

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

«Система управління авіакомпанією»

Виконав студент групи ПЗ-21-1
КОБИЛКО Марія Павлівна

Керівник роботи:
ВАКАЛЮК Тетяна Анатоліївна

Рецензент:
ВЛАСЕНКО Олег Васильович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії програмного
забезпечення

Тетяна Вакалюк

«26» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **КОБИЛКО Марія Павлівна**

Керівник роботи: **Вакалюк Тетяна Анатоліївна**

Тема роботи: **«Система управління авіакомпанією»,**

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: **об'єктом дослідження є процеси управління авіакомпанією. Предметом дослідження є засоби, алгоритми та методи, що забезпечують ефективну роботу системи під час управління рейсами, автоматизації процесів бронювання, моніторингу доступності літаків та екіпажу.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Вакалюк Т.А.	27.02.2025	27.02.2025
2	Вакалюк Т.А.	27.02.2025	27.02.2025
3	Вакалюк Т.А.	27.02.2025	27.02.2025

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебдодатку «Система управління авіакомпанією» та пояснювальної записки. Пояснювальна записка до випускної роботи містить 66 сторінок, 56 ілюстрацій та 36 таблиць. Список використаних джерел містить 30 найменувань.

Метою роботи є розробка системи для ефективного управління основними процесами авіакомпанії.

В ході виконання роботи було сформульовано завдання, проведено аналіз аналогічних систем у цій сфері, обрано відповідний технологічний стек, що задовільняє всі функціональні вимоги до системи. Спроектовано та реалізовано базу даних, розроблено алгоритми для реалізації ключової бізнес-логіки, а також описано основні компоненти програмного коду.

У результаті було створено вебдодаток – систему управління авіакомпанією, що дозволяє автоматизувати основні процеси та мінімізувати людський фактор.

КЛЮЧОВІ СЛОВА: АВІАКОМПАНІЯ, СИСТЕМА, БАЗА ДАНИХ, ВЕБДОДАТОК, SPRING BOOT, HIBERNATE, MVC.

					ІПЗ.КР.Б-121-25-ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата	Система управління авіакомпанії			Лім.	Арк.	Аркушів	
Розроб.		Кобилко М.П									
Перевір.		Вакалюк Т. А.								3	66
Керівник								ФІКТ Гр. ІПЗ-21-1			
Н. контр.		Тичина О.П.									
Зав. каф.		Вакалюк Т. А.									

ABSTRACT

The Bachelor's qualification thesis consists of a web application Airline management system and an explanatory note. The explanatory note to the thesis contains 66 pages, 56 illustrations, and 36 tables. The list of references contains 30 entries.

The purpose of the work is to develop a system for effective management of the main processes of an airline.

During the work, the task was formulated, an analysis of similar systems in this field was conducted, an appropriate technology stack was chosen that meets all the functional requirements of the system. The database was designed and implemented, algorithms for the implementation of key business logic were developed, and the main components of the program code were described.

As a result, a web application – an airline management system – was created that allows automating the main processes and minimizing the human factor.

KEYWORDS: AIRLINE, SYSTEM, DATABASE, WEB APPLICATION, SPRING BOOT, HIBERNATE, MVC.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СИСТЕМИ УПРАВЛІННЯ АВІАКОМПАНІЇ.....	9
1.1. Постановка задачі.....	9
1.2. Аналіз аналогів програмного продукту	11
1.3. Вибір архітектури системи управління авіакомпанією	16
Висновок до першого розділу.....	21
РОЗДІЛ 2. ПРОЄКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ АВІАКОМПАНІЇ	22
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	22
2.2 Розробка бази даних системи.....	31
2.3 Проектування та реалізація алгоритмів роботи системи	39
2.4 Реалізація системи.....	42
Висновки до другого розділу	45
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ СИСТЕМИ УПРАВЛІННЯ АВАКОМПАНІЄЮ.....	46
3.1 Структура інтерфейсу системи управління авіакомпанією	46
3.2 Тестування роботи програмного комплексу	58
Висновки до третього розділу	61
ВИСНОВКИ.....	62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	63
ДОДАТКИ.....	66
Додаток А.....	67

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

MVC – Model view controller

БД – База даних

API – Application Programming Interface (Прикладний програмний інтерфейс)

REST API – Representational State Transfer Application Programming Interface
(архітектурний стиль взаємодії клієнта з сервером через HTTP)

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

JPA – Java Persistence API (інтерфейс для роботи з об'єктно-реляційним
відображенням (ORM) у Java)

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

Актуальність. Сьогодні авіація вважається найбезпечнішим видом транспорту. А суворі стандарти контролю, технічне обслуговування літаків і кваліфікаційний персонал мінімізують будь-які ризики. Авіап перевезення підтримують розвиток туризму та сприяють інтеграції країн у світову економіку.

Для того щоб залишатися конкурентоспроможним, а також відповідати високим очікуванням пасажирів в якості обслуговування, точності рейсів і доступності цін, авіакомпаніям потрібно впроваджувати нові технології. Важливо автоматизувати та вдосконалювати процеси. Такими технологіями є система управління авіакомпанією. Це складний програмний продукт, що охоплює великий спектр функцій, необхідних для ефективного управління авіап перевезеннями.

Крім того наявність системи управління гарантує безпеку польотів. Вона дозволяє зменшити ризики, слідкувати за технічним станом літаків і швидко реагувати на будь-які зміни чи проблеми. Ще один актуальний аспект є те, що всі дані - розклад, завантаження літаків, технічний стан, зміни маршрутів доступні в одному місці. Це дає змогу оперативно приймати рішення в режимі реального часу. Окрім безпеки польотів, система також відповідає за захист персональних даних з дотриманням певних міжнародних і національних нормативів.

Основною метою даної роботи є розробка системи управління авіакомпанією, що забезпечить автоматизацію ключових бізнес-процесів, підвищить ефективність операційної діяльності компанії та покращить взаємодію з пасажирами й персоналом.

Предметом дослідження є методи та засоби автоматизації процесів управління авіакомпанією, а також інформаційні технології, що застосовуються для розробки таких систем.

Об'єктом дослідження є процеси управління авіакомпанією, включаючи адміністрування рейсів, бронювання та продаж квитків, контроль розкладу польотів, управління екіпажем та облік ресурсів компанії.

Завданнями роботи є дослідження основних вимог до програмного

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

забезпечення для управління авіакомпанією. Вибір та обґрунтування інструментальних засобів, що відповідають специфіці авіаційної галузі. Проведення аналізу функціональних та нефункціональних вимог до програмного продукту. Проведення порівняльного аналізу з аналогами. Розроблення бази даних системи.

За темою випускної кваліфікаційної роботи бакалавра було опубліковано тези [1, 2].

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СИСТЕМИ УПРАВЛІННЯ АВІАКОМПАНІЇ

1.1. Постановка задачі

Авіаційна індустрія є складною і технологічно розвиненою галуззю. Авіакомпанії щодня стикаються з процесами, що вимагають точності та відсутності помилок. Наприклад, формування розкладу рейсів, продаж квитків, управління екіпажем, обробка багажу та багато іншого. Недоліки та помилки в цих процесах можуть призвести до фінансових втрат, незадоволеності клієнтів та навіть порушення безпеки польотів, що є недопустимим.

Враховуючи складність всіх зазначених процесів, використання інформаційних технологій є важливим для забезпечення роботи авіакомпаній.

Сучасні авіакомпанії зазвичай використовують інформаційні системи, які забезпечують автоматизацію більшості процесів.

Проте також мають свої недоліки - висока вартість. Крім того великий набір функцій який є у даних рішеннях, не завжди є необхідними для малих авіаперевізників та навіть є неефективним. Також варто відмітити складність у впровадженні, адже впровадження таких систем потребує значного часу та висококваліфікованого персоналу. Крім вищезазначених проблем, важливим аспектом є і питання гнучкості масштабування. Багато з існуючих рішень не враховують потреби компаній, що перебувають на етапі росту, коли необхідно поступово додавати нові функції без повної перебудови інфраструктури. Це створює ризик втрати інвестицій або потреби у повній заміні програмного забезпечення після певного періоду експлуатації.

Саме через низку таких причин виникає потреба у розробці гнучкої, доступної та ефективної системи управління авіакомпанією, яка дозволить автоматизувати основні бізнес-процеси та забезпечити високу якість обслуговування пасажирів.

Основна мета розробки є створення системи управління авіакомпанією у вигляді веб-застосунку.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

Невід’ємною складовою є розробка зручного користувацького інтерфейсу, що дозволить мінімізувати час навчання персоналу та зробить взаємодію із системою інтуїтивно зрозумілою аби не ускладнити щоденну роботу, знижуючи загальну продуктивність.

Система матиме три рівні доступу – адміністратор, що матиме повний контроль над системою; пасажир, що купуватиме квитки, переглядатиме рейси; робітники (пілоти та бортпровідники), що будуть переглядати свій розклад рейсів на які їх було назначено.

Основними завданнями, які вирішує система є:

1. Реєстрація та авторизація користувачів.
2. Управління аеропортами, літаками.
3. Управління рейсами (створення, редагування, скасування рейсів; планування маршрутів і розкладу; контроль за доступністю літаків та екіпажу для виконання конкретного рейсу).
4. Продаж квитків (інтеграція з платіжними системами та керування бронюваннями).
5. Управління екіпажем (встановлення графіку роботи, введення реєстру сертифікатів пілотів на літак, облік робочих годин, щоб уникнути перевантаження).
6. Управління персоналом (відстеження спроможності персоналу перебувати на тому чи іншому рейсів та облік ліцензій на пілотування певними моделями літаків (для пілотів)).
7. Контроль багажу.

Розробка та впровадження запропонованого функціоналу системи забезпечить авіакомпаніям можливість значно вдосконалити свої бізнес-процеси, знизити вплив людського фактору на управлінські функції завдяки автоматизації ключових операцій.

В результаті буде створено систему який стане альтернативою існуючим аналогам.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				10
Змн.	Арк.	№ докум.	Підпис	Дата		

1.2. Аналіз аналогів програмного продукту

Система управління авіакомпанією поєднує багато функцій. Було проведено порівняльний аналіз з найбільш популярними системами.

Спершу варто розглянути аналоги з точки зору користувача.

Одним з аналогів є авіакомпанія Wizz Air (рис.1.1), що має власну систему бронювання. Це один із популярних лоукост-операторів, який дозволяє клієнтам швидко купувати квитки через інтуїтивно зрозумілий інтерфейс [3].

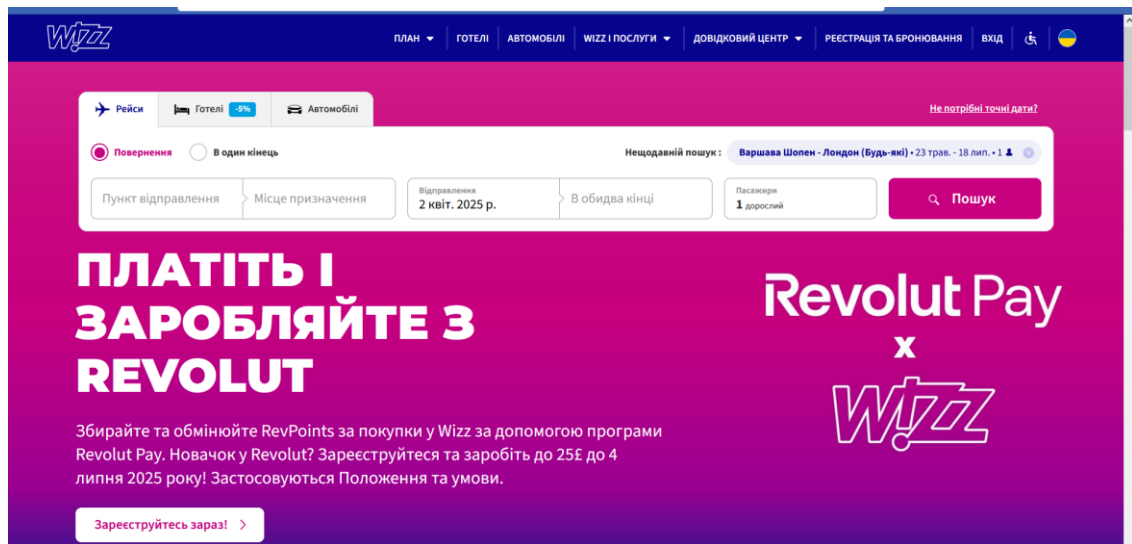


Рис. 1.1 – Головна сторінка сайту Wizz Air

Даний веб-застосунок (рис.1.2) є популярним засобом для бронювання квитків. Він дозволяє користувачам переглянути доступні маршрути, забронювати квиток, вибрати сидіння, оформити багаж. Однак платформа не має розширеної функціональності для адміністратора, що є критично важливими для повноцінної системи управління авіакомпанією. Через це її можна розглядати лише як користувацький інтерфейс для бронювання, а не як комплексне рішення для внутрішньої роботи компанії.

Перевагами Wizz Air є інтуїтивно зрозумілий інтерфейс для бронювання квитків. Динамічна зміна ціна, що адаптується до попиту. Інтеграція з програмами лояльності та CRM-системами.

Недоліками Wizz Air є відсутність підтримки бронювань для корпоративних клієнтів. Обмежена можливість зміни рейсів або комбінування маршрутів. Відсутність API для інтеграції з іншими платформами.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

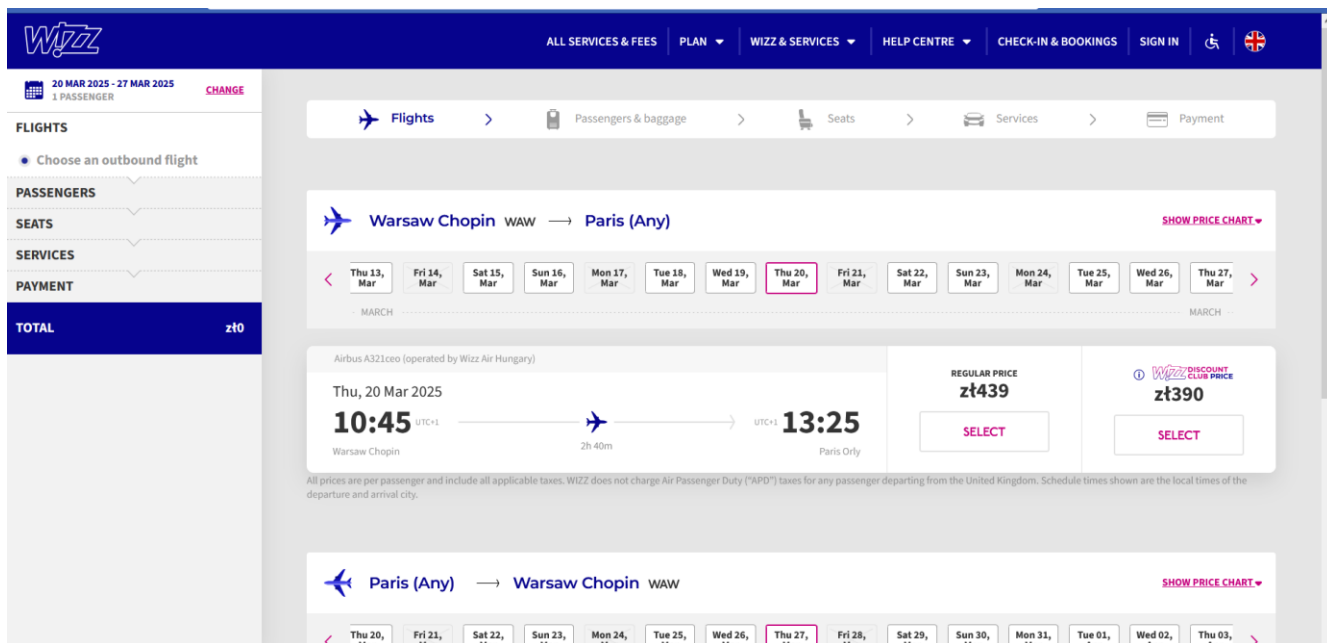


Рис. 1.2 – Сторінка перегляду доступних рейсів на сайті Wizz Air

Ryanair (рис.1.3) – ще одна популярна система бронювання квитків на рейси, що також орієнтована на лоукост-стратегію [4].

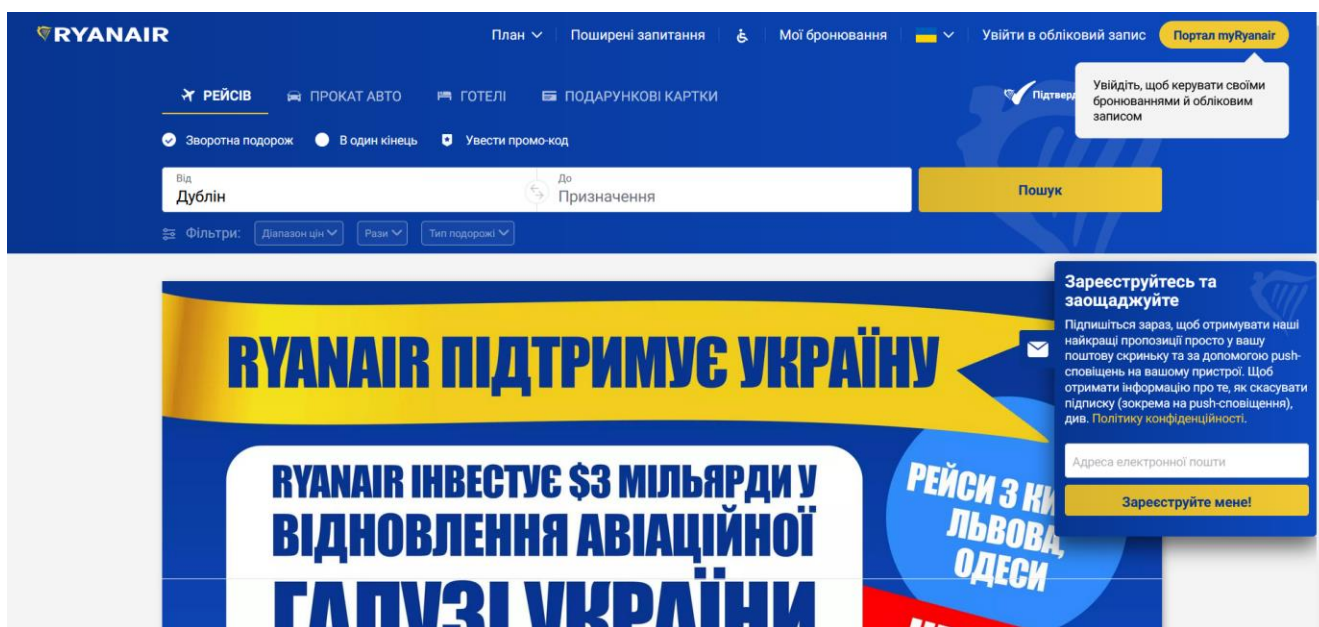


Рис. 1.3 – Головна сторінка сайту Ryanair

До переваг Ryanair можна віднести простий і зручний інтерфейс для швидкого бронювання. Гнучкість у виборі тарифів і додаткових послуг. Можливість перегляду доступних рейсів та вибору місць на борту.

Серед недоліків Ryanair можна окреслити обмежені можливості для керування маршрутом або змін на етапі бронювання. Складність в управлінні великими обсягами запитів на платформі через високу популярність.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		
						12

Lufthansa (рис.1.4) - авіакомпанія з більш комплексною системою управління для пасажирів з додатковими послугами та можливостями [5].

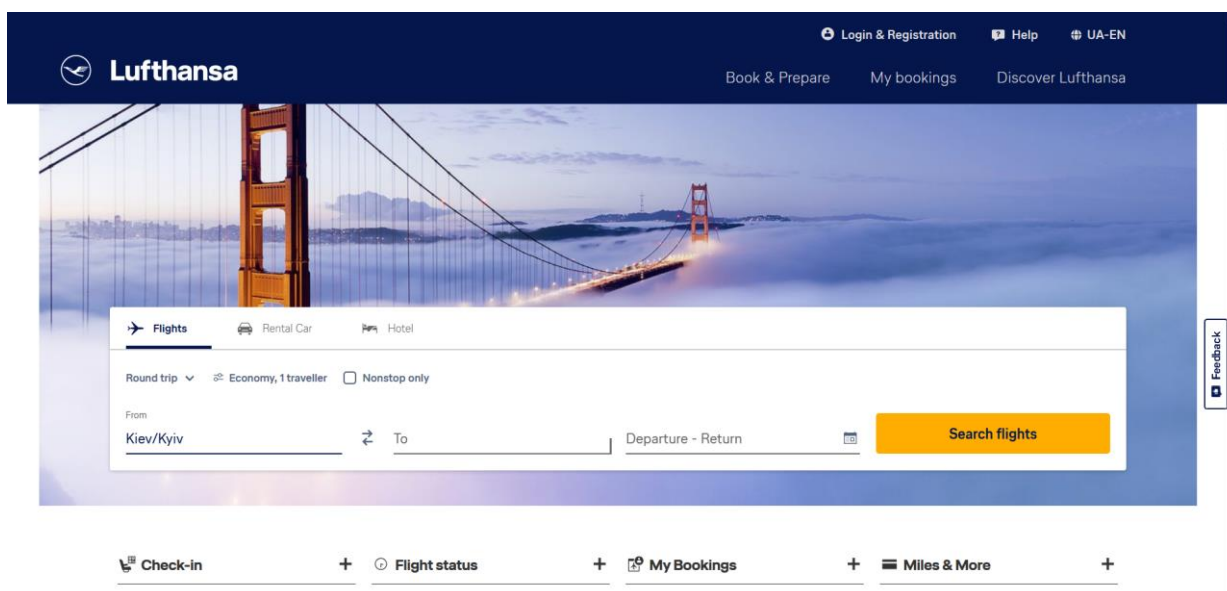


Рис. 1.4 – Головна сторінка сайту Lufthansa

Основні плюси Lufthansa полягають у наявності системи накопичення миль та програми лояльності, що стимулюють повторні покупки. Можливість бронювання квитків, вибору сидінь, оформлення багажу. Інтеграція з іншими платформами, що дозволяє отримувати дані про рейси з різних джерел.

Серед мінусів Lufthansa є складність навігації через велику кількість доступних функцій. Висока вартість квитків у порівнянні з лоукостами. Обмежені можливості для зміни або відміни рейсів без значних додаткових витрат.

Далі варто оглянути аналоги з точки зору персоналу та адміністраторів.

Sabre Airline Solutions (рис.1.5) надає програмне забезпечення для управління авіакомпанією.

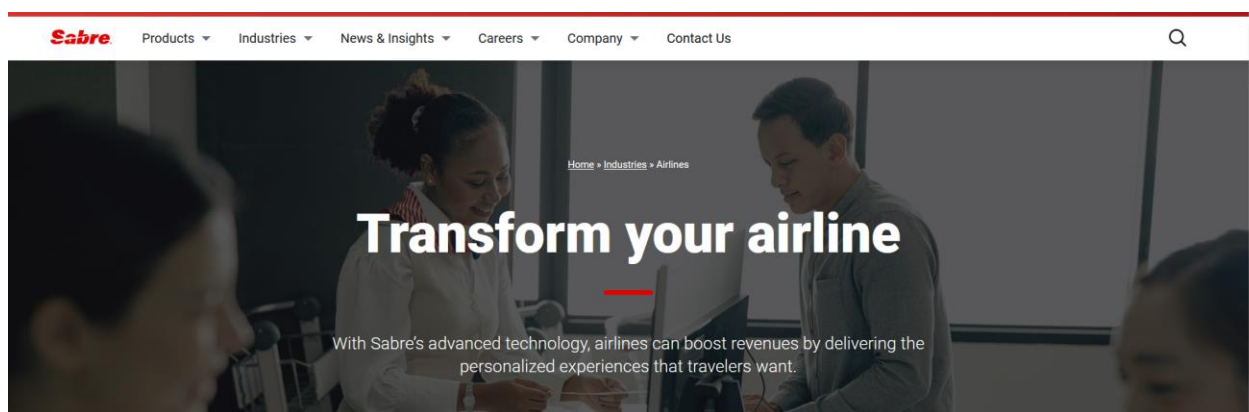


Рис. 1.5 – Головна сторінка сайту Sabre

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

Дана платформа дозволяє керувати бронюваннями, управліннями рейсів, контролювати розклад та управляти персоналом. Також доступний модуль аналіз доходів та бронювання квитків.

Перевагами Sabre Airline є поєднання з іншими програмами авіакомпанії, включаючи CRM та системи управління лояльністю. Підтримка функцій для управління рейсами, персоналом і багажем. Оптимізація маршрутів і розкладів.

А недоліками Sabre Airline є висока вартість ліцензії та складність налаштування. Необхідність висококваліфікованого персоналу для налаштування та обслуговування.

Іншим прикладом є SITA Horizon (рис.1.6) – це система для управління пасажирськими перевезеннями, екіпажем та ресурсами. Вона забезпечує повний цикл обслуговування: від бронювання квитків до контролю посадки та роботи екіпажу.

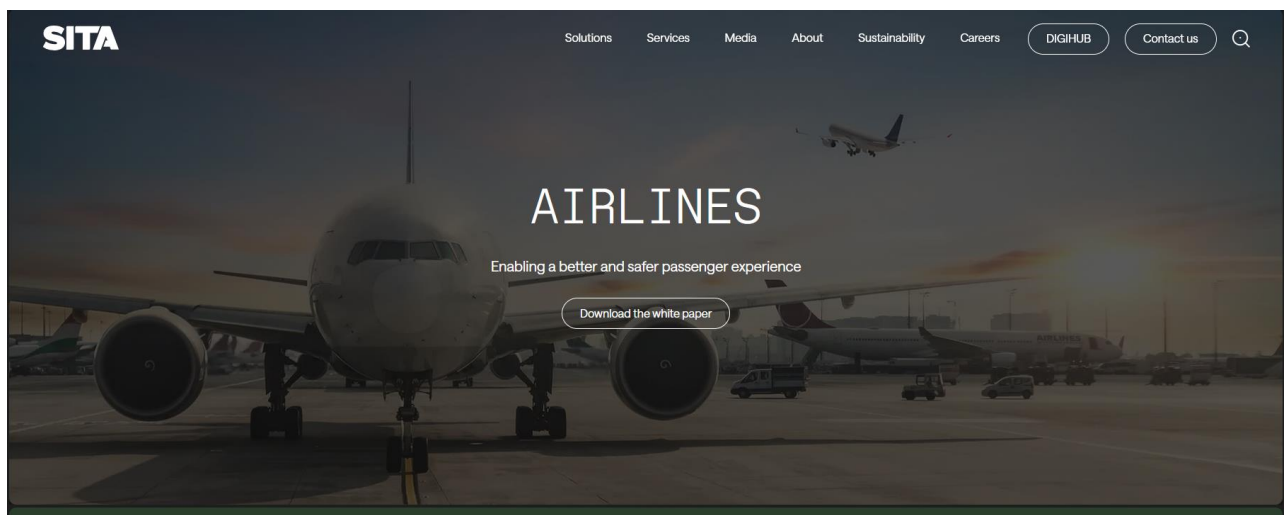


Рис. 1.6 – Головна сторінка сайту SITA

Основні плюси SITA Horizon полягають в автоматизованому управлінні графіками персоналу. Моніторинг ефективності роботи екіпажу. Інтеграція з іншими операційними системами авіакомпанії.

До недоліків SITA Horizon можна віднести важкість у впровадженні та підтримці. Відсутність гнучкої кастомізації під специфічні потреби компаній. Складний інтерфейс для нових користувачів. Також потребує спеціалізованого навчання персоналу та може бути надлишковою для компаній з менш складною структурою.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				14
Змн.	Арк.	№ докум.	Підпис	Дата		

Виходячи з наявних переваг та недоліків, було складено порівняльну таблицю програмних аналогів у таблиці 1.1. Адже з розглянутих популярних систем управління авіакомпанією, кожна з яких має свої переваги та недоліки.

Таблиця 1.1

Таблиця порівняння платформ для управління авіакомпанією

Характеристика	Wizz Air	RyanAir	Lufthansa	Sabre Airline	SITA Horizon	Моя система
Додавання шаблону сидінь літака	-	-	-	-	-	+
Додавання літака	-	-	-	+	+	+
Пошук аеропорту по відкритому апі і його збереження	-	-	-	-	-	+
Додавання шляху для рейсів	-	-	-	+	+	+
Створення рейсу (технічний/не технічний)	-	-	-	+	-	+
Додавання літака на рейс (автоматичне формування квитків)	-	-	-	+	+	+
Додавання робітників на рейс	-	-	-	+	+	+
Зміна статусу рейса	-	-	-	-	-	+
Пошук рейсів по місту і даті	-	+	-	+	+	+
Вибір рейсу для покупки квитка (односторонній і двосторонній)	+	+	+	+	+	+
Вибір місця в літаку на обраний рейс	+	+	+	+	+	+
Вибір багажу на рейс	+	+	+	+	+	+
Введення даних пасажира	+	+	+	+	+	+
Оплата квитка	+	+	+	+	+	+
Перегляд розкладу робітників	-	-	-	+	+	+
Додавання сертифікатів пілотами	-	-	-	+	+	+

Найбільш зручними для пасажирів є Wizz Air і Lufthansa. Вони забезпечують простоту бронювання для користувачів. Системи з більшим функціоналом, такі як Sabre Airline і SITA Horizon, мають велику кількість функцій для управління рейсами, персоналом та багажем, але мають високу складність у використанні.

Розробка власної системи дозволяє поєднати переваги: забезпечити інтуїтивно зрозумілий користувацький інтерфейс і гнучке налаштування внутрішніх бізнес-процесів. Крім того, важливо адаптувати систему під специфіку компанії та автоматизувати операції, зменшуючи навантаження на персонал.

1.3. Вибір архітектури системи управління авіакомпанією

Для створення надійної системи управління авіакомпанією було обрано трирівневу архітектуру. Вона складається з клієнтської частини, серверної частини та бази даних. Саме така модель дозволяє чітко і логічно розділити функціональність та відповідно зменшити зв'язаність компонентів між собою.

Однією з ключових переваг цієї архітектури є мінімальна взаємозалежність між різними компонентами системи, що дає змогу змінювати або оновлювати один з елементів (наприклад, замінити клієнтський інтерфейс або змінити тип бази даних) без значного впливу на інші частини системи. Це особливо важливо для систем де постійно змінюються вимоги, навантаження та вимоги до безпеки. Так клієнтська частина обробляє взаємодію з користувачем, серверна – внутрішню логіку та обробку запитів, а база даних відповідає за збереження інформації та доступ до даних.

У якості мови програмування для серверної частини було обрано мову програмування Java. Серед її головних переваг є: стабільність, масштабованість, продуктивність та підтримка великої кількості інструментів для розробки [6]. Крім того, Java це об'єктно-орієнтована мова програмування [7]. А саме така мова програмування дозволяє створювати систему, кожен елемент якої (інформація про рейси, пасажирів, літаки, аеропорти, тощо) є окремим об'єктом зі своїми визначеними функціями та властивостями.

Загалом ООП – це парадигма програмування, основними принципами якої є інкапсуляція, спадкування, поліморфізм та абстракція [8].

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				16
Змн.	Арк.	№ докум.	Підпис	Дата		

Інкапсуляція – це певний механізм, що дозволяє приховати внутрішню реалізацію об’єкта і надати доступ лише до певних частин через публічні методи. Це зменшує ймовірність помилок та спрощує підтримку коду.

Спадкування дозволяє створювати нові класи на основі вже існуючих. Тобто можна повторно використовувати код, що зменшує дублювання.

Поліморфізм забезпечує можливість використовувати екземпляри різних класів через один інтерфейс.

Абстракція ж дозволяє описувати лише ключові характеристики об’єктів без деталізації їх реалізації.

Тож основною перевагою ООП є можливість додавання нового функціоналу без порушення основної частини програми, що забезпечує чистоту коду [9]. Використання цього підходу дає можливість розділити складну задачу на простіші, які можна окремо розробляти, тестувати та оновлювати.

Крім того, Java має велику екосистему бібліотек і фреймворків, ключовим серед яких є Spring з його розширенням Spring Boot, яке значно спрощує розробку застосунків [10].

Також є Spring Web/Spring MVC для побудови RESTful веб-застосунків.

Spring Data та Hibernate - для роботи з базами даних і Spring Security для безпеки системи.

У якості бази даних для системи було обрано MySQL, як найбільш придатну для даного типу системи [11]. Це тип бази даних, де інформація зберігається в структурованих таблицях [12]. Вони пов’язані між собою через первинні ключі (унікальні ідентифікатори записів) та зовнішні ключі (посилання на записи в інших таблицях). Тому з даними легко працювати навіть за умови складних запитів. Також MySQL легко поєднується з багатьма мовами програмування та відомими фреймворками, завдяки чому вона є гнучким та зручним інструментом.

Для реалізації клієнтської частини системи управління авіакомпанією було вибрано React. Це одна з найпопулярніших JavaScript-бібліотек для створення інтерфейсів користувача [13].

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

Цей список технології для розробки забезпечує побудову сучасної, гнучкої та безпечної системи, що буде підтримувати всі необхідні механізми роботи в авіакомпанії.

1.4. Обґрунтування вибору інструментальних засобів

Головна мета обраного технологічного стеку – створити надійну та гнучку систему. Важливо також аби програма ефективно обробляла великий обсяг даних та підтримувала високу доступність і головне – гарантувати безпеку інформації.

Spring Boot був обраний як основа серверної частини. Він має вбудований сервер Tomcat який забезпечує обробку HTTP-запитів, управління сесіями та підтримку REST API [14, 15]. Це значно пришвидшує процес розгортання додатку. Завдяки цьому систему можна легко перемістити на інші сервери або хмарні платформи, наприклад, у випадку розширення інфраструктури авіакомпанії, тощо.

В якості ORM-рішення було застосовано Hibernate, який дозволяє працювати з реляційними базами даних. Він автоматично перетворює Java класи (моделі) в записи бази даних і навпаки.

Окрім Hibernate, для роботи з реляційними базами даних було використано Spring Data. Це модуль, що спрощує роботу з реляційними базами даних за допомогою специфікації JPA (Java Persistence API) [16]. Тобто Spring Data містить реалізацію простих запитів (CRUD) до бази даних без необхідності написання SQL-коду вручну. Складніші ж запити до бази даних вже пишуться за допомогою JPQL (Java Persistence Query Language),

Інструментом для реалізації безпеки в системі є Spring Security. Саме він забезпечує контроль доступу, автентифікацію та авторизацію користувачів, захист REST API, а також реалізацію ролей користувачів (адміністратор, пасажир, робітник, тощо) [17]. За допомогою конфігурацій і анотацій можна гнучко налаштувати всі перевірки та забезпечити конфіденційність та цілісність даних.

Доповнює стек технології – Spring Web (а також Spring MVC), що забезпечує реалізацію архітектури Model-View-Controller.

Виходячи з наявних переваг та недоліків, було складено порівняльну таблицю (табл. 1.2).

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				18
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця порівняння фреймворків

Фреймворк	Перваги	Недоліки
Spring Boot	Мультиплатформеність	Вища складність налаштування
	Підтримка різних баз даних і інструментів	
	Наявність екосистема для мікросервісів	Більші вимоги до пам'яті для великих проектів
	Масштабованість	
ASP.NET	Інтеграція з екосистемою Microsoft	Обмежена мультиплатформеність
	Висока продуктивність	
	Зручне середовище розробки (Visual Studio)	
Node.js	Висока продуктивність при роботі з асинхронними операціями	Менш ефективним при обробці складних бізнес-логік
	Легкість у створенні RESTful API	

Для клієнтський інтерфейсу – React дозволяє створювати динамічні веб-інтерфейси сторінки яких оновлюються без повного перезавантаження [18].

Також варто відмітити, одну з ключових переваг, технологію DOM. Вона дозволяє оновлювати лише ті частини сторінки, які були змінені, а не всю відразу. Саме тому отриманий інтерфейс буде швидким та плавним навіть при великому обсязі даних. Крім того React інтегрується з RESTful API, який реалізовується на серверній частині через Spring Boot. Це забезпечує зручний обмін даними між фронтендом і бекендом.

Тож React це оптимальний вибір для побудови сучасного, швидкого та гнучкого веб-додатку для системи управління авіакомпанією.

Також додатково варто зазначити використання CSS – мову стилів, яка використовується для опису зовнішнього вигляду HTML-документів. Вона дозволяє змінювати кольори, шрифти, розміри, відступи, розташування елементів, а також створювати адаптивні інтерфейси та анімації. CSS відокремлює структуру веб-сторінки від її оформлення, що робить код чистішим і легшим для підтримки.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				19
Змн.	Арк.	№ докум.	Підпис	Дата		

При розробці системи управління авіакомпанією значним етапом є правильний вибір типу бази даних, оскільки вона відіграє ключову роль у зберіганні, обробці та захисті інформації.

Було обрано реляційну базу даних – MySQL. Адже крім наведених в попередньому розділі характеристик, наявна підтримка транзакцій, бо, наприклад, при бронюванні квитка одразу змінюються декілька таблиць (пасажери, квитки, статус місця) [19]. Важливо, щоб ці зміни виконувалися цілісно та без помилок.

Окремо варто виділити, що доступ до бази даних може бути здійснений з різних пристроїв (через клієнтсько-серверну архітектуру), що означає, що бд зберігається централізовано на сервері.

Підсумовуючи, можна сказати, що реляційна база даних найкраще підходить для системи управління авіакомпанією, так як система працює з великим обсягом структурованих даних, таких як рейси, бронювання, тарифи та розклади.

На противагу, NoSQL не підтримує складні транзакції, а це важливо для цілісності операцій. Також у таких системах складно реалізувати зв'язки між різними типами даних (наприклад, рейси і бронювання). До того ж NoSQL бази не гарантують ізольованість і довговічність даних.

Виходячи з наявних переваг та недоліків, було складено порівняльну таблицю баз даних (табл. 1.3).

Таблиця 1.3

Таблиця порівняння баз даних

Назва	Тип	Переваги	Недоліки
MySQL	Реляційна	Підтримка складних транзакцій	Обмежені можливості для зберігання неструктурованих даних
		Висока цілісність даних	
		Простота налаштування та використання	
		Широка документація	
PostgreSQL	Реляційна	Підтримка складних запитів	Складна в налаштуванні
		Підтримка великих даних та високої надійності	

Продовження таблиці 1.3

Назва	Тип	Переваги	Недоліки
MongoDB	NoSQL	Гнучкість для зберігання неструктурованих даних	Не підтримує складні транзакції
		Легке масштабування	Проблеми з цілісністю даних при великих навантаженнях і складних зв'язках між даними
		Висока швидкість для простих операцій	
Oracle DB	Реляційна	Висока продуктивність і надійність	Висока вартість ліцензії
		Велика кількість функціональних можливостей	Складність у налаштуванні та адмініструванні

Робота з базою даних здійснюється через MySQL Workbench. Він дозволяє спроектувати структуру бази даних, виконувати SQL-запити, створювати ER-діаграми, керувати доступом, аналізувати продуктивність.

Розробка вебдодатку здійснюється у середовищі JetBrains, яке є потужним редактором з підтримкою вебтехнологій, вбудованими інструментами для налагодження коду та інтеграцією з Git.

Висновок до першого розділу

У першому розділі було проведено аналіз аналогів, їх переваги та недоліки. Це дозволило чітко визначити функціональні вимоги до майбутньої системи, враховуючи всі потреби. На основі цих даних було обрано стек технологій та інструментальні засоби, за допомогою яких можна реалізувати систему управління авіакомпанією.

Також вибір технологій, таких як Java з використанням Spring Boot, React для клієнтської частини, а також реляційна база даних MySQL був обґрунтованим.

Таким чином, у результаті першого розділу створено основу для подальшої розробки, яка забезпечить створення системи управління авіакомпанією.

Опис діаграми варіантів використання наведено в таблицях

Таблиця 2.1

Опис прецеденту «Вхід/Реєстрація»

Назва варіанту використання	Вхід/реєстрація
Основні актори	Адміністратор, пілот, бортпровідник, пасажир
Короткий опис	Новий пасажир може зареєструватися, а зареєстровані користувачі (пасажир, працівники, адміністратори) — увійти до системи.

Таблиця 2.2

Опис прецеденту «Вхід у кабінет/зміна власних даних»

Назва варіанту використання	Вхід у кабінет/зміна власних даних
Основні актори	Адміністратор, пілот, бортпровідник, пасажир
Короткий опис	Пасажир, адміністратор, працівник (пілоти і бортпровідники) можуть зайти до кабінету та змінити свої дані: ім'я, дату народження, пароль.

Таблиця 2.3

Опис прецеденту «Додавання конфігурації розміщення сидінь»

Назва варіанту використання	Додавання конфігурації розміщення сидінь
Основні актори	Адміністратор
Короткий опис	Адміністратор створює шаблон розміщення сидінь в салоні літака, вказуючи: загальну кількість місць, кількість рядів бізнес-класу, схему сидінь у ряді (де числа вказують кількість сидінь між проходами)

Таблиця 2.4

Опис прецеденту «Додавання літаку»

Назва варіанту використання	Додавання літаку
Основні актори	Адміністратор
Короткий опис	Адміністратор додає новий літак, вказує його конфігурацію сидінь та домашній аеропорт

Таблиця 2.5

Опис прецеденту «Додавання аеропорту»

Назва варіанту використання	Додавання аеропорту
Основні актори	Адміністратор
Короткий опис	Адміністратор вводить ІАТА-код, система отримує дані з відкритого API, після чого аеропорт зберігається в базу.

Таблиця 2.6

Опис прецеденту «Додавання працівника»

Назва варіанту використання	Додавання працівника
Основні актори	Адміністратор
Короткий опис	Адміністратор додає працівника (пілота або бортпровідника).

Таблиця 2.7

Опис прецеденту «Створення нового маршрута»

Назва варіанту використання	Створення нового маршрута
Основні актори	Адміністратор
Короткий опис	Адміністратор створює новий маршрут, вказуючи аеропорт відправлення і прибуття. Автоматично формується зворотній маршрут.

Таблиця 2.8

Опис прецеденту «Створення нового рейсу»

Назва варіанту використання	Створення нового рейсу
Основні актори	Адміністратор
Короткий опис	Адміністратор створює рейс, прив'язує літак, призначає екіпаж та встановлює дату і час. Автоматично формується зворотній рейс.

Таблиця 2.9

Опис прецеденту «Перегляд списку рейсів/літаків/працівників/аеропортів»

Назва варіанту використання	Перегляд списку рейсів/літаків/працівників/аеропортів
Основні актори	Адміністратор
Короткий опис	Адміністратор перегляди список всіх наявних літаків, працівників, тощо

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				24
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 2.10

Опис прецеденту «Пошук рейсів»

Назва варіанту використання	Пошук рейсів
Основні актори	Пасажир
Короткий опис	Пасажир шукає рейс за містом відправлення, містом прибуття та датою, також може знайти зворотній рейс

Таблиця 2.11

Опис прецеденту «Перегляд та вибір рейсів»

Назва варіанту використання	Перегляд та вибір рейсів
Основні актори	Пасажир
Короткий опис	Пасажир переглядає рейси за вибраними критеріями і обирає для купівлі квитка

Таблиця 2.12

Опис прецеденту «Купівля квитка»

Назва варіанту використання	Купівля квитка
Основні актори	Пасажир
Короткий опис	Пасажир купує квиток на рейс. Що включає в себе: вибір місця в літаку, вибір багажу чи його відсутність, введення паспортних даних пасажирів, оплата, отримання квитка на e-mail

Таблиця 2.13

Опис прецеденту «Перегляд своїх квитків»

Назва варіанту використання	Перегляд своїх квитків
Основні актори	Пасажир
Короткий опис	Пасажир може переглянути куплені квитки

Таблиця 2.14

Опис прецеденту «Перегляд розкладу своїх рейсів»

Назва варіанту використання	Перегляд розкладу своїх рейсів
Основні актори	Працівник (пілот, борпроводник)
Короткий опис	Працівник може переглянути розклад рейсів на які його було назначено

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				25
Змн.	Арк.	№ докум.	Підпис	Дата		

Таблиця 2.15

Опис прецеденту «Додавання сертифікатів на літак»

Назва варіанту використання	Додавання сертифікатів на літак
Основні актори	Працівник (пілот)
Короткий опис	Пілот додає сертифікат, що підтверджує право на керування певною моделлю літака.

Таблиця 2.16

Опис прецеденту «Перегляд сертифікатів пілотів/їх підтвердження»

Назва варіанту використання	Перегляд сертифікатів пілотів/їх підтвердження
Основні актори	Адміністратор
Короткий опис	Адміністратор переглядає сертифікати, що були додані пілотами та підтверджує їх або відхиляє

Функціональні вимоги

Визначають набір можливостей, реалізованих у системі. Авторизація та реєстрація користувачів з подальшим призначенням ролі (пасажир або адміністратор).

Управління літаками. Додавання нового літака із зазначенням конфігурації сидінь, створення шаблону конфігурації розміщення сидінь, можливість деактивації літака якщо він вийшов з ладу чи в ремонті. Додавання нового аеропорту.

Управління працівниками. Додавання нового працівника (пілота, бортпровідника) із вказанням e-mail, ролі та домашнього аеропорту, також перегляд списку всіх працівників та призначення працівників на рейси. Додавання пілотом сертифіката на право керування певною моделлю літака. Перегляд сертифікатів адміністратором. Підтвердження або відхилення сертифікатів адміністратором.

Управління маршрутами та рейсами. Створення маршруту між двома аеропортами з автоматичним формуванням зворотного маршруту. Створення нового рейсу з прив'язкою літака, екіпажу та встановленням дати/часу. Автоматичне створення зворотного рейсу. Перегляд списку рейсів.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				26
Змн.	Арк.	№ докум.	Підпис	Дата		

Пошук та бронювання квитків (для пасажирів). Пошук рейсів за містом відправлення, прибуття та датою. Вибір рейсу зі списку. Бронювання та купівля квитка, що включає в себе вибір місця, вибір наявності багажу та його типу, введення паспортних даних, оплата квитка та отримання його на е-mail.

Нефункціональні вимоги

Впливають на зручність, продуктивність і надійність системи.

Інтерфейс має бути інтуїтивно зрозумілим і забезпечувати мінімальний час на навчання – до 2 годин для нових користувачів. Система повинна бути стабільною, з рівнем доступності не менше ніж 92% часу, середнім часом безвідмовної роботи до 28 днів та максимально допустимою нормою помилок не більше 1 на 20 тисяч рядків коду. Щодо продуктивності – система має підтримувати до 50 одночасно активних користувачів без зниження швидкодії. Масштабованість передбачає здатність витримувати зростання навантаження при збільшенні кількості пасажирів або рейсів.

Характеристика об'єкта комп'ютеризації - авіакомпанія, діяльність якої включає керування парком літаків, координацію персоналу, обслуговування клієнтів, організацію рейсів і продаж квитків. Всі ці процеси потребують автоматизації для підвищення ефективності роботи, мінімізації людського фактора та зниження операційних витрат.

Система має дозволяти легко інтегруватися з іншими сервісами, такими як платіжні системи, зовнішні API.

Має бути забезпечений захист даних користувачів та комерційної інформації за допомогою шифрування. Контроль доступу до різних функцій системи залежно від ролі користувача.

Крім того чудовим доповненням була б можливість ведення журналів роботи та логів помилок для швидкого виявлення і усунення несправностей.

Таким чином, система управління авіакомпанією реалізує комплексний функціонал, спрямований на задоволення потреб усіх категорій користувачів, дотримання бізнес-вимог і забезпечення надійної роботи в умовах реального навантаження.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

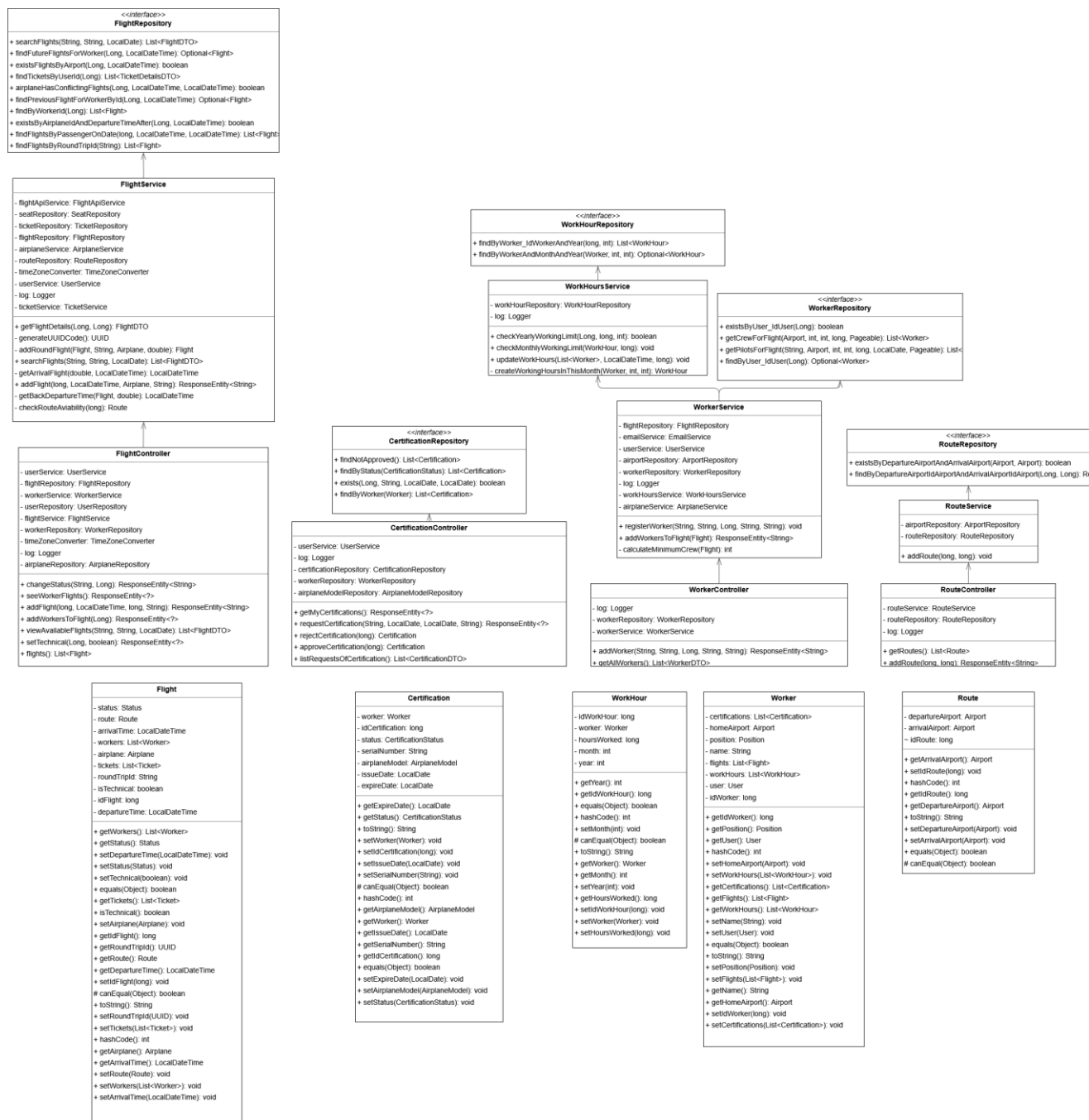


Рис. 2.2 – Діаграма класів системи за паттерном MVC (наявні не всі класи)

Дана діаграма класів частково описує структуру програми, реалізовану за принципами архітектури Model-View-Controller (MVC). Загалом система поділена на чотири головні шари: моделі, контролери, сервіси та репозиторії [22, с. 437]. Кожен із цих шарів виконує свою функцію в обробці даних, бізнес-логіці та взаємодії з користувачем або іншими компонентами.

Моделі - це класи, що описують бізнес-сутності системи: літак (Airplane),

		Кобилко М. П.			Арк.
		Вакалюк Т. А.			28
Змн.	Арк.	№ докум.	Підпис	Дата	

рейс (Flight), маршрут (Route), працівник (Worker), квиток (Ticket), сертифікат (Certification), робочі години (WorkHour) тощо. З метою покращення читабельності на діаграмі наведено лише частину моделей, що входять до складу системи.

Контролери (AirplaneController, FlightController, CertificationController, WorkerController, TicketController, RouteController) реалізують зовнішній інтерфейс взаємодії. Вони приймають запити, в даній системі також делегують обробку проміжному шару – сервісам і формують відповідь. Наприклад, AirplaneController відповідає за активацію/деактивацію літаків, додавання нових моделей, FlightController керує створенням рейсів і зміною їх статусу, а TicketController – за логіку вибору місць та перегляду квитків.

Контролери напряму не містять складної бізнес-логіки, вони виконують роль посередника між HTTP-запитами та сервісним шаром. Іноді контролери використовують DTO класи (Data Transfer Object). В системі це, наприклад, FlightDTO, CertificationDTO, щоб передавати лише потрібну інформацію назовні, не розкриваючи внутрішню структуру моделей.

DTO є структурним патерном, що дозволяє зменшити кількість запитів і зробити передачу даних між клієнтом і сервером більш контрольованою та безпечною [23].

Сервісний шар (AirplaneService, FlightService, TicketService, тощо), про який було згадано вище, відповідає за бізнес-логіку системи. Він обробляє запити з контролерів, взаємодіє з репозиторіями, виконує перевірки, створення об'єктів, їх зміну або валідацію. Наприклад, FlightService відповідає за створення та пошук рейсів, генерацію UUID-кодів, перевірку доступності маршрутів; AirplaneService перевіряє конфлікти між рейсами, додає нові літаки, TicketService генерує квитки, перевіряє доступність місць, проводить бронювання.

Сервіси не взаємодіють з HTTP напряму. Вони забезпечують інкапсуляцію складної логіки та полегшують тестування й підтримку коду.

Репозиторії (AirplaneRepository, FlightRepository, CertificationRepository, тощо) - це інтерфейси, що реалізують шар доступу до даних. Вони абстрагують взаємодію з базою даних за допомогою Spring Data JPA.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				29
Змн.	Арк.	№ докум.	Підпис	Дата		

Наприклад, FlightRepository дозволяє шукати рейси за ідентифікатором пасажирів, дати, ID екіпажу тощо. TicketRepository перевіряє наявність бронювання, кількість заброньованих місць та конфлікти рейсів для пасажирів.

Репозиторії не містять логіки обробки даних, лише доступ до них. Вони дозволяють швидко реалізовувати складні запити через ключові слова в іменах методів або за допомогою JPQL.

Варто зазначити, що використання архітектури MVC у цьому випадку є доцільнішим, ніж, наприклад, MVVM (Model–View–ViewModel). MVVM є більш актуальним для клієнтських застосунків з графічною взаємодією, де ViewModel виконує роль посередника між графічним інтерфейсом і моделлю. У випадку з веб-додатком для керування авіакомпанією, паттерн MVC забезпечує кращу структурованість, простоту тестування і ефективну обробку запитів.

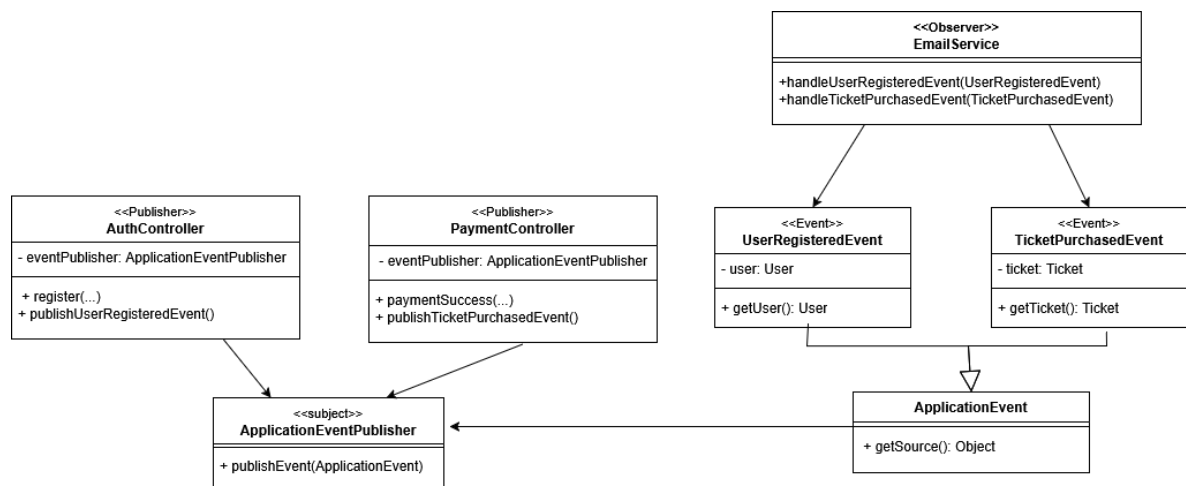


Рис. 2.3 – Діаграма класів паттерну Observer

У представленій UML-діаграмі зображено реалізацію шаблону проектування Observer (Спостерігач). Цей паттерн реалізує механізм в якому один об'єкт (subject) повідомляє інші об'єкти (observers) про зміну свого стану [22, с. 217]. Варто зазначити, що об'єкт не знає хто ці спостерігачі. Це дозволяє досягти низького зв'язування між компонентами системи, а це один з принципів SOLID.

На зображеній діаграмі (рис. 2.3) клас ApplicationEventPublisher реалізує роль subject у шаблоні Observer. Він відповідає за публікацію події, які потім будуть оброблятися тими класами, що підписані на ці події. Це ядро системи подій у Spring.

Два класи: AuthController та PaymentController є так званими видавцями подій і позначені відповідно як «Publisher». Так AuthController створює подію UserRegisteredEvent, коли користувач реєструється, а PaymentController створює TicketPurchasedEvent, коли користувач здійснює покупку квитка на рейс. Тобто класи UserRegisteredEvent і TicketPurchasedEvent є подіями, які наслідують ApplicationEvent. Вони містять інформацію, яка буде передана так званим підписникам.

Головним спостерігачем (observer) у системі є EmailService. Він реагує на події. Коли одна з цих подій відбувається і вона «публікується» через ApplicationEventPublisher, Spring автоматично викликає відповідний метод у EmailService. Завдяки цьому EmailService не залежить від контролерів, і логіку відправки листів можна легко змінювати окремо від решти системи. В результаті користувачі отримують на пошту лист реєстрації та лист з квитком.

2.2 Розробка бази даних системи

Одним із ключових етапів проектування програмного забезпечення є розробка база даних. Адже саме вона дозволяє зберігати, обробляти та зв'язувати між собою всі необхідні об'єкти авіаційної інфраструктури.

Розробка бази даних для системи управління авіакомпанією є основою для забезпечення ефективного зберігання та обробки великих обсягів інформації про пасажирів, рейси, літаки, аеропорти та інші сутності. Правильна організація таблиць і встановлення зв'язків між ними дозволяє забезпечити надійність і швидкість обробки запитів. Система використовує реляційну модель даних, що дозволяє забезпечити гнучкість при роботі з різними типами запитів, таких як фільтрація, сортування, обчислення ціни квитків, моніторинг робочого часу персоналу та інші важливі операції.

Ще одним важливим аспектом є забезпечення масштабованості та гнучкості системи. Структура бази даних передбачає можливість її розширення, що є критично важливим для розвитку авіакомпанії.

База даних (рис.2.2) реалізована за допомогою системи управління базами даних MySQL, що є зручною, масштабованою та відкритою платформою.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

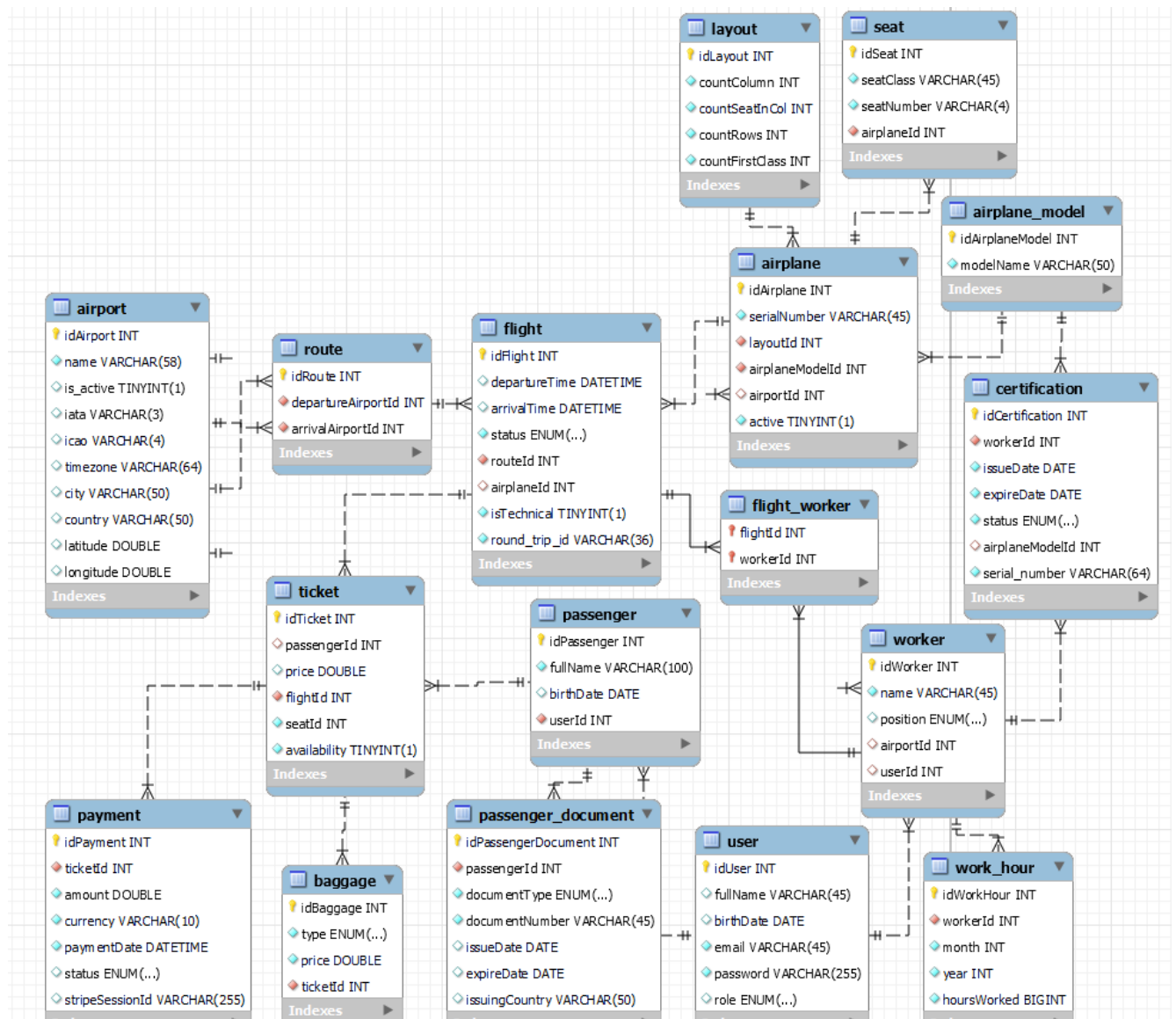


Рис. 2.4 – База даних системи управління авіакомпанії

Таблиця User (табл. 2.17) зберігає інформацію про користувачів системи.

Таблиця 2.17

Структура полів таблиці User

Назва поля	Тип даних	ПК	ЗК	Опис
idUser	INT	+	-	ID користувача
fullName	VARCHAR(45)	-	-	Повне ім'я користувача
birthDate	DATE	-	-	Дата народження
email	VARCHAR(45)	-	-	Емейл користувача
password	VARCHAR(255)	-	-	Пароль користувача
role	ENUM(ADMIN, WORKER, PASSANGER)	-	-	Роль користувача

Таблиця Passenger (табл. 2.18) призначена для зберігання персональних даних пасажирів, які здійснили купили квиток на рейс. Кожен запис у таблиці відповідає одному пасажиру та містить основну інформацію, яка необхідна для ідентифікації особи.

Таблиця 2.18

Структура полів таблиці Passenger

Назва поля	Тип даних	ПК	ЗК	Опис
idPassenger	INT	+	-	ID користувача
fullName	VARCHAR(100)	-	-	Повне ім'я пасажирів
birthDate	DATE	-	-	Дата народження
userId	INT	-	+	Зв'язок з таблицею user

Таблиця worker (табл. 2.19) містить інформацію про працівників авіакомпанії (пілотів і бортпроводників). Ці дані використовуються для управління розкладом, формування екіпажів, контролю персоналу та призначення робочих змін. Таблиця містить ім'я, посаду, закріплення аеропортів працівників.

Таблиця 2.19

Структура полів таблиці Worker

Назва поля	Тип даних	ПК	ЗК	Опис
idWorker	INT	+	-	ID працівника
name	VARCHAR(45)	-	-	Ім'я працівника
position	ENUM(PILOT, CREWMEMBER)	-	-	Посада (PILOT, CREWMEMBER)
airportId	INT	-	+	Зв'язок з таблицею airport. Домашній аеропорт працівника
userId	INT	-	+	Зв'язок з таблицею user

Таблиця airport (табл. 2.20) зберігає інформацію про аеропорти, що використовуються в системі. У таблиці зберігаються назви аеропортів, географічне розташування, часові пояси та авіаційні коди. Додатково передбачене поле is_active, яке вказує, чи функціонує даний аеропорт на поточний момент.

Таблиця 2.20

Структура полів таблиці Airport

Назва поля	Тип даних	ПК	ЗК	Опис
idAirport	INT	+	-	ID аеропорту
name	VARCHAR(58)	-	-	Назва аеропорту
is_active	TINYINT(1)	-	-	Стан аеропорту (активний/неактивний)
iata	VARCHAR(3)	-	-	IATA код
icao	VARCHAR(4)	-	-	ICAO код
timezone	VARCHAR(64)	-	-	Часовий пояс
city	VARCHAR(50)	-	-	Місто
country	VARCHAR(50)	-	-	Країна
latitude	DOUBLE	-	-	Географічна широта
longitude	DOUBLE	-	-	Географічна довгота

Таблиця layout (табл. 2.21) описує схему посадки в літаку, включаючи кількість рядів, колон, місць у колоні та кількість місць першого класу. Ця таблиця дає змогу стандартизувати конфігурації літаків.

Таблиця 2.21

Структура полів таблиці Layout

Назва поля	Тип даних	ПК	ЗК	Опис
idLayout	INT	+	-	ID конфігурації літака
countColumn	INT	-	-	Кількість колонок сидінь у ряду
countSeatICol	INT	-	-	Кількість сидінь у кожній колонці
countRows	INT	-	-	Кількість рядів у літаку
countFirstClass	INT	-	-	Кількість сидінь першого класу

Таблиця AirplaneModel (табл. 2.22) зберігає моделі літаків у системі.

Таблиця 2.22

Структура полів таблиці AirplaneModel

Назва поля	Тип даних	ПК	ЗК	Опис
idAirplaneModel	INT	+	-	ID моделі літака
modelName	VARCHAR(50)	-	-	Назва моделі

Таблиця airplane (табл. 2.23) містить інформацію про кожен літак: серійний номер, посилання на таблицю моделі та схему розміщення сидінь. Також є статус активності літака.

Таблиця 2.23

Структура полів таблиці Airplane

Назва поля	Тип даних	ПК	ЗК	Опис
idAirplane	INT	+	-	ID літака
serialNumber	VARCHAR(45)	-	-	Серійний номер
layoutId	INT	-	+	Зв'язок з таблицею layout
airplaneModelId	INT	-	+	Зв'язок з таблицею модель літака (airplane_model)
airportId	INT	-	+	Домашній аеропорт літака
active	TINYINT(1)	-	-	Статус активності літака

Таблиця route (табл. 2.24) визначає маршрут рейсу шляхом вказання аеропорту відправлення та прибуття.

Таблиця 2.24

Структура полів таблиці Route

Назва поля	Тип даних	ПК	ЗК	Опис
idRoute	INT	+	-	ID маршруту
departureAirportId	INT	-	+	Аеропорт відправлення Зв'язок з таблицею airport
arrivalAirportId	INT	-	+	Аеропорт прибуття Зв'язок з таблицею (airport)

Таблиця flight (табл. 2.25) зберігає інформацію про всі рейси авіакомпанії. Вона є однією з ключових таблиць у структурі бази даних, оскільки відображає кожну окрему подію вильоту та прибуття літака, пов'язану з конкретним маршрутом, літаком та працівниками.

Таблиця 2.25

Структура полів таблиці Flight

Назва поля	Тип даних	ПК	ЗК	Опис
idFlight	INT	+	-	ID рейсу

Назва поля	Тип даних	ПК	ЗК	Опис
departureTime	DATETIME	-	-	Час відправлення
arrivalTime	DATETIME	-	-	Час прибуття
status	ENUM ('ON_TIME', 'DELAYED','CANCELED')	-	-	Статус рейсу
routeId	INT	-	+	Маршрут
airplaneId	INT	-	+	Літак
isTechnical	TINYINT(1)	-	-	Чи технічний рейс
round_trip_id	VARCHAR(36)	-	-	ID для ідентифікації пари рейсів

Таблиця ticket (табл. 2.26) представляє квитки, які містять ціну, пасажирів якому належить квиток, номер місця та пов'язаний рейс. Таким чином можна точно визначити, хто і де сидить у конкретному рейсі.

Таблиця 2.26

Структура полів таблиці Ticket

Назва поля	Тип даних	ПК	ЗК	Опис
idTicket	INT	+	-	ID квитка
passengerId	INT	-	+	Пасажир
price	DOUBLE	-	-	Ціна квитка
flightId	INT	-	+	Рейс
seatId	INT	-	+	Місце
availability	TINYINT(1)	-	-	Доступність квитка

Таблиця baggage (табл. 2.27) описує багаж, прив'язаний до квитка. Вказується вага, розмір, а також номер відповідного квитка.

Таблиця 2.27

Структура полів таблиці Baggage

Назва поля	Тип даних	ПК	ЗК	Опис
idBaggage	INT	+	-	ID багажу
type	ENUM(SMALL, MEDIUM, LARGE)	-	-	Тип багажу
price	DOUBLE	-	-	Вартість
ticketId	INT	-	+	Квиток

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		36

Таблиця certifications (табл. 2.28) фіксує ліцензію пілотів на керування конкретних літаків. Включає дати видачі та закінчення дії, а також статус (підтверджено чи не підтверджено адміністратором).

Таблиця 2.28

Структура полів таблиці Certifications

Назва поля	Тип даних	ПК	ЗК	Опис
idCertification	INT	+	-	ID сертифікату
workerId	INT	-	+	Працівник
issueDate	DATE	-	-	Дата видачі
expireDate	DATE	-	-	Дата закінчення дії
airplaneModelId	INT	-	+	Модель літака
status	ENUM('PENDING', 'APPROVED', 'REJECTED')	-	-	Статус сертифікату
serial_number	VARCHAR(64)	-	-	Номер сертифікату

Таблиця flight_worker (табл. 2.29) – зв’язувальна таблиця між flight та worker і слугує пов’язувальним елементом для зв’язку багато до багатьох.

Таблиця 2.29

Структура полів таблиці flight_worker

Назва поля	Тип даних	ПК	ЗК	Опис
flightId	INT	+	+	Рейс
workerId	INT	+	+	Працівник

Таблиця PassengerDocument (табл. 2.30) містить інформацію про документ пасажирів.

Таблиця 2.30

Структура полів таблиці PassengerDocument

Назва поля	Тип даних	ПК	ЗК	Опис
idPassengerDocument	INT	+	-	ID сертифікату
passengerId	INT	-	+	Пасажир
documentType	ENUM('PASSPORT', 'ID_CARD')	-	-	Тип документа
documentNumber	VARCHAR(100)	-	-	Зашифрований номер документа
issueDate	DATE	-	-	Дата видачі

Назва поля	Тип даних	ПК	ЗК	Опис
expireDate	DATE	-	-	Дата закінчення дії
issuingCountry	VARCHAR(50)	-	-	Країна видачі

Таблиця Payment (табл. 2.31) зберігає інформацію про транзакції оплати квитків. А саме: сума, валюта, коли відбулася оплата, а також який статус має платіж (оплачено, скасовано або очікує підтвердження). Крім того, зберігається ідентифікатор сесії в платіжній системі Stripe, що дає змогу перевірити або повторити платіж у разі потреби.

Таблиця 2.31

Структура полів таблиці Payment

Назва поля	Тип даних	ПК	ЗК	Опис
idPayment	INT	+	-	ID платежу
ticketId	INT	-	+	Квиток
amount	DOUBLE	-	-	Сума
currency	VARCHAR(10)	-	-	Валюта
paymentDate	DATETIME	-	-	Дата оплати
status	ENUM ('PENDING', 'PAID', 'FAILED', 'CANCELED')	-	-	Статус оплати
stripeSessionId	VARCHAR(255)	-	-	ID сесії Stripe

Таблиця seat (табл. 2.32) – таблиця з усіма сидіннями на борту кожного літака. Зберігає клас місця, його номер, а також статус доступності.

Таблиця 2.32

Структура полів таблиці Seat

Назва поля	Тип даних	ПК	ЗК	Опис
idSeat	INT	+	-	ID сидіння
seatClass	Varchar(45)	-	-	Клас сидіння (економ, бізнес, перший тощо)
seatNumber	Varchar(4)	-	-	Номер місця (наприклад, 12A)
airplaneId	INT	-	+	Літак

Таблиця work_hour (табл. 2.33) дозволяє вести облік відпрацьованих годин кожного працівника в конкретний місяць та рік.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		38

Структура полів таблиці Work_Hour

Назва поля	Тип даних	ПК	ЗК	Опис
idWorkHour	INT	+	-	ID запису годин
workerId	INT	-	+	Працівник
month	INT	-	-	Місяць
year	INT	-	-	Рік
hoursWorked	BIGINT	-	-	Кількість відпрацьованих годин

Усі таблиці бази даних пов'язані між собою за допомогою зовнішніх ключів, що гарантує цілісність даних та запобігає логічним помилкам, як-от видалення аеропорту, який використовується в маршруті.

У перспективі база даних може бути розширена. Наприклад можна додати модулі статистики, історії змін або навіть модуля лояльності для пасажирів. Завдяки продуманій структурі таблиць і коректному застосуванню зовнішніх ключів, такі доповнення не вимагатимуть великих змін в архітектурі.

2.3 Проектування та реалізація алгоритмів роботи системи

Проектування алгоритмів для системи управління авіакомпанією здійснено для спрощення користування, адже відповідальність за виконання складних процесів на адміністраторі буде тоді мінімальна. У контексті системи авіакомпанії важливими є алгоритми для роботи з базою даних, а також для інтеграції з платіжними системами для обробки фінансових транзакцій. Крім того, алгоритми мають враховувати різні сценарії поведінки користувачів: пасажирів, адміністрації та персоналу, забезпечуючи відповідний рівень доступу та функціональності для кожної ролі.

Процес проектування алгоритмів вимагає ретельного визначення взаємодії між компонентами системи: від користувацького інтерфейсу до обробки запитів сервером і взаємодії з базою даних. Наприклад, при бронюванні рейсу, система повинна коректно обробляти запити на вибір місць, враховуючи їх доступність і дані про літаки та рейси.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				39
Змн.	Арк.	№ докум.	Підпис	Дата		

Одним із ключових бізнес-процесів, доступних авторизованому користувачу, є покупка квитка. На рисунку 2.3 зображено діаграму активності цього процесу. Він розпочинається з перегляду доступних рейсів.

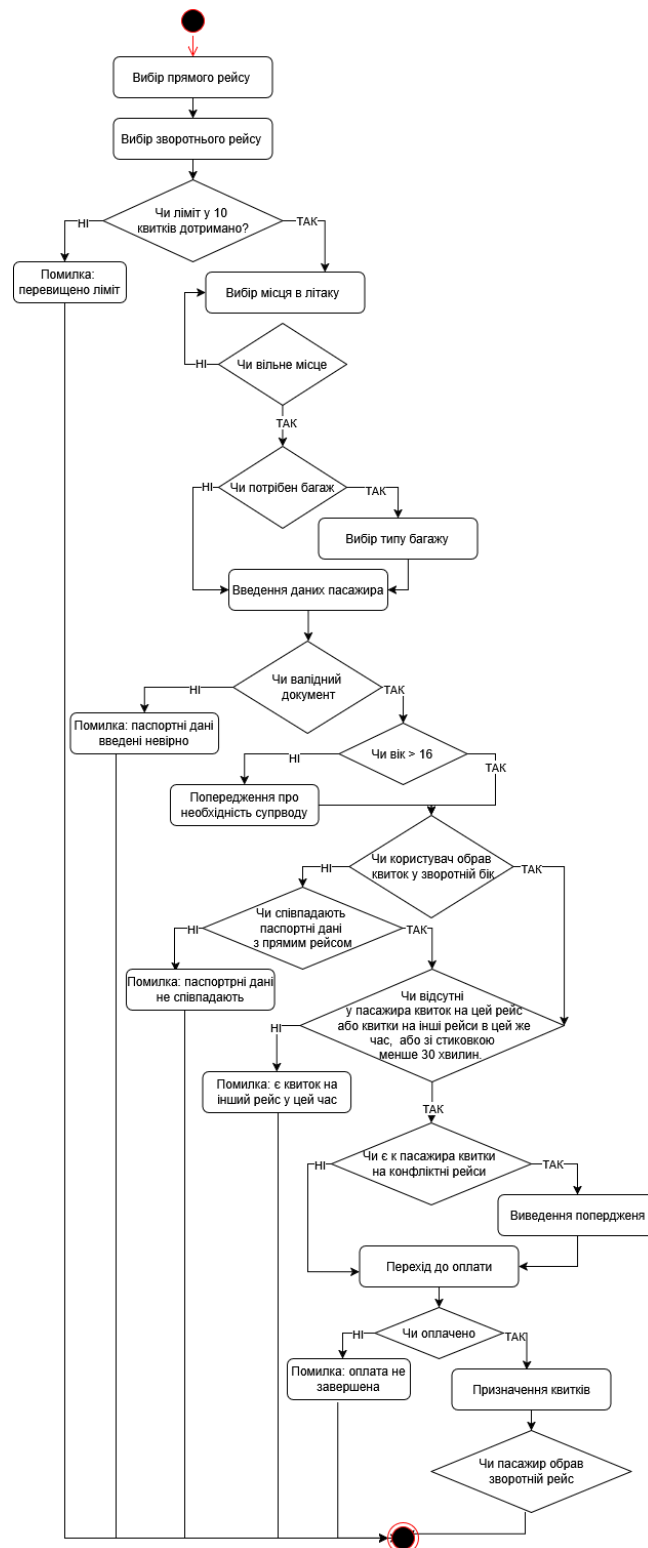


Рис. 2.5 – Діаграма активності покупки двох квитків: прямого і зворотнього

Діаграма активності використовується для моделювання послідовності дій, які відбуваються при виконанні певного процесу, включаючи як дії користувача, так і реакції системи [24].

Процес починається з перегляду наявних рейсів. Після авторизації через Spring Security, пасажир має змогу фільтрувати рейси за параметрами дати, місто відправлення, місто прибуття, чи потрібен квиток на зворотній рейс і якщо так – дата зворотнього рейсу.

Після вибору основного рейсу пасажир вибирає за потреби зворотній. Перед переходом до вибору місця виконується перевірка чи не придбав цей користувач вже більше ніж 10 квитків на обраний рейс (мається на увазі, що квитки могли бути оформлені на різних осіб, оскільки квиток є іменним).

Наступним кроком є вибір багажу. Пасажир може його додати, обравши бажаний тип. Після чого потрібно ввести паспортні дані. Якщо йдеться про зворотній рейс, додатково перевіряється, чи збігаються введені паспортні дані з тими, що були вказані при купівлі квитка на рейс в першу сторону, оскільки обидва квитки мають належати тій самій особі. Також якщо пасажир молодше 16-ти років, то виникає повідомлення про необхідність супроводу, аби батьки попідкувалися про це.

Окремо виконується перевірка на наявність конфліктуючих рейсів у цей день. Наприклад, якщо у пасажирів вже придбано інший квиток і аеропорт прибуття цього рейсу не збігається з аеропортом вильоту нового рейсу, система попереджає про потенційний конфлікт у маршруті. Водночас, якщо час нового рейсу перетинається з часом вже придбаного квитка, система не дозволяє пасажирові продовжити оформлення, оскільки це створює неможливість для логістичного виконання обох рейсів. Ну і звісно стиківка між рейсами має бути як мінімум 30 хвилин.

Останнім етапом є оплата. Використовується сторонній платіжний сервіс Stripe. Система створює Stripe-сесію, а статус оплати зберігається як «очікує». Користувач автоматично перенаправляється на сторінку Stripe Checkout для завершення оплати.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		41

Після успішної оплати система отримує callback, підтверджує платіж та перевіряє, чи не була оплата вже проведена раніше. Якщо така оплата вже зафіксована, система зупиняє обробку з відповідним повідомленням. Якщо ж ні – квиток бронюється, а його статус змінюється на «оплачено».

Якщо у замовленні зазначено зворотній рейс, вся описана послідовність дій повторюється і для зворотного квитка.

Також на рисунку 2.4 наведена діаграма активності процесу додавання нового літака. Спочатку адміністратор вводить модель літака якщо вона існує в базі даних то новий літак буде закріплений за цією моделлю. У разі відсутності моделі, створюється нова. Потім здійснюється вибір конфігурації місць у салоні. Після перевірки на дублікат створюється новий літак.

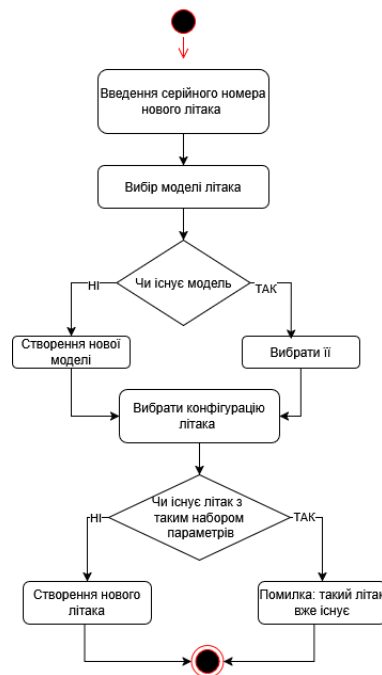


Рис. 2.6 – Діаграма активності додавання нового літака

2.4 Реалізація системи

В додатку важливим етапом є авторизація. Реалізації цього в системі здійснюється за допомогою JWT (JSON Web Token) та Spring Security [25]. Це забезпечує безпечний вхід, реєстрацію користувачів, генерацію токенів, їх оновлення. А обробка запитів здійснюється через фільтр без створення сесій.

В програмі є клас-сервіс JWTService (див. додаток А, клас JWTService), що відповідає за створення та валідацію токенів.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				42
Змн.	Арк.	№ докум.	Підпис	Дата		

AuthController (див. додаток А, клас AuthController) контролер, що відповідає за запити, пов'язані з аутентифікацією: логін, реєстрація, отримання інформації про поточного користувача і вихід із системи. Під час логіну система перевіряє правильність введених облікових даних. Після чого формуються JWT (на 30 хв) і refresh (на 7 днів) токени, які встановлюються у вигляді HTTP-only cookies. Так само відбувається під час реєстрації нового користувача.

Компонент JwtFilter (див. додаток А, клас JwtFilter) є фільтром, який перехоплює всі вхідні HTTP-запити. Він знаходить токен у cookies. Якщо токен присутній, з нього отримується адреса електронної пошти, по якій у базі даних знаходиться користувач. Якщо токен відсутній або ж недійсний, але при цьому присутній токен оновлення, то генерується новий JWT і додається у cookie.

Конфігурація безпеки реалізована в класі SecurityConfig (див. додаток А, клас SecurityConfig). У цьому класі вимикається CSRF-захист, оскільки система не зберігає сесії й працює з токенами JWT [26]. Доступ до всіх запитів, крім логіну, реєстрації та виходу, дозволений лише після автентифікації. Для цього використовується власний провайдер автентифікації, що шифрує паролі за допомогою BCrypt.

Наступний алгоритм наявний у системі є призначення працівників (бортпроводників і пілотів) на рейс. Цей процес реалізовано в контролері FlightController (див. додаток А, клас FlightController) у методі addWorkersToFlight. Даний метод доступний лише користувачам з роллю адміністратора через анотацію. Контролер знаходить відповідний рейс у базі даних і перевіряє, чи прив'язаний до нього літак. Якщо літак ще не додано, повертається помилка з повідомленням. У workerService (див. додаток А, клас workerService), в методі addWorkersToFlight визначає аеропорт відправлення рейсу та знаходить зворотний рейс за UUID ключем. Визначається початковий і кінцевий час подорожі і обчислюється її загальна тривалість. Це необхідно аби в подальшому обчислити час зайнятості працівників. Далі на основі кількості місць у літаку обчислюється мінімальна кількість бортпроводників, яка потрібна для обслуговування рейсу.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

Наступним кроком є отримання працівників для додавання їх на рейс з бази даних. За допомогою запиту в WorkerRepository (див. додаток А, клас WorkerRepository) надсилається два запити з використанням інтерфейсу JpaRepository, що є частиною Spring Data JPA [27]. Перший – getPilotsForFlight, метод для вибору пілотів. Здійснюється вибірка з таблиці працівників (Worker), фільтруючи лише тих, хто має посаду «PILOT». До цієї таблиці додається інформація з таблиці сертифікатів, оскільки пілот має мати сертифікат на відповідну модель літака, який буде задіяний в рейсі. Запит перевіряє, чи термін дії сертифіката актуальний на дату вильоту. Також використовуються додаткове приєднання (LEFT JOIN) до таблиці робочих годин (WorkHour) - для перевірки кількості робочих годин у поточному місяці і в поточному році. Це дозволяє переконатися, що обраний працівник не перевищить встановлені норми: не більше 100 годин на місяць і 1000 годин на рік. Ці перевірки реалізовано через WHERE та HAVING умови з COALESCE, що дозволяє коректно обробити працівників без записів у таблиці годин. Для уникнення ситуацій, коли одні й ті ж пілоти постійно працюють разом або на одних рейсах, результати рандомізуються.

Другий запит - це getCrewForFlight для отримання бортробітників. У запиті здійснюється LEFT JOIN до таблиці WorkHour про яку вже було згадано вище. Серед додаткових перевірок є посада «CREWMEMBER», аеропорт вильоту має співпадати з домашнім аеропортом працівника і ті ж самі обмеження по робочих годинах. Отриманні робітники також є рандомними. Це було здійснено задля безпеки аби одна й та ж команда на мала багато спільних рейсів.

При додаванні робітників, за допомогою workHoursService (див. додаток А, клас workHoursService) оновлюються години роботи як для пілотів, так і для екіпажу, з урахуванням тривалості подорожі. Успішне виконання операції завершується відповіддю з повідомленням про успішне призначення персоналу.

Крім цього, варто відзначити клас SeatService (див. додаток А, клас SeatService), що відповідає за логіку генерації сидінь у літаку згідно з його конфігурацією (Layout).

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				44
Змн.	Арк.	№ докум.	Підпис	Дата		

Також присутній алгоритм створення та збереження у базі даних усіх сидінь у літаку на основі заданого макету салону. Він реалізований у методі `seatsGeneration` (див. додаток А, метод `seatsGeneration`). Процес послідовно проходить по кожному ряду, кожній колонці та кількості сидінь у колонці, формуючи унікальні номери місць «1А», «1В» і так далі. Тобто кожне сидіння отримує унікальний номер, що складається з номера ряду та відповідної букви з алфавіту. Для кожного місця також визначається клас або перший, або економ, залежно від того, в якому ряді воно знаходиться. Після формування повного списку місць вони зберігаються у базі, що дозволяє далі керувати ними при бронюванні та інших операціях.

Висновки до другого розділу

В другому розділі було визначено основні варіанти використання системи. Та описано взаємодію користувачів із функціями системи.

Наступним кроком стало створення діаграми класів системи з урахуванням патернів, такі як MVC та Observer. Так за допомогою MVC-патерну було розділено систему на три основні частини: моделі, контролери і репозиторії, а також додатковий шар – сервіси для реалізації додаткової логіки.

Також було розроблено базу даних, яка містить всі необхідні таблиці для збереження інформації про рейси, пасажирів, персонал, літаки, бронювання, тощо.

Окрім цього, було описано деякі алгоритми наявні в системі. Такі як: авторизація та аутентифікація, додавання робітників на рейс (формування їх розкладу) та генерація сидінь.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ СИСТЕМИ УПРАВЛІННЯ АВАКОМПАНІЄЮ

3.1 Структура інтерфейсу системи управління авіакомпанією

Зайшовши на сайт, користувач бачить головну сторінку (рис.3.1) на якій він може відразу почати пошук рейсів за містом відправлення, містом призначення та датою відправлення. Також може розширити запит пошуком зворотнього рейсу. Однак щоб переглянути рейс та придбати квиток пасажир має зареєструватися.

Рис. 3.1 – Головна сторінка програми

Тож користувачі мають можливість увійти до системи (рис.3.2) або зареєструватися (рис.3.3). Вхід до системи передбачає введення логіну (електронної пошти) та пароля. Для реєстрації також слід заповнити дані поля, що дозволяє створити новий акаунт.

Рис. 3.2 – Сторінка логіну

Рис. 3.3 – Сторінка реєстрації

Після успішного входу користувач потрапляє на головну сторінку системи. У верхній частині інтерфейсу розташоване меню навігації, воно залежить від рівня доступу користувача. Так адміністратор має доступ до ключових розділів: шаблони/конфігурацій, літаки, аеропорти, маршрути, рейси, працівники та

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

сертифікати (рис.3.4). Саме верхнє меню допомагає швидко переміщатися між різними частинами системи.

Рис. 3.4 – Головна сторінка сайту. Поточний користувач: адміністратор

Адміністратору доступні сторінки на яких він може модифікувати дані системи. Так, наприклад, сторінка аеропортів відображає повний перелік усіх аеропортів, що зареєстровані в системі (рис.3.5).

Аеропорти

		Пошук		Додати аеропорт				
# 1	IATA	ICAO	Ім'я	Місто	Країна	Часова зона	Активний	Actions
1	LHR	EGLL	London Heathrow Airport	London	GB	Europe/London	Так	Деактивувати
2	JFK	KJFK	John F Kennedy International Airport	New York	US	America/New_York	Так	Деактивувати
3	VIE	LOWW	Vienna International Airport	Vienna	AT	Europe/Vienna	Так	Деактивувати
4	LCA	LCLK	Larnaca International Airport	Larnaca	CY	Asia/Nicosia	Так	Деактивувати
5	PRG	LKPR	Ruzyně International Airport	Prague	CZ	Europe/Prague	Так	Деактивувати
6	SJJ	LQSA	Sarajevo International Airport	Sarajevo	BA	Europe/Sarajevo	Так	Деактивувати
7	BRU	EBBR	Brussels Airport	Brussels	BE	Europe/Brussels	Так	Деактивувати

Рис. 3.5 – Сторінка аеропортів

Можна додати новий аеропорт. Це відбувається у декілька кроків: адміністратор вводить IATA код аеропорту, після чого система через зовнішнє API отримує всі дані про аеропорт (включно з таймзоною, довготою та широтою) і дана інформація виводиться аби можна було її переглянути (рис.3.6) і зберегти у базі даних (рис.3.7). Це значно спрощує процес додавання, мінімізуючи ризик помилок.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				47
Змн.	Арк.	№ докум.	Підпис	Дата		

✕

Додати аеропорт

CDG

Пошук

Вибрати аеропорт

Вибрати один

Вибрати один

Charles de Gaulle International Airport (CDG) - Paris, FR

Додати

Рис. 3.6 – Додавання нового аеропорту

7	CDG	LFPG	Charles de Gaulle International Airport	Paris	FR	Europe/Paris	Так	Додати аеропорт
8	CDG	LFPG	Charles de Gaulle International Airport	Paris	FR	Europe/Paris	Так	Деактивувати

Рис. 3.7 – Додано новий запис про аеропорт

Розділ шаблони/конфігурації містить опис внутрішнього устрою літаків, а саме – розташування сидінь (рис.3.8). Таблиця містить кількість: колонок, сидінь у кожній колонці, рядів у салоні та рядів першого класу. Адміністратор може додавати нові шаблони (рис.3.9), що дозволяє гнучко налаштовувати планування салону під різні моделі літаків.

Шаблони/Конфігурації

Пошук

Додати

Кількість колонок	Кількість сидінь в колонці	Кількість рядів	Кількість рядів першого класу
2	3	21	4
3	2	23	6

Рис. 3.8 – Сторінка шаблонів

✕

Додати шаблон/
конфігурацію

Кількість колонок

наприклад 2

Кількість сидінь в колонці

наприклад 3

Кількість рядів

Ряди

Кількість рядів першого класу

Перший клас

Додати

Рис. 3.9 – Додавання нового шаблону

Сторінка літаків відображає всі наявні літаки авіакомпанії (рис.3.10). Вказані також їх основні характеристики, включаючи домашній аеропорт де простоє літак. Авіалайнер можна деактивувати і активувати (рис.3.12 - рис.3.13) якщо

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				48
Змн.	Арк.	№ докум.	Підпис	Дата		

раптом той вийшов з ладу, має технічну несправність або був виведений з експлуатації.

Літаки

Пошук

Додати новий

# 1	Серійний номер	Модель	Шаблон/Конфігурація	Домашній аеропорт	Actions
1	1111111	Boeing 737	2x21	London Heathrow Airport	Деактивувати
2	2435345346	Boeing 747	3x23	London Heathrow Airport	Деактивувати

Рисунок 3.10 – Сторінка літаків

При додавання нового літака обов’язково обирається його конфігурації сидінь та вказується модель (рис.3.11). Це забезпечує відповідність літака реальним параметрам та забезпечує можливість продавати квитки на реальні місця наявні в літаку.

Додати літак

Серійний номер

121131232313

Модель

Boeing 737-800

Шаблон/Конфігурація

2x21

Домашній аеропорт

Charles de Gaulle International Airport (Paris, FR)

Додати

Рис. 3.11 – Додавання нового літака

3	121131232313	Boeing 737-800	2x21	Charles de Gaulle International Airport	Деактивувати
---	--------------	----------------	------	---	--------------

Рис. 3.12 – Деактивація літака

3	121131232313	Boeing 737-800	2x21	Charles de Gaulle International Airport	Активувати
---	--------------	----------------	------	---	------------

Рис. 3.13 – Активація літака

Сторінка маршрутів містить інформацію про всі маршрути якими оперує авіакомпанія (рис.3.14)

Маршрути

Пошук по аеропорту

Додати

#	Аеропорт відправлення	Аеропорт прибуття
1	London Heathrow Airport	Brussels Airport
2	Brussels Airport	London Heathrow Airport

Рис. 3.14 – Сторінка маршрутів

При додаванні нового маршрута (рис.3.15) автоматично генерується зворотній (рис.3.16). Це суттєво економить час та зменшує кількості помилок, що може допустити адміністратор.

Рис. 3.15 – Додавання нового маршрута

2	Brussels Airport	London Heathrow Airport
3	Sarajevo International Airport	Charles de Gaulle International Airport
4	Charles de Gaulle International Airport	Sarajevo International Airport

Рис. 3.16 – Сформовані маршрути туди і зворотній

Розділ працівників містить список всіх пілотів та бортпровідників компанії (рис.3.17). У їх особових даних вказано посаду, що вони обіймають та домашній аеропорт та ідентифікаційні дані: ім'я та пошта.

Рис. 3.17 – Сторінка інформації про працівників

Адміністратор також може додати нового співробітника (рис.3.18) вводячи його дані, посаду, а також вказавши його пошту та пароль для доступу в систему. Аутентифікаційні дані введені адміністратором буде автоматично надіслано робітнику на пошту, після чого той має змогу їх змінити у власному кабінеті після авторизації в системі (рис.3.19).

Рис. 3.18 – Додавання нового працівника

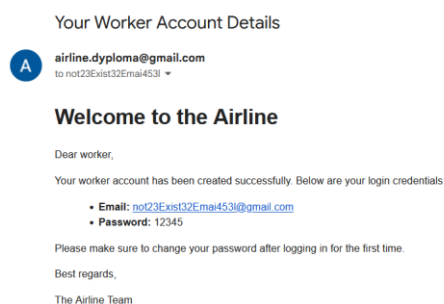


Рис. 3.19 – Лист з даними для аутентифікації в системі

Сторінка сертифікатів містить інформацію про наявні сертифікати у пілоті (рис.3.20). Адміністратор бачить запити від пілота на підтвердження його сертифікату, він може підтвердити або відхилити його, якщо наприклад документу немає в базі Державної авіаційної служби.

Сертифікати пілотів

Працівник	Модель літака	Серійний номер	Дата видачі	Дата закінчення	Статус	Actions
PilotLondon1	Boeing 737	675765696567956757	2025-05-25	2026-06-01	APPROVED	—
PilotLondon2	Boeing 747	43563636444364	2025-05-25	2026-06-01	APPROVED	—
Pilot4	Boeing 737	4353534536345654	2025-05-26	2026-06-02	APPROVED	—
Pilot3	Boeing 737	4534654654654	2025-05-26	2026-06-02	APPROVED	—

Рис. 3.20 – Сторінка сертифікатів пілотів

Формування розкладу рейсів – ключова функція системи, її серце (рис.3.21). Тут адміністратор може переглянути всі доступні польоти, додати нові та додати робітників на вже створені рейси.

Рейси

Пошук

Додати новий рейс

Час відправлення (UTC)	Аеропорт відправлення	Час прибуття (UTC)	Аеропорт прибуття	Статус	Літак	Технічний	Працівники	Actions
6/30/2025, 2:00:00 AM	London Heathrow Airport London, GB	6/30/2025, 2:53:00 AM	Brussels Airport Brussels, BE	ON_TIME	1111111 Boeing 737	Hi	CrewWorkerLondon1 - CREWMEMBER CrewWorkerLondon3 - CREWMEMBER CrewWorkerLondon4 - CREWMEMBER Pilot3 - PILOT Pilot4 - PILOT	—
6/30/2025, 3:53:00 AM	Brussels Airport Brussels, BE	6/30/2025, 4:46:00 AM	London Heathrow Airport London, GB	ON_TIME	1111111 Boeing 737	Hi	CrewWorkerLondon1 - CREWMEMBER CrewWorkerLondon3 - CREWMEMBER CrewWorkerLondon4 - CREWMEMBER Pilot3 - PILOT Pilot4 - PILOT	—

Рис. 3.21 – Сторінка розкладу рейсів

При додаванні нового рейсу (рис.3.22), адміністратор вибирає час відправлення, необхідний маршрут з наявних та літак. Система автоматично перевіряє, чи співпадає домашній аеропорт літака з аеропортом відправлення, що забезпечує логічність і коректність планування. При додаванні польоту, автоматично створюються в обидва напрямки - прямий та зворотній. Час прибуття рейсу, а також час відправлення і прибуття зворотного рейсу визначаються автоматично системою. Ці параметри розраховуються на основі тривалості польоту, яку надає відкрите API. У випадку, якщо політ триває понад 10 годин, система встановлює мінімальний інтервал у 10 годин між прибуттям літака та відправленням зворотного рейсу. Це забезпечує необхідний відпочинок екіпажу, а також гарантує дотримання нормативних вимог безпеки польотів.

Додати рейс

Маршрут:
Charles de Gaulle International Airport → Sarajevo International Airport

Літак:
Boeing 737-800 — Домашній аеропорт: Charles de Gaulle International Airport

Час відправлення (UTC):
07 / 05 / 2025, 01 : 00 AM

Чи технічний політ:
Жоден
Політ туди
Зворотній рейс
Обидва

Додати

Рис. 3.22 – Додавання нового рейсу

7/5/2025, 1:00:00 AM	Charles de Gaulle International Airport Paris, FR	7/5/2025, 3:04:00 AM	Sarajevo International Airport Sarajevo, BA	ON_TIME	121131232313 Boeing 737-800	Hi	Працівники ще не призначені	Додати працівників
7/5/2025, 4:04:00 AM	Sarajevo International Airport Sarajevo, BA	7/5/2025, 6:08:00 AM	Charles de Gaulle International Airport Paris, FR	ON_TIME	121131232313 Boeing 737-800	Hi	Працівники ще не призначені	Додати працівників

Рис. 3.23 – Щойно доданий рейс

Окрім всього вищесказаного, пілоту система надає можливість додати свій сертифікат, що підтверджує право на пілотування певною моделлю літака. Даний сертифікат буде розглянутий та затверджений (чи відхилений) адміністратором (рис.3.24 – рис.3.25).

Рис. 3.24 – Додавання сертифікату пілотом

AIRLINE				
Сертифікати		Сертифікати	Календар	EN mertete213.kde123rtretertl@gmail.com
Пошук по моделі чи ста Додати новий сертифікат				
Модель літака	Статус	Серійний номер	Дата видачі	Дата завершення дії
Boeing 737	APPROVED	4534654654654	2025-05-26	2026-06-02
Boeing 737-800	PENDING	76556655656	2025-05-25	2025-08-30

Рис. 3.25 – Сторінка сертифікатів пілота

Крім того, робітники (як пілоти так і бортпровідники) можуть переглянути в системі свій розклад на рейси, на які вони були призначені (рис.3.26 - рис.3.27).

Розклад ресів						
Today	Back	Next	June 2025			
Month	Week	Day				
Sun	Mon	Tue	Wed	Thu	Fri	Sat
01	02	03	04	05	06	07
08	09	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	01	02	03	04	05
London Heathrow Airport ... Brussels Airport → London...						

Рис. 3.26 – Розклад рейсів працівника (місяць)

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				53
Змн.	Арк.	№ докум.	Підпис	Дата		

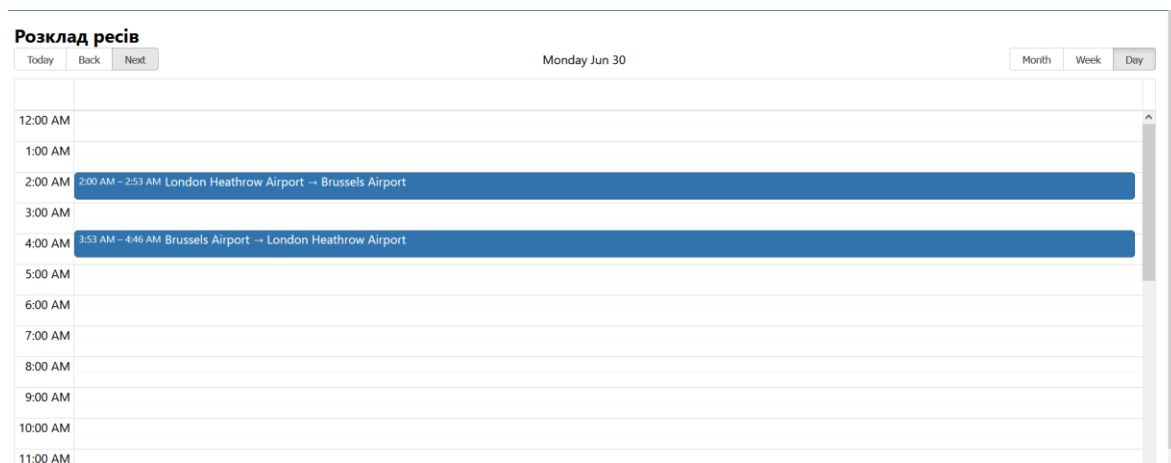


Рис. 3.27 – Розклад рейсів працівника (день)

Будь який користувач, будь то адміністратор, працівник або пасажир мають персональний кабінет, де можуть переглядати та редагувати власні дані, змінювати пароль та управляти особистою інформацією (рис.3.28).

Рис. 3.28 – Кабінет користувача

Пасажири можуть здійснювати пошук рейсів за містом відправлення, прибуття та датою (рис.3.29). А також може вказати дату зворотнього рейсу і купити квитки відразу туди й назад. Наявні варіанти можна відфільтрувати на максимальною ціною, за аеропортом відправлення та прибуттям (адже вказується саме місто для пошуку).

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				54
Змн.	Арк.	№ докум.	Підпис	Дата		

Доступні рейси

Максимальна ціна:

Аеропорт відправлення:

Аеропорт прибуття:

Максимальна ціна

Пошук

Пошук

London → Brussels

заданим: London Heathrow Airport

До: Brussels Airport

Час відправлення: 2025-06-30 03:00

Час прибуття: 2025-06-30 04:53

Ціна: \$200.00

Забронювати

← Повернутися

Рис. 3.29 – Доступні рейси за пошуком

Після вибору рейсу пасажир перенаправляється на сторінку вибору місця в літаку, де він може обрати будь-яке доступне сидіння (рис.3.30).

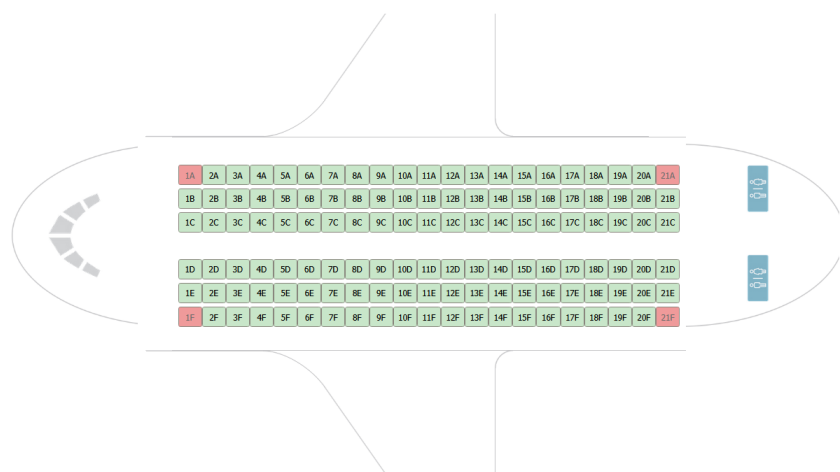


Рис. 3.30 – Сторінка обрання сидіння на літак

Далі пасажир переходить до сторінки вибору багажу, де він може вибрати бажаний за розміром або його відсутність (рис.3.31).

ВІДПРАВЛЕННЯ

Аеропорт: London Heathrow Airport

Час: 30 червня 2025 р. о 03:00

ПРИБУТТЯ

Аеропорт: Brussels Airport

Час: 30 червня 2025 р. о 04:53

SEAT

Seat: 7F

Виберіть ваш багаж

No baggage

Багаж не вибрано

Безкоштовно

Small

Макс вага: 10 kg

Макс розмір: 40x30x20 cm

Ціна: \$10

Medium

Макс вага: 20 kg

Макс розмір: 60x40x30 cm

Ціна: \$20

Big

Макс вага: 30 kg

Макс розмір: 70x50x40 cm

Ціна: \$30

Підтвердити

Рис. 3.31 – Сторінка вибору багажу

Після чого пасажир має ввести дані людини, що буде безпосередньо летіти на рейсі (рис.3.32). Присутня можливість купити квиток не тільки для себе, але й

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				55
Змн.	Арк.	№ докум.	Підпис	Дата		

для інших осіб - до 10 пасажирів на один рейс. Оскільки квиток іменний, для однієї людини на один рейс може бути лише один квиток.

Варто також зазначити, що якщо у пасажира вже є рейс у той самий день, і аеропорт прибуття попереднього рейсу не співпадає з аеропортом відправлення нового (або навпаки), система видає попередження, щоб користувач міг переглянути свої плани (рис. 3.33). Проте, незважаючи на це попередження, користувач може продовжити процес купівлі квитка.

ВІДПРАВЛЕННЯ

Аеропорт: London Heathrow Airport

Час: 30 червня 2025 р. о 03:00

ПРИБУТТЯ

Аеропорт: Brussels Airport

Час: 30 червня 2025 р. о 04:53

SEAT

Seat: 7F

Повне ім'я:

Анна Захарено

Дата народження:

10 / 10 / 2000

Тип документа:

Passport

Номер документа:

35346345734575

Дата випуску:

05 / 25 / 2025

Дата завершення дії:

07 / 05 / 2025

Країна випуску:

ua

Перейти до оплати

Рис. 3.32 – Введення даних пасажира

Дата народження:

10 / 10 / 2000

Тип документа:

Passport

Номер документа:

Увага

Ви прибуваєте в інший аеропорт, ніж той, з якого відправляється ваш наступний рейс.

ОК

Дата завершення дії:

07 / 05 / 2025

Рис. 3.33 – Попередження: аеропорт прильоту/вильоту рейсу на який вже куплено квиток на співпадає з тим на який планує купити новий

Також є наступні перевірки (у разі виникнення цих помилок продовжити бронювання неможливо). Наприклад, якщо пасажир має квиток на рейс із часом вильоту чи прильоту з різницею менше ніж 30 хвилин (рис. 3.34), або якщо пасажир

уже має квиток на цей самий рейс (рис. 3.35), або має квиток на інший рейс, що відбувається у той самий час (рис. 3.36).

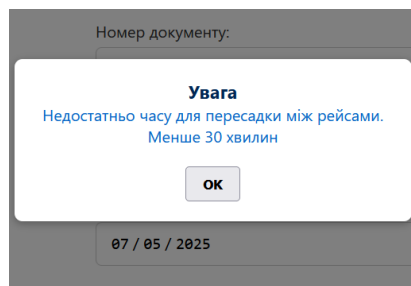


Рис. 3.34 – Помилка: пасажир має квиток на рейс, що має час вильоту чи прильоту з різницею менше 30 хвилин

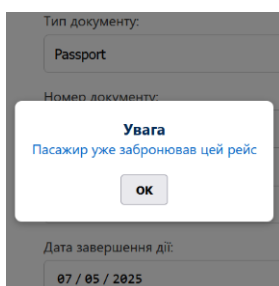


Рис. 3.35 – Помилка: пасажир вже має квиток на цей рейс

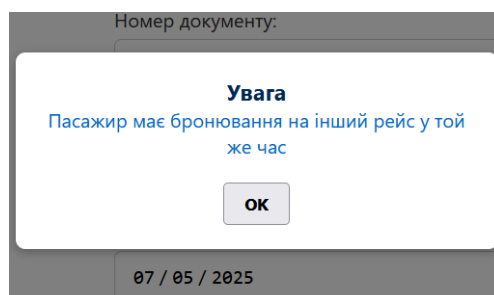


Рис. 3.36 – Помилка: у пасажирів вже є квиток на інший рейс в цей час

Після коректного заповнення усіх даних користувача система перенаправляє пасажирів на сторінку оплати (рис.3.37 – рис.3.38)

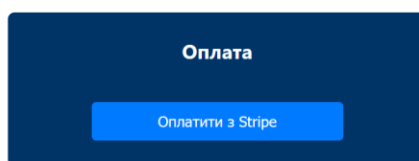


Рис. 3.37 – Перенаправлення на сторінку оплати

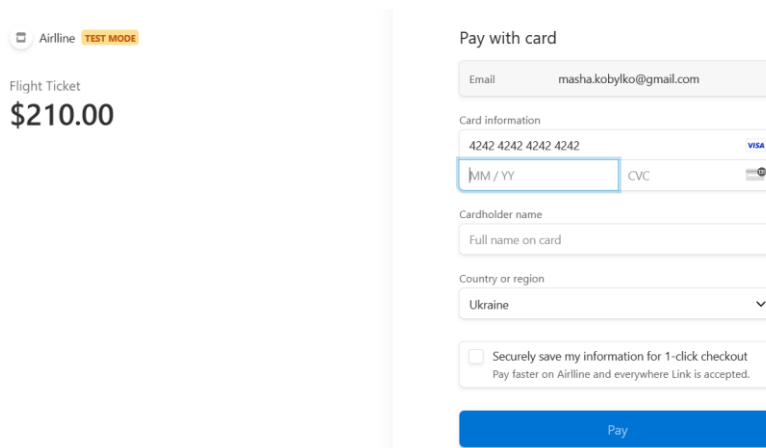


Рис. 3.38 – Сторінка оплати

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				57
Змн.	Арк.	№ докум.	Підпис	Дата		

Оплата відбувається на сторонньому сервісі – Stripe [28]. Вартість оплати формується з урахуванням вибраного багажу, а також застосовується знижка 10% при купівлі квитка в обидва кінці маршруту.

Придбаний квиток приходить на пошту (рис.3.39) і зберігається у персональному кабінеті (рис.3.40).

Привіт!

Ваш квиток успішно заброньовано! Ось деталі:

Ім'я пасажир: Анна Захарено
Пасажирський документ: ПАСПОРТ - 35346345734575
Аеропорт вильоту: Лондонський аеропорт Хітроу
Час відправлення: 2025-06-30T03:00
Аеропорт прибуття: Брюссельський аеропорт
Час прибуття: 2025-06-30T04:53
Номер місця: 7F
Всього сплачено: 210,0 доларів США

Дякуємо, що обрали нашу авіакомпанію. Бажаємо вам приємного польоту!

Рис. 3.39 – Квиток на пошті

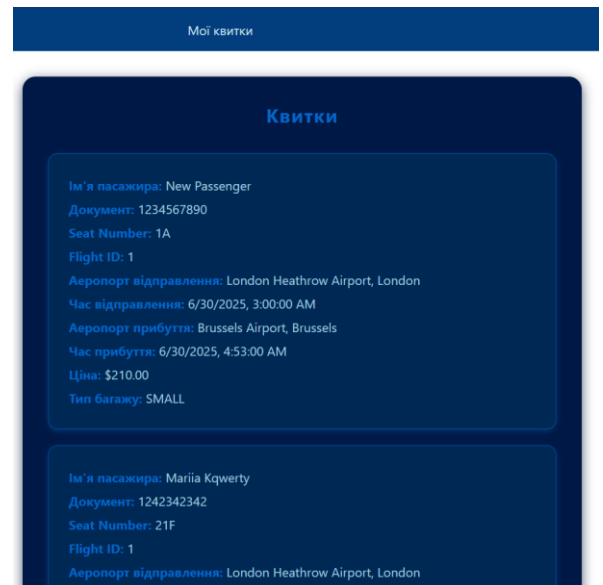


Рис. 3.40 – Квитки в кабінеті

3.2 Тестування роботи програмного комплексу

Тестування програмного забезпечення є невід’ємною частиною розробки. Воно дозволяє виявити помилки, перевірити функціональність, а також забезпечити стабільність і безпеку роботи [29].

У випадку із системою управління авіакомпанією особливу увагу приділено тестуванню критичних процесів, таких як створення рейсів, обробка сертифікатів пілотів, управління квитками, формування розкладів та обробка користувацьких ролей. Метою тестування є перевірка відповідності фактичної поведінки системи очікувань, забезпечити коректне функціонування всіх процесів, виявити дефекти і усунути їх.

У ході тестування використовувалися як ручні методи тестування інтерфейсу, так і модульні. Було створено мок тести для перевірки роботи певних методів системи. Мок тести - це юніт тести, у яких залежності класу (наприклад, репозиторії, сервіси) замінені на мок-об’єкти (імітації), що дозволяє контролювати

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

їхню поведінку і тестувати логіку методу без взаємодії з реальними базами даних або зовнішніми системами [30].

Було створено набір юніт-мок тестів для контролера (LayoutController), який відповідає за додавання конфігурацій розташування місць (Layout) (рис.3.41).

Було реалізована перевірка декількох сценаріїв: успішне додавання нового шаблону; додавання дубліката шаблону (імітується наявність в бд конфігурації з такими ж параметрами, що й в того який хочуть додати), контролер повинен повернути помилку статусом 400; перевірка логіки валідації; перевірка валідації полів (чи контролер коректно реагує на порушення обмежень валідації).

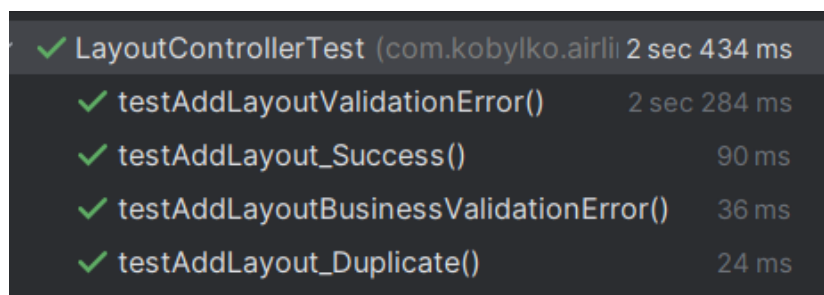


Рис.3.41 – Пройдені тести для перевірки Layout

Також реалізовано інтеграційні тести для репозиторію WorkerRepository із застосуванням анотації @DataJpaTest, що дозволяє тестувати роботу з базою даних у пам'яті або з реальною БД.

Таким чином, тест getPilotsForFlight перевіряє логіку вибору пілотів для конкретного рейсу. А саме - чи пілоти мають сертифікат на модель літака, яка використовується для рейсу, їх домашній аеропорт – аеропорт вильоту, також перевіряється чи не буде перевищено ліміт їх робочих годин.

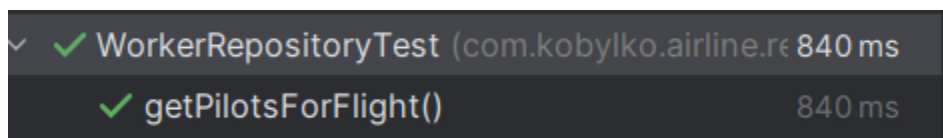


Рис.3.42 – Пройдений тест додавання пілотів на рейс

Тест AirplaneServiceTest реалізовує юніт-тести для сервісного шару, який відповідає за керування літаками. Для імітації залежностей використовується бібліотека Mockito, що дозволяє ізолювати тестований клас від зовнішніх ресурсів і перевірити логіку на рівні сервісу.

Основний тест `addAirplaneTest_Success` перевіряє коректне додавання нового літака до системи (рис.3.43).

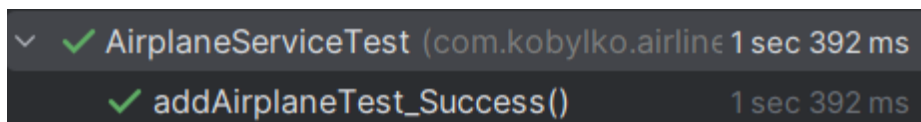


Рис.3.43 – Пройдений тест – додавання літака

У класі `PassengerServiceTest` реалізовано юніт-тести для сервісу, який відповідає за верифікацію та створення пасажирів під час бронювання квитків.

Тест `verifyOrCreatePassenger_existingValidPassenger_returnsPassenger` перевіряє поведінку сервісу у випадку, коли пасажир з вказаними даними вже існує у системі.

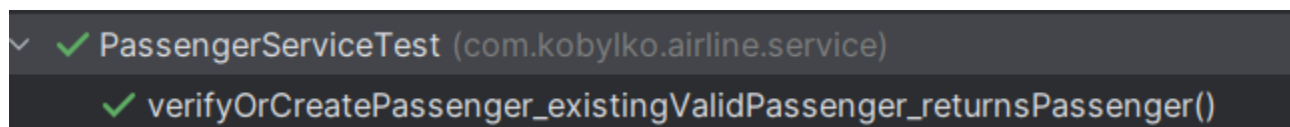


Рис.3.44 – Пройдений тест перевірки пасажирів

Чек-лист (табл.3.1) демонструє приклади деяких тестових сценаріїв, що були перевірені.

Таблиця 3.1

Чек-лист функціонального тестування

Тестування	Кроки тестування	Тестові дані	Очікуваний результат	Фактичний результат	Pass / Fail
Авторизація	1. Перейти на сторінку логіну	Логін: anna.ko@gmail.com	Успішно. Користувач потрапляє на головну сторінку	Вхід успішний, перенаправлення на домашню	Pass
	2. Ввести email та пароль	Пароль: 23@1w2D23			
	3. Натиснути кнопку "Увійти"				
Додавання аеропорту	1. Перейти на сторінку "Аеропорти"	IATA-код "AMS"	Аеропорт знайдено через API, збережено в БД, виведен до таблиці	Аеропорт додано до списку	Pass
	2. Ввести IATA-код "AMS"				
	3. Натиснути "Знайти"				
	4. Отримати аеропорт та вибрати його				
	5. Зберегти				

Продовження таблиці 3.1

Тестування	Кроки тестування	Тестові дані	Очікуваний результат	Фактичний результат	Pass / Fail
Додавання літака	1. Перейти на сторінку "Літаки"	Модель: Boeing 737	Літак створено, відображається у списку	Літак додано до бази даних, відображається в таблиці	Pass
	2. Вказати модель, домашній аеропорт, шаблон	Аеропорт: AMS (id=15)			
	3. Зберегти	Шаблон: 3*2 (id=1)			
Формування маршруту	1. Перейти на сторінку "Маршрути"	Аеропорт вильоту: LHR	Створено два рейси (туди й назад)	Маршрут згенеровано	Pass
	2. Обрати аеропорти відправлення та прибуття	Аеропорт прибуття: AMS			
	3. Натиснути "Додати"				
Додавання робітника	1. Перейти на сторінку "Робітники"	Ім'я: Андрій	Працівника додано до списку, email з логіном/паролем надіслано	Запис збережено, лист надійшов	Pass
	2. Ввести дані	Посада: Pilot			
	4. Натиснути «Зберегти»	Email: pilot@test.com Пароль: Pilot123			
Створення рейсу	1. Обрати маршрут	Маршрут: Амстердам - Лондон	Два рейси (туди й назад) створено, враховано час польоту та відпочинку, літак додано	Рейси згенеровано, час поставлено, літак додано	Pass
	2. Вибрати літак	Літак (id=15)			
	3. Вказати дату та час вильоту	Дата вильоту: 15.07.2025, 10:00			

Висновки до третього розділу

У третьому розділі було розглянуто інтерфейс та порядок роботи вебдодатку. Детально описано, як виглядає програма та які дії вона реалізує. Інтерфейс розроблено так, щоб ним можна було легко користуватися без попередньої підготовки або з мінімальним поясненням.

Також у цьому розділі проведено тестування програмного комплексу. Було перевірено, як працюють основні функції системи. Тестування показало, що система працює стабільно, без збоїв і помилок.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				61
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

У випускній кваліфікаційній роботі бакалавра розроблено веборієнтовану систему управління авіакомпанією.

Було розглянуто і проаналізовано декілька існуючих аналогів, визначено їх переваги та недоліки, сформовано список вимог до системи.

Було визначено ролі користувачів та представлено їх у вигляді use case діаграми, що дозволило розглянути функції системи та взаємодію між користувачами і програмою.

Для розробки системи обрано сучасний технологічний стек, який включає Spring Boot, Spring Data, Hibernate, MySQL та React

Було задіяно патерни програмування такі як MVC та Спостерігач, а також DTO. Особливу увагу приділено розробці певних алгоритмів програми. А також для наочності та кращого розуміння логіки роботи системи представлено діаграми активності, які відображають послідовність дій користувачів і системи в ключових сценаріях.

Розроблений програмний комплекс є повністю функціональним та готовим до впровадження. Його можна використовувати у малих авіакомпаніях для ефективного управління операціями, підвищення якості обслуговування пасажирів та оптимізації внутрішніх процесів. Завдяки модульній архітектурі та застосуванню перевірених патернів програмування, система легко масштабується і адаптується під конкретні потреби користувачів.

У подальшому даний програмний продукт може удосконалюватися шляхом створення додаткових модулів, таких як система аналітики та звітності, підтримка мобільних платформ, впровадження алгоритмів машинного навчання для прогнозування попиту на рейси та оптимізації маршрутів та багато іншого.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				62
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кобилко М.П., Вакалюк Т.А. Аналіз веб-додатків управління авіакомпанією та продажу квитків. Інформаційні технології – 2025: зб. тез XII Всеукраїнської науково-практичної конференції молодих учених, 15 трав. 2025 р., м. Київ / Київський столичний університет імені Бориса Грінченка; Відповід. за вип.: М.М. Астаф'єва, Д.М. Бодненко, О.М. Глушак, І.В. Машкіна, О.В. Локазюк, В.В. Прошкін, С.М. Шевченко. Київ : Київський столичний університет імені Бориса Грінченка, 2025. С. 136-137.
<https://zcit.kubg.edu.ua/index.php/journal/issue/view/13/23> (тези - техн)

2. Кобилко М.П., Вакалюк Т.А. Архітектурні підходи до побудови системи управління авіакомпанією. Інформаційні технології – 2025: зб. тез XII Всеукраїнської науково-практичної конференції молодих учених, 15 трав. 2025 р., м. Київ / Київський столичний університет імені Бориса Грінченка; Відповід. за вип.: М.М. Астаф'єва, Д.М. Бодненко, О.М. Глушак, І.В. Машкіна, О.В. Локазюк, В.В. Прошкін, С.М. Шевченко. Київ : Київський столичний університет імені Бориса Грінченка, 2025. С. 138-139.
<https://zcit.kubg.edu.ua/index.php/journal/issue/view/13/23> (тези - техн)

3. Wizzair [Електронний ресурс] – Режим доступу:
<https://www.wizzair.com/uk-ua>

4. Ryanair [Електронний ресурс] – Режим доступу:
<https://www.ryanair.com/ua/uk>

5. Lufthansa [Електронний ресурс] – Режим доступу:
<https://www.lufthansa.com/ua/en/homepage?>

6. Oracle Java Documentation [Електронний ресурс] – Режим доступу:
<https://docs.oracle.com/en/java/>

7. Loy, Marc, Patrick Niemeyer, and Daniel Leuck. Learning Java: An Introduction to Real-World Programming with Java. 5th ed., O'Reilly Media, 2021. ISBN-13: 978-1098145530, ISBN-10: 1098145534.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

8. Horstmann, Cay S. Object-Oriented Design and Patterns, 2nd Edition. Wiley, 2022. ISBN-13: 978-1119793658.
9. What is OOP (Object Oriented Programming)? [Електронний ресурс] – Режим доступу: <https://docs.oracle.com/en/java/>
10. Ottinger, Joseph B., and Andrew Lombardi. Beginning Spring 6: From Beginner to Pro, Second Edition. Apress, 2023.
11. Schwartz, Baron, Peter Zaitsev, and Vadim Tkachenko. High Performance MySQL: Optimization, Backups, and Replication, 3rd Edition. O'Reilly Media, 2020. ISBN-13: 978-1492083069.
12. Rahman, Seyed M.M. Learning MySQL: Get a Handle on Your Data. O'Reilly Media, 2021.
13. Hinkula, Juha. Full Stack Development with Spring Boot and React, Third Edition: Build Modern and Scalable Full Stack Applications Using the Power of Spring Boot and React. Packt Publishing, 2023.
14. Gutierrez, Felipe. Pro Spring Boot 2: An Authoritative Guide to Building Microservices, Web and Enterprise Applications, and Best Practices. Apress, 2020. ISBN-13: 978-1484259664.
15. Heckler, Mark. Spring Boot: Up and Running: Building Cloud Native Java and Kotlin Applications. O'Reilly Media, 2021. ISBN-13: 978-1492076988
16. Tudose, C. Java Persistence with Spring Data and Hibernate / C. Tudose, C. Bauer, G. King, G. Gregory. — Revised and extended new ed. — Shelter Island: Manning Publications Co., 2023.
17. Spilca, Laurentiu. Spring Security in Action, Second Edition. Manning Publications, 2024. ISBN-13: 978-1633437975
18. Stefanov, Stoyan. React - Up & Running: Building Web Applications. O'Reilly Media, 2021. ISBN-13: 978-1492051725.
19. MySQL Documentation [Електронний ресурс] – Режим доступу: <https://dev.mysql.com/doc/>
20. Sundaramoorthy, S. UML Diagramming: A Case Study Approach. Taylor & Francis, 2022. 416 с. ISBN 978-1032120782.

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				64
Змн.	Арк.	№ докум.	Підпис	Дата		

21. What is Use Case Diagram? [Електронний ресурс]. – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/use-case-diagram/>
22. Vaskaran Sarcar Java Design Patterns, Third Edition. Packt Publishing, 2022. ISBN 978-1839219598.
23. The DTO Pattern (Data Transfer Object) [Електронний ресурс] – Режим доступу: <https://www.baeldung.com/java-dto-pattern>
24. Activity Diagram | Guide [Електронний ресурс]. – Режим доступу: <https://creately.com/guides/activity-diagram-tutorial/>
25. OAuth 2.0 Resource Server JWT [Електронний ресурс]. – Режим доступу: <https://docs.spring.io/spring-security/reference/servlet/oauth2/resource-server/jwt.html>
26. Spring Security With JWT for REST API [Електронний ресурс] – Режим доступу: <https://www.toptal.com/spring/spring-security-tutorial>
27. Spring Data JPA [Електронний ресурс] – Режим доступу: <https://spring.io/projects/spring-data-jpa>
28. Stripe [Електронний ресурс] – Режим доступу: <https://stripe.com>
29. Riecks, Philip. Testing Spring Boot Applications Demystified: Avoiding Pitfalls, Implementing Recipes, and Embracing Best Practices. Leanpub, 2023.
30. Testing Spring Boot App [Електронний ресурс] – Режим доступу: <https://docs.spring.io/spring-boot/reference/testing/spring-boot-applications.html>

ДОДАТКИ

		Кобилко М. П.			ІПЗ.КР.Б-121-25-ПЗ	Арк.
		Вакалюк Т. А.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

Клас JWTService

```

package com.kobylko.airline.service;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

import javax.crypto.SecretKey;
import java.util.Base64;
import java.util.Date;
import java.util.Map;
import java.util.function.Function;

@Service
public class JWTService {

    @Value("${jwt.secret}")
    private String secretKey;
    private SecretKey getKey() {
        byte[] keyBytes = Decoders.BASE64.decode(secretKey);
        return Keys.hmacShaKeyFor(keyBytes);
    }

    public String generateToken(String username) {
        return buildToken(username, 30 * 60 * 1000);
    }

    public String generateRefreshToken(String username) {
        return buildToken(username, 7L * 24 * 60 * 60 * 1000);
    }

    private String buildToken(String username, long expirationMillis) {
        return Jwts.builder()
            .claims()
            .add(Map.of())
            .subject(username)
            .issuedAt(new Date(System.currentTimeMillis()))
            .expiration(new Date(System.currentTimeMillis() +
expirationMillis))
            .and()
            .signWith(getKey())
            .compact();
    }

    public String extractUserName(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public boolean validateToken(String token, UserDetails userDetails) {
        final String userName = extractUserName(token);
        return (userName.equals(userDetails.getUsername()) &&
!isTokenExpired(token));
    }

    public boolean isTokenExpired(String token) {
        Date expirationDate = extractClaim(token, Claims::getExpiration);
        return expirationDate.before(new Date());
    }

    private <T> T extractClaim(String token, Function<Claims, T> claimResolver) {

```

```

        final Claims claims = extractAllClaims(token);
        return claimResolver.apply(claims);
    }

    private Claims extractAllClaims(String token) {
        return Jwts.parser()
            .verifyWith(getKey())
            .build()
            .parseSignedClaims(token)
            .getPayload();
    }
}

```

Клас AuthController

```

package com.kobylko.airline.controller;

import com.kobylko.airline.event.UserRegisteredEvent;
import com.kobylko.airline.model.UserPrincipal;
import com.kobylko.airline.model.UserRole;
import org.springframework.context.ApplicationEventPublisher;
import com.kobylko.airline.event.TicketPurchasedEvent;
import com.kobylko.airline.model.User;
import com.kobylko.airline.repository.UserRepository;
import com.kobylko.airline.service.UserService;
import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.*;
import java.util.Map;
import java.util.Optional;

@Slf4j
@RequiredArgsConstructor
@RestController
@RequestMapping("/auth")
public class AuthController {
    private final UserRepository userRepository;
    private final UserService userService;
    private final ApplicationEventPublisher eventPublisher;

    @GetMapping("/me")
    public Map<String, Object> getMe() {
        Long userId = userService.getAuthenticatedUserId();
        if (userId == null) {
            return Map.of("role", "guest", "position", "", "fullName", "");
        }
        Optional<User> finderUser = userRepository.findById(userId);
        if (finderUser.isEmpty()) {
            return Map.of("role", "guest", "position", "", "fullName", "");
        }
        User user = finderUser.get();
        String role = user.getRole().name().toLowerCase();
        String position = "";

        if (user.getRole() == UserRole.WORKER && user.getWorker() != null) {
            position = user.getWorker().getPosition().name().toLowerCase();
        }
        return Map.of(

```

```

        "role", role,
        "position", position,
        "fullName", user.getFullName() != null ? user.getFullName() :
user.getEmail()
    );
}

    public static void addJwtToCookie(String token, HttpServletResponse response)
{
    Cookie jwtCookie = new Cookie("jwt", token);
    jwtCookie.setHttpOnly(true);
    jwtCookie.setPath("/");
    jwtCookie.setMaxAge(30 * 60);
    jwtCookie.setSecure(true);
    response.addCookie(jwtCookie);
}

    public static void addRefreshTokenToCookie(String token, HttpServletResponse
response) {
    Cookie refreshCookie = new Cookie("refresh", token);
    refreshCookie.setHttpOnly(true);
    refreshCookie.setPath("/");
    refreshCookie.setMaxAge(7 * 24 * 60 * 60);
    refreshCookie.setSecure(true);
    response.addCookie(refreshCookie);
}

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestParam String email, @RequestParam
String password, HttpServletResponse response) {

        User user = new User();
        user.setEmail(email);
        user.setPassword(password);
        String token = userService.verify(user);
        addJwtToCookie(token, response);
        String refreshToken = userService.generateRefreshToken(user.getEmail());
        addRefreshTokenToCookie(refreshToken, response);
        return ResponseEntity.ok().body("Login successful");
    }

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestParam String email, @RequestParam
String password, HttpServletResponse response) {
        if (userRepository.findByEmail(email) != null) {
            return
ResponseEntity.status(HttpServletResponse.SC_CONFLICT).body("Email already in
use");
        }
        User user = new User();
        user.setEmail(email);
        user.setPassword(password);
        userService.register(user, UserRole.PASSENGER);
        String token = userService.verifyForRegister(email, password);
        addJwtToCookie(token, response);
        String refreshToken = userService.generateRefreshToken(user.getEmail());
        addRefreshTokenToCookie(refreshToken, response);
        eventPublisher.publishEvent(new UserRegisteredEvent(this, user));
        return ResponseEntity.ok().body("Registration successful");
    }

    @PostMapping("/logout")
    public ResponseEntity<String> logout(HttpServletRequest request,
HttpServletResponse response) {
        Cookie jwtCookie = new Cookie("jwt", null);
        jwtCookie.setHttpOnly(true);
        jwtCookie.setSecure(true);
        jwtCookie.setPath("/");

```

```

        jwtCookie.setMaxAge(0);

        Cookie refreshCookie = new Cookie("refresh", null);
        refreshCookie.setHttpOnly(true);
        refreshCookie.setSecure(true);
        refreshCookie.setPath("/");
        refreshCookie.setMaxAge(0);
        response.addCookie(jwtCookie);
        response.addCookie(refreshCookie);
        SecurityContextHolder.clearContext();
        return ResponseEntity.ok("Logout successful");
    }
}

```

Клас JwtFilter

```

package com.kobylko.airline.config;

import com.kobylko.airline.controller.AuthController;
import com.kobylko.airline.service.JWTService;
import com.kobylko.airline.service.MyUserDetailsService;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
import org.springframework.web.util.WebUtils;

import java.io.IOException;

@Component
@Slf4j
@RequiredArgsConstructor
public class JwtFilter extends OncePerRequestFilter {

    private final JWTService jwtService;
    private final MyUserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
        HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {

        String token = null;
        String refreshToken = null;
        String email = null;
        String authHeader = request.getHeader("Authorization");
        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            token = authHeader.substring(7);
        }
        if (token == null) {
            Cookie cookie = WebUtils.getCookie(request, "jwt");
            if (cookie != null) {

```

```

        token = cookie.getValue();
    }
    Cookie refreshCookie = WebUtils.getCookie(request, "refresh");
    if (refreshCookie != null) {
        refreshToken = refreshCookie.getValue();
    }

    if (token != null) {
        email = jwtService.extractUserName(token);
    }
    if (email == null && refreshToken != null) {
        email = jwtService.extractUserName(refreshToken);
    }
    if (email != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {

        UserDetails userDetails =
userDetailsService.loadUserByUsername(email);

        if (token == null || !jwtService.validateToken(token, userDetails)) {
            log.warn("Token expired/missing: {}", trying to generate a new
one", email);
            if (refreshToken == null) {
                log.warn("Missing refresh token: {}", email);
            } else {
                if (!jwtService.validateToken(refreshToken, userDetails)) {
                    log.warn("Invalid refresh token: {}", email);
                } else {
                    token = jwtService.generateToken(email);
                    AuthController.addJwtToCookie(token, response);
                    log.info("Successfully refreshed token: {}", email);
                }
            }
        }

        if (token != null && jwtService.validateToken(token, userDetails)) {
            UsernamePasswordAuthenticationToken authToken =
                new UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
            authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
            SecurityContextHolder.getContext().setAuthentication(authToken);
            log.info("Authentication successful for user: {}", email);
        }
    }

    if (token == null) {
        log.warn("Unable to authenticate: {}", email);
        SecurityContextHolder.clearContext();
    }

    filterChain.doFilter(request, response);
}
}

```

Клас SecurityConfig

```

package com.kobylko.airline.config;

import jakarta.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;

```

```

import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSe
curity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.config.annotation.authentication.configuration.Authen
ticationConfiguration;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilt
er;

@Configuration
@EnableMethodSecurity
public class SecurityConfig {

    @Autowired
    private JwtFilter jwtFilter;
    @Autowired
    private UserDetailsService userDetailsService;
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        return http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(
                    "/auth/login",
                    "/auth/register",
                    "/",
                    "/auth/logout"
                ).permitAll()
                .anyRequest().authenticated()
            )
            .exceptionHandling(exception -> exception
                .authenticationEntryPoint((request, response,
authException) -> {
                    response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
                    response.getWriter().write("Unauthorized");
                })
            )
            .sessionManagement(session -> session
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            )
            .addFilterBefore(jwtFilter,
UsernamePasswordAuthenticationFilter.class)
            .build();
    }

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

    @Bean

```

```

    public AuthenticationManager authenticationManager(AuthenticationConfiguration
config) throws Exception {
        return config.getAuthenticationManager();
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(12);
    }
}

```

Клас FlightController

```

package com.kobylko.airline.controller;

import com.kobylko.airline.dto.ErrorResponse;
import com.kobylko.airline.dto.FlightDTO;
import com.kobylko.airline.model.*;
import com.kobylko.airline.repository.AirplaneRepository;
import com.kobylko.airline.repository.FlightRepository;
import com.kobylko.airline.repository.UserRepository;
import com.kobylko.airline.repository.WorkerRepository;
import com.kobylko.airline.service.*;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.*;
import org.springframework.security.access.prepost.PreAuthorize;

@RestController
@RequiredArgsConstructor
@Slf4j
@RequestMapping("/flights")
public class FlightController {

    private final FlightRepository flightRepository;
    private final FlightService flightService;
    private final AirplaneRepository airplaneRepository;
    private final WorkerService workerService;
    private final UserService userService;
    private final UserRepository userRepository;
    private final WorkerRepository workerRepository;
    private final TimeZoneConverter timeZoneConverter;

    @PreAuthorize("hasRole('ADMIN')")
    @GetMapping("/all")
    public List<Flight> flights() {
        return flightRepository.findAll();
    }

    @PreAuthorize("hasRole('ADMIN')")
    @PostMapping("/add")
    public ResponseEntity<String> addFlight(@RequestParam long routeId,
                                             @RequestParam @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE_TIME) LocalDateTime departureTime,
                                             @RequestParam long airplaneId,
                                             @RequestParam(defaultValue = "NONE")
String technicalType) {
        Airplane airplane =
airplaneRepository.findAirplaneByIdAirplane(airplaneId);

```

```

        if (airplane == null) {
            return ResponseEntity.badRequest().body("Airplane with the given
serial number not found.");
        }
        return flightService.addFlight(routeId, departureTime, airplane,
technicalType.toUpperCase());
    }

    @PreAuthorize("hasRole('ADMIN')")
    @PostMapping("/changeStatus")
    public ResponseEntity<String> changeStatus(@RequestParam("status") String
status,
                                              @RequestParam("flightId") Long
flightId) {
        Flight flight = flightRepository.findById(flightId)
            .orElseThrow(() -> new IllegalArgumentException("Flight with this
ID does not exist."));
        try {
            Status newStatus = Status.valueOf(status.toUpperCase());
            flight.setStatus(newStatus);
            flightRepository.save(flight);

            return ResponseEntity.ok("Flight status updated successfully.");
        } catch (IllegalArgumentException e) {
            return ResponseEntity.badRequest().body("Invalid status value.");
        }
    }

    @PreAuthorize("hasRole('ADMIN')")
    @PostMapping("/addWorkersToFlight")
    public ResponseEntity<?> addWorkersToFlight(@RequestParam("flightId") Long
flightId) {

        Flight flight = flightRepository.findById(flightId)
            .orElseThrow(() -> new RuntimeException("Flight not found"));

        if(flight.getAirplane() == null){
            return ResponseEntity.badRequest().body(
                new ErrorResponse("ADD_ERROR", "Firstly add airplane",
"error.worker.add"));
        }

        return workerService.addWorkersToFlight(flight);
    }

    @GetMapping("/available")
    public List<FlightDTO> viewAvailableFlights(
        @RequestParam("departureAirport") String departureAirportCity,
        @RequestParam("arrivalAirport") String arrivalAirportCity,
        @RequestParam("departureTime") @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate departureDate) {
        List<FlightDTO> availableFlights =
flightService.searchFlights(departureAirportCity, arrivalAirportCity,
departureDate);

        return availableFlights;
    }

    @PreAuthorize("hasRole('WORKER')")
    @GetMapping("/me")
    public ResponseEntity<?> seeWorkerFlights() {
        Long userId = userService.getAuthenticatedUserId();
        if (userId == null) {
            return ResponseEntity.status(401).body("User not authenticated");
        }
        Worker worker = workerRepository.findByUser_IdUser(userId)

```

```

        .orElseThrow(() -> new IllegalArgumentException("Worker not
found"));

        List<Flight> flights =
flightRepository.findById(worker.getIdWorker());

        return ResponseEntity.ok(flights);
    }

    @PreAuthorize("hasRole('ADMIN')")
    @PostMapping("/setTechnical")
    public ResponseEntity<?> setTechnical(@RequestParam Long flightId,
@RequestParam boolean isTechnical) {
        Flight flight = flightRepository.findById(flightId)
            .orElseThrow(() -> new RuntimeException("Flight not found"));

        if (isTechnical && flight.getAirplane() != null) {
            return ResponseEntity.badRequest().body("Cannot set flight as
technical after airplane is assigned");
        }
        flight.setTechnical(isTechnical);
        flightRepository.save(flight);
        return ResponseEntity.ok("Flight is technical.");
    }
}

```

Клас workerService

```

package com.kobylko.airline.service;

import com.kobylko.airline.exception.EmailAlreadyExistsException;
import com.kobylko.airline.exception.ResourceNotFoundException;
import com.kobylko.airline.model.*;
import com.kobylko.airline.repository.AirportRepository;
import com.kobylko.airline.repository.FlightRepository;
import com.kobylko.airline.repository.WorkerRepository;
import jakarta.transaction.Transactional;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.data.domain.PageRequest;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import java.time.Duration;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class WorkerService {

    private final WorkerRepository workerRepository;
    private final UserService userService;
    private final AirportRepository airportRepository;
    private final EmailService emailService;
    private final AirplaneService airplaneService;
    private final WorkHoursService workHoursService;
    private final FlightRepository flightRepository;

    @Transactional
    public ResponseEntity<String> addWorkersToFlight(Flight flight) {
        Airport departureAirport = airplaneService.getDepartureAirport(flight);

        List<Flight> roundTripFlights =

```

```

airplaneService.getFlightsByRoundTrip(flight);
    LocalDateTime roundTripStart = roundTripFlights.get(0).getDepartureTime();
    LocalDateTime roundTripEnd = roundTripFlights.get(roundTripFlights.size()
- 1).getArrivalTime();
    long tripDuration = Duration.between(roundTripStart,
roundTripEnd).toHours();

    log.info("TRIP DURATION - {}", tripDuration);

    int month = roundTripStart.getMonthValue();
    int year = roundTripStart.getYear();
    int crewNeeded = calculateMinimumCrew(flight);
    List<Worker> crew = workerRepository
        .getCrewForFlight(departureAirport, month, year, tripDuration,
PageRequest.of(0, crewNeeded));

    if (crew.size() < crewNeeded) {
        return ResponseEntity.badRequest().body("Not enough crew available");
    }
    log.info("workers");
    for (int i = 0; i < crew.size(); i++) {
        log.info(crew.get(i).getName());
    }
    List<Worker> pilots = workerRepository
        .getPilotsForFlight(flight.getAirplane().getModel().getModelName(),
departureAirport, month, year, tripDuration,
flight.getArrivalTime().toLocalDate(), PageRequest.of(0, 2));
    if (pilots.size() < 2) {
        return ResponseEntity.badRequest().body("Not enough pilots
available");
    }
    for (int i = 0; i < pilots.size(); i++) {
        log.info(pilots.get(i).getName());
    }
    for (Flight f : roundTripFlights) {
        if (f.getWorkers() == null) {
            f.setWorkers(new ArrayList<>());
        }
        for (Worker pilot : pilots) {
            if (!f.getWorkers().contains(pilot)) {
                f.getWorkers().add(pilot);
            }
        }
        for (Worker crewMember : crew) {
            if (!f.getWorkers().contains(crewMember)) {
                f.getWorkers().add(crewMember);
            }
        }
        flightRepository.save(f);
    }
    workHoursService.updateWorkHours(pilots, roundTripStart, tripDuration);
    workHoursService.updateWorkHours(crew, roundTripStart, tripDuration);
    return ResponseEntity.ok("Workers successfully assigned to the flight");
}
private int calculateMinimumCrew(Flight flight) {
    int seats = flight.getAirplane().getLayout().getCountAllSeats();
    return (int) Math.ceil(seats / 50.0);
}

@Transactional
public void registerWorker(String name, String position, Long homeAirportId,
                           String email, String password) {

    User workerUser = new User();
    workerUser.setEmail(email);

```

```

        workerUser.setPassword(password);
        try {
            userService.register(workerUser, UserRole.WORKER);
        } catch (DataIntegrityViolationException ex) {
            if
(ex.getMostSpecificCause().getMessage().contains("user.unique_email")) {
                throw new EmailAlreadyExistsException("A user with this email
already exists.");
            }
            throw ex;
        }
        Worker worker = new Worker();
        worker.setName(name);
        worker.setPosition(Position.valueOf(position));
        Airport homeAirport = airportRepository.findById(homeAirportId)
            .orElseThrow(() -> new ResourceNotFoundException("Airport with ID
" + homeAirportId + " not found"));
        worker.setHomeAirport(homeAirport);
        worker.setUser(workerUser);
        workerRepository.save(worker);
        emailService.sendWorkerRegistrationEmail(email, password);
    }
}

```

Клас WorkerRepository

```

package com.kobylko.airline.repository;

import com.kobylko.airline.model.Airport;
import com.kobylko.airline.model.Worker;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import java.time.LocalDate;
import java.util.List;
import java.util.Optional;

public interface WorkerRepository extends JpaRepository<Worker, Long> {
    boolean existsByUser_IdUser(Long idUser);
    Optional<Worker> findByUser_IdUser(Long idUser);
    @Query("""
        SELECT DISTINCT w FROM Worker w
        JOIN w.certifications c
        LEFT JOIN WorkHour wh ON wh.worker = w AND wh.month = :month AND
wh.year = :year
        LEFT JOIN WorkHour wh2 ON wh2.worker = w AND wh2.year = :year
        WHERE w.position = 'PILOT'
        AND w.homeAirport = :departureAirport
        AND c.airplaneModel.modelName LIKE :airplaneModelName
        AND c.issueDate <= :flightDate
        AND c.expireDate >= :flightDate
        AND COALESCE(wh.hoursWorked, 0) + :tripDurationHours <= 100
        GROUP BY w
        HAVING COALESCE(SUM(wh2.hoursWorked), 0) + :tripDurationHours <=
1000
        ORDER BY FUNCTION('RAND')
    """)
    List<Worker> getPilotsForFlight(
        @Param("airplaneModelName") String airplaneModelName,
        @Param("departureAirport") Airport departureAirport,
        @Param("month") int month,
        @Param("year") int year,
        @Param("tripDurationHours") long tripDurationHours,
        @Param("flightDate") LocalDate flightDate,
        Pageable pageable
    )
}

```

```

    );
    @Query("""
        SELECT DISTINCT w FROM Worker w
        LEFT JOIN WorkHour wh ON wh.worker = w AND wh.month = :month AND
wh.year = :year
        LEFT JOIN WorkHour wh2 ON wh2.worker = w AND wh2.year = :year
        WHERE w.position = 'CREWMEMBER'
            AND w.homeAirport = :departureAirport
            AND COALESCE(wh.hoursWorked, 0) + :tripDurationHours <= 100
        GROUP BY w
        HAVING COALESCE(SUM(wh2.hoursWorked), 0) + :tripDurationHours <= 1000
        ORDER BY FUNCTION('RAND')
        """)
    List<Worker> getCrewForFlight(
        @Param("departureAirport") Airport departureAirport,
        @Param("month") int month,
        @Param("year") int year,
        @Param("tripDurationHours") long tripDurationHours,
        Pageable pageable
    );
}

```

Клас SeatService, метод seatsGeneration

```

package com.kobylko.airline.service;

import com.kobylko.airline.model.Airplane;
import com.kobylko.airline.model.Layout;
import com.kobylko.airline.model.Seat;
import com.kobylko.airline.repository.SeatRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service
@RequiredArgsConstructor
@Slf4j
public class SeatService {
    private final SeatRepository seatRepository;
    public void seatsGeneration(Layout layout, Airplane airplane) {
        String seatsNumber = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        List<Seat> seatInAirplane = new ArrayList<>();
        for (int i = 0; i < layout.getCountRows(); i++) {
            for(int j = 0; j < layout.getCountColumn(); j++){
                for (int k = 0; k < layout.getCountSeatInCol(); k++) {
                    Seat seat = new Seat();
                    seat.setSeatNumber((i + 1) + "" + seatsNumber.charAt(j *
layout.getCountSeatInCol() + k));
                    seat.setAirplane(airplane);
                    if (i < layout.getCountFirstClass()) {
                        seat.setSeatClass("FirstClass");
                    } else {
                        seat.setSeatClass("Economy");
                    }
                    seatInAirplane.add(seat);
                }
            }
        }
        seatRepository.saveAll(seatInAirplane);
    }
}

```

Клас FlightRepository

```
package com.kobylko.airline.repository;

import com.kobylko.airline.dto.FlightDTO;
import com.kobylko.airline.dto.TicketDetailsDTO;
import com.kobylko.airline.model.Flight;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;

public interface FlightRepository extends JpaRepository<Flight, Long> {

    @Query("SELECT COUNT(f) > 0 FROM Flight f " +
        "JOIN f.route r " +
        "WHERE (r.departureAirport.idAirport = :airportId OR " +
        r.arrivalAirport.idAirport = :airportId) " +
        "AND f.departureTime >= :now")
    boolean existsFlightsByAirport(@Param("airportId") Long airportId,
        @Param("now") LocalDateTime now);

    @Query("SELECT new com.kobylko.airline.dto.FlightDTO( " +
        "f.idFlight, " +
        "depAir.name, " +
        "depAir.city, " +
        "arrAir.name, " +
        "arrAir.city, " +
        "f.departureTime, " +
        "f.arrivalTime, " +
        "MAX(t.price), " +
        "depAir.timezone, " +
        "arrAir.timezone) " +
        "FROM Flight f " +
        "JOIN f.route r " +
        "JOIN r.departureAirport depAir " +
        "JOIN r.arrivalAirport arrAir " +
        "JOIN Ticket t ON t.flight.idFlight = f.idFlight " +
        "WHERE depAir.city = :departureAirportCity " +
        "AND arrAir.city = :arrivalAirportCity " +
        "AND DATE(f.departureTime) = :departureTime " +
        "GROUP BY f.idFlight, depAir.name, depAir.city, depAir.country, " +
        "arrAir.name, arrAir.city, arrAir.country, f.departureTime, " +
        f.arrivalTime")
    List<FlightDTO> searchFlights(
        @Param("departureAirportCity") String departureAirportCity,
        @Param("arrivalAirportCity") String arrivalAirportCity,
        @Param("departureTime") LocalDate departureTime);

    List<Flight> findFlightsByRoundTripId(String roundTripId);

    @Query("SELECT COUNT(f) > 0 FROM Flight f WHERE f.airplane.idAirplane = " +
        ":airplaneId AND " +
        "(:start BETWEEN f.departureTime AND f.arrivalTime OR " +
        ":end BETWEEN f.departureTime AND f.arrivalTime OR " +
        "f.departureTime BETWEEN :start AND :end OR " +
        "f.arrivalTime BETWEEN :start AND :end)")
    boolean airplaneHasConflictingFlights(@Param("airplaneId") Long airplaneId,
        @Param("start") LocalDateTime start,
        @Param("end") LocalDateTime end);
```

```

@Query("SELECT f FROM Flight f JOIN f.workers w WHERE w.idWorker = :workerId")
List<Flight> findByWorkerId(@Param("workerId") Long workerId);

@Query("SELECT f FROM Flight f " +
        "JOIN f.tickets t " +
        "WHERE t.passenger.idPassenger = :passengerId " +
        "AND f.arrivalTime BETWEEN :startOfDay AND :endOfDay")
List<Flight> findFlightsByPassengerOnDate(@Param("passengerId") long
passengerId,
                                         @Param("startOfDay") LocalDateTime
startOfDay,
                                         @Param("endOfDay") LocalDateTime
endOfDay);

@Query("""
        SELECT new com.kobylko.airline.dto.TicketDetailsDTO(
            t.idTicket,
            COALESCE(p.amount, t.price),
            pas.fullName,
            doc.documentNumber,
            s.seatNumber,
            f.idFlight, f.departureTime, f.arrivalTime,
            depAirport.name, depAirport.city,
            arrAirport.name, arrAirport.city,
            b.type,
            depAirport.timezone,
            arrAirport.timezone)
        FROM Ticket t
        JOIN t.flight f
        JOIN f.route r
        JOIN r.departureAirport depAirport
        JOIN r.arrivalAirport arrAirport
        JOIN t.seat s
        JOIN t.passenger pas
        LEFT JOIN PassengerDocument doc ON doc.passenger = pas
        LEFT JOIN Payment p ON p.ticket.idTicket = t.idTicket
        LEFT JOIN t.baggage b
        WHERE t.passenger.user.idUser = :userId
        """)
List<TicketDetailsDTO> findTicketsByUserId(@Param("userId") Long userId);

@Query("SELECT COUNT(f) > 0 FROM Flight f WHERE f.airplane.idAirplane =
:airplaneId AND f.departureTime >= :now")
boolean existsByAirplaneIdAndDepartureTimeAfter(@Param("airplaneId") Long
airplaneId,
                                                  @Param("now") LocalDateTime
now);

@Query("SELECT f FROM Flight f " +
        "JOIN f.workers w " +
        "WHERE w.idWorker = :workerId " +
        "AND f.departureTime <= :newFlightDepartureTime " +
        "ORDER BY f.departureTime DESC")
Optional<Flight> findPreviousFlightForWorkerById(
    @Param("workerId") Long workerId,
    @Param("newFlightDepartureTime") LocalDateTime
newFlightDepartureTime);

@Query("SELECT f FROM Flight f " +
        "JOIN f.workers w " +
        "WHERE w.idWorker = :workerId " +
        "AND f.departureTime > :newFlightDepartureTime " +
        "ORDER BY f.departureTime ASC")

```

```

        Optional<Flight> findFutureFlightsForWorker(
            @Param("workerId") Long workerId,
            @Param("newFlightDepartureTime") LocalDateTime
newFlightDepartureTime);
    }

```

Клас AirportApiService

```

package com.kobylko.airline.service;

import com.kobylko.airline.dto.AirportApiDTO;
import com.kobylko.airline.exception.AirportAlreadyExistsException;
import com.kobylko.airline.model.Airport;
import com.kobylko.airline.repository.AirportRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
@Service
@Slf4j
@RequiredArgsConstructor
public class AirportApiService {
    private final RestTemplate restTemplate;
    private final AirportRepository airportRepository;
    private static final String API_URL = "https://api.api-
ninjas.com/v1/airports?iata={iata}";
    @Value("${api.ninjas.key}")
    private String apiKey;
    public List<AirportApiDTO> fetchAirportsByIata(String iata) {
        HttpHeaders headers = new HttpHeaders();
        headers.set("X-API-Key", apiKey);
        HttpEntity<String> entity = new HttpEntity<>(headers);
        ResponseEntity<AirportApiDTO[]> response = restTemplate.exchange(
            API_URL, HttpMethod.GET, entity, AirportApiDTO[].class, iata);

        return response.getBody() != null ? Arrays.asList(response.getBody()) :
Collections.emptyList();
    }
    public String saveAirport(AirportApiDTO dto) {
        if (airportRepository.existsByName(dto.getName())) {
            throw new AirportAlreadyExistsException("Airport already exists with
name: " + dto.getName());
        }
        Airport airport = new Airport();
        airport.setName(dto.getName());
        airport.setCity(dto.getCity());
        airport.setCountry(dto.getCountry());
        airport.setIata(dto.getIata());
        airport.setIcao(dto.getIcao());
        airport.setTimezone(dto.getTimezone());
        airport.setLatitude(dto.getLatitude());
        airport.setLongitude(dto.getLongitude());
        airport.setActive(true);
        airportRepository.save(airport);

        return "Airport already exists";
    }
}

```

Клас FlightApiService

```
package com.kobylko.airline.service;

import com.kobylko.airline.dto.FlightTimeResponse;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.*;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
import java.util.HashMap;
import java.util.Map;

@Service
@RequiredArgsConstructor
@Slf4j
public class FlightApiService {

    private final RestTemplate restTemplate;

    @Value("${api.aviapages.flight_calculator.url}")
    private String apiUrl;

    @Value("${tokenAuth}")
    private String apiKey;

    public double fetchFlightTime(String departureAirport, String arrivalAirport,
String aircraft) {

        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);
        headers.set("Authorization", "Token " + apiKey);
        Map<String, Object> requestBody = new HashMap<>();
        requestBody.put("departure_airport", departureAirport);
        requestBody.put("arrival_airport", arrivalAirport);
        requestBody.put("aircraft", aircraft);
        requestBody.put("airway_time", true);
        HttpEntity<Map<String, Object>> requestEntity = new
HttpEntity<>(requestBody, headers);
        try {
            ResponseEntity<FlightTimeResponse> response = restTemplate.exchange(
                apiUrl,
                HttpMethod.POST,
                requestEntity,
                FlightTimeResponse.class
            );
            FlightTimeResponse body = response.getBody();
            if (body != null && body.getTime() != null &&
body.getTime().getAirway() != null) {
                double dbl = body.getTime().getAirway();
                log.info("Flight time (airway) received: {} minutes", dbl);
                return dbl;
            } else {
                log.info("No flight time received from API");
                return 0;
            }
        } catch (Exception e) {
            log.info("Error during API call: {}", e.getMessage(), e);
            return 0;
        }
    }
}
```