НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютеризовані системи управління»
спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології»
на тему: «Система обробки JSON документів»

Виконав (-ла):
студент (-ка) IV курсу, групи ІА-61
Вовк Дмитро Андрійович _____

Керівник:
кандидат технічних наук, доцент
Букасов Максим Михайлович _____

Рецензент:
кандидат технічних наук, доцент
Тетяна ЛІХОУЗОВА

_____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 151 «Автоматизація та комп'ютерно-інтегровані технології»

Освітньо-професійна програма «Комп'ютеризовані системи управління»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Вовку Дмитру Андрійовичу

1. Тема проєкту «Система обробки JSON документів», керівник проєкту Букасов Максим Михайлович, к.т.н., доцент, затверджені наказом по університету від «_7_»__травня__2020 р. №1081-с

2. Термін подання студентом проєкту 09.06.2020

3. Вихідні дані до проєкту застосування мікросервісної архітектури; впровадження архітектурного стилю REST; незалежність від операційної системи; можливість інтеграції до зовнішної системи; зберігання JSON документів; виконання усіх базових дій над JSON документами.

4. Зміст пояснювальної записки аналіз існуючих рішень. Пошук та обґрунтування обраних технологій. Розроблення структури системи. Розроблення алгоритму оброблення запитів до системи. Аналіз якості та тестування отриманої системи.

5. Перелік графічного матеріалу: схема структурна, алгоритм оброблення запиту до системи, діаграма послідовності, діаграма варіантів використання.

6. Дата видачі завдання 05.03.2020.

Календарний план

№	Назва етапів виконання	Термін виконання	Примітка
1	Аналіз існуючих рішень	30.04.20 – 05.05.20	
2	Пошук технологій для системи	05.05.20 – 08.05.20	
3	Проектування структури системи	08.05.20 – 13.05.20	
4	Розроблення схеми структурної	13.05.20 – 15.05.20	
5	Розроблення серверної частини	15.05.20 – 22.05.20	
	системи		
6	Аналіз якості системи	22.05.20 – 23.05.20	
7	Оформлення текстової	23.05.20 – 01.06.20	
	документації		

Студент Дмитро ВОВК

Керівник Максим БУКАСОВ

АНОТАЦІЯ

Вовк Д.А. Система обробки JSON документів. КПІ ім. Ігоря Сікорського, Київ, 2020.

Дипломний проєкт виконано на 75 аркушах, він містить 4 додатки та перелік посилань на використані джерела з 28 найменувань. У проєкті наведено 76 рисунків.

Метою даного дипломного проєкту є створення програмного забезпечення для оброблення JSON документів.

У проєкті було проведено аналіз існуючих реалізацій даного завдання іншими компаніями та розробниками. Проведено аналіз існуючих рішень та виконано порівняння недоліків та переваг відносно розроблюваної системи. Для реалізації завдання було обрано існуючі низькорівневі бібліотеки написані мовою програмування Java.

У процесі розроблення проекту було спроектовано та розроблено програмне забезпечення для розв’язку задачі дипломного проєкту. Отриманий програмний продукт було протестовано та проаналізовано результати виконання.

Ключові слова: JSON, HTTP, REST, API.

SUMMARY

Vovk D. A. JSON Document Processing System. Igor Sikorsky KPI, Kyiv, 2020.

The thesis is completed on 75 pages, it contains 4 annexes and a list of references to the used sources of 28 items. The paper contains 76 images.

The purpose of this thesis is to create software for processing JSON documents.

The analysis of existing implementations of this task by other companies and developers was carried out in the work. The analysis of existing solutions is carried out and the disadvantages and advantages are compared with respect to the developed system. To implement the task, the existing low-level libraries written in the Java programming language were selected and used.

In the course of the work, the software for solving the thesis problem was designed and developed. The obtained software product was tested and the results of execution were analyzed.

Keywords: JSON, HTTP, REST, API

Номер рядка	Формат	Позначення	Найменування	Кільк. аркушів	Номер екзем.	Примітка
1			Документація загальна			
2						
3			Знову розроблена			
4						
5	А4	ІА61.060БАК.005 ПЗ	Пояснювальна записка	75		
6						
7	А3	ІА61.060БАК.005 Д1	Структурна схема	1		
8						
9	А3	ІА61.060БАК.005 Д2	Діаграма оброблення запиту до	1		
10			системи			
11	А3	ІА61.060БАК.005 Д3	Діаграма послідовності	1		
12						
13	А3	ІА61.060БАК.005 Д4	Діаграма варіантів використання	1		
14			системи			
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						
25						
26						
27						
28						

Зм.	Аркуш	№ докум.	Підпис	Дата	ІА61.060БАК.005 ТП		
Розроб.		Вовк			Система обробки JSON документів	Літ.	Аркуш	Аркушів
Перевір.		Букасов				Т	1	1
Реценз.						«КПІ ім. І. Сікорського» ФІОТ група ІА-61		
Н. Контр.					Відомість технічного проекту			
Затв.								

Пояснювальна записка
до дипломного проєкту
на тему: «Система обробки JSON документів»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

API – Application Programming Interface;

CRUD – Create, Read, Update, Delete;

HTML – Hypertext Markup Language;

HTTP – HyperText Transfer Protocol;

JSON – JavaScript Object Notation;

REST – Representational State Transfer;

UI – User Interface;

URL – Uniform Resource Locator.

ВСТУП

У даний час інформаційні технології досягли значних вершин та продовжують свій розвиток. Сьогоднішні комп'ютеризовані системи неможливо уявити без використання JSON – формату, який реалізує структуроване текстове представлення даних, засноване на принципі ключ-значення. Наразі можна сказати, що JSON став формальним стандартом обміну даними між веб-клієнтами та веб-сервером, також JSON широко використовується як формат для збереження конфігурації об'єкта або збереження даних в таких базах даних як PostgreSQL, MySQL, ArangoDB та MongoDB. Використання JSON базами даних дуже спрощує зберігання даних без налаштування відношень між таблицями. Без JSON не можуть функціонувати і системи побудовані на мікросервісній архітектурі, які використовуюсь даний формат для забезпечення передачі даних між сервісами. Завдяки широкому поширенню JSON його підтримують усі сучасні браузери та мови програмування. Такі задачі як зміна існуючої конфігурації або існуючого об'єкту в базі даних потребують модифікації JSON документів.

На даний час існує досить багато аналогів для редагування документів формату JSON, але всі вони розроблені для використання тільки людиною та не можуть бути інтегровані в іншу систему. Іншою проблемою існуючих редакторів документів є неможливість швидко редагувати великі документи, що дуже ускладнює на перший погляд просту задачу у вигляді редагування документу. Окрім того наразі існує багато реалізованих систем побудованих на мікросервісній архітектурі мають можливість редагування таких документів, але така можливість буде коштувати їм дублюванням коду, а потім і пересиланням великих документів між сервісами, що буде коштувати системі часу, розподіленим навантаженням на такі важливі ресурси системи як процесорний час, оперативна пам'ять та пропускна здатність мережі. Рішенням проблеми ресурсів найменшими затратами та проблеми редагування великих документів користувачем є розроблення системи з підтримкою мікросервісної архітектури.

У даному проекті проаналізовано доступні комерційні та безкоштовні реалізації системи оброблення та збереження JSON документів. За основу системи узято мікросервісну архітектуру, мову програмування Java разом з бібліотеками Spring та SimpleJSON, для забезпечення можливості системою оброблення HTTP запитів використано архітектурний стиль REST та базу даних ArangoDB для збереження даних.

1 ПОСТАНОВКА ЗАДАЧІ

Завданням для дипломного проекту є розроблення системи оброблення JSON документів. Система буде мати можливість виконувати свої функції як при роботі з користувачем у вигляді людини, так і будучі інтегрованою до зовнішньої системи. Система повинна мати можливість запуску на будь-якій операційній системі, включаючи такі поширені як Windows, Linux, MacOS. Користувач або інтегрована система повинні мати змогу завантажувати та видаляти документи з системи оброблення та модифікувати існуючі документи. Система не повинна займати багато ресурсів системи для функціонування та повинна мати можливість запуску у будь-якій операційній системі. Для функціонування системи у вигляді сервера потрібно реалізувати взаємодію за REST протоколом. Система повинна оброблювати запити від користувача та зовнішньої системи за одним алгоритмом. Система не повинна припиняти своє виконання при виникненні помилок, для цього система має містити обробник помилок у кожному сервісі. При виникненні помилок система повинна повідомити користувача або зовнішню систему про помилку за допомогою HTTP статусу та інформативним повідомленням про помилку. При успішному обробленню запиту повинен бути повернутий успішний HTTP статус до користувача або зовнішньої системи та тіло запиту. Для збереження, видалення та модифікації документів до системи за допомогою драйверу бази даних потрібно підключити базу даних з підтримкою збереження даних у форматі JSON. Драйвер бази даних не повинен мати тонких налаштувань для забезпечення його швидкої роботи та мати можливість виконувати усі базові дії над документами у базі даних. Взаємодія користувача з системою повинна виконуватись за допомогою графічного інтерфейсу, потрібно щоб графічний інтерфейс описував API та документував його для полегшення інтеграції розроблюваної системи до зовнішньої системи. Система повинна мати можливість працювати у мережі «Інтернет» та підтримувати роботу з мобільними клієнтами браузерів.

2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

Існує досить багато додатків для оброблення JSON документів. По більшій частині ці додатки являють собою редактори тексту з додатковими налаштуваннями для покращення роботи з документом.

2.1 Редактор Notepad++

Notepad++ – це редактор розроблений Don Ho для редагування документів різних форматів, в тому числі й формату JSON. Редактор може проводити усі базові операції над документами, такі як зміна документу, його збереження в різних форматах, також в редакторі впроваджено зручну навігацію по документу(рисунок 2.1)[1].

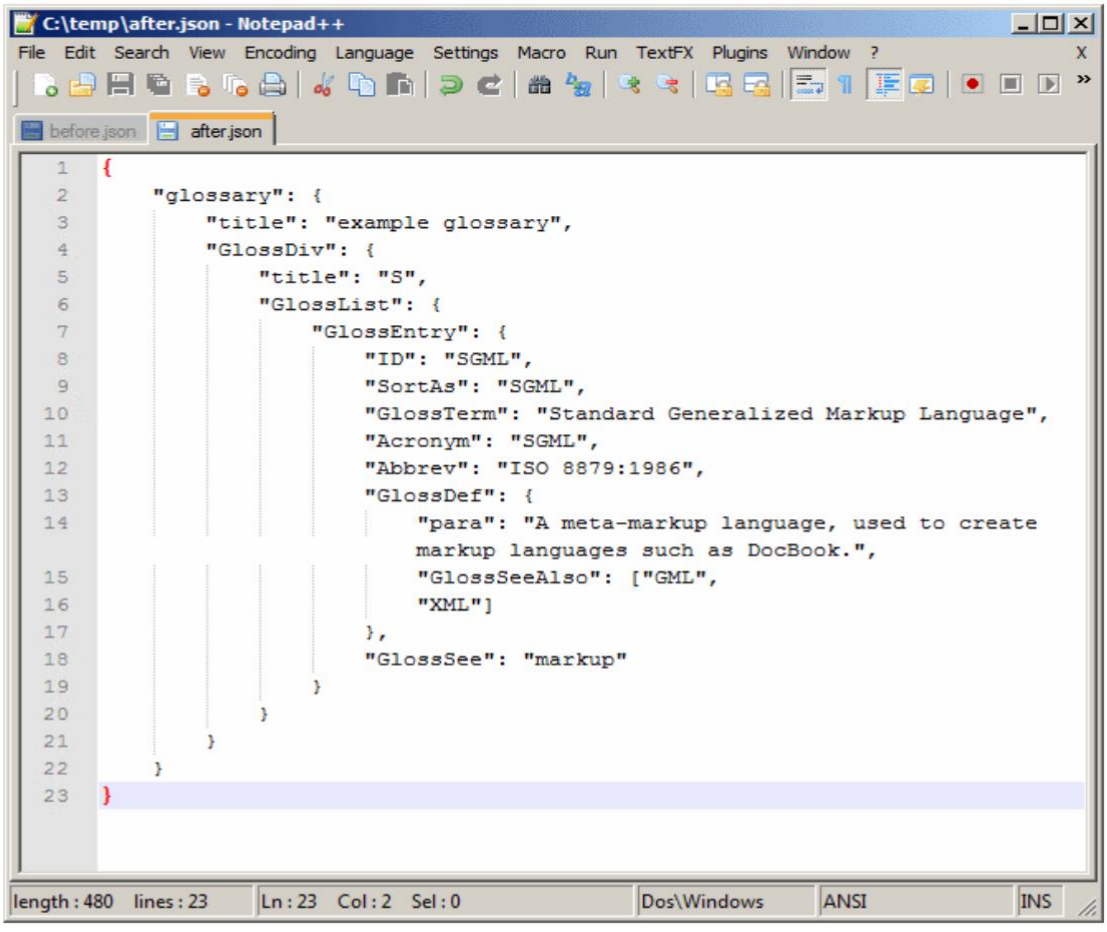

Рисунок 2.1 – Скріншот інтерфейсу Notepad++

Найкраще підходить для швидкого редагування майже будь якого документу, широко налаштовуєтеся та не потребує багато ресурсів.

Переваги редактору документів Notepad++:

–розповсюджується на безкоштовній основі;

–зрозумілий інтерфейс;

–добре підходить для швидкого редагування документу;

–редактор може бути налаштований для роботи з багатьма форматами;

–маленький розмір встановленої програми.

Недоліки редактору документів Notepad++:

– стабільність роботи редактора залежить напряму від його налаштування та розміру документу;

–немає підтримки роботи зі шляхом Json;

Стабільність роботи залежіть від потужності комп’ютера користувача.

2.2 Редактор Visual Code

Visual Code – це рішення від Microsoft яке представляє собою редактор з автоматичним форматуванням тексту та розширеними можливостями перегляду документів. Він встановлюється та використовується на персональних пристроях. Користувач з допомогою додатку може виконувати базові дії над документами різноманітного формату. Його важливим бонусом можна вважати підтримку “git”. Це дозволяє розробнику зручно користуватися усіма можливостями git без встановлення додаткових клієнтів та іншого програмного забезпечення. Також Visual Code підтримує розробку багатьма мовами програмування, тому у додатку реалізована функція налагодження програми, що дозволяє швидко знайти проблему.. Додаток пропонує зручну навігацію по документу у правій частині екрану за допомогою проекції зменшеної частини документа(рисунок 2.2)[2].

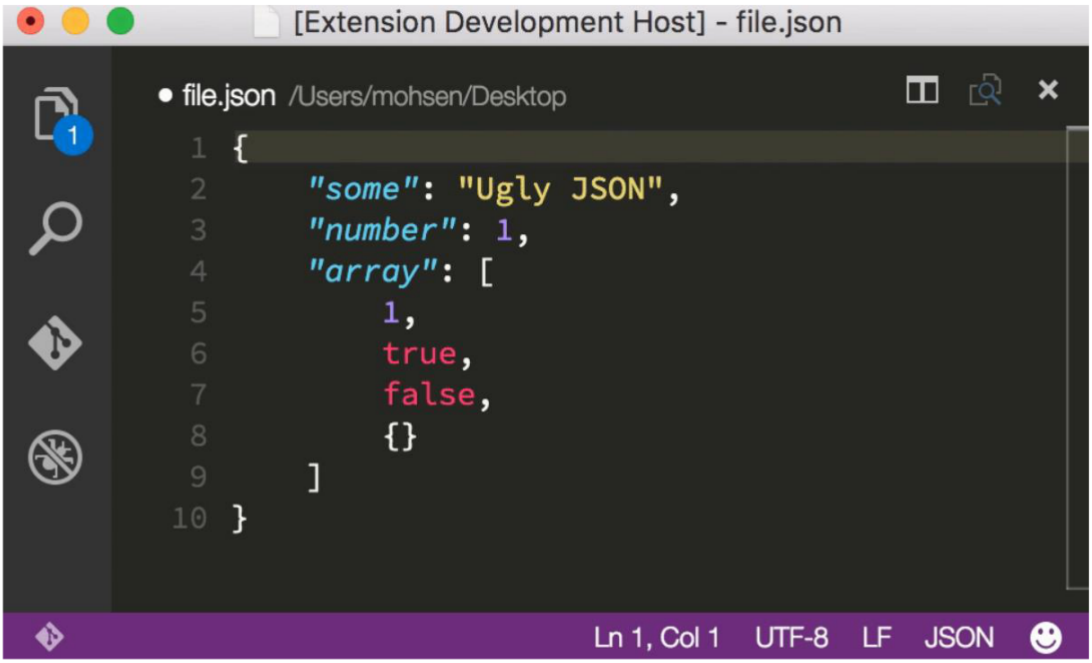

Рисунок 2.2 – Скріншот інтерфейсу Visual Code

Додаток добре підходить для розробників, які працюють з Visual Code для виконання своїх основних задач, так як їм не потрібно встановлювати та налаштовувати новий редактор для будь-якого редагування документу формату JSON або іншого.

Переваги редактору документів Visual Code:

–розповсюджується на безкоштовній основі;

–розроблюється Microsoft;

–зрозумілий інтерфейс для просунутих користувачів;

–добре підходить для швидкого редагування документу а також коду;

–редактор може бути налаштований для роботи з багатьма форматами;

–досить невеликий розмір встановленої програми.

Недоліки редактору документів Visual Code:

– стабільність роботи редактора залежить напряму від його налаштування та розміру документу;

–немає підтримки роботи зі шляхом Json;

Стабільність роботи залежіть від потужності комп’ютера користувача.

2.3 Редактор Sublime Text

Sublime Text – це черговий редактор документів розроблений Jon Skinner. Він встановлюється та використовується на персональних пристроях. Комерційний редактор документів, має той самий набор базових функцій що й аналоги, але працює за принципом попереднього завантаження та форматування документу, тому при початку роботи з документом редактор займає деякий час для обробки документа, і лише після цього дозволяє проводити маніпуляції над документом(рисунок 2.3)[3].

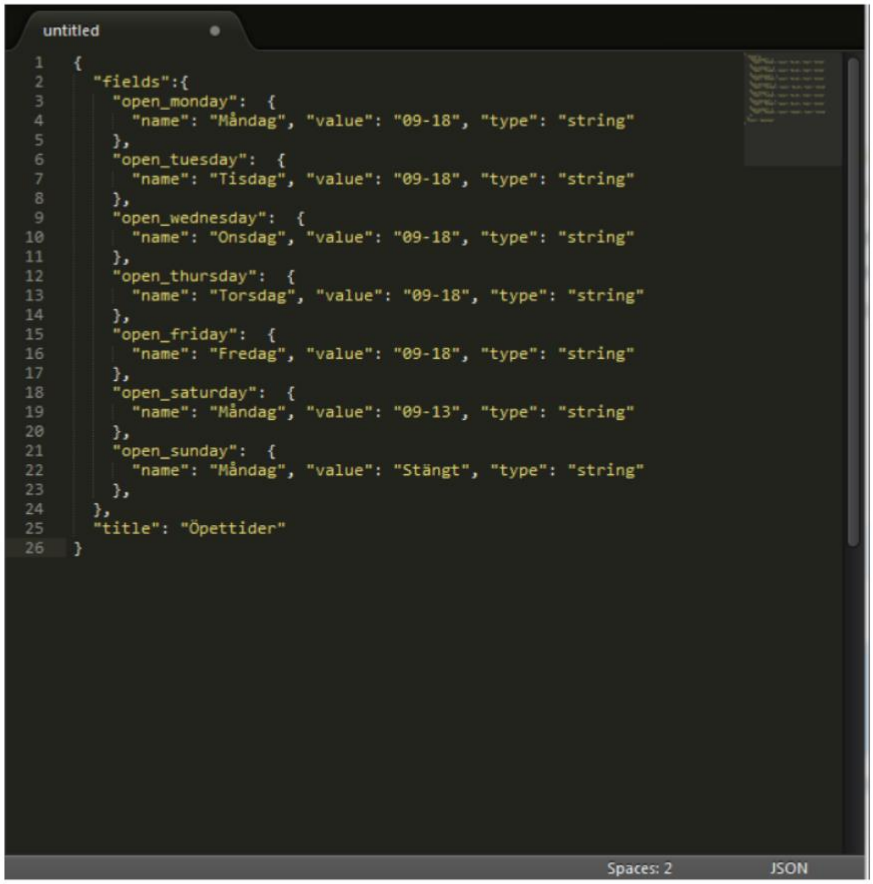

Рисунок 2.3 – Скріншот інтерфейсу Sublime Text

Добре підходить для роботи з досить великими документами які навантажують редактор.

Переваги редактору документів Sublime Text:

–зрозумілий інтерфейс;

–добре підходить для редагування великих документів;

–редактор може бути налаштований для роботи з багатьма форматами;

–досить невеликий розмір встановленої програми.

Недоліки редактору документів Sublime Text:

– час опрацювання документу може бути великим при великому розмірі документа і низькій продуктивності системи користувача;

–немає підтримки роботи зі шляхом Json;

–стабільність роботи залежіть від потужності комп’ютера користувача;

–потребує придбання ліцензії.

2.4 Онлайн редактор JSONPath Online Evaluator

JSONPath Online Evaluator – це онлайн додаток розроблений Казукі Хамасакі. Особливістю виступає можливість редагування документів без встановлення самого редактору, за допомогою інтернету. Редактор дозволяє редагувати лише документи невеликого розміру, а також на відміну від аналогів має підтримку роботи зі шляхом Json(рисунок 2.4)[4].

Рисунок 2.4 – Скріншот інтерфейсу JSONPath Online Evaluator

Найкраще для: швидкого редагування документу, перевірки його синтаксису або пошуку окремого значення по шляху JSON. Також може бути корисним для розробників при використанні шляху JSON.

Переваги інтернет редактору документів JSONPath Online Evaluator:

–розповсюджується на безкоштовній основі;

–зрозумілий інтерфейс;

–добре підходить для редагування маленьких документів;

–не потребує встановлення;

–аналізує синтаксис;

–підтримує роботу зі шляхом JSON.

Недоліки інтернет редактору документів JSONPath Online Evaluator:

–немає підтримки роботи з великими документами;

–немає підтримки завантаження документу;

Стабільність роботи залежіть від браузера користувача.

2.5 Онлайн редактор JSON Editor Online

JSON Editor Online – це веб-інструмент для перегляду, редагування і форматування JSON. JSON Editor Online розроблено у 2011 році Джос де Чонгом. Він показує ваші дані пліч-о-пліч в ясному редагованому дереві або в редакторі коду. Готовий документ можна завантажити на комп’ютер користувача. Користувач може зберігати документи одразу після його редагування. JSON Editor Online дозволяє зберігати документи локально або в хмарі, що демонструє схожість з розроблюваною системою яка є завданням цього дипломного проєкту. Завантажений до системи документ можна отримати за допомогою UUID ключа, що

є досить зручною функцією, коли користувачеві потрібно повернутися до редагування документу пізніше. Додатковою можливістю JSON Editor Online є перегляд документу у вигляді коду або у вигляді дерева, а також сортування та форматування документу.(рисунок 2.5)[5].

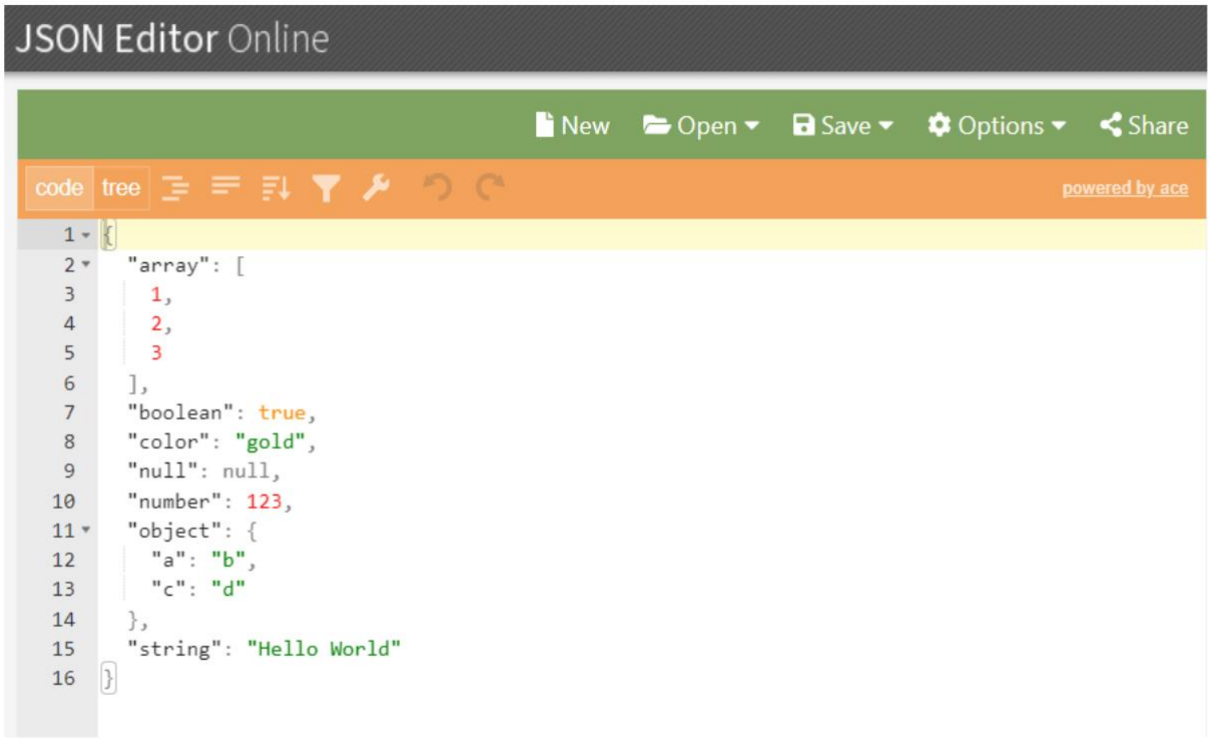

Рисунок 2.5 – Скріншот інтерфейсу JSON Editor Online

JSON Editor Online найкраще підходить для: швидкого редагування документу, перевірки його синтаксису а також збереження документу у хмарі. Також може бути корисним для розробників при порівнянні двух JSON документів.

Переваги інтернет редактору документів JSON Editor Online:

–розповсюджується на безкоштовній основі;

–зрозумілий інтерфейс;

–добре підходить для редагування маленьких документів;

–не потребує встановлення;

–аналізує синтаксис;

–дозволяє зберігати документи у хмарі.

Недоліки інтернет редактору документів JSON Editor Online:

–немає підтримки роботи з великими документами;

–немає підтримки завантаження документу;

–не підтримує роботу зі шляхом JSON.

Стабільність роботи залежіть від браузера користувача.

2.6 Онлайн редактор Online JSON Viewer

Online JSON Viewer – це онлайн додаток розроблений Габором Турі. Його головною особливістю виступає можливість редагування документів без встановлення самого редактору, за допомогою інтернету а також перетворення документів в більш зручний для читання формат. Редактор дозволяє редагувати лише документи невеликого розміру(рисунок 2.6)[6].

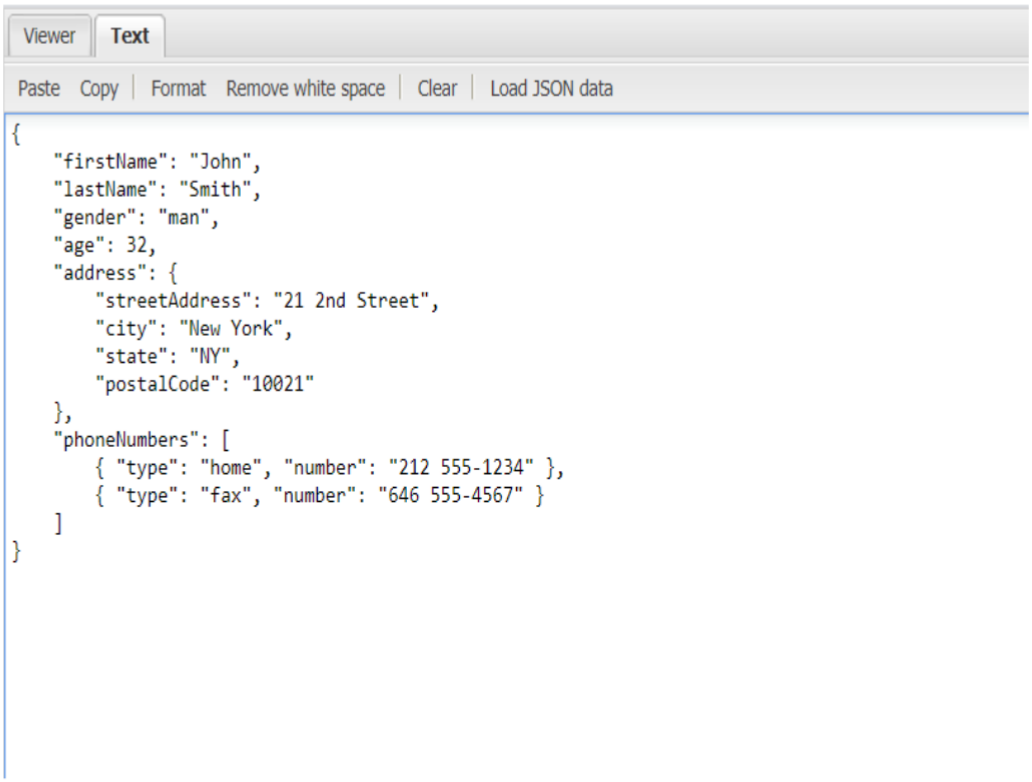

Рисунок 2.6 – Скріншот інтерфейсу Online JSON Viewer

Переваги інтернет редактору документів JSON Editor Online:

–розповсюджується на безкоштовній основі;

–зрозумілий інтерфейс;

–добре підходить для редагування маленьких документів;

–не потребує встановлення;

–аналізує синтаксис;

–дозволяє зберігати документи у хмарі;

–дозволяє переводити документи в більш зручний вигляд для читання.

Недоліки інтернет редактору документів JSON Editor Online:

–немає підтримки роботи з великими документами;

–немає підтримки завантаження документу;

–не підтримує роботу зі шляхом JSON.

Стабільність роботи залежіть від браузера користувача.

2.7 Висновки до розділу

У розділі проведено аналіз та огляд існуючих рішень, було виділено основних конкурентів, визначено їх недоліки та переваги. Кожен з перелічених додатків має можливість редагування JSON документу, більшість має можливість приведення табуляції до нормального, більш зручного для читання вигляду. Але жодне з перерахованих рішень не має можливості бути інтегрованим в іншу систему, лише декілька рішень підтримують шлях JSON та лише одне рішення дозволяє зберігати документи у хмарному сховищі.

Кожен із редакторів володіє певними йому унікальними можливостями, і ці можливості іноді бувають дуже корисні, але загалом усі розглянуті редактори припускають лише один варіант роботи і тільки з користувачем. Така модель оброблення документів має недоліки у вигляді проблем з обробленням великих JSON документів та мають незручності при редагуванні документів з мобільного телефону.

За результатом аналізу можна зробити висновок, що системи оброблення JSON документів у такому вигляді, у якому вона розроблюється в даному проекті – немає.

3 ПРОЕКТУВАННЯ СИСТЕМИ

3.1 Структура системи

Дану систему було написано мовою програмування Java з використанням Spring Framework.

3.1.1 Мова програмування Java

Java – об’єктно-орієнтована мова програмування яка була ініційована Джеймсом Гослінгом і випущена в 1995 році як основний компонент платформи Java Sun Microsystems (Java 1.0). Компанія Sun випустила першу публічну реалізацію як Java 1.0 у 1995 році. Мова наслідує значну частину свого синтаксису від C та C ++, ці мови програмування також вплинули і на більшість об'єктно-орієнтованих функцій Java, але потрібно відзначити – Java має менше можливостей низького рівня, C або C ++. Java проповідує принцип Write Once, Run Anywhere (WORA), за допомогою того що компільований код Java може працювати на всіх платформах, які підтримують Java, не потребуючи рекомпіляції[7]. Наприклад, код Java може бути написано та скомпільовано на UNIX та а після цього запущений на машині Microsoft Windows, Macintosh або UNIX без будь-яких змін вихідного коду. WORA досягається компіляцією програми Java на проміжну мову, яку називають байт-кодом. Формат байт-коду не залежить від платформи. JVM знаходиться всередині JRE разом з пакетами java (бібліотеками). JVM – це абстрактна машина, на якій виконуються байт-коди Java. Складається із специфікаційного документа, що описує реалізацію JVM, фактичної програми впровадження та екземпляру JVM, де можна запустити основну програму.

Остання версія – Java 14, випущена в березні 2020 року. Але також слід виділити Java 11, яка підтримується в даний час довгостроковою підтримкою (LTS), була реалізованою 25 вересня 2018 року. Однією з найпоширеніших версій Java є 8 версія яка випущена 18 березня 2014 року, ця версія також мала довгострокову підтримку та отримала останне безкоштовне публічне оновлення в січні 2019 року для комерційного використання, а для особистого користування оновлення плануються до грудня 2020 року. Oracle та інші компанії рекомендують видалити

старі версії Java, оскільки вони містять невирішені проблеми безпеки, що тягне за собою великі ризики. Оскільки Java 9 (і 10) більше не підтримується, Oracle радить користувачам негайно перейти до Java 11 (Java 12 також не є LTS).

Мова програмування Java має багато переваг. Такі, як:

– простота використання: Java полегшила життя, видаливши всі складності, такі як вказівники, перевантаження оператора, яке можна побачити на
C ++ або будь-якій іншій мові програмування;

– портативність: Java – незалежна платформа, що означає, що будь-яка програма, написана на одній платформі, може бути легко перенесена на іншу платформу;

– об'єктно-орієнтована мова: все вважається "об'єктом", який має певний стан, поведінку і всі операції виконуються за допомогою цих об'єктів;

– безпечна: весь код після компіляції перетворюється в байт-код, який не читається людиною. і Java не використовує явного вказівника і запускає програми всередині пісочної скриньки, щоб запобігти будь-якій діяльності з недовірених джерел. Це дає змогу розробляти системи або програми, що не містять вірусів;

– динамічність: Java має можливість адаптуватися до розвиваючого середовища, яке підтримує динамічний розподіл пам’яті, за рахунок чого витрата пам’яті зменшується та підвищується продуктивність програми;

– розподіленість: Java надає функцію, яка допомагає створювати розподілені програми. За допомогою віддаленого виклику методу (RMI) програма може викликати метод іншої програми в мережі та отримати вихід звернувшись до методів з будь-якої машини в Інтернеті[8];

– надійність: Java має потужну систему управління пам'яттю. Це допомагає усунути помилки, оскільки вона перевіряє код під час компіляції та виконання;

– висока продуктивність: Java досягає високої продуктивності завдяки використанню байт-коду, який можна легко перевести на рідний машинний код. За допомогою компіляторів JIT (Just-In-Time) Java забезпечує високу продуктивність;

– інтерпретовананість: Java компілюється в байт-коди, які інтерпретуються середовищем виконання Java;

–багатопоточність: Java підтримує кілька потоків виконання (наприклад, легкі процеси), включаючи набір примітивів синхронізації. Це значно спрощує програмування з потоками.

3.1.2 Java бібліотека Spring Framework

Spring Framework – це програмний каркас, який містить у собі контейнери для підтримки інверсії управління[9]. Для роботи програмного забезпечення, яке реалізує розв’язок завдання на дипломний проект, було використано основні технології даного фреймворку:

–Spring Boot. Це забезпечує гнучкий спосіб налаштування Java Beans, конфігурацій XML та транзакцій з базами даних. Він забезпечує потужну пакетну обробку та керує кінцевими точками REST. У Spring Boot все налаштовано автоматично; ніякі конфігурації вручну не потрібні. Spring Boot впроваджує конфігурування на основі анотацій. Полегшує управління залежностями, включає вбудований контейнер сервлетів завдяки якому розворот та конфігурація серверу становиться максимально швидкою[10];

– Spring Data. Це проект, метою якого є уніфікація та полегшення доступу до різних типів постійних сховищ, як реляційних систем баз даних, так і сховищ даних NoSQL. Це полегшує використання технологій доступу до даних, реляційних та нереляційних баз даних та хмарних служб передачі даних. Це парасольковий проект, який містить багато підпроектів, характерних для певної бази даних[11];

– Spring MVC. Spring MVC забезпечує архітектуру патерну Model–View–Controller (Модель – Відображення – Контролер) за допомогою слабо пов'язаних готових компонентів. Патерн MVC розділяє аспекти додатка (логіку введення, бізнес-логіку і логіку UI), забезпечуючи при цьому вільний зв'язок між ними[12].

Spring Boot це проект, побудований на вершині Spring Framework. Він надає більш простий і швидкий спосіб налаштування, налаштування та запуску як простих, так і веб-програм[13].

У базовій системі Spring потрібно налаштувати всі речі для кожної системи. Отже, у розробника може бути багато файлів конфігурації, таких як дескриптори XML. Це одна з основних проблем, яку вирішує для розробників Spring Boot.

Spring Boot розумно вибирає залежності, автоматично налаштовує всі функції, які Spring Boot хоче використовувати, і дає змогу запустити свою систему одним натисканням миші. Крім того, це також спрощує процес розгортання вашої програми.

Дана технологія має багато переваг такі, як:

– скорочує час розробки та збільшує загальну продуктивність команди розробників;

– допомагає автоматично налаштувати всі компоненти для Spring системи промислового рівня;

–полегшує розробникам створення та тестування програм на базі Java, надаючи налаштування за замовчуванням для тестів для модулів та інтеграції;

– уникає написання великої кількості стандартного коду, анотацій та конфігурації XML;

–поставляється з вбудованими HTTP-серверами, такими як Tomcat або

– додає безліч плагінів, які розробники можуть легко використовувати для роботи з вбудованими базами даних і базами даних в пам'яті. Spring Boot дозволяє легко підключатися до таких служб баз даних та черг, як Oracle, PostgreSQL, MySQL, MongoDB, Redis, Solr, ElasticSearch, Rabbit MQ, ActiveMQ та багато іншого;

– додає адміністраторську підтримку, а саме можливість керувати через віддалений доступ до системи.

Головна проблема для багатьох розробників при використанні Spring Boot – відсутність контролю. Він встановлює безліч додаткових залежностей, які, як він

вважає, вам знадобляться. Встановлюючи всі ці додаткові залежності (які іноді залишаються невикористаними), розмір виконуваного файлу розгортання може стати дуже великим.

Однією з основних переваг Spring Framework є підтримка інверсії управління. Інверсія управління – це принцип в розробці програмного забезпечення, при якому управління об'єктами або частинами програми передається в контейнер або середу.

Найчастіше він використовується в контексті об'єктно-орієнтованого програмування.

На відміну від традиційного програмування, в якому наш користувальницький код виконує виклики бібліотеки, IoC дозволяє інфраструктурі контролювати потік програми і здійснювати запити до призначеного для користувача коду[14]. Щоб включити це, фреймворки використовують абстракції з вбудованим додатковим поведінкою. Якщо ми хочемо додати нашу власна поведінка, нам потрібно розширити класи фреймворка або підключити наші власні класи.

Однією з реалізацій інверсії керування є впровадження залежностей (Dependency injection), яке використовується в Spring Framework. Впровадження залежностей (Dependency injection) – це процес, за допомогою якого об'єкти визначають свої залежності, тобто інші об'єкти, з якими вони працюють, тільки через аргументи конструктора, аргументи методу фабрики або властивості, які встановлюються в екземплярі об'єкта після його створення або повернення. Потім контейнер впроваджує ці залежності при створенні компонента. Цей процес в своїй основі є зворотним, звідси і назва Inversion of Control (IoC) самого компонента, який самостійно управляє створенням або розташуванням своїх залежностей за допомогою прямого конструювання класів.

Код чистіше з принципом Dependency injection, і розділення на об’єкти є більш ефективним, коли об'єктам надано їх залежності. Об'єкт не шукає свої залежності і не знає розташування або клас залежностей. Таким чином, класи стають простіше для тестування, зокрема, коли залежності знаходяться на інтерфейсах або абстрактних базових класах, які дозволяють використовувати реалізації-заглушки або макети в модульних тестах.

На даний час існує три способи для впровадження залежностей:

–через конструктор;

–через метод (setter);

–через поле класу.

Переваги даного шаблону проектування:

– дозволяє клієнту гнучкість налаштування. Фіксується лише поведінка клієнта;

–тестування може проводитися за допомогою макетних об'єктів;

–слабо поєднана архітектура;

–зниження складності модуля;

–підвищене повторне використання модуля;

– не вимагає змін у поведінці коду, його можна застосувати до застарілого коду як рефакторинг;

– дозволяє клієнту видалити всі знання про конкретну реалізацію, яку потрібно використовувати. Це більш багаторазовий використання, більш тестований, більш читабельний код;

–дозволяє усунути або принаймні зменшити непотрібні залежності;

–дозволяє одночасно чи самостійно розвиватися;

–зменшує зв’язок між класом та його залежність.

3.1.3 Java бібліотека Swagger

Swagger – це набір інструментів з відкритим кодом, побудованих навколо специфікації OpenAPI, які можуть допомогти розробити, створити, документувати та споживати API REST[15]. До основних інструментів Swagger належать:

– Swagger Editor – редактор на базі браузера, куди можна писати характеристики OpenAPI;

– інтерфейс Swagger – надає специфікації OpenAPI як інтерактивну документацію API;

– Swagger Codegen – генерує заглушки сервера та бібліотеки клієнтів із специфікації OpenAPI.

Специфікація OpenAPI (раніше специфікація Swagger) – це формат опису API для API REST. Файл OpenAPI дозволяє описати весь API, включаючи:

– доступні кінцеві точки (/ користувачів) та операції на кожній кінцевій точці (GET / користувачі, POST / користувачі);

–параметри роботи як вхід і вихід для кожної операції;

–методи аутентифікації;

–контактна інформація, ліцензія, умови використання та інша інформація.

Специфікації API можуть бути записані YAML або JSON. Формат простий у вивченні та читанні як для людей, так і для машин. Впровадження Swagger специфікації за допомогою Swagger UI дозволить користуватися системою безпосередньо з браузера користувача, а також спростить інтеграцію з іншими системами завдяки тому, що кожен розробник зможе власноруч спробувати використати кожен «REST Endpoint», побачити що потребується на вхід та що буде на його виході.

Swagger UI – це один із найпопулярніших інструментів для генерації інтерактивної документації з вашого документа OpenAPI. Інтерфейс Swagger створює інтерактивну консоль API, щоб користувачі могли швидко дізнатися про ваш API та експериментувати із запитами. Крім того, інтерфейс Swagger (це активно керований проект з ліцензією Apache 2.0) підтримує останню версію специфікації OpenAPI (3.x) та інтегрується з іншими інструментами Swagger(рисунок 3.1)[16].

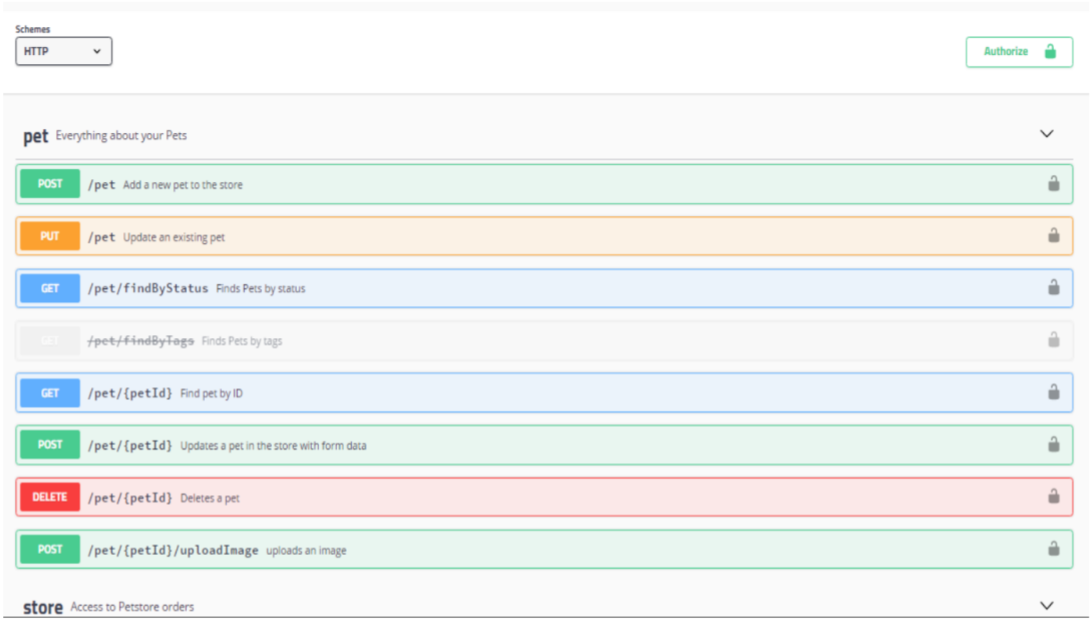

Рисунок 3.1 – Приклад інтерфейсу Swagger UI

Потрібно також зазначити, що Swagger UI динамічно генерує інтерфейс та документує API, завдяки чому розробник економить час на розробку нового інтерфейсу та документації.

3.1.4 Архітектурний стиль REST

Для оптимальної роботи програмного забезпечення, як веб-додатку, була використана архітектура REST.

REST – це архітектурний стиль для проектування мережевих додатків. REST розшифровується як Передача представницького стану (іноді його називають «REST»). Він заснований на протоколі обміну даними з клієнтом і сервером, не залежить від стану, і в переважній більшості випадків використовується протокол HTTP.

Ідея полягає в тому, що замість використання складних механізмів, таких як CORBA, RPC або SOAP, використовується простий HTTP для здійснення запитів між машинами.

HTTP означає протокол передачі гіпертексту. HTTP є базовим протоколом, використовуваним в World Wide Web, і цей протокол визначає, як повідомлення форматуються і передаються, і які дії веб-сервери і браузери повинні виконувати у

відповідь на різні команди. HTTP дозволяє відправляти документи назад і вперед в Інтернеті[17]. І REST – це архітектура, яка використовує протокол HTTP для забезпечення функціональності, яка залежить від вимог програмних додатків або веб-сторінок дуже легким, простим і ефективним способом.

Програми з REST використовують запити HTTP для розміщення даних (створення та / або оновлення), зчитування даних (наприклад, здійснення запитів) та видалення даних. Таким чином, REST використовує HTTP для всіх чотирьох операцій CRUD (Create / Read / Update / Delete)[18]. За допомогою представлення користувач взаємодіє з системою(рисунок 3.2).

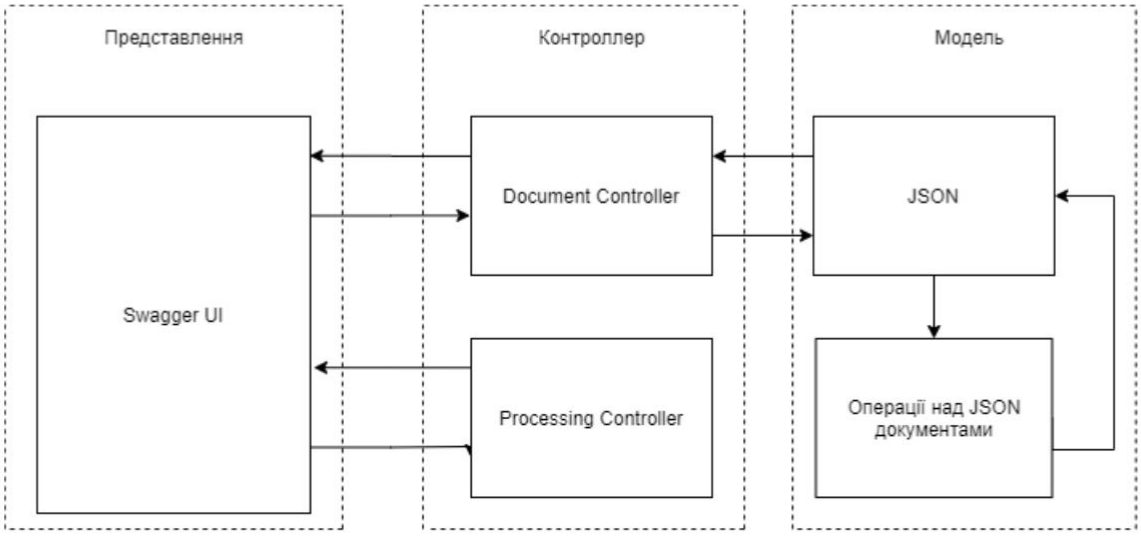

Рисунок 3.2 – Діаграма використання патерну MVC у системі

Загалом використовують такі HTTP запити як:

– GET – запрошує уявлення зазначеного ресурсу. GET-запити повинні тільки отримувати дані, запит є іденпотентним, це означає, що при використанні цього запиту будуть повертатися ті ж самі дані, якщо запит не було змінено;

–POST – передає дані на сервер, запит не є іденпотентним;

– PUT – використовується для повної зміни даних на сервері, запит іденпотентний;

–PATCH – використовується для часткової зміни даних на сервері;

–DELETE – використовується для видалення даних з серверу;

– HEAD – метод аналогічний до GET, за тим винятком що відвовідь з серверу приходить без тіла;

–TRACE – потрібен для налагодження запитів усередині системи;

–OPTIONS – використовується для описання комунікації з ресурсом;

–CONNECT – потрібен при використанні проксі-серверів.

Переваги використання архітектури REST:

– REST веб-сервіси: архітектура REST була реакцією на більш важкі стандарти на основі SOAP[19]. У веб-сервісах REST акцент робиться на простому точковому зв'язку через HTTP за допомогою простого XML. Крім того, REST дозволяє багато різних форматів даних, тоді як SOAP дозволяє лише XML;

–простота REST: архітектуру REST набагато простіше розробити, ніж SOAP. Однією з головних причин популярності REST є простота та простота використання, оскільки це розширення власних веб-технологій, таких як HTTP;

– REST архітектура наближається до дизайну до Інтернету: REST – це архітектурний стиль самого Інтернету, тому розробник, який володіє знаннями з веб-архітектури, природно розвиватиметься в архітектурі REST;

–масштабованість: Оскільки REST забороняє стан розмови, що означає, що ми можемо масштабувати дуже широко, додаючи додаткові серверні вузли за балансиром навантаження;використання API як сервіси HTTP: Коли розробникам потрібні універсальні можливості з мінімальними зусиллями, враховуючи той факт, що API RESTful виставляються як HTTP Services, що практично присутнє майже на всіх платформах[20].

Для впровадження REST у систему було використано один з компонентів Spring Framework, а саме використовуючи анотацію @RestController над класом, виконуючим роль контролера у системі[21]. При використанні анотації також бажано вказати URL контролера, для цього потрібно використати анотацію @RequestMapping це допоможе розділити префікс URL для кожного контролера за його призначенням. Наприклад контролер обробки змісту документів має анотацію @RequestMapping("/v1/process"), а контролер оброблення самих документів має анотацію @RequestMapping("/v1/documents"). Окрім цього важну роль грає префікс «/v1» – він є маркером версії контролеру, це дозволить не зламати зворотну

сумістність при будь-яких кардинальних змінах в контролері або внутрішньої логіки програми.

Реалізовані контролери надають змогу системі оброблювати усі можливі HTTP запити, це робить контролер важливим елементом взаємодії з системою та важливим елементом безпосередньо всередині системи, оскільки контролер бере на себе оброблення запитів які клієнт у вигляді людини або інтегрованої системи надсилає на сервер за допомогою HTTP запитів на визначений URL залежно від бажаного результату дії.

3.1.5 Java бібліотека SimpleJSON

Для виконання дій над JSON документами було використано SimpleJSON Framework який розроблено компанією Google. Бібліотеку реалізовано з дотримкою стандарту RFC 7159[22]. Вона дозволяє швидко формувати JSON документ із вхідних даних контролера та оброблювати його, головні переваги цього фреймворку перелічено нижче:

–гнучка, проста та зручна у використанні, використовуючи інтерфейси

–підтримує потокове виведення тексту JSON;

–аналізатор на основі heap-у;

–висока продуктивність;

–відсутність залежності від зовнішніх бібліотек;

–і вихідний, і двійковий код сумісні з JDK1.2;

–легкість використання;

–легковажність.

3.1.6 База даних ArangoDB

ArangoDB є багатомодельною базою даних. Багатомодельна вона, тому що ArangoDB надає можливості графічної бази даних, бази даних документів,

зберігання ключа-значення в одному ядрі C ++. В ArangoDB користувач може використовувати та вільно комбінувати всі підтримувані моделі даних навіть в одному запиті. Завдяки ArangoDB користувач може легко змінити свою стратегію доступу до даних просто змінивши запит.

Основою багатомодельної моделі в ArangoDB є гнучкість JSON. Користувачі можуть зберігати довільної складності дані та навіть використовувати вкладені властивості в ArangoDB.

Всі дані в ArangoDB зберігаються у вигляді документів JSON та аналогічно структурованих документів, які можна об'єднати в колекції – аналогічно таблиці в реляційних базах даних. ArangoDB можна використовувати як сховище транзакційних документів. Дані можна запитувати за допомогою AQL, мови запитів ArangoDB. AQL підтримує CRUD, агрегацію, складні умови фільтра, вторинні індекси та реальні операції JOIN.

Також ArangoDB надає широкий спектр можливостей баз даних графів: проходження графів, найкоротший шлях, узгодження шаблонів та розподілення графа за допомогою Pregel. Користувачі також можуть взяти результат операції JOIN, геопросторового запиту, пошуку тексту або будь-якого іншого шаблону доступу як вихідну точку для подальшого аналізу графів і навпаки – все в одному запиті, якщо це потрібно. Це перевага багатомодельної бази даних, як ArangoDB. Графік можна візуалізувати та керувати безпосередньо в WebUI ArangoDB. WebUI пропонує безліч конфігурацій для відображення країв і вершин.

Ключові можливості ArangoDB[23]:

– встановити ArangoDB на кластер так само просто, як встановити додаток на мобільний телефон;

– гнучке моделювання даних: моделювання даних у вигляді комбінації пар ключових значень, документів або графіків - ідеально підходить для встановлення відносин в об’єкті;

–потужна мова запитів (AQL) для отримання та зміни даних;

– виконування запитів на декількох документах або колекціях з необов'язковою транзакційною послідовністю та ізольованістю;

– реплікація: налаштовуйте базу даних в конфігурацію головного підлеглого або розподіляйте великі набори даних на декілька серверів;

– налаштування довговічності: можливість додатку вирішувати, чи потребує він більшої довговічності або більшої продуктивності;

– ArangoDB використовує всю потужність сучасного обладнання для зберігання даних, як SSD та великі кеші;

– ArangoDB може бути легко розгорнутий як відмовостійкий розподілений державний апарат, який може служити мозком тварин розподілених приладів;

–ArangoDB – це відкритий код (Apache License 2.0).

Так як ArangoDB зберігає дані в такому NoSQL форматі як JSON можна сказати, що ця база даних ідеально підходить до розроблюваної системи, оскільки типові CRUD операції зі ключ-значення даними можуть виконуватися ефективно, підвищуючи загальну швидкість системи.

3.2 Обґрунтування вибраних технологій

Використання Java у поєднанні зі Spring обумовлено в першу чергу можливістю побудови ресурсо-ефективних програм з підтримкою багатьох патернів проектування у неявному вигляді за допомогою Spring. Окрім цього Spring значно спрощує побудову систем у вигляді серверу. Завдяки Java буде розроблено систему, яка може бути запущена на майже будь-якій платформі включаючи UNIX та Windows системи, разом з цим Java ефективно використовує пам’ять та працює не повільніше ніж такі аналоги як C# та інші.

Існує досить багато бібліотек для оброблення JSON на Java, але майже усі вони мають досить багато додаткових функцій які не будуть використані в системі. Бібліотека SimpleJSON не залежить від інших бібліотек, працює з Java heap та підтримує JSON специфікацію RFC4627, саме тому SimpleJSON є ідеальним вибором як бібліотека для використання у розроблюваній системі.

Вибір бази даних ArangoDB обумовлений тим, що ця БД зберігає дані у форматі JSON, тому не буде ніякої потреби в будь якому форматуванні існуючих даних, що добре позначиться на швидкості та ефективності використання даної бази даних.

REST використано з тої причини, що це найпоширеніший та простий у розумінні та використанні архітектурний стиль[24]. Це дасть змогу розробникам максимально швидкої інтеграції розроблюваної системи до будь-якої іншої[25].

Swagger використано для використання та можливістю тестування API іншими розробниками для поліпшення інтеграції з іншою системою. Окрім цього Swagger забезпечує легковажний та зручний UI інтерфейс для користувача.

3.3 Формат зберігання даних

Система має лише одну модель об’єкту для зберігання в базі даних. Це сутність документа. Сутність складається з унікального ідентифікатора у форматі UUID та самого документа(рисунок 3.3)[26].

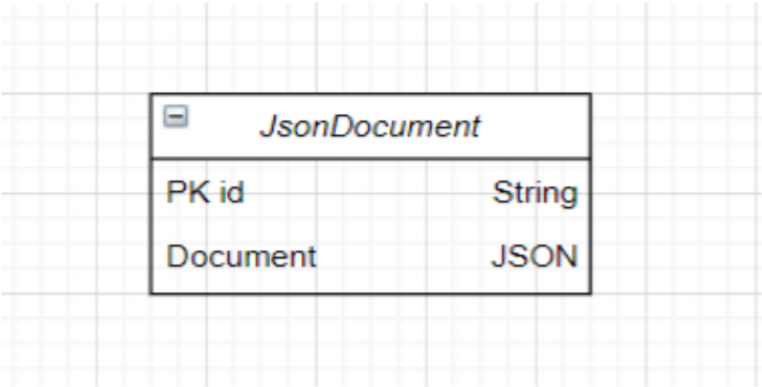

Рисунок 3.3 – Формат зберігання даних

3.4 Структура системи

За допомогою вибраних технологій було побудовано систему. Взаємодія з системою відбувається за допомогою HTTP протоколу. Користувач використовує систему за допомогою Swagger UI, це дозволяє йому не генерувати REST запити

власноруч та делегувати це на Swagger UI. Усі надіслані REST запити до системи у першу чергу потрапляють до Dispatcher Servlet. На цьому рівні вирішується до котрого контролеру далі піде запит, це вирішується за допомогою URL запиту та URL адрес існуючих контролерів, якщо знаходиться збігання то запит направляється від Dispatcher Servlet до існуючого контролеру, інакше користувачу або інтегрованій системі повертається HTTP статус або, в рідких випадках запит направляється до середньо-рівневих сервісів(рисунок 3.4)[27].

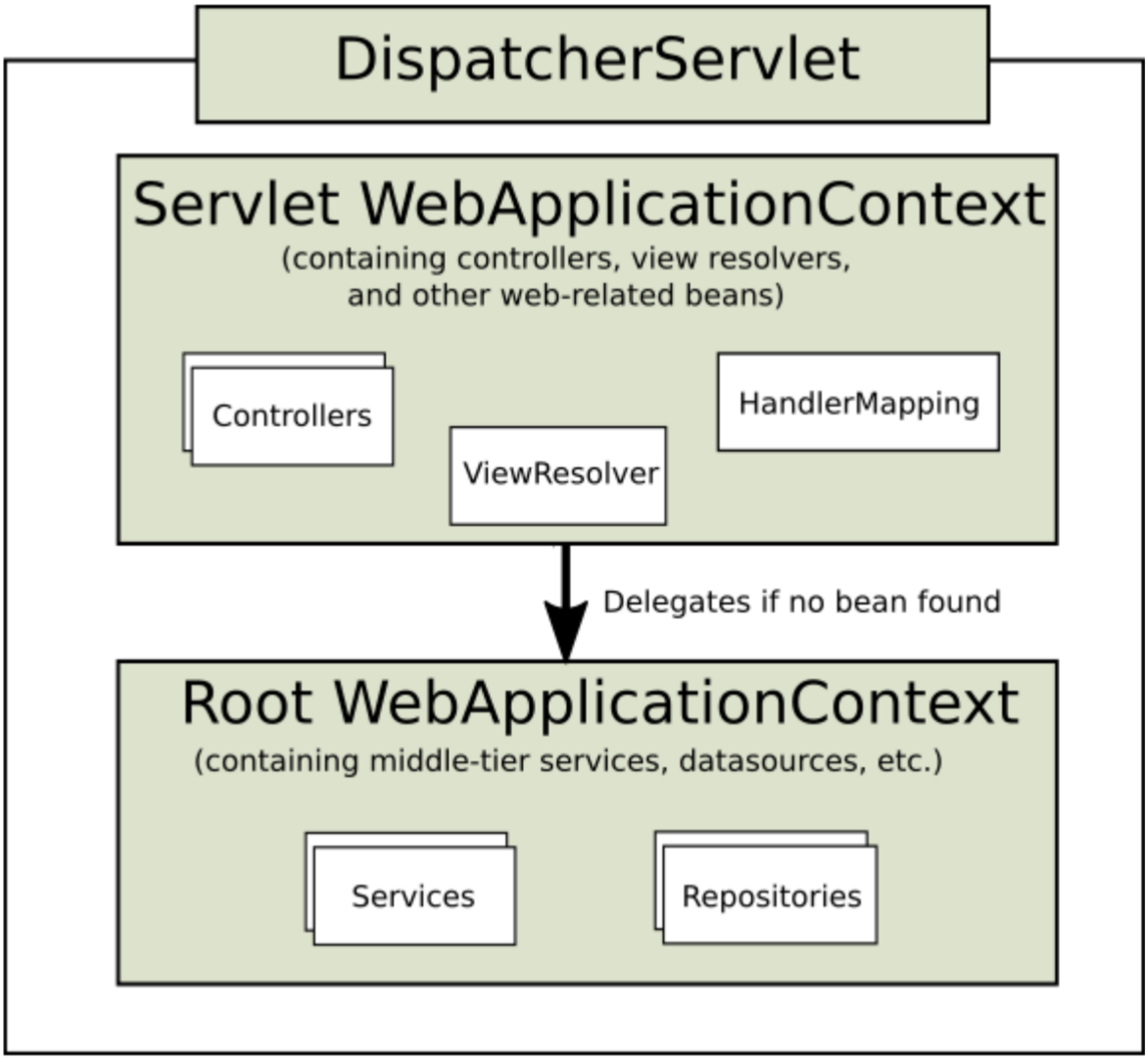

Рисунок 3.4 – Структура Dispatcher Servlet

При потраплянні запиту до контролеру продовжується пошук кінцевої точки по URL, при її знаходженні йде валідація запиту – перевірка його тіла, формату та іншого. Якщо запит провалідовано, і контролер приймає рішення, що система може обробити запит то він відправляється нижче по ланцюжку до сервісів.

На рівні сервісів існує 2 сервіси, вони виконують роботу по остаточній валідації запиту на більш низькому рівні та оброблення документів. Сервіс по обробленню вмісту документів виконує модифікацію існуючого документу завдяки взаємодії з Java бібліотекою SimpleJSON. Інший сервіс виконує такі дії над існуючими документами як створення, видалення та завантаження. При виконанні дії над документами сервіси спочатку перевіряють чи існує такий документ задля того щоб своєчасно попередити користувача про помилку та не виконувати непотрібних дій. При виникненні помилки сервіс відправляє рекомендацію до формування відповіді контролеру помилок. Контролер помилок у свою чергу формує відповідь та надсилає користувачеві статус код помилки та повідомлення про помилку.

Сервіси не можуть функціонувати без драйверу зв’язку з базою даних ArangoDB, він слугує як прошарок між розроблюваною системою та базою даних та використовується при усіх діях над документами. Драйвер зв’язку з базою даних являє собою систему яка має можливість відправляти запити по мережі безпосередньо до самої бази даних ArangoDB. Драйвер зв’язку знаходиться на найнижчому рівні в системі. Для отримання запитів у ArangoDB налаштовано REST API, завдяки якому виконується перевірка чи запит сформовано правильно та чи існує документ над яким користувач збирається виконати дію. Якщо документ існує то база даних виконує дію над документом та повертає успішну відповідь до системи, далі у системі буде сформовано успішну відповідь до користувача для інформування що запит оброблено правильно. Інакше база даних ArangoDB сформує відповідь з інформацією про помилку, яка буде оброблена системою та повернена користувачеві разом з повідомленням про помилку для того, щоб користувач мав змогу виправити помилку та спробувати надіслати запит до системи ще раз.(рисунок 3.5).

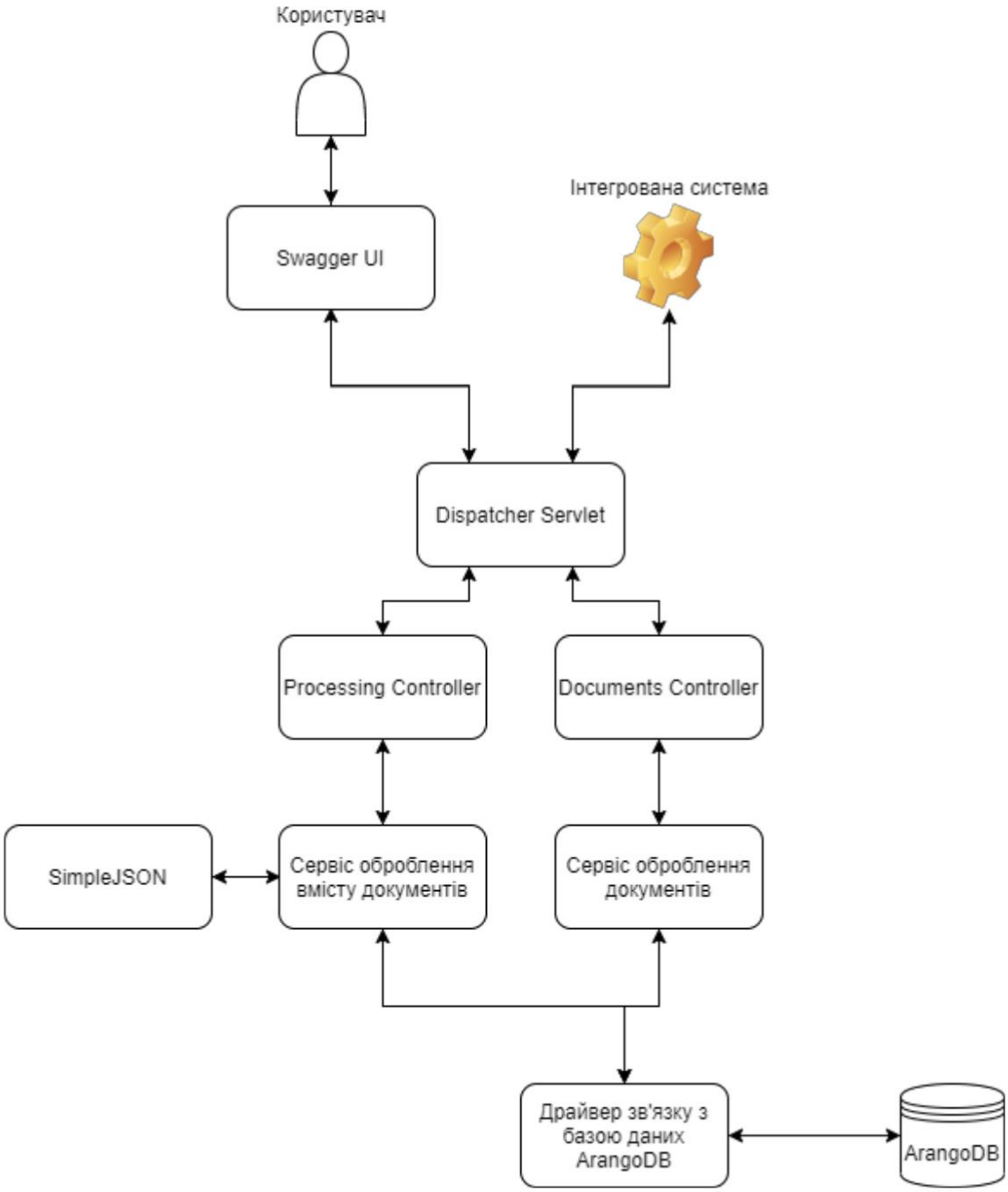

Рисунок 3.5 – Структурна схема системи

3.5 Сценарії використання системи

Система має 2 основних сценарії використання, так як існують варіанти взаємодії з системою як користувача так і інтегрованої системи, то потрібно розглянути кожен окремо.

При використання системи користувачем передбачаються наступні умови: користувачу потрібно змінити власноруч дуже великий JSON документ. Користувач знає структуру документу, але, він зіштовхується з проблемами вже при спробі хоча би відкрити документ використовуючи такі поширені редактори як Notepad++, Visual Code та інші. Таким чином користувачу стає дуже складно виконати таку просту задачу як зміна JSON документу і перед ним стає потреба у системі, яка могла б виконувати дії над документами з мінімальним навантаженням на комп'ютер. Для забезпечення мінімального навантаження на комп'ютер, «редактор» не повинен використовувати форматування або відображення документу у реальному часі.

Іншим сценарієм використання системи користувачем є додаткова потреба у зберіганні документів не на локальному комп'ютері. Це може бути потрібно користувачу при потребі редагуванні та зберіганні великої кількості великих документів. У такому сценарії до редагування цих документів додається ще й необхідність їх зберігання, що вимагатиме додатково багато місця на локальному комп'ютері.

Тому оптимальним варіантом для вирішення даних задач для користувача буде використання системи оброблення JSON документів. Головною перевагою розроблюваної системи стає можливість редагування документу без потреби його відкривання на локальному комп'ютері, завдяки чому користувач у десятки та сотні разів швидше виконувати дії над великими документами. Ще одною, але не менш важливою можливістю системи є зберігання документів.

При використанні системи іншою системою, тобто при інтеграції розроблюваної системи до іншої системи передбачаються наступні умови: при розробленні систем на мікросервісній архітектурі виникає проблема навантаження мережі при, наприклад, ситуаціях коли один сервіс повертає об'єкт іншому сервісу для виконання певних дій, після цього об'єкт надсилається до третього сервісу і далі – нечастий випадок, але існуючий. Окрім навантаження мережі у сервісі з'являється потреба у інструменті який і буде виконувати дії над документом, для цього завжди використовується бібліотека для редагування

документів залежно від обраної мови програмування для сервісу. Використання різних бібліотек для виконання одних і тих же дій над документом призведе у першу чергу до складності підтримки сервісів загалом, також з’явиться проблема дублювання коду, яка може бути достатньо розвинутою коли перелік сервісів йде на десятки[28].

В такому випадку оптимальним рішенням було би використовувати окремий сервіс у який буде винесено можливість редагування документів а також їх зберігання. Це допоможе не пересилати документ від сервісу до сервісу для редагування його частини, а редагувати одразу. Після редагування кожен сервіс може запросити результуючий документ для його подальшого використання.

3.6 Взаємодія з системою

Як було зазначено, взаємодія з системою відбувається по REST API, тому для визначення чи було виконано дію користувачем або інтегрованою системою успішно – було впроваджено використання HTTP статусів.

HTTP статус код – це трицифрове ціле число, де перша цифра статус коду визначає клас відповіді, а останні дві цифри не виконують жодної ролі категоризації.

Існує 5 значень для першої цифри:

–1xx: Інформаційні, означають, що запит отримано, і процес триває;

– 2xx: Успішні, означають, що запит було успішно отримано, суть запита зрозуміла та запит був оброблений без помилок;

– 3xx: Переадресаційні, означають, що для виконання запиту необхідно зробити подальші дії;

– 4xx: Помилки користувача, означають, що запит містить неправильний синтаксис або не може бути виконаний;

– 5xx: Помилки серверу, означають, що серверу не вдалося виконати явно допустимий запит;

У системі явно використано статус коди 2xx та 4xx для повернення статусу результату операції або дії, наприклад серед 2xx кодів використано такий код як 202 Accepted, він інформує, що запит прийнято та більш ніяких дій зі сторони користувача не потрібно.

Уся взаємодія з системою виконується тільки через REST, іншого варіанту не призначено по тій причині, що не буде дотримано оптимальне функціонування системи як веб-сервісу, та може бути ускладнено налаштування системи, її розгортування та використання(рисунок 3.6).

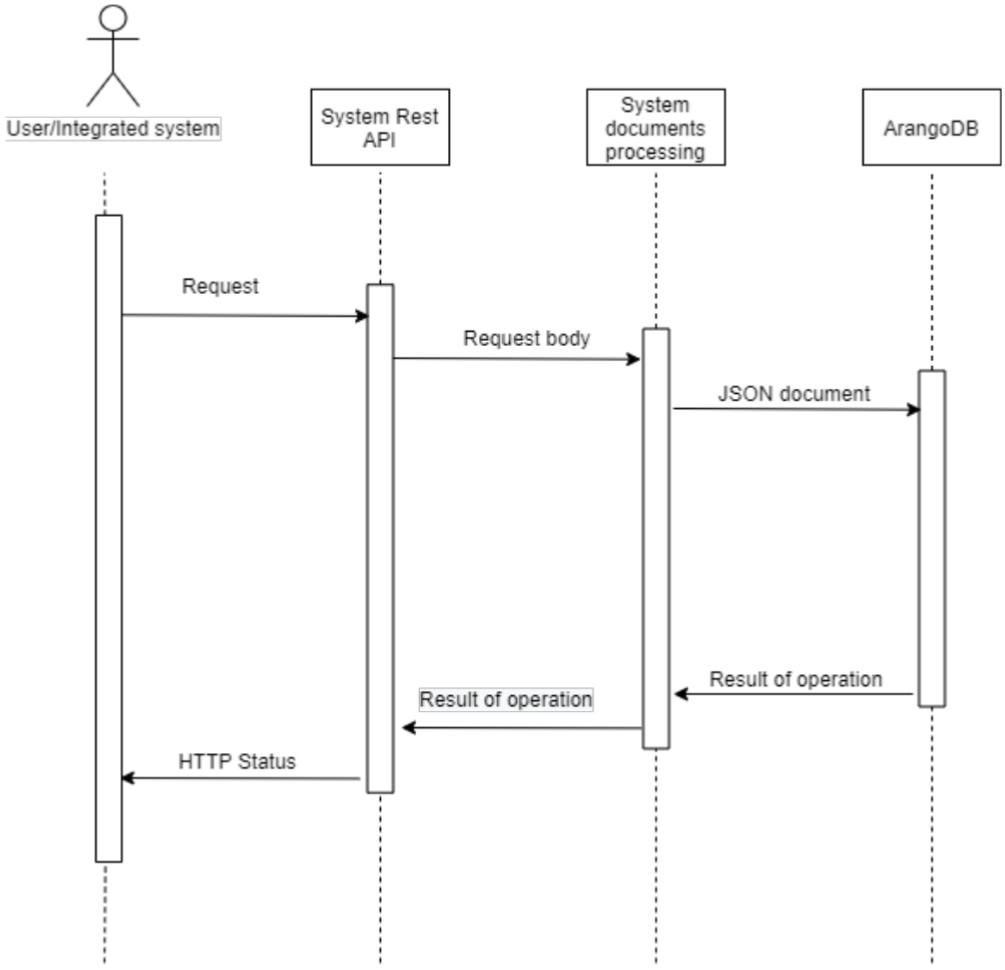

Рисунок 3.6 – Діаграма послідовності взаємодії користувача/інтегрованої системи з системою

За діаграмою послідовності взаємодії користувача з системою видно, що система не виконує жодних дій доки не отримає запит від користувача або інтегрованої системи. При отриманні запита на одній з кінцевих точок REST API система почне оброблення документу, в залежності від вибраної користувачем або

інтегрованої системою дії. Після оброблення JSON документ буде збережено у базу даних ArangoDB за допомогою запиту зі сторони системи до бази даних з певними параметрами, залежно від вибраної дії. База даних приймає запит, оброблює та зберігає документ, який вона отримує у тілі запита. Результатом запиту до бази буде повернено до більш високо рівню системи, позначеного як System documents processing, На цьому рівні йде аналіз відповіді від бази даних та первісна підготовка результату дії, який буде повернено користувачеві. Результат первісного аналізу дії та його доповнення виконує рівень REST API системи, який вже і формує результат дії користувача.

Для безпомилкового використання системи користувач повинен переконатися, що він використовує відповідний до його вимог контролер. Для завантаження або видалення документу з системи потрібно використовувати DocumentController, для цього в запиті до системи потрібно вказати URL який буде складатися із адреси системи до якої буде конкатеновано “/v1/documents”, що і є вказівником на DocumentController. Для модифікації існуючих документів потрібно використовувати ProcessingController, для цього в запит до системи потрібно вказати URL який буде складатися із адреси системи до якої буде конкатеновано “/v1/processing”, що і є вказівником на ProcessingController.

Після закінчення системою оброблення запиту формується відповідь у форматі вихідних даних JSON та/або статусу виконання дії(Д 2).

3.6.1 Взаємодія користувача з системою

Для початку використання системи користувачу потрібно відкрити браузер та ввести у строку пошуку адресу системи. Якщо систему встановлено на локальному комп’ютері, то достатньо буде запустити систему, ввести адресу http://localhost:8080/swagger-ui.html#/ та натиснути Enter. Після цих дій користувач побачить головну сторінку системи. Для користувача будуть доступні 2 контролери, а саме document-controller та processing-controller(рисунок 3.7).

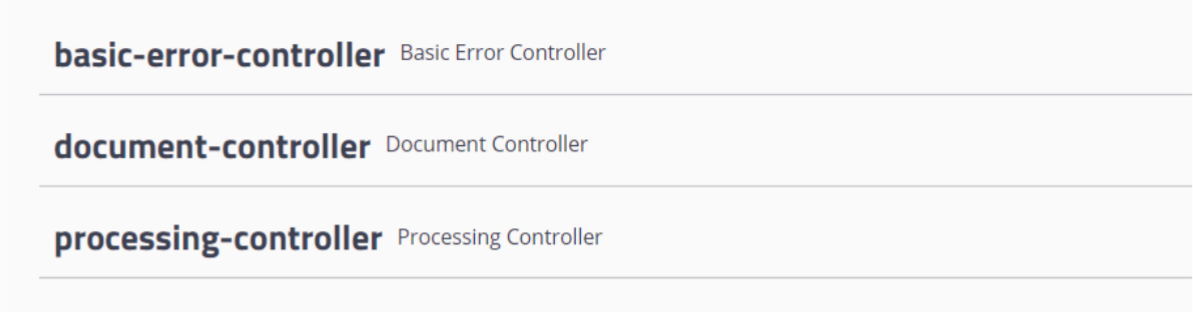

Рисунок 3.7 – Доступні для користувача контролери

Натиснувши на будь-який з них користувач може розвернути усі їх доступні методи. Переглянувши кожен контролер користувач може побачити які дії система може виконувати над документами та визначити той, який потрібен саме йому. Система не має безлічі можливих дій над документами, але при використанні багатьох дій одразу користувач може досить гнучко та швидко модифікувати існуючий документ без звертання до сторонніх систем. Контролери мають надписи на кожній дії, тому користувачу буде легко знайти потрібну дію. Коли користувач знайде її йому потрібно буде натиснути на область інтерфейсу цієї дії для розгортання та подальшого використання. Перелік можливих дій зображено на рисунку 3.8.

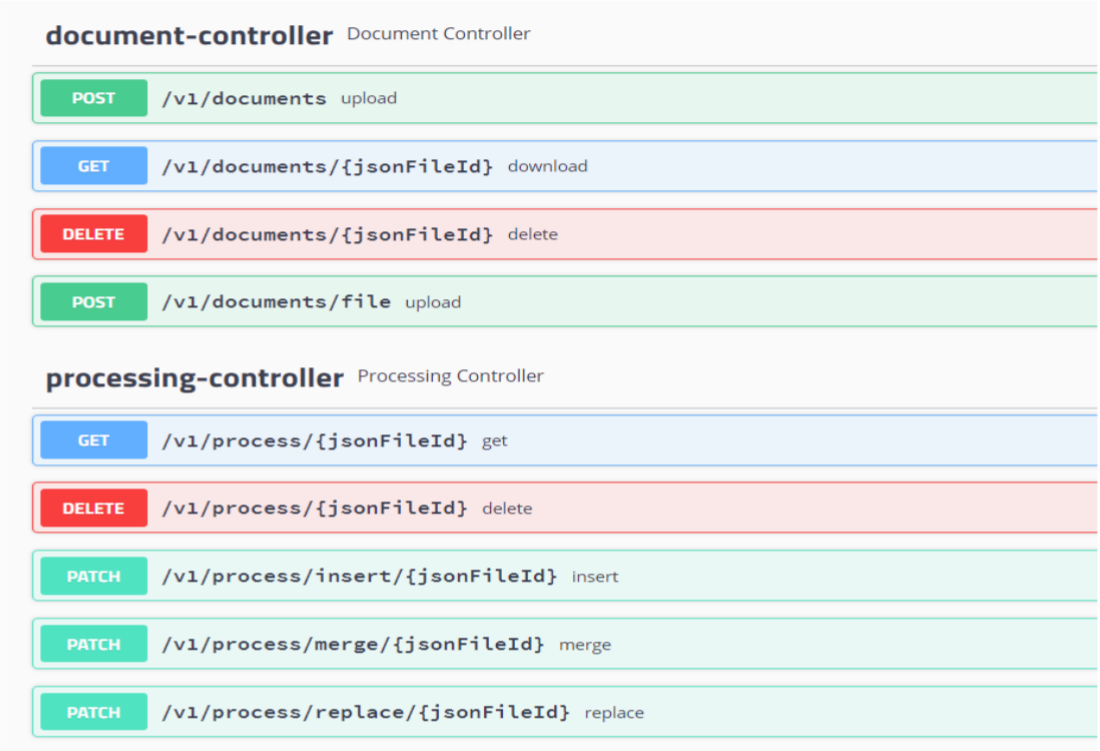

Рисунок 3.8 – Перелік можливих дій

При виборі дії, а саме при натисканні на неї користувачу буде показано усі обов’язкові до заповнення поля та можливі відповіді системи у вигляді статус кодів, що особливо буде зручно для розробників. У прикладі буде використано дію GET у processing-controller(рисунок 3.9)(рисунок 3.10).

Рисунок 3.9 – Обов’язкові до заповнення поля

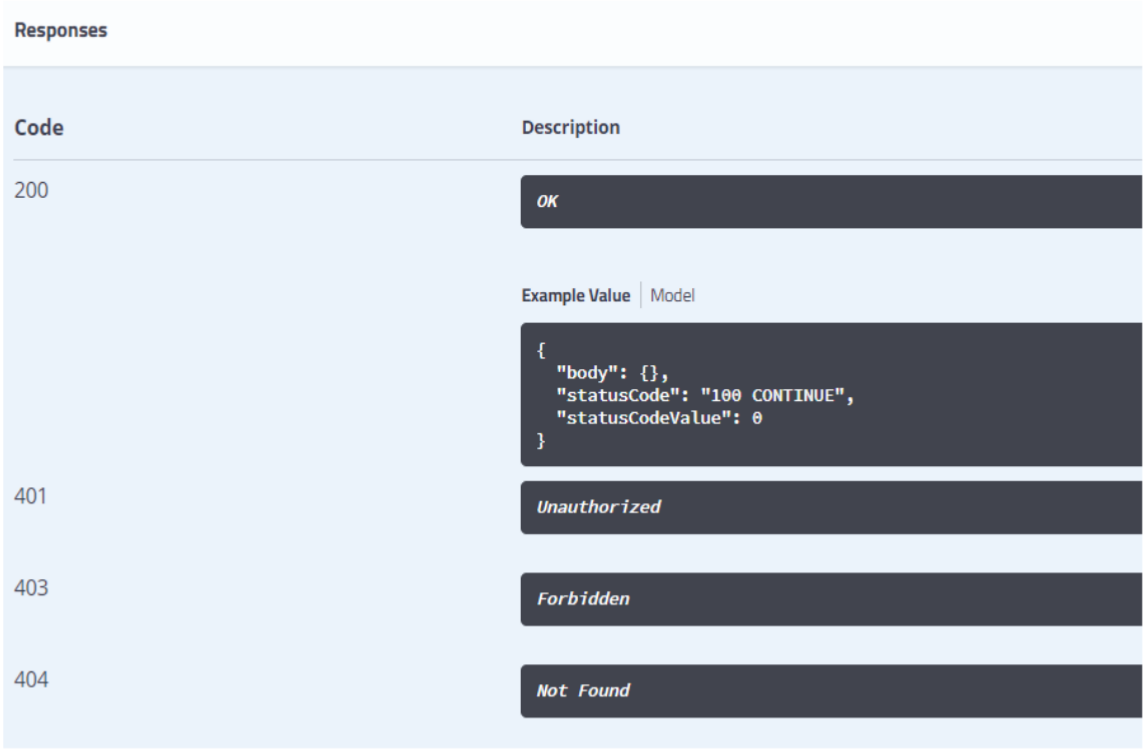

Рисунок 3.10 – Статус коди відповіді системи

При виконанні дій над вже завантаженими документами до системи передбачається що користувач знає унікальний ідентифікатор документу у системі, який він введе у поле «jsonFileId». Інше поле «path» повинно бути заповнено існуючим у документі шляхом. Передбачається, що користувач знає про існуючі

шляхи документу, інакше він може завантажити увесь документ та спробувати знайти потрібний йому шлях. Шлях заповнюється згідно стандарту заповнення у вигляді язику запитів JSONPath. Цей язик запитів дозволяє досить просто та швидко використовувати дії зчитування над документом у будь-якій його частині, в залежності від того як був сформований запит та від документу на який буде застосовано сформований запит(рисунок 3.11).

Рисунок 3.11 – Заповнені поля

Результатом виконання дії системою буде код 2xx серії та Response Body, який буде сформовано в залежності від застосованої до документу дії, в деяких випадках Response Body може не бути(рисунок 3.12).

Рисунок 3.12 – Успішне виконання запиту

При виникненні помилки при запиті до системи, наприклад, якщо документ не було знайдено по переданому унікальному ідентифікатору – користувач отримає відповідний статус та відповідь від системи(рисунок 3.13).

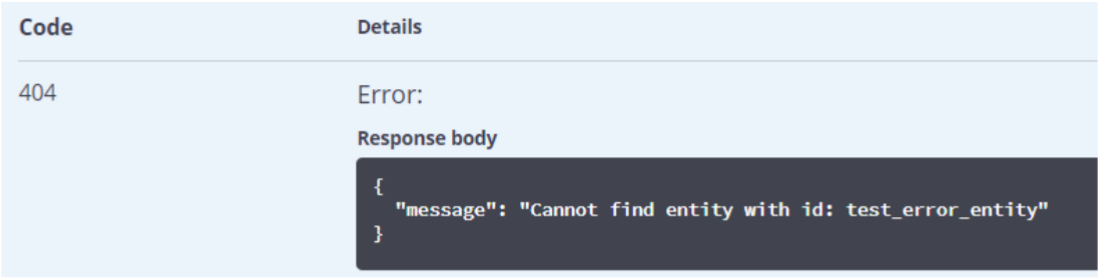

Рисунок 3.13 – Отримання помилки при виконанні запиту

3.6.2 Взаємодія інтегрованої системи з системою

Для інтегрування системи оброблення документів до зовнішньої системи розробник повинен визначитися з тими функціями системи, які будуть використовуватися. Для цього розробник повинен звернутися до Swagger UI по URL http://localhost:8080/swagger-ui.html#/. На головній сторінці буде доступно перелік основних контролерів та методів. На кожному з доступних методів позначено тип HTTP запиту який буде використано при використанні метода. Також на кожному з методів зазначено його URL. На цьому етапі розробник остаточно визначається з тим методом або методами які йому потрібно використати(рисунок 3.14).

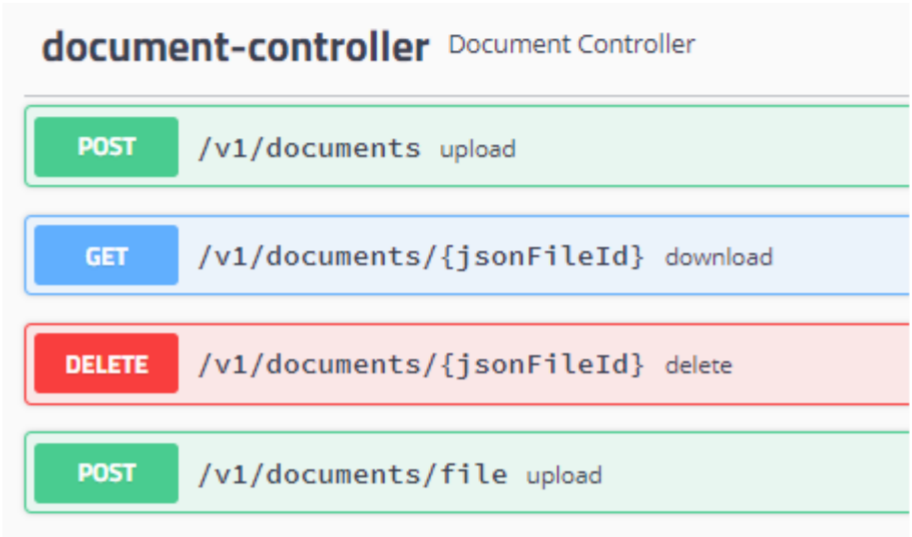

Рисунок 3.14 – Перелік методів та їх URL

Далі розробник формує URL для запиту із своєї системи. Буде розглянуто приклад формування запиту для видалення документу з системи. Для цього розробнику потрібно конкатенувати адресу сервісу з адресою метода. Якщо система розгорнута локально, то її адреса буде http://localhost:8080/, саме до неї потрібно конкатенувати «/v1/process/», у результаті буде отримано URL «http://localhost:8080/v1/process/». Для того щоб URL був повністю сформований потрібно додати унікальний ідентифікатор документу, як це зазначено на методі у формі «{jsonFileId}». Результуючим URL буде «http://localhost:8080/v1/process/90598a16-36dd-47e1-9e6c-6f0d394d1d41».

На даному етапі розробнику залишається тільки коректно вибрати HTTP метод запиту, в нашому випадку це буде «DELETE». Після цього розробнику залишається лише надіслати запит(рисунок 3.15).

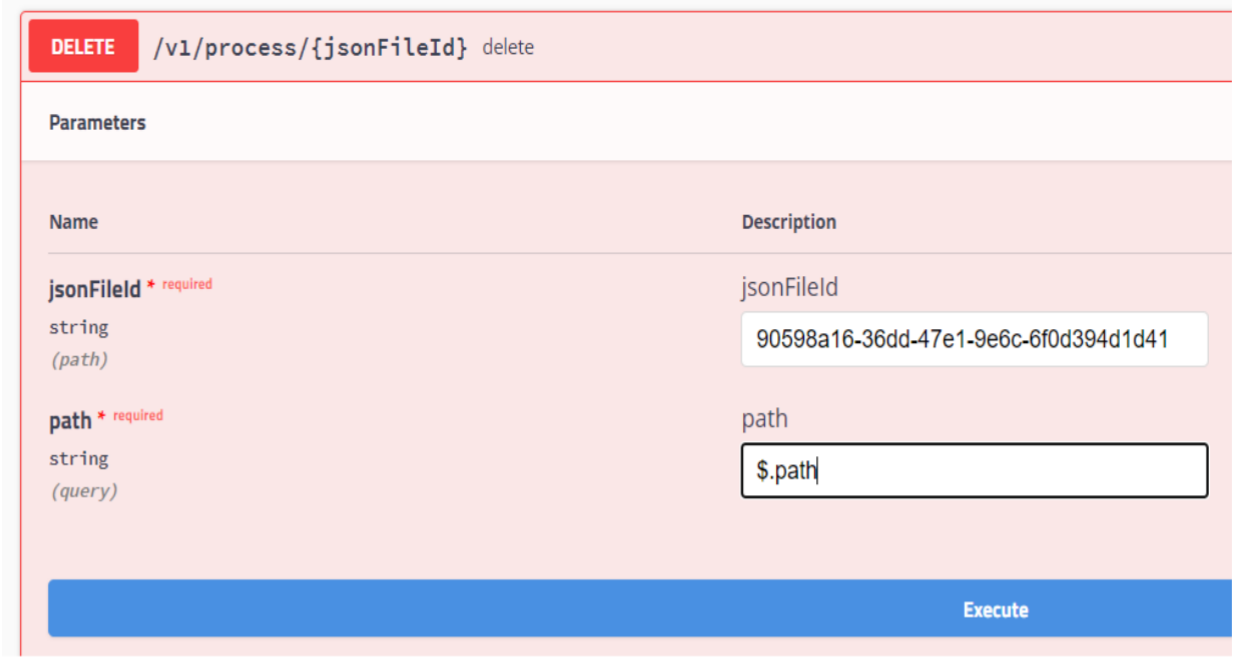

Рисунок 3.15 – Метод DELETE

Існує ще один сценарій будування запиту, це відбувається в тих випадках, до системи потрібно також надіслати тіло запиту. Для прикладу буде використано метод PATCH контролеру processing-controller(рисунок 3.16).

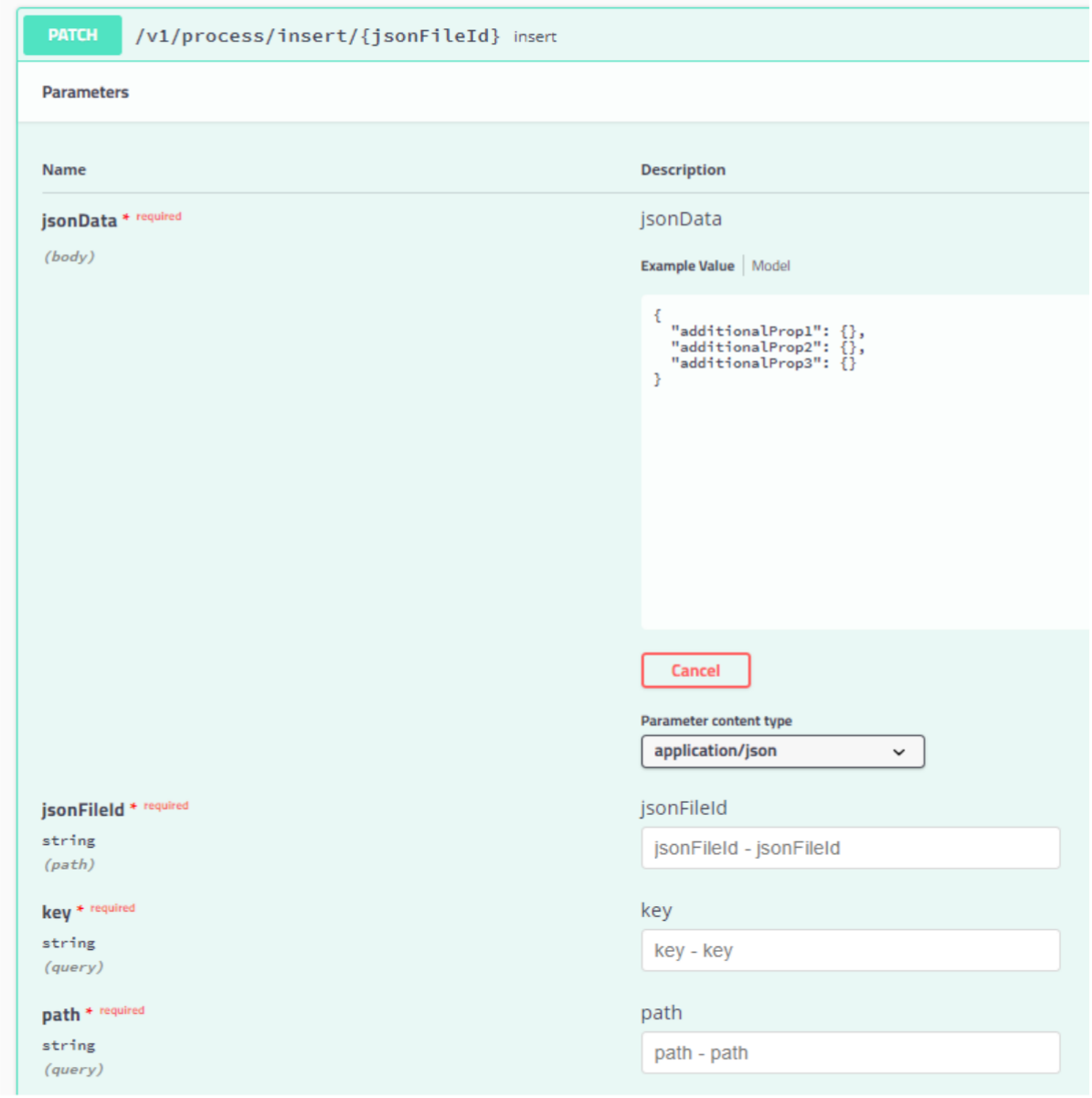

Рисунок 3.16 – Метод PATCH з тілом jsonData

У цьому випадку розробнику потрібно не тільки вказати jsonFileId в URL запиту, а ще й вказати його тіло у вигляді jsonData. Swagger дозволяє розробнику побачити у якому вигляді потрібно надсилати тіло запиту. Тобто наступне тіло запиту буде зрозуміло для системи та буде успішно оброблено:

```
{
  "additionalProp1": {},
  "additionalProp2": {},
  "additionalProp3": {}
}
```

Також за допомогою Swagger UI розробник може побачити відповідь від системи та обробити її у своїй системі, наприклад при надсиланні запиту система повернула статус код 204, така ж відповідь буде повернута і при надсиланні коректного запиту на цей ж метод із системи розробника(рисунок 3.17).

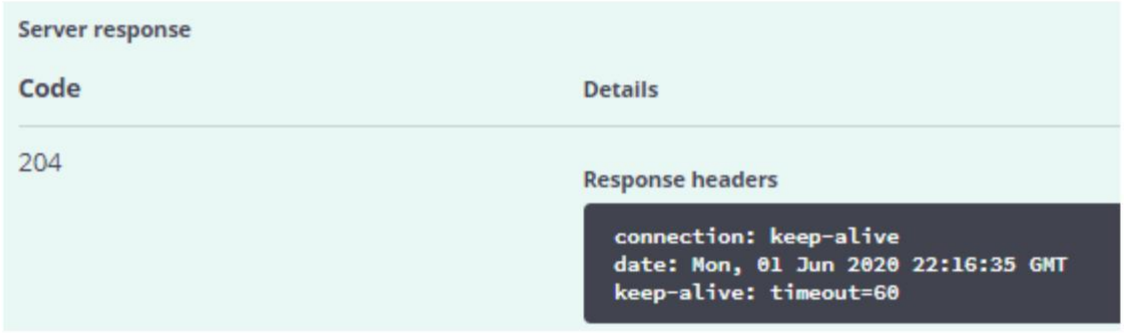

Рисунок 3.17 – Відповідь від системи

3.7 Висновки до розділу

У розділі проведено опис та обґрунтування використаних технологій для розроблення системи оброблення документів. Систему реалізовано на Java у купі з використанням Spring Framework та внутрішніх Spring компонент. Використано 8 версію Java як найбільш стабільну та популярну на сьогоднішній день. Для збереження інформації використано NoSQL базу даних ArangoDB. Для виконання головної задачі системи у вигляді оброблення JSON документів використано бібліотеку від Google під назвою SimpleJSON Framework, яка ідеально підходить під вимоги системи. Також були використані такі стилі як REST та MVC для функціонування системи як веб-сервісу. Було розглянуто та сформовано структуру системи, наведено її сценарії використання як користувачем, так і іншою інтегрованою системою.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Підготовка до випробування системи

Для тестування було виконано запущено базу даних ArangoDB(рисунок 4.1). Після цього запущено систему оброблення JSON документів.

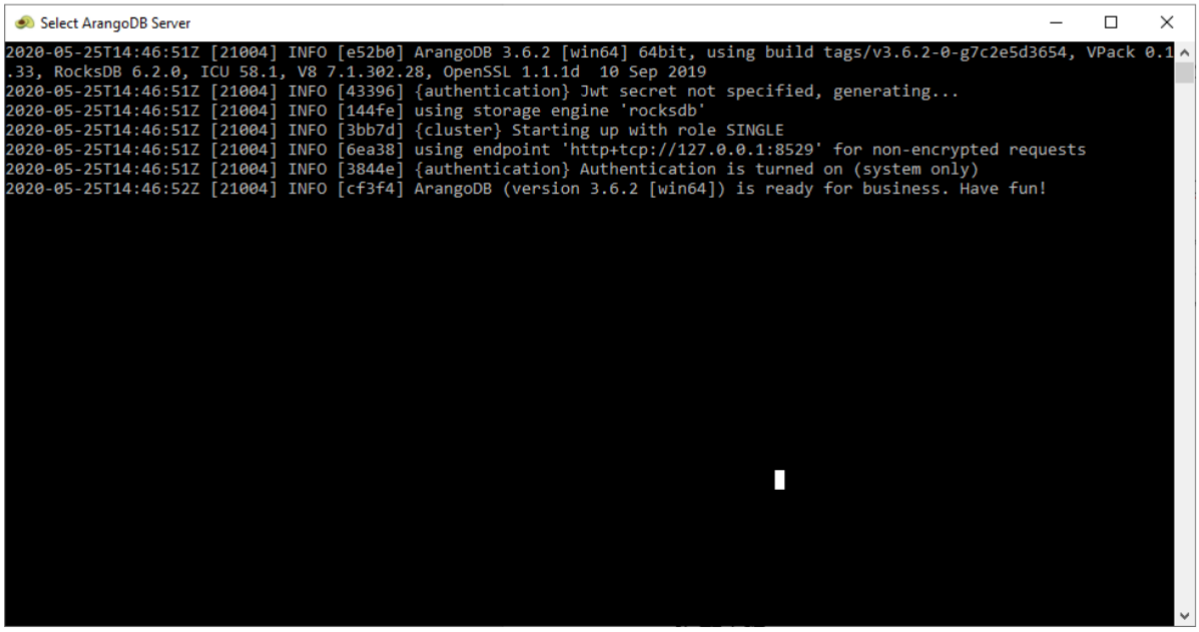

Рисунок 4.1 – Ініціалізація бази даних ArangoDB

В якості інструменту тестування було використано Postman, та Swagger для демонстрації справної роботи системи незалежно від клієнту(рисунок 4.2). Для тествування у Swagger використано Google Chrome. Postman являє собою потужний інструмент для тестування API, він підтримує усі базові HTTP запити, а саме:

–GET

–POST

–PATCH

–PUT

–DELETE

–HEAD

–OPTIONS

–TRACE

–CONNECT

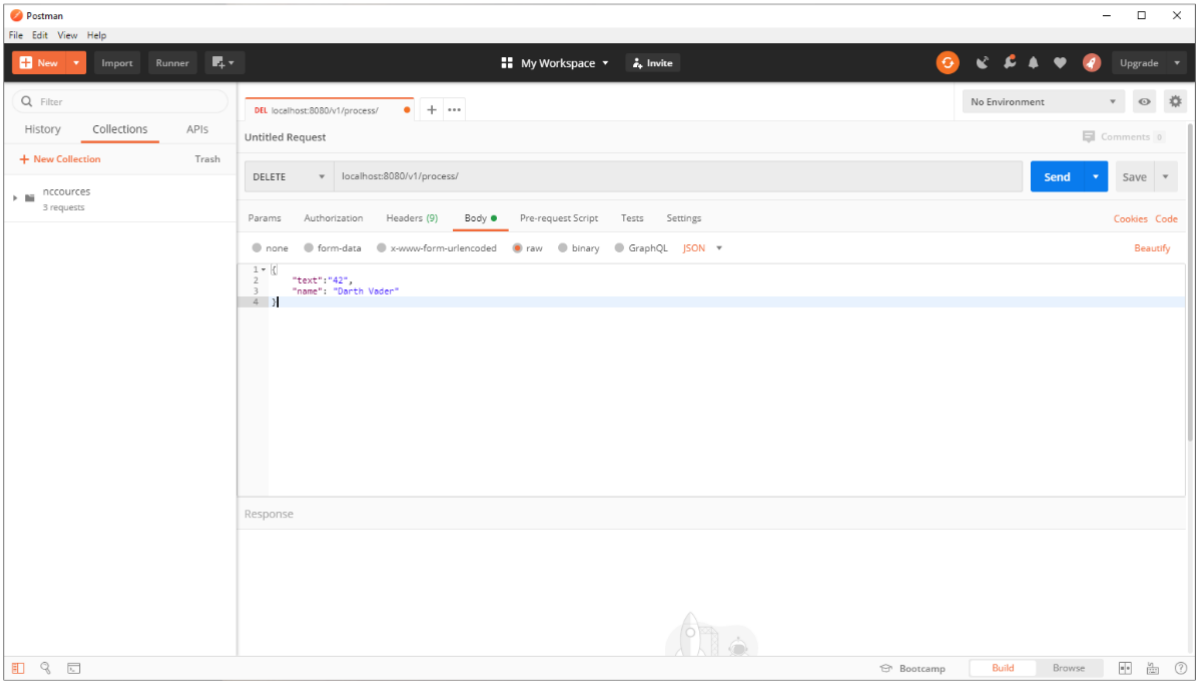

Рисунок 4.2 – Головний екран інструменту Postman

Для початку роботи з системою було сформовано початковий JSON документ у вигляді:

```
{
        "big_brother" : {
          "is" : "watching you",
          "says" : "twice two equals five"
         }
       }
```

4.2 Завантаження документу у вигляді тексту до системи

Одним з варіантів завантаження документу до системи є заватаження документу у вигляді тексту.

4.2.1 За допомогою Postman

При використанні Postman документ було вміщено у тіло запиту, для завантаження документу використано DocumentController, для цього в URL запиту було задано шлях “/v1/documents” та метод запиту POST(рисунок 4.3).

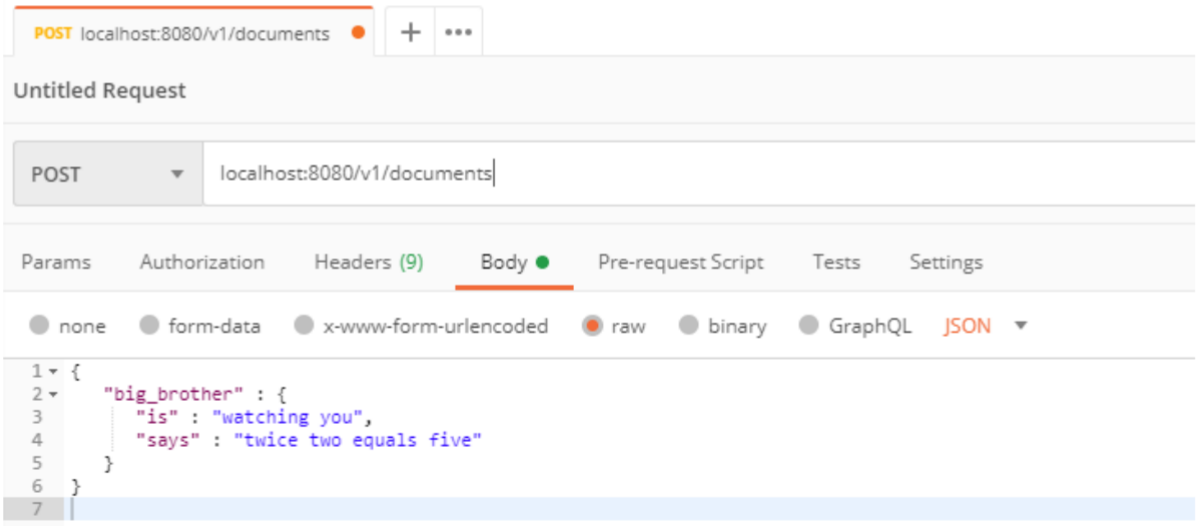

Рисунок 4.3 – Сформований запит до системи

Запит надіслано на сервер, отримано відповідь(рисунок 4.4):

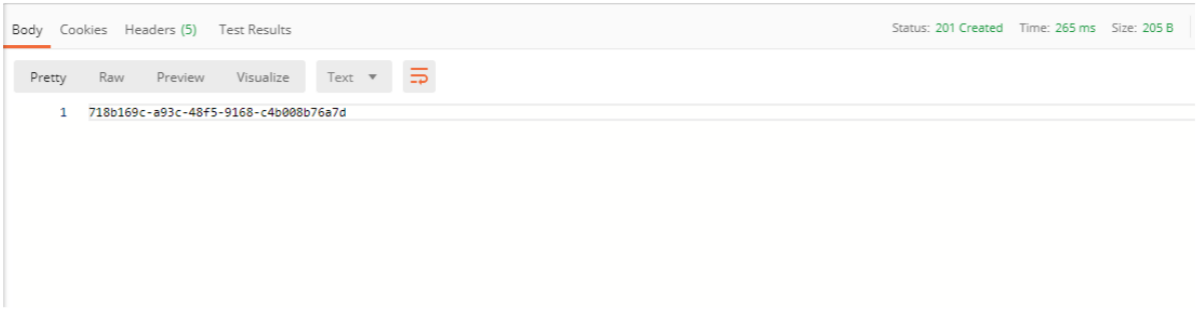

Рисунок 4.4 – Відповідь від системи

Отримано статус 201, що свідчіть про успішну операцію, та результат у вигляді UUID, що є ідентифікатором завантаженого документу.

Для перевірки чи дійсно создано документ потрібно звернутися до бази даних ArangoDB, знайти документ з тим ключем який повернула система, в нашому випадку це 718b169c-a93c-48f5-9168-c4b008b76a7d(рисунок 4.5).

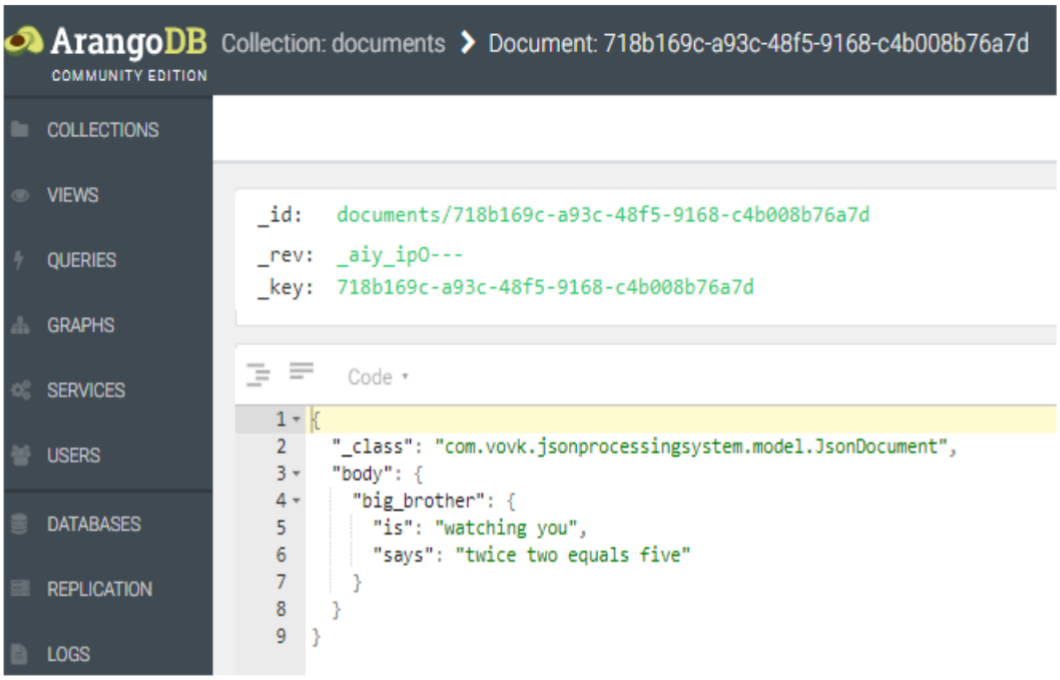

Рисунок 4.5 – Шуканий документ

4.2.2 За допомогою Swagger

У Google Chrome введено адресу системи у вигляді http://localhost:8080/swagger-ui.html#/. Вибрано document-controller(рисунок 4.6).

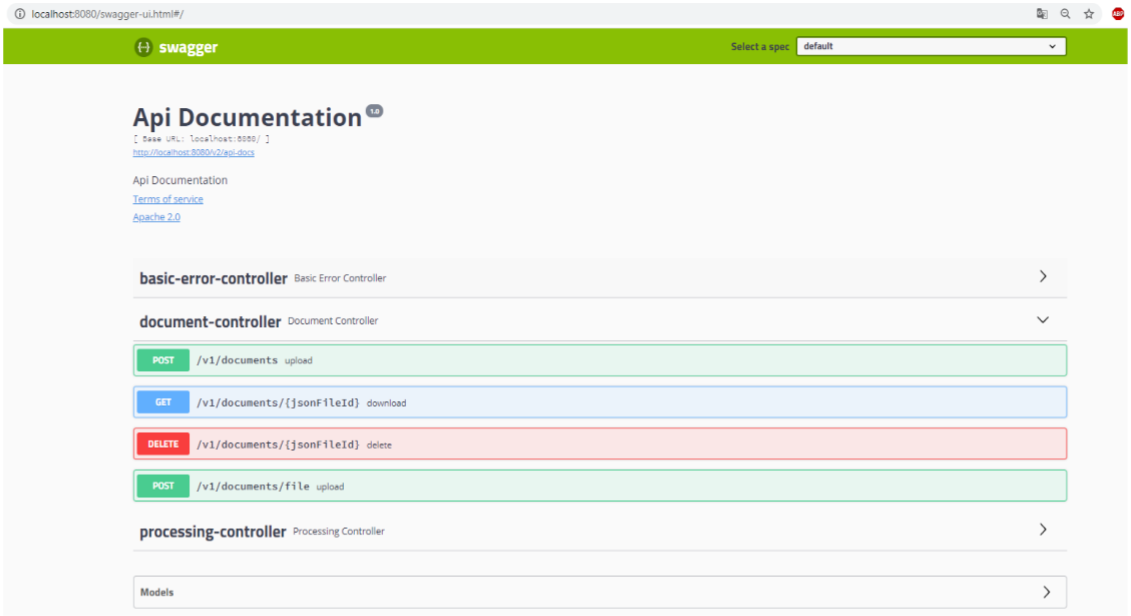

Рисунок 4.6 – Доступні методи document-controller

Для завантаження документу у вигляді тексту потрібно вибрати блок POST “/v1/documents” та натиснуто кнопку «Try it out». Після цього з’являється форма для формування запиту(рисунок 4.7).

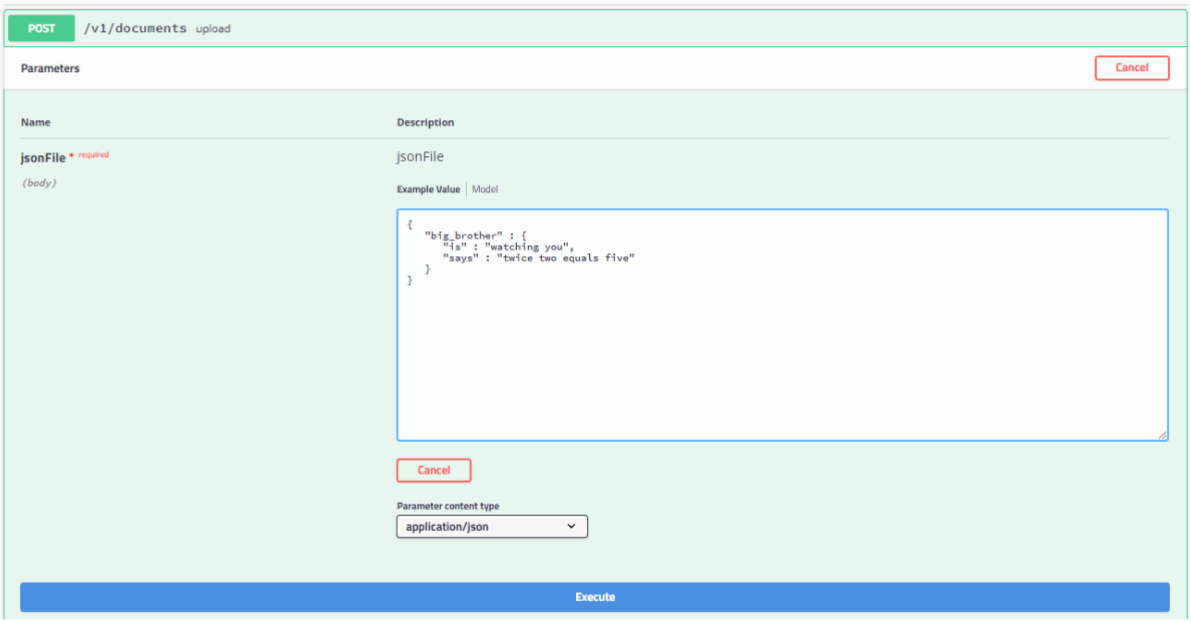

Рисунок 4.7 – Форма для формування запиту

У поле jsonFile введено сформований раніше JSON документ. Натиснуто кнопку «Execute». Отримано відповідь від системи(рисунок 4.8).

Рисунок 4.8 – Відповідь від системи

Отримано статус 201, що свідчіть про успішну операцію, та результат у вигляді UUID, що є ідентифікатором завантаженого документу.

4.3 Завантаження документу у вигляді файлу до системи

Іншим варіантом завантаження документу до системи є завантаження у вигляді файлу з розширенням .json.

4.3.1 За допомогою Postman

При використанні Postman документ було вміщено у файл з розширенням .json, який у свою чергу було вміщено у тіло запиту, для завантаження документу використано DocumentController, для цього в URL запиту було задано шлях “/v1/documents/file” та метод запиту POST(рисунок 4.9).

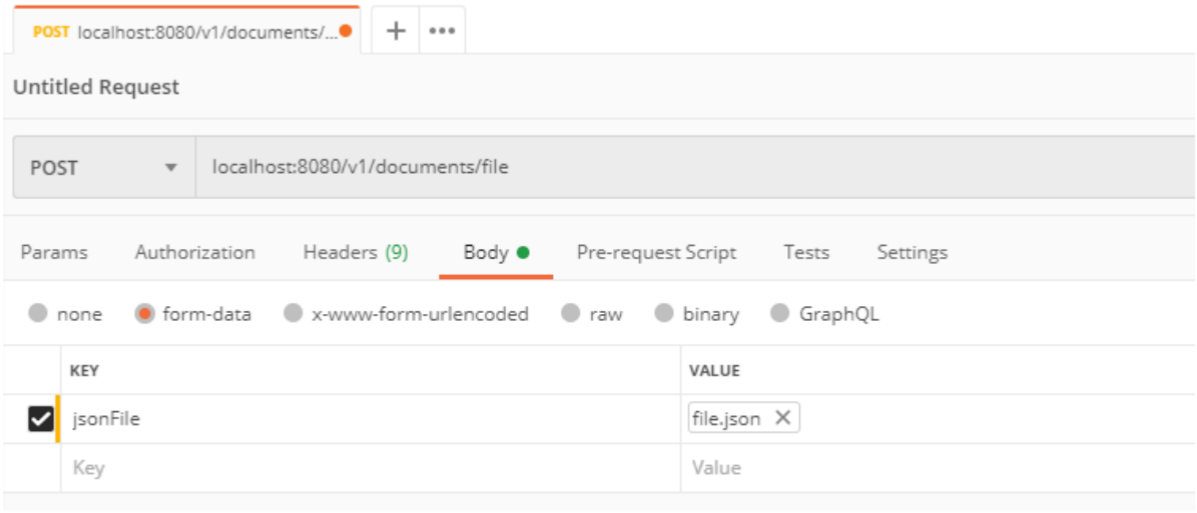

Рисунок 4.9 – Сформований запит до системи

Запит надіслано на сервер, отримано відповідь(рисунок 4.10).

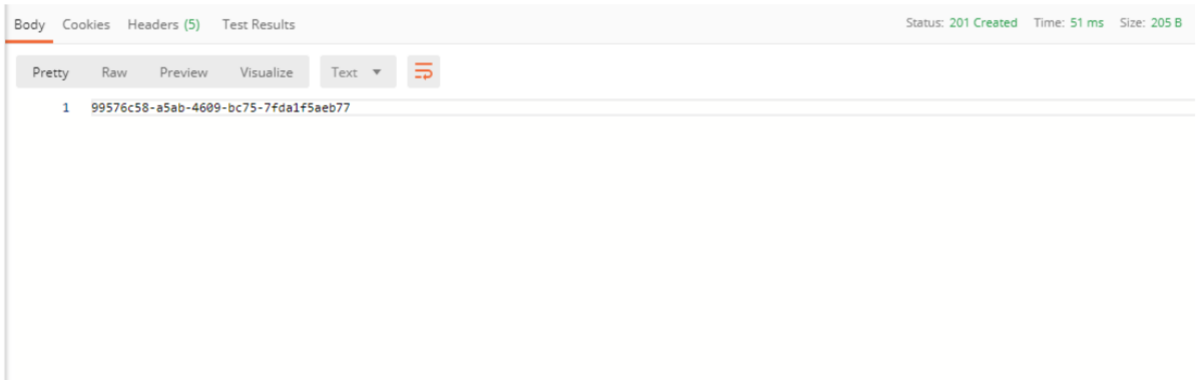

Рисунок 4.10 – Відповідь від системи

Отримано статус 201, що свідчіть про успішну операцію, та результат у вигляді UUID, що є ідентифікатором завантаженого документу.

4.3.2 За допомогою Swagger

Для завантаження документу у вигляді тексту потрібно вибрати блок POST “/v1/documents/file” та натиснуто кнопку «Try it out». Після цього з’являється форма для формування запиту(рисунок 4.11).

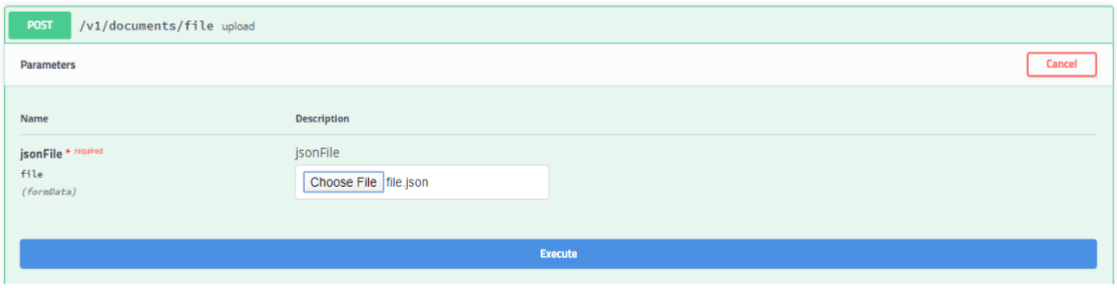

Рисунок 4.11 – Форма для формування запиту з вікном для завантаження файлу

Ця форма має вікно для завантаження файлу, що відрізняє її від вище наведеної форми. У вікні було вибрано сформаваний раніше документ у вигляді файлу з розширенням .json та натиснуто кнопку «Execute» для надсилання запиту(рисунок 4.12).

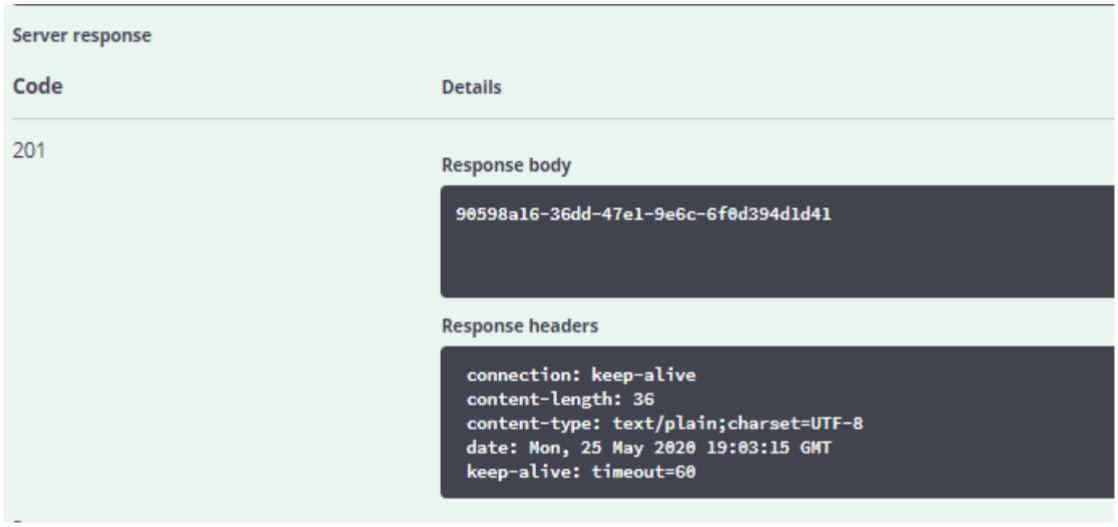

Рисунок 4.12 – Відповідь від системи

Отримано статус 201, що свідчіть про успішну операцію, та результат у вигляді UUID, що є ідентифікатором завантаженого документу.

4.4 Завантаження документу з системи

Для отримання результату оброблення документу системою використовується метод завантаження документу з системи.

4.4.1 За допомогою Postman

При використанні Postman в URL запиту було задано шлях “/v1/documents/{id}” та метод запиту GET(рисунок 4.13)..

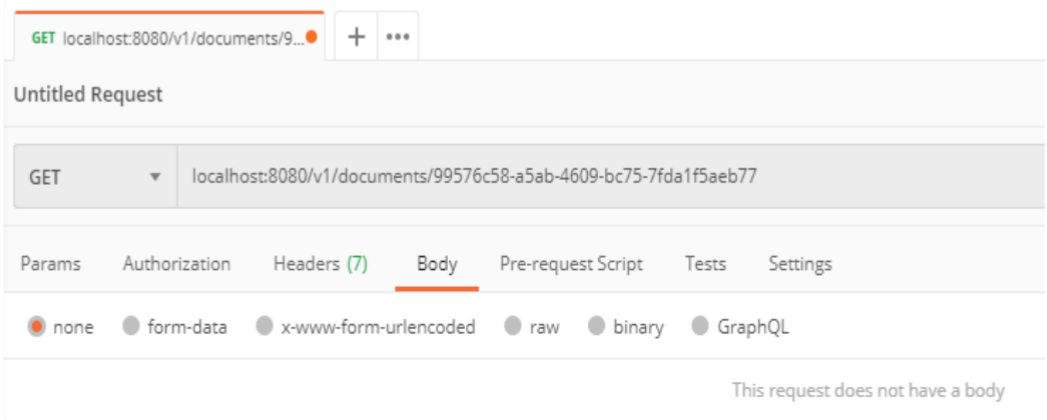

Рисунок 4.13 – Сформований запит до системи

Запит надіслано на сервер, отримано відповідь(рисунок 4.14).

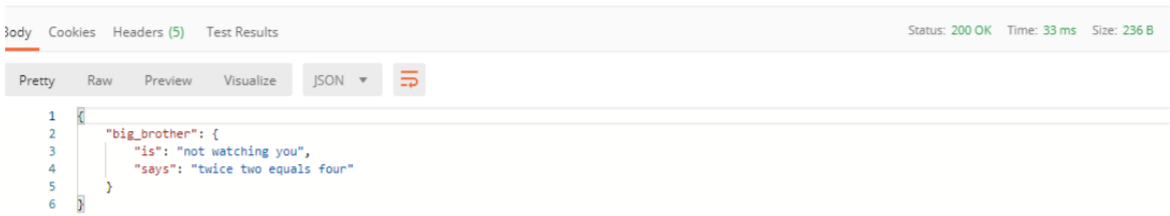

Рисунок 4.14 – Відповідь від системи

Отримано статус 200, що свідчіть про успішну операцію, та результат завантаженого раніше документу.

4.4.2 За допомогою Swagger

Для завантаження документу у вигляді тексту потрібно вибрати блок GET “/v1/documents/{id}” та натиснуто кнопку «Try it out». Після цього з’являється форма для формування запиту(рисунок 4.15).

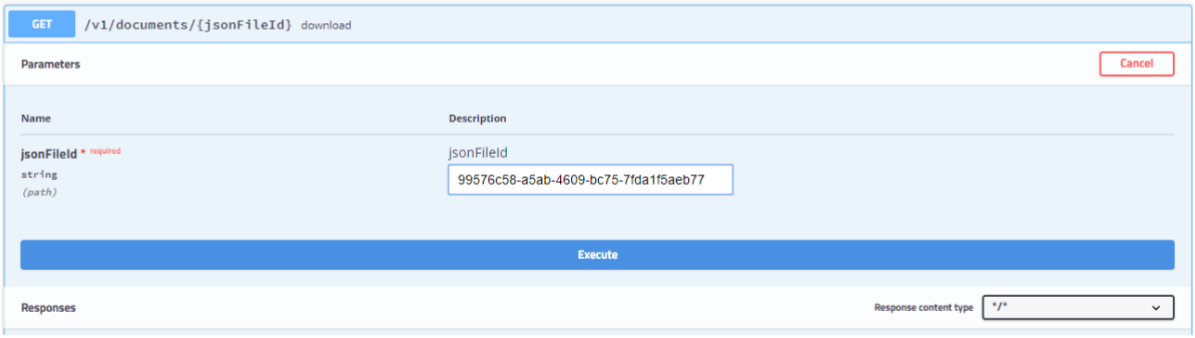

Рисунок 4.15 – Форма для формування запиту з вікном для завантаження файлу

У поле jsonFileId введено UUID сформованого раніше JSON документу. Натиснуто кнопку «Execute». Отримано відповідь від системи(рисунок 4.16).

Рисунок 4.16 – Відповідь від системи

Отримано статус 200, що свідчіть про успішну операцію, та результат у вигляді раніше завантаженого документу.

Також є можливість завантажити документ натиснувши кнопку «Download» (рисунок 4.17).

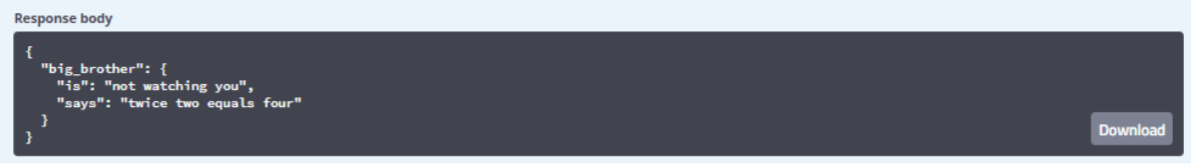

Рисунок 4.17 – Відповідь від системи з можливістю завантаження на клієнт у вигляді файлу

При натисканні кнопки «Download» буде завантажено документ з розширенням .json, який містить раніше сформований текст(рисунок 4.18)..

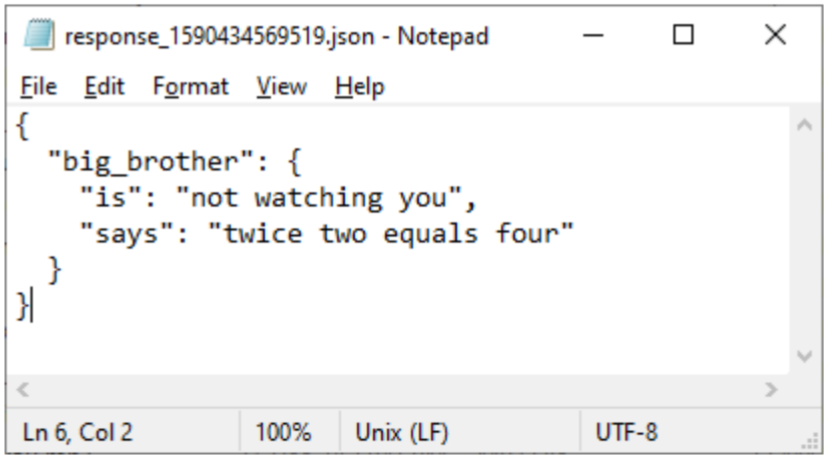

Рисунок 4.18 – Завантажений файл відкрито у Notepad 4.5 Видалення документу з системи

Після того, як користувач більше не планує використовувати завантажений файл у системі – він може його видалити.

4.5.1 За допомогою Postman

При використанні Postman в URL запиту було задано шлях “/v1/documents/{id}” та метод запиту DELETE(рисунок 4.19)..

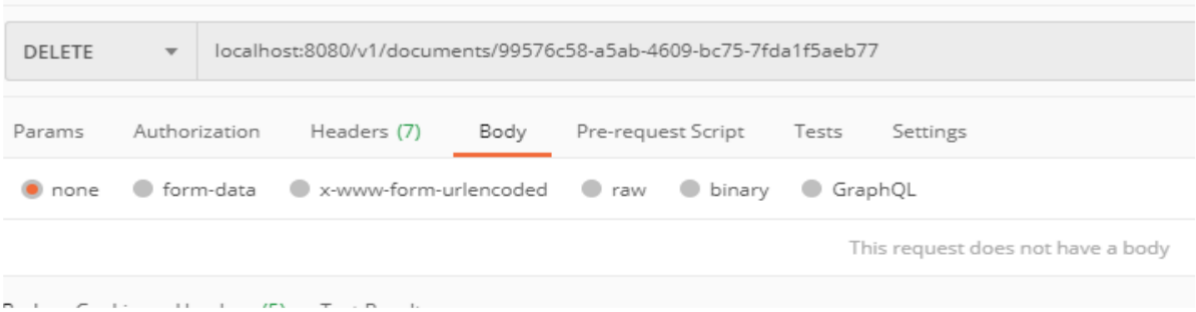

Рисунок 4.19 – Сформований запит до системи

Запит надіслано на сервер, отримано відповідь(рисунок 4.20).

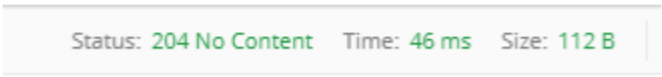

Рисунок 4.20 – Відповідь від системи

Отримано статус 204, що свідчіть про успішну операцію, та який означає що сервер не повертає тіло запиту при даній операції. Для перевірки чи видалено документ потрібно знову звернутися до бази даних ArangoDB, для цього в інтерфейсі потрібно вибрати фільтр та відфільтрувати документи по полю _key, результат не повинен містити жодного документу(рисунок 4.21).

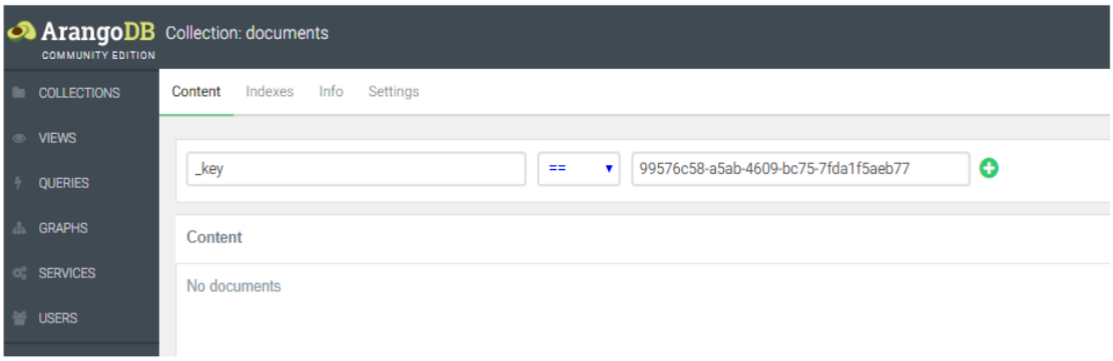

Рисунок 4.21 – Результат фільтрування

Результатом виконаних дій стало повне видалення документу з системи.

4.5.2 За допомогою Swagger

Для видалення документу з системи потрібно вибрати блок DELETE “/v1/documents/{id}” та натиснути кнопку «Try it out». Після цього з’являється форма для формування запиту(рисунок 4.22)

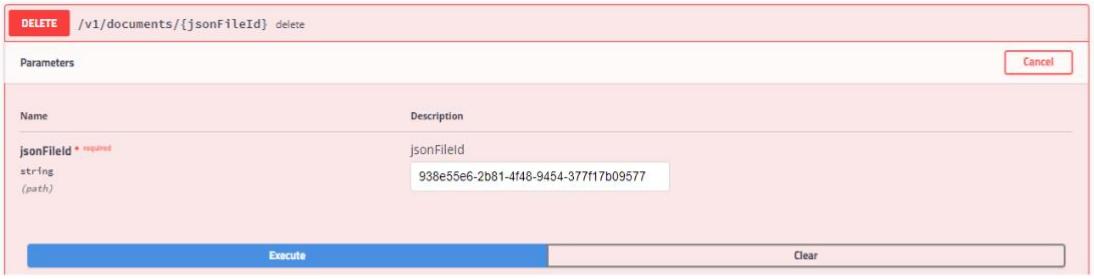

Рисунок 4.22 – Форма для формування запиту з полем для ідентифікатора документу

Видно, що форма позначена червоним кольором, який запобігає небажаним діям. У поле jsonFileId введено UUID сформованого раніше JSON документу. Натиснуто кнопку «Execute». Отримано відповідь від системи(рисунок 4.23).

Рисунок 4.23 – Відповідь від системи

Отримано статус 204, що свідчить про успішну операцію. Перевіряємо за допомогою фільтрування, чи залишився документ в базі даних ArangoDB після видалення(рисунок 4.24).

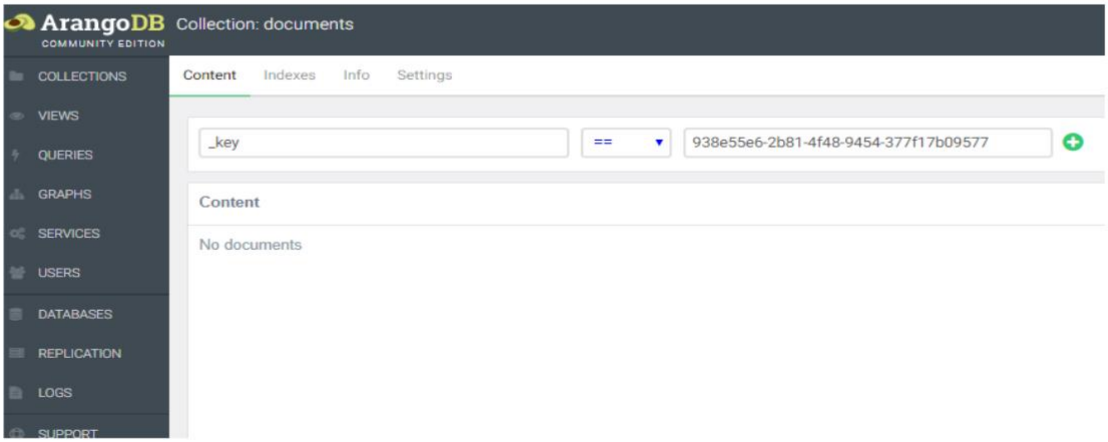

Рисунок 4.24 – Результат фільтрування

4.6 Додавання частини до документу

4.6.1 За допомогою Postman

При використанні Postman в URL запиту було задано шлях “/v1/process/insert/{id}” та метод запиту PATCH для проведення часткової зміни документу. Цей запит також вимагає додаткових параметрів в URL у вигляді key та path, вони додаються в URL за правилом конкатенування параметрів до URL запиту(рисунок 4.25)

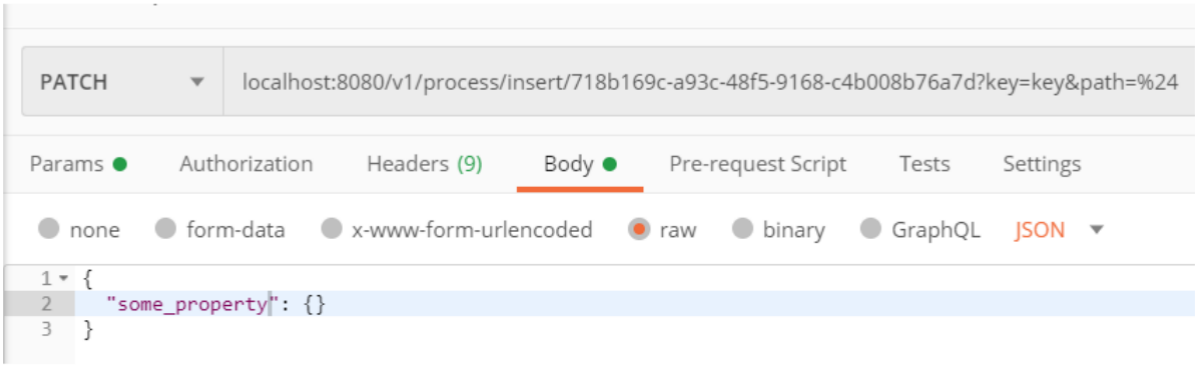

Рисунок 4.25 – Сформований запит до системи

Запит надіслано на сервер, отримано відповідь(рисунок 4.26)

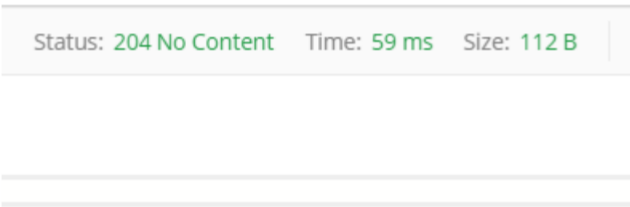

Рисунок 4.26 – Відповідь від системи

Отримано статус 204, що свідчіть про успішну операцію, та який означає що сервер не повертає тіло запиту при даній операції. При перегляді документу базі даних ArangoDB результат повинен містити параметр some_property у середені параметру key JSON документу(рисунок 4.27).

Рисунок 4.27 – Результат додавання частини JSON до документу

Результатом виконаних дій стало додавання нової частини до існуючого документу в системі.

4.6.2 За допомогою Swagger

Для видалення документу з системи потрібно вибрати блок PATCH “/v1/process/insert/{id}” та натиснути кнопку «Try it out». Після цього з’являється форма для формування запиту(рисунок 4.28).

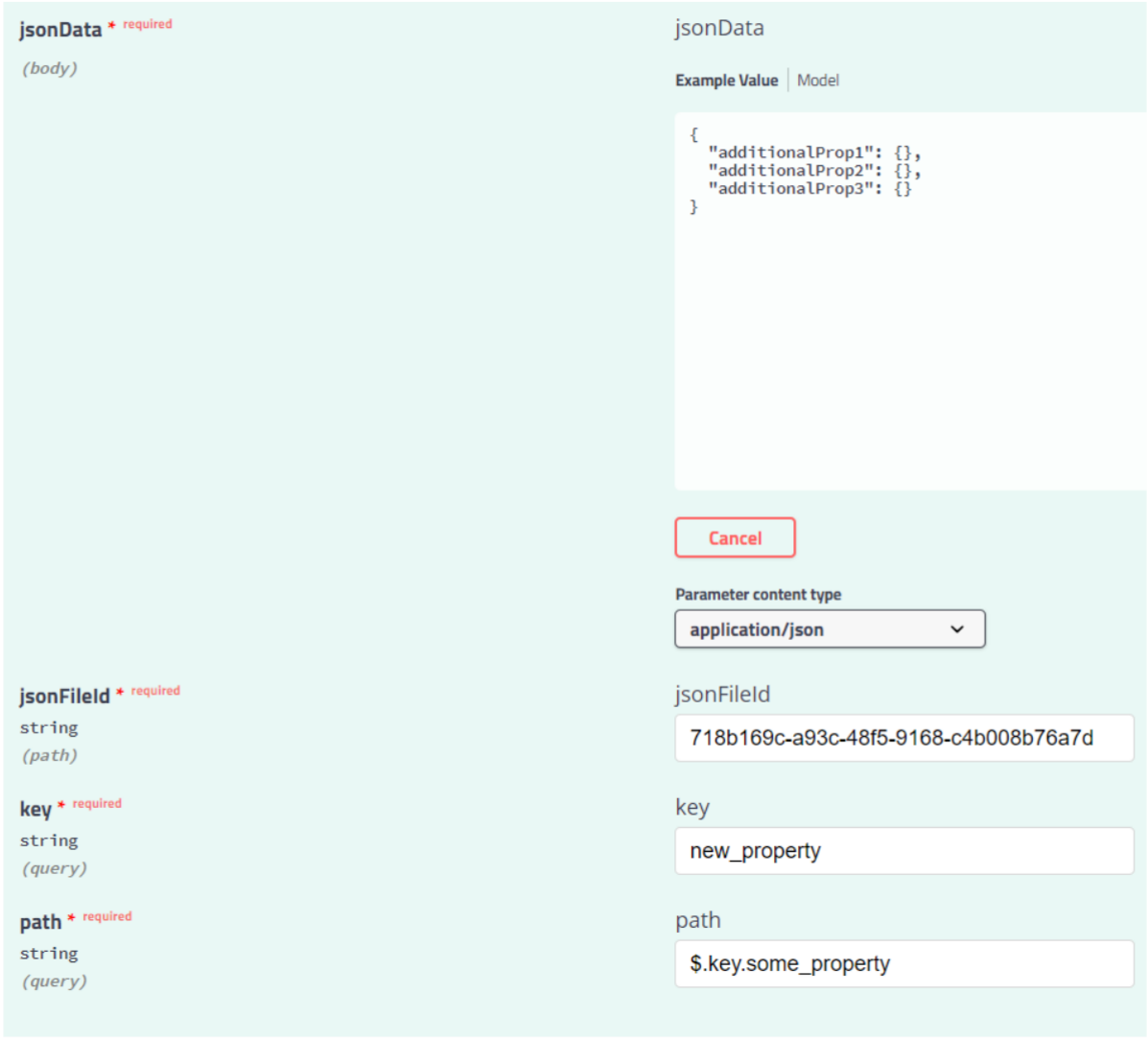

Рисунок 4.28 – Форма для формування запиту

У поле jsonFileId введено UUID сформованого раніше JSON документу. Також заповнено поля key, path та додано тіло у вигляді jsonData. Натиснуто кнопку «Execute». Отримано відповідь від системи(рисунок 4.29).

Рисунок 4.29 – Відповідь від системи

Отримано статус 204, що свідчить про успішну операцію. Проведено перевірку за допомогою фільтрування, чи змінився документ в базі даних ArangoDB після проведення над ним дії(рисунок 4.30).

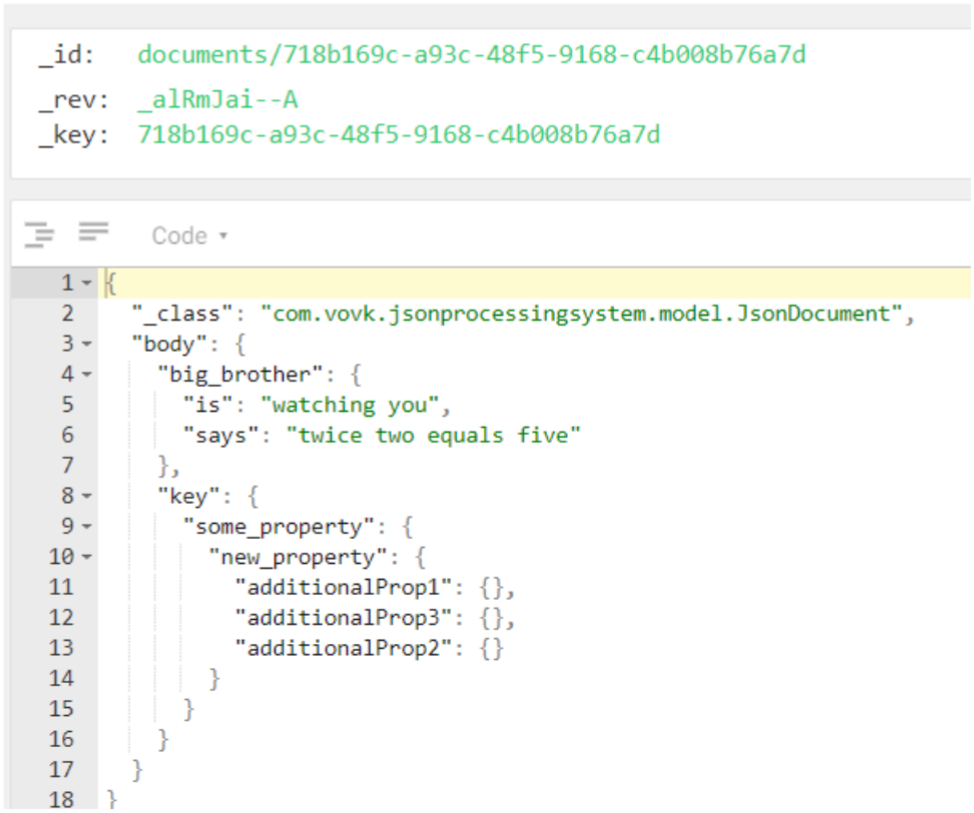

Рисунок 4.30 – Результат додавання частини до документу

4.7 Видалення частини документу

4.7.1 За допомогою Postman

При використанні Postman в URL запиту було задано шлях “/v1/process/{id}” та метод запиту DELETE. До шляху в URL додано поле path зі значенням $.key.some_property.new_property.additionalProp1. При виконанні такого запиту буде видалено частину документу, а саме ту, яку буде вказано у полі path. У цьому випадку це будуть усі дані, які знаходяться у документі з переданим ідентифікатором по шляху key.some_property.new_property.additionalProp1(рисунок 4.31).

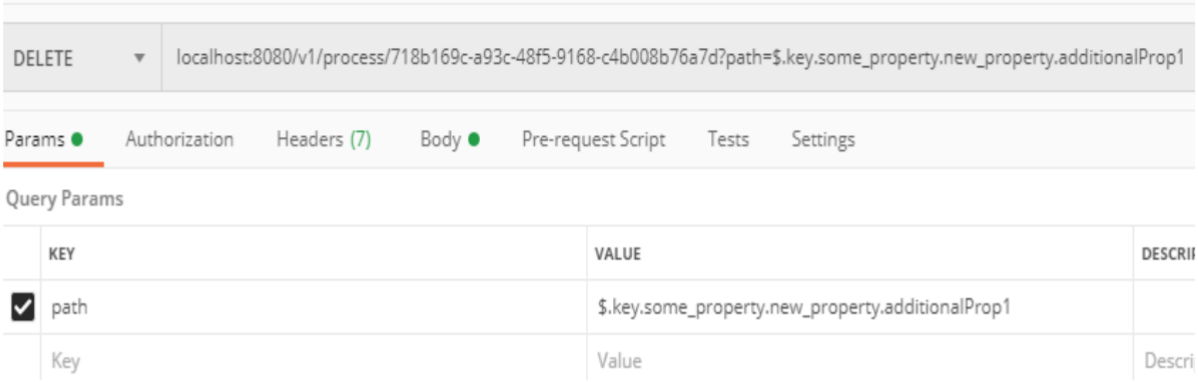

Рисунок 4.31 – Сформований запит до системи

Запит надіслано на сервер, отримано позитивну відповідь від системи, що свідчить про те що запит оброблено правильно та без помилок. Статус 204 свідчить про те що запит оброблено правильно, але відповідь від системи не передбачає надсилання тіла запиту до користувача або зовнішньої інтегрованої системи. Час оброблення запиту 24 мілісекунди, що свідчить про швидке оброблення запиту системою(рисунок 4.32).

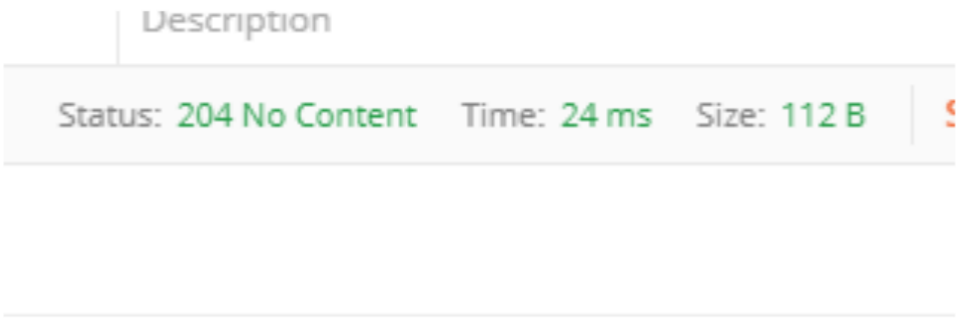

Рисунок 4.32 – Відповідь від системи

Отримано статус 204, що свідчіть про успішну операцію, та який означає що сервер не повертає тіло запиту при даній операції. Для перевірки чи видалено поле по шляху $.key.some_property.new_property.additionalProp1 у документі потрібно знову звернутися до бази даних ArangoDB, та перевірити, чи є таке поле у документі з ідентифікатором 718b169c-a93c-48f5-9168-c4b008b76a7d(рисунок 4.33).

Рисунок 4.33 – Результат дії над документом

Результатом виконаних дій стало видалення частини документу з системи.

4.7.2 За допомогою Swagger

Для видалення документу з системи потрібно вибрати блок DELETE “/v1/process/{id}” та натиснути кнопку «Try it out». Після цього з’являється форма для формування запиту(рисунок 4.34).

Рисунок 4.34 – Форма для формування запиту з полем для ідентифікатора документу

Ця форма позначена червоним кольором як і усі форми які виконують функцію видалення. У поле jsonFileId введено UUID сформованого раніше JSON документу. У поле path введено існуючий у документі шлях. Натиснуто кнопку «Execute». Отримано відповідь від системи(рисунок 4.35):

Рисунок 4.35 – відповідь від системи

Отримано статус 204, що свідчить про успішну операцію. Перевірено за допомогою ArangoDB, чи відбулася зміна документу(рисунок 4.36).

Рисунок 4.36 – Результат фільтрування

Результатом виконаних дій стало видалення частини документу з системи.

4.8 Об’єднання двох документів зі збереженням у системі

4.8.1 За допомогою Postman

При використанні Postman в URL запиту було задано шлях “/v1/process/merge/{id}” та метод запиту PATCH(рисунок 4.37).

Рисунок 4.37 – Сформований запит до системи

Запит надіслано на сервер, отримано відповідь(рисунок 4.38).

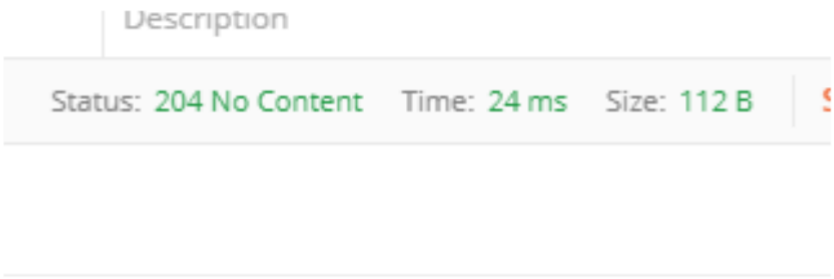

Рисунок 4.38 – Відповідь від системи

Отримано статус 204, що свідчіть про успішну операцію, та який означає що сервер не повертає тіло запиту при даній операції. Перед виконанням дії документ мав лише один ключ по шляху $.key, а саме some_property3(рисунок 4.39).

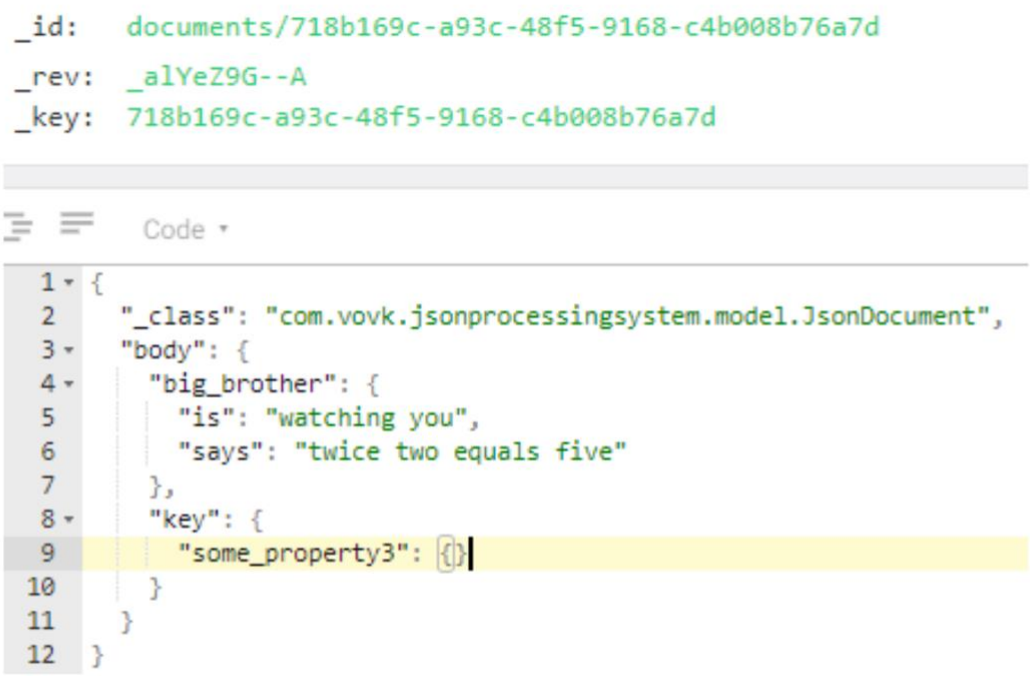

Рисунок 4.39 – Стан документу до злиття з іншим документом

Результатом виконаних дій стало повинно бути злиття документу з тіла запита та існуючого документу в системі(рисунок 4.40).

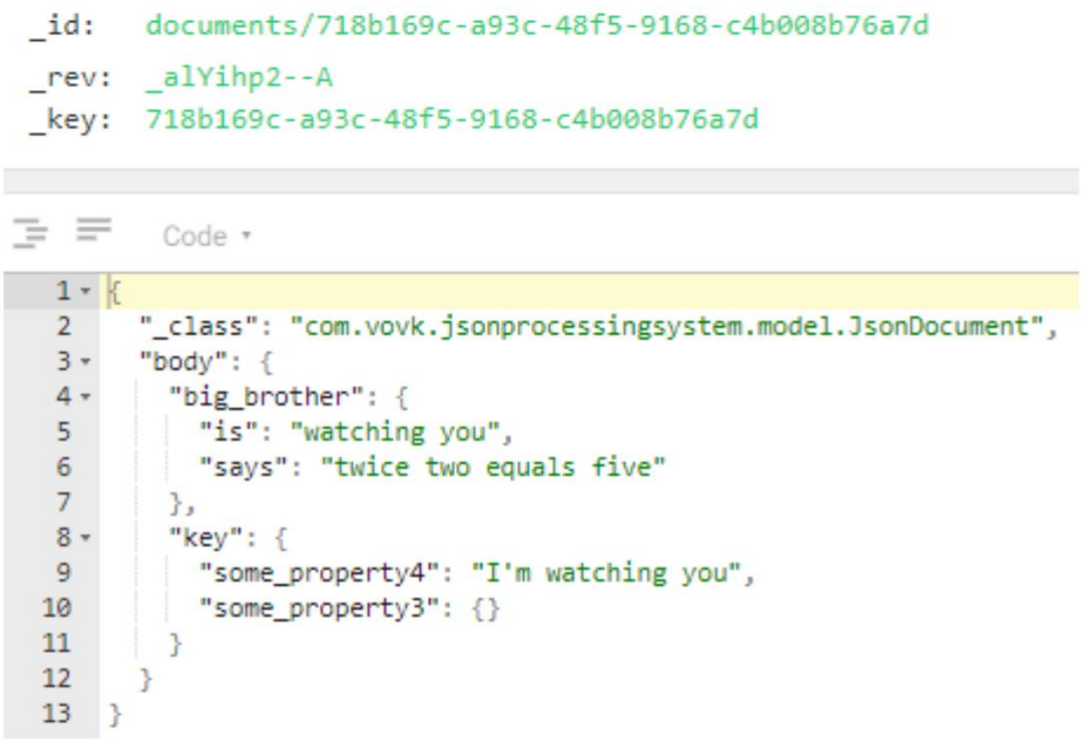

Рисунок 4.40 – Стан документу після злиття з іншим документом

4.8.2 За допомогою Swagger

Для видалення документу з системи потрібно вибрати блок PATCH “/v1/process/merge/{id}” та натиснути кнопку «Try it out». Після цього з’являється форма для формування запиту(рисунок 4.41).

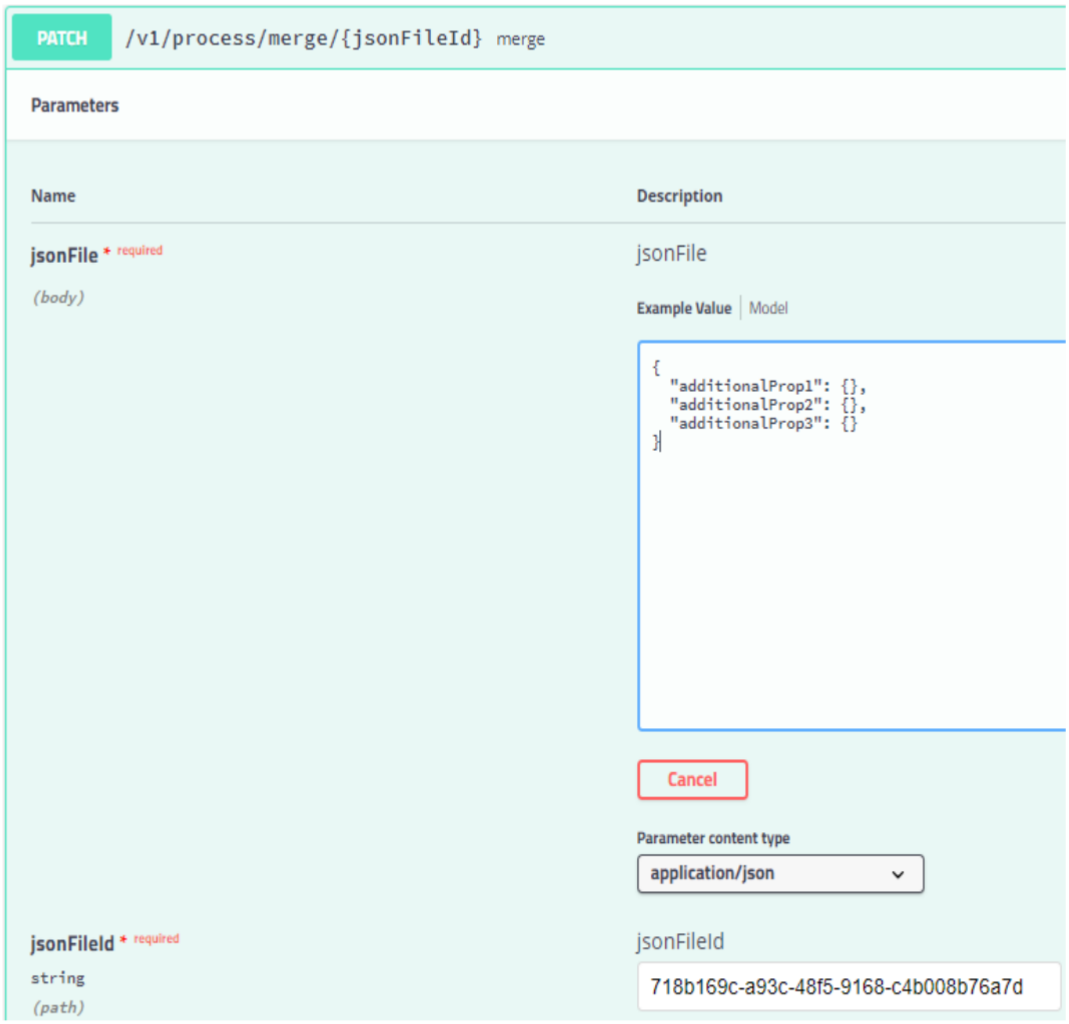

Рисунок 4.41 – Форма для формування запиту з полем для ідентифікатора документу

У поле jsonFileId введено UUID сформованого раніше JSON документу. У поле jsonFile введено документ який потрібно об’єднати з існуючим. Натиснуто кнопку «Execute». Отримано відповідь від системи(рисунок 4.42):

Рисунок 4.42 – Відповідь від системи

Отримано статус 204, що свідчить про успішну операцію. Перевірено за допомогою ArangoDB, чи відбулася зміна документу(рисунок 4.43).

Рисунок 4.43 – Результат злиття документів

4.9 Зміна документу

4.9.1 За допомогою Postman

При використанні Postman в URL запиту було задано шлях “/v1/process/replace/{id}” та метод запиту PATCH(рисунок 4.44).

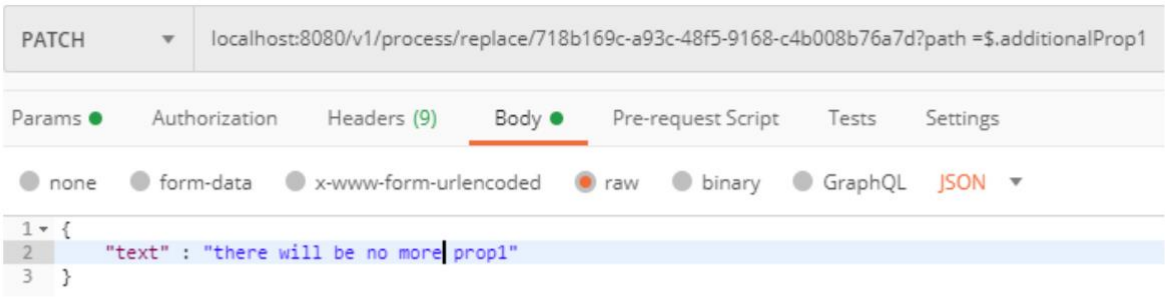

Рисунок 4.44 – Сформований запит до системи

Запит надіслано на сервер, отримано відповідь(рисунок 4.45).

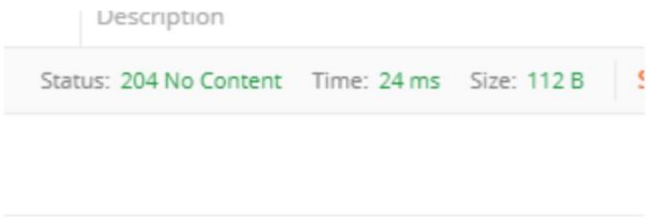

Рисунок 4.45 – Відповідь від системи

Отримано статус 204, що свідчіть про успішну операцію. Після виконання дії документ повинен мати документ переданий йому в запиті по шляху $.additionalProp1(рисунок 4.46).

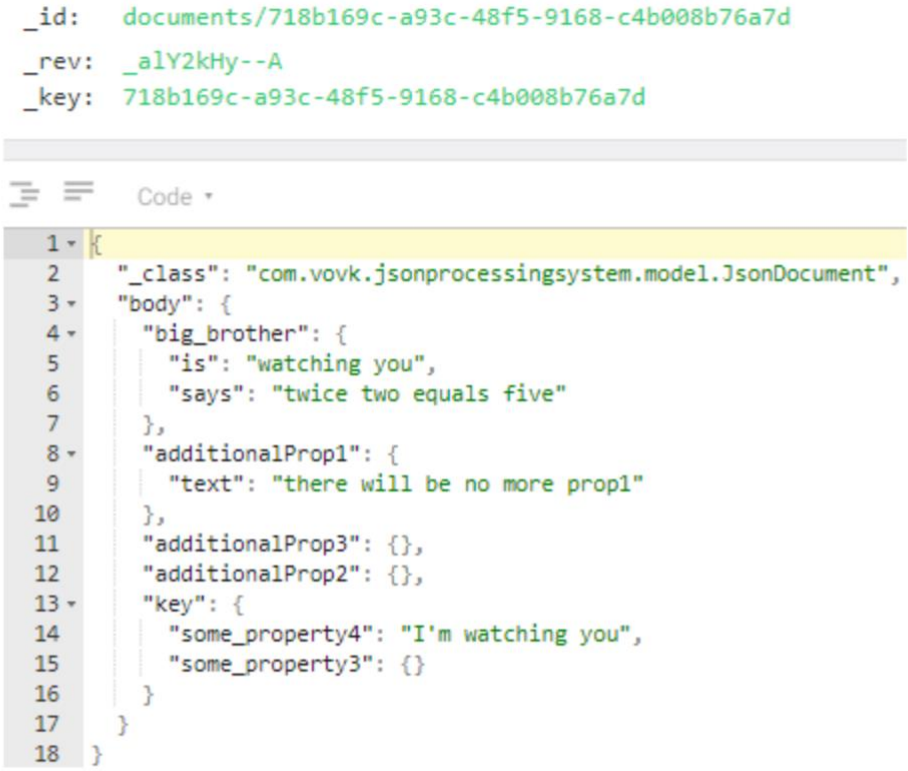

Рисунок 4.46 – Стан документу після виконання зміни

4.9.2 За допомогою Swagger

Для видалення документу з системи потрібно вибрати блок PATCH “/v1/process/replace/{id}” та натиснути кнопку «Try it out». Після цього з’являється форма для формування запиту(рисунок 4.47).

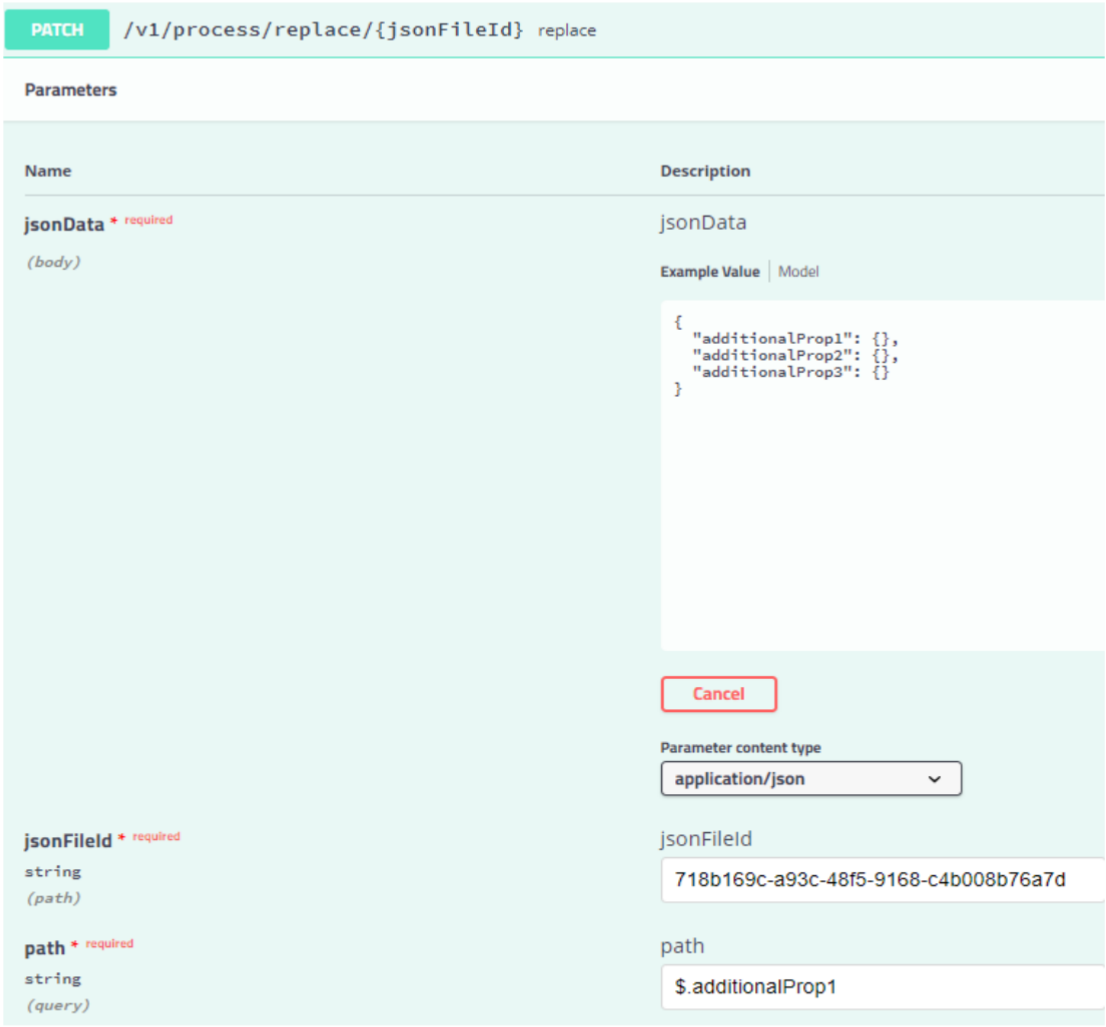

Рисунок 4.47 – Форма для формування запиту з полем для ідентифікатора документу

У поле jsonFileId введено UUID сформованого раніше JSON документу. У поле jsonData введено те що потрібно замінити у ключа по шляху $.additionalProp1. Натиснуто кнопку «Execute». Отримано відповідь від системи(рисунок 4.48).

Рисунок 4.48 – Відповідь від системи

Отримано статус 204, що свідчить про успішну операцію. Перевірено за допомогою ArangoDB, чи відбулася зміна документу(рисунок 4.49).

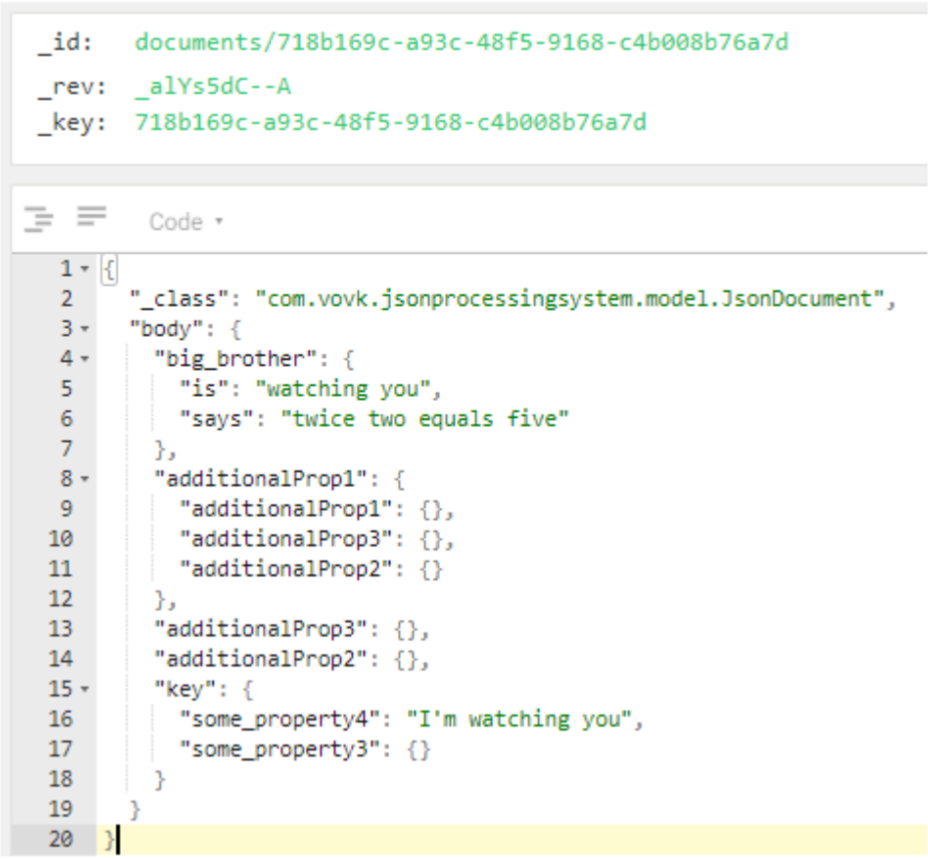

Рисунок 4.49 – Результат зміни частини документа

4.10 Завантаження частини документу

Над документом буде проводитися лише дія зчитування, тому його стан залишиться незмінним після її закінчення(рисунок 4.50).

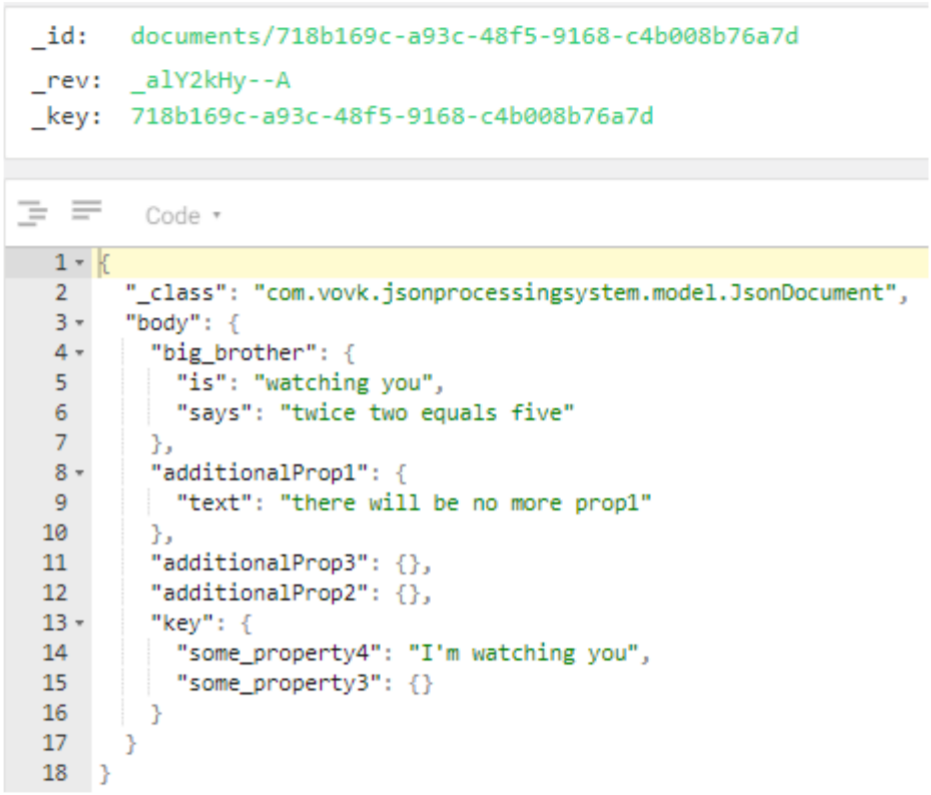

Рисунок 4.50 – Тіло документу

4.10.1 За допомогою Postman

Для формування запиту використано метод GET та адресу “/v1/process/{id}” до якої конкатеновано шлях за правилами будування URL(рисунок 4.51).

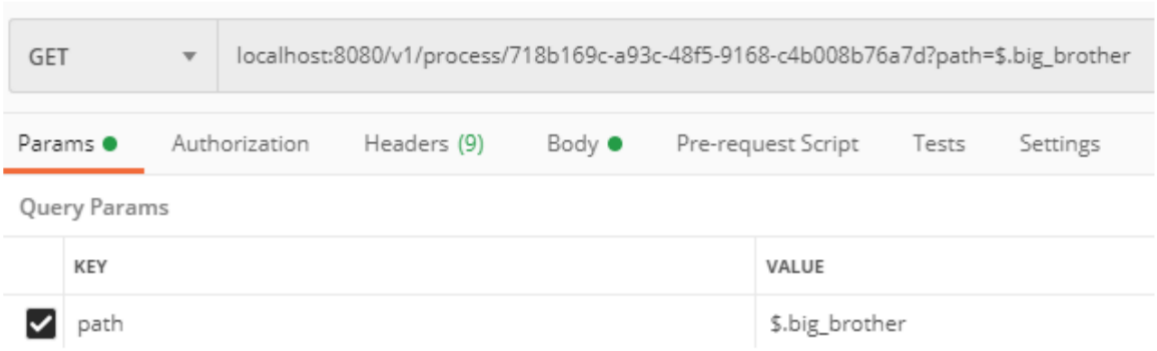

Рисунок 4.51 – Сформований запит

Запит надіслано та отримано успішну відповідь від системи зі статусом 200 та тілом відпіповіді(рисунок 4.52).

```
{
    "says": "twice two equals five",
    "is": "watching you"
}
```


Рисунок 4.52 – Відповідь системи

4.10.2 За допомогою Swagger

Для видалення документу з системи потрібно вибрати блок GET “/v1/process/{id}” та натиснути кнопку «Try it out». Після цього з’являється форма для формування запиту(рисунок 4.53).

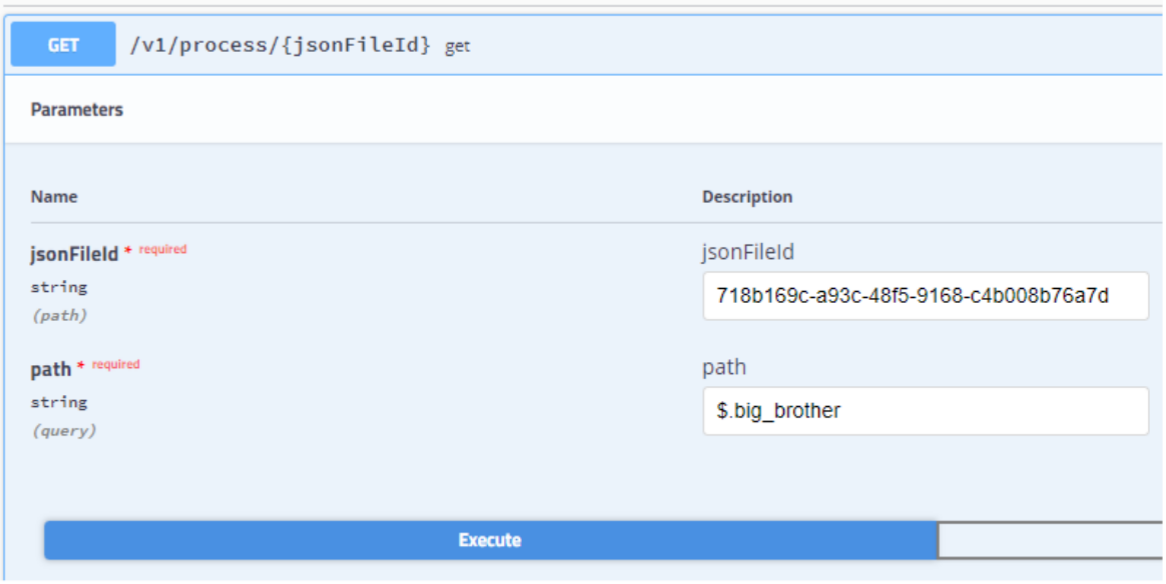

Рисунок 4.53 – Сформований запит

Після формування запиту його надіслано до системи, отримано код 200 та частину документу, яка існує по шляху $.big_brother(рисунок 4.54).

Рисунок 4.54 – Відповідь від системи

4.11 Висновки до розділу

У даному розділі було протестовано розроблену систему оброблення JSON документів.

Було перевірено кожний метод, який може бути використаний іншою системою або користувачем, перевірено що документи успішно зберігаються в базі даних ArangoDB, модифікуються та видаляються за запитом. Перевірено, що система видає ті ж самі результати, як при тестуванні через Postman, так і при тестуванні через Swagger, що свідчить про її стабільність при роботі як з користувачем так і з інтегрованою системою.

Розроблене програмне забезпечення повністю відповідає тим вимогам, які було поставлено для програмного забезпечення яке знаходиться в основі дипломного проекту.

ВИСНОВКИ

У дипломному проекті було проведено аналіз існуючих рішень, їх переваги та недоліки, на основі розглянутих існуючих систем розроблено вимоги до розроблюваної системи. Для розроблення системи проведено пошук найбільш оптимальних технологій, бібліотек та патернів проектування. У результаті пошуку знайдено та обґрунтовано вибір найбільш оптимальних і найновіших технологій та інструментів для розроблення системи. Спроектовано систему з урахуванням вимог зазначених у завданні до проєкту, патернів проектування та найкращих практик, описано її сценарії використання.

Розроблено систему з можливістю завантаження документів зі зберіганням у базі даних системи та можливістю видалення документів. Реалізовано можливість зміни, видалення або додавання до них частини переданого документу. Додатковою можливістю системи є об’єднання двох документів зі збереженням у системі. Для зручності користувача система зберігає завантажені документи, тому вони можуть бути змінені, завантажені або видалені у разі потреби.

Впроваджено можливість як інтегрування розробленої системи до складу іншої системи так і можливість використання системи безпосередньо користувачем. Обидва варіанти використання детально описано та перевірено за допомогою Swagger UI та Postman на предмет виникнення помилок в залежності від клієнту використання системи.

Розроблена система не потребує багато ресурсів для функціонування та взагалі не навантажує систему клієнту при виконанні будь-яких дій над документами. Таке архітектурне рішення дозволяє редагувати великі JSON документи швидко, зручно та без встановлення будь-якого програмного забезпечення до системи клієнту.

За результатами тестування визначено, що розроблена система стабільно працює в незалежності від клієнту, працює зі стабільно високою швидкістю та надійністю, а мету дипломного проекту досягнуто в повному обсязі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Офіційний сайт Notepad++. URL: https://notepad-plus-plus.org/ (дата звернення: 13.04.2020)
2. Офіційний сайт Vusual Code. URL: https://code.visualstudio.com/ (дата звернення: 13.04.2020)
3. Офіційний сайт Sublime Text. URL: https://www.sublimetext.com/ (дата звернення: 14.04.2020)
4. Офіційний сайт JSONPath Online Evaluator. URL: https://jsonpath.com/ (дата звернення: 14.04.2020)
5. Офіційний сайт JSON Editor Online. URL: https://jsoneditoronline.org/ (дата звернення: 14.04.2020)
6. Офіційний сайт Online JSON Viewer. URL: http://jsonviewer.stack.hu/ (дата звернення: 15.04.2020)
7. Шилдт Г. Java 8. Повне керівництво. – К. : Діалетіка, 2015. – 1376 с.
8. Java Remote Method Invocation API Guide. URL: https://docs.oracle.com/en/java/javase/14/rmi/index.html#RMIUG-GUID-70825E99-57CF-470F-871A-5CCB6C2771BE (дата звернення: 18.04.2020)
9. Rod J., Jürgen. H., Alef A., Risberg T., Sampaleanu C. Spring Framework Reference Documentation. 2016. URL: https://docs.spring.io/autorepo/docs/spring-framework/5.0.0.M1/spring-frameworkreference/pdf/spring-framework-reference.pdf (дата звернення: 19.04.2020)
10. Amit H. Spring Framework Ecosystem - Introduction to Spring Modules. URL: http://springtutorials.com/introduction-to-spring-modules/ (дата звернення: 23.04.2020)
11. Ципанов С. І. На горе стоит Spring Boot. URL: https://habr.com/ru/post/439918/ (дата звернення: 21.04.2020)
12. Чернишов В. А. Sping MVC – основные принципы. URL: https://habr.com/ru/post/336816/ (дата звернення: 22.04.2020)
13. Raffai Z. Spring Boot: The Most Notable Features You Should Know. URL: https://dzone.com/articles/what-is-spring-boot (дата звернення: 23.04.2020)

14. Prasanna D. Dependency Injection – Manning Publications, 2009, 352 pp.

15. Swagger documentation. URL: https://swagger.io/docs/specification/about/ (дата звернення: 24.04.2020)

16. Swagger UI. URL: https://swagger.io/tools/swagger-ui/ (дата звернення: 24.04.2020)

17. Quercia V., Spainhour S. WebMaster in a Nutshell: A Desktop Quick Reference. Third edition. – O`Reilly Media, 2002, 520 pp.

18. Richardson L., Amundsen M., Ruby S. RESTful Web – O`Reilly Media, 2013, 408 pp.

19. Феоктістов А. Р. REST vs SOAP. Часть 1. URL: https://habr.com/ru/post/131343/ (дата звернення: 24.04.2020)

20. Jim W. REST in Practice: Hypermedia and Systems Architecture. 1st Edition. –

21. Stoyanchev R., Brannen S. Annotation Type RestController. URL: https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html (дата звернення: 26.04.2020)

22. JSON Стандарт RFC 7159.

23. Abramov. A. Introduction to ArangoDB. URL: https://www.arangodb.com/docs/stable/ (дата звернення: 27.04.2020)

24. REST API documentation. Доступ: https://docs.spring.io/spring-restdocs/docs/current/reference/html5/ (дата звернення: 27.04.2020)

25. Макарова Д. АРХІТЕКТУРА МІКРОСЕРВІСІВ : Матеріали III Всеукраїнської науково-технічної конференції ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ РАДІОТЕХНІКИ І ПРИЛАДОБУДУВАННЯ, 26 травня 2017 р. м. Київ с. 18.

26. UUID Стандарт RFC 4122.

27. Rod J., Jürgen. H., Alef A., Risberg T., Sampaleanu C. Spring Framework Documentation. 2019. URL: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html (дата звернення: 27.04.2020)

28. Олексій А.О. Переход от монолита к микросервисам. URL: https://habrahabr.ru/post/305826/. (дата звернення: 29.04.2020)