Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Факультет інформаційних технологій
(факультет)

Кафедра системного аналізу та управління
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра

Здобувача вищої освіти Колягіна Данила Сергійовича

академічної групи 124-21-1

спеціальності 124 Системний аналіз

за освітньо-професійною програмою Системний аналіз

на тему: «Розробка мікросервісу для бронювання квитків на авіарейси за допомогою зовнішнього API»

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*к.т.н., доц. Желдак Т.А.*			
розділів:				
Інформаційно-аналітичний	*к.т.н., доц. Желдак Т.А.*			
Спеціальний розділ	*к.т.н., доц. Желдак Т.А.*			
Рецензент	*д.т.н.,проф. Алексєєв М.А.*			
Нормоконтролер	*к.ф.-м.н., доц. Хом'як Т.В.*			

Дніпро
2025

ЗАТВЕРДЖЕНО:
завідувач кафедри
Системного аналізу та управління
(повна назва)

_____ *к.т.н., доц. Желдак Т.А.*
(підпис) (прізвище,ініціали)

«____» _____20___року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня *бакалавра*

здобувачу вищої освіти *Колягіну Д. С.* **академічної групи** *124- 21-1*

спеціальності: *124 Системний аналіз*
за освітньо-професійною програмою *Системний аналіз*
на тему «*Розробка мікросервісу для бронювання квитків на авіарейси за допомогою зовнішнього API»*
затверджену наказом ректора НТУ «Дніпровська політехніка» від *05.05.2025 р. №336-с*

Розділ	Зміст	Терміни виконання
1. Інформаційно-аналітичний розділ	*Проаналізувати структуру об'єкта дослідження. Визначити предметну область дослідження та проблему, що розв'язується. Обґрунтувати методи виконання поставлених завдань*	10.01.2025 – 01.03.2025
2.Спеціальний розділ	*Розв'язати поставлені задачі: розробити алгоритми та створити систему для автоматизації пасажирських перевезень, враховуючи різні фактори, які необхідні для зручного переміщення людей.*	01.03.2025 – 10.06.2025

Завдання видано _____ *доц. Желдак Т.А.*
(підпис) (прізвище, ініціали)

Дата видачі: *06.12.2024 р.*
Дата подання до екзаменаційної комісії: _____
Прийнято до виконання _____ Колягін Д. С.
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Кваліфікаційна робота містить 82 сторінки, 4 таблиці, 26 рисунків, 2 додатки. Список використаних джерел нараховує 14 найменувань.

Мета кваліфікаційної роботи створення повнофункціонального back-end мікросервісу з використанням сучасних технологій (Java, Spring, Hibernate, JWT тощо), що забезпечує взаємодію з авіаційним API та реалізує основні бізнес-процеси онлайн-бронювання квитків.

Об'єкт дослідження – Процес розробки back-end мікросервісу для бронювання авіаквитків з використанням зовнішнього API та програмна реалізація веб-сервісу для бронювання авіаквитків.

Предмет дослідження – технічні та програмні аспекти реалізації функціонального back-end мікросервісу на сучасних Java-технологій.

Методи дослідження: У кваліфікаційній роботі використано сукупність теоретичних і практичних методів дослідження. Аналіз літературних джерел та документації дозволив обґрунтувати вибір технологій для розробки мікросервісу. Системний підхід застосовувався для визначення архітектури та функціональних компонентів сервісу.

Проєктування та моделювання використовувалися для створення структури бази даних і логіки взаємодії модулів.

Практична частина реалізовувалася за допомогою емпіричних методів — через тестування функціональності системи (JUnit, WireMock), перевірку безпеки, обробку помилок та роботу з API. Завершальним етапом був економічний аналіз, у якому за допомогою розрахунково-аналітичного методу визначено трудомісткість, витрати та кінцеву вартість розробки.

***Ключові слова*:** Java, Spring Boot, Hibernate, JWT, Swagger, Maven, Lombok, PostgreSQL, H2, REST API, JSON, DTO, WireMock, Logback, мікросервіс, авторизація, автентифікація, контролер, репозиторій, фасад, SOLID, тестування, собівартість, калькуляція, охорона праці.

ABSTRACT

The qualification work contains 82 pages, 4 tables, 26 figures, 2 appendices. The list of references includes 14 titles.

The purpose of the qualification work is the creation of a fully functional back-end microservice using modern technologies (Java, Spring, Hibernate, JWT, etc.) that provides interaction with an aviation API and implements the main business processes of online ticket booking.

The object of research is the process of developing a back-end microservice for booking airline tickets using an external API and the software implementation of a web service for booking airline tickets.

The subject of research is the technical and software aspects of implementing a functional back-end microservice with modern Java technologies.

Research methods: The qualification work uses a combination of theoretical and practical research methods. The analysis of literary sources and documentation allowed substantiating the choice of technologies for microservice development. A systems approach was applied to determine the architecture and functional components of the service.

Design and modeling were used to create the database structure and the logic of module interaction.

The practical part was implemented using empirical methods — through system functionality testing (JUnit, WireMock), security verification, error handling, and API interaction. The final stage was an economic analysis, in which the labor intensity, costs, and final development cost were determined using the calculation-analytical method.

Keywords: Java, Spring Boot, Hibernate, JWT, Swagger, Maven, Lombok, PostgreSQL, H2, REST API, JSON, DTO, WireMock, Logback, microservice, authorization, authentication, controller, repository, facade, SOLID, testing, cost price, calculation, occupational safety.

ЗМІСТ

ВСТУП

Мікросервісна архітектура є підхідом до розробки програмного забезпечення, що дає змогу створювати складні додатки, розбиваючи їх на невеликі незалежні компоненти, відомі як мікросервіси. Кожен мікросервіс виконує окрему функціональність і може працювати відокремлено від інших мікросервісів, спілкуючись з ними за допомогою мережевих інтерфейсів.

У моїй кваліфікаційній роботі я розробив мікросервіс, призначений для бронювання квитків на реальні авіарейси з зовнішнього API [1]. Цей мікросервіс забезпечує реєстрацію нових користувачів, автентифікацію та авторизацію, зберігання профілів користувачів, шукає авіарейси по фільтру, бронює квитки та дає можливість сплачувати платежі.

Для реалізації цього мікросервісу я використав мову Java та такі технології як: Apache Maven, Spring, Hibernate, JWT, Swagger, Lombok, Junit 5, WireMock, Logback. Також використав дві бази даних: PostgreSQL для продакшену та H2 для тестів.

Один з ключових аспектів мікросервісної архітектури - це розподіленість. Мій мікросервіс може працювати незалежно, запускаючись на своєму окремому сервері або контейнері. Він також може масштабуватись горизонтально, що дозволяє збільшити його продуктивність шляхом додавання більшої кількості екземплярів мікросервісу при збільшенні навантаження.

Для забезпечення комунікації між мікросервісами застосовується протокол HTTP, а дані передаються у форматі JSON. Завдяки цьому мікросервіси можуть легко взаємодіяти один з одним і передавати необхідну інформацію для виконання комплексних функцій.

Створення мікросервісу в рамках моєї кваліфікаційної роботи вимагало від мене не тільки знання програмування, але й розуміння принципів мікросервісної архітектури та здатність до системного аналізу та організації розробки. Цей досвід став важливим кроком у моєму професійному розвитку і підготував мене до роботи з сучасними технологіями у сфері розробки програмного забезпечення.

1 ІНФОРМАЦІЙНО-АНАЛІТИЧНИЙ РОЗДІЛ

1.1 Постановка завдання

Тема моєї кваліфікаційної роботи полягає в розробці мікросервісу, який призначений для бронювання квитків на реальні авіарейси за допомогою зовнішнього API [1]. Зовнішнє API потрібно для пошуку самих авіарейсів. Цей мікросервіс можна використовувати як back-end сервер в мікросервісній архітектурі. Він забезпечує реєстрацію нових користувачів, автентифікацію та авторизацію, ролі користувачів, зберігання профілів, пошук авіарейсів за заданим фільтром, можливість бронювання квитків та оплату платежів для них.

Так як це back-end мікросервіс, для роботи з його кінцевими точками найбільш зручним інтерфейсом є Swagger UI. Його документація відображає усі URL адреси у програмі, та усі моделі, які використовуються у контролерах. Сторінку документації можна побачити на рисунку 1.1.

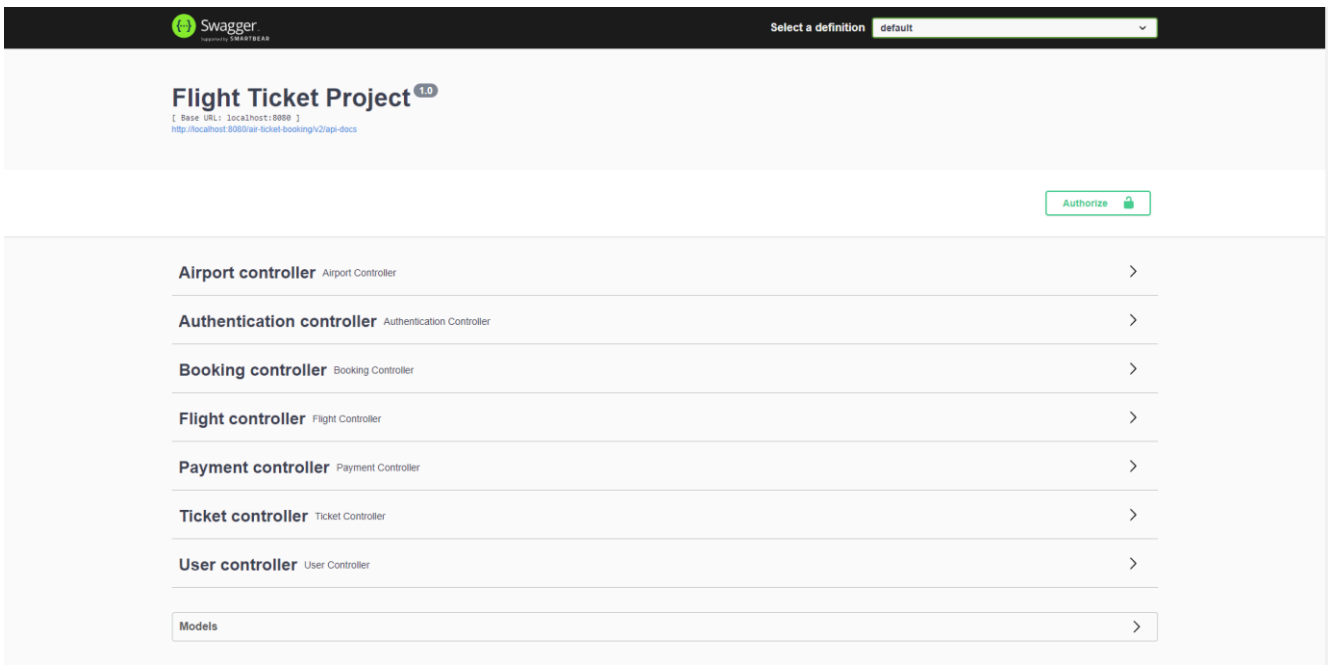


Рисунок 1.1 – Сторінка документації Swagger

1.2 Вибір програмного забезпечення

Для розробки проекту було використано об'єктно-орієнтовану мову програмування Java та такі технології як: Apache Maven, Spring, Hibernate, JWT, Swagger, Lombok, Junit 5, WireMock, Logback. Також використано дві бази даних: PostgreSQL для продакшену та H2 для тестів.

1.2.1 Об'єктно-орієнтована мова програмування Java

Java - Це сучасна мова програмування з об’єктно-орієнтованим підходом, створена компанією Sun Microsystems, яка пізніше стала частиною Oracle. Java відома тим, що її застосунки можна запускати на різних операційних системах, таких як Windows, macOS та Linux, без необхідності перекомпіляції [2].

Однією з ключових переваг Java є її сумісність із платформою Java Virtual Machine (JVM). Код, написаний на цій мові, перетворюється на байт-код, який можна запускати на будь-якому пристрої, де встановлена JVM, незалежно від операційної системи. Це дозволяє розробникам розповсюджувати свої програми безпосередньо у вигляді байт-коду, забезпечуючи високу переносимість і сумісність між різними платформами.

Java також володіє багатою екосистемою інструментів і бібліотек, які спрощують розробку програм і дозволяють використовувати готові компоненти для вирішення різних завдань. Наприклад, вона має велику стандартну бібліотеку, яка надає засоби для операцій введення-виведення, мережевого програмування, обробки рядків, роботи з базами даних та багато іншого.

Java Широко застосовується для створення вебзастосунків, мобільних програм (зокрема на базі Android), корпоративного програмного забезпечення та

різноманітних інших цифрових рішень. Вона пропонує розширені можливості для роботи з багатопотоковістю, безпекою, обробкою винятків та іншими важливими аспектами програмування. Завдяки своїй надійності, стабільній роботі та широкому розповсюдженню Java досі входить до переліку найпопулярніших мов програмування у світі. Вона має велике співтовариство розробників, активну підтримку та широку базу знань, що допомагає розробникам вирішувати проблеми та знаходити необхідну документацію.

1.2.2 Засіб автоматизації роботи Apache Maven

Apache Maven - це потужний інструмент для управління проектами і збирання програмного забезпечення в середовищі Java. Він надає автоматизований спосіб керування залежностями, компіляцією, пакуванням та розгортанням проектів [3].

Одним з головних принципів Maven є конвенція над конфігурацією. За допомогою стандартних структур каталогів та файлів, Maven забезпечує послідовність та легкість розгортання проектів. Це дозволяє розробникам швидше розпочати проект і швидше орієнтуватися в його структурі.

Maven використовує файл конфігурації pom.xml (Project Object Model), де вказується інформація про проект, його залежності, плагіни та різні конфігураційні параметри. Це дозволяє легко управляти проектом, його залежностями та процесами збирання.

Однією з ключових особливостей Maven є централізоване сховище залежностей – Maven Repository. Він містить широкий вибір бібліотек та компонентів, які можуть бути використані в проектах. Maven автоматично завантажує необхідні залежності з цього репозиторію, що спрощує управління залежностями та забезпечує їх актуальність [4].

1.2.3 Spring Framework

Spring Framework - це фреймворк розробки додатків на мові Java, який надає широкий спектр інструментів і компонентів для побудови сучасних, масштабованих та ефективних програмних рішень [5].

Spring Boot - це розширення Spring Framework, яке спрощує процес налаштування та розгортання додатків. Він надає конвенції над конфігурацією, що дозволяє розробникам швидко створювати самостійні додатки з мінімальними зусиллями.

Spring Web - модуль, який надає підтримку для розробки веб-додатків на основі Spring Framework. Він включає в себе інструменти для обробки HTTP-запитів, керування сесіями, обробки форм та інтеграції з веб-серверами.

Spring Rest - модуль, який допомагає розробляти RESTful API відповідно до архітектурного стилю REST. Він надає анотації та інструменти для створення легких, масштабованих та стандартизованих веб-сервісів.

Spring Data JPA - модуль, який спрощує взаємодію з базами даних за допомогою Java Persistence API (JPA). Він автоматично генерує SQL-запити на основі репозиторіїв та надає зручний спосіб роботи з об'єктно-реляційним відображенням (ORM).

Spring Security - модуль, який забезпечує механізми автентифікації, авторизації та захисту веб-додатків. Він дозволяє налаштовувати права доступу, керувати сесіями користувачів та забезпечувати безпеку даних.

Загалом, Spring Framework забезпечує велику гнучкість і розширюваність для розробників, дозволяючи створювати різноманітні додатки, від простих веб-сайтів до складних корпоративних систем. Він є одним з найпопулярніших фреймворків в сфері розробки на мові Java завдяки своїй потужності, співтовариству розробників та багатому набору функцій.

1.2.4 Hibernate ORM

Hibernate - це потужна бібліотека для об'єктно-реляційного відображення (ORM) у середовищі Java. Він надає зручний спосіб взаємодії з базами даних, де об'єкти Java можуть бути збережені та витягнуті з бази даних без необхідності писати складні SQL-запити вручну [6].

Hibernate дозволяє розробникам працювати з базами даних, використовуючи об'єктно-орієнтований підхід, що полегшує розробку і збереження даних. Він автоматично відображає об'єкти Java на відповідні таблиці бази даних і забезпечує механізми зчитування, запису та оновлення даних безпосередньо з об'єктів.

За допомогою Hibernate у Spring Data JPA, розробники можуть використовувати анотації та інші специфікації JPA для визначення сутностей, їх зв'язків та маппінгу на таблиці бази даних. Hibernate автоматично генерує SQL-запити та виконує їх для забезпечення збереження, зчитування та оновлення даних.

1.2.5 JWT

JWT, або "JSON Web Token", є стандартом обміну даними у форматі JSON, який використовується для передачі токенів між сторонами. Цей токен має компактну структуру і складається з трьох основних частин: заголовка (header), клейма (payload) і підпису (signature) [7].

Заголовок JWT містить метадані про тип токена та алгоритм шифрування, використовуваний для генерації підпису. Заголовок закодований у форматі Base64Url.

Клейм JWT містить корисну інформацію, таку як ідентифікатор користувача, дозволи або будь-які інші дані, які потрібні для ідентифікації або авторизації користувача. Клейм також закодований у форматі Base64Url.

Підпис JWT генерується за допомогою секретного ключа або приватного ключа, що підписує заголовок та клейм. Цей підпис перевіряється стороною, яка отримує токен, щоб забезпечити його цілісність і автентичність.

JWT має багато переваг, включаючи простоту використання, переносимість та безпеку. Він може бути використаний для автентифікації та авторизації користувачів у веб-додатках та API. Крім того, JWT дозволяє передавати додаткові дані разом з токеном, що робить його гнучким і розширюваним.

1.2.6 Специфікація OpenAPI Swagger

Swagger, також відомий як OpenAPI Specification, є набором інструментів і специфікацій для опису та документування RESTful API. Він дозволяє розробникам створювати, визначати та управляти API зрозумілою і стандартизованою манерою [8].

Swagger дозволяє описувати ресурси, операції та параметри API за допомогою читабельного формату YAML або JSON. Ця специфікація надає велику кількість деталей, таких як URL-шаблони, методи HTTP, параметри запиту, формати даних тощо, що допомагає розробникам та споживачам API зрозуміти його функціональність і використовувати його належним чином.

Він автоматично генерує інтерактивну документацію для API на основі наданої специфікації. Ця документація включає опис доступних шляхів, параметрів, відповідей та схем даних, що спрощує розуміння та спілкування з API для розробників та інших зацікавлених сторін.

Крім того, Swagger надає можливість виконувати тестування API безпосередньо з документації, включаючи надання зразків запитів та перевірку результатів. Це допомагає розробникам швидше розробляти та тестувати API, а також сприяє зменшенню помилок та полегшенню інтеграції з іншими системами.

1.2.7 Lombok

Lombok - це бібліотека для мови програмування Java, яка допомагає зменшити шаблонний та повторюваний код, спрощуючи процес розробки [9].

Lombok надає аннотації, які можна використовувати в класах Java для автоматичного створення стандартного коду, такого як геттери, сеттери, конструктори та інші методи доступу до полів. Замість того, щоб писати цей код вручну, розробники можуть просто додати анотацію Lombok до класу, і Lombok згенерує цей код автоматично під час компіляції. Це дозволяє розробникам сконцентруватися на основній логіки програми, замість того, щоб тратити час на писання однотипного коду. Крім того, використання Lombok допомагає зменшити кількість помилок, пов'язаних з ручним створенням такого типу коду.

Також він надає інші корисні функції, наприклад, можливість автоматично додавати анотації для обробки винятків, генерації методів toString(), equals() та hashCode(), а також для роботи з іммутабельними класами. Ці функції допомагають покращити продуктивність розробіфвки та якість коду.

1.2.8 Бібліотека для тестування програмного забезпечення JUnit 5

JUnit 5 - це фреймворк для тестування одиниць програмного забезпечення (unit testing) у мові Java, який надає потужні інструменти для створення та виконання тестів [10].

Фреймворк включає в себе багато нових функцій та поліпшень порівняно з попередніми версіями JUnit. Він підтримує розширену модель анотацій, що

дозволяє легко визначати й налаштовувати тести, включаючи фікстури перед і після тесту, передачу параметрів, декларативне тестування тощо.

JUnit 5 також має багате API, яке дозволяє розробникам писати тестові методи з використанням різних способів перевірки очікуваних результатів, включаючи стандартні перевірки рівності, порівняння зі значенням null, винятків тощо. Це забезпечує гнучкість і зручність при написанні тестів.

Ще одною корисною функцією JUnit 5 є розширення (extensions), які дозволяють розширити можливості фреймворку і використовувати спеціалізовані розширення для роботи з контекстом тестування, зовнішніми ресурсами, логуванням, покриттям коду тощо.

1.2.9 Бібліотека для тестування програмного забезпечення WireMock

WireMock - це бібліотека та сервер, які дозволяють створювати імітовані HTTP-сервери для тестування та симуляції зовнішніх API. Вона дозволяє розробникам легко створювати та налаштовувати штучні відповіді згідно з визначеними сценаріями [11].

Бібліотека надає можливість визначити очікувані запити, включаючи URL-шаблони, методи HTTP, заголовки та параметри. Розробники можуть задати відповіді, включаючи коди статусу, тіла відповіді та заголовки, які будуть повернуті. Це дозволяє точно налаштувати поведінку імітованого сервера під час тестування.

Крім того, WireMock має потужні можливості для валідації запитів, таких як перевірка наявності та значень заголовків, валідація тіла запиту за допомогою регулярних виразів або зіставлення з фіксованими значеннями. Це дозволяє перевірити, чи взаємодіє програма з зовнішніми сервісами правильно.

WireMock також підтримує запам'ятовування запитів, що дозволяє перевірити, скільки разів був зроблений певний запит, а також отримати доступ до його параметрів та деталей. Це допомагає розробникам виконувати більш складні перевірки та аналізувати взаємодію зовнішнього API.

1.2.10 Logback

Logback - це бібліотека для обробки логування в додатках, написаних на мові Java. Вона надає гнучкі налаштування, широкі можливості та високу продуктивність для реєстрації подій та відладки програмного забезпечення [12].

Logback підтримує різні рівні логування, такі як TRACE, DEBUG, INFO, WARN та ERROR, що дозволяє розробникам точно контролювати, які події включати до логів в залежності від режиму роботи програми. Це забезпечує гнучкість при відладці та моніторингу додатків.

Бібліотека також має розширену систему фільтрації та форматування логів. Розробники можуть налаштувати фільтри, які виключають або включають певні події в залежності від їх важливості або характеру. Крім того, Logback дозволяє налаштовувати формат виведення логів, включаючи дату, час, рівень логування, повідомлення та інші додаткові деталі.

Іншою важливою особливістю Logback є можливість налаштування різних апендерів (appenders), які визначають, куди будуть записуватися логи. Logback підтримує різні типи апендерів, такі як файловий апендер, консольний апендер, апендер для виводу на віддалений сервер і багато інших. Це дозволяє розробникам гнучко керувати місцем збереження та виведення логів.

1.2.11 Об'єктно-реляційна система керування базами даних PostgreSQL

PostgreSQL - це об'єктно-реляційна система керування базами даних (ОРСБД), яка надає широкий набір можливостей для зберігання та управління даними у програмах [13].

PostgreSQL відома своєю надійністю, стабільністю та високою продуктивністю. Вона підтримує ACID (Atomicity, Consistency, Isolation, Durability) властивості, що забезпечують надійність транзакцій та цілісність даних. База даних може обробляти великий обсяг даних та високе навантаження, забезпечуючи

швидкий доступ до інформації. Вона підтримує розширення SQL-стандарту, що дозволяє виконувати складні операції з даними, включаючи складні запити, підзапити, агрегацію та з'єднання таблиць. Вона також має розширені можливості для керування індексами, забезпечуючи швидкий пошук та фільтрацію даних.

PostgreSQL підтримує різні типи даних, включаючи числа, рядки, дати, часи, JSON, XML та багатовимірні геометричні об'єкти. Це робить її потужним інструментом для різних типів додатків, включаючи веб-додатки, аналітичні системи та геоінформаційні системи.

Також вона надає розширені можливості для управління безпекою даних, включаючи визначення ролей, управління доступом та шифрування. Вона підтримує автентифікацію засобами ОС, а також може працювати в режимі мережевого доступу, що дозволяє забезпечити безпеку даних у розподілених середовищах.

1.2.12 Кросплатформова СУБД H2

H2 - це легка вбудовувана система керування базами даних (СКБД), написана на мові Java. Вона надає можливість зберігати та управляти даними в програмах з низькими вимогами до продуктивності та масштабованості [14].

H2 пропонує широкі можливості для створення та управління реляційними базами даних. Вона підтримує різні типи даних, такі як числа, рядки, дати, часи, бінарні дані, XML та CLOB/BLOB. Вона також підтримує транзакційні операції з використанням ACID властивостей для забезпечення цілісності даних.

Одна з головних переваг H2 полягає у його портативності. Вона може бути вбудована безпосередньо в програму або працювати у режимі сервера, забезпечуючи доступ до бази даних через TCP/IP або вбудований режим. Це робить H2 зручним інструментом для розробки та тестування додатків, а також для невеликих проектів з обмеженими вимогами до продуктивності.

2 СПЕЦІАЛЬНИЙ РОЗДІЛ

2.1 Проектування бази даних

Розробка будь-якого back-end сервера починається з побудови моделей та бази даних. Спочатку я вибрав СКБД PostgreSQL, так як на мою думку вона має потужний інструментарій для реалізації бази даних. Потім я проаналізував відповіді від зовнішнього API і вирішив створити 4 таблиці, які будуть мати співвідношення між собою. Для візуального проектування таблиць у базі даних я використати сервіс drawSQL. У ньому можна швидко та зручно спроектувати таблиці так їх відношення між собою. Схему таблиць можна побачити на рисунку 2.1.

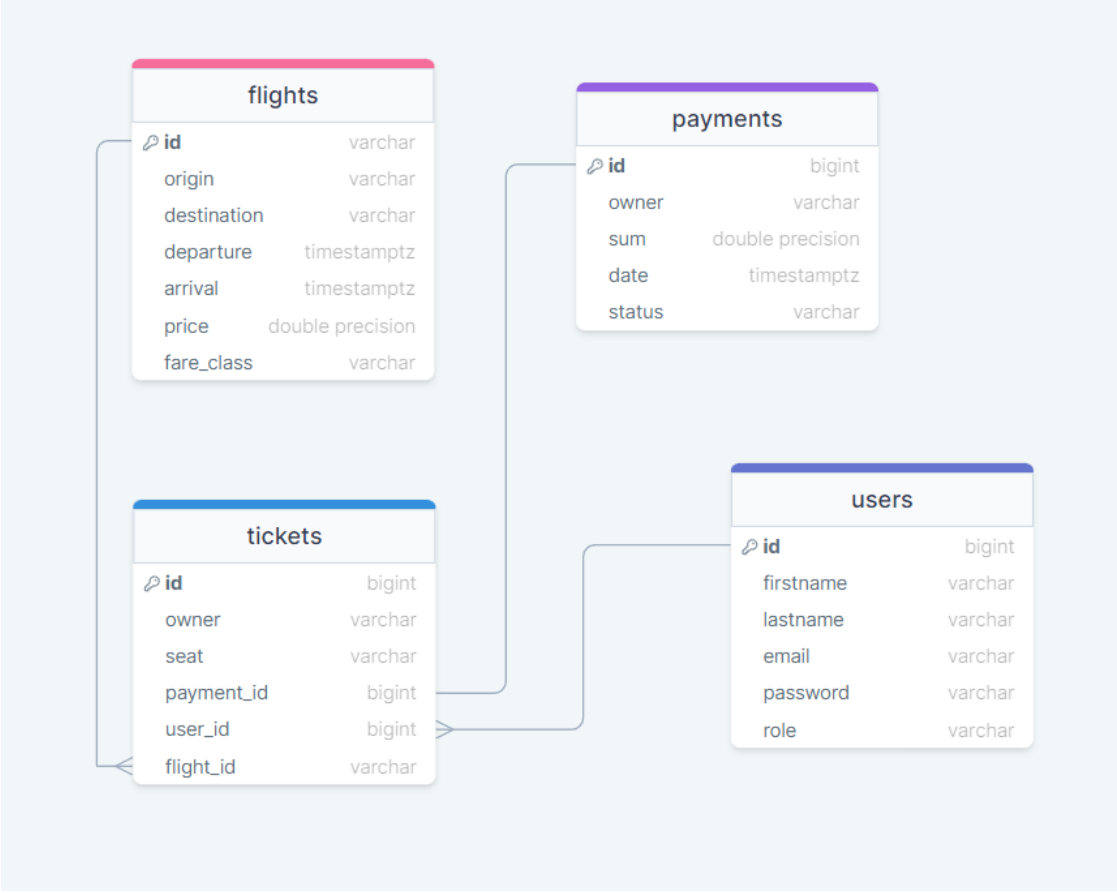


Рисунок 2.1 – Схема таблиць в базі даних

Основною таблицею для співвідношення буде «tickets», оскільки через неї можна отримати дані з будь якої таблиці для користувача. Таблиця «tickets» має поля «payment_id», «user_id» та «flight_id» які звязані з полями «id» у інших таблицях. Це зроблено для того, щоб створити такі співвідношення:

- one-to-one;
- one-to-many.

В one-to-one відношенні кожен запис в одній таблиці відповідає точно одному запису в іншій таблиці. Це означає, що існує пряма залежність між цими двома таблицями. Наприклад, у таблиці «tickets» поле «payment_id», яке посилається на унікальний ідентифікатор у таблиці «payments». Кожен квиток має лише один платіж, і навпаки, тому для кожного запису в таблиці «tickets» існує тільки один відповідний запис в таблиці «payments».

У one-to-many відношенні кожен запис в одній таблиці може мати безліч записів у зв'язаній таблиці. Це означає, що записи в першій таблиці посилаються на записи в другій таблиці за допомогою зовнішнього ключа. Наприклад, у таблиці «tickets» поле «flight_id», яке посилається на унікальний ідентифікатор авіарейсу у таблиці «flights». Оскільки у квитка може бути всього один рейс, а у рейса безліч квитків.

Після створення схеми я вирішив створити саму базу даних та ці таблиці. Спочатку треба створити сервер для бази даних, використавши pgAdmin. Вікно створення можна побачити на рисунку 2.2.

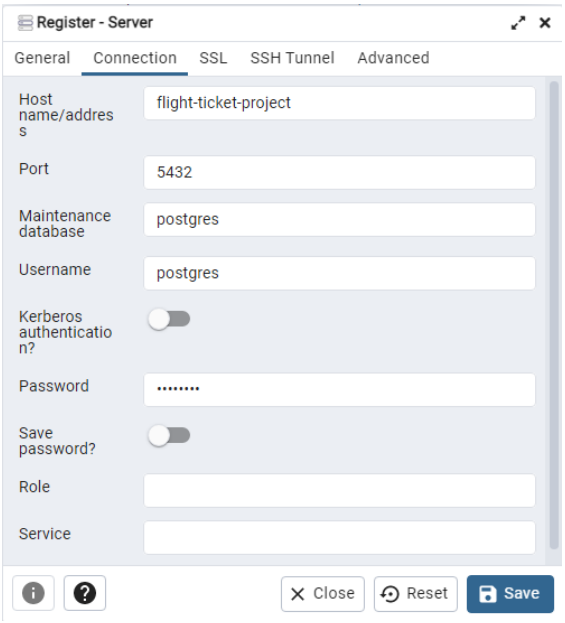
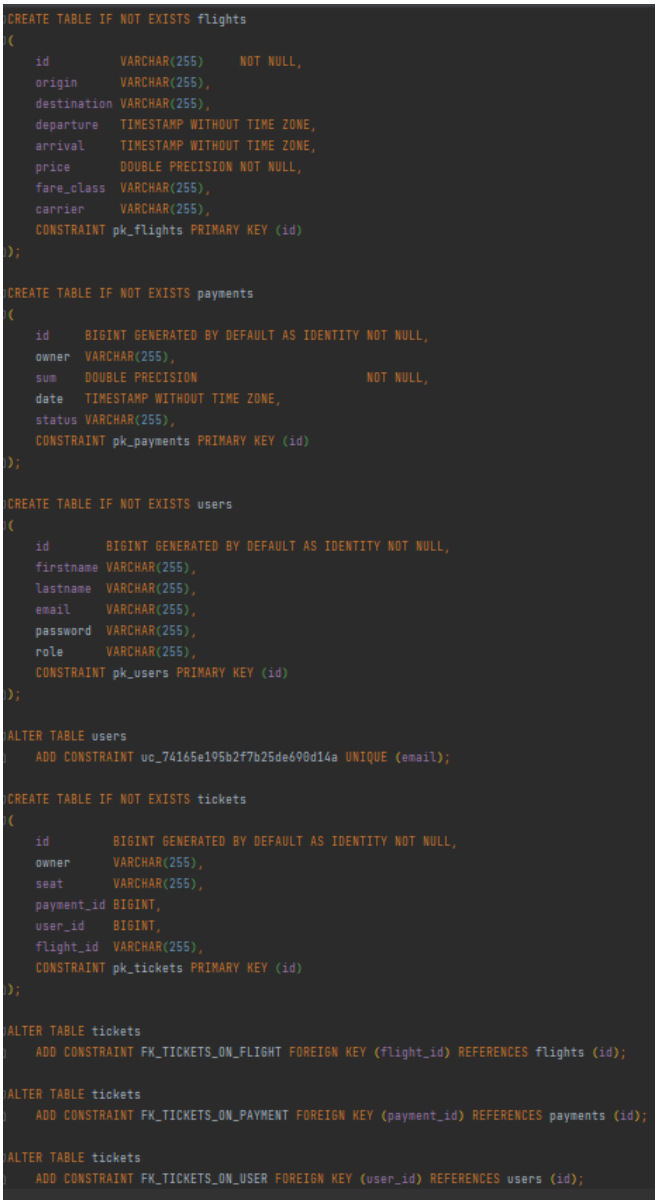


Рисунок 2.2 – Вікно створення сервера PostgreSQL

Сервер потрібен для того, щоб створити в ньому базу даних та приєднатися до нього через IntelliJ IDEA — інтегроване середовище розробки. Це потрібно для того, щоб виконувати SQL запити через середовище розробки.

SQL (Structured Query Language) – це мова програмування, що використовується для роботи з реляційними базами даних. Він дозволяє виконувати операції зі створення, модифікації та вилучення даних, а також управляти структурою та відносинами між таблицями.

Приєднавшись до серверу можна виконати скрипт «schema.sql», який знаходиться в каталогу ресурсів проекту. Він створить усі таблиці та відношення між ними. Зміст скрипта можна побачити на рисунку 2.3.



```
CREATE TABLE IF NOT EXISTS flights
(
    id          VARCHAR(255)     NOT NULL,
    origin      VARCHAR(255),
    destination VARCHAR(255),
    departure   TIMESTAMP WITHOUT TIME ZONE,
    arrival     TIMESTAMP WITHOUT TIME ZONE,
    price       DOUBLE PRECISION NOT NULL,
    fare_class  VARCHAR(255),
    carrier     VARCHAR(255),
    CONSTRAINT pk_flights PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS payments
(
    id     BIGINT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
    owner  VARCHAR(255),
    sum    DOUBLE PRECISION                       NOT NULL,
    date   TIMESTAMP WITHOUT TIME ZONE,
    status VARCHAR(255),
    CONSTRAINT pk_payments PRIMARY KEY (id)
);

CREATE TABLE IF NOT EXISTS users
(
    id        BIGINT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
    firstname VARCHAR(255),
    lastname  VARCHAR(255),
    email     VARCHAR(255),
    password  VARCHAR(255),
    role      VARCHAR(255),
    CONSTRAINT pk_users PRIMARY KEY (id)
);

ALTER TABLE users
    ADD CONSTRAINT uc_74165e195b2f7b25de690d14a UNIQUE (email);

CREATE TABLE IF NOT EXISTS tickets
(
    id         BIGINT GENERATED BY DEFAULT AS IDENTITY NOT NULL,
    owner      VARCHAR(255),
    seat       VARCHAR(255),
    payment_id BIGINT,
    user_id    BIGINT,
    flight_id  VARCHAR(255),
    CONSTRAINT pk_tickets PRIMARY KEY (id)
);

ALTER TABLE tickets
    ADD CONSTRAINT FK_TICKETS_ON_FLIGHT FOREIGN KEY (flight_id) REFERENCES flights (id);

ALTER TABLE tickets
    ADD CONSTRAINT FK_TICKETS_ON_PAYMENT FOREIGN KEY (payment_id) REFERENCES payments (id);

ALTER TABLE tickets
    ADD CONSTRAINT FK_TICKETS_ON_USER FOREIGN KEY (user_id) REFERENCES users (id);
```

Рисунок 2.3 – Скрипт створення таблиць

Оскільки сервер бази даних та таблиці створені, можна виконати наступний скрипт «data.sql», який додасть начальні дані до моїх таблиць. Зміст скрипта можна побачити на рисунку 2.4.

```sql
INSERT INTO flights (id, origin, destination, departure, arrival, price, fare_class, carrier)
VALUES  no usages
       ('C1', 'London', 'New-York', NOW() + INTERVAL '1' DAY, NOW() + INTERVAL '2' DAY, 500.50, 'ECONOMY', 'Iberia Express'),
       ('2', 'Berlin', 'Paris', NOW() + INTERVAL '2' DAY, NOW() + INTERVAL '3' DAY, 700, 'ECONOMY', 'British Airways'),
       ('3', 'Barcelona', 'Rome', NOW() + INTERVAL '3' DAY, NOW() + INTERVAL '4' DAY, 200.25, 'ECONOMY', 'Iberia Express');

INSERT INTO payments (owner, sum, date, status)  no usages
VALUES  no usages
       ('Danil Admin', 700, NOW(), 'NEW'),
       ('Danil Koliahin', 500.50, NOW(), 'DONE'),
       ('Danil User', 200.25, NOW(), 'ARCHIVE');

INSERT INTO users (firstname, lastname, email, password, role)  no usages
VALUES  no usages
       ('Danil', 'Admin', 'admin@gmail.com', '$2a$10$Yd44hCeBpiEXsafVDuJ.Je3P3fxK/aEH4v/OEaSYWEL54cgURUzg2', 'ADMIN'),
       ('Danil', 'User', 'user@gmail.com', '$2a$10$LtZIU.WVSx952CRKqmhbqeEB8NiWWCH7i5HTLBLhFkfIEUJKm25FW', 'USER');

INSERT INTO tickets (owner, seat, payment_id, user_id, flight_id)  no usages
VALUES  no usages
       ('Danil Koliahin', 'F10', 2, 1, 2),
       ('Danil Koliahin', 'F11', 1, 1, 2),
       ('Danil User', 'F12', 3, 2, 3);
```

Рисунок 2.4 – Скрипт заповнення таблиць

2.2 Розробка back-end серверу

Розробка сервера розпочинається з налаштування залежностей та плагінів. Для цього у файл pom.xml треба описати всі залежності між бібліотеками, які будуть використані у проекті (рисунок 2.5). Також через цей файл треба налаштувати версії бібліотек, JDK та плагіни. Після додавання залежностей Maven завантажить їх з свого сховища.

```xml
<properties>
    <java.version>17</java.version>
    <springfox.version>3.0.0</springfox.version>
    <jwt.version>0.11.5</jwt.version>
    <wiremock.version>2.35.0</wiremock.version>
    <postgresql.version>42.5.1</postgresql.version>
</properties>

<dependencies>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
```

Рисунок 2.5 – Вміст файлу pom.xml

Завжди після додавання залажностей треба налаштувати файл application.yml у каталозі «resources» (рисунок 2.6). Через цей файл налаштовується увесь проект, включаючи: яку базу даних буде використовувати Hibernate та діалект SQL, контекстний шлях для сервера, різні значення змінних які можуть змінюватися та інші налаштування. Наприклад, секретний ключ, який використовується в реалізації JWT.

```yaml
## Swagger
spring:
  mvc:
    pathmatch:
      matching-strategy: ANT_PATH_MATCHER

  ## Postgresql Datasource
  datasource:
    url: jdbc:postgresql://localhost:5432/flight_ticket_project
    username: postgres
    password: postgres
    driver-class-name: org.postgresql.Driver

  ## JPA
  jpa:
    database-platform: org.hibernate.dialect.PostgreSQLDialect
    defer-datasource-initialization: true
    show-sql: true

## Goflightlabs
url:
  access-key:
  base: https://app.goflightlabs.com

## Server
server:
  servlet:
    context-path: /air-ticket-booking

## Security secret key
jwt:
  secret:
    key: 7639792442264528482B4D6251655468576D5A7134743777217A25432A462D4A
```

Рисунок 2.6 – Вміст файлу application.yml

Файл logback-spring.xml слугує для налаштування логів (журналу) у проекті. Через нього я налаштував інший колір для них, який буде відображатися в консолі запуску, рівень відображення, та файл, в який вони будуть зберігатися. Це було

зроблено для того, щоб розробники могли слідкувати за роботою мікросервісу, та дивитися історію роботи. Окрім того, логі допомагають відслідковувати помилки у проекті. Вміст можна побачити на рисунку 2.7.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <property name="LOGS" value="./logs"/>

    <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
        <layout class="ch.qos.logback.classic.PatternLayout">
            <Pattern>
                %white(%d{ISO8601}) %highlight(%-5level) [%blue(%t)] %yellow(%C{1.}): %msg%n%throwable
            </Pattern>
        </layout>
    </appender>

    <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>./logs/FlightApi.log</file>
        <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
            <Pattern>%d %p %C{1.} [%t] %m%n</Pattern>
        </encoder>

        <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${LOGS}/archived/spring-boot-logger-%d{yyyy-MM-dd}.%i.log
            </fileNamePattern>
            <timeBasedFileNamingAndTriggeringPolicy class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>10MB</maxFileSize>
            </timeBasedFileNamingAndTriggeringPolicy>
        </rollingPolicy>
    </appender>

    <root level="INFO">
        <appender-ref ref="FILE"/>
        <appender-ref ref="CONSOLE"/>
    </root>

</configuration>
```

Рисунок 2.7 – Вміст файлу logback-spring.xml

Після налаштування базових речей у проекті я приступив до створення сек'юріті. Перше що я зробив це створив сутність «User», яка імплементує інтерфейс Spring Framework «UserDetails». Імплементація потрібна для того, щоб Spring розумів, що цей об'єкт буде використовуватися у Spring Security. Також з імплементацією потрібно реалізувати методи цього інтерфейсу, завдяки яким можна буде отримати ролі користувача, статус його профіля та його пошту.

На рисунку 2.8 можна побачити клас «User» та анотації із бібліотеки javax.presistence та Lombok. Анотації javax.presistence слугують для налаштування

сутності в Hibernate. При запуску серверу, Hibernate буде розуміти, що цей клас є сутністю «@Entity», яка належить до таблиці «@Table(name = "users", uniqueConstraints = @UniqueConstraint(columnNames = "email"))» з унікальним значенням пошти. Анотації «@Column» та «@OneToMany» прив'язують поля класу до полів у таблиці, а також показують яке відношення у них з іншими сутностями. В даному випадку відношення з сутністю «Ticket». Анотації з бібліотеки Lombok «@Data», «@AllArgsConstructor», «@NoArgsConstructor» та «@Builder» потрібні для того, щоб генерувати конструктори та шаблонний код при компіляції серверу.

```java
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Table(name = "users", uniqueConstraints = @UniqueConstraint(columnNames = "email"))
public class User implements UserDetails {

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

    2 usages
    @Column(name = "firstname")
    private String firstname;

    2 usages
    @Column(name = "lastname")
    private String lastname;

    3 usages
    @Column(name = "email")
    private String email;

    2 usages
    @Column(name = "password")
    private String password;

    3 usages
    @Enumerated(EnumType.STRING)
    @Column(name = "role")
    private UserRole role;

    2 usages
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL)
    private List<Ticket> tickets = new ArrayList<>();
```

Рисунок 2.8 – Клас «User» та його поля

Для того щоб можна було отримувати ці сутності з бази даних, я створив інтерфейс репозіторія, який успадковує інтерфейс «JpaRepository». Spring Data JPA автоматично згенерує шаблонний код для цього репозіторія, все що залишається, це створити потрібні методи та написати через анотації SQL запити. Приклад інтерфейсу «UserRepository» можна побачити на рисунку 2.9.

```
public interface UserRepository extends JpaRepository<User, Long> {


    @Query(value = "SELECT * FROM users WHERE email = ?", nativeQuery = true)
    Optional<User> findByEmail(String email);
```

Рисунок 2.9 – Інтерфейс репозіторія «UserRepository»

Після цього я створив інтерфейс «UserService» та його імплементацію «UserServiceImpl», для того щоб створити проміжний шар в архітектурі. Інтерфейс вказує на те, яку поведінку повинна мати реалізація. Цей приклад відноситься до принципу об'єктно-орієнтованого програмування SOLID.

SOLID - це абревіатура, що представляє п'ять принципів об'єктно-орієнтованого програмування, розроблених Робертом Мартіном. Ці принципи допомагають створювати гнучкі, розширювані та підтримувані програмні системи. В даному випадку використано принцип розподілу інтерфейсу (Interface Segregation Principle, ISP). Клієнти не повинні залежати від інтерфейсів, які вони не використовують. Краще мати багато невеликих спеціалізованих інтерфейсів, ніж один загальний інтерфейс. Також згідно цього принципу, якщо далі у розробці потрібно буде замінити саму реалізацію інтерфейсу, треба буде просто створити новий клас, а інтерфейс залишиться незмінним. Інтерфейс та його реалізацію можна побачити на рисунках 2.10 та 2.11.

```
10 usages   1 implementation
public interface UserService {

    1 implementation
    List<User> findAll();

    1 implementation
    User findById(Long userId);

    1 implementation
    User save(User user);

    1 implementation
    void update(User user);

    1 implementation
    User findByEmail();
}
```

Рисунок 2.10 – Інтерфейс сервісу «UserService»

```
@RequiredArgsConstructor
@Service
public class UserServiceImpl implements UserService {
    5 usages
    private final UserRepository userRepository;


    @Override
    public List<User> findAll() {
        return userRepository.findAll();
    }


    @Override
    public User findById(Long userId) {
        return userRepository
                .findById(userId)
                .orElseThrow(() -> new ResourceNotFound("User with ID = " + userId + " not found"));
    }


    @Override
    public User save(User user) {
        return userRepository.save(user);
    }


    @Override
    public void update(User user) {
        userRepository.save(user);
    }


    @Override
    public User findByEmail() {
        return userRepository
                .findByEmail(JwtAuthenticationFilter.getCurrentUserEmail())
                .orElseThrow(() -> new ResourceNotFound("User not found"));
    }
}
```

Рисунок 2.11 – Реалізація сервісу «UserServiceImpl»

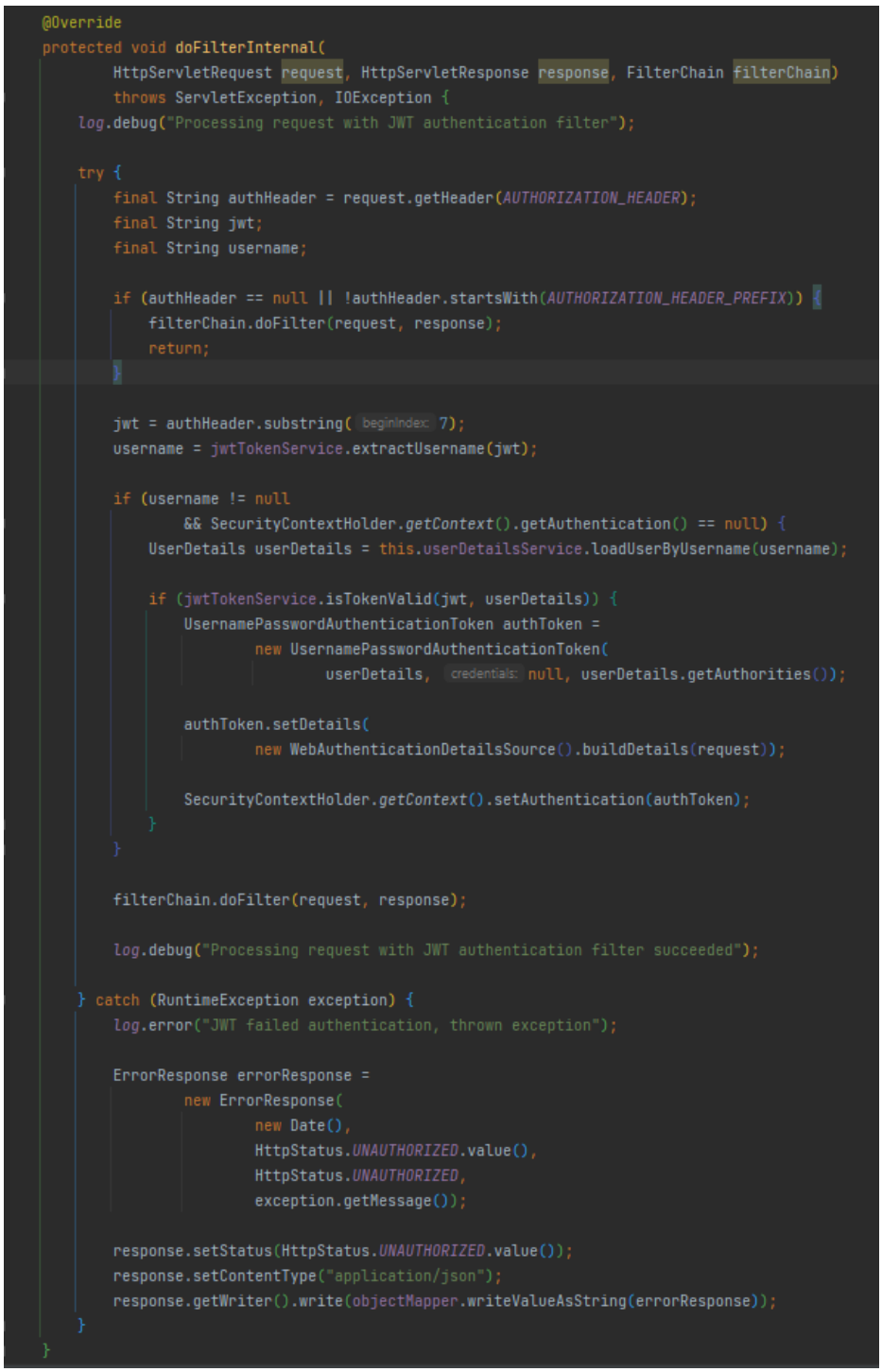
Далі я створив клас «JwtTokenService» який потрібен для самої реалізації сек'юріті. Методи цього класу генерують токін для користувача, налаштовують його термін, перевіряють термін та вміст самого токіну (рисунок 2.12). Цей клас використовується у фільтрі, який ловить усі запити до мікросервісу, аналізує користувача по токіну, та вирішує давати йому доступ чи ні. Сам токін дійсний всього одну добу, тому користувач повинен повторно авторизуватися, якщо термін токіну минув. Якщо перевірити отриманий токін після авторизації через сервіс jwt.io, можна побачити яке корисне навантаження він зберігає у себе (рисунок 2.13). Наприклад, на цьому рисунку можна побачити пошту, з якою авторизувався користувач.

```
@Service
public class JwtTokenService {

    1 usage
    @Value("763979244226452848 2B4D6251655468576D5A71347437772 17A25432...")
    private String secretKey;

    1 usage
    private static final long TOKEN_VALIDITY = 1000 * 60 * 24;

    5 usages  Andrey Kravchenko
    public String generateToken(UserDetails userDetails) {
        return generateToken(new HashMap<>(), userDetails);
    }

    1 usage  Andrey Kravchenko
    private String generateToken(Map<String, Object> extraClaims, UserDetails userDetails) {
        return Jwts.builder()
                .setClaims(extraClaims)
                .setSubject(userDetails.getUsername())
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + TOKEN_VALIDITY))
                .signWith(getSignInKey(), SignatureAlgorithm.HS256)
                .compact();
    }

    2 usages  Andrey Kravchenko
    private Key getSignInKey() {
        byte[] keyBytes = Decoders.BASE64.decode(secretKey);
        return Keys.hmacShaKeyFor(keyBytes);
    }

    7 usages  Andrey Kravchenko
    public boolean isTokenValid(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername())) && !isTokenExpired(token);
    }
```

Рисунок 2.12 – Реалізація сервісу «JwtTokenService»

Рисунок 2.13 – Корисна нагрузка токену

Метод фільтра «JwtAuthenticationFilter» перевіряє перед кожним запитом чи існує такий користувач у системі, перевіряє токін на валідність та дає доступ, в залежності від ролі користувача. Токін береться з кожного хедеру запита, він передається приховано для користувача. Також, якщо токін буде з минувшим терміном, метод зловить виняток, та надішле користувачу повідомлення. Цей метод можна побачити на рисунку 2.14.



```
@Override
protected void doFilterInternal(
        HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {
    log.debug("Processing request with JWT authentication filter");

    try {
        final String authHeader = request.getHeader(AUTHORIZATION_HEADER);
        final String jwt;
        final String username;

        if (authHeader == null || !authHeader.startsWith(AUTHORIZATION_HEADER_PREFIX)) {
            filterChain.doFilter(request, response);
            return;
        }

        jwt = authHeader.substring( beginIndex: 7);
        username = jwtTokenService.extractUsername(jwt);

        if (username != null
                && SecurityContextHolder.getContext().getAuthentication() == null) {
            UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

            if (jwtTokenService.isTokenValid(jwt, userDetails)) {
                UsernamePasswordAuthenticationToken authToken =
                        new UsernamePasswordAuthenticationToken(
                                userDetails,  credentials: null, userDetails.getAuthorities());

                authToken.setDetails(
                        new WebAuthenticationDetailsSource().buildDetails(request));

                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }

        filterChain.doFilter(request, response);

        log.debug("Processing request with JWT authentication filter succeeded");

    } catch (RuntimeException exception) {
        log.error("JWT failed authentication, thrown exception");

        ErrorResponse errorResponse =
                new ErrorResponse(
                        new Date(),
                        HttpStatus.UNAUTHORIZED.value(),
                        HttpStatus.UNAUTHORIZED,
                        exception.getMessage());

        response.setStatus(HttpStatus.UNAUTHORIZED.value());
        response.setContentType("application/json");
        response.getWriter().write(objectMapper.writeValueAsString(errorResponse));
    }
}
```

Рисунок 2.14 – Метод фільтру «JwtAuthenticationFilter»

Після цього я створив конфігурацію самого сек'юріті. В конфігурації я налаштував кінцеві точки доступу для користувачів по їх ролям, додав фільтр для токіну, створив реалізацію біна «AuthenticationProvider», для того щоб Spring розумів мої налаштування автентифікації користувача.

З такими налаштуваннями сек'юріті, користувач, коли спробую отримати доступ без автентифікації, отримає відмову у доступі. Потім, в залежності від ролі, він зможе виконувати різні дії. Якщо це «admin», він зможе отримати доступ к усім кінцевим точкам серверу, тобто: переглянути усіх існуючих користувачів, усі авіарейси, квитки, платежі тощо. Користувач з роллю «user» може переглядати свої дані, шукати рейси, бронювати квитки та сплачувати платежі. Цю конфігурацію можна побачити на рисунку 2.15.

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http.csrf()
            .disable()
            .authorizeHttpRequests()
            .antMatchers(
                    "/api/authentication/**",
                    "/v2/api-docs",
                    "/configuration/ui",
                    "/swagger-resources/**",
                    "/configuration/security",
                    "/swagger-ui/**",
                    "/webjars/**")
            .permitAll()
            .antMatchers(
                    "/api/airports/**",
                    "/api/booking/**",
                    "/api/flights/search",
                    "/api/flights/user",
                    "/api/payments/execute",
                    "/api/payments/user",
                    "/api/tickets/user",
                    "/api/users/info")
            .hasAnyAuthority("USER", "ADMIN")
            .antMatchers("/api/**")
            .hasAuthority("ADMIN")
            .anyRequest()
            .authenticated()
            .and()
            .sessionManagement()
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class);

    return http.build();
}


@Bean
public AuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
    authProvider.setUserDetailsService(userDetailsService);
    authProvider.setPasswordEncoder(passwordEncoder());
    return authProvider;
}
```

Рисунок 2.15 – Конфігурація сек'юріті

Для того, щоб користувач міг зареєструватися та авторизуватися, я створив сервіс «AuthenticationServiceImpl», який має два методи бізнес логіки для цих процесів. Кожен з методів приймає як параметр та повертає об'єкти DTO. Методи повернуть JWT токін користувача, який він зможе використати для доступу у додатку. Ці методи можна побачити на рисунку 2.16.

DTO (Data Transfer Object) – це шаблон проектування, який використовується для передачі даних між шарами чи компонентами у додатку. Основна мета DTO - спростити та стандартизувати передачу даних, зменшити кількість запитів та покращити продуктивність. DTO є простим контейнером даних, який містить кілька полів і відповідні методи доступу до цих полів. Він часто використовується для організації та передачі даних між клієнтською та серверною частинами програми, а також між різними шарами програми, такими як шар уявлення, шар сервісу та шар доступу до даних.

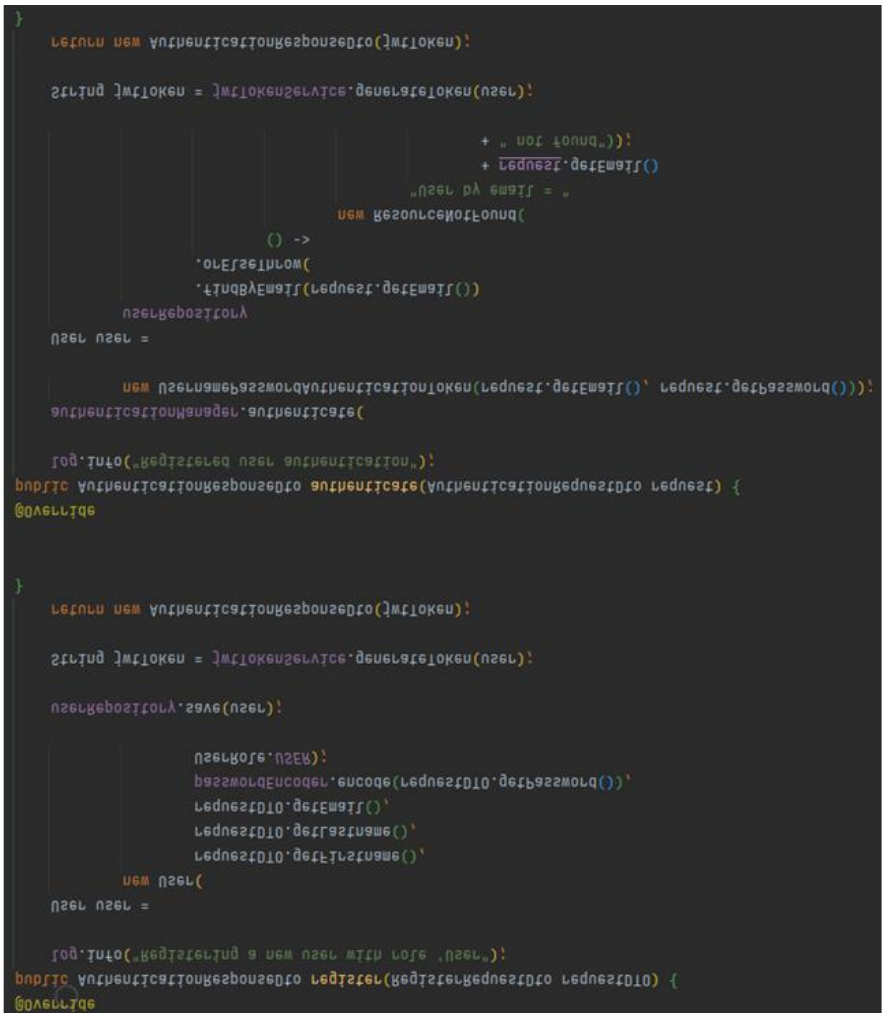


Рисунок 2.16 – Методи сервісу автентифікації

Останнім класом в сек'юріті є кінцева точка у вигляді контролера (рисунок 2.17). У класі «AuthenticationController» я створив два методи, які відправляють POST запити, через які і відбувається автентифікація користувачів. Також в класі налаштовані URL адреса для запитів, валідація запитів для повернення користувачу помилок та те, що цей контролер відправляє і приймає запити у вигляді JSON.

JSON (JavaScript Object Notation) - це формат обміну даними, що базується на синтаксисі об'єктів JavaScript. Він використовується для передачі структурованих даних між клієнтом та сервером по мережі. JSON є текстовим форматом, який легко читається і створюється як людиною, так і комп'ютером. Він складається з пар ключ-значення і може містити списки, рядки, числа і вкладені об'єкти і масиви. JSON широко застосовується у веб-розробці, API та інших додатках, де потрібна передача та зберігання даних у зручному та універсальному форматі.

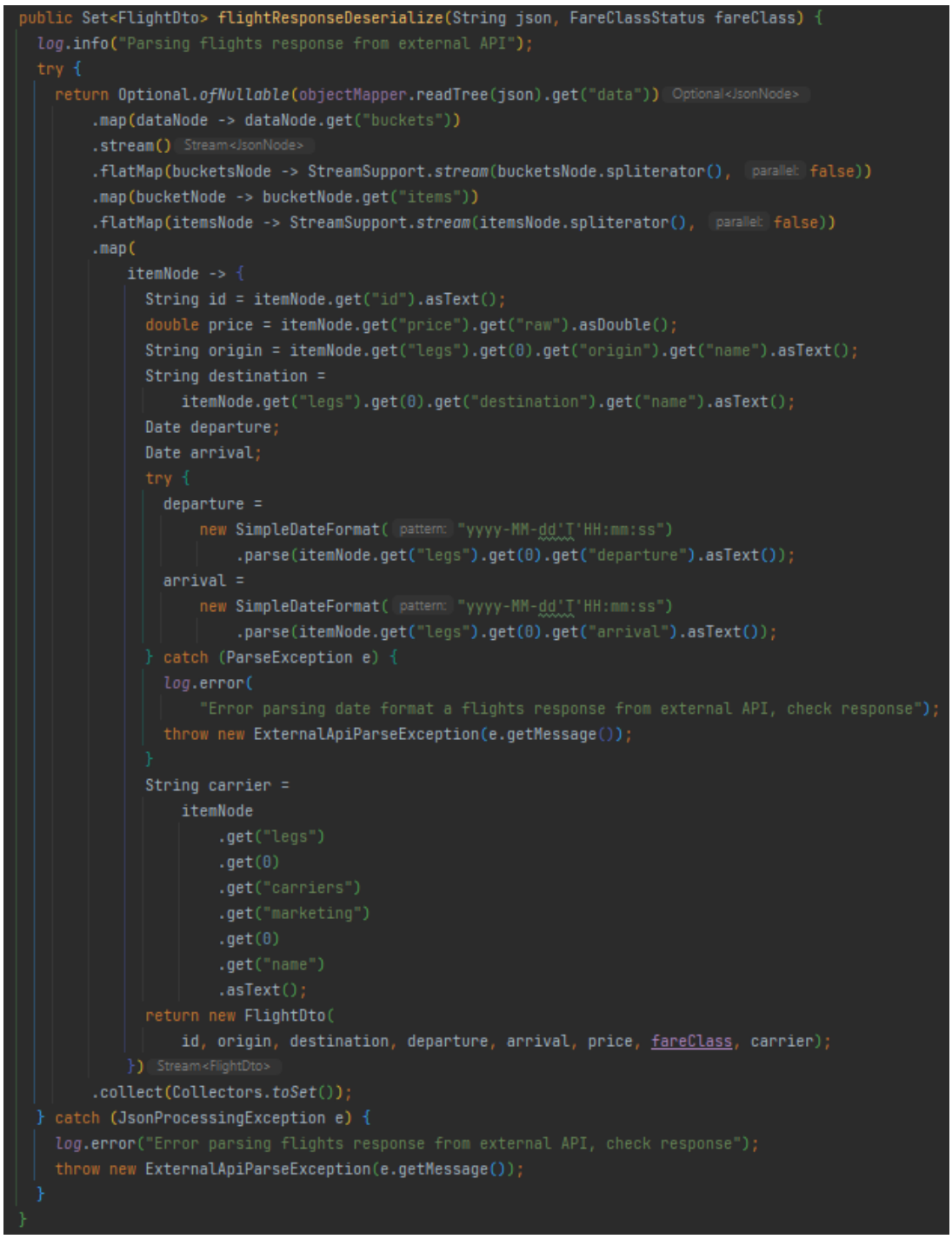
```java
@RequiredArgsConstructor
@Api(tags = "Authentication controller")
@RequestMapping("/api/authentication")
@RestController
@Slf4j
public class AuthenticationController {

    2 usages
    private final AuthenticationService authenticationService;

    Andrey Kravchenko
    @PostMapping("/register")
    public ResponseEntity<AuthenticationResponseDto> register(
            @RequestBody @Valid RegisterRequestDto requestDto) {
        log.info("Received request to register new user");

        AuthenticationResponseDto responseDto = authenticationService.register(requestDto);

        return ResponseEntity.ok(responseDto);
    }

    Andrey Kravchenko
    @PostMapping("/authenticate")
    public ResponseEntity<AuthenticationResponseDto> authenticate(
            @RequestBody @Valid AuthenticationRequestDto requestDto) {
        log.info("Received request to authenticate user");

        AuthenticationResponseDto responseDto = authenticationService.authenticate(requestDto);

        return ResponseEntity.ok(responseDto);
    }
}
```

Рисунок 2.17 – Контролер автентифікації

Процес розробки функціоналу пошуку авіарейсів починається з класу «GoflightlabsClientService», метод якого шукає рейси в зовнішньому API. Спочатку створюється URL запит, через фільтр пошуку користувача, потім об’єкт «RestTemplate» виконує запит і отримує строку відповіді. Цю строку десеріалізує мій клас «GoflightlabsResponseCustomDeserializer», він перетворює її на колекцію «FlightDto», так як основна сутність «Flight» має поля, які не потрібен бачити користувач. Метод сервіс класу та десеріалізатор можна побачити на рисунках 2.18 та 2.19.

```
public Set<FlightDto> findFlightsByFilter(
        String adults,
        String origin,
        String destination,
        String departureDate,
        String fareClass) {
    log.info(
            "Finding flights in Goflightlabs by filter: adults = {}, origin = {}, destination = {}, departureDate = {}, fareClass = {}",
            adults,
            origin,
            destination,
            departureDate,
            fareClass);

    String path =
            "/search-best-flights?access_key={accessKey}&adults={adults}&origin={origin}&destination={destination}&departureDate={departureDate}&cabinClass={fareClass}";

    Map<String, String> params =
            Map.of(
                    k1: "accessKey", accessKey,
                    k2: "adults", adults,
                    k3: "origin", origin,
                    k4: "destination", destination,
                    k5: "departureDate", departureDate,
                    k6: "fareClass", fareClass.toLowerCase());

    String url =
            UriComponentsBuilder.fromHttpUrl(baseUrl) UriComponentsBuilder
                    .path(path)
                    .buildAndExpand(params) UriComponents
                    .toUriString();

    String json = restTemplate.getForObject(url, String.class);

    return customDeserializer.flightResponseDeserialize(
            json, FareClassStatus.valueOf(fareClass.toUpperCase()));
}
```

Рисунок 2.18 – Метод пошуку авіарейсів



```
public Set<FlightDto> flightResponseDeserialize(String json, FareClassStatus fareClass) {
  log.info("Parsing flights response from external API");
  try {
    return Optional.ofNullable(objectMapper.readTree(json).get("data")) Optional<JsonNode>
        .map(dataNode -> dataNode.get("buckets"))
        .stream() Stream<JsonNode>
        .flatMap(bucketsNode -> StreamSupport.stream(bucketsNode.spliterator(), parallel: false))
        .map(bucketNode -> bucketNode.get("items"))
        .flatMap(itemsNode -> StreamSupport.stream(itemsNode.spliterator(), parallel: false))
        .map(
            itemNode -> {
              String id = itemNode.get("id").asText();
              double price = itemNode.get("price").get("raw").asDouble();
              String origin = itemNode.get("legs").get(0).get("origin").get("name").asText();
              String destination =
                  itemNode.get("legs").get(0).get("destination").get("name").asText();
              Date departure;
              Date arrival;
              try {
                departure =
                    new SimpleDateFormat( pattern: "yyyy-MM-dd'T'HH:mm:ss")
                        .parse(itemNode.get("legs").get(0).get("departure").asText());
                arrival =
                    new SimpleDateFormat( pattern: "yyyy-MM-dd'T'HH:mm:ss")
                        .parse(itemNode.get("legs").get(0).get("arrival").asText());
              } catch (ParseException e) {
                log.error(
                    "Error parsing date format a flights response from external API, check response");
                throw new ExternalApiParseException(e.getMessage());
              }
              String carrier =
                  itemNode
                      .get("legs")
                      .get(0)
                      .get("carriers")
                      .get("marketing")
                      .get(0)
                      .get("name")
                      .asText();
              return new FlightDto(
                  id, origin, destination, departure, arrival, price, fareClass, carrier);
            }) Stream<FlightDto>
        .collect(Collectors.toSet());
  } catch (JsonProcessingException e) {
    log.error("Error parsing flights response from external API, check response");
    throw new ExternalApiParseException(e.getMessage());
  }
}
```

Рисунок 2.19 – Метод десеріалізації

Після того, як авіарейси були знайдені, відповідь повертається в контролер «FlightController» (рисунок 2.20), де всі знайдені рейси зберігаються в створений мною кеш.

Кеш був створений через графік роботи метода сервісу (schedule). Кожні три хвилини в таблиці «flights» видаляються всі авіарейси, на які не було заброньовано жодного квитка. Це було зроблено для того, щоб користувач бронював далі квиток

на рейс через його id, в іншому контролеру. Метод графіку можна побачити на рисунку 2.21.

```
@GetMapping("/search")
public ResponseEntity<Set<FlightDto>> getFlightsByFilter(
        @RequestParam(value = "adults") @NotBlank String adults,
        @RequestParam(value = "origin") @NotBlank String origin,
        @RequestParam(value = "destination") @NotBlank String destination,
        @RequestParam(value = "departureDate")
                @Pattern(regexp = "^[0-9]{4}-(0[1-9]|1[0-2])-(0[1-9]|[1-2][0-9]|3[0-1])$")
                String departureDate,
        @RequestParam(value = "fareClass") @NotBlank String fareClass) {
    log.info(
            "Received request to get flights by filter: adults = {}, origin = {}, destination = {}, departureDate = {}, fareClass = {}",
            adults,
            origin,
            destination,
            departureDate,
            fareClass);

    Set<FlightDto> flightsDto =
            goflightlabsClientService.findFlightsByFilter(
                    adults, origin, destination, departureDate, fareClass);

    flightService.cacheAll(
            flightsDto.stream().map(FlightDto::toEntity).collect(Collectors.toSet()));

    return ResponseEntity.ok(flightsDto);
}
```

Рисунок 2.20 – Метод контролеру пошуку авіарейсів

```
@RequiredArgsConstructor
@EnableScheduling
@Service
public class DatabaseScheduler {

    1 usage
    private final FlightRepository flightRepository;


    @Transactional
    @Scheduled(fixedRate = 3, timeUnit = TimeUnit.MINUTES)
    public void deleteUnrelatedFlights() {
        flightRepository.deleteAllUnrelated();
    }
}
```

Рисунок 2.21 – Метод графіку кєшу

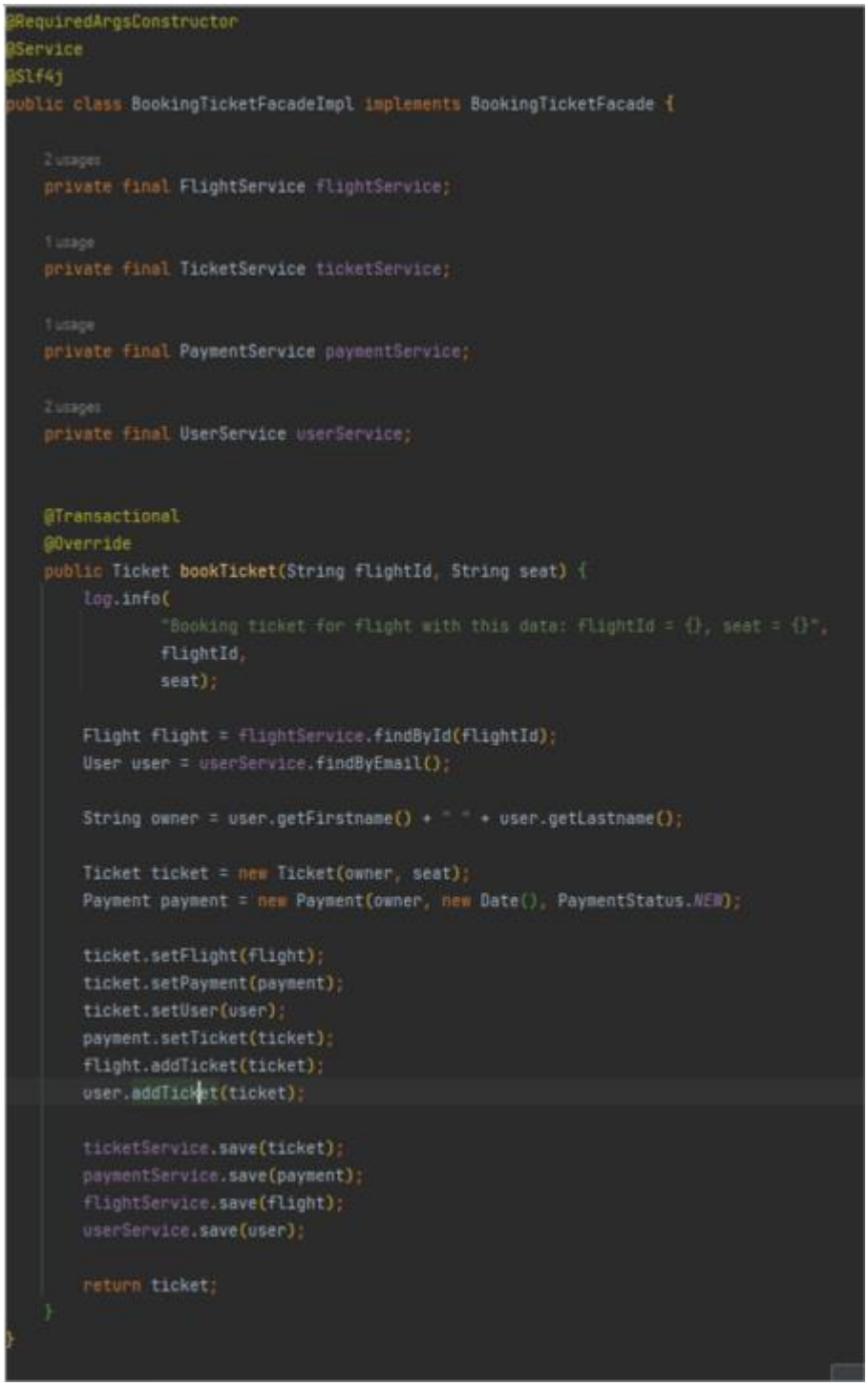
Процес розробки функціоналу бронювання квитка розпочався з того, що я створив подібні сутності, репозіторії та сервіси як для процесу с сек'юріті. Головним класом бізнес логіки для бронювання квитка став фасад «BookingTicketFacade».

Шаблон проектування фасаду в програмуванні є структурним шаблоном, який надає уніфікований інтерфейс для доступу до складної підсистеми. Фасад дозволяє приховати складність внутрішньої структури системи, надаючи простий інтерфейс взаємодії з нею. Основна ідея шаблону фасаду полягає в тому, щоб створити клас, який агрегує функціональність декількох класів і надає спрощений інтерфейс для роботи з ними. Фасад дозволяє клієнтському коду взаємодіяти з підсистемою через один об'єкт, замість викликати різні методи кожного класу підсистеми безпосередньо. Переваги використання шаблону фасаду включають спрощення складних систем, покращення розширюваності та підтримуваності коду, а також забезпечення ізоляції клієнтського коду від внутрішніх деталей підсистеми.

Клас імплемінтації має чотири сервіс класи у себе всередині:

- «FlightService»;
- «TicketService»;
- «PaymentService»;
- «UserService».

Коли метод бронювання починає працювати, спочатку у системі шукається авіарейс по його id, та шукається користувач. Якщо об’єкти було знайдено, створюється квиток з ім’ям людини, яка бронює білет, та платіж, який буде створений у системі. Після цього всі сутності зберігаються в базу даних. Цей фасадний клас та метод можна побачити на рисунку 2.22.



```java
@RequiredArgsConstructor
@Service
@Slf4j
public class BookingTicketFacadeImpl implements BookingTicketFacade {

    2 usages
    private final FlightService flightService;

    1 usage
    private final TicketService ticketService;

    1 usage
    private final PaymentService paymentService;

    2 usages
    private final UserService userService;


    @Transactional
    @Override
    public Ticket bookTicket(String flightId, String seat) {
        log.info(
                "Booking ticket for flight with this data: flightId = {}, seat = {}",
                flightId,
                seat);

        Flight flight = flightService.findById(flightId);
        User user = userService.findByEmail();

        String owner = user.getFirstname() + " " + user.getLastname();

        Ticket ticket = new Ticket(owner, seat);
        Payment payment = new Payment(owner, new Date(), PaymentStatus.NEW);

        ticket.setFlight(flight);
        ticket.setPayment(payment);
        ticket.setUser(user);
        payment.setTicket(ticket);
        flight.addTicket(ticket);
        user.addTicket(ticket);

        ticketService.save(ticket);
        paymentService.save(payment);
        flightService.save(flight);
        userService.save(user);

        return ticket;
    }
}
```

Рисунок 2.22 – Фасадний клас та його метод

Після цього процес бронювання переходить до контролеру «BookingController» (рисунок 2.23), а саме до POST методу «bookTicket», де користувач водить дані для бронювання квитка та номер рейсу через DTO об‘єкт, та отримає інформацію про квиток та платіж, який був створений.

```java
@RequiredArgsConstructor
@Api(tags = "Booking controller")
@RequestMapping("/api/booking")
@RestController
@Slf4j
public class BookingController {

    1 usage
    private final BookingTicketFacade bookingTicketFacade;


    @PostMapping
    public ResponseEntity<TicketInfoWithPaymentIdDto> bookTicket(
            @RequestBody @Valid BookTicketDto bookTicketDto) {
        log.info(
                "Received request to book a ticket for flight with id = {}",
                bookTicketDto.getFlightId());

        Ticket ticket =
                bookingTicketFacade.bookTicket(
                        bookTicketDto.getFlightId(), bookTicketDto.getSeat());

        return ResponseEntity.status(HttpStatus.CREATED)
                .body(ticket.toDtoWithPaymentId(ticket.getPayment().getId()));
    }
}
```

Рисунок 2.23 – Контролер бронювання квитка

У мікросервісі налаштована обробка винятків через клас з анотацією «@ControllerAdvice». Цей клас відловлює усі винятки та відображає їх у читальному вигляді для користувача. У кожному методі обробки винятка створюється об’єкт «ErrorResponse», який несе у собі дані про помилку для користувача, а саме: дату, значення коду помилки, код помилки та повідомлення, яке може підказати користувачу, що він робить не так. Приклад одного з методів обробки можна побачити на рисунку 2.24.

```java
@ExceptionHandler(UnauthorizedAccessException.class)
public ResponseEntity<ErrorResponse> handleUnauthorizedAccessException(
        UnauthorizedAccessException exception) {
    ErrorResponse errorResponse =
            new ErrorResponse(
                    new Date(),
                    HttpStatus.UNAUTHORIZED.value(),
                    HttpStatus.UNAUTHORIZED,
                    exception.getMessage());

    return new ResponseEntity<>(errorResponse, HttpStatus.UNAUTHORIZED);
}
```

Рисунок 2.24 – Метод обробки винятка

Писати тести для додатків на Java має ряд важливих переваг і дозволяє підвищити якість і надійність програмного забезпечення, що розробляється. Ось кілька основних причин, чому тестування є важливою практикою під час роботи:

- забезпечення коректності функціональності
- запобігання регресії
- поліпшення підтримки коду
- забезпечення стабільності при змінах

Для тестування бізнес логіки функціоналу пошуку авіарейсів я використав бібліотеку WireMock. Завдяки ній можна створити штучне API, яке буде відправляти налаштовані відповіді, а потім протестувати їх на своєму функціоналу.

Спочатку треба налаштувати сервер WireMock (рисунок 2.25), вказати його порт та запускати його перед кожним тестом. Це можна зробити завдяки анотаціям JUnit 5, а саме через «@BeforeEach». Після цього треба в методі тесту налаштувати відповідь від штучного API, та реалізувати сам тест, використовуючи тестові дані. Відповідь API знаходиться в «flights.json. Метод тесту можна побачити на рисунку 2.26.

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@WireMockTest
@ActiveProfiles(profiles = "test")
class GoflightlabsClientServiceTest {

    5 usages
    private WireMockServer wireMockServer;

    2 usages
    private final String accessKey = "my-access-key";

    2 usages
    @Autowired private GoflightlabsClientService goflightlabsClientService;

    Andrey Kravchenko
    @BeforeEach
    void setUp() {
        wireMockServer = new WireMockServer( port: 9090);
        wireMockServer.start();
    }

    Andrey Kravchenko
    @AfterEach
    void tearDown() {
        wireMockServer.stop();
    }
```

Рисунок 2.25 – Налаштувати сервер WireMock

```
@Test
void testShouldFindFlightsByFilter() {
    String adults = "1";
    String origin = "LOND";
    String destination = "MAD";
    String departureDate = "2023-03-14";
    String fareClass = "economy";

    wireMockServer.stubFor(
            get(urlPathEqualTo( testUrl: "/search-best-flights"))
                    .withQueryParam( s: "access_key", equalTo(accessKey))
                    .withQueryParam( s: "adults", equalTo(adults))
                    .withQueryParam( s: "origin", equalTo(origin))
                    .withQueryParam( s: "destination", equalTo(destination))
                    .withQueryParam( s: "departureDate", equalTo(departureDate))
                    .withQueryParam( s: "cabinClass", equalTo(fareClass))
                    .willReturn(
                            aResponse()
                                    .withStatus(HttpStatus.OK.value())
                                    .withHeader( key: "Content-Type",  ...values: "application/json")
                                    .withBodyFile( fileName: "flights.json")));

    Set<FlightDto> result =
            goflightlabsClientService.findFlightsByFilter(
                    adults, origin, destination, departureDate, fareClass);

    assertNotNull(result);
}
```

Рисунок 2.26 – Метод тесту пошуку авіарейсів

3 ЕКСПЕРЕМЕНТАЛЬНО-АНАЛІТИЧНИЙ РОЗДІЛ

3.1 Інструкція з використання мікросервісу

Після того як було запущено мікросеервіс потрібно перейти по адресі, яка вказана у файлі «README.md». Коли користувач перейде по ній, він побачить головне меню Swagger UI, його можна побачити на рисунку 3.1.

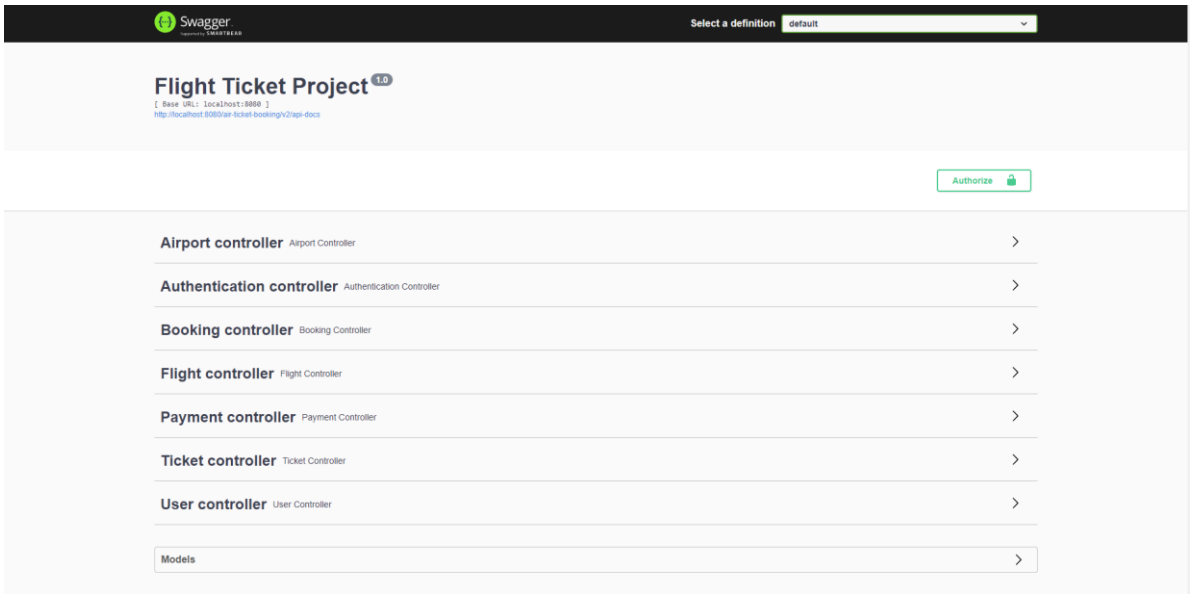


Рисунок 3.1 – Головна сторінка Swagger UI

Для того, щоб отримати доступ до кінцевих точок, користувачу потрібно пройти автентифікацію у контролері. Наприклад, я зареєструю нового користувача та отримаю токін доступу. Приклад реєстрації можна побачити на рисунку 3.2.

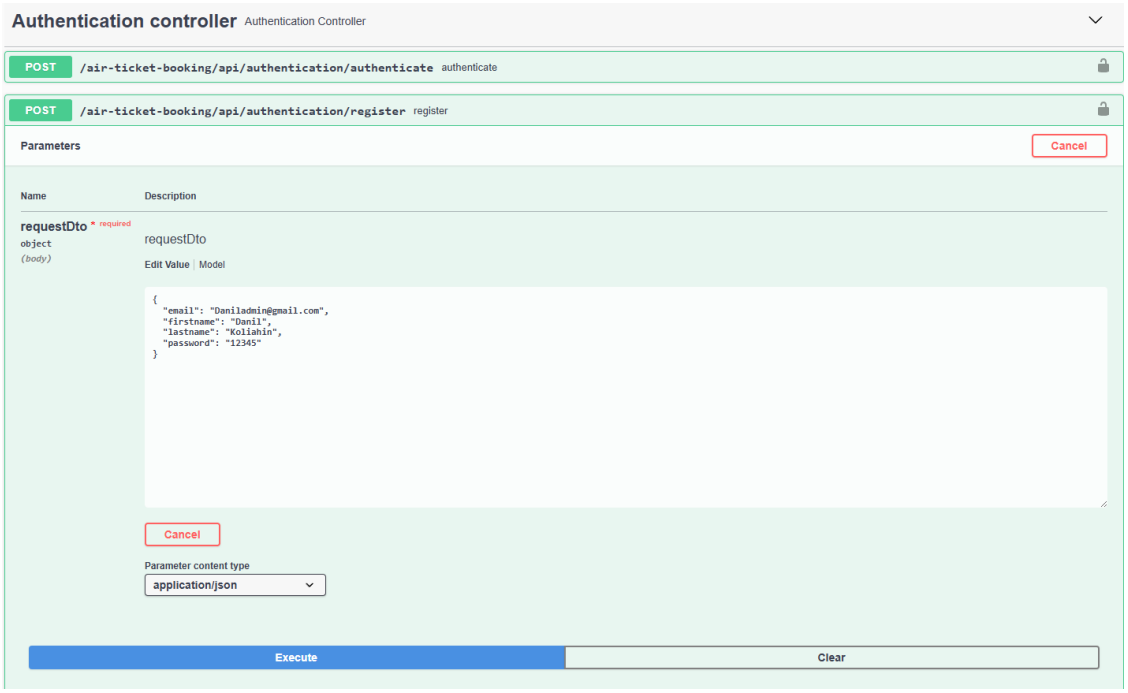


Рисунок 3.2 – Приклад реєстрації

Після цього користувач отримає токін доступу, який наведено на рисунку 3.3. Також можна бачить статус HTTP, адресу запиту та його параметри.

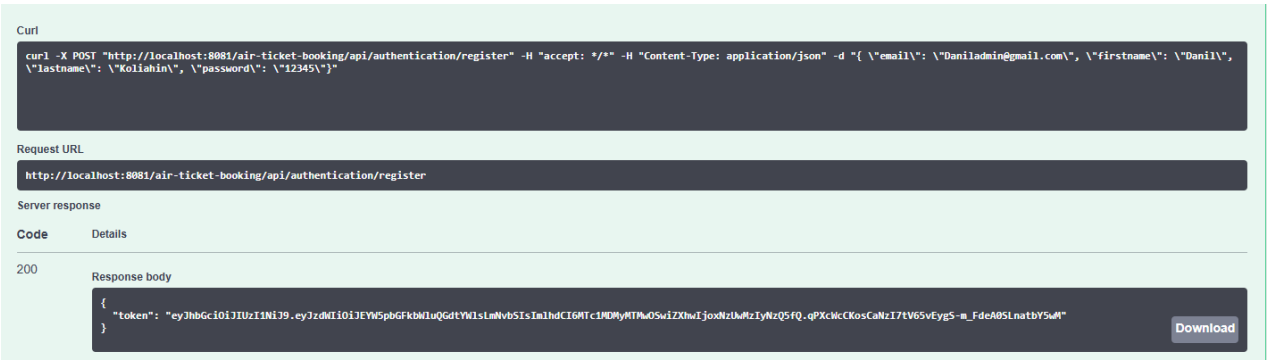


Рисунок 3.3 – Отриманий токін користувача

Далі користувачу потрібно використати цей токен. Це можно зробити через кнопку «Authorize», вставивши токін після слова «Bearer». Процес наведено на рисунку 3.4

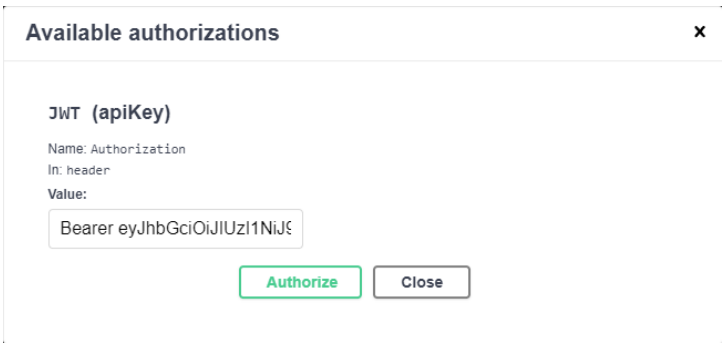


Рисунок 3.4 – Використання токіну

Відтепер у користувача є доступ до всіх наведених методів контролерів, окрім тих, які може виконати тільки адмін. Наприклад, можна відправити запит на пошук аеропортів у контролері «Airport Controller», вписати назву міста та отримати код аеропорту як відповідь, яка наведена на рисунку 3.5.

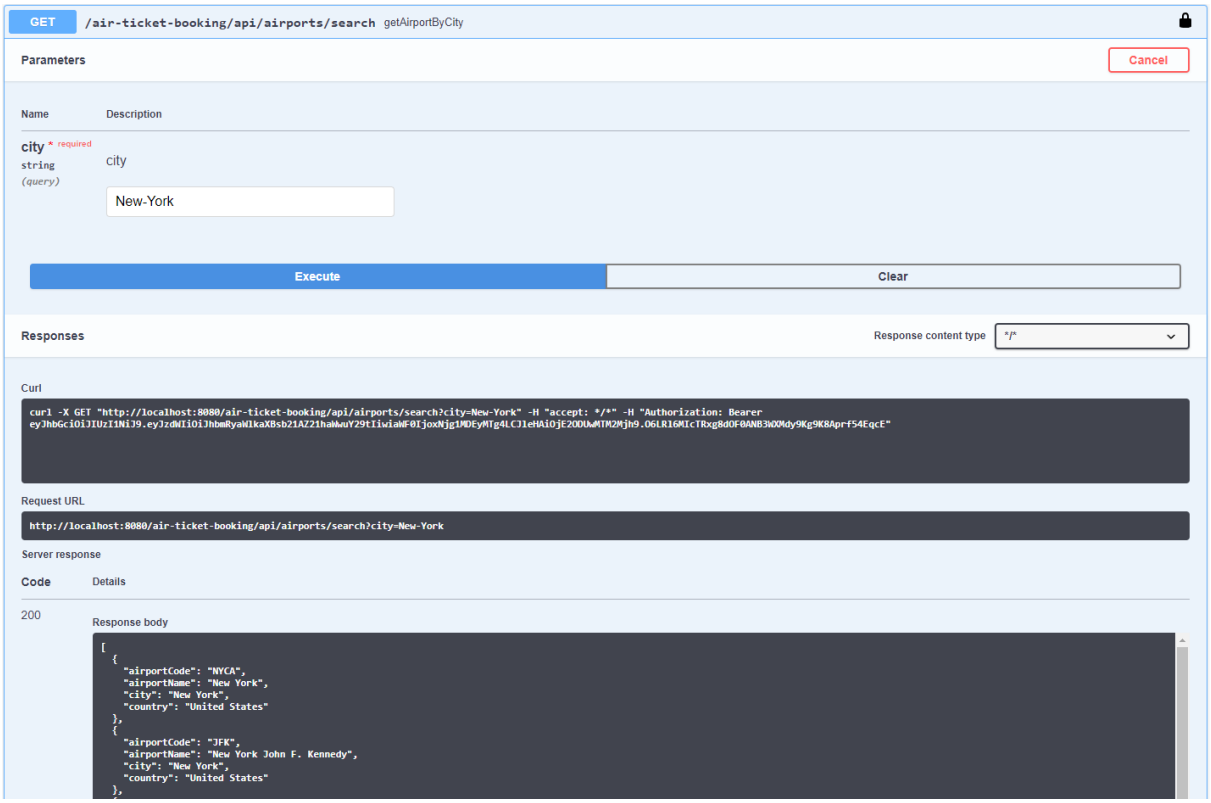


Рисунок 3.5 – Пошук аеропорту за назвою міста

Далі можна знайти авіарейси, через контролер «Flight Controller», використавши фільтр пошуку. Користувачу потрібно ввести кількість дорослих на рейс, дату відправки, аеропорти відправки та прильоту, а також клас яким він буде

летіти. Приклад пошуку авіарейсів з Нью Йорку до Лондону по фільтру наведено на рисунку 3.6.

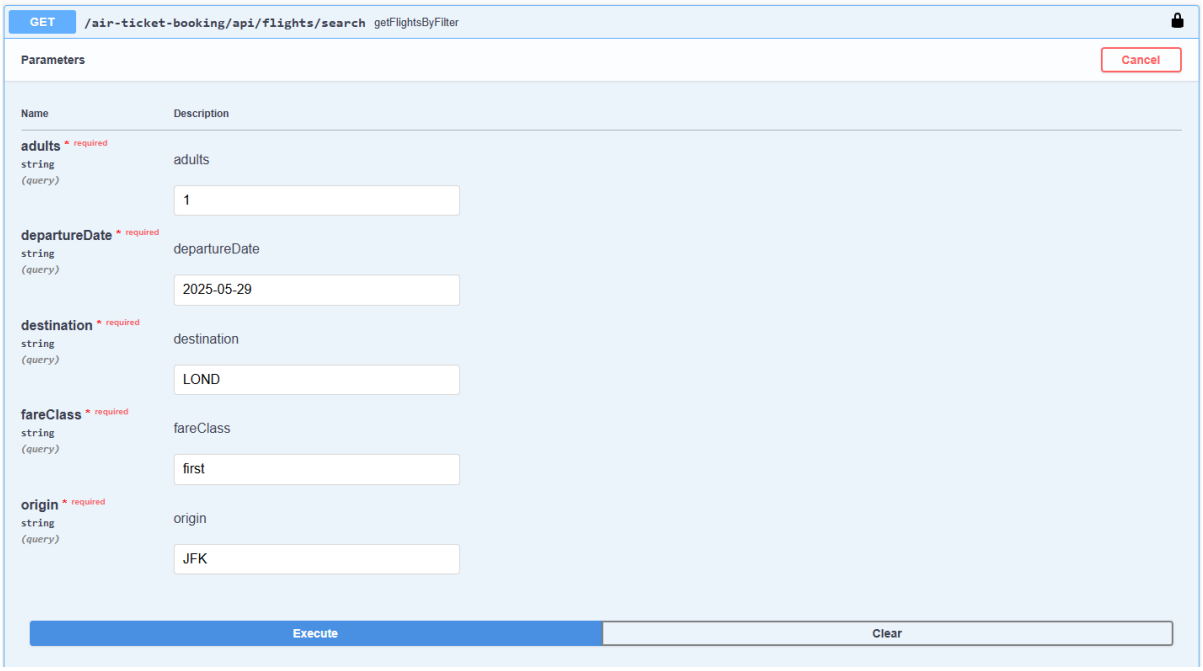


Рисунок 3.6 – Фільтр пошуку авіарейсів

У відповідь приходить колекція JSON об'єктів, яка несе у собі інформацію о авіарейсах. Відповідь наведено на рисунку 3.7.

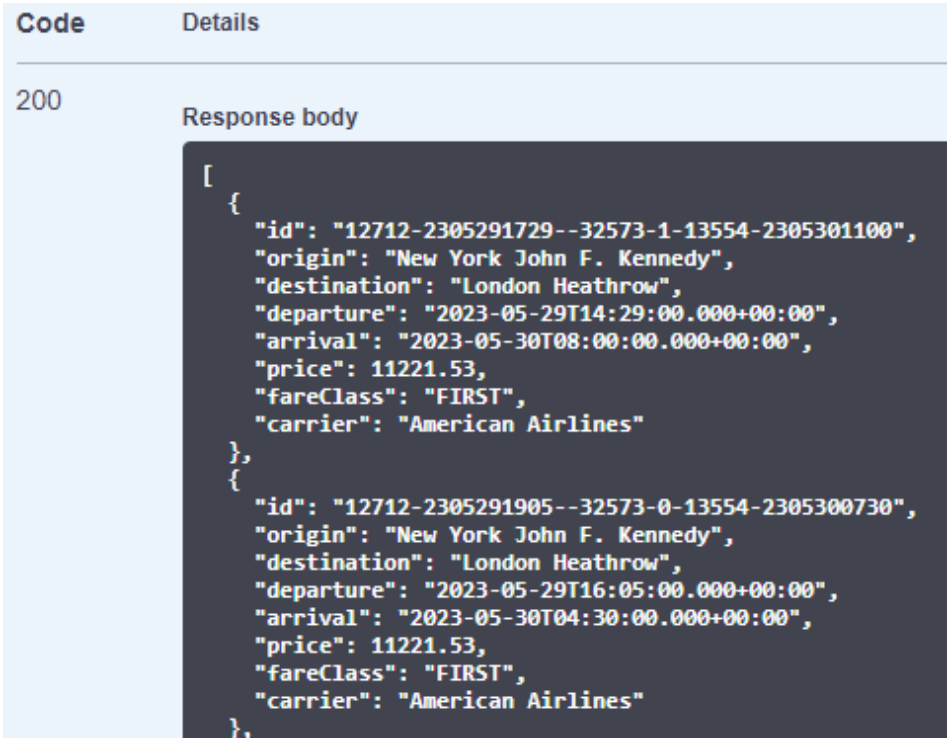


Рисунок 3.7 – Результат пошуку авіарейсів

Після цього можна забронювати білет на один з цих рейсів у «Booking Controller». Користувачу потрібно ввести id та місце, де він буде сидіти. Приклад запиту наведено на рисунку 3.8.

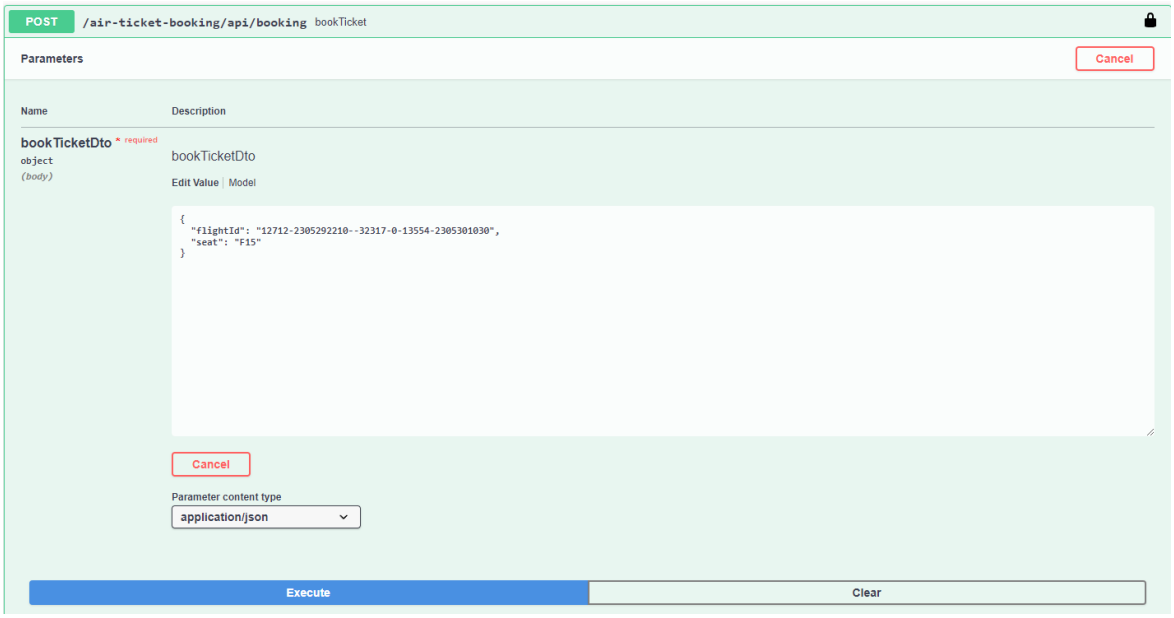


Рисунок 3.8 – Запит на бронювання квитка

У відповідь до користувача приходить інформація о квитку та id платежу (рисунок 3.9), який він може оплатити у контролері «Payment Controller». Для цього йому потрібно ввести id платежу та суму, яка буда зазначена у пошуку авіарейсу. Приклад запиту на оплату білета можна побачити на рисунку 3.10.

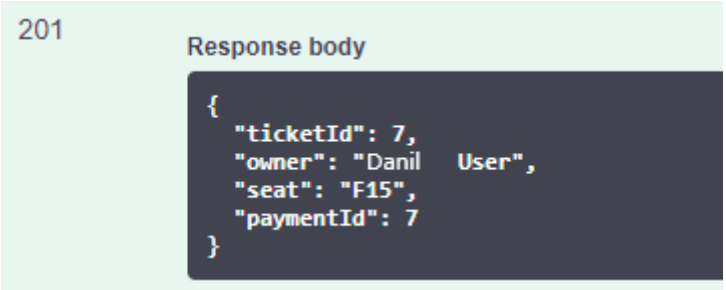


Рисунок 3.9 – Відповідь після бронювання

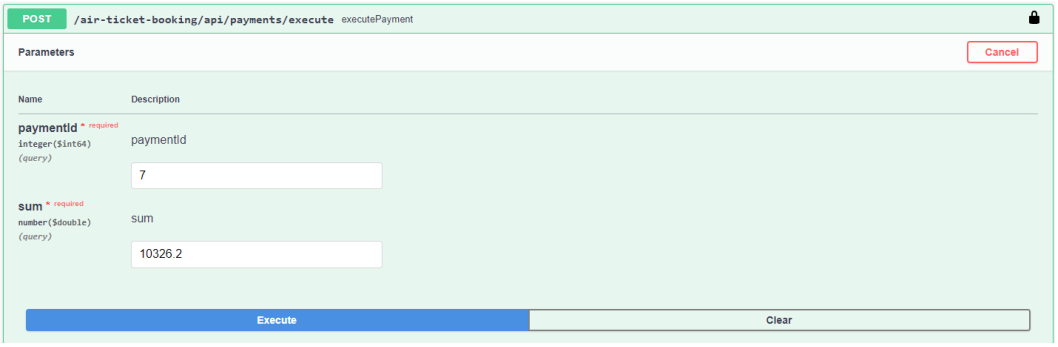


Рисунок 3.10 – Запит на оплату квитка

У відповідь йому приходить інформація о платежі: ім’я, його статус, дата оплати та сума. Приклад наведено на рисунку 3.11.

Рисунок 3.11 – Відповідь після сплати платежу

Відтепер користувач може відправити запит на пошук усіх рейсів, квитків, платежів. Наприклад, можна зробити пошук усіх авіарейсів, на які були заброньовані квитки через «Flight Controller» використавши метод «getAllFlightByUser». Приклад відповіді наведено на рисунку 3.12.

Рисунок 3.12 – Відповідь на пошук всіх своїх авіарейсів

3.2 Економічний вплив

Розрахунок калькуляції собівартості та ціни розробки мікросервісу для бронювання квитків на авіарейси

Для виконання розрахунку використовуються наступні початкові дані, які оформлено у таблиці 4.1

Таблиця 4.1

Початкові дані

Найменування даних	Значення	Джерело отримання
1 Середня місячна ставка програміста, грн.	23 033	Дані сервісного центру
2 Кількість робочих годин за місяць	184	Розрахунок проведено для травня місяця 2025 року. Кількість робочих днів – 23
3 Премія, %	30	Дослідно – статистичні
4 Додаткова заробітна плата %	10	Дослідно – статистичні
5 Відрахування до єдиного внеску, %	22	Нормативні дані по Україні
6 Загальновиробничі витрати, %	40	Досвідно-статистичні
7 Податок на додану вартість, %	20	Закон України
8 Прибуток, %	25	Дані сервісного центру
9 Адміністративні витрати, %	30	Дані сервісного центру
10 Інші витрати, %	2	Дані сервісного центру
11 Витрати на збут, %	10	Дані сервісного центру

3.2.1 Розрахунок трудомісткості створення мікросервісу для бронювання квитків на авіарейси

Розрахунок трудомісткості виконується на підставі спеціального розділу кваліфікаційної роботи.

Розрахунок зведений у таблицю 4.2.

Таблиця 4.2

Розрахунок трудомісткості розробки мікросервісу для бронювання квитків на авіарейси

Найменування операцій	Норма часу, год
1 Збір інформації	10
2 Пошук ПЗ	2
3 Встановлення програмного забезпечення	1
4 Дослідження робочого простору програмного забезпечення	2
5 Проектування бази даних	5
6 Написання коду згідно із поставленим завданням	80
7 Тестування та внесення коректівів	15
Разом	115

3.2.2 Розрахунок калькуляції собівартості створення мікросервісу для бронювання квитків на авіарейси

При створенні мікросервісу для бронювання квитків на авіарейси (матеріальні, нематеріальні, трудових ресурсів). Найбільш питому вагу в загальних витратах підприємства мають операційні витрати.

За економічним змістом розрізняють такі операційні витрати: матеріальні, на оплату праці, на соціальні потреби, загальновиробничі, адміністративні та інші операційні витрати [15].

Для визначення собівартості створення мікросервісу для бронювання квитків на авіарейси використовується розрахунково-аналітичний метод розрахунку калькуляції по статтям. Ці статті відрізняються між собою функціональною роллю у виробничому процесі.

Собівартість розробки мікросервісу для бронювання квитків на авіарейси:

$$С = М + З_О + З_д + В_{соц} + В_{зв} + В_{адм} + В_{зб} + В_{інш} \text{ грн.},$$

де М — вартість матеріалів, покупних комплектуючих елементів та виробів, грн;

$З_О$ — основна заробітна плата робітників, зайнятих розробкою мікросервісу для бронювання квитків на авіарейси, грн;

$З_д$ — додаткова заробітна плата цих робітників, грн;

$В_{соц}$ — відрахування до єдиного соціального внеску, грн;

$В_{зв}$ — загальновиробничі витрати, грн;

$В_{адм}$ — адміністративні витрати;

$В_{зб}$ — витрати на збут, грн;

$В_{інш}$ — інші оперативні витрати, грн.

3.2.2.1 Розрахунок вартості матеріалів, покупних елементів та виробів

Витрати на матеріали, покупні комплектуючі елементи та вироби не використовуються, тому вартість складає 0.

3.2.2.2 Розрахунок основної заробітної плати

Основна заробітна плата, згідно із законом України «Про оплату праці», встановлюється у вигляді тарифних ставок (окладів) та відрядних розцінок. Для розрахунку основної заробітної плати використовується формула:

$$З_О = t_{шт} \cdot С_{год} 1 + \left(\frac{П\%}{100}\right) \text{ грн.},$$

де $t_{шт}$ — трудомісткість мікросервісу для бронювання квитків на авіарейси, год;

П% — премія, %;

$С_{год}$ — годинна тарифна ставка, грн.

Годинна тарифна ставка розраховується за формулою:

$$С_{год} = \frac{С_м}{Т_м} \text{ грн.},$$

де $С_м$ — місячний посадовий оклад робітника, грн;

$Т_м$ — кількість робочих годин за місяць, год.

$$С_{год} = \frac{23\,033}{186} = 123.83 \text{ грн.}$$

$$З_О = 115 \cdot 123.83 \left(1 + \frac{30}{100}\right) = 18{,}512.585 \text{ грн.}$$

Основна заробітна плата за розробку мікросервісу для бронювання квитків на авіарейси 9363,18 грн.

3.1.3.3 Розрахунок додаткової заробітної плати виробничих робітників

Додаткова заробітна плата – це винагорода за роботу понад встановленої норми, за трудові успіхи, особливі умови праці. До неї входять передбачені чинним законодавством доплати, надбавки, гарантійні та компенсаційні витрати.

Додаткова заробітна плата розраховується у відсотках від основної за формулою:

$$З_д = З_О \cdot \frac{\%З_д}{100} \text{ грн.},$$

де $З_д$ — додаткова заробітна плата, грн;

$\% З_д$ — відсоток додаткової заробітної плати, грн;

$З_о$ — основна заробітна плата робітника, грн;

$$З_д = 18{,}512.585 \cdot \frac{10}{100} = 1{,}851.25 \text{ грн.}$$

Додаткова заробітна плата за трудові успіхи у розробці мікросервісу для бронювання квитків на авіарейси складає 936,31 грн.

3.2.2.3 Розрахунок єдиного соціального внеску

Єдиний соціальний внесок включає відрахування на пенсійне забезпечення, на соціальне страхування, страхування на випадок безробіття та на соціальне страхування від нещасних випадків і встановлюється у відсотках від витрат на оплату праці за формулою [16]:

$$В_{соц} = (З_О + З_д) \cdot \frac{\%В_{соц}}{100} \text{ грн.},$$

де $З_о$ — основна заробітна плата робітника, грн;

$З_д$ — додаткова заробітна плата, грн;

$\%В_{соц}$ — відсоток відрахувань до на соціальні заходи, встановлений чинним законодавством на 2025 рік складає 22%.

$$В_{соц} = (18{,}512.585 + 936{,}31) \cdot \frac{22}{100} = 4{,}278.75 \text{ грн.}$$

3.2.2.4 Розрахунок загальновиробничих витрат

Загальновиробничі витрати — це витрати на організацію виробництва, на утримання та експлуатацію устаткування загальновиробничого призначення, визначаються в відсотках від основної заробітної плати.

$$В_{зв} = З_О \cdot \frac{\%В_{зв}}{100} \text{ грн.},$$

де $З_О$ — основна заробітна плата робітника, грн;

$\%В_{зв}$ — відсоток загальновиробничих витрат,%;

$$В_{зв} = 18{,}512.585 \cdot \frac{40}{100} = 7{,}405.03 \text{ грн.}$$

3.2.2.5 Розрахунок виробничої собівартості

Виробнича собівартість охоплює витрати на виробництво або здійснення послуг у межах підприємства. Витрати на заробітну плату (основну и додаткову), єдиний соціальний внесок та загальновиробничі витрати складають виробничу собівартість, яка розраховується за формулою:

$$С_{вир} = М + З_О + З_д + В_{соц} + В_{зв} \text{ грн.},$$

$$С_{вир} = 0 + 18{,}512.585 + 936{,}31 + 4{,}278.75 + 7{,}405.03 = 31{,}132.67 \text{ грн.}$$

3.2.2.6 Розрахунок адміністративних витрат

Адміністративні витрати пов’язані з утриманням управлінського персоналу, розраховуються за формулою:

$$В_{адм} = З_О \cdot \frac{\%В_{адм}}{100\%}, \text{грн.},$$

де $З_О$ — основна заробітна плата робітників, грн;

$В_{адм}$ — відсоток адміністративних витрат,%;

$$В_{адм} = 18{,}512.585 \cdot \frac{30}{100} = 5{,}553.77 \text{ грн.}$$

3.2.2.7 Розрахунок інших операційних витрат

Інші операційні витрати розраховуються у відсотках від виробничої собівартості за формулою:

$$В_{інші} = С_{ВИР} \cdot \frac{\%В_{інші}}{100\%} \text{ грн.,}$$

$$В_{інші} = 31{,}132.67 \cdot \frac{2}{100} = 622.65 \text{ грн.}$$

3.2.2.8 Розрахунок витрат на збут

Витрати на збут пов'язані з витратами на рекламу, плануються у відсотках від виробничої собівартості і розраховуються за формулою:

$$В_{зб} = С_{вир} \cdot \frac{\%В_{зб}}{100}, \text{грн}$$

де: $С_{вир}$- виробнича собівартість, грн.;

$\%В_{зб}$ - відсоток витрат на збут, складає 10%.

$$В_{зб} = 31{,}132.67 \cdot \frac{10}{100} = 3{,}113.26 \text{ грн.}$$

3.2.2.9 Розрахунок повної собівартості

Повна собівартість послуг підприємства – це сукупність усіх витрат, виробничу собівартість та позавиробничі витрати (адміністративні, витрати на збут та інші операційні витрати).

$$С_п = С_{вир} + В_{адм} + В_{інш} + В_{зб} \text{ грн.,}$$

$$С_П = 16310,64 + 5,553.77 + 622.65 + 3,113.26 = 25,600.32 \text{ грн.}$$

3.2.3 Розрахунок калькуляції ціни розробки мікросервісу для бронювання квитків на авіарейси

Ціна — це грошовий вираз вартості товару. В умовах ринкової економіки ціна є з найважливіших показників, що впливають на фінансовий стан підприємства. Від рівня цін залежить розмір прибутку підприємства, конкурентоспроможність продукції або послуг і фінансова стійкість підприємства. Ціна складається з окремих елементів. Основним з них є собівартість і прибуток.

При розрахунку калькуляції ціни, розробки мікросервісу для бронювання квитків на авіарейси використовується метод «середні витрати + прибуток» - найпоширеніший при розрахунку ціни.

За цим методом ціна обчислюється за формулою:

$$Ц = С + П \text{ грн.,}$$

де С — собівартість продукції або послуг, грн;

П — величина прибутку в ціні, який встановлює підприємство або обмежує держава, грн.

3.2.3.1 Розрахунок прибутку

Прибуток від операційної діяльності є джерелом усіх фінансових ресурсів підприємства та розраховується за формулою:

$$П = С_п \cdot \frac{\%П}{100} \text{ грн.},$$

де П — рівень прибутковості підприємства у відсотках, %.

$$П = 25{,}600.32 \cdot \frac{25}{100} = 6{,}400.08 \text{ грн.}$$

3.2.3.2 Розрахунок оптової ціни (вільної)

Оптова (вільна) ціна на послугу встановлюється підприємством на договірній основі із врахуванням попиту і пропозиції на ринку послуг, орієнтовані на економічну зацікавленість виробників у збільшенні обсягу послуг. Вони можуть змінюватися за взаємною згодою сторін.

$$Ц_О = С_П + П \text{ грн.},$$

де $С_п$ — повна собівартість послуги, грн;

П — сума прибутку, грн.

$$Ц_О = 25{,}600.32 \ + 6{,}400.08 = 32{,}000.40 \text{ грн.}$$

3.2.3.3 Розрахунок податку на додану вартість

Податок на додану вартість (ПДВ) є частиною новоствореної вартості, яка сплачується у державний бюджет на кожному етапі виробництва продукції або виконання послуг [17]. ПДВ включається у ціну за встановленою ставкою.

Законодавством України встановлена (станом на 2025 рік) основна ставка ПДВ – 20 % до оптової ціни без ПДВ.

ПДВ розраховується за формулою:

$$ПДВ = Ц_о \cdot \frac{20\%}{100}, грн$$

де: $Ц_о$ - ціна оптова на послугу, грн.;

20% - основна ставка ПДВ.

$$ПДВ = 32{,}000.40 \cdot \frac{20}{100} = 6{,}400.08, грн.$$

3.2.3.4 Розрахунок відпускної ціни реалізації (для замовника)

Відпускна ціна реалізації на послугу для замовника буде складати:

$$Ц_Р = Ц_О + ПДВ, грн$$

де: $Ц_О$ - ціна оптова на послугу, грн.;

ПДВ- податок на додану вартість, грн.

$$Ц_Р = 32{,}000.40 + 6{,}400.08 = 38{,}400.48 \text{ грн}$$

Результати усіх попередніх розрахунків зведено у таблицю 4.3.

Таблиця 4.3

Калькуляція собівартості та ціни створення сайту візитівки графічних робіт

Статті калькуляції	Сума, грн.
1 Вартість матеріалів покупних комплектуючих елементів і виробів	0
2 Основна заробітна плата	18,512.585
3 Додаткова заробітна плата	1,851.25

Продовження табл. 4.3

4 Єдиний соціальний внесок	4,278.75
5 Загальновиробничі витрати	7,405.03
6 Виробнича собівартість	16310,64
7 Адміністративні витрати	5,553.77
8 Інші операційні витрати	622.65
9 Витрати на збут	3,113.26
10 Собівартість повна	25,600.32

11 Прибуток	6,400.08
12 Ціна оптова (вільна)	32,000.40
13 ПДВ	5269,21
14 Ціна реалізації підприємства для замовника	38,400.48

Висновок: в цьому розділі було розраховано калькуляцію собівартості та ціни створення мікросервісу для бронювання квитків на авіарейси. Ціна для замовника складає 38,400.48 гривень.

3.3 Організації робочого місця користувача

Встановлене устаткування: системний блок комп'ютера, монітор, необхідні периферійні пристрої, призначені для розробки програмного продукту. Робоче місце обладнане столом, шафами та поворотним стільцем.

Робота пов’язана з розумовою працею, що потребує уваги і пам’яті, активізації процесів мислення без значних фізичних зусиль.

Категорія робіт за ступенем важкості, яка проводиться на робочому місці відноситься до легкої – 1 категорії за енерговитратами до 150 Ккал/год.

Зорова напруга відноситься до 5 категорії – малої точності, найменший розмір розпізнавання об'єкту від 1,0 до 5,0 мм.

Робоче місце організовано для одного працівника, за фахом системотехніка, має розмір 3,0 х 2,8 х 3,2 м.

3.4 Техніка безпеки

Охорона праці (ОП) – нормативна дисципліна, яка вивчається з метою формування у майбутніх фахівців із вищою освітою необхідного в їхній подальшій професійній діяльності рівня знань та умінь із правових та організаційних питань з охорони праці, основ фізіології, гігієни праці, виробничої санітарії, безпеки процесів праці та пожежної безпеки.

На ділянці можливі такі причини травмування:

- технічні – невідповідальність нормам безпеки конструкції технологічного устаткування, технологічного оснащення, невідповідальність конструкції

устаткування ергономічним вимогам, відсутність вказівок про способи і засоби безпечного виконання роботи, несправність технологічного устаткування;

- організаційні – відсутність або неякісне проведення інструктажу і навчання, відсутність необхідної технічної документації, інструкції з охорони праці, порушення правил проведення робіт, також режимів праці і відпочинку, незадовільна організація і утримання робочих місць, включаючи незабезпечення необхідних санітарно-гігієнічних умов праці, недостатній контроль охорони праці;

- санітарно-гігієнічні пов’язані з перевищенням рівня шуму, вібрації, щільності електромагнітного випромінювання, поганого освітлення, незадоволеного мікроклімату;

- психофізіологічні причини, до яких належать фізичні та нервово-психологічні перенавантаження.

Згідно з Законом України “Про охорону праці” ст. 15, служба ОП створюється роботодавцем або уповноваженим ним органом на підприємствах, установах, організаціях незалежно від форм власності та видів їх діяльності для організації виконання правових, організаційно-технічних, санітарно-гігієнічних, соціально-економічних і лікувально-профілактичних заходів, спрямованих на запобігання нещасних випадків, професійним захворюванням і аваріям під час праці.

На підприємствах виробничої сфери з кількістю працівників до 50 осіб (невиробничої сфери – до 100 осіб та навчальних закладах) функції служби ОП виконують особи, котрі пройшли перевірку знань з ОП, тобто з відповідною підготовкою за сумісництвом [18].

На підприємствах (у виробничих і науково-виробничих об’єднаннях) при чисельності працівників від 51 до 500 осіб включно (невиробнича сфера, зокрема навчальні заклади від 101 до 500 осіб) таку службу повинен представляти один спеціаліст з ОП з інженерно-технічною освітою та стажем роботи за профілем виробництва не менше трьох років. На підприємствах, де використовуються вибухові матеріали або сильнодіючі отруйні речовини, в такій службі повинно бути двоє спеціалістів.

За невиконання питань з охорони праці на порушників накладаються наступні міри відповідальності:

- адміністративна (штраф, попередження, позбавлення окремих прав, зупинка роботи);
- дисциплінарна (догана, звільнення з роботи);
- матеріальна (розмір виплат не більше ніж місячна ставка);
- кримінальна (максимальний термін покарання за порушення законів ОП – 7 років);
- заходи суспільної дії (збори колективу).

Заходи по боротьбі з виробничим травматизмом розробляються на підставі їх аналізу конкретних ситуацій та конкретних умов праці і узгоджуються з професійними спілками. Такі заходи, залежно від конкретних умов виробничої діяльності можуть включати як технічні, санітарно-гігієнічні так і організаційні методи та засоби запобігання реалізації небезпечних ситуацій та небажаних подій.

До організаційних заходів належать – дотримання трудової та технологічної дисципліни, правил та норм з охорони праці, проведення планово-запобіжних ремонтів, рівень кваліфікації штатних працівників, відомчий та громадський контроль за виконанням робіт, відповідне навчання та інструктаж працюючих та ін.

Працівники підприємства під час прийняття на роботу і періодично в процесі праці проходять за рахунок роботодавця навчання і перевірку знань, інструктажі з охорони праці, надання першої медичної допомоги потерпілим від нещасних випадків і правил поведінки у разі виникнення аварії. Види інструктажу показані у таблиці 5.1.

Таблиця 5.1

Види інструктажу

Види інструктажу	Призначення	Ким проводиться	Терміни проведення
Вступний	Проводиться серед працівників, що прибули у відрядження і студентами. Учнями скерованими на виробничу практику. Роз'ясняється значення виробничої, трудової дисципліни, ознайомлення з умовами роботи, правила внутрішнього розпорядку, правила електробезпеки, надання першої допомоги, особиста гігієна, вимога до утримання робочих місць, основні вимоги пожежної безпеки	Інженер з охорони праці або особа на яку покладені його обов'язки	Перед початком роботи. По прибуттю на підприємство
Первинний	Для тільки прибулих у цех робітників, або переведених з іншого, які беруть безпосередню участь у виробничому процесі. Також проводять студентам при виробничій практиці	Керівник підрозділу. На невеликих підприємствах безпосередньо керівник з наступним опитуванням	На робочому місці до початку роботи
Повторний	Цей інструктаж на робочому місці повинні проходити усі працівники незалежно від кваліфікації, освіти і стану	Керівник робочого місця (майстер). З наступним опитуванням	За умов роботи з підвищеною небезпекою – один раз за квартал на інших роботах раз за півроку

Продовження табл. 5.1

Позаплановий	- введення нових правил або внесення змін; - при зміні технології та устаткування; - порушення техніки безпеки; - на вимогу державного нагляду, якщо виявлене недостатнє знання; - при перерві у роботі понад 30 днів для працюючих з підвищеною небезпекою; - для решти робіт – у випадку перерви понад 60 днів	Керівник робочого місця (з наступним опитуванням)	По необхідності
Цільовий	Проводиться з працівниками перед виконанням робі, не пов'язаних з безпосередніми функціональними обов'язками (вантажно – розвантажувальні роботи, прибирання території від будівельного сміття тощо)	Керівник робочого місця з наступним опитуванням	По необхідності

Оператор комп'ютерного набору повинен проходити періодичний медогляд (відповідно до наказу Міністерства охорони здоров'я України від 21.05.2007 р. № 246) раз на два роки, комісію в складі терапевта, невропатолога та офтальмолога. Жінки що працюють з ВДТ обов'язково оглядаються акушером-гінекологом раз на 1 рік. Жінки з часу встановлення вагітності та в період годування дитини до виконання всіх робіт, пов'язані з використанням ВДТ, не допускаються.

До основних форм контролю за станом охорони праці належать:

а) державний нагляд – генеральний прокурор і підпорядковані йому прокурори. Його органи є незалежними від державних адміністрацій, господарських, громадських, політичних організацій і діють відповідно до положень, затверджених кабінетом міністрів України [19];

б) відомчий контроль – міністерство, комітет об'єднання та інше;

в) громадській контроль – трудові колективи, через обраних ними уповноважених осіб;

г) регіональний контроль – місцеві державні адміністрації та ради народних депутатів, через посадових осіб, що відповідають за охорону праці;

д) адміністративно – громадський трьохступеневий контроль.

Важливе місце у створенні безпечних умов праці відводиться технологічним режимам, які мають створювати необхідні умови:

а) злагодженості технічного устаткування;

б) безвідмовної дії технологічного устаткування і засобів захисту протягом строку передбаченого технічною документацією;

в) запобігання можливого загорання або пожежі тощо.

У кожному підприємстві щорічно розробляються заходи щодо профілактики виробничого травматизму й професійних захворювань які включаються в колективні договори, забезпечуються технічною документацією, джерелами фінансування та матеріальними ресурсами.

При організації праці, що пов’язана з використанням ВДТ (відеотерміналів), ЕОМ (електронно-обчислювальних машин) і ПЕОМ (персональних електронно-обчислювальних машин), для збереження здоров’я працюючих, запобігання професійним захворюванням і підтримання працездатності передбачено внутрішньо змінні режими при 8-годинному робочому дні залежно від характеру праці:

– для розробників програм – 15 хвилин перерви через кожну годину роботи;

– для операторів ЕОМ – 15 хвилин через кожні 2 години роботи;

– для операторів комп’ютерного набору – 10 хвилин перерви через кожну годину роботи.

У всіх випадках, коли виробничі обставини не дозволяють застосовувати регламентовані перерви, тривалість безперервної роботи з ВДТ не може перевищувати 4 години.

Працівники мають бути проінформовані та проінструктовані щодо дій, необхідних у разі виникнення на підприємстві аварійних ситуацій, пов’язаних з загрозою для їх життя і здоров’я, та про вжиті або такі, що мають бути вжитими, запобіжні і захисні заходи.

До технічних засобів попередження травмування відноситься огородження всіх небезпечних ділянок на виробництві, безпечне розміщення обладнання ділянки.

Електричні дроти заізольовані, робоче місце обладнано розетками 220 В, над розетками є табличка 220 В, металеві корпуса електроустаткування заземлені, дільниця має захисне відключення обладнання яке призначене для відключення електроустановок при пошкодженні ізоляції і переході напруги на неелектропровідні елементи.

Розрахунок заземлення виконаний на ПЕОМ приведений на рисунку 5.1.

Початкові дані:

1. Питомий опір ґрунту, $\rho = 30$ Ом·м
2. Довжина вертикального стрижня $L = 2{,}7$ м
3. Глибина залягання горизонтального заземлювача, $t_0 = 0{,}7$ м
4. Діаметр вертикального стрижня (ширина полки), $D(B) = 52$ мм
5. Ширина горизонтального заземлювача, $b = 24$ мм
6. Відстань між стрижнями, $S = 2{,}7$ м
7. Коеф. сезонності для вертикальних стрижнів, $\psi_в = 1{,}3$
8. Коеф. сезонності горизонтального смугового електрода $\psi_г = 1{,}4$

Результати розрахунку:

1. Опір одиночного вертикального заземлювача, $R_0 = 11{,}47$ Ом
2. Опір горизонтального смугового електрода, $R_{смуг} = 11{,}28$ Ом
3. Опір групового заземлювача, $R_{гр} = 2{,}91$ Ом
4. Дійсна кількість вертикальних електродів, $N = 4$ шт.
5. Довжина смугового електрода, $L_{смуг} = 8{,}51$ м

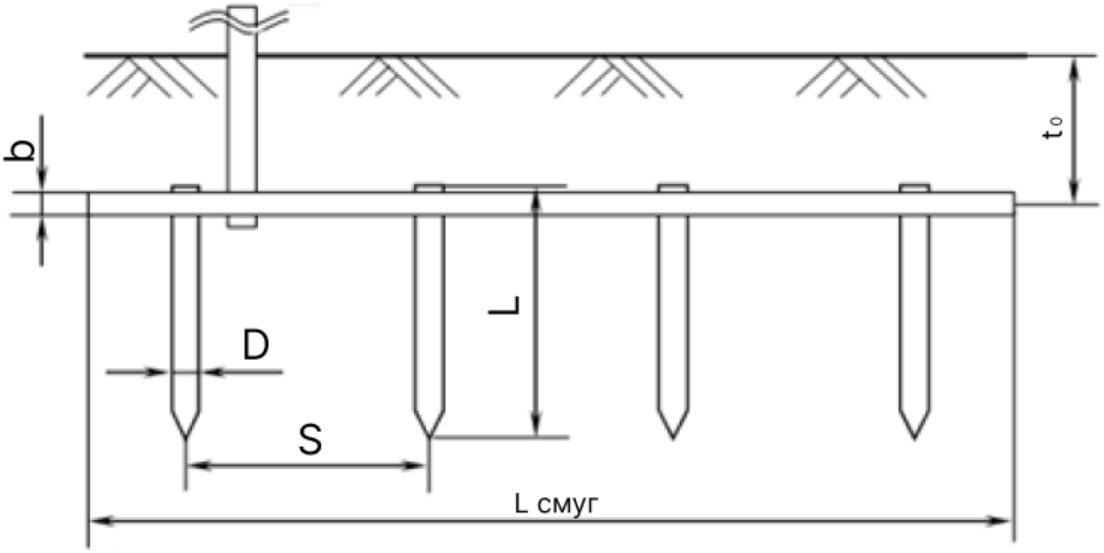


Рисунок 5.1 – Схема заземлюючого пристрою

Висновок: опір групового заземлювача $R_{гр}$ = 2,91 Ом, що відповідає вимогам ПУЕ, так як одержана величина $R_{гр}$ < 4 Ом.

3.5 Виробнича санітарія та гігієна праці

Наказ Мінсоцполітики від 14.02.2018 року № 207 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» (далі – Вимоги), зареєстрований в Міністерстві юстиції України 25 квітня 2018 року № 508/31960, поширюється на всіх суб’єктів господарювання незалежно від форм власності, організаційно-правової форми і видів діяльності та встановлюють мінімальні вимоги безпеки та захисту здоров’я під час здійснення роботи, пов’язаної з використанням екранних пристроїв незалежно від їхнього типу і моделі. Роботодавець зобов’язаний забезпечити в приміщеннях з ПК оптимальні параметри виробничого середовища. Оптимальні параметри виробничого середовища наведені у таблиці 5.2.

Таблиця 5.2

Оптимальні параметри виробничого середовища

Пора року	Категорія робіт	Температура повітря, °С, не більше	Відносна вологість повітря, %	Швидкість руху повітря, м/с
Холодна	Легка 1-я	22-24	40-60	0,1
Тепла	Легка 1-я	23-25	40-60	0,1

Для підтримки таких параметрів дільниця облаштована водяним опаленням та загально-обмінною вентиляцією. Підлога на дільниці вкрита паркетом, стіни пофарбовані масляною фарбою сірого цвіту. Шум не перевищує 65 дБ. Значення напруженості електростатичного поля на робочих місцях з ВДТ не перевищує 10 Вт/м^2. Площа на одне робоче місце становить не менше ніж 6 м^2, а об'єм не менш ніж 20 м^3.

Освітлення виробничих приміщень для роботи з ВДТ має бути природним і штучним.

Природне освітлення має здійснюватися через світлові прорізи, орієнтовані на північ або північний схід, що забезпечують КПО ≥1,5 %. Необхідно використовувати одностороннє бокове природне освітлення з площею світлових прорізів – 25 % від площі підлоги.

Віконні прорізи обладнують регульованими пристроями (жалюзі, завіски, зовнішні козирки), а робочі столи розміщують подалі від вікон і так, щоб вікна були зліва від них.

Для внутрішнього оздоблення слід використовувати дифузно-відбивні матеріали з коефіцієнтом відбиття: для стелі – 0,7–0,8; для стін – 0,5–0,6; для підлоги 0,3–0,5.

Нормована освітленість на поверхні робочого столу в зоні розміщення документів має бути 300–500 лк.

Виробничі приміщення повинні обладнуватись шафами для зберігання документів, електронних носіїв інформації, полицями, стелажами, тумбами тощо, з урахуванням вимог до площі приміщень.

Умови праці операторів під час роботи з відеотерміналами визначаються характеристиками обладнання, яке використовується, якістю робочих матеріалів, розміщенням елементів обладнання і матеріалів у робочій зоні, конструкцією меблів та їх розмірами.

Розмір екрана за діагоналлю має бути не меншим за 38 см. Збільшення розміру екрана призводить до таких недоліків: погіршується відбиваюча здатність екрана (утворення відблисків), деформуються знаки на периферії екрана, екран складно розмістити у нормальному полі зору.

З ергономічної точки зору найкращим є плоский екран, який поглинає зовнішні світлові потоки, чим зменшує кількість відблисків.

Оптимальна висота розташування екрана має відповідати спрямованості зору оператора в секторі 50 – 35° відносно горизонталі. Якщо верхній край екрана вищий за рівень очей, то зчитування інформації з екрана може викликати стан дискомфорту.

При організації робочого місця враховують антропометричні дані операторів, а також розміщення елементів обладнання залежно від характеру роботи, яку виконують. Робочий стіл повинен мати стабільну конструкцію: дошка стола повинна мати розміри 180 × 90 см і регулюватися за висотою в діапазоні 65 – 85 см, висота від горизонтальної лінії зору до робочої поверхні стола має становити 450 – 500 см. Висота сидіння регулюється за висотою в межах 420 – 550 см.

Покриття поверхні стола повинно бути матовим з коефіцієнтом відбиття 20 – 50 %, легко чиститися, кути і передня панель дошки стола – заокруглені. Сидіння – комфортне, із заокругленими краями – має нахилятися за горизонталлю вперед на 20° і назад на 140°, розмір його не більший ніж 400 х 400 см. Висота спинки крісла становить 480 – 500 см від поверхні сидіння. Підніжка крісла повинна мати п’ять опор для запобігання падінню.

Робочі місця з ВДТ мають відповідати таким вимогам:

а) відстань між бічними поверхнями ВДТ – 1,2 м;

б) відстань від тильної поверхні одного ВДТ до екрана іншого ВДТ – 2,5 м;

в) прохід між рядами робочих місць має бути не меншим за 1 м;

г) висота робочої поверхні робочого стола – 680–800 мм;

д) ширина робочої поверхні робочого стола – 600–1400 мм;

е) глибина робочої поверхні робочого стола – 800–1000 мм;

ж) простір для ніг: заввишки – 600 мм; завширшки – 500 мм, завглибшки – 450 мм.

Конструкція робочого місця користувача ВДТ має забезпечувати підтримання оптимальної робочої пози з такими ергономічними характеристиками:

а) ступні ніг – на підставці для ніг або на підлозі;

б) стегна – в горизонтальній площині;

в) передпліччя – вертикально;

г) лікті – під кутом 70–90° до вертикальної площини;

д) зап’ястя – зігнуті під кутом не більше 20° відносно горизонтальної площини;

е) нахил голови – 15–20° відносно вертикальної площини.

Екран та клавіатуру розташовують на оптимальній відстані від очей користувача, але не ближче ніж 600 мм, з урахуванням розміру алфавітно-цифрових знаків і символів, а також розміру екрана за діагоналлю. Клавіатура повинна розташовуватися в 10-15 сантиметрах від краю столу. Відстань яка повинна бути від екрана до очей користувача залежно від розміру екрана ВТД наведена у таблиці 5.3.

Таблиця 5.3

Відстань від екрана до очей користувача залежно від розміру екрана ВТД

Розмір екрана по діагоналі	Відстань від екрана до очей, мм
35/38 см (14"/15")	600–700
43 см (17")	700–800

Продовження табл. 5.3

48 см (19")	800–900
53 см (21")	900–1000

У приміщеннях з ВДТ слід щоденно робити вологе прибирання.

Приміщення з ВДТ мають бути оснащені аптечками першої допомоги.

Приміщення з ВДТ мають бути обладнанні побутовими приміщеннями для відпочинку під час роботи-кімнатою психологічного розвантаження. В кімнаті психологічного розвантаження слід передбачити встановлення пристроїв для приготування й роздачі тонізуючих напоїв, а також місця для занять фізичною культурою (ДСанПін 3.3.2007-98).

Дільниця відноситься до екологічно чистих виробництв.

3.6 Пожежна безпека

На дільниці можливі причини неелектричного й електричного характеру.

До причин неелектричного характеру відносяться:

– необережне та халатне використання вогню;

– неправильний пристрій і несправність вентиляційної системи;

– самозаймання речовин і матеріалів.

До причин електричного характеру відносяться:

– короткі замикання;

– несправність перевантаження електроустаткування й електромереж;

– іскріння й електричні дуги;

– загорання внаслідок грозових розрядів, розрядів статичної електрики;

– великі перехідні опори в місцях з’єднань, відгалужень, у контактах електромашин і апаратів, що призводять до локального перегріву.

Правила пожежної безпеки в Україні відповідно до ГОСТ 12.1.004-91 визначають заходи пожежної безпеки, в які входить:

– дільниця відноситься до категорії приміщень та будівель за вибухопожежною і пожежною небезпекою (Д), згідно ОНТП 24-86, пожежонебезпечного класу П-IIа за ПБЄ;

– звукопоглинальне облицювання стін та стелі слід виготовляти з негорючих або важкозаймистих матеріалів;

– приміщення мають бути обладнані системою автоматичної пожежної сигналізації з димовими пожежними сповіщувачами та вогнегасниками з розрахунку 1 шт. на 200 м^2 площі.

Для запобігання пожежі проводяться заходи які мають: організаційний, технічний, експлуатаційний і режимний характер.

До організаційних заходів відносяться навчання робітників та службовців пожежної безпеки, проведення бесід, лекцій, інструктажу. На дільниці організовується пожежна дружина не менш чим з п’яти чоловік.

Експлуатаційні заходи передбачають правильну експлуатацію устаткування, своєчасні регулярні огляди установок і апаратів, їхній огляд, ремонт і іспити, правильне розташування будинків і територій.

Усунення електричних причин пожежі проводиться по різному. Попередження короткого замикання здійснюється правильним вибором, монтажем і експлуатацією мереж електроустановок, застосуванням захисту схем у виді швидкодіючих реле, вимикачів, безконтактних автоматичних схем захисту, плавких запобіжників. Для захисту електромережі від перевантажень застосовуються автоматичні вимикачі, теплові реле, запобіжники. Зменшення контактного опору досягається збільшенням площі контактів, застосуванням нероз’ємних з’єднань проводів пайкою, зварюванням.

На дільниці встановлений пожежний щит, що за кількістю задовольняє вимозі 1 щит на 200 м^2. Він укомплектований різними засобами пожежогасіння: ручними вогнегасниками ВВК-5, сокирою, азбестовим покривалом 2 х 2 м, багром. Мається ручна система сигналізації. Дільниця обладнана датчиками теплової дії. Двері приміщення відкриваються назовні, мається запасний евакуовихід. Застосовується звукова сигналізація, що оповіщає про пожежу, на будинку цеху

мається блискавковідвід. На дільниці є протипожежний кран, зовні будинку знаходиться гідрант в п’яти метрах від будинку, встановлена спринклерна система пожежогасіння.

До технічних заходів відносяться: дотримання протипожежних правил і норм при проектуванні будинків і споруд, наявність у будівлях евакуаційних виходів та сходів, систем освітлення, вентиляції, опалення, кондиціювання.

До заходів режимного характеру відносяться заборона проведення електрогазозварювальних та інших вогненебезпечних робіт в приміщеннях, заборона паління в невстановлених місцях.

3.7 Захист навколишнього середовища

Оцінювання впливів на навколишнє середовище на різних стадіях життєвого циклу виробничого процесу або продукту є важливим інструментом управління охороною навколишнього середовища. Це допомагає ідентифікувати, аналізувати і квантифікувати потенційні впливи на довкілля, що можуть виникнути на різних етапах.

Оцінювання впливів на навколишнє середовище можна проводити на наступних стадіях життєвого циклу:

1. Добування сировини: Оцінка включає аналіз впливу добування сировини на природні ресурси, використання енергії, забруднення повітря і води, зниження біорізноманіття та зміну використання землі.
2. Виробництво: Оцінка впливу виробництва на довкілля включає аналіз використання енергії, води і розсіяння відходів, емісії парникових газів, забруднення повітря та води, відходів, викидів шкідливих речовин та відходів, а також оцінку якості продукту і можливостей його утилізації.
3. Транспортування: Оцінка включає аналіз викидів шкідливих речовин та парникових газів, використання енергії, забруднення повітря, шуму та води,

впливу на транспортну інфраструктуру та можливості зменшення негативного впливу шляхом застосування екологічно чистих видів транспорту.

4. Використання та експлуатація: Оцінка включає аналіз використання енергії, води, викидів та відходів, а також впливу на здоров'я людей та довкілля під час використання продукту.

5. Утилізація та повторне використання: Оцінка включає аналіз можливостей повторного використання, переробки або утилізації продукту, включаючи вплив на відходи, забруднення води, повітря та ґрунту.

Всі ці стадії життєвого циклу повинні бути враховані при оцінці впливів на навколишнє середовище, щоб ідентифікувати можливі негативні наслідки і розробляти стратегії для зменшення цих впливів. Оцінка впливів на навколишнє середовище допомагає підприємствам і організаціям приймати раціональні рішення щодо виробництва, дизайну продукту та використання ресурсів з метою забезпечення сталого розвитку і збереження навколишнього середовища.

ВИСНОВОК

У даній кваліфікаційній роботі було розроблено мікросервіс для бронювання квитків на реальні авіарейси за допомогою зовнішнього API. Проект був успішно реалізований, і його ефективність була перевірена шляхом тестування та оцінки його функціональності і продуктивності.

Під час розробки мікросервісу було враховано різні аспекти, включаючи дизайн архітектури, безпеку даних та оптимальну взаємодію з зовнішнім API. Мікросервіс був реалізований з використанням сучасних технологій і фреймворків, забезпечуючи гнучкість і розширюваність системи.

Отримані результати дозволяють зробити висновок про успішне впровадження функціоналу для бронювання квитків на авіарейси. Його можна використовувати в реальних умовах для забезпечення зручного та ефективного процесу бронювання квитків для користувачів. Дана робота відкриває можливості для подальшого розширення та вдосконалення функціональності системи, з метою забезпечення ще більшої зручності та задоволення потреб користувачів.

Згідно даних розрахунку економічного розділу повна собівартість мікросервісу складає 25,600.32 грн.

В розділі «Охорона праці» розглянуті питання організації робочого місця користувача ЕОМ, розрахунок заземлюючого пристрою до заданого завдання, що складає $R_{гр}$ = 2,91 Ом. Також розглянуті правила пожежної безпеки та виробничої санітарії.

Список використаних джерел

1 Кваліфікаційна робота бакалавра [https://do.nmu.org.ua] : методичні рекомендації для здобувачів ступеня бакалавра освітньо-професійної програми «Системний аналіз» зі спеціальності 124 Системний аналіз / уклад.: Т.А. Желдак, Т.В. Хом’як, А.В. Малієнко ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2025. – 32 с.

2 Flight Labs API. app.goflightlabs.com - документація. URL: https://app.goflightlabs.com/dashboard

3 Об'єктно-орієнтована мова програмування Java. Матеріал з Вікіпедії — вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Java

4 Apache Maven. Матеріал з Вікіпедії — вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Apache_Maven

5 Maven Repository. Apache Maven Project. URL: https://maven.apache.org/guides/introduction/introduction-to-repositories.html

6 Spring Framework. spring.io – сайт фреймворку. URL: https://spring.io

7 Hibernate. Матеріал з Вікіпедії — вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Hibernate

8 Introduction to JWT. jwt.io – сайт технології. URL: https://jwt.io/introduction

9 API Documentation Swagger. swagger.io – сайт документації. URL: https://swagger.io/solutions/api-documentation/

10 Project Lombok. projectlombokз – сайт технології. URL: https://projectlombok.org/

11 JUnit 5. junit.org - сайт технології. URL: https://junit.org/junit5/

12 WireMock. wiremock.org - сайт технології. URL: https://wiremock.org/

13 Logback. logback.qos - сайт технології. URL: https://logback.qos.ch/

14 PostgreSQL. Матеріал з Вікіпедії — вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/PostgreSQL

15 Database H2. h2database - сайт технології. URL: https://www.h2database.com/html/main.html

16 Калина, А. В. Економіка підприємства: Навчальний посібник для студентів вищих навчальних закладів: учебное пособие / Калина А.В., Котвицький А.А., Стожок О.З. – К. : Знання України, 2007. - 323 с

17 Мацибора, В. І. Економіка підприємства: Навч. посібник для студентів вищ. навч. закл.: учебное пособие / В.І. Мацибора, В.К.Збарський, Т.В. Мацибора. – К. : Каравела, 2008. - 312 с. – (Вища освіта в Україні)

18 Шваб, Л. І. Економіка підприємства: навчальний посібник для студентів вищих навчальних закладів / Л.І. Шваб. – 4-е вид. – К. : Каравела, 2007. - 584 с

19 Гандзюк, М. П. Основи охорони праці. Підручник. – К.: Каравела, 2004. – 408 с

20 Законодавство України про охорону праці: Збірник нормативних документів. – К. : Основа, 2003, в 4-х томах

Додаток А. Відомість матеріалів кваліфікаційної роботи

№ з/п	Позначення	Найменування	Кількість аркушів	Примітки
1				
2		Документація		
3				
4	САУ.КР.25.28.ПЗ	Пояснювальна записка	82	Формат А4
5				
6		Демонстраційний матеріал	12	Презентація на CD-R
7				
8		Копія роботи	1	Диск CD-R
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				

					САУ.КР.25.28.ДА.ПЗ.
Змін.	Аркуш	№ докум.	Підпис	Дата	

Розроб.	Колягін Д.С.			**Матеріали кваліфікаційної роботи**	Літ.		Аркуш	Аркушів
К. розд.	Слєсарєв В.В.							
Керівн.	Слєсарєв В.В.				НТУ «ДП», 12; 124-21-1			
Н.контр.	Хом’як Т.В.							
Зав. каф.	Жедлак Т.А.							

Додаток Б

ЛІСТИНГ КОДУ

1. PaymentProcessorFacadeImpl

```
@RequiredArgsConstructor
@Service
@Slf4j
public class PaymentProcessorFacadeImpl implements PaymentProcessorFacade
{

   private final FlightService flightService;

   private final PaymentService paymentService;

   @Override
   public Payment executePayment(long paymentId, double sum) {
      log.info("Execute payment with this data: paymentId = {}, sum = {}",
paymentId, sum);

      Payment payment = paymentService.findById(paymentId);

      checkOwner(payment.getTicket().getUser().getEmail());

      checkExecution(payment.getStatus());

      double priceFlight = flightService.findPriceFlightByPaymentId(paymentId);
      checkSum(sum, priceFlight);

      payment.setSum(sum);
      payment.setDate(new Date());
      payment.setStatus(PaymentStatus.DONE);

      return paymentService.save(payment);
   }

   private void checkExecution(PaymentStatus paymentStatus) {
      if (!paymentStatus.equals(PaymentStatus.NEW)) {
         log.error(
               "Payment status is not 'New', throwing exception. Payment status =
```

```
{}",
                paymentStatus);
            throw new PaymentAlreadyExecuteException(
                "Payment already execute or is in the archive");
        }
    }

    private void checkSum(double sum, double priceFlight) {
        if (sum != priceFlight) {
            log.error(
                "Payment sum does not match the price of the flight. Sum = {},
priceFlight = {}",
                sum,
                priceFlight);
            throw new InvalidSumException(
                "Invalid sum = " + sum + " because price of the flight = " +
priceFlight);
        }
    }

    private void checkOwner(String email) {
        String currentUserEmail = JwtAuthenticationFilter.getCurrentUserEmail();
        if (!email.equals(currentUserEmail)) {
            log.error(
                "User not authorized to execute payment. Owner payment = {},
current user = {}",
                email,
                currentUserEmail);
            throw new UnauthorizedAccessException("User not authorized to execute
payment");
        }
    }
}
```

2. PaymentController

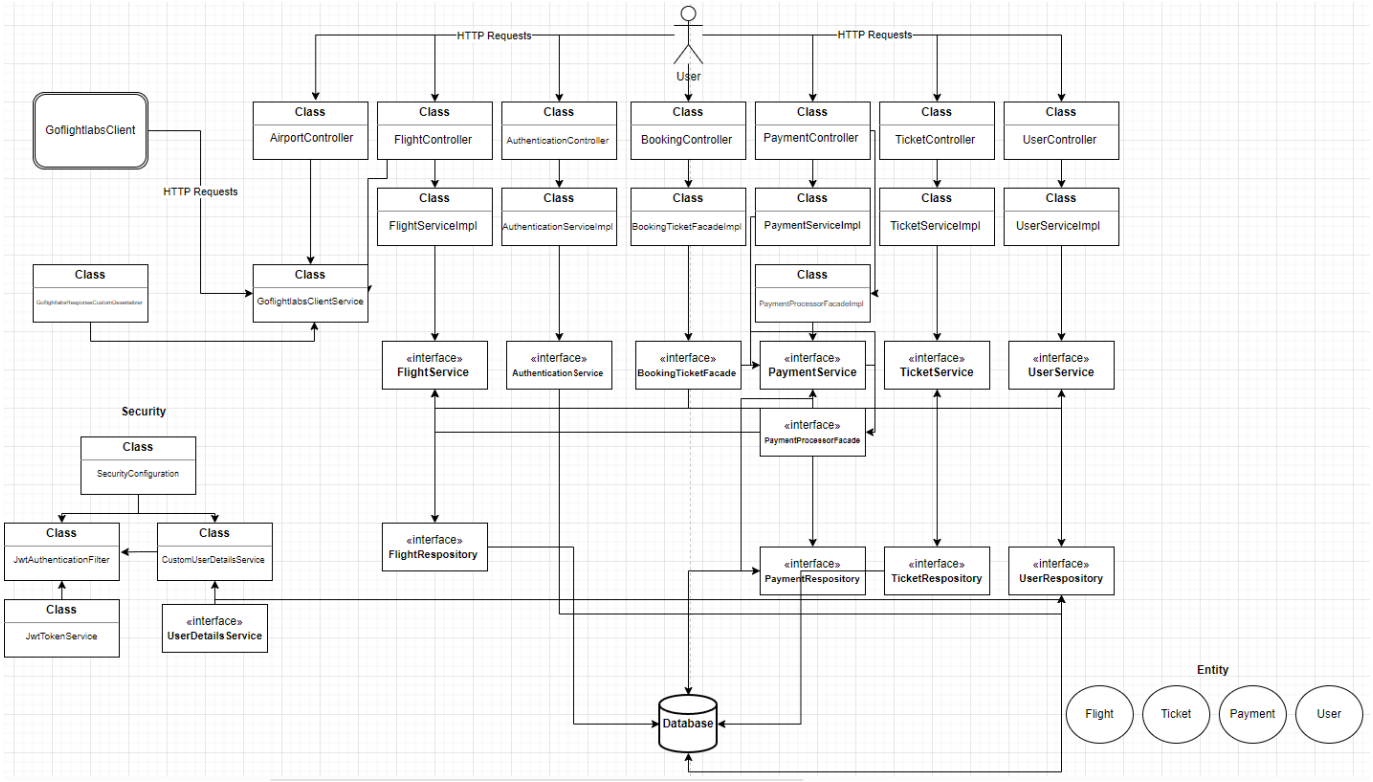
```
@RequiredArgsConstructor
@Api(tags = "Payment controller")
@RequestMapping("/api/payments")
@RestController
@Slf4j
public class PaymentController {

    private final PaymentService paymentService;

    private final PaymentProcessorFacade paymentProcessorFacade;
```

```
    @GetMapping
    public ResponseEntity<List<PaymentDto>> getAllPayments() {
        List<PaymentDto> paymentsDto =
                paymentService.findAll().stream().map(Payment::toDto).toList();

        return ResponseEntity.ok(paymentsDto);
    }

    @GetMapping("/user")
    public ResponseEntity<List<PaymentDto>> getAllPaymentsByUser() {
        List<PaymentDto> paymentsDto =

paymentService.findAllByUser().stream().map(Payment::toDto).toList();

        return ResponseEntity.ok(paymentsDto);
    }

    @GetMapping("/{paymentId}")
    public ResponseEntity<PaymentDto> getPaymentById(@PathVariable
@Min(1) long paymentId) {
        Payment paymentById = paymentService.findById(paymentId);

        return ResponseEntity.ok(paymentById.toDto());
    }

    @PostMapping
    public ResponseEntity<PaymentDto> savePayment(@Valid @RequestBody
PaymentDto paymentDto) {
        Payment payment = paymentDto.toEntity();

        Payment savedPayment = paymentService.save(payment);

        return
ResponseEntity.status(HttpStatus.CREATED).body(savedPayment.toDto());
    }

    @PostMapping("/execute")
    public ResponseEntity<PaymentDto> executePayment(
            @Min(1) @RequestParam(value = "paymentId") long paymentId,
            @RequestParam(value = "sum") @Min(1) double sum) {
        log.info("Received request to execute a payment with id = {}", paymentId);

        Payment payment = paymentProcessorFacade.executePayment(paymentId,
sum);
```

```
        return ResponseEntity.ok(payment.toDto());
    }
}
```

Додаток В

СТРУКТУРНА СХЕМА АРХІТЕКТУРИ



Додаток Г

Відгук
на кваліфікаційну роботу бакалавра
здобувача вищої освіти групи *124 – 21 – 1*
спеціальності *124 Системний аналіз*

Тема кваліфікаційної роботи: _____
_____
Обсяг кваліфікаційної роботи _____ стор.
Мета кваліфікаційної роботи: _____
_____
Актуальність теми _____
_____
_____
Тема кваліфікаційної роботи безпосередньо пов'язана з об'єктом діяльності бакалавра спеціальності 124 Системний аналіз, оскільки _____
_____
_____
Виконані в кваліфікаційній роботі завдання відповідають вимогам ступеня бакалавра. Оригінальність наукових рішень полягає в _____
_____
_____
_____
Практичне значення результатів кваліфікаційної роботи полягає в _____
_____
_____
Висновки підтверджують можливість використання результатів роботи в _____
_____
_____
Оформлення пояснювальної записки та демонстраційного матеріалу до неї виконано згідно з вимогами. Роботу виконано самостійно, відповідно до завдання та у повному обсязі (*в разі невідповідності – вказати*)
У роботі відзначено такі недоліки: _____
_____
_____
Кваліфікаційна робота в цілому заслуговує оцінки:_____

З урахуванням висловлених зауважень автор (не) заслуговує присвоєння кваліфікації «бакалавр з системного аналізу».

Керівник кваліфікаційної роботи бакалавра,
науковий ступінь, вчене звання, посада _____ / ПІБ