МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

Катерина НЕСТЕРЕНКО
“____”_____2023 р.

КВАЛІФІКАЦІЙН РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСНИКА ОСВІТНЬОГО СТУПЕНЯ
МАГІСТР

Тема: “Система для створення, редагування та збору аналітики про польоти для авіакомпаній”

Виконавець: Жерибор Іван Віталійович

Керівник: к.т.н. доцент Борковська Любов Олексіївна

Нормоконтролер: Кравченко Ольга Сергіївна

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки та програмної інженерії
Кафедра інженерії програмного забезпечення
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Катерина НЕСТЕРЕНКО
"___" _____ 2023 р

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Жерибора Івана Віталійовича

1. Тема кваліфікаційної роботи: «Система для створення, редагування та збору аналітики про польоти для авіакомпаній»
затверджена наказом ректора від 29.09.2023 р. № 1994/ст.

2. Термін виконання проекту: з 02.10.2023 р. по 31.12.2023 р.

3. Вихідні данні до роботи: графічний інтерфейс користувача розроблена за допомогою бібліотеки React з використанням мови програмування TypeScript, серверна частина, яка була розроблена за допомогою платформи ASP.NET Core використовуючи мову програмування – C#.

4. Зміст пояснювальної записки:
1. Аналіз проблеми створення, редагування та збору аналітики про польоти для авіакомпаній.
2. Огляд алгоритмів розпізнавання та класифікації зображень користувацьких інтерфейсів.
3. Реалізація системи для створення, редагування та збору аналітики про польоти для авіакомпаній

5. Перелік обов'язкових слайдів презентації:
1. Аналіз доменної області.
2. Опис використаних технологій.
3. Огляд скріншотів інтерфейсу готового продукту.

6. Календарний план-графік

№ пор	Завдання	Термін виконання	Відмітка про виконання
1.	Розробка та затвердження графіка роботи	25.10.2023 – 31.10.2023	
2.	Написання 1 розділу, представлення керівнику	1.11.2023 – 15.11.2023	
3.	Написання 2 розділу, представлення керівнику	15.11.2023 – 25.11.2023	
4.	Написання 3 розділу, представлення керівнику	26.11.2023 – 01.12.2023	
6.	Загальне редагування та друк пояснювальної записки, графічного матеріалу	01.12.2023 – 09.12.2023	
7.	Проходження нормо-контролю, перевірка на антиплагіат, перепліт пояснювальної записки.	11.12.2023 – 15.12.2023	
8.	Розробка тексту доповіді. Оформлення графічного матеріалу для презентації	15.12.2023 – 20.12.2023	
9.	Отримання відгуку керівника, рецензії.	20.12.2023	
10.	Підготовка матеріалів для передачі секретарю ДЕК (ПЗ, CD-R з електронними копіями ПЗ, презентації, відгук керівника, рецензія) в папці	21.12.2023	
11	Захист дипломної роботи перед ЕК.	21.12.2023 – 27.12.2023	

Дата видачі завдання 02.10.2023

Керівник дипломної роботи: к.т.н. доцент Любов БОРКОВСЬКА

Завдання прийняв до виконання: Іван ЖЕРИБОР

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Система для створення, редагування та збору аналітики про польоти для авіакомпаній»: 40 с., 15 рис., 8 табл., 4 інформаційних джерел.

СИСТЕМА ДЛЯ СТВОРЕННЯ, РЕДАГУВАННЯ ТА ЗБОРУ АНАЛІТИКИ ПРО ПОЛЬОТИ ДЛЯ АВІАКОМПАНІЙ

Об’єкт розробки – Система для створення, редагування та збору аналітики про польоти для авіакомпаній.

Мета дипломної роботи – підвищити доступність збору аналітики про польоти для авіакомпаній.

Метод дослідження – розробка системи для створення, редагування та збору аналітики про польоти для авіакомпаній.

Результати роботи можуть бути використані при розробці системи для створення, редагування та збору аналітики про польоти для авіакомпаній в галузях авіаційних перевезень.

Розробка та дослідження проводилися під управлінням ОС Windows 11. Розробка програми проводилася у середовищі JetBrains Rider C#, на мові програмування C#.

ABSTRACT

Explanatory note to the diploma project "System for creating, editing and collecting analytics about flights for airlines": 40 p., 15 figures, 8 tables, 4 information sources.

SYSTEM FOR CREATING, EDITING AND COLLECTING FLIGHT ANALYTICS FOR AIRLINES

Property development – a system for creating, editing and collecting flight analytics for airlines.

Purpose – to increase the availability of flight analytics collection for airlines.

Research method – the development of a system for creating, editing and collecting flight analytics for airlines.

The results of the work can be used in the development of a system for creating, editing and collecting analytics about flights for airlines in the air transportation industry.

The development and research was carried out under Windows 11 operating system. The development of the program was carried out in the JetBrains Rider C# environment, in the C# programming language.

ЗМІСТ

ПЕРЕЛІК ПРИЙНЯТИХ СКОРОЧЕНЬ

КІ – користувацький інтерфейс.

FE – front-end.

BE – back-end.

БД – база даних.

ВСТУП

Перед початком вибору теми для дипломної роботи спочатку потрібно було проаналізувати область, у сфері якої буде проводитись розробка програмного забезпечення. У випадку цієї дипломної роботи – це галузь авіаперельотів та авіакомпаній.

Серед основних трендів розвитку транспортних систем наразі є розширення діяльності національних та міжнародних авіаційних мереж для того, щоб підвищити ефективність всіх видів перевезень та раціонального забезпечення наявних потреб у економіки і населення. Наразі світова економіка має мережеву модель, яка ґрунтується на глобальних фінансових, економічних, виробничих та управлінських інститутах і має на увазі об’єднання в спільні структури різної конфігурації як національних так і міжнародних установ, при цьому стаючи головною і за рахунок постійного вдосконалення різних елементів інформаційної та інноваційної економіки та активного розвитку комунікаційних систем та їх підтримки створює необхідні передумови для ефективного подолання всіх існуючих глобальних загроз. Нинішні тенденції розвитку авіаційних систем (кількість пасажирських перевезень у 2019 р. збільшилася на 44% порівняно з 2016 р., вантажів – на 22%) впливають на зміни типових поглядів на їх значимість у процесах формування світового продукту, задоволення наявних потреб, сприянню конкурентних переваг. Значно більшої уваги потребує розвиток та розширення європейської авіаційної системи, на яку, наприклад, у 2018 році приходилося понад 35% всіх світових пасажирських перевезень та близько 8% вантажних перевезень. Створення єдиного європейського повітряного простору, який також включає в себе створення крупних міжнародних авіа хабів, удосконалення європейського економічного середовища, поглиблення євроінтеграційної стратегії Україною вимагають постійних досліджень, сфокусованих на пошуку тенденцій розвитку та розширення діяльності європейської авіаційної системи, обґрунтування

засобів збільшення її ефективності, визначення результативності і доцільності цих процесів в галузі авіації.

Попит на компанії, які розробляють ІТ-рішення для авіаційної галузі, може бути високим через кілька ключових факторів:

1. Потреба в оптимізації операцій – авіакомпанії шукають рішення для оптимізації своїх операцій, включаючи планування маршрутів, управління ресурсами та підтримку обслуговування повітряних суден.

2. Посилення конкуренції – збільшення конкуренції в авіаційній галузі заохочує авіакомпанії шукати інноваційні технології для підвищення ефективності та привертання клієнтів.

3. Підвищення клієнтської вимогливості – пасажири очікують високого рівня сервісу та зручності при плануванні та виконанні подорожей, що ставить виклик перед авіакомпаніями впроваджувати нові технології.

4. Безпека та регулювання - зростаючі вимоги до безпеки та дотримання регуляторних стандартів створюють потребу в технологіях, які допомагають у вирішенні цих завдань.

5. Сталий розвиток – зростає усвідомлення екологічних проблем, і авіакомпанії шукають технологічні рішення для зменшення впливу своєї діяльності на навколишнє середовище.

6. Використання аналітики та штучного інтелекту – попит на розвинені аналітичні та ШІ-рішення зростає, оскільки компанії хочуть здати та аналізувати великі обсяги даних для прийняття стратегічних рішень.

7. Глобальна природа авіації – сприяння міжнародних подорожей та глобальна природа авіації робить технології для авіаційної галузі особливо важливими та потрібними.

Отже, у світі існує попит на ІТ-рішення, які можуть надавати інноваційні рішення для авіаційної галузі, особливо ті, що спрямовані на підвищення ефективності, безпеки та комфорту пасажирів.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМИ СТВОРЕННЯ, РЕДАГУВАННЯ ТА ЗБОРУ АНАЛІТИКИ ПРО ПОЛЬОТИ ДЛЯ АВІАКОМПАНІЙ

1.1. Аналіз створення, редагування та збору аналітики про польоти для авіакомпаній

Цивільна авіація є одним із найбільш дорогих видів транспорту, тому великі обсяги авіаперевезень притаманні країнам з високим економічним розвитком. Так, 37% від загального обсягу перевезень пасажирів на міжнародних і внутрішніх регулярних лініях було виконано авіакомпаніями Азії – Тихого океану, європейськими авіакомпаніями – 26%, авіакомпаніями Північної Америки – 23%, Латинської Америки та Карибського басейну – 8%, Близького Сходу – 4%, а частка авіакомпаній Африки склала майже 2%.

Система для створення, редагування та збору аналітики про авіаперельоти розв'язує ряд важливих завдань та виконує функції, що дозволяють авіакомпаніям та іншим учасникам авіаційного сектору ефективно управляти та аналізувати інформацію. Ця система покращить документообіг для кожної авіакомпанії, а також покращить вирішування проблем, які виникали під час рейсі.

Після аналізу доменної області авіаперельотів було визначена низку питань, які будуть вирішуватись системою створення, редагування та збору аналітики про польоти для авіакомпаній. Ось кілька ключових деталей та завдань, які може розв'язувати така система:

1. Бронювання та управління рейсами:

- Система дозволяє здійснювати бронювання рейсів, управляти розкладом та маршрутами. Вона повинна слідкувати за доступністю місць та ефективно планувати рейси.

2. Системи реєстрації та обслуговування пасажирів:

- Забезпечує функції онлайн та офлайн реєстрації пасажирів, а також здійснює управління обслуговуванням пасажирів під час польоту.

3. Моніторинг та відстеження польотів:

- Забезпечує систему для моніторингу та відстеження реального часу руху літаків, включаючи інформацію про місцезнаходження, шлях польоту, висоту і швидкість.

4. Збір та аналіз даних безпеки:

- Здійснює збір та аналіз даних, пов'язаних з безпекою польотів, щоб виявляти потенційні ризики та забезпечити високий стандарт безпеки.

5. Системи Інтеграції з Іншими Авіаційними Сервісами:

- Взаємодіє з іншими авіаційними сервісами, такими як системи бронювання, служби авіа-навігації та контролю, для забезпечення повноцінної інтеграції та обміну даними.

6. Системи Збору та Обробки Даних Пасажирів:

- Забезпечує збір та обробку даних пасажирів для забезпечення комфортного та безпечного перельоту, а також для аналізу вимог та зручностей пасажирів.

7. Аналітика та Звітність:

- Надає засоби для аналізу даних та створення звітів. Це може включати в себе візуалізацію даних, виявлення тенденцій, попередження про можливі проблеми та прийняття управлінських рішень.

8. Електронний Документообіг та Автоматизація:

- Автоматизує процеси створення та обробки документів, пов'язаних з польотами, включаючи електронні квитки, дозвільні документи та інші важливі документи.

9. Системи Забезпечення Доступу та Конфіденційності:

- Забезпечує надійний захист даних та забезпечення конфіденційності, використовуючи механізми шифрування та системи керування доступом.

Ці функції спрямовані на поліпшення ефективності операцій, безпеки польотів, зручності для пасажирів та прийняття інформованих управлінських рішень в авіаційній галузі.

Однак більш логічно та правильно було б створювати більш вузько направлені системи для вирішення кожної з проблем. Бо кожна із проблем заслуговує на вирішення командою професіоналів, яка має досвід роботи у окремій частині домену або просто не буде відволікатись на інші частини цього домену.

Розробка сервісів у більш вузькому домені, відомому також як "нишева розробка", має кілька переваг порівняно з розробкою у більш загальному або універсальному контексті. Ось деякі з основних переваг:

- глибше розуміння клієнтських потреб: робота у вузькому домені дозволяє команді розробників глибше розуміти специфіку потреб та вимог користувачів у цьому конкретному сегменті ринку. Вони можуть краще адаптувати свої продукти до унікальних вимог та викликів цього ринку;
- швидше використання та Ринкова Зрілість: розробка у вузькому домені може дозволити швидше використання продукту на ринок, оскільки конкретна спеціалізація дозволяє уникнути складнощів, пов'язаних з великою кількістю можливих варіантів та конфігурацій;
- можливість вирішення конкретних проблем: фокус на вузькому домені дозволяє команді розробників більше концентруватися на вирішенні конкретних проблем або викликів, з якими стикаються користувачі в цьому сегменті. Це сприяє створенню продуктів, які є більш ефективними та відповідають реальним потребам;
- легше встановлення авторитету: робота в конкретному сегменті може дозволити команді розробників стати експертами у своїй галузі. Вони можуть отримати визнання та довіру спільноти, що допомагає позиціонувати їх як ключових гравців в цьому сегменті ринку;
- зменшення конкуренції: вузький домен може вигідно впливати на конкурентоспроможність, оскільки конкуренція може бути меншою

порівняно з більш загальними ринковими сегментами. Це дозволяє продукту легше знаходити своє місце та вибиватися серед інших гравців;

- ефективне використання ресурсів: робота у вузькому домені може зменшити кількість необхідних ресурсів для розробки та підтримки продукту. Замість того, щоб вирішувати широкий спектр завдань, команда може фокусуватися на конкретних аспектах своєї галузі;
- збільшення шансів на успіх: працюючи в вузькому домені, розробники можуть стати основними гравцями у своїй галузі, що збільшує їх шанси на успіх та визнання на ринку.

Звісно, важливо збалансувати фокус на вузькому домені з потребами ринку та забезпечити, щоб продукт все ще відповідав зростаючим вимогам користувачів.

Тому було обрано вирішувати лише частину із проблем, а саме моніторинг та відстеження польотів, аналітика та звітність.

1.2. Аналіз існуючих застосунків

Проте, в даній галузі існують певні типові рішення та системи, які можуть використовуватися авіакомпаніями, аеропортами та іншими гравцями авіаційного сектору. Деякі приклади включають:

Amadeus Altéa Suite

Amadeus Altéa — це інтегрована платформа для авіакомпаній, яка включає в себе рішення для бронювання, обслуговування пасажирів, управління рейсами та інші функції (сайт на рис. 1.1). Вона дозволяє авіакомпаніям ефективно управляти всіма аспектами їх бізнесу.

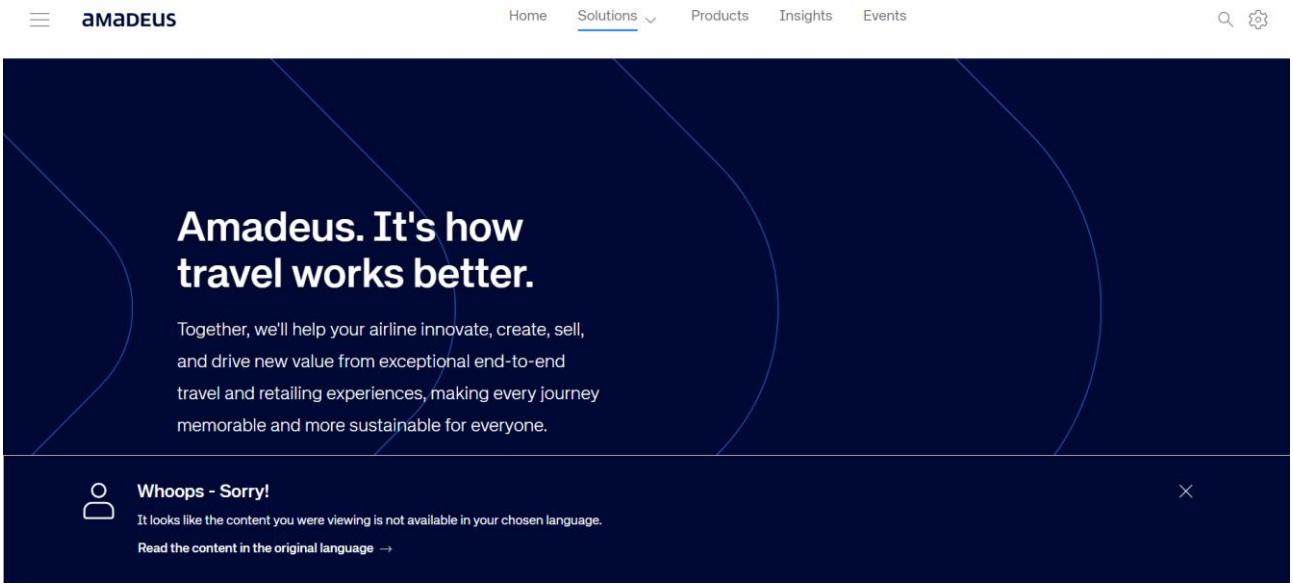

Рис. 1.1. Amadeus Altéa офіційний сайт

Amadeus Altéa Suite є інтегрованою платформою для авіакомпаній, яка включає в себе різноманітні рішення для ефективного управління різними аспектами авіаційного бізнесу. Ця платформа розроблена компанією Amadeus, однією з провідних у світі у сфері технологій для туризму та авіації. Основні компоненти Amadeus Altéa Suite включають такі:

- Amadeus Altéa Reservation: цей компонент дозволяє авіакомпаніям керувати системами бронювання та обслуговування пасажирів. Він включає в себе інтерфейс для онлайн-бронювання, управління маршрутами, продаж квитків, а також роботу з лояльністю пасажирів.
- Amadeus Altéa Inventory: цей модуль відповідає за управління інвентарем авіакомпанії, включаючи маршрути, доступність місць на рейсах, тарифи та інші аспекти, пов'язані з розкладом.
- Amadeus Altéa Departure Control: даний компонент розроблений для оптимізації процесу відправлення рейсів. Він включає в себе функції реєстрації пасажирів, обробки багажу та забезпечення відправлення літаків за графіком.
- Amadeus Altéa Flight Management: цей модуль забезпечує управління рейсами та оперативною інформацією під час польотів.

Він може включати в себе інструменти для планування маршрутів, взаємодії з екіпажем та підтримки при прийнятті рішень під час польоту.

- Amadeus Altéa Customer Management: цей компонент призначений для управління взаємодією авіакомпанії з пасажирами. Він включає в себе інструменти для реалізації програм лояльності, обслуговування пасажирів та створення персоналізованих послуг.
- Amadeus Altéa Network Revenue Management: модуль відповідає за оптимізацію доходів авіакомпанії. Він аналізує дані та розробляє стратегії тарифів для максимізації прибутку в рамках мережі рейсів.

Amadeus Altéa Suite спрямована на те, щоб забезпечити авіакомпаніям інтегрований та ефективний підхід до управління усіма аспектами їх діяльності. Вона дозволяє авіакомпаніям оптимізувати свої операції, покращувати обслуговування пасажирів та ефективніше конкурувати на ринку авіаційних послуг.

Sabre AirVision

Sabre AirVision — це серія продуктів від компанії Sabre, яка надає рішення для управління рейсами, аналітики, оптимізації маршрутів та інших важливих аспектів авіаційної діяльності. Їх сайт представлено на рис. 1.2.

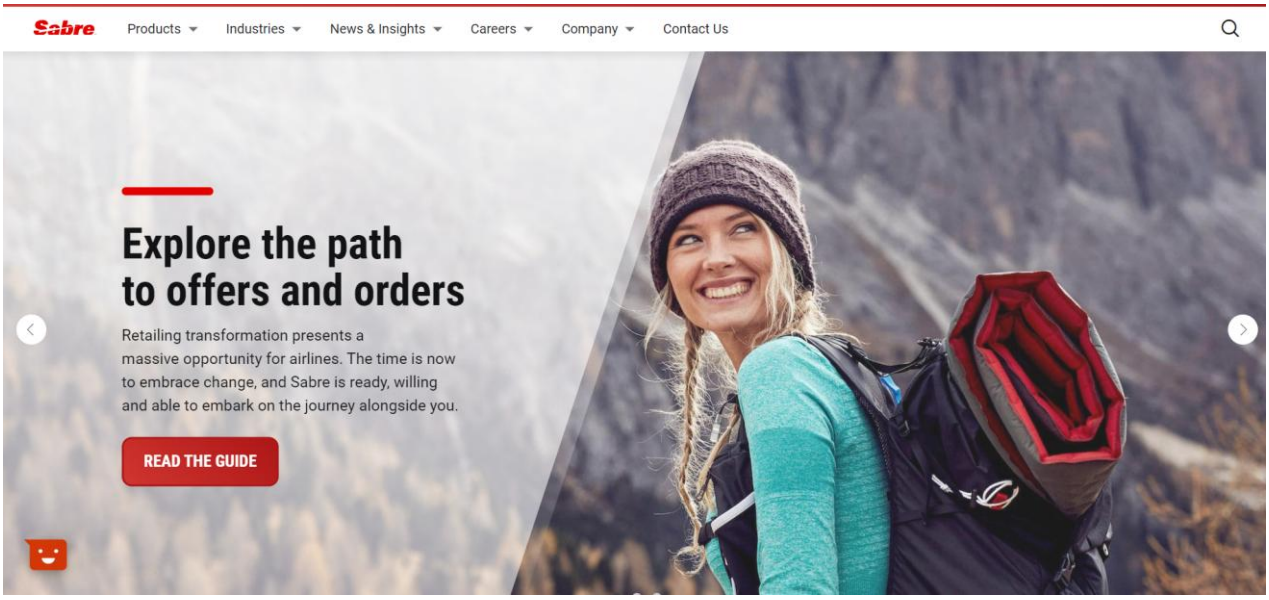

Рис. 1.2. Sabre AirVision офіційний сайт

Основні напрямки діяльності компанії включають одну із найбільших світових систем глобального розподілу, що дозволяє туристичним агентствам та іншим користувачам в галузі бронювати авіабілети, готелі, автомобілі та інші послуги. Їхні технології використовуються багатьма агентствами для забезпечення доступу до широкого спектру туристичних продуктів.

Крім авіаційного сектору, Sabre також надає рішення для інших сегментів туризму, включаючи готелі, оренду автомобілів, круїзи та інші послуги. Їхні технології допомагають у взаємодії між різними постачальниками послуг та туристичними агентствами.

Компанія надає рішення для авіакомпаній, такі як системи управління резервуваннями, планування рейсів, управління ресурсами та інші технології, спрямовані на покращення ефективності операцій та збільшення доходів.

Також компанія працює у галузі гостинності, надаючи рішення для готелів та інших гостьових установ. Ці рішення включають в себе системи бронювання, управління резервуваннями та інші технології для оптимізації управління готелями.

Sabre використовує дані та аналітику для надання своїм клієнтам інформованих стратегічних рішень. Їхні аналітичні інструменти допомагають бізнесам в галузі туризму вивчати тенденції, розуміти попит та удосконалювати свої послуги.

Компанія вміло розробляє мобільні застосунки та рішення для полегшення подорожей для кінцевих користувачів. Це може включати в себе мобільні додатки для бронювання, інформаційні сервіси та інші зручності для мандрівників.

Своєю діяльністю Sabre спрямовується на покращення ефективності та конкурентоспроможності гравців в галузі туризму та транспорту, надаючи їм інноваційні технології та послуги.

Sabre AirVision - це серія продуктів та рішень, розроблених компанією Sabre Corporation для авіакомпаній з метою оптимізації управління різними аспектами їхньої діяльності. Ці продукти призначені для забезпечення

авіакомпаніям інструментів та аналітичних рішень, які допомагають в оптимізації різних операційних процесів та прийнятті стратегічних рішень. Основні компоненти Sabre AirVision включають:

- Sabre AirVision Planning and Scheduling: цей продукт допомагає авіакомпаніям в оптимізації процесів планування та розкладу рейсів. Він враховує різні фактори, такі як попит на рейси, ефективність використання літаків, конкуренція та інші, для створення оптимального розкладу.
- Sabre AirVision Fleet Management: цей компонент допомагає в управлінні флотом літаків авіакомпанії. Він включає в себе інструменти для моніторингу технічного стану літаків, оптимізації розподілу літаків між рейсами та планування технічного обслуговування.
- Sabre AirVision Passenger Service System (PSS): цей компонент включає рішення для управління обслуговуванням пасажирів, бронюванням та емісією квитків. Він спрямований на забезпечення зручності для пасажирів та оптимізацію обслуговування.
- Sabre AirVision Revenue Management (рис. 1.3): цей продукт допомагає авіакомпаніям управляти тарифами та доходами. Він аналізує попит на рейси та розробляє стратегії ціноутворення для максимізації доходів.

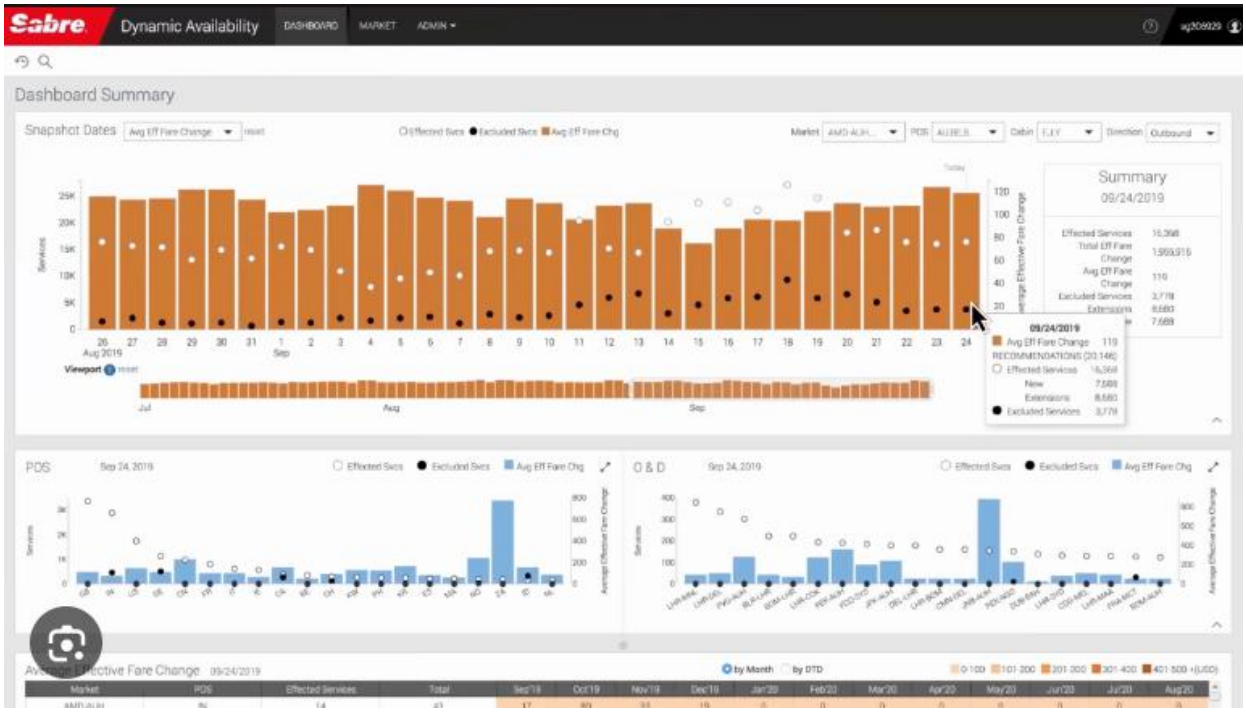

Рис. 1.3. Sabre Revenue Optimizer

- Sabre AirVision Commercial Analytics: цей компонент надає інструменти для аналізу даних та виявлення тенденцій в різних аспектах бізнесу авіакомпаній, таких як продажі, попит, тарифи та інші.
- Sabre AirVision Operations: цей продукт допомагає в управлінні різними операційними аспектами авіакомпанії, включаючи моніторинг польотів, роботу з екіпажем та інші аспекти операцій.

Загалом, Sabre AirVision допомагає авіакомпаніям покращити ефективність своїх операцій, взаємодіяти з пасажирами, ефективно використовувати свій флот, оптимізувати доходи та приймати інформовані стратегічні рішення.

Lufthansa Systems NetLine/Plan

NetLine/Plan — це рішення для оптимізації маршрутів та планування рейсів від компанії Lufthansa Systems. Воно допомагає авіакомпаніям ефективно розподіляти ресурси та оптимізувати їх мережу рейсів. Офіційний сайт на рис. 1.4.

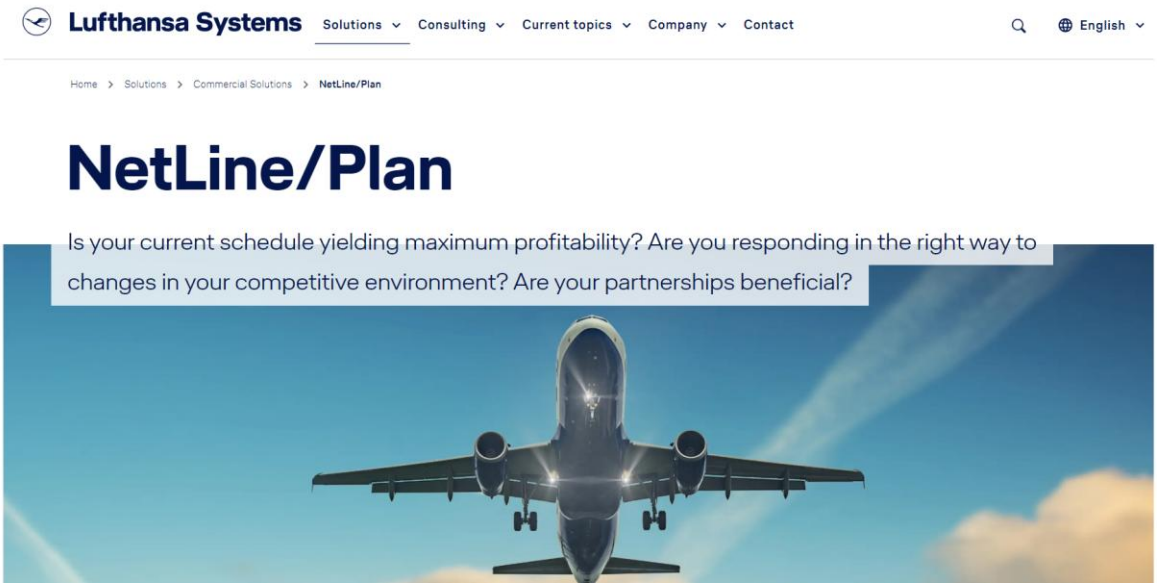

Рис. 1.4. Офіційний сайт Lufthansa Systems NetLine/Plan

Lufthansa Systems є визнаним лідером у сфері інформаційних технологій для авіаційної індустрії та представляє собою ключового гравця в складі Lufthansa Group, однієї з найбільших авіаційних компаній у світі. Заснована в 1995 році, компанія здобула репутацію завдяки своєму інноваційному підходу та високоякісним рішенням для оптимізації операцій авіакомпаній.

Портфель продуктів Lufthansa Systems широкий і охоплює всі ключові аспекти авіаційного бізнесу. Одним із ключових продуктів є Lufthansa Systems NetLine/Plan, який спеціалізується на плануванні рейсів та оптимізації використання ресурсів. Це дозволяє авіакомпаніям ефективно розподіляти літаки, управляти резервуваннями та мінімізувати витрати.

Однією з ключових переваг Lufthansa Systems є їхня глобальна присутність і обслуговування клієнтів по всьому світі. Компанія партнерствує з різними авіаційними суб'єктами, включаючи авіакомпанії, аеропорти та туроператорів.

Lufthansa Systems активно вкладає в дослідження та розвиток, спрямовуючи свою увагу на інновації в галузі технологій. Висока експертиза компанії в галузі безпеки та захисту даних є важливим аспектом, особливо в контексті обробки конфіденційної інформації в авіаційній сфері.

Загальна спрямованість Lufthansa Systems полягає в тому, щоб бути надійним та інноваційним партнером для авіаційної галузі, допомагаючи своїм клієнтам вдосконалювати операційні процеси, оптимізувати використання ресурсів та досягати стратегічних цілей.

Lufthansa Systems є провідним постачальником інформаційних технологій та рішень для авіаційної індустрії. Одним з їхніх ключових продуктів для авіакомпаній є Lufthansa Systems NetLine/Plan. Давайте розглянемо цей продукт більш детально:

1. Планування Рейсів:

- NetLine/Plan спеціалізується на оптимізації та автоматизації процесів планування рейсів. Він дозволяє авіакомпаніям розробляти ефективні розклади рейсів, враховуючи різноманітні обмеження і умови, такі як наявність літаків, аеропортові обмеження, попит на рейси та інші фактори.

2.Оптимізація роботи флоту:

- Система допомагає авіакомпаніям ефективно розподіляти літаки між рейсами та мінімізувати час простою літаків. Це дозволяє оптимізувати використання флоту та зменшувати витрати.

3. Управління ресурсами:

- NetLine/Plan включає інструменти для ефективного управління різними ресурсами, такими як екіпажі, пальне, обслуговуючий персонал та інші ресурси, що впливають на операційні витрати авіакомпанії.

4. Моніторинг та оптимізація маршрутів:

- Система дозволяє автоматично моніторити роботу рейсів та, при необхідності, вносити зміни в маршрути для оптимізації польотів та ресурсів.

5. Аналітика та звітність:

- NetLine/Plan надає інструменти для аналізу даних про рейси та вирішення задач звітності. Це допомагає авіакомпаніям здійснювати інформовані стратегічні рішення та вдосконалювати їх операційну ефективність.

6. Підтримка прийняття рішень:

- Засоби прийняття рішень у системі допомагають авіакомпаніям робити оптимальні вирішення в умовах змінного середовища та ринкової конкуренції.

Lufthansa Systems NetLine/Plan спрямований на оптимізацію операцій авіакомпаній, забезпечуючи їм ефективне використання ресурсів та досягання стратегічних цілей.

FlightAware

FlightAware – це технологічна компанія, що спеціалізується на наданні інформації та послуг для відстеження польотів та вивчення авіаційних даних. Заснована в 2005 році, компанія стала важливим гравцем у галузі авіаційних технологій. Офіційний сайт FlightAware на рис. 1.5.

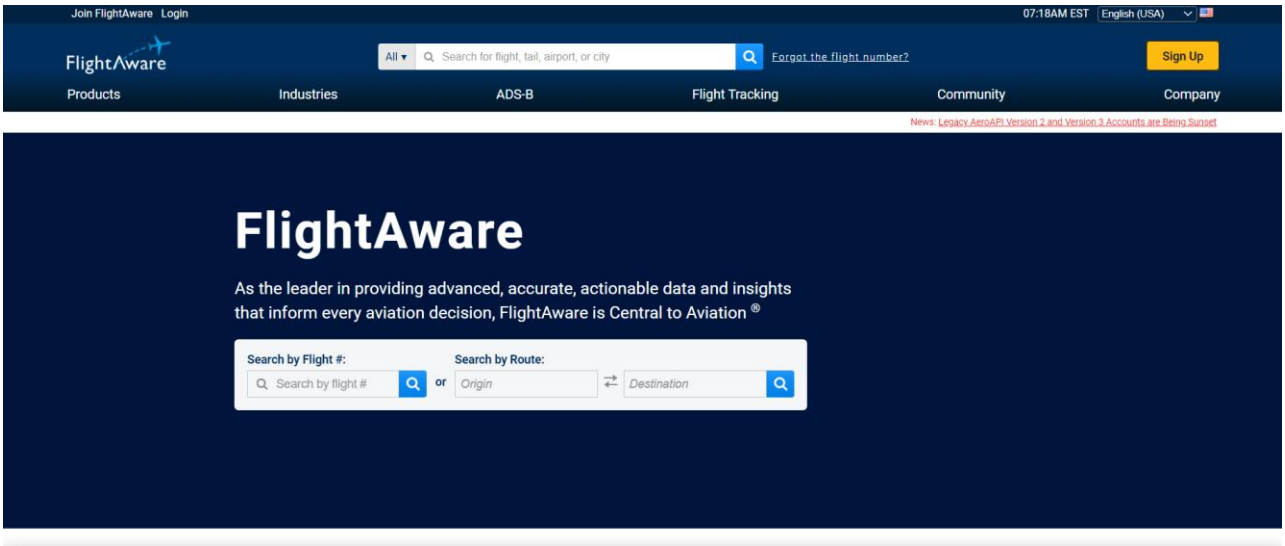

Рис. 1.5. Офіційний сайт FlightAware.

Основні аспекти FlightAware включають:

1. Відстеження Польотів:

- FlightAware пропонує високоточні дані щодо розташування літаків у режимі реального часу. Система використовує дані з різних джерел, таких як ADS-B, MLAT, та інші, для точного відстеження руху літаків по всьому світу. На сайті можна побачити сторінку з відстеженням польотів навіть незареєстрованому користувачу, рис. 1.6.

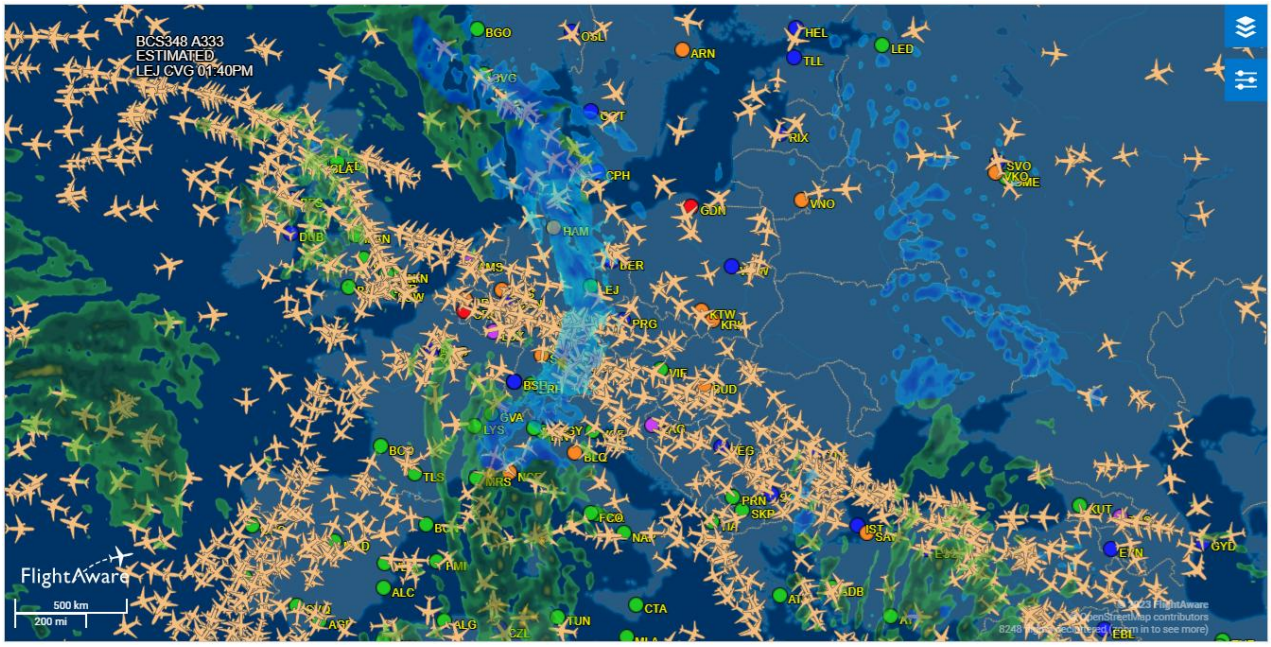

Рис. 1.6. Сторінка відстеження авіаперельотів.

2. Повідомлення та Сповіщення:

- Користувачі можуть отримувати інформацію про польоти через веб-сайт, мобільні додатки або електронні листи. FlightAware також надає можливість встановлювати сповіщення для конкретних польотів.

3. Глобальне Покриття:

- Сервіс FlightAware охоплює польоти по всьому світу, забезпечуючи користувачам доступ до інформації про літаки навіть в найвіддаленіших куточках планети.

4. Інформація про Аеропорти:

- Крім відстеження польотів, FlightAware також надає корисну інформацію про аеропорти, включаючи погоду, розклади рейсів, термінали та інше. Також збирає дані про кількість вильотів із аеропортів і надає цю інформацію у вільному доступі, рис. 1.7.

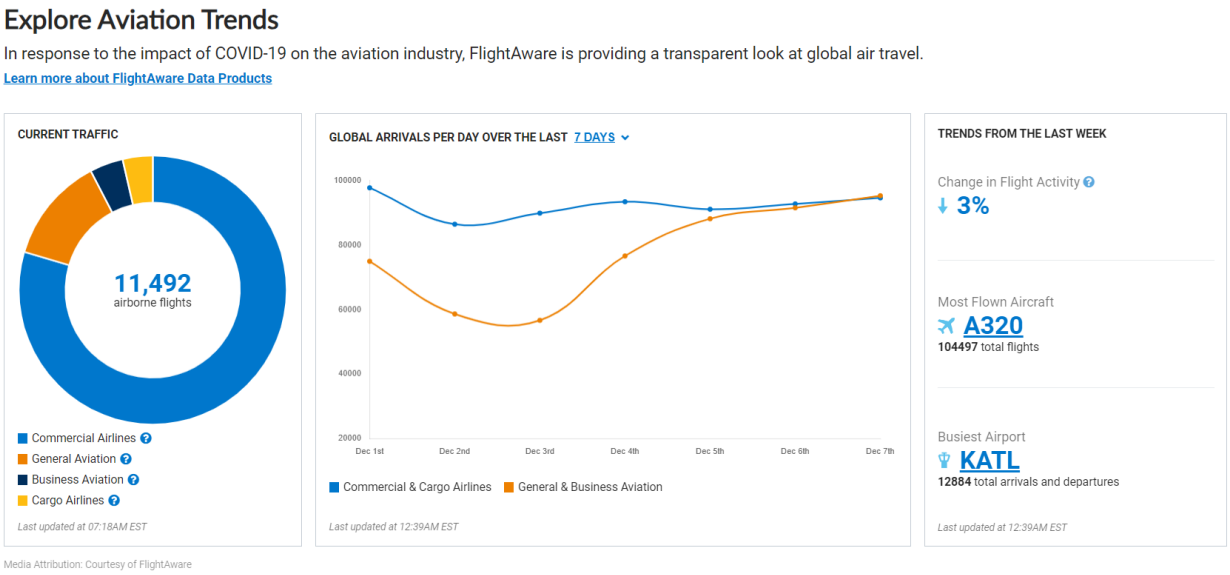

Рис. 1.7. Інформація про авіаперельоти

5. Дані для Авіаційних Професіоналів:

- Компанія надає послуги для авіаційних професіоналів, таких як авіакомпанії, аеропорти та інші гравці галузі. Їхні рішення включають в себе інструменти для відстеження флоту, аналізу даних та оптимізації операцій.

FlightAware ставить своєю метою забезпечити широкий коло користувачів якісною та актуальною інформацією про польоти та пов'язані дані, що робить його популярним серед авіаційних ентузіастів, подорожуючих, а також фахівців галузі.

1.3. Аналіз фінансового ринку авіаперельотів

Авіаційна галузь, яка розцвітала на фоні сучасного світу, не лише перетворює глобальний транспорт, але й стає каталізатором економічного зростання для багатьох країн. Авіакомпанії, як ключові учасники цієї індустрії, активно розвиваються, пристосовуючись до вимог ринку та технологічних інновацій.

Однією з головних складових успіху авіакомпаній є їхні здатності до заробітку. Пригадавши популярну фразу "час - гроші", можна сказати, що для авіакомпаній кожна секунда має величезне значення. Заробіток цих компаній

ґрунтується на декількох ключових аспектах, що формують їхню фінансову успішність.

По-перше, продаж квитків на пасажирські та вантажні перевезення становить значний внесок у доходи. Літаки авіакомпаній регулярно здійснюють тисячі польотів, переносяючи пасажирів та вантажі в різні куточки світу. Важливою частиною їхньої стратегії є ефективне управління розкладами, ціноутворенням та забезпеченням комфортного обслуговування пасажирів.

По-друге, вантажні перевезення відкривають додаткові можливості для авіакомпаній. Вони важливі для глобального логістичного ланцюга, переносячи товари та пошту з одного континенту на інший. Заробіток від вантажних перевезень стає критичним елементом для багатьох авіакомпаній, що розширюють свої мережі та розбудовують флот літаків.

По-третє, розуміння та використання технологій стає невід'ємною частиною успіху авіакомпаній. Інновації в області інформаційних технологій дозволяють вдосконалювати системи бронювання, покращувати безпеку польотів та ефективно управляти флотом. Застосування аналітики та штучного інтелекту сприяє оптимізації різних аспектів бізнесу.

Однак, як у будь-якій галузі, авіаційна індустрія також стикається з викликами. Зміни в цінах на пальне, непередбачувані обставини, такі як пандемії або природні катастрофи, можуть суттєво впливати на фінансове становище авіакомпаній.

Заробіток авіакомпаній у світі може значно варіюватися в залежності від різних факторів, таких як розмір компанії, маршрутна мережа, типи літаків, кон'юнктура ринку, конкуренція, а також економічні та геополітичні умови. Ось деякі ключові аспекти заробітку авіакомпаній:

- одним із основних джерел доходів для більшості авіакомпаній - це пасажирські перевезення, продаж квитків на пасажирські рейси є основним джерелом прибутку, деякі авіакомпанії також отримують дохід від вантажних перевезень;

- авіакомпанії визначають ціни на квитки враховуючи різні фактори, такі як вартість пального, обслуговування пасажирів, конкуренція на конкретних маршрутах та сезонні аспекти;
- деякі авіакомпанії отримують додатковий дохід від надання додаткових послуг, таких як обслуговування на борту, пріоритетний доступ до аеропорту, додаткові послуги для бізнес-класу тощо;
- авіакомпанії можуть укладати комерційні угоди з іншими авіакомпаніями, спільними підприємствами, туроператорами та іншими партнерами для оптимізації прибутку та маршрутних мереж;
- сезонність грає важливу роль, зокрема в туристичних напрямках. Глобальні тенденції, такі як зростання або зменшення попиту на подорожі, можуть впливати на прибутковість.
- вартість пального є значущим чинником в затратах авіакомпаній. Зміни в цінах на нафту можуть значно впливати на їхню прибутковість.

Важливо враховувати, що авіаційна галузь є висококонкурентною, і успіх компанії залежить від ефективного управління вартістю, вдосконалення обслуговування пасажирів та адаптації до змін в економічних та соціокультурних умовах.

Авіакомпанії можуть стикатися з різними факторами, які призводять до фінансових втрат. Деякі з найбільш поширених причин включають:

- вартість пального: зміни в цінах на нафту та вартість пального можуть значно впливати на витрати авіакомпаній, оскільки пальне є одним з основних факторів в їхніх експлуатаційних витратах;
- конкуренція та ціноутворення:висока конкуренція в галузі може призводити до пониження цін на квитки, що впливає на прибутковість авіакомпаній;

- пандемії та кризи: глобальні події, такі як пандемії, кризи та економічні спади, можуть призводити до суттєвого зменшення попиту на авіаперевезення.
- несприятливі погодні умови: погодні умови, такі як грози, снігопади або тумани, можуть призводити до затримок та скасувань рейсів, що впливає на пасажиропотік та може призводити до втрат;
- високі витрати на технічне обслуговування: технічне обслуговування літаків та їхнє підтримання є дорогими процесами, особливо при використанні сучасних та технологічно складних літаків;
- обслуговування боргів та витрати на капіталовкладення: сплати за кредити, обслуговування боргів та витрати на придбання нових літаків можуть бути значними фінансовими тягарами;
- соціальні та трудові конфлікти: страйки або конфлікти з трудовими організаціями можуть призводити до зупинок у роботі та втрат.
- обмеження доступу до даних та кіберзагрози: кібератаки та обмеження доступу до важливих даних можуть впливати на операційну діяльність та призводити до фінансових втрат;
- правові та регуляторні зміни: зміни в законодавстві та регулюванні, такі як введення нових податків чи обмежень, можуть впливати на фінансовий стан авіакомпаній;
- природні катастрофи: природні катастрофи, такі як землетруси або вулканічні виверження, можуть призводити до припинення діяльності або змінювати маршрути.

Ці фактори взаємодіють та можуть призводити до складнощів у фінансовому управлінні авіакомпаній. Успішна стратегія вимагає гнучкості, ефективного управління витратами та виробничими процесами, а також виваженого підходу до ризиків.

1.4. Висновки до розділу

У першому розділі було розглянуто доменну область створення, редагування та збору аналітики про авіаперельоти, було визначено основні проблеми цієї галузі та визначено актуальність розробки даної системи.

Також було розглянуто існуючі аналоги на ринку, які надають дані про свою функціональність публічно, проаналізувавши конкурентів можна зробити висновок, що основна частина цих компаній та їх продуктів стали або багатофункціональними або мають окремі підрозділи під продукти, які вирішують певну проблему галузі. На їх прикладі можна зрозуміти, що ця галузь має у собі великий фінансовий потенціал, а отже у майбутньому будуть з’являтись нові продукти та компанії.

Наявність великих та сильних конкурентів у галузі сприяє компанії до створення інноваційних та унікальних рішень, які будуть їх відрізняти від решти. Також конкуренція змушує компанії зосереджуватись на покращенні якості своїх продуктів та послуг, щоб зберігати та привертати клієнтів. Не менш важливим фактором є і розширення вибору для клієнтів, оскільки вони можуть обирати серед різних варіантів, а це стимулює компанії надавати більше опцій та вдосконалювати свою пропозицію.

Наведені аспекти підтримки конкуренції ведуть до створення здорового та динамічного бізнес-середовища, яке призводить до вигод для споживачів та розвитку ринку в цілому.

Також було розглянуто фінансову сторону галузі авіаперевезень у світі у якій було визначено основні прибутки та втрати грошей на ринку авіаперевезень.

Отже, якщо узагальнити, у першому розділі було розглянуто домену область та конкурентів на ринку.

РОЗДІЛ 2
ОГЛЯД АЛГОРИТМІВ РОЗПІЗНАВАННЯ ТА КЛАСИФІКАЦІЇ ЗОБРАЖЕНЬ КОРИСТУВАЦЬКИХ ІНТЕРФЕЙСІВ

2.1. Загальний опис продукту

Основним завданням продукту є створення, редагування та збору аналітики про польоти для авіакомпаній, це комплексне програмне рішення, яке спрямоване на оптимізацію та поліпшення управління авіаперевезеннями для авіакомпаній. Його загальний опис можна розглядати через призму ключових функцій та переваг.

Однією із переваг є впровадження сучасних засобів автоматизованої системи документування для забезпечення високого рівня безпеки та відповідності регуляторним вимогам.

Також це забезпечення зручного та ефективного інтерфейсу для редагування даних про рейси, пасажирів, екіпажі та інші важливі параметри. Можливість автоматизованого збору та аналізу різноманітних даних, таких як статистика перевезень, інформація про пасажирів і екіпажі.

Розробка зручного клієнтського інтерфейсу для пасажирів та служба підтримки для вирішення запитань та проблем.

Такий продукт створений для того, щоб сприяти оптимізації роботи авіакомпаній, підвищенню їхньої конкурентоспроможності, забезпеченню безпеки та комфорту для пасажирів, а також для ефективного управління всіма аспектами авіаційної діяльності.

Веб-застосунок для створення, редагування та збору аналітики про польоти для авіакомпаній має потенційно широкий спектр користувачів, кожен з яких відіграє свою роль у процесі управління авіаційними операціями. Основними користувачами можуть бути, як адміністратори та менеджери авіакомпаній, оператори та диспетчери рейсів і бортпровідники, а також інший персонал авіакомпаній. Далі буде наведено декілька причин, для чого систему можуть використовувати співробітники авіакомпаній.

Адміністратори та менеджери авіакомпаній відповідають за стратегічне та оперативне управління компанією, тому вони використовуватимуть систему для планування рейсів, редагування розкладів, аналізу даних та прийняття стратегічних рішень для оптимізації функціонування авіакомпанії.

З іншого боку оператори та диспетчери використовуватимуть систему для відстеження рейсів у режимі реального часу, вирішення технічних питань, планування маршрутів та координації з екіпажем.

Наприклад, бортпровідники можуть використовувати систему для реєстрації та керування пасажирськими даними, отримання інформації про авіаперельот та обслуговування пасажирів, внесення даних про рейс. Завжди можуть бути неточності у налаштуваннях даних про рейс, часто літаки затримують або переносять виліт через негоду, тощо.

Інженери та технічний персонал можуть використовувати систему для планування та виконання технічного обслуговування літаків, вони можуть змінювати статус літака після технічного огляду та ремонту.

Спеціалісти з обробки даних та аналітики потребують у системі для збору та аналізу великого обсягу даних про польоти для прийняття стратегічних рішень, виявлення тенденцій та покращення ефективності.

Розглядаючи різноманітних користувачів, система повинна забезпечити інтерфейси та функціонал, спрямовані на конкретні потреби кожного користувача, забезпечуючи їм зручність в роботі та необхідний функціонал для ефективного виконання їхніх обов'язків. Також ця система має бути інтуїтивна зрозуміла, а при необхідності мати документації по роботі з нею, або спеціальне навчання для цього. В ідеалі у КІ має змогти розібратись людину віком з 18 до 65, тому при розробці слід на це звернути увагу.

З іншого боку у системі має бути система ролей, яка не дозволятиме всім співробітникам мати доступ до зміни інформації про рейси, літаки, пілотів, тощо. Має бути зрозуміла лінійна градація ролей, яку визначає адміністрація авіакомпанії, яка буде користуватись систему, з можливістю розширення та пониження доступності до даних.

У системі має бути змога створювати рейс, з даними про літак та пілотів, з даними про час вильоту та прильоту, та звичайно звідки та куди він летить, також має бути система статусів для кожного рейсу, які будуть обмежувати менеджерів та операторів у змінах цього рейсу. Наприклад, коли літак вилетів, то не можна змінити номер літаку у застосунку, або пілотів.

Також продукт має зберігати список наявних рейсів, список наявних пілотів, які працюють на авіакомпанію, з їх кваліфікацією, також список літаків, доступний до перельотів із своїми статусами, можливість додавання нових аккаунтів адміністраторами та надання ролей користувачам системи.

Паролі користувачів мають зберігатись у хешованому вигляді. Зберігання паролів у вигляді хешей у базі даних є сучасною та безпечною практикою з точки зору кібербезпеки. Ось кілька причин, чому це важливо:

1. Безпека Паролів:

- Хешування паролів перетворює їх на непереборний, випадковий набір символів, навіть якщо зловмиснику вдалося отримати доступ до бази даних. При належному використанні сучасних хеш-функцій, вирішення знаходження відповідного паролю за його хешем шляхом "зворотного перетворення" (реверс-інженіринг) стає практично неможливим.

2. Захист від атак "пошук за перебором":

- Якщо паролі зберігаються у вигляді простого тексту, зловмисник, який отримав доступ до бази даних, може використовувати атаку "пошук за перебором" (brute force), спробуючи різні комбінації паролів. Хеші роблять цей процес надзвичайно складним та часово затратним.

3. Безпека при витоку даних:

- У випадку витоку бази даних, хеші паролів залишаються відносно безпечними, оскільки важко визначити оригінальний пароль, навіть якщо він знаходиться в розкодованому вигляді.

4. Виключення ключів управління доступом:

- Якщо паролі зберігаються у вигляді хешей, навіть адміністратор бази даних не має доступу до реальних паролів користувачів. Це підвищує рівень конфіденційності та безпеки.

5. Застосування солі:

- Додавання унікального випадкового значення, відомого як сіль (salt), до кожного паролю перед хешуванням дозволяє запобігти атакам з використанням таблиць радужних хешів (rainbow tables), оскільки це створює унікальний хеш для одного й того ж паролю в різних контекстах.

2.2. Користувацькі вимоги

Користувацькі вимоги до системи - це опис того, як система повинна взаємодіяти з користувачем та виконувати його потреби. Такі вимоги допомагають розробникам створити продукт, який задовольнить очікування та потреби користувачів. Вони можуть включати в себе різноманітні аспекти від інтерфейсу користувача до функціональності та ефективності системи.

Визначимо користувацькі вимоги для веб-системи створення, редагування та збору аналітики про польоти для авіакомпаній:

1. Інтерфейс:

- Інтуїтивний інтерфейс: система повинна мати інтуїтивно зрозумілий та легко навігаційний інтерфейс, щоб користувачі могли швидко орієнтуватися та використовувати всі функції.
- Зручність використання мають забезпечити, щоб користувачі могли ефективно виконувати завдання з мінімальною кількістю помилок.
- Підтримка різних пристроїв, тобто система повинна бути адаптована до різних типів пристроїв, таких як комп'ютери, планшети та смартфони.
- Мінімалізм та Простота, тобто Простий та лаконічний дизайн, з фокусом на основних функціях та легкості використання.

- Візуальна Єдність, тобто єдність стилів, шрифтів, кольорів та інших дизайнерських елементів для створення єдиної та легко впізнаваної марки.
- Зручність Навігації: Інтуїтивна та легка навігація, яка дозволяє користувачам швидко знаходити потрібну інформацію.
- Мультимедійні Елементи: Використання зображень, відео та інших мультимедійних елементів для створення привабливого та цікавого інтерфейсу.
- Персоналізація: Можливість персоналізації інтерфейсу користувачем, включаючи зміну тем та налаштувань.
- Реакція на Дії Користувача: Інтерактивність та анімації, які реагують на взаємодію користувача для покращення враження від використання.
- Інтеграція з Сучасними Технологіями: Використання новітніх технологій, таких як штучний інтелект, віртуальна реальність, аналітика даних та інші, для поліпшення функціональності та забезпечення передових можливостей.
- Безпека та Конфіденційність: Забезпечення надійного зберігання та обробки конфіденційної інформації, як інформації користувача, так і фінансової інформації.
- Спільнота та соціальні функції: інтеграція соціальних мереж, можливість обговорення та обміну інформацією між користувачами.

Ці вимоги відображають стрімкі зміни в технологічному середовищі та вподобання користувачів, спрямовані на створення ефективних, зручних та привабливих веб-застосунків.

2. Безпека та конфіденційність:

- Захист даних: гарантування захисту особистих та фінансових даних користувачів за допомогою шифрування та інших засобів безпеки.

- Система контролю доступу: визначення і реалізація системи контролю доступу, щоб гарантувати, що кожен користувач має доступ тільки до відповідних йому ресурсів.

3. Підтримка користувачів:

- Онлайн підтримка: надання можливості отримання підтримки через онлайн-чат, електронну пошту або телефон.
- Навчання та документація: забезпечення доступу до інструкцій та документації, які допомагають користувачам ознайомитися з системою та вирішувати можливі проблеми.

Ці користувацькі вимоги визначаються конкретними потребами користувачів та є ключовим елементом успіху системи, оскільки вони визначають, наскільки ефективно користувачі можуть взаємодіяти з продуктом.\

2.3. Нефункціональні вимоги

Нефункціональні вимоги або NFR (non-functional requirements) – це набір специфікацій, які описують робочі можливості та обмеження системи та намагаються покращити її функціональність. Це в основному вимоги, які визначають, наскільки добре працюватиме продукт якщо врахувати, наприклад, швидкість, безпеку, надійність, цілісність даних тощо. Структура стандартів ISO/IEC 25000 визначає нефункціональні вимоги як вимоги до якості системи та якості програмного забезпечення.

Нефункціональні вимоги (також відомі як атрибути якості або вимоги до якості послуг) часто асоціюються з системними рішеннями, але вони також ширше застосовуються як до процесів, так і до людських аспектів рішень. Вони доповнюють функціональні вимоги до рішення, визначають обмеження на ці вимоги або описують атрибути якості, які має демонструвати рішення на основі цих функціональних вимог.

До загальних категорій нефункціональних вимог відносяться наступні:

- Доступність (Availability): ступінь, до якого рішення є працездатним і доступним, коли воно потрібне для використання, часто виражається у відсотках часу, протягом якого рішення є доступним.
- Сумісність (Compatibility): ступінь, до якого рішення ефективно працює з іншими компонентами у своєму середовищі, наприклад, один процес з іншим.
- Функціональність (Functionality): ступінь, до якого функції рішення відповідають потребам користувача, включаючи аспекти придатності, точності та інтероперабельності.
- Придатність підтримки (Maintainability): легкість, з якою рішення або компонент може бути модифікований для виправлення помилок, покращення продуктивності або інших характеристик, або адаптації до зміненого середовища.
- Ефективність роботи (Performance Efficiency): ступінь, до якого рішення або компонент виконує свої функції з мінімальним споживанням ресурсів. Може бути визначена на основі контексту або періоду, наприклад, пікове, середнє або непікове використання.
- Переносимість (Portability): легкість, з якою рішення або компонент може бути перенесений з одного середовища в інше.
- Надійність (Reliability): здатність рішення або компонента виконувати необхідні функції в заданих умовах протягом певного періоду часу, наприклад, середній час напрацювання на відмову пристрою.
- Масштабованість (Scalability): ступінь, з яким рішення здатне рости або розвиватися для виконання зростаючих обсягів роботи.
- Безпека (Security): аспекти рішення, які захищають вміст рішення або його компоненти від випадкового або зловмисного доступу, використання, модифікації, знищення або розкриття.

- Зручність використання (Usability): легкість, з якою користувач може навчитися користуватися рішенням. Сертифікація: обмеження на рішення, які необхідні для відповідності певним стандартам або галузевим конвенціям.
- Відповідність (Compliance): регуляторні, фінансові або юридичні обмеження, які можуть відрізнятися залежно від контексту або юрисдикції.
- Локалізація (Localization): вимоги, пов'язані з місцевими мовами, законами, валютами, культурою, правописом та іншими особливостями користувачів, що вимагає уваги до контексту.
- Угоди про рівень обслуговування (Service Level Agreements): обмеження організації, що обслуговується рішенням, які офіційно узгоджені як постачальником, так і користувачем рішення.
- Розширюваність (Extensibility): здатність рішення включати нову функціональність.

Сильні сторони нефункціональних вимог:

- Чітко визначає обмеження, які застосовуються до набору функціональних вимог.
- Надає вимірні вирази того, наскільки добре повинні виконуватися функціональні вимоги, залишаючи за функціональними вимогами опис того, що повинно робити рішення або як воно повинно поводитися. Це також матиме сильний вплив на те, чи буде рішення прийняте користувачами.

Обмеження нефункціональних вимог:

- Чіткість і корисність нефункціональної вимоги залежить від того, що зацікавлені сторони знають про потреби в рішенні і наскільки добре вони можуть висловити ці потреби.
- Очікування різних користувачів можуть суттєво відрізнятися, а досягнення згоди щодо атрибутів якості може бути складним через суб'єктивне сприйняття якості користувачами.

- Набір нефункціональних вимог може мати внутрішні конфлікти і потребувати узгодження.
- Надмірно суворі вимоги або обмеження можуть призвести до збільшення часу та вартості рішення, що може мати негативні наслідки та послабити його прийняття користувачами.
- Багато нефункціональних вимог є якісними, а тому їх важко виміряти за шкалою, і вони можуть викликати певну суб'єктивність користувачів щодо того, як, на їхню думку, конкретні вимоги в кінцевому підсумку задовольняють їхні потреби.

Після аналізу того, що таке нефункціональні вимоги можемо легко визначити їх для нашої веб-системи створення, редагування та збору аналітики про польоти для авіакомпаній:

- конфіденційність клієнтів та їх даних, вся інформація, яка не є публічною, має бути захищена від зловмисників, на прикладі нашої системи – це пароль та логіні аккаунтів менеджерів, адміністраторів та інших співробітників авіакомпанії, також інформація про пілотів;
- сайт має бути адаптивний для всіх пристроїв, тобто для телефонів, планшетів та моніторів різних масштабів має відображатись коректно;
- веб-система має бути масштабована, тобто мати змогу розширювати свій функціонал відповідно до вимог клієнтів авіакомпаній, а також відповідно до змін ринку у доменній сфері цього продукту, ця можливість закладається на самому початку розробки веб-системи, методом вибору технологій та технічних рішень;
- доступність системи має бути 23 години 50 хвилин на добу, бо кожна хвилина може коштувати багато для авіакомпаній, якщо вони не встигнуть змінити рейс чи перенести, тощо, тому система має бути доступною майже весь день, ця доступність гарантується

налаштуваннями системи на сервері розгортання, а також тестуванням перед релізами, аби система не падала;

- база даних має зберігати інформацію про транзакції 12 місяці та за потреби це значення має бути адаптивним, мати можливість збільшуватись або зменшуватись;
- локалізація системи може бути додана у майбутньому, на даний момент буде лише англійська мова, розширення цієї функції потребуватиме доробок зі сторони ФЕ та БЕ;
- система має бути сумісною з іншими сервісами, які можуть інтегруватись, для цього має бути можливість додавання публічного http клієнту або інший метод спілкування з іншими системами, наприклад, через Kafka-повідомлення;
- зручність та простота використання буде оцінюватись тим, скільки дій потрібно користувача для виконання певного функціоналу, цей коефіцієнт не має перевищувати натискань 7-ми кнопок, поля для вводу не рахуємо (варіанти розглянемо у наступному розділі);
- система матиме змогу легко переноситись із одного середовища розгортання на інше, бо матиме налаштовані DockerFile та Docker-compose файли, також БЕ-у все-одно, яка система буде надсилати йому запити, це у варіанті коли потенційна авіакомпанія матиме готовій ФЕ і захоче користуватись функціями нашої системи.

2.4. Функціональні вимоги

Функціональні вимоги описують, що система повинна робити, тоді як нефункціональні вимоги описують, як саме вона повинна це робити. Ці вимоги є ключовими для підтвердження, що система відповідає своєму призначенню та забезпечує позитивний досвід користування. Вони також відіграють важливу роль у визначенні загального успіху проєкту.

Функціональні вимоги – це певні функції, які система повинна виконувати. Ці вимоги описують вхідні й вихідні дані та взаємодію з користувачем.

Найголовніші вимоги до веб-системи це:

- створення та редагування рейсів, до цієї вимоги відноситься також відміна рейсу, до того, як літак вилетів із аеропорту;
- введення списку доступних літаків у авіакомпанії;
- введення списку пілотів літаків;
- система авторизації та реєстрації нових співробітників;
- система допомоги користувачам системи.

Користувач під час взаємодії із системою буде вносити дані, а саме:

- інформація про рейси;
- інформація про літаки;
- інформація про пілотів;
- дані авторизації, це пошта та пароль.

Далі розглянемо КІ, який було вирішено поділити на прості логічні блоки для того, аби мати незалежні частини системи і у разі чого мати змогу без проблем їх замінювати іншими, представлено діаграму на рис. 2.1.

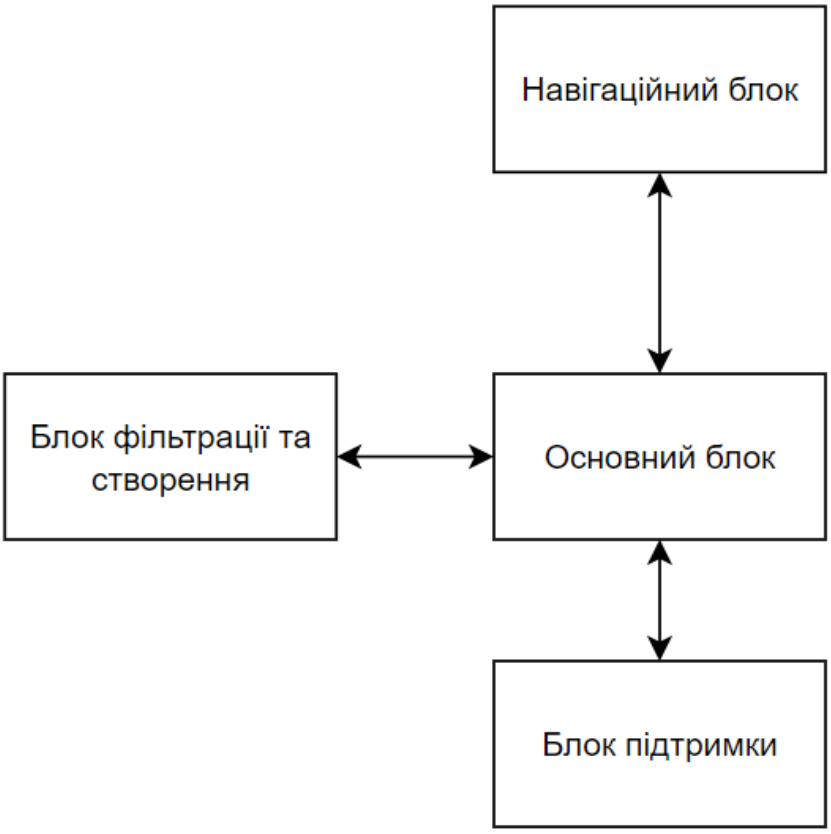

Рис. 2.1. Логічні блоки КІ

Представлена схема показує, що КІ було поділено на чотири частини, у якій є основний блок на який впливають всі інші, цей блок містить у собі корисну інформацію для користувача, наприклад, там буде відображений список рейсів.

Навігаційний блок – це частина КІ, яка містить у собі кнопки для переходів між сторінками та виходом або входом у систему.

Блок фільтрації та створення – це блок у якому знаходяться всі потрібні засоби для фільтрації, пошуку або створення елементів, які знаходяться у основному блоці.

Блок підтримки перекидає користувача на сторінку у якій він може або задати питання до центру підтримки або задокументувати проблему системи, тобто баг і повідомити про нього.

Далі для опису нашої систему скористаємось корисним засобом візуалізації користувачів та їх дій – діаграмою прецедентів, рис. 2.2.

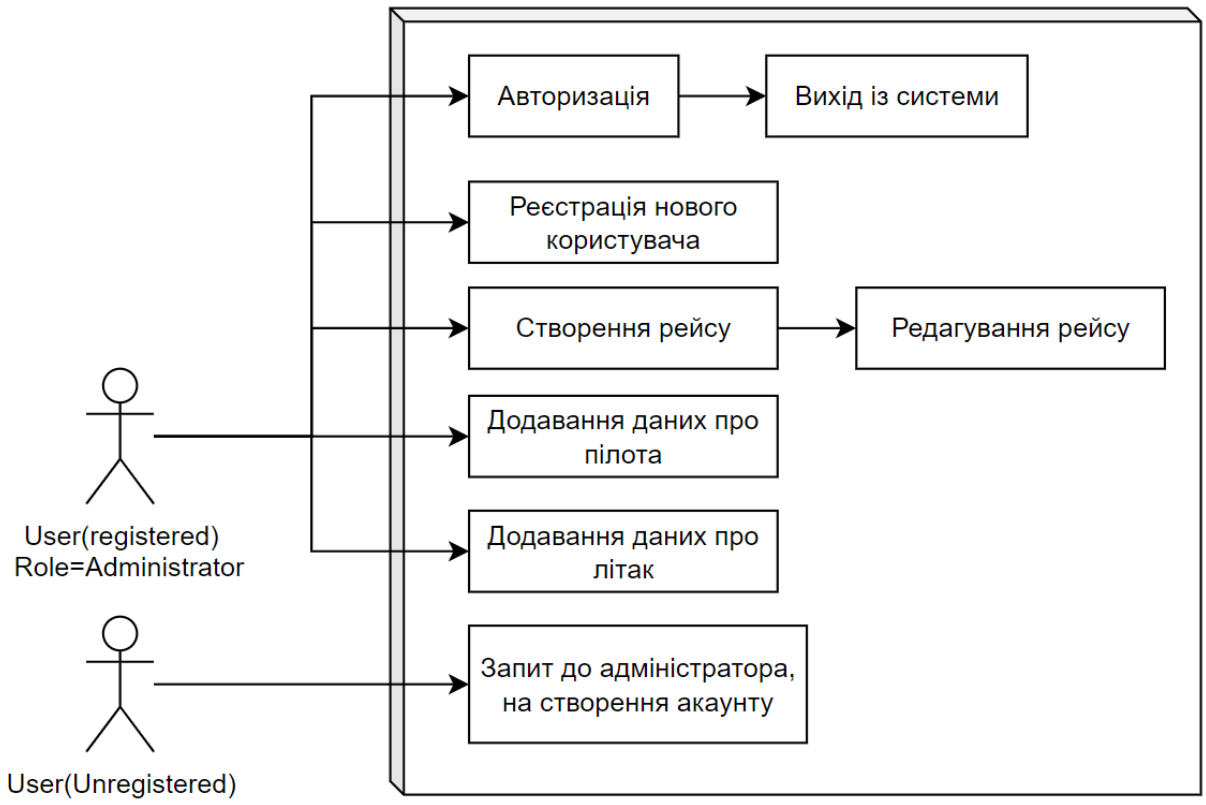

Рис. 2.2. Діаграма прецедентів

Діаграма прецедентів використовує 2 основних елементи:

1) Actor (учасник) — множина логічно пов'язаних ролей, виконуваних при взаємодії з прецедентами або сутностями (система, підсистема або клас). Учасником може бути людина, роль людини в системі чи інша система, підсистема або клас, які представляють щось поза сутністю.

2) Use case (прецедент) — опис окремого аспекту поведінки системи з точки зору користувача. Прецедент не показує, "як" досягається певний результат, а тільки "що" саме виконується.

Велике значення діаграма прецедентів має для специфікування, візуалізації та документування поведінки системи. Використовуючи її, розробнику легше розуміти систему, підсистему або класи, а також поглянути ззовні на переваги використання елементів для того чи іншого контексту. Подібна UML-діаграма представляє особливу важливість для проведення тестування виконуваних систем при прямому проектуванні, а також для

кращого розуміння їх внутрішнього пристрою, особливо при зворотному проектуванні.

Можно побачити на діаграмі основні варіанти використання веб-системи користувачем. На діаграмі може побачити трьох різних користувачів, давайте опишемо кожного з них:

- User(registered) – це зареєстрований у системі користувач, який має може авторизуватись, створювати та редагувати рейси, додавати інформацію про літаки та пілотів.
- User(registered) (Role = Administrator) – це зареєстрований у системі користувач, який має роль адміністратора, він володіє всіма функція та додатково може реєструвати нових користувачів у систему;
- User(Unregistered) – незареєстрований користувач, він не може користуватись системою, лише запросити створення акаунта у адміністратора через канали авіакомпанії, якій надано доступ до системи.

Наступним кроком опису систему буде написання Use case прецедентів. Use Case описує сценарій взаємодії учасників (як правило - користувача та системи). Учасників може бути 2 та більше. Користувачем може бути як людина, і інша система.

Навіщо та кому потрібно описувати Use cases системи?

Розробникам набагато зручніше, коли гілляста вимога описана за допомогою основного та альтернативного потоку подій. Все чітко і зрозуміло: хто, коли і що викликає, і що виходить у результаті. У моєму останньому проекті менеджер сповідував підхід: мінімум описів, максимум спілкування. Але для кількох складних сценаріїв розробники самі просили зробити докладний опис, і юзкейси чудово підійшли.

Замовники можуть побачити опис системи людською мовою, замовник вчасно може підтвердити, що це саме те, на що він чекає, або поправити.

Також розберемо у який випадках потрібно описувати сценарії системи:

- якщо компанії потрібна якісна повна специфікація вимог — use cases чудово в цьому допоможуть. Є такі системи, для розробки та підтримки яких специфікація вимог, що містить модель даних, опис інтерфейсу, інтеграції з іншими системами та use case – дуже гарний варіант;
- для підтримки системи. Щоб виявити помилку, розібратися, якому кроку пішло негаразд;
- якщо потрібно описати певну частину функціональності, роботи користувача з інтерфейсом, тощо. у вигляді сценарію. Тоді ви можете взяти шаблон use case за основу та використати його для опису сценарію. Але виконання деяких функцій має багато нюансів, які потрібно додатково описати за допомогою таблички: "дія - відгук системи", або навіть поєднати таку табличку зі сценарієм.

Розглянемо нижче декілька сценаріїв, які описані у наступних таблицях 2.1-2.4 для авторизації та створення аккаунта для колеги адміністратором.

Сценарії авторизації представлено у таблицях 2.1-2.2, для авторизації користувачеві потрібно заповнити два поля, ввести пошту, за якою був зареєстрований акаунт та пароль до акаунту. Після чого система перевіряє дані і в залежності від правильності вводу або авторизує користувача або ні.

Таблиця 2.1

Use Case авторизації, успішна

Актори	Користувач, Система
Ціль	Увійти в систему
Передумова	Немає
Сценарій: 1. Користувач заповнює поля для входу 2. Користувач відправляє заповнену форму 3. Система отримає дані користувача, ідентифікує і дозволяє вхід 4. Користувач входить в систему.	
Результат	Користувач успішно авторизується

Таблиця 2.2

Use Case авторизації, неуспішна

Актори	Користувач, Система
Ціль	Увійти в систему
Передумова	Немає
Сценарій: 1. Користувач заповнює поля для входу 2. Користувач відправляє заповнену форму 3. Система отримає дані користувача, ідентифікує і віддає помилку системи про невалідність даних або такого користувача немає у системі 4. Користувач не зміг увійти в систему	
Результат	Користувач не зміг авторизуватись

У таблицях 2.3-2.4 описано сценарії створення акаунту користувачу, для створення акаунта у користувача має бути роль Адміністратора.

Таблиця 2.3

Use Case створення акаунту користувачу, успішна

Актори	Користувач, Система
Ціль	Увійти в систему та створити аккаунт для колеги
Передумова	Створити аккаунт для колеги
Сценарій: 1. Користувач заповнює поля для входу 2. Користувач відправляє заповнену форму 3. Система отримує дані користувача, ідентифікує і дозволяє вхід, користувач має роль Адміністратора 4. Користувач заповнює форму для створення аккаунта для колеги 5. Користувач відправляє заповнену форму 6. Система отримує дані користувача, валідує та створює аккаунт 7. Користувач зміг створити аккаунт для колеги	
Результат	Користувач створив аккаунт для колеги

Таблиця 2.4

Use Case створення акаунту користувачу, неуспішна

Актори	Користувач, Система
Ціль	Увійти в систему та створити аккаунт для колеги
Передумова	Створити аккаунт для колеги
Сценарій: 1. Користувач заповнює поля для входу 2. Користувач відправляє заповнену форму 3. Система отримує дані користувача, ідентифікує і дозволяє вхід, користувач має роль Адміністратора 4. Користувач заповнює форму для створення аккаунта для колеги 5. Користувач відправляє заповнену форму 6. Система отримує дані користувача, валідує та надсилає помилку про те, що такий акаунт створений або введені некоректні дані 7. Користувач не зміг створити аккаунт для колеги	
Результат	Користувач не зміг створити аккаунт для колеги

У наступних сценаріях буде описана одна із основних функцій системи, а саме – створення рейсу (таблиці 2.5-2.6).

Таблиця 2.5

Use Case створення рейсу, успішна

Актори	Користувач, Система
Ціль	Увійти в систему та створити рейс
Передумова	Створити рейс
Сценарій: 1. Користувач авторизується 2. Користувача переходить на вкладку з рейсами 3. Користувач відкриває та заповнює форму для створення рейсу 4. Система отримує дані користувача, валідує їх та створює рейс 5. Користувач зміг створити рейс і він відображається у загальному списку	
Результат	Користувач створив рейс

Таблиця 2.6

Use Case створення рейсу, неуспішна

Актори	Користувач, Система
Ціль	Увійти в систему та створити рейс
Передумова	Створити рейс
Сценарій: 1. Користувач авторизується 2. Користувача переходить на вкладку з рейсами 3. Користувач відкриває та заповнює форму для створення рейсу 4. Система отримує дані користувача, валідує їх та повертає помилку, дані були введені неправильно 5. Користувач не зміг створити рейс	
Результат	Користувач не зміг створити рейс

Наступні сценарії будуть відноситись до додавання літаків та пілотів у систему (таблиці 2.7-2.10).

Таблиця 2.7

Use Case додавання літаку у систему, успішна

Актори	Користувач, Система
Ціль	Увійти в систему та додати літак
Передумова	Додати літак у систему
Сценарій: 1. Користувач авторизується 2. Користувача переходить на вкладку з літаками 3. Користувач відкриває та заповнює форму для додавання літака 4. Система отримує дані користувача, валідує їх та додає літак у систему 5. Користувач зміг додати літак у систему і тепер він відображається у системі	
Результат	Користувач додав літак у систему

Таблиця 2.8

Use Case додавання літаку у систему, неуспішна

Актори	Користувач, Система
Ціль	Увійти в систему та додати літак
Передумова	Додати літак у систему
Сценарій: 1. Користувач авторизується 2. Користувача переходить на вкладку з літаками 3. Користувач відкриває та заповнює форму для додавання літака 4. Система отримує дані користувача, валідує їх та отримує помилку, про те що дані невалідні 5. Користувач не зміг додати літак у систему	
Результат	Користувач не додав літак у систему

Таблиця 2.9

Use Case додавання пілота у систему, успішна

Актори	Користувач, Система
Ціль	Увійти в систему та додати пілота
Передумова	Додати пілота у систему
Сценарій: 6. Користувач авторизується 7. Користувача переходить на вкладку з пілотами 8. Користувач відкриває та заповнює форму для додавання пілота 9. Система отримує дані користувача, валідує їх та додає пілота у систему 10. Користувач зміг додати пілота у систему і тепер він відображається у системі	
Результат	Користувач додав пілота у систему

Таблиця 2.10

Use Case додавання пілота у систему, неуспішна

Актори	Користувач, Система
Ціль	Увійти в систему та додати пілота
Передумова	Додати пілота у систему
Сценарій: 6. Користувач авторизується 7. Користувача переходить на вкладку з пілотами 8. Користувач відкриває та заповнює форму для додавання пілота 9. Система отримує дані користувача, валідує їх та отримує помилку, про те що дані невалідні 10. Користувач не зміг додати пілота у систему	
Результат	Користувач не додав пілота у систему

2.5. Системні вимоги

Для початку треба розібратись, що визначає цей термін і для чого такі вимоги потрібні.

Системні вимоги – це більш деталізований опис користувацьких вимог. Вони зазвичай є основою для укладення контракту на розробку програмної системи і тому повинні представляти максимально повну специфікацію системи в цілому. Системні вимоги також використовуються в якості відправної точки на етапі проектування системи.

Специфікація системних вимог може будуватися на основі різних системних моделей, таких, як об'єктна модель або модель потоків даних. Різні моделі, використовувані при розробці специфікації системних вимог.

У принципі системні вимоги визначають, що повинна робити система, не показуючи при цьому механізму її реалізації. Але, з іншого боку, для повного опису системи потрібна деталізована інформація про неї, яка по можливості повинна включати всю інформацію про системну архітектуру. На те існує ряд причин.

- Первісна архітектура системи допомагає структурувати специфікацію вимог. Системні вимоги повинні описувати підсистеми, з яких складається розроблювальна система.

- У більшості випадків розроблювальна система повинна взаємодіяти із уже існуючими системами. Це накладає певні обмеження на архітектуру нової системи.
- У якості зовнішньої системної вимоги може виступати умова використання для розроблювальної системи спеціальної архітектури.

Для використання майбутньої веб-системи користувача потрібен пристрій з виходом у інтернет та монітор для відображення інформації, такі системні характеристики будуть мінімальними для веб-системи:

- оперативна пам’ять пристрою, від 2048 мб;
- інтернет з’єднання зі швидкістю від 10 мегабіт на секунду;
- пристрій має підтримувати користуванням браузера та мати встановлений браузер, будь-який із тих, що наявні на ринку;
- монітор, клавіатура та мишко або сенсорна панель для вводу данних.

2.6. Висновки до розділу

У другому розділі було представлено опис вимог до веб-системи створення, редагування та збору аналітики про авіаперельоти для авіакомпаній, було описано більшість бізнес вимог до системи, а саме:

- користувацькі;
- нефункціональні;
- функціональні.

Також було представлено графічно логічні блоки КІ системи, створено діаграму прецедентів, описано основну частину функціоналу за допомогою Use case сценарії у таблицях

Визначено системні вимоги для комфортного та зручного використання системи клієнтами. Весь цей опис буде слугувати документацією того, що

клієнт може очікувати від майбутньої системи, але також у подальшому клієнти можуть вносити правки у цю документацію, бо кожна окрема компанія буде мати свої унікальні особливості використання системи у роботі, у випадку правок система матиме додаткову технічну роботу, яка займе деякий час, що теж варто враховувати.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ СТВОРЕННЯ, РЕДАГУВАННЯ ТА ЗБОРУ АНАЛІТИКИ ПРО ПОЛЬОТИ ДЛЯ АВІАКОМПАНІЙ

3.1. Використані інструменти для створення веб-системи

Обираючи інструменти для створення веб-системи, важливо враховувати ряд факторів, щоб забезпечити ефективність, масштабованість та зручність розробки і подальшого використання. Ось декілька кроків, які були корисними при виборі технологій:

- Визначення вимог до проекту, у попередньому розділі було з'ясовано, які саме функції та можливості повинна надавати веб-система. Визначення вимог допомогла зрозуміти, які інструменти та технології будуть оптимальними для досягнення цілей системи.
- Було розглянуто власний досвід та експертизу і обрано інструменти, які раніше було вивчені та освоєні на практиці під час навчання в Національному Авіаційному Університеті.
- Дослідження технологічного стеку, а саме – проведено дослідження різних технологій та технічних стеків. Перевірено їхню актуальність, популярність та спільноту розробників.
- Враховано масштаби та ефективність, тобто було розглянуто, як добре обрані технології масштабуються та чи вони відповідають потребам проекту з точки зору продуктивності та швидкодії.
- Було звернуто увагу на спільноту та підтримку технології. Це може бути важливо для отримання допомоги та вирішення проблем під час розробки та покращення системи.
- Ключовою деталлю була безпека та відповідність, враховано вимоги до безпеки та відповідність. Обрано інструменти, які допоможуть забезпечити високий рівень захисту даних користувачів у нашій системі.

- Було проаналізовано та вивчено вартість ліцензій для використання технології. Перевірено вартість користування та ліцензійні умови для обраних інструментів, таких як база даних, редактор для написання коду. Важливо врахувати бюджет та вартість підтримки систему у подальшому при використанні певних технологій, також варто враховувати вартість на майбутнє збільшення навантаження системи.
- Ще одним важливим аспектом була надійність та стабільність обраних інструментів. Важливо мати на увазі, що вони підтримуються та не мають серйозних проблем безпеки.
- Було враховано тривалість життєвого циклу технологій. Важливо вибрати технічний стек, який буде підтримуватися та розвиватись у майбутньому.

Обережне врахування цих аспектів допоможе здійснити оптимальний вибір інструментів для вашого проекту, забезпечуючи успішний розвиток та ефективне використання веб-системи.

Після аналізу було обрано такі технології для створення веб-системи:

- ASP.NET Core з EntityFramework для написання БЕ;
- React native для написання ФЕ.

Розглянемо ці технології більш детально, щоб зрозуміти їх переваги та недоліки при використанні.

3.1.1. Технології БЕ

Почнемо із БЕ частини, для розробки з використанням ASP.NET Core було використано об’єктно-орієнтовну мову програмування – C#.

Загальний опис

C# (вимовляється "сі-шарп") – це потужна мова програмування, яка була розроблена корпорацією Microsoft і вперше представлена у 2000 році. Ця мова базується на сім'ї мов C, включаючи C, C++, та Java. Розглянемо основні характеристики та використання C# у світі програмування.

Об'єктно-орієнтоване програмування та C#

Однією з ключових особливостей C# є її повна об'єктно-орієнтована природа. Вона сприяє створенню структурованого та легко розширюваного коду, що полегшує підтримку та розвиток проектів. Для кращого розуміння, що таке об'єктно-орієнтована мова програмування розберемо, що ж таке об'єктно-орієнтоване програмування.

Об'єктно-орієнтоване програмування (ООП) – це парадигма програмування, яка базується на концепції "об'єктів", які представляють реальні або абстрактні сутності та взаємодіють один з одним. Мови програмування, які підтримують ООП, мають численні переваги, серед яких:

1. Модульність та підтримка розширення:

- Об'єкти в ООП можуть бути розглянуті як модулі, що містять дані та функції, які з ними пов'язані. Це сприяє модульності програмного коду та полегшує його розширення без змін в інших частинах системи.

2. Поліморфізм та віртуальні методи:

- Мови ООП дозволяють використовувати поліморфізм, тобто можливість об'єктів одного класу використовувати методи, які визначені в класі-батьку. Це полегшує обробку об'єктів різних типів з однаковим інтерфейсом.

3. Інкапсуляція та захист даних:

- Інкапсуляція дозволяє обмежити доступ до деяких частин об'єкта та приховати реалізацію. Це забезпечує захист даних та методів від неправильного використання та забезпечує їх контрольований доступ.

4. Підтримка класів та об'єктів:

- Класи та об'єкти дозволяють структурувати код та організовувати дані за концепцією об'єктів. Це сприяє легшому розумінню та утриманню програмного коду.

5. Підтримка спадкування:

- Спадкування дозволяє створювати новий клас на основі існуючого, спадкуючи його властивості та методи. Це спрощує рефакторинг коду та забезпечує його перевикористання.

6. Розподілена розробка:

- ООП сприяє розподіленій розробці, де різні розробники можуть працювати над різними класами та об'єктами, не заважаючи один одному. Це полегшує роботу великих команд.

7. Підтримка системного аналізу:

- ООП полегшує впровадження понять та моделей, що використовуються у системному аналізі. Об'єктні моделі можуть бути легко переносимі з аналітичного етапу розробки до фази реалізації.

8. Спрощена підтримка багатозадачності:

- Об'єктно-орієнтоване програмування надає зручні інструменти для створення багатозадачних програм, де об'єкти можуть взаємодіяти асинхронно.

Загалом, об'єктно-орієнтоване програмування сприяє покращенню структури коду, полегшенню розвитку та утримання програм, а також спрощенню співпраці великих розробницьких команд.

.NET Framework та C#

Також C# тісно пов'язана з .NET Framework, що надає розробникам доступ до великої стандартної бібліотеки класів. Це дозволяє ефективно розробляти різноманітні додатки на основі готових рішень. .NET Framework - це платформа розробки програмного забезпечення, розроблена компанією Microsoft. Вона надає середовище виконання та бібліотеки класів для розробки та виконання різноманітних типів додатків, включаючи десктопні додатки, веб-додатки та служби.

Основні аспекти та функції .NET Framework:

1. Крос-платформенність:

- .NET Core, який є модернізованою версією .NET Framework, є крос-платформеним і підтримує роботу на різних операційних системах, таких як

Windows, macOS та Linux. Це робить .NET більш гнучким для розробки додатків на різних платформах.

2. Мовна Незалежність:

- .NET Framework підтримує різні мови програмування, такі як C#, VB.NET, F#, інші мови .NET. Розробники можуть вибирати мову, яка найкраще відповідає їхнім потребам та навичкам.

3. Безпека та Керування Пам'яттю:

- Використання керування пам'яттю у .NET Framework допомагає уникнути багатьох типових помилок, пов'язаних із невірним використанням пам'яті, і забезпечує більш високий рівень безпеки виконання програм.

4. Бібліотеки Класів:

- .NET надає обширні бібліотеки класів, які включають різноманітні функції та інструменти для розробників. Це робить розробку додатків ефективною та швидкою, оскільки велику частину функціоналу можна використовувати "з коробки".

5. Віртуальна Машина .NET (CLR):

- Common Language Runtime (CLR) використовується для виконання програм на .NET. Вона забезпечує управління пам'яттю, обробку винятків, безпеку та інші важливі функції для забезпечення стабільності та ефективності виконання.

6. Розподілена Архітектура:

- .NET підтримує розробку розподілених додатків, що можуть використовувати веб-служби та інші механізми для обміну даними та функціональністю між різними компонентами.

7. Інтеграція з Інструментами Розробки:

- Платформа інтегрується з різними інструментами розробки, такими як Visual Studio, що забезпечує зручну роботу розробників та підтримку широкого спектру інструментів.

.NET Framework є ключовим інструментом для розробки додатків під Windows та іншими платформами, що надає зручний інструментарій та

інфраструктуру для створення різноманітних застосунків. Бібліотека класів Framework .NET – Framework Class Library (FCL) містить понад 7 000 типів (класів, структур, інтерфейсів, перелічених типів 9 і делегатів), які поділено між "просторами імен", кожен з яких відповідає за служби з певної області. Простори імен, які автоматично додаються до нового проекту, наведені в табл. 3.1.

Таблиця 3.1

Основні простори імен .NET

Простір імен	Опис
System	Містить основні типи та класи
System.Collections	Кешовані таблиці, динамічні масиви та інші контейнери
System.Collections.Generic	Містить інтерфейси та класи, що визначають універсальні колекції, які дозволяють користувачам створювати строго типізовані колекції
System.Linq	Містить класи й інтерфейси, які підтримують LINQ (Language-Integrated Query)
System.Text	Містить класи для кодування символів і для управління рядками. Дочірній простір імен дозволяє обробляти текст з використанням регулярних виразів
System.Threading.Tasks	Містить класи, які спрощують роботу з написання паралельного й асинхронного коду

Асинхронне програмування у C#

Мова підтримує концепції асинхронного програмування, що дозволяє ефективно працювати з багатозадачними програмами та оптимізувати ресурси системи. Асинхронне програмування дозволяє уникнути появи вузьких місць продуктивності та збільшити загальну швидкість реагування програми. Однак традиційні методи створення асинхронних додатків можуть виявитися складними як у плані написання коду, так і в плані налагодження та обслуговування.

C# підтримує спрощений підхід, асинхронне програмування, яке використовує асинхронну підтримку серед виконання .NET. Компілятор виконує складну роботу, яку раніше робив розробник, при цьому логічна структура програмного коду схожа на синхронний код. Тобто можна користуватися всіма перевагами асинхронного програмування зі значно меншими, ніж зазвичай, трудовитратами.

Асинхронність необхідно використовувати за наявності дій, що потенційно блокують роботу, наприклад при здійсненні доступу до Інтернету. Доступ до веб-ресурсу іноді здійснюється повільно або із затримкою. Якщо така дія блокується в синхронному процесі, вся програма змушена чекати. В асинхронному процесі програма може перейти до наступної операції, яка не залежить від веб-ресурсу, до завершення блокуючого завдання.

Для чого використовується C#

C# застосовується для створення десктопних додатків за допомогою технологій, таких як Windows Forms та Windows Presentation Foundation (WPF). Це дозволяє створювати інтерактивні та графічно багаті додатки для операційної системи Windows.

За допомогою ASP.NET, C# використовується для створення веб-додатків та служб. ASP.NET надає потужні засоби для розробки та підтримки високопродуктивних веб-сайтів.

Мова програмування C# широко використовується в ігровій індустрії, зокрема, завдяки платформі розробки ігор Unity. Розробники можуть використовувати C# для створення ігор для різних платформ.

Xamarin, що базується на C#, дозволяє розробникам створювати мобільні додатки для різних платформ, таких як Android та iOS, використовуючи спільний код.

Недоліки C#

Сильні сторони цього інструменту було описані вище тепер треба розібратись і з недоліками, які можуть виникнути при застосуванні цієї технології.

Один із головних недоліків є чи не найнижчий поріг входу для новачків. У .NET не можна просто створити файл index.html, відкрити його у браузері та відразу побачити готовий результат. Microsoft послідовно працює над зниженням порогу входу до платформи, але він все ще високий.

Наявність великої кількості технологій та підходів для розробки десктопних додатків. Зараз паралельно існують Windows Forms, WPF та MAUI. Без попереднього аналізу складно сказати, що з фреймворків могло краще підійти під завдання. У Інтернеті, наприклад, починаючи з .NET Core, Web API і MVC уніфікували в один фреймворк, і тепер для Інтернету альтернатив за великим рахунком немає.

Періодична смерть технологій у межах платформи. Silverlight, XNA, project.json (новий формат файлів проекту для .NET Core), постійні зміни парадигм у SDK для MS Office аддонів, що вмирає WCF, невизначеність із Blazor/MVC – це лише частина прикладів. У цілому нині процес природний, але в непідготовлених людей може викликати питання.

Було спеціально винесено окремо C#, тому що він є як перевагою, так і нестачею платформи. З погляду переваг ми маємо чудовий компілятор, багату стандартну бібліотеку та непоганий синтаксис. Крім того, це універсальна мова, якою можна вирішувати завдання з будь-яких областей програмування.

З недоліків, на мій погляд, можна відзначити надто велике багатство синтаксису, що підвищує поріг входу, особливо в останніх версіях мови. Номінальна система типів періодично змушує писати коду більше, ніж необхідно (до речі, творець C# таки зробив структурну систему типів у своїй наступній C# мові — TypeScript). Але це за великим рахунком причіпки, тому що якщо потрібно використовувати більше функціонального підходу, то завжди можна зробити збірку на F# і завдяки CLI та CLR ці мови можна комбінувати в рамках одного проекту.

За підсумками сьогодні платформа .NET є досить конкурентним рішенням для сучасної веб-розробки, особливо серверної частини. Швидкодія середовища виконання, простота розробки, баланс між складністю мови та її можливостями роблять .NET хорошим вибором як для невеликих проектів, так і для великих enterprise-рішень.

Чому обрано IDE JetBrains Rider

Для розробки використовувалось IDE JetBrains Rider, але на ринку є його прямий конкурент і це Visual Studio.

Rider і Visual Studio є відомими IDE у спільноті розробників. Кожен пропонує свої унікальні функції та можливості. Коли ви орієнтуєтесь у світі кодування, розуміння відмінностей і сильних сторін кожного може бути неоціненним. Давайте дослідимо ці два інструменти та подивимося, як вони співвідносяться один з одним.

Звісно, Visual Studio є однією з найпопулярніших інтегрованих розробницьких середовищ (IDE) для платформи .NET та інших мов програмування. Ось деякі з плюсів використання Visual Studio:

- Visual Studio підтримує ряд мов програмування, включаючи C#, Visual Basic, F#, C++, Python, JavaScript, TypeScript та інші. Це робить його універсальним інструментом для розробки різноманітних застосунків.
- Як продукт від Microsoft, Visual Studio має глибоку інтеграцію з платформою .NET та іншими технологіями Microsoft, такими як

Azure. Це полегшує розробку та розгортання додатків в хмарному середовищі.

- У Visual Studio є інструменти для швидкої розробки інтерфейсу користувача (UI). Наприклад, Windows Forms і WPF дозволяють легко створювати графічні інтерфейси десктопних додатків.
- Visual Studio надає широкий спектр інструментів для розробки, тестування та відлагодження коду. Включає в себе інтегровані системи управління версіями, тестування, аналіз коду, редагування графічного інтерфейсу та інше.
- Visual Studio Code (VS Code) - легка версія Visual Studio, яка підтримує широкий вибір мов програмування та платформ. Вона ідеально підходить для швидкої розробки та роботи з різноманітними технологіями.
- Visual Studio має розширювану архітектуру, що дозволяє вам використовувати різноманітні плагіни та додатки для покращення функціоналу. Visual Studio Marketplace надає велику кількість розширень від спільноти та компаній.
- Visual Studio має потужні інструменти для розробки ігор, зокрема, інтеграцію з Unity для створення ігор на платформі Unity.
- Visual Studio Team Services та Team Foundation Server надають інструменти для командної розробки, включаючи системи управління проектами, тестування та CI/CD (Continuous Integration/Continuous Deployment).

Загалом, Visual Studio (рис. 3.1) вважається потужним та універсальним інструментом для розробників.

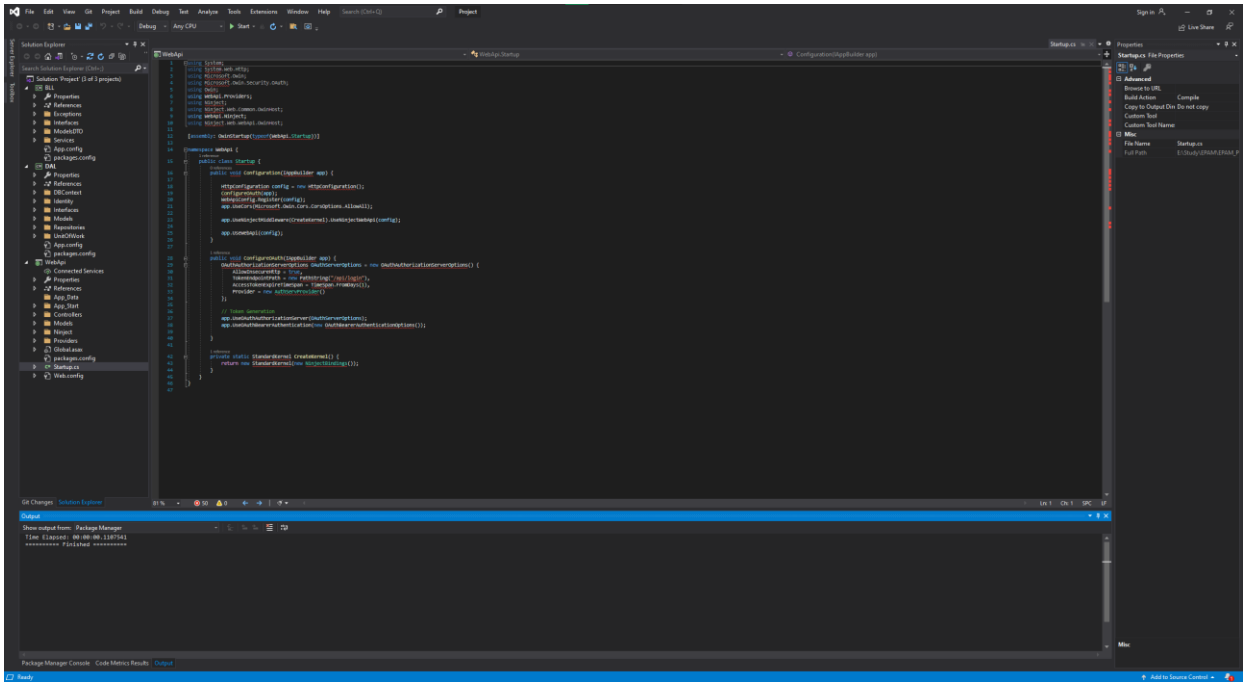

Рис. 3.1. Інтерфейс Visual Studio

Але і у Visual Studio є низка недоліків:

- Великі вимоги до ресурсів комп'ютера, особливо при роботі з великими проектами, що може впливати на продуктивність на менш потужних системах.
- Крос-платформенність обмежена, зокрема версії для Windows мають більше можливостей порівняно з версією для Mac.
- Тісна інтеграція з екосистемою Microsoft, що може бути обмежуючим для розробників, які працюють із іншими технологіями або мовами програмування.
- Великий обсяг функціоналу, що може виглядати складним для освоєння, особливо для новачків.
- Деякі продвинуті функції доступні тільки у платних версіях, таких як Professional або Enterprise, що може впливати на доступність деяких інструментів.
- Інтеграція з іншими інтегрованими розробницькими середовищами може бути не такою зручною, зокрема для тих, хто працює у різних середовищах розробки.

Проаналізувавши всю інформацію про Visual Studio можу зробити висновок, що цей інструмент найкраще підходить для розробників, які використовують Windows та створюють системи інтегровані із Azure, також це ідеальний вибір для розробників на WPF та WindowsForms, бо Visual Studio має свій власний інструмент для роботи з цими технологіями.

Далі розглянемо переваги Rider:

- Rider є повністю крос-платформеним, що означає, що ви можете використовувати його на різних операційних системах, включаючи Windows, macOS і Linux.
- Часто відзначається високою швидкістю компіляції проектів та роботою з кодом, що сприяє загальній продуктивності розробника.
- Підтримує широкий спектр мов програмування, включаючи C#, F#, VB.NET, JavaScript, TypeScript, HTML, CSS та інші.
- Хороша інтеграція з іншими популярними інструментами розробки, такими як системи контролю версій, Docker, NuGet та багато інших.
- Має потужний та сучасний редактор коду з підсвічуванням синтаксису, автодоповненням, аналізом коду та іншими корисними функціями.
- Надає розширені інструменти для рефакторингу коду, що полегшує покращення структури та якості коду.
- Включає в себе широкий функціонал для розробки та можливість встановлювати різні плагіни для розширення можливостей.
- Має активну спільноту користувачів, а регулярні оновлення вдосконалюють функціонал та виправляють помилки.
- Rider надає додаткову підтримку для розробки ігор на платформі Unity.
- JetBrains надає студентам можливість використовувати Rider безкоштовно за спеціальною ліцензією.

Але і у цієї IDE є ряд своїх недоліків про які варто знати:

- Крос-платформенність: Хоча Rider є крос-платформеним, деякі користувачі вказують на те, що на деяких платформах він може виявлятися менш оптимізованим чи менш стабільним.
- Великий обсяг ресурсів: Зауважується, що Rider може вимагати значних ресурсів комп'ютера, що може впливати на продуктивність на менш потужних системах.
- Інтеграція з Azure та іншими продуктами Microsoft: Хоча Rider підтримує певну інтеграцію з платформою Microsoft, у порівнянні з Visual Studio відзначається меншою підтримкою деяких продуктів та сервісів Microsoft.
- Вартість платних версій: Хоча є можливість використовувати Rider безкоштовно для окремих користувачів, платні версії можуть виявитися дорожчими для деяких розробників порівняно із забезпеченням Microsoft.
- Менша кількість шаблонів проектів: Деякі розробники відзначають, що у Rider менша кількість шаблонів проектів порівняно з Visual Studio, що може затримати початківців.
- Перехід від інших IDE: Для тих, хто вже звик до інших інтегрованих середовищ розробки, перехід до Rider може вимагати часу для освоєння нового інтерфейсу та функціоналу.
- Обмежена підтримка деяких технологій: У порівнянні з Visual Studio, Rider може виявлятися менш підтриманим у деяких технологіях чи мовах програмування.

Судячи з наданої вище інформації можу зазначити, що Rider більш корисний для користувачів, які мають Mac та розроблюють системи без використання Azure. Із власного досвіду можу сказати, що не дивлячись на те, що Rider займає великий обсяг ресурсів, але він запускається дещо швидке аніж Visual Studio, у таблиці 3.2 наведено порівняння цих IDE.

Таблиця 3.2.

Порівняння швидкість відгуку Visual Studio та Rider

	Visual Studio 2022 with no extension	Visual Studio 2022 with Resharper extension	Rider 2022.1.2
Запуск, доки початкове вікно не реагує	**4 сек**	6 сек	5 сек
Запуск з готовим рішенням: адаптивний редактор коду та оглядач рішень	17 сек	25 сек	**13 сек**
Перезборка всього рішення	**22.2 сек**	22.3 сек	23.4 сек
Запуск із рішенням NopCommerce: адаптивний редактор коду та Solution Explorer	18 сек	24 сек	**13 сек**
Перебудувати все рішення NopCommerce	49 сек	52 сек	Перебудувати все рішення NopCommerce (після чистого рішення)

Тому було обрано Rider (рис. 3.2) для розробки веб-системи створення, редагування та збору аналітики про авіаперельоти. У цьому проєкті не використовується інтеграція з Azure та із власного досвіду Rider більше зручний у розробці веб-застосунків та систем.

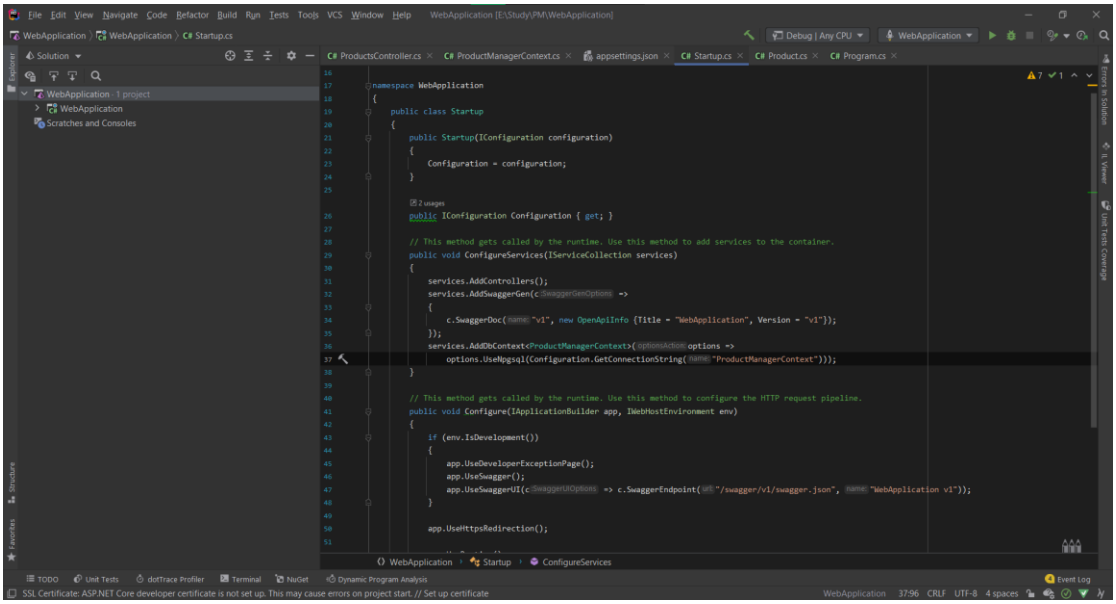

Рис. 3.2. Інтерфейс JetBrains Rider

3.1.2. Технології ФЕ

Для розробки користувацького інтерфейсу за основу було обрано React з використанням мови Typescript та HTML/CSS.

Обираючи інструмент для розробки користувацького інтерфейсу (UI), такого як React, треба враховувати кілька переваг цієї бібліотеки:

- React використовує декларативний синтаксис, що полегшує створення та розуміння коду. Ви описуєте, як повинен виглядати ваш інтерфейс у різних станах, і React бере на себе управління оновленням DOM.
- React використовує віртуальний DOM, що дозволяє ефективно оновлювати лише ті елементи, які змінилися, замість повного перерендерингу. Це призводить до високої продуктивності та швидкої реакції на зміни.
- React побудований на основі компонентної архітектури, де інтерфейс розділяється на невеликі та самодостатні компоненти. Це полегшує управління та модифікацію коду, а також сприяє повторному використанню компонентів.

- React має велику та активну спільноту розробників. Це означає, що ви можете легко знаходити допомогу, різноманітні плагіни та інструменти, які полегшать розробку.
- React широко використовується в індустрії та має підтримку від багатьох компаній. Це означає, що навчання та реалізація React можуть бути вигідними з точки зору кар'єрного росту.
- Існує багато інструментів та розширень для розробки на React, таких як React DevTools, які допомагають відлагоджувати та оптимізувати ваш код.
- React можна використовувати з різними технологіями, такими як Redux для управління станом, React Router для роботи з маршрутами, і інші.

Обираючи React, ви отримуєте ефективний та гнучкий інструмент для створення модерних та швидких користувацьких інтерфейсів. Він відрізняється від більшості конкурентів, що це саме бібліотека, а не фреймворк і наповнення цього інструменту залежить від вимог, а не як із Angular, де отримуєш одразу всі доступні можливості, які можеш навіть не використовувати у своєму проєкті.

Але все ж таки розглянемо недоліки цього інструменту:

- Деякі розробники вказують на те, що розмір React може бути великим для деяких проектів, особливо якщо ви використовуєте його разом із багатьма іншими бібліотеками та залежностями.
- Для деяких початківців React може здатися складним. Концепції, такі як віртуальний DOM, компоненти та стейт, можуть потребувати часу для освоєння.
- Використання єдиного об'єкту стану може призвести до складнощів у великих додатках, особливо коли стан стає складним та глибоко вкладеним.

- React не надає офіційного рішення для управління глобальним станом додатка, і для цього часто використовується додаткова бібліотека, така як Redux.
- Для використання React вам потрібно комбінувати JavaScript та JSX (розширений синтаксис JavaScript), що може вимагати адаптації та привести до помилок, особливо для тих, хто не знайомий із JSX.
- React рекомендує використовувати невзаємодіючий підхід для зміни стану, що може зробити код менш інтуїтивним та більш складним.
- Деякі розробники вказують на те, що для ефективного використання React може бути потрібно велику кількість налаштувань та конфігурацій, що може збільшити час настройки проекту.
- React не має вбудованого рішення для управління станом, що може вимагати використання сторонніх бібліотек та підходів для управління складним станом додатка.

Під час розробки було зроблено рішення використовувати TypeScript замість звичайного JavaScript і ось чому:

- TypeScript додає статичну типізацію до JavaScript, що дозволяє визначати типи змінних, параметрів та інших об'єктів на етапі розробки. Це допомагає виявляти помилки та покращує розуміння коду.
- TypeScript підтримує всі сучасні функції та стандарти JavaScript, а також дозволяє використовувати нові фічі до їх офіційного виходу.
- TypeScript має багатий функціонал об'єктно-орієнтованого програмування, що полегшує створення та підтримку об'єктно-орієнтованого коду.
- TypeScript можна використовувати в будь-якому проекті, який використовує JavaScript. Ви можете почати з декількох файлів

TypeScript у вже існуючому проекті та поступово переходити до повного використання TypeScript.

- TypeScript надає додаткові можливості для редагування коду, такі як автодоповнення, підказки та рефакторинг, що робить розробку більш продуктивною.
- Визначення типів допомагає в розумінні та читабельності коду. Коли ви читаєте код, ви можете легко визначити типи даних та параметрів функцій.
- Багато популярних інструментів розробки та IDE (Integrated Development Environment), таких як Visual Studio Code, мають вбудовану підтримку TypeScript, що полегшує розробку.

Також було використано технологію Redux. Redux - це бібліотека для управління станом (state management) в додатках на основі JavaScript та React. Вона часто використовується в додатках з великою кількістю взаємодій із станом, таких як односторінкові додатки (SPA) або додатки на основі React.

Основні концепції та складові Redux:

1. Store (Сховище):

- Це об'єкт, що утримує стан вашого додатка. Зміни в стані відбуваються лише через спеціальні об'єкти, відомі як "дії" (actions).

2. Actions (Дії):

- Дії є об'єктами, які описують зміни в стані. Вони відправляються до сховища і спричиняють оновлення стану.

3. Reducers (Редуктори):

- Редуктори - це функції, які обробляють дії і визначають, як саме стан додатка повинен бути оновлений. Кожен редуктор відповідає за певну частину стану.

4. Middleware (Проміжники):

- Проміжники - це шари логіки, які виконуються між відправленням дії та обробкою її редуктором. Вони можуть використовуватися для логування, асинхронних операцій тощо.

5. Selectors (Виборці):

- Виборці - це функції, які використовуються для вибору конкретних частин стану.

Чому Redux використовується:

- Redux зберігає стан додатка в одному централізованому сховищі, що полегшує відслідковування та управління станом.
- Зміни в стані відбуваються тільки через дії, що робить процес розробки передбачуваним та допомагає уникнути непередбачених змін.
- Логіка керування станом відокремлюється від компонентів, що полегшує управління та тестування.
- Дії та зміни стану можна легко відстежувати та аналізувати, що полегшує виявлення та усунення помилок.
- Редуктори можна комбінувати та розширювати, що дозволяє легко розширювати функціональність додатка.

Redux допомагає вирішити проблеми керування станом у великих та складних додатках, зроблюючи код більш організованим, підтримуючи прозорість та предсказуваність змін стану.

Для написання коду ФЕ було використано також IDE з сім'ї JetBrains – WebStorm. У загальному це було зроблено через оформлення студентської ліцензії на продукти JetBrains та через зручність, бо інтерфейси цих IDE схожі та інтуїтивно зрозумілі (рис. 3.3).

WebStorm - це інтегроване середовище розробки (IDE) для веб-розробників, розроблене компанією JetBrains. Ось деякі переваги використання WebStorm:

- WebStorm надає повну підтримку JavaScript, включаючи ES6, TypeScript, а також популярні фреймворки, такі як React, Angular, Vue.js, і інші.

- Можливість автоматично завершувати код та отримувати інтелектуальні підказки полегшує роботу розробників та зменшує ймовірність помилок.
- WebStorm включає в себе вбудовані інструменти для роботи з системою контролю версій Git, що дозволяє вам легко взаємодіяти з репозиторіями, вносити зміни та вирішувати конфлікти.
- Веб-Storm дозволяє відлагоджувати код як локально, так і на віддаленому сервері, що полегшує виявлення та виправлення помилок.
- Окрім JavaScript, WebStorm має підтримку HTML та CSS, забезпечуючи розширені можливості редагування для всіх важливих мов веб-розробки.
- IDE підтримує багато мов та технологій, що дозволяє розробникам працювати з різноманітними проектами.
- WebStorm дозволяє використовувати Live Templates та Code Snippets для автоматизації рутинних завдань та прискорення написання коду.
- IDE надає засоби для тестування коду, включаючи інструменти для запуску та аналізу тестів.
- WebStorm підтримує Node.js, дозволяючи легко розробляти серверні застосунки та взаємодіяти з серверним середовищем.
- Широкий вибір розширень та плагінів дозволяє розширювати функціональність IDE за потреби розробника.

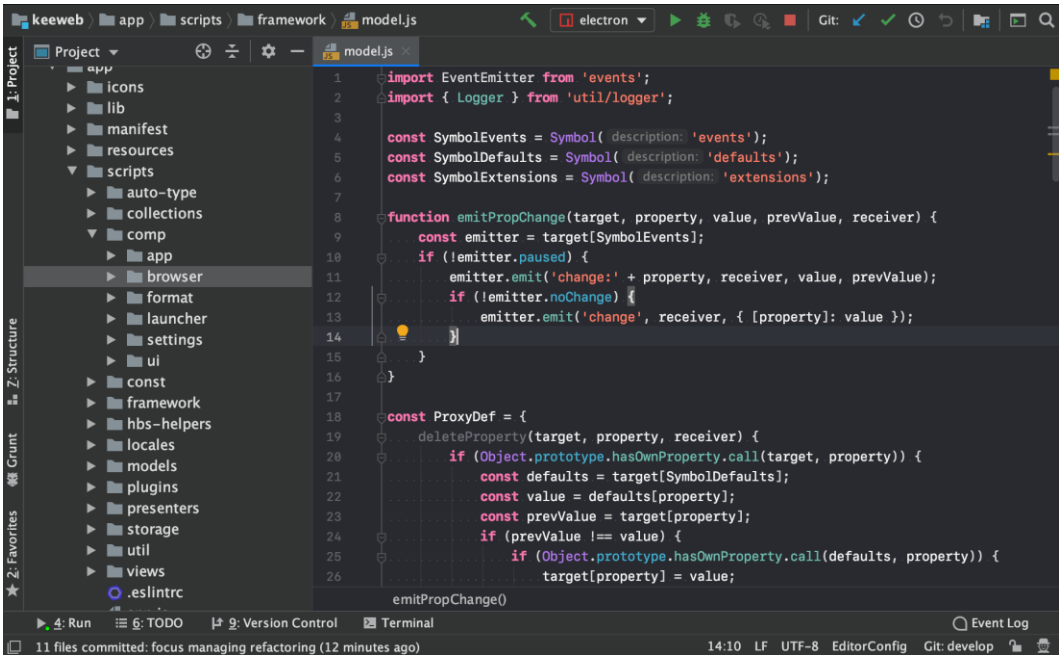

Рис. 3.3. Інтерфейс JetBrains WebStorm

3.1.3. База даних

Одним із важливих інструментів при розробці веб-системи є база даних, тому під час вибору її потрібно проаналізувати аналоги та обрати ту, яка найбільше підійде під вимоги майбутньої системи.

Існує кілька типів баз даних в залежності від їх структури та організації даних. Основні типи баз даних включають:

1. Реляційні бази даних (RDBMS) – вони базуються на моделі реляційної бази даних, де дані організовані у вигляді таблиць зі зв'язками між ними. Найпопулярнішим представником є MySQL, PostgreSQL, Microsoft SQL Server, Oracle.

2. Нереляційні бази даних (NoSQL) – ці бази даних не використовують традиційні таблиці зі зв'язками. Вони можуть бути документ-орієнтованими (наприклад, MongoDB), ключ-значеннями (наприклад, Redis), стовпчастими (наприклад, Apache Cassandra) чи графовими (наприклад, Neo4j).

3. Інтерактивні бази даних – це бази даних, які спроектовані для обробки запитів користувачів у реальному часі. До прикладів відносяться бази даних, які використовуються для систем управління великими мережами та комунікаційними системами.

4. Об'єктно-орієнтовані бази даних – вони орієнтовані на роботу з об'єктами, де дані представлені як об'єкти. Це може бути корисним для додатків, які використовують об'єктно-орієнтований підхід в програмуванні.

5. Текстові бази даних – вони орієнтовані на зберігання та обробку текстової інформації. Такі бази даних можуть використовуватися для повнотекстового пошуку або обробки великих об'ємів текстової інформації.

6. Інтернет-орієнтовані бази даних – ці бази даних оптимізовані для використання у великих веб-застосунках. Вони можуть включати в себе технології, такі як розподілена обробка та можливості шкалювання.

7. Вбудовані бази даних – це бази даних, які вбудовані безпосередньо в додаток чи програмне забезпечення, що використовує їх для зберігання та обробки даних. Наприклад, SQLite є популярною вбудованою базою даних.

8. Децентралізовані бази даних – це бази даних, де управління та зберігання даних розподілені між різними вузлами чи серверами. Це може використовуватися для створення більш стійких та масштабованих систем.

Після аналізу всіх типів баз даних було обрано реляційну модель, бо використання реляційної бази даних (RDBMS) має багато плюсів. Їх розглянемо нижче.

Першою перевагою є те, що реляційні бази даних забезпечують структуроване управління даними у вигляді таблиць, що спрощує створення, зміну та запити даних. Вони дозволяють застосовувати нормалізацію, що сприяє уникненню дублювання даних та збереженню цілісності бази даних. А це важливо, бо зберігання зайвих даних має свої наслідки, а саме, те що база даних буде розростатись і на підтримку її потрібно буде більше коштів.

SQL (Structured Query Language) використовується для взаємодії з реляційними базами даних, і він є мовою, яка легко зрозуміла та вивчається, спрощуючи створення запитів до даних, а також використовуючи Entity Framework для роботи з базою даних запити створювати та підтримувати ще простіше.

Запити SQL дозволяють виконувати різні види запитів, такі як вибірка (SELECT), вставка (INSERT), оновлення (UPDATE) та видалення (DELETE), що робить можливим роботу з різноманітними вимогами до даних.

Реляційні бази даних дозволяють використовувати індексацію, що поліпшує швидкість виконання запитів та пошуків. Індекси у свою чергу поділяються на кластеризовані та некластеризовані.

Кластеризований забезпечує фізичний порядком обраним полем, такий індекс може бути тільки один у таблиці і сортує таблиці лише одним способом.

Некластеризований індекс зберігає дані в одному місці, а індекси - в іншому. Індекс містить вказівники на розташування цих даних. В одній таблиці може бути багато некластеризованих індексів, оскільки індекс у некластеризованому індексі зберігається в різних місцях.

Індекс некластеризації визначений у полі упорядкування таблиці. Цей тип методу індексації допомагає покращити продуктивність запитів, які використовують ключі, які не призначені як первинний ключ. Некластеризований індекс дозволяє додавати унікальний ключ для таблиці.

RDBMS підтримують концепцію транзакцій, яка забезпечує атомарність, консистентність, ізольованість та довіреність (ACID properties). Це гарантує правильне відновлення даних у випадку помилок чи відміни транзакцій. Для початку треба розібратись, що таке транзакція і для чого вона потрібна.

Транзакція — це єдина логічна одиниця роботи, яка отримує доступ і, можливо, змінює вміст бази даних. Транзакції отримують доступ до даних за допомогою операцій читання та запису. Після чого краще більш детально описати ACID абревіатуру, де кожна буква відповідає за окрему ознаку транзакції.

A(Atomicity) – атомарність, під цим мається на увазі, що або вся транзакція відбувається відразу, або не відбувається взагалі. Проміжного шляху немає, тобто транзакції не відбуваються частково. Кожна транзакція вважається одним цілим і виконується до кінця, або не виконується взагалі. Він включає наступні дві операції:

- Abort: якщо транзакція переривається, зміни, внесені до бази даних, не відображаються.
- Commit: якщо транзакцію закріплено, внесені зміни видно.

Атомарність також відома як правило «все або нічого».

C(Consistency) – консистентність даних, це означає, що необхідно підтримувати обмеження цілісності, щоб база даних була узгодженою до і після транзакції. Це відноситься до коректності бази даних.

I(Isolation) – ізоляція, ця властивість гарантує, що кілька транзакцій можуть відбуватися одночасно, не призводячи до неузгодженості стану бази даних. Транзакції відбуваються самостійно без стороннього втручання. Зміни, що відбуваються в конкретній транзакції, не будуть видимі для жодної іншої транзакції, доки ця конкретна зміна в цій транзакції не буде записана в пам’ять або не буде зафіксовано. Ця властивість гарантує, що одночасне виконання транзакцій призведе до стану, еквівалентного досягнутому стану, коли вони виконувалися послідовно в певному порядку.

D(Durability) – довговічність, ця властивість гарантує, що після завершення виконання транзакції оновлення та модифікації бази даних зберігаються на диску та записуються на диск, і вони зберігаються навіть у разі збою системи. Тепер ці оновлення стають постійними та зберігаються в енергонезалежній пам’яті. Таким чином, наслідки транзакції ніколи не втрачаються.

Реляційні бази даних забезпечують рівень безпеки, включаючи можливість визначення рівнів доступу для користувачів та ролей. Використання схеми даних дозволяє визначати структуру бази даних, включаючи таблиці, поля та відносини між ними. Це полегшує розуміння та підтримку бази даних.

Після вибору типу бази даних потрібно обрати, яку саме із представлених на ринку слід обрати? Найбільш поширені бази це:

- MySQL: відкрита реляційна база даних, яка використовує мову SQL. Є частиною Oracle Corporation.

- PostgreSQL: об'єктно-реляційна база даних з відкритим вихідним кодом, яка підтримує розширення SQL та багато інших функцій.
- Microsoft SQL Server: реляційна база даних від Microsoft, яка пропонує різні версії для різних потреб, включаючи підтримку підприємств та розробки.
- Oracle Database: одна з найбільш розповсюджених реляційних баз даних, яку розробляє компанія Oracle. Використовує мову SQL та має багато розширених можливостей.
- SQLite: вбудована реляційна база даних, яка зазвичай використовується для невеликих проектів та мобільних додатків.

Було обрано використовувати PostgreSQL, бо вона має відкритий вихідний код, для її використання не потрібно купувати ліцензію і є досвід використовування її у проєктах.

Для того щоб працювати з БД є декілька різних підходів, але для мене більш зручний та простий – через IDE, як і з кодом для ФЕ та БЕ. Тому потрібно було обрати потрібну IDE.

Із основних IDE, які є на ринку більше всього підходили JetBrains DataGrip, Dbeaver, PgAdmin. Для початку я проаналізував для чого кожна із програм:

DataGrip:

- Розробник: JetBrains.
- Інтелектуальний редактор SQL.
- Підтримка багатьох систем управління базами даних (СУБД), включаючи PostgreSQL, MySQL, SQLite та інші.
- Інтеграція з іншими продуктами JetBrains.
- Зручний інтерфейс та інтуїтивне використання.
- Відладка SQL-запитів.

DBeaver:

- Розробник: DBeaver Corp.

- Універсальна платформа для роботи з базами даних, яка підтримує багато СУБД.
- Відкритий вихідний код.
- Графічний інтерфейс для роботи зі схемами баз даних.
- Підтримка великої кількості драйверів баз даних.
- Можливості експорту та імпорту даних.

pgAdmin:

- Розробник: The pgAdmin Development Team.
- Адміністративний інтерфейс для PostgreSQL.
- Вбудована підтримка великої кількості функцій для адміністрування та моніторингу PostgreSQL.
- Відкритий вихідний код.
- Можливість виконання SQL-запитів та перегляд даних.
- Зручний інтерфейс для керування користувачами, таблицями та іншими об'єктами бази даних.

Порівняємо цільову аудиторію програм – DataGrip та DBeaver позиціонуються як універсальні інструменти для роботи з різними СУБД, в той час як pgAdmin спеціалізується на адмініструванні PostgreSQL.

Також не варто забувати, що більшість програм можна використовувати лише із ліцензією, тому на це варто звертати увагу. DataGrip та DBeaver мають комерційні та безкоштовні версії, в той час як pgAdmin є відкритим вихідним кодом. Виходить, що кожну із програм можливо використовувати безкоштовно, але DataGrip має лише тріальну версію безкоштовною, якщо не оформити підписку на JetBrains використовуючи студентський.

Судячи із опису маємо дуже схожі програми, потрібно розглянути різницю у функціональності. DataGrip та DBeaver можуть бути використані для роботи з різними СУБД, включаючи PostgreSQL. Вони надають інтегровані інструменти для відладки та управління проектами. pgAdmin спеціалізується на PostgreSQL та надає багато функцій для його адміністрування.

Далі порівняємо інтерфейс цих систем(рис. 3.4 – 3.6). Всі три інструменти мають зручний графічний інтерфейс, але можуть відрізнятися за структурою та стилем використання.

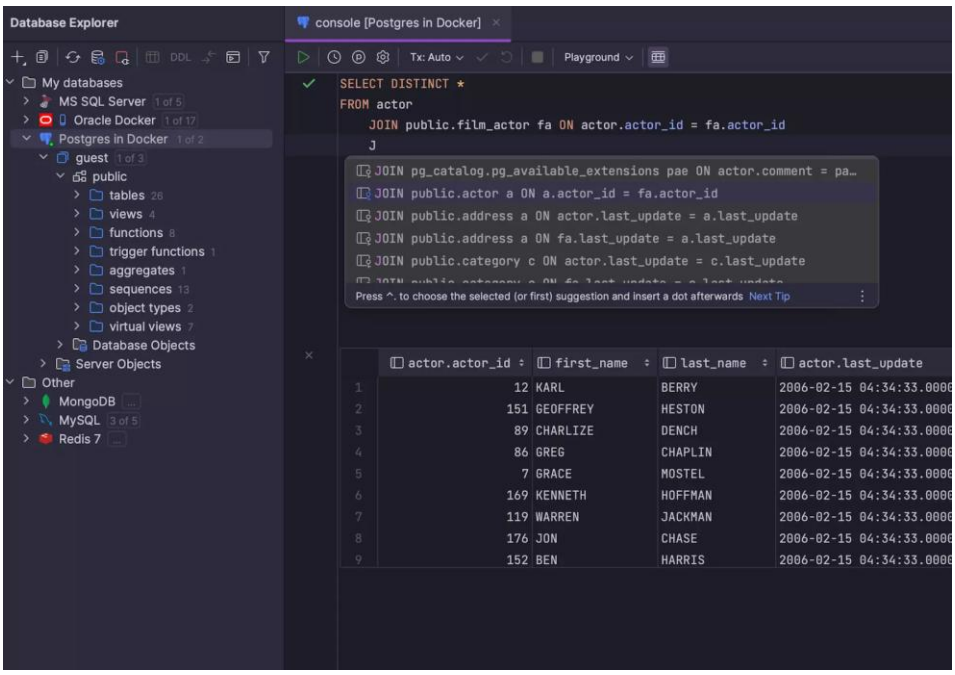

Рис. 3.4. Графічний інтерфейс DataGrip

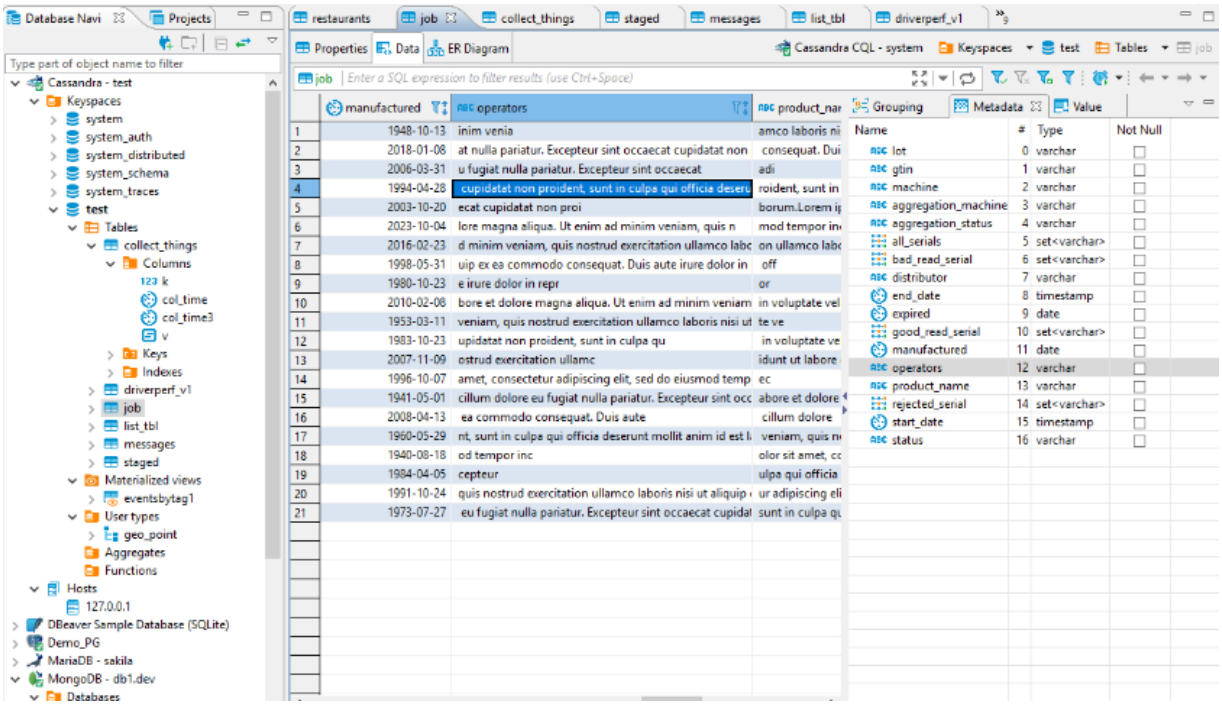

Рис. 3.5. Графічний інтерфейс DBeaver

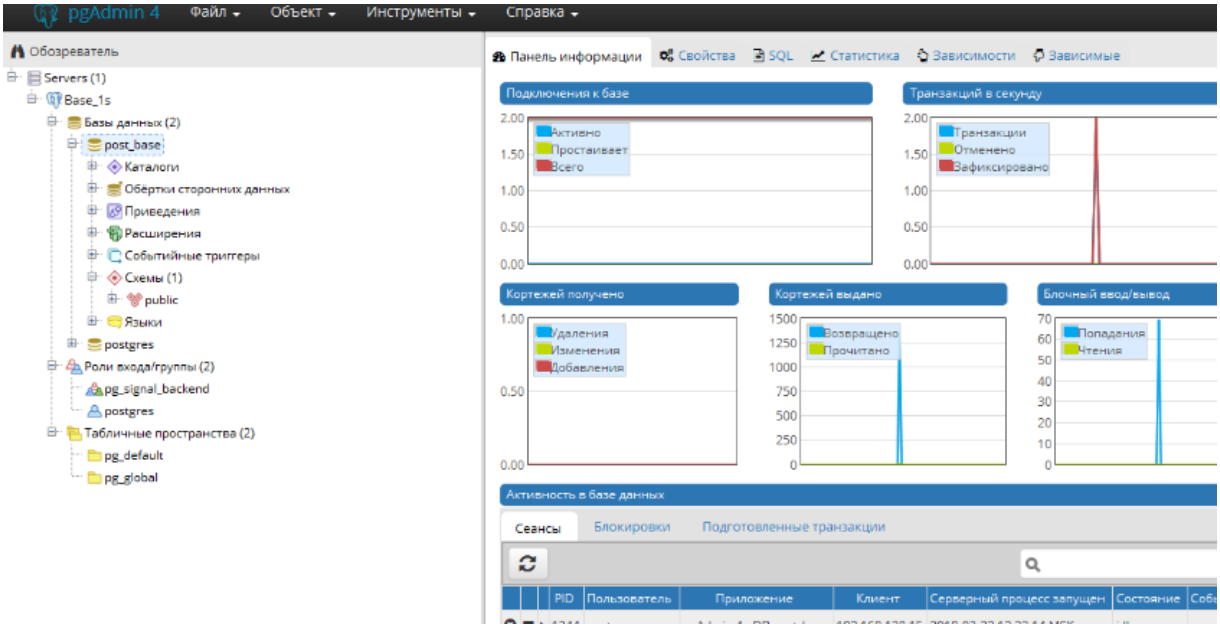

Рис. 3.6. Графічний інтерфейс pgAdmin

DataGrip та DBeaver мають певні можливості інтеграції з іншими інструментами розробки, такими як IDE від JetBrains. pgAdmin зазвичай використовується самостійно для адміністрування PostgreSQL.

DataGrip та DBeaver підтримують багато різних СУБД, в той час як pgAdmin орієнтований на PostgreSQL.

Вибір між цими інструментами був очевидний, бо було обрано працювати із БД PostgreSQL, так як pgAdmin спеціалізується тільки на цій БД, то ця програма ідеально підійде.

3.1.4. Контейнери та Docker

Docker - це платформа для розробки, доставки та запуску застосунків у контейнерах. Контейнери є віртуальними середовищами, які утримують всі необхідні залежності для виконання програми, включаючи код, бібліотеки та інші налаштування. Docker надає зручний і стандартизований спосіб упаковувати та розгортати програми.

Основні концепції та складові Docker:

1. Контейнер - це запускний екземпляр образу, який містить програму та всі її залежності. Контейнери ізолюють програму від зовнішнього середовища, що полегшує перенос та розгортання.

2. Образ - це великий файл, що включає в себе код, бібліотеки, середовище виконання та всі інші необхідні ресурси для роботи програми. Образ використовується для створення контейнерів.

3. Dockerfile - це текстовий файл, що містить інструкції для побудови Docker-образу. Він описує кроки для установки та налаштування програми та її залежностей.

4. Docker Hub - це реєстр образів Docker, де можна знаходити та ділитися готовими образами. Також можна використовувати власні приватні реєстри.

Для чого Docker використовується:

1. Відокремленість та Ізоляція:

- Docker дозволяє ізолювати програми в контейнерах, забезпечуючи відокремленість середовища виконання. Це робить конфлікти між різними додатками менш імовірними.

2. Портативність:

- Контейнери Docker можна запускати на будь-якій системі, де встановлено Docker, що полегшує перенос та розгортання додатків між різними середовищами.

3. Швидкість Розгортання:

- Контейнери можна швидко створювати та запускати, що робить процес розгортання дуже ефективним.

4. Масштабованість:

- Docker спрощує процес масштабування додатків за рахунок використання контейнерів. Можна швидко запускати нові контейнери або відключати їх за потреби.

5. Легке Управління Залежностями:

- Docker дозволяє визначати та управляти залежностями програми в Dockerfile, забезпечуючи стандартизацію та консистентність.

Docker є потужним інструментом для розробки та розгортання додатків, особливо у великих та складних проектах.

Для серверної частини, яка розроблена на мові C# був створений DockerFile для подальшого розгортання системи на віддаленому сервері. Під час створення було використано останні образи для .NET 8, які можна легко знайти на офіційному сайті Mircosoft.

Також було додано docker-compose файл для розгортання БД на будь-якому пристрої. Docker Compose — це інструмент, який допомагає керувати декількома контейнерами Docker одночасно за допомогою правил, визначених у файлі docker-compose.yml. Завдяки цьому файлу можна створити повністю кастомізоване середовище з декількома контейнерами, які можуть спільно використовувати мережі та обсяги даних.

Якщо порівнювати DockerFile та Docker Compose, то Docker використовують, щоб управляти окремим сервісом (контейнером), з якого складається застосунок.

Docker Compose використовують, щоб одночасно управляти декількома контейнерами одного і того ж застосунку. У Docker Compose такі ж можливості як в Docker, але він дозволяє працювати з набагато складнішими застосунками.

Для того аби локально підняти ці файли треби виконати декілька команд:

- Для docker-compose – це “docker-compose up” у папці із відповідним файлом;
- Для DockerFile спочатку треба побудувати образ “docker build -t avian-server .”, а потім його запустити “docker run -dp 127.0.0.1:3000:3000 avian-server”, де ‘avian-server’ це назва образу, який ми збудували, а 3000:3000 це порти за якими буде доступний наш сервер

3.2. Створення програмного застосунку

Веб-система для створення, редагування та збору аналітики про авіаперельоту буде називатись “Avian”. Проєкт буде зберігатись на GitHubі у публічному доступі.

GitHub – це веб-платформа для розробки програмного забезпечення, яка надає засоби для керування версіями коду, співпраці розробників та ведення проектів.

Основні характеристики та функції GitHub включають:

- Репозиторії - це місця для зберігання та керування проектами. Кожен проект може мати свій репозиторій.
- Вбудований інструмент для керування версіями коду, що дозволяє відстежувати зміни, створювати гілки та об'єднувати їх (merge).
- Розробники можуть створювати гілки для ізоляції нового функціоналу чи виправлення помилок та об'єднувати їх після завершення.
- Розробники можуть пропонувати зміни (pull requests), ініціювати обговорення та запроси на об'єднання коду в основну гілку.
- Надає API для взаємодії з платформою та підтримує багато інтеграцій з іншими інструментами.

Після чого було створено проєкт для БЕ у IDE Rider, рис. 3.7.

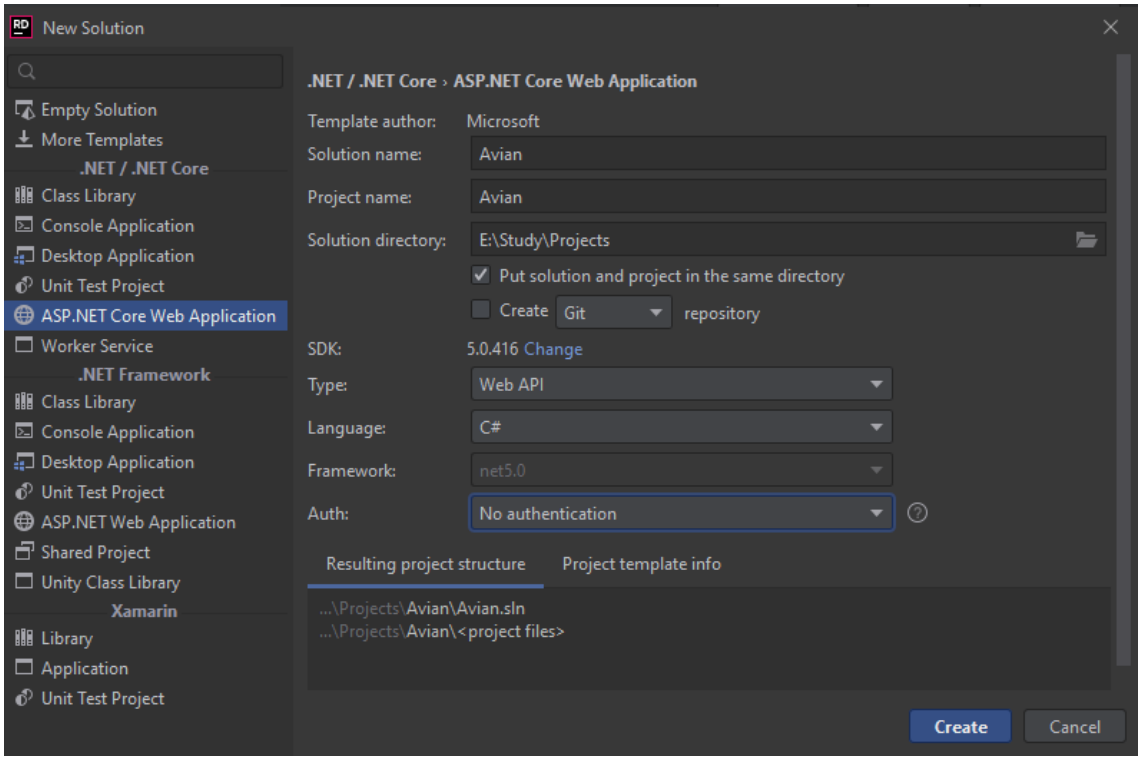

Рис. 3.7. Створення проєкту у Rider

Після чого було обрано рішення створення багато-шарової архітектури, проєкт поділено на 4 частини(рис. 3.8):

- Avian – проєкт на якому розташовані всі контроллери та налаштування системи;
- Avian.Application – проєкт на якому виконується логіка системи, така як, витягування даних з бази, валідація даних, тощо;
- Avian.Dal – проєкт на якому лежать моделі, які знаходяться у базі даних і тут знаходиться налаштована база даних, а також лежать міграції до неї;
- Avian.Domain – проєкт доменних моделей, у доменних моделях лежить вся бізнес логіка, у подальшому для написання тестів на бізнес логіку потрібно буде тестувати либо об’єкти з цього шару.

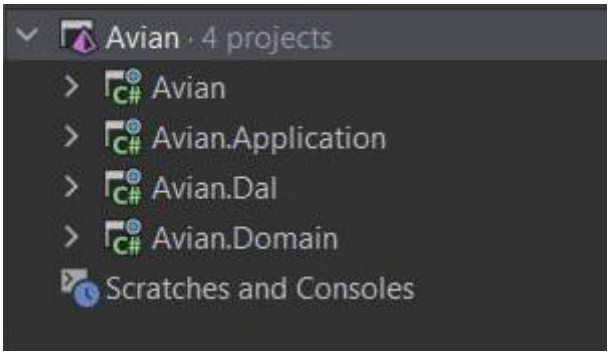

Рис. 3.8. Структура рішення

Першим ділом було створено логіку роботи користувача, а саме – авторизація, доменна модель користувача, моделі для бази, першим кроком була створена та протестована функціональність авторизації, скрін авторизації на рис. 3.9.

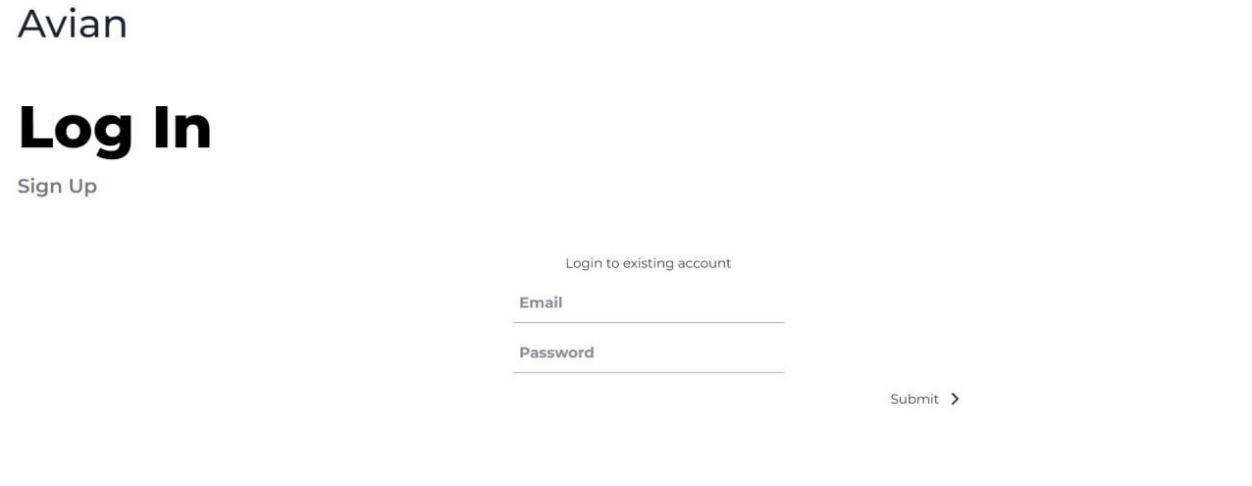

Рис. 3.9. Екран авторизації

Для авторизації користувач вводить пошту та пароль і якщо дані вірні система його авторизує. Функція реєстрації доступна лише для адміністраторів, після авторизації у адміністратор з’являється можливість створювати акаунти через модальне вікно, рис. 3.10.

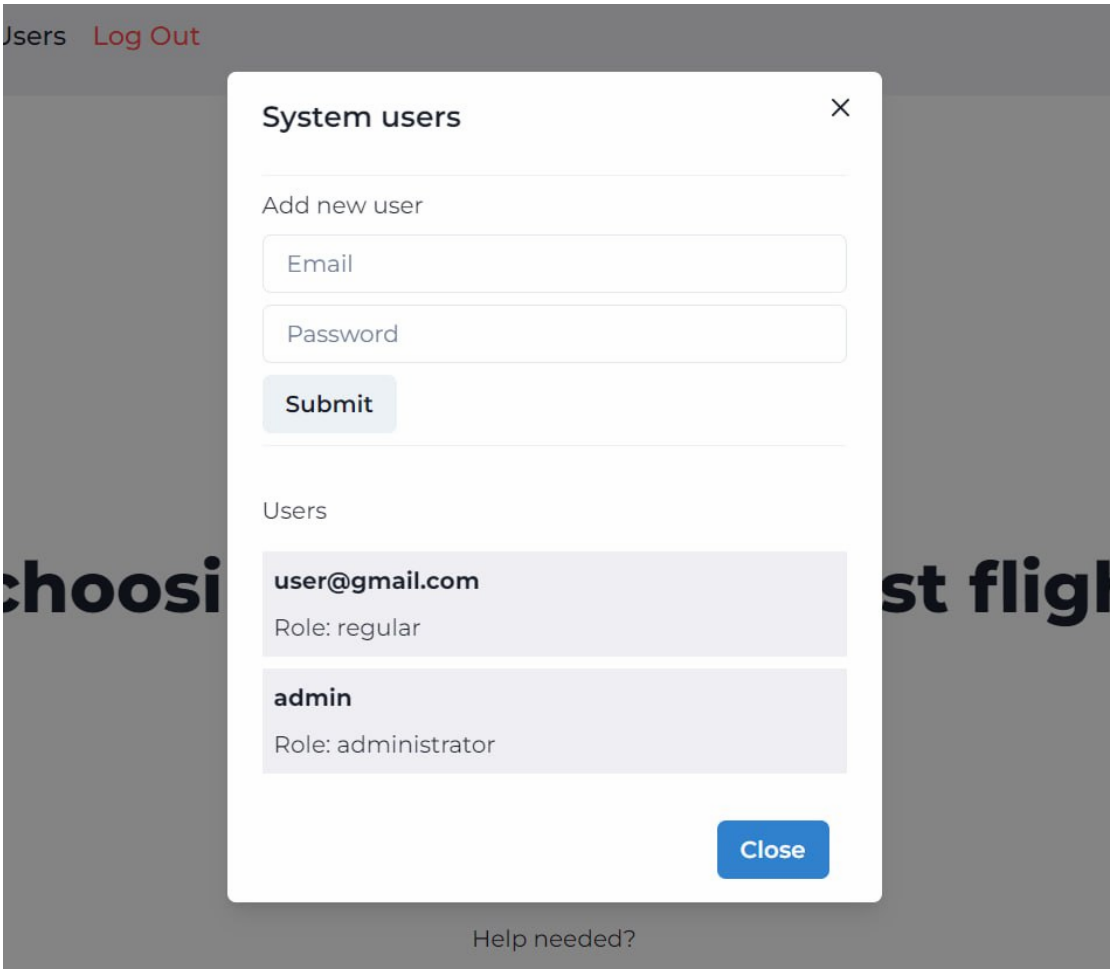

Рис. 3.10. Створення акаунта користувача

Після авторизації користувач потрапляє на головний екран, він може побачити навігаційну зону зверху екрана, рис. 3.11.

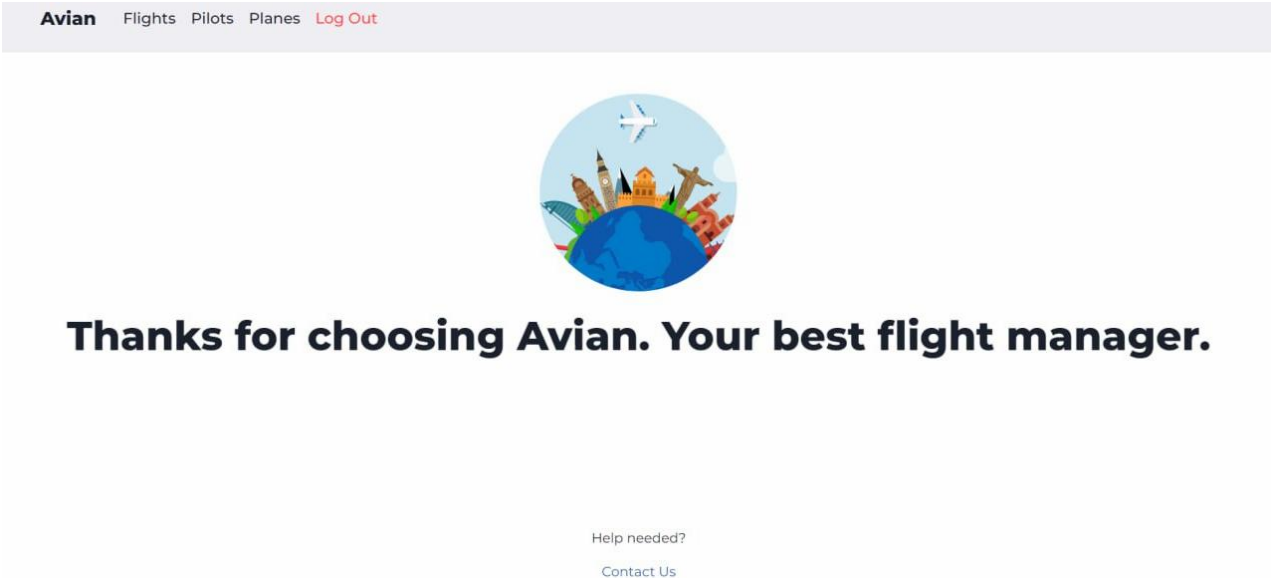

Рис. 3.11. Головний екран

При переході на вкладку Flights користувач попадає на сторінку з рейсами, де може створити рейс, переглядати список рейсів, видаляти та змінювати статус, рис. 3.12.

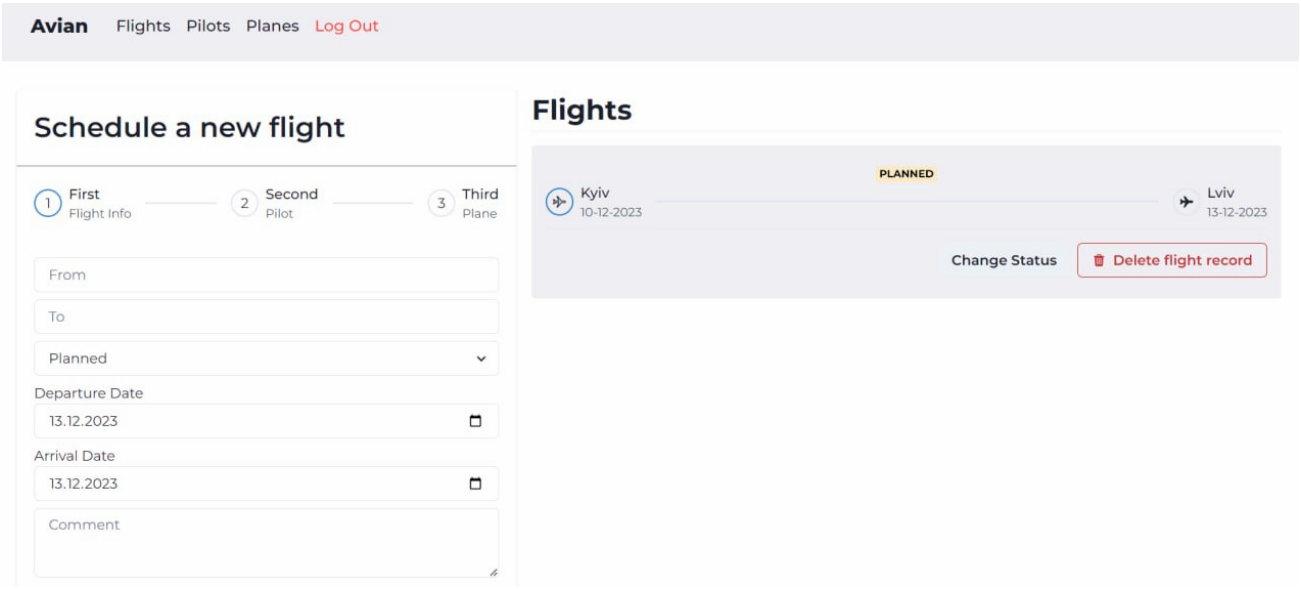

Рис. 3.12. Сторінка рейсів

При переході на вкладку Pilots користувач потрапляє на сторінку з пілотами, де може створити пілота, видалити та переглядати список пілотів, рис. 3.13.

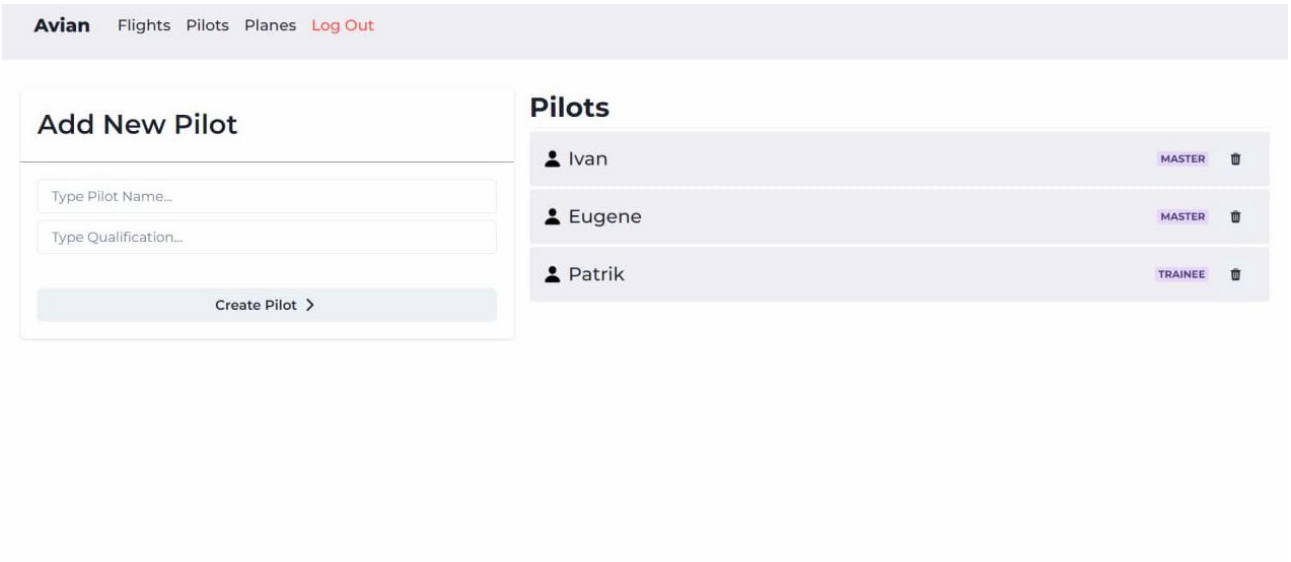

Рис. 3.13. Сторінка пілотів

При натисканні кнопки Planes користувача перенаправляє на сторінку літаків, де він може створити літак, видалити та переглядати список літаків, рис. 3.14.

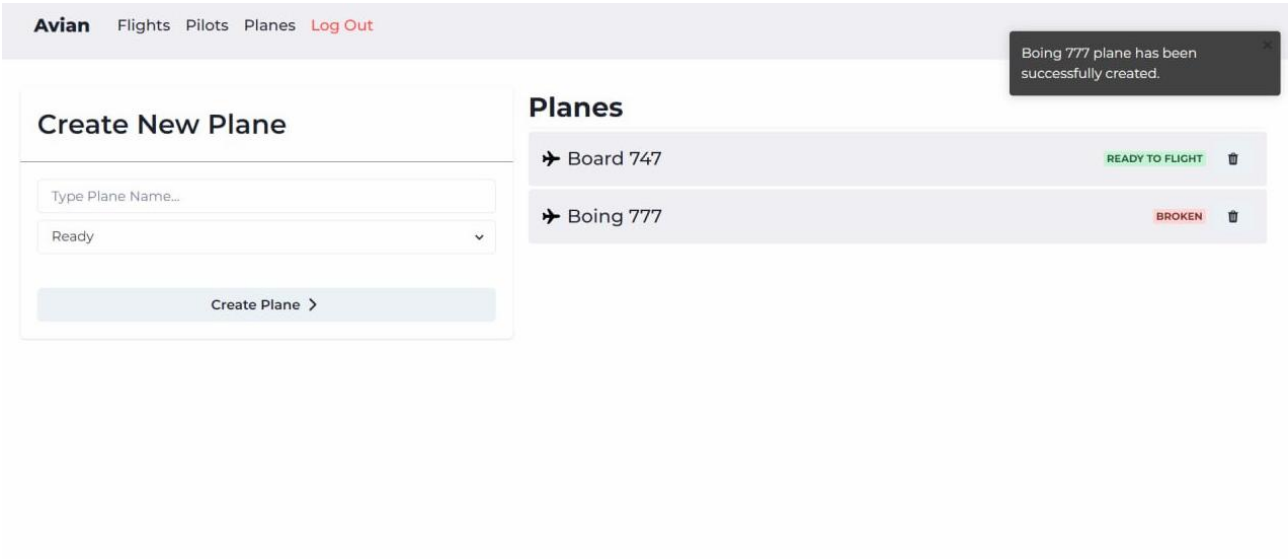

Рис. 3.14. Сторінка літаків

Для того аби користувач вийшов із системи потрібно натиснути червону кнопку “Log Out”, після чого підтвердити свою дію на модальному вікні і користувача вилогінить із системи, рис. 3.15. Після виходу із системи користувач потрапляє на вікно, яке зображене на рис. 3.9.

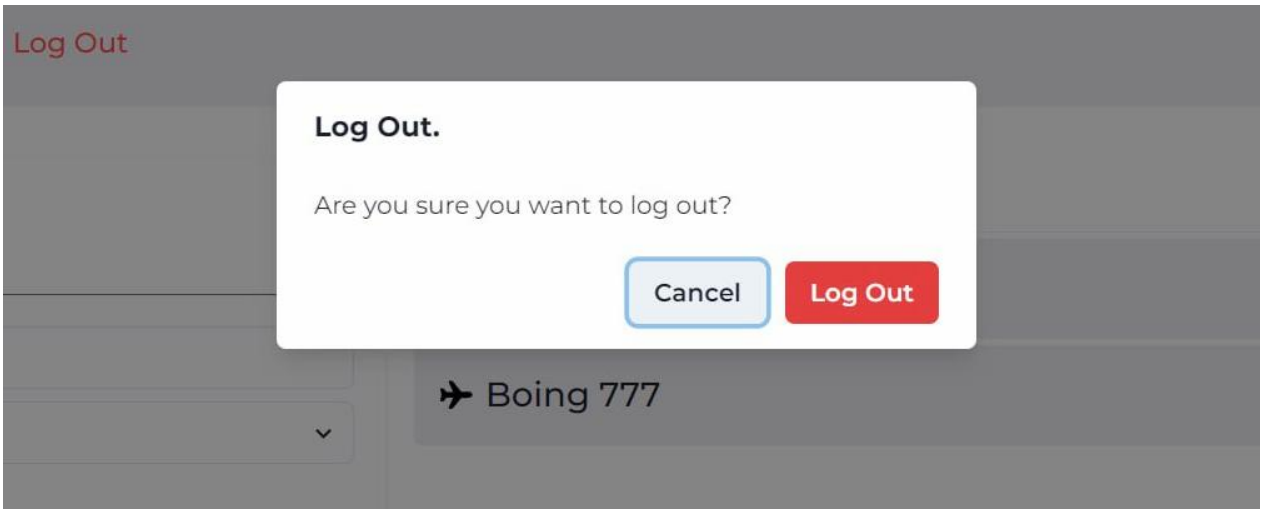

Рис. 3.15. Модальне вікно для виходу із системи

3.3. Висновки до розділу

У останньому третьому розділі було проаналізовано та обрано технології для розробки веб-системи створення, редагування та збору аналітики про авіаперельоти. Спочатку було описано переваги мов програмування, які було обрано. Після чого проаналізовано та обрано IDE у котрому треба було розроблювати систему, було описано і розібрано аналоги та аргументовано чому саме цей IDE було обрано.

У другій частині розділу був описаний процес створення самої системи, описано систему та представлені скріншоти розроблених екранів інтерфейсу готового продукту.

ВИСНОВКИ

У першому розділі було проведено аналіз предметної області, а саме, авіаперельотів та авіакомпаній, проаналізовано актуальність створення веб-системи для створення, редагування та збору аналітики про польоти для авіакомпаній, знайдено та проаналізовано існуючі аналоги, а також визначено необхідні характеристики майбутнього веб-застосунку і проведено дослідження на відповідність існуючих аналогів. Крім цього було проаналізовано та описано фінансову складову авіакомпаній, як складову доменної області.

У другому розділі було описано вимоги до веб-системи для створення, редагування та збору аналітики про польоти для авіакомпаній, спершу описано користувацькі вимоги, нефункціональні та функціональні вимоги – створено діаграми прецедентів, сценарії використання системи користувачами, описано вимоги для пристроїв користувачів аби вони могли зручно та ефективно користуватись системою.

У третьому розділі було проаналізовано, описано та обрано інструменти для розробки веб-системи для створення, редагування та збору аналітики про польоти для авіакомпаній. Було визначено, які технології будуть використовуватись при створенні системи, а саме – для серверної частини це – ASP.NET Core з використанням мови програмування C# та бібліотека React для розробки інтерфейсу з використанням мови програмування TypeScript та використанням технології Redux, також у ролі БД було обрано PostgreSQL. Також було обрано середу розробки, як для FE так і для BE, і для БД.

Наступним кроком була сама розробка веб-системи і реалізація основного функціоналу, а саме, авторизація, створення, редагування, видалення та перегляд рейсів, створення та видалення літаків та пілотів, створення акаунту адміністратором, вихід з системи.

Після чого розпочато реалізацію КІ, який взаємодіє з сервером. Було створено html сторінки на які було додано стилі для кращого вигляду системи,

також розроблено функціональність на React для роботи TypeScript із html-сторінками.

Останнім кроком було локальне тестування та опис створеної веб-системи.

Отже, мета роботи є досягнутою, всі завдання є реалізованими, система описана, доменна область проаналізована, а тому роботу можна вважати успішно завершеною.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Amadeus global information about company [Електронний ресурс]. – Відкритий доступ: https://amadeus.com/en/about/
2. Sabre global information about company [Електронний ресурс]. – Відкритий доступ: https://www.sabre.com/about/
3. Нефункціональні вимоги: приклади, типи, підходи [Електронний ресурс]. – Відкритий доступ: https://training.qatestlab.com/blog/technical-articles/non-functional-requirements-examples-types-approaches/
4. Бізнес-аналітики і нефункціональні вимоги в agile розробці [Електронний ресурс]. – Відкритий доступ: https://www.ba.in.ua/2023/08/18/biznes-analityky-i-nefunkczionalni-vymogy-v-agile-rozrobczi/
5. UML для бізнес-моделювання: для чого потрібні діаграми процесів [Електронний ресурс]. – Відкритий доступ: https://evergreens.com.ua/ua/articles/uml-diagrams.html
6. UML-діаграма прецедентів [Електронний ресурс]. – Відкритий доступ: https://gazette.com.ua/rizne/uml-diagrama-pretsedentiv.html
7. Системні вимоги [Електронний ресурс]. – Відкритий доступ: https://studfile.net/preview/4452595/page:9/
8. Бібліотека класів платформи .NET [Електронний ресурс]. – Відкритий доступ: http://repository.hneu.edu.ua/jspui/bitstream/123456789/23847/1/2019-%D0%A9%D0%B5%D1%80%D0%B1%D0%B0%D0%BA%D0%BE%D0%B2%20%D0%9E%20%D0%92%2C%20%D0%9F%D0%B0%D1%80%D1%84%D1%8C%D0%BE%D0%BD%D0%BE%D0%B2%20%D0%AE%20%D0%95%2C%20%D0%A4%D0%B5%D0%B4%D0%BE%D1%80%D1%87%D0%B5%D0%BD%D0%BA%D0%BE%20%D0%92%20%D0%9C.pdf
9. React documentation [Електронний ресурс]. – Відкритий доступ: https://legacy.reactjs.org/docs/getting-started.html

10. Main features for Dbeaver [Електронний ресурс]. – Відкритий доступ: https://dbeaver.com/features/
11. PgAdmin documentation [Електронний ресурс]. – Відкритий доступ: https://www.pgadmin.org/docs/pgadmin4/8.0/index.html
12. DataGrip documentation [Електронний ресурс]. – Відкритий доступ: https:/ https://www.jetbrains.com/datagrip/
13. ACID Properties in DBMS [Електронний ресурс]. – Відкритий доступ: https://byjus.com/gate/acid-properties-in-dbms-notes/#:~:text=In%20the%20transaction%20processing%20context,and%20correctness%20of%20any%20database.
14. ACID Properties in DBMS [Електронний ресурс]. – Відкритий доступ: https://www.geeksforgeeks.org/acid-properties-in-dbms/
15. Visual Studio vs. Jetbrains Rider Performance [Електронний ресурс]. – Відкритий доступ: https://blog.ndepend.com/visual-studio-vs-jetbrains-rider-performance/
16. Переваги та недоліки .NET [Електронний ресурс]. – Відкритий доступ: https://dou.ua/lenta/articles/pros-and-cons-of-dotnet/
17. Install SASS [Електронний ресурс]. – Відкритий доступ: https://sass-lang.com/install/
18. Основи Redux [Електронний ресурс]. – Відкритий доступ: https://devzone.org.ua/post/osnovi-redux
19. Завантаження NodeJs [Електронний ресурс]. – Відкритий доступ: https://nodejs.org/en
20. Найпростіші типи баз даних [Електронний ресурс]. – Відкритий доступ: https://senior.ua/articles/11-tipv-suchasnih-baz-danih-korotkiy-opis-shemi--prikladi-bd
21. 9 причин вивчити мову C# [Електронний ресурс]. – Відкритий доступ: https://foxminded.ua/9-prichin-vivchiti-movu-c/
22. Redux. Simple as a rake [Електронний ресурс]. – Відкритий доступ: https://habr.com/ru/articles/439104/

23. Чому топові компанії використовують Node.js [Електронний ресурс]. – Відкритий доступ: https://prjctr.com/mag/whynode
24. PostgreSQL [Електронний ресурс]. – Відкритий доступ: https://brander.ua/technologies/postgresql