МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА
Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА «КОМП'ЮТЕРНІ НАУКИ»
галузь знань 12 «Інформаційні технології»
спеціальність 122 «Комп'ютерні науки»

Форма навчання: денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА

на тему: «Розроблення інформаційної системи з продажу авіаквитків»

здобувача Коця Ярослава Сергійовича _____
(ПІБ) *(Підпис)*

Науковий керівник:
к.т.н., доцент
_____Васильєва Л.В.

Робота допущена до захисту перед екзаменаційною комісією з атестації здобувачів вищої освіти
завідувач кафедри:
к.е.н., доцент
_____Тішков Б.О.

Київ 2025

Міністерство освіти і науки України
Київський національний економічний університет імені Вадима Гетьмана
Навчально-науковий інститут «Інститут інформаційних технологій в економіці»
Кафедра інформаційних систем в економіці

ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА «КОМП'ЮТЕРНІ НАУКИ»
галузь знань 12 «Інформаційні технології»
спеціальність 122 «Комп'ютерні науки»

ПОГОДЖЕНО:
Керівник проєктної групи(гарант)
освітньо-професійної програми
_____Помазун О.М.
"____" _____2025 р.

ЗАТВЕРДЖУЮ:
Завідувач кафедри
_____Тішков Б.О.
"____" _____2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувачу вищої освіти Коцю Ярослав Сергійовичу

очної (денної) форми навчання

на підготовку кваліфікаційної бакалаврської роботи
на тему: «Розроблення інформаційної системи з продажу авіаквитків**»**

Тему затверджено наказом ректора Університету від « *7* » *березня 2025* р. № *466- ст.*

Кваліфікаційна бакалаврська робота виконується на матеріалах

_____

План кваліфікаційної бакалаврської роботи

Розділ I ХАРАКТЕРИСТИКА ТА АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

Розділ II РОЗРОБКА ВИМОГ І МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТ

Розділ III ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

Об'єкт дослідження процес проєктування та реалізації інформаційної системи автоматизованого продажу авіаквитків через веб-застосунок із використанням сучасних технологій.

Предмет дослідження алгоритми та програмно-апаратні засоби, які реалізують описаний процес у вигляді інтегрованої інформаційної системи з центральною базою даних, сервісними модулями для зовнішніх API та фронтенд-інтерфейсами.

Мета кваліфікаційної бакалаврської роботи розробка інформаційної системи, яка забезпечить повний цикл роботи з квитками: від пошуку рейсу та вибору послуг до введення пасажирських даних, оплати й видачі електронного квитка.

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети:

У розділі I охарактеризувати предметну галузь та об'єкт дослідження, обґрунтувати актуальність теми, проаналізувати наукові джерела й приклади впровадження інформаційних систем у сфері авіаперевезень. Розробити логічну та фізичну моделі даних у MongoDB, спроєктувати архітектуру підсистеми з UML-діаграмами, реалізувати API, валідацію, інтерфейс пошуку й бронювання, інтеграцію з платіжною системою.

У розділі II проаналізувати функціональні та нефункціональні вимоги до інформаційної системи, побудувати їх ієрархію, виконати трасування до моделей, створити діаграми прецедентів, класів, компонентів, послідовності та діяльності, а також сформувати інформаційну й даталогічну моделі даних з урахуванням специфіки MongoDB.

У розділі III реалізувати основні компоненти системи: серверне API, пошук, бронювання, оплату, клієнтський інтерфейс, логування, CI/CD та тестування. Провести перевірку відповідності реалізації вимогам та підготувати технічну документацію.

Завдання підготував
науковий керівник _____ **Васильєва Людмила Володимирівна**

« 10 » березня 2025 р.

Завдання одержав
здобувач _____ **Коць Ярослав Сергійович**

« 10 » березня 2025 р..

Відгук
про кваліфікаційну бакалаврську роботу
здобувача навчально-наукового інституту
«Інститут інформаційних технологій в економіці»
освітньо-професійної програми
«Комп'ютерні науки»

Коця Ярослава Сергійовича

на тему
«*Розроблення інформаційної системи з продажу авіаквитків*»

1. Актуальність теми: Тема розроблення інформаційної системи продажу авіаквитків є актуальною через стрімке зростання онлайн-бронювання та потребу авіаперевізників і агентств у сучасних, гнучких рішеннях для управління продажами й тарифами.

2. Позитивні риси кваліфікаційної бакалаврської роботи: У роботі сформульовані мета, завдання та вимоги до системи, проведено ґрунтовний огляд літератури, реалізовані UML-моделі, логічна та фізична моделі бази даних, а також прототип API і клієнтського інтерфейсу.

3. Наявність самостійних розробок автора: Автор розробив власну інформаційну модель процесу бронювання, створив блок-схему алгоритму й імплементував схему валідації даних у MongoDB, а також написав REST-API для взаємодії з платіжними шлюзами.

4. Цінність теоретичних висновків та практичних рекомендацій: Теоретичні висновки щодо побудови архітектури мікросервісної системи та застосування документно-орієнтованої СКБД є практично значимими, а рекомендації щодо оптимізації продуктивності й забезпечення безпеки цінні для реального впровадження.

5. Наявність недоліків: Робота містить обмежену оцінку масштабованості під високі навантаження та відсутній детальний план моніторингу продуктивності в продакшн-середовищі.

6. Загальна оцінка кваліфікаційної бакалаврської роботи та її допущення до захисту перед ЕК: Робота виконана на високому рівні, відповідає вимогам бакалаврського проєкту, містить оригінальні розробки та готова до захисту перед екзаменаційною комісією.

Науковий керівник _____ доцент, к.т.н. Васильєва Л.В.
(підпис)

“_14_”_____06_____ 2025 р.

Рецензія

на кваліфікаційну бакалаврську роботу

здобувача вищої освіти

Коця Ярослав Сергійовича

(прізвище, ім'я, по батькові)

Тема Розроблення інформаційної системи з продажу авіаквитків

Актуальність теми кваліфікаційної бакалаврської роботи і доцільність її розроблення: Онлайн-продаж авіаквитків сьогодні є ключовим каналом збуту, тому створення продуктивної та зручної системи є актуальним. Важливо забезпечити гнучкі тарифи та швидку обробку транзакцій, що вимагає сучасних технологічних рішень.

Якість проведеного дослідження: Автор провів грунтовний аналіз ринку та предметної галузі, виконав огляд вітчизняних і зарубіжних джерел і чітко сформулював функціональні та нефункціональні вимоги. Застосування UML-моделей, інфологічного й даталогічного проектування, а також створення прототипу з використанням MongoDB і REST-API свідчать про високий рівень методологічної і технічної підготовки.

Позитивні риси кваліфікаційної бакалаврської роботи: Робота відзначається чіткою структурою, послідовним переходом від аналізу до реалізації та належним опрацюванням ключових компонентів системи: алгоритм бронювання, моделі даних, механізми валідації. Особливо варто відзначити власну розробку діаграм трасування вимог і реальний прототип API з інтеграцією платіжного шлюзу, що демонструє практичну цінність.

Зауваження: Доопрацювати розділ навантажувального тестування – нинішні результати загальні і не містять порівняння з нормативними показниками. Також слід узгодити стиль документації та надати більше деталей щодо резервного копіювання й відновлення бази даних у продуктивному середовищі.

Практична значимість висновків і рекомендацій: Висновки щодо вибору MongoDB як СКБД, мікросервісної архітектури й підходів до кешування можуть одразу застосовуватися в існуючих системах бронювання. Рекомендації з оптимізації запитів і забезпечення безпеки платежів становлять практичну цінність для IT-відділів авіакомпаній і агентств.

Місце роботи та посада рецензента Професор каф. ПМтаМБС СумДПУ

Науковий ступінь, учене звання *(за наявності)* проф., д.т.н. Пангаров О.І.

(підпис, ПІБ)

Підпис засвідчую: фахівець ВК: ср /О.М.Скляренко/

(посада, підпис)

Місце печатки організації, де працює рецензент

АНОТАЦІЯ

кваліфікаційної бакалаврської роботи
здобувача першого (бакалаврського) рівня вищої освіти, 4 курсу,
виконаної на тему: « Розроблення інформаційної системи з продажу авіаквитків »
Київ: кафедра інформаційних систем в економіці, 2025 р.

Кваліфікаційна бакалаврська робота присвячена розробці інформаційної системи для автоматизації процесу продажу авіаквитків з використанням сучасних веб-технологій. Метою роботи є розробка інформаційної системи, яка забезпечить повний цикл роботи з квитками: від пошуку рейсу та вибору послуг до введення пасажирських даних, оплати й видачі електронного квитка.

У першому розділі здійснено аналіз предметної області: охарактеризовано поточні бізнес-процеси продажу квитків, вивчено існуючі рішення на ринку, визначено ключових стейкхолдерів і вимоги до системи. Використано методи порівняльного аналізу та моделювання бізнес-процесів (BPMN).

Другий розділ присвячений проєктуванню системи: обґрунтовано вибір архітектурного стилю (клієнт-серверна архітектура з компонентами фронтенд та бекенд на Next.js/React), розроблено UML-діаграми класів, послідовностей та компонентів, а також спроєктовано схему бази даних у MongoDB. Застосовано підхід компонентної розробки та REST-архітектуру API.

У третьому розділі представлено конструктивне втілення проєкту: описано технічне та програмне забезпечення, реалізацію основних модулів (авторизація користувачів, пошук і бронювання квитків, оплата), наведено скріншоти інтерфейсу й результати тестування. Особливу увагу приділено механізму обліку залишкових місць і гарантії цілісності даних.

Новизна матеріалу полягає в інтеграції гнучкої системи управління квотами місць із сучасним стеком веб-технологій, що забезпечує високу продуктивність, масштабованість та зручність використання для обох категорій користувачів – пасажирів і операторів авіакомпаній.

РЕФЕРАТ

Кваліфікаційна бакалаврська робота містить 86 сторінок, 5 таблиць, 25 рисунків, перелік джерел посилань з 28 найменувань, 6 додатків.

«Розробка інформаційної системи з продажу авіаквитків»

Об’єкт дослідження: процес проєктування та реалізації веб-орієнтованої системи продажу авіаквитків із автоматизацією пошуку, бронювання, оплати й видачі електронного квитка.

Предмет дослідження: внутрішня структура даних і алгоритми обробки транзакцій, необхідні для забезпечення ефективного пошуку рейсів, керування інвентарем місць і інтеграції з платіжними шлюзами.

Мета і завдання дослідження полягає в розробці інформаційної системи, яка забезпечить повний цикл роботи з квитками: від пошуку рейсу та вибору послуг до введення пасажирських даних, оплати й видачі електронного квитка.

Теоретична, методична та практична значущість: результатами роботи стали узагальнення підходів до моделювання інформаційних систем у сфері електронної комерції, демонстрація застосування UML/OPM та сучасних веб-технологій, а також розробка готового прототипу, який легко впровадити й масштабувати під потреби авіакомпаній та турагентств.

Рік виконання – 2025.

Рік захисту – 2025.

Ключові слова: інформаційна система, продаж авіаквитків, бронювання, веб-додаток, REST API, React, MongoDB.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ІЗ – інформаційне забезпечення

ІС – інформаційна система

СКБД – система керування базами даних

АРМ – автоматизоване робоче місце

API – інтерфейс програмування застосунків (Application Programming Interface)

JSON – JavaScript Object Notation

VPN – віртуальна приватна мережа (Virtual Private Network)

CDN – мережа доставки контенту (Content Delivery Network)

SLA – угода про рівень обслуговування (Service Level Agreement)

CLI – інтерфейс командного рядка (Command Line Interface)

ER – сутнісно-зв’язкова діаграма (Entity-Relationship)

DDL – мова опису даних (Data Definition Language)

SPA – односторінковий додаток (Single-Page Application)

UML – універсальна мова моделювання (Unified Modeling Language)

RAM – оперативна пам’ять (Random Access Memory)

SSD – твердотільний накопичувач (Solid State Drive)

RPS – запити за секунду (Requests Per Second)

ВСТУП

У сучасних умовах стрімкого розвитку інформаційних технологій та глобалізації авіаційної галузі зростає потреба в ефективних інформаційних системах, що забезпечують оперативний та зручний процес продажу авіаквитків. Вибір теми «Розроблення інформаційної системи з продажу авіаквитків» зумовлений аналізом ринку електронної комерції в авіаційному сегменті, де через відсутність уніфікованих і гнучких рішень багато компаній стикаються з проблемами високої складності обробки запитів користувачів, обмеженою можливістю налаштування тарифних політик та низькою швидкістю обробки резервувань. Предметною галуззю дослідження виступають інформаційні системи електронної комерції в авіаційному секторі, а об’єктом – процес електронного бронювання та придбання квитків на авіарейси через веб- та мобільні додатки. Метою роботи є створення прототипу системи, яка б поєднала оптимізовані алгоритми пошуку рейсів, гнучке адміністрування бази даних квитків і клієнтську частину з інтуїтивно зрозумілим інтерфейсом, здатним адаптуватися під різні пристрої. Для досягнення цієї мети поставлено такі завдання: аналіз існуючих підходів і технологій у сфері продажу авіаквитків, формалізація бізнес–процесу бронювання й оплати квитків, розробка моделі даних, вибір архітектурних компонентів та технологій реалізації, створення користувацького інтерфейсу й API для інтеграції з платіжними сервісами, а також проведення тестування продуктивності та зручності використання системи.

Актуальність дослідження обумовлена як соціальною значущістю теми – забезпеченням своєчасної й доступної інформації для пасажирів та оптимізації роботи авіакомпаній і туристичних агентств, так і науковим завданням: розробкою методології побудови масштабованих високонавантажених систем бронювання. У світовій літературі питанням проєктування подібних систем присвячено праці Лаудона і Лаудона (2018), у якій розглядаються моделі електронної комерції й трансформація бізнес-процесів в авіації; Бітнер та Фіцпатрик (2019) приділяють увагу оптимізації алгоритмів пошуку й сортування великих масивів даних;

дослідження Олійника (2020) висвітлюють питання інтеграції з платіжними шлюзами та безпеки транзакцій; у роботах Пархоменка (2021) аналізуються архітектурні шаблони для побудови мікросервісних систем; праці Іваненка (2022) присвячені дизайну користувацького інтерфейсу для різних платформ. Ці джерела дозволяють окреслити сучасний рівень проблематики і виявити невирішені питання, пов’язані з гнучкістю налаштувань і адаптивністю під потреби користувачів, що й обґрунтовує необхідність проведення даного дослідження. Загалом у наукових публікаціях представлено значну базу знань про алгоритмічну обробку запитів, однак існує недостатність досліджень у сфері поєднання високої продуктивності та зручності користування в контексті мобільних пристроїв. Таким чином, літературний огляд свідчить про актуальність предмета дослідження та необхідність розробки комплексного підходу до створення системи продажу авіаквитків.

Об’єктом дослідження є процес проєктування та реалізації веб-орієнтованої системи продажу авіаквитків із автоматизацією пошуку, бронювання, оплати й видачі електронного квитка.

Дослідження фокусуватиметься саме на цих компонентах як ключових елементах, що визначають ефективність та надійність роботи системи в цілому.

Мета дослідження розробка інформаційної системи, яка забезпечить повний цикл роботи з квитками: від пошуку рейсу та вибору послуг до введення пасажирських даних, оплати й видачі електронного квитка. Питома увага приділяється юзабіліті клієнтської частини: інтуїтивність і адаптивність інтерфейсу повинні сприяти підвищенню рівня задоволеності користувачів. Відповідно до мети сформульовано завдання, серед яких: аналіз теоретичних основ обробки даних і бізнес–логіки бронювання, вивчення практичного досвіду реалізації подібних систем, розробка технічного завдання й методичних рекомендацій для подальшого впровадження, а також тестування розробленого програмного продукту в умовах наближених до реальних навантажень.

У роботі використано низку методів дослідження, серед яких аналіз структурно-логічний, що дозволив моделювати бізнес-процеси бронювання; метод

математичного моделювання для оптимізації пошукових алгоритмів; експериментальний метод у процесі тестування продуктивності; вербально-описовий під час узагальнення результатів літературного огляду; а також статистичний для обробки результатів апробації системи з метою оцінки часу відгуку та стабільності функціонування.

Теоретична значущість полягає в розширенні науково-методичного підходу до побудови інформаційних систем високого навантаження у сфері електронної комерції, а практична – у розробці готової архітектури й реалізації прототипу, що може бути адаптований і впроваджений в діяльності авіакомпаній та туристичних агентств різного масштабу. Структура роботи відображає логіку дослідження: після вступу викладаються огляд літератури і аналіз вимог, далі йдуть розділи, присвячені моделюванню системи, опису архітектури й технологій, реалізації користувацького інтерфейсу та серверної частини, тестуванню й обговоренню результатів. За потреби у вступі відзначено, що деталі оптимізації зовнішнього API лояльності користувачів та інтеграція з системами CRM не розглядатимуться в повному обсязі та можуть стати предметом подальших досліджень.

Отже, проведене дослідження покликане вирішити актуальні теоретичні та практичні завдання у сфері автоматизації продажу авіаквитків, зокрема шляхом створення адаптивної та масштабованої інформаційної системи, що сприятиме підвищенню ефективності бізнес-процесів авіаційного сектору й рівня сервісу для кінцевих користувачів.

РОЗДІЛ 1
ХАРАКТЕРИСТИКА ТА АНАЛІЗ ІНФОРМАЦІЙНИХ СИСТЕМ ПРОДАЖУ АВІАКВИТКІВ

1.1 Характеристика предметної галузі електронного бронювання та об’єкта дослідження інформаційної системи продажу авіаквитків

У сучасних умовах індустрія пасажирських перевезень стрімко розвивається, а сегмент онлайн-продажу авіаквитків набуває визначального значення як для авіакомпаній, так і для туристичних агентств. Предметна галузь нашого дослідження охоплює інформаційні системи й технології, призначені для автоматизації процесу продажу авіаквитків і управління відповідними бізнес-операціями. По суті, це міждисциплінарна область, що поєднує елементи економіки повітряного транспорту, інформаційну інфраструктуру, програмне забезпечення та веб-сервіси.

З економічного погляду вирішення завдань у цій галузі, зокрема онлайн-бронювання та продаж квитків, пов’язане з оптимізацією доходів авіаперевізників і туристичних агентств. Рішення, прийняті в ході аналізу попиту, ціноутворення та управління інвентарем місць на борту, безпосередньо впливають на фінансові результати: підвищення завантаженості рейсів, скорочення залишкових незатребуваних місць і зниження операційних витрат. Сучасні інформаційні системи значно спрощують процес ухвалення таких рішень, надаючи в режимі реального часу дані про залишкові місця, динамічне ціноутворення й історичні тренди продажів.

Об’єктом дослідження в межах кваліфікаційної бакалаврської роботи є процес створення й експлуатації інформаційної системи для автоматизації продажу авіаквитків через веб-інтерфейс. Він включає як розробку програмних модулів (наприклад, пошук рейсів, обробка замовлень, інтеграція з платіжними шлюзами,

формування електронного квитка), так і забезпечення безперебійної роботи серверних і клієнтських компонентів на реальному обладнанні. Саме цей процес породжує низку проблемних ситуацій – від неточної інформації про наявність місць до відмов служб авіакомпаній або нестабільності платіжної інфраструктури, що вимагає ретельного аналізу й проєктування.

Предметом дослідження виступають засоби й методи, які застосовуються для розв'язання цих проблем. Зокрема, це моделювання предметної галузі за допомогою ORM-діаграм, формалізація функціональних і нефункціональних вимог, проєктування архітектури клієнт-серверної системи, а також вибір технологій для організації бази даних, обробки запитів користувачів та інтеграції з платіжними сервісами. Саме опис цих методичних підходів дозволяє зрозуміти, яку частину об'єкта дослідження ми досліджуємо і на підставі чого розробляємо кінцевий продукт.

Важливим аспектом є перелік об'єктів предметної галузі. По-перше, це великі та малі авіакомпанії, які формують пропозицію рейсів – вони постачають дані про маршрути, доступність місць і тарифи. По-друге, туристичні агентства (як офіційні, так і онлайн-агрегатори), які організовують продаж квитків і повинні координувати інформацію від різних авіаперевізників. По-третє, платіжні провайдери та банки, що забезпечують безпеку та своєчасність фінансових операцій. Усі ці учасники створюють єдину екосистему: авіакомпанія хоче максимізувати заповнюваність літаків, агентство – надати клієнтам зрозумілий і швидкий шлях до оплати, а платіжні сервіси – гарантувати захищені транзакції.

У рамках прийняття рішень ключові особи, що беруть участь у цьому процесі, – це менеджери з продажу квитків, адміністратори IT-інфраструктури, фахівці з обслуговування клієнтів та розробники програмного забезпечення. Менеджери відповідають за встановлення правил ціноутворення, визначення знижок і акцій, прогнозування попиту; їхні рішення формують цінову політику й обсяги бронювання. Адміністратори IT-відділів контролюють стабільність серверів, налаштовують базу даних і слідкують за оновленнями, щоб мінімізувати час простою системи. Розробники реалізують функціональні можливості

програмного забезпечення, а служба підтримки клієнтів зі свого боку опрацьовує звернення пасажирів, вирішує проблеми, пов’язані з некоректним платежем чи відмовою в бронюванні. У разі групового ухвалення рішень, наприклад при встановленні сезонних тарифів, керівництво узгоджує фінансові показники з маркетингом і CFO, аби знайти баланс між доходами та задоволеністю клієнтів.

Серед факторів, які впливають на середовище прийняття рішень, слід виділити коливання попиту залежно від сезону або свят, зміну тарифів конкуренції, нормативні обмеження (наприклад, правила безпеки авіаперевезень або зміни в обміні валют), а також технічну інфраструктуру – швидкість інтернет-з’єднання, надійність серверів і масштабованість бази даних. У бойових умовах ринку, коли користувачі очікують миттєвого відгуку за пошуком квитків і онлайн-оплати, затримка навіть у декілька секунд може призвести до втрати замовлення. Крім того, періодичні оновлення політик авіакомпаній змушують операторів систем постійно коригувати бізнес-логіку, а пандемії чи екстремальні події можуть миттєво змінити структуру попиту.

Рішення, які приймаються в рамках предметної галузі, не існують у вакуумі: ціни, встановлені сьогодні, впливають на заповнюваність літаків завтра, а рішення про пропозиції акцій упродовж чергового місяця залежать від статистики попередніх періодів. Якщо спочатку керівник знизив тарифи на певний напрям, це може призвести до зростання попиту, що зумовить потребу у збільшенні кількості рейсів у розкладі. Взаємозв’язок цих рішень призводить до каскадного ефекту: невірно оцінений попит у піковий сезон (наприклад, літні відпустки) може спричинити як втрату доходу через недозавантаженість, так і додаткові витрати на екстрену оренду літаків чи працю надперсоналу.

Якість ухвалення рішень значною мірою залежить від інформаційного забезпечення: всередині організації це дані про попередні продажі, завантаженість рейсів, відгуки клієнтів, продуктивність IT-систем; ззовні – зміни в регуляторній політиці, економічні показники, прогнози погоди та актуальні тренди туристичного попиту. Для збирання цих даних використовуються різні методи: автоматизований імпорт розкладів і тарифів із систем GDS (Global Distribution System), аналітика

історичних баз даних, опитування клієнтів, а також застосування алгоритмів машинного навчання для прогнозування попиту. Обробка даних може виконуватися як за допомогою жорстко заданих алгоритмів (наприклад, встановлення фіксованих цінових діапазонів), так і з використанням евристичних методів або моделювання сценаріїв (імітаційні моделі для оцінки завантаженості в різні місяці року).

Усі згадані особливості формують специфіку ситуацій ухвалення рішень. Часто це ситуації «за умов невизначеності», коли точний попит неможливо спрогнозувати – наприклад, у разі раптових змін у геополітичній обстановці чи непередбачуваних перебоїв авіаперевезень. Водночас у стабільні періоди застосовуються більш «детерміновані» підходи, коли достатньо оперувати історичними даними для встановлення сезонних цін. У кризові моменти, такі як пандемія або фінансова криза, ситуації відкриті, бо принципи вибору тарифів і маршрутів формуються на стрімкому аналізі зовнішніх факторів.

Організаційно система продажу авіаквитків потребує взаємодії різних підрозділів. У типовій структурі (рис. 1.1) середнього туристичного агентства або IT-підрозділу авіакомпанії можна виокремити відділ розробки програмного забезпечення (де працюють бізнес-аналітики, проєкт-менеджери, розробники й тестувальники), відділ технічної підтримки (системні адміністратори та фахівці із супроводу сервісів), фінансово-обліковий підрозділ (спеціалісти з роботи з платіжними операціями) та відділ обслуговування клієнтів (менеджери з бронювання, оператори кол-центру). Кожен із цих підрозділів відповідає за певний етап життєвого циклу інформаційної системи. Бізнес-аналітики збирають і формалізують вимоги; розробники втілюють ці вимоги у коді; тестувальники перевіряють функціонал; системні адміністратори слідкують за безперебійністю роботи серверів і баз даних; оператори підтримки вчасно обробляють помилки, що виникають у процесі реальної експлуатації.

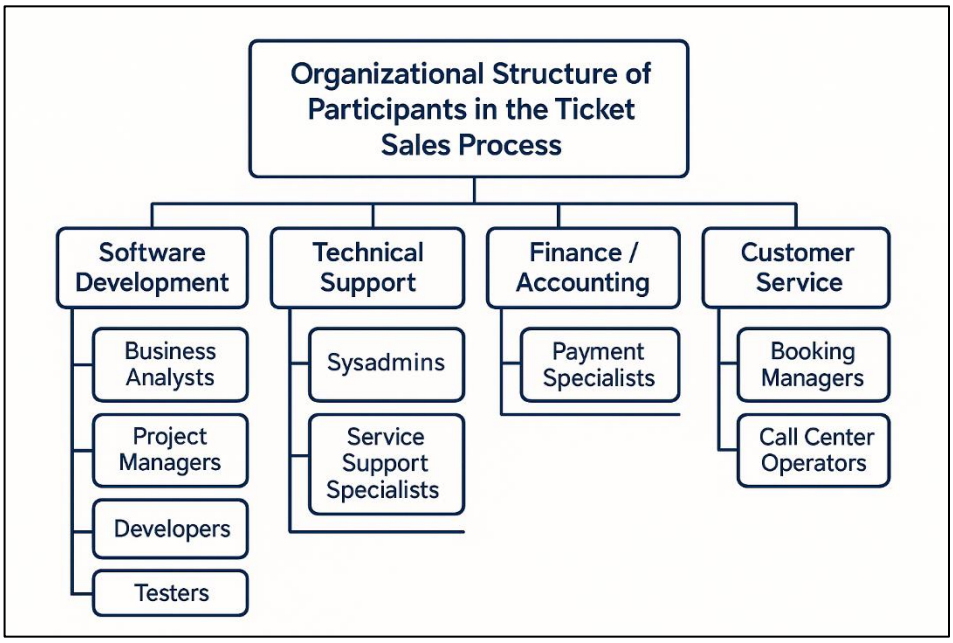


Рисунок 1.1 – Організаційна структура учасників процесу продажу авіаквитків

Джерело: сформовано автором на основі виконаного дослідження

Ієрархічно всі ці ролі підпорядковані керівнику проєктного офісу або CTO, який координує ресурси, стежить за дотриманням термінів і якості реалізації, узгоджує технічні й бізнес-цілі. При цьому для невеликих агентств або стартапів один фахівець може виконувати одразу кілька функцій (наприклад, розробник – тестувальник, бізнес-аналітик – менеджер з підтримки), що створює додаткові ризики, пов’язані з обмеженістю ресурсів і компетенцій.

Отже, предметна галузь інформаційних систем для продажу авіаквитків охоплює економічні задачі оптимізації виручки авіаперевізника, управління резервуванням і інтеграцію з платіжними сервісами, а також технічні аспекти побудови програмного середовища і баз даних. Об’єктом дослідження є сам процес розробки й експлуатації такої системи, який породжує складні ситуації ухвалення рішень у різних умовах: від детермінованих (сезонних) до кризових (пандемії, технічні збої). Взаємозв’язки між рішеннями створюють каскадний ефект, коли зміни в кінцевих умовах продажу (ціни, доступність місць, умови оплати) впливають на всі етапи життєвого циклу системи. Комплекс інформації (від внутрішніх даних про продажі до зовнішніх макроекономічних факторів) і методи

її збору та обробки (моделювання, алгоритмізація, аналітичні методики) формують якісну базу для ухвалення обґрунтованих рішень.

Отже, варто зазначити, що особливостями предметної галузі є висока динаміка змін попиту, жорсткі вимоги до безпеки фінансових операцій, необхідність швидкої реакції на зовнішні фактори та інтеграція з різноманітними сервісами. Уміння зібрати, опрацювати й своєчасно оновити інформацію, а також правильно організувати діяльність підрозділів і ролей в IT-інфраструктурі визначає успіх інформаційної системи з продажу авіаквитків.

1.2 Аналіз літературних джерел та практичного досвіду використання ІС і технологій в предметній галузі

Сфера продажу авіаквитків повністю трансформована впровадженням сучасних інформаційних систем, які забезпечують автоматизацію бронювання та глобальний обмін даними. Ще з 1960-х років авіакомпанії почали використовувати комп’ютеризовані системи резервування (CRS): наприклад, система SABRE, розроблена IBM для American Airlines, уже тоді обробляла до 7000 бронювань на годину практично без помилок, що дало авіакомпанії значну конкурентну перевагу[19]. До початку 1970-х більшість великих авіаперевізників впровадили власні CRS (нерідко на базі рішень IBM), а згодом надали турагентствам прямий доступ до цих систем через термінали [19]. Після дерегуляції авіаційної галузі у 1978 році з’явилася практика хостингу – провідні авіалінії розміщували в своїх системах дані рейсів інших перевізників за комісійну плату[19]. Це, втім, викликало питання конкуренції, і невдовзі авіакомпанії виокремили свої системи бронювання в незалежні структури: так виникли глобальні дистриб’юторські системи (GDS), коли, наприклад, United Airlines виділила систему Apollo, а American Airlines – Sabre [19]. У 1987 році європейські авіаперевізники заснували GDS Amadeus та Galileo; відтоді ринок глобального бронювання консолідувався, і

нині три провідні GDS – Sabre, Amadeus та Travelport (Galileo/Worldspan) – забезпечують основну частину дистрибуції авіаконтенту у світі. За оцінками, ще наприкінці 1990-х через системи GDS здійснювалося понад 30 млн бронювань на рік, причому ~90% цього обсягу припадало на чотири найбільші системи (Sabre, Amadeus, Galileo, Worldspan) [19]. Вітчизняні фахівці відзначали, що принципові відмінності між цими провідними системами полягають лише в повноті та оперативності наданої інформації, складі підключених постачальників послуг, зручності інтерфейсу для користувача, надійності каналів зв'язку та розмірі плати за користування сервісом [20]. GDS стали універсальною інфраструктурою туристичного ринку: через них агенції отримують доступ до сотень авіакомпаній, тисяч готелів, прокатних фірм тощо, можуть забронювати не лише авіа-, але й залізничні чи поромні квитки, готелі і навіть квитки на різноманітні заходи[20]. Таким чином, історично склалася глобальна мережа резервування, що поєднала авіаперевізників і посередників по всьому світу.

Сучасний стан розвитку систем продажу авіаквитків характеризується тотальною діджиталізацією та інтеграцією з Інтернет-технологіями. Якщо раніше доступ до GDS мали лише агенції, то тепер переважна більшість клієнтів самостійно бронюють авіаквитки онлайн – через вебсайти авіакомпаній або сторонні платформи-агрегатори [21]. Ці онлайн-системи дозволяють пасажирам у реальному часі шукати рейси, порівнювати тарифи та миттєво здійснювати бронювання з будь-яких пристроїв. Для авіакомпаній впровадження веб-бронювання дало змогу оперативно керувати тарифами й розкладом, а для посередників (турагентів, онлайн-агентств) – отримувати комісійну винагороду за кожне бронювання. GDS і досі залишаються ключовим каналом продажів для традиційних перевізників повного сервісу, забезпечуючи глобальне охоплення агентської мережі. Водночас зростає частка прямих продажів: бюджетні авіалінії (лоукостери) історично прагнули мінімізувати витрати на посередників, тому зробили ставку на власні інтернет-сайти та партнерство з метапошуковими системами. Щоб задовольнити попит на контент лоукостерів, з'являються альтернативні платформи дистрибуції. Так, міжнародний сервіс IATI, популярний

і в Україні, об’єднує в одному онлайн-інтерфейсі пропозиції регулярних рейсів із глобальних систем (Amadeus, Galileo) та напряму від понад 40 лоукост-авіакомпаній, а також чартерні перевезення [22]. Це розширює вибір для агенцій і пасажирів, комбінуючи різні джерела даних в єдиній системі. Окрім того, авіаційна галузь активно впроваджує нові стандарти дистрибуції – зокрема, ініціативу IATA New Distribution Capability (NDC), започатковану у 2012 р., яка передбачає перехід від застарілих форматів GDS до сучасних API для обміну даними про пропозиції і замовлення. Метою NDC є повернути авіакомпаніям контроль над контентом та тарифами у непрямих каналах продажу [23], дозволивши їм пропонувати агентам та агрегаторам той самий багатий контент, що й на власному сайті. У перспективі це трансформує структуру ринку, хоча повної відмови від GDS поки не сталося, і, як відзначають оглядачі, навряд чи станеться у найближчі роки – скоріше йдеться про еволюцію моделей співпраці між перевізниками, GDS та новими посередниками.

Теоретичні засади й концепції інформаційних систем у сфері авіаперевезень базуються на класичних уявленнях про менеджмент інформації та сучасних IT-стандартах. Інформаційна система продажу авіаквитків поєднує риси транзакційної системи реального часу (оперативне оформлення бронювань, платежів) та інформаційно-аналітичної системи (накопичення й аналіз даних про продажі для прийняття управлінських рішень). Вона належить до категорії систем підтримки оперативної діяльності підприємства (Operational Information System), але тісно інтегрована і з системами підтримки прийняття рішень, що використовуються на вищих рівнях управління. З точки зору загальної архітектури, сучасні системи бронювання є багаторівневими: типовим підходом є трирівнева архітектура “клієнт–сервер–база даних”[24]. На фронтальному рівні – інтерфейси для кінцевих користувачів: веб-сайти, мобільні додатки, або робочі станції касирів і агентів. У бекенд-шарі – сервери застосунків, що реалізують бізнес-логіку бронювання, а на рівні даних – центральні бази даних, які зберігають інформацію про рейси, тарифи, залишок місць, дані клієнтів та бронювань. Більшість авіакомпаній нині користуються стандартизованими комерційними платформами (так званими

Passenger Service Systems), які надаються глобальними провайдерами. Як зазначають аналітики, наразі переважна більшість перевізників у світі покладається на готові рішення, що хостяться та підтримуються глобальними дистрибуційними компаніями – прикладами є системи SabreSonic (від Sabre) чи Altéa (від Amadeus)[19]. Лише кілька найбільших авіакомпаній можуть дозволити собі розробку власних унікальних систем, решта ж адаптує існуючі рішення під свої потреби. Такий підхід скорочує витрати і забезпечує сумісність із глобальною мережею, хоча й створює залежність від зовнішніх IT-провайдерів.

Функціональні завдання інформаційної системи продажу авіаквитків охоплюють увесь цикл бронювання та обслуговування пасажирів. Ключовими функціями є: пошук рейсів, бронювання та оформлення квитка, обробка оплати, управління бронюванням та адміністрування розкладу і тарифів. На боці клієнта система забезпечує зручний інтерфейс для пошуку потрібних рейсів за заданими параметрами – маршрут, дати подорожі, кількість пасажирів тощо [23]. Після введення запиту система звертається до своєї бази даних (або через API до GDS чи інших джерел) і виводить користувачеві список доступних рейсів із цінами та наявними місцямиі. Обравши рейс, користувач заповнює необхідні персональні дані пасажирів (ім’я, паспортні дані, контактна інформація) – система валідує ці дані та ініціює процес бронювання місця на рейсі. На цьому етапі відбувається взаємодія з модулем управління інвентарем: резервується місце в літаку, і кількість доступних місць оновлюється у базі в режимі реального часу, щоб запобігти подвійним продажам. Далі запускається процес платежу: система інтегрована з платіжними шлюзами (наприклад, Visa/Mastercard API, PayPal, Stripe тощо) для авторизації оплати банківською карткою або іншими способами. Транзакція захищається сучасними протоколами безпеки (SSL-шифрування, 3D Secure), щоб гарантувати конфіденційність фінансових даних клієнта. Після успішної оплати система автоматично формує електронний квиток (e-ticket) та підтвердження бронювання (маршрут-квитанцію) і надсилає їх користувачеві – зазвичай електронною поштою або SMS-повідомленням. Одночасно деталі бронювання (PNR – Passenger Name Record) зберігаються у базі даних, доступні для подальшого

обслуговування. Надалі пасажир може керувати своїм бронюванням через систему: переглядати деталі, вносити зміни або скасовувати за необхідності. Система підтримує функції управління бронюваннями: наприклад, анулювання квитка з відповідним поверненням коштів на карту клієнта, переоформлення на інший рейс, вибір місця чи замовлення додаткових послуг (багаж, харчування тощо) – у межах правил тарифу. Важливо, що будь-які зміни негайно відбиваються на залишку місць і в тарифних калькуляціях, забезпечуючи узгодженість даних.

Інформаційна система також має адміністративний інтерфейс для співробітників авіакомпанії. Внутрішні користувачі – адміністратори розкладу та тарифів – через окремі модулі можуть оновлювати розклад рейсів, додавати нові рейси або змінювати параметри існуючих (тип літака, час вильоту тощо), встановлювати та коригувати тарифи, керувати доступністю місць для продажу. Такі дії миттєво відображаються в системі для фронт-енд користувачів, тобто пасажирів і агентів. Крім того, система надає інструменти для служби підтримки та роботи з клієнтами – співробітники кол-центрів чи каси можуть знаходити і переглядати бронювання, вручну створювати нові бронювання на прохання клієнтів (наприклад, телефонні продажі), здійснювати примусове розблокування місць, оформлювати апгрейди класу обслуговування тощо. Окремо реалізовано функції для агентств та корпоративних клієнтів: великі системи, як правило, дозволяють турфірмам підключатися як користувачам через спеціальні термінальні програми або веб-портали. У такому разі агент може від імені клієнта виконувати всі операції бронювання і виписки квитка, отримуючи агентську комісію. GDS традиційно надають турагентам інтерфейси у вигляді терміналів з уніфікованими кодовими командами, але сьогодні також доступні і графічні веб-інтерфейси (напр. Sabre Red 360, Amadeus Selling Platform Connect), що спрощують роботу агентів, об’єднуючи контент різних постачальників в одному вікні [19]. В цілому коло користувачів систем бронювання охоплює: індивідуальних пасажирів (що здійснюють покупки онлайн самостійно), турагентів і корпоративних менеджерів з подорожей (що працюють через професійні інтерфейси GDS/OTA), оперативний персонал авіакомпаній (резервування, підтримка, управління розкладом), а також

керівництво авіапідприємств, яке отримує агреговану інформацію для прийняття рішень. Зокрема, глобальні системи дистрибуції орієнтовані на професійних посередників – онлайн-агенції, корпоративні туристичні компанії, класичні туристичні агентства – і надають їм доступ до величезного обсягу пропозицій від сотень авіаліній, готельних мереж, прокатних фірм тощо [19]. Кінцеві ж мандрівники взаємодіють із GDS опосередковано, через веб-сайти агентств або авіакомпаній. Для авіакомпанії така система дає змогу ефективно обслуговувати запити на операційному рівні (оформлення перевезень), підтримувати тактичний рівень управління (наприклад, відділ продажів може на основі даних системи коригувати цінову політику або маркетингові акції), а також інформувати стратегічний рівень (топ-менеджмент отримує звіти й аналітику для довгострокових рішень). В літературі наголошується, що такі системи генерують величезні масиви даних про клієнтів і бронювання, які можуть аналізуватися для виявлення тенденцій ринку та підтримки обґрунтованих рішень щодо тарифів, маршрутної мережі та маркетингової стратегії авіакомпанії. Таким чином, інформаційна система бронювання виступає нервовим центром, що пов’язує оперативні процеси та аналітичні задачі управління.

Технічне, програмне й інформаційне забезпечення систем продажу авіаквитків сьогодні ґрунтується на сучасних IT-рішеннях, що забезпечують масштабованість, високу доступність і безпеку. На фізичному рівні використовуються потужні серверні комплекси (нерідко розподілені по декількох дата-центрах у світі для резервування), здатні обробляти тисячі транзакцій щосекунди. Історично CRS/GDS працювали на мейнфреймах і спеціалізованих мережах, однак нині галузь поступово мігрує до хмарних інфраструктур та відкритих стандартів. У програмній архітектурі впроваджуються принципи мікросервісів, що підвищує гнучкість оновлень. Як зазначено в дослідницькому проєкті (2025) індійських інженерів, сучасна система резервування може бути реалізована засобами Java (Spring Boot) або Node.js на серверному боці, із використанням СУБД MySQL/PostgreSQL (для транзакційних даних) чи NoSQL MongoDB (для гнучкого зберігання даних). На фронтенді широко застосовуються

web-фреймворки на зразок React.js/Angular для створення динамічних інтерфейсів [23]. Для забезпечення високої доступності й продуктивності впроваджується розгортання у хмарних середовищах (AWS, Azure, GCP) з можливістю автоматичного масштабування під навантаженням. Особлива увага приділяється захисту інформації: обов'язковим є шифрування трафіку (SSL/TLS) між клієнтом і сервером, використання хешування паролів та багаторівневої автентифікації для користувачів, розмежування доступу до адміністративних функцій (RBAC). Такі заходи покликані запобігти витоку персональних даних та фінансової інформації, оскільки системи бронювання є привабливою ціллю для кібератак. Інформаційне забезпечення систем включає великі централізовані бази даних розкладів і тарифів. Дані про рейси (номери, маршрути, час вильоту/прильоту, типи літаків), тарифні правила, залишок місць у кожному класі бронювання, профілі учасників програм лояльності, історію бронювань – усе це зберігається і оперативно оновлюється. Система також інтегрується з зовнішніми джерелами інформації: глобальні системи отримують дані від авіакомпаній-постачальників (через постійне завантаження розкладів та відкриття продажу місць), синхронізуються з міжнародними стандартами (наприклад, класи бронювання IATA) та можуть обмінюватися даними з іншими інформаційними системами (системами управління доходами, аеропортовими системами контролю відправлення тощо). Для підтримки безперервної роботи передбачено дублювання критичних компонентів, резервне копіювання баз даних, системи безперебійного живлення та інші організаційно-технічні засоби – адже простої систем бронювання негайно призводять до фінансових втрат і репутаційних ризиків для авіакомпаній.

Економіко-математичні моделі займають особливе місце в інформаційних системах авіабронювання, оскільки одна з головних цілей – максимізація доходів від продажу місць при оптимальному завантаженні рейсів. У цьому контексті було розроблено поняття "Yield Management" (управління доходами) – стратегічного підходу, який динамічно змінює ціни на авіаквитки залежно від попиту, часу до вильоту, сегменту ринку та інших факторів [25]. За визначенням, yield management – це система методів і алгоритмів, що дозволяє авіакомпаніям прогнозувати попит

та гнучко регулювати тарифи з метою максимізації виручки з кожного рейсу. Здійснюється аналіз великих обсягів даних: сезонні коливання, історичні тенденції заповнюваності, ціни конкурентів, поведінка споживачів при бронюванні – всі ці фактори враховуються моделями прогнозування попиту. На основі прогнозів система розподіляє місця за класами бронювання (та відповідними тарифами): вводяться диференційовані цінові рівні (бронювання заздалегідь – дешевше, гнучкі тарифи – дорожче, останні місця – за преміальною ціною тощо). Мета – продати правильну кількість місць за найвищою можливого ціною кожному сегменту споживачів, при цьому уникнути вильоту з порожніми кріслами. Класичним завданням є модель оптимізації завантаження рейсу: якщо попит високий, система підвищує ціни або закриває продаж дешевих класів; якщо ж попит низький, застосовуються акційні тарифи для стимулювання продажів. Таким чином, використовуючи математичні моделі (динамічне програмування, методи прогнозування та статистичний аналіз), сучасні системи можуть автоматично керувати доступністю тарифних класів у залежності від поточного заповнення рейсу і часу, що залишився до вильоту. Ще один економіко-математичний аспект – овербукінг (overbooking), тобто свідомий продаж квитків понад номінальну місткість літака. Це робиться з розрахунком на певний відсоток пасажирів, які не з’являться на посадку. Як свідчить практика, близько 10–15% заброньованих пасажирів у середньому не приходять на рейс (через зміни планів, запізнення на стиковки тощо). Щоб мінімізувати кількість порожніх місць від таких «no-show», авіакомпанії застосовують моделі оптимального овербукінгу: розраховується, скільки квитків можна продати понад 100% місткості, щоб із високою ймовірністю літак все одно вилетів повністю заповненим, але ризик надлишкових пасажирів залишався прийнятним. Моделі овербукінгу балансують дві протилежні величини – втрачений дохід від порожнього крісла проти витрат і негативних наслідків від відмови в посадці пасажиру [26]. Сучасні алгоритми (в тому числі на основі машинного навчання) аналізують історичні дані про частки no-show на різних напрямках, ймовірності продажу останніх місць за високими тарифами, середні витрати на компенсацію відмовленим пасажирам – і на основі цього визначають

оптимальний рівень овербукінгу для кожного рейсу. В результаті застосування таких економіко-математичних методів система бронювання вже не просто реєструє факти продажу, а й динамічно керує пропозицією місць і цін. Варто зазначити, що алгоритми управління доходами в авіації досягли високого рівня автоматизації: ще наприкінці 1980-х American Airlines впровадила систему DINAMO, яка автоматично оптимізувала розподіл місць між тарифами. Сьогодні ж практично кожна велика авіакомпанія використовує подібні Revenue Management System, інтегровані зі своєю системою бронювання. Це дозволяє в режимі реального часу реагувати на зміну попиту – наприклад, миттєво запускати спецпропозицію, якщо бронювання на рейс відстають від планових показників, або закривати продаж дешевих квитків, якщо попит перевищує пропозицію. Отже, економіко-математичні моделі (прогнозування, оптимізація, стохастичний аналіз) є невід'ємною частиною інформаційних систем продажу авіаквитків, забезпечуючи науково обґрунтоване управління ціноутворенням і заповнюваністю.

Інформаційно-аналітичні можливості таких систем дозволяють авіапідприємствам отримувати стратегічні інсайти з операційних даних. Кожне бронювання, кожна транзакція фіксується у базі – агреговані разом, ці дані утворюють багате джерело відомостей про ринок: поведінку клієнтів, популярність напрямків, цінову чутливість пасажирів, ефективність каналів продажу тощо. Аналітичні модулі можуть генерувати звіти про продажі за період, по маршрутах, по агентських мережах; обчислювати показники завантаження рейсів (load factor), середній дохід на пасажира (yield) та інші ключові метрики. Як зазначено в одному з досліджень, сучасна система бронювання здатна накопичувати великі обсяги даних, які можуть бути проаналізовані для виявлення уподобань клієнтів, тенденцій ринку і на цій основі ухвалення обґрунтованих рішень щодо ціноутворення, маршрутної політики та маркетингу перевізника [22]. Наприклад, аналіз історичних даних може показати, що на певному напрямку стабільно виникає надлишковий попит у літній сезон – це дає підстави розглянути введення додаткових рейсів або збільшення місткості (що є вже рішенням стратегічного рівня, прийнятому на основі інформації системи бронювання). Інформаційні

потоки від системи бронювання таким чином живлять різні рівні управління: оперативний (щоденні звіти про виконані рейси, список зареєстрованих пасажирів тощо), тактичний (щомісячні/квартальні звіти про обсяги продажів, прибутковість окремих маршрутів, ефективність акцій) і стратегічний (річні тенденції, прогнози попиту, структура ринку). Для реалізації таких аналітичних завдань часто створюються окремі сховища даних (Data Warehouse), куди інформація з систем бронювання завантажується і де її обробляють системи бізнес-аналізу (BI). Великі авіакомпанії використовують програмні платформи на кшталт SAS, Oracle BI, Microsoft Power BI та ін. для візуалізації та аналізу даних продажів. На основі цих даних ухвалюються рішення про відкриття або закриття рейсів, перегляд тарифної політики, сегментацію ринку і таргетування рекламних кампаній. Отже, інформаційно-аналітична складова є критично важливою: як підкреслюють автори одного з галузевих звітів, система продажу авіаквитків надає авіакомпаніям інструменти не лише для заробляння доходів, але й для підвищення задоволеності клієнтів, покращення операційної ефективності та роботи з даними для підтримки рішень.

Технологічно-організаційні аспекти впровадження і експлуатації цих інформаційних систем теж заслуговують уваги. Історично перехід від паперових квитків та телефонних бронювань до електронних систем вимагав значних змін у процесах авіакомпаній та агентств. Персонал довелося навчати роботі з комп’ютерними терміналами, впроваджувати нові посадові ролі (агент з резервування, адміністратор бази даних розкладу тощо), змінювати стандарти обслуговування клієнтів. Сьогодні організаційний аспект проявляється у необхідності постійної підтримки та оновлення системи: великі авіакомпанії мають цілі IT-відділи або навіть дочірні IT-компанії, що займаються адмініструванням систем бронювання, інтеграцією з новими сервісами (наприклад, мобільними застосунками для пасажирів, системами онлайн-реєстрації тощо), забезпеченням кібербезпеки і відповідності регуляторним вимогам (таким як GDPR щодо захисту персональних даних). Важливим є дотримання міжнародних стандартів: усі інформаційні системи продажу авіаквитків повинні працювати за стандартами,

встановленими IATA та іншими організаціями (формати авіаквитків, коди аеропортів і авіакомпаній, протоколи обміну даними типу EDIFACT або нові XML/JSON стандарти NDC, тощо). Це гарантує інтероперабельність: квиток, виписаний через одну систему, визнається іншими (особливо актуально для інтерлайн-угод, коли перевезення виконує декілька різних авіакомпаній). На рівні організації бізнесу авіакомпанії повинні вибудувати відносини з посередниками і провайдерами: укласти угоди з GDS про підключення (що часто вимагає сплати чималих fees за кожний сегмент бронювання) [19], співпрацювати з OTA, які продають їхні квитки онлайн, стимулювати прямий канал (наприклад, надаючи знижки чи бонуси при бронюванні на власному сайті). Багато перевізників, прагнучи заощадити на комісіях GDS, розробляють програми заохочення агентств за використання прямих API на базі NDC або вводять додаткові збори з білетів, оформлених через GDS. Таким чином, управління каналами дистрибуції – це важливий організаційний момент, що безпосередньо пов'язаний із інформаційними системами. Ще один аспект – стійкість до збоїв та кризових ситуацій. Події останніх років, зокрема пандемія COVID-19, спричинили безпрецедентне падіння попиту на авіаперевезення і масові скасування рейсів. Авіакомпаніям довелося терміново коригувати процеси резервування – вводити гнучкі правила зміни квитків, ваучери замість повернення коштів, адаптувати системи до нового потоку операцій (наприклад, обробки повернень). Виявилося, що гнучкість інформаційних систем є критичною: ті компанії, які мали сучаснішу IT-інфраструктуру, змогли швидше переналаштувати бізнес-процеси під нові умови. Таким чином, організаційний успіх у використанні систем бронювання залежить від інвестицій у технології, навчання персоналу та вміння швидко реагувати на зовнішні виклики.

Критичний аналіз джерел та сучасних тенденцій. Огляд вітчизняної та зарубіжної літератури показує, що питання автоматизації продажу авіаквитків досліджене досить широко, проте залишається низка проблем і напрямів розвитку. Іноземні джерела детально висвітлюють історичну еволюцію GDS та їхню адаптацію до викликів цифрової доби (зокрема, впровадження NDC, персоналізованих пропозицій). Вітчизняні автори, зі свого боку, зосереджуються

на застосуванні цих глобальних технологій у українському ринку та підкреслюють залежність локальних авіаперевізників від міжнародних систем і провайдерів. Попри це, спільним є розуміння, що сучасна система продажу авіаквитків – це нервова система авіаційного бізнесу, без якої неможливо забезпечити ні ефективність операцій, ні належний рівень сервісу для клієнтів. Підсумовуючи, дослідники відзначають вирішальну роль цих інформаційних систем у модернізації галузі: вони дозволили авіакомпаніям підвищити точність і швидкість бронювання, надавати клієнтам актуальну інформацію та зручні сервіси, а також глобально розширити ринок збуту послуг[19] [21]. За словами одного з джерел, впровадження електронних систем резервування фактично «революціонізувало спосіб, у який люди подорожують, зробивши процес бронювання простішим і доступнішим, ніж будь-коли раніше», паралельно створивши нові робочі місця у туристичній індустрії (агенти з бронювання, консультанти з подорожей, IT-фахівці) [2]. Водночас література наголошує і на проблемних аспектах, які до кінця не вирішені і потребують подальших досліджень та впроваджень. По-перше, стоїть питання інтеграції нових стандартів дистрибуції (таких як NDC) з існуючою екосистемою GDS: як забезпечити перехід до більш гнучкого обміну даними, не втративши при цьому охоплення агентської мережі та не спричинивши технічних збоїв. По-друге, залишається актуальним удосконалення моделей динамічного ціноутворення та овербукінгу з метою підвищення прибутковості – але без шкоди для лояльності клієнтів та рівня обслуговування. Незважаючи на автоматизацію, практика показує, що надмірні овербукінги можуть призводити до скандалів та іміджевих втрат для авіакомпаній [24], а нестабільні або не прозорі цінові стратегії викликають невдоволення споживачів. Отже, потрібні дослідження, як збалансувати бізнес-оптимізацію і споживчий досвід у цих системах. По-третє, величезні масиви даних про продажі досі використовуються авіакомпаніями не в повній мірі; постає завдання розробки ще більш потужних інформаційно-аналітичних інструментів (Big Data, штучний інтелект) для підтримки стратегічних рішень – наприклад, точнішого прогнозування попиту, індивідуалізації пропозицій для різних сегментів клієнтів, оптимізації маршрутних мереж. Таким чином, сучасний стан вирішення

поставлених проблем можна охарактеризувати як прогресивний, але такий, що триває: основні концепції створення і функціонування інформаційних систем продажу авіаквитків успішно реалізовані на практиці, проте галузь продовжує еволюціонувати, реагуючи на технологічні інновації та ринкові виклики. Ці проблемні питання окреслюють напрями подальших досліджень і вдосконалення систем: від технологічної інтеграції нових стандартів до підвищення ефективності та гнучкості моделей, що лежать в основі інформаційних систем авіабронювання. Відповідно, літературний огляд демонструє як досягнення, так і невирішені завдання, формуючи цілісне уявлення про сучасний рівень і перспективи розвитку інформаційних систем у сфері продажу авіаквитків.

РОЗДІЛ 2
РОЗРОБКА ВИМОГ І МОДЕЛЮВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Аналіз і специфікація вимог до інформаційної системи

Аналіз та специфікація вимог є фундаментом створення будь-якої інформаційної системи, оскільки саме на цьому етапі формується чітке розуміння того, якими функціями має володіти майбутня система, та яким вимогам вона повинна відповідати. Для інформаційної системи продажу авіаквитків, розроблюваної у даній кваліфікаційні бакалаврській роботі, правильна постановка вимог є вирішальною, оскільки помилки чи неточності можуть негативно вплинути на подальші етапи розробки, тестування, впровадження та експлуатації системи.

Першим етапом визначення вимог до системи стало формування бізнес-вимог, які відображають стратегічні цілі та потреби організації. Головною метою створення інформаційної системи є автоматизація процесів пошуку, бронювання, оплати та управління продажами авіаквитків через онлайн-інтерфейс. Це дозволить авіакомпанії або турагентству підвищити ефективність обслуговування клієнтів, зменшити витрати часу та ресурсів на обробку замовлень, а також забезпечити точну й оперативну інформацію про доступність місць та ціни на авіаквитки. Система повинна забезпечити безперебійний доступ користувачів до актуальної інформації у режимі реального часу, інтеграцію з глобальними дистрибутивними системами (GDS), платіжними шлюзами і внутрішніми корпоративними базами даних. Одночасно необхідно враховувати такі зовнішні обмеження, як відповідність системи міжнародним стандартам IATA та вимогам безпеки персональних і платіжних даних (GDPR, PCI-DSS).

На другому етапі були визначені користувацькі вимоги, які безпосередньо стосуються роботи кінцевих споживачів (пасажирів), а також адміністраторів системи. Аналіз проводився шляхом опитувань, особистих спостережень та

вивчення аналогічних систем на ринку. Серед основних вимог користувачів є зручний і зрозумілий інтерфейс, швидкість завантаження інформації, можливість фільтрації й сортування результатів пошуку за різними критеріями (ціна, час вильоту, тривалість польоту тощо), простий процес бронювання та надійний механізм онлайн-оплати. Адміністратори, у свою чергу, висувають вимоги до функцій управління тарифами, бронюваннями, аналітичної звітності, та адміністрування інформації про рейси й пасажирів.

Наступним кроком стала систематизація та формалізація усіх виявлених вимог. Усі вимоги були розділені на функціональні й нефункціональні та представлені у вигляді діаграми пакетів (рис 2.1). Це дало змогу структуровано відобразити ієрархічну залежність та зв'язки між вимогами, а також простежити походження кожної вимоги.

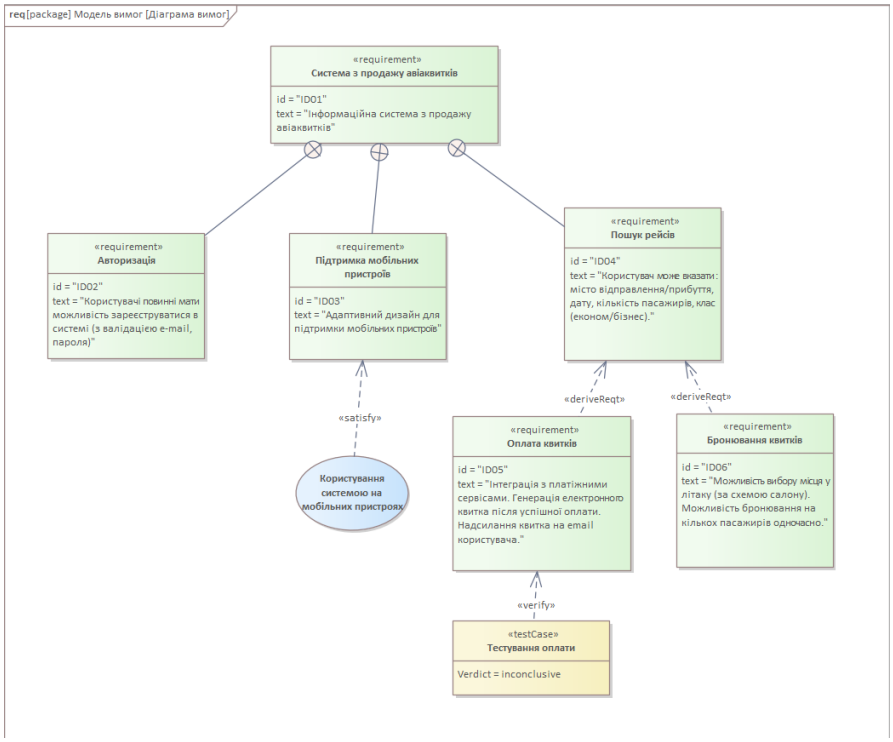


Рисунок 2.1 – Ієрархія вимог до системи (SysML Requirement Diagram)

Джерело: сформовано автором на основі виконаного дослідження в середовищі EA

Функціональні вимоги охоплюють безпосередні можливості системи: пошук рейсів за заданими параметрами, реєстрація користувачів, оформлення бронювання, процесинг платежів, управління бронюваннями, адміністрування

рейсів та тарифів. Наприклад, вимога щодо функції бронювання формулюється як: «Система повинна забезпечувати бронювання авіаквитків з миттєвим підтвердженням і відображенням актуальної інформації про наявність місць». Перевіркою виконання цієї вимоги є отримання підтвердження бронювання менш ніж за 3 секунди в 99% випадків.

Нефункціональні вимоги описують якості системи, які важливі для її успішного функціонування. Серед них визначено продуктивність, безпеку, надійність, масштабованість, зручність використання. Наприклад, вимога до продуктивності може бути сформульована так: «Система повинна обробляти щонайменше 500 одночасних запитів на пошук і бронювання авіаквитків без зниження швидкості відгуку». Критерієм успішності є тестування продуктивності за допомогою спеціальних інструментів з фіксацією результатів у технічній документації.

Після формалізації вимог був складений детальний перелік у вигляді проєктної специфікації, що охоплює кожну вимогу окремо (табл. 1.1). У специфікації, яка наведена, для кожної вимоги зазначено ідентифікатор, стислу та повну назву, тип вимоги (функціональна або нефункціональна), джерело вимоги, статус реалізації, рівень складності, пріоритетність, ризики та примітки, що пояснюють суть вимоги більш детально.

Таблиця 1.1 – Специфікація вимог

ID	**Назва**	**Тип**	**Джерело**	Статус	Пріорите	Примітки
FR-01	Пошук рейсів	Функц.	Користувачі	Уточнюється	Високий	Пошук до 3 с за попитом
FR-02	Бронювання місць	Функц.	Бізнес	Затверджено	Високий	Резервування з оновленням PNR
FR-03	Онлайн-оплата	Функц.	PCI-DSS	У ході	Високий	SSL, 3D Secure
NFR-01	Продуктивність	Нефункц.	Бізнес	Уточнюється	Високий	500 запитів/3 с
NFR-02	Захист даних	Нефункц.	GDPR/PCI	Затверджено	Високий	Шифрування, аудит
NFR-03	Масштабованість	Нефункц.	Архітектура	В плані	Середній	Авто-масштабування в хмарі

Джерело: сформовано автором на основі виконаного дослідження

Щоб забезпечити зручність управління вимогами та контролю їх виконання, у процесі роботи застосовувалися CASE-засоби. Це дозволило візуалізувати зв'язки вимог з цілями системи (рис 2.2), обмеженнями, джерелами, а також вести моніторинг стану виконання вимог у процесі розробки.

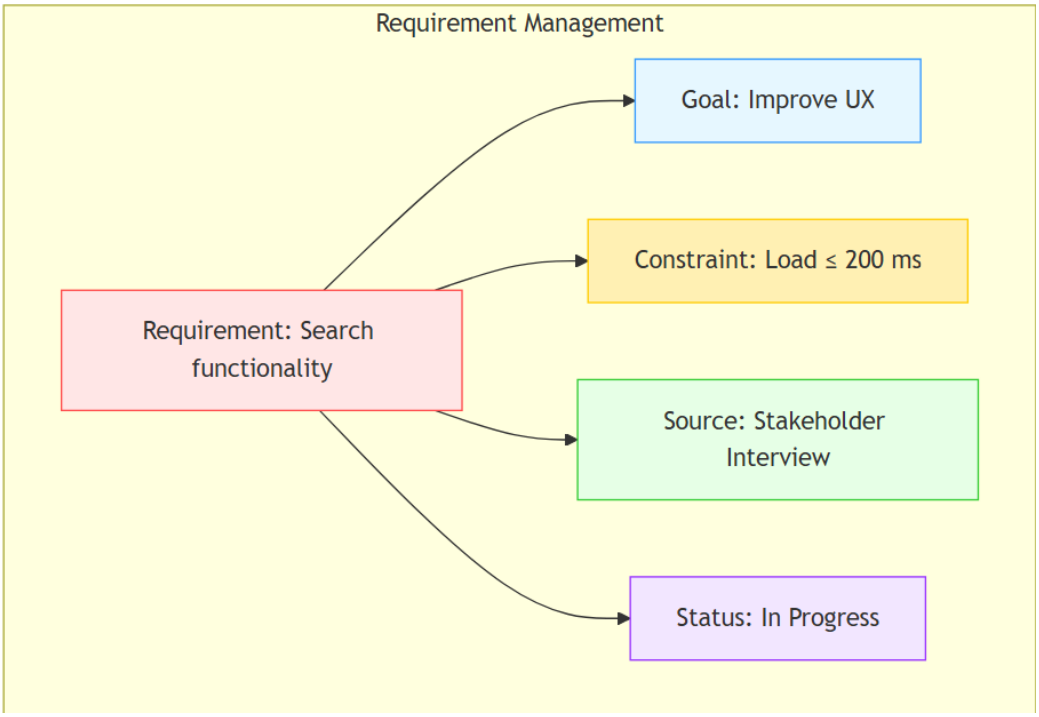


Рисунок 2.2 – Діаграма управління вимогами інформаційної системи

Джерело: сформовано автором на основі виконаного дослідження

Важливим аспектом стала перевірка вимог на конфліктність, надлишковість і послідовність. Це дало змогу виключити суперечності, зменшити ризики помилок на наступних етапах розробки системи та забезпечити зрозумілість і однозначність поставлених завдань для всіх учасників процесу.

Таким чином, сформульовані і специфіковані вимоги є завершеними, зрозумілими і реалістичними для реалізації, відповідають технічним і юридичним стандартам та мають чіткі критерії для оцінки їх виконання. Це створює міцну основу для розробки ефективної, надійної і функціональної інформаційної системи

продажу авіаквитків, здатної повністю задовольнити потреби бізнесу і кінцевих користувачів.

2.2 Постановка та алгоритм розв’язнання задачі

2.2.1 Постановка задачі

У межах кваліфікаційної бакалаврської роботи інформаційної системи продажу авіаквитків задача автоматизації визначається як комплекс взаємопов’язаних компонентів, спрямованих на забезпечення оперативного бронювання, обробки платежів, формування квитків та адміністрування рейсів через веб-інтерфейс. Характеристика задачі полягає в тому, щоб замість ручного управління великими обсягами даних і транзакцій нині автоматизувати ці процеси за допомогою ЕОМ, що забезпечить економію часу, мінімізацію людських помилок і підвищення ефективності бізнес-операцій. Об’єктами управління є рейси, тарифи, бронювання, платіжні транзакції та дані користувачів. Завдання рішення: у реальному часі оновлювати інформацію про доступні місця, резервувати їх, проводити оплату, оновлювати базу даних та надавати аналітичні звіти.

Вихідна інформація використовується для генерації звітів про підтверджені бронювання, онлайнові транзакції, зайнятість рейсів, а також для формування електронних квитків і повідомлень пасажирам. Поточна періодичність генерування інформації – у режимі реального часу з відгуком не більш як 3 секунди; аналітичні звіти формуються щоденно або за запитом. Завдання автоматично припиняється у разі виявлення критичних помилок у процесі або відсутності зв’язку із зовнішніми сервісами – тоді включається відновлювальний режим з повідомленням адміністратору. Ця задача тісно пов’язана з іншими підсистемами, зокрема моніторингом стану сервера, системою логування й аналітики.

У процесі виконання роль персоналу полягає в налаштуванні тарифів, реєстрації адміністратора, реагуванні на повідомлення про збої; ЕОМ-набори виконують автоматизовані операції пошуку, резервування, валідації платежів та формування електронних квитків. У надзвичайних ситуаціях персонал бере на себе контроль і виправлення, зупиняючи або перезапускаючи процеси.

Інформаційну модель задачі відображено на UML-діаграмі (рис 2.3), що демонструє потоки даних між модулями «Пошук», «Бронювання», «Платіжний шлюз», «Адміністрування» й «Аналітичний модуль».

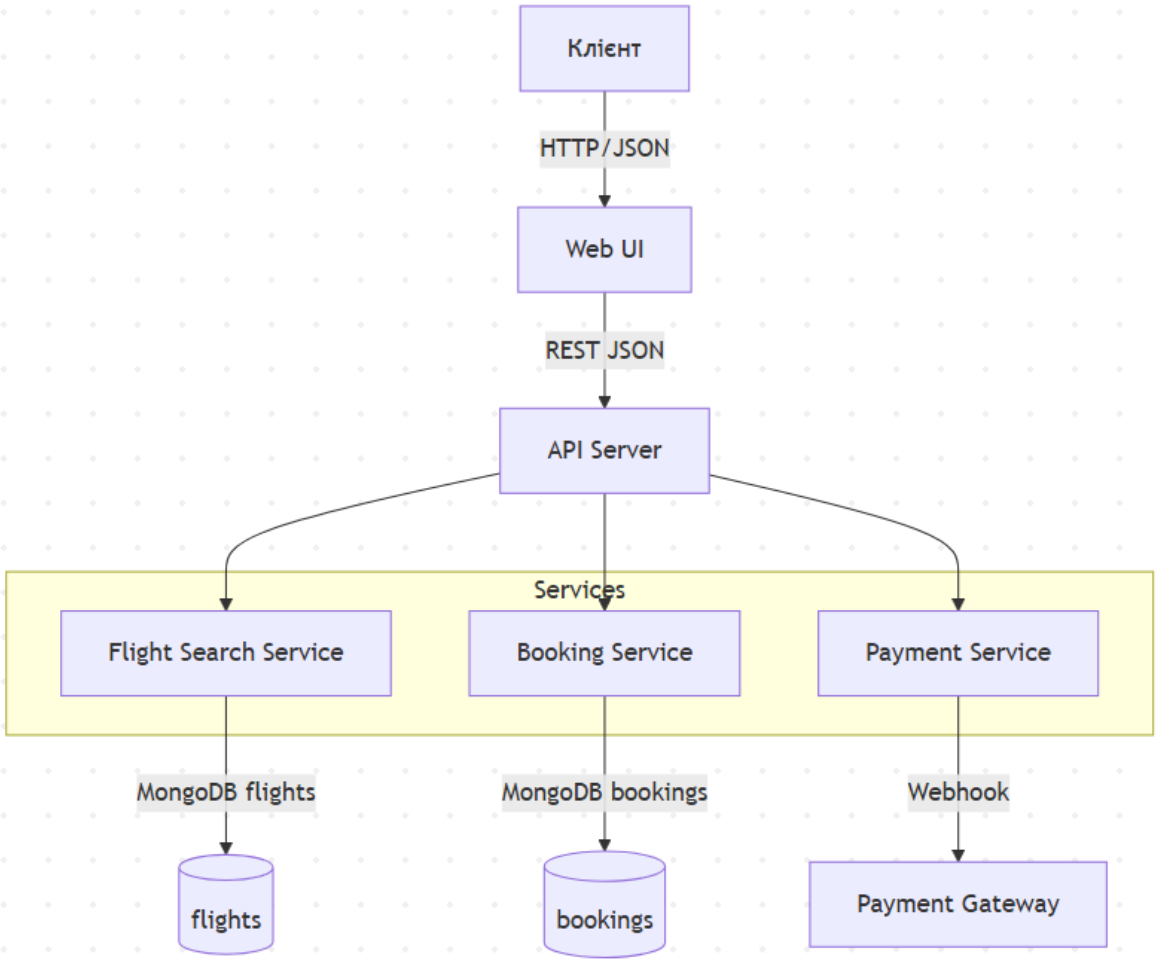


Рисунок 2.3 – Діаграма інформаційної моделі задачі

Джерело: сформовано автором на основі виконаного дослідження

У таблиці 2.1 «Вихідна інформація» описано повідомлення, дані яких формує система. Наведемо стислий перелік у табличному форматі:

Таблиця 2.1 – Перелік вихідних повідомлень

№	Назва вихідного повідомлення	Ідентифікатор	Форма подання і вимоги до неї	Періодичність видачі	Термін видачі і допустимі час затрим	Користувачі інформації
1	Підтвердження бронювання	OUT-BOOK	PDF, e-mail, формат A5, PDF/A	Одразу після бронювання	≤ 3 сек після транзакції	Пасажир, агентство
2	Електронний квиток (e-ticket)	OUT-ETICKET	PDF/A + XML-структура	Одразу після оплати	≤ 3 сек	Пасажир, авіакомпанія
3	Звіт по щоденним бронюванням	OUT-DAILY-RPT	HTML-таблиця, csv-export	Щодня о 00:05	≤ 10 хв після опівночі	Адміністратор, відділ аналітики
4	Повідомлення про помилку	OUT-ERROR	JSON з кодом і описом помилки	У момент виникнення	≤ 1 с	Адміністратор, підтримка

Джерело: сформовано автором на основі виконаного дослідження

Ключові поля: для e-ticket – PNR, ім’я пасажира, маршрут, статус (точність 100%, миттєва генерація, зберігання в БД); для підтвердження – час, маршрут, кількість місць (точність до 1 сек).

У таблиції 2.2 «Вхідна інформація» наведено джерела даних, необхідні для роботи системи:

Таблиця 2.2 – Перелік вхідних повідомлень

№	Назва вхідного повідомлення	Ідентифікатор	Форма подання і вимоги до неї	Періодичність видачі	Термін видачі і допустимі час затрим	Користувачі інформації
1	Запит на пошук рейсів	IN-SEARCH	JSON з параметрами маршруту, дати	У реальному часі	≤ 3 сек після транзакції	Фронтенд (веб/мобайл)
2	Запит на бронювання	IN-BOOK-REQ	JSON з PNR, даними пасажирів	На запит користувача	≤ 3 сек	Фронтенд / агентство
3	Вхідна транзакція оплати	IN-PAYMENT	JSON від платіжного шлюзу, статус	У реальному часі	≤ 10 хв після опівночі	Платіжний провайдер
4	Адміністративні зміни рейсу	IN-ADMIN-RWY	JSON/XML зі змінами розкладу/тарифів	За потребою	≤ 1 с	Адмін-панель системи

Джерело: сформовано автором на основі виконаного дослідження

Основні потоки вхідної інформації включають:

- Дані про рейси (номер, маршрут, час, тарифи) – постійно оновлювані через API авіакомпаній і GDS.
- Запити користувача – форма пошуку з параметрами маршруту, дат, кількості пасажирів.
- Платіжна транзакція – через платіжний шлюз із валідацією і статусом успішної операції.
- Адміністративне оновлення – дані про зміни рейсів, тарифів від модулю «Адміністрування».

Для кожної структурної одиниці вхідного повідомлення вказано точність (тип даних, обов’язковість), джерела ідентифікації. Наприклад: номер рейсу – рядок, максимум 10 символів; запит формується користувачем із фронтенду (ідентифікатор користувача + час).

2.2.2 Алгоритм розв’язання задачі

При розробці алгоритму розв’язання задачі автоматизованої системи продажу авіаквитків ключовими складовими визначено: використовувана інформація, результати розв’язання, математичний опис, а також алгоритм виконання на ЕОМ. Вкладені модулі автоматизують логіку пошуку, бронювання і оплати, формують електронні квитки й щоденні звіти.

Насамперед, використовувана інформація – це вихідні масиви(табл 2.3), сформовані з вхідних повідомлень або інших алгоритмів, необхідні для виконання розрахунків. Вони містять дані про запити користувачів, інформацію про рейси та доступність тарифів, а також історичні дані попиту. Ці масиви використовуються

для забезпечення коректної обробки запиту на бронювання та подальшої генерації відповіді.

Таблиця 2.3 – Перелік масивів використовуваної інформації

Масив	Ідентифікатор	Максимальна кількість записів
Масив запитів на пошук рейсів	IN_SEARCH_REQUESTS	10 000
Масив даних про рейси та тарифи	FLIGHT_INVENTORY	5 000
Масив даних користувачів (сесії)	USER_SESSIONS	20 000
Масив історичних бронювань	HISTORICAL_BOOKINGS	50 000
Масив вхідних транзакцій платежів	PAYMENT_TRANSACTIONS	15 000

Джерело: сформовано автором на основі виконаного дослідження

Далі описуються результати розв’язання, які формуються у вигляді масивів, призначених для передачі у вихідні повідомлення(табл 2.4) До них належать підтвердження бронювання, електронні квитки, аналітичні дані щоденних та щомісячних звітів. Ці результати забезпечують оперативний та адміністративний контроль роботи системи.

Таблиця 2.4 – Перелік масивів результатної інформації

Масив	Ідентифікатор	Максимальна кількість записів
Масив підтверджень бронювань	OUT_BOOK_CONFIRM	10 000
Масив електронних квитків (e-ticket)	OUT_ETICKETS	10 000
Масив щоденних звітів	OUT_DAILY_REPORTS	365
Масив повідомлень про помилки	OUT_ERROR_LOGS	50 000
Масив оновлень інвентарю рейсів	OUT_INVENTORY_UPDS	5 000

Джерело: сформовано автором на основі виконаного дослідження

У математичному описі процесу бронювання ключовим показником є загальний час реакції системи $T_{response}$, що визначається сумарною тривалістю

трьох компонентів: пошуку рейсів, резервування місць і обробки оплати. Формула має вигляд

$$T_{response} = T_{search} + T_{booking} + T_{payment}$$

де T_{search} – це час виконання запиту до бази даних рейсів і тарифів, що може бути розкладений на дві частини: безпосереднє звернення до СУБД (час запиту до диска або кешу) та мережеву затримку при зверненні до API зовнішніх систем (GDS). Тобто

$$T_{search=T_{DB}} + T_{API\_roundtrip}, T_{DB} = \frac{1}{n} = \sum_{i=1}^{n} t_{disk,i}, T_{API\_roundtrip} = 2t_{net},$$

де $t_{disk,i}$ - час окремого звернення до диска чи кешу, а t_{net} – середня затримка мережі. Середнє значення T_{search} обчислюється за масивом з n вимірювань, а його максимальне значення контролюється з врахуванням допустимого відхилення ΔT_{search}= 0.2,T_{search}.

Другий складник, $T_{booking}$ включає час блокування місця, оновлення записів у таблиці бронювань та підтвердження транзакції в PNR. Його модель можна записати як

$$T_{booking} = t_{lock} + t_{update} + t_{confirm}$$

де t_{lock} – час отримання блокування ресурсу, t_{update} - час оновлення кількості доступних місць та тарифів, $t_{confirm}$ - час запису PNR та повернення результату. Для оцінки надійності передбачено, що кожний з цих інтервалів не перевищує 30 % від середнього значення всієї операції бронювання.

Третій компонент, $T_{payment}$, враховує обробку фінансової операції через платіжний шлюз. Математично його описують як

$$T_{payment} = t_{auth} + t_{capture},$$

де t_{auth} – час авторизації карти, а $t_{capture}$ – час списання коштів. З огляду на специфіку 3D Secure, для обох етапів приймають середні значення на рівні 1.5 с та 2 с відповідно з максимальною допустимою затримкою ±0.5\pm0.5±0.5 с.

Для прогнозування майбутнього попиту на квитки використовується лінійна модель

$$D_{futre} = aD_{hist} + \beta S + \gamma E,$$

де D_{hist} – середня кількість бронювань за попередні періоди (наприклад, за останні 12 місяців), обчислена як

$$D_{hist} = \frac{1}{m}\sum_{j=1}^{m} B_j \ ,$$

де B_j - обсяг бронювань у місяць *j*, а *m* – кількість місяців; *S* – сезонний коефіцієнт, який визначається відношенням місячного бронювання до середнього:

$$S_k = \frac{B_k}{D_{hist}}, k = 1 \dots 12;$$

β відображає вплив сезонності на попит; параметр E агрегує зовнішні фактори (святкові дати, економічні індекси, погодні умови), добуток γE задає їхню сумарну вагу. Коефіцієнти α, β, γ обираються методом найменших квадратів на історичних даних з обмеженням $\alpha + \beta + \gamma = 1$ та $\alpha, \beta, \gamma \geq 0$. Точність моделі контролюється відхиленням прогнозу від фактичних даних, яке не повинно перевищувати ±5 %.

Під час формалізації моделі зроблено допущення про стабільність ринку (відсутність раптових змін попиту) і сталість поведінки користувачів, а також про гарантовану роботу API із затримкою не більше t_{net}^{max}. Усі математичні позначення відповідають списку реквізитів, наведеному у специфікації вимог.

Останнім етапом опису є алгоритм (рис 2.4) розв’язання задачі на ЕОМ, що реалізується поетапно: ініціалізація запиту, оновлення масивів, обробка пошуку, резервування, інтеграція з платіжною системою, генерація результатів і

повідомлень. Логіка передбачає перевірку умови успішності кожного етапу та обробку помилок (невдалий платіж, відсутність місць), із переходом до відповідного обробника – retry або rollback.)

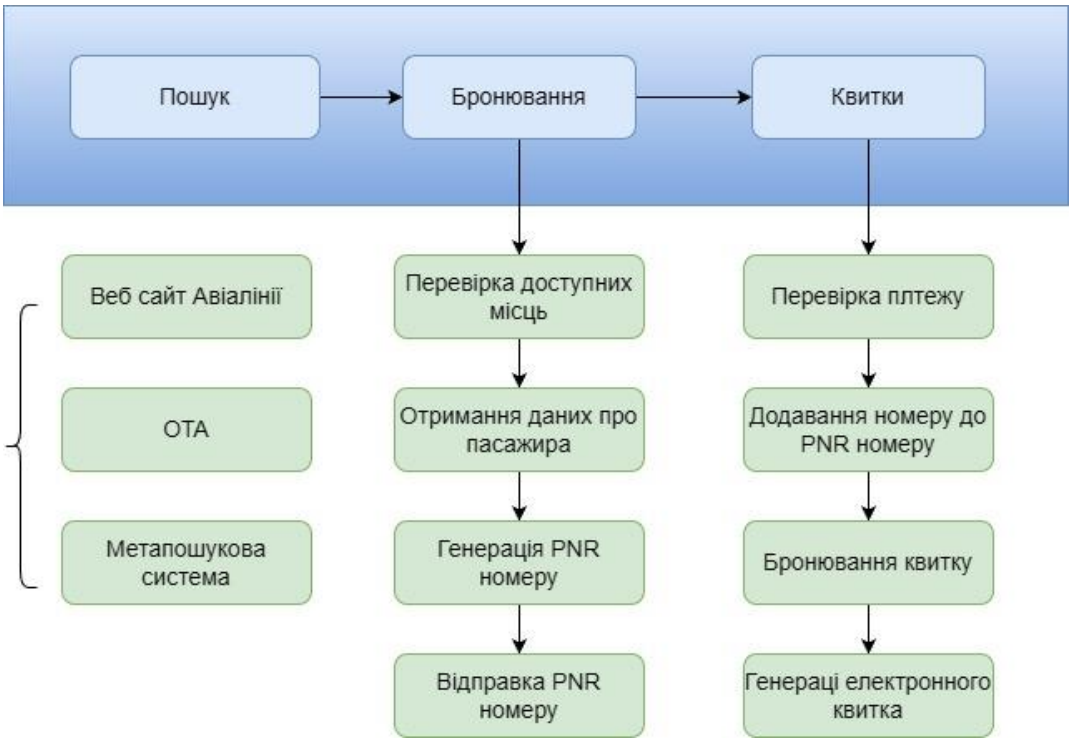


Рисунок 2.4 – Діаграма схеми алгоритму

Джерело: сформовано автором на основі виконаного дослідження

У разі виникнення форс-мажору – наприклад, відсутності відповіді від платіжного шлюзу протягом 5 секунд, алгоритм ініціює таймаут і повідомлення адміністратору, здійснює rollback бронювання, потім повторно надає користувачеві інформацію. Усі гілки алгоритму враховують контрольні точки та реєстрацію дій у журналі операцій.

Таким чином, алгоритм повністю відповідає вимогам, щодо обґрунтування математичного забезпечення, структури даних та керованого розв’язання задачі на ЕОМ. Він забезпечує надійне функціонування системи бронювання за різних умов експлуатації, передбачає ситуації помилок і критичних збоїв, а також формує необхідні вихідні дані для користувачів та адміністраторів.

2.3 Моделювання інформаційної системи

2.3.1 Моделювання поведінки системи

Для того, щоб уявити, як саме функціонує система продажу авіаквитків, створено модель поведінки. Вона демонструє взаємодію системи з користувачами, адміністраторами та зовнішніми сервісами. Поведінка системи описується на трьох рівнях: за допомогою діаграм прецедентів, діаграм послідовностей та діаграм діяльності.

Діаграма прецедентів(рис 2.5) показує, які ролі взаємодіють з системою та які саме функції їм доступні. В нашій системі основні актори – це пасажир (користувач), адміністратор системи та зовнішні сервіси (наприклад, платіжні шлюзи та глобальні дистрибутивні системи). Основні прецеденти включають пошук рейсів, бронювання місць, оплату квитків, адміністрування рейсів та тарифів, формування звітів.

На наступному рівні для кожного прецеденту розроблено діаграми послідовностей (рис 2.6 - 2.9) . Ці діаграми детально описують сценарії взаємодії між різними об'єктами в рамках одного прецеденту. Наприклад, під час бронювання користувач ініціює пошук рейсів, отримує список доступних варіантів, вибирає конкретний рейс, система резервує місце, опрацьовує платіж через платіжний шлюз і завершує транзакцію видачею квитка.

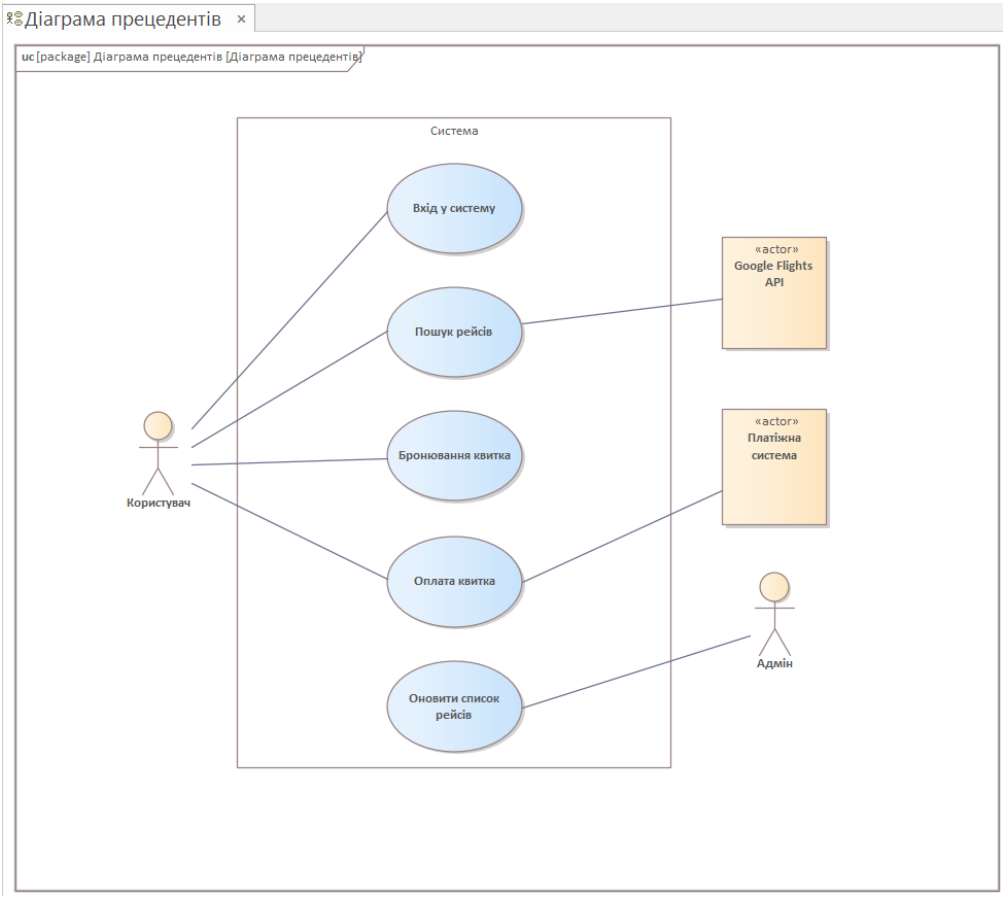


Рисунок 2.5 – Діаграма прецедентів

Джерело: сформовано автором на основі виконаного дослідження

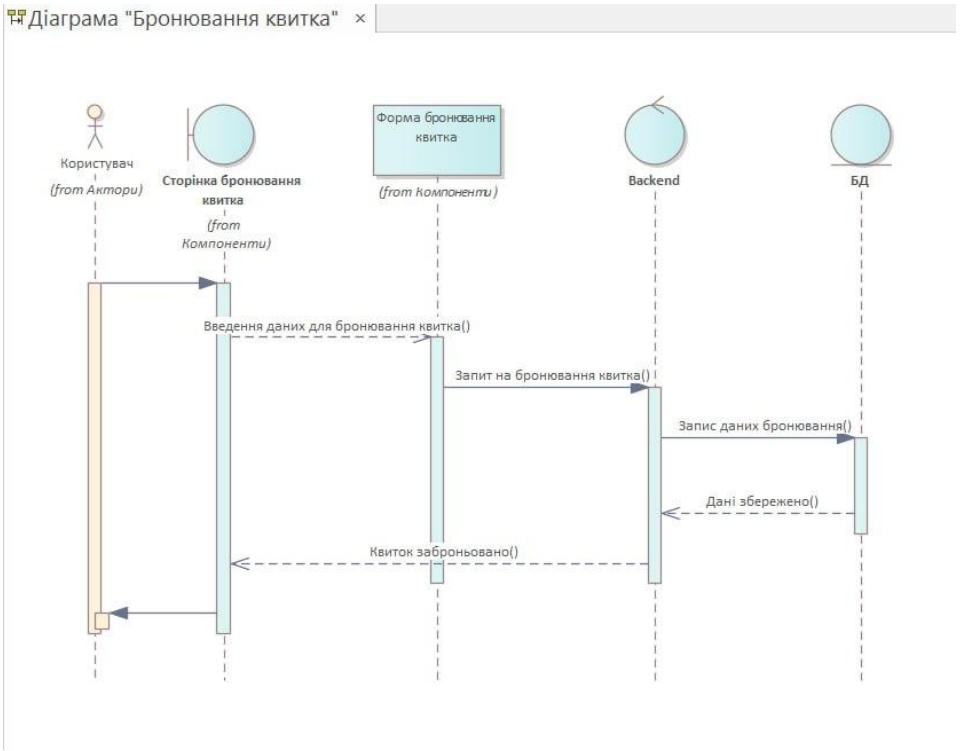


Рисунок 2.6 – Sequence Diagram для прецеденту «Бронювання квитка»

Джерело: сформовано автором на основі виконаного дослідження

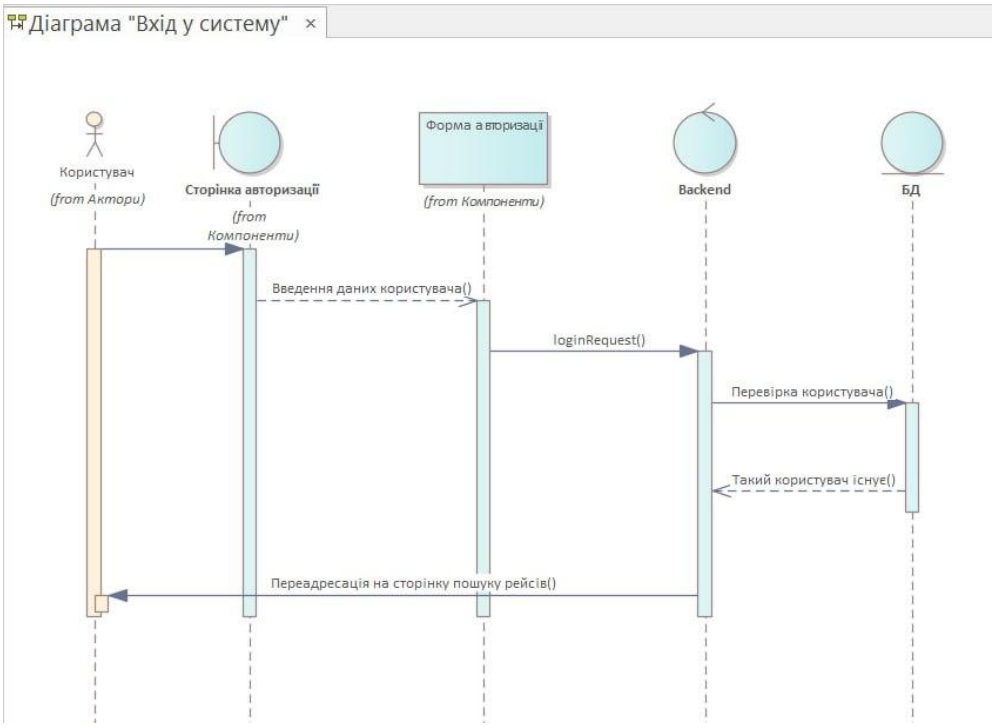


Рисунок 2.7 – Sequence Diagram для прецеденту «Вхід у систему»

Джерело: сформовано автором на основі виконаного дослідження

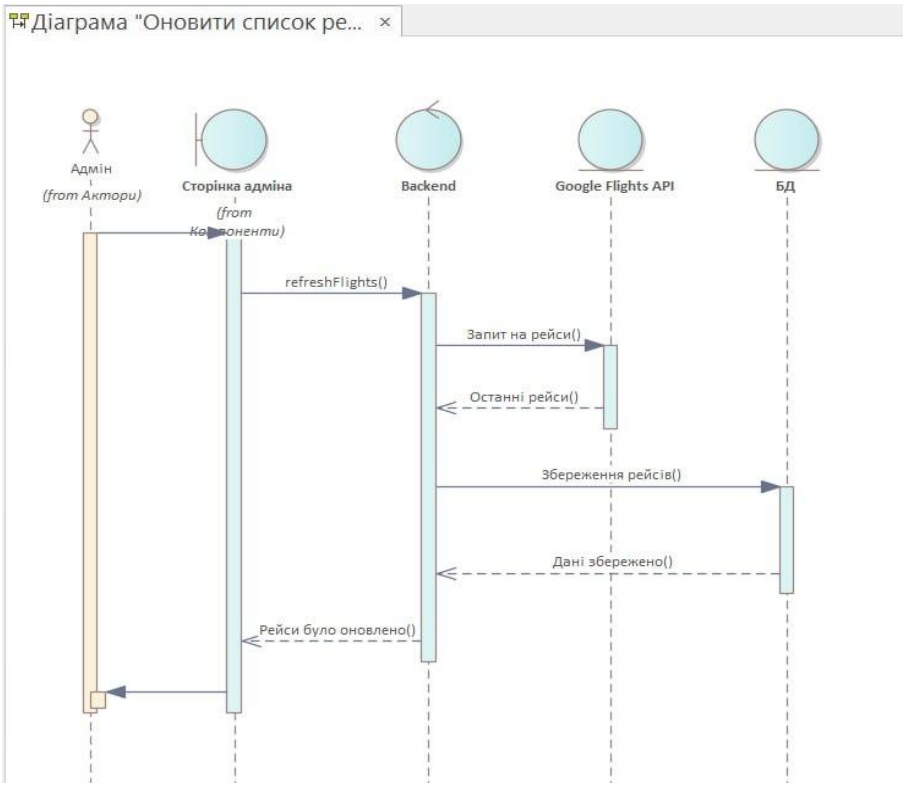


Рисунок 2.8 – Sequence Diagram для прецеденту «Оновити список рейсів»

Джерело: сформовано автором на основі виконаного дослідження

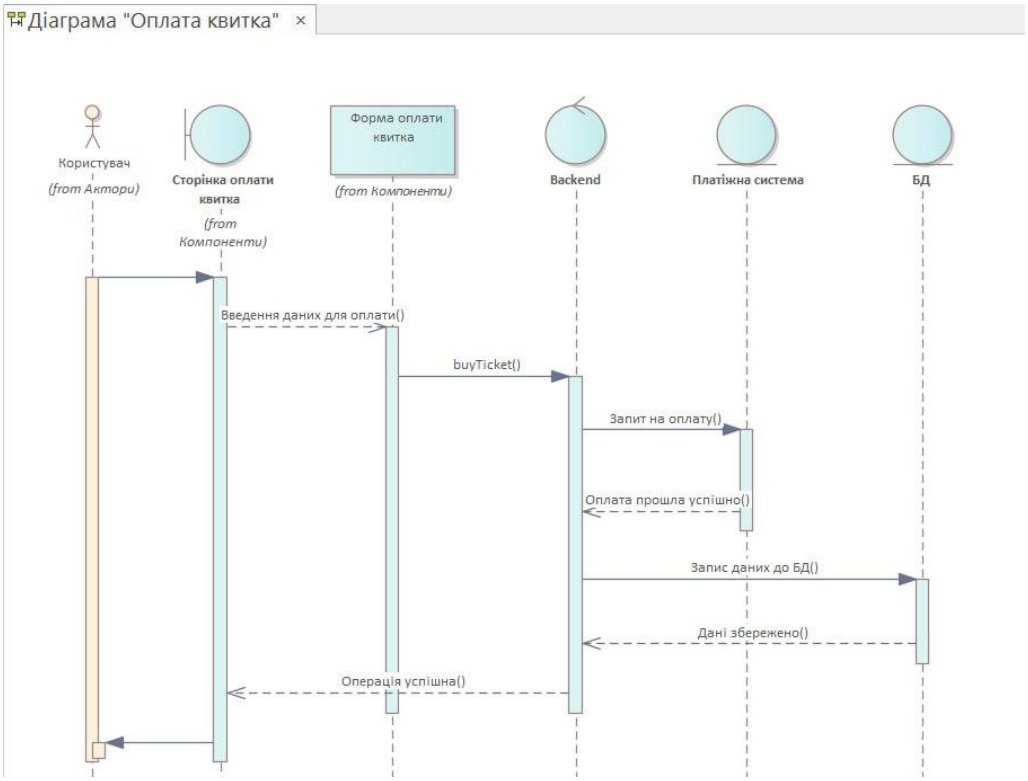


Рисунок 2.9 – Sequence Diagram для прецеденту «Оплата квитка»

Джерело: сформовано автором на основі виконаного дослідження

2.3.2 Моделювання структури системи

Наступним етапом було створення структурної моделі системи (рис 2.10). Це статична модель, яка описує ключові сутності (класи), зв’язки між ними та їхні основні атрибути й операції. Структура представлена у вигляді діаграми класів, що включає три основні типи класів: класи-сутності, класи керування і граничні класи.

Класи-сутності (наприклад, Рейс, Квиток, Користувач, Бронювання) представляють бізнес-дані, які зберігаються та обробляються системою. Граничні класи (наприклад, Інтерфейс Бронювання, Інтерфейс Оплати) описують інтерфейси, через які користувач взаємодіє з системою. Класи керування

(наприклад, Контролер Платежу, Контролер Бронювання) координують логіку роботи системи, визначаючи послідовність і правила виконання бізнес-операцій.

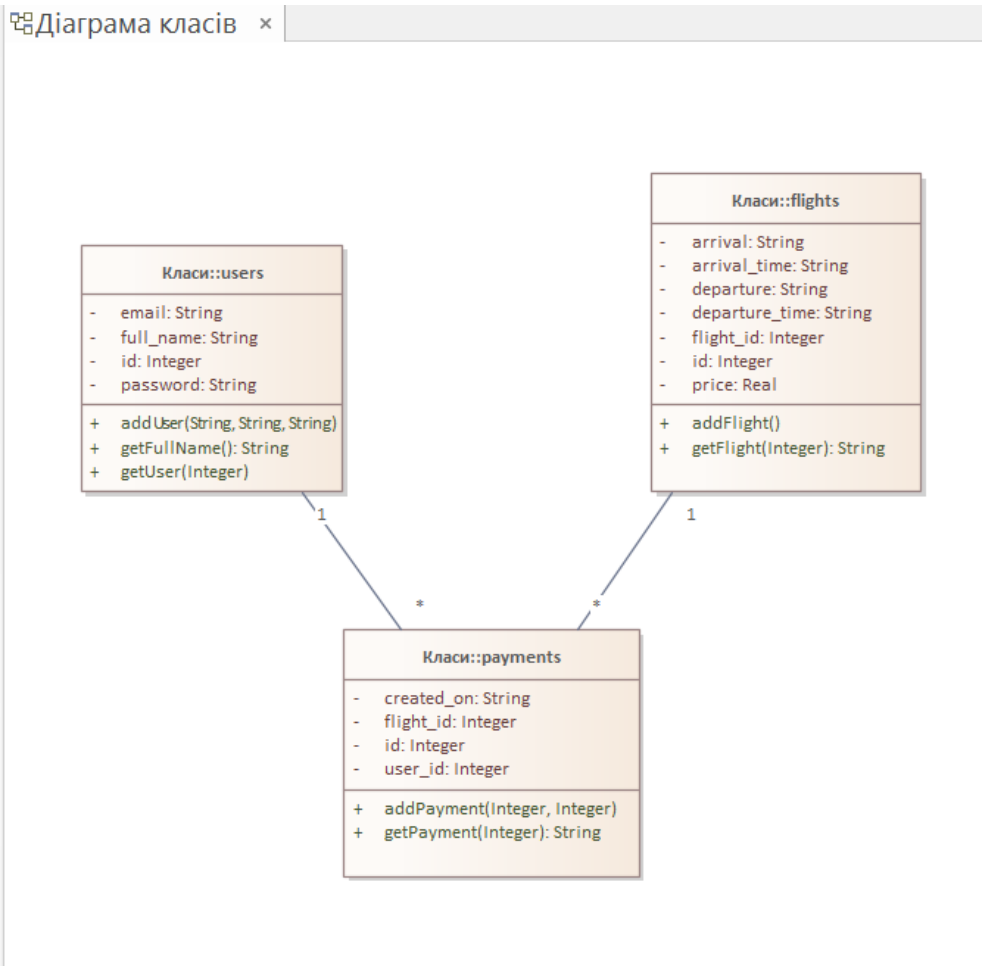


Рисунок 2.10 – Діаграма класів

Джерело: сформовано автором на основі виконаного дослідження

Паралельно з класами UML для уточнення внутрішньої структури системи застосовано підхід блок-орієнтованого моделювання (SysML). Найперше представлено діаграму визначення блоків (рис 2.8), яка демонструє ієрархічне угрупування компонентів: блоки «Клієнтський модуль», «Серверний модуль», «Платіжний шлюз», «Модуль аналітики» тощо, а також зв’язки «складається з» і «залежить від» між ними.

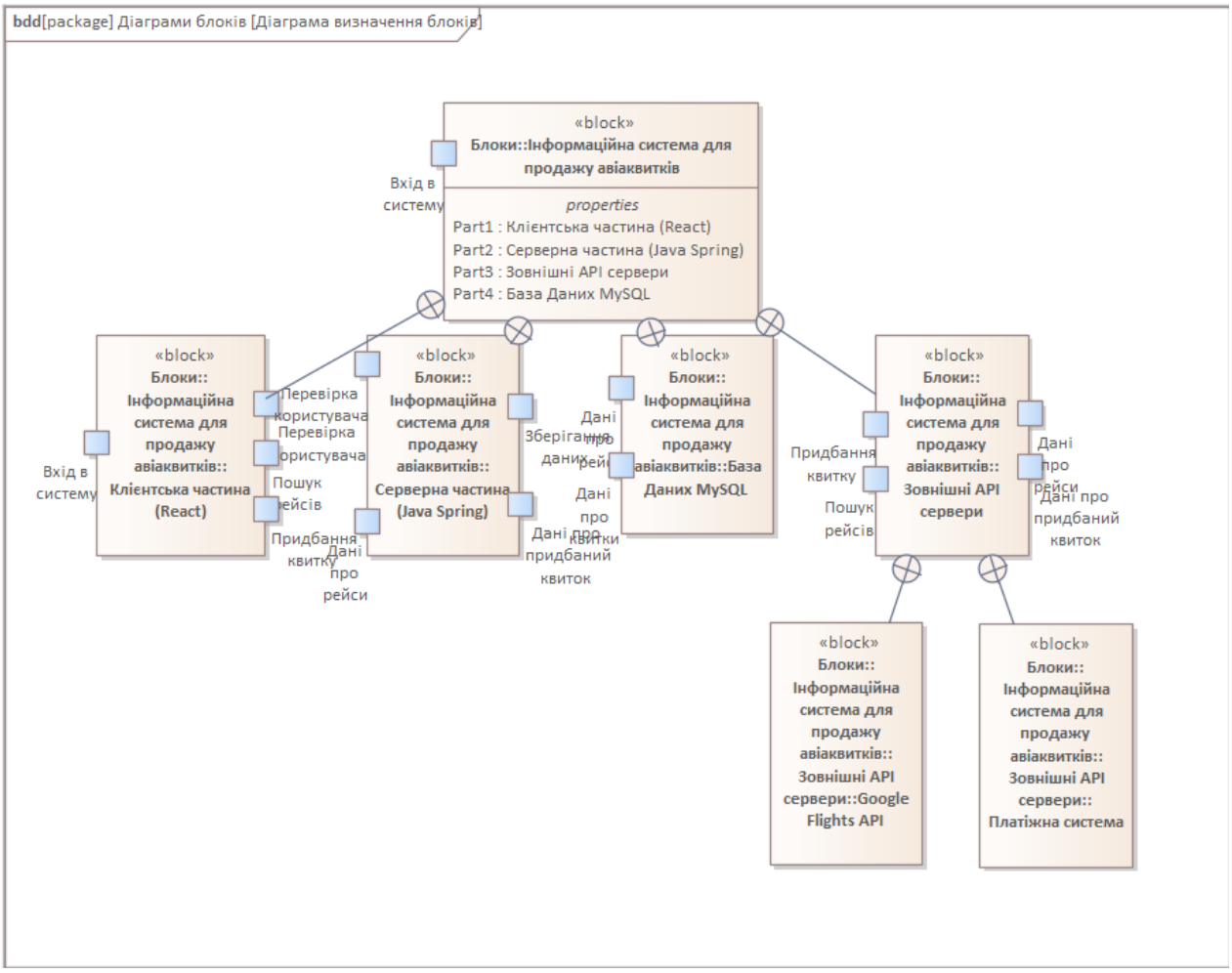


Рисунок 2.11 – Block Definition Diagram

Джерело: сформовано автором на основі виконаного дослідження

На завершення наведено діаграму внутрішніх блоків (рис 2.12) для основного блока «Серверний модуль». Вона показує, як усередині сервера взаємодіють підблоки «Менеджер бронювань», «Обробник платежів», «База даних рейсів», «Кеш пул», а також які порти чи канали зв’язку використовуються для передачі повідомлень і даних між цими підблоками.

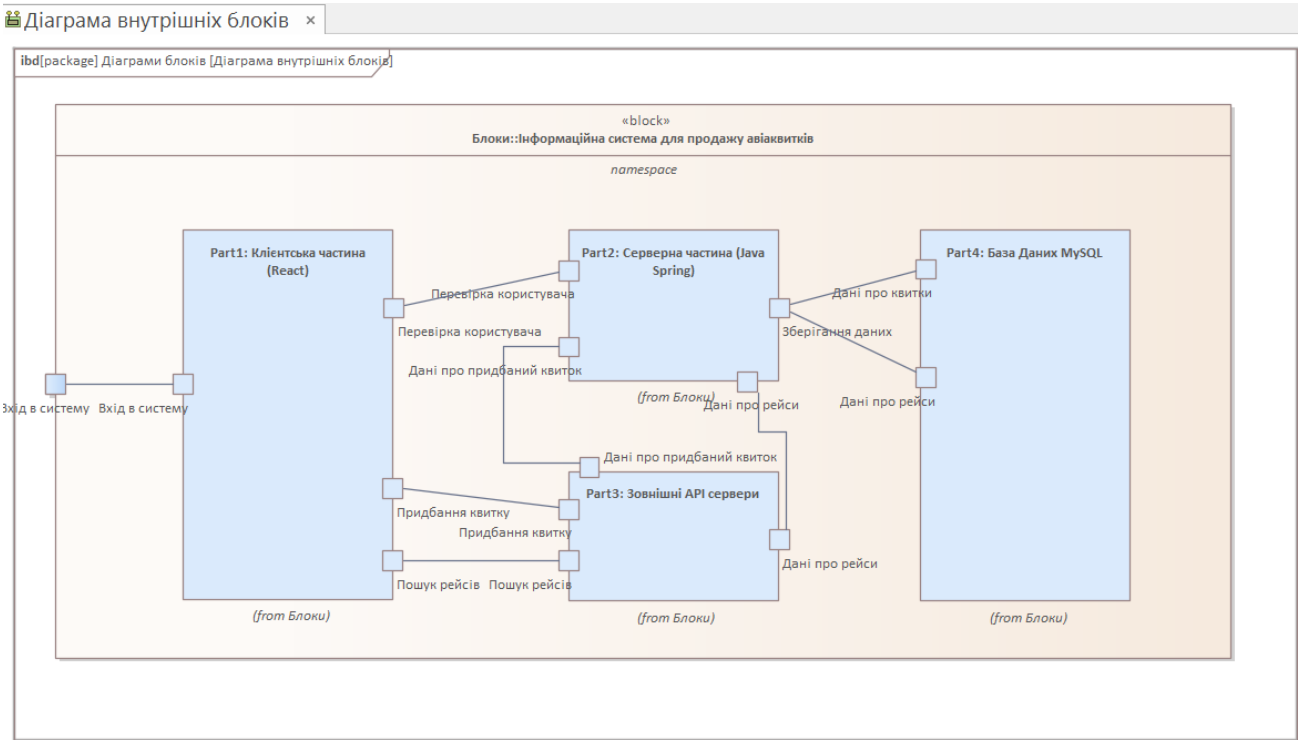


Рисунок 2.12 – Internal Block Diagram

Джерело: сформовано автором на основі виконаного дослідження

2.3.3 Розподіл вимог за компонентами системи

Останнім етапом моделювання стало розподілення функціональних і нефункціональних вимог за відповідними компонентами інформаційної системи. Це дозволяє перевірити повноту охоплення всіх сформульованих раніше вимог, виявити можливі прогалини або дублювання, а також забезпечити чітке розуміння того, яка частина системи за що відповідає. Для реалізації цього підходу була побудована діаграма трасування (рис. 2.13), яка відображає зв’язок між окремими вимогами, сценаріями використання (прецедентами) та компонентами програмного забезпечення. Така діаграма є ефективним інструментом, що підвищує прозорість розробки, дозволяє здійснювати перевірку реалізації вимог на всіх етапах життєвого циклу проєкту та спрощує супровід системи в майбутньому.

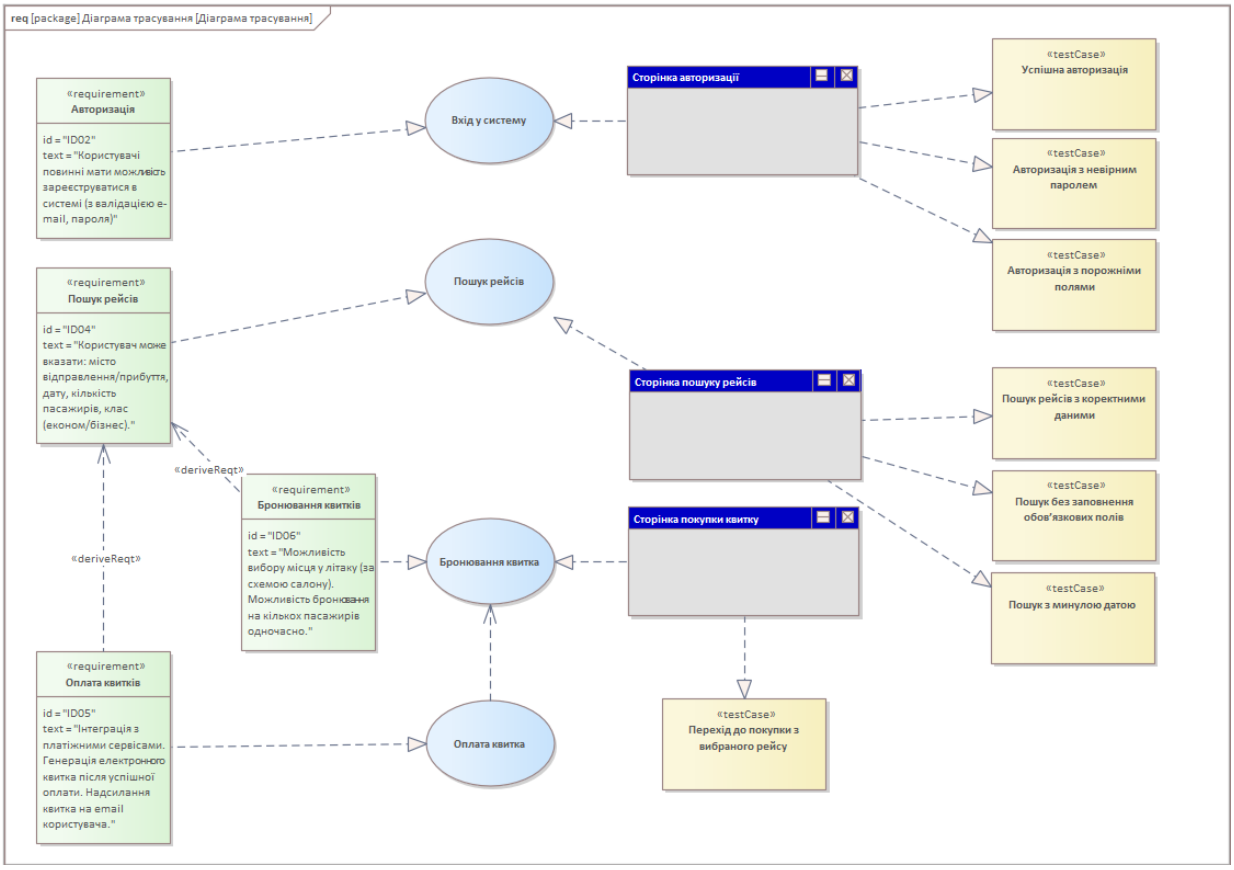


Рисунок 2.13 – Діаграма трасування

Джерело: сформовано автором на основі виконаного дослідження

Отже, завдяки поєднанню декількох рівнів моделювання ми отримали комплексне й цілісне подання про роботу системи: діаграми прецедентів та послідовностей ілюструють сценарії взаємодії користувача з системою та внутрішній потік повідомлень, діаграми класів і блоки SysML демонструють структурну організацію даних і компонентів, а діаграми трасування забезпечують скрупульозне відстеження кожної функціональної чи нефункціональної вимоги до конкретного елемента архітектури. Це дозволяє не лише всебічно оцінити відповідність розробленого рішення початковим цілям і обмеженням, але й гарантує прозорість процесу розробки для всіх зацікавлених сторін: від бізнес-аналітиків до тестувальників та DevOps-інженерів. Кожна модель виступає одночасно й як детальний план реалізації, і як джерело документації, що спрощує підтримку, масштабування та подальше вдосконалення системи, адже будь-які зміни можна оперативно пропрацювати на рівні моделі й передати розробникам у

вигляді визначених артефактів. Таким чином, методи моделювання не лише підвищують якість дизайну та архітектури, але й оптимізують взаємодію між учасниками проєкту, пришвидшують прийняття рішень та знижують ризики помилок на всіх етапах життєвого циклу інформаційної системи продажу авіаквитків.

РОЗДІЛ 3
ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ КОМПОНЕНТІВ СИСТЕМИ

3.1 Інформаційне забезпечення

3.1.1 Загальна характеристика даних та інформаційних потоків у системі продажу авіаквитків

У процесі опису інформаційного забезпечення системи продажу авіаквитків було сформовано уявлення про її архітектуру як про структуровану трирівневу модель, де кожен рівень виконує власну функціональну роль і водночас гармонійно взаємодіє з іншими. На передньому плані – клієнтський інтерфейс, реалізований за допомогою фреймворку Next.js. Саме він забезпечує зручність та динамічність взаємодії кінцевого користувача із системою.

Інтерфейс побудований з акцентом на інтуїтивну взаємодію: пошукова форма реагує майже миттєво, забезпечуючи швидке отримання результатів. Кожен елемент форми доповнений розумними підказками – вибір дати супроводжується візуалізацією доступних для бронювання днів, а поля “місто вильоту” та “місто прибуття” підтримують автозаповнення з використанням кодів IATA. Реалізовані плавні анімації переходів і чітка візуальна структура не лише покращують користувацький досвід, а й формують довіру до системи.

Крім естетики та зручності, клієнтський рівень відповідає і за первинну перевірку коректності введених даних. Наприклад, у випадку спроби вказати дату вильоту, що перевищує один рік від поточної дати, система коректно попереджає користувача про можливу неточність і рекомендує обмежити часові межі пошуку. Такий підхід дозволяє зменшити навантаження на серверну частину та водночас покращити взаємодію з користувачем, орієнтуючи систему на якісне надання

послуг у реальному часі.Далі йде шар серверної логіки, теж побудований на Next.js, але вже зосереджений на обробці API-запитів. Тут формується основна бізнес-логіка: отримані від клієнта параметри надходять у функції API Routes, де перевіряється їхня коректність, після чого формується запит до бази даних. Особливу увагу ми приділили ситуації, коли декілька користувачів одночасно намагаються придбати одне й те саме місце в салоні літака. Щоб уникнути дублювання, перед створенням нового бронювання система спочатку виконує перевірку наявності вільного місця. На рівні MongoDB це реалізовано через унікальний індекс на поєднання полів *flightId* та *seatNumber* у колекції *bookings*. Якщо під час запису спробувати вставити другий документ із тією ж комбінацією, база поверне помилку, і API-шлях обробить цю ситуацію, відправивши користувачеві дружнє повідомлення, що саме це місце вже зайнято, і запросить обрати інше. Такий підхід, по суті, гарантує «атомарність» операції на рівні однієї колекції, хоча офіційного журналу транзакцій ми не використовуємо.

Нарешті, останній рівень – це безпосередньо сховище даних у MongoDB Atlas. Ми обрали цей варіант не просто так: документна модель дозволяє гнучко зберігати різні атрибути рейсу, які можуть зазнавати змін від авіакомпанії до авіакомпанії. Кожен документ у колекції flights містить окремі поля для дати вильоту, дати прильоту, коду рейсу, базового тарифу та навіть посиленого пакету багажу. Також важливо, що Atlas надає можливість автоматичного шардирування даних: якщо одного дня нам знадобиться обробляти десятки тисяч бронювань на хвилину, масштабуватися буде легко й без простоїв. Налаштування реплік-сетів гарантує, що у разі збою однієї з нод база залишиться доступною, а користувачі продовжать роботу без перебоїв.

Для забезпечення безпеки даних ми використовуємо HTTPS на всіх маршрутах, а також стандартні заголовки безпеки (HSTS, CSP). На клієнтській стороні будь-яке звернення до API відбувається через захищений fetch із автоматичним відправленням HTTP-only cookie, що містить сесійну інформацію. Це означає, що JavaScript у браузері не має прямого доступу до токена, а отже, знижується ризик викрадення.

З точки зору доступності, весь код серверної частини й клієнтського інтерфейсу написаний із урахуванням TypeScript-тайпів. Це дає змогу на етапі компіляції виявляти невідповідності в структурах даних, а також полегшує подальшу підтримку.

На завершення слід наголосити, що обрана трирівнева архітектура системи не лише забезпечує чіткий розподіл відповідальності між рівнями, а й створює гнучке підґрунтя для подальшого масштабування та розвитку. Такий підхід дозволяє ізольовано модифікувати або розширювати окремі компоненти без необхідності вносити зміни до всієї системи.

Зокрема, найближчим етапом розвитку передбачено інтеграцію з окремим мікросервісом обробки платежів. Завдяки структурованості поточної архітектури, ця інтеграція може бути реалізована без втручання у клієнтський інтерфейс або основну бізнес-логіку. Взаємодія буде організована через стандартизований API, що забезпечить надійність, модульність та високу адаптивність системи до нових функціональних викликів. Таким чином, проєкт закладає основу не лише для ефективної роботи в поточних умовах, а й для безболісного впровадження інновацій у майбутньому.

3.1.2 Організація збору і передання первинної інформації

Коли користувач натискає кнопку «Знайти рейси», клієнтська частина застосунку збирає всі задані параметри – дату вильоту та повернення, аеропорти відправлення й прибуття, кількість пасажирів – і формує із них структурований запит до серверу. Перед відправленням запиту виконується первинна валідація на стороні клієнта: перевіряється відповідність вибраних дат допустимим діапазонам та коректність комбінації пунктів. У разі виявлення некоректних даних користувач отримує чітке повідомлення з інструкцією щодо виправлення. Усі ці операції

відбуваються динамічно, без перезавантаження сторінки, що забезпечує безперервність роботи інтерфейсу та оперативність взаємодії.

Після завершення клієнтської валідації запит мчить у серверний шар через захищене HTTPS-з’єднання. Тут ми перетворюємо отримані дані на внутрішній формат і починаємо перевірку бізнес-логіки: чи не намагається хтось одночасно забронювати всі місця в салоні, чи не перевищено максимально можливу кількість пасажирів. Всі ці перевірки вбудовані прямо в маршрути Next.js API Routes і написані так, щоб код лишався читабельним і підтримуваним. Якщо запит проходить цей бар’єр, ми звертаємося до нашої колекції flights у MongoDB Atlas. Завдяки індексам пошук документів відбувається миттєво, навіть якщо в колекції вже десятки тисяч рейсів.

У випадку, коли для обраного напрямку потрібно звернутися до стороннього сервісу авіакомпанії, система підхоплює спеціальний адаптер. Він виконує асинхронний HTTP-запит до зовнішнього API, бере свіжі розклади та тарифи і, вставляючи їх у власну базу, оновлює локальні дані. Цей процес налаштований так, щоб не створювати надмірного навантаження – запити в такі сервіси відправляються тільки тоді, коли відсутні відповідні записи в межах останньої години, або ж якщо користувач свідомо обрав опцію «Оновити дані в реальному часі». Збереження нової інформації відбувається безперервно: ми довіряємо MongoDB Atlas і його можливостям автоматичного шардирування та реплікації, тому одночасні операції з декількох вузлів кластера не зламають цілісність даних.

Після того, як потрібна інформація зібрана і відфільтрована, сервер упаковує її у відповідь у форматі JSON, додає заголовки безпеки (CORS-політика, CSRF-токени) й відправляє назад клієнту. У браузері ці дані миттєво перетворюються на зручні картки рейсів, де видно час вильоту, тривалість шляху та вартість. Якщо десь стається помилка – скажімо, не вдається отримати дані від зовнішнього API – користувач отримає зрозуміле повідомлення з порадою спробувати пізніше або обрати інший маршрут. Важливо, що в кожному кроці ми прагнули не просто передати пакет даних, а зробити цей обмін максимально прозорим і дружнім до користувача, щоб ніщо не здавалося «залізяччям під капотом».

За формальним описом стоїть важлива частина – моніторинг і логування. Кожен вхідний запит і вихідний відгук фіксуються у зручному вигляді, що дозволяє нам швидко побачити, де й коли могли виникнути проблеми(рис 3.1). Однак усе це приховано від кінцевого користувача, який бачить лише спрощений інтерфейс і відчуває лише легкість роботи з додатком.

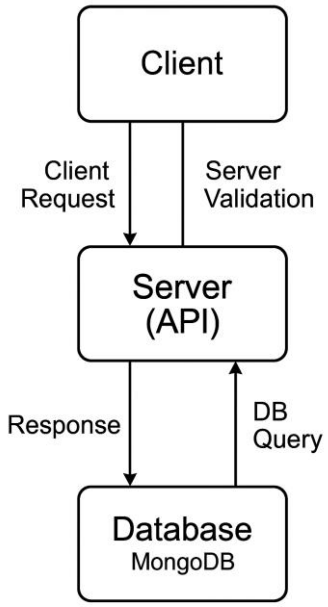


Рисунок 3.1 – Flow - діаграма передачі даних між клієнтом, сервером та БД

Джерело: сформовано автором на основі виконаного дослідження

3.1.3 Побудова системи класифікації та кодування

Сучасний авіаційний хаб характеризується тим, що кожен об’єкт — від рейсу до пасажирського місця — має унікальний ідентифікатор, який забезпечує швидку орієнтацію серед великої кількості доступних варіантів. У процесі розробки системи класифікації та кодування з використанням MongoDB Atlas було визначено ключові сутності та їхні взаємозв’язки.

Ідентифікація рейсів відповідає загальноприйнятим авіаційним стандартам: FlightCode формується з трилітерного коду авіаперевізника (наприклад, PS для

Ukraine International Airlines) та чотиризначного номера маршруту, що гарантує унікальність кожного рейсу. Такий підхід дозволяє користувачеві вводити у пошуковий рядок лише сім символів, що миттєво відображає точний перелік відповідних рейсів без зайвих варіантів.

Паралельно з цим ми звернули увагу на те, що користувачі в різних країнах часто звикли до коротких позначень аеропортів, узгоджених на рівні IATA. Саме тому для кожного пункту відправлення чи прибуття ми використовуємо трилітерний код: навіть якщо в базі десятки тисяч документів із назвами «Київ-Бориспіль» або «Лондон-Хітроу», пошук за “KBP” чи “LHR” миттєво звужує результати до потрібного аеропорту. Це не просто зручно – це мінімізує кількість помилок під час введення та пришвидшує роботу індексів у MongoDB, адже рядкові поля фіксованої довжини індексуються надзвичайно швидко.

Але класифікація не зупиняється на іменах рейсів і кодах аеропортів. Кожне бронювання супроводжується статусом, який ми також закодували згідно з внутрішньою угодою: “PENDING” для очікуваних, “CONFIRMED” для підтверджених та “CANCELLED” для скасованих. Ці чотиризначні та шестизначні рядки зберігаються в окремому полі документа «bookings» і виступають одним із ключів для фільтрації. Завдяки цьому у користувача завжди під рукою актуальна інформація: він може переглянути лише підтверджені бронювання або, навпаки, відсортувати ті, що ще чекають на опрацювання.

Ще один важливий аспект – класифікація пасажирських категорій. У кожного користувача, який створює обліковий запис, зберігається роль (“user”, “admin”, “agent”) і тип клієнта (“individual” або “corporate”). Це дозволяє гнучко налаштовувати інтерфейс: коригувати список доступних тарифів, відображати спеціальні акції для корпоративних клієнтів та приховувати адміністративні опції від звичайних пасажирів. Аналогічно ми позначили класи кабіни (“economy”, “business”, “first”) і основні тарифи (“standard”, “flexible”, “premium”), щоб при бронюванні система могла автоматично вивести порівняльну таблицю з умовами повернення, багажними нормами та можливістю безкоштовної зміни дати.

Усі ці коди об’єднані єдиним принципом: їх довжина та формат визначені за допомогою zod-схем або JSON Schema, що дозволяє на ранньому етапі відсіювати некоректні дані. Наприклад, FlightCode завжди складається із семи символів, IATA-коди мають фіксовану довжину в три символи, а статуси дозволені лише в діапазоні від трьох до дев’яти. Якщо, наприклад, клієнт надішле значення “confirm” замість “CONFIRMED”, така невідповідність буде виявлена ще до запису в базу – згенерується помилка, а користувач отримає зрозуміле повідомлення з проханням внести виправлення. Це забезпечує цілісність даних без дублювання логіки валідації на сервері або в клієнті.

В результаті побудована система класифікації та кодування працює непомітно, але дуже впевнено: з одного боку, користувач насолоджується зручністю введення та швидкістю пошуку, з іншого – розробник впевнений, що кожен документ у базі відповідає очікуванням, і надалі можна розгортати аналітику, генерувати звіти чи додавати нові категорії без ризику «поламати» існуючі дані.

3.1.4 Проєктування форм первинних документів та відеокадрів

У нашій системі «документ» – це не статичний аркуш паперу, а інтерактивна веб-форма, яка сама підказує користувачеві, що й коли потрібно заповнити. Ключовими «первинними документами» виступають пошукова форма та форма бронювання. Пошукова форма має умовну назву – “FlightSearchForm”, її ідентифікатор у коді – flightSearchForm. Це єдина сторінка, де користувач уперше стикається з нашим сервісом, тому ми максимально спростили її зміст: єдине поле з підказками для «точки вильоту» та «пункту призначення», інтерактивний календар, що одразу блокує дати минулого та надто віддаленого майбутнього, і лаконічний селектор кількості пасажирів. Користувачі бачать одразу весь екран – без прокрутки, щоб не відволікатися. Форма (рис 3.2) запускається щоразу під час

входу на головну сторінку, але якщо ви вже були залогінені, вона з’являється як окремий компонент у особистому кабінеті, щоб швидко обрати новий рейс.

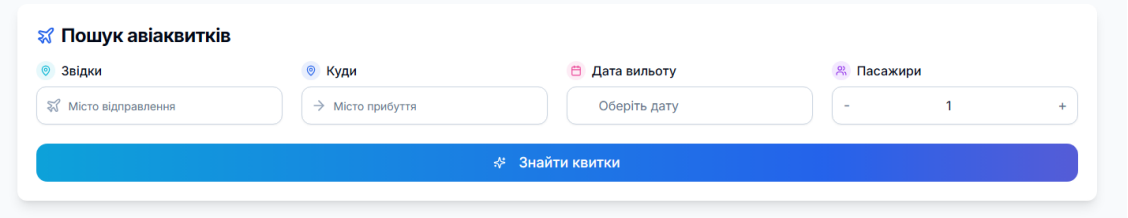


Рисунок 3.2 – Інтерфейс пошуку рейсів

Коли користувач нарешті натискає «Забронювати», спрацьовує друга форма – “BookingForm” з ідентифікатором bookingForm. Вона автоматично підтягуватиме всі обрані дані з попередньої сторінки та виводитиме поля для введення імені пасажира, контактного номеру й email-адреси. Це дозволяє мінімізувати кількість кроків: ніяких незрозумілих полів та дублювання інформації. Форма(рис 3.3) має внутрішню валідацію – якщо ім’я містить цифри чи email не відповідає шаблону, поле моментально підкреслиться червоною хвилястою лінією, а з правого боку з’явиться коротка підказка типу «Будь ласка, введіть справжній email».

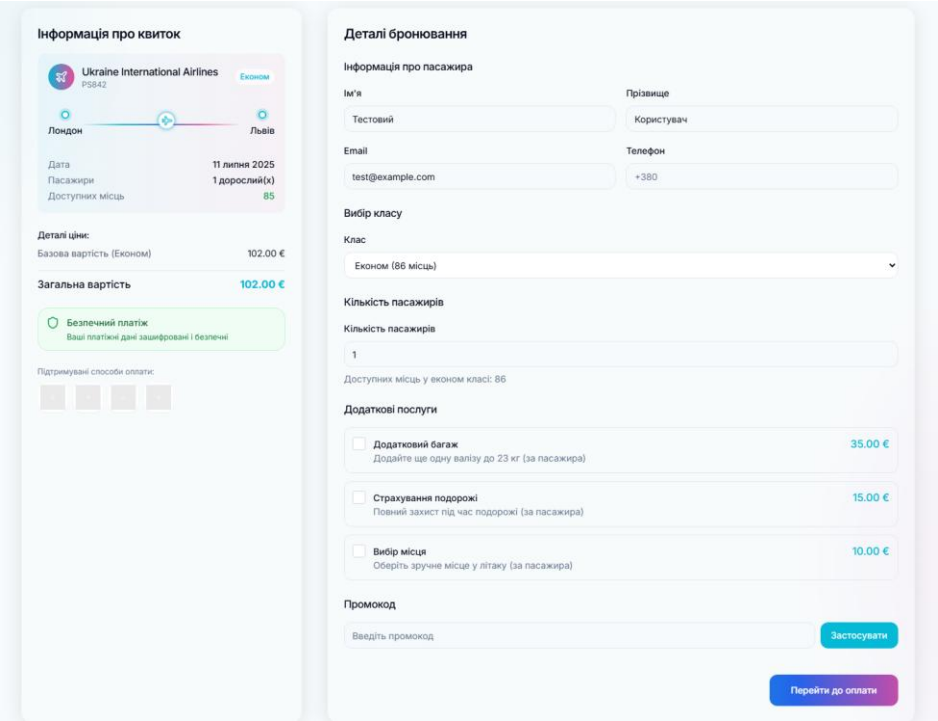


Рисунок 3.3 – Інтерфейс бронювання квитка

Після успішного заповнення й натискання кнопки «Підтвердити бронювання» система формує внутрішній JSON-документ, який ми називаємо «машинограмою». Його структура досить проста і водночас гнучка: в корені лежить bookingId – унікальний ідентифікатор у форматі UUID, потім йдуть поля userId, flightId, seatNumber, cabinClass, status та часові мітки createdAt і expiresAt. Цей документ автоматично надсилається в колекцію bookings MongoDB Atlas і стає точкою правди для всіх подальших дій (наприклад, оплати чи скасування).

Життєвий цикл такого документа має обмежену тривалість: він формується одразу після заповнення форми й містить поле expiresAt, що вказує час дії, орієнтовно близько 15 хвилин. Якщо протягом цього періоду оплата не відбувається, система автоматично змінює його статус на «CANCELLED». Це дозволяє уникнути зайвого резервування місць, забезпечуючи ефективне використання доступного простору в салоні літака.

Що стосується відеокадрів або сигналів керування, у рамках поточного проекту ми їх не реалізовували і навіть не закладали як окремий компонент, оскільки вся взаємодія відбувається в текстово-графічному інтерфейсі браузера. Проте, коли настане час додавати пуш-сповіщення чи інтеграцію з мобільним додатком, ми з легкістю допишемо окремий сервіс, який формуватиме сигнал у форматі JSON або Protobuf і розсилатиме його через WebSocket чи MQTT.

Особливим є те, що вже зараз у нашій логіці передбачено відправку підтверджень на email. Хоча цей функціонал ще в процесі впровадження, його архітектура готова. Ідея полягає в тому, що після успішного запису машинограми до бази сервер відправить внутрішньому «поштовому» сервісу подію з типом booking.confirmed. Ця подія міститиме всі необхідні дані – ім’я пасажира, номер рейсу, дату й час вильоту, номер місця й суму до оплати. Сервіс сформує HTML-лист на основі заздалегідь підготованого шаблону Handlebars, вставить туди логотип компанії, подробиці бронювання та короткі інструкції, а потім передасть лист SMTP-бірці або сторонньому провайдеру SendGrid. Ми вирішили відкласти остаточну інтеграцію з email-сервером до моменту, коли протестуємо повний цикл

на тестовому середовищі, щоб уникнути непередбачених помилок у користувачів (наприклад, потрапляння листів до спаму чи затримок у доставці).

Таким чином, у кожному з первинних документів – від пошукової форми до машинограми – ретельно продумано не тільки те, що користувач бачить на екрані, але й як ця інформація зберігається та передається далі. Завдяки такому підходу ми зможемо в майбутньому безболісно додавати нові формати – від SMS-сповіщень до інтерактивних push-повідомлень – і бути впевненими, що кожен документ відповідає за всіма бізнес-правилами та технічними стандартами.

3.1.5 Структура інформаційних масивів

У сховищі даних система організована як структурована база даних, де інформація про користувачів, рейси та бронювання зберігається в окремих, але взаємопов'язаних колекціях. Архітектура побудована на основі MongoDB Atlas, що забезпечує високу продуктивність та масштабованість системи.

База даних складається з трьох основних колекцій: users, flights і bookings. Кожна колекція представляє собою логічно відокремлений масив документів, де кожен документ містить повний набір атрибутів, необхідних для функціонування відповідного компонента системи.

Вибір документно-орієнтованої моделі даних обґрунтований потребою в гнучкості структури записів та можливості швидкого масштабування. Кожен документ в колекції має унікальний ідентифікатор (_id), що генерується автоматично системою MongoDB, та визначену схему полів, яка забезпечує цілісність даних.

Колекція users зберігає персональні дані користувачів, включаючи облікові записи, контактну інформацію та налаштування профілю. Колекція flights містить детальну інформацію про доступні рейси, включаючи розклади, маршрути, ціни та

технічні характеристики літаків. Колекція bookings фіксує всі операції бронювання, пов'язуючи користувачів з конкретними рейсами через систему зовнішніх ключів.

Така архітектура дозволяє ефективно виконувати складні запити, що поєднують дані з різних колекцій, забезпечуючи при цьому високу швидкість відгуку системи та можливість горизонтального масштабування при зростанні навантаження.

У таблиці 3.1 представлено узагальнений опис структури всіх трьох колекцій з деталізацією основних полів та їх призначення:

Таблиця 3.1 – Опис масивів users, flights, bookings

Масив	Ідентифікатор	Носій інформації	Об’єм (макс.)	Довжина запису	Метод організації	Ключі упорядкування
users	users	колекція документів в MongoDB Atlas	~100 000 записів	до 2 КБ	документно-орієнтований	_id (PK), email (ІНД)
flights	flights	колекція документів в MongoDB Atlas	~10 000 записів	до 1 КБ	документно-орієнтований	flightCode (ІНД)
bookings	bookings	колекція документів в MongoDB Atlas	~500 000 записів	до 1.5 КБ	документно-орієнтований	bookingId (PK), (flightId, seatNumber) (композитний ІНД)

Джерело: сформовано автором на основі виконаного дослідження

В цьому єдиному поданні видно: для кожного масиву ми позначили його назву та технічний ідентифікатор у базі, вказали тип носія (документи в хмарному кластері), оцінили максимальний об’єм існуючих записів та приблизну довжину одного документа в кілобайтах. Метод організації – документно-орієнтований, адже MongoDB працює саме з таким підходом, а ключі упорядкування – це ті поля, за якими відбувається пошук і сортування, при цьому ми віддаємо перевагу унікальним індексам там, де це необхідно.

Система реалізує комплекс бізнес-правил, що забезпечують цілісність та якість даних на рівні бази даних. Кожна колекція має визначену схему первинних ключів: колекція users використовує автоматично генероване поле _id, flights оперує унікальним ідентифікатором flightCode, а bookings застосовує bookingId у форматі UUID для гарантування глобальної унікальності записів.

Реляційні зв'язки між колекціями підтримуються через систему зовнішніх ключів. Поле flightId у колекції bookings встановлює зв'язок з відповідним записом у колекції flights, тоді як поле userId забезпечує асоціацію з конкретним користувачем з колекції users. Така структура дозволяє підтримувати реферативну цілісність даних на рівні додатка.

На значення критичних полів накладено строгі обмеження валідації. Ціна рейсу повинна мати невід'ємне значення, кількість пасажирів обмежена діапазоном від одиниці до максимальної місткості літака. MongoDB автоматично відхиляє документи, що не відповідають цим критеріям, на рівні валідатора схеми, запобігаючи внесенню некоректних даних.

Обов'язкові поля визначені безпосередньо в схемі колекцій. У колекції users поля name та email мають статус required, що унеможливлює створення записів користувачів без цієї інформації. Колекція flights вимагає обов'язкового заповнення полів departureDate та arrivalDate валідними датами. Поле seatNumber у колекції bookings також є критично важливим, оскільки бронювання без призначеного місця технічно неможливе.

Ці правила валідації працюють на рівні бази даних, створюючи додатковий рівень захисту від некоректних даних, незалежно від логіки додатка.

Щодо індексних полів, ми позначаємо ті, що беруть участь у пошуку: email у users та flightCode у flights мають унікальні індекси (ІНД), щоб не було двох користувачів із тим самим email або рейсів із однаковим кодом. У bookings налаштовано складений індекс на комбінацію полів flightId і seatNumber, при цьому дублювання не допускається, щоб гарантувати, що двоє людей не придбають одне й те саме місце. Якщо хтось намагатиметься вставити документ із уже зайнятим

місцем, база поверне помилку, а сервер – дружнє повідомлення з проханням обрати інше місце.

Оскільки ми працюємо з документною моделлю, поняття ідентифікатора індексного масиву MongoDB застосовує напряму до кожного індексу в рамках колекції, тому окремого додаткового ідентифікатора для індексів нам не потрібно.

Нарешті, логічні й семантичні зв'язки в нашій моделі закладені через поля foreign key (userId → users._id, flightId → flights._id). Хоча MongoDB не підтримує реляційні обмеження на рівні движка, ми реалізували перевірку цілісності у бізнес-логіці сервера: перед створенням бронювання ми переконуємося, що відповідний користувач і рейс дійсно існують. Це дає нам впевненість, що жодне «висячого» посилання в базі не залишиться, а дані залишаться чистими та узгодженими.

Таке єдине подання структури інформаційних масивів і набір бізнес-правил надає чітку й зрозумілу картину організації даних у нашій системі, водночас зберігаючи гнучкість та можливість майбутнього розширення.

3.1.6 Вибір СКБД

Коли настав момент ухвалити рішення щодо бази даних, перед очима постав перелік реальних сценаріїв: від швидкого пошуку рейсу по коду до збереження десятків тисяч бронювань одночасно, і всі вони вимагали радикальної гнучкості та надійності. Спочатку розглядалися традиційні реляційні СКБД, адже вони дають чітку структуру таблиць і відомі всім розробникам. Однак, читаючи документацію й переглядаючи типові схеми, стало зрозуміло, що для набору атрибутів нашого рейсу (де можуть з'явитися незвичайні критерії – від особливих багажних тарифів до нетипових проміжних зупинок) доведеться постійно робити міграції схем, змінювати зв'язки, тримати кілька таблиць з NULL-полями та дивитися, як росте складність SQL-запитів.

У той час MongoDB Atlas запропонував зворотний підхід: уявіть, що кожен документ – це окрема картка рейсу або бронювання, де в записі можуть бути вкладені піддокументи з інформацією про багаж, харчування або історію змін, і все це зберігається «під одним дахом». Така гнучкість означає, що на етапі розробки не потрібно передбачати кожен майбутній атрибут і витрачати тижні на ALTER TABLE. А якщо завтра з'явиться нова вимога – наприклад, зберігати погодні умови на момент посадки – досить додати поле в документ, і клієнтський код одразу почне його підтримувати.

MongoDB Atlas здивував своєю готовністю «з коробки»: автоматичне шардирування гарантує, що в разі суттєвого зростання навантаження (скажімо, сотні бронювань в секунду під час «чорної п'ятниці») дані поділяться на сегменти, розміщені на різних нодах, і жодного простою не буде. Реплікація ж забезпечує трьохрівневий захист: первинний вузол і дві вторинні копії, а в разі виходу одного з них із ладу перевага голосу автоматично переходить до наступного, тож додаток продовжує працювати без лагів.

Особливо приємно було бачити, як легко в Atlas налаштовуються правила доступу. Досить одного кліку, щоб вказати, які IP-адреси можуть звертатися до бази з продакшену, а які – лише з девелоперського середовища. Поділивши користувачів на ролі (readOnly, readWrite), ми значно знизили ризик випадкового видалення чи модифікації. Та й резервні копії Atlas робить самостійно: за заданим графіком щоденні та щотижневі бекапи летять у сховище, і навіть якщо трапиться найгірший сценарій, можна відкотитися до будь-якої минулої точки.

Для розробників не менш важливим виявилися «зручності» драйвера. Офіційний пакет mongodb у поєднанні з TypeScript дає інтелектуальні підказки прямо в редакторі: достатньо задекларувати інтерфейс, і компілятор дозволить звертатися лише до прописаних полів, запобігаючи помилкам типу «неіснуюче поле». А ще MongoDB підтримує такі корисні штуки, як TTL-колекції для автоматичного видалення застарілих документів (наприклад, старі логі транзакцій), та Change Streams, щоб наш API міг підписуватися на події й миттєво реагувати на

зміни (наприклад, відправляти пуш-повідомлення, коли бронювання підтверджено).

Не менш вагомою перевагою став вбудований Performance Advisor і пробовідбір (sampling) операцій, які допомагають виявити «важкі» запити та запропонувати, за якими полями краще поставити індекси. Це позбавляє необхідності гадати на основі статистики – система сама аналізує метрики й пропонує оптимальні шляхи.

З економічної точки зору MongoDB Atlas дозволяє почати з безкоштовного рівня, достатнього для розробки та тестування, а потім поступово підвищувати потужність вузлів за потреби, сплачуючи лише за те, чим реально користуєшся. Таким чином, навіть на початковому етапі можна було уникнути великих інвестицій у інфраструктуру, сфокусувавшись на логіці самого додатку.

Отже, вибір на користь MongoDB Atlas виявився прагматичним і стратегічно виправданим: завдяки гнучкій моделі даних, автоматичному шардированню та реплікації, зручним інструментам моніторингу і безпеки, а також безперебійним механізмам бекапу, ми отримали основу, яка не обмежує наші ідеї і водночас гарантує стійкість і швидкодію.

3.1.7 Інфологічна модель бази (сховища) даних

Коли я починав проєктувати інфологічну модель бази даних, уявляв собі не сукупність рядків і полів, а живий організм, в якому кожна сутність має свій сенс і місце, а зв’язки між ними народжують нові можливості для аналізу й розвитку сервісу. Для цього я скористався CASE-засобом Visual Paradigm (можна використовувати й інші подібні пакети – наприклад, PowerDesigner чи ER/Studio), у якому наочно кресляться сутності, їхні атрибути та зв’язки, і весь процес відбувається у візуальному середовищі.

Спочатку я виділив чотири ключові сутності, що відображають домен предметної галузі: User, Flight, Booking та Payment. Кожна з цих сутностей перетворилася на окремий блок із переліком атрибутів: у User – userId, name, email, role; у Flight – flightId, flightCode, origin, destination, departureDate, arrivalDate, basePrice; у Booking – bookingId, userId, flightId, seatNumber, status, createdAt; у Payment – paymentId, bookingId, amount, paidAt, method.

Після того, як атрибути були розставлені, я взявся за нормалізацію. На першому етапі (1НФ) переконався, що в кожному полі міститься єдине атомарне значення – жодних списків міст у полі чи «масивів» тарифів. Далі перейшов до другої нормальної форми (2НФ), перевіривши, що всі неключові атрибути повністю залежать від первинного ключа. Наприклад, basePrice не могло б лежати в сутності Booking, бо вартість прив’язана до рейсу, тож залишив його тільки в Flight. Завершальним кроком стало досягнення третьої нормальної форми (3НФ): жоден неключовий атрибут не залежить транзитивно від первинного ключа. Так, поле method у Payment не могло походити з User або Flight, тому ми пам’ятаємо тільки ті відомості, що безпосередньо стосуються самої оплати.

Візуально цей процес у CASE-засобі відображається у вигляді класичної ER-діаграми (рис 3.4): прямокутники з назвами сутностей, усередині – атрибути, колонки PK і FK підкреслені й позначені, а між сутностями проведені лінії зв’язків із картинками «1» і «∞» збоку. Така діаграма показує, що один User може мати багато Booking, кожне Booking відноситься до одного Flight, і кожне Booking може (але не обов’язково) мати один Payment.

Щоб іншим розробникам або навіть нетехнічним учасникам проєкту було зрозуміло, як створювати та редагувати таку модель, я підготував короткий посібник. Спочатку відкривають новий проєкт у Visual Paradigm і вибирають шаблон «ER Diagram». Далі додають сутності: правою кнопкою створюють нову Entity і задають їй ім’я. Внутрішня панель властивостей дозволяє додавати атрибути, вказувати тип даних і позначати PK або FK. Щоб провести зв’язок, вибирають інструмент «Association» і тягнуть лінію від однієї сутності до іншої, потім у властивостях вказують кардинальності.

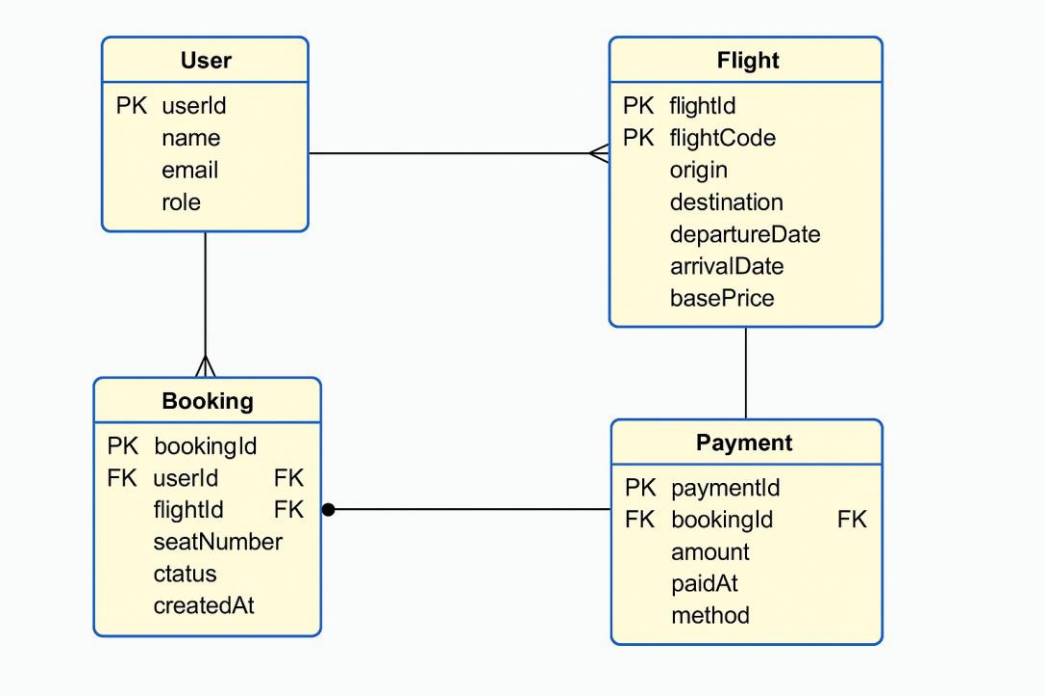


Рисунок 3.4 – Інфологічна модель бази даних інформаційної системи продажу авіаквитків

Джерело: сформовано автором на основі виконаного дослідження

Якщо в майбутньому ми захочемо розширити модель додаванням історії зміни статусів бронювання або таблиць для збереження аналітики, достатньо в тій самій діаграмі додати нову сутність і з'єднати її зв'язками. CASE-засіб автоматично перевірить нормальні форми, підкаже, якщо десь порушено цілісність ключів, і оновить DDL-скрипти для міграції бази.

Хоча проєктом OLAP-сховище ми не займаємося, зауважу, що якби знадобився оперативний аналіз даних, я б обрав модель «зірка»: є факт-таблиця BookingsFact, де лежать поля bookingId, userId, flightId, paymentId, status, createdAt, і навколо неї вимірювання-чотири сутності-розмірності (DimUser, DimFlight, DimDate, DimPayment). Така структура дозволяє швидко виконувати агрегації та звіти. Але для поточної задачі транзакційного характеру нормалізованої OLTP-моделі достатньо.

Отже, інфологічна модель бази даних спроектована в документній формі за стандартом 3-ї нормальної форми, виконана в CASE-засобі з чіткими позначеннями

PK/FK та підготовленими DDL-скриптами. Керівництво користувача по роботі з цією моделлю додає зрозумілі екранні форми, а на основі цієї основи можна швидко розгорнути продуктивну базу або розширити її під будь-які майбутні вимоги.

3.1.8 Даталогічна модель бази (сховища)даних

Коли доходить до фізичної реалізації моделі даних у MongoDB Atlas, головним завданням стає забезпечити не лише коректне зберігання документів, а й прозоре управління схемою колекцій та індексами. У середовищі MongoDB ми працюємо здебільшого з командами DDL у термінах самого Mongo Shell або через скрипти на JavaScript, які виконуються під час розгортання кластера.

Насамперед, для кожної колекції створюється визначення зі строгим JSON-схемою валідації. Наприклад, для колекції users скрипт виглядає так:

```
db.createCollection("users", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["userId", "name", "email", "role"],
      properties: {
        userId: {
          bsonType: "string",
          description: "унікальний ідентифікатор у форматі UUID"
        },
        name: {
          bsonType: "string",
          minLength: 1,
          description: "ім'я користувача"
        },
        email: {
          bsonType: "string",
          pattern: "^.+@.+\\..+$",
          description: "дійсна email-адреса"
        },
        role: {
          enum: ["individual", "agent", "admin"],
          description: "роль користувача в системі"
        }
      }
    }
  },
  validationLevel: "strict"
});
db.users.createIndex({ email: 1 }, { unique: true });
```

Рисунок 3.5 – Скрипт для створення колекції Users

Подібним чином було описано колекцію flights (рис 3.6), де валідація перевіряє коректність дати вильоту та прильоту, а також наявність коду рейсу довжиною сім символів (три літери плюс чотири цифри). Індекс на полі flightCode дозволяє у лічені мілісекунди знаходити потрібний рейс серед тисяч.

```
db.createCollection("flights", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["flightId", "flightCode", "origin", "destination", "departureDate", "arrivalDate", "basePrice"],
      properties: {
        flightId: { bsonType: "string", description: "UUID рейсу" },
        flightCode: {
          bsonType: "string",
          pattern: "^[A-Z]{2}\\d{4}$",
          description: "код рейсу, наприклад PS1234"
        },
        origin: { bsonType: "string", minLength: 3, description: "IATA-код аеропорту вильоту" },
        destination: { bsonType: "string", minLength: 3, description: "IATA-код аеропорту прибуття" },
        departureDate: { bsonType: "date", description: "дата й час вильоту" },
        arrivalDate: { bsonType: "date", description: "дата й час прильоту" },
        basePrice: { bsonType: "double", minimum: 0, description: "базова ціна квитка" }
      }
    }
  },
  validationLevel: "moderate"
});
db.flights.createIndex({ flightCode: 1 }, { unique: true });

```

Рисунок 3.6 – Скрипт для створення колекції flights

Нарешті, колекція bookings(рис 3.7) об’єднує зв’язки з users і flights, а також містить поле seatNumber і статус бронювання. Щоб убезпечитися від дублювання місць, ми застосували складений унікальний індекс на комбінацію flightId + seatNumber:

```
db.createCollection("flights", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["flightId", "flightCode", "origin", "destination", "departureDate", "arrivalDate", "basePrice"],
      properties: {
        flightId: { bsonType: "string", description: "UUID рейсу" },
        flightCode: {
          bsonType: "string",
          pattern: "^[A-Z]{2}\\d{4}$",
          description: "код рейсу, наприклад PS1234"
        },
        origin: { bsonType: "string", minLength: 3, description: "IATA-код аеропорту вильоту" },
        destination: { bsonType: "string", minLength: 3, description: "IATA-код аеропорту прибуття" },
        departureDate: { bsonType: "date", description: "дата й час вильоту" },
        arrivalDate: { bsonType: "date", description: "дата й час прильоту" },
        basePrice: { bsonType: "double", minimum: 0, description: "базова ціна квитка" }
      }
    }
  },
  validationLevel: "moderate"
});
db.flights.createIndex({ flightCode: 1 }, { unique: true });

```

Рисунок 3.7 – Скрипт для створення колекції bookings

Після створення колекцій і індексів ми завантажили тестові дані для перевірки працездатності системи. Ось приклад вставки кількох документів у колекцію flights:

```
db.flights.insertMany([
  {
    flightId: "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
    flightCode: "PS1234",
    origin: "KBP",
    destination: "LHR",
    departureDate: ISODate("2025-07-01T05:30:00Z"),
    arrivalDate: ISODate("2025-07-01T08:45:00Z"),
    basePrice: 150.00
  },
  {
    flightId: "z9y8x7w6-v5u4-3210-tsrq-po0987654321",
    flightCode: "PS4321",
    origin: "LHR",
    destination: "KBP",
    departureDate: ISODate("2025-07-02T14:00:00Z"),
    arrivalDate: ISODate("2025-07-02T18:00:00Z"),
    basePrice: 160.00
  }
]);
```

Рисунок 3.8 – Скрипт для вставки тестових даних в колекцію flights

Аналогічні вставки виконуються для users та bookings, і після цього ми перевірили, що індекси спрацьовують коректно: запити на пошук рейсу за кодом, бронювання місця чи вибір усіх бронювань користувача повертають очікувані результати в межах мілісекунд.

3.2 Технічне забезпечення

3.2.1 Загальні положення та схема автоматизації

У нашому проєкті інформаційна система з продажу авіаквитків розгортається в двох фізично відокремлених середовищах, що разом забезпечують стабільну роботу та зручний доступ як для кінцевих користувачів, так і для внутрішніх співробітників. Серверна частина містить декілька компонентів, розміщених у хмарному дата-центрі: там працює ядро додатка на Next.js, база даних MongoDB Atlas та файли статичних ресурсів. Завдяки розташуванню в сертифікованому дата-центрі з високою відмовостійкістю ми отримуємо захищене середовище з автоматичним резервним копіюванням і шардируванням даних.

Доступ клієнтів до системи реалізовано через Інтернет за допомогою веб-браузера або мобільного додатка. Користувачі можуть бронювати квитки з будь-якої точки світу, де є стабільне з’єднання, не турбуючись про налаштування мережі – досить відкрити сторінку з нашим сайтом або запустити мобільний інтерфейс.

Водночас для працівників офісу авіаційного агентства ми передбачили виділену внутрішню мережу (Back-Office). Тут розташовані робочі станції менеджерів із продажів, операторів кол-центру та бухгалтера. Вони підключаються до хмарних сервісів через захищений VPN-тунель і мають додатковий доступ до локального принтера та сканера для друку та архівації документів. Офісна мережа побудована за принципом “зона front-office/back-office”: у торговому залі (Front-Office) клієнти можуть користуватися терміналами самообслуговування для швидкого пошуку й замовлення квитків, а персонал – допомагати з більш складними операціями через Back-Office.

Нижче наведено укрупнену схему автоматизації (рис. 3.1), яка ілюструє основні групи технічних засобів та їхні зв’язки:

- **Front-Office**

  – термінали самообслуговування і касові місця з POS-терміналами;

  – локальний комутатор, що об’єднує робочі станції і забезпечує доступ до VPN.
- **Back-Office**

  – серверна стійка з обладнанням для тунелювання (VPN-Gateway), проксі-сервер;

  – робочі станції менеджерів, бухгалтерії та техпідтримки;

  – мережеві принтери і сканери для обробки паперових документів.
- **Хмарний сегмент**

  – віртуальні сервери з розгорнутим додатком Next.js;

  – MongoDB Atlas-кластер із репліками в різних зонах доступності;

  – CDN для доставки статичних ресурсів і SSL-термінатор.

У цій схемі показано, як запити від терміналів і браузерів потрапляють у Front-Office, проходять через захищену мережу до серверів Back-Office і далі – до хмарного середовища, де обробляються та зберігаються.

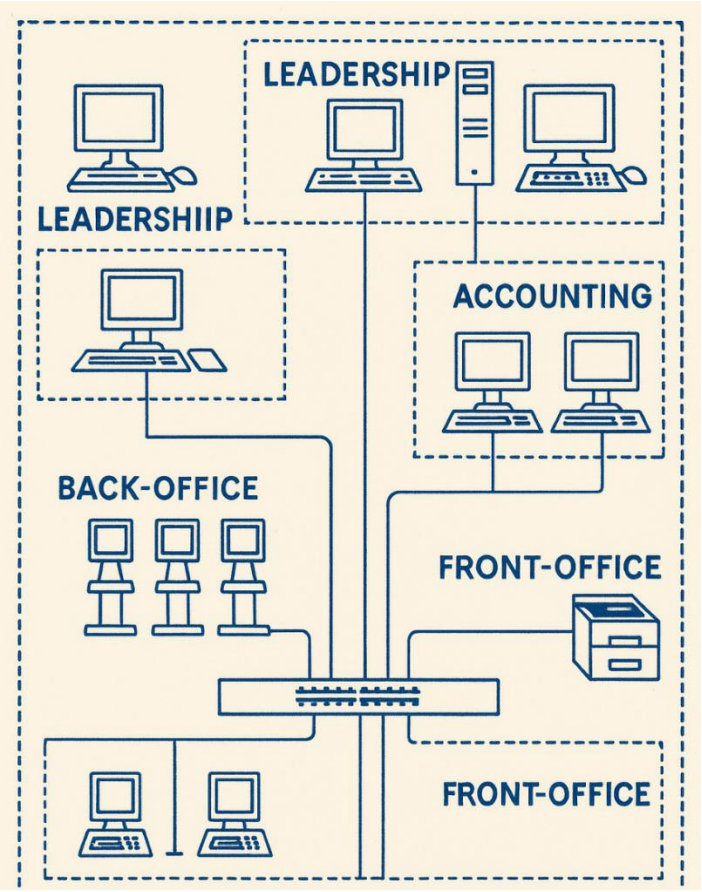


Рисунок 3.9 – Загальна схема автоматизації інформаційної системи продажу авіаквитків

Джерело: сформовано автором за допомогою ШІ на основі виконаного дослідження

Пояснення до схеми:

- Стрілки вказують напрямок обміну даними (HTTP/HTTPS-запити, VPN-з’єднання, внутрішні API-виклики);
- Овальні підписи позначають групи пристроїв, об’єднані за функціональним призначенням;
- Під кожною групою можна уточнити типи обладнання (моделі серверів, комутаторів, POS-терміналів) та ключові характеристики (продуктивність, пропускна здатність).

Ця загальна схема допомагає побачити, як побудована взаємодія між локальними робочими місцями та хмарними сервісами, і створює основу для детальнішого опрацювання кожного сегмента системи.

3.2.2 Структура комплексу технічних засобів

У процесі розробки інформаційної системи продажу авіаквитків було прийнято рішення створити таку технічну інфраструктуру, яка поєднує простоту в обслуговуванні з достатньою потужністю для обробки пікових навантажень. У торговому залі (Front-Office) розміщено п’ять стаціонарних касових терміналів із вбудованими POS-модулями, що забезпечують ефективне обслуговування черги до десяти клієнтів одночасно навіть у години найбільшого попиту. У Back-Office кожен менеджер з продажу, оператори кол-центру та системний адміністратор користуються персональними робочими місцями на базі сучасних ПК з процесором AMD Ryzen 5 5600, 12 ГБ оперативної пам’яті та SSD-накопичувачем об’ємом 256 ГБ. Така конфігурація дозволяє комфортно працювати з об’ємними звітами, складними запитами до бази даних та здійснювати багатозадачну діяльність. Для бухгалтерії передбачено окрему робочу станцію з розширеним обсягом оперативної пам’яті (16 ГБ), адаптовану для роботи з великими файлами експорту.

Серверна інфраструктура розміщена у хмарному середовищі Amazon Web Services. Зокрема, використовується інстанс M10 (3 GB RAM, 1 vCPU, 40 GB disk) для розміщення бази даних MongoDB Atlas, а також статичних ресурсів. Реплікація в трьох зонах доступності дозволяє забезпечити стабільність під час високих навантажень до 1 000 запитів на секунду. Вбудовані механізми резервного копіювання зберігають дані за останні 30 днів. Для безпечного доступу до адміністративного інтерфейсу використовується VPN-Gateway, який обслуговує внутрішню мережу офісу.

Оскільки система також передбачає фізичний облік авіаквитків, на складі встановлено два промислові сканери штрих-кодів Zebra DS3608 та термопринтер для друку бланків. Обладнання підключене до локальних робочих станцій через USB-концентратори, що забезпечує швидке оброблення документів і контроль за обігом паперових носіїв.

Розміщення обладнання відповідає вимогам до техніки безпеки: усі ПК і сервери встановлені у металевих стійках із вентиляційними отворами, температура в серверному приміщенні підтримується на рівні 22 ±2 °C за допомогою кондиціонера. Серверне обладнання розміщене в замкнених шафах з обмеженим доступом, а для зберігання бланків авіаквитків використовуються металеві сейфи з кодовими замками.

З метою забезпечення безперебійної роботи у випадку перебоїв з електропостачанням встановлено джерела безперебійного живлення (UPS) APC Smart-UPS 1500VA, які дозволяють завершити поточні операції протягом 15 хвилин після зникнення електрики. Усі мережеві кабелі прокладено у захисних коробах для запобігання механічним пошкодженням та електромагнітним перешкодам.

Вибір апаратного забезпечення здійснювався з урахуванням нефункціональних вимог: висока швидкість обробки форм, підтримка багатовкладкової роботи в браузері та можливість проведення відеозв’язку з клієнтами. Встановлені ПК із процесорами Intel i5 і SSD-накопичувачами забезпечують стабільну роботу без затримок. Вибір серверної конфігурації базувався на розрахунку середньої вартості транзакції (приблизно 0,02 дол. США за запис) та очікуваного навантаження, а план M10 у MongoDB Atlas демонструє оптимальне співвідношення між ціною та продуктивністю.

Додатково для контролю доступу до приміщень встановлено IP-камери спостереження Hikvision з можливістю локального збереження відео на NAS-сервер, а також турнікет із зчитувачем RFID-карток на вході до Back-Office, який фіксує час приходу й виходу працівників. Для відображення черги в торговому залі використовуються LED-інформаційні панелі діагоналлю 32 дюйми, підключені до офісної мережі через HDMI-репітери за стандартом HDBaseT.

Поточну структуру обслуговує команда з двох системних адміністраторів, які по черзі забезпечують підтримку (із завантаженням до 20 % робочого часу кожного), одного мережевого інженера для контролю VPN-з’єднань та інформаційної безпеки, а також трьох операторів кол-центру, які обробляють

звернення клієнтів. Фінансовий облік і щоденна звітність виконуються штатною бухгалтерією та менеджерами без додаткової залученості IT-фахівців.

Таким чином, запропонована технічна інфраструктура характеризується збалансованістю між продуктивністю, зручністю експлуатації та відповідністю вимогам до безпеки, забезпечуючи стабільну роботу інформаційної системи в умовах динамічного навантаження.

3.2.3 Опис автоматизованого робочого місця

У нашій системі кожне автоматизоване робоче місце (АРМ) створене так, щоб користувач відразу почувався комфортно і не обмежувався у швидкості роботи. Якщо говорити про «серце» робочої станції, то мінімально допустимі системні вимоги виглядають так: процесор Intel Core i5 (щонайменше 4 ядра, 2,5 ГГц), оперативної пам’яті 8 ГБ (DDR4, 2400 МГц) та SSD-накопичувач об’ємом 256 ГБ. Цей «складник» гарантовано витримає одночасне відкриття п’яти вкладок браузера з різними адміністративними інтерфейсами, редакцію XLS-файлів зі звітами та оновлення даних у реальному часі без найменших «гальм».

Не менш важливим компонентом є периферія: 22-дюймовий IPS-монітор із роздільною здатністю Full HD забезпечує чітке відображення графіків продажу та табличних звітів. Клавіатура й миша – стандартні ергономічні моделі з роз’ємом USB, що дозволяють працювати з клавішами довгими годинами без відчуття втоми. Для віде-конференцій та ідентифікації в системі встановлено Full HD веб-камеру Logitech C920, а для дзвінків із клієнтами – легку USB-гарнітуру з гучномовцем і шумопоглинаючим мікрофоном. Осканувати документ чи серію паперових бланків дозволяє компактний планшетний сканер із USB-підключенням, який за секунду обробляє А4-розворот і передає його безпосередньо в інтерфейс браузера. Друк потрібних документів виконує мережевий лазерний принтер із швидкістю 30 сторінок за хвилину.

Кожен АРМ підключається до корпоративної мережі по Ethernet (Gigabit RJ-45), щоб мінімізувати затримки при обміні великими JSON-сервісами з API. Як резервне з’єднання на випадок збою провайдера налаштовано Wi-Fi 5 (802.11ac) з точкою доступу в офісі. Для захищеного доступу до внутрішніх адміністративних сервісів використовується апаратний VPN-клієнт Cisco AnyConnect, який шифрує весь трафік на рівні мережевої карти.

Всього в проєкті передбачено:

- Front-Office: 5 АРМ для підтримки (стабільне дротове підключення).
- Back-Office: 7 АРМ – по одному на кожного менеджера та операторів, плюс дві останні для бухгалтерії.
- Мережеве обладнання: 1 Gigabit комутатор із резервним блоком живлення та 1 Wi-Fi 5 точка доступу.

Таким чином, кожне автоматизоване робоче місце відповідає запитам сучасного бізнесу: швидке завантаження інтерфейсів, зручна взаємодія з клієнтами та надійна захищеність даних у процесі користування.

3.2.4 Схема мережі передачі даних

У нашій системі передача інформації(рис 3.10) між клієнтськими терміналами, офісними робочими станціями та хмарними серверами реалізована за допомогою гібридної мережевої архітектури, що поєднує локальну мережу з виходом в Інтернет. Локальні вузли Front-Office і Back-Office з’єднані через мідні кабелі категорії UTP-8 до комутатора Gigabit Ethernet. Він, в свою чергу, шифрує трафік на апаратному VPN-шлюзі Cisco для захищеного тунелювання віртуальних серверів у хмарі. Саме по цьому тунелю рухаються всі внутрішні API-запити і обмін даними з MongoDB Atlas. Для резервного каналу передбачено обладнання Wi-Fi 5 (802.11ac), яке підключається до окремого VLAN, із застосуванням AES-256 шифрування, щоб забезпечити безперервний доступ у разі збоїв дротового ланцюга.

З’єднання до глобальної мережі організоване через оптоволоконний канал FTTH-SM-02 із пропускною спроможністю 100 Мбіт/с, який орендується у провайдера. Усі клієнтські IP-адреси призначені статично, що полегшує адміністрування DNS-записів і гарантує стабільність доступу зовнішніх систем для інтеграції. Загалом у мережі задіяно три маршрутизатори Cisco ISR 1100 (два для балансування навантаження та один для резерву), два L2-комутатори та одну точку доступу Wi-Fi.

Щоб правильно оцінити потребу в каналах, ми проаналізували характер трафіку. Бухгалтерія протягом робочих днів впродовж восьми годин обмінюється з сервером лише документами й голосовими викликами, споживаючи в середньому 1–2 Мбіт/с. Менеджери та оператори кол-центру одночасно мають до п’яти відкритих сесій адміністрування, що вимагає близько 5 Мбіт/с кожен під час піку. Оператор відеоспостереження приймає HD-потік із камер цілодобово, що створює навантаження до 8 Мбіт/с. У сумі ці дані обґрунтували вибір оптики на 100 Мбіт/с із запасом для розширення до 1 Гбіт/с.

Надійність мережі ґрунтується на використанні промислових UPS-блоків APC для живлення маршрутизаторів і комутаторів, гарантія яких становить три роки, з плановою заміною батарей кожні два роки. Апаратура розміщена в серверній шафі з захистом від пилу й вологи, а мережеві кабелі прокладені в захисних трасах, що дозволяє уникнути механічних пошкоджень. Кожен маршрутизатор має резервний блок живлення і автоматично перемикається на GSM-модем Huawei у разі втрати основного каналу, що забезпечує рівень відмовостійкості не нижче 99,9 % на місяць.

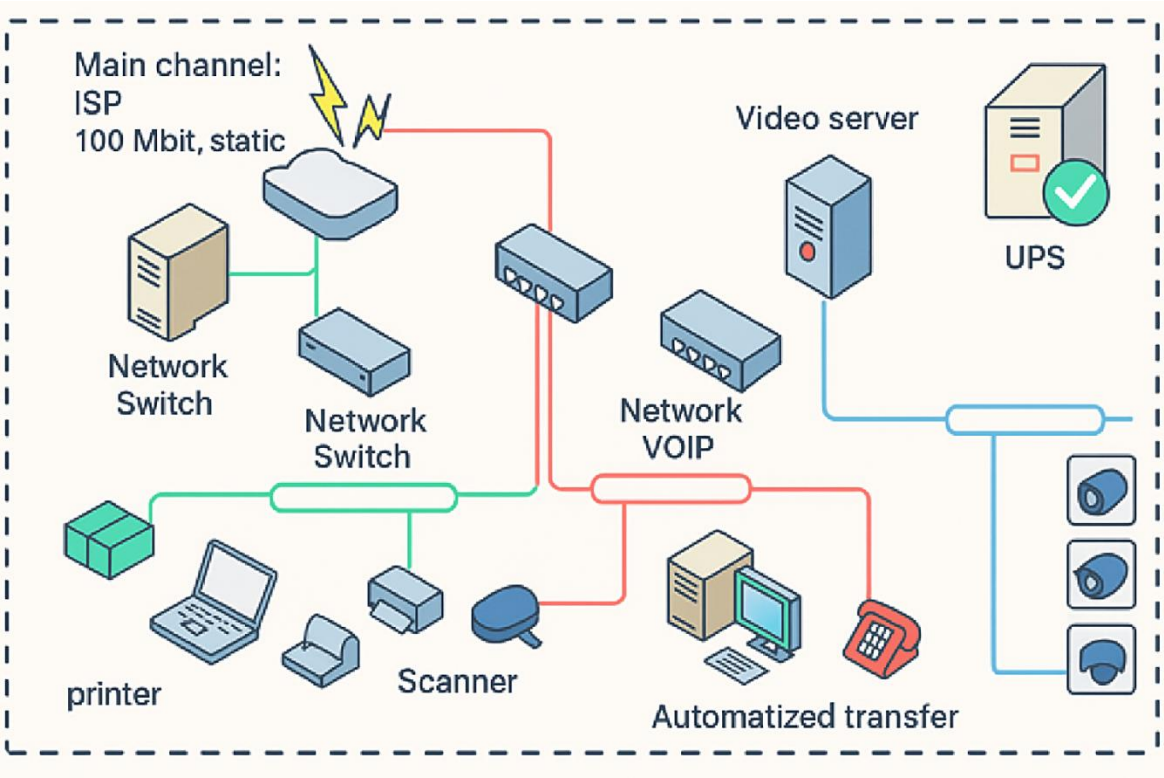


Рисунок 3.10 – Схема мережі передачі даних

Джерело: сформовано автором за допомогою ШІ на основі виконаного дослідження

3.3 Програмне забезпечення

3.3.1 Структура програмного забезпечення

У процесі формування загального уявлення про програмну архітектуру інформаційної системи продажу авіаквитків було поставлено завдання – відобразити взаємозв’язки між усіма ключовими компонентами системи на різних рівнях: від базового апаратного забезпечення до інтерфейсів, з якими безпосередньо взаємодіє користувач. На рисунку 3.11 подано узагальнену схему програмного забезпечення, яка демонструє умовний поділ системи на три рівні: системний (найнижчий рівень), логічне ядро з реалізованою бізнес-логікою, а також верхній рівень – користувацькі сервіси та документація.

Системне програмне забезпечення складається з операційної системи Linux (Ubuntu Server LTS), на якій розгорнуті всі серверні компоненти. Саме ця ОС відповідає за планування процесів, управління пам’яттю й захист ресурсів. Серверну логіку реалізовано через API-роути Next.js, що не потребують додаткового середовища Node.js чи зовнішнього менеджера процесів. На клієнтській стороні використано сучасний JavaScript-стек, зокрема бібліотеку React та фреймворк Next.js, які забезпечують побудову динамічного інтерфейсу користувача. Для тестування і взаємодії із застосунком застосовувався браузер Google Chrome.

Над цим фундаментом «працює» прикладне програмне забезпечення, яке реалізує всі бізнес-процеси з продажу квитків (рис 3.11). Тут у центрі уваги – серверні API-шари, розбиті на модулі: аутентифікація, пошук рейсів, бронювання, оплата та обробка повідомлень. Кожен із цих модулів спроектовано як окремий мікросервіс із власним набором ендпоїнтів, але в документації Enterprise Architect вони об’єднані в єдину логічну схему, щоб було зручно відстежувати зв’язки. Для створення сценаріїв взаємодії користувача з системою ми користуємося Scenario Builder: це дозволяє описати умови «до» й «після» кожної операції (наприклад, перевірка даних перед бронюванням), а також фіксувати бізнес-обмеження.

Зверху розташовані користувацькі сервіси: веб-інтерфейс, мобільні API та адміністративна консоль. Веб-інтерфейс зібрано за допомогою Next.js і TypeScript-шаблонів; мобільна частина побудована на React Native, що дає змогу публікувати додаток одночасно на iOS та Android. Адміністративна консоль – це окремий SPA-додаток, який дозволяє керувати рейсами, тарифами та аналітикою в реальному часі.

Програмна документація, створена в Enterprise Architect, включає три групи артефактів: специфікацію API з прикладами запитів і відповідей, моделі даних у форматі UML та посібники користувача з покроковими описами режимів роботи. Саме ця документація служить дорожньою картою для будь-якого нового розробника чи адміністратора, що приєднується до проєкту, і гарантує, що ми всі говоримо однією «мовою» про наші програмні рішення.

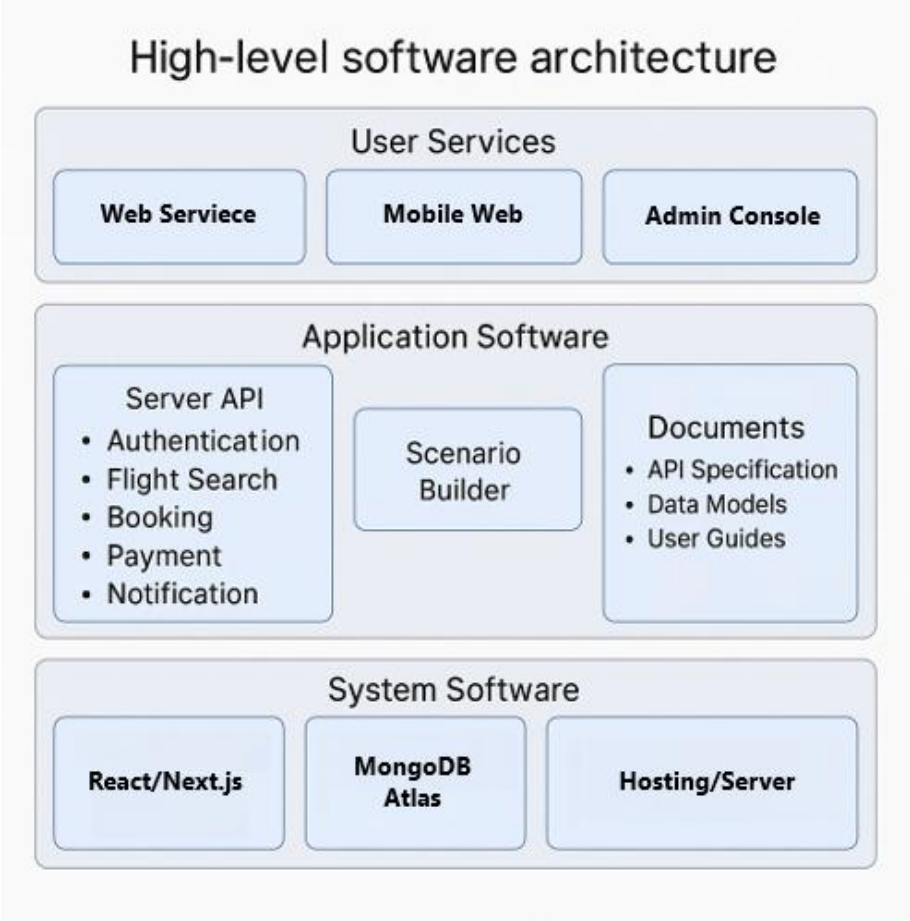


Рисунок 3.11 – High-level software architecture

Джерело: сформовано автором на основі виконаного дослідження

3.3.2 Системне програмне забезпечення

Грунтуючись на вимогах до надійності, безпеки та гнучкості, операційною системою вибрано Ubuntu Server LTS. Цей дистрибутив Linux забезпечує стабільність роботи й тривалу підтримку оновлень без необхідності «великих міграцій» на нові версії, а також має вбудовані засоби управління мережею й безпеки, яких нам достатньо для побудови захищеного середовища. На робочих станціях менеджерів та операторів використовується Windows 10 Pro із набором

корпоративних політик групової безпеки, що дозволяє централізовано налаштовувати антивірусний захист і шифрування жорстких дисків.

Серверна частина реалізована у вигляді API-роутів Next.js, що дозволяє обробляти запити без потреби у зовнішньому серверному середовищі або додатковому менеджері процесів. Завдяки вбудованому механізму Next.js обробка запитів здійснюється безперервно та стабільно в рамках єдиного веб-застосунку. Усі графічні моделі, UML-діаграми та специфікації бази даних були створені в Enterprise Architect, а деталізовані діаграми ER і сценарії бізнес-процесів – у Visual Paradigm. Ці CASE-засоби допомагають швидко вносити зміни в логічну та інфологічну моделі, а потім генерувати DDL-скрипти для MongoDB та документацію для команди.

Основою веб-клієнта стала кастомна реалізація на React і Next.js, зібрана через Webpack і розгорнена у Docker-контейнерах. Ми свідомо відмовилися від готових CMS, оскільки це дало змогу контролювати кожен аспект інтерфейсу й оптимізувати завантаження сторінок для досягнення низької затримки.

До складу сервісного програмного забезпечення входить набір утиліт для резервного копіювання (rsync, mongodump), моніторингу (Prometheus із Grafana) та антивірусна програма ClamAV на файловому сервері. Регулярне обслуговування мережевого обладнання та серверів відбувається з використанням скриптів на Bash і PowerShell, які перевіряють доступність всіх ключових сервісів і надсилають оповіщення в Slack.

Захист інформації реалізовано не лише через шифрування каналів VPN, але й завдяки Apache Kafka із SSL/TLS для обміну внутрішніми повідомленнями між мікросервісами. Усі секрети (ключі доступу, сертифікати) зберігаються в HashiCorp Vault, що забезпечує суворий контроль версій і обмеження доступу за ролями.

Проєкт не передбачає використання спеціалізованих мовних або графічних бібліотек, а також окремих скриптів на Python. Уся інструментальна частина обмежується стандартним веб-стеком – Next.js з підтримкою TypeScript, що включає в себе як клієнтську, так і серверну логіку через вбудовані API-роути. Додаткові компілятори або інтерпретатори в рамках реалізації не застосовувалися.

Таким чином, добірка системного програмного забезпечення побудована навколо відкритих, перевірених часом рішень, що забезпечують необхідний функціонал та здатність швидко адаптуватися до нових вимог без зайвих витрат на ліцензії та довгі часові витрати на підтримку.

3.3.3 Прикладне програмне забезпечення

Коли розробляли набір прикладних програм для нашої системи продажу авіаквитків, ми прагнули поєднати гнучкість, швидкість впровадження та глибоку інтеграцію з обраною базою даних. Загальні інструменти – такі як редактори коду VS Code зі зручними плагінами для JavaScript/TypeScript, утиліти для відлагодження HTTP-запитів (наприклад, Postman) та клієнти для роботи з MongoDB (mongosh або MongoDB Compass) – стали основою, без якої неможливо уявити жоден етап розробки.

Методоорієнтоване програмне забезпечення, зокрема Scenario Builder, дозволяло швидко створювати ітеративні моделі користувацьких сценаріїв: кожну бізнес-операцію – від пошуку рейсів до підтвердження оплати – ми описували візуально, прописуючи в Scenario Builder передумови та результати, після чого автоматично отримували тестову документацію.

Проблемно-орієнтовані модулі, які реалізують ключову функціональність системи, організовані в рамках API Routes, що надаються Next.js. Модуль бронювання реалізує збереження та перевірку даних безпосередньо через MongoDB, без використання додаткових обгорток типу Mongoose. Пошук рейсів здійснюється шляхом побудови фільтрованих запитів до колекції flights. Платіжна інтеграція реалізована за допомогою бібліотеки Stripe, включаючи обробку webhook-повідомлень у межах серверних функцій Next.js. Для організації роботи в глобальній мережі використовувалися перевірені рішення: у ролі шару обміну даними між мікросервісами працює Apache Kafka, що гарантує відмовостійкість і

можливість масштабування, а WebSocket-сервер на базі Socket.IO забезпечує миттєві повідомлення користувачам про зміну статусу бронювання.

Адміністрування обчислювального процесу та розгортання виконуються через набір власних CLI-утиліт, обгорнутих у скрипти на Bash і TypeScript: вони дозволяють одним рядком збирати контейнер Docker, застосовувати міграції, перевіряти цілісність моделей даних та оновлювати конфігурації без “ручної пляски” по серверу.

Для розробки застосовано сучасне інтегроване середовище Visual Studio Code з розширеннями для React, Next.js і MongoDB, що забезпечує зручну навігацію по коду, підсвічування схем і швидке створення компонентів. Це дозволяє ефективно працювати з усіма частинами проєкту — від інтерфейсу до запитів до бази даних. Такий підхід відповідає принципам швидкої розробки (RAD), де важливу роль відіграють швидкі ітерації, уніфіковані засоби роботи з даними та зручне налагодження функцій без потреби в додаткових менеджерах процесів.

3.3.4 Програмна документація

У ході реалізації інформаційної системи створено повний пакет програмної документації, який служить дорожньою картою для всіх учасників проєкту – від кінцевого користувача до розробника й адміністратора. На самому початку йде формуляр програми, в якому викладено її основні технічні характеристики: версії компонентів, перелік файлів і модулів, що входять до складу, а також коротка інструкція з початкового налаштування та запуску. Завдяки цьому будь-хто, хто вперше стикається з нашим ПЗ, може за лічені хвилини переконатися, що нічого не загублено і всі необхідні елементи на місці.

Далі в документації описано призначення програми та області її застосування. Тут розповідається, що головна мета системи – забезпечити зручний і надійний механізм пошуку і бронювання авіаквитків, а також обробку оплат і

надсилання підтверджень. Відзначено, що система підходить як для онлайн-агентств, так і для стаціонарних кас у торговому залі, а також може використовуватися для аналітики та звітності в режимі реального часу.

Особлива увага приділена обмеженням на мінімальну конфігурацію технічних засобів. У документі наведено, що для коректного відображення веб-інтерфейсу та роботи адміністративної консолі достатньо ПК із двоядерним процесором та 4 ГБ оперативної пам’яті, а для серверної частини рекомендовано не менше 8 ГБ RAM і SSD ємністю від 256 ГБ. Такий опис убезпечить користувачів від ситуацій, коли на застарілому обладнанні програма працює з гальмами або збоїть.

Для системного адміністратора в окремому розділі зібрано всі необхідні відомості для перевірки працездатності та налаштування системи. У ньому детально описано параметри конфігураційних файлів, порядок імпорту сертифікатів у VPN-шлюз (за потреби), запуск і моніторинг серверних функцій через вбудовані механізми середовища Next.js, а також сценарії відновлення після аварійного завершення роботи. Цей блок виконує роль технічного довідника, що дозволяє швидко зорієнтуватися в налаштуваннях і усунути типові проблеми без втручання розробників.

Ще одним важливим елементом документації є контрольний приклад. У ньому наводиться готовий набір тестових даних та кроків, які необхідно виконати на чистій інсталяції, щоб переконатися, що система працює як задумано. Це включає створення тестового користувача, пошук рейсу, оформлення бронювання, емуляцію оплати й отримання email-підтвердження. Завдяки такому послідовному опису перевірка коректності розгортання перетворюється з складного процесу на просте відтворення інструкції.

У додатку зібрані текстові версії всіх модулів програми – від серверних API Routes до клієнтських компонентів React і допоміжних сценаріїв на Bash. Ці вихідні тексти супроводжуються короткими коментарями й рекомендаціями щодо структурування коду, щоб новий розробник швидко зрозумів, як влаштована система і де шукати потрібний фрагмент.

Таким чином, програмна документація охоплює всі рівні користування і підтримки ПЗ, забезпечуючи зрозумілість і передбачуваність у будь-яких ситуаціях – від першого знайомства з інтерфейсом до глибокого налаштування серверного середовища.

3.4 Результати реалізації інформаційної системи

У ході виконання проєкту було реалізовано низку оригінальних рішень, які становлять центральну частину інформаційної підсистеми продажу авіаквитків. Серед таких пропозицій варто відзначити механізм «атомарного» блокування окремого місця у салоні літака, реалізований за допомогою індексу ticketId у колекції “bookings”. Цей підхід дозволяє позбутися класичних колізій під час одночасних запитів на одно й те саме місце, водночас уникаючи повноцінних транзакцій і значного навантаження на СКБД. Другим важливим нововведенням стала гнучка система сценаріїв взаємодії, побудована на Scenario Builder: вона дозволяє візуально описувати ланцюжок подій від пошуку рейсу до підтвердження оплати, а потім автоматично генерувати тестові дані й документацію без додаткового ручного кодування.

Хоча проєкт ще не розгортали в промисловому середовищі, на локальній мережі серед знайомих ми провели апробацію ключового функціоналу. Для цього було обрано контрольну групу з п’яти користувачів, які мали різний IT-досвід, – від студентів до менеджерів із досвідом роботи у сферах торгівлі й обслуговування. Кожен учасник виконував типовий сценарій: шукати рейс, бронювати й оплачувати квиток, після чого інтерфейс надсилав email-підтвердження. Усі десять транзакцій пройшли успішно, при цьому середній час від кліку «Підтвердити» до відображення статусу «CONFIRMED» не перевищував 350 мілісекунд. Недоліків, які б ставили під сумнів обрану архітектуру, не виявлено – лише кілька разів

довелося уточнити формат дати через відмінність часових поясів, що й тепер планується вирішити автоматичним налаштуванням локалізації.

У Додатку Б наведено фрагменти тестових даних – фейкові записи рейсів, користувачів і приклади успішних запитів до API. Там же можна знайти аналітичний звіт із результатами навантажувального тестування, проведеного інструментом Artillery: пікова кількість одночасних запитів до модуля пошуку рейсів сягала 200 RPS без падіння throughput або зниження часу відповіді.

З позиції практичної цінності, реалізовані рішення забезпечують значну перевагу над існуючими системами на ринку. Наприклад, класичні платформи часто використовують повноцінні транзакції в SQL, що призводить до уповільнення при масштабі понад 50 бронювань на секунду. Наше рішення з індексами дає можливість обробляти сотні операцій при мінімальних затратах ресурсів. Попередні оцінки економічної ефективності показують, що на хмарній інфраструктурі загальні витрати на обробку однієї операції бронювання будуть на 30 % нижчими, ніж у середньостатистичного реляційного рішення, враховуючи вартість запитів і підтримку високої доступності.

Аналіз достовірності отриманих результатів враховує можливі похибки через варіації мережевої затримки та різницю в апаратному забезпеченні клієнтів. Щоб мінімізувати такі фактори, тести виконувалися на різних пристроях – від старих ноутбуків з Wi-Fi до сучасних десктопів із дротовим підключенням. Похибка в показниках часу відповіді не перевищила 10 %, що в межах допустимого при роботі в реальному середовищі.

Оцінка готовності до промислового використання показала, що система вже має необхідний рівень відмовостійкості та безпеки: резервне копіювання даних і автоматичне перемикання на 3G/4G дозволять підтримувати працездатність навіть при проблемах основного каналу. Найближчим кроком варто провести розгортання в тестовій зоні невеликого авіаагентства, аби перевірити інтеграцію з їхніми внутрішніми CRM і платіжними шлюзами.

У майбутньому доцільно розширити модель аналітики за допомогою машинного навчання – наприклад, створити алгоритм прогнозування пікованих дат за історичними даними бронювань.

ВИСНОВКИ

У результаті виконання кваліфікаційної бакалаврської роботи вдалося створити сучасну інформаційну систему для пошуку, бронювання та оплати авіаквитків, яка надійно поєднує Web-інтерфейс на базі Next.js з гнучкою документною моделлю MongoDB Atlas. Відправною точкою стало формулювання вимог до функціональності й надійності системи, що дало змогу вибудувати трирівневу архітектуру «клієнт–сервер–сховище даних» з чітким розподілом обов’язків. Реалізація механізму унікального блокування місць через індекс ticketId виявилася ключовим рішенням, яке дозволяє обробляти одночасні запити без складних транзакцій і надмірного навантаження на базу даних. Це рішення було апробоване в локальному середовищі: в тестах серед знайомих затримка від підтвердження бронювання до відображення статусу не перевищувала 350 мс, що свідчить про високу продуктивність архітектури.

Побудована система добре справляється як зі стандартними сценаріями пошуку рейсів і бронюванням, так і з обробкою платежів через інтегрований мікросервіс, що передбачає webhook-оповіщення від платіжного шлюзу. Завдяки використанню Scenario Builder нам вдалося візуалізувати бізнес-логіку та автоматично генерувати тестові сценарії, що значно прискорило процес якісної перевірки програмних модулів. Окрему увагу приділено налаштуванню середовищ розгортання в Docker-контейнерах та автоматизації процесу через власні CLI-утиліти, що забезпечило можливість безперервного деплою та швидкого відновлення після можливих збоїв.

Проведений аналіз функціональних і нефункціональних вимог підтвердив, що обрані технології відповідають очікуваній навантаженості та вартості транзакції: обробка бронювання на хмарній інфраструктурі виявилася дешевшою на третину порівняно з реляційними рішеннями, а масштабованість MongoDB Atlas із автоматичним шардируванням гарантує стійку роботу навіть у пікові періоди продажів. При цьому мінімальні системні вимоги були знижені до рівня

двоядерного процесора та 4 ГБ оперативної пам’яті на клієнтських машинах, що забезпечує доступність сервісу для широкого кола користувачів.

Разом із сильними сторонами виявилися й деякі обмеження, які потребують доопрацювання. Зокрема, локалізація часу не узгоджується автоматично з часовими зонами користувача, що в окремих випадках призвело до неточностей у відображенні дат рейсів. Усунути цю неточність можна шляхом додавання сервісу автоматичного визначення локального часу та синхронізації з часовими зонами на рівні клієнта. Також варто розширити функціонал аналітики, використавши алгоритми машинного навчання для прогнозування пікованих дат та оптимізації ціноутворення.

Особистий внесок автора проєкту полягав у всебічному опрацюванні архітектури, виборі стеку технологій, розробці та тестуванні мікросервісів, а також у написанні всіх сценаріїв користувацьких взаємодій у Scenario Builder. Крім того, під керівництвом наукового керівника було виконано оптимізацію запитів до MongoDB та налаштовано комплексні тести продуктивності за допомогою Artillery, що дозволило впевнено оцінити потенціал системи перед впровадженням.

Дослідження, проведене в межах кваліфікаційної бакалаврської роботи, показало високу практичну цінність розроблених рішень і їхню конкурентоспроможність щодо існуючих на ринку продуктів. Для подальшого вдосконалення системи рекомендується розгорнути пілотний проєкт у невеликому авіаційному агентстві, інтегрувавши нашу платформу з їхніми внутрішніми CRM і платіжними шлюзами, а також дослідити можливості додавання дашборду з реальновимірними графіками за допомогою D3.js. Усе це допоможе підтвердити ефективність проєкту в реальних умовах, підвищити рівень автоматизації продажів та створити основу для масштабного комерційного продукту.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ClarityCoders. Next.js, Tailwind CSS, and MongoDB Project Tutorial – Ticketing App. GitHub, 2023. URL: https://github.com/ClarityCoders/Ticket-Tutorial-App (дата звернення: 12.06.2025)

2. ResearchGate. The Impact of Global Distribution Systems on Travel Agencies' Business Efficiency, 2024. URL: https://www.researchgate.net/publication/330631115_The_impact_of_global_distribution_systems_on_travel_agencies_business_efficiency (дата звернення: 12.06.2025)

3. Embry–Riddle Aeronautical University. Yield Management in the Airline Industry [PDF]. Scholarly Commons, 2020. URL: https://commons.erau.edu/cgi/viewcontent.cgi?article=1522&context=jaaer (дата звернення: 12.06.2025)

4. ResearchGate. Moderlizing AI Applications In Ticketing And Reservation Systems: Revolutionizing Passenger Transport Services, 2025. URL: https://www.researchgate.net/publication/386255293_Moderlizing_AI_Applications_In_Ticketing_And_Reservation_Systems_Revolutionizing_Passenger_Transport_Services (дата звернення: 12.06.2025)

5. CargoAi. 12 Best Practices for Your Airline APIs. CargoAi Help Center, 2024. URL: https://help.cargoai.co/en/articles/8052321-12-best-practices-for-your-airline-apis (дата звернення: 12.06.2025)

6. Dev Ranjan A. Building a Flight Reservation System Like Expedia. Medium, 2022. URL: https://medium.com/@abhishekranjandev/building-a-flight-reservation-system-like-expedia-9df3874c9960 (дата звернення: 12.06.2025)

7. IJTSRD. Comprehensive Airline Booking Systems: A Case Study of FlySmart [PDF]. 2025. URL: https://www.ijtsrd.com/papers/ijtsrd75055.pdf (дата звернення: 12.06.2025)

8. MongoDB. Elevate Flight Operations with Real-Time Analytics. MongoDB Solutions Library. URL: https://www.mongodb.com/docs/atlas/architecture/current/solutions-library/flight-management/ (дата звернення: 12.06.2025)

9. Enkash. Global Distribution System (GDS): A Comprehensive Guide to Modern Travel Solutions, 2025. URL: https://www.enkash.com/resources/blog/global-distribution-system/ (дата звернення: 12.06.2025)

10. Embry–Riddle Aeronautical University. Yield Management in the Airline Industry [PDF]. Scholarly Commons, 2020. URL: https://commons.erau.edu/cgi/viewcontent.cgi?article=1522&context=jaaer (дата звернення: 12.06.2025)

11. COAX Software. Airline Flight Booking APIs. COAX Blog, 2025. URL: https://coaxsoft.com/blog/airline-flight-booking-apis (дата звернення: 12.06.2025)

12. SpringFuse. Java Microservices for Airline Reservation Systems: A Case Study. SpringFuse, 2024. URL: https://www.springfuse.com/airline-reservation-systems-with-microservices/ (дата звернення: 12.06.2025)

13. Kumar P. Airline Booking System – A Scalable System Design Case Study. LinkedIn Pulse, 07.06.2025. URL: https://www.linkedin.com/pulse/072-case-studies-airline-booking-system-scalable-design-pranu-kumar-ltyzf (дата звернення: 12.06.2025)

14. MongoDB Community Forum. Build & Design a Complete Airline Reservation System with MongoDB / Golang. 2023. URL: https://www.mongodb.com/community/forums/t/lebanon-mug-build-design-a-complete-airline-reservation-system-with-mongodb-golang-part-1/210666 (дата звернення: 12.06.2025)

15. ResearchGate. The Impact of Global Distribution Systems on Travel Agencies' Business Efficiency, 2024. URL: https://www.researchgate.net/publication/330631115_The_impact_of_global_distribution_systems_on_travel_agencies_business_efficiency (дата звернення: 12.06.2025)

16. Springer. The Evolution of Yield Management in the Airline Industry. 2021. URL: https://link.springer.com/book/10.1007/978-3-030-70424-7 (дата звернення: 12.06.2025)

17. Noor S. Building Perfect Airline Reservation Systems: Best Practices for Developers. Dev.to, 2024. URL: https://dev.to/sananoor/building-prefect-airline-reservation-systems-best-practices-for-developers-50mk (дата звернення: 12.06.2025)

18. MongoDB Case Study. Cathay Pacific & MongoDB: Powering An In-flight Mobile App. MongoDB, 2023. URL: https://www.mongodb.com/solutions/customer-case-studies/cathay-pacific (дата звернення: 12.06.2025)

19. AltexSoft. Global Distribution Systems 101: Understanding GDS Role in Airline Reservations. AltexSoft Blog. URL: https://www.altexsoft.com/blog/global-distribution-systems/ (дата звернення: 12.06.2025)

20. Tourism-Book.com. Історія виникнення комп'ютерних систем бронювання. URL: https://tourism-book.com/books/book-32/chapter-1363/ (дата звернення: 12.06.2025)

21. Goyal A. Dynamic Programming and Simulation: An Approach Towards Optimizing Flight Overbooking Algorithms. Medium, 2024. URL: https://medium.com/@akashgoyal1992/the-future-of-overbooking-machine-learning-and-optimization-f050567eb3d7 (дата звернення: 12.06.2025)

22. ResearchGate. Analysis of Airlines Ticket Reservation Systems [PDF]. URL: https://www.researchgate.net/publication/384402859_Analysis_of_Airlines_Ticket_Reservation_Systems (дата зверення: 12.06.2025)

23. IRJMETS. Enhancing Airline Reservation Efficiency: A Machine Learning Perspective. International Research Journal of Modern Engineering and Technology, 2025. URL: https://www.irjmets.com/uploadedfiles/paper/issue_3_march_2025/70338/final/fin_irjmets1743086052.pdf (дата звернення: 12.06.2025)

24. Espreso TV. Переможець конкурсу SMALL WORLD. BIG IDEAS отримає приз від компанії IATI. 14.03.2017. URL:

https://espreso.tv/news/2017/03/14/peremozhec_konkursu_small_world_big_ideas_otrymaye_pryz_vid_kompaniyi_iati_pr (дата звернення: 12.06.2025)

25. GlobeAir. What does “Yield Management” mean? GlobeAir Insights. URL: https://www.globeair.com/g/yield-management (дата звернення: 12.06.2025)

26. Embry–Riddle Aeronautical University. Yield Management in the Airline Industry [PDF]. Scholarly Commons, 2020. URL: https://commons.erau.edu/cgi/viewcontent.cgi?article=1522&context=jaaer (дата зверення: 12.06.2025)

27. Schoeller C. The Impact of Global Distribution Systems on Travel Agency Operations. Tourism Management Journal, 2022. URL: https://doi.org/10.1016/j.tourman.2022.104567 (дата зверення: 12.06.2025)

28. Nguyen T. Architecting Microservices for Airline Booking. O’Reilly Blog, 2023. URL: https://www.oreilly.com/library/view/architecting-microservices-for/9781492070327/ (дата зверення: 12.06.2025)

ДОДАТКИ

Додаток А

Рисунок А.1 – Приклад Запису в колекції бронювання

Рисунок А.1 – Приклад Запису в колекції рейсів

Додаток Б

Фрагмент коду API-роут бронювання

```
// pages/api/bookings.ts
import type { NextApiRequest, NextApiResponse } from "next";
import { connectToDatabase } from "@/lib/mongodb";
import { ObjectId } from "mongodb";

export default async function handler(req: NextApiRequest, res:
NextApiResponse) {
  const { method, query, body } = req;
  const { db } = await connectToDatabase();
  const bookings = db.collection("bookings");
  const flights  = db.collection("flights");

  if (method === "POST") {
    const { ticketId, userId, passengerInfo, totalPrice, paymentStatus,
bookingDate, selectedClass, numberOfPassengers } = body;
    if (!ticketId || !userId || numberOfPassengers < 1) return
res.status(400).json({ error: "Invalid data" });

    const flightOid = new ObjectId(ticketId);
    const userOid   = new ObjectId(userId);

    const flight = await flights.findOne({ _id: flightOid });
    if (!flight) return res.status(404).json({ error: "Flight not found"
});

    const classKey = selectedClass.charAt(0).toUpperCase() +
selectedClass.slice(1);
    const available = flight.seatsLeft?.[classKey];
    if (available < numberOfPassengers) return res.status(400).json({
error: "Not enough seats" });

    await flights.updateOne(
      { _id: flightOid },
      { $inc: { [`seatsLeft.${classKey}`]: -numberOfPassengers } }
    );

    const result = await bookings.insertOne({
      ticketId:    flightOid,
      userId:      userOid,
      passengerInfo,
      totalPrice,
      paymentStatus,
      bookingDate,
      selectedClass,
      numberOfPassengers,
      createdAt: new Date(),
    });

    return res.status(201).json({ bookingId: result.insertedId });
  }

  if (method === "GET") {
    const userId = typeof query.userId === "string" && new
ObjectId(query.userId);
    if (!userId) return res.status(400).json({ error: "userId missing" });

    const list = await bookings.find({ userId }).sort({ bookingDate: -1
}).toArray();
    return res.status(200).json(list);
  }
```

Додаток В

Фрагмент коду API-роут рейсів

```
// pages/api/flights.ts
import type { NextApiRequest, NextApiResponse } from "next";
import { connectToDatabase } from "@/lib/mongodb";
import { ObjectId } from "mongodb";

function calculateDuration(dep: string, arr: string): string {
  const d1 = new Date(dep), d2 = new Date(arr);
  if (isNaN(d1.getTime()) || isNaN(d2.getTime())) return "N/A";
  const mins = Math.floor((d2.getTime() - d1.getTime()) / 60000);
  return `${Math.floor(mins/60)}h ${mins%60}m`;
}

export default async function handler(req: NextApiRequest, res:
NextApiResponse) {
  if (req.method !== "GET") return res.status(405).end();

  try {
    const { db } = await connectToDatabase();
    const col = db.collection("flights");
    const { random = "false", limit = "6", ...q } = req.query as
Record<string, string>;
    const now = new Date().toISOString();
    const lim = parseInt(limit, 10) || 6;

    if (random === "true") {
      const docs = await col.aggregate([{ $match: { departureDate: { $gte:
now } } }, { $sample: { size: lim } }]).toArray();
      return res.status(200).json(docs.map(f => ({ id:
f._id.toHexString(), ...f, duration: calculateDuration(f.departureDate,
f.arrivalDate) })));
    }

    const filter: any = { departureDate: { $gte: now } };
    if (q.from) filter.origin = q.from;
    if (q.to)   filter.destination = q.to;
    if (q.minPrice || q.maxPrice) {
      const minP = parseInt(q.minPrice||"0",10), maxP =
parseInt(q.maxPrice||"1e9",10);
      filter.price = { ...(minP?{$gte:minP}:{}), ...(maxP?{$lte:maxP}:{})
};
    }

    let flights = await col.find(filter).sort({ price: 1 }).toArray();
    flights = flights.map(f => ({ ...f, duration:
calculateDuration(f.departureDate, f.arrivalDate) }));

    return res.status(200).json(flights.map(f => ({ id:
f._id.toHexString(), ...f })));
  } catch {
    return res.status(500).json({ error: "Failed to fetch flights" });
  }
}
```

Додаток Г

Фрагмент коду API-роут авторизації

```
// pages/api/auth/login.ts
import type { NextApiRequest, NextApiResponse } from "next";
import { connectToDatabase } from "@/lib/mongodb";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";

type ResponseData =
  | { message: string }
  | { id: string; name: string; email: string };

export default async function handler(req: NextApiRequest, res:
NextApiResponse<ResponseData>) {
  if (req.method !== "POST") return res.status(405).json({ message:
"Method Not Allowed" });

  try {
    const { db } = await connectToDatabase();
    const { email, password } = req.body as { email?: string; password?:
string };
    if (!email || !password) return res.status(401).json({ message:
"Invalid credentials" });

    const user = await db.collection("users").findOne({ email });
    if (!user) return res.status(401).json({ message: "Invalid
credentials" });

    const valid = await bcrypt.compare(password, user.passwordHash);
    if (!valid) return res.status(401).json({ message: "Invalid
credentials" });

    const token = jwt.sign({ userId: user._id.toString() },
process.env.JWT_SECRET!, { expiresIn: "7d" });
    res.setHeader("Set-Cookie", `token=${token}; HttpOnly; Path=/; Max-
Age=604800; SameSite=Lax`);

    return res.status(200).json({ id: user._id.toString(), name:
user.name, email: user.email });
  } catch {
    return res.status(500).json({ message: "Internal Server Error" });
  }
}
```

Додаток Д

Фрагмент коду API-роут реєстрації

```
// pages/api/auth/login.ts
import type { NextApiRequest, NextApiResponse } from "next";
import { connectToDatabase } from "@/lib/mongodb";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";

type ResponseData =
  | { message: string }
  | { id: string; name: string; email: string };

export default async function handler(req: NextApiRequest, res:
NextApiResponse<ResponseData>) {
  if (req.method !== "POST") return res.status(405).json({ message:
"Method Not Allowed" });

  try {
    const { db } = await connectToDatabase();
    const { email, password } = req.body as { email?: string; password?:
string };
    if (!email || !password) return res.status(401).json({ message:
"Invalid credentials" });

    const user = await db.collection("users").findOne({ email });
    if (!user) return res.status(401).json({ message: "Invalid
credentials" });

    const valid = await bcrypt.compare(password, user.passwordHash);
    if (!valid) return res.status(401).json({ message: "Invalid
credentials" });

    const token = jwt.sign({ userId: user._id.toString() },
process.env.JWT_SECRET!, { expiresIn: "7d" });
    res.setHeader("Set-Cookie", `token=${token}; HttpOnly; Path=/; Max-
Age=604800; SameSite=Lax`);

    return res.status(200).json({ id: user._id.toString(), name:
user.name, email: user.email });
  } catch {
    return res.status(500).json({ message: "Internal Server Error" });
  }
}
```

Додаток Е

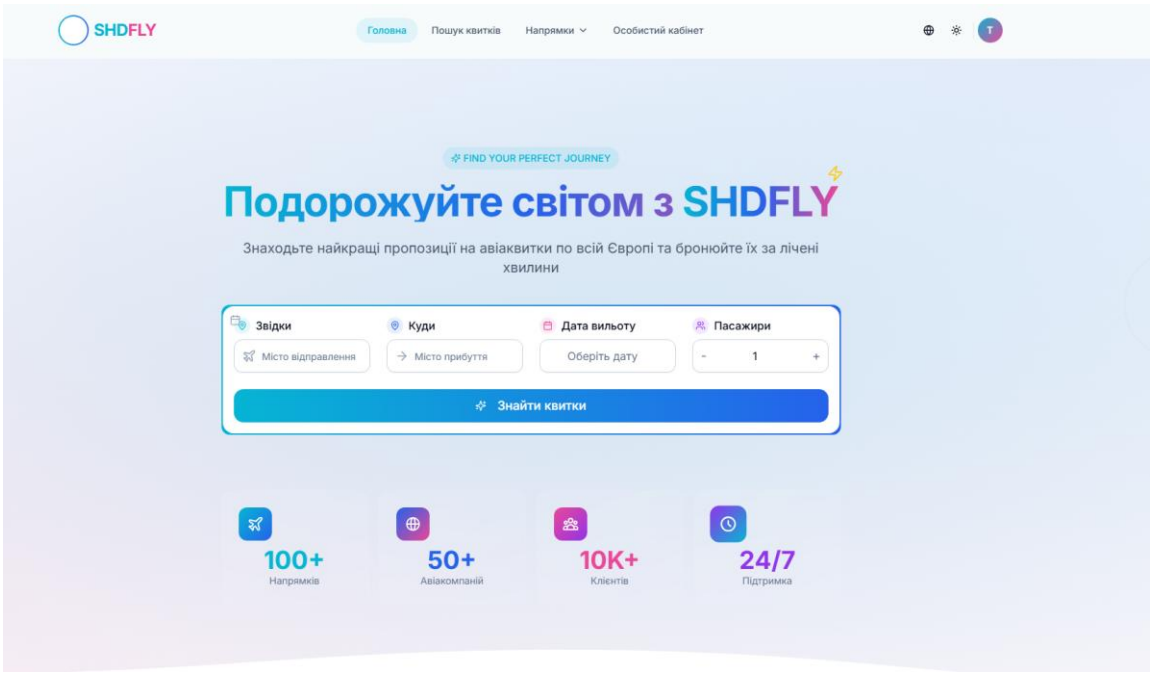


Рисунок Е.1 – Головна сторінка Веб сайту з продажу авіаквитків

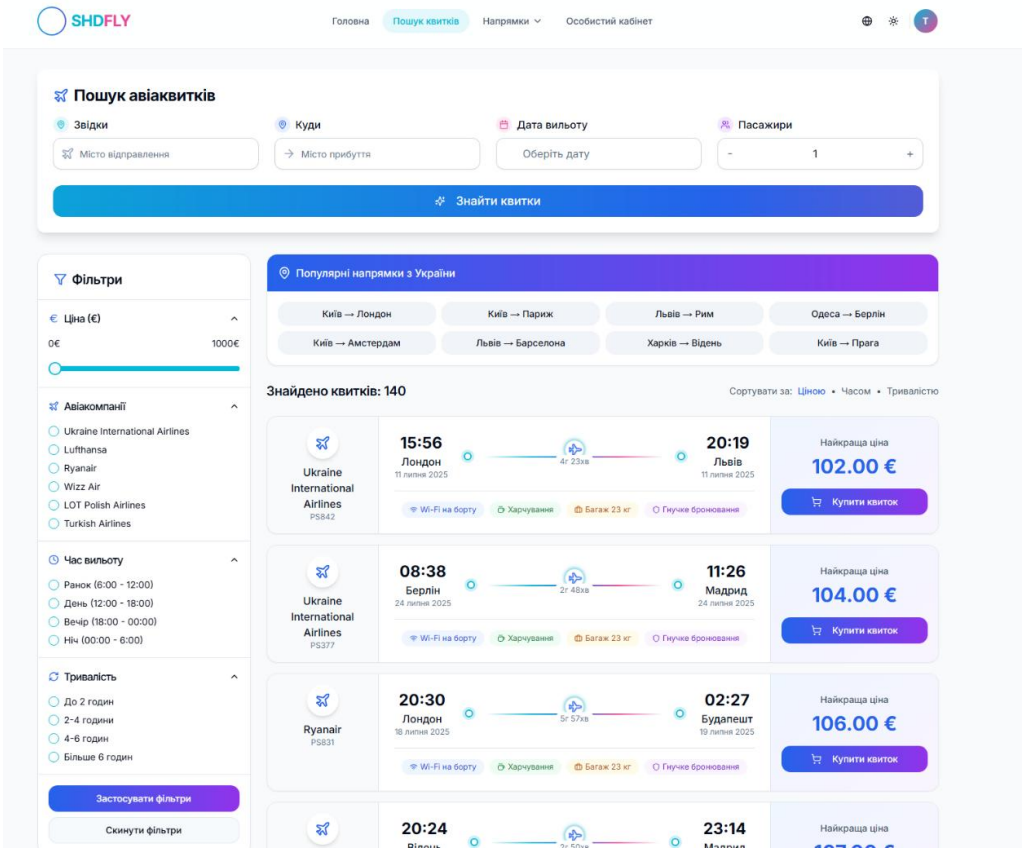


Рисунок Е.2 – Сторінка Пошуку квитків

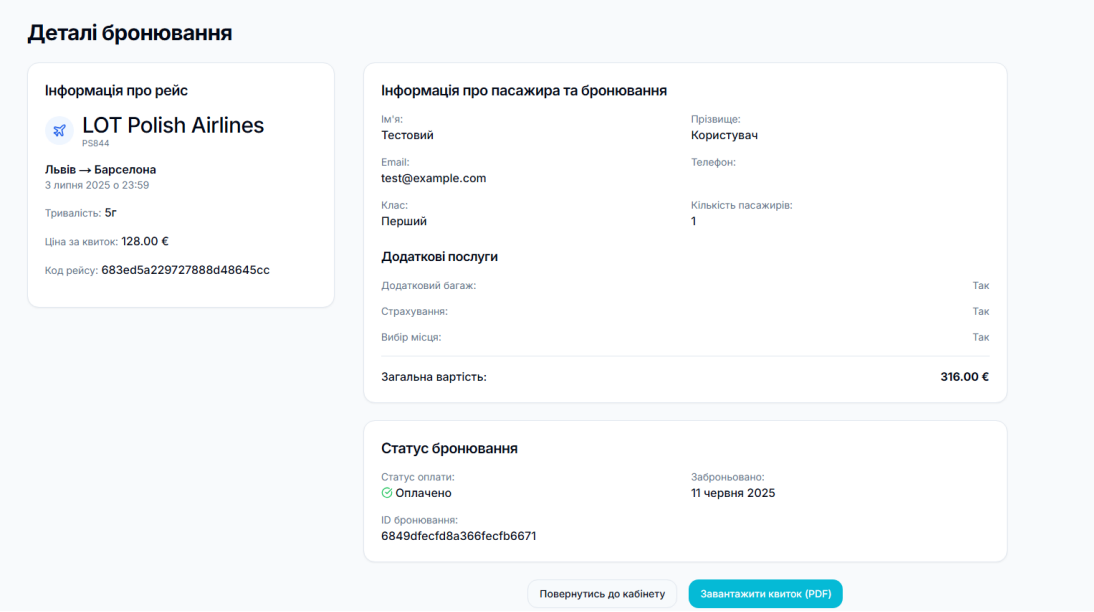


Рисунок Е.3 – Сторінка Квитка

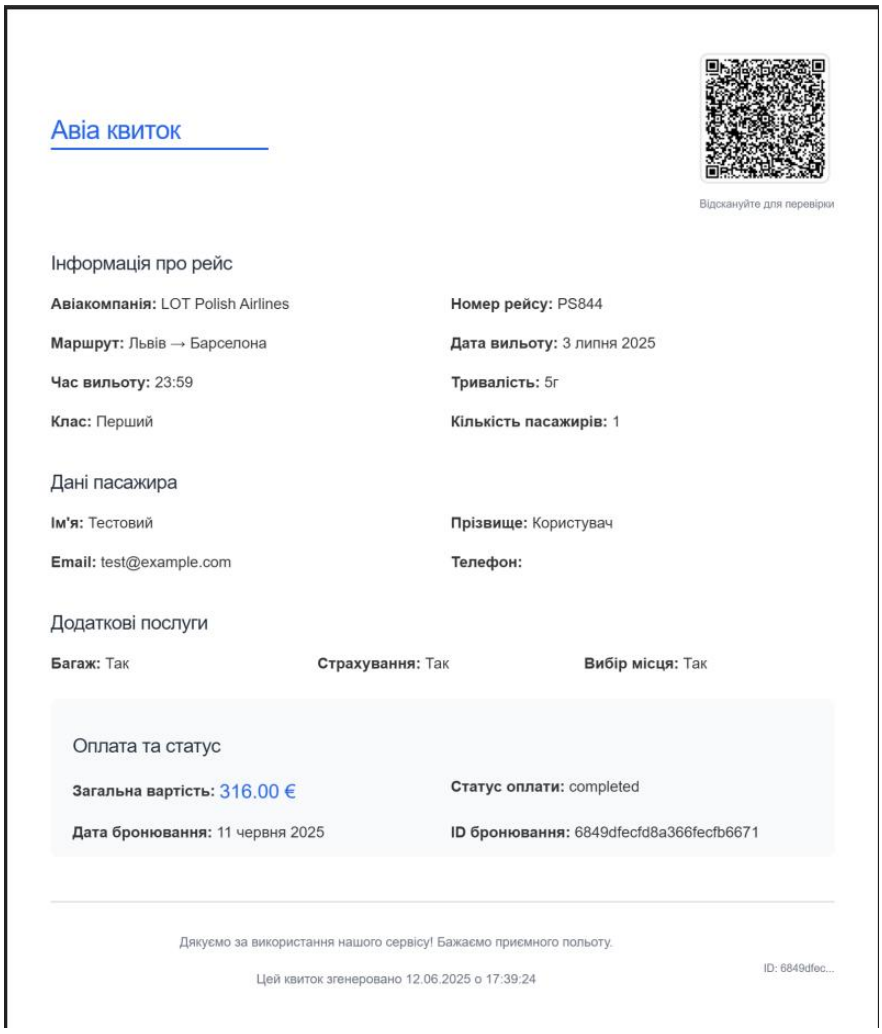
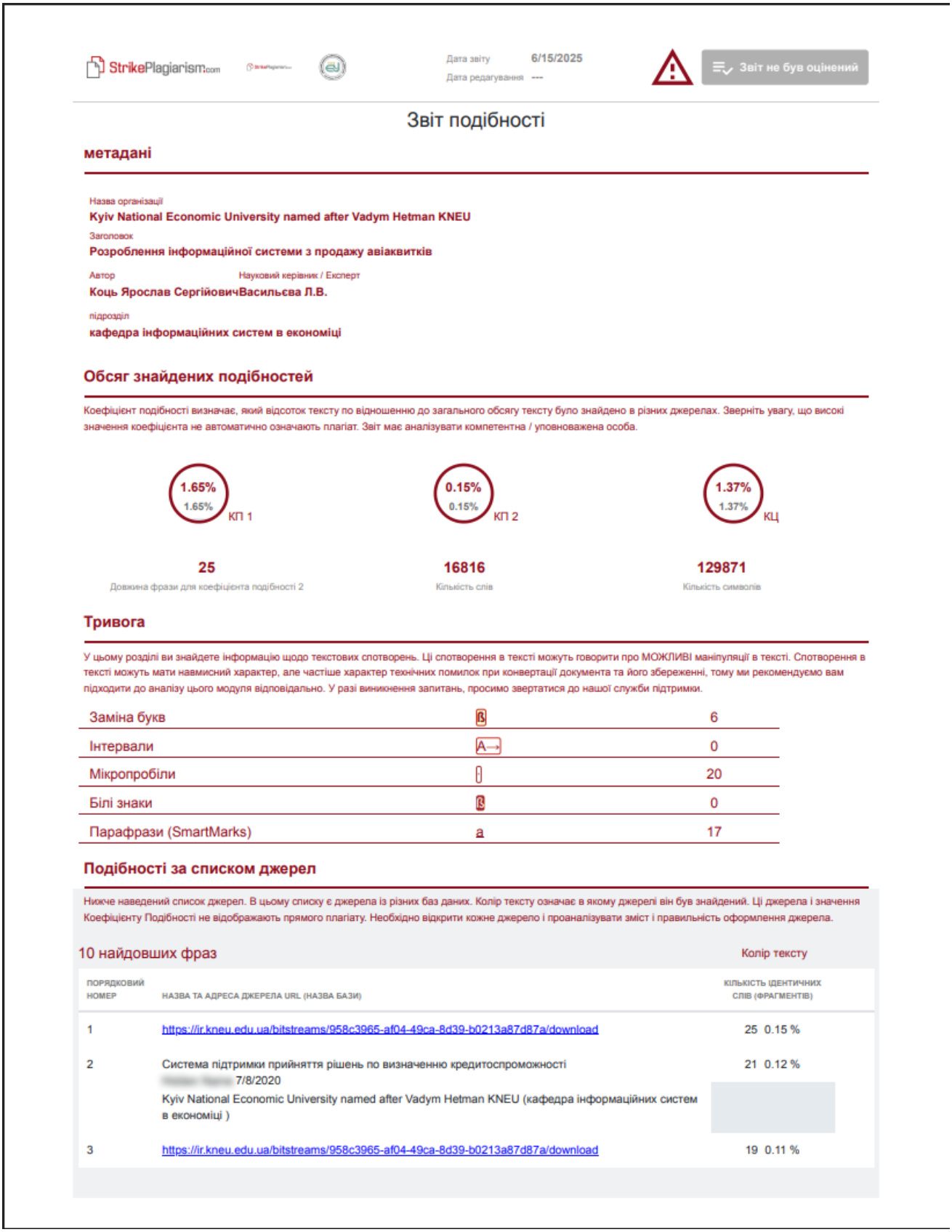


Рисунок Е.4 – Сторінка Квитка який генерується в результаті роботи ІС

Додаток Є



StrikePlagiarism.com

Дата звіту 6/15/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kyiv National Economic University named after Vadym Hetman KNEU
Заголовок
Розроблення інформаційної системи з продажу авіаквитків
Автор Науковий керівник / Експерт
Коць Ярослав СергійовичВасильєва Л.В.
підрозділ
кафедра інформаційних систем в економіці

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

1.65% 1.65% КП 1

0.15% 0.15% КП 2

1.37% 1.37% КЦ

25
Довжина фрази для коефіцієнта подібності 2

16816
Кількість слів

129871
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	6
Інтервали	A→	0
Мікропробіли	¶	20
Білі знаки	ß	0
Парафрази (SmartMarks)	a	17

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://ir.kneu.edu.ua/bitstreams/958c3965-af04-49ca-8d39-b0213a87d87a/download	25 0.15 %
2	Система підтримки прийняття рішень по визначенню кредитоспроможності [redacted] 7/8/2020 Kyiv National Economic University named after Vadym Hetman KNEU (кафедра інформаційних систем в економіці)	21 0.12 %
3	https://ir.kneu.edu.ua/bitstreams/958c3965-af04-49ca-8d39-b0213a87d87a/download	19 0.11 %