КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій

Кафедра прикладних інформаційних систем
122 «Комп'ютерні науки»

(шифр і назва спеціальності)

«Прикладні інформаційні системи»

(назва освітньої програми)

Кваліфікаційна робота бакалавра

на тему: «Веб-сервіс пошуку та бронювання авіа білетів»

Виконав _____
(Підпис)
Єфремов Дмитро Андрійович
(прізвище, ім'я, по батькові)

Керівник Краснощок Віктор Миколайович
(прізвище, ім'я, по батькові)

_____
(Резолюція «До захисту»)

Попередній захист:

_____
(Висновок: "До захисту в екзаменаційній комісії ")
Завідувач кафедри _____ Плескач В.Л. _____
(Підпис) (Прізвище, ініціали) (Дата)

Засвідчую, що у цьому курсовому проекті немає запозичень з праць інших авторів без відповідних посилань. ___%
Студент _____
(підпис)

Київ – 2021

Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій
Кафедра прикладних інформаційних систем

Назва теми: «Веб-сервіс пошуку та бронювання авіа білетів»

Освітня програма: Прикладне програмування

Спеціальність: Комп’ютерні науки

ПІБ	Підпис
Єфремов Дмитро Андрійович	

Назва роботи українською та англійською мовами

Веб-сервіс пошуку та бронювання авіа білетів

Web service for searching and booking airline tickets

Мета бакалаврської роботи, завдання

Мета бакалаврської роботи: Підвищення ефективності пошуку та бронювання авіа білетів

План роботи:

1. Сучасні підходи до розроблення і впровадження веб сервісів
2. Аналіз архітектурних рішень і вибір програмних засобів для реалізації веб – систем
3. Програмна реалізація веб-сервісу пошуку та бронювання авіа білетів

ПІБ, ступінь, звання наукового керівника роботи: Краснощок В.М., к.т.н., доцент

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА

Номер	Назва етапів кваліфікаційної роботи бакалавра	Термін виконання етапів кваліфікаційної роботи бакалавра	Відмітка про виконання
1.	Вибір теми та наукового керівника кваліфікаційної роботи бакалавра	26.10.2020	виконав
2.	Видача завдання кваліфікаційної роботи бакалавра	23.11.2020	заява
3.	Настановча групова співбесіда з питань кваліфікаційної роботи бакалавра	01.12.2020	виконав
4.	Затвердження плану кваліфікаційної роботи бакалавра	18.02.2021	виконав
5.	Підбір та вивчення літературних та інших джерел з теми дослідження	25.02.2021	виконав
6.	Підготовка і подання науковому керівнику першого варіанту І розділу роботи	05.03.2021	виконав
7.	Підготовка і подання науковому керівнику першого варіанту ІІ розділу роботи	09.04.2021	виконав
8.	Підготовка і подання науковому керівнику першого варіанту ІІІ розділу роботи	07.05.2021	виконав
9.	Подання роботи у першому варіанті	11.05.2021	виконав
10.	Оформлення пояснювальної записки кваліфікаційної роботи бакалавра	12.05.2021	виконав
11.	Подання кваліфікаційної роботи бакалавра на попередній захист	**24.05.2021**	
12.	Врахування зауважень керівника і подання роботи в остаточному варіанті (з відповідним висновком про допуск) на кафедру	28.05.2021	
13.	Затвердження роботи в цілому (підготовка письмового відгуку керівника, письмова рецензія на бакалаврської роботу)	11.06.2021	
14.	Захист кваліфікаційної роботи бакалавра	16.06.2021	

Здобувач вищої освіти _____
(підпис)

Керівник _____
(підпис)

ВІДОМІСТЬ ДИПЛОМНОЇ РОБОТИ

Зміст пояснювальної записки (перелік питань під час дослідження)

Складові частини дипломної роботи	Обсяг, арк. 74
Титульний аркуш	1 ст.
Завдання до дипломної роботи (календарний план проекту)	1 ст.
Відомість дипломної роботи	1 ст.
Анотація	1 ст.
Анотація (іноземною мовою-англійською)	1 ст.
Зміст	2 ст.
Словник термінів	2 ст.
Вступ	2 ст.
1. Сучасні підходи до розроблення і впровадження веб-сервісів. Аналіз рішень реалізації веб-сервісів	18 ст.
2. Вибір програмних засобів для реалізації веб-сервісу пошуку та бронювання авіа білетів	26 ст.
3. Програмна реалізація веб-сервісу пошуку та бронювання авіа білетів	10 ст.
Висновки	1 ст.
Перелік посилань	5 ст.
Додатки	3 ст.

				ДП XXXX 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.				Відомість дипломної роботи	Лист	Листів
Керівн.						
Н/контр.	Макаренко С.А.					
Зав.каф.	Плескач В.Л.					

АНОТАЦІЯ (РЕФЕРАТ)

До бакалаврської дипломної роботи Єфремова Дмитра Андрійовича на тему «Веб-сервіс пошуку та бронювання авіа білетів»

Дана дипломна робота присвячена створенню універсальних веб-сервісів із використанням сучасних технологій та розробці бази даних для цього сервісу.

Для розробки оптимізованого веб-сервісу було проведено дослідження всіх актуальних, на даний час, технологій, що дозволяють взаємодіяти з веб-сервісами. Після досліджень було здійснено порівняння самих популярних компонентів класичного веб-сервісу, їх швидкість, простота реалізації та взаємодії, рівень захищеності, актуальність на різних сегментах ринку та інше.

Вибір технологій для реалізації дипломної роботи був зроблений по цим основним критеріям: масштабованість, наявні можливості, програмне забезпечення та супроводження системи, а також вирішення проблем які можуть виникнути при розробці веб-сервісу.

Для розробки веб-сервісу із пошуку та бронювання авіа білетів було використано архітектуру REST.

В даній роботі приведені результати досліджень та порівнянь двох варіантів реалізації веб-сервісу, програмне супроводження та приклад впровадження веб-сервісу та детальний опис встановлення та налаштування розробленого шаблона, функцій, компонентів, модулів, контролерів, сервісів та стратегій.

Загальний обсяг роботи: 3 розділи загальною кількістю 74 сторінок, 35 рисунків, 2 таблиці, 56 посилань.

Ключові слова: REST, SOAP, веб-сервіс, Nest.js, HTTP.

ANNOTATION (ABSTRACT)

To the bachelor's thesis of Yefremov Dmytro Andriyovych on the topic "Web service of search and booking of air tickets"

This thesis is devoted to the creation of universal web services using modern technologies and the development of a database for this service.

To develop an optimized web service, a study of all current technologies that allow you to interact with web services was conducted. After the research, a comparison was made for the most popular components of the classic web service, their speed, ease of implementation and interaction, the level of security, relevance in different market segments and more.

The choice of technologies for the implementation of the thesis was made according to the following basic criteria: scalability, available capabilities, software and system support, as well as solving problems that may arise during the development of a web service.

The REST concept was used to develop a web service for searching and booking airline tickets.

This work presents the results of research and comparisons of two options for implementing a web service, software support and an example of realizing a web service and a detailed description of installing and configuring the developed template, functions, components, modules, controllers, services and strategies.

Total volume of work: 3 sections with a total of 74 pages, 35 figures, 2 tables, 56 links.

Keywords: REST, SOAP, web service, Nest.js, HTTP.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

SOA (Service Oriented Architecture) – сервіс орієнтована архітектура

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту

HTTPS (HyperText Transfer Protocol Secure) – розширення протоколу HTTP

URL (Uniform Resource Locator) – єдиний вказівник на ресурс

SOAP (Simple Object Access Protocol) – простий протокол доступу до об'єктів

REST (Representational State Transfer) – передача стану уявлення

W3C (World Wide Web Consortium) – консорціум всесвітньої павутини

WSDL (Web Services Description Language) – мова опису веб-сервісів

XML (Extensible Markup Language) – розширювана мова розмітки

JSON (JavaScript Object Notation) – текстовий формат обміну даними, заснований на JavaScript

CRUD (Create, Read, Update, Delete) – 4 основні функції управління даними – створення, читання, оновлення і вилучення

UDDI (Universal Description Discovery and Integration) – платформо-незалежний інструмент для розміщення описів веб-сервісів

SAML (Security Assertion Markup Language) – мова розмітки декларації безпеки

RMI (Remote Method Invocation) – програмний інтерфейс виклику віддалених методів

DCOM (Distributed Component Object Model) – розширення моделі компонентного об'єкта

CORBA (Common Object Request Broker Architecture) – загальна архітектура брокера об'єктних запитів

FTP (File Transfer Protocol) – протокол передачі файлів по мережі

B2B (Business-to-business) – модель взаємодії на ринку комерції між юридичними особами

EDI (Electronic data interchange) – електронний обмін даними

SMTP (Simple Mail Transfer Protocol) – мережевий протокол, призначений для передачі електронної пошти

EJB (Enterprise JavaBeans) – специфікація технології написання і підтримки серверних компонентів, що містять бізнес-логіку

SSL (Secure Sockets Layer) – рівень захищених сокетів

MIME (Multipurpose Internet Mail Extensions) – багатоцільові розширення інтернет-пошти

ACID (Atomicity, Consistency, Isolation, Durability) – атомарність, узгодженість, ізольованість, міцність

SWS (Semantic Web Services) – закінчені веб-сервіси з описаною семантикою

IOPE (Inputs Outputs Preconditions Effects) – вхідні, вихідні параметри, попередні умови та ефекти виконання

OWL-S (Web Ontology Language) – мова веб-онтології

SAWSDL (Semantic Annotations for WSDL and XML Schema) – семантичні анотації для схеми WSDL та XML

WSMO (Web Service Modeling Ontology) – онтологія моделювання веб-служб

API (Application Programming Interface) – програмний інтерфейс створення додатків

FP (Functional programming) – функціональне програмування

FRP (Functional reactive programming) – функціональне реактивне програмування

ECMAScript – вбудована розширювана мова програмування, що не має засобів введення-виведення

MIT (Massachusetts Institute of Technology) – Массачусетський Технологічний Інститут

DTO (Data Transfer Object) – об'єкт передачі даних

SPA (Single Page Application) – односторінковий додаток

ІС – інформаційна система

БД – база даних

ПЗ – програмне забезпечення

СКБД – система керування базою даних

ВСТУП

Веб-сервіси – це технологія інтеграції веб-застосунків, яка може використовуватися в Інтернет. Використання веб-сервісів має безліч переваг – від очевидного підвищення якості обслуговування клієнтів і простоти розробки до стандартизації та масштабованості.

Актуальність досліджень веб-сервісів можна показати на прикладі пошуку та бронювання авіа білетів. Так, для виконання цих дій вам доведеться відвідати велику кількість різних авіакомпанії та застосунків. Усі ці дії можуть зайняти досить багато часу, перш ніж ви досягнете мети. Ця проблема полягає в тому, що авіа компанії, що спеціалізуються на окремих областях в більшості випадків замкнуті і використовують власні засоби зберігання і представлення даних. Зручніше було б запустити один застосунок, який б взяв від вас необхідну інформацію і виконав усі рутинні дії автоматично, без вашої участі. Саме тому й набули такої популярності веб-сервіси. Тоді, наприклад, авіакомпанія надає веб-сервіс, що дозволяє додаткам отримувати список рейсів між двома містами для заданої дати. У цьому випадку більше не потрібно звертатися до веб-вузла авіакомпанії і вказувати різні критерії пошуку – вся необхідна інформація доступна у вигляді єдиного документа.

Теоретичне значення даної роботи представлено у дослідженні та аналізі питань, що стосуються розробки веб-сервісів, вибору актуальних технологій у створенні веб-сервісу, які будуть корисними у застосуванні при різних сценаріях.

Практичне значення дипломної роботи можна охарактеризувати створенням веб-сервісу, який дозволить отримувати інформацію про всі доступні рейси, зареєстровані авіалінії, наявні аеропорти чи літаки в запасі. Додатково реалізовано процес бронювання та повернення квитка і надано можливість оновлювати усі можливі сутності в базі даних.

Метою дипломної роботи є аналіз та впровадження веб-сервісу з пошуку та бронювання авіа білетів.

Завдання дослідження:

- аналіз існуючих компонентів веб-сервісів, їх принципів роботи та взаємодії;
- опис програмно-технічних рішень, різних видів забезпечення веб-сервісу;
- ознайомлення з існуючими веб-сервісами;
- ознайомлення з процесом реалізації веб-сервісу від створення прототипу до проведення тестування;
- реалізація на практиці веб-сервісу з пошуку та бронювання авіа білетів.

Об’єктом дослідження у цій роботі є процес пошуку та бронювання авіа білетів, а **предметом дослідження** – програмні засоби пошуку та бронювання авіабілетів, компоненти веб-сервісів, а також програмо-технічні, організаційні засади, принципи, концептуальні підходи до побудови веб-сервісів даної тематики.

Методи дослідження полягають в системному аналізі та синтезі, теорії систем та застосуванні клієнт-серверної моделі архітектурного рішення досліджуваної системи, методах математичної статистики для збору даних веб-сервісу.

У процесі виконання дипломної роботи було використано: редактор коду VisualCode, документацію з використання технологій веб-сервісів, мови програмування JavaScript та TypeScript, NoSQL базу даних MongoDB, фреймворк Nest.js, бібліотеки Mongoose, Moment.js, Fuse.js та інших, міжнародні стандарти з побудови веб-сервісів, а саме архітектури Rest та загальні концепції з написання чистого коду.

Структура дипломної роботи складається із вступу, трьох розділів, висновку, додатків і списку використаних джерел.

РОЗДІЛ 1: СУЧАСНІ ПІДХОДИ ДО РОЗРОБЛЕННЯ І ВПРОВАДЖЕННЯ ВЕБ-СЕРВІСІВ. АНАЛІЗ РІШЕНЬ РЕАЛІЗАЦІЇ ВЕБ-СЕРВІСІВ

Що таке веб-сервіси? Термін веб-сервіс можна охарактеризувати як послугу, що пропонується електронним пристроєм іншому електронному пристрою, що взаємодіє між собою через всесвітню павутину, або як сервер, що працює на комп'ютерному пристрої, прослуховує запити через певний порт через мережу, обслуговує веб-документи (HTML, JSON, XML, зображення).

У веб-службі веб-технологія, така як HTTP, використовується для передачі машиночитаних форматів файлів, таких як XML та JSON. На практиці веб-служба зазвичай надає об'єктно-орієнтований веб-інтерфейс серверу баз даних, що використовується, наприклад, іншим веб-сервером або мобільним додатком, що забезпечує інтерфейс для кінцевого користувача. Іншим додатком, пропонованим кінцевому користувачеві, може бути комбінування, коли веб-сервер споживає кілька веб-служб на різних машинах і компілює вміст в один користувальницький інтерфейс.

Підсумовуючи, веб-сервіси (рисунок 1.1) – це автономні, модульні, розподілені, динамічні додатки, які можна описувати, публікувати, розміщувати або викликати по мережі для створення продуктів, процесів і ланцюжків поставок.

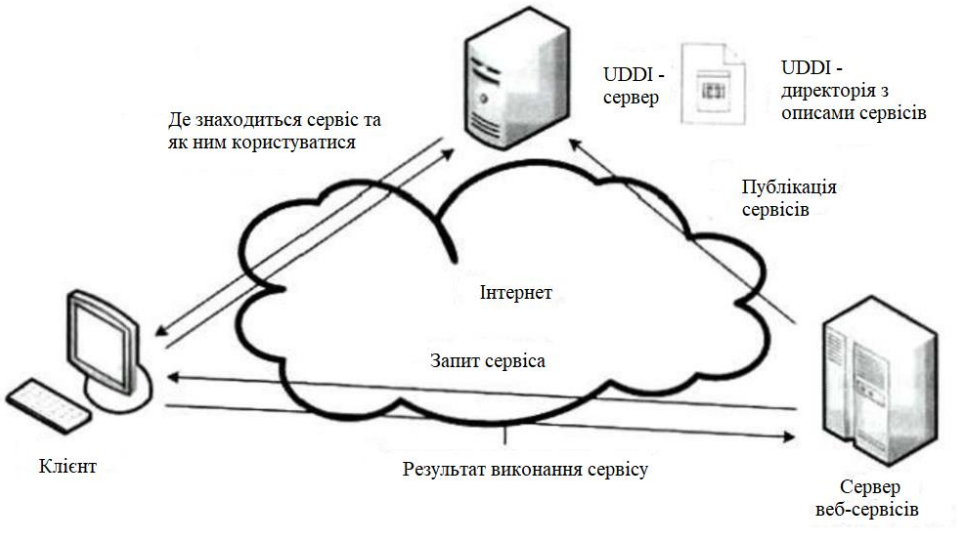

Рисунок 1.1 – Узагальнена схема роботи веб-сервісів

1.1 Основні поняття веб-сервісу

Веб-сервіс або веб-служба – це система, доступна в інтернет-просторі і працює на основі спеціальної програми зі стандартизованими інтерфейсами, ідентифікація якої виконується за допомогою URL-адреси. Пошук здійснюється іншими ресурсами, основним завданням є взаємодія програмних систем на різних платформах, для чого використовуються відкриті протоколи. До веб-сервісів відносять пошуковики, хостинги, електронні пошти, хмарні сховища, календарі та інші сервіси [1].

Ці додатки можуть бути локальними, розподіленими або мережевими. Веб-сервіси побудовані на основі відкритих стандартів, таких як TCP / IP, HTTP, Java, HTML і XML. Веб-служба є одиницею модульності при використанні SOA додатку.

Сервіс-орієнтована архітектура або **SOA** – це стиль дизайну програмного забезпечення, при якому послуги надаються іншим компонентам за допомогою компонентів додатків через протоколи зв'язку через мережу. Її принципи не залежать від постачальників та інших технологій. В архітектурі, орієнтованій на послуги, низка служб взаємодіють між собою одним із двох способів: шляхом передачі даних або через дві або більше служб, що координують діяльність. Це лише одне визначення сервісно-орієнтованої архітектури. [7]

Хоча визначальні концепції сервісно-орієнтованої архітектури різняться в залежності від компанії, існує шість основних принципів, що охоплюють широке поняття архітектури, орієнтованої на послуги. Ці основні цінності включають:

- ділова цінність;
- стратегічні цілі;
- внутрішня взаємодіяльність;
- спільні послуги;
- гнучкість;

- еволюційне вдосконалення.

Набула поширення в кінці 1990-х – початку 2000-х років. З середини 2010-х років набула популярності мікросервісна архітектура – варіант SOA, що базується на застосування наскільки це можливо мінімальних за розміром сервісів. У цьому вживанні термін вимагає уточнення, чи йде мова про пошук, веб-пошту, зберігання документів, файлів, закладок і т.д. Такими веб-сервісами можна користуватися незалежно від комп'ютера, браузера або місця доступу в Інтернет.

Семантичні веб-служби або **SWS** є складовою семантичної мережі, оскільки вони використовують розмітку, яка робить дані доступними для аналізу та роботи з ними. Семантичні веб-служби використовуються для просування загальних форматів даних та дозволяє включати семантичний вміст, що описує формат, на веб-сторінках. Семантичні веб-сервіси побудовані навколо універсальних стандартів для обміну семантичними даними, що полегшує програмістам поєднання даних з різних джерел та служб, не витрачаючи зайвих зусиль.

Веб-служби можуть бути налаштовані та активовані "за лаштунками", коли веб-браузер робить запит до веб-сервера, який потім використовує інші веб-сервіси для створення більш повної відповіді, ніж він міг би зробити це самостійно. Семантичні веб-служби можуть також використовуватися автоматичними програмами, які працюють без будь-якого підключення до веб-браузера. Виділяють 2 характерних семантичним веб-сервісам поняття – хореографію (choreography) та оркестровку (orchestration).

Хореографія займається описом зовнішньої видимої поведінки служб, як набору обмінів повідомленнями, необов'язково слідуючих шаблону обміну повідомленнями, з точки зору функціональних можливостей споживача.

Оркестровка займається описом того, як низка служб, дві або більше, співпрацюють та спілкуються з метою досягнення спільної мети.

Існує кілька мов опису семантичних веб-сервісів: SAWSDL, OWL-S, WSMO. Всі ці мови орієнтовані на взаємодію з WSDL. З цих мов найбільш широкими можливостями мають OWL-S і WSMO.

При використанні цієї онтології і мов розмітки семантика сервісу характеризується наступними 4 компонентами:

- inputs – вхідні параметри;
- outputs – вихідні параметри;
- preconditions – попередні умови;
- effects – ефекти виконання.

Концепція веб-сервісів має на увазі, що окремі веб-сервіси пропонують сукупність операцій, що володіють певною обмеженою функціональністю. У деклараціях, пов'язаних з потенційними можливостями семантичних веб-сервісів, можливість автоматичної композиції стоїть на першому місці. При цьому під терміном «семантичний веб-сервіс» розуміються не тільки веб-сервіси з незалежно викликаються операціями, а й композитні веб-сервіси.

Термін веб-сервіс було введено організацією W3C та в основному на початку відносився до великої кількості різноманітних систем, а в першу чергу до клієнтів та серверів, у яких процес взаємодії проходив завдяки повідомленням протоколу SOAP. При цьому припускається, що в обох випадках також існує опис доступних операцій формату WSDL. Наявність цього опису не є визначальною вимогою до стандарту SOAP, а радше складовою для автоматичної генерації коду на деяких платформах на стороні клієнта.

Приклади веб-сервісів:

- Маркетплейс – це інтернет-магазин, який дозволяє стороннім компаніям-розробникам ПО реалізувати свій продукт;
- Стрімінг – це надання послуг з видачі потокового мультимедіа;
- Магазин додатків – це надання торгового веб-майданчика для торгівлі онлайн.

1.2 Переваги і недоліки веб-сервісів

Переваги веб-сервісів:

- Викриття існуючої функціональності в мережі. Веб-сервіс є одиницею керованого коду, який може бути віддалено викликаний за допомогою HTTP, тобто його можна активувати за допомогою HTTP-запитів. Веб-сервіси дозволяють виявляти функціональність існуючого коду через мережу. Це значна перевага, порівняно з такими технологіями, як CORBA, DCOM або Java RMI. З іншого боку, веб-служби не прив'язані намертво до HTTP – можуть використовуватися і інші протоколи.
- Взаємодія. Веб-сервіси дозволяють різним програмам спілкуватися одна з одною та обмінюватися даними та службами між собою. Інші програми також можуть використовувати веб-служби.
- Веб-служби забезпечують взаємодію програмних систем незалежно від платформи та технології.
- Стандартизований протокол. Веб-сервіси використовують стандартний галузевий протокол для зв'язку. Всі чотири шари (сервісний транспорт, XML-повідомлення, опис сервісу та шари виявлення служб) використовують чітко визначені протоколи у стеку протоколів веб-сервісів. Ця стандартизація стек протоколу дає бізнесу безліч переваг, таких як широкий вибір варіантів, зменшення витрат за рахунок конкуренції та підвищення якості. Так, завдяки використанню XML досягається простота розробки і налагодження веб-служб.
- Низька вартість спілкування. Веб-сервіси використовують протокол SOAP за протоколом HTTP, тому ви можете використовувати свій існуючий недорогий інтернет для впровадження веб-служб. Це рішення набагато менш дорого, ніж у фірмових рішеннях, таких як EDI/B2B. Крім SOAP через HTTP, веб-сервіси також можуть бути реалізовані на інших надійних транспортних механізмах, таких як FTP.

Недоліки веб-сервісів:

- Менша продуктивність і більший розмір мережевого трафіку в порівнянні з технологіями RMI, CORBA, DCOM за рахунок використання текстових XML-повідомлень. Однак на деяких веб-серверах можливе налаштування зниження мережевого трафіку.
- Аспекти безпеки. Відповідальні веб-служби повинні використовувати кодування, можливо – вимагати аутентифікації користувача. Чи достатньо тут застосування HTTPS, або кращі такі рішення, як XML Signature, XML Encryption або SAML – має бути вирішено самостійно кожним розробником.

1.3 Ролі веб-сервісу

Загалом можна виділити три основні інстанції, які взаємодіють в рамках веб-служби(рисунок 1.2):

- Замовник або користувач (service requester) – це будь-який споживач веб-сервісу. Замовник використовує наявну веб-службу, відкриваючи мережеве з'єднання та надсилаючи запит XML;
- Виконавець або постачальник (service provider) – це постачальник веб-сервісу. Постачальник послуг реалізує цю послугу та робить її доступною в Інтернеті;
- Каталог або брокер (service broker) – це логічно централізована служба реєстру послуг. Каталог забезпечує центральне місце, де розробники можуть публікувати нові сервіси або знаходити існуючі. Таким чином, він є центральним кліринговим центром для компаній та їх послуг.

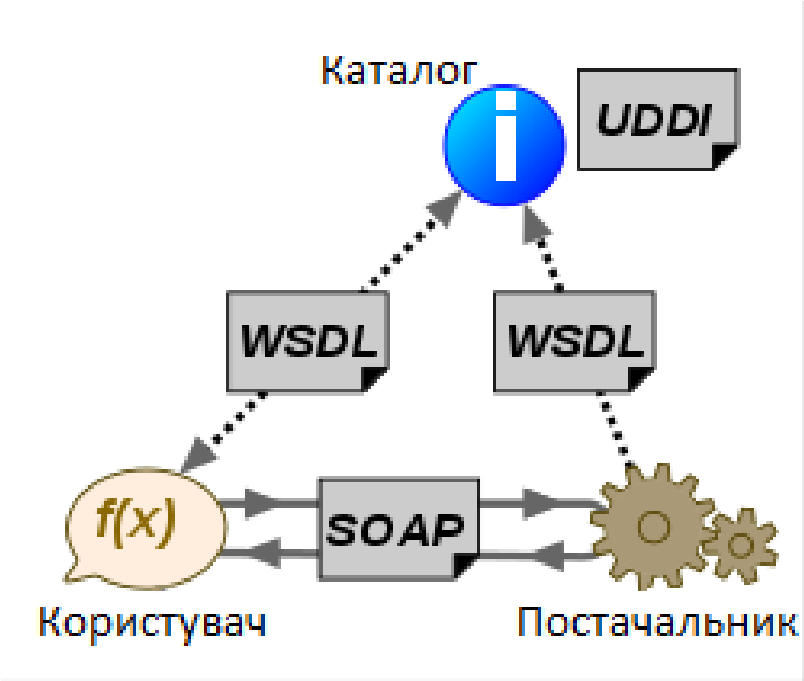

Рисунок 1.2 – Веб-служба на основі UDDI і SOAP

1.4 Технології реалізації та платформи

Існують засоби автоматизації розробки веб-служб, які поділяються на дві основні групи. При розробці знизу-вгору спочатку пишуться імплементовані класи, а з їх вихідних даних скалдається WSDL-файли, що описують та документують службу. Недоліком цього методу є схильність деяких класів до частих змін. При підході "зверху вниз" спочатку готується WSDL, а з нього генерується скелет певного класу, імплементує службу. Цей шлях вважається більш важким, зате призводить до більш чистим і краще захищеним від змін рішенням. Поки формат повідомлень, якими обмінюються замовник і виконавець, не змінюється, зміни в кожному з них не порушують взаємодії. Ця техніка називається іноді «contract first», так як вихідною точкою є WSDL («договір» між замовником і виконавцем). Також існує розробка веб-служби на основі SDK для розпізнавання документів (OCR).

Основною більшості платформ веб-сервісів є XML + HTTP. Найпопулярніші приклади платформ веб-сервісів:

- ColdFusion від Adobe;

- DotGNU від GNU Project;
- GlassFish – від компанії Oracle;
- Google App Engine – платформа для масштабованих додатків, що використовують інфраструктуру компанії Google;
- IBM Lotus Notes лінійка ПО для організації спільної роботи;
- JBoss – компанії Red Hat;
- Mono – платформа розробки від Xamarin (раніше Novell);
- .NET Framework сервери від Microsoft;
- Web Application Server від SAP;
- WebLogic від компанії Oracle (продукт BEA Systems поглиненої Oracle);
- webMethods Integration Platform від Software AG;
- WebSphere Application Server від IBM (заснований на Apache і платформі J2EE);

1.5 Сценарії застосування веб-сервісів

Веб-сервіси представляють 3 наступних сценарія застосування веб-сервісів:

- Веб-сервіси як реалізація логіки програми (рисунок 1.3). Тобто створення принципово нового додатка з бізнес-логікою, який реалізований у веб-службі;

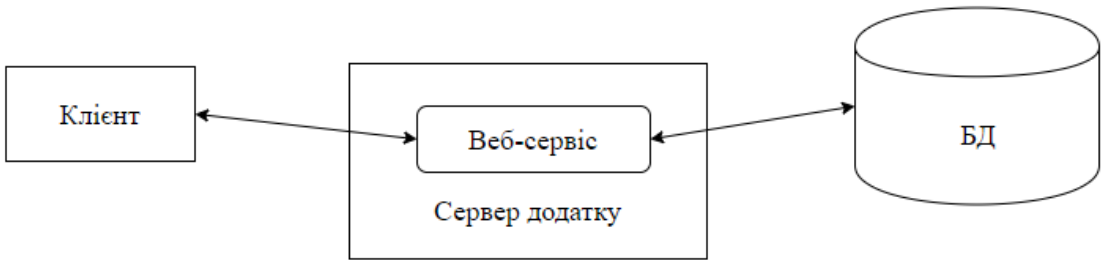

Рисунок 1.3 – Веб-сервіси як реалізація логіки програми

- Веб-сервіси як засіб інтеграції (рисунок 1.4). Тобто використання веб-сервісу як способу доступу до віддалених клієнтів до внутрішньої ІС компанії або для організації взаємодії компонента (наприклад, EJB, COM-компонента) з різними віддаленими клієнтами;

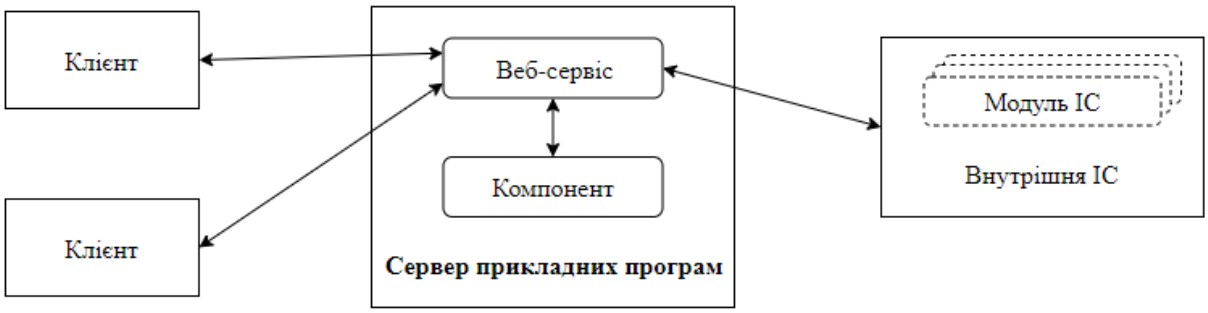

Рисунок 1.4 – Веб-сервіси як засіб інтеграції

- Використання веб-сервісу як будівельного блоку при створенні програми (рисунок 1.5). Програма може використовувати веб-сервіси як окремі компоненти, що забезпечують певну функціональність. Існують різні служби, які забезпечують якісне вирішення таких завдань, як аутентифікація, календар, обмін повідомленнями, пошук, переклад тощо.

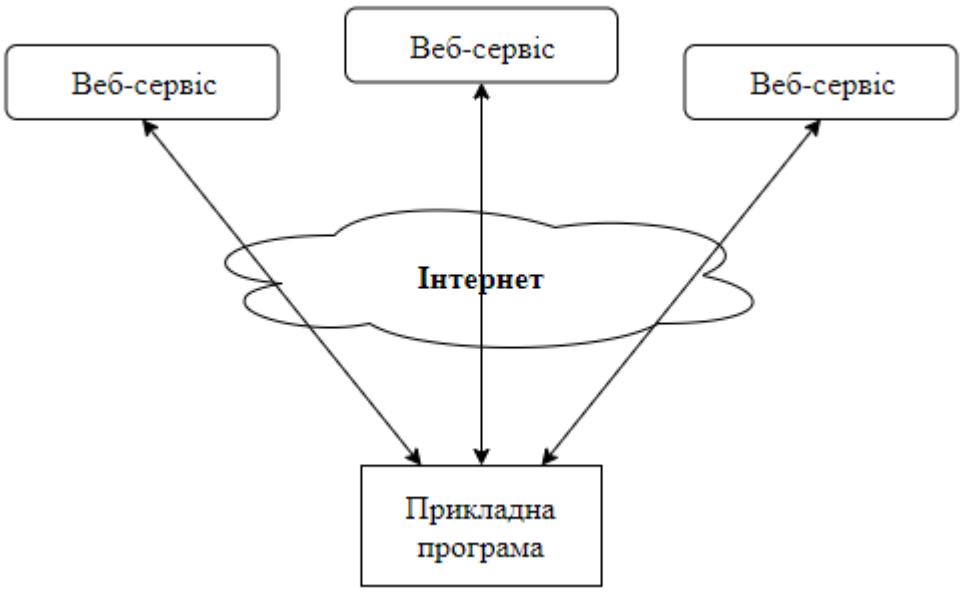

Рисунок 1.5 – Використання веб-сервісу як будівельного блоку при створенні програми

1.6 Стандарти та концепції веб-сервісів

Багато звичайних веб-сервісів працюють за допомогою наступних стандартів (рисунок 1.6):

- XML (Extensible Markup Language);
- SOAP (Simple Object Access Protocol);
- WSDL (Web Services Description Language);
- UDDI (Universal Description, Discovery and Integration) ;
- XML-RPC (XML Remote Procedure Call).

Безумовно, існують і інші протоколи, але, оскільки вони не набули широкого поширення та є кілька «застарілим», зупинимося в цьому огляді на цих основних стандартах.

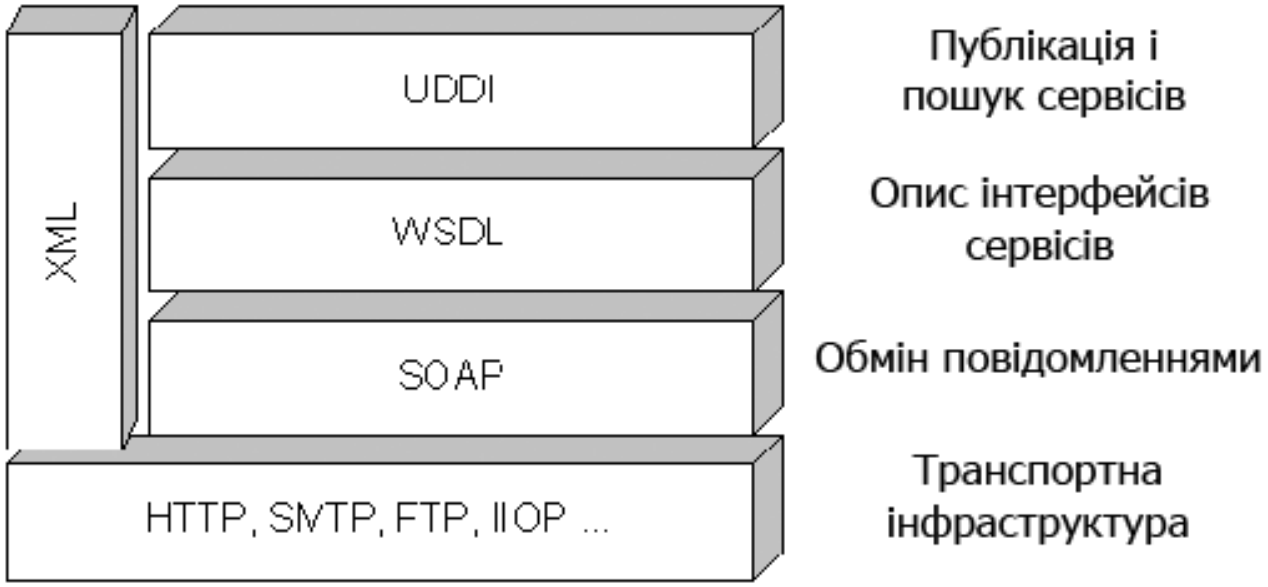

Рисунок 1.6 – Взаємозв'язок компонентів веб-сервісу

Так, **WSDL** – це мова опису веб-сервісів і доступу до них, заснований на мові XML. Опис типу WSDL є основою SOAP [8]. Кожен документ WSDL можна розбити на наступні логічні частини:

- визначення типів даних (types) – це визначення виду відправлених і отриманих сервісом XML-повідомлень;

- елементи даних (message) – це повідомлення, що використовуються веб-сервісом;
- абстрактні операції (portType) – це список операцій, які можуть бути виконані з повідомленнями;
- зв'язування сервісів (binding) – це спосіб, яким повідомлення буде доставлено.

UDDI – це інструмент для розташування описів веб-сервісів (WSDL) для подальшого їх пошуку іншими організаціями та інтеграції до своїх систем.
UDDI – це багатоплатформність, основана на XML. UDDI є відкритим проектом, спонсором якого є OASIS, який дозволяє організаціям публікувати WSDL для подальшого їх пошуку іншими організаціями та інтеграції в свої системи, а також визначає, як сервіси або додатки взаємодіють через Internet [6].

UDDI був спочатку запропонований як основний стандарт веб-сервісу. Він призначений для опитування SOAP повідомленнями і для забезпечення доступу до WSDL документам, що описує прив'язки протоколів і форматів повідомлень, необхідних для взаємодії з веб-послугами, переліченими в його каталозі.

XML – це розширювана мова розмітки, яка визначає набір правил для кодування документів у форматі, який читається як людиною, так і машинно. Специфікація XML 1.0 W3C 1998 р. та кілька інших пов'язаних специфікацій – усі з них безкоштовні відкриті стандарти – визначають XML. Цілі дизайну XML підкреслюють його простоту, загальність та зручність використання в Інтернеті. Хоча дизайн XML зосереджений на документах, мова широко використовується для подання довільних структур даних, таких як ті, що використовуються у веб-сервісах [11]. MIME-тип application/xml, text/xml.

XML-RPC – це стандарт/протокол виклику віддалених процедур, що використовує XML для кодування своїх повідомлень і HTTP в якості транспортного механізму. Відрізняється винятковою простотою в застосуванні.

У XML-RPC клієнт виконує RPC, надсилаючи HTTP-запит на сервер, який реалізує XML-RPC і отримує відповідь HTTP. У порівнянні з протоколами RESTful, де передаються подання ресурсів (документи), XML-RPC призначений для виклику методів. Яскравою технічною відмінністю між типовими протоколами RESTful та XML-RPC є те, що протокол RESTful використовує HTTP URI для інформації про параметри, тоді як для XML-RPC URI просто ідентифікує сервер. JSON-RPC схожий на XML-RPC [28].

1.6.1 Стандарт SOAP

SOAP – це специфікація протоколу обміну повідомленнями для обміну структурованою інформацією при реалізації веб-сервісів у комп'ютерних мережах. Для формату повідомлень він використовує набір інформації XML і покладається на протоколи прикладного рівня, найчастіше протокол передачі гіпертексту (HTTP), хоча деякі застарілі системи обмінюються повідомленнями за допомогою простого протоколу передачі пошти (SMTP) для узгодження та передачі повідомлень [5].

SOAP дозволяє розробникам викликати процеси, що працюють у різних операційних системах (таких як Windows, macOS та Linux), для автентифікації, авторизації та спілкування за допомогою розширюваної мови розмітки (XML). Оскільки веб-протоколи, такі як HTTP, встановлюються та працюють у всіх операційних системах, SOAP дозволяє клієнтам викликати веб-служби та отримувати відповіді незалежно від мови та платформ.

Повідомлення SOAP виглядає так:

- Envelope – кореневий елемент, який визначає повідомлення і простір імен, використане в документі;
- Header – містить атрибути повідомлення, наприклад: інформація про безпеку або про мережевої маршрутизації;
- Body – містить повідомлення, яким обмінюються програми;

- Fault – необов'язковий елемент, який надає інформацію про помилки, які мали місце під час обробки повідомлень.

1.6.2 Архітектурний стиль REST

REST – це концепція (рисунок 1.7), в основі якої лежить архітектурний стиль, а не нова технологія. Зазвичай він використовується для створення інтерактивних додатків, які використовують веб-сервіси. Веб-служба, яка відповідає цим інструкціям, називається RESTful. Така веб-служба повинна надавати свої веб-ресурси у текстовому поданні та дозволяти їх читати та модифікувати за допомогою протоколу без стану та заздалегідь визначеного набору операцій. Цей підхід забезпечує взаємодію між комп'ютерними системами в Інтернеті, які надають ці послуги. REST є альтернативою, наприклад, SOAP як спосіб доступу до веб-служби [29].

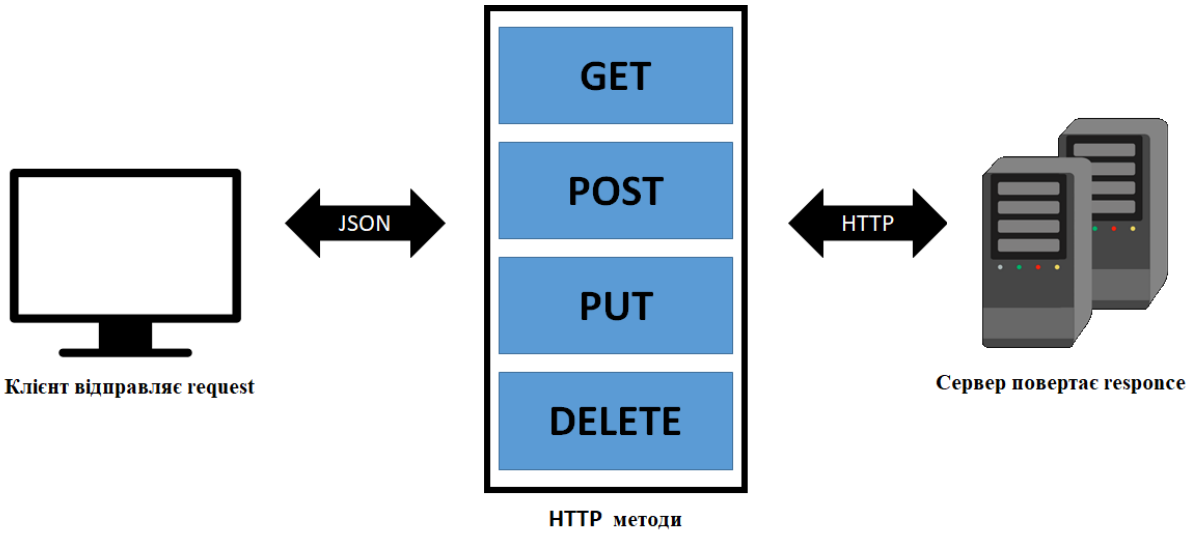

Рисунок 1.7 – Схема роботи RESTful веб-сервісу

Веб-ресурси вперше були визначені у Всесвітній павутині як документи або файли, ідентифіковані за їхніми URL-адресами. Сьогодні це визначення є набагато загальнішим і абстрактнішим і включає кожну річ, сутність чи дію, які можна ідентифікувати, назвати, звернутись до них, обробити чи виконати будь-яким способом в Інтернеті. У веб-службі RESTful запити, надіслані на URI

ресурсу, викликають відповідь із корисним навантаженням, відформатованим у форматі HTML, XML, JSON або іншому форматі. Наприклад, відповідь може підтвердити, що стан ресурсу змінено. Відповідь може також містити гіпертекстові посилання на відповідні ресурси. Найпоширенішим протоколом для цих запитів та відповідей є HTTP. Він забезпечує такі операції (методи HTTP), як GET, POST, PUT та DELETE. Використовуючи протокол без стану та стандартні операції, системи RESTful прагнуть до швидкої продуктивності, надійності та здатності рости шляхом повторного використання компонентів, якими можна керувати та оновлювати їх, не впливаючи на систему в цілому, навіть під час її роботи. Метою REST є підвищення продуктивності, масштабованості, простоти, модифікації, видимості, портативності та надійності.

Термін REST був введений і визначений у 2000 році Роєм Філдінгом у своїй докторській дисертації. Дисертація Філдінга пояснила принципи REST, які були відомі як "об'єктна модель HTTP", починаючи з 1994 року, і використовувалися при розробці стандартів HTTP 1.1 та Uniform Resource Identifiers (URI). Цей термін має на меті створити уявлення про те, як поводиться добре розроблений веб-додаток: це мережа веб-ресурсів (віртуальний автомат стану), де користувач просувається через додаток, вибираючи ідентифікатори ресурсів та операції з ресурсами, такі як GET або POST (переходи стану додатка), в результаті чого подання наступного ресурсу (наступний стан програми) передається кінцевому користувачеві для їх використання.

JSON – це загальний формат для представлення значень і об'єктів. Як і багато інших текстових форматів, JSON зрозумілий для людей. Формат JSON був розроблений Дугласом Крокфордом [18]. MIME-тип application/json.

Обов'язковими умовами-обмеженнями REST архітектури за Роєм Філдінгом є:

- Клієнт – серверна архітектура. Відокремлення логіки додатку від різних клієнтів, код більш чистий та універсальний, а структура серверу

простіша і масштабована. Розробка клієнтів і сервера може вестися абсолютно незалежно.

- Stateless – сервер. Стан клієнта не повинен зберігатися на сервері, ні в якому вигляді, цим займається виключно сам клієнт. Це спрощує доопрацювання і супровід сервера, робить його більш стабільним;
- Кешування. Повинна бути розроблена чітка система кешування запитів до сервера, що дозволяє значно поліпшити продуктивність;
- Багатошарова структура. Наявність/відсутність проміжних серверів кешування, балансування навантаження, додаткового проксінгу має залишатися абсолютно непоміченим з боку клієнтів;
- Єдиний інтерфейс включає в себе:
  - Ідентифікація ресурсів. Кожен ресурс має унікальний ідентифікатор URI.
  - Взаємодія з ресурсами через уявлення. Кожен ресурс має свою унікальну адресу URI (подання), і кілька визначень для управління цим поданням.
  - Самоописуючі повідомлення (MIME type). Кожна відповідь має містити всю необхідну інформацію, щоб його можна було правильно обробити, не звертаючись до допомоги к іншим ресурсам.
- Код на вимогу. Опціональний елемент структури. Дозволяє отримувати програмний код для подальшого його виконання на клієнті.

Таким чином, якщо ваш додаток відповідає всім вимогам (обмеженням), описаним вище, воно сміливо може називатися RESTful. Виняток становить лише код на вимогу – цей параметр опціональний.

1.6.3 Порівняння SOAP та REST

Для того щоб можно було порівнювати ці терміни необхідно дати оцінку кожному з них. Так REST має наступні визначаючі риси:

- стабільність (за рахунок відсутності збереженної інформації про стан клієнта, яка може бути втрачена);
- продуктивність (за рахунок кешування);
- масштабованість;
- легка складність системи взаємодії;
- простота та відкритість інтерфейсів;
- портативність компонентів та моливість легко внеосити зміни;
- здатність пристосовуватися до нових вимог;
- відсутність специфікації;
- неоднозначність методів управління даними.

Натомість SOAP включає в себе:

- галузевий стандарт за версією W3C;
- наявність суворої специфікації;
- широка підтримка в продуктах Microsoft,
- однозначність;
- складність реалізації;
- порівняно старий стандарт;
- складність/ресурсомісткість парсинга XML-даних.

Так, який спосіб реалізації використовувати? Це питання необхідно розглядати в контексті реалізованої системи і її обмежень. Зазвичай, SOAP використовується в великих корпоративних системах зі складною логікою, коли потрібні чіткі стандарти, підкріплені часом. XML-RPC, вже скоріш застарів і не має сенсу через наявність JSON-RPC. RPC-протоколи підійдуть для зовсім простих систем з малою кількістю одиниць інформації і API-методів.

Якщо ж розробляється публічне API і логіка взаємодії багато в чому покривається четвіркою методів CRUD – можна сміливо вибирати REST. Він найбільш популярний в WEB. Так, Яндекс, Google та інші використовують саме його для свого API.

REST проти SOAP можна перефразувати як "Простота проти Стандарту". У разі REST (простота) у вас буде швидкість, розширюваність і підтримка багатьох форматів. У випадку з SOAP у вас буде більше можливостей з безпеки (WS-security) і безпеки транзакцій (ACID).

Висновки

Таким чином, повноцінний веб-сервіс – це будь-яка послуга, яка:

- Доступна через Інтернет або приватні (інтрамережі) мережі;
- Використовує стандартизовану систему обміну повідомленнями;
- Не прив'язаний до жодної операційної системи або мови програмування;
- Самоописується через загальну граматику;
- Виявляється за допомогою простого механізму пошуку.

Основною платформою веб-сервісів є XML + HTTP. Більшість веб-сервісів працюють за допомогою наступних стандартів:

- XML – це розширювана мова розмітки, призначена для зберігання і передачі структурованих даних
- SOAP – це протокол обміну повідомленнями на базі XML
- WSDL – це мова опису зовнішніх інтерфейсів веб-сервісів на базі XML
- UDDI – це універсальний інтерфейс розпізнавання, опису та інтеграції. Каталог веб-сервісів і відомостей про компанії, що надають веб-сервіси в загальне користування або конкретним компаніям.

Різниця в концепції REST та стандарті SOAP:

- REST підтримує різні формати: text, JSON, XML; SOAP – тільки XML;
- REST працює тільки по HTTP(S), а SOAP може працювати з різними протоколами;

- REST може працювати з ресурсами. Кожен URL це уявлення будь-якого ресурсу. SOAP працює з операціями, які реалізують якусь бізнес логіку за допомогою декількох інтерфейсів,
- SOAP на основі читання не може бути поміщена в кеш, а REST в цьому випадку може бути закешована;
- SOAP підтримує SSL і WS-security, в той час як REST – тільки SSL;
- SOAP підтримує ACID. REST підтримує транзакції, але не один з ACID несумісний з двох фазовим коммітом.

Так, підсумовуючи, можна сказати, що відправка даних по мережі в форматі JSON дешевше, ніж відправка їх в форматі XML щодо корисного навантаження. SOAP підтримує тільки XML, але REST підтримує різні формати, такі як текст, JSON, XML і т. д.

Важливим моментом є те, що одним з переваг SOAP є використання транспорту "generic", а REST використовує HTTP/HTTPS. SOAP може використовувати майже будь-який транспорт для відправки запиту, але REST не може. Таким чином, тут ми отримали перевагу використання SOAP.

SOAP підтримує SSL так само, як і REST, крім того, він також підтримує WS-Security, який додає деякі функції корпоративної безпеки. WS-Security забезпечує захист від створення повідомлення до його споживання. Таким чином, для безпеки транспортного рівня будь-яка лазівка, яку ми знайшли, може бути відвернена за допомогою WS-Security. Крім того, оскільки REST обмежений протоколом HTTP, його підтримка транзакцій не є ні сумісної з ACID. А SOAP має всебічну підтримку як для управління транзакціями на основі ACID для короткострокових транзакцій, так і для управління транзакціями на основі компенсації для довгострокових транзакцій.

Виконавши порівняння стандарту SOAP та архітектури REST та проаналізувавши усі переваги та недоліки кожного компонента, я вирішив обрати для своєї дипломної роботи REST архітектуру.

РОЗДІЛ 2: ВИБІР ПРОГРАМНИХ ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-СЕРВІСУ ПОШУКУ ТА БРОНЮВАННЯ АВІА БІЛЕТІВ

В даному розділі проводиться опис програмних засобів по розробці веб-сервісу пошуку та бронювання квитків. Для якісної розробки будуть проведені наступні роботи:

- аналіз вихідних даних;
- опис технологій та додатків веб-сервісу;
- розробка стратегії створення веб-сервісу;
- опис структури веб-сервісу;
- опис функціоналу;
- опис системних елементів та програмного забезпечення;
- тестування веб-сервісу.

2.1 Аналіз вихідних даних

Веб-сервіс пошуку та бронювання авіа білетів являє собою відкрите ПЗ, яка розташовується в мережі Інтернет. До основної цільової аудиторії веб-сервісу можливо віднести розробників веб-дизайну та FrontEnd частини.

Для створення веб-сервісу було виведено наступну реалізацію архітектури та компонентів веб-сервісу, які повинні відповідати наступним вимогам:

- виконання 6-ти основних умов-обмежень для реалізації Rest архітектури;
- пошук авіа білету, подальше його бронювання та можливість повернення;
- отримання актуальної інформації про рейс, доступні аеропорти, їх авіапарк;
- додатковий вибір класу та місця в літаку;
- можливість вносити інформацію до кожної з сутностей БД;

- розробка системи авторизації та автентифікації;
- забезпечення чистоти, читаності, простоти та повторного використання коду;
- рефакторинг та оптимізація контролерів;
- забезпечення відповідного рівня безпеки веб-сервісу;
- розробка наглядної документація та карти з користування веб-сервісу.

2.2 Опис технологій та допоміжних засобів веб-сервісу

Для реалізації проекту, було використано технології та бібліотеки для створення веб-сервісів: мови програмування JavaScript та TypeScript, NoSQL базу даних MongoDB, фреймворк Nest.js, бібліотеки Mongoose, Moment.js, Fuse.js та інші. За допомогою даних технологій можна виконати усі поставлені задачі практичного завдання роботи. Для ознайомлення з використаними технологіями у розробці застосунку, надається повний опис кожної з використаних технологій, їх функціонал та обгрунтування того з якою метою вони були залучені.

2.2.1 Мови програмування JavaScript та TypeScript

JavaScript (JS) – є мовою програмування, яка відповідає специфікації ECMAScript. JavaScript є високорівневою, мультипарадигменною мовою програмування. Він має синтаксис фігурних дужок, динамічне введення тексту, орієнтацію на об’єкти на основі прототипу та функції першого класу [22].

Поряд з HTML та CSS, JavaScript є однією з основних технологій Всесвітньої мережі. Понад 97% веб-сайтів використовують його на стороні клієнта для поведінки веб-сторінок , часто включаючи сторонні бібліотеки. Усі основні веб-браузери мають спеціальний механізм JavaScript для виконання коду на пристрої користувача.

Як мультипарадигменна мова, JavaScript підтримує керовані подіями, функціональні та імперативні стилі програмування. Він має інтерфейси програмування програм (API) для роботи з текстом, датами, регулярними виразами, стандартними структурами даних та обєктною моделлю документа (DOM). Стандарт ECMAScript не включає введення/виведення, наприклад, мережеві, сховища чи графічні засоби. На практиці веб-браузер або інша система виконання забезпечує API JavaScript для вводу-виводу. Спочатку двигун JavaScript використовувався лише у веб-браузерах, але зараз вони є основними компонентами інших програмних систем, зокрема серверів та різноманітних додатків.

Моливості JavaScript:

- завантаження нового вмісту веб-сторінки без перезавантаження сторінки через Ajax або WebSocket. Наприклад, користувачі соціальних мереж можуть надсилати та отримувати повідомлення, не виходячи з поточної сторінки;
- анімація веб-сторінок, наприклад, вицвітання та зменшення об’єктів, зміна розміру та переміщення;
- браузерні ігри;
- керування відтворенням потокового мультимедіа;
- створення спливаючих вікон;
- перевірка вхідних значень веб-форми перед надсиланням даних на веб-сервер;
- запис даних про поведінку користувача, а потім надсилання їх на сервер.

TypeScript (TS) – це мова програмування, розроблена та підтримувана корпорацією Майкрософт [32]. Це синтаксична набір до JavaScript, що додає до мови необов’язкову статичну типізацію. TypeScript призначений для розробки великих додатків та перекомпіляції в JavaScript. Оскільки TypeScript є надмножиною JavaScript, існуючі програми JavaScript також є дійсними

програмами TypeScript. Розробником мови TypeScript є Андерс, який створив раніше Turbo Pascal, Delphi і C#.

TypeScript є зворотно сумісною з JavaScript і компілюється в останню. Код експериментального компілятора, який транслює TypeScript в JavaScript, поширюється під ліцензією Apache. TypeScript відрізняється від JavaScript можливістю явного статичного призначення типів, підтримкою використання повноцінних класів (як в традиційних об'єктно-орієнтованих мовах), а також підтримкою підключення модулів, що покликане підвищити швидкість розробки, полегшити читаність, рефакторинг і повторне використання коду, допомогою в здійснені пошуку помилок на етапі розробки і компіляції, і прискорення виконання програм.

Таким чином, програма JavaScript також є правильною програмою TypeScript, і програми TypeScript можуть легко включати JavaScript. TypeScript компілює ES3-сумісний JavaScript. З TypeScript можна використовувати існуючий JavaScript-код, включати популярні бібліотеки JavaScript, і викликати TypeScript-код, згенерований з інших JavaScript. Оголошення типів для цих бібліотек поставляються разом з вихідним кодом. TypeScript являє собою розширення мови ECMAScript 5 та має наступні опції:

- анотації типів і перевірка їх узгодження на етапі компіляції;
- класи;
- інтерфейси;
- декоратори;
- узагальнене програмування;
- модулі;
- скорочений синтаксис анонімних стрілочних функцій або лямбда функцій;
- розширені можливості пошуку і параметри за замовчуванням;
- кортежі.

2.2.2 Програмна платформа Node.js, фреймворк Express.js та менеджер пакетів npm

Node або **Node.js** – це кросплатформенне середовище виконання з відкритим вихідним кодом, яка дозволяє розробникам створювати різні серверні інструменти і додатки використовуючи мову JavaScript. Середовище виконання призначена для використання поза контекстом браузера (тобто виконується безпосередньо на комп'ютері або на серверної ОС). Таким чином, середа виключає API-інтерфейси JavaScript для браузера і додає підтримку більш традиційних OS API-інтерфейсів, включаючи бібліотеки HTTP і файлових систем.

Раніше розподілений проект розробки Node.js керувався Node.js Foundation, і тепер він об’єднався з JS Foundation, щоб сформувати OpenJS Foundation, що сприяє програмі спільних проектів Linux Foundation [33].

Express.js, або просто Express – це внутрішня веб-програма для Node.js, випущена як безкоштовне програмне забезпечення з відкритим кодом під ліцензією MIT. Він призначений для створення веб-додатків та API. Його визначають як де-факто стандартною серверною структурою для Node.js. Представляє собою популярний веб-фреймворк, написаний на JavaScript і працює в середовищі виконання Node.js. Цей модуль освітлює деякі ключові переваги цього фреймворку, встановлює середовище розробки та виконує основні завдання веб-розробки. Express є внутрішнім компонентом популярних стеків розробки, таких як стек MEAN, MERN або MEVN, разом із програмним забезпеченням бази даних MongoDB та інтерфейсною структурою або бібліотекою JavaScript [34].

npm – це менеджер пакетів для Node.js. Він був створений у 2009 році як проект з відкритим кодом, щоб допомогти розробникам JavaScript легко ділитися упакованими модулями коду. npm є важливою частиною спільноти JavaScript і допомагає підтримувати одну з найбільших екосистем розробників у світі. Реєстр npm – це загальнодоступна колекція пакетів коду з відкритим

кодом для Node.js, інтерфейсних веб-програм, мобільних додатків, роботів, маршрутизаторів та безлічі інших потреб спільноти JavaScript. npm – це клієнт командного рядка, який дозволяє розробникам встановлювати та публікувати ці пакети [55].

2.2.3 Фреймворк Nest.js

За стратегію створення веб-сервісу було обрано фреймворк Nest, що надає готову масштабовану архітектуру для подальшої реалізації застосунків. Nest або **Nest.js** – це основа для створення ефективних, масштабованих серверних додатків Node.js. Він використовує прогресивний JavaScript, повністю підтримує та побудований на TypeScript і поєднує елементи OOP, FP та FRP. Під капотом Nest використовує надійний HTTP Server фреймворк Express [35].

Nest забезпечує рівень абстракції над цими загальними фреймворками Node.js, але також надає їх API безпосередньо розробнику. Це надає розробникам свободу використовувати безліч сторонніх модулів, доступних для базової платформи. Так, в останні роки, завдяки Node.js, JavaScript став популярним в Інтернеті як для фронтенд, так і для бекенд-додатків. Це породило чудові проекти, такі як Angular, React та Vue, які покращують продуктивність розробників і дозволяють створювати швидкі, масштабовані та розширювані фронтендні програми. Однак, хоча для Node існує безліч чудових бібліотек, помічників та інструментів, жоден з них ефективно не вирішує головну проблему – архітектуру. Nest надає нестандартну архітектуру додатків, яка дозволяє розробникам та командам створювати високотестовані, масштабовані, вільно пов'язані та легко обслуговуванні додатки. Архітектура сильно натхненна Angular.

Структура Nest.js складається з наступних елементів:

1. Контролери (Controllers). Контролери (рисунок 2.1) відповідають за обробку вхідних запитів та повернення відповідей клієнту. Призначення

контролера – отримувати конкретні запити на додаток. Механізм маршрутизації контролює, який контролер отримує які запити. Часто кожен контролер має більше одного маршруту, і різні маршрути можуть виконувати різні дії;

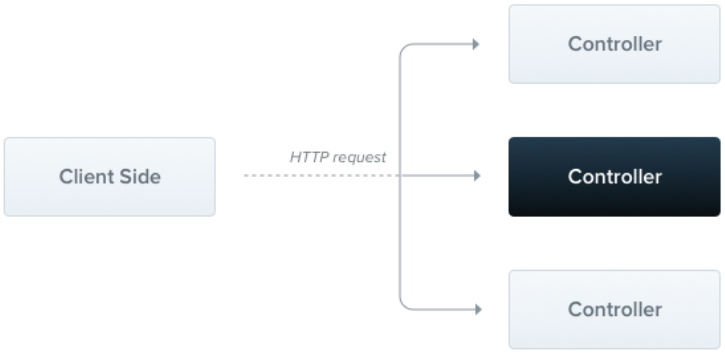

Рисунок 2.1 – Схема контролеру

2. Постачальники (Providers) – це основне поняття Nest (рисунок 2.2). Багато базових класів Nest можуть розглядатися як постачальник – послуги, сховища, фабрики, помічники тощо. Основна ідея провайдера полягає в тому, що він може вводити залежності – це означає, що об'єкти можуть створювати різні взаємозв'язки між собою, а функцію "підключення" екземплярів об'єктів можна в основному делегувати системі виконання Nest;

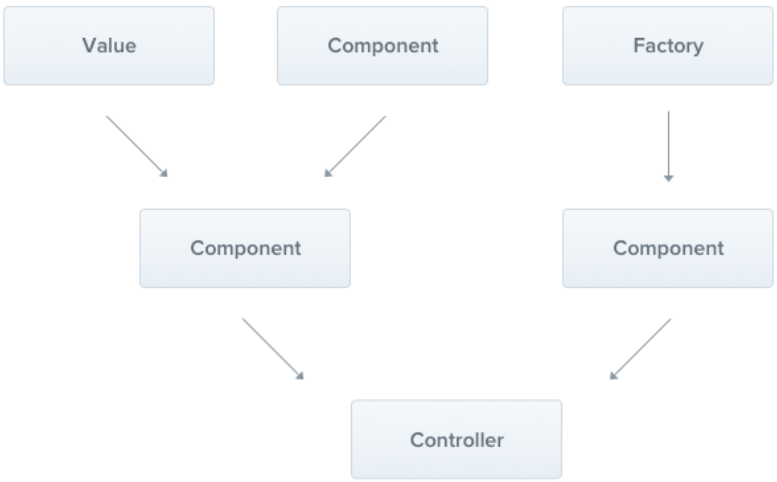

Рисунок 2.2 – Схема постачальника

3. Модулі (Modules) – це клас, котрий коментується декоратором @Module(). Декоратор @Module() надає метадані, які Nest використовує для

організації структури програми (рисунок 2.3). Кожен додаток має принаймні один модуль, кореневий модуль. Кореневий модуль є вихідною точкою, яку Nest використовує для побудови графіку програми – внутрішньої структури даних, яку Nest використовує для вирішення взаємозв'язків та залежностей модуля та постачальника;

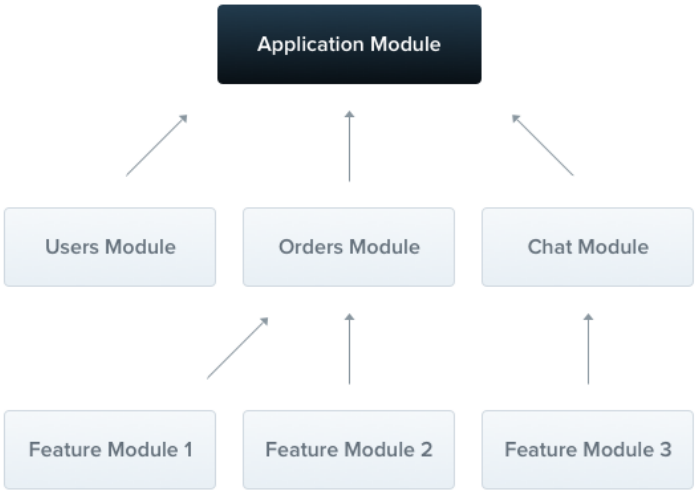

Рисунок 2.3 – Схема модуля

4. Проміжне програмне забезпечення (Middleware) – це функція, яка викликається перед обробником маршруту (рисунок 2.4). Функції проміжного програмного забезпечення мають доступ до об'єктів запиту, відповіді та next() функції проміжного програмного забезпечення у циклі запиту-відповіді програми. Наступна функція проміжного програмного забезпечення зазвичай позначається змінною з іменем next;

Рисунок 2.4 – Схема проміжного програмного забезпечення

5. Фільтри винятків (Exception filters). Nest постачається із вбудованим шаром винятків, який відповідає за обробку всіх необроблених винятків у програмі (рисунок 2.5). Коли виняток не обробляється кодом

програми, він перехоплюється цим шаром, який потім автоматично надсилає відповідну зручну відповідь;

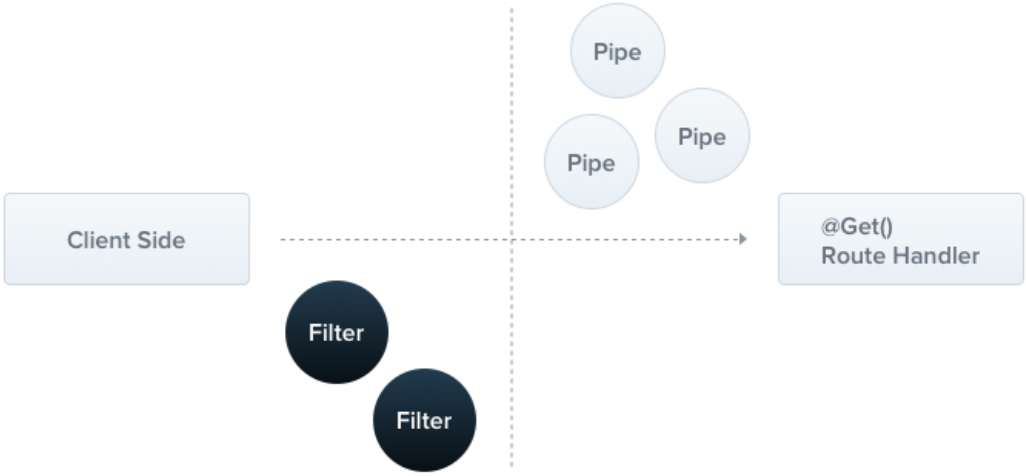

Рисунок 2.5 – Схема фільтрів винятку

6. Конвеєр (Pipes) – це клас, котрий коментується декоратором @Injectable(). Конвеєри повинні реалізовувати інтерфейс PipeTransform. Конвеєри мають два типові варіанти використання:

- перетворення: перетворення вхідних даних у бажану форму (наприклад, із рядка в ціле число) ;
- перевірка: оцінка вхідних даних і, якщо вони дійсні, просто передача їх без змін; в іншому випадку викид винятку, коли дані неправильні.

В обох випадках канали працюють на аргументах, що обробляються обробником маршрутизації контролера. Nest вставляє конвеєр безпосередньо перед викликом методу, і тоді конвеєр отримує аргументи, призначені для методу, і оперує ними. Будь-яка операція перетворення або перевірки відбувається в той час, після чого обробник маршруту викликається з будь-якими потенційно перетвореними аргументами.

7. Страж (Guards) – це клас, котрий коментується декоратором @Injectable(). Стражі повинні реалізувати інтерфейс CanActivate (рисунок 2.6). Стражі несуть єдину відповідальність. Вони визначають, чи буде обробляти

даний запит обробник маршруту чи ні, залежно від певних умов, наприклад, дозволів, ролей, ACL тощо, наявних під час виконання. Авторизація, як правило, обробляється проміжним програмним забезпеченням у традиційних програмах Express.

Рисунок 2.6 – Схема стражів

8. Перехоплювач (Interceptors) – це клас, котрий коментується декоратором @Injectable(). Перехоплювачі повинні реалізовувати інтерфейс NestInterceptor (рисунок 2.7). Перехоплювачі мають набір корисних можливостей, натхненних технікою аспектно-орієнтованого програмування (AOP). Вони дають можливість:

- прив’язати додаткову логіку до/після виконання методу;
- перетворити результат, повернутий із функції;
- перетворити виняток, викинутий із функції;
- розширити основну функцію поведінки;
- повністю замінити функцію залежно від конкретних умов (наприклад, для кешування).

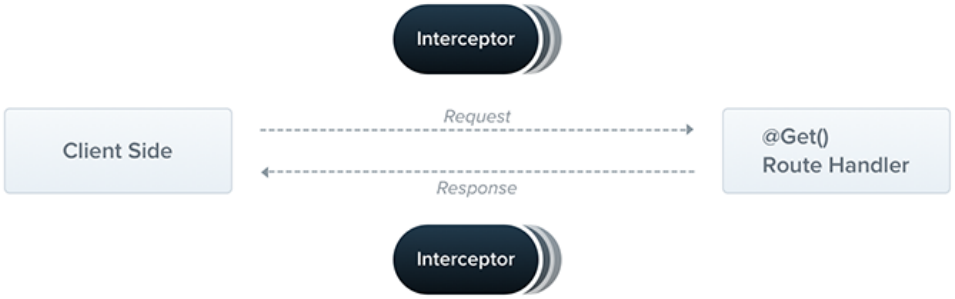

Рисунок 2.7 – Схема перехоплювачів

Data Transfer Object, або об’єкт передачі даних (DTO) - один з шаблонів проектування, використовується для передачі даних між підсистемами додатки.

Entities, або сутність - це клас, який відображається у колекції. Визначається декоратором @Entity().

2.2.4 NoSQL база даних MongoDb та сервіс MongoDb Atlas

MongoDB – це доступна для джерел міжплатформена документ-орієнтована база даних. Класифікується як програма баз даних NoSQL, MongoDB використовує подібні до JSON документи з необов'язковими схемами. MongoDB розроблено компанією MongoDB Inc. та ліцензовано під загальною ліцензією на сторону сервера (SSPL) [35]. Основні особливості:

- Спеціальні запити. MongoDB підтримує пошук за полями, діапазонами та регулярними виразами. Запити можуть повертати певні поля документів, а також включати визначені користувачем функції JavaScript;
- Індексація. Поля в документі MongoDB можна індексувати за допомогою первинних та вторинних індексів;
- Реплікація MongoDB забезпечує високу доступність наборів реплік. Набір реплік складається з двох або більше копій даних. Кожен член набору реплік може виступати в ролі первинної або вторинної репліки в будь-який час;
- Балансування навантаження. MongoDB масштабується горизонтально. Користувач вибирає ключ-фрагмент, який визначає спосіб розподілу даних у колекції. MongoDB може працювати на декількох серверах, балансуючи навантаження або продублюючи дані, щоб підтримувати роботу системи в разі несправності обладнання;
- Зберігання файлів. MongoDB може використовуватися як файлова система, що називається GridFS, із функціями балансування навантаження та реплікації даних на декількох машинах для зберігання файлів. Ця функція, яка називається сітковою файловою системою, входить до складу драйверів MongoDB. MongoDB надає розробникам функції для обробки

файлів та вмісту. Доступ до GridFS можна отримати за допомогою утиліти mongofiles або плагінів для Nginx та lighttpd. GridFS ділить файл на частини або фрагменти та зберігає кожен із цих фрагментів як окремий документ;

- Агрегація. MongoDB пропонує три способи виконання агрегації: конвеєр агрегування, функція зменшення та одноцільові методи агрегування;
- Виконання JavaScript на стороні сервера. JavaScript можна використовувати в запитах, функціях агрегування (наприклад, MapReduce) і надсилати безпосередньо до бази даних для виконання;
- Обмежені колекції. MongoDB підтримує колекції фіксованого розміру, які називаються обмеженими колекціями. Цей тип колекції підтримує порядок вставки та, як тільки досягнуто вказаного розміру, діє як черга.

Є офіційні драйвери для основних мов програмування (C, C ++, C#, Go, Java, Node.js, Perl, PHP, Python, Ruby, Rust, Scala, Swift). Існує також велика кількість неофіційних або підтримуваних спільнотою драйверів для інших мов програмування і фреймворків.

Основним інтерфейсом до бази даних була командна оболонка «mongo». Існують продукти і сторонні проекти, які пропонують інструменти з GUI для адміністрування і перегляду даних.

MongoDB Atlas – це глобальний хмарний сервіс баз даних для додатків. За допомогою MongoDB Atlas можна розгорнути керовану базу даних MongoDB на таких хмарних сервісах, як AWS, Azure або GCP. З MongoDB Atlas бази даних створюються швидше, і менше витрачається часу на їх управління [37].

MongoDB Atlas – це платформа MongoDB Database-as-a-Service, це означає, що сервіс автоматично налаштовує і розміщує базу даних, а єдине, що потрібно зробити користувачеві – це заповнити базу даних вмістом. MongoDB

Atlas знімає з плеч клієнтів навантаження з управління базами NoSQL і дає сфокусуватися на додатках.

Налаштування MongoDB Atlas та створений кластер можна побачити на рисунку 2.8.

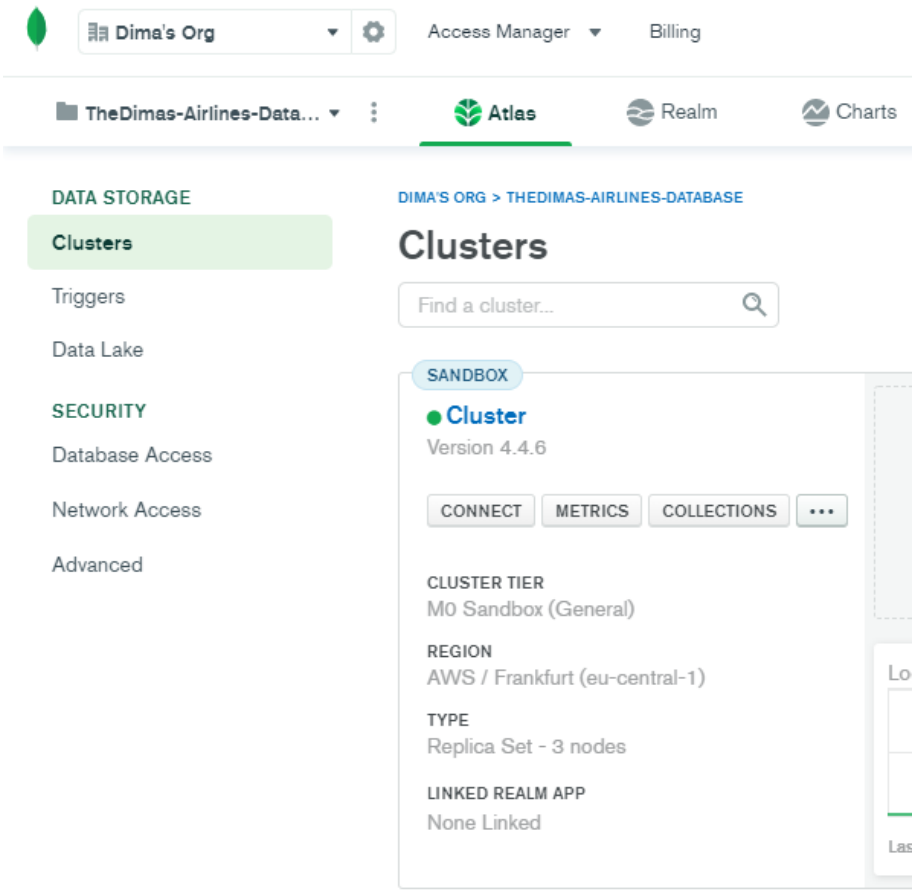

Рисунок 2.8 – Конфігурація кластера

Для виконання дипломної роботи було створено наступні колекції – авіалінії (airlines), літаки (airplanes), аеропорти (airports), рейси (flights), місця (seats), секції (sections), квитки (tickets), користувачі (users). Графічне зображення усіх колекцій можна побачити на рисунку 2.9.

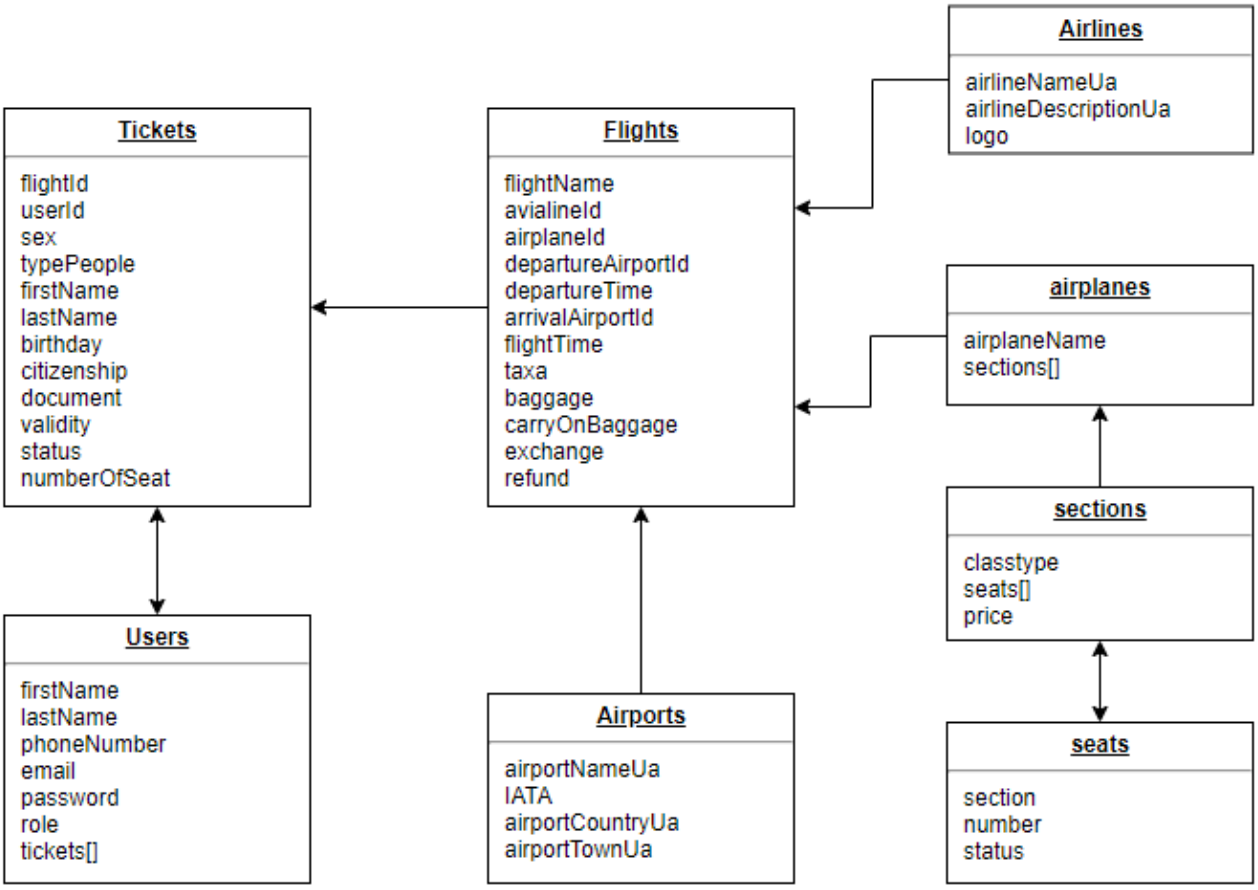

Рисунок 2.9 – Структура БД

Кожна з колекцій має власні ключі, завдяки яким, наприклад, можливо показати всі квитки одного користувача або доступні рейси за певну дату. Вся інформація представлена у вигляді JSON-подібних документах (рисунок 2.10) – BSON (binary JSON). Дані в цих колекціях можливо додавати, видаляти або редагувати. Також в кожній колекції (рисунок 2.11) набір іменованих ключів, які, наприклад, зберігають корисну інформацію починаючи від імені користувача та закінчуючи поштою користувача.


```
_id: ObjectId("60a252a2e0eb1b26d8672661")
airlineNameUa: "МАУ (Міжнародні Авіалінії України)"
descriptionUa: "Міжнародні Авіалінії України працюють з 1 жовтня 1992 року і є флагман..."
logo: "airline/1d83ded3-583c-4763-9ded-b06be95da4e9.png"
__v: 0
```


Рисунок 2.10 – Приклад одного з документів колекції

Додатково було реалізовано для колекцій аеропортів та авіаліній можливі ключі для різних мов, а саме російську та англійську мови.

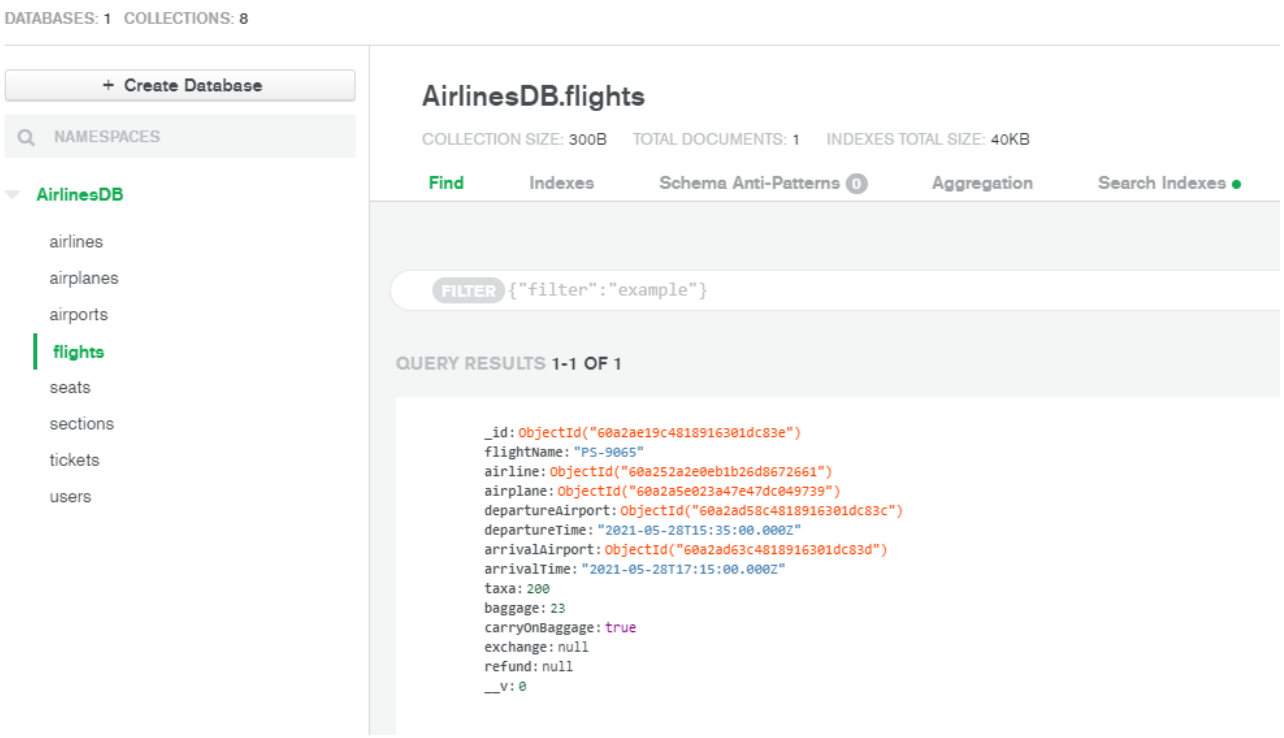

Рисунок 2.11 – Вигляд БД та колекції рейсів

Для розробки веб-сервісу та полегшення взаємодії з БД MongoDB додатково використовується бібліотека **Mongoose** – це інструмент об'єктного моделювання MongoDB, призначений для роботи в асинхронному середовищі. Mongoose підтримує як обіцянки (promises), так і зворотні дзвінки (callbacks) [49].

2.2.5 Налаштування веб-сервісу

Для розробки веб-сервісу в фреймворку Nest.js на мові TypeScript було використані модульний пакет webpack, cross-env, транскопілятор Babel, набір визначень та декораторів TypeScript та Nest.js.

Babel – це безкоштовний транскопілятор JavaScript із відкритим кодом, який в основному використовується для компіляції коду ECMAScript 2015+ (ES6+) у сумісну для більшості браузерів версію JavaScript та яку можуть запускати старіші механізми JavaScript. Babel – це популярний інструмент для використання новітніх функцій мови програмування JavaScript [38].

Плагіни Babel використовуються для перетворення синтаксису, який не підтримується широко, у версію, сумісну із зворотною стороною. Нестандартний синтаксис JavaScript, такий як JSX, також може бути перетворений.

webpack – це інструмент, що дозволяє скомпілювати, JavaScript модулі в єдиний JS-файл. webpack також відомий як збирач модулів. При великій кількості файлів він створює один об'ємний файл (або кілька файлів) для запуску вашого застосування. Він також здатний виконувати безліч інших операцій:

- допомагає зібрати воєдино усі ресурси;
- стежить за змінами і повторно виконує завдання;
- може виконати транспіляцію JavaScript наступного покоління до старішого стандарту JavaScript (ES5) за допомогою Babel;
- може конвертувати вбудовані зображення в data URI;
- може запустити webpack-dev-server ;
- може працювати з Hot Module Replacement (заміна гарячого модуля);
- може розділити вихідний файл (output file) на кілька файлів, щоб уникнути повільного завантаження сторінки через великого розміру JS-файлу;
- може виконати Tree Shaking.

Webpack не обмежується одним лише фронтенда, його також успішно застосовують в бекенд розробці на Node.js.

Додатки часто працюють в різних середовищах. Залежно від середовища слід використовувати різні налаштування конфігурації. Оскільки конфігураційні змінні змінюються, найкращою практикою є зберігання змінних конфігурації у середовищі. Хорошим підходом для використання цієї техніки в Nest є пакет **cross-env** – це набір модулів, що запускає сценарії, які встановлюють і використовують змінні середовища на різних платформах.

2.2.6 Написання та рефакторинг коду

Для забезпечення забезпечення чистоти, читаності, простоти та повторного використання коду був використаний лінтер **ESLint**. Цей інструмент аналізує код для того, щоб допомогти виявити проблемні патерни, які не відповідають правилам і стандартам. Працює він для більшості мов програмування [41].

Лінтінг – це вид статичного аналізу коду, який часто використовують для знаходження проблемних патернів проектування або коду, якого слід уникати за певними посібникам зі стилю.

Так як JavaScript – це динамічний мову програмування зі слабкою типізацією, код, написаний на ньому, схильний до помилок, які допускають розробники. JavaScript – інтерпретована мова, тому синтаксичні та інші помилки в коді зазвичай виявляються тільки після запуску цього коду. Лінтера, на зразок ESLint, дозволяють розробникам знаходити проблеми в коді не запускаючи його.

Додатково для автоматичного редагування коду було використано **Prettier** – це засіб для форматування коду, яке націлений на використання жорстко заданих правил по оформленню програм [40].

Ось які можливості і особливості Prettier дозволяють говорити про корисність цього інструменту:

- Приведення в порядок існуючої кодової бази. Подібне, за допомогою Prettier, можна виконати буквально однією командою. Ручна обробка великих обсягів коду займе набагато більше часу.
- Prettier легко впровадити. Prettier використовує «усереднений», найменш спірний підхід до стилю при форматуванні коду.
- Prettier дозволяє зосередитися на написанні коду, а не на його форматуванні. Багато хто просто не усвідомлюють того, як багато часу і сил витрачається на форматування коду. Використання Prettier дозволяє не думати про форматування, а займатися замість цього програмуванням.

- Prettier досить легкий в розумінні та освоєнні.

Весь код було розроблено в редакторі **Visual Studio Code** – це редактор вихідного коду, розроблений Microsoft для Windows, Linux і macOS. Studio Code спроектований на Electron і реалізується через веб-редактор Monaco, розроблений для Visual Studio Online [56].

Visual Studio Code відмінний вибір для початківця програміста, має необхідний мінімум:

- непогану документацію;
- автодоповнення коду (з використанням IntelliSense);
- підсвічування синтаксису;
- вбудований відладчик;
- розширення функціоналу за рахунок плагінів;
- управління системою контролю версій git;
- багатоплатформний;
- безкоштовний, з відкритим вихідним кодом.

Такий редактор адаптований для веб-розробки і цілком підходить для серйозних проектів як основний інструмент редагування коду.

2.2.7 Забезпечення безпеки веб-сервісу

Для забезпечення потужної безпеки веб-сервісу були здійснені наступні заходи та виконанні наступні пункти:

- аутентифікація;
- авторизація;
- хешування;
- а також налаштування додаткових модулів та додатків.

Аутентифікація є важливою частиною більшості програм. Існує багато різних підходів та стратегій для обробки автентифікації. Підхід цього проекту представлено аутентифікацією на основі стратегії бібліотеки Passport.js.

Passport.js – це найпопулярніша бібліотека автентифікації node.js, добре відома спільнотою та успішно використовується у багатьох виробничих додатках [42].На високому рівні Passport виконує ряд кроків:

- аутентифікація користувача, перевіривши його "облікові дані" – (ім'я користувача/пароль, JSON (JWT);
- керування станом автентифікації;
- додання інформації про автентифікованого користувача до об’єкта запиту для подальшого використання в обробниках маршрутів.

JSON Web Token (JWT) – це запропонований стандарт Інтернету для створення даних з необов’язковим підписом та/або необов’язковим шифруванням, корисне навантаження якого містить JSON, який заявляє певну кількість претензій (рисунок 2.12). Токени підписуються або за допомогою приватного секрету, або відкритого/приватного ключа. Веб-токени JSON можуть містити стан сеансу. JWT спирається на інші стандарти, засновані на JSON: веб-підпис JSON та веб-кодування JSON [45].

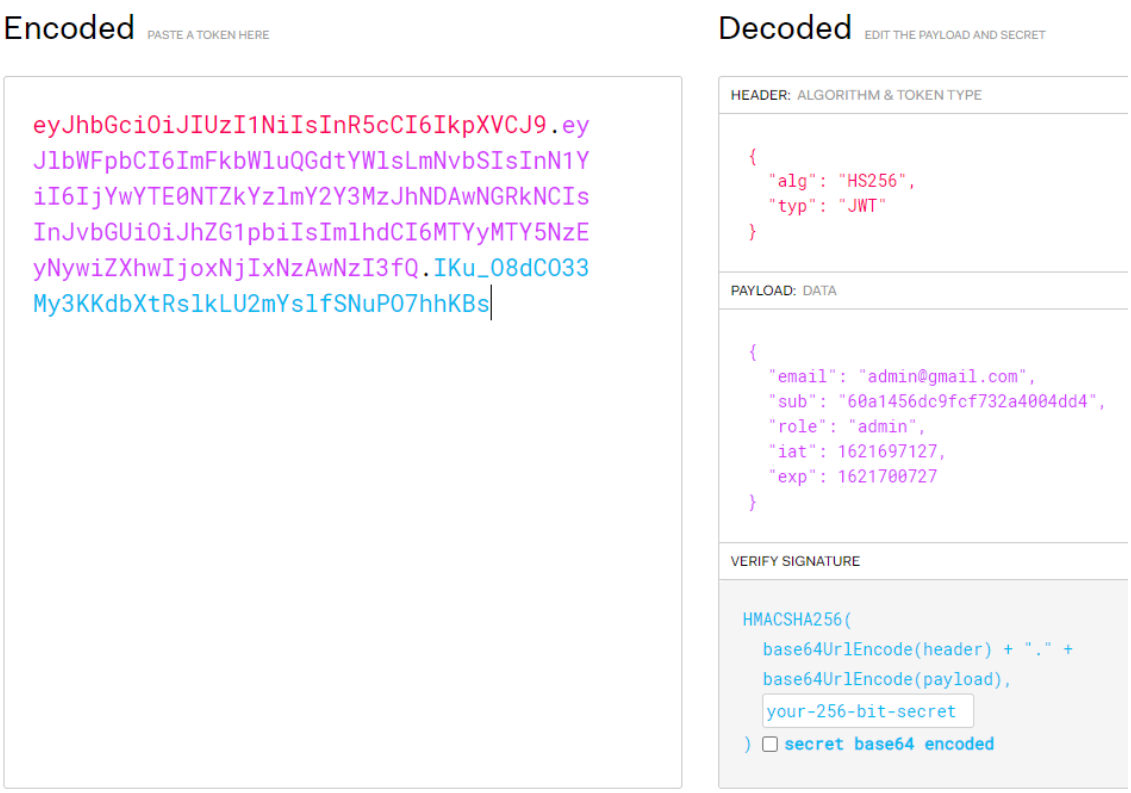

Рисунок 2.12 – Загальний вигляд токену

Авторизація відноситься до процесу, який визначає, що може зробити користувач. Авторизація є ортогональною і незалежною від автентифікації. Однак ауторизація вимагає механізму автентифікації. Обраний підхід в даній дипломній роботі представлений RBAC, або рольовий контроль доступу.

RBAC (Role-based access control) – це нейтральний до політики механізм контролю доступу, визначений навколо ролей та привілеїв.

Хешування - це процес перетворення заданого ключа в інше значення. Хеш-функція використовується для генерації нового значення відповідно до математичного алгоритму. Після того, як було зроблено хешування, перехід від вихідного сигналу до вхідного може бути неможливим. Для реалізації алгоритму хешування було використано бібліотеку **bcrypt.js** [43].

Helmet (Шолом) захищає додаток від деяких відомих веб-вразливостей, встановивши належним чином заголовки HTTP. Як правило, Helmet – це лише сукупність 14 менших функцій проміжного програмного забезпечення, які встановлюють заголовки HTTP, пов’язані з безпекою [44].

CORS (Cross-origin resource sharing), або спільний доступ до ресурсів – це механізм, який дозволяє запитувати ресурси з іншого домену.

Cookies, або кукі-файл HTTP – це невеликий фрагмент даних, що зберігається в браузері користувача. Файли cookie були розроблені, щоб стати надійним механізмом з запам'ятовування інформацію для веб-сайтів, що містить інформацію. Коли користувач знову відвідує веб-сайт, файл cookie автоматично надсилається разом із запитом.

Поширеним методом захисту додатків від атак грубої сили є обмеження швидкості. Цей метод представлений пакетом **throttler**.

2.2.8 Оптимізація архітектури та контролерів

Для оптимізації архітектури та роботи контролерів було використано наступні заходи:

- кешування запитів;

- оптимізація запитів;
- перевірка вхідних та вихідних даних;
- заміна компілятора на гарячий модуль.

Кешування – це чудовий і простий прийом, який допомагає покращити продуктивність вашого додатка. Він діє як тимчасове сховище даних, що забезпечує високопродуктивний доступ до даних.

Стиснення може значно зменшити розмір тіла відповіді, тим самим збільшуючи швидкість веб-програми.

Найкращою практикою є **перевірка правильності** будь-яких даних, що надсилаються у веб-програму. Для автоматичної перевірки вхідних запитів Nest пропонує ValidationPipe. ValidationPipe використовує потужний пакет перевірки класів та його декоративні декоратори перевірки. ValidationPipe забезпечує зручний підхід до забезпечення правил перевірки для всіх вхідних корисних навантажень клієнта, де конкретні правила оголошуються простими анотаціями в деклараціях локального класу/DTO в кожному модулі. Для забезпечення коректної роботи цього конвеєру було використано бібліотеки **class-validator** та **class-transformer** [47].

Додатково, для оптимізації розробки веб-сервісу було використано **webpack HMR** (Hot-Module Replacement) – це заміна гарячого модуля, що полегшує перекомпілювання проекту. Це значно зменшує кількість часу, необхідного для створення екземпляра програми, та значно полегшує ітеративну розробку [39].

2.2.9 Бібліотеки для полегшення розробки веб-сервісу

Додатково було використано бібліотеки Fuse.js, Moment.js та пакет uuid.

Fuse.js – це потужна, легка бібліотека нечіткого пошуку з нульовими залежностями. Ця бібліотека надає можливості нечіткого пошуку (більш формально відомий як приблизне узгодження рядків) – це техніка пошуку рядків, які приблизно дорівнюють заданому шаблону, але не точно [51].

Moment.js – це бібліотека для аналізу, перевірки та маніпуляції, і відображення дати та час у JavaScript [52].

UUID (Universally unique identifier), або універсальний унікальний ідентифікатор – це стандарт, що використовується для ідентифікації в комп'ютерних системах. UUID можна використовувати в якості первинних ключів в базах даних, як унікальних імен завантажуваних файлів так й будь-яких веб-джерел. Хоча ймовірність того, що UUID буде продубльовано, не дорівнює нулю, вона досить близька до нуля. Багато обчислювальних платформ забезпечують підтримку генерування та аналізу їх текстового представлення. Пакет **uuid** надає доступ до цього стандарту ідентифікації [53].

Для обробки завантаження файлів Nest надає вбудований модуль на основі багатопрофільного проміжного програмного забезпечення для Express. Так, **Multer** обробляє дані, розміщені у форматі multipart/form-data, які в основному використовується для завантаження файлів через запит HTTP POST.

Для того, щоб обслуговувати статичний вміст, як Single Page Application (SPA), було використано ServeStaticModule із пакета **serve-static**.

2.3 Функціональні можливості веб-сервісу

Для вирішення завдань проектованого веб-сервісу було проаналізовано декілька якісних наявних веб-сервісів з можливостями пошуку та бронювання авіа білетів та виділено функціонал, який найкраще підходить для проектованої системи, не ускладнює роботу користувачу та не зменшує продуктивність всього веб-сервісу.

Таблиця 2.1

Функціонал веб-сервісу пошуку та бронювання авіа білетів для користувача

Назва функціоналу	Опис функціоналу
Реєстрація користувача	Функція, яка дозволяє створювати облікові записи для кожного користувача.

Продовження таблиці 2.1

Бронювання квитка	Функція, яка дозволяє забронювати квиток з обраними послугами та знижкою.
Повернення квитка	Функція, яка дозволяє повернути квиток та відновити куплене місце для повторної купівлі.
Пошук квитка	Функція, яка дозволяє віднайти за заданими параметрами потрібний квиток.
Зміна персональних даних	Функція, яка дозволяє змінити пошту, пароль та ім’я для унікальної ідентифікації користувача.
Авторизація користувача	Для можливості ідентифікування користувачів та збереженням їх даних на сервері кожен користувач повинен пройти авторизації.

Це є головний функціонал, що вимагається від будь-якого веб-сервісу по пошуку та бронюванню авіа білетів. Він дозволить без труднощів шукати та забронювати квиток для користувача, та забезпечить надійну роботу адміністратора з базою даних.

Таблиця 2.2

Функціонал веб-застосунку купівлі залізничних квитків для адміністратора

Назва функціоналу	Опис функціоналу
Перегляд зареєстрованих користувачів	Функція, яка дозволяє слідкувати за зареєстрованими користувачами.
Перегляд продажів	Функція, яка дозволяє переглянути продажі квитків.
Перегляд, додавання, зміна та видалення для кожного елементу рейсу	Функція, яка дозволяє переглянути, додати, змінити чи видалити певний елемент рейсу.

2.4 Тестування веб-сервісу

Для тестування веб-сервісу було використано Postman – це платформа для тестування розробки API. Postman спрощує кожен крок побудови API та впорядковує співпрацю, щоб допомагає створювати кращі API – швидше. За допомогою розширення можна складати і редагувати прості або складні HTTP-запити [46]. Приклад тестування запиту можна побачити на рисунку 2.13.

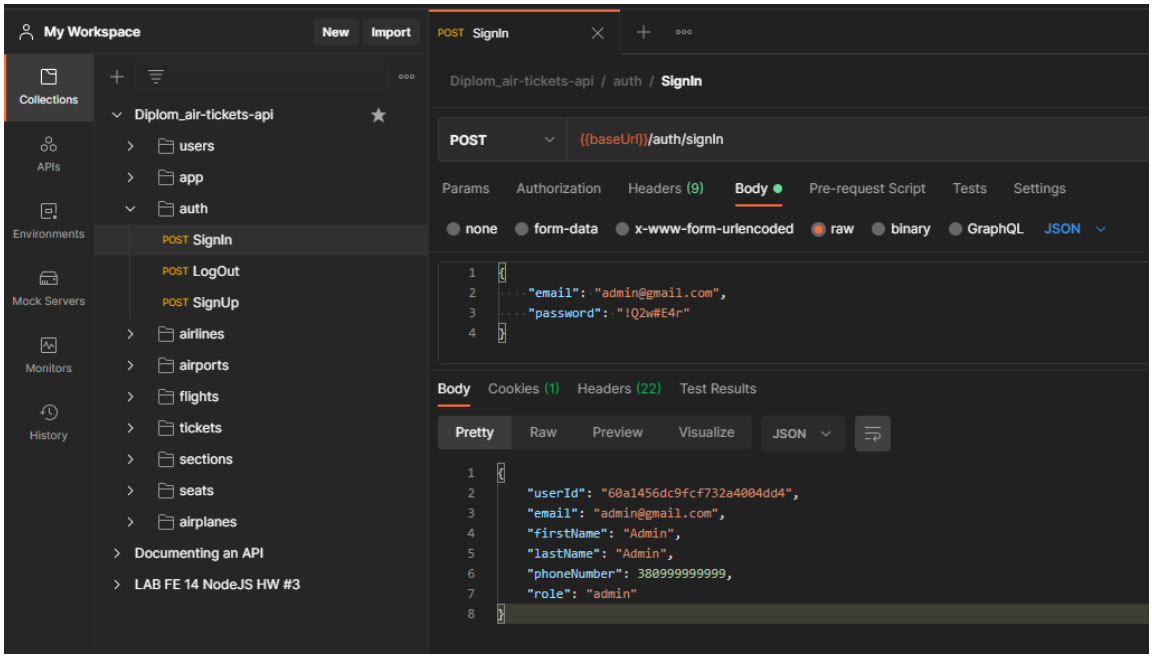

Рисунок 2.13 – Приклад тестування авторизації веб-сервісу

Postman надає доступ до повного налаштування запитів. Так, ми можемо бачити заголовки (рисунок 2.14), тіло (рисунок 2.15) запитів та відповідей та маємо можливість переглядати cookies (рисунок 2.16).

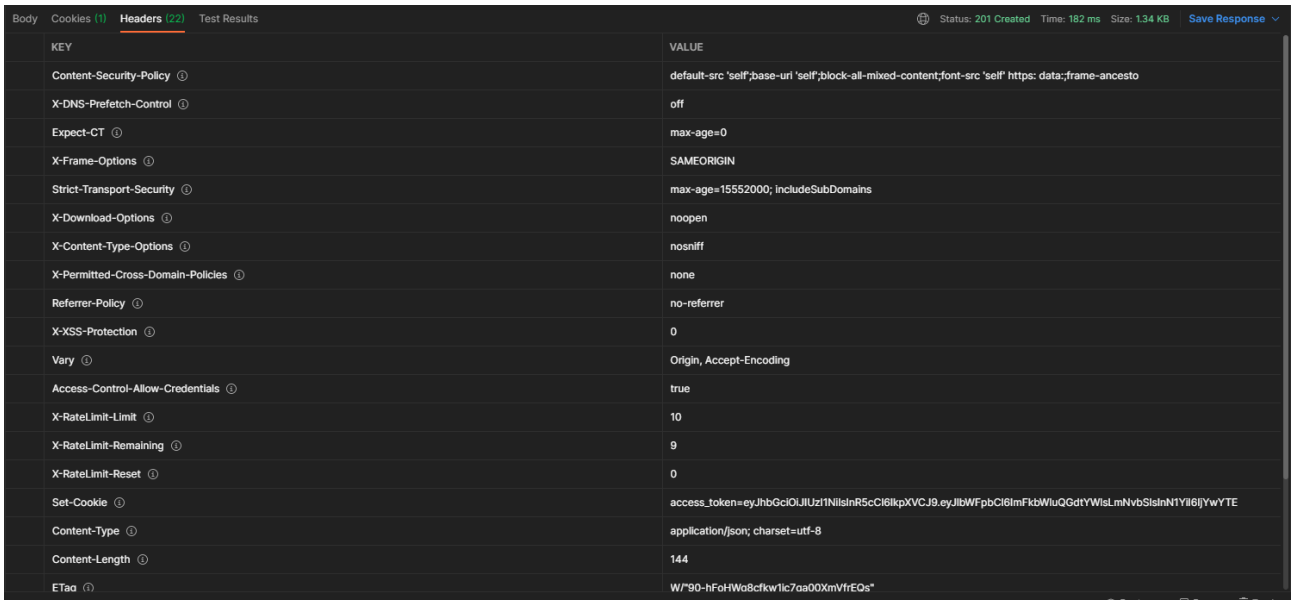

Рисунок 2.14 – Перегляд заголовків запитів

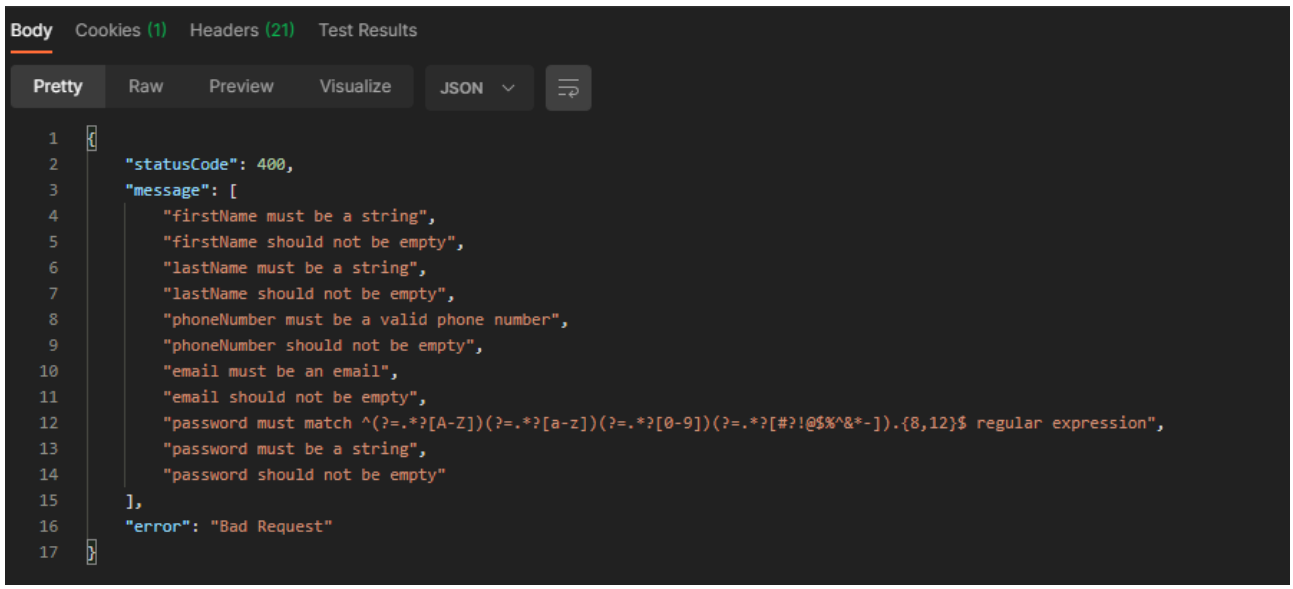

Рисунок 2.15 – Перегляд тіла відповіді

Рисунок 2.16 – Перегляд cookies

Висновки

Виконавши аналіз вхідної інформації стало відомо особливості розробки веб-сервісу. Зважаючи на це було створено стратегію веб-сервісу з реалізації архітектури Rest.

Було описано усі використані програмні засоби при розробці веб-сервісу. Так, було обрано мову програмування JavaScript та TypeScript, БД MongoDB та СКБД MongoDB Atlas, обгортку Nest.js та фреймворк Express до платформи Node.js.

Для створення динамічного ресурсу, який можливо оновлювати та додавати нову інформацію було використано додаткові бібліотеки та пакети, які сприяють полегшенню розробці.

Для покращення функціоналу застосунку було проведено роботи з встановленням додатків які відповідають за рівень безпеки, оптимізація та рефакторинг коду та структури веб-сервісу, регулярне створення резервної копії веб-застосунку та інше.

Для перевірки виконаних робіт було проведено тестування веб-застосунку з підключенням інших розробників та користувачів, що дозволило виправити отримані помилки та запобігти їх виникнення в майбутньому.

Розробка веб-сервісу з потужним функціоналом потребує якісної підготовки та чіткого плану по виконанню всіх робіт, які стосуються створення прототипу застосунку.

РОЗДІЛ 3: ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ ПОШУКУ ТА БРОНЮВАННЯ АВІА БІЛЕТІВ

В розділі 3 проведено аналіз основного функціоналу веб-сервісу із пошуку та бронювання авіа білетів. Додатково проведено розбір адміністрування веб-сервісу, що дозволить оцінити доступність, зрозумілість та простоту в його використанні для адміністратора.

3.1 Документація OpenAPI веб-сервісу

Специфікація OpenAPI – це мовно-агностичний формат визначення (рисунок 3.1), який використовується для опису RESTful API. Nest пропонує спеціальний модуль, який дозволяє генерувати таку специфікацію, використовуючи декоратори [50].

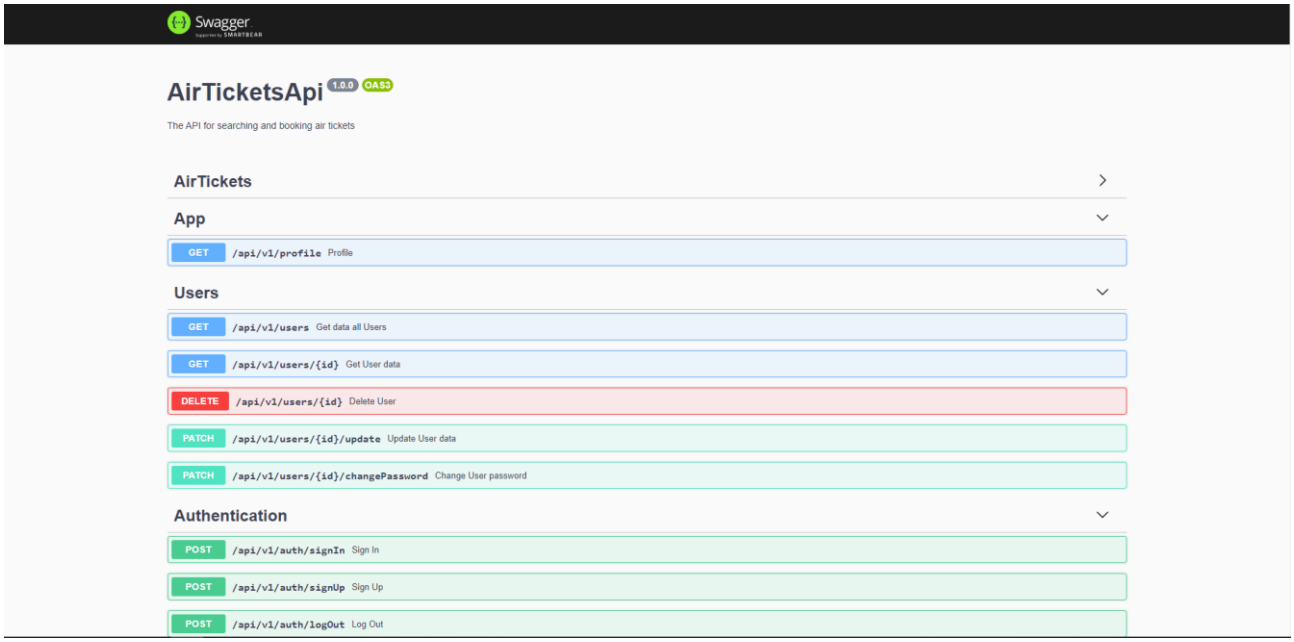

Рисунок 3.1 – Swagger UI

The OpenAPI Specification – це формалізована специфікація і екосистема безлічі інструментів, що надає інтерфейс між front-end системами, кодом бібліотек низького рівня і комерційними рішеннями у вигляді API. Разом з тим,

специфікація побудована таким чином, що не залежить від мов програмування, і зручна у використанні як людиною, так і машиною.

Щодо призначення, OpenAPI розглядається як універсальний інтерфейс для користувачів (клієнтів) по взаємодії з сервісами (серверами). Якщо спроектована специфікація для деякого сервісу, то на її підставі можна генерувати вихідний код для бібліотек клієнтських додатків, текстову документацію для користувачів, варіанти тестування та ін. Для цих дій є великий набір інструментів для різних мов програмування і платформ.

Ця специфікація надає можливість працювати з інтерфейсами напряму (рисунок 3.2). Наприклад, можно спробувати різні методи представлені в специфікації.

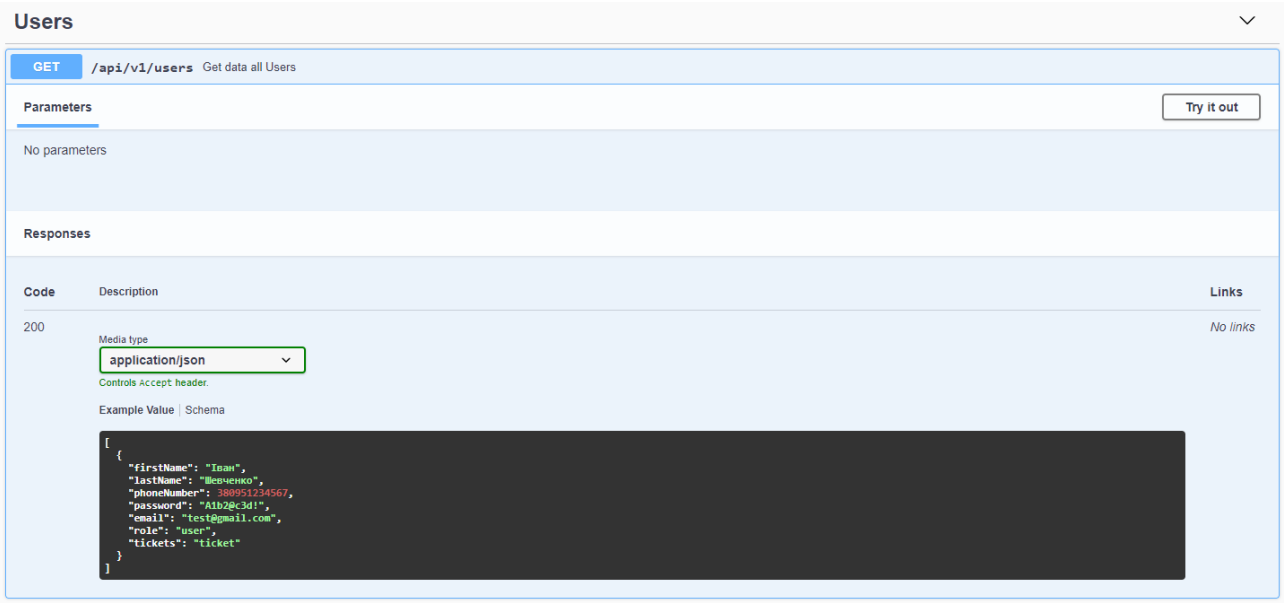

Рисунок 3.2 – Приклад опису HHTP-методу

OpenAPI є формалізованою специфікацію і повноцінним фреймворком для опису, створення, використання і візуалізації веб-сервісів REST. Її завданням є дозвіл клієнтським системам і документації синхронізувати свої оновлення зі змінами на сервері. Це досягається тим, що методи, параметри, моделі та інші елементи за допомогою OpenAPI інтегруються з програмним забезпеченням сервера і весь час з ним синхронізуються. Специфікація не залежить від мови програмування, і може бути використана поза протоколом

HTTP. OpenAPI одночасно застосовується для клієнта, сервера і відповідної документації інтерфейсу, створеного відповідно до REST.

Специфікація декларативна, і тому може бути використана клієнтами без знань особливостей серверної реалізації. При цьому працювати з OpenAPI можуть як розробники, так і рядові користувачі через готові інструменти і надані інтерфейси. Як формат використовується JSON, але в загальному випадку може бути обрана і інша мова розмітки. Детальний опис вхідних, вихідних даних, можливих відповідей, помилок та виключень значно полегшує роботу з дослідження інтерфейсів (рисунок 3.3).

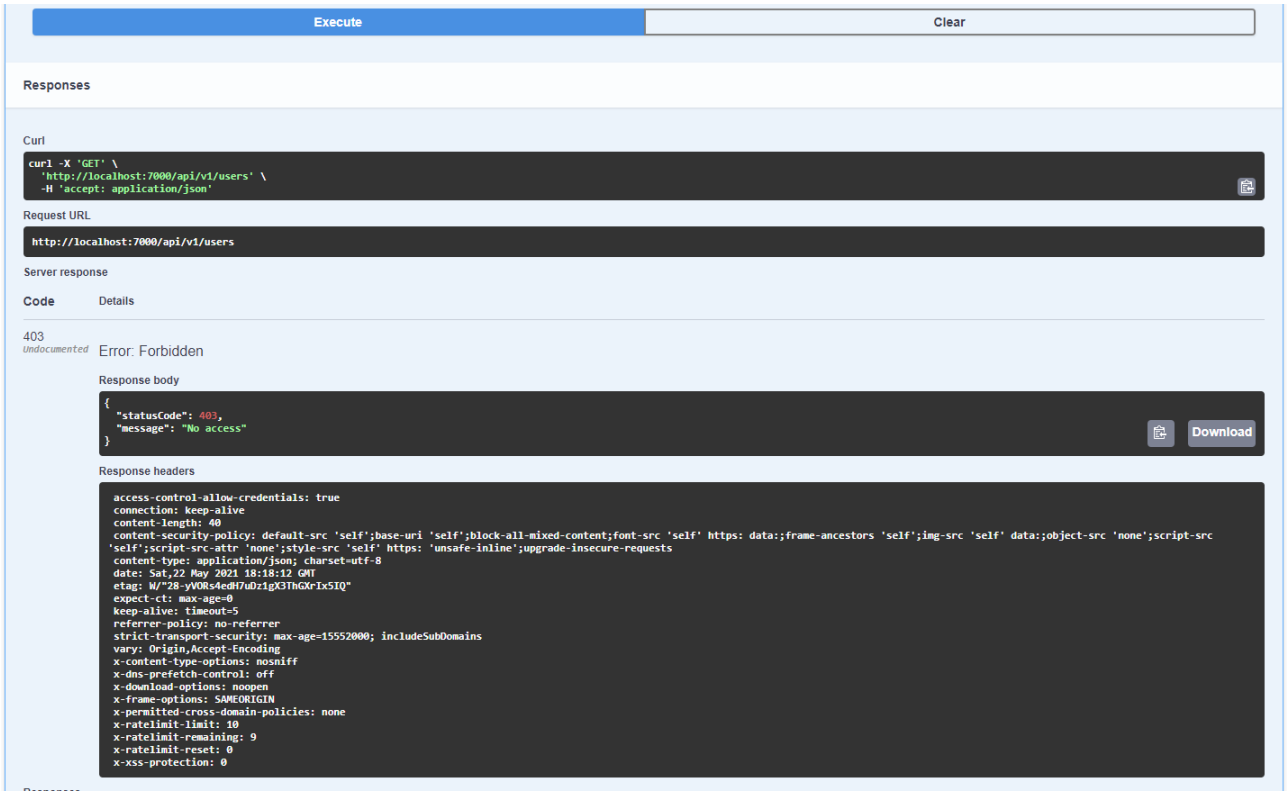

Рисунок 3.3 – Результат виконання HHTP-методу

Також, Swagger надає додаткові можливості, які дозволяють копіювати, редагувати та завантажувати методи. Усього було створено методи для:

- створення, редагування, видалення та отримання відповідної інформації користувачів, літаків, секції та авіаліній;
- зміна поточного статусу сидіння;
- пошук, створення, оновлення та видалення рейсів;

- реєстрації, входу та виходу;
- створення квитка, зміни статусу, оновлення, отримання інформації та видалення.

Також можна побачити усі наявні сутності та DTO (рисунок 3.4).

Рисунок 3.4 – Схеми веб-сервісу

OpenAPI надає доступ до кожної схеми та представляє інформацію про кожні поля, ключі, обмеження та приклади (рисунок 3.5).

Flight {

flightName*	string *example: PS-9065* Flight name
airline*	> {...} *example: UIA (Ukraine International Airlines)*
airplane*	> {...} *example: Aerospatiale/Alenia ATR 72*
departureAirport*	> {...} *example: Zhulyani/Жуляни/IEV*
departureTime*	string *example: 2021-05-16T15:35:00.000Z* Departure time
arrivalAirport*	> {...} *example: Sheremetyevo/Шереметьево/SVO*
arrivalTime*	string *example: 2021-05-21T09:15:00.000Z* Arrival time
taxa*	number *example: 200* Ticket taxa
baggage*	number *example: 23* Baggage properties
carryOnBaggage*	boolean *example: true* Hand luggage
exchange*	number *example: 0.7* Ticket exchange
refund*	number *example: 0.5* Ticket return

}

Рисунок 3.5 – Структура схеми

3.2 Документація Compodoc веб-сервісу

Compodoc − це інструмент документації для Angular додатків. Оскільки Nest та Angular мають схожі структури проектів та коду, Compodoc також працює з додатками Nest [54].

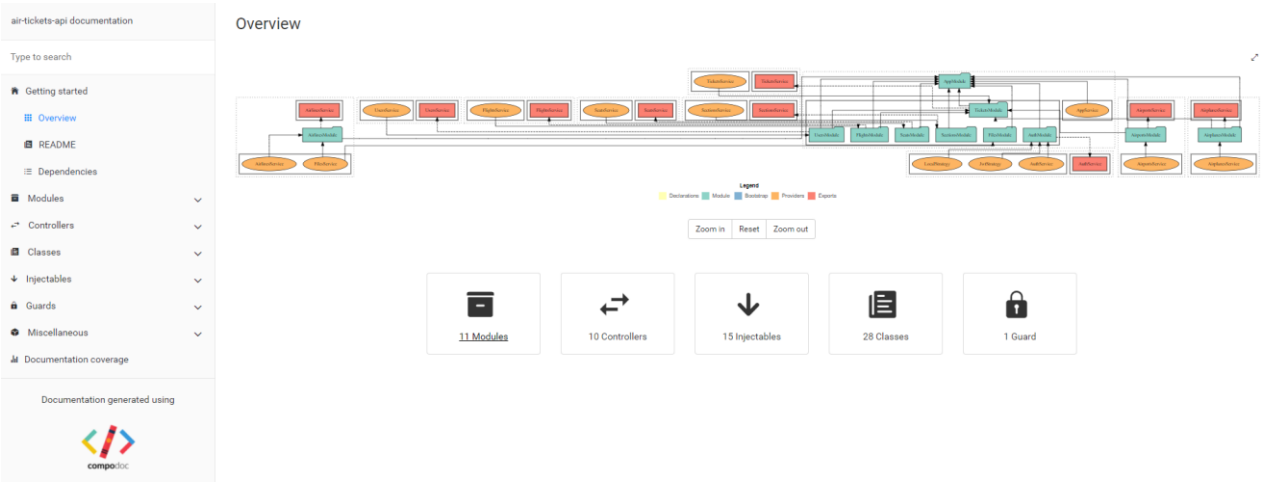

Рисунок 3.6 – Головна сторінка Compodoc

Compodoc генерує статичну документацію до додатку. Переваги та функції:

- чистий, простой дизайн;
- красиві теми;
- наявний пошук включає в себе потужний пошуковий движок для швидкого пошуку того, що ви шукаєте;
- автоматичне оглавлення – це оглавлення API, що генерується з використанням елементів, найдених в розроблених файлах;
- легка підтримка JSDoc – це підтримка тегів @param, @returns, @link, @ignore і @example;
- покриття документації – це отримання звіту про покриття документації проекту;
- документація генерується повністю в автономному режимі;
- відкритий вихідний код і на npm.

Загальний вид контролеру, де можна побачити шлях до файлу, префікс методів, доступні методи, їх опис, декоратори, параметри, вхідні та вихідні дані (рисунок 3.7).

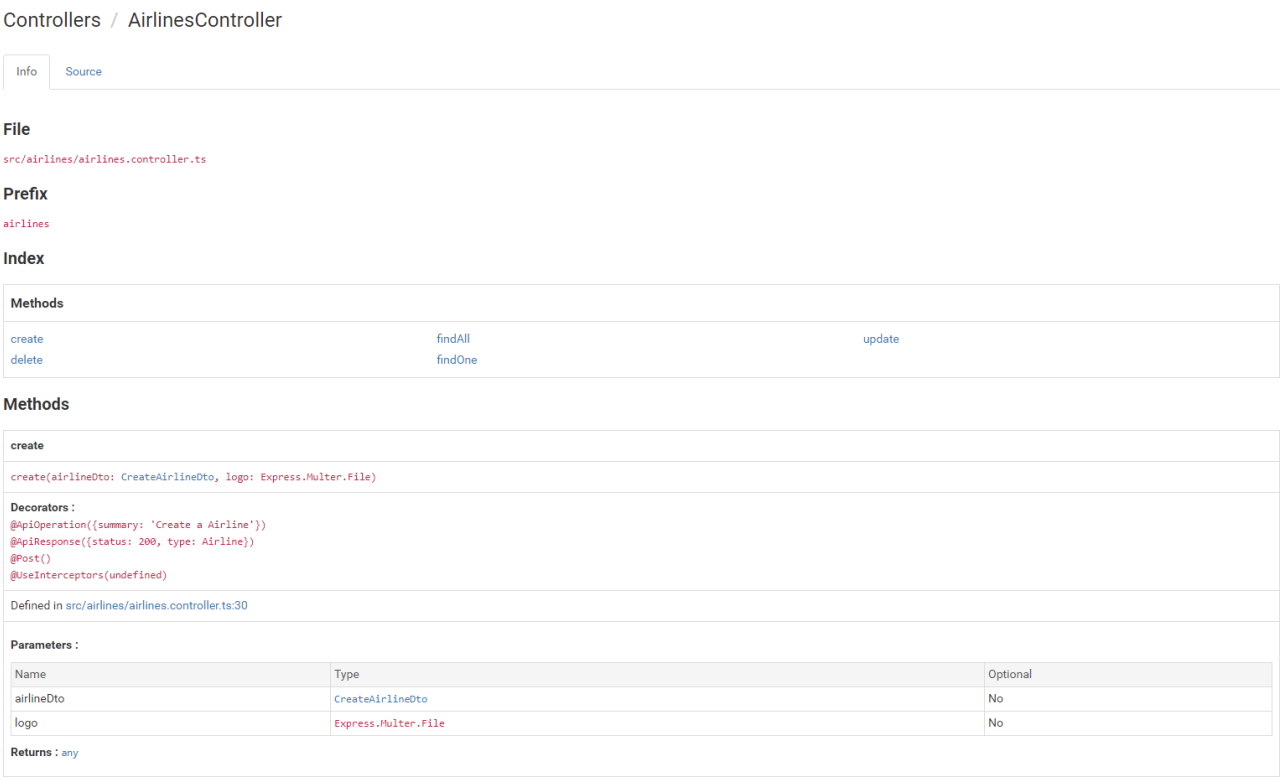

Рисунок 3.7 – Вигляд контролеру веб-сервісу

Загальний вид модуля, де можна бачити його структуру, інформацію про код, шлях до файлів, їх список, (рисунок 3.8).

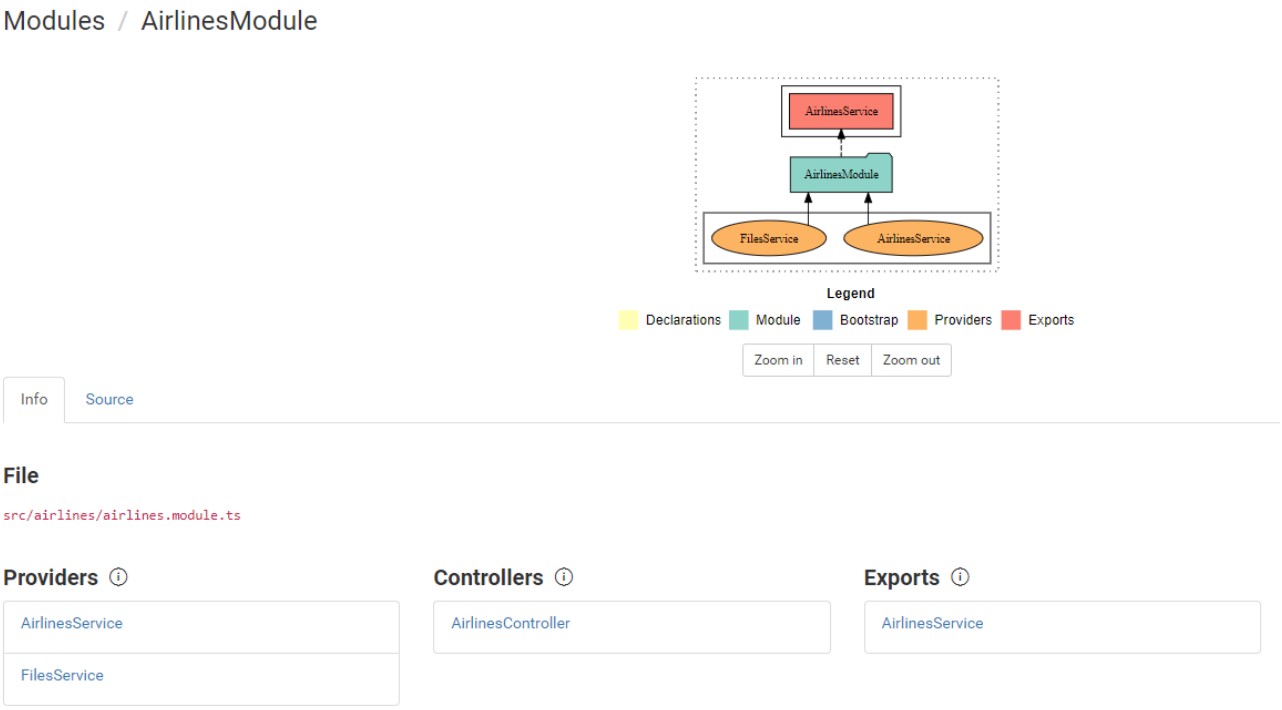

Рисунок 3.8 – Вигляд модулю веб-сервісу

Загальний вид класу, де можна бачити його структуру, властивості, тип, шлях до файлу та декоратори (рисунок 3.9).

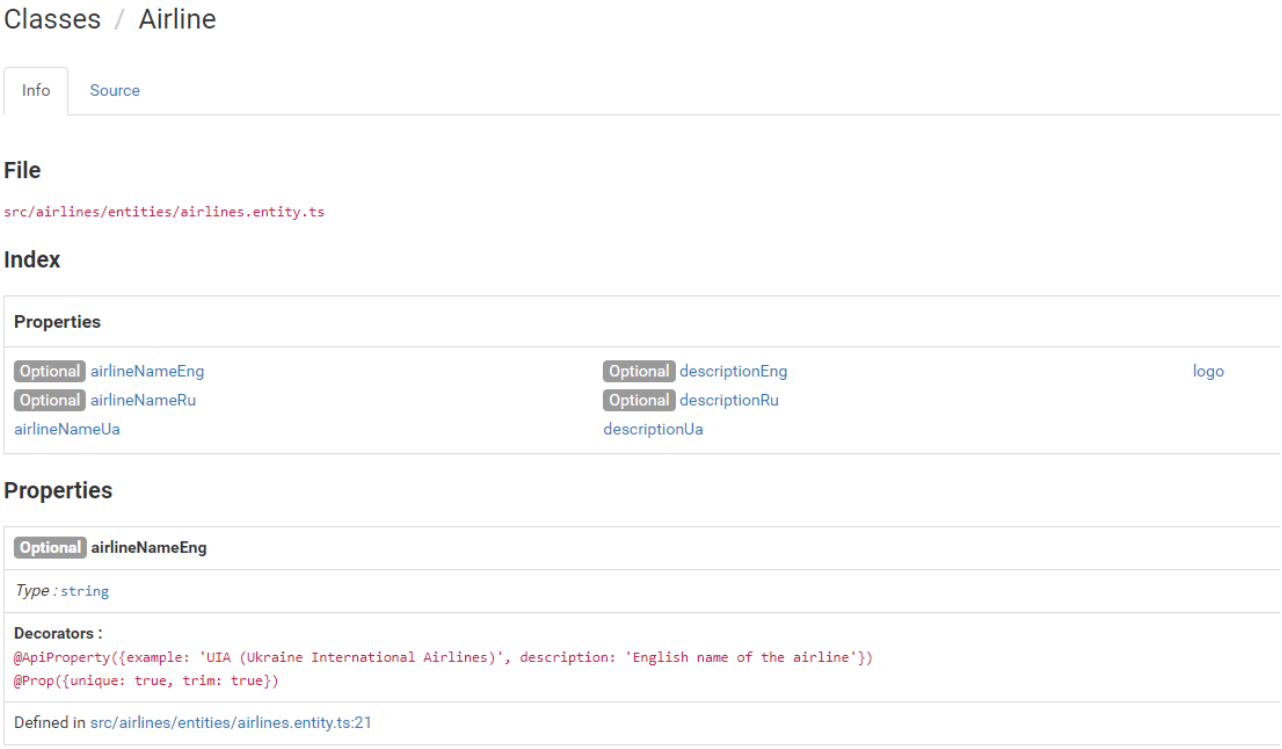

Рисунок 3.9 – Вигляд класу веб-сервісу

Загальний вид ін'єкції, де можна бачити шлях до файлу, його структуру, методи, конструктори та параметри (рисунок 3.10).

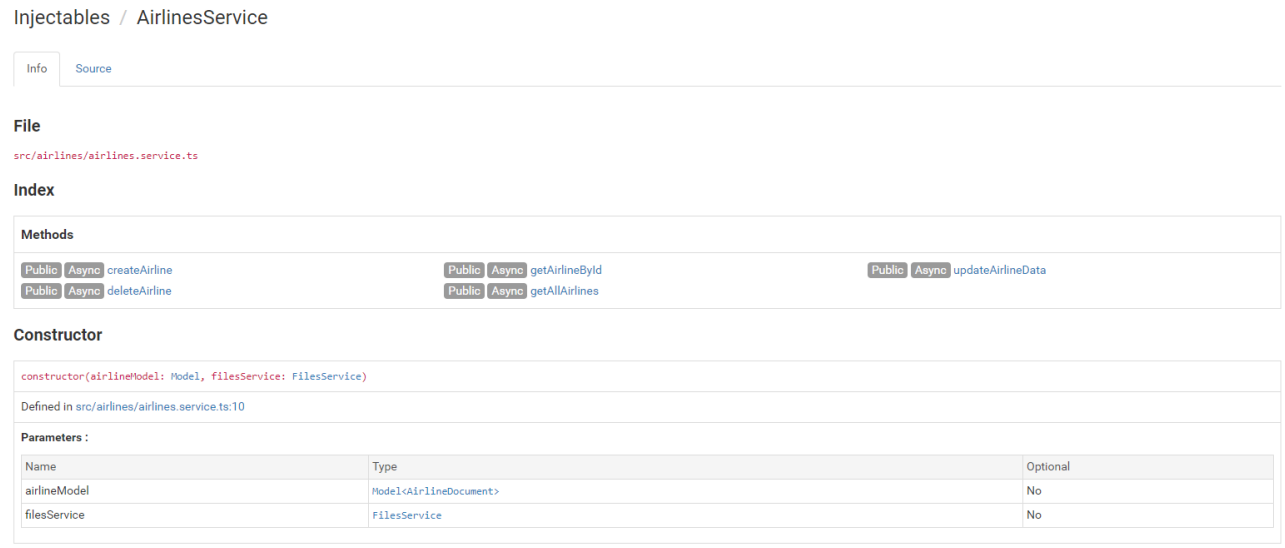

Рисунок 3.10 – Вигляд ін’єкції веб-сервісу

Загальний вид стражів, де можна бачити шлях до файлу, його методи, конструктори та параметри (рисунок 3.11).

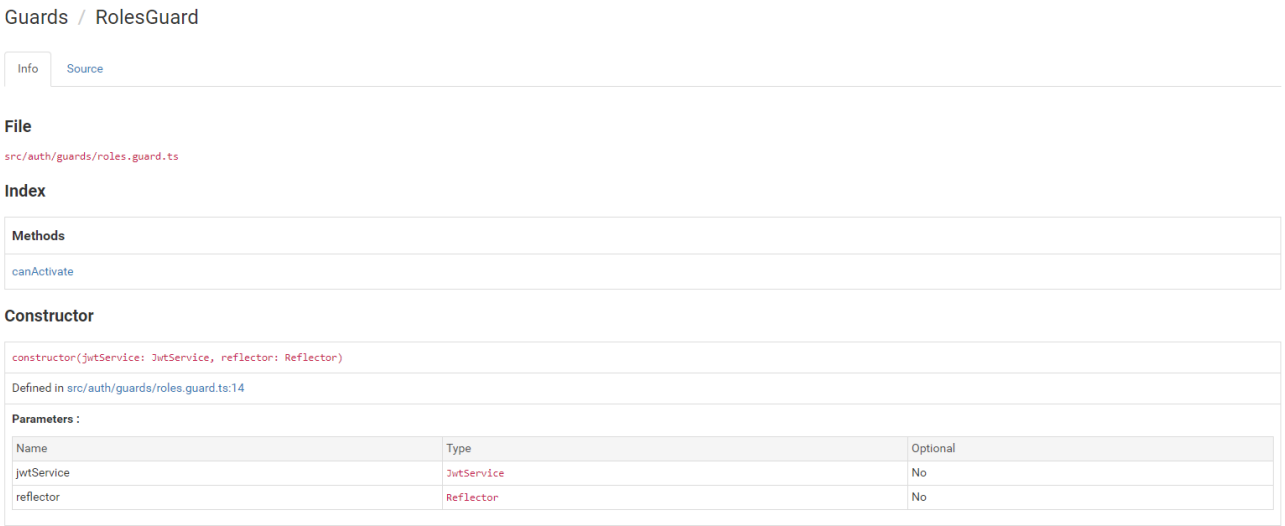

Рисунок 3.11 – Вигляд стражів веб-сервісу

Додатково є розділ з різним, де можна бачити використані переліки, функції, псевдоніми та змінні (рисунок 3.12).

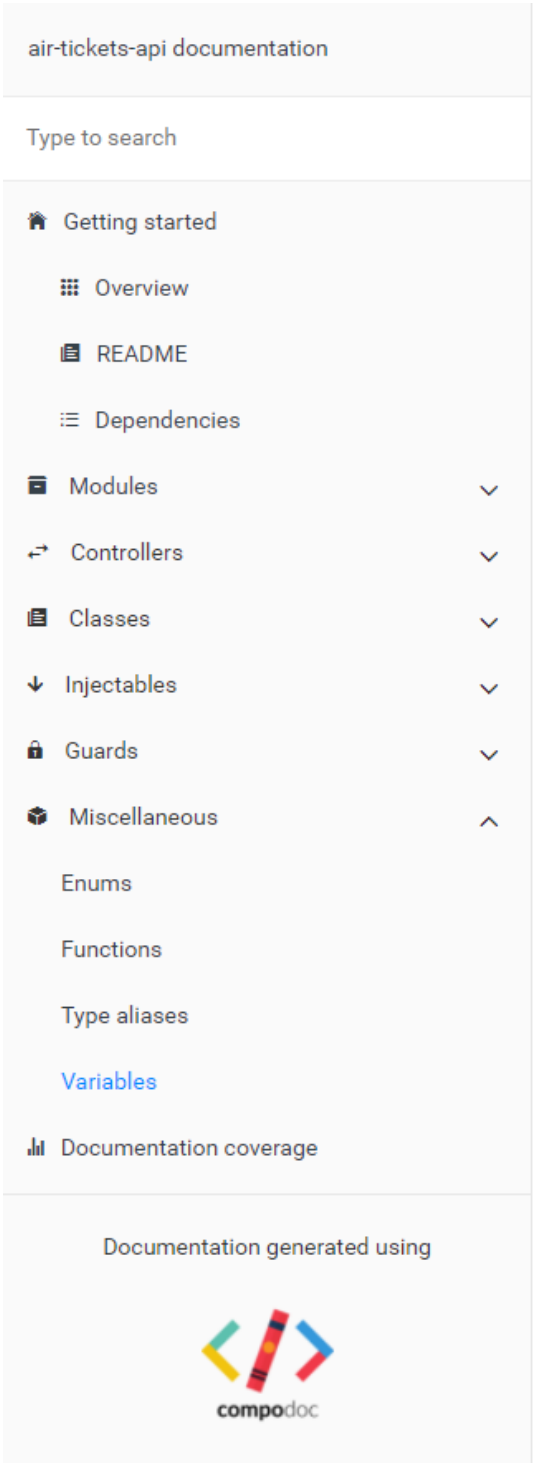

Рисунок 3.12 – Структура документації

ВИСНОВКИ

В даній дипломній роботі було проаналізовано та розглянуто різноманітні способи та технології для створення веб-сервісу. Було описано основні переваги та недоліки та порівняно компоненти веб-сервісів – Rest та Soap.

Досліджено які технології є більш зручними для створення веб-сервісу під різні потреби та функціонал. Також було наведено наочні та актуальні приклади найпопулярніших та розповсюджених інструментів для створення веб-сервісів.

Здійснено аналіз основних засобів, що входить до рефакторингу, оптимізації веб-сервісу, підняття його параметрів та розглянуто загальні параметри безпеки та методи, які дозволяють її покращити.

Було реалізовано власний тип записів, що дозволяє додавати до веб-сервісу тільки потрібну інформацію та спільну базу знань, що вміщує всі введені дані.

Робота виконана на фреймворку Nest.js, який базується на бібліотеці Express платформи Node.js, а також, було використано NoSQL базу даних MongoDB і використано пакети та бібліотеки, які було описано раніше.

Після проведення тестування швидкості роботи та оптимізації веб-сервісу було розроблено повноцінну документацію з користування програмними інтерфейсами. Всі компоненти веб-сервісу працювали коректно, швидко та безперебійно.

СПИСОК ЛІТЕРАТУРИ

1. Веб-служба [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%BB%D1%83%D0%B6%D0%B1%D0%B0 (дата звернення: 08.03.2021);

2. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language [Електронний ресурс]. – Режим доступу : https://www.w3.org/TR/wsdl20/ (дата звернення: 08.03.2021);

3. Simple Object Access Protocol (SOAP) 1.1 [Електронний ресурс]. – Режим доступу : https://www.w3.org/TR/2000/NOTE-SOAP-20000508/ (дата звернення: 08.03.2021);

4. Веб-сервисы – Краткое руководство [Електронний ресурс]. – Режим доступу : https://coderlessons.com/tutorials/veb-razrabotka/izuchite-veb-servisy/veb-servisy-kratkoe-rukovodstvo (дата звернення: 10.03.2021);

5. SOAP Message Transmission Optimization Mechanism [Електронний ресурс]. – Режим доступу : http://www.w3.org/TR/soap12-mtom/ (дата звернення: 10.03.2021);

6. UDDI Version 3.0.2 [Електронний ресурс]. – Режим доступу : http://uddi.org/pubs/uddi_v3.htm (дата звернення: 18.03.2021);

7. Что такое SOA? [Електронний ресурс]. – Режим доступу : http://www.finecosoft.ru/SOA (дата звернення: 18.03.2021);

8. WSDL [Електронний ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/WSDL (дата звернення: 18.03.2021);

9. HTTP [Електронний ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/HTTP (дата звернення: 23.03.2021);

10. Обзор протокола HTTP [Електронний ресурс]. – Режим доступу : https://developer.mozilla.org/ru/docs/Web/HTTP/Overview (дата звернення: 23.03.2021);

11. XML [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/XML (дата звернення: 23.03.2021);

12. Extensible Markup Language (XML) [Електронний ресурс]. – Режим доступу : https://www.w3.org/XML/ (дата звернення: 23.03.2021);

13. XML Введение [Електронний ресурс]. – Режим доступу : https://developer.mozilla.org/ru/docs/Web/XML/XML_introduction (дата звернення: 23.03.2021);

14. Примеры SOAP-запросов веб-сервиса [Електронний ресурс]. – Режим доступу : https://www.elma-bpm.ru/KB/help/Platform/content/Designer_SOA_Web_Service_Primer_index.html (дата звернення: 23.03.2021);

15. Architectural Styles and the Design of Network-based Software Architectures [Електронний ресурс]. – Режим доступу : https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 23.03.2021);

16. REST: простым языком [Електронний ресурс]. – Режим доступу : https://medium.com/@andr.ivas12/rest-%D0%BF%D1%80%D0%BE%D1%81%D1%82%D1%8B%D0%BC-%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%BC-90a0bca0bc78 (дата звернення: 03.04.2021);

17. REST API: что это такое простыми словами, примеры запросов, варианты использования сервиса, методы [Електронний ресурс]. – Режим доступу : https://www.cleverence.ru/articles/elektronnaya-kommertsiya/rest-api-chto-eto-takoe-prostymi-slovami-primery-zaprosov-varianty-ispolzovaniya-servisa-metody/ (дата звернення: 03.04.2021);

18. JSON [Електронний ресурс]. – Режим доступу : https://uk.wikipedia.org/wiki/JSON (дата звернення: 03.04.2021);

19. Работа с JSON [Електронний ресурс]. – Режим доступу : https://developer.mozilla.org/ru/docs/Learn/JavaScript/Objects/JSON (дата звернення: 08.04.2021);

20. API [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/API (дата звернення: 08.04.2021);

21. Что такое API [Електронний ресурс]. – Режим доступу : https://habr.com/ru/post/464261/ (дата звернення: 08.04.2021);

22. JavaScript [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/JavaScript (дата звернення: 09.04.2021);

23. Что такое JavaScript? [Електронний ресурс]. – Режим доступу : https://developer.mozilla.org/ru/docs/Learn/JavaScript/First_steps/What_is_JavaScript (дата звернення: 09.04.2021);

24. To UDDI Spec TC page OASIS UDDI Specifications TC – Committee Specifications [Електронний ресурс]. – Режим доступу : https://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm (дата звернення: 14.04.2021);

25. UDDI [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/UDDI (дата звернення: 14.04.2021);

26. UDDI — Краткое руководство [Електронний ресурс]. – Режим доступу : https://coderlessons.com/tutorials/akademicheskii/vyuchi-uddi/uddi-kratkoe-rukovodstvo (дата звернення: 14.04.2021);

27. What is XML-RPC? [Електронний ресурс]. – Режим доступу : http://xmlrpc.com/ (дата звернення: 19.04.2021);

28. XML-RPC — Краткое руководство [Електронний ресурс]. – Режим доступу : https://coderlessons.com/tutorials/xml-tekhnologii/izuchite-xml-rpc/xml-rpc-kratkoe-rukovodstvo (дата звернення: 19.04.2021);

29. REST [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/REST (дата звернення: 19.04.2021);

30. RESTful Web Services Tutorial with REST API Example [Електронний ресурс]. – Режим доступу : https://www.guru99.com/restful-web-services.html (дата звернення: 19.04.2021);

31. Typed JavaScript at Any Scale [Електронний ресурс]. – Режим доступу : https://www.typescriptlang.org/ (дата звернення: 26.04.2021);

32. TypeScript [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/TypeScript (дата звернення: 26.04.2021);

33. Node.js [Електронний ресурс]. – Режим доступу : https://nodejs.org/uk/ (дата звернення: 28.04.2021);

34. Express [Електронний ресурс]. – Режим доступу : https://expressjs.com/ru/ (дата звернення: 28.04.2021);

35. Nest.js [Електронний ресурс]. – Режим доступу : https://nestjs.com/ (дата звернення: 28.04.2021);

36. MongoDB [Електронний ресурс]. – Режим доступу : https://www.mongodb.com/ (дата звернення: 28.04.2021);

37. MongoDB Atlas [Електронний ресурс]. – Режим доступу : https://www.mongodb.com/cloud/atlas (дата звернення: 05.05.2021);

38. Babel [Електронний ресурс]. – Режим доступу : https://babeljs.io/ (дата звернення: 05.05.2021);

39. Webpack 5 [Електронний ресурс]. – Режим доступу : https://webpack.js.org/ (дата звернення: 05.05.2021);

40. Prettier [Електронний ресурс]. – Режим доступу : https://prettier.io/ (дата звернення: 11.05.2021);

41. ESLint [Електронний ресурс]. – Режим доступу : https://eslint.org/ (дата звернення: 13.05.2021);

42. Passport [Електронний ресурс]. – Режим доступу : http://www.passportjs.org/ (дата звернення: 15.05.2021);

43. node.bcrypt.js [Електронний ресурс]. – Режим доступу : https://www.npmjs.com/package/bcrypt (дата звернення: 15.05.2021);

44. Helmet [Електронний ресурс]. – Режим доступу : https://www.npmjs.com/package/helmet (дата звернення: 16.05.2021);

45. JSON Web Token [Електронний ресурс]. – Режим доступу : https://ru.wikipedia.org/wiki/JSON_Web_Token (дата звернення: 16.05..2021);

46. Postman [Електронний ресурс]. – Режим доступу : https://www.postman.com/ (дата звернення: 16.05.2021);

47. class-validator [Електронний ресурс]. – Режим доступу : https://github.com/typestack/class-validator (дата звернення: 17.05.2021);

48. class-transformer [Електронний ресурс]. – Режим доступу : https://github.com/typestack/class-transformer (дата звернення: 18.05.2021);

49. Mongoose [Електронний ресурс]. – Режим доступу : https://mongoosejs.com/ (дата звернення: 19.05.2021);

50. Swagger [Електронний ресурс]. – Режим доступу : https://swagger.io/ (дата звернення: 19.05.2021);

51. Fuse.js [Електронний ресурс]. – Режим доступу : https://fusejs.io/ (дата звернення: 21.052021);

52. Moment.js [Електронний ресурс]. – Режим доступу : https://momentjs.com/ (дата звернення: 21.05.2021);

53. UUID [Електронний ресурс]. – Режим доступу : https://www.npmjs.com/package/uuid (дата звернення: 22.05.2021);

54. Compodoc [Електронний ресурс]. – Режим доступу : https://compodoc.app/ (дата звернення: 22.05.2021);

55. npm [Електронний ресурс]. – Режим доступу : https://www.npmjs.com/ (дата звернення: 23.05.2021);

56. VS Code [Електронний ресурс]. – Режим доступу : https://code.visualstudio.com/ (дата звернення: 23.05.2021);

ДОДАТОК А. КОД МЕТОДУ ПОШУКУ БІЛЕТУ

```
      public async searchFlights(flightDto: SearchFlightDto):
Promise<Flight[]> {
        let secondDepTime: string = flightDto.firstDepTime;
        if (flightDto.secondDepTime) {
          secondDepTime = flightDto.secondDepTime;
        }

        const initialFlights = await this.flightModel
          .find({
            departureTime: {
              $gte: moment(flightDto.firstDepTime)
                .startOf('day')
                .add(3, 'hours')
                .toISOString()
                .toString(),
              $lte: moment(secondDepTime)
                .endOf('day')
                .add(3, 'hours')
                .toISOString()
                .toString(),
            },
          })
          .select({ __v: false })
          .populate({
            path: 'airline',
            select: '-__v',
          })
          .populate({
            path: 'airplane',
            select: '-__v',
            populate: {
              path: 'sections',
              select: '-__v',
              populate: { path: 'seats', select: '-__v' },
            },
          })
          .populate({ path: 'departureAirport', select: '-__v'
})
          .populate({  path:  'arrivalAirport',  select:  '-__v'
})
```

```
        .exec();

      const arr = [];
      initialFlights.forEach((a) => {
        if (flightDto.class.length > 1) {
          if (a.airplane.amountOfSeat >= flightDto.count) {
            a.airplane.sections.forEach((s) => {
              for (const q of flightDto.class)
                if (s.class === q) {
                  arr.push(a);
                  break;
                }
            });
          }
        } else {
          a.airplane.sections.forEach((s) => {
            for (const q of flightDto.class)
              if (s.class === q) {
                if (s.seats.length >= flightDto.count) {
                  arr.push(a);
                }
              }
          });
        }
      });
      const flights = Array.from(new Set(arr));

      const optionsForDeparture = {
        includeScore: true,
        keys: [
          'departureAirport.airportNameUa',
          'departureAirport.airportNameEng',
          'departureAirport.airportNameRu',
          'departureAirport.IATA',
          'departureAirport.airportTownUa',
          'departureAirport.airportTownEng',
          'departureAirport.airportTownRu',
        ],
      };

      let fuse = new Fuse(flights, optionsForDeparture);
      let result = fuse.search(flightDto.departure).map((r)
=> r.item);
```

```
          const optionsForArrival = {
            includeScore: true,
            keys: [
              'arrivalAirport.airportNameUa',
              'arrivalAirport.airportNameEng',
              'arrivalAirport.airportNameRu',
              'arrivalAirport.IATA',
              'arrivalAirport.airportTownUa',
              'arrivalAirport.airportTownEng',
              'arrivalAirport.airportTownRu',
            ],
          };

          fuse = new Fuse(result, optionsForArrival);
          result  =  fuse.search(flightDto.arrival).map((r)  =>
r.item);
          return result;
        }
```