService, UserDetailsServiceImpl, StorageStatus. Приклад реалізації даних класів у системі зображено на рисунку 2.2.7.

```
public class StorageService {

    public String save(Path dirPath, String fileExt, MultipartFile file) throws IOException {
        if (!Arrays.asList(ContentType.IMAGE_JPEG.getMimeType(),
                ContentType.IMAGE_PNG.getMimeType()).contains(file.getContentType()))
            throw new IOException("Wrong file format");
        if (!Files.exists(dirPath))
            Files.createDirectories(dirPath);
        String fileName = "";
        try (
                InputStream inputStream = file.getInputStream()) {
            Path filePath;
            do {
                fileName = UUID.randomUUID() + "." + fileExt;
                filePath = dirPath.resolve(fileName);
            } while (Files.exists(filePath));
            Files.copy(inputStream, filePath);
        } catch (
                IOException ex) {
            ex.printStackTrace();
        }
        return fileName;
    }

    public void delete(Path path) throws IOException {
        Files.deleteIfExists(path);
    }
```

Рисунок 2.2.7 – Приклад реалізації класів-сервісів

Класи валідаторів використовуються для перевірки коректності даних, які передаються у контролери чи сервіси. Клас DateTimeValidator є реалізацією валідатора для анотації DateTimeMatch, яка перевіряє коректність дат і часу в

об'єкті, який анотований цією анотацією. Він використовується для перевірки правильності встановлення дати та часу в об'єктах. Аналогічно створені класи LoginValidator для валідації логінів користувачів, NameValidator для імен користувачів та PasswordValidator для перевірки правильності написання паролів.

2.2.2. Розробка GUI

Для реалізації роботи користувацького інтерфейсу, додано класи, які відповідають за основний функціонал на сторінках.

Клас WebAppConfig є конфігураційним класом у Spring, який реалізує інтерфейс WebMvcConfigurer для налаштування веб-компонентів додатка, головні методи якого зображені на рисунку 2.2.8. Метод addViewControllers налаштовує прості контролери перегляду для певних URL-шляхів, а також встановлює порядок обробки цих контролерів на найвищий пріоритет. Метод addResourceHandlers визначає, де шукати статичні ресурси в додатку. Метод localeResolver визначає компонент LocaleResolver, який використовує сесію для збереження поточної локалі. За замовчуванням встановлюється українська локаль ("uk", "UA"). Метод localeChangeInterceptor створює інтерцептор для зміни локалі додатка на основі параметра запиту "lang". Інтерцептор перевіряє наявність цього параметра і змінює локаль, якщо значення валідне. Метод addInterceptors додає інтерцептор для зміни локалі до реєстру інтерцепторів, забезпечуючи його активацію для всіх запитів. Метод messageSource налаштовує компонент MessageSource, який забезпечує підтримку інтернаціоналізації, зчитуючи повідомлення з файлів ресурсів з базовим ім'ям "messages" та використовуючи кодування UTF-8.

```
@Bean
public LocaleChangeInterceptor localeChangeInterceptor() {
    List<String> localeList = Arrays.asList("uk", "en");
    LocaleChangeInterceptor lci = preHandle(request, response, handler) -> {
            String locale = request.getParameter(this.getParamName());
            if (locale != null && localeList.contains(locale)) {
                LocaleResolver localeResolver = RequestContextUtils.getLocaleResolver(request);
                if (localeResolver != null) {
                    localeResolver.setLocale(request, response, StringUtils.parseLocaleString(locale));
                }
            }
            return true;
    };

    lci.setParamName("lang");
    return lci;
}

@Override  no usages
public void addInterceptors(InterceptorRegistry registry) { registry.addInterceptor(localeChangeInterceptor()); }

@Bean
public MessageSource messageSource() {
    ResourceBundleMessageSource rbms = new ResourceBundleMessageSource();
    rbms.setBasename("messages");
    rbms.setDefaultEncoding("UTF-8");
    return rbms;
}
```

Рисунок 2.2.8 – Ключові методи реалізації зміни мови інтерфейсу

Клас AdminPanelController є контролером, який обробляє запити користувачів, авторизованих як адміністратори, щодо адміністрування панелі управління. Він відповідає за відображення інформації про об'єкти, які наразі модифікуються в системі. Контролер використовує анотації Spring для мапінгу HTTP-запитів (@GetMapping) на метод getAdminPanel, який збирає поточну інформацію про модифіковані об'єкти (круїзи, лайнери, маршрути, квитки, користувачі) і передає її в представлення з допомогою об'єкта Model. Це дозволяє відображати актуальний статус цих об'єктів у веб-інтерфейсі адміністративної панелі.

Інтерфейс користувача веб-додатка круїзної компанії є добре структурованим. Директорії проекту поділені за функціональними зонами: користувачі, круїзи, квитки, фрагменти, шляхи та сторінки обробки помилок. Користувачі, круїзи, квитки, фрагменти, шляхи мають окремі сторінки для користувачів-адмінів. Така організація дозволяє легко орієнтуватися в проекті та швидко знаходити необхідні файли для модифікацій або оновлень.

Інформаційні сторінки, такі як about.html та index.html, знаходяться в кореневій директорії.

У директорії user розміщені сторінки для управління користувачами, такі як сторінка входу(login.html) та сторінка з усіма даними користувача (readMy.html).

У директорії cruise містяться сторінки для управління круїзами, включаючи перегляд всіх круїзів (readAll.html) та детальну інформацію про конкретний круїз (readById.html). Ці сторінки надають зручний інтерфейс для перегляду та адміністрування круїзів.

Сторінки обробки помилок у директорії error відповідають за відображення відповідних HTTP-статусів помилок, таких як 404.html для "Not Found" та 500.html для "Internal Server Error", що покращує користувацький досвід у разі виникнення проблем.

Директорія fragments містить спільні компоненти інтерфейсу, такі як футер (footer.html), хедер (header.html) та навігаційна панель (navbar.html). Ці компоненти використовуються на інших сторінках для забезпечення єдиного вигляду та зручності навігації.

У директорії ticket розташовані сторінки для створення квитків (create.html) та перегляду особистих квитків користувача (readMy.html).

Також, у додатку є директорія admin, яка включає сторінки для адміністративного управління різними аспектами круїзів. У піддиректоріях, таких як cruise, liner, route, routePoint і ticket, розміщені сторінки для створення (create.html), перегляду (readAll.html, readById.html) та оновлення (update.html) відповідних сутностей, а також адміністративна панель (panel.html). Це забезпечує зручне адміністрування та підтримку актуальних даних користувачів та дозволяє адміністраторам додавати нові круїзи, лайнерів, маршрути, точки маршрутів та квитки, а також редагувати існуючі дані.

Файли конфігурації (application.properties) та логування (log4j2-spring.xml) забезпечують налаштування додатка, а ресурсні файли для інтернаціоналізації (messages_en.properties, messages_uk.properties) дозволяють підтримувати кілька мов, роблячи додаток доступним для користувачів з різних країн. Така структура

файлів полегшує підтримку, масштабування та розширення функціоналу додатка, забезпечуючи при цьому високу якість користувацького досвіду. Наразі у додатку присутні дві мови: українська (за замовчуванням) та англійська.

Файл style.css є основним CSS-файлом для веб-додатка круїзної компанії. Він відповідає за оформлення всіх HTML-сторінок і забезпечує єдиний стиль та зовнішній вигляд інтерфейсу користувача. Використовуючи цей файл, можна контролювати кольорову гаму, шрифти, відступи, розміри елементів та їх розташування на сторінці. Це дозволяє створити гармонійний і привабливий дизайн, який покращує користувацький досвід.

Файл style.css забезпечує централізоване управління стилями, що полегшує підтримку та внесення змін. При необхідності зміни дизайну достатньо відредагувати цей файл, і зміни автоматично застосуються до всіх сторінок додатка. Це робить процес розробки більш ефективним та організованим, знижуючи ймовірність виникнення стилістичних помилок.

2.2.3. Тестування програмного забезпечення та оцінка якості

Тестування програмного забезпечення є невід'ємною частиною процесу розробки, що має на меті виявлення та виправлення дефектів, забезпечення відповідності продукту заданим вимогам та підвищення його якості. Ефективне тестування дозволяє мінімізувати ризики, пов'язані з впровадженням нових функцій, і гарантує стабільність та надійність програмного забезпечення. Важливою складовою тестування є оцінка якості, яка включає не тільки виявлення помилок, але й аналіз продуктивності, безпеки та зручності використання системи.

Ручне тестування веб-додатка круїзної компанії виконувалося покроково, з ретельною перевіркою кожної функціональності та інтерфейсу користувача. Тестування було спрямоване на виявлення можливих багів, перевірку коректності

роботи всіх компонентів та забезпечення оптимальної взаємодії користувача з додатком.

Спочатку було проведено тестування сторінок автентифікації користувачів. Перевірялися форми реєстрації та входу до системи. Було протестовано різні сценарії, включаючи введення правильних і неправильних даних, спроби входу з неіснуючими обліковими записами та перевірку функціональності відновлення пароля. Усі повідомлення про помилки та підтвердження коректного входу були ретельно перевірені.

Наступним кроком було тестування інтерфейсу адміністратора. Перевірялися функції створення, редагування та видалення круїзів, лайнери, маршрутів та точок маршруту. Особливу увагу приділяли формам вводу даних, перевіряючи їх на наявність валідації та коректне збереження інформації. Також тестувалися сценарії оновлення статусів круїзів залежно від часу та зміни кількості доступних квитків.

Для користувацького інтерфейсу були перевірені сторінки перегляду та бронювання круїзів, перегляд інформації про користувача та управління особистими даними. Тестувалися функції перегляду доступних круїзів, їх деталей, а також процес бронювання квитків. Важливо було переконатися, що всі ці дії виконуються без помилок і зручні для користувача.

Окремо було протестовано сторінки помилок (400, 403, 404, 409, 500) для перевірки коректного відображення повідомлень про помилки. Перевірялися різні сценарії, які могли б спричинити помилки, та забезпечувалося, щоб користувач отримував зрозумілу інформацію про проблему та можливі шляхи її вирішення.

Після цього виконувалися тестування адаптивності дизайну. Використовуючи різні пристрої та розміри екранів, перевірялося, чи коректно відображається інтерфейс на мобільних пристроях, планшетах і десктопах. Особливу увагу приділяли навігації, формам вводу та таблицям, щоб вони залишалися зручними та функціональними на всіх пристроях.

Заключним етапом було тестування функціональності завантаження та видалення файлів, яке виконувалося через сервіс зберігання файлів. Перевірялося

завантаження зображень різних форматів, їх відображення на сторінках додатка та можливість видалення файлів без помилок.

Після завершення тестування можна було дійти декількох наступних висновків. Тестування автентифікації пройшло успішно, проте так, як система не прив’язує користувача до пошти може бути не повністю конфіденційним. Під час тестування інтерфейсу адміністратора, користувача, функціоналу завантаження файлів та сторінки посилок усі функції виконувались коректно. Тестування адаптивності показало, що сайт підтримує лише декстопкну версію, а у мобільній користування веб-додатком є не дуже зручним.

Автоматичне тестування веб-додатку круїзної компанії було виконано для перевірки функціональності різних компонентів системи. Це тестування охоплювало контролери та сервіси додатку, забезпечуючи якість та стабільність коду. Нижче наведено детальний опис автоматичних тестів, які були проведені.

SpringCruiseApplicationTests проводило загальне тестування функціональності всього додатку, включаючи коректний запуск контексту додатку та інтеграцію різних компонентів додатку.

AuthControllerTest зосереджувався на перевірці автентифікації користувачів. Було протестовано успішний вхід та реєстрацію користувачів, а також перевірку валідації даних під час входу. CruiseControllerTest включав тести на створення, оновлення та видалення круїзів, перевірку правильності відображення списку круїзів і обробку помилок при некоректних даних. TicketControllerTest забезпечував перевірку створення та видалення квитків, отримання списку квитків і коректності обробки помилок.

CruiseServiceTest зосереджувався на бізнес-логіці, пов'язаній з круїзами, включаючи правильність створення, оновлення та видалення круїзів, а також тести на фільтрацію та сортування круїзів. LinerServiceTest перевіряв функціональність сервісу лайнери, включаючи створення, оновлення та видалення лайнерів і коректність валідації даних. RoutePointServiceTest забезпечував тести на обробку точок маршруту, додавання, оновлення та видалення точок маршруту. RouteServiceTest перевіряв функціональність сервісу маршрутів, зокрема

створення, оновлення та видалення маршрутів, а також обробку помилок. StorageServiceTest включав тести на збереження та видалення файлів, перевірку правильності обробки різних форматів файлів і валідацію даних при завантаженні файлів. TicketServiceTest зосереджувався на перевірці функціональності сервісу квитків, включаючи створення, оновлення та видалення квитків і коректність відображення списку квитків. UserDetailsServiceImplTest забезпечував перевірку автентифікації та авторизації користувачів, завантаження користувачів по логіну і правильність обробки даних користувачів. UserServiceTest перевіряв бізнес-логіку, пов'язану з користувачами, включаючи створення, оновлення та видалення користувачів, а також валідацію даних.

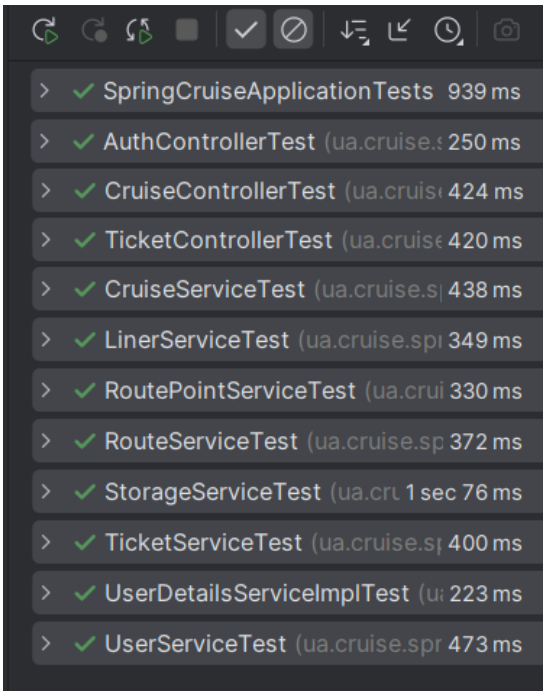


Рисунок 2.2.9 – Результати автоматичного тестування

Автоматичне тестування веб-додатку круїзної компанії виявило, що всі основні функціональні можливості працюють коректно. Тести на контролери та сервіси підтвердили, що додаток відповідає вимогам і може коректно обробляти дані користувачів, квитків, круїзів та інших компонентів. Результати автоматичного тестування зображено на рисунку 2.2.9.

2.2.4. Результати розробки

У процесі розробки веб-додатку круїзної компанії на Java з використанням Spring Framework було успішно виконано поставлені завдання та додаток відповідає вимогам проекту. Основні функціональні можливості додатку включають управління круїзами, бронювання квитків, адміністрування користувачів та обробку маршрутів. Ці функції забезпечують користувачам можливість зручного та ефективного управління своїми подорожами, а адміністраторам — можливість легко керувати системою.

Архітектура системи побудована на багатошаровій моделі, що забезпечує високу модульність і легкість у супроводі та розширенні. Основними компонентами архітектури є контролери, сервіси, репозиторії та сутності, кожен з яких відповідає за певний аспект роботи системи. Використання Spring Framework дозволило реалізувати такі принципи, як інверсія керування та впровадження залежностей, що значно спрощує розробку та тестування окремих модулів системи. Для доступу до бази даних використовується Spring Data JPA, що забезпечує зручний та ефективний механізм роботи з даними.

Інтерфейс користувача розроблений з урахуванням принципів зручності та інтуїтивності. Основні елементи інтерфейсу включають сторінки для перегляду та бронювання круїзів, управління особистим кабінетом користувача та адміністрування системи. Для візуалізації даних використовуються сучасні технології, такі як HTML5, CSS3 та JavaScript, що забезпечує швидку та зручну взаємодію користувача з системою.

У процесі тестування було проведено як ручне, так і автоматичне тестування компонентів системи. Тести охоплювали перевірку функціональності контролерів, сервісів та репозиторіїв, а також інтеграційні тести для перевірки коректної взаємодії між компонентами. Результати тестування підтвердили стабільність та надійність системи, а також відповідність функціональних можливостей вимогам

проекту. Було виявлено та виправлено кілька незначних помилок, що дозволило підвищити загальну якість коду.

Під час розробки особлива увага приділялася оптимізації продуктивності системи. Було проведено аналіз використання ресурсів та виявлено можливі "вузькі місця". На основі цього аналізу було здійснено оптимізацію алгоритмів обробки даних та зменшення часу відгуку системи. В результаті вдалося досягти високої продуктивності та швидкодії веб-додатку, що є важливим фактором для користувачів.

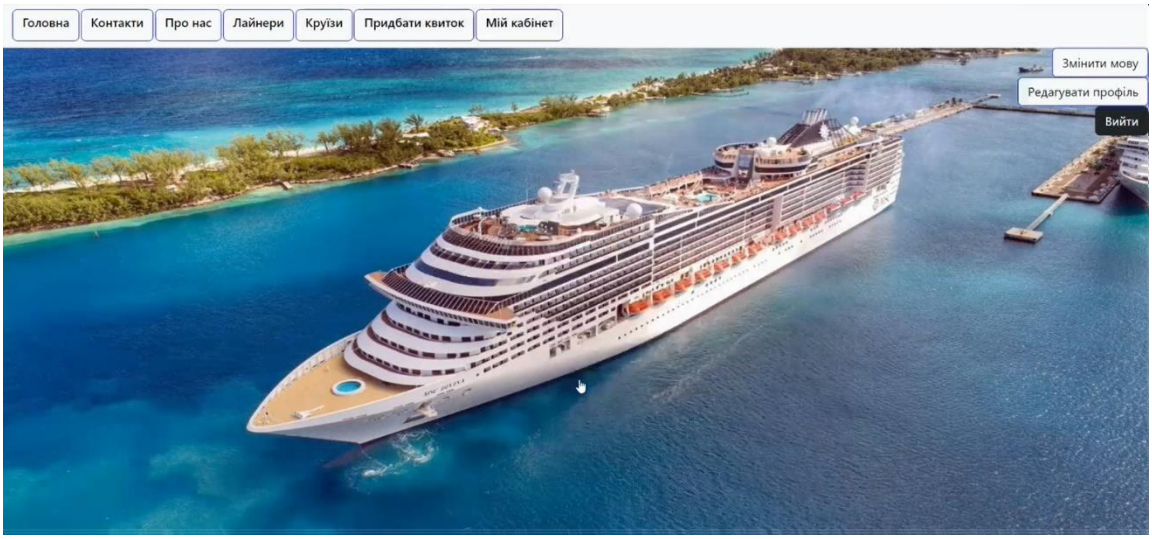


Рисунок 2.2.10 – Головна сторінка веб-додатку

Розроблене програмне забезпечення повністю відповідає вимогам та специфікації проекту. Всі основні функції реалізовані відповідно до поставлених задач, і додаток успішно справляється з обробкою даних користувачів, квитків та круїзів. Система забезпечує надійну та безпечну роботу, що підтверджується результатами тестування.

У порівнянні з аналогічними рішеннями, розроблений веб-додаток демонструє конкурентні переваги, такі як висока швидкодія, зручний інтерфейс та гнучкість у налаштуванні. Це робить його привабливим вибором для круїзних компаній, які прагнуть забезпечити своїм клієнтам якісне та зручне обслуговування.

3 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

3.1 Управління та нагляд за безпекою життєдіяльності в Україні

Контроль за дотриманням законодавства у сфері безпеки життєдіяльності в Україні здійснюється різними державними та громадськими організаціями. Серед цих організацій виділяються:

Органи загальної компетенції: Верховна Рада, Кабінет Міністрів, виконавчі комітети місцевих рад народних депутатів, місцеві адміністрації.

Органи спеціальної компетенції: ці органи контролюють діяльність підприємств, установ, організацій і громадян щодо охорони праці, охорони здоров'я, захисту навколишнього природного середовища.

Органи галузевої компетенції: ці структури контролюють питання охорони праці, здоров'я та екології в межах конкретних галузей народного господарства.

Центральним виконавчим органом, що займається управлінням безпекою і захистом у надзвичайних ситуаціях, є Державна служба України з надзвичайних ситуацій (ДСНС). Цей орган відповідає за реалізацію державної політики у сферах цивільного захисту, запобігання і ліквідації надзвичайних ситуацій, пожежної та техногенної безпеки, діяльності аварійно-рятувальних служб, а також гідрометеорологічної діяльності. ДСНС має право видавати накази, організовувати їх виконання та здійснювати контроль за їх реалізацією.

Координацію діяльності виконавчої влади в області цивільного захисту здійснюють Рада національної безпеки і оборони України та Кабінет Міністрів.

Державне управління охороною праці здійснюють Кабінет Міністрів, Міністерство соціальної політики, міністерства та інші центральні органи виконавчої влади, місцеві державні адміністрації та органи самоврядування. Кабінет Міністрів відповідає за реалізацію державної політики в сфері охорони праці, затверджує національні програми з поліпшення стану безпеки, гігієни праці та виробничого середовища, визначає функції міністерств і інших органів у створенні безпечних умов праці та контролює їх дотримання.

Для розробки і реалізації державної системи управління охороною праці створена Національна рада з питань безпечної життєдіяльності населення при Кабінеті Міністрів, яку очолює віце-прем'єр-міністр України. Міністерство соціальної політики здійснює державну експертизу умов праці, контролює якість атестації робочих місць і бере участь у розробці нормативних актів про охорону праці. У міністерствах та інших органах державної виконавчої влади створюються служби охорони праці для координації та вдосконалення роботи в цій сфері.

Відповідальність за реалізацію державної політики у сфері охорони здоров'я несуть органи державної виконавчої влади, а кінцеву відповідальність – Президент України. Кабінет Міністрів організовує розробку та впровадження загальнодержавних програм, створює необхідні механізми для ефективної діяльності у сфері охорони здоров'я, забезпечує розвиток мережі медичних закладів та координує міжнародне співробітництво з питань охорони здоров'я.

Міністерства, відомства та інші центральні органи виконавчої влади розробляють програми та прогнози у сфері охорони здоров'я, встановлюють державні стандарти, критерії та вимоги до охорони здоров'я населення, формують державні замовлення для матеріально-технічного забезпечення галузі, здійснюють державний контроль та іншу виконавчо-розпорядчу діяльність. Міністерство охорони здоров'я України є спеціально уповноваженим органом державної виконавчої влади в цій галузі, його компетенція визначається відповідним положенням, затвердженим Кабінетом Міністрів України.

Державну санітарно-епідеміологічну службу України очолює державний санітарний лікар України – перший заступник міністра охорони здоров'я. Основні напрями діяльності цієї служби включають здійснення санітарно-епідеміологічного нагляду, визначення пріоритетних заходів у профілактиці захворювань, охорону здоров'я населення від шкідливого впливу факторів навколишнього середовища, вивчення та прогнозування показників здоров'я населення залежно від стану середовища життєдіяльності людини, встановлення санітарно-гігієнічних вимог до продукції з відходів та видачу гігієнічного сертифіката на неї.

Управління охороною навколишнього середовища в Україні здійснюють Кабінет Міністрів, Міністерство екології та природних ресурсів, місцеві ради народних депутатів зі своїми виконавчими органами, а також спеціалізовані державні органи, що відповідають за охорону довкілля і раціональне використання природних ресурсів. Міністерство екології та природних ресурсів координує всі природоохоронні заходи в країні, розробляє пропозиції для Кабінету Міністрів щодо збереження природи і ефективного використання водних ресурсів, створює рекомендації для вдосконалення господарського управління природокористуванням, встановлює екологічні норми, правила та стандарти, розробляє довгострокові державні програми з охорони довкілля, проводить екологічну експертизу планів розвитку та розміщення продуктивних сил, здійснює контроль за дотриманням екологічних стандартів при розробці нової техніки, технологій та матеріалів, а також проводить екологічну оцінку проектів новобудов і існуючих промислових об'єктів. Міністерство має повноваження забороняти будівництво, реконструкцію або розширення промислових та інших об'єктів, а також проведення робіт з експлуатації природних ресурсів, якщо вони порушують природоохоронне законодавство. Воно також може притягати до відповідальності організації та громадян за порушення природоохоронних норм. Міністерство взаємодіє з Міністерством охорони здоров'я та його санітарно-епідеміологічними службами, Міністерством аграрної політики, Державним комітетом з водного господарства, Державним комітетом із земельних ресурсів та Міністерством транспорту України.

Державтоінспекція МВС України виступає спеціалізованим державним органом, відповідальним за контроль і нагляд у сфері автомобільного дорожнього руху. Генеральний прокурор України разом зі своїми підлеглими прокурорами здійснюють вищий нагляд за дотриманням і правильним застосуванням законодавства, включаючи закони з безпеки життєдіяльності.

3.2 Оцінка травмонебезпеки технологічного процесу

Оцінка травмонебезпеки технологічного процесу є важливою складовою забезпечення безпеки праці на виробництві. Вона включає аналіз потенційних небезпек та визначення заходів для мінімізації ризиків травмування працівників.

Травмонебезпеки технологічного процесу можна класифікувати за різними критеріями, такими як технічні, організаційні, санітарно-гігієнічні та психофізіологічні причини. Технічні причини включають недосконаліть технологічного процесу, конструктивні недоліки обладнання, недостатню механізацію важких робіт, недосконале огородження та відсутність спеціальних захисних засобів.

Організаційні причини, які часто стають причиною травмування, включають незадовільний стан території, порушення правил експлуатації обладнання, порушення технологічного регламенту, недостатнє технічне обслуговування та ремонт обладнання, а також недоліки в навчанні робітників безпечним методам праці.

Санітарно-гігієнічні причини включають перевищення норм запиленості та загазованості повітря робочої зони, недостатнє природне освітлення, підвищені рівні шуму та вібрації, ультразвукові та інфразвукові коливання на робочому місці, а також підвищений рівень радіації.

Психофізіологічні причини включають фізичні та нервово-психічні перевантаження працівників, які можуть бути пов'язані з втомленістю, підвищеною дратливістю або хворобливим станом працівників.

Для оцінки травмонебезпеки технологічного процесу використовуються різні методи, включаючи: аналіз причин виробничого травматизму; планування профілактичних заходів; впровадження заходів з охорони праці.

Аналіз причин виробничого травматизму дозволяє виявити основні фактори, що призводять до травмування працівників. Більшість нещасних випадків трапляється через організаційні причини, зокрема порушення трудової та

виробничої дисципліни, правил безпеки руху та вимог безпеки під час експлуатації устаткування.

На основі результатів аналізу травматизму плануються профілактичні заходи, спрямовані на створення безпечних та нешкідливих умов праці на робочих місцях. Це включає впровадження технічних та організаційних заходів для попередження травмування працівників.

Впровадження заходів з охорони праці та створення безпечних умов праці є важливою складовою забезпечення безпеки на виробництві. Це включає створення системи управління охороною праці, навчання працівників безпечним методам роботи, забезпечення засобами індивідуального захисту та здійснення технічного нагляду за безпекою робіт.

У даній темі описується веб-додаток круїзної компанії, тому розглянемо травмонебезпеки технологічного процесу для судна, що є критично важливим для забезпечення безпеки екіпажу та збереження судна в належному технічному стані.

Перше, що потрібно зробити, це ідентифікувати можливі джерела небезпеки на судні. До них можна віднести технічні збої, неправильну експлуатацію обладнання, несприятливі погодні умови, недотримання техніки безпеки тощо. Наприклад, небезпеку можуть становити такі фактори, як робота з вантажопідйомними механізмами, обслуговування двигунів, використання електричного обладнання та виконання робіт на палубі під час шторму.

Технічні причини травмонебезпеки на судні можуть включати зношеність або несправність обладнання, відсутність необхідних захисних огороджень, недостатню механізацію важких робіт. Для зниження технічних ризиків необхідно регулярно проводити технічне обслуговування та ремонт обладнання, модернізувати застарілі системи, забезпечувати екіпаж засобами індивідуального захисту та встановлювати сучасні засоби контролю та автоматизації.

Організаційні причини травматизму можуть включати незадовільний стан робочих місць, порушення правил експлуатації обладнання, недостатнє навчання екіпажу. Для усунення цих причин необхідно забезпечити належну організацію робочого процесу, регулярні інструктажі та тренінги для екіпажу, суворий

контроль за дотриманням правил безпеки. Також важливо запровадити систему моніторингу та аудиту умов праці.

Санітарно-гігієнічні умови праці на судні мають велике значення для збереження здоров'я екіпажу. Підвищена вологість, недостатнє освітлення, шум та вібрація можуть негативно вплинути на самопочуття працівників та підвищити ризик травм. Для зниження цих ризиків необхідно забезпечити відповідні санітарні умови, встановити системи вентиляції, шумоізоляції та віброізоляції.

Робота на судні часто пов'язана з фізичними та нервово-психічними навантаженнями. Постійне перебування в замкненому просторі, робота у важких умовах, віддаленість від берегу можуть вплинути на психічний стан екіпажу. Для зниження психофізіологічних ризиків важливо забезпечити належний режим роботи та відпочинку, організувати підтримку морального стану екіпажу, проводити психологічні тренінги.

Виконаємо оцінку деяких ризиків на пасажирський суднах. Для розрахунків скористаємося наступними даними:

- Середня кількість пасажирів на борту: 200 осіб.
- Кількість інцидентів за рік: 10 випадків падіння, 2 випадки ураження струмом, 1 випадок травми при евакуації.
- Середня кількість робочих днів у році: 300 днів.

Формула ймовірності виникнення інциденту:

$$P_{інциденту} = \frac{\text{Кількість інцидентів}}{\text{Кількість днів у році} \times \text{Кількість пасажирів}} \quad (3.1)$$

Розрахунок ймовірності виникнення падінь:

$$P_{\text{падіння}} = \frac{10}{300 \times 200} = 0.000167 \quad (3.2)$$

Розрахунок ймовірності ураження струмом:

$$P_{струм} = \frac{2}{300 \times 200} = 0.000033 \quad (3.3)$$

Розрахунок ймовірності травм при евакуації:

$$P_{евакуація} = \frac{1}{300 \times 200} = 0.000017 \quad (3.4)$$

Враховуємо такі параметри для оцінки тяжкості:

- Падіння: середня тяжкість (5 балів за 10-бальною шкалою).
- Ураження струмом: висока тяжкість (8 балів за 10-бальною шкалою).
- Травми при евакуації: дуже висока тяжкість (10 балів за 10-бальною шкалою).

Далі визначаємо інтегральний показник ризику (R), обчислюється як добуток ймовірності та тяжкості наслідків.

Ризик падіння:

$$R_{\text{падіння}} = P_{\text{падіння}} \times S_{\text{падіння}} = 0.000167 \times 5 = 0.000835 \quad (3.5)$$

Ризик ураження струмом:

$$R_{\text{струм}} = P_{\text{струм}} \times S_{\text{струм}} = 0.000033 \times 8 = 0.000264 \quad (3.6)$$

Ризик травм при евакуації:

$$R_{\textit{евакуація}} = P_{\text{евакуація}} \times S_{\text{евакуація}} = 0.000017 \times 10 = 0.00017 \quad (3.7)$$

Загальний ризик визначається як сума всіх інтегральних показників ризику.

$$R_{\text{заг}} = R_{\text{падіння}} + R_{\text{струм}} + R_{\text{евакуація}} = 0.000835 + 0.000264 + \\ + 0.00017 = 0.001269 \quad (3.8)$$

На основі проведених розрахунків можна зробити висновок, що загальний ризик травмонебезпеки технологічного процесу для пасажирського судна є прийнятним, проте потребує подальших заходів для зниження ймовірності виникнення небезпек. Зокрема, необхідно посилити контроль за виконанням техніки безпеки, проводити регулярне навчання екіпажу та забезпечити належне технічне обслуговування обладнання.

Таким чином, оцінка травмонебезпеки технологічного процесу є необхідною для забезпечення безпеки праці на виробництві. Вона включає аналіз причин травматизму, планування профілактичних заходів та впровадження заходів з охорони праці, що сприяють зменшенню ризиків травмування працівників.

ВИСНОВКИ

У процесі виконання дипломної роботи на тему "Розробка веб-додатку круїзної компанії на Java з використанням Spring Framework" було досягнуто поставлених цілей. Розроблено повнофункціональний веб-додаток, що відповідає сучасним вимогам до програмного забезпечення. Використання Spring Framework дозволило створити масштабовану та гнучку систему з чіткою архітектурою та високою продуктивністю.

Реалізовані функціональні можливості додатку забезпечують повний цикл обслуговування клієнтів круїзної компанії, включаючи управління круїзами, бронювання квитків та керування маршрутами. Інтерфейс користувача розроблений з урахуванням зручності та інтуїтивності, що робить систему легкою у використанні.

Автоматичне тестування підтвердило стабільність та надійність роботи додатку, що є важливим фактором для його впровадження у реальне середовище. Проведена оптимізація дозволила досягти високих показників продуктивності та ефективності використання ресурсів.

Загалом, розроблений веб-додаток є конкурентоспроможним продуктом, що може бути успішно використаний у сфері круїзних перевезень. Подальший розвиток системи може включати додавання нових функцій та інтеграцію з іншими сервісами, що дозволить ще більше покращити якість обслуговування клієнтів та підвищити ефективність роботи компанії.

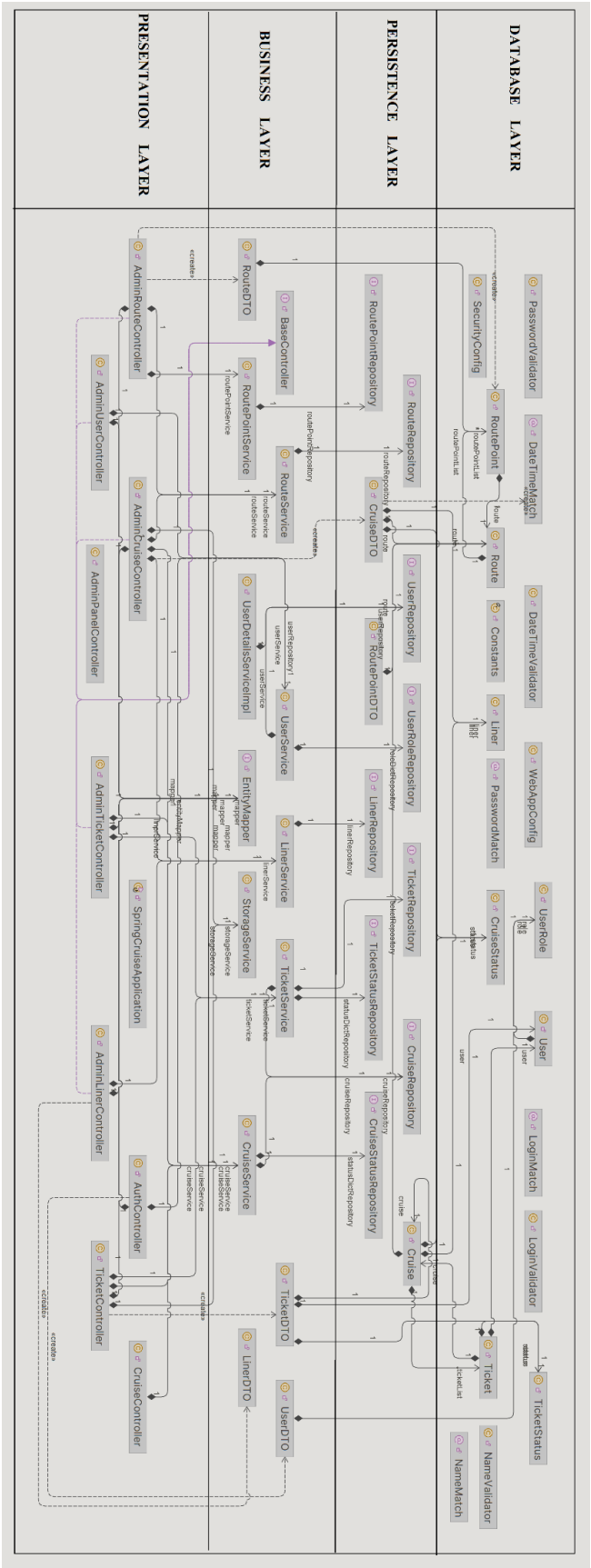
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання атестаційної роботи магістра спеціальності 121 – Інженерія програмного забезпечення (Освітньо-професійна програма - «Програмне забезпечення систем», Освітньо-наукова програма -«Інженерія програмного забезпечення») для студентів усіх форм навчання / Упор.: М.Р. Петрик, Д.М. Михалик, О.Ю. Петрик, Г.Б. Цуприк - Тернопіль: ТНТУ, 2020-51с
2. Інформація про компанію FOUR GATES UKRAINE LTD [Електронний ресурс] – Режим доступу до ресурсу: https://4gates.com.ua/ukr/information/.
3. Chhetri Y. Global Tourism Statistics 2024:Facts and Forecasts [Електронний ресурс] / Yam Chhetri // WordPress Travel. – 2024. – Режим доступу до ресурсу: https://wptravel.io/global-tourism-industry-statistics/.
4. Java Language Specification [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/specs/jls/se11/html/jls-1.html.
5. Spring [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/.
6. Hibernate ORM Documentation - 7.0 [Електронний ресурс] – Режим доступу до ресурсу: https://hibernate.org/orm/documentation/7.0/.
7. Maven Documentation [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://maven.apache.org/guides/.
8. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://www.postgresql.org.
9. IntelliJ IDEA overview | IntelliJ IDEA Documentation [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://www.jetbrains.com/help/idea/discover-intellij-idea.html.
10. Клієнт-серверна архітектура [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture/.

11. Apache Log4j [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://logging.apache.org/log4j/2.x/.
12. Ghimire Y. Software Architecture Patterns. Layered Architecture [Електронний ресурс] / Yubraj Ghimire // Medium.com. – 2021. – Режим доступу до ресурсу: https://medium.com/etribes-tech/software-architecture-patterns-c2ec7d0ca537.
13. Unified Modeling Language [Електронний ресурс] – Режим доступу до ресурсу: http://www.omg.org/spec/UML/.
14. Моралес Д. UML Class Diagram [Електронний ресурс] / Джейд Моралес // MindOnMap. – 2023. – Режим доступу до ресурсу: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/.
15. Гандзюк М.П., Желібо Є.П., Халімовський М.О. Основи охорони праці: Підручник. 5-е вид. / За ред. М.П. Гандзюка. – К.: Каравела, 2011. – 384 с.
16. Іванова О. В. Конспект лекцій з дисципліни “Безпека життєдіяльності та основи охорони праці” / О. В. Іванова, Н. М. Ювченко. – Одеса, 2018. – 156 с.
17. Навчальний посібник «Безпека життєдіяльності» / Т. Є.Стиценко, Г. В. Пронюк, Н. М. Сердюк, І. І. Хондак. – Харків, 2018. – 200 с.
18. Сєріков Я. О. Безпека життєдіяльності та охорона праці / Я. О. Сєріков, Л. Ф. Коженевські, М. В. Хворост. – Харків, 2021. – 255 с.

ДОДАТКИ

ДОДАТОК А


DATABASE LAYER
PERSISTENCE LAYER
BUSINESS LAYER
PRESENTATION LAYER
PasswordValidator
DateTimeMatch
SecurityConfig
RoutePoint
Route
Constants
Liner
PasswordMatch
UserRole
CruiseStatus
User
LoginMatch
LoginValidator
TicketStatus
Ticket
NameValidator
NameMatch
DateTimeValidator
WebAppConfig
RouteRepository
RoutePointRepository
CruiseDTO
RoutePointDTO
UserRepository
UserRoleRepository
LinerRepository
TicketRepository
TicketStatusRepository
CruiseRepository
CruiseStatusRepository
Cruise
RouteDTO
BaseController
RoutePointService
RouteService
UserDetailsServiceImpl
UserService
EntityMapper
LinerService
StorageService
TicketService
CruiseService
TicketDTO
LinerDTO
UserDTO
AdminRouteController
AdminUserController
AdminCruiseController
AdminPanelController
AdminTicketController
SpringCruiseApplication
AdminLinerController
AuthController
TicketController
CruiseController

ДОДАТОК Б

Лістинг коду:

Файл SpringCruiseApplication.java

```
package ua.cruise.springcruise;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;

@EnableScheduling
@SpringBootApplication
public class SpringCruiseApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringCruiseApplication.class, args);
    }

}
```

Файл SecurityConfig.java

```
package ua.cruise.springcruise.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.builders.HttpSecu
rity;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;

/**
 * A configuration class that contains custom implementation of
default settings related to security
 * @version 1.1
 * @see Configuration
 */

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws
Exception {
        http.csrf().disable().authorizeRequests()
```

```
                .antMatchers("/admin-panel**", "/admin-cruise**",
"/admin-liner/**", "/admin-route/**",
                        "/admin-routepoint/**", "admin-ticket/**",
"/admin-user/**")
                .hasAnyRole("OWNER", "ADMIN")
                .antMatchers("/ticket/**").hasAnyRole("OWNER",
"ADMIN", "USER")
                .antMatchers("/", "/about", "/auth/**",
"/cruise/**", "/data/**", "/error/*").permitAll()
                .anyRequest().authenticated()
                .and()
                .formLogin()
                .loginPage("/auth/signIn")
                .usernameParameter("login")
                .defaultSuccessUrl("/", false)
                .permitAll()
                .and()
                .logout()
                .logoutUrl("/auth/signOut")
                .invalidateHttpSession(false)
                //.deleteCookies("JSESSIONID")
                .logoutSuccessUrl("/")
                .permitAll();
        return http.build();
    }

    @Bean
    public BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }


}
```

Файл WebAppConfig.java

```
package ua.cruise.springcruise.config;

import org.springframework.context.MessageSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.context.support.ResourceBundleMessageSource;
import org.springframework.core.Ordered;
import org.springframework.util.StringUtils;
import org.springframework.web.servlet.LocaleResolver;
import
org.springframework.web.servlet.config.annotation.InterceptorRegistr
y;
import
org.springframework.web.servlet.config.annotation.ResourceHandlerReg
istry;
```

```
import
org.springframework.web.servlet.config.annotation.ViewControllerRegi
stry;
import
org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
import org.springframework.web.servlet.i18n.LocaleChangeInterceptor;
import org.springframework.web.servlet.i18n.SessionLocaleResolver;
import org.springframework.web.servlet.support.RequestContextUtils;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.Arrays;
import java.util.List;
import java.util.Locale;

/**
 * A configuration class that contains custom implementation of
default settings related to web
 * @version 1.1
 * @see Configuration
 * @see
org.springframework.web.servlet.config.annotation.WebMvcConfigurer
 */

@Configuration
public class WebAppConfig implements WebMvcConfigurer {

    @Override
    public void addViewControllers(ViewControllerRegistry registry)
{

registry.addViewController("/auth/signIn").setViewName("user/login")
;
        registry.addViewController("/about").setViewName("about");
        registry.setOrder(Ordered.HIGHEST_PRECEDENCE);
    }

    @Override
    public void addResourceHandlers(final ResourceHandlerRegistry
registry) {

registry.addResourceHandler("/data/**").addResourceLocations("file:D
:/data/spring/");
    }

    @Bean
    public LocaleResolver localeResolver() {
        SessionLocaleResolver slr = new SessionLocaleResolver();
        Locale ukrainianLocale = new Locale("uk", "UA");
        slr.setDefaultLocale(ukrainianLocale);
        return slr;
    }
```

```
    @Bean
    public LocaleChangeInterceptor localeChangeInterceptor() {
        List<String> localeList = Arrays.asList("uk", "en");
        LocaleChangeInterceptor lci = new LocaleChangeInterceptor()
{
            @Override
            public boolean preHandle(HttpServletRequest request,
HttpServletResponse response, Object handler) {
                String locale =
request.getParameter(this.getParamName());
                if (locale != null && localeList.contains(locale)) {
                    LocaleResolver localeResolver =
RequestContextUtils.getLocaleResolver(request);
                    if (localeResolver != null) {
                        localeResolver.setLocale(request, response,
StringUtils.parseLocaleString(locale));
                    }
                }
                return true;
            }
        };

        lci.setParamName("lang");
        return lci;
    }

    @Override
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(localeChangeInterceptor());
    }

    @Bean
    public MessageSource messageSource() {
        ResourceBundleMessageSource rbms = new
ResourceBundleMessageSource();
        rbms.setBasename("messages");
        rbms.setDefaultEncoding("UTF-8");
        return rbms;
    }
}
```

Файл AdminCruiseController.java

```
package ua.cruise.springcruise.controller;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.util.ConcurrentReferenceHashMap;
import org.springframework.util.StringUtils;
```

```
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.server.ResponseStatusException;
import ua.cruise.springcruise.dto.CruiseDTO;
import ua.cruise.springcruise.entity.Cruise;
import ua.cruise.springcruise.entity.Liner;
import ua.cruise.springcruise.entity.Route;
import ua.cruise.springcruise.entity.dictionary.CruiseStatus;
import ua.cruise.springcruise.service.StorageService;
import ua.cruise.springcruise.util.EntityMapper;
import ua.cruise.springcruise.service.CruiseService;
import ua.cruise.springcruise.service.LinerService;
import ua.cruise.springcruise.service.RouteService;
import ua.cruise.springcruise.util.Constants;

import javax.validation.Valid;
import java.io.IOException;
import java.nio.file.Path;
import java.util.*;

/**
 * An controller class that controls authorized as administrators
users' requests related to cruise administration
 * (such as CRUD operations) and redirects them to requested
services. Controls view layer.
 * @version 1.1
 * @see Controller
 */

@Log4j2
@Controller
@RequiredArgsConstructor
@RequestMapping("/admin-cruise")
public class AdminCruiseController implements BaseController {
    private final CruiseService cruiseService;
    private final LinerService linerService;
    private final RouteService routeService;
    private final StorageService storageService;
    private final EntityMapper entityMapper;

    protected static final Map<Long, String>
currentlyModifiedCruises = new ConcurrentReferenceHashMap<>();


    private static final String REDIRECT_URL = "redirect:/admin-
cruise";

    @GetMapping("")
    public String readAll(Model model) {
        List<Cruise> cruiseList = cruiseService.findAll();
        model.addAttribute("cruiseList", cruiseList);
        return "admin/cruise/readAll";
```

```
    }

    @GetMapping("/{id}/edit")
    public String updateForm(@PathVariable Long id, Model model) {
        checkModifiedObjectsConflict(currentlyModifiedCruises, id);
        Cruise cruise = cruiseService.findById(id);
        CruiseDTO cruiseDTO = entityMapper.cruiseToDTO(cruise);
        List<Liner> linerList = linerService.findAll();
        List<Route> routeList = routeService.findAll();
        List<CruiseStatus> statusList =
cruiseService.findAllStatuses();
        model.addAttribute("cruiseDTO", cruiseDTO);
        model.addAttribute("linerList", linerList);
        model.addAttribute("routeList", routeList);
        model.addAttribute("statusList", statusList);
        return "admin/cruise/update";
    }


    @PatchMapping("/{id}")
    public String update(@PathVariable("id") long id,
@ModelAttribute("cruiseDTO") @Valid CruiseDTO cruiseDTO,
                         BindingResult result, @RequestParam(value =
"image", required = false) MultipartFile file) {
        checkModifiedObjectsConflict(currentlyModifiedCruises, id);
        if (result.hasErrors())
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "CruiseDTO is not
valid");
        Cruise cruise = entityMapper.dtoToCruise(cruiseDTO);
        cruise.setId(id);
        if (cruiseService.existsByName(cruise.getName()) &&

!Objects.equals(cruiseService.findByName(cruise.getName()).getId(),
cruise.getId()))
            throw new
ResponseStatusException(HttpStatus.NOT_ACCEPTABLE, "Failed to update
cruise: name is already taken");
        Path fileDirectory;
        String fileName;
        if (file != null && !file.isEmpty()) {
            String fileExt =
StringUtils.getFilenameExtension(file.getOriginalFilename());
            fileDirectory = Path.of(Constants.DATA_PATH +
"/cruise/");
            try {
                fileName = storageService.save(fileDirectory,
fileExt, file);
                String oldFileName = cruise.getImageName();
                cruise.setImageName(fileName);
                cruiseService.update(cruise);
                storageService.delete(Path.of(fileDirectory +
oldFileName));
```

```
            } catch (IOException ex) {
                log.info("Failed to save image for cruise", ex);
                throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Failed to create
cruise: cant save image");
            }
        } else {

cruise.setImageName(cruiseService.findById(cruise.getId()).getImageN
ame());
            cruiseService.update(cruise);
        }
        currentlyModifiedCruises.remove(id);
        return REDIRECT_URL;
    }

    @GetMapping("/new")
    public String createForm(Model model) {
        CruiseDTO cruiseDTO = new CruiseDTO();
        List<Liner> linerList = linerService.findAll();
        List<Route> routeList = routeService.findAll();
        model.addAttribute("cruiseDTO", cruiseDTO);
        model.addAttribute("linerList", linerList);
        model.addAttribute("routeList", routeList);
        return "admin/cruise/create";
    }

    @PostMapping()
    public String create(@ModelAttribute("cruiseDTO") @Valid
CruiseDTO cruiseDTO, BindingResult result,
                         @RequestParam("image") MultipartFile file)
{
        if (result.hasErrors()) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "CruiseDTO is not
valid");
        }
        Cruise cruise = entityMapper.dtoToCruise(cruiseDTO);

cruise.setStatus(cruiseService.findStatusById(Constants.CRUISE_DEFAU
LT_STATUS_ID));
        if (cruiseService.existsByName(cruise.getName()))
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Failed to create
cruise: name is already taken");
        String fileExt =
StringUtils.getFilenameExtension(file.getOriginalFilename());
        Path fileDirectory;
        String fileName;
        try {
            fileDirectory = Path.of(Constants.DATA_PATH +
"/cruise/");
```

```
            fileName = storageService.save(fileDirectory, fileExt,
file);
            cruise.setImageName(fileName);
        } catch (IOException ex) {
            log.info("Failed to save image for cruise", ex);
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Failed to create
cruise: cant save image");
        }
        cruiseService.create(cruise);
        return REDIRECT_URL;
    }

    @DeleteMapping("/{id}")
    public String delete(@PathVariable Long id) {
        checkModifiedObjectsConflict(currentlyModifiedCruises, id);
        cruiseService.delete(id);
        currentlyModifiedCruises.remove(id);
        return REDIRECT_URL;
    }

    /**
     * Updates status of all cruises and has been started or ended
     * @return String that contains URL to requested view
     */

    @GetMapping("/statusUpdate")
    public String statusUpdate() {
        if (currentlyModifiedCruises.isEmpty())
            cruiseService.updateAllStartedAndEndedCruiseStatuses();
        else
            throw new ResponseStatusException(HttpStatus.CONFLICT,
"Failed to force update all cruise statuses: cruise(s) is being
modified");
        return REDIRECT_URL;
    }
}
```

Файл AdminLinerController.java

```
package ua.cruise.springcruise.controller;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.util.ConcurrentReferenceHashMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.server.ResponseStatusException;
import ua.cruise.springcruise.dto.LinerDTO;
```

```
import ua.cruise.springcruise.entity.Liner;
import ua.cruise.springcruise.util.EntityMapper;
import ua.cruise.springcruise.service.LinerService;

import javax.validation.Valid;
import java.util.List;
import java.util.Map;
import java.util.Objects;

/**
 * An controller class that controls authorized as administrators
users' requests related to liner administration
 * (such as CRUD operations) and redirects them to requested
services. Controls view layer.
 * @version 1.1
 * @see Controller
 */

@Log4j2
@Controller
@RequiredArgsConstructor
@RequestMapping("/admin-liner")
public class AdminLinerController implements BaseController {
    private final LinerService linerService;
    private final EntityMapper mapper;

    protected static final Map<Long, String> currentlyModifiedLiners
= new ConcurrentReferenceHashMap<>();

    private static final String REDIRECT_URL = "redirect:/admin-
liner";

    @GetMapping("")
    public String readAll(Model model) {
        List<Liner> linerList = linerService.findAll();
        model.addAttribute("linerList", linerList);
        return "admin/liner/readAll";
    }

    @GetMapping("/{id}/edit")
    public String updateForm(@PathVariable Long id, Model model) {
        checkModifiedObjectsConflict(currentlyModifiedLiners, id);
        Liner liner = linerService.findById(id);
        LinerDTO linerDTO = mapper.linerToDTO(liner);
        model.addAttribute("linerDTO", linerDTO);
        return "admin/liner/update";
    }

    @PatchMapping("/{id}")
    public String update(@PathVariable("id") Long id,
@ModelAttribute("linerDTO") @Valid LinerDTO linerDTO,
                         BindingResult result) {
        checkModifiedObjectsConflict(currentlyModifiedLiners, id);
```

```
        if (result.hasErrors())
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "LinerDTO is not
valid");
        Liner liner = mapper.dtoToLiner(linerDTO);
        liner.setId(id);
        if (linerService.existsByName(liner.getName()) &&

!Objects.equals(linerService.findByName(liner.getName()).getId(),
liner.getId()))
            throw new
ResponseStatusException(HttpStatus.NOT_ACCEPTABLE, "Failed to update
liner: name already taken [name= " + liner.getName() + "]");
        try {
            linerService.update(liner);
        } catch (ResponseStatusException ex) {
            return "/error/501";
        }
        currentlyModifiedLiners.remove(id);
        return REDIRECT_URL;
    }

    @GetMapping("/new")
    public String createForm(Model model) {
        LinerDTO linerDTO = new LinerDTO();
        model.addAttribute("linerDTO", linerDTO);
        return "admin/liner/create";
    }

    @PostMapping()
    public String create(@ModelAttribute("linerDTO") @Valid LinerDTO
linerDTO, BindingResult result) {
        if (result.hasErrors())
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "LinerDTO is not
valid");
        Liner liner = mapper.dtoToLiner(linerDTO);
        if (linerService.existsByName(liner.getName()))
            throw new
ResponseStatusException(HttpStatus.NOT_ACCEPTABLE, "Failed to create
liner: name already taken [name=" + liner.getName() + "]");
        try {
            linerService.create(liner);
        } catch (ResponseStatusException ex) {
            ex.printStackTrace();
        }
        return REDIRECT_URL;
    }

    @DeleteMapping("/{id}")
    public String delete(@PathVariable Long id) {
        checkModifiedObjectsConflict(currentlyModifiedLiners, id);
        linerService.delete(id);
```

```
        currentlyModifiedLiners.remove(id);
        return REDIRECT_URL;
    }
}
```

Файл AdminPanelController.java

```
package ua.cruise.springcruise.controller;

import lombok.extern.log4j.Log4j2;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

/**
 * An controller class that controls authorized as administrators
users' requests related to administration panel
 * (such as currently modified objects) and redirects them to
requested services. Controls view layer.
 * @version 1.1
 * @see Controller
 */

@Log4j2
@Controller
@RequestMapping("/admin-panel")
public class AdminPanelController {

    @GetMapping("")
    public String getAdminPanel(Model model){
        model.addAttribute("currentlyModifiedCruises",
AdminCruiseController.currentlyModifiedCruises);
        model.addAttribute("currentlyModifiedLiners",
AdminLinerController.currentlyModifiedLiners);
        model.addAttribute("currentlyModifiedRoutes",
AdminRouteController.currentlyModifiedRoutes);
        model.addAttribute("currentlyModifiedTickets",
AdminTicketController.currentlyModifiedTickets);
        model.addAttribute("currentlyModifiedUsers",
AdminUserController.currentlyModifiedUsers);
        return "admin/panel";
    }

}
```

Файл AdminRouteController.java

```
package ua.cruise.springcruise.controller;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.http.HttpStatus;
```

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.util.ConcurrentReferenceHashMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.server.ResponseStatusException;
import ua.cruise.springcruise.dto.RouteDTO;
import ua.cruise.springcruise.dto.RoutePointDTO;
import ua.cruise.springcruise.entity.Route;
import ua.cruise.springcruise.entity.RoutePoint;
import ua.cruise.springcruise.service.RoutePointService;
import ua.cruise.springcruise.util.EntityMapper;
import ua.cruise.springcruise.service.RouteService;

import javax.validation.Valid;
import java.util.List;
import java.util.Map;
import java.util.Objects;

/**
 * An controller class that controls authorized as administrators
users' requests related to route and routepoint administration
 * (such as CRUD operations) and redirects them to requested
services. Controls view layer.
 * @version 1.1
 * @see Controller
 */

@Log4j2
@Controller
@RequiredArgsConstructor
@RequestMapping("/admin-route")
public class AdminRouteController implements BaseController {
    private final RouteService routeService;
    private final RoutePointService routePointService;
    private final EntityMapper mapper;

    protected static final Map<Long, String> currentlyModifiedRoutes
= new ConcurrentReferenceHashMap<>();

    private static final String REDIRECT_URL = "redirect:/admin-
route";

    @GetMapping("")
    public String readAll(Model model) {
        List<Route> routeList = routeService.findAll();
        model.addAttribute("routeList", routeList);
        return "admin/route/readAll";
    }

    @GetMapping("/{id}/edit")
    public String updateForm(@PathVariable Long id, Model model) {
        checkModifiedObjectsConflict(currentlyModifiedRoutes, id);
```

```
        Route route = routeService.findById(id);
        RouteDTO routeDTO = mapper.routeToDTO(route);
        model.addAttribute("routeDTO", routeDTO);
        return "admin/route/update";
    }

    @PatchMapping("/{id}")
    public String update(@PathVariable("id") Long id,
@ModelAttribute("routeDTO") @Valid RouteDTO routeDTO, BindingResult
result) {
        checkModifiedObjectsConflict(currentlyModifiedRoutes, id);
        if (result.hasErrors())
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "RouteDTO is not
valid");
        Route route = mapper.dtoToRoute(routeDTO);
        route.setId(id);
        if (routeService.existsByName(route.getName()) &&

!Objects.equals(routeService.findByName(route.getName()).getId(),
route.getId()))
            throw new
ResponseStatusException(HttpStatus.NOT_ACCEPTABLE, "Failed to update
route [id=" + id + "]");
        routeService.update(route);
        currentlyModifiedRoutes.remove(id);
        return REDIRECT_URL;
    }

    @GetMapping("/new")
    public String createForm(Model model) {
        RouteDTO routeDTO = new RouteDTO();
        model.addAttribute("routeDTO", routeDTO);
        return "admin/route/create";
    }

    @PostMapping()
    public String create(@ModelAttribute("routeDTO") @Valid RouteDTO
routeDTO, BindingResult result) {
        if (result.hasErrors())
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "RouteDTO is not
valid");
        Route route = mapper.dtoToRoute(routeDTO);
        if (routeService.existsByName(route.getName()))
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Failed to create
route: name already taken [id=" + route.getId() + "]");
        try {
            routeService.create(route);
        } catch (ResponseStatusException ex) {
            ex.printStackTrace();
        }
```

```
        return REDIRECT_URL;
    }

    @DeleteMapping("/{id}")
    public String delete(@PathVariable Long id) {
        checkModifiedObjectsConflict(currentlyModifiedRoutes, id);
        routeService.delete(id);
        currentlyModifiedRoutes.remove(id);
        return REDIRECT_URL;
    }

    @GetMapping("/{routeId}/point")
    public String readPointByRouteId(@PathVariable Long routeId,
Model model) {
        List<RoutePoint> routePointList =
routePointService.findByRouteId(routeId);
        model.addAttribute("routeId", routeId);
        model.addAttribute("routePointList", routePointList);
        return "admin/routePoint/readById";
    }

    @GetMapping("/{routeId}/point/{pointId}/edit")
    public String pointUpdateForm(@PathVariable("routeId") Long
routeId,
                                  @PathVariable("pointId") Long
pointId, Model model) {
        checkModifiedObjectsConflict(currentlyModifiedRoutes,
routeId);
        RoutePoint routePoint = routePointService.findById(pointId);
        RoutePointDTO routePointDTO =
mapper.routePointToDTO(routePoint);
        List<RoutePoint> routePointList =
routePointService.findByRouteId(routeId);
        model.addAttribute("routePointList", routePointList);
        model.addAttribute("routePointDTO", routePointDTO);
        return "admin/routePoint/update";
    }

    @PatchMapping("/{routeId}/point/{pointId}")
    public String pointUpdate(@PathVariable("routeId") Long routeId,
                              @PathVariable("pointId") Long pointId,
                              @ModelAttribute("routePointDTO")
@Valid RoutePointDTO routePointDTO, BindingResult result) {
        checkModifiedObjectsConflict(currentlyModifiedRoutes,
routeId);
        if (result.hasErrors())
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "RoutepointDTO is
not valid");
        RoutePoint routePoint =
mapper.dtoToRoutePoint(routePointDTO);
        routePoint.setId(pointId);
        routePoint.setRoute(routeService.findById(routeId));
```

```
        if (routePointService.existsByName(routePoint.getName()) &&

!Objects.equals(routePointService.findByName(routePoint.getName()).g
etId(), routePoint.getId()))
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Failed to update
routepoint: name already taken[name=" + routePoint.getName() + "]");
        try {
            routePointService.update(routePoint);
        } catch (ResponseStatusException ex) {
            ex.printStackTrace();
        }
        currentlyModifiedRoutes.remove(routeId);
        return REDIRECT_URL + "/" + routeId + "/point";
    }

    @PostMapping("/{routeId}/point")
    public String pointCreate(@PathVariable Long routeId) {
        checkModifiedObjectsConflict(currentlyModifiedRoutes,
routeId);
        try {
            RoutePoint routePoint = new RoutePoint();
            Route route = routeService.findById(routeId);
            routePoint.setRoute(route);
            routePointService.create(routePoint);
        } catch (ResponseStatusException ex) {
            ex.printStackTrace();
        }
        return REDIRECT_URL + "/" + routeId + "/point";
    }

    @DeleteMapping("/{routeId}/point/{pointId}")
    public String pointDelete(@PathVariable("routeId") Long routeId,
                              @PathVariable("pointId") Long pointId)
{
        checkModifiedObjectsConflict(currentlyModifiedRoutes,
routeId);
        routePointService.delete(pointId);
        currentlyModifiedRoutes.remove(routeId);
        return REDIRECT_URL + "/" + routeId + "/point";
    }
}
```

Файл AdminTicketController.java

```
package ua.cruise.springcruise.controller;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
```

```
import org.springframework.util.ConcurrentReferenceHashMap;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.server.ResponseStatusException;
import ua.cruise.springcruise.dto.TicketDTO;
import ua.cruise.springcruise.entity.Ticket;
import ua.cruise.springcruise.entity.dictionary.TicketStatus;
import ua.cruise.springcruise.service.CruiseService;
import ua.cruise.springcruise.util.EntityMapper;
import ua.cruise.springcruise.service.TicketService;

import java.util.List;
import java.util.Map;

/**
 * An controller class that controls authorized as administrators
users' requests related to ticket administration (
 * such as CRUD operations) and redirects them to requested
services. Controls view layer.
 * @version 1.1
 * @see Controller
 */

@Log4j2
@Controller
@RequiredArgsConstructor
@RequestMapping("/admin-ticket")
public class AdminTicketController implements BaseController {
    private final TicketService ticketService;
    private final CruiseService cruiseService;
    private final EntityMapper mapper;

    protected static final Map<Long, String>
currentlyModifiedTickets = new ConcurrentReferenceHashMap<>();

    private static final String REDIRECT_URL = "redirect:/admin-
ticket";

    @GetMapping("")
    public String readAll(Model model) {
        List<Ticket> ticketList = ticketService.findAll();
        model.addAttribute("ticketList", ticketList);
        return "admin/ticket/readAll";
    }

    @GetMapping("/{id}/edit")
    public String updateForm(@PathVariable Long id, Model model) {
        checkModifiedObjectsConflict(currentlyModifiedTickets, id);
        Ticket ticket = ticketService.findById(id);
        TicketDTO ticketDTO = mapper.ticketToDTO(ticket);
        List<TicketStatus> statusList =
ticketService.findStatusDict();
        model.addAttribute("ticketDTO", ticketDTO);
```

```
        model.addAttribute("statusList", statusList);
        return "admin/ticket/update";
    }

    @PatchMapping("/{id}")
    public String update(@PathVariable("id") Long id,
@ModelAttribute("ticketDTO") TicketDTO ticketDTO, BindingResult
result) {
        checkModifiedObjectsConflict(currentlyModifiedTickets, id);
        Ticket ticket = ticketService.findById(id);
        TicketStatus status =
ticketService.findStatusById(ticketDTO.getStatus().getId());
        ticket.setStatus(status);
        try {
            ticketService.update(ticket);

cruiseService.updateCruiseStatusDueToCapacity(ticketService.findById
(id).getCruise());
        } catch (ResponseStatusException ex) {
            ex.printStackTrace();
        }
        currentlyModifiedTickets.remove(id);
        return REDIRECT_URL;
    }

    /**
     * Updates status of all tickets contains cruises that have been
started or ended
     * @return String that contains URL to requested view
     */

    @GetMapping("/statusUpdate")
    public String statusUpdate() {
        if (!currentlyModifiedTickets.isEmpty())
            throw new ResponseStatusException(HttpStatus.CONFLICT,
"Failed to force update all ticket statuses: tickets(s) is being
modified");
        ticketService.updateAllOutdatedTicketStatuses();
        return REDIRECT_URL;
    }
}
```

Файл AdminUserController.java

```
package ua.cruise.springcruise.controller;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.util.ConcurrentReferenceHashMap;
import org.springframework.web.bind.annotation.*;
```

```
import org.springframework.web.server.ResponseStatusException;
import ua.cruise.springcruise.dto.UserDTO;
import ua.cruise.springcruise.entity.User;
import ua.cruise.springcruise.entity.dictionary.UserRole;
import ua.cruise.springcruise.util.EntityMapper;
import ua.cruise.springcruise.service.UserService;

import java.util.List;
import java.util.Map;

/**
 * An controller class that controls authorized as administrators
users' requests related to user administration
 * (such as CRUD operations) and redirects them to requested
services. Controls view layer.
 * @version 1.1
 * @see Controller
 */

@Log4j2
@Controller
@RequiredArgsConstructor
@RequestMapping("/admin-user")
public class AdminUserController implements BaseController {
    private final UserService userService;
    private final EntityMapper mapper;

    protected static final Map<Long, String> currentlyModifiedUsers
= new ConcurrentReferenceHashMap<>();

    private static final String REDIRECT_URL = "redirect:/admin-
user";

    @GetMapping("")
    public String readAll(Model model) {
        List<User> userList = userService.findAll();
        model.addAttribute("userList", userList);
        return "admin/user/readAll";
    }

    @GetMapping("/{id}/edit")
    public String updateForm(@PathVariable Long id, Model model) {
        checkModifiedObjectsConflict(currentlyModifiedUsers, id);
        User user = userService.findById(id);
        UserDTO userDTO = mapper.userToDTO(user);
        List<UserRole> roleDict = userService.findRoleDict();
        model.addAttribute("userDTO", userDTO);
        model.addAttribute("roleDict", roleDict);
        return "admin/user/update";
    }

    @PatchMapping("/{id}")
```

```
    public String update(@PathVariable("id") Long id,
@ModelAttribute("userDTO") UserDTO userDTO) {
        checkModifiedObjectsConflict(currentlyModifiedUsers, id);
        User user = userService.findById(id);
        UserRole role =
userService.findRoleById(userDTO.getRole().getId());
        user.setRole(role);
        try {
            userService.update(user);
        } catch (ResponseStatusException ex) {
            ex.printStackTrace();
        }
        currentlyModifiedUsers.remove(id);
        return REDIRECT_URL;
    }

}
```

Файл AuthController.java

```
package ua.cruise.springcruise.controller;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.http.HttpStatus;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.server.ResponseStatusException;
import ua.cruise.springcruise.dto.UserDTO;
import ua.cruise.springcruise.entity.User;
import ua.cruise.springcruise.util.Constants;
import ua.cruise.springcruise.util.EntityMapper;
import ua.cruise.springcruise.service.UserService;

import javax.validation.Valid;

/**
 * An controller class that controls unauthorized users' requests
related to authorization (such as sign up)
 * and redirects them to requested services. Controls view layer.
 * @version 1.1
 * @see Controller
 */

@Log4j2
@Controller
@RequiredArgsConstructor
@RequestMapping("/auth")
public class AuthController {
```

```
    private final UserService userService;
    private final PasswordEncoder passwordEncoder;
    private final EntityMapper mapper;

    private static final String REDIRECT_URL =
"redirect:/auth/signIn";

    @GetMapping("/signUp")
    public String signUpForm(Model model) {
        UserDTO userDTO = new UserDTO();
        model.addAttribute("userDTO", userDTO);
        return "user/create";
    }

    @PostMapping("/signUp")
    public String signUp(@ModelAttribute("userDTO") @Valid UserDTO
userDTO, BindingResult result) {
        if (result.hasErrors())
            return "user/create";
        User user = mapper.dtoToUser(userDTO);

user.setPassword(passwordEncoder.encode(user.getPassword()));

user.setRole(userService.findRoleById(Constants.USER_DEFAULT_ROLE_ID
));
        if (userService.existsByLogin(user.getLogin()))
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Failed to create
user: login already taken [login=" + user.getLogin() + "]");
        try {
            userService.create(user);
        } catch (ResponseStatusException ex) {
            ex.printStackTrace();
        }
        return REDIRECT_URL;
    }

}
```

Файл BaseController.java

```
package ua.cruise.springcruise.controller;

import org.springframework.http.HttpStatus;
import
org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Controller;
import org.springframework.web.server.ResponseStatusException;

import java.util.Map;
```

```
/**
 * A base interface for controllers that contains common method(s)
with default implementation
 * @version 1.1
 * @see Controller
 */

public interface BaseController {
    default void checkModifiedObjectsConflict(Map<Long, String> map,
Long id) throws ResponseStatusException {
        Object principal =
SecurityContextHolder.getContext().getAuthentication().getPrincipal(
);
        String login;
        if (principal instanceof UserDetails) {
            login = ((UserDetails) principal).getUsername();
        } else {
            login = principal.toString();
        }

        if (map.containsKey(id)) {
            if (!map.get(id).equals(login))
                throw new
ResponseStatusException(HttpStatus.CONFLICT, "Object modifing
conflict [object_id=" + id + ", login=" + login + "]");
        } else {
            map.put(id, login);
        }
    }
}
```

Файл CruiseController.java

```
package ua.cruise.springcruise.controller;

import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.data.domain.Page;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import ua.cruise.springcruise.entity.Cruise;
import ua.cruise.springcruise.service.CruiseService;
import ua.cruise.springcruise.util.Constants;

import java.time.LocalDateTime;
import java.time.format.DateTimeParseException;
import java.time.temporal.ChronoUnit;

/**
 * An controller class that controls authorized users' requests
related to cruise (such as view all cruises) and redirects
```

```
 * them to requested services. Controls view layer.
 * @version 1.1
 * @see Controller
 */

@Log4j2
@Controller
@RequiredArgsConstructor
@RequestMapping("/cruise")
public class CruiseController {
    private final CruiseService cruiseService;

    @GetMapping("")
    public String readAll(@RequestParam(value = "dateTimeSign",
required = false) String dateTimeSignAsString,
                          @RequestParam(value = "startDateTime",
required = false) String dateTimeAsString,
                          @RequestParam(value = "durationSign",
required = false) String durationSignAsString,
                          @RequestParam(value = "duration", required
= false) Integer duration,
                          @RequestParam(value = "durationType",
required = false) String durationTypeAsString,
                          @RequestParam(value = "page", required =
false) Integer pageId,
                          @RequestParam(value = "perPage", required
= false) Integer countPerPage,
                          Model model) {
        Constants.equalitySign dateTimeSign;
        try {
            dateTimeSign =
Constants.equalitySign.valueOf(dateTimeSignAsString);
        } catch (IllegalArgumentException | NullPointerException ex)
{
            dateTimeSign = null;
        }
        LocalDateTime dateTime;
        try {
            dateTime =
LocalDateTime.parse(dateTimeAsString).truncatedTo(ChronoUnit.SECONDS
);
        }catch (DateTimeParseException | NullPointerException ex){
            dateTime = null;
        }
        Constants.equalitySign durationSign;
        try {
            durationSign =
Constants.equalitySign.valueOf(durationSignAsString);
        } catch (IllegalArgumentException | NullPointerException ex)
{
            durationSign = null;
        }
        Constants.durationType durationType;
```

```
        if (duration == null)
            duration = 0;
        int durationAsDays = 0;
        try {
            durationType =
Constants.durationType.valueOf(durationTypeAsString);
            if (durationType.equals(Constants.durationType.WEEKS))
                durationAsDays = duration * 7;
            else if
(durationType.equals(Constants.durationType.MONTHS))
                durationAsDays = duration * 30;
            else
                durationAsDays = duration;
        } catch (IllegalArgumentException | NullPointerException ex)
{
            durationType = null;
        }
        if (pageId == null)
            pageId = 1;
        if (countPerPage == null)
            countPerPage = 5;
        Page<Cruise> cruisePage =
cruiseService.findByStartDateTimeAndDuration(dateTimeSign, dateTime,
durationSign, durationAsDays, pageId-1, countPerPage);
        model.addAttribute("cruiseList", cruisePage);
        model.addAttribute("dateTimeSign", dateTimeSign);
        model.addAttribute("dateTime", dateTime);
        model.addAttribute("durationSign", durationSign);
        model.addAttribute("duration", duration);
        model.addAttribute("durationType", durationType);
        model.addAttribute("signList",
Constants.equalitySign.values());
        model.addAttribute("durationTypeList",
Constants.durationType.values());
        model.addAttribute("page", pageId);
        model.addAttribute("perPage", countPerPage);
        return "cruise/readAll";
    }

    @GetMapping("/{id}")
    public String readById(@PathVariable Long id, Model model) {
        Cruise foundCruise = cruiseService.findById(id);
        model.addAttribute("cruise", foundCruise);
        return "cruise/readById";
    }
}
```

Файл TicketController.java

```
package ua.cruise.springcruise.controller;

import lombok.RequiredArgsConstructor;
```

```
import lombok.extern.log4j.Log4j2;
import org.springframework.http.HttpStatus;
import org.springframework.security.core.Authentication;
import
org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.util.StringUtils;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.server.ResponseStatusException;
import ua.cruise.springcruise.dto.TicketDTO;
import ua.cruise.springcruise.entity.Cruise;
import ua.cruise.springcruise.entity.Ticket;
import ua.cruise.springcruise.entity.User;
import ua.cruise.springcruise.service.StorageService;
import ua.cruise.springcruise.util.Constants;
import ua.cruise.springcruise.util.EntityMapper;
import ua.cruise.springcruise.service.CruiseService;
import ua.cruise.springcruise.service.TicketService;

import java.io.IOException;
import java.nio.file.Path;
import java.util.List;

/**
 * An controller class that controls authorized users' requests
related to ticket (such as create ticket) and redirects
 * them to requested services. Controls view layer.
 * @version 1.1
 * @see Controller
 */

@Log4j2
@Controller
@RequiredArgsConstructor
@RequestMapping("/ticket")
public class TicketController {
    private final TicketService ticketService;
    private final CruiseService cruiseService;
    private final StorageService storageService;
    private final EntityMapper mapper;

    private static final String REDIRECT_URL = "redirect:/ticket";

    @GetMapping("")
    public String readByUserId(Model model) {
        Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
        User user = (User) auth.getPrincipal();
        List<Ticket> ticketList =
ticketService.findByUserId(user.getId());
        model.addAttribute("userLogin", user.getLogin());
```

```
        model.addAttribute("ticketList", ticketList);
        return "ticket/readMy";
    }

    @GetMapping("{id}/new")
    public String createForm(@PathVariable("id") Long id, Model
model) {
        TicketDTO ticketDTO = new TicketDTO();
        Cruise cruise = cruiseService.findById(id);
        ticketDTO.setCruise(cruise);
        Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
        User user = (User) auth.getPrincipal();
        ticketDTO.setUser(user);
        List<Ticket> ticketList =
ticketService.findByCruiseActual(cruise);
        model.addAttribute("ticketDTO", ticketDTO);
        model.addAttribute("ticketList", ticketList);
        return "ticket/create";
    }

    @PostMapping("/{id}")
    public String create(@PathVariable("id") long id,
                         @ModelAttribute("ticketDTO") TicketDTO
ticketDTO,
                         @RequestParam("image") MultipartFile file)
{
        String fileExt =
StringUtils.getFilenameExtension(file.getOriginalFilename());
        Ticket ticket = mapper.dtoToTicket(ticketDTO);
        Cruise cruise = cruiseService.findById(id);
        ticket.setCruise(cruise);
        Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
        User user = (User) auth.getPrincipal();
        ticket.setUser(user);
        ticket.setPrice(cruise.getPrice());

ticket.setStatus(ticketService.findStatusById(Constants.TICKET_DEFAU
LT_STATUS_ID));
        if (ticketService.exists(cruise, ticket.getPosition()))
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Failed to create
ticket: position already taken [pos=" + ticket.getPosition() + "]");
        Path fileDirectory;
        String fileName;
        try {
            fileDirectory = Path.of(Constants.DATA_PATH + "/docs/");
            fileName = storageService.save(fileDirectory, fileExt,
file);
            ticket.setImageName(fileName);
            ticketService.create(ticket);
            cruiseService.updateCruiseStatusDueToCapacity(cruise);
```

```
        } catch (ResponseStatusException | IOException ex) {
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Failed to save
image for ticket", ex);
        }
        return REDIRECT_URL;
    }

    @PatchMapping("{id}")
    public String changeStatus(@PathVariable Long id,
@RequestParam("status") long statusId) {
        if (statusId == Constants.TICKET_PAYED_STATUS)
            ticketService.pay(id);
        else if (statusId == Constants.TICKET_CANCELED_STATUS)
            ticketService.cancel(id);
        else
            throw new
ResponseStatusException(HttpStatus.BAD_REQUEST, "Failed to update
ticket status: wrong id [id=" + id + "]");

cruiseService.updateCruiseStatusDueToCapacity(ticketService.findById
(id).getCruise());
        return REDIRECT_URL;
    }
}
```

ДОДАТОК В

Диск з роботою