МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ СИСТЕМ ТА МЕРЕЖ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри

_____ Ігор ЖУКОВ

«_____» _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ МАГІСТР

ЗА СПЕЦІАЛЬНІСТЮ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

Тема: «Система візуалізації та аналізу даних з рейсових записів польотів»

Виконавець: Денис КОСІНСЬКИЙ

Керівник: Ольга РУСАНОВА

Нормоконтролер: Василь МАЛЯРЧУК

Засвідчую, що у кваліфікаційній роботі немає запозичень праць інших авторів без відповідних посилань
Студент _____ Косінський Д.Л.

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій

Кафедра комп'ютерних систем та мереж

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Ігор ЖУКОВ
«_____»_____ 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

Косінського Дениса Леонідовича

(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема кваліфікаційної роботи: «Система візуалізації та аналізу даних з рейсових записів польотів»

затверджена наказом ректора від «29» серпня 2023 р. № 1521/ст

2. Термін виконання роботи: з 2 жовтня 2023 р. по 31 грудня 2023 р.

3. Вихідні дані до роботи: технічна документація, тестові дані, програмні продукти.

4. Зміст пояснювальної записки: вступ, аналіз предметної області, аналіз інструментарію, опис програмної реалізації, робота користувача з програмною системою, розробка веб-додатку для аналізу даних рейсових польотів літаків, тестування веб-додатку, висновки.

5. Перелік обов'язкового графічного (ілюстрованого) матеріалу:

– головний інтерфейс при роботі модуля;

– структура БД;

– діаграма варіантів використання;

– діаграма класів;

– схема алгоритму роботи веб-додатку.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1	Проаналізувати існуючі системи візуалізації та аналізу даних рейсових польотів та сформувати основні вимоги до програмного модуля	02.10.2023 – 10.10.2023	
2	Визначити структуру та розробити програмний модуль	11.10.2023 – 01.11.2023	
3	Розробка розділу 1: Аналіз предметної області	02.11.2023 – 13.11.2023	
4	Розробка розділу 2: Аналіз інструментарію	14.11.2023 – 20.11.2023	
5	Розробка розділу 3: Опис програмної реалізації	21.11.2023 – 01.12.2023	
6	Розробка розділу 4: Робота користувача з програмною системою	02.12.2023 – 12.12.2023	
7	Оформлення пояснювальної записки	13.12.2023 – 17.12.2023	
8	Розробка презентації для захисту роботи та підготовка до захисту	18.12.2023 – 31.12.2023	

7. Дата видачі завдання: « 02 » жовтня 2023 р.

Керівник дипломного дослідження _____ Ольга РУСАНОВА

Завдання прийняв до виконання _____ Денис КОСІНСЬКИЙ

РЕФЕРАТ

Пояснювальна записка до дипломного дослідження «Система візуалізації та аналізу даних з рейсових записів польотів»: 90 сторінок, 28 рисунків, 1 таблиця, 23 використаних джерел.

Веб-додаток, *.NET CORE, ENTITY FRAMEWORK, AUTOMAPPER, MSSQL, C#, API, DEPENDENCY INJECTION, Blazor, Azure, ServiceBus, LeafletJS.*

Мета дипломного дослідження – створення системи візуалізації та аналізу даних з рейсових записів польотів з метою підвищення ефективності та доступності інформації про авіаційний рух.

Об’єкт дослідження – рейсові записи польотів, аналіз та візуалізація.

Предмет дослідження – веб-додаток системи візуалізації та аналізу даних з рейсових записів польотів.

Практична значущість дослідження – практичне застосування веб-додатку для ефективності роботи авіаторів, авіа-любителів.

Технології, технічні та програмні засоби, задіяні в дослідженні– мова програмування *C#, C#.NetCore, Entity Framework 6, MsSql, Object-Oriented Programming, Pattern Repository, Dependency injection, Azure, ServiceBus, LeafletJS.*

Прогнозовані припущення про подальший розвиток об’єкта дослідження – розширення вибірки даних.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

UI — графічний інтерфейс користувача

DB — база даних

COM — об’єктна модель компонентів

DLL — бібліотека динамічної компоновки.

API — програмний інтерфейс додатку

LINQ — запити інтегровані в мову

MVP — шаблон проєктування

MVC — шаблон проєктування

XML — розширювана мова розмітки

Валідація — перевірка коректності

Фреймворк — зовнішній програмний продукт

ВСТУП

У сучасному світі, де технологічний прогрес нестабільно розвивається, комп'ютерна інженерія займає центральне місце у вирішенні складних завдань та розвитку нових технологій. Разом з тим, авіаційна промисловість не залишається осторонь цього процесу, ставлячи перед собою завдання покращення безпеки, ефективності та функціональності своїх систем.

У контексті цього еволюційного прогресу, дослідник ставить перед собою виклик розробити та впровадити систему візуалізації та аналізу даних з рейсових записів польотів. Актуальність цього завдання полягає в потребі підвищення рівня безпеки та ефективності авіаційних процесів шляхом інноваційного використання сучасних технологій.

Ця магістерська робота присвячена розробці системи візуалізації та аналізу даних з рейсових записів польотів. Метою даного дослідження є створення веб-додатку, який надасть можливість зчитувати та обробляти дані з рейсових записів польотів, а також візуалізувати ці дані на зручному веб-інтерфейсі. Застосування фреймворку *ASP.NET Core* для розробки серверної частини та використання *Blazor* для створення інтерактивних графіків та діаграм дозволить реалізувати мету роботи та отримати корисні результати.

Метою даної магістерської роботи є створення системи візуалізації та аналізу даних з рейсових записів польотів з метою підвищення ефективності та доступності інформації про авіаційний рух. Головні цілі цього дослідження включають:

– розробку функціональності системи, яка дозволить ефективно відстежувати рейси, отримувати інформацію про аеропорти та літаки, інтегрувати великий обсяг даних та представляти їх у зрозумілій та корисній формі для користувачів;

– створення інфографіки для кожного аеропорту та літака з метою поліпшення сприйняття та аналізу авіаційних даних;

– забезпечення безкоштовним доступом до системи для розширення кола користувачів та забезпечення їм ефективним інструментом для вивчення та аналізу

авіаційного руху;

– збільшення обсягу доступних даних та їх аналіз з метою поліпшення розуміння та використання інформації про рейси та авіаційний рух для підвищення ефективності вітчизняного та міжнародного повітряного транспорту.

Це дослідження спрямоване на створення ефективного інструмента для відстеження та аналізу авіаційних даних з метою покращення якості доступної інформації та її використання в різних сферах авіаційної діяльності. В результатах цієї магістерської роботи передбачається отримання практичних рекомендацій щодо розробки веб-додатків для візуалізації та аналізу даних у сфері авіаційних систем. Також буде проведено оцінку ефективності та функціональності розробленої системи на основі тестування та апробації на реальних даних.

Авіаційна галузь, на сьогоднішній день, переживає стрімкий розвиток, супроводжуваний значним обсягом даних, які стають ключовим елементом прийняття рішень та вдосконалення авіаційної безпеки. Запровадження системи візуалізації та аналізу рейсових записів польотів є важливим кроком у забезпеченні інноваційного підходу до управління та безпеки в авіації.

Аналіз сучасних наукових публікацій та технічних документів в області візуалізації даних та авіаційних систем дозволяє визначити тенденції та найновіші розробки. Результати літературного огляду обґрунтовують вибір конкретних технологій та підходів, які будуть використані у розробці системи.

У роботі визначено ключові завдання, перед якими стоїть дослідження. Серед них – забезпечення високої ефективності системи, інтеграція різноманітних даних, та створення інтуїтивно зрозумілого інтерфейсу для кінцевого користувача.

Магістерська робота складається з чотирьох основних розділів, кожен з яких відображає певний етап розробки системи візуалізації та аналізу даних з рейсових записів польотів. Кожен розділ охоплює конкретний аспект роботи, від методології та архітектури до використаних технологій і результатів дослідження.

У вступі визначено основні поняття та терміни, що використовуються в роботі, з метою уникнення непорозумінь та забезпечення єдності термінології в усій магістерській роботі.

Від цього дослідження очікуються практичні результати, що визначать можливості та переваги використання розробленої системи для візуалізації даних у сфері авіаційних систем. Очікується, що отримані результати внесуть вагомий внесок у розробку веб-додатків та покращення доступності інформації про авіаційний рух.

У вступі розглядаються обмеження, що можуть вплинути на результати дослідження, а також вказуються можливі напрямки для подальших досліджень у цій області.

Цей вступ визначає контекст та фундамент для подальших розділів, а також надає читачеві чітке уявлення про мету та значення магістерської роботи.

У контексті сучасного технологічного розвитку та постійного зростання обсягу даних, комп'ютерна інженерія визначає нові стандарти у вирішенні складних завдань та сприяє еволюції нових технологій. Авіаційна промисловість, в свою чергу, визнає важливість та актуальність інновацій у вдосконаленні безпеки, ефективності та функціональності своїх систем. У цьому контексті, ця магістерська робота присвячена розробці системи візуалізації та аналізу даних з рейсових записів польотів.

Основною метою даного дослідження є створення веб-додатку, який спрямований на зчитування та обробку даних з рейсових записів польотів, а також їх візуалізацію на зручному веб-інтерфейсі. Застосування фреймворку *ASP.NET Core* для розробки серверної частини та використання *Blazor* для створення інтерактивних графіків та діаграм дозволить досягти високої ефективності та отримати значущі результати.

Основні цілі дослідження включають розробку функціональності системи для ефективного відстеження рейсів, отримання інформації про аеропорти та літаки, інтеграцію великого обсягу даних та їх зручне представлення. Створення інфографіки для кожного аеропорту та літака спростить сприйняття та аналіз авіаційних даних, а безкоштовний доступ до системи розширить коло користувачів та забезпечить їм ефективний інструмент для вивчення та аналізу авіаційного руху.

Очікується, що розроблена система не тільки полегшить доступ до інформації про авіаційний рух, але й сприятиме підвищенню розуміння та використання цих даних для покращення ефективності авіаційного транспорту, враховуючи його зростаючу складність та обсяг.

Забезпечення безкоштовного доступу до системи сприятиме розширенню кола користувачів та наданню їм ефективного інструменту для вивчення та аналізу авіаційного руху. Збільшення обсягу доступних даних та їх аналіз покликані поліпшити розуміння та використання інформації про рейси та авіаційний рух для підвищення ефективності вітчизняного та міжнародного повітряного транспорту.

Орієнтована на практичні результати, магістерська робота передбачає отримання практичних рекомендацій щодо розробки веб-додатків для візуалізації та аналізу даних в авіаційних системах. Тестування та апробація на реальних даних сприятимуть оцінці ефективності та функціональності системи.

Ця магістерська робота визначає сучасний контекст поєднання комп'ютерної інженерії та авіаційної галузі, а також пропонує практичний інструмент для розробників веб-додатків та фахівців відповідних галузей. Усі ці аспекти визначають важливість та актуальність даного дослідження у сучасному технологічному ландшафті.

В цілому, ця магістерська робота відображає актуальність теми в контексті поєднання комп'ютерної інженерії та авіаційної сфери, а також має практичне значення для розробників веб-додатків та відповідних галузей.

Це дослідження має на меті створення ефективного інструмента для відстеження та аналізу авіаційних даних з метою покращення якості доступності інформації та її використання в різних сферах авіаційної діяльності. В результаті цього дослідження передбачається отримання практичних рекомендацій, які будуть сприяти розробці веб-додатків для візуалізації та аналізу даних у сфері авіаційних систем. Очікується, що отримані результати внесуть вагомий внесок у покращення якості доступної інформації та її використання в різних аспектах авіаційної сфери.

Важливою складовою вступу є визначення обмежень, які можуть вплинути на результати дослідження. З урахуванням технічних, економічних та соціальних

обмежень роботи, буде проведено вдосконалення системи в подальших етапах її розвитку.

В контексті дальших напрямків досліджень, визначено можливості розширення функціоналу системи, а також оптимізації для впровадження в інші галузі авіаційної індустрії та суміжних областей. Здійснення такого дослідження сприятиме розвитку та впровадженню нових технологій в авіаційних системах, а також покращенню безпеки та ефективності авіаційного руху.

Отже, дана магістерська робота присвячена розробці та вдосконаленню системи візуалізації та аналізу даних з рейсових записів польотів. Її результати та рекомендації мають великий потенціал у покращенні якості та ефективності управління авіаційним рухом, а також внесенні новаторських рішень у сучасну авіаційну галузь.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Вступ до предметної області

Авіаційна індустрія є однією з найбільш складних та важливих галузей, яка забезпечує швидкий та безпечний перевезення пасажирів та вантажів по всьому світу. В рамках авіаційної сфери рейсові записи польотів виконують важливу функцію – вони фіксують інформацію про кожен здійснений політ, включаючи дані про маршрут, тривалість польоту, паливну економію, параметри повітряного судна, погодні умови та інші параметри, що мають значення для безпеки та ефективності польотів.

Однак, обробка та аналіз великого обсягу даних з рейсових записів можуть бути складними завданнями, які вимагають спеціалізованих інструментів та методів. Візуалізація даних з рейсових записів може сприяти зрозумінню складних взаємозв'язків між різними показниками, виявленню аномалій та прийняттю обґрунтованих рішень в авіаційному секторі.

Основним завданням даного дослідження є розробка системи візуалізації та аналізу даних з рейсових записів польотів. Використання сучасних веб-технологій, таких як *ASP.NET Core* та *Blazor*, надає можливість створити веб-додаток, який дозволить зручно та ефективно працювати з великим обсягом даних та отримувати важливу інформацію для авіаційних спеціалістів.

У подальших розділах роботи будуть розглянуті основні вимоги до системи візуалізації та аналізу даних з рейсових записів, аналіз існуючого інструментарію та опис програмної реалізації системи. Крім того, буде досліджено взаємодію користувача з програмною системою та надано оцінку її ефективності та корисності у контексті авіаційного сектору.

Актуальність визначається шляхом аналізу сучасних потреб і викликів, що існують у предметній області, а також прогресу технологій та розвитку авіаційної

галузі.

Авіаційна галузь є однією зі стратегічно важливих сфер транспорту і має значний вплив на економіку та суспільство в цілому. З кожним роком зростає кількість авіаперельотів, а разом з нею збільшується обсяг даних, які збираються під час польотів. Аналіз та візуалізація цих даних може допомогти виявляти тенденції, забезпечувати безпеку польотів та оптимізувати роботу авіакомпаній.

Однак, на сьогоднішній день багато систем, які використовуються для аналізу та візуалізації даних, не спеціалізовані на рейсових записах польотів і не забезпечують необхідного функціоналу для авіаційного сектору. Тому є актуальна потреба у розробці спеціалізованої програмної системи, яка враховуватиме особливості цих даних та вміятиме їх ефективно аналізувати та візуалізувати.

Крім того, швидкий технологічний прогрес у галузі веб-розробки, зокрема використання *ASP.NET Core* та *Blazor*, надає нові можливості для створення інтерактивних та потужних веб-додатків. Це дозволяє розробнику ефективно використовувати сучасні інструменти для створення програмного продукту, який буде легко доступний для користувачів з різних пристроїв та забезпечуватиме зручну взаємодію з даними.

Отже, розробка програмного продукту для візуалізації та аналізу даних з рейсових записів польотів є актуальною завданням, що відповідає потребам авіаційного сектору та використовує сучасні технології веб-розробки. Вона сприятиме поліпшенню процесів аналізу та прийняття рішень у цій галузі та може мати значний позитивний вплив на авіаційну безпеку, ефективність та пасажирський комфорт.

1.2. Огляд та аналіз існуючих рішень

У цьому підрозділі проводиться огляд існуючих рішень та інструментарію, які використовуються для візуалізації та аналізу даних з рейсових записів польотів. Це дозволяє отримати уявлення про наявні можливості, переваги та недоліки існуючих систем і з'ясувати, які прогалини можуть бути заповнені у процесі розробки

програмної системи.

На сьогоднішній день існує кілька популярних інструментів та платформ для аналізу та візуалізації даних, таких як *Tableau, Power BI, QlikView* та інші. Ці системи надають широкий спектр можливостей для створення інтерактивних графіків, діаграм, звітів та інших візуалізаційних компонентів. Вони дозволяють користувачам швидко та зручно аналізувати дані, виявляти тенденції та залежності, робити прогнози та приймати обґрунтовані рішення.

Проте, враховуючи специфіку авіаційної галузі та особливості рейсових записів польотів, можливе виникнення проблем з використанням загальноприйнятих рішень. Наприклад, не всі системи мають достатню гнучкість для роботи зі структурою даних рейсових записів або не забезпечують необхідні можливості для аналізу показників, специфічних для авіаційного сектору.

Тому в рамках даної магістерської роботи вивчаються існуючі рішення та їхні можливості з точки зору аналізу та візуалізації даних з рейсових записів польотів. Проводиться порівняльний аналіз їхніх переваг та недоліків з урахуванням вимог і особливостей авіаційного сектору. Такий огляд і аналіз існуючих рішень допоможе визначити недоліки та прогалини, які можуть бути вирішені у процесі розробки програмної системи для візуалізації та аналізу даних з рейсових записів.

1.2.1. *FlightAware*

FlightAware – це компанія та онлайн-платформа, яка надає послуги в області відстеження авіаційних рейсів та надання інформації про авіаперельоти. Вони пропонують широкий спектр послуг і інструментів, які стосуються авіації, включаючи такі функції:

Відстеження рейсів: *FlightAware* дозволяє користувачам в режимі реального часу слідкувати за рухом пасажирських і вантажних літаків. Ви можете ввести номер рейсу або іншу інформацію про рейс, і отримувати актуальну інформацію про його місцезнаходження, статус, швидкість і висоту [22].

Інформація про аеропорти: *FlightAware* також надає важливу інформацію про аеропорти, включаючи погодні умови, статуси рейсів, інформацію про вимушені затримки і вибіркову статистику щодо аеропортів.

План політів: пасажири та інші користувачі можуть використовувати FlightAware для планування подорожей, знаходження доступних рейсів і отримання інформації про ціни на квитки.

Служби для авіакомпаній: *FlightAware* пропонує рішення для авіакомпаній та операторів літаків, включаючи інструменти для відстеження флоту, оптимізації маршрутів, контролю палива та інші.

Інтеграція з аеропортами та повітряними службами: компанія також співпрацює з аеропортами і повітряними службами для покращення управління рухом літаків і безпекою в повітрі.

FlightAware створена для різних категорій користувачів, включаючи авіаційних ентузіастів, пасажирів, авіакомпанії, аеропорти і громадських служб повітряного руху. Платформа надає доступ до величезної кількості даних про авіаперельоти та пов'язані послуги для полегшення навігації в світі авіації.

1.2.2. *OpenSky Network*

OpenSky Network (ОпенСкай Нетворк) – це міжнародна ініціатива, спрямована на створення та підтримку безкоштовної та відкритої системи для збору та розповсюдження даних про повітряний рух. Головною метою *OpenSky Network* є забезпечення доступу до важливої інформації про рух літаків у реальному часі для великої кількості користувачів, таких як дослідники, громадські служби, авіакомпанії, розробники програмного забезпечення та інші.

Основні особливості *OpenSky Network* описані нижче.

– Відкриті дані: *OpenSky Network* надає відкритий доступ до даних про рух літаків у реальному часі. Це означає, що будь-який користувач може отримати доступ до цих даних без обмежень та безкоштовно.

– Світовий охоплення: *OpenSky Network* володіє мережею земних станцій та супутникових засобів, розташованих по всьому світу. Це дозволяє системі збирати дані з різних регіонів та забезпечує глобальний охоплення.

– Дані про рух літаків: *OpenSky Network* збирає дані про положення та параметри літаків, такі як швидкість, висота, маршрут, ідентифікаційні номери і багато іншого. Ці дані можуть бути використані для великої кількості цілей,

включаючи дослідження, навігацію, відслідковування польотів і безпеку повітряного руху.

– Доступ через *API*: *OpenSky Network* надає *API* (інтерфейс програмування додатків), який дозволяє розробникам створювати програми та сервіси, що використовують дані про рух літаків. Це дозволяє створювати різноманітні застосунки та інструменти.

OpenSky Network займається активним залученням громадськості, науковців та інженерів для спільної розробки і використання цих даних з метою покращення авіаційної безпеки, досліджень та інновацій у сфері повітряного руху.

1.2.3. *Airport-Data*

Airport-Data – це онлайн-ресурс та база даних, яка містить інформацію про аеропорти з усього світу. Ця платформа надає доступ до різноманітних даних, пов'язаних з аеропортами, що описані нижче.

– Опис аеропортів: *Airport-Data* надає короткий опис кожного аеропорту, включаючи його назву, місцезнаходження (координати), класифікацію (наприклад, міжнародний або внутрішній), висоту над рівнем моря, види послуг і обладнання, наявність злітно-посадкових смуг та іншу загальну інформацію.

– Інформація про злітно-посадкові смуги: для кожного аеропорту *Airport-Data* надає детальну інформацію про всі наявні злітно-посадкові смуги, включаючи їхні розміри, типи покриття і їхні орієнтації.

– Фотографії: платформа містить фотографії аеропортів, які дозволяють користувачам побачити їх зовнішній вигляд та інфраструктуру.

– Інформація про навігацію: *Airport-Data* також може включати дані про навігаційні засоби та навігаційну інфраструктуру, доступну на аеропорту для літаків.

– Авіакомпанії та маршрути: деякі версії *Airport-Data* можуть містити інформацію про авіакомпанії, які обслуговують конкретний аеропорт, та маршрути, які вони обслуговують.

Ця інформація корисна для пасажирів, авіаційних ентузіастів, пілотів, дослідників і всіх, хто має інтерес до аеропортів та авіації загалом. *Airport-Data*

може бути використана для планування подорожей, досліджень, навігації та вивчення аеропортів у всьому світі.

1.2.4. *Flightradar24*

Flightradar24 – це популярна онлайн-платформа та мобільний додаток, які надають можливість відстежувати рух цивільних літаків у реальному часі. Ця платформа надає користувачам доступ до широкого спектру інформації про повітряний рух, включаючи такі основні функції, що описані нижче.

– Відстеження рейсів: *Flightradar24* дозволяє користувачам в реальному часі спостерігати за рухом пасажирських, вантажних і приватних літаків по всьому світу. Ви можете ввести номер рейсу, індекс літака або іншу інформацію про рейс, щоб відслідковувати його маршрут, висоту, швидкість та інші параметри.

– Інформація про літаки: платформа надає докладну інформацію про кожний літак, включаючи модель літака, виробника, рік випуску, реєстраційний номер, власника, поточну швидкість, висоту і іншу важливу статистику.

– Інформація про аеропорти: *Flightradar24* також містить дані про аеропорти, включаючи їхні координати, коди *IATA*/*IKAO*, висоту над рівнем моря, розмір злітно-посадкових смуг, статус рейсів та погодні умови.

– Інтерактивна карта: користувачі можуть використовувати інтерактивну карту *Flightradar24* для відстеження руху літаків у своєму регіоні або в будь-якому місці світу. Карта показує літаки, їхні траєкторії і інші важливі дані.

– Метеорологічні дані: платформа може надавати інформацію про погоду в реальному часі для аеропортів і літаків.

Flightradar24 користується популярністю серед авіаційних ентузіастів, пасажирів, пілотів, журналістів і дослідників. Вона надає можливість відстежувати польоти родичів та друзів, контролювати стан повітряного руху під час подорожей, досліджувати авіаційні події і багато іншого. Базуючись на передових технологіях відстеження, *Flightradar24* забезпечує доступ до важливої інформації про авіаційний рух у реальному часі.

1.2.5. *PlaneFinder*

PlaneFinder – це популярний онлайн-ресурс та мобільний додаток для

відстеження руху цивільних літаків у реальному часі. Ця платформа подібна до *Flightradar24* і надає користувачам можливість слідкувати за польотами, дізнаватися інформацію про літаки і отримувати різноманітні дані про авіаційний рух. Основні функції *PlaneFinder* включають в себе наступне:

– відстеження рейсів: *PlaneFinder* дозволяє користувачам в реальному часі відстежувати рух літаків, введучи номер рейсу або іншу інформацію про рейс. Ви можете переглядати траєкторію польоту, висоту, швидкість та інші параметри;

– інформація про літаки: платформа надає докладну інформацію про кожен літак, включаючи модель літака, реєстраційний номер, виробника, рік випуску, авіакомпанію-експлуатанта, поточну швидкість і висоту;

– інтерактивна карта: *PlaneFinder* має інтерактивну карту, яка відображає рух літаків у реальному часі. Користувачі можуть навігувати картою, масштабувати та вибирати конкретні літаки для отримання додаткової інформації;

– метеорологічні дані: платформа може надавати інформацію про погоду в реальному часі для аеропортів і літаків, що може бути корисним для пасажирів та пілотів;

– події та сповіщення: *PlaneFinder* може надсилати користувачам сповіщення про події, такі як вильоти, прибуття і плановані рейси.

PlaneFinder служить як корисний інструмент для пасажирів, авіаційних ентузіастів, пілотів та інших осіб, які мають інтерес до авіаційного руху. Платформа дозволяє легко відстежувати рух літаків, отримувати актуальну інформацію про авіаційний рух і навіть дізнаватися деталі про конкретні літаки.

1.2.6. *AirNav Network Radarbox*

AirNav RadarBox є популярними системами для відстеження руху цивільних літаків у реальному часі та збирання інформації про них. Ця система включає в себе різні продукти і послуги для відстеження руху літаків та надає користувачам доступ до важливих даних про авіаційний рух. Нижче подано докладну інформацію про *AirNav RadarBox*:

– відстеження рейсів в реальному часі: *AirNav RadarBox* дозволяє користувачам слідкувати за рухом цивільних літаків у реальному часі. Ви можете

ввести номер рейсу, індекс літака або іншу інформацію про рейс, щоб відстежувати його маршрут, висоту, швидкість і інші параметри;

– повний набір інформації про літаки: система надає докладну інформацію про кожний літак, включаючи модель літака, реєстраційний номер, виробника, рік випуску, авіакомпанію-експлуатанта, маршрут і багато іншої статистики;

– історія польоту: *AirNav RadarBox* дозволяє переглядати історію польотів для конкретного літака, включаючи попередні рейси та відомості про попередні польоти;

– аварійні ситуації та попередження: система може надсилати сповіщення про аварійні ситуації або події, такі як екстрені зниження висоти літака, виліт за межі маршруту і інші надзвичайні обставини;

– метеорологічна інформація: *AirNav RadarBox* може надавати користувачам інформацію про погоду в реальному часі на аеропортах та для конкретних літаків;

– мережа користувачів: в системі існує можливість взаємодії з мережею користувачів, де ви можете обмінюватися даними та інформацією про авіаційний рух.

AirNav RadarBox надає інформацію про цивільний авіаційний рух і може бути корисною для пасажирів, авіаційних ентузіастів, пілотів і дослідників. Вона допомагає відслідковувати рух літаків, дізнаватися деталі про конкретні літаки та моніторити безпеку в повітрі.

1.2.7. Висновок до існуючих систем

FlightAware, OpenSky Network, Airport-Data, Flightradar24, PlaneFinder, AirNav Network Radarbox сервіси і системи надають інформацію про авіаційний рух та аеропорти, проте вони можуть відрізнятися за функціональністю, обсягом доступної інформації та спрямованістю.

FlightAware: *FlightAware* – це компанія, яка спеціалізується на відстеженні рейсів та наданні інформації про авіаперельоти. Вона надає широкий спектр послуг для авіаційних професіоналів та пасажирів.

OpenSky Network: *OpenSky Network* – це міжнародна ініціатива, яка надає безкоштовний та відкритий доступ до даних про повітряний рух та пов'язаної

інформації для спільного використання та досліджень.

Airport-Data: *Airport-Data* – це онлайн-ресурс і база даних, яка містить інформацію про аеропорти з усього світу, включаючи дані про злітно-посадкові смуги та інфраструктуру.

Flightradar24: *Flightradar24* – це популярна платформа для відстеження руху цивільних літаків у реальному часі, надаючи докладну інформацію про рейси та літаки.

PlaneFinder: *PlaneFinder* – це інший популярний інструмент для відстеження руху літаків, який дозволяє користувачам слідкувати за літаками, отримувати інформацію про них та досліджувати авіаційний рух.

AirNav Network RadarBox: *AirNav RadarBox* – це система для відстеження руху літаків, яка надає доступ до інформації про рух літаків у реальному часі, повний набір інформації про літаки і інші функції для користувачів.

Детальна порівняльна інформація описана нижче (табл. 1.1).

Таблиця 1.1

Порівняльна таблиця

Сервіс	**Спрямованість**	**Основні Функції**	**Доступність даних**
FlightAware	Авіаційні професіонали, пасажири	Відстеження рейсів, інформація про аеропорти	Широкий спектр даних
OpenSky Network	Дослідники, громадські служби	Збір даних про повітряний рух, дослідження	Світовий охоплення
Airport-Data	Авіаційні ентузіасти, дослідники	Інформація про аеропорти, злітно-посадкові смуги	Широкий спектр аеропортів
1	**2**	**3**	**4**

Закінчення таблиці 1.1

1	**2**	**3**	**4**
Flightradar24	Авіаційні ентузіасти	Відстеження рейсів	Глобальний охоплення
Flightradar24	Авіаційні ентузіасти, пасажири	Відстеження рейсів, інформація про літаки	Глобальний охоплення
PlaneFinder	Авіаційні ентузіасти, пілоти	Відстеження руху літаків, інформація про літаки	Інтерактивна мапа
AirNav Network RadarBox	Авіаційні ентузіасти, пасажири	Відстеження руху літаків, повна інформація про літаки	Деталізовані дані про рух
Розробляєма система	Пасажири, Авіаційні діячі, Статисти, Пілоти	Відстеження рейсів, інформація про аеропорти, Інформація про літаки. Інфографіка по кожному аєропорту/літаку. Безоплатність користуваня. Великий обсяг даних.	Глобальне охоплення, широка вибірка даних.

Таким чином можна зробити висновок: розроблювана система візуалізації та аналізу даних з рейсових записів польотів пропонує унікальний набір функцій і переваг для користувачів. Вона забезпечує відстеження рейсів, надає інформацію

про аеропорти та літаки, а також надає можливість отримувати інфографіку для кожного аеропорту і літаку. Важливою особливістю цієї системи є її безоплатність для користувачів, що робить її доступною для широкого кола зацікавлених осіб. Крім того, система володіє великим обсягом даних, що забезпечує користувачів інформацією, необхідною для вивчення авіаційного руху та пов'язаних аспектів. Завдяки цим перевагам, розроблювана система стає цінним інструментом для досліджень, моніторингу та аналізу авіаційного сектору, відкриваючи нові можливості для користувачів у галузі авіаційної інформації та даних.

1.3. Визначення основних вимог та функціональності

У цьому пункті розглядається процес визначення основних вимог та функціональності системи візуалізації та аналізу даних з рейсових записів польотів. Цей етап є критичним, оскільки від правильного визначення вимог залежить успішне виконання проекту та задоволення потреб користувачів [23].

Перш за все, визначаються основні функції, які повинна виконувати програмна система. Серед них можуть бути наступні: зчитування та імпорт даних з рейсових записів польотів у систему.

– Обробка та аналіз даних з метою виявлення тенденцій, паттернів та аномалій.

– Візуалізація даних у вигляді графіків, діаграм, карт і інших візуальних елементів.

– Надання інтерактивного інтерфейсу користувачу для взаємодії з даними та виконання різних запитів.

– Забезпечення можливості фільтрації, сортування та пошуку даних з рейсових записів.

– Забезпечення зручного відображення деталей польотів та пов'язаних параметрів.

Крім того, розглядаються основні вимоги до системи, які включають:

– ефективність та швидкодію обробки даних навіть при великому обсязі інформації;

– гнучкість та можливість розширення системи для врахування майбутніх потреб;

– надійність та безпеку зберігання та обробки конфіденційних даних;

– зручний та інтуїтивно зрозумілий інтерфейс для користувачів з різним рівнем технічної підготовки;

– сумісність з різними платформами та пристроями для забезпечення доступності з будь-якого пристрою з Інтернет-підключенням.

У результаті аналізу вимог і функціональності будуть визначені ключові характеристики та можливості системи візуалізації та аналізу даних з рейсових записів польотів, що дозволить забезпечити задоволення потреб користувачів та досягнення мети даної магістерської роботи.

Висновки за розділом

У цьому розділі проведено аналіз сучасних можливостей та вимог до систем візуалізації та аналізу даних з рейсових записів польотів. Цей аналіз дозволив визначити, що існують різні підходи та інструменти для відстеження авіаційного руху та надання інформації про рейси та літаки.

Встановлено, що існуючі системи, такі як *FlightAware, OpenSky Network, Airport-Data, Flightradar24*, *PlaneFinder, AirNav Network Radarbox*, відповідають на різні потреби та мають свої переваги та обмеження. *FlightAware* спрямований на різні категорії користувачів, *OpenSky Network* – на дослідників та спільноту, *Airport-Data* – на аеропорти, а *Flightradar24* та *PlaneFinder* – на відстеження літаків у реальному часі. *AirNav Network Radarbox* надає інформацію про рух літаків у реальному часі та інші функції для користувачів, а розробляєма система – відстеження рейсів, надання інформації про аеропорти, літаки. Інфографіка по кожному аеропорту/літаку. Безоплатність користуваня. Великий обсяг даних.

РОЗДІЛ 2
АНАЛІЗ ІНСТРУМЕНТАРІЮ

2.1. Платформа .*Net*

Microsoft .NET [3], представлена корпорацією *Microsoft*, є платформою, призначеною для полегшення розробки різноманітних додатків для персональних комп'ютерів, веб-додатків і порталів. Важливою концепцією *.NET* є його здатність забезпечувати сумісність між службами, написаними різними мовами програмування. Ця перевага активно впроваджується *Microsoft* як частина *.NET* framework.

2.1.1. Загальні відомості про платформу .*NET Framework*

Платформа розробки *.NET* охоплює широкий спектр інструментів, мов програмування та бібліотек, які дозволяють створювати різноманітні програми.

.NET було реалізовано різними способами, що дозволяє виконувати код .NET на багатьох платформах, включаючи Linux, macOS, Windows, iOS, Android та багато інших.

Оригінальною реалізацією .NET є .NET Framework, яка дозволяє виконувати різноманітні програми, такі як веб-сайти, служби та настільні програми, виключно в Windows.

Реалізація .NET розроблена таким чином, щоб бути сумісною з кількома платформами, що дозволяє запускати веб-сайти, служби та консольні програми в операційних системах Windows, Linux і macOS. Варто зазначити, що .NET тепер є проектом з відкритим кодом, розміщеним на GitHub. Важливо зазначити, що .NET раніше називався .NET Core.

Xamarin/Mono — це комплексна реалізація .NET, розроблена для безперебійного виконання програм у всіх основних мобільних операційних системах, включаючи IOS та Android.

Стандарт .NET служить авторитетним керівництвом для API, які є універсальними для різних реалізацій .NET. Це гарантує безпроблемне

використання коду та бібліотек у різних реалізаціях.

Діаграма на рисунку 2.1 ілюструє зв’язок між середовищем CLR, бібліотекою класів і системою в цілому, а також те, як керований код працює в ширшій архітектурній структурі, як зображено на тому ж малюнку (рис. 2.1).

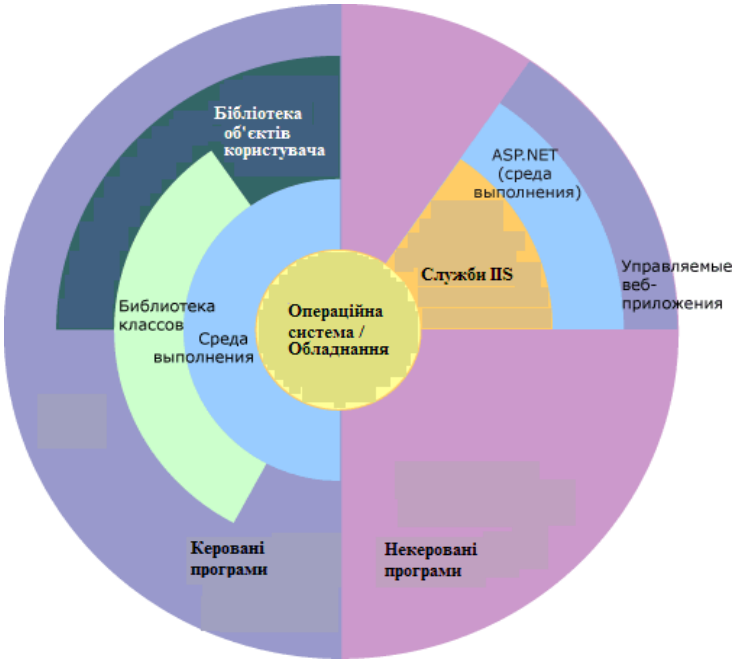

Рис. 2.1. Модель виконання *.Net Framework*

2.1.2. Середовище *CLR*

Середовище CLR [3] контролює пам'ять, виконання потоків, виконання коду, перевірку безпеки коду, компіляцію проекту та різні системні процеси (рис. 2.2). Ці процеси є внутрішніми для керованого коду, що виконується CLR.

Платформа .NET надає середовище виконання, яке називається середовищем виконання спільної мови, що виконує код і надає послуги для полегшення розробки додатків.

Мовні компілятори та інструменти розкривають можливості середовища виконання спільної мови і дозволяють писати код, який використовує це кероване середовище виконання. Код, який ви розробляєте за допомогою компілятора мови, орієнтованого на середовище виконання, називається керованим кодом. Керований код має такі переваги, як міжмовна інтеграція, обробка міжмовних винятків, підвищена безпека, підтримка версій та розгортання, спрощена модель взаємодії компонентів, а також сервіси налагодження та профілювання.

Для того, щоб середовище виконання могло надавати послуги керованому коду, компілятори мов повинні випускати метадані, які описують типи, члени та посилання у вашому коді. Метадані зберігаються разом з кодом; кожен портативний виконуваний файл (PE) містить метадані. Середовище виконання використовує метадані для пошуку та завантаження класів, розміщення екземплярів у пам'яті, обробки викликів методів, генерування власного коду, забезпечення безпеки та встановлення меж контексту виконання.

Середовище виконання автоматично обробляє компонування об'єктів і керує посиланнями на об'єкти, звільняючи їх, коли вони більше не використовуються. Об'єкти, часом життя яких керують у такий спосіб, називаються керованими даними. Збірка сміття усуває витоки пам'яті, а також деякі інші поширені помилки програмування. Якщо ваш код є керованим, ви можете використовувати керовані дані, некеровані дані або як керовані, так і некеровані дані у вашій .NET програмі. Оскільки компілятори мов надають власні типи, такі як примітивні типи, ви не завжди знаєте (або потребуєте знати), чи є ваші дані керованими.

Спільне середовище виконання полегшує створення компонентів і додатків, об'єкти яких взаємодіють між мовами. Об'єкти, написані різними мовами, можуть взаємодіяти один з одним, а їхня поведінка може бути тісно інтегрована. Наприклад, ви можете визначити клас, а потім використати іншу мову, щоб отримати клас з вашого оригінального класу або викликати метод в оригінальному класі. Ви також можете передати екземпляр класу методу класу, написаному іншою мовою. Така міжмовна інтеграція можлива завдяки тому, що компілятори мов та інструменти, орієнтовані на середовище виконання, використовують спільну систему типів, визначену середовищем виконання, і дотримуються правил середовища виконання для визначення нових типів, а також для створення, використання, збереження та зв'язування типів.

Середовище виконання надає наступні переваги:

– покращена продуктивність;

– можливість легко використовувати компоненти, розроблені іншими мовами;

– розширювані типи, що надаються бібліотекою класів;

– можливості мови, такі як успадкування, інтерфейси та перевантаження для об'єктно-орієнтованого програмування;

– явна підтримка паралельного програмування, що дозволяє писати багатопотокові та масштабовані додатки;

– підтримка структурованої обробки винятків ;

– підтримка користувацьких атрибутів;

– збір сміття;

– використання делегатів замість вказівників на функції для безпечних за типом зворотних викликів;

– підтримка асинхронного програмування: Сучасні мови та середовища виконання можуть надавати підтримку асинхронного програмування, що дозволяє ефективно взаємодіяти з асинхронними операціями та подіями;

– інструменти для профілювання та аналізу вашого коду: Рантайми можуть постачатися з інструментами для профілювання коду, аналізу його продуктивності та виявлення можливих помилок або проблем зі споживанням ресурсів.

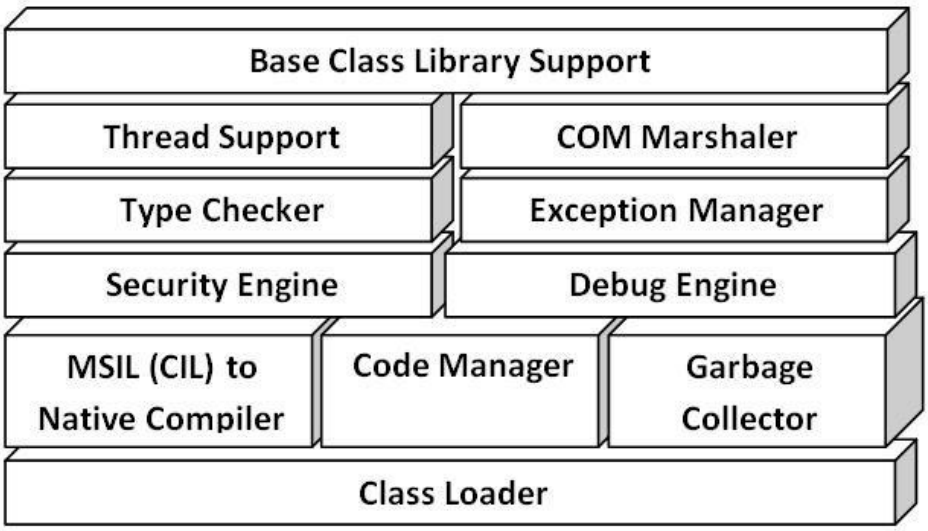

Рис. 2.2. Архітектура *CLR*

2.2. Платформа *Blazor*

Blazor є сучасною технологією розробки веб-застосунків [4], яка була представлена *Microsoft*. Ця платформа дозволяє розробникам створювати

інтерактивні клієнтські та серверні веб-додатки, використовуючи *C#* замість *JavaScript*. *Blazor* входить до складу *ASP.NET Core* і є частиною *.NET* платформи.

– Основні концепції застосування *Blazor*: ізоляція логіки та представлення**.** Шаблони *Blazor* використовують *Razor* синтаксис, дозволяючи легко інтегрувати *HTML* з *C#*, що сприяє ефективній роботі із компонентами та реюзабельністю коду.

– Компонентний підхід. *Blazor* спирається на компонентний підхід при створенні веб-інтерфейсів. Кожен компонент може мати свій власний логіку і стан, і він може бути легко перевикористаний у різних частинах застосунку.

– Використання новітніх можливостей *.NET*. *Blazor* надає можливість використовувати всі можливості *.NET*, такі як збір безпеки типів, управління пам'яттю, і велику кількість бібліотек і *API*, доступних для *.NET* розробників.

Варіативність *Blazor*:

Blazor пропонує дві основні схеми розробки: *Blazor Server* і *Blazor WebAssembly*.

– *Blazor Server*. При *Blazor Server* логіка додатку виконується на сервері, а з клієнтом відбувається взаємодія через *SignalR*. Це забезпечує великі можливості для масштабування і використання потужностей серверів, але може збільшити кількість затримок при відповідях інтерфейсу через необхідність звернення до сервера.

– *Blazor WebAssembly. Blazor WebAssembly* дозволяє виконувати *C#* код прямо в браузері, використовуючи технологію *WebAssembly*. Це дозволяє створювати веб-додатки, які можуть працювати офлайн і мають більш низькі затримки у реакції на користувацькі дії, оскільки логіка виконується локально, без необхідності звернення до сервера.

Blazor ефективний для створення комплексних односторінкових програм (*SPA*), які потребують швидкої відповіді від користувача, і комплексної взаємодії на клієнтській стороні без перезавантаження сторінки. Також *Blazor* добре підходить для проектів, де необхідно використовувати існуючі *C#* навички чи кодову базу без необхідності перенесення логіки на *JavaScript*.

Blazor є потужною і гнучкою платформою для створення масштабованих веб-застосунків із зосередженням на використанні *C#* на клієнтській та серверній

стороні. Вона надає значні переваги розробникам, дозволяючи пришвидшити процес розробки шляхом повторного використання коду та бібліотек, а також зменшити залежність від зовнішніх скриптових мов.

2.3. Хмарне рішення *MicrosoftAzure*

Microsoft Azure – хмарна платформа компанії *Microsoft* [5]. Надає можливість розробки, виконання додатків і зберігання даних на серверах, розташованих у розподілених дата-центрах.

Хмару *Azure* було анонсовано в жовтні 2008 року під кодовою назвою «*Project Red Dog*». Реліз відбувся 1 лютого 2010 року під назвою «*Windows Azure*». У 2014 році платформа була перейменована на *Microsoft Azure.*

Microsoft Azure реалізує хмарні моделі платформи як сервісу (*PaaS*) та інфраструктури як сервісу (*IaaS*). Можливе використання як сторонніх, так і сервісів *Microsoft* як моделі ПЗ як сервісу (*SaaS*). Працездатність платформи *Microsoft Azure* забезпечує глобальна мережа розподілених дата-центрів *Microsoft.*

Крім базових функцій операційних систем, *Microsoft Azure* має і додаткові: виділення ресурсів на вимогу для масштабування, автоматичну синхронну реплікацію даних для підвищення відмовостійкості, обробку відмов інфраструктури для забезпечення постійної доступності та інше.

Модель надання інфраструктури (*IaaS*) реалізує можливість оренди таких ресурсів, як сервери, пристрої зберігання даних і мережеве обладнання. Керування всією інфраструктурою здійснює постачальник, споживач керує тільки операційною системою та встановленими додатками.

Для віртуальних машин доступні образи таких операційних систем: *Windows Server, CoreOS, Ubuntu Server, Red Hat, Clear Linux OS, Debian, SUSE Linux Enterprise Server, Oracle Linux.*

Практично всі сервіси *Microsoft Azure* мають інтерфейс взаємодії *API*, побудований на основі обмежень для розподілених систем *REST*, що дає змогу розробникам використовувати хмарні сервіси з будь-якої операційної системи,

пристрою та платформи.

Крім того, користувачі можуть створювати та керувати власними сервісами, користуючись візуальним веб-інтерфейсом порталу *Azure*. Портал дає змогу налаштовувати сервіси, редагувати права доступу, відстежувати стан ресурсів і керувати білінгом.

Підтримувані мови та платформи розробки:

– *.NET;*
– *Java;*
– *Node.js (JavaScript);*
– *Python;*
– *PHP;*
– *Go.*

В інших розділах буде розписано про сервіси *Azure*, які будуть використанні в цьому дипломному дослідженні.

2.3.1. *Azure SQL Server*

Azure SQL – масштабована служба реляційної бази даних, розроблена компанією *Microsoft* на основі ядра *MS SQL* [6]. Застосовується як для управлінням малих персональних баз, так і для великих комерційних додатків.

Перевагами *Azure SQL* є:

– оптимізована для великих масивів даних та роботи на хмарній;
– платформі, що надає високу швидкість витягу даних;
– висока інтеграція з мовою програмування *C#* та бібліотекою;
– *EFCore*, що спрощує розробку;
– функції управління СУБД автоматизовані;
– високий рівень цілісності та захисту даних;
– використання розширеної мови *Transact SQL*;
– реплікації даних.

Основним недоліком *Azure SQL* є відсутність *SQL* агенту, за рахунок чого виконання задач за заданим часом є складно реалізованим.

2.3.2. *Azure ServiceBus*

Azure Service Bus – це послуга, яка надає можливості для реалізації асинхронного обміну повідомленнями між різними складовими розподіленої системи [7]. Це частина облакового платформного рішення *Microsoft Azure*. Основна мета *Azure Service Bus* – це забезпечення надійного і масштабованого обміну повідомленнями в розподіленому середовищі, такому як хмарні застосунки, мікросервіси та інші розподілені системи.

Основні функції та можливості *Azure Service Bus* включають:

– асинхронний обмін повідомленнями: *Azure Service Bus* дозволяє асинхронно обмінюватися повідомленнями між різними компонентами системи, що дозволяє їм взаємодіяти, не чекаючи одне на одне;

– надійність та масштабованість: *Azure Service Bus* надає механізми для забезпечення надійності обміну повідомленнями, включаючи можливість зберігання повідомлень у випадку тимчасових збоїв та їхнє повторне відправлення. Сервіс також масштабується відповідно до потреб вашого додатку;

– підтримка різних протоколів: *Azure Service Bus* підтримує різні протоколи обміну повідомленнями, такі як *AMQP*, *MQTT*, та *HTTP*, що робить його гнучким у використанні для різних застосунків;

– теми та підписки: *Azure Service Bus* дозволяє використовувати теми та підписки для групування та розсилання повідомлень. Це особливо корисно в сценаріях, де одне повідомлення має бути оброблено різними компонентами;

– фільтрація та вирази вибору повідомлень: спеціальні функції, такі як фільтрація повідомлень та вирази вибору (*SQL*-подібні вирази), дозволяють вибирати конкретні повідомлення для обробки.

Azure Service Bus використовується для реалізації розподіленої інтеграції, мікросервісної архітектури, а також для побудови розподілених застосунків, які вимагають асинхронного обміну повідомленнями між різними компонентами. Його можна використовувати для забезпечення взаємодії між додатками, розгорнутими в різних місцях, і для забезпечення надійного обміну даними в умовах високих вимог до доступності та масштабованості.

Поглиблюючи знання про *Azure Service Bus*, важливо відзначити додаткові аспекти, які підкреслюють його значущість та використання в розподіленому середовищі:

– глобальна розгортка та доступність: *Azure Service Bus* надає можливість глобального розгортання, що дозволяє вам обмінюватися повідомленнями між різними регіонами та областями. Це робить сервіс ідеальним для побудови розподілених систем з глобальним охопленням, забезпечуючи високий рівень доступності;

– підтримка гарантованої доставки: *Azure Service Bus* дозволяє налаштовувати гарантії доставки повідомлень, включаючи можливість доставки повідомлень лише один раз (*at-most-once delivery*) або гарантовано один раз (*exactly-once delivery*). Це важливо для сценаріїв, де важлива точність та цілісність обміну даними;

– інтеграція з *Azure Logic Apps* та іншими службами: *Azure Service Bus* ідеально поєднується з іншими сервісами *Azure*, такими як *Azure Logic Apps*, що дозволяє створювати складні бізнес-логіку та автоматизовані процеси на основі обміну повідомленнями. Це створює можливості для гнучкості та розширення функціональності;

– захист даних та конфіденційність: *Azure Service Bus* надає високий рівень захисту даних, використовуючи різноманітні механізми шифрування та аутентифікації. Це важливо для забезпечення конфіденційності та цілісності обміну конфіденційною інформацією між системними компонентами;

– підтримка складних сценаріїв маршрутизації: *Azure Service Bus* дозволяє налаштовувати складні сценарії маршрутизації повідомлень на основі вмісту повідомлення або його властивостей. Це полегшує гнучке управління обробкою повідомлень в залежності від конкретних умов та потреб вашої системи.

Azure Service Bus стає ключовим елементом інфраструктури для проєктів, які вимагають надійного та масштабованого асинхронного обміну повідомленнями в розподіленому середовищі *Azure*.

2.3.3. *Azure WebApp*

Azure Web App – це послуга обчислення в хмарному середовищі *Microsoft*

Azure [8], яка дозволяє легко розгортати та керувати веб-застосунками без необхідності управління інфраструктурою. *Azure Web App* надає середовище виконання для ваших веб-застосунків і дозволяє швидко розгортати, масштабувати та управляти вашими веб-додатками з хмарного середовища *Azure*.

Основні характеристики та можливості *Azure Web App* включають:

– простота розгортання: за допомогою *Azure Web App* ви можете легко розгортати свій веб-додаток, навіть якщо у вас немає глибоких знань про інфраструктуру хмарних обчислень;

– підтримка різних мов та платформ: *Azure Web App* підтримує різні мови програмування, такі як *.NET, Java, Node.js, Python, PHP* та інші. Ви можете вибрати мову, яка найкраще підходить для вашого проекту;

– автоматична масштабованість*: Azure Web App* надає можливості автоматичної масштабованості, що дозволяє вашому веб-додатку адаптуватися до змін обсягу трафіку без необхідності ручного втручання. Ви можете легко масштабувати свій додаток вертикально (збільшуючи потужність серверів) або горизонтально (збільшуючи кількість серверів);

– підтримка інтеграції з іншими службами *Azure: Azure Web App* легко інтегрується з іншими службами *Azure*, такими як *Azure Database*, *Azure Storage*, *Azure Active Directory* та іншими, для поліпшення функціональності вашого додатку;

– підтримка різних середовищ: ви можете використовувати *Azure Web App* для розгортання веб-застосунків в різних середовищах, таких як продакшн, розробка чи тестування.

Azure Web App використовується для розгортання та управління веб-додатками різного роду, від простих статичних сайтів до складних корпоративних веб-застосунків. Він робить процес розгортання та управління інфраструктурою простішим, дозволяючи розробникам та командам швидко виводити свої веб-додатки в хмарному середовищі.

Удосконалюючи опис можливостей та характеристик *Azure Web App*, важливо відзначити додаткові переваги та аспекти, які роблять цю послугу привабливою для розробників та бізнесів:

– гнучкість налаштувань середовища: *Azure Web App* дозволяє розробникам гнучко налаштовувати середовище виконання для своїх веб-додатків, включаючи параметри виконання, змінні середовища та параметри безпеки. Це дозволяє максимально враховувати специфічні вимоги та унікальні особливості кожного проєкту;

– інтегровані засоби моніторингу та аналізу: *Azure Web App* забезпечує вбудовані засоби моніторингу, які дозволяють вам відстежувати роботу вашого веб-додатку в реальному часі. Це включає в себе метрики продуктивності, стан серверів та інші корисні дані для забезпечення оптимальної ефективності та надійності;

– інтеграція з *DevOps* і іншими інструментами: *Azure Web App* ідеально вписується в процеси розробки *DevOps*, надаючи можливості автоматизації розгортання, тестування та виведення виробленого коду. Інтеграція з такими інструментами, як *Azure DevOps*, *GitHub Actions* та іншими, сприяє створенню безперервного потоку розробки;

– сховище для контейнерів: *Azure Web App* підтримує контейнеризацію, дозволяючи розробникам розгортати свої веб-додатки в контейнерах *Docker*. Це надає більше гнучкості та перенесення для додатків, а також сприяє використанню сучасних архітектурних підходів;

Azure Web App стає надійним та потужним інструментом для розробників, які цінують швидкість розгортання, гнучкість конфігурації та безпеку своїх веб-додатків в хмарному середовищі *Microsoft Azure*.

2.3.4. *Azure DevOps, CI/CD*

Для програмних систем, що складаються з великої кількості частин, постає питання складності розгортання, що було раніше зазначено у пункті 2.3 для мікросервісної та функціональної архітектури. Оскільки кожен елемент таких систем є незалежним і потребує окремої розгортки для кожного елемента, швидкість розгортки такої системи є прямопропорційною до кількості компонентів. Для спрощення цього завдання, існують методи автоматизації побудованих за принципом *CI/CD* (англ. «*continuous integration / continuous delivery*» – укр.

«безперервна інтеграція/безперервна доставка») [9].

Принцип безперервної інтеграції та доставки побудований на розробці вихідного коду малими ітераціями, які розгортаються у короткі строки. За рахунок цього частота тестування та перевірки коду підвищується, що надає можливість випускати надійніші та якісніші програмні системи. Для реалізації принципу *CI/CD* зазвичай застосовується опис послідовних інструкцій з тестування та розгортання системи. Отриманий процес має бути простим, оскільки він буде часто виконуватися автоматизованою системою, як *Azure Pipelines, GitHub Actions, Gitlab CI/CD.*

В рамках розробки створюваного вебдодатку було використано систему Azure Pipelines для опису процесу *CI/CD* для кожного з компонентів розроблюваної системи та його виконання. Описаний процес складається з наступних етапів та зображений на рис. 2.3:

– компіляція коду розроблюваного компоненту;
– запуск юніт-тестів та публікація результатів тестування;
– перевірка готовності інфраструктури до розгортки компонента;
– завантаження необхідних для компоненту секретів, як строка;
– підключення до БД;
– розгортка нової версії додатку.

У представленому процесі етапи 1-2 є частиною безпервної інтеграцій, а 3-5 частинами доставки. Процес автоматизований та дозволяє провести коректну інтеграцію коду при спробі додавання до основної гілки кодової бази та виконує розгортання компоненту при будь-якій зміні основної гілки.

Зокрема, процес *CI/CD* не лише полегшує інтеграцію та розгортання, але й сприяє покращенню якості програмного забезпечення. Використання частих ітерацій із застосуванням юніт-тестів підвищує стабільність та надійність коду, що в свою чергу призводить до зменшення кількості помилок та підвищення продуктивності розробки.

Принцип безперервної інтеграції також визначає активний взаємозв'язок інфраструктури з процесом розгортання. Завдяки перевірці готовності

інфраструктури до розгортки компонента перед введенням змін, уникнуто можливих конфліктів та забезпечено зручні умови для внесення змін.

Окрім того, використання системи *Azure Pipelines* у розробці вебдодатку дозволяє автоматизувати і керувати всіма етапами *CI/CD* для кожного компонента. Це забезпечує консистентність та ефективність усього процесу, а також полегшує відслідковування та аналіз результатів розгортання.

Отже, принцип *CI/CD* є не лише засобом автоматизації, а й потужним інструментом для підвищення якості та швидкості розробки програмного забезпечення, зменшення ймовірності помилок та ефективного впровадження нових функціональностей в робочий продукт.

Перевірка готовності інфраструктури до розгортки компонента: перед введенням змін в основний репозиторій важливо перевірити, чи готова інфраструктура для розгортання нової версії. Це включає в себе перевірку доступності необхідних ресурсів та сервісів.

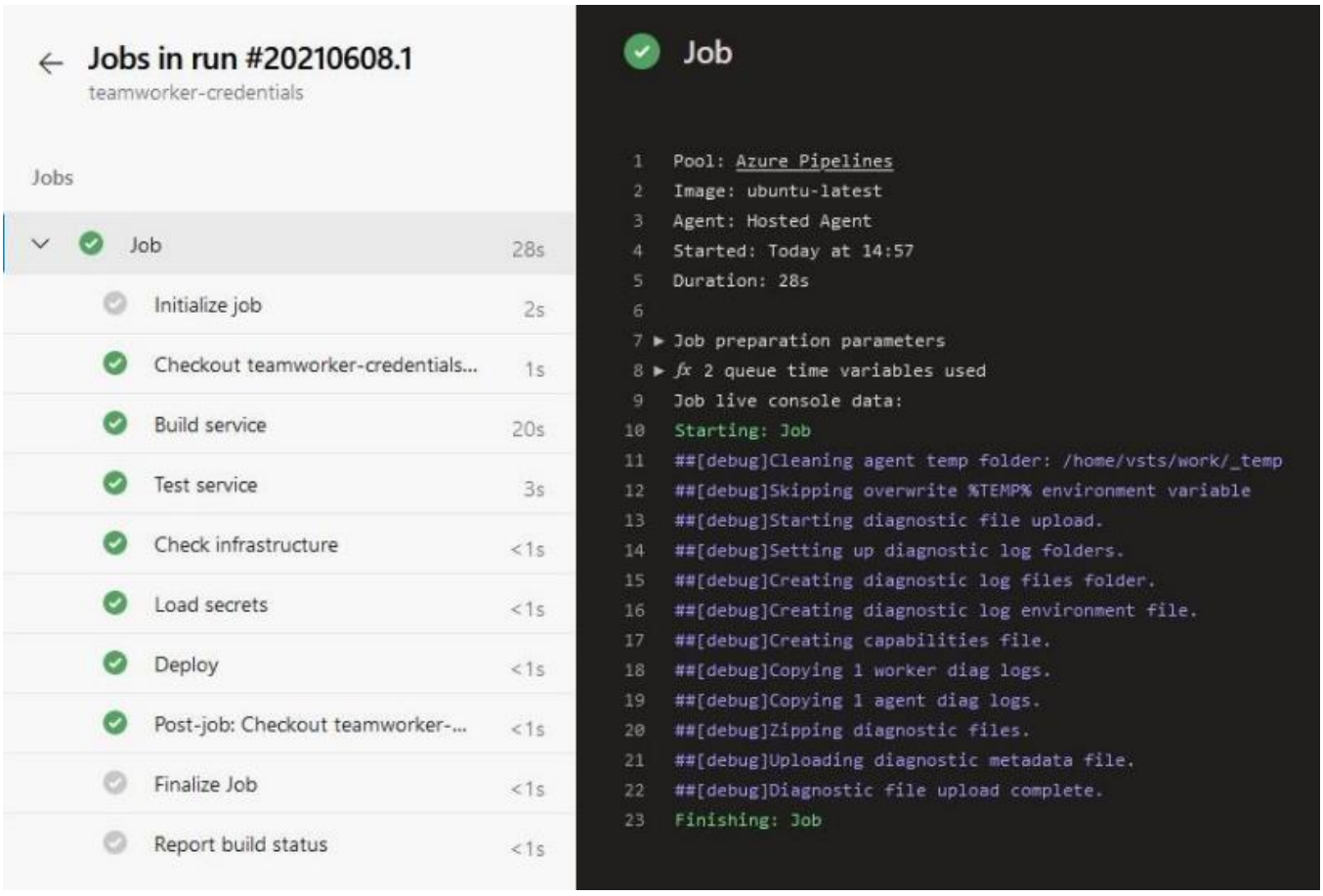

Рис 2.3. Етапи розробленого *CI/CD* процесу

Поглибивши розгляд принципу *CI/CD* у контексті розробки вебдодатку, важливо звернутися до конкретних етапів, що становлять основу цього принципу.

Компіляція коду розроблюваного компоненту: цей етап визначається необхідністю перетворення вихідного коду виконуваного файлу. Він дозволяє виявити синтаксичні помилки та виконати перевірку коду на наявність потенційних проблем ще до введення його в основний репозиторій.

Запуск юніт-тестів та публікація результатів тестування: цей етап становить ключовий компонент забезпечення якості коду. Запуск юніт-тестів дозволяє виявити та виправити помилки на ранніх етапах розробки, а публікація результатів надає зрозумілу звітність команді розробників.

Завантаження необхідних для компоненту секретів: управління конфіденційною інформацією, такою як ключі *API* або паролі, є критичним аспектом безперервної доставки.

2.4. *ElasticSearch*.

Одним з найбільш використовуваних механізмів моніторингу та логування систем у більшості компанії є *ELK Stack* [10], який складається з наступних основних компонентів: *Elasticsearch, Logstash* та *Kibana*. Загалом *ELK Stack* використовується для розробки інструментів моніторингу за журналами (так званими логами). *ELK Stack* має перевагу в тому, що він може розгортатися як *SaaS* в більшості хмарних провайдерах.

Elastic стек – це набір інструментів з відкритим кодом, розроблений *Elastics*, який дозволяє збирати дані з будь-якого типу джерела та в будь-якому форматі для здійснення пошуку, аналізу та візуалізації даних у режимі реального часу.

Він складається з таких продуктів:

– *Elasticsearch*: розповсюджена пошукова система, побудована на *Apache Lucene*. Він дозволяє індексувати та витягувати документи *JSON* (без жорсткої схеми) у різних форматах. Він заснований на *Java* і забезпечує доступний *RESTFUL*

інтерфейс;

– *Logstash*: це двигун, який дозволяє збирати дані з різних джерел, нормалізувати їх і поширювати. Спочатку оптимізовано для файлівжурналів, хоча він може читати дані з багатьох типів джерел;

– *Kibana*: інструмент, що дозволяє візуалізувати та досліджувати в реальному часі велику кількість даних. За допомогою графічного зображення дозволяє легко візуалізувати складні набори даних;

– *Beats*: це завантажувач даних, який встановлюється на сервери і функціонує як агент та надсилають оперативну інформацію в *Elasticsearch*. *Elastic*-стек дозволяє отримувати дані з журнальних файлів за допомогою *Logstash* та зберігати їх в аналітичній пошуковій системі *Elasticsearch*. Крім того, це дозволяє виконувати візуалізацію, моніторинг та управління даними в режимі реального часу через веб-інтерфейс *Kibana*.

З іншого боку, пропонується включити *ElastAlert*, інструмент з відкритим вихідним кодом, інтегрованим з *Elastic*-стек і дозволяє виявити аномалії та невідповідності даних *Elasticsearch*, а також виконувати сповіщення.

Набір інструментів *Elastic*-стеку, а також *ElastAlert* дозволяє створити систему моніторингу для вилучення, зберігання та експлуатації даних процесів або систем, що підлягають моніторингу. Це дозволяє отримувати дані з журнальних файлів через *Logstash* та зберігати їх у системі пошуку та аналізу *Elasticsearch*. Крім того, це дозволяє візуалізацію, моніторинг та експлуатацію даних у режимі реального часу через *Kibana* та конфігурацію попереджень з *ElastAlert*.

Для зберігання даних у *Elasticsearch Logstash* використовує вихідний плагін *Elasticsearch*, який дозволяє надсилати дані через протокл *HTTP* та *HTTP*. *Elasticsearch* буде відповідальним за індексацію даних, отриманих із журналів, та можливість їх пошуку та аналізу. Схема обробки даних за допомогою *ELK* стеку наведена на рис. 2.4.

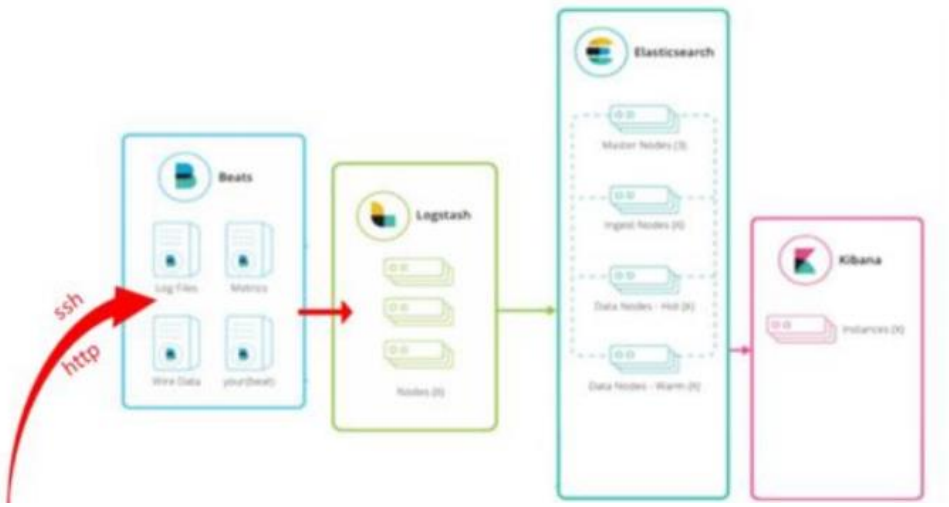

Рис 2.4. Схема обробки даних за допомогою *Elastic* стеку

2.5. Мова програмування *C#*

Мова програмування C# є об'єктно-орієнтованою та компонентно-орієнтованою. Вона надає мовні конструкції для підтримки цих концепцій, роблячи C# природною для створення та використання програмних компонентів. З моменту свого виникнення C# активно розвивалася, додаючи функції для підтримки нових робочих завдань та методів програмного забезпечення.

Однією з переваг C# є можливість створення надійних та довговічних програм. Автоматичний збір сміття відновлює пам'ять, звільнену невикористаними об'єктами. Типи, які допускають значення NULL, захищають від змінних, які не посилаються на виділені об'єкти. Обробка винятків забезпечує структурований підхід до виявлення та відновлення помилок.

Мова підтримує лямбда-вирази для функціонального програмування, а синтаксис мовного інтегрованого запиту (LINQ) створює єдиний шаблон для роботи з даними з будь-якого джерела. Підтримка асинхронних операцій дозволяє побудову розподілених систем.

C# має уніфіковану систему типів, де всі типи успадковуються від одного кореневого типу об'єкта. Це дозволяє зберігати, транспортувати та використовувати значення будь-якого типу узгоджено. Мова також підтримує як визначені

користувачем типи посилань, так і типи значень.

Ітератори, що надає C#, дозволяють реалізаторам колекцій визначати користувацьку поведінку для клієнтського коду. Зокрема, C# акцентує на версійності для забезпечення сумісності програм та бібліотек у часі. Аспекти дизайну, такі як віртуальні модифікатори та модифікатори перевизначення, сприяють цій версійності.

Типи визначають структуру та поведінку даних у C#. Змінні можуть бути типів значень або посилань. Змінні типів значень містять свої дані напряму, тоді як змінні типів посилань зберігають посилання на свої дані (об'єкти). Це дозволяє двом змінним посилатися на один і той же об'єкт, а операції з однієї змінної впливати на іншу. З типами значень кожна змінна має власну копію даних, що забезпечує їх ізольованість.

Ідентифікатори визначають імена змінних та можуть бути зарезервованими словами C#, якщо використовують префікс @. Це корисно для взаємодії з іншими мовами.

Система типів C# включає в себе прості та складені типи, і надає можливість робити визначені користувачем типи, структури, перерахування, класи, інтерфейси, масиви та делегати.

2.6. Середовище розробки *Visual Studio*

Microsoft Visual Studio [14] представляє інтегроване середовище розробки (IDE) від компанії Microsoft. Використовується для створення комп'ютерних програм, веб-сайтів, веб-додатків, веб-сервісів та мобільних додатків. Visual Studio базується на платформах розробки програмного забезпечення Microsoft, таких як Windows API, Windows Forms, Windows Presentation Foundation, Windows Store і Microsoft Silverlight. Здатне генерувати як рідний, так і керований код.

У складі Visual Studio включено редактор коду, що підтримує IntelliSense (автоматичне завершення коду) та можливості рефакторингу коду. Інтегрований налагоджувач працює на рівні джерела та на рівні машини. Інші вбудовані

інструменти включають профайлер коду, конструктор для створення додатків із графічним інтерфейсом користувача, веб-дизайнер, конструктор класів і дизайнер схем баз даних. Відкритий до плагінів, що розширюють функціональність на різних рівнях, включаючи підтримку систем контролю версій (наприклад, Subversion і Git) та додавання нових інструментів, таких як редактори та візуальні дизайнери для мов, пов'язаних з доменом, або інструментів для різних аспектів життєвого циклу розробки програмного забезпечення (наприклад, Azure DevOps: Team Explorer).

Visual Studio підтримує 36 різних мов програмування і надає можливість редактору коду та налагоджувачу підтримувати (в різній мірі) практично будь-яку мову програмування за умови наявності спеціальної мови. Вбудовані мови включають C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML і CSS. Підтримка інших мов, таких як Python, Ruby, Node.js та M, серед інших, здійснюється через плагіни. Раніше була підтримка для Java (і J#).

Найпростіша версія Visual Studio, Community, доступна безкоштовно і призначена для студентів, відкритих та індивідуальних розробників. Гасло видання Visual Studio Community – «Безкоштовна повнофункціональна IDE для студентів, відкритих і індивідуальних розробників».

На момент 8 листопада 2021 року актуальна версія Visual Studio - 2022, а старіші версії, такі як 2013 і 2015, перебувають у розширеній підтримці, а 2017 та 2019 - у основній підтримці. Рис. 2.3 показує початок роботи та створення нового проєкту в Visual Studio.

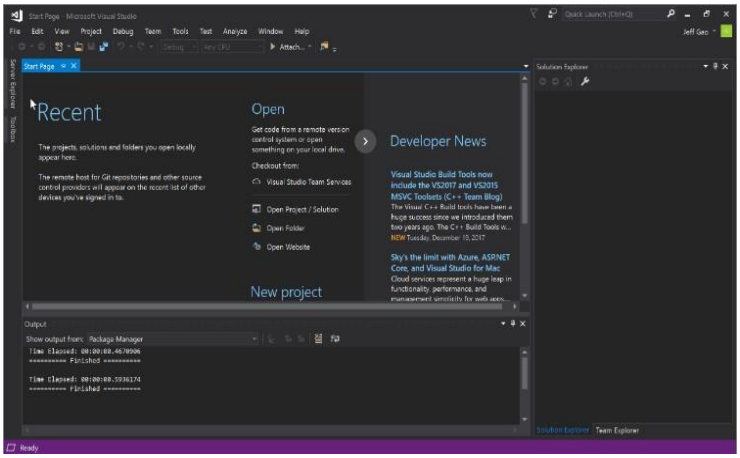

Рис. 2.5. *Visual Studio*

2.7. Технологія для розробки інтерфейсу

MudBlazor та *AntBlazor* представляють собою бібліотеки компонентів інтерфейсу користувача для фреймворку *Blazor*, який дозволяє розробникам використовувати мову програмування *C#* та *.NET* для створення веб-застосунків, що виконуються в браузері. Основні характеристики кожної з цих бібліотек варто розглянути більш детально:

MudBlazor визначається своєю розширеною функціональністю та гнучкістю. Вона надає розбірливий та легкий у використанні *API* для розробки інтерфейсу користувача, включаючи різноманітні компоненти, такі як кнопки, форми, таблиці та інші. Завдяки *MudBlazor*, розробники можуть швидко та ефективно створювати стильні веб-додатки з допомогою компонентів, що допомагають спростити рутинні завдання фронтенд-розробки.

AntBlazor, як ім'я вказує, базується на дизайн-мові *Ant Design*, яка визначає стандарти дизайну, що часто використовуються в інтерфейсах користувача. Ця бібліотека пропонує компоненти, які відповідають цим стандартам, що дозволяє розробникам легко інтегрувати сучасний та стильний дизайн в свої веб-застосунки. *AntBlazor* забезпечує консистентність та високий рівень професіоналізму у вигляді та поведінці компонентів, що є характерною рисою *Ant Design*.

В обох випадках, вибір між *MudBlazor* та *AntBlazor* може бути залежний від конкретних потреб проекту, дизайн-вимог, а також від особистих вподобань розробника. Обидві бібліотеки спрощують процес фронтенд-розробки в середовищі *Blazor*, пропонуючи готові рішення для різних елементів інтерфейсу користувача.

LeafletJS – це відкрите програмне забезпечення для створення інтерактивних карт веб-сайтів. Ця *JavaScript* бібліотека надає простий та легкий у використанні інтерфейс для відображення картографічної інформації та взаємодії з користувачем на веб-сторінці. *LeafletJS* була розроблена з урахуванням простоти використання, легкої інтеграції та високої продуктивності.

Основні характеристики та можливості *LeafletJS* включають:

– легкість використання: *LeafletJS* пропонує простий *API*, що дозволяє

розробникам швидко інтегрувати карти на свої веб-сторінки. Зазначення координат, додавання шарів та інші операції відбуваються з легкістю;

– підтримка різних постачальників карт: бібліотека підтримує використання різних постачальників картографічних даних, таких як *OpenStreetMap*, *Mapbox*, *Bing Maps* та інших. Це дозволяє розробникам вибирати найбільш підходящий постачальник для їхніх потреб;

– інтерактивність та анімації: *LeafletJS* дозволяє створювати інтерактивні карти, де користувачі можуть взаємодіяти з об'єктами на карті, такими як маркери, полігони та лінії. Також надається можливість додавання анімаційних ефектів для покращення користувацького досвіду;

– розширюваність через плагіни: *LeafletJS* підтримує використання плагінів, які дозволяють розширювати функціональність бібліотеки. Це дозволяє додавати нові можливості та функції в залежності від потреб проекту.

LeafletJS часто використовується веб-розробниками для вбудовування інтерактивних карт у веб-сайти та веб-додатки, такі як туристичні ресурси, сервіси доставки та інші проекти, де картографічна інформація є важливою частиною користувацького інтерфейсу.

Фреймворк *AntBlazor.Chart*.

AntBlazor.Chart – це безкоштовний і відкритий фреймворк для створення графіків і діаграм у веб-додатках, написаних на *Blazor*. Він заснований на бібліотеці *G2Plot*, яка забезпечує широкий спектр можливостей для створення візуальних представлень даних.

Фреймворк *AntBlazor.Chart* підтримує широкий спектр типів графіків і діаграм, включаючи:

– стовпчасті графіки;

– лінійні графіки;

– кругові діаграми;

– порівняльні діаграми;

– розподільчі діаграми;

– інтерактивні графіки.

Фреймворк також забезпечує широкий спектр налаштувань для стилізації графіків і діаграм, включаючи:

– кольори;
– шрифти;
– формати даних;
– розміри;
– межі;
– шкала.

Для використання фреймворка *AntBlazor.Chart* потрібно створити компонент, який розширює клас *BaseChart*. Цей клас надає методи для додавання даних до графіка, налаштування його стилю та керування інтерактивністю.

Фреймворк *AntBlazor.Chart* також забезпечує підтримку інтерактивності. Наприклад, можна додати можливість користувачам переміщувати курсор миші над графіком, щоб побачити додаткову інформацію про дані.

AntBlazor.Chart – це потужний і гнучкий інструмент для створення графіків і діаграм у веб-додатках, написаних на *Blazor*. Він забезпечує широкий спектр можливостей для створення візуальних представлень даних, включаючи підтримку різних типів графіків і діаграм, налаштування стилю та інтерактивності.

2.8. Бібліотека *Entity framework*

Entity Framework [15] є спеціалізованою об'єктно-орієнтованою технологією, побудованою на базі фреймворку .NET, призначеною для роботи з даними. Основний відмінник від традиційних інструментів ADO.NET полягає в тому, що ADO.NET взаємодіє з базами даних через створення підключень, команд та інших об'єктів, тоді як Entity Framework надає вищий рівень абстракції. Це дозволяє абстрагуватися від конкретної бази даних і працювати з даними, незалежно від типу сховища. В Entity Framework вже на концептуальному рівні взаємодіємо з об'єктами, використовуючи об'єктно-орієнтований підхід.

Перша версія Entity Framework вийшла у 2008 році і мала базову підтримку

ORM (object-relational mapping - відображення даних на об'єкти) та лише один підхід до взаємодії з БД - Database First. З версією 4.0 у 2010 році з'явилися нові можливості: підходи Model First та Code First, і Entity Framework став рекомендованою технологією для доступу до даних.

Випуск версії 5.0 у 2012 році приніс додаткові покращення функціоналу, а у 2013 році вийшов Entity Framework 6.0, який підтримує асинхронний доступ до даних.

Ключовим поняттям в Entity Framework є "сутність" (entity). Сутність - це набір даних, пов'язаних з конкретним об'єктом, і відображається на клас у програмному коді. Таким чином, Entity Framework дозволяє працювати не з таблицями та стовпцями, а з об'єктами та їхніми наборами.

Кожна сутність має свої властивості, які визначають її характеристики. Ці властивості можуть представляти не лише прості дані (наприклад, числа), але і складні структури даних. Кожна сутність також може мати ключі, які унікально визначають цю сутність.

Entity Framework використовує мову запитів LINQ для вибірки даних з бази даних. LINQ дозволяє витягувати не лише рядки з бази даних, але і об'єкти, пов'язані з різними асоціативними зв'язками.

Ще однією ключовою концепцією є Entity Data Model, який відображає класи сутностей на реальні таблиці в базі даних. Цей модель складається з концептуального, рівня сховища і рівня зіставлення (мапінгу). Концептуальний рівень визначає класи сутностей у програмі, рівень сховища визначає структуру бази даних, а рівень зіставлення встановлює відповідності між ними.

Entity Framework пропонує три основні способи взаємодії з базою даних:

- Database First: генерація кл

асів для моделі конкретної бази даних.

- Model First: спочатку створення моделі бази даних, яка потім використовується для генерації реальної бази даних.

- Code First: визначення класів моделі даних у програмному коді, які потім використовуються для створення бази даних.

Для використання Entity Framework слід завантажити відповідні бібліотеки за допомогою NuGet Package Manager, що є системою управління пакетами для платформ розробки Microsoft, зокрема для бібліотек .NET Framework.

2.9. Середовище *Microsoft SQL Server*

Microsoft SQL Server [16] - це система управління реляційною базою даних (RDBMS), яка широко використовується для обробки транзакцій, бізнес-аналітики та аналітики в корпоративних IT-середовищах. Вона є однією з провідних технологій баз даних на ринку, разом із Oracle Database і IBM DB2.

Основна характеристика Microsoft SQL Server - це те, що вона побудована на основі SQL (Structured Query Language), стандартизованої мови програмування, яку використовують адміністратори баз даних та інші IT-фахівці для керування базами даних і виконання запитів до них. Microsoft SQL Server використовує Transact-SQL (T-SQL), реалізацію SQL від Microsoft, яка додає власні розширення програмування до стандартної мови.

Основний компонент Microsoft SQL Server - це SQL Server Database Engine, який контролює зберігання, обробку та безпеку даних. Він включає реляційний механізм для обробки команд та запитів, а також механізм зберігання для управління файлами баз даних, таблицями, індексами, буферами даних та транзакціями. До складу Database Engine входять збережені процедури, тригери, уявлення та інші об'єкти бази даних.

На рівні операційної системи SQL Server або SQLOS відповідає за функції нижнього рівня, такі як управління пам'яттю, вводом-виводом, планування завдань та блокування даних. На рівні мережевого інтерфейсу використовується протокол табличного потоку даних Microsoft для взаємодії з серверами баз даних.

Microsoft SQL Server також включає різноманітні інструменти для управління даними, бізнес-аналітики та аналітичних інструментів. До них входять SQL Server Analysis Services для обробки даних в бізнес-аналітичних додатках, SQL Server

Reporting Services для створення та доставки звітів BI, а також інші служби для інтеграції, якості даних і основних даних.

Microsoft пропонує різні версії SQL Server, включаючи безкоштовні версії для розробників та експрес-версію для запуску невеликих баз даних. Інші версії включають стандартну, веб-версію, версію Business Intelligence та Enterprise, яка має повний набір функцій. Пакет оновлень 1 для SQL Server 2016 розширив доступність деяких функцій, раніше обмежених версією Enterprise, в стандартні та експрес-версії.

Висновки за розділом

У цьому розділі розглянуті обрані інструменті та технології, які були необхідні для створення програмного продукту. Розроблений програмний продукт написано об’єктно-орієнтованою мовою програмування *C#,* платформи *.NET Framework*. Розробка проводилась з використанням середовища *Microsoft Visual Studio*. Для роботи з базою даних було обрано та використано Microsoft *SQL Server* та бібліотеку *Entity Framework.*

Додатково, важливим аспектом обраного набору технологій є їхня висока сумісність та інтеграція, що сприяє ефективному взаємодії різних компонентів програмного продукту. Використання платформи *.NET* дозволяє забезпечити зручний та швидкий обмін інформацією, що робить продукт ще більш доступним для користувачів.

Мова програмування *C#* обрана не лише через її об'єктно-орієнтований підхід, але й через великий екосистем *.NET*, що включає багато готових бібліотек та інструментів, що полегшують розробку та підтримку програмного продукту.

Використання *Microsoft SQL Server* та *Entity Framework* забезпечує надійність та ефективність в роботі з базою даних, а також спрощує роботу зі зміною схеми бази даних та виконанням розширень. Це важливо для забезпечення гнучкості та швидкості реагування на зміни вимог користувачів чи умов використання.

Загальний вибір технологій та інструментів підтримує високий рівень професіоналізму та компетентності в розробці програмного продукту, а також

створює надійну основу для подальшого розвитку та вдосконалення системи.

Крім того, важливо відзначити, що в процесі розробки програмного продукту вдалося успішно впровадити принципи *Clean Code* та *SOLID*, що сприяло підтримці високого рівня читабельності коду, його модульності та можливості ефективного розширення в майбутньому.

Використання середовища *Microsoft Visual Studio* надало можливість використовувати різноманітні інструменти для аналізу та відлагодження коду, що спростило виявлення та усунення помилок, а також підвищило продуктивність розробки.

Важливим аспектом є також врахування принципів безпеки та конфіденційності даних у процесі вибору технологій для зберігання та обробки інформації. *Microsoft SQL Server* та *Entity Framework* забезпечують високий рівень захисту та дозволяють ефективно керувати доступом до конфіденційних даних.

Узагальнюючи, обраний стек технологій дозволяє створити не лише функціонально високоякісний продукт, але й забезпечити його стабільність, безпеку та готовність до майбутніх розширень. Результати цієї роботи ставлять під сумнів високий потенціал застосування розробленого рішення в практичних умовах та підкреслюють значущість вибору технологій у контексті досягнення мети даного дослідження.

РОЗДІЛ 3
ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

3.1. Архітектура додатку для роботи з БД

Архітектура Onion (Цибуля) є формою багатошарової архітектури, яку було вперше представлено Джеффрі Палермо для подолання проблем традиційного підходу до N-шарової архітектури. Ця архітектура може бути уявлена у вигляді концентричних кіл, а кожен шар виконує певну функцію в додатку.

В архітектурі Onion виділяються чотири основні шари:

1. Рівень домену:

- Найвнутрішніший шар, представляє основну логіку додатку, включаючи бізнес-логіку та сутності.

- Тут знаходяться об'єкти домену, які відображають ключові поняття та бізнес-правила.

2. Рівень сервісу:

- Містить сервіси, які надають конкретний функціонал для рівня презентації.

- Ці сервіси використовують об'єкти домену для виконання конкретних завдань.

3. Рівень інфраструктури:

- Відповідає за зовнішні технічні аспекти, такі як зберігання даних, комунікація з базою даних та інші технічні операції.

- Тут можна знаходити код для роботи з базами даних, зовнішніми сервісами та інші технічні аспекти.

4. Шар презентації:

- Забезпечує взаємодію з користувачем, включаючи інтерфейс користувача та логіку обробки користувацьких взаємодій.

- Використовує сервіси для виконання конкретних дій та відображення даних.

Ключовою концепцією архітектури Onion є використання інверсії залежностей (Dependency Inversion). Усі шари взаємодіють лише через інтерфейси, визначені на рівнях нижче, зі строго спрямованим потоком залежностей до ядра Onion. Це дозволяє прозоро вимкнути реалізацію під час виконання та створює гнучкість у зміні частин додатку без впливу на інші частини.

Основні переваги архітектури Onion включають прозорість взаємодії між шарами, легкість тестування та зміни, а також зменшення залежностей від конкретних реалізацій усіх компонентів системи.

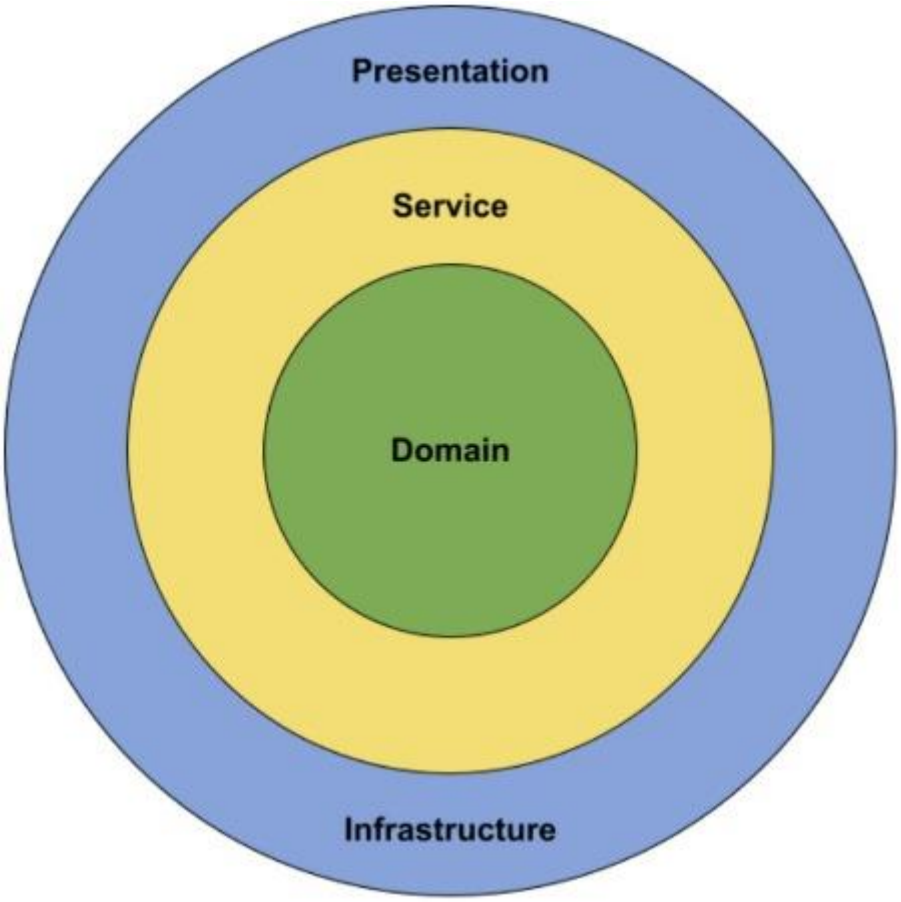

Рис. 3.1. Шари архітектури *Onion*

Високий рівень тестованості забезпечується архітектурою Onion завдяки великому значенню абстракцій. Зокрема, можливість насміювання абстракцій за

допомогою бібліотек, таких як Moq, сприяє легкості написання бізнес-логіки без необхідності враховувати деталі реалізації. Створюючи інтерфейс для зовнішніх систем або служб, ми можемо використовувати його без необхідності займатися деталями реалізації, оскільки вищі шари Onion відповідають за прозору реалізацію цих інтерфейсів.

Основна ідея архітектури Onion полягає в потоці залежностей між її шарами. Чим глибше шар розташований всередині Onion, тим менше він залежить від інших шарів. Доменний рівень, зокрема, є ізольованим від зовнішніх шарів і дозволяє їм звертатися лише до шарів, які знаходяться нижче в ієрархії.

Загалом можна визначити, що всі залежності в архітектурі Onion протікають всередину системи. Потік залежностей визначає, які дії може виконувати кожен шар архітектури, обмежуючи їх методами, які надаються нижчими шарами.

Використовуючи цей підхід, ми можемо інкапсулювати бізнес-логіку на рівнях домену та сервісу, уникнувши деталей реалізації. На рівні сервісу ми залежимо лише від інтерфейсів, які визначені доменним рівнем.

Діаграма варіантів використання системи для візуалізації та аналізу даних з рейсових записів польотів представлена на рис. 3.2.

Концептуальна модель, яка є уявленням про систему, складається з концепцій для полегшення розуміння або моделювання предмету. Фізичні моделі, навпаки, представляють фізичні об'єкти, такі як іграшкові моделі. У штучному інтелекті концептуальні моделі використовуються для побудови експертних систем і систем, заснованих на знаннях, для відтворення експертної думки..

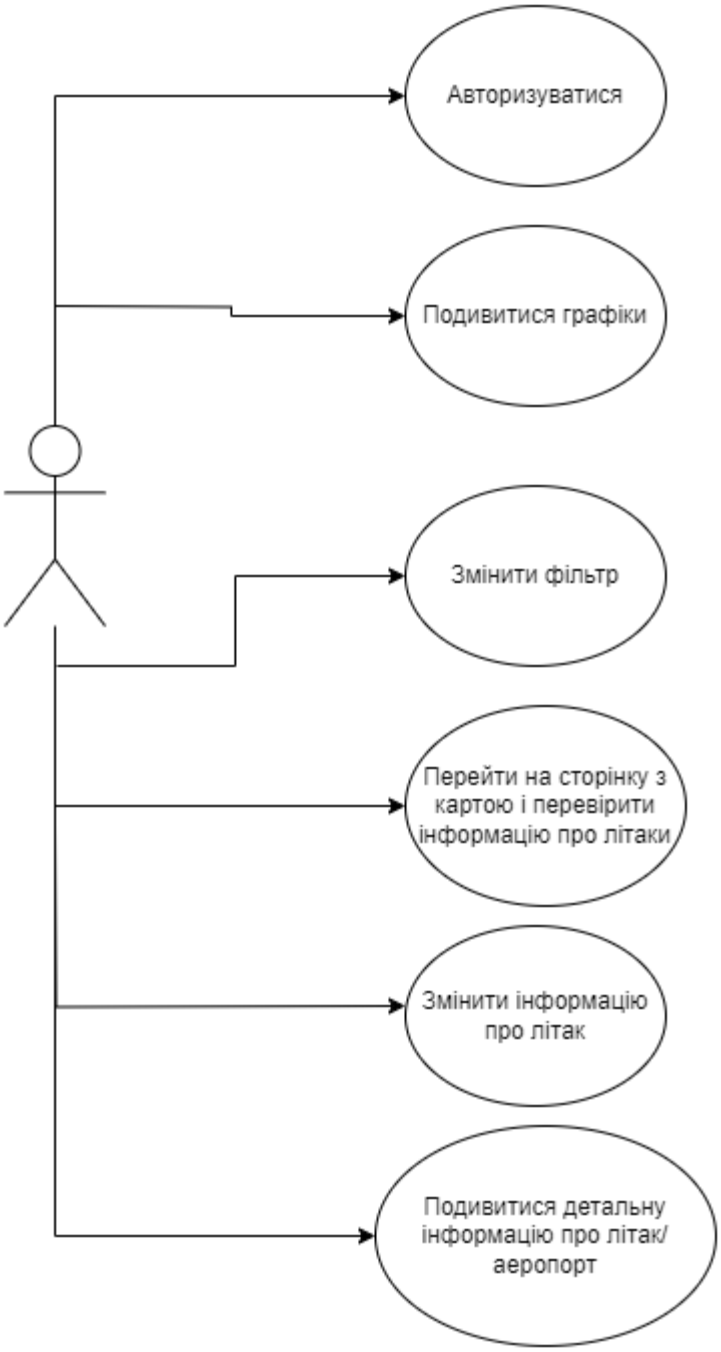

Рис. 3.2. Діаграма варіантів використання

Термін може вказувати на моделі, які виникають після процесу концептуалізації або узагальнення. Концептуальні моделі часто є абстракціями реальних об'єктів у фізичному чи соціальному світі. Семантичні дослідження стосуються різних етапів формування понять, зокрема, семантика зосереджена на значеннях, які мислячі істоти присвоюють різним елементам свого досвіду.

Концептуальна модель створеного програмного продукту відображена на рис. 3.3.

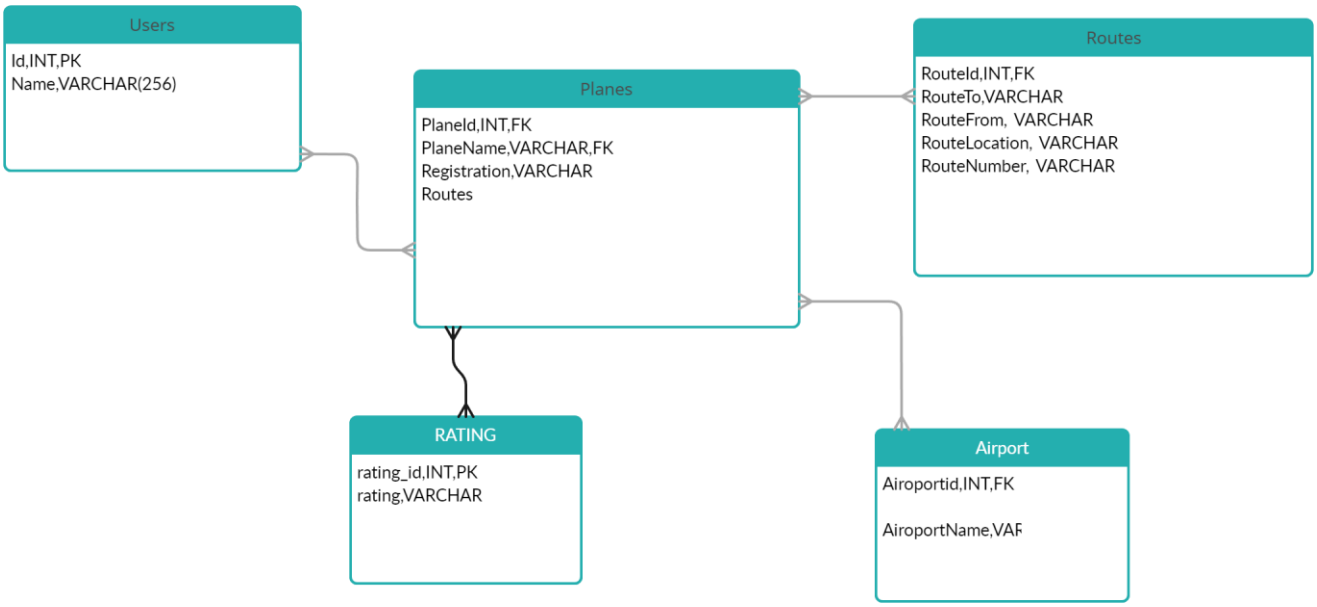

Рис. 3.3. Концептуальна модель

3.1.2. Шаблон *MVP*

Шаблон MVP [19] подібний до шаблону MVC, виходячи із якого контролер замінюється представником. Мінімально життєздатний продукт, або MVP, представляє собою продукт із достатньою кількістю функцій для залучення клієнтів на ранньому етапі та підтвердження ідеї продукту у початковому циклі розробки. У сферах, таких як програмне забезпечення, MVP допомагає команді продукту отримувати швидкі відгуки користувачів для подальших ітерацій та вдосконалення продукту.

Оскільки методологія agile базується на перевірці та ітераціях продуктів з врахуванням відгуків користувачів, MVP відіграє ключову роль у розробці за принципами agile. Ерік Райс, автор концепції мінімально життєздатного продукту, в рамках методології Lean Startup, визначає його мету як версію нового продукту, що дозволяє збирати максимальну кількість підтверджених знань про клієнтів з мінімальними зусиллями.

Розробка мінімально життєздатного продукту виправдана у випадках, коли команда продукту має намір:

- Швидко вивести продукт на ринок;

- Перевірити ідею з реальними користувачами перед виділенням великого

бюджету на повну розробку продукту;

- Дізнатися, що відповідає цільовому ринку компанії та що ні.

Крім того, MVP допомагає мінімізувати час та ресурси, які можна було б витратити на створення продукту, який може не досягти успіху. Компоненти перегляду, такі як активності, фрагменти та інші, реалізують цей інтерфейс та передають дані у відповідний спосіб (рис. 3.4).

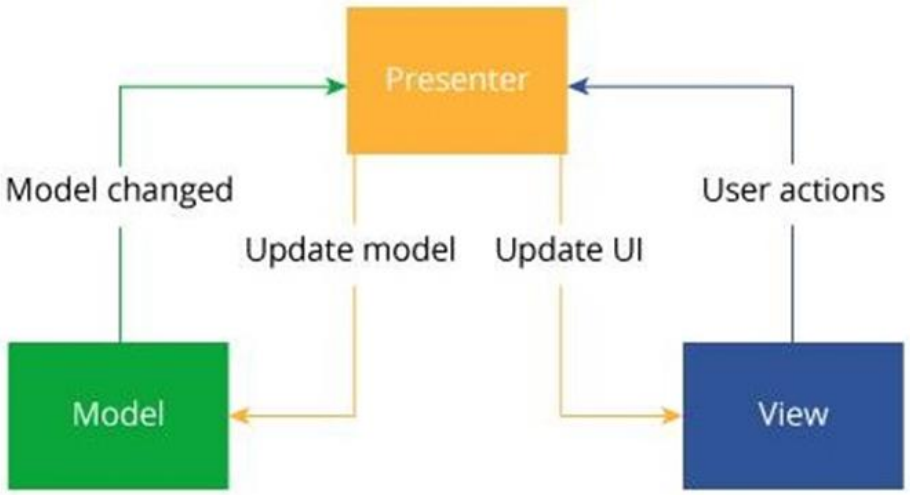

Рис. 3.4. Шаблон *MVP*

У шаблоні дизайну *MVP* представник маніпулює моделлю, а також оновлює подання. У *MVP View* і *Presenter* повністю відокремлені один від одного і спілкуються один з одним за допомогою інтерфейсу.

У контексті шаблону *MVP*, розглядається важливий аспект взаємодії між компонентами. Презентер виступає як посередник між моделлю та представленням, керуючи введенням від користувача, викликами *API*, та іншими подіями.

Важливо зазначити, що *MVP* дозволяє визначити контракт між представленням та презентером, що робить код більш розширюваним та обслуговуваним. Це також забезпечує можливість тестування кожного компонента окремо.

Застосування *MVP* особливо актуально на етапах ранньої розробки, коли важливо швидко пристосовуватися до змін і отримувати швидку зворотню зв'язок від користувачів. *MVP* допомагає уникнути зайвого коду в представленні та моделі, забезпечуючи чистий та структурований дизайн.

Загалом, застосування шаблону *MVP* важливе для команд, які прагнуть швидко розвивати та оптимізувати свої продукти, отримуючи при цьому максимальну цінність від взаємодії з користувачами та ринком.

Крім того, шаблон *MVP* дозволяє створювати більш гнучкі та переносні рішення. Передавання логіки введення користувача на рівень презентера дозволяє ефективно використовувати цю логіку у різних частинах програми. Наприклад, один і той самий презентер може використовуватися в різних частинах програми, що полегшує переносність коду та зменшує його дублювання.

Застосування *MVP* дозволяє підтримувати високий рівень відокремленості між різними компонентами програми. Це забезпечує можливість змінювати один компонент без впливу на інші, що робить код більш масштабованим та обслуговуваним у великих проектах.

Крім того, *MVP* відкриває можливості для реакції на зміни вимог користувачів. Благодаря чітко визначеному інтерфейсу між презентером та представленням, зміни в інтерфейсі користувача можна ефективно впроваджувати без впливу на бізнес-логіку та модель даних.

У цілому, шаблон *MVP* створює збалансовану архітектуру для розробки програм, забезпечуючи ефективну роботу команд та швидкий розвиток продуктів.

3.1.3. Шаблон Репозиторій

Шаблон Repository [20] призначений для створення абстракції між рівнем доступу до даних та рівнем бізнес-логіки програми, щоб захистити програму від змін у сховищі даних і полегшити автоматичне модульне тестування або розробку, орієнтовану на тестування (TDD).

Для кожного типу сутності створюється інтерфейс репозитарію та його реалізація. Контролер отримує екземпляр сховища, використовуючи інтерфейс, щоб приймати посилання на будь-який об'єкт, який реалізує інтерфейс репозитарію. Наприклад, контролер, що працює на веб-сервері, може використовувати репозитарій, який взаємодіє з Entity Framework, а контролер для модульного тестування може використовувати імітаційне сховище з легкістю маніпулювати для тестування.

На прикладі курсу та відділу в контролері курсу використовується кілька сховищ та одиничний робочий клас. Одиниця роботи координує роботу сховищ, створюючи єдиний клас контексту бази даних для всіх них. Використання інтерфейсів для цих класів полегшує автоматичне модульне тестування, але на практиці класи можуть використовуватися без інтерфейсів для спрощення реалізації.

Використання шаблону Repository відзначається тим, що він створює абстракцію для роботи з даними, що дозволяє контролеру легко взаємодіяти з різними сховищами. Це не тільки спрощує реалізацію, але й забезпечує єдинообразний інтерфейс для роботи з даними незалежно від конкретного сховища.

Важливо відзначити, що використання шаблону Repository полегшує підтримку єдинообразного інтерфейсу для взаємодії з даними, спрощуючи розробку та утримання програмного продукту. Цей підхід також робить систему більш гнучкою до змін, дозволяючи легко переходити від одного сховища даних до іншого.

3.2. Використання залежностей *Dependency Injection*

Впровадження залежностей (Dependency Injection, DI) є шаблоном проєктування програмного забезпечення, підтримуваним ASP.NET Core. Цей шаблон спрямований на досягнення інверсії керування (IoC) між класами та їх залежностями. Коли класи створюють свої об'єкти за допомогою ключового слова "new", це може призвести до тісно пов'язаного коду, що ускладнює його перевірку, підтримку та розширення.

Ін'єкція залежностей (Dependency Injection) - це принцип, за яким об'єкти, які потрібні класу, передаються в нього ззовні, а не створюються всередині нього. Це зазвичай відбувається через конструктор, де клас отримує необхідні об'єкти як параметри конструктора.

Проблема, яку DI намагається вирішити, полягає в уникненні тісного пов'язання між класами та їх залежностями. Коли змінні-члени оголошуються за допомогою інтерфейсів, це робить класи тісно пов'язаними. Це ускладнює

підтримку та розширення коду, оскільки будь-які зміни в реалізації можуть вимагати змін в усіх класах, які створюють екземпляри цього класу.

Ін'єкція залежностей дозволяє подавати залежності через конструктор, роблячи код більш гнучким та легким для тестування. Крім того, це відокремлює відповідальність за створення залежностей від класів, дотримуючись принципу єдиної відповідальності.

Ін'єкція залежностей полегшує тестування, оскільки можна висміювати поведінку інтерфейсів в тестах, не створюючи фактичні об'єкти. Це робить тести більш гнучкими та піддає їхню структуру змінам.

Загальна перевага DI полягає в тому, що він сприяє створенню слабкосв'язаних та легко тестових класів, роблячи програмний код більш гнучким, розширюваним та обслуговуваним.

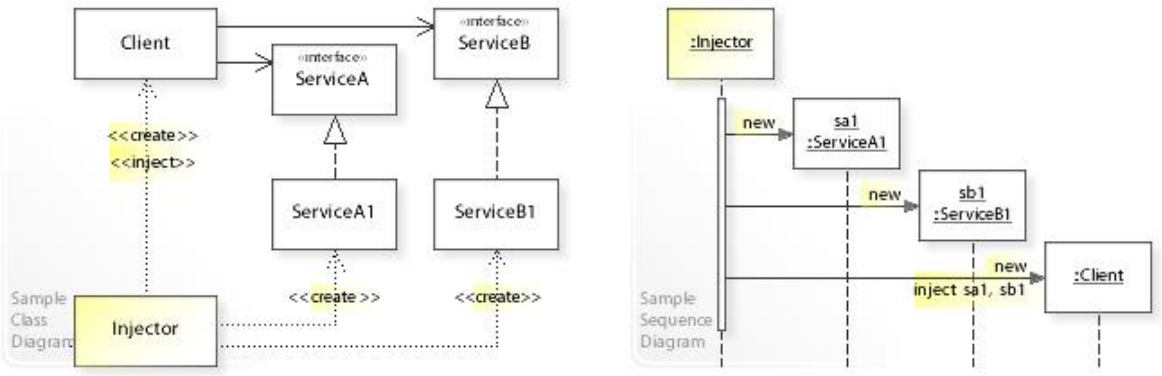

Рис. 3.5. Впровадження залежностей

3.3. Шар доступу до даних

Шар доступу до даних (Data Access Layer – DAL) в програмному забезпеченні виступає як проміжний шар, який надає спрощений доступ до даних, що зберігаються в постійному сховищі, такому як реляційна база даних. Його структура та компоненти включають різні елементи для забезпечення ефективної роботи з базою даних. Нижче наведено короткий опис ключових компонентів шару доступу до даних:

1. Моделі Даних (Data Models): Це представлення об'єктів та їх взаємозв'язків у програмі, які відображають сутності в базі даних. Моделі даних допомагають у визначенні структури та характеристик даних, які будуть використовуватися в програмі.

2. Об'єктно-Реляційне Відображення (ORM): Використання ORM-фреймворку дозволяє автоматизовано взаємодіяти між об'єктами програми та записами в базі даних. Це спрощує роботу з базою даних, конвертуючи об'єкти в записи бази даних та навпаки.

3. Класи Репозиторіїв (Repository Classes): Репозиторії надають інтерфейс для взаємодії з базою даних через методи, що дозволяють виконувати операції читання та запису. Вони служать проміжним рівнем між об'єктами програми та базою даних.

4. Контекст Бази Даних (Database Context): Цей компонент відображає сесію взаємодії з базою даних. Він відстежує зміни та забезпечує управління транзакціями, дозволяючи групувати кілька операцій в одну транзакцію.

5. З'єднання та Пул Підключень (Connection and Connection Pooling): Ці компоненти забезпечують управління з'єднаннями до бази даних та ефективне використання ресурсів через пул підключень. Пул підключень дозволяє ефективно використовувати підключення, уникати їхнього постійного відкриття та закриття.

Шар доступу до даних є ключовим елементом проєкту, що відображає сучасні стандарти та кращі практики в програмуванні. Використання об'єктно-реляційного відображення (ORM) дозволяє ефективно взаємодіяти з базою даних, спрощуючи роботу розробників та роблячи програмний код більш читабельним та підтримуваним.

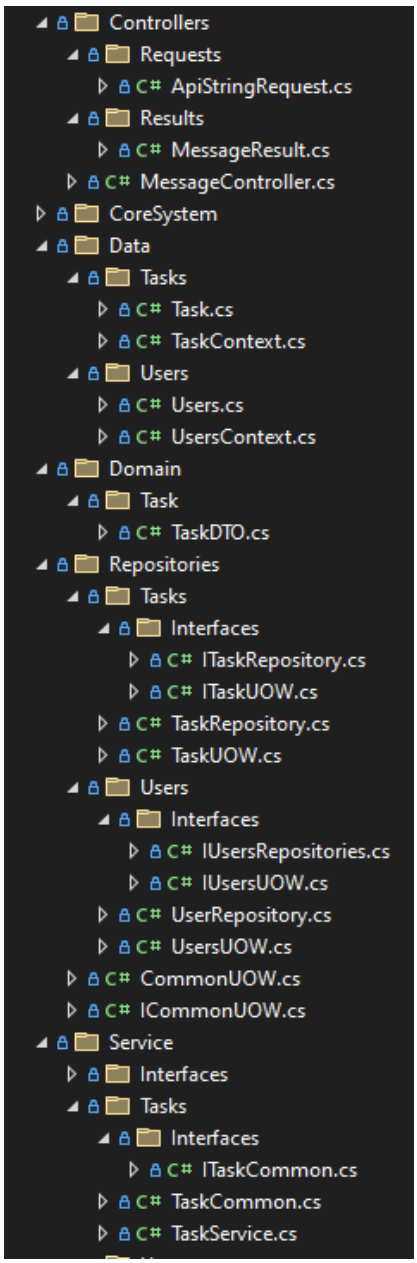

Рис. 3.6. Доступ до даних

Шар доступу до даних в системі відповідає за ефективне управління та взаємодію з даними, використовуючи відповідні моделі, що вичерпно описують всі необхідні сутності. Цей шар володіє важливим функціоналом, який включає у себе не лише зберігання моделей даних, але й ведення репозиторіїв.

3.4. Шар бізнес-логіки

Шар бізнес-логіки є критичним компонентом, який визначає ключові аспекти обробки даних та взаємодії з базою даних у програмному продукті (рис. 3.7). Цей

шар виконує функцію обробки та взаємодії із данними, що надходять від рівня представлення, і реалізує всю необхідну логіку додатка.

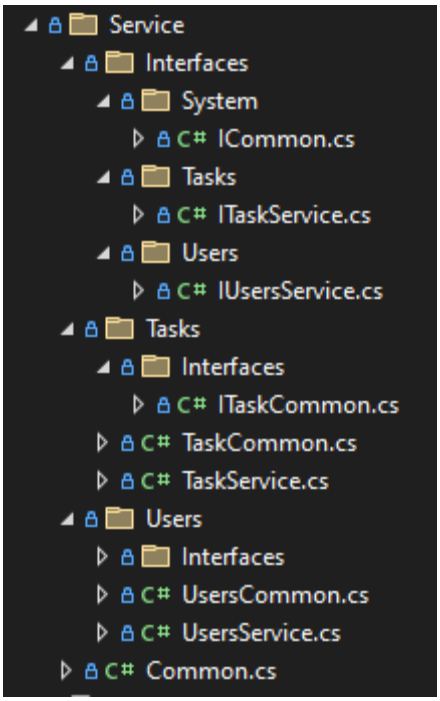

Рис. 3.7. Шар бізнес-логіки

На рівні бізнес-логіки присутні ключові сервіси, такі як «*MapService*», що відповідає за роботу з завданнями, що формуються менеджером. «*MapService*» взаємодіє з «*MapRepository*», обробляє дані, які повертає останній, і заповнює доменну модель «*MapDto*» інформацією, якої не вистачає в моделі бази даних.

«*MapService*» оброблює дані, які вертає «*MapRepository*», заповнює доменну модель «*MapDto*» даними, які не вистачає в моделі БД.

Сервіс «*UserService*» вертає модель користувача, обновлює модель користувача, додає модель користувача.

Додатково, сервіс «*UserService*» управляє моделями користувачів, включаючи їх додавання, оновлення та отримання. Використання таких сервісів на цьому рівні дозволяє використовувати їх як універсальні компоненти для обробки бізнес-логіки (рис. 3.8).

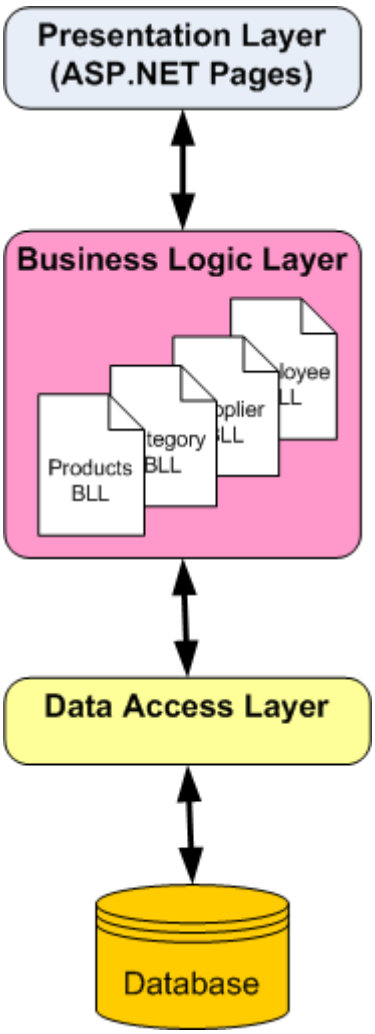

Рис. 3.8. *BLL* відокремлює рівень представлення від рівня доступу до даних і накладає бізнес-правила

Додатково, важливо зазначити, що в шарі бізнес-логіки використовується підхід *DDD* (*Domain-Driven Design*), що дозволяє виокремити ключові бізнес-елементи та їхні взаємодії в межах додатка. Сервіс «*MapService*» ілюструє цей підхід, оскільки він взаємодіє з об'єктами доменної моделі «*MapDto*», вирізняючи їхні ключові атрибути та функціональні можливості.

Важливим компонентом шару бізнес-логіки є також сервіс «*UserService*», який забезпечує роботу з моделями користувачів. Цей сервіс додає, оновлює та повертає моделі користувачів, визначаючи їхні характеристики та поведінку в системі. Використання сервісів на цьому рівні дозволяє використовувати їх як універсальні компоненти для обробки бізнес-логіки та взаємодії з іншими рівнями додатка.

Разом із тим, шар бізнес-логіки служить місцем впровадження бізнес-правил, які накладають обмеження та умови на обробку даних. Це дозволяє забезпечити

консистентність та коректність операцій, здійснюваних у системі, і визначає поведінку різних компонентів додатка відповідно до визначених бізнес-вимог.

Шар бізнес-логіки відіграє ключову роль у забезпеченні високої ефективності та легкості розширення додатка. Використання сервісів, таких як «*MapService*» та «*UserService*», дозволяє ізолювати бізнес-логіку від інших компонентів системи, що сприяє збереженню чистоти коду та його легшій тестовості.

Інтеграція підходу *DDD* в шар бізнес-логіки полегшує розуміння та управління складністю домену, забезпечуючи ясну відокремленість між різними бізнес-концепціями. Це зроблено для того, щоб забезпечити легкість розвитку та зміни системи відповідно до змін у бізнес-вимогах.

Шар бізнес-логіки, керуючи взаємодією з базою даних через репозиторії, впроваджує найкращі практики для забезпечення оптимальної продуктивності та швидкості обробки даних. Використання патернів проектування та оптимальних архітектурних рішень сприяє стабільності та надійності розроблюваного програмного продукту.

Важливим аспектом розгляду є також роль шару бізнес-логіки у введенні та реалізації бізнес-правил. Ці правила стають своєрідними страждущими над цілісністю та правильністю даних, а також визначають, як різні компоненти системи повинні взаємодіяти між собою. Це забезпечує не тільки відділення логічної частини додатка від його представлення та доступу до даних, але й надає централізоване місце для встановлення стандартів та правил обробки інформації.

Інтеграція підходу *Domain-Driven Design* визначає, як об'єкти доменної моделі взаємодіють, і це особливо видно у роботі сервісу «*MapService*». Цей сервіс використовує об'єкти «*MapDto*», які виокремлюють ключові атрибути та функціональність, щоб забезпечити зрозумілу та ефективну роботу з даними.

Важливою перевагою шару бізнес-логіки є його роль у забезпеченні стійкості та надійності додатка. Використання патернів проектування, таких як «репозиторії», сприяє оптимізації продуктивності та забезпеченню швидкої обробки великих обсягів даних.

Шар бізнес-логіки є місцем реалізації стратегій масштабування та

розширення. Завдяки його модульній структурі, можливості додатка можуть легко розширюватися, що робить систему адаптованою до зростання обсягів даних та різноманітних вимог користувачів.

Усі ці аспекти додають значущий внесок у високий рівень функціональності, надійності та ефективності програмного продукту. Шар бізнес-логіки стає не лише структурою для реалізації бізнес-правил, але і стратегічним елементом, що визначає успішність та розвиток додатка в умовах постійних змін та вимог користувачів.

3.5. Шар інтерфейсу користувача

Рівень представлення – реалізований через фреймворк *Blazor* на базі *BlazorServer.* Всі діаграми виконані через фрейморк *Ant.Design.Charts.*

На рис. 3.9 представлено головне вікно програмного продукту, яке служить візуальним інтерфейсом для взаємодії користувача з системою. Головне вікно відображає ключові елементи і функціонал програми, спрощуючи навігацію та взаємодію з основними можливостями продукту.

Шар інтерфейсу користувача взаємодіє з іншими рівнями архітектури, передаючи введені дані та отримуючи відповіді від бізнес-логіки. Це забезпечує високий рівень абстракції для розробників і кінцевих користувачів, спрощуючи процес розробки та полегшуючи використання програмного продукту.

Інтерактивний інтерфейс, побудований на основі взаємодії шару інтерфейсу користувача з іншими рівнями архітектури, впевнено передає введені дані та отримує відповіді від бізнес-логіки. Це забезпечує високий рівень абстракції, що полегшує як розробку, так і використання програмного продукту. Користувачі можуть ефективно взаємодіяти з системою, використовуючи інтуїтивний та зручний інтерфейс, що робить процес користування програмою максимально комфортним та продуктивним.

Фреймворк *Ant.Design.Charts* використовується для створення діаграм та графіків, що додає візуальну ефективність та легкість розуміння складних даних. Цей фреймворк дозволяє ефективно візуалізувати інформацію та надає різноманітні можливості представлення даних, сприяючи зрозумілості та доступності інформації

для користувачів.

Однією з важливих функціональних складових є механізм авторизації, який гарантує безпеку та конфіденційність інформації. Успішна реалізація цього механізму визначається як критичний аспект для забезпечення доступу лише авторизованим користувачам. Використання фреймворка *Blazor Server* дозволяє зберігати чутливі дані на сервері, зменшуючи ризик витоку інформації, а також надає можливість легко впроваджувати засоби шифрування та інші заходи безпеки.

Застосування механізму авторизації стає гарантом того, що лише користувачі з належними правами можуть отримувати доступ до важливих функціональних можливостей, зберігаючи цілісність та безпеку оброблюваних даних.

Функція фільтрації даних виявилася не лише функціональною, але й ефективною, дозволяючи користувачам вибирати конкретні дані для подальшого аналізу. Це сприяє зручності в роботі з програмою, оскільки користувачі можуть швидко та точно визначати параметри для відображення інформації, що робить процес аналізу більш гнучким та ефективним.

Оцінка діаграм та графіків підтверджує їхню спроможність точно та інтуїтивно відображати складні дані. Використання фреймворку *Ant.Design.Charts* додає естетичний аспект до візуалізації даних, роблячи її більш зрозумілою та привабливою для користувачів.

Брифінг та мапа відзначаються не лише надійністю роботи, але й можливістю надати користувачам зрозумілу та достовірну інформацію щодо маршрутів та географічних елементів. Можливість взаємодії з літаком на мапі додає інтерактивний аспект, що підвищує зручність використання та робить процес спостереження за польотами більш захоплюючим для користувачів.

Отримані результати свідчать про високий рівень функціональності та відповідність програмного продукту визначеним вимогам, а детальний аналіз цих аспектів служить основою для подальшого вдосконалення та розвитку системи для забезпечення найвищої якості обслуговування та задоволення потреб користувачів.

Рис. 3.9. Головне вікно програми

Висновки за розділом

Для вирішення поставленої задачі було розроблено програмний продукт для візуалізації та аналізу даних з рейсових записів польотів.

Також була представлена загальна структура програмного продукту, яка складається з декількох модулів, такі як:

– модуль для візуалізації;

– модуль серверної частини та бази даних;

– модуль контролю нових рейсів, які приходять від стороніх сервісів;

– модуль для роботи с користувачами.

Досягнуто збільшення зручності завдяки мінімізації і реалізації тільки необхідних інструментів, такий підхід полегшує розуміння програмного продукту з боку користувача та надає змогу працювати тільки з тим функціоналом, який потребує користувач.

Крім того, важливим аспектом розробленого програмного продукту є його висока масштабованість та гнучкість. Модульна структура дозволяє легко розширювати функціональність та додавати нові можливості, що робить систему адаптованою до зростання обсягу даних та змінних потреб користувачів.

Напрямок подальших вдосконалень полягає у постійному моніторингу відгуків користувачів та врахуванні їхніх потреб для розширення функціональності. Такий підхід сприяє вдосконаленню якості програмного продукту та забезпечує високий рівень задоволення користувачів.

Надалі, планується проведення додаткових тестувань та оптимізацій для забезпечення високої швидкодії та надійності системи. Також у планах розгляд впровадження нових технологій та підходів, що можуть підвищити продуктивність та розширити можливості веб-додатку.

Завершення розробки та успішна імплементація цього програмного продукту є важливим етапом у вдосконаленні системи візуалізації та аналізу даних з рейсових записів польотів. Отримані результати свідчать про високий потенціал застосування розробленого інструменту в сучасному авіаційному секторі, а також створюють основу для подальших досліджень та удосконалень в області візуалізації та аналізу авіаційних даних.

Під час розробки програмного продукту для візуалізації та аналізу даних з рейсових записів польотів, ми акцентували увагу на ключових аспектах, що визначають його високий функціонал та ефективність.

Щодо візуального представлення інформації, наш продукт використовує фреймворк *Blazor* на базі *BlazorServer* та *Ant.Design.Charts* для реалізації діаграм та графіків. Головне вікно програми було створено з урахуванням простоти навігації та легкої взаємодії з основним функціоналом. Шар інтерфейсу користувача взаємодіє з іншими рівнями архітектури, що забезпечує високий рівень абстракції для розробників та кінцевих користувачів.

Авторизація, як один із важливих елементів системи, гарантує безпеку та конфіденційність інформації, а функція фільтрації даних дозволяє користувачам вибирати конкретні дані для подальшого аналізу, сприяючи зручності в роботі з

програмою.

Результати оцінки діаграм та графіків підтверджують їхню точність та інтуїтивність, надаючи користувачам засіб для визначення та аналізу залежностей. Робота з мапою та можливість взаємодії з літаком, представлені у програмі, ретельно протестовані, забезпечуючи користувачам зрозумілу та достовірну інформацію щодо маршрутів та географічних елементів.

Підвищення зручності користування та надання тільки суттєвого функціоналу, враховуючи потреби кожного користувача, визначають високий ступінь адаптованості продукту. Ми залишаємося відкритими до відгуків користувачів та готові до подальших вдосконалень для досягнення найвищого рівня задоволення від використання нашої системи.

Завершення розробки цього програмного продукту створює основу для впровадження його в сучасних умовах авіаційного сектору. Наш продукт вже сьогодні відповідає високим вимогам та надає рішення для важливих завдань візуалізації та аналізу даних у сфері авіації, але ми прагнемо йти далі. Подальші плани включають тестування для забезпечення стабільної роботи та оптимізації, а також впровадження новітніх технологій для збільшення продуктивності та функціональних можливостей веб-додатку.

РОЗДІЛ 4
РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

4.1. Системні вимоги та інсталяція

Для запуску програмного продукту потрібно відвідати веб-сайт і пройти авторизацію. Мінімальні системні вимоги для використання телеграму на платформі Android – версія Android 4.0. Для Windows Phone – мінімальна версія 7.1, а для iPhone – IOS 6.

Запуск на комп'ютері або ноутбуці вимагає наявності процесора Intel Pentium 1.4 GHz і обсягу оперативної пам'яті не менше 1 Гб.

Щодо запуску у браузері, рекомендується використовувати сучасні браузери, такі як Chrome, Firefox, Safari або Edge, які підтримують WebAssembly. Деякі вимоги включають оновлену версію браузера, налаштування для активації WebAssembly та JavaScript, включену підтримку JavaScript, ввімкнені Cookies для правильної роботи додатків, підтримку Local Storage для локального зберігання даних та можливість роботи через протоколи HTTP/HTTPS, і хоча додатки можуть працювати через HTTP, рекомендується використовувати HTTPS для забезпечення безпеки.

4.2. Сценарії роботи з програмним продуктом

Запуск програмного додатку виконується через сторінку авторизації (рис. 4.1).

Далі потрібно авторизуватися. Ввести свій логін і пароль, який було вказано при регістрації. Якщо користувач в базі данних має роль менеджер, то він авторизується як менеджер, і буде мати вповноваження заходити на сторінку налаштувань додатку і змінювати певні налаштування, якщо авторизувався звичайний користувач, і буде бачити для себе звичайні сторінки з візуалізацію даних.

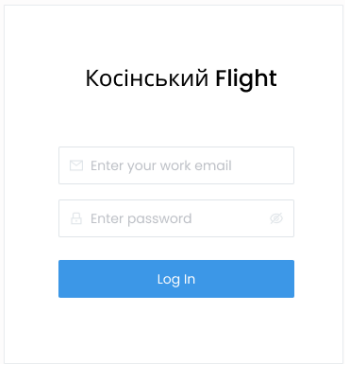

Рис. 4.1. Сторінка входу в додаток

Після того, як користувач авторизувався в систему, він потрапляє на головну сторінку додатку (рис. 4.2) на ній детально візуалізовано данні рейсових записів польотів у вигляді діаграм. Детально по кожній діаграмі буде описано нижче.

Рис. 4.2. Головне вікно програми

Перш ніж почати роботу з діаграмами, потрібно налаштувати фільтр по даті, рейсу, аеропорту, літаку, часу, бортовому номеру, номеру квитка (рис. 4.3).

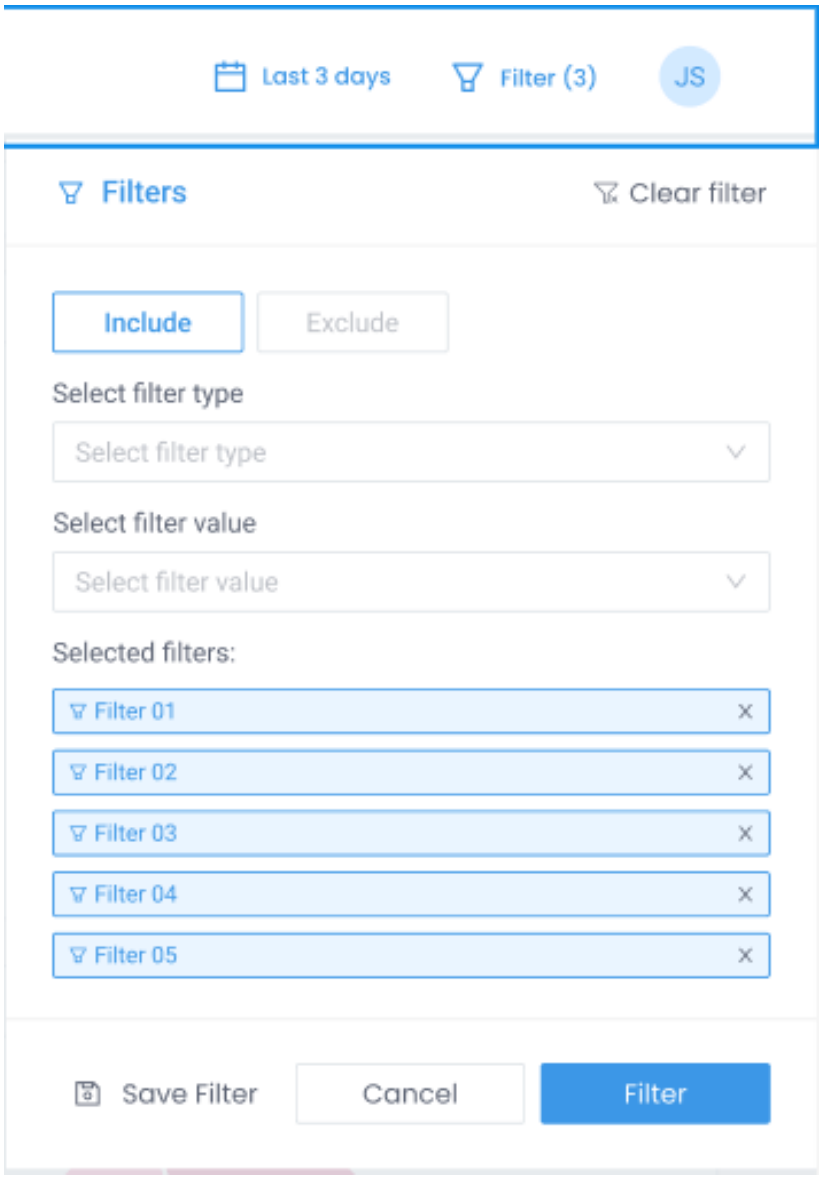

Рис. 4.3. Фільтри

У фільтра є два режима роботи: 1) *Include*, 2) *Exclude* (рис. 4.4), тобто включає і не включає. Можна вибрати фільтр, яий буде включати або виключати той чи інший аеропорт, літак і так далі. Фільтри можна зберігати, оновлювати, видаляти, копіювати. Все це зроблено для зручності користування і фільтрації інформації.

Рис. 4.4. *Include/Exclude*

Зараз детально буде розібрана головна сторінка додатку. На рисунку 4.5 зображено перша візуально представлена інформація про рейсові записи польтовів у вигляді статистики по кількості активних аеропортів в даному фільтрі, який користувач зазначив до цього, кількість літаків у польторі, кількість запланованих рейсів і кількість скасованих рейсів.

Рис. 4.5. Статистика на головній сторінці

На данній діаграмі (рис. 4.6) візуально зображено графік, для певного фільтру, який зазначив користувач до цього, на якому відображаються данні про рейси на проміжок часу. Затримані рейси, рейси в польоті, рейси, які вже виконані.

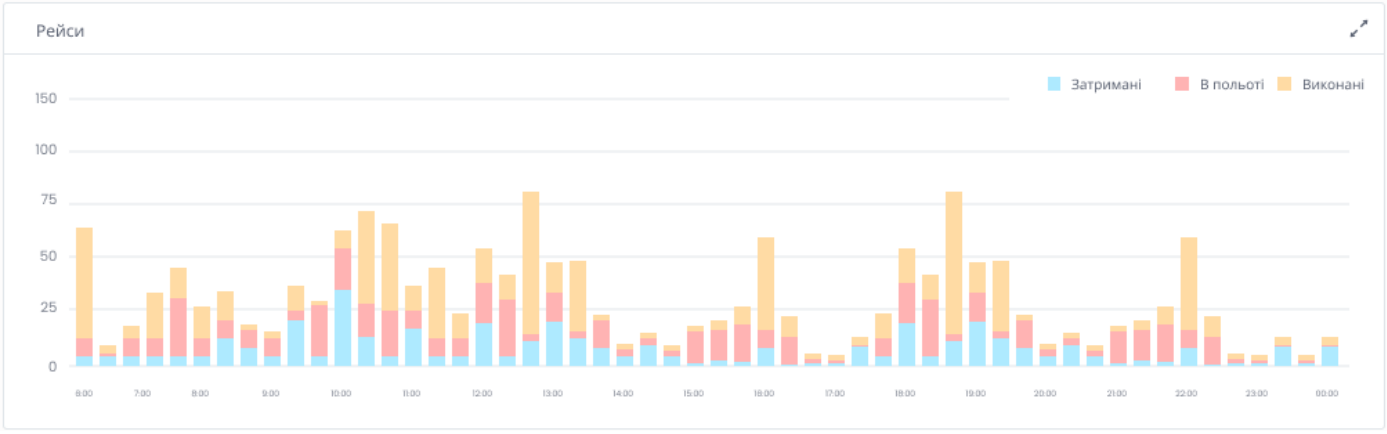

Рис. 4.6. Діаграма рейсів

На данній діаграмі (рис. 4.7) візуально зображено графік, для певного фільтру, який зазначив користувач до цього, на якому відображаються данні про топ аеропортів по завантаженості на проміжок часу.

У представленій діаграмі (рис. 4.8) наочно відтворено графік, побудований для конкретного фільтру, визначеного користувачем на попередніх етапах. Цей графік висвітлює інформацію про провідні аеропорти за показниками затримок протягом визначеного періоду часу.

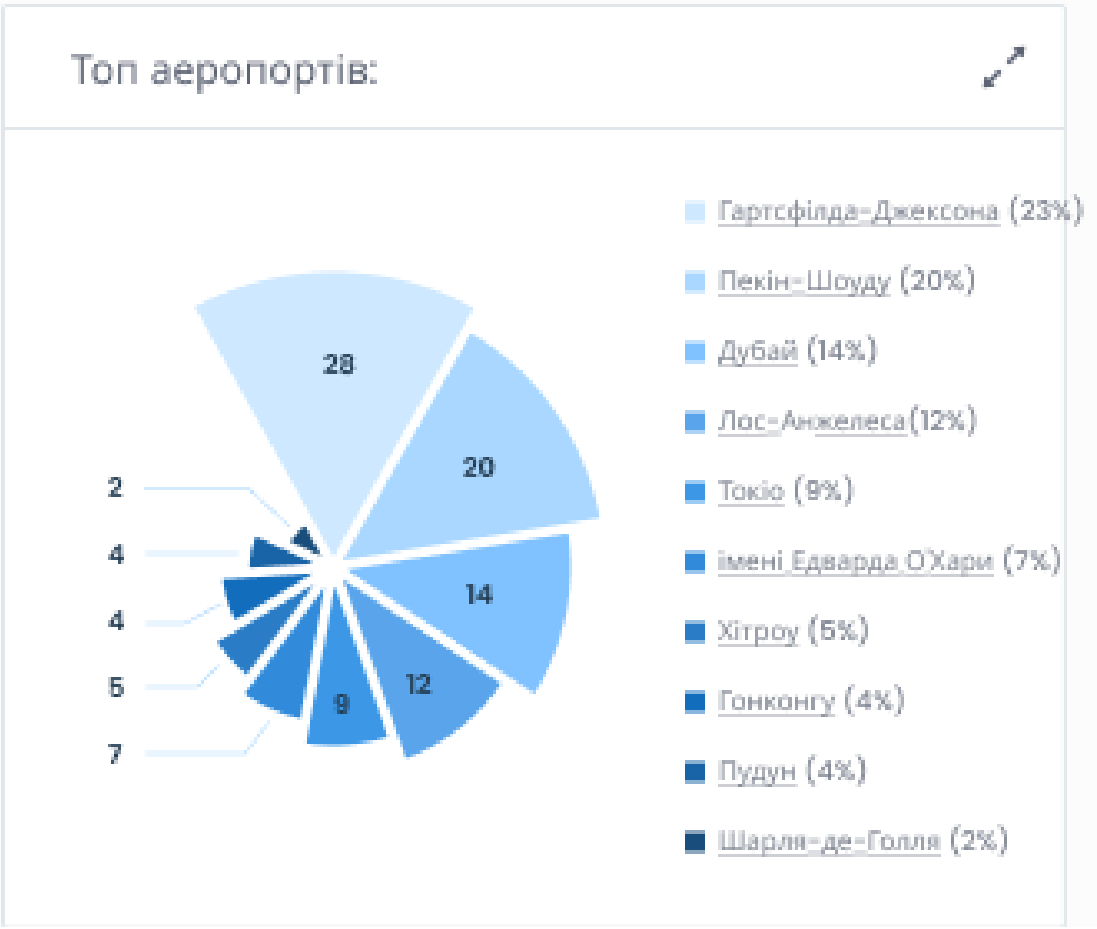

Рис. 4.7. Діаграма топ аеропортів

Рис. 4.8. Діаграма топ аеропортів по затримках

На даній діаграмі (рис. 4.9) візуально зображено графік, для певного фільтру, який зазначив користувач до цього, на якому відображаються данні про статистику рейсів на проміжок часу.

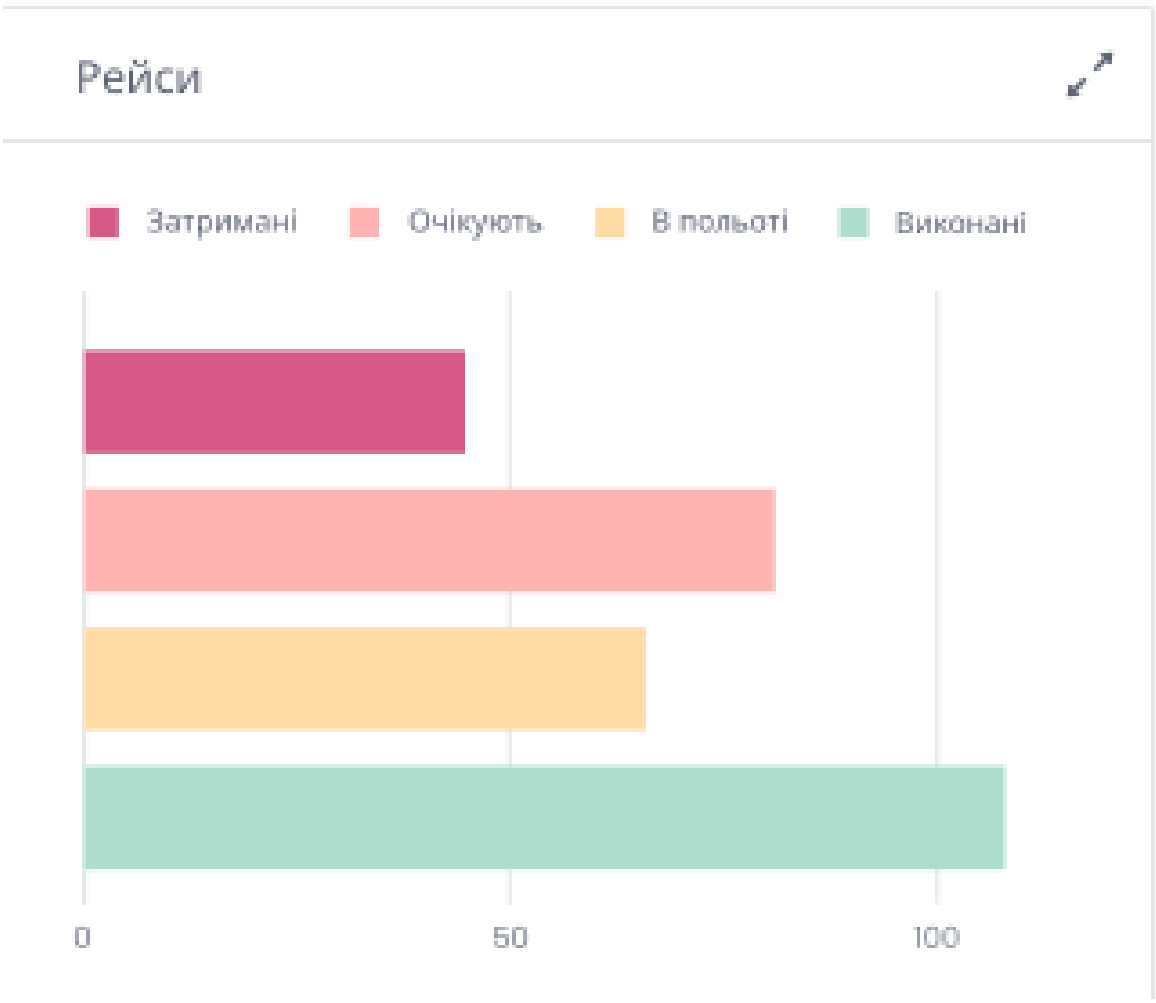

Рис. 4.9. Діаграма статистики рейсів

На даній діаграмі (рис. 4.10) візуально зображено графік, для певного фільтру, який зазначив користувач до цього, на якому відображаються статистика авіакомпаній по кількості перевезень на проміжок часу.

На рис. 4.11 надано компактний огляд останніх ключових подій в авіандустрії, який стане надзвичайно корисним для авіалюбителів та всіх, хто цікавиться розвитком авіаційних технологій. Цей візуальний брифінг слугує як важливий інформаційний ресурс, який дозволяє швидко та ефективно освіжити або розширити знання про недавні події у світі авіації.

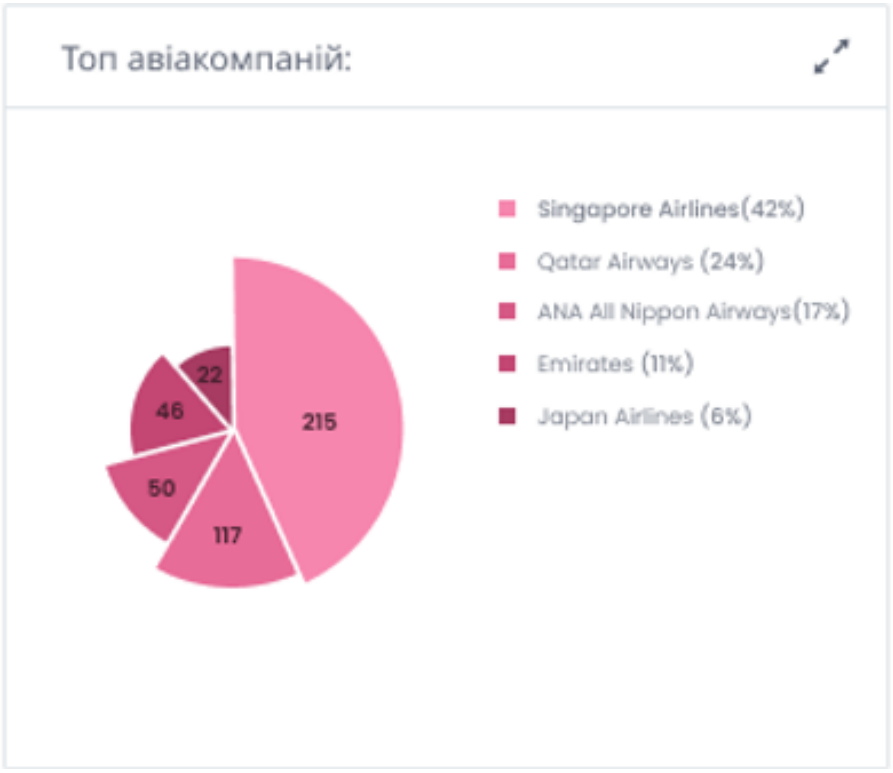

Рис. 4.10. Діаграма статистики авіакомпаній по кількості перевезень

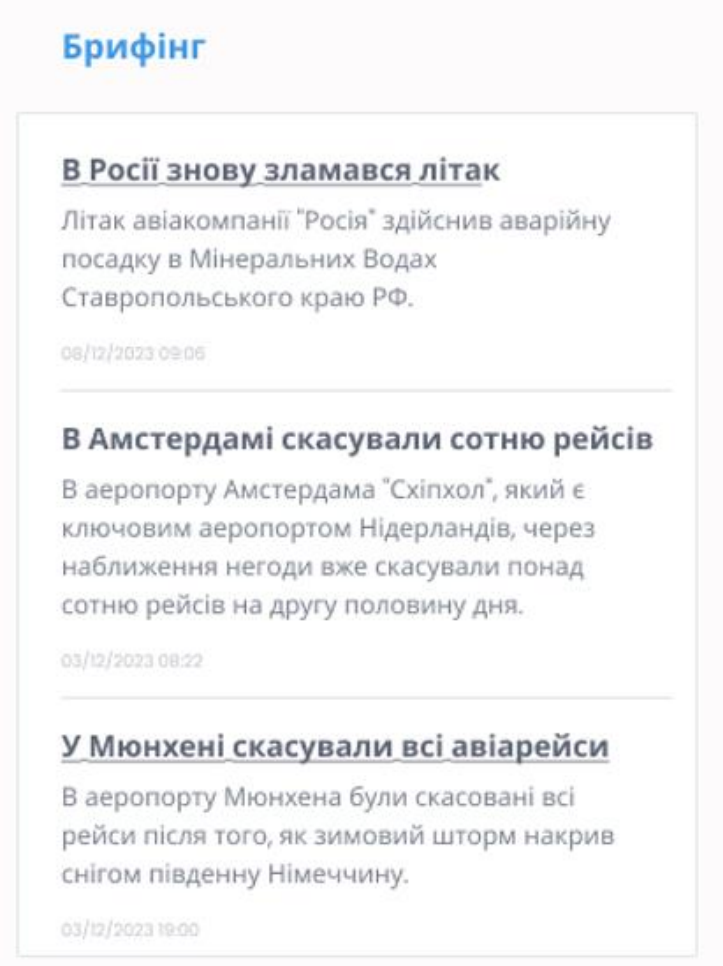

Рис. 4.11. Короткий брифінг подій

Перейдемо на іншу вкладку програмного продукту під назвою «*GeoMap*» (рис. 4.12). На цій сторінці візуально і інтерактивно відображається інформація про рух повітряного транспорту.

Основний функціонал сторінки:

– карта з відображенням літаків та аеропортів: літаки, які були в повітрі і аеропорти, які приймали ці літаки;

– інтерактивні елементи: взаємодія з мапою, зумуючи або рухаючи, щоб отримати більше деталей про конкретні області чи літаки;

– інформація про літаки (рис. 4.13): при кліку на конкретний літак можна переглянути деталі, такі як номер рейсу, модель літака, швидкість, висота, напрямок руху і інша важлива інформація;

– фільтри та пошук: сервіс дозволяє фільтрувати літаки за різними критеріями, такими як авіакомпанія, тип літака, аеропорт вильоту чи призначення;

– інформація про аеропорти;

– можливість вибору різних шарів на карті.

Висновки за розділом

Цей розділ становить критичну складову в оцінці програмного продукту, спрямованого на систему візуалізації та аналізу даних з рейсових записів польотів. Зазначені ключові аспекти роботи програми виявили величезний потенціал, охоплюючи широкий спектр функціональних можливостей для забезпечення повноти та точності обробки інформації.

Одним із критичних елементів є механізм авторизації, який ефективно гарантує безпеку та конфіденційність інформації. Успішна реалізація цього механізму визначається як важливий крок у забезпеченні доступу лише авторизованим користувачам, а це, в свою чергу, формує основу для надійності системи.

Функція фільтрації даних не лише виявилася функціональною, але й вкрай ефективною. Користувачі мають можливість вибирати конкретні дані для

подальшого аналізу, що відзначається високою зручністю в роботі з програмою та підвищує загальний рівень її використання.

Оцінка діаграм та графіків підтверджує їхню спроможність точно та інтуїтивно відображати складні дані. Це надає користувачам потужний інструмент для визначення та аналізу залежностей між різними параметрами, сприяючи більш глибокому розумінню авіаційних показників.

Функціонал брифінгу та взаємодія з мапою були об'єктом ретельного тестування, що призвело до надання користувачам зрозумілої та достовірної інформації про маршрути та географічні елементи. Зокрема, можливість взаємодії з літаком на мапі дозволяє користувачам ефективно виконувати різноманітні функції та отримувати додаткові дані, покращуючи взаємодію з програмою.

Отримані результати, які детально охоплюють авторизацію, фільтрацію даних, діаграми, графіки, брифінг та мапу, підтверджують високий рівень функціональності та відповідність програмного продукту визначеним вимогам. Цей детальний аналіз слугує основою для подальшого удосконалення та розвитку системи, щоб забезпечити найвищу якість обслуговування та задоволення потреб користувачів.

Оцінка надійності продукту визначила його стійкість та стабільність при обробці великої кількості даних. Навіть при значному обсязі інформації, програмний продукт демонструє ефективність та швидкодію обробки даних, що є важливим параметром для забезпечення безперебійної роботи та комфортного користування.

Інтерактивний інтерфейс виявився додатковим сильним боком програми. Надання користувачам інструментів для взаємодії з даними та виконання різноманітних запитів, сприяє відмінній користувацькій зручності. Зокрема, можливість фільтрації, сортування та пошуку даних, а також інтерактивні можливості маніпуляції інформацією роблять продукт гнучким та відповідним різним потребам користувачів.

На підставі результатів дослідження можна визначити напрями для подальшого вдосконалення системи. Розширення функціональності може включати в себе аналіз даних з інших джерел, таких як метеорологічні дані та дані про стан

повітряного простору, для більш повного уявлення про фактори, що впливають на авіаційну діяльність.

Подальше вдосконалення інтерфейсу користувача може збільшити його інтуїтивність та зручність для різних категорій користувачів. Забезпечення безпеки даних є актуальним завданням, і використання сучасних технологій у цьому плані є критичним.

Додаткові можливості, такі як розробка мобільної версії системи, інтеграція з іншими системами та розробка нових методів аналізу даних, можуть розширити функціональність системи та розширити її застосування.

Основні можливості програми, які були перевірені:

– авторизація;
– фільтрація даних;
– перевірка роботи діаграм і графіків;
– перевірка роботи брифінгу;
– перевірка роботи мапи;
– перевірка роботи кліка на літак на мапі.

На основі вивчення вимог та функціональності системи визначено ключові характеристики, що роблять програмний продукт ефективним та надійним інструментом для візуалізації та аналізу даних з рейсових записів польотів. Розроблена система не лише відповідає поставленим завданням, але й виявляє великий потенціал для розвитку та удосконалення. Подальша реалізація рекомендацій та перспективних напрямків розвитку може зробити її ще більш потужним та конкурентоспроможним інструментом для авіакомпаній та авіалюбителів.

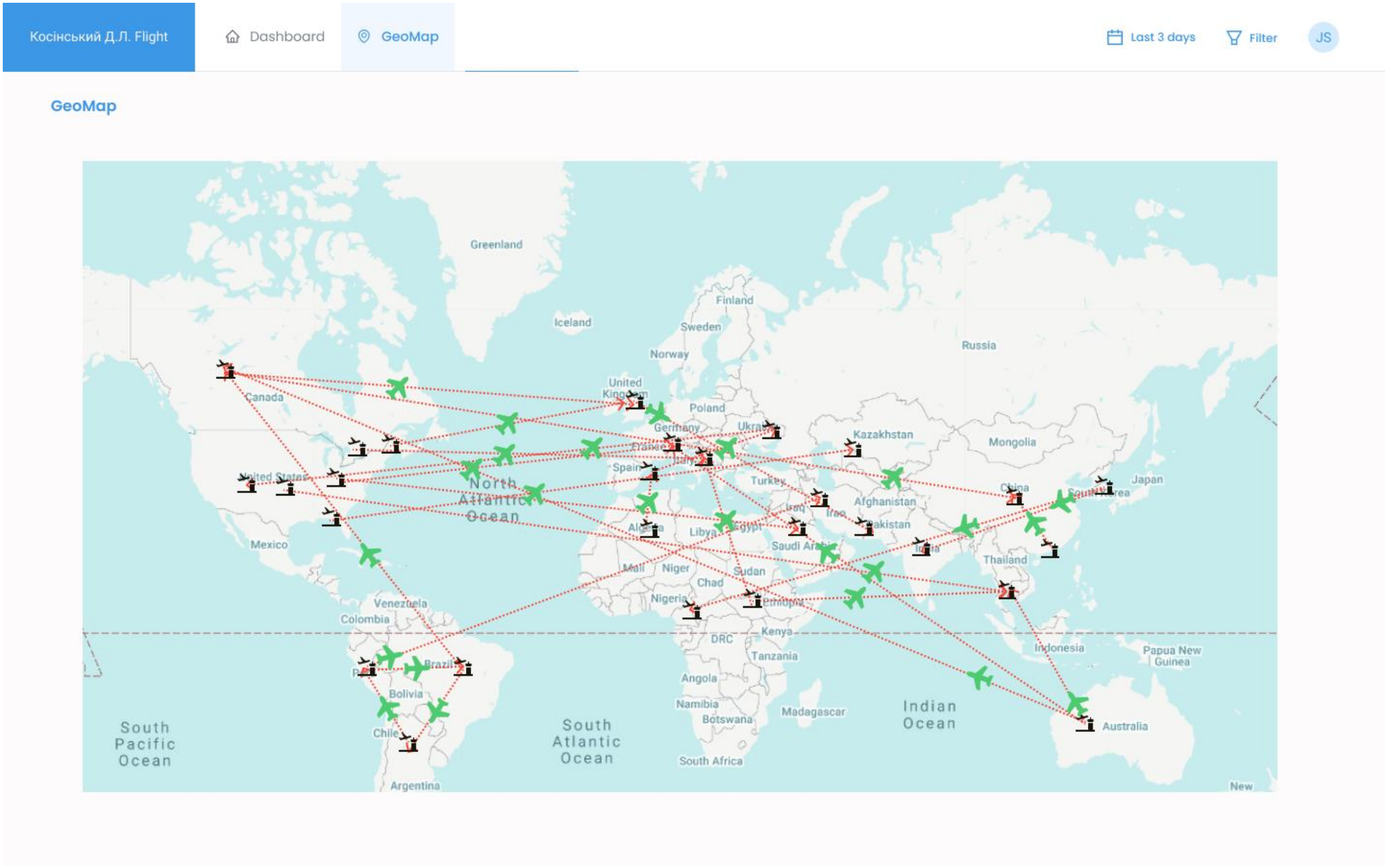

Рис. 4.12. Сторінка «*GeoMap*»

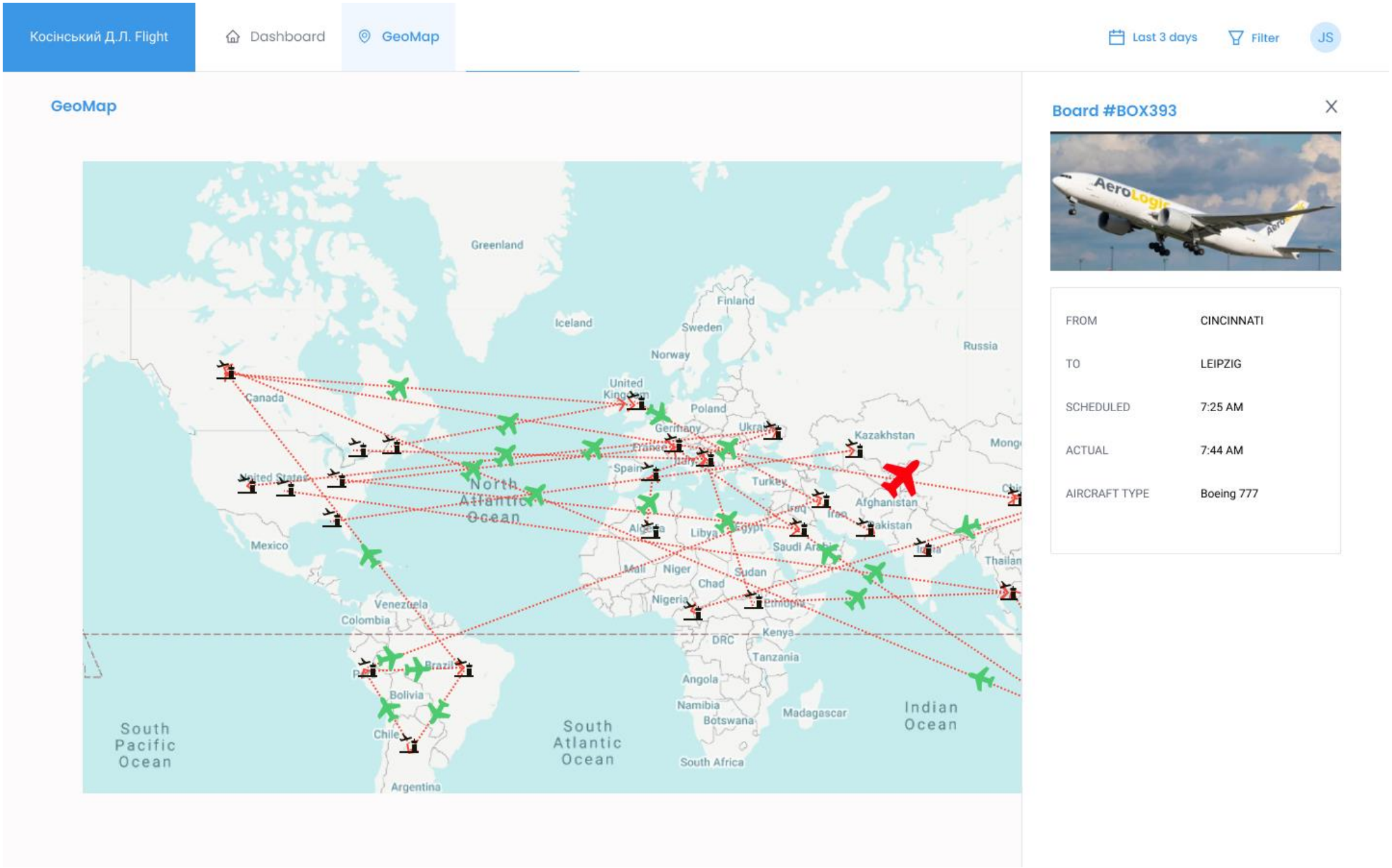

Рис. 4.13. Деталі про літак

ВИСНОВКИ

У даній магістерській роботі було розглянуто процес визначення основних вимог та функціональності системи візуалізації та аналізу даних з рейсових записів польотів.

Розвиток авіації та зростання обсягів даних з рейсових записів створюють нові можливості для візуалізації та аналізу цих даних. Розроблена в рамках магістерської роботи система візуалізації та аналізу даних з рейсових записів польотів дозволяє задовольнити потреби користувачів у наступних областях:

– аналіз тенденцій та прогнозування: система може використовуватися для виявлення тенденцій у даних рейсових записів, таких як зміни в частоті польотів, тривалості рейсів, затримках тощо. Ці дані можуть використовуватися для прогнозування майбутньої діяльності авіакомпаній;

– аналіз безпеки: система може використовуватися для виявлення аномалій у даних рейсових записів, які можуть вказувати на потенційні проблеми з безпекою. Ці дані можуть використовуватися для прийняття заходів для поліпшення безпеки польотів;

– управління ресурсами: система може використовуватися для управління ресурсами авіакомпаній, такими як літаками, екіпажами та пасажирськими перевезеннями. Ці дані можуть використовуватися для підвищення ефективності роботи авіакомпаній.

У результаті аналізу вимог і функціональності були визначені ключові характеристики та можливості системи:

Функціональність:

– зчитування та імпорт даних з рейсових записів польотів у систему;

– обробка та аналіз даних з метою виявлення тенденцій, паттернів та аномалій;

– візуалізація даних у вигляді графіків, діаграм, карт і інших візуальних елементів;

– надання інтерактивного інтерфейсу користувачу для взаємодії з даними та

виконання різних запитів;

– забезпечення можливості фільтрації, сортування та пошуку даних з рейсових записів;

– забезпечення зручного відображення деталей польотів та пов'язаних параметрів.

Вимоги:

– ефективність та швидкодію обробки даних навіть при великому обсязі інформації;

– гнучкість та можливість розширення системи для врахування майбутніх потреб;

– надійність та безпеку зберігання та обробки конфіденційних даних;

– зручний та інтуїтивно зрозумілий інтерфейс для користувачів з різним рівнем технічної підготовки;

– сумісність з різними платформами та пристроями для забезпечення доступності з будь-якого пристрою з Інтернет-підключенням.

Зазначені характеристики та можливості дозволять системі задовольнити потреби користувачів у наступних областях:

– аналіз тенденцій та прогнозування: система може використовуватися для виявлення тенденцій у даних рейсових записів, таких як зміни в частоті польотів, тривалості рейсів, затримках тощо. Ці дані можуть використовуватися для прогнозування майбутньої діяльності авіакомпаній;

– аналіз безпеки: система може використовуватися для виявлення аномалій у даних рейсових записів, які можуть вказувати на потенційні проблеми з безпекою. Ці дані можуть використовуватися для прийняття заходів для поліпшення безпеки польотів;

– управління ресурсами: система може використовуватися для управління ресурсами авіакомпаній, такими як літаками, екіпажами та пасажирськими перевезеннями. Ці дані можуть використовуватися для підвищення ефективності роботи авіакомпаній.

На основі проведеного дослідження можна зробити наступні рекомендації щодо вдосконалення системи:

– розширити функціональність системи: додати можливість аналізу даних з інших джерел, таких як метеорологічні дані, дані про стан повітряного простору тощо. Це дозволить отримати більш повне уявлення про фактори, що впливають на діяльність авіакомпаній;

– покращити інтерфейс користувача: зробити інтерфейс більш інтуїтивно зрозумілим та зручним для використання користувачами з різним рівнем технічної підготовки;

– забезпечити безпеку даних: використовувати сучасні технології для захисту даних від несанкціонованого доступу, модифікації або знищення.

Реалізація цих рекомендацій дозволить зробити систему більш корисною та ефективною для користувачів.

Крім зазначених рекомендацій, доцільно також розглянути такі можливості:

– мобільна версія системи: це дозволить користувачам отримувати доступ до системи з будь-якого пристрою з Інтернет-підключенням;

– інтеграція з іншими системами: це дозволить обмінюватися даними з іншими системами, такими як системи бронювання, системи управління ресурсами тощо;

– реалізація розширеної системи оповіщення: включення розширених систем сповіщення для користувачів про зміни у рейсових записах, важливі події чи новини в авіаційній галузі. Це дозволить користувачам бути завжди в курсі та оперативно реагувати на зміни;

– розробка модулів аналізу погоди: додавання можливостей аналізу погодних умов та їх впливу на авіаперельоти допоможе створити більш повне та інформативне зображення факторів, що впливають на авіаційну безпеку та ефективність;

– енергоефективність: вдосконалення системи з точки зору енергоефективності та оптимізації використання ресурсів дозволить зменшити витрати та забезпечити стабільну роботу системи навіть при інтенсивному

завантаженні;

– глобальна інтеграція: розгляд можливостей глобальної інтеграції системи з іншими аналогічними платформами та сервісами, що дозволить створити єдиний інформаційний простір для спільноти авіаційних експертів та зацікавлених сторін;

– оптимізація продуктивності: забезпечення високої продуктивності системи є критичним аспектом, особливо при роботі з великим обсягом даних. Оптимізація алгоритмів обробки та вдосконалення методів зберігання може значно покращити швидкість реакції системи та сприяти зручності користування;

– розширення аналітичних можливостей: включення нових методів аналізу даних, таких як машинне навчання чи статистичні моделі, може збагатити функціональність системи та допомогти в отриманні більш глибоких та точних висновків з рейсових записів;

– вдосконалення системи безпеки: в контексті обробки конфіденційних даних, важливо посилити заходи безпеки. Використання шифрування та інших сучасних методів захисту даних забезпечить конфіденційність та запобігання несанкціонованому доступу;

– співпраця зі спільнотою користувачів: залучення користувачів до процесу розвитку системи, врахування їхніх пропозицій та фідбеку, може сприяти точному визначенню потреб аудиторії та поліпшенню користувацького досвіду;

– інтеграція з розширеними джерелами даних: розгляд можливостей інтеграції системи з іншими джерелами даних, такими як соціальні мережі, геолокаційні сервіси чи інші технології, може розширити спектр інформації та покращити здатність системи до глибшого аналізу;

– розробка нових методів аналізу даних: це дозволить отримувати більш глибокі висновки з даних.

Узагальнюючи вищезазначене, подальший розвиток системи має ґрунтуватися на збільшенні її ефективності, розширенні аналітичних можливостей та поліпшенні заходів безпеки. Співпраця з користувачами та інтеграція з новими джерелами даних стануть ключовими факторами для забезпечення системи стати ще більш потужним та відповідати вимогам сучасного авіаційного середовища.

Розробка та реалізація цих можливостей дозволить зробити систему ще більш потужним і корисним інструментом для авіакомпаній та авіалюбителів. За результатами аналізу вимог і функціональності системи визначено ключові характеристики та можливості, що дозволяють забезпечити повне задоволення потреб користувачів та досягти поставленої мети магістерської роботи. Розроблена система, враховуючи визначені параметри, володіє великим потенціалом для успішної реалізації завдань візуалізації та аналізу даних з рейсових записів польотів, забезпечуючи при цьому високу ефективність та надійність функціонування.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Положення про дипломні роботи (проекти) випускників Національного Авіаційного Університету. Київ: НАУ, 2017.
2. ДСТУ 3008-95 Документація. Звіти у сфері науки і техніки. Структура і правила оформлення / Видавничий дім «Держстандарт України», 1995. – 88с
3. *.NET Framework Guide* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://docs.microsoft.com/en-us/dotnet/framework/* (дата звернення 23.11.2023). – Назва з екрану.
4. *ASP.NET Core Blazor* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://learn.microsoft.com/ru-ru/aspnet/core/blazor/?view=aspnetcore-8.0* (дата звернення 24.11.2023). – Назва з екрану.
5. *Microsoft Azure* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://azure.microsoft.com/en-us/* (дата звернення 25.11.2023). – Назва з екрану.
6. *Microsoft Azure SQL Server* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://learn.microsoft.com/uk-ua/azure/azure-sql/azure-sql-iaas-vs-paas-what-is-overview* (дата звернення 26.11.2023). – Назва з екрану.
7. *Microsoft Azure Service Bus* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://learn.microsoft.com/uk-ua/azure/service-bus-messaging/service-bus-messaging-overview* (дата звернення 27.11.2023). – Назва з екрану.
8. *Microsoft Azure Web App* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://learn.microsoft.com/uk-ua/azure/app-service/overview* (дата звернення 28.11.2023). – Назва з екрану.
9. *CI/CD baseline architecture with Azure Pipelines* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://learn.microsoft.com/uk-ua/azure/devops/pipelines/architectures/devops-pipelines-baseline-architecture* (дата звернення 29.11.2023). – Назва з екрану.
10. *Elastic Documentaion* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://www.elastic.co/guide/index.html* (дата звернення 30.11.2023). – Назва з екрану.

11. *Common Language Runtime (CLR) overview* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://docs.microsoft.com/en-us/dotnet/standard/clr* (дата звернення 01.12.2023). – Назва з екрану.

12. Ріхтер Д. *CLR* via *C#* Програмування на платформі *Microsoft .NET Framework 2.0* на мові *C#* / Джеффрі Ріхтер. – Київ, 2008. – 656 с. – (2).

13. Троелсен Э. Мова програмування *C#* і платформа *.NET 4.5* / Єндрю Троелсен. – Київ: вільямс, 2014. – 1312 с. – (6).

14. *Visual Studio* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://visualstudio.microsoft.com/ua/* (дата звернення 02.12.2023). – Назва з екрану.

15. *Entity Framework Documentation* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://docs.microsoft.com/en-us/ef/* (дата звернення 05.12.2023). – Назва з екрану.

16. *SQL Server 2017* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://www.microsoft.com/ru-ru/sql-server/sql-server-2017* (дата звернення 07.12.2023). – Назва з екрану.

17. *Understanding Onion Architecture* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://www.codeguru.com/csharp/csharp/cs_misc/designtechniques/understanding- onion-architecture.html* (дата звернення 10.12.2023). – Назва з екрану.

18. Концептуальна модель систем [Електронний ресурс] / Електрон. дані – Режим доступу: *http://studepedia.org/index.php?vol=1&post=2123* (дата звернення 11.12.2023). – Назва з екрану.

19. *MVC, MVP and MVVM Design Pattern* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://medium.com/@ankit.sinhal/mvc-mvp-and-mvvm- design-pattern-6e169567bbad* (дата звернення 15.12.2023). – Назва з екрану.

20. *UnitOfWork And Repository Pattern* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://medium.com/@utterbbq/c-unitofwork-and-repository-pattern- 305cd8ecfa7a* (дата звернення 15.12.2023). – Назва з екрану.

21. *Dependency Injection* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection*

(дата звернення 15.12.2023). – Назва з екрану.

22. Особливості розвитку авіаційної галузі на міжнародному та національному рівні в умовах глобалізації [Електронний ресурс] / Електрон. дані – Режим доступу: *https://ev.nmu.org.ua/docs/2017/4/EV20174_092-099.pdf* (дата звернення 15.12.2023). – Назва з екрану.

23. *Heuristics for tactical time slot management: a periodic vehicle routing problem view* [Електронний ресурс] / Електрон. дані – Режим доступу: *https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12403* (дата звернення 15.12.2023). – Назва з екрану.

ДОДАТКИ

Додаток А

Лістинг файлу MessageController.cs

```
using System.Linq;


namespace TeamContactTelegramBot.CoreSystem
{
   class StartUp
   {
      static ITelegramBotClient bot = new
TelegramBotClient("5301791346:AAHYNy6OLsS50qnLVcQFbET61zZ99BHge-Q");
      static MessageController messController = new MessageController();
      public static async Task HandleUpdateAsync(ITelegramBotClient botClient, Update
update, CancellationToken cancellationToken)
      {
Console.WriteLine(Newtonsoft.Json.JsonConvert.SerializeObject(update.Message));
         if (update.Type == UpdateType.Message)
         {
            var message = update.Message;
            var procces = await messController.ProcessText(new ApiStringRequest {
MessageRequest = message, botClient = botClient });
            if (!string.IsNullOrEmpty(procces.Message))
               await botClient.SendTextMessageAsync(message.Chat, procces.Message);


            // ---------------------------
            if (procces.Role != 0)
            {
```

```
            ReplyKeyboardMarkup keyboard = new(new[]
            {
                new KeyboardButton[] {
                    procces.Role == 1 ? "Створити задачу" : procces.Role == 2 ? "Активні задачі" : "",
                    procces.Role == 1 ? "Закрити задачу" : procces.Role == 2 ? "Заплановані зустрічі" : "Заплановані зустрічі" },
                new KeyboardButton[] {
                    procces.Role == 1 ? "Створити віртуальну зустріч" : procces.Role == 2 ? "Поставити статус Need Testing" : "Задачі, які потребують тестуванню",
                    procces.Role == 1 ? "Запланувати віртуальну зустріч" : procces.Role == 2 ? "" : "" },
                new KeyboardButton[] {
                    procces.Role == 1 ? "Подивитися всі задачі" : procces.Role == 2 ? "" : "",
                    procces.Role == 1 ? "Додати нового співробітника" : procces.Role == 2 ? "" : "" }
            })
            {
                ResizeKeyboard = true
            };
            await botClient.SendTextMessageAsync(message.Chat.Id, "Вибір:", replyMarkup: keyboard);
            return;
        }
        // ---------------------------

        return;
    }

    if (update.Type == UpdateType.CallbackQuery)
```

```
        {
            //var message = update.Message;
            //var procces = await messController.ProcessText(new ApiStringRequest { StringRequest = message.Text });
            //await botClient.SendTextMessageAsync(message.Chat, procces);
            //return;
        }
    }

    public static async Task HandleErrorAsync(ITelegramBotClient botClient, Exception exception, CancellationToken cancellationToken)
    {
        Console.WriteLine(Newtonsoft.Json.JsonConvert.SerializeObject(exception));
    }

    static void Main(string[] args)
    {

        Console.WriteLine("Запущен бот " + bot.GetMeAsync().Result.FirstName);

        var cts = new CancellationTokenSource();
        var cancellationToken = cts.Token;
        var receiverOptions = new ReceiverOptions
        {
            AllowedUpdates = { }, // receive all update types
        };
        bot.StartReceiving(
            HandleUpdateAsync,
            HandleErrorAsync,
```

```
        receiverOptions,
        cancellationToken
    );
    Console.ReadLine();
}
```

Лістинг файлу UsersService.cs

```
using TeamContactTelegramBot.Repositories;
using TeamContactTelegramBot.Service.Interfaces.Users;

namespace TeamContactTelegramBot.Service.Users
{
    public class UsersService : IUsersService
    {
        private ICommonUOW Common { get; }
        public UsersService(ICommonUOW sender)
        {
            Common = sender;
        }
        public string Send()
        {
            return string.Empty;
        }

        public async Task<List<Data.Users.Users>> GetListAsync()
        {
            return await Common.UsersRepository.Users.GetListAsync();
        }
        public async Task<(bool, byte, int)> CheckIfRegAsync(string log, string pass)
        {
            return await Common.UsersRepository.Users.CheckIfRegAsync(log, pass);

        }
        public async Task<List<Data.Users.Users>> GetAllProgrammersAsync()
```

```
        {
            return await Common.UsersRepository.Users.GetAllProgrammersAsync();
        }
        public async Task<List<Data.Users.Users>> GetAllAnalystsAsync()
        {
            return await Common.UsersRepository.Users.GetAllAnalystsAsync();
        }

        public async Task<Data.Users.Users> AddAsync(Data.Users.Users user)
        {
            return await Common.UsersRepository.Users.AddAsync(user);
        }
    }
}
```

Лістинг файлу IUsersService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TeamContactTelegramBot.CoreSystem;

namespace TeamContactTelegramBot.Service.Interfaces.Users
{
    public interface IUsersService : ITeamContactDI
    {
        Task<List<Data.Users.Users>> GetListAsync();

        Task<(bool, byte, int)> CheckIfRegAsync(string log, string pass);

        Task<List<Data.Users.Users>> GetAllProgrammersAsync();

        Task<List<Data.Users.Users>> GetAllAnalystsAsync();

        Task<Data.Users.Users> AddAsync(Data.Users.Users user);
    }
}
```

Лістинг файлу UsersRepository.cs

```
using AutoMapper;
using AutoMapper.Extensions.ExpressionMapping;
using Microsoft.EntityFrameworkCore;
using TeamContactTelegramBot.Data.Users;
using TeamContactTelegramBot.Repositories.Users.Interfaces;

namespace TeamContactTelegramBot.Repositories.Users
{
    public class UserRepository : IUsersRepository
    {
        private IUsersUOW _uow;
        private UsersContext _context;
        private static MapperConfiguration _conf;
        private static IMapper _mapper;

        public UserRepository(IUsersUOW usersUOW, UsersContext context)
        {
            _uow = usersUOW;
            _context = context;
            if (_conf == null)
            {
                _conf = new MapperConfiguration(cfg =>
                {
                    cfg.AddExpressionMapping();
                    cfg.CreateMap<Data.Users.Users, UsersDTO>().ReverseMap();
```

```
            });
            _conf.CompileMappings();
            _mapper = _conf.CreateMapper();
        }
    }

    public async Task<List<Data.Users.Users>> GetAllProgrammersAsync()
    {
        using UsersContext context = new UsersContext();

        var programmers = await context.DbUsers.Where(x => x.Role ==
2).ToListAsync();

        if (programmers.Count == 0)
            return new List<Data.Users.Users>();

        return programmers;
    }

    public async Task<List<Data.Users.Users>> GetAllAnalystsAsync()
    {
        using UsersContext context = new UsersContext();

        var analysts = await context.DbUsers.Where(x => x.Role == 3).ToListAsync();

        if (analysts.Count == 0)
            return new List<Data.Users.Users>();

        return analysts;
```

```
    }

    public async Task<Data.Users.Users> GetAsync(int userRcd)
    {
        using UsersContext context = new UsersContext();

        var user = await context.DbUsers.FirstOrDefaultAsync(x => x.UsersId ==
userRcd);

        if (user == null)
            return new Data.Users.Users();

        return user;
    }

    public async Task<Data.Users.Users> AddAsync(Data.Users.Users user)
    {
        using UsersContext context = new UsersContext();

        await context.AddAsync(user);
        await context.SaveChangesAsync();

        return user;
    }

    public async Task<List<Data.Users.Users>> GetListAsync()
    {
        using UsersContext context = new UsersContext();
```

```
            var usersInformation = await context.DbUsers.ToListAsync();

            if (usersInformation.Count == 0)
                return new List<Data.Users.Users>();

            return usersInformation.ToList();
        }

        public async Task<(bool, byte, int)> CheckIfRegAsync(string log, string pass)
        {
            using UsersContext context = new UsersContext();

            var user = await context.DbUsers.FirstOrDefaultAsync(x => x.Login == log &&
x.Password == pass);

            if (user == null)
                return (false, 0, 0);
            else
                return (true, user.Role, user.UsersId);
        }
    }
}
```

Схема алгоритму роботи веб-додатку