Державний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмний комплекс для розробки API»

Студента 2м курсу, 2 групи, спеціальності 121 «Інженерія програмного забезпечення» освітньої програми «Інженерія програмного забезпечення» _____ Александрова Андрія Юрійовича
підпис студента

Науковий керівник кандидат педагогічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ Котенко Наталія Олексіївна
підпис керівника

Гарант освітньої програми кандидат педагогічних наук, доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ Котенко Наталія Олексіївна
підпис гаранта

КИЇВ – 2023

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Освітня програма 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«13» грудня 2022 р.

Завдання
на випускну кваліфікаційну роботу студентові

Александрову Андрію Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема випускної кваліфікаційної роботи «Програмний комплекс для розробки API»

Затверджена наказом ректора від «06» грудня 2022 р. № 3285

2. Строк здачі студентом закінченої роботи 1 грудня 2023

3. Цільова установка та вихідні дані до роáоти

Мета роботи піддати науковому огляду взаємодію сучасних програмних засобів та розробити інструмент для підвищення ефективності розробки прикладних програмних інтерфейсів як способу взаємодії на основі отриманої інформації.

Об'єкт дослідження проєктування та програмна реалізація усіх компонентів сервісу розробки та впровадження програмних інтерфейсів.

Предмет дослідження розробка програмного забезпечення.

4. Консультанти роботи із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускної кваліфікаційної роботи (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1 АНАЛІЗ ТЕХНІЧНОЇ ДОКУМЕНТАЦІЇ ТА СТАНДАРТІВ ЯК ФУНДАМЕНТУ ФУНКЦІОНУВАННЯ ПРОГРАМНИХ ІНТЕРФЕЙСІВ

1.1. Аналіз способів взаємодії програмних систем. Клієнт серверна взаємодія

1.2. Функціональні програмні інтерфейси. Бізнес-логіка

1.3. Порівняння аналогічного програмного забезпечення

1.4. Висновки до розділу 1

РОЗДІЛ 2 РОЗРОБКА АРХІТЕКТУРИ СИСТЕМИ РОЗРОБКИ ТА ВПРОВАДЖЕННЯ ПРОГРАМНИХ ІНТЕРФЕЙСІВ

2.1. 2.1. Високорівневий огляд архітектури

2.2. Розробка моделей діючих компонентів, об’єктів системи та проєктування взаємозв’язків між ними

2.3. Обґрунтування вибору операційної системи, мови програмування та інших технологій

2.4. Висновки до розділу 2

РОЗДІЛ 3 ДАНІ. СКЛАД, СТРУКТУРА ТА ПРИНЦИПИ ОРГАНІЗАЦІЇ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ, ВИБІР ТЕХНОЛОГІЙ ДОСТУПУ ДО БАЗ ДАНИХ

3.1. Високорівневе проєктування баз даних

3.2. Проєктування концептуальної моделі реляційної бази даних

3.3. Проєктування логічної моделі реляційної бази даних

3.4. Проєктування фізичної моделі реляційної бази даних

.

6. Календарний план виконання роботи

№ пор.	Назва етапів випускної кваліфікаційної роботи	Строк виконання етапів роботи	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*07.11.2022*	*07.11.2022*
2.	*Розробка та затвердження завдання на роботу магістра (стац/заоч)*	*13.12.2022*	*13.12.2022*
3.	*Вступ та перелік літературних джерел*	*24.02.2023*	*24.02.2023*
4.	*Розробка технічного завдання*	*15.03.2023*	*15.03.2023*
5.	*Розділ 1. Аналіз технічної документації та стандартів як фундаменту функціонування програмних інтерфейсів*	*10.04.2023*	*10.04.2023*
6.	*Розділ 2. Розробка архітектури системи розробки та впровадження програмних інтерфейсів*	*24.05.2023*	*24.05.2023*
7.	*Розділ 3. Дані. склад, структура та принципи організації інформаційного забезпечення, вибір технологій доступу до баз даних*	*06.09.2023*	*06.09.2023*
8.	*Розділ 4. Опис розробки програмного забезпечення*	*06.09.2023*	*06.09.2023*
9.	*Розробка програми та методики тестування*	*18.10.2023*	*18.10.2023*
10.	*Написання наукової статті*	*17.05.2023*	*17.05.2023*
11.	*Керівництво користувача*	*25.10.2023*	*25.10.2023*
12.	*Висновки та пропозиції*	*01.11.2023*	*01.11.2023*
13.	*Здача випускної кваліфікаційної роботи на кафедру (перша перевірка)*	*06.11.2023*	*06.11.2023*
14.	*Підготовка автореферату та презентації доповіді*	*06.11.2023*	*06.11.2023*
15.	*Попередній захист випускної кваліфікаційної роботи*	*20.11.2023 – 24.11.2023*	*20.11.2023 – 24.11.2023*
16.	*Здача зброшурованої випускної кваліфікаційної роботи*	*27.11.2023*	*27.11.2023*
17.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*29.11.2023*	*29.11.2023*
18.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*05.12.2023-06.12.2023*	*05.12.2023-06.12.2023*

7. Дата видачі завдання «13» грудня 2022 р.

8. Науковий керівник випускної кваліфікаційної роботи Котенко Н.О.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Котенко Н.О.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Александров А. Ю.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускної кваліфікаційної роботи

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускну кваліфікаційну роботу

Випускна кваліфікаційна робота студента _____Александров А. Ю._____

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Котенко Н.О._____

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О. В._____

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Робота присвячена дослідженню взаємодії сучасних програмних систем та їх частин, включає проєктування та реалізацію сервісу розробки та впровадження прикладних програмних інтерфейсів як способу передачі даних та взаємодії систем.

Розробка здійснена з використанням мов програмування Python та JavaScript, бібліотек Django, FastAPI та React.

Випускна кваліфікаційна робота на тему «Програмний комплекс для розробки API» загальним обсягом 50 сторінок містить:

- 16 рисунків
- 3 таблиці

Ключові слова: інтерфейс, взаємодія

ABSTRACT

The work explores the interaction of contemporary software systems and their components, including the design and development of a service for developing and implementing application programming interfaces, which serve as a means of data transmission and system interaction.

The development process utilized Python and JavaScript programming languages, Django, FastAPI, and React libraries.

Final qualification paper, entitled "Software suite for API development", spans 50 pages and contains:

- 16 figures
- 3 tables

Keywords: interface, interaction

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

REST API - Representational State Transfer Application Programming Interface

URI – Uniform Resource Identifier.

HTTP – HyperText Transfer Protocol.

TCP/IP – Transmission Control Protocol/Internet Protocol.

OSI – The Open Systems Interconnection model

RFC – Request for Comments.

ОС – операційна система.

AWS – Amazon Web Services.

SQL – Structured Query Language.

UML – Unified Modeling Language.

MVC – Model View Controller.

СУБД – система управління базами даних.

TOAST – The Oversized-Attribute Storage Technique.

ACID – Atomicity, Consistency, Isolation, Durability.

JWT – JSON Web Token.

HTML – HyperText Markup Language.

DOM – Document Object Model.

UI – User Interface

ASGI – Asynchronous Server Gateway Interface.

JSON – JavaScript Object Notation.

ЗМІСТ

					ДТЕУ 121 02-1.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		01.11.23	*Програмний комплекс для розробки API*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Котенко Н.О.		01.11.23		*Зміст*	*3*	*50*
Гарант		Котенко Н.О.		01.11.23				
Розробив		Александров А. Ю.		01.11.23	*Зміст*	*Факультет інформаційних технологій 2м курс, 2 група*		

ВСТУП

Передача даних між вузлами – основна ідея глобальної мережі Інтернет, дані є фундаментом функціонування сучасних складних програмних систем, вони обробляють їх, зберігають та надають користувачам або іншим системам для подальшого використання.

Розроблюваний проєкт покликаний зробити імплементацію систем, які вимагають передачі даних у мережі, тривіальною задачею шляхом надання послуг зі створення програмних інтерфейсів, ресурсів для їх повноцінного впровадження. Користувачам не обов'язково глибоко володіти знаннями в області управління серверами, мережевої архітектури, безпеки чи програмування, для ефективного використання достатньо простого набору навичок.

Використання прикладних програмних інтерфейсів (API) як способу передачі даних є найбільш поширеним при створенні вебзастосунків, мобільних чи настільних додатків - для збереження даних у відповідь на користувацький запит, для їх відображення у необхідному форматі, чи внутрішньої взаємодії у системі.

Реалізація сервісу розробки та впровадження програмних інтерфейсів з можливістю їх зміни в реальному часі стає можливою через наявність значної кількості ресурсів у сучасних комп'ютерних систем – швидкодії збереження, обробки, читання та аналізу даних.

Актуальність. Прикладні програмні інтерфейси є основою взаємодії сучасних програмних застосунків, результатом дослідження є інструмент, який значно спрощує їх реалізацію та впровадження.

Мета дослідження: піддати науковому огляду взаємодію сучасних програмних засобів та розробити інструмент для підвищення ефективності розробки прикладних програмних інтерфейсів як способу взаємодії на основі отриманої інформації.

Об'єктом дослідження є компоненти сервісу розробки та впровадження програмних інтерфейсів.

Предмет дослідження – розробка програмного забезпечення.

Предметна область: хмарні обчислення, інтеграції програмних систем.

Пошуки шляхів досягнення мети обумовили необхідність визначення **наступних завдань**:

- піддати науковому аналізу способи та засоби взаємодії програмних систем, включно з клієнт-серверною;
- детальне проєктування компонентів програмного забезпечення - архітектури власне застосунку та моделей бази даних;
- імплементація компонентів, які разом складуть систему реалізації та впровадження прикладних програмних інтерфейсів.

Наукова новизна дослідження – підвищення рівня абстракції для розробки та впровадження прикладних програмних інтерфейсів, зменшення порогу знань, необхідних для цього при збільшенні ефективності.

Методи дослідження: порівняння, аналіз та синтез, сходження від абстрактного до конкретного.

РОЗДІЛ 1
АНАЛІЗ ТЕХНІЧНОЇ ДОКУМЕНТАЦІЇ ТА СТАНДАРТІВ ЯК ФУНДАМЕНТУ ФУНКЦІОНУВАННЯ ПРОГРАМНИХ ІНТЕРФЕЙСІВ

1.1. Аналіз способів взаємодії програмних систем. Клієнт серверна взаємодія

Взаємодія різних систем є ключовим аспектом у роботі програмного забезпечення. Системи можуть виробляти, обробляти та зберігати різні типи даних, і взаємодія між ними дозволяє обмінюватися цими даними для досягнення певних цілей. Кожна система має свої особливі можливості та обмеження, і взаємодія з іншими системами дозволяє розширити її функціональність, інтеграція з зовнішніми сервісами може доповнити існуючу систему новими можливостями.

Залежно від конкретного контексту та потреб системи використовуються різні підходи до взаємодії. Одним із найпоширеніших підходів є використання клієнт-серверної моделі. З таким підходом передбачається дві ролі, які отримують системи, - клієнт та сервер.

Клієнтська сторона відповідає за взаємодію з користувачем і формування запитів до сервера. Це може бути програмний додаток, веббраузер, мобільний додаток або будь-який інший інтерфейс, який дозволяє користувачу взаємодіяти з системою. Клієнтська сторона може забезпечувати візуальне відображення даних, обробку подій.

Серверна сторона відповідає за обробку запитів, виконання бізнес-логіки та надання відповідей клієнту. Це може бути фізичний сервер, хмарний сервіс або будь-який інший ресурс, який забезпечує необхідні послуги.

Клієнт-серверна модель дозволяє забезпечити розподілену обробку завдань, де різні сервери можуть виконувати окремі функції та обробляти запити від багатьох клієнтів. Це сприяє масштабованості та ефективності системи, спрощує розробку, тестування та підтримку системи.

У контексті клієнт-серверної взаємодії розроблюване програмне забезпечення покликане повністю або, у випадку розподілених системи - частково, заповнити роль сервера, обробляючи користувацькі запити на основі заздалегідь визначених структур програмних інтерфейсів.

Програмні інтерфейси - API (Application Programming Interface) - є важливим аспектом взаємодії між різними системами. API визначає набір правил та протоколів, які використовуються для взаємодії між програмними компонентами. Використання API дозволяє системам обмінюватися даними та виконувати операції через стандартизовані інтерфейси. Серед ряду наявних опцій - RPC, gRPC, GraphQL, SOAP та REST.

REST API (Representational State Transfer) є одним з найпоширеніших видів API для взаємодії між системами. REST API - архітектурний стиль, заснований на ряді обмежень, які й визначають вимоги та можливості, які будуть надаватися користувачам при побудові власних програмних інтерфейсів (подано на Рис. 1.1).

Рис. 1.1. Модель REST API застосунку

Серед обмежень, визначених архітектурним стилем:

- клієнт-сервер, або принцип розподілення відповідальності – відокремлення проблем користувацького інтерфейсу від проблем зберігання даних. Це дозволяє максимізувати портативність інтерфейсів та

масштабованість серверу, а також надає можливість розвиватися компонентам окремо;

- відсутність стану – комунікація між сервером та клієнтом не повинна мати збереженого стану, а вся інформація повинна міститися в кожному запиті, не покладаючись на контекст на сервері;
- кешування – визначення даних як таких, які або можуть бути або кешованими, або ні, і можливість перевикористання клієнтом отриманих даних при еквівалентних запитах. Перевага додавання обмежень кешу полягає в тому, що він може частково або повністю усунути деякі взаємодії, підвищуючи ефективність, масштабованість і продуктивність, яку сприймає користувач, за рахунок зменшення середньої затримки серії взаємодій. Однак компроміс полягає в тому, що кеш може знизити надійність, якщо застарілі дані в кеші значно відрізняються від даних, які були б отримані, якби запит було надіслано безпосередньо на сервер;
- уніфікований інтерфейс – використання стандартних URI для ідентифікації ресурсів, HTTP для опису комунікації, таким чином певна операція описується HTTP методом та URI;
- багатошарова система - поділ системи на різні шари, які не знають про деталі імплементації один одного [1].

Для правильного отримання, обробки та повернення даних потрібно визначити ряд стандартів та умов, дотримання яких забезпечує правильне, очікуване функціонування програмних інтерфейсів. Передача даних у мережі та загалом концепція функціонування мережі описується моделлю OSI (The Open Systems Interconnection model), а конкретна імплементація функціонуючої мережі забезпечується стеком протоколів TCP/IP (подано на Рис. 1.2).

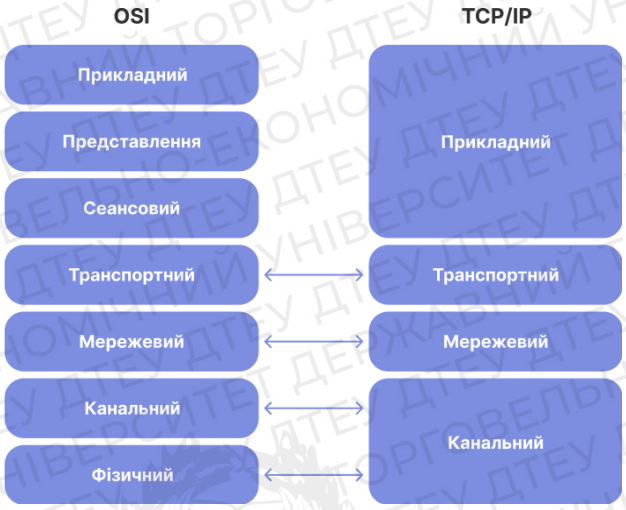


Рис. 1.2. Модель функціонування мережі

На найвищому рівні абстракції програмні інтерфейси оперують з прикладним рівнем моделі TCP/IP та OSI – рівнем взаємодії людини з комп'ютером, надають інтерфейс до даних для взаємодії з іншими серверами, іншим програмним забезпеченням або користувачами. На прикладному рівні моделі TCP/IP доступ до інтерфейсу забезпечується протоколом HTTP, описаним в стандарті RFC 9110.

HTTP – протокол прикладного рівня без збереження стану для розподілених спільних гіпертекстових інформаційних систем. HTTP приховує деталі того, як реалізовується служба, надаючи клієнтам єдиний інтерфейс, який не залежить від типів наданих ресурсів. Так само серверам не потрібно знати мету кожного клієнта: запит можна розглядати ізольовано, а не пов'язувати з конкретним типом клієнта чи попередньо визначеною послідовністю кроків програми. Це дозволяє ефективно використовувати реалізації загального призначення в багатьох різних контекстах, зменшує складність взаємодії та забезпечує незалежну еволюцію з часом [2].

Будь-яке повідомлення в рамках протоколу HTTP є або запитом або відповіддю - клієнт створює запит з необхідними даними та направляє його на потрібний шлях, сервер зчитує запит, формує та повертає відповідь клієнту.

Запит складається з таких основних частин:

- метод – індикатор наміру користувача – GET, POST, DELETE для читання, створення чи видалення даних відповідно, та ряд інших методів;
- ціль запиту – ідентифікатор ресурсів;
- набір заголовків – додаткова інформація для передачі на сервер;
- тіло запиту – основна інформація для передачі на сервер.

Відповідь складається з таких основних частин:

- статус – індикатор реакції серверу на відправлений раніше запит, може сигналізувати про успіх чи помилку та причини;
- набір заголовків;
- тіло відповіді.

Протокол HTTP покладається на нижчі рівні стеку TCP/IP і потребує їх для правильного функціонування. TCP – для транспортування та IP для ідентифікації серверу та клієнту.

1.2. Функціональні програмні інтерфейси. Бізнес-логіка

За виключенням примітивних програмних систем, функціональність яких полягає в здійсненні запитів до бази даних, складніші системи потребують виконання певної бізнес-логіки, яка визначає способи створення, представлення та зміни даних.

В умовах трирівневної архітектури шар бізнес-логіки є проміжним між шаром представлення та шаром доступу до даних у базі даних, на цьому рівні виконуються розрахунки, здійснюються логічні рішення. Шар визначає, яким чином використовуються дані з бази даних та як вони потрапляють у неї (подано на Рис. 1.3).

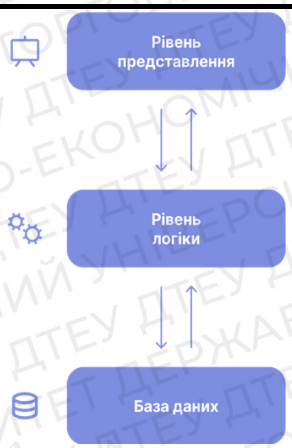


Рис. 1.3. Модель трирівневої архітектури програмної системи

Можливість виконання нестандартної бізнес-логіки потребує збору, збереження та виконання користувацьких інструкцій. Це створює проблему запуску ненадійного коду на використовуваному апаратному забезпеченні, що потенційно може призвести до компрометації даних чи будь-яких інших, руйнівних дій щодо працюючого апаратного та програмного забезпечення.

Вирішення такої проблеми потребує достатнього рівня ізоляції на різних рівнях, – по-перше, ізоляція від працюючого застосунку, по-друге, ізоляція запущених інструкцій між собою. Вирішення проблеми такого вигляду може реалізовуватися віртуалізацією чи контейнеризацією.

Контейнер – це стандартна одиниця програмного забезпечення, яка пакує код і всі його залежності, щоб програма швидко й надійно запускалася з одного обчислювального середовища в інше [3]. Віртуалізація сервера може забезпечити деякі переваги безпеки. Запуск сервера в гіпервізорі забезпечує пісочницю, яка може обмежити вплив компрометації, гіпервізор може забезпечити меншу поверхню для атаки, ніж хост-операційна система, зменшуючи можливість розширення успішної компрометації за межі гостьової ОС [4].

Таблиця 1.1.

Порівняння віртуальних машин та контейнерів

Особливість	***Віртуальні машини***	***Контейнери***
Розмір	Значно більший	Менший
Операційна система	Окрема	З основної системи
Швидкість запуску	Менша	Більша
Рівень ізоляції	Значний	Ізоляція присутня, але менша
Використання пам'яті та інших ресурсів	Вище, кожна віртуальна машина потребує ядро операційної системи та усе необхідне програмне забезпечення	Нижче, використовується базова операційна система
Збереження даних	Дані зберігаються між перезавантаженнями	Зазвичай без збереження стану
Мережева взаємодія	Окремі мережеві інтерфейси	Мережеві інтерфейси базової системи

Завдяки повній відокремленості віртуальних машин від операційної системи основної машини вони надають значний рівень ізоляції, при цьому кожна окремо запущена віртуальна машина потребує власної операційної системи, файлової системи та програмного забезпечення. Контейнеризація, з

іншого боку, більш легка та забезпечує менший рівень ізоляції, використовуючи основну операційну систему.

1.3. Порівняння аналогічного програмного забезпечення

Проведення порівняльного аналізу існуючих аналогічних програмних засобів дозволяє набути кращого розуміння про очікування потенційних користувачів, визначити поточні стандарти та практики у відповідній області – технічних вимог, щодо вибору мов програмування, побудови архітектури системи чи інших аспектів, а також бізнес-вимог. Окрім цього, аналіз аналогів дозволяє визначити можливі ризики та проблеми, які можуть виникнути під час розробки, впровадження чи функціонування програмного забезпечення, та мінімізувати їх.

Серед аналогічних, та таких, що можуть вважатися потенційно конкурентними розроблюваному програмному забезпеченню проєктів можна виділити наступні:

- AWS API Gateway;
- Directus.

AWS (Amazon Web Services) API Gateway - сервіс для створення, публікації, підтримки, моніторингу і захисту REST, HTTP і WebSocket API будь-якого масштабу [5]. Важливою особливістю є орієнтованість на роботу з іншими AWS-сервісами – базами даних (DynamoDB), безсерверними функціями (Lambda) та безліччю інших. Така особливість може вважатися перевагою, якщо брати до уваги різносторонність сервісів та, відповідно, можливість створення застосунків будь-яких розмірів в одному місці - AWS. Проте пов’язування сервісів між собою та з власне API Gateway, а також простота налаштування не є очевидними. Налаштування сервісів AWS не є елементарним процесом, вимагає визначеного набору знань та призначене для спеціалістів певного профілю.

Directus – сервіс для створення програмних інтерфейсів до SQL (structured query language) бази даних без застосування навичок програмування. Перевагою сервісу є простота налаштування, яка забезпечується примітивністю виконуваних задач, – створення REST API для записування, читання, оновлення, а також видалення даних з існуючої бази даних, – що, при цьому, є найбільш поширеним варіантом використання програмних інтерфейсів, і є достатнім для більшості виконуваних кінцевих завдань – створення вебсайтів, мобільних застосунків тощо. Для більш складних систем є можливість здійснення пошуку, фільтрування, агрегування даних, зміна схеми таблиць, налаштування автентифікації, шифрування.

Перевагою розроблюваного програмного забезпечення можна визначити простоту використання та налаштування програмних інтерфейсів, наявність вбудованої бази даних, інших ресурсів та засобів, складність розуміння, налаштування та впровадження яких зростає поступово.

Таблиця 1.2.

Порівняння аналогічних розроблюваному програмних засобів

Особливість	***AWS API Gateway***	***Directus***	***Розроблюваний сервіс***
Джерело даних	Зовнішнє, з ряду доступних в AWS баз даних	зовнішня SQL база даних	Вбудоване внутрішнє
Складність налаштування	Висока	Середня	Низька
Додаткова валідація/кастомізація даних	Так, використовуючи інші сервіси	Так	Так

Контроль доступу до даних	Так, використовуючи інші сервіси	Так	Ні
Автоматична документація	Ні	Ні	Так
Підтримувані типи API	REST, Websocket	REST, GraphQL, Websocket	REST

1.4. Висновки до розділу 1

У ході першого розділу було проаналізовано наявну документацію, якою забезпечуються основні компоненти розроблюваного програмного забезпечення та загалом його концепція. Визначено стандарти та підходи, які будуть забезпечувати працюючий сервіс зі створення програмних інтерфейсів.

Визначено основу роботи програмного забезпечення – мережеву взаємодію систем, для якої передбачається використання стеку протоколів TCP/IP як основи передачі даних в мережі та сутності програмних інтерфейсів, конкретного протоколу HTTP, який забезпечує передачу даних до та від програмних інтерфейсів. REST API – архітектурний стиль клієнт-серверної взаємодії, дотримання якого буде потребуватися від користувачів, а також є основоположною вимогою до реалізації програмних інтерфейсів з точки зору послуг сервісу. Проведено порівняльний аналіз віртуалізації та контейнеризації для забезпечення можливості виконання користувацьких інструкцій, бізнес-логіки без впливу на працююче програмне та апаратне забезпечення.

Проведено порівняльний аналіз з аналогічними програмними засобами та визначено ряд переваг та недоліків кожного, що дозволяє зрозуміти очікування користувачів та покращити їхній досвід, виявити ще не реалізовані можливості.

РОЗДІЛ 2
РОЗРОБКА АРХІТЕКТУРИ СИСТЕМИ РОЗРОБКИ ТА ВПРОВАДЖЕННЯ ПРОГРАМНИХ ІНТЕРФЕЙСІВ

2.1. Високорівневий огляд архітектури

Проєктований програмний продукт забезпечує можливість розробки та впровадження програмних інтерфейсів, що передбачає розмежування системи щодо двох основних видів користувачів - розробників, які створюють та налаштовують програмні модулі та кінцевих користувачів, які можуть їх використовувати.

Передбачається можливість надання послуги відразу багатьом розробникам API та можливість обробляти запити для кожного з користувачів таких API. Замість надання кожному окремому розробнику окремих екземплярів сервісу надається перевага багатоклієнтській архітектурі (multitenant architecture), де всі користувачі обслуговуються одним екземпляром сервісу. Такий підхід дозволяє значно полегшити розробку та впровадження програмних інтерфейсів.

Поділ сервісу на частини для кожного з користувачів відбувається створенням об’єктів “проєктів”, також на цьому рівні відбувається ізоляція даних - одні дані доступні в межах проєктів, але не поза ними. Кожен проєкт може містити більше одного користувача для можливості колаборації.

Задля можливості розробникам API створювати більш практичні архітектури, а також для стимулювання використання користувачами розподілених систем, кожен проєкт ділиться на модулі.

Модулі не знаходяться в ізоляції один від одного, можуть взаємодіяти з даними з інших, і є логічним поділом проєкту на частини. Модулі є найменшим структурним елементом та включають реалізації програмних інтерфейсів, обсягів даних та обробників даних.

2.2. Розробка моделей діючих компонентів, об'єктів системи та проєктування взаємозв'язків між ними

Проєктування архітектури програмного забезпечення є важливим етапом у розробці складних систем, це допомагає чітко розуміти структуру системи перед її реалізацією.

Для моделювання передбачається використання Unified Modeling Language (UML) діаграм, які фактично є галузевим стандартом візуалізації програмних систем. Створення UML діаграм надає наступні переваги:

- визначення структури системи, включаючи компоненти, модулі, їх взаємозв'язки та взаємодію, що може використовуватися для пришвидшення імплементації;
- забезпечення розширюваності та модифікованості – правильно спроєктована архітектура забезпечує гнучкість та можливість легко внести зміни до системи в майбутньому, зміни окремих компонентів не потребуватимуть значних змін змін у всій системі;
- виявлення ризиків та проблем – проєктування архітектури допомагає виявити потенційні ризики та проблеми ще на цьому етапі, це дозволяє прийняти правильні рішення та внести необхідні зміни до архітектури для запобігання проблемам під час реалізації та експлуатації системи;
- комунікація між командами – використання UML-діаграм дозволяє командам розробників, архітекторів та іншим зацікавленим сторонам взаємодіяти та спілкуватися на єдиній мові.

UML-діаграми дозволяють візуалізовувати програмні системи, створювати документації до них, при цьому спрощуючи, розкладаючи на менші, зрозуміліші компоненти. Діаграми дозволяють поглянути на систему з різних боків. Задовільні моделі можна охарактеризувати як:

- точні – вони повинні точно і правильно описувати систему, яку представляють;
- зрозумілі – повинні бути максимально простими;
- послідовні – різні представлення не повинні виражати речі, які суперечать одне одному;
- модифіковані – повинні бути легко змінюваними [6].

Основним компонентом об’єктно-орієнтованих систем є класи. UML-діаграма класів – графічна нотація для візуалізації таких компонентів та їх взаємозв’язків, класи інкапсулюють атрибути (стан) та поведінку (операції). Розроблена діаграма класів відображає структуру основного сервісу – сервісу обробки користувацьких запитів та роботи з даними.

Початковий клас – “ASGIApplication” репрезентує вхідну точку в застосунок. Клас може мати тільки один об’єкт, який створюється при запуску сервісу. Призначення класу – первинна обробка HTTP-запитів та повернення відповідей користувачам.

Клас “ASGIApplication” напряму взаємодіє з класом “Router”, який призначений для подальшої обробки запиту, перевірки чи запит з такою сигнатурою може бути оброблений.

Якщо існують дані, які описують, як обробити запит, “Router” взаємодіє з класом “View”. Даний клас може вважатися таким, що на високому рівні абстракції виконує такі функції, як і однойменний компонент моделі Model-View-Controller (MVC) або аналогічною. Призначення класу – обробка метаданих, серіалізація даних у взаємодії з класом “Serializer”, а також визначення і виконання потрібної дії у відношенні до бази даних,

повернення відповіді або помилки у форматі HTTP-відповіді. Окрім цього, клас взаємодіє з класом “PipeExecutor”, який виконує обробку даних відповідно до визначених розробниками інструкцій.

Останній клас “VolumeClient” репрезентує систему керування базою даних та безпосередньо виконує запити до неї, надаючи зручний інтерфейс класам вищого рівня. Однією з конкретних імплементацій клієнту для доступу до бази даних є клас “MongoVolumeClient” (подано на Рис. 2.1).

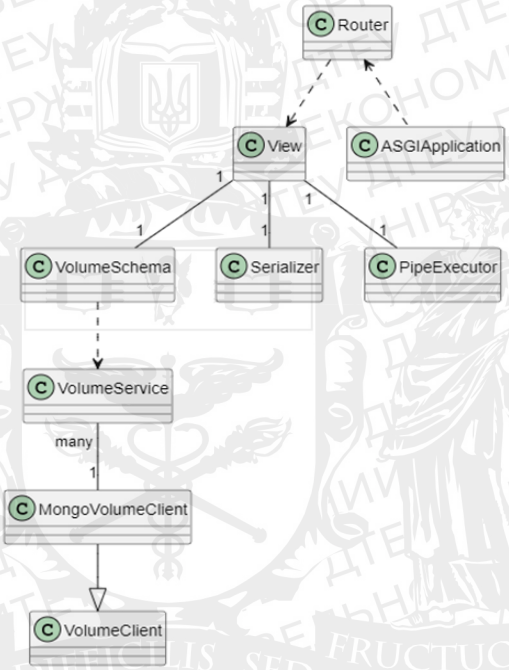


Рис. 2.1. Діаграма класів

Увесь потік даних через класи з діаграми класів можна розглянути на діаграм активностей. Діаграма активностей є потужним інструментом для візуалізації взаємодії з системою. Діаграма відображає шлях від запиту на створення даних до відповіді користувачу та включає:

- перевірку існування інтерфейсу – повинен існувати активний програмний інтерфейс із заданим шляхом та методом для поточного проєкту;
- перевірку правильності введених даних – поля повинні співпадати з визначеними для відповідного інтерфейсу розробниками API;

- пошук активного визначеного у базі даних обробника даних для запитів та його запуск;

- збереження даних у базу даних для можливості їх подальшого використання (подано на Рис. 2.2).

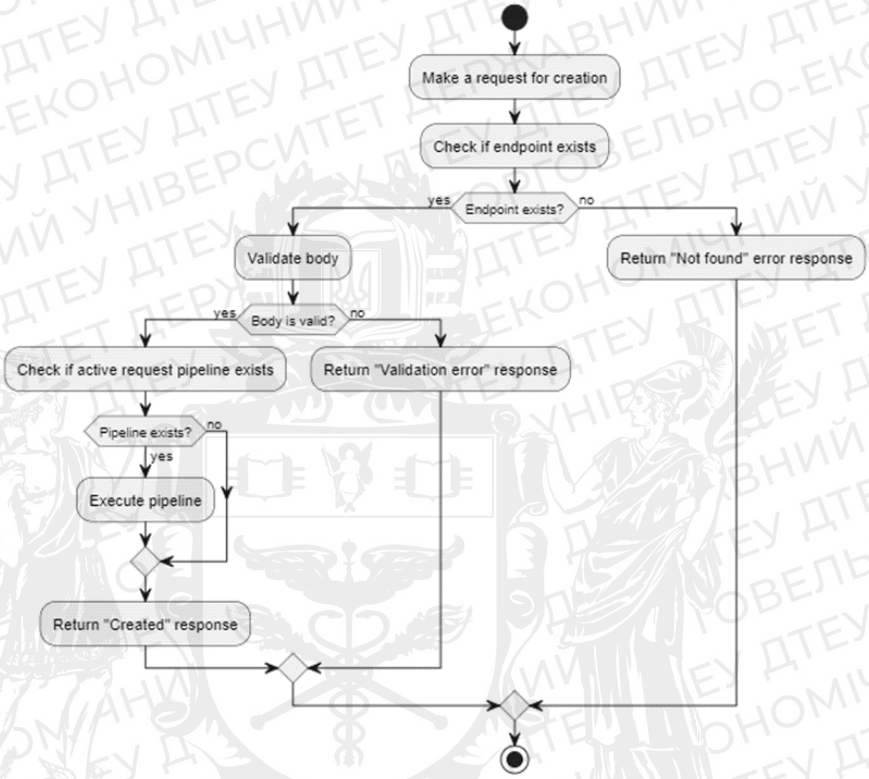


Рис. 2.2. Діаграма активностей записування даних

Наступна діаграма активностей відображає шлях від запиту на відображення списку раніше створених даних до відповіді та повернення списку або помилки користувачу та включає:

- перевірку існування активного програмного інтерфейсу в базі даних;

- отримання даних з бази даних по визначеному обсягу даних;

- серіалізацію даних у відповідності до визначеної розробниками інтерфейсів схеми;

- пошук активного визначеного у базі даних обробника даних для відповідей та його запуск;

- повернення даних (подано на Рис. 2.3).

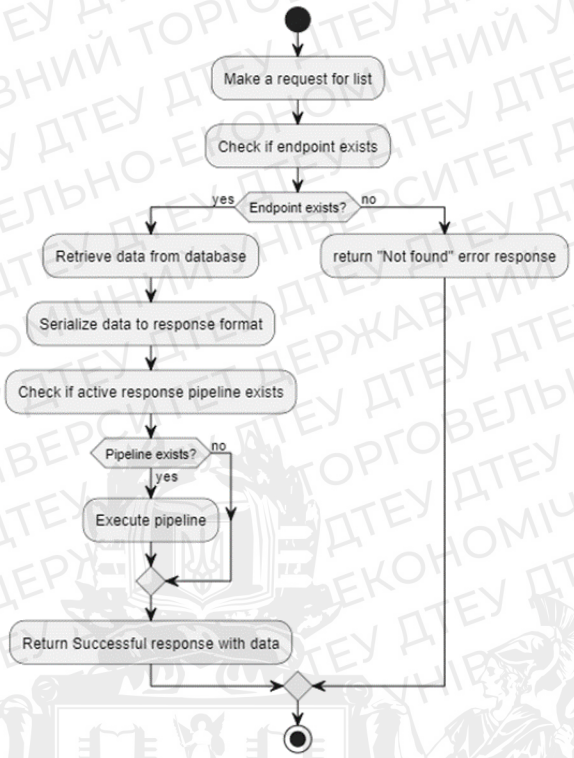


Рис. 2.3. Діаграма активностей читання даних

Іншим важливим компонентом є модель сценаріїв використання програмного забезпечення. Модуль для обробки запитів представлений одним актором – кінцевим користувачем попередньо імплементованих програмний інтерфейсів – таким користувачем може бути людина, у випадку прямих запитів, або ж інша система. Актору доступний ряд сценаріїв використання, які передбачають взаємодію з обсягами даних у відповідності до структури, визначеної користувачем-розробником:

- створення нового запису в обсязі даних, який в подальшому можна унікально ідентифікувати;
- читання конкретного запису за його унікальним ідентифікатором, визначеним при створенні;
- перегляд списку записів;
- оновлення запису – перевизначення даних, наданих при створенні, або попередніх оновленнях;
- видалення запису – безповоротне знищення запису з обсягу даних.

Модель сценаріїв використання є важливим компонентом процесу розробки програмного забезпечення, оскільки вона допомагає зрозуміти потреби та очікування користувачів, а також забезпечує чіткість та послідовність взаємодії з системою (подано на Рис. 2.4).

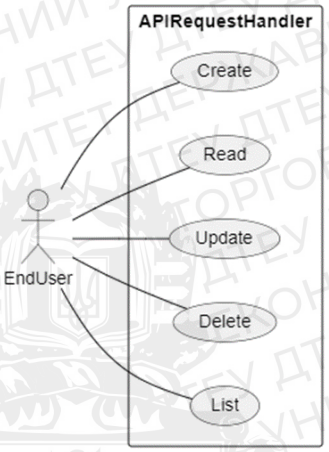


Рис. 2.4. Діаграма сценаріїв використання кінцевого користувача

Серед сценаріїв використання, доступних користувачам-розробникам, в контексті членства у певному проєкті, – можливість взаємодіяти у будь-якому форматі із рядом ресурсів – створювати, редагувати, переглядати або видаляти схеми програмних інтерфейсів, обсягів даних, обробників даних та груп обробників даних, модулів, а також можливість запрошувати нових користувачів до проєкту [Додаток А].

Схема розгортання зображає високорівневу архітектуру системи, поділеної на дві частини - для розробників програмних інтерфейсів та для кінцевих користувачів.

Система для користувачів-розробників включає серверну частину, яка надає програмні інтерфейси та користувацький інтерфейс, який їх використовує, на цьому рівні передбачається база даних для схем різних компонентів.

Система для кінцевих користувачів включає серверну частину, яка надає програмні інтерфейси на основі створених схем, на цьому рівні

передбачається база даних для користувацьких даних. Окремим компонентом є сервіс для динамічної побудови документації (подано на Рис. 2.5).

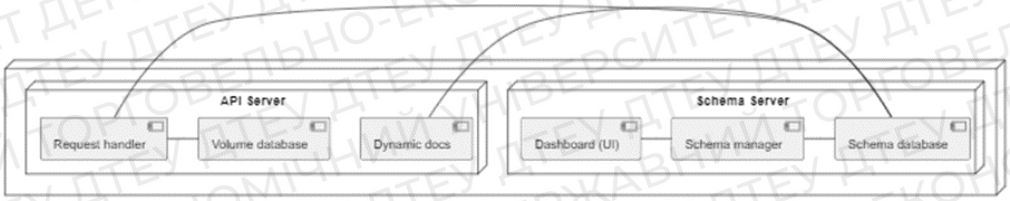


Рис. 2.5. Діаграма розгортання

Діаграми послідовності, використовуючи компоненти з діаграми розгортання та включаючи функціональні деталі з діаграми використання відображає процес створення програмного інтерфейсу користувачем-розробником, який взаємодіє з користувацьким інтерфейсом, а також використання даного програмного інтерфейсу кінцевим користувачем, який робить запит до нього напряму.

Для створення програмного інтерфейсу користувач-розробник повинен використати користувацький інтерфейс – заповнити форму, при заповненні її правильними даними користувацький інтерфейс здійснює запит до серверу, який зберігає дані в базі даних та повертає відповідь користувачу (подано на Рис. 2.6).

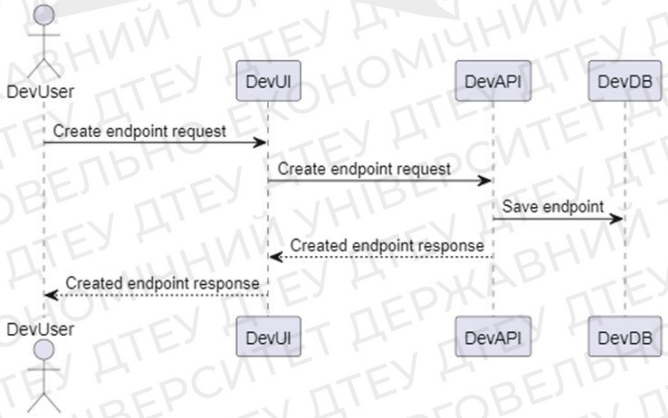


Рис. 2.6. Діаграма послідовності створення програмного інтерфейсу

Для використання програмного інтерфейсу кінцевий користувач або ж розроблений користувачем користувацький інтерфейс повинен здійснити запит на сервіс розробки та впровадження програмних інтерфейсів, який

отримає з бази даних дані, визначені розробником у відповідності до визначеної схеми, обробить їх за потреби, та поверне у відповіді користувачу [Додаток Б].

Важливою функцією для можливості взаємодії на рівні проєктів є додавання нових користувачів шляхом відправки запрошення. Для цього розробник, взаємодіючи з користувацьким інтерфейсом, надає електронну пошту потенційного користувача. Електронна пошта зберігається локально разом з довільним унікальним значенням – токеном – для подальшої перевірки. На вказану адресу разом з токеном відправляється посилання для можливості прийняти запрошення і стати учасником проєкту [Додаток В].

2.3. Обґрунтування вибору операційної системи, мови програмування та інших технологій

Вибір апаратного та програмного забезпечення для реалізації проєкту має вирішальне значення для забезпечення оптимальної продуктивності та масштабованості, особливо в умовах високої навантаженості.

З метою забезпечення максимальної стабільності та безпеки системи для розгортання програмного забезпечення, було обрано операційну систему з сімейства Linux. Linux відомий своєю надійністю та широкою підтримкою спільноти користувачів та розробників. Це забезпечує широкі можливості для підтримки та розвитку проєкту у майбутньому.

Мова програмування Python була обрана через свою простоту та зручність для розробки вебдодатків. Python є популярною мовою в галузі розробки вебдодатків та машинного навчання, що забезпечує доступність багатьох бібліотек та інструментів для швидкої та ефективної розробки.

Python, як мова для основної частини програмної системи має наступні переваги:

- швидкодія працюючого застосунку – Python може легко справлятися з обробкою великої кількості запитів;

- простота розробки та підтримки - Python має простий та зрозумілий синтаксис;

- широка спільнота – Python має велику та активну спільноту розробників, ресурсів, форумів, бібліотек та інструментів.

Для забезпечення зручності та полегшення реалізації передбачається використання додаткових бібліотек. Django – для серверної частини інтерфейсу розробників програмних інтерфейсів, бібліотека надає зручні можливості для доступу до бази даних та для швидкої розробки шаблонних програмних інтерфейсів для створення, редагування та видалення різних об’єктів з бази даних. Окрім цього, Django надає зручний функціонал для відправки електронної пошти. Іншою бібліотекою є FastAPI – для сервісу динамічної документації, який не вимагає більшості можливостей, які надає Django, та має бути відокремлений від нього для масштабованості.

Задля ефективної розробки користувацького інтерфейсу передбачається використання мови JavaScript, серед переваг якої:

- підтримуваність – JavaScript надає можливість створювати функціональні та інтерактивні сторінки, які працюватимуть у всіх сучасних браузерах;

- динамічність – JavaScript дозволяє створювати динамічні сторінки, які можуть змінюватися у реальному часі відповідно до дій користувачів, що робить користувацький досвід більш цікавим та зручним;

- розширюваність – JavaScript має велику кількість бібліотек та фреймворків, які допомагають спростити розробку та розширення функціональності вебінтерфейсу.

Враховуючи ці переваги, мова програмування JavaScript є ідеальним вибором для розробки користувацького інтерфейсу проєкту будь-якої складності. Для легкого та ефективного використання усіх переваг JavaScript передбачається використання бібліотеки React.

Використання JavaScript разом зі зручною інтеграцією з мовою Python та операційною системою Linux створює потужну комбінацію для ефективної та стабільної розробки програмного забезпечення.

Крім цього, для забезпечення ізольованості, масштабованості та зручності розгортання різних частин програмного забезпечення існує потреба у контейнеризації. Такий підхід передбачає використання ядра операційної системи, в якій відбувається запуск, що є більш оптимальним порівняно з віртуалізацією. Найбільш поширеним інструментарієм для виконання такої задачі є Docker. Він надає зручний інтерфейс для створення, запуску та подальшої підтримки контейнерів. Серед інших переваг використання контейнеризації з Docker:

- портативність розгортання – програмні застосунки, побудовані всередині контейнерів, є надзвичайно портативними, переміщуються між системами як єдине ціле, і цей рух не впливає на продуктивність;

- швидкість доставки – через стандартизований формат контейнерів, командам розробки не потрібно турбуватися про задачі один одного - розробники турбуються про програми всередині контейнерів, а адміністратори лише про розгортання серверів з контейнерами;

- швидший час створення – контейнери дуже маленькі і швидко створюються, що дозволяє скоротити час тестування, розробки та розгортання [7].

У середовищі для виконання користувацьких інструкцій обробників даних ізоляція з використанням контейнерів є недостатньою, є потреба у максимальному відокремленні виконуваного коду, як від основної операційної системи так і від коду, виконуваного у інших середовищах. Рішенням, яке може забезпечити максимальну ізоляцію, є віртуалізація – створення віртуального середовища, програмної реалізації комп’ютера, яка виконує команди подібно до справжнього, має власну операційну систему.

Для цього обрано Firecracker – технологію. віртуалізації з відкритим вихідним кодом, яка спеціально створена для запуску та керування захищеними багатокористувацькими контейнерними та функціональними службами [8]. Firecracker дозволяє запускати ізольовані віртуальні машини з мінімальним використанням ресурсів. Для зручності взаємодії з віртуальними машинами Firecracker передбачається використання Weave Ignite - менеджеру віртуальних машин, який дозволяє взаємодіяти з ними, як з контейнерами. З таким набором інструментів зі зручними інтерфейсами використання ізольованих віртуальних машин стає схожим на використання контейнерів.

2.4. Висновки до розділу 2

У ході розробки даного розділу було створено архітектуру програмного забезпечення у відповідності до вимог та потреб – ряд UML-діаграм, як відображають основні структурні елементи на різних рівнях абстракції, а також зв’язки між ними, ролі користувачів у процесах.

Обрано мови програмування та бібліотеки для серверної та клієнтської частин, які забезпечать ефективність розробки та зручність подальшої підтримки програмного забезпечення. Визначено підходи та відповідні технології для ізоляції, для частин програмного застосунку – контейнеризація, для ізоляції виконуваних користувацьких інструкцій – віртуалізація.

Спроєктована архітектура, імплементована з використанням обраних програмних та апаратних засобів, дозволить створити всі умови для успішної розробки програмного забезпечення та виконання усіх вимог.

РОЗДІЛ 3
ДАНІ. СКЛАД, СТРУКТУРА ТА ПРИНЦИПИ ОРГАНІЗАЦІЇ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ, ВИБІР ТЕХНОЛОГІЙ ДОСТУПУ ДО БАЗ ДАНИХ

3.1. Високорівневе проєктування баз даних

Ефективність роботи програмного забезпечення залежить від правильної структури баз даних, їх організації. Недостатня якість бази даних може призвести до більших витрат часу та коштів, тому ретельне проєктування бази даних є необхідністю для успіху розробки та роботи програмного забезпечення.

Програмний комплекс передбачає використання даних двох різних видів – структурованих та неструктурованих. Для зберігання інформації, важливої для загального функціонування сервісу, такої як, дані про користувачів, а також схем користувацьких структур є потреба у структурованій базі даних з чітко визначеними таблицями. Для зберігання користувацьких даних, де розробники програмних інтерфейсів визначають структуру таблиць, можуть змінювати її без значних обмежень у будь-який час, потрібна динамічна база даних.

Найбільш ефективною опцією зберігання структурованих даних є використання реляційної бази даних, де кожен рядок даних відображає певну сутність, а стовпець – конкретний атрибут, який стосується всіх сутностей.

Для створення, читання, оновлення чи видалення даних використовується проста та ефективна мова запитів SQL (Structured Query Language). Для забезпечення структури, яка б оптимально відповідала потребам додатка та користувачів необхідно спроєктувати таблиці бази даних. Правильність структури дозволяє ефективно виконувати запити до бази даних, а також забезпечує легкість модифікації та розширення програмного забезпечення в майбутньому.

В контексті предметної області, сутність “Проєкт” відіграє роль основного контейнера для групування модулів. Кожен проєкт може мати декілька модулів, що дозволяє організувати структуру та створити ієрархію даних. Проєкт може представляти вебдодаток, а його модулі - окремі функціональні частини такі як аутентифікація, створення постів або оплата послуг.

Сутність "Project Invitation" надає можливість надсилати запрошення до проєктів для співпраці.

"Endpoint" визначає метадані для створення прикладних програмних інтерфейсів. Ця сутність включає в себе інформацію про тип HTTP-запиту (GET, POST, PUT, DELETE тощо) та місце зберігання або джерело даних, з якого програмний інтерфейс отримує або зберігає інформацію – один з обсягів даних, а також шлях, за яким можна отримати доступ до програмного інтерфейсу.

Сутність "Volume" визначає метадані для таблиць, які використовуються програмними інтерфейсами. Вона включає в себе список полів (атрибутів), які можуть бути використані для зберігання або отримання даних.

Сутності "Pipeline" та "Pipe" відіграють важливу роль у обробці даних під час взаємодії з програмними інтерфейсами. "Pipeline" представляє групу обробників даних, які можуть бути виконані для певного програмного

інтерфейсу для обробки та маніпуляції з даними. “Ріре" містить код, який виконується при обробці запиту користувача до програмного інтерфейсу. Цей код може включати в себе валідацію, зміни або перетворення даних залежно від бізнес-правил та вимог системи, яку будують користувачі.

Сутність “User” представляє загального користувача у системі, без прив’язки до проєкту. Користувач може бути членом декількох проєктів, а проєкт може мати декілька користувачів, що дозволяє встановлювати відносини співпраці та обміну даними між різними учасниками системи.

Всі ці сутності та їх взаємозв'язки є важливими для визначення структури та функціональності бази даних сервісу створення API. Ретельний аналіз предметної області та вимог дозволяє зрозуміти особливості домену, встановити вірні взаємозв'язки між сутностями та визначити необхідні атрибути та обмеження. Цей етап є основою для подальшого проєктування концептуальної, логічної та фізичної моделей бази даних.

3.2. Проєктування концептуальної моделі реляційної бази даних

Концептуальна модель реляційної бази даних є першим кроком у процесі проєктування бази даних і відображає сутності, їх атрибути та зв'язки між ними. Ця модель є абстрактною та незалежною від конкретної реалізації бази даних і дозволяє уявити загальну структуру даних та їх взаємозв'язки у предметній області.

У концептуальній моделі, кожна сутність представляється як окрема таблиця. Зв'язки між сутностями відображаються у вигляді зв'язків між таблицями. Цей зв'язок представляється за допомогою зовнішнього ключа, що посилається на первинний ключ іншої таблиці.

На розробленій концептуальній моделі можна спостерігати усі основні сутності, які використовуються в системі, та як вони взаємодіють між собою. Первинною сутністю є користувачі, які створюються в

результаті успішної реєстрації. Кожен з користувачів може створювати безліч проєктів та, з іншого боку, кожен з проєктів може мати безліч користувачів як своїх учасників. Кожен з проєктів може вміщати безліч модулів, а модулі – найменших функціональних елементів – програмних інтерфейсів та обсягів даних. Обсяги даних визначаються на рівні модулів та можуть використовуватися великою кількістю програмних інтерфейсів одночасно.

Окрім цього, кожен з програмних інтерфейсів може мати декілька груп обробників даних, які включають власне обробники (подано на Рис. 3.1).

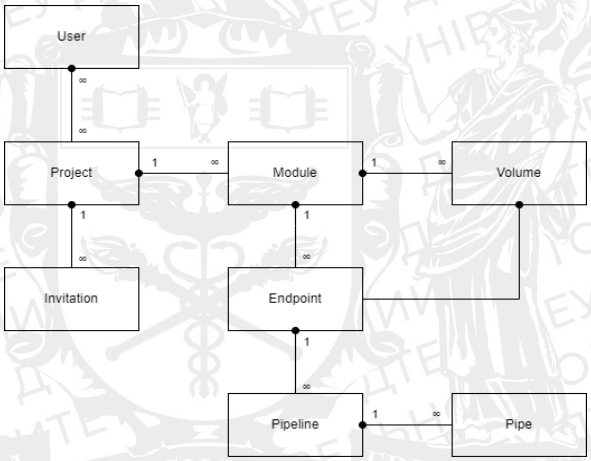


Рис. 3.1. Концептуальна модель даних

3.3. Проєктування логічної моделі реляційної бази даних

Логічна модель реляційної бази даних конкретизує концептуальну модель, враховуючи деталі реалізації бази даних. Вона визначає структуру таблиць, відношення між ними та обмеження, які накладаються на дані.

У логічній моделі, кожна сутність концептуальної моделі відображається як таблиця зі своїми атрибутами. Логічна модель бази даних важлива для подальшого фізичного проєктування бази даних.

Таким чином, логічна модель реляційної бази даних деталізує структуру та взаємозв'язки даних, що були визначені у концептуальній моделі, і створює основу для фізичного реалізації бази даних.

На логічній моделі можна спостерігати конкретні атрибути сутностей, включно з ключами, які забезпечують взаємозв’язки між таблицями. Кожна з таблиць має первинний ключ – атрибут, який однозначно ідентифікує запис в межах таблиці. Окрім первинного, деякі з таблиць мають також зовнішній ключ, який визначає відношення до іншої таблиці.

Більшість з таблиць включають поле з назвою для зручності ідентифікації сутностей користувачами, таблиці програмних інтерфейсів та обробників даних, які використовуються кінцевими користувачами мають поля, які визначають їх активність, визначення таких сутностей як неактивних виключає потребу їх повного видалення та забезпечує гнучкість розробки (подано на Рис. 3.2).

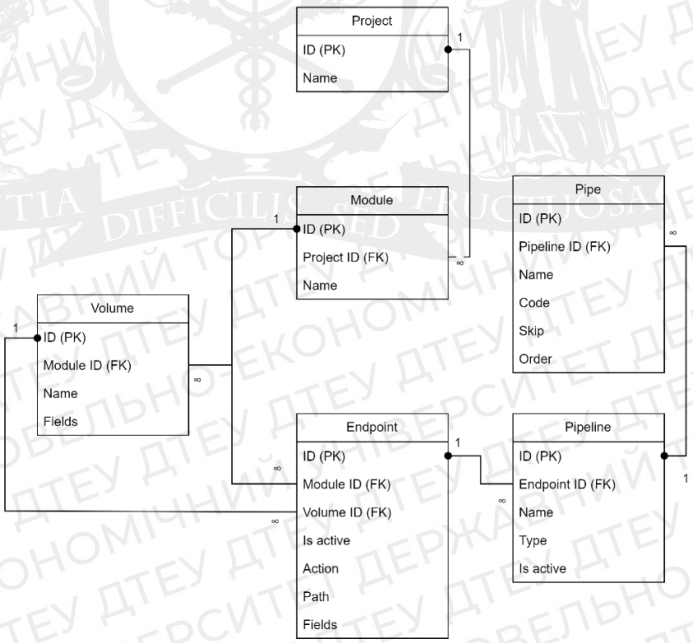


Рис. 3.2. Логічна модель даних у контексті проєкту

Окремою частиною логічної моделі даних є модель для відображення зв’язків проєкту та загального користувача системи, включає таблицю, яка пов’язує їх напряму та репрезентує членство користувача у проєкті. Окрім

цього, наявна таблиця запрошення, що є наміром додавання нового учасника до проєкту, містить електронну адресу користувача, куди відправляється запрошення, токен – значення-ідентифікатор запрошення, а також дату створення і дату, коли запрошення стає неактивним [Додаток Г].

3.4. Проєктування фізичної моделі реляційної бази даних

Фізична модель реляційної бази даних визначає спосіб фізичного збереження даних на певній платформі системи управління базами даних (СУБД) з урахуванням особливостей конкретної реалізації.

У фізичній моделі визначаються таблиці з їх структурою, типами даних, обсягами, індексами, обмеженнями та іншими аспектами, що впливають на ефективність роботи з базою даних. Кожна таблиця має поля (стовпці) з відповідними типами даних. Для забезпечення цілісності даних встановлюються обмеження, такі як унікальність, зовнішні ключі та правила цілісності.

На цьому етапі проєктування враховується оптимізація запитів та використання пам'яті, визначається стратегія фізичного збереження даних, розміщення таблиць на диску, розподіл даних на фізичні пристрої та створення індексів для підвищення продуктивності запитів. Фізична модель бази даних є остаточним кроком перед реалізацією бази даних і визначає деталі збереження та оптимізації даних, необхідні для ефективної роботи з базою даних на певній платформі СУБД.

На розробленій моделі даних, яка відображає таблиці у контексті проєктів додатково до даних з логічної моделі, відображаються типи даних з обмеженнями – максимальними розмірами рядків, можливістю запису пустих значень, унікальність даних. Усі обмеження визначені у відповідності до можливих значень у контексті предметної області (подано на Рис. 3.3).

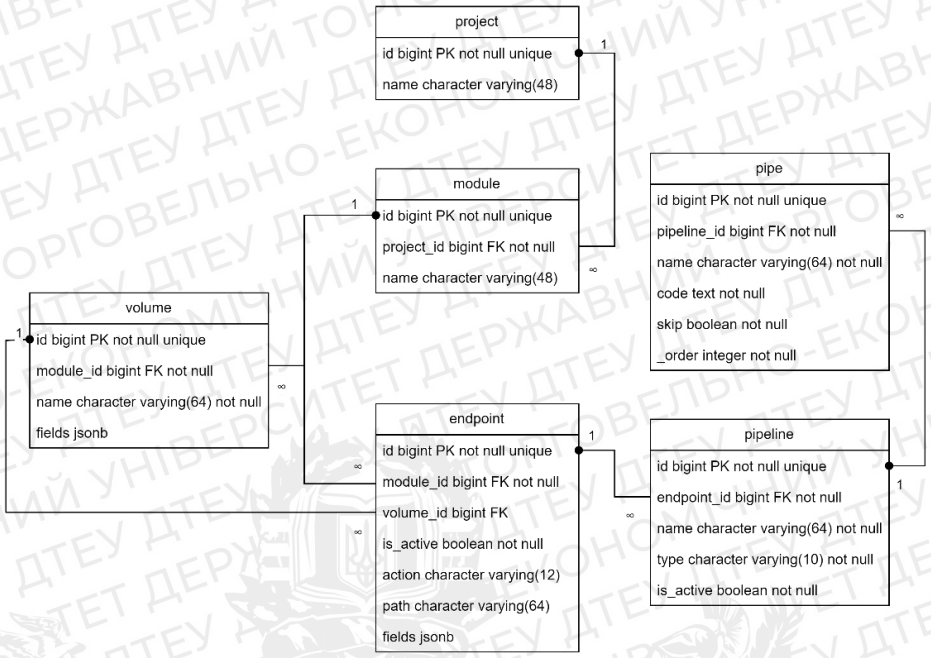


Рис. 3.3. Фізична модель даних у контексті проєкту

Використовуючи PostgreSQL передбачається використання TOAST (The Oversized-Attribute Storage Technique), що надає можливість обрати стратегію для зберігання кожного із стовпців у таблицях. Для усіх типів даних, які це підтримують обрано стратегію «extended», що дозволяє проводити компресію збережених даних і зменшувати використовуване місце на диску. Окрім цього, для збільшення ефективності виконання запитів, кожен з первинних та зовнішніх ключів мають індекси.

Окремою частиною фізичної моделі даних є зображення зв'язку проєктів та користувачів [Додаток Д].

3.5. Архітектура NoSQL бази даних кінцевих користувачів

Для збереження даних кінцевих користувачів найбільш доцільним є використання NoSQL бази даних. NoSQL бази даних відрізняються від реляційних баз даних тим, що не потребують жорсткої, наперед встановленої схеми для таблиць, є динамічними і готові до непередбачуваних змін у структурі даних. Замість жорсткої фіксації схеми

таблиць, NoSQL бази даних дозволяють додавати, видаляти або змінювати поля в документах без необхідності перетворення всіх даних.

Крім того, NoSQL бази даних добре масштабуються. Вони забезпечують можливість розподіленої обробки даних, що дозволяє працювати з великими обсягами інформації. Це важливо для сервісу, який має обробляти багато запитів та забезпечувати швидкий доступ до даних.

Таблиця 3.1.

Порівняння NoSQL баз даних

Назва	***Модель даних***	***Підтримка ACID***	***Open source***
MongoDB	Документ	Так	Так, з обмеженнями
Cassandra	З широким стовпчиком	Ні	Так
Redis	Ключ-значення	Так	Так
Couchbase	Документ	Так	Так
Neo4j	Граф	Так	Так, з обмеженнями
Amazon DynamoDB	Ключ-значення	Так	Ні

З усіх перелічених баз даних, найбільш задовільним варіантом для визначених сценаріїв є MongoDB. Його гнучка схема та можливість динамічно змінювати структуру документів забезпечують легкість розширення та адаптації до змін вимог користувачів. MongoDB пропонує горизонтальну масштабованість та високу доступність даних, що робить його привабливим вибором для вебдодатків з великою кількістю запитів. При цьому, широка підтримка, активна спільнота та розгалужена екосистема роблять його потужним інструментом для обробки та можливого подальшого аналізу великих обсягів даних.

Окрім цього, MongoDB підтримує ACID, що забезпечує зберігання правильного стану бази даних при виконанні групи операцій (транзакції) навіть при виникненні помилок.

Користувацька база даних не передбачає визначення попередніх моделей даних, структура визначається динамічно у відповідності до створених користувачами-розробниками обсягів даних.

3.6. Висновок до розділу 3

Ретельний аналіз предметної області дозволив зрозуміти основні вимоги та особливості сервісу створення програмних інтерфейсів. Визначення концептуальної моделі дозволило ідентифікувати ключові сутності та взаємозв'язки, які будуть використовуватися в системі. Це стало основою для подальшої деталізації у логічній моделі, де були визначені специфікації сутностей, їх атрибути та відношення. Створення фізичної моделі бази даних вирішила питання фізичної реалізації бази даних на конкретній платформі.

Проведені дослідження та проєктування бази даних допомагають створити стійку, ефективну та масштабовану систему. Вони забезпечують коректне збереження даних, їх консистентність та доступність для різних функцій та процесів сервісу. Крім того, вони дозволяють врахувати специфічні потреби бізнесу та забезпечити відповідність функціональним вимогам системи, користувачів. Правильно спроєктована база даних є ключовим елементом успішної розробки та подальшої експлуатації програмного забезпечення.

РОЗДІЛ 4
ОПИС РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Імплементація системи розробки програмних інтерфейсів

Система розробки програмних інтерфейсів, побудована на основі Django, розділена на декілька Django-додатків для кращої організації функціональності та легкості розвитку системи. Кожен додаток відповідає за конкретну сферу системи і має свою визначену роль.

Додаток "accounts" включає функціональність, пов'язану з автентифікацією та управлінням користувачами-розробниками. Він містить моделі для зберігання даних користувачів, представлення та контролери для обробки реєстрації та автентифікації користувачів.

Автентифікація забезпечується використанням JWT (JSON Web Token). JWT-токени забезпечують безсесійну автентифікацію – сервер не зберігає стан сесії для кожного користувача, замість цього, токен надсилається разом з кожним запитом і перевіряється сервером для підтвердження автентичності та повноважень користувача.

Однією з переваг використання JWT-токенів є їх самодостатність. Усі необхідні дані для автентифікації містяться безпосередньо в токені, що дозволяє зменшити навантаження на сервер та знизити частоту обміну даними.

Додаток "projects" включає функціональність, пов'язану з керуванням проєктами. Він містить моделі для зберігання даних проєктів та модулів, представлення та контролери для їх створення, оновлення та видалення, роботу з учасниками проєктів.

Можливість взаємодії декількох користувачів над одним проєктом включає потребу у створенні та відправці запрошень користувачам поза проєктом. Робота запрошень забезпечується унікальними ідентифікаторами, які зберігаються у базі даних. Запрошення у проєкт можуть мати визначений користувачем час, після якого воно стає неактивним, а способом їх доставки є відправка електронною поштою на визначені електронні адреси [Додаток Е].

Для забезпечення найкращого користувацького досвіду відправка запрошень також вимагає HTML-шаблону, який відображає наміри відправника електронного листа та містить елементи, які надають можливість прийняти запрошення [Додаток Ж].

Додаток "endpoints" включає функціональність, пов'язану з керуванням програмними інтерфейсами. Він містить моделі для зберігання даних про програмні інтерфейси, представлення та контролери для їх створення, оновлення та видалення. Окрім програмних інтерфейсів, у цьому додатку здійснюється менеджмент обробників даних та їх груп.

Додаток "volumes" включає функціональність, пов'язану з управлінням ємностями даних. Він містить моделі для зберігання метаданих таблиць, представлення та контролери для визначення структури таблиць, роботу з полями.

Така організація системи на рівні Django додатків дозволяє розділити функціональність на логічні модулі, які можуть бути розроблені та модифіковані незалежно один від одного. Крім того, це сприяє кращому керуванню залежностями, полегшує розробку, тестування та підтримку системи. Кожен Django-додаток має свою визначену функціональність і може бути розглянутий як окремий модуль, який вносить свій внесок у роботу системи в цілому.

Використання Django додатків дозволяє забезпечити високу перевикористовуваність коду, кожен додаток може бути використаний в інших проєктах або навіть самостійно. Це сприяє швидкому розвитку нових проєктів і спрощує роботу з вже наявними.

4.2. Імплементація інтерфейсу розробнкиів програмних інтерфейсів

Користувацький інтерфейс для розробників програмних інтерфейсів імплементується з використанням JavaScript React. Одним з основних принципів React є використання компонентного підходу до розробки. Компоненти є основними будівельними блоками додатку і дозволяють структурувати інтерфейс на невеликі, повторно використовувані частини. Кожен компонент має свій стан, властивості та методи, які визначають його поведінку та взаємодію з іншими компонентами. Завдяки компонентній архітектурі, розробка складних інтерфейсів стає більш організованою та керованою.

Одна з ключових переваг React полягає в його віртуальному DOM (Document Object Model) – механізмі, який дозволяє ефективно оновлювати інтерфейс, враховуючи зміни в стані додатку. Використання віртуального DOM забезпечує швидке перемальовування елементів та мінімізує навантаження на браузер.

Окрім React, для розробки використовується Material UI – високоефективний інструмент для розробки користувацьких інтерфейсів, надає великий набір готових компонентів, які значно пришвидшують реалізацію користувацьких інтерфейсів. Це дозволяє ефективно будувати інтерфейси без необхідності великих обсягів власного коду і зусиль.

Окрім компонентів, розробка інтерфейсу включає створення сторінок - компонентів, які відповідають за відображення цілісних сторінок у

програмному забезпеченні. Вони складаються з інших компонентів і можуть виконувати специфічні функції.

Серед основних функціональних сторінок користувацького інтерфейсу є:

- сторінка реєстрації з формою для створення нового користувача;
- сторінка автентифікації з формою для вводу даних вже існуючого користувача (подано на Рис. 4.1);

Рис. 4.1. Форма автентифікації користувача

- список проєктів – сторінка з можливістю створювати новий проєкт та перейти до деталей існуючого [Додаток И];
- деталі проєкту – сторінка зі списком модулів обраного проєкту, з можливістю створити новий модуль, перейти до існуючого модуля, а також налаштуваннями проєкту, включно з можливістю відправляти запрошення і переглядати наявних членів проєкту;
- список кінцевих точок – сторінка зі списком кінцевих точок та можливістю створювати, редагувати, видаляти та активувати їх (подано на Рис. 4.2);

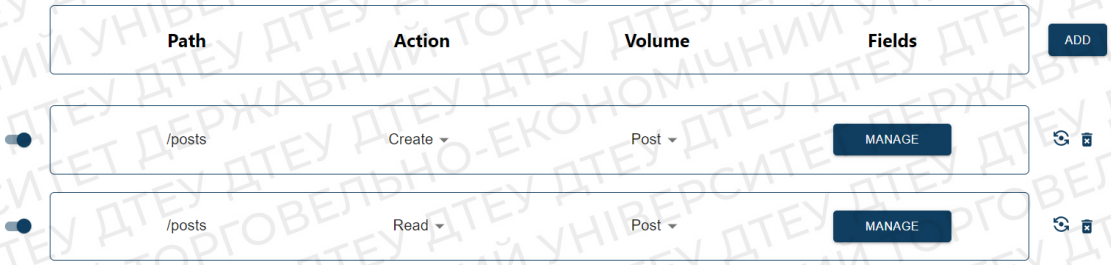


Рис. 4.2. Форма редагування програмних інтерфейсів

- список обсягів даних – сторінка зі списком обсягів даних та можливістю створювати, редагувати та видаляти їх;
- сторінка з обробниками даних для їх створення та налаштування для обраних програмних інтерфейсів (подано на Рис. 4.3);

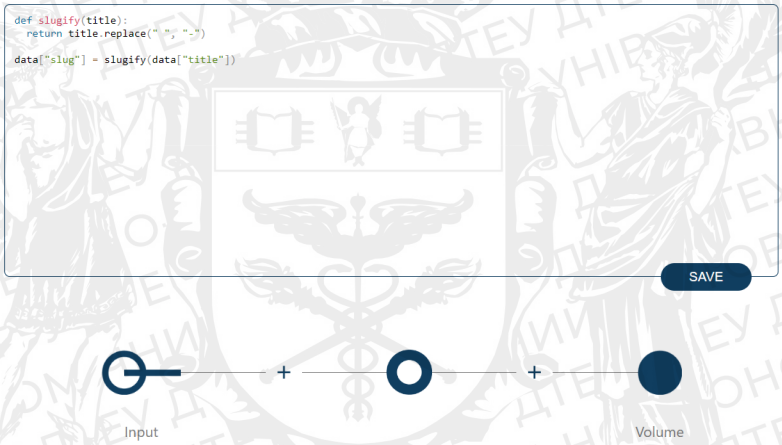


Рис. 4.3. Форма обробника даних програмного інтерфейсу

Кожна з форм передбачає валідацію введених користувачами даних – формату, довжини чи інших атрибутів. Помилки валідації проведені на рівні користувацького інтерфейсу або повернуті з серверу після запиту відображаються користувачу для розуміння проблеми та можливості її вирішення (подано на Рис. 4.4).

Рис. 4.4. Сповіщення про помилку авторизації

Іншою важливим архітектурним компонентом є хуки (hooks) – спеціальні функції, які мають ряд переваг – оптимізація завантаження сторінок через відсутність повторних створень таких функцій, можливість

використовувати хуки, які надає React, всередині. Вони дозволяють виділити загальну логіку в окремі функції, які можна використовувати в багатьох компонентах, спрощують управління станом компонентів, можуть відповідати за запити на програмні інтерфейси – зчитування даних з бази даних, запис даних та інші операції.

4.3. Імплементація системи динамічних програмних інтерфейсів

Працездатність програмних інтерфейсів, створених розробниками, забезпечується Python-застосунком, який імплементує ASGI (Asynchronous Server Gateway Interface) – стандартний інтерфейс, який дозволяє відділити серверну частину від бізнес-логіки програмного забезпечення. ASGI-клас – вхідна точка до системи, яка реалізовує динамічні програмні інтерфейси, відповідає за ініціалізацію програмного забезпечення при запуску сервера – підключення та перевірку стану баз даних, а також можливі дії при зупинці сервера. Окрім цього, здійснює первинну обробку запитів, визначає проєкт, в контексті якого здійснюється запит, використовуючи піддомен, відправляє об’єкт запиту на подальшу обробку і повертає відповідь користувачам.

Дана частина програмної системи є найбільш навантажуваною та потребує ефективної роботи, вона відділена від інтерфейсу розробників програмних інтерфейсів в окремий сервіс задля можливості масштабування. Реалізація системи без використання окремих бібліотек для обробки запитів на рівні Python-застосунку дозволяє зробити цей процес більш ефективним, без використання зайвої логіки [Додаток К].

Для розробки використовується асинхронний підхід. Це дозволяє оптимізувати роботу з мережевими операціями, забезпечити високу продуктивність і швидкодію системи, оптимальне використання ресурсів та задоволення вимог користувачів. Застосування асинхронного підходу в

Python здійснюється за допомогою вбудованої бібліотеки asyncio, яка надає зручні механізми для виконання асинхронних операцій.

Система динамічних програмних інтерфейсів підтримується NoSQL базою даних MongoDB для запису та отримання даних кінцевих користувачів. Ізоляція даних між проєктами забезпечується створенням окремих колекцій на рівні бази даних.

Сервіс динамічних програмних інтерфейсів включає отримання, обробку запитів, запис даних в базу даних та повернення відповідей. Формат програмних інтерфейсів – запитів чи відповідей, обсягів даних – структури таблиць бази даних, та обробників даних визначається через інтерфейс розробників та може змінюватися без прямого втручання у роботу сервера.

4.4. Імплементація системи автоматичної документації динамічних програмних інтерфейсів

Для забезпечення зручності взаємодії кінцевих користувачів з програмними інтерфейсами передбачається автоматично згенерована документація, яка надає детальний опис щодо використання API. Вона відображає зміни, які відбуваються у метаданих в реальному часі.

Документація для API є невід'ємною складовою розробки програмного забезпечення, посібником для розробників, який надає інформацію про доступні інтерфейси, параметри запитів, структуру даних та інші важливі деталі, необхідні для успішної інтеграції з API. Окрім цього, документація є важливим елементом для спілкування з іншими розробниками, клієнтами та командою підтримки.

Сервіс динамічної документації є FastAPI Python сервісом, який на основі метаданих з бази даних генерує специфікацію OpenAPI у форматі JSON. Згенерований об’єкт включає такі дані як тіло запиту з вказаним типом вмісту та списком полів, а також можливі відповіді з тілом відповіді

та кодами стану. Окрім цього, специфікація документації візуалізовується у зручний користувацький інтерфейс.

4.5. Імплементація обробників даних

Робота обробників даних частково інтегрована в сервіс динамічних програмних інтерфейсів, обробники даних викликаються на кожен запит та відповідь кінцевих користувачів, якщо потреба у них визначена користувачами-розробниками.

Обробка даних на основі користувацького коду здійснюється в окремих ізольованих віртуальних машинах. Ізоляція забезпечується в межах одного запиту конкретного користувача, віртуальні машини не використовуються повторно для обробки інших запитів, а вже застосовані середовища підлягають видаленню.

Окрім звичайної обробки даних також передбачається повернення помилок з використанням спеціальної Python-помилки, доступної розробникам, де вказується HTTP-статус та детальний текст помилки для відображення кінцевим користувачам.

Віртуальні машини запускаються з використанням Firecracker та Ignite, що забезпечує ефективність їх запуску, роботи та видалення, а взаємодія з ними схожа на роботу зі звичайними контейнерами.

Для віртуальних машин створено окрему файлову систему, яка містить усі необхідні бібліотеки, включаючи Python. Окрім цього, на рівні файлової системи забезпечується запуск серверу при кожному запуску віртуальної машини. Сервер відповідає за роботу з HTTP-запитами з інших сервісів та здійснює обробку даних відповідно до переданих інструкцій.

Для забезпечення максимальної швидкодії запитів віртуальні машини створюються раніше, для цього розроблено окремий сервіс, який постійно перевіряє кількість доступних віртуальних машин та створює нові

за потреби, завжди підтримуючи визначену кількість активних віртуальних машин. Сервіс записує IP-адреси доступних віртуальних машин у чергу в базі даних Redis.

Сервіс динамічних програмних інтерфейсів не має тісної прив'язки до реалізації обробників даних з використанням Firecracker та може використовувати будь-які середовища, де доступний сервер, який може обробляти інструкції.

4.6. Висновок до розділу 4

Отже, на основі розроблених раніше UML-діаграм, а також моделей баз даних було реалізовано функціональні елементи, які забезпечують роботу ряду сервісів, включаючи систему розробки програмних інтерфейсів із користувацьким інтерфейсом, які разом забезпечують зручну можливість користувачам-розробникам створювати та редагувати метадані динамічних компонентів, з можливістю взаємодії з іншими користувачами.

Окрім цього, реалізовано сервіс для обробки запитів кінцевих користувачів, який, використовуючи наявні метадані, дозволяє кінцевим користувачам взаємодіяти з обсягами даних через програмні інтерфейси та обробники даних. Крім того, для забезпечення зручності використання користувачами програмних інтерфейсів, була розроблена автоматично генерована документація.

Для роботи обробників даних було реалізовано ряд окремих сервісів, які відповідають за запуск віртуальних машин для обробки даних за інструкціями, підтримку визначеної кількості віртуальних машин активними.

Усі ці технології та рішення спільно працюють, щоб забезпечити зручну, ефективну та надійну розробку програмних інтерфейсів.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Однією з основ функціонування сучасних програмних систем є взаємодія між їх внутрішніми компонентами, а також взаємодія систем одна з одною. Фундаментальним підходом до вирішення цієї проблеми є клієнт-серверна взаємодія.

Розроблена система розробки та впровадження програмних інтерфейсів виводить таку взаємодію на новий рівень, коли для реалізації серверної частини більше не вимагаються значні знання у програмуванні чи інших сферах. Маючи бізнес-вимоги до системи, будь-який користувач може реалізувати програмні інтерфейси, які можуть використовуватися у взаємодії з іншими сервісами, користувацьким інтерфейсом або напряму. Окрім цього, програмне забезпечення може бути корисним для зручного тестування взаємодії систем в процесі розробки, як сервер-”макет”.

У ході виконання проєкту було проаналізовано велику кількість теоретичних відомостей про взаємодію програмних систем, клієнт-серверну взаємодію зокрема. Розглянуто мережеву взаємодію, протокол HTTP та REST як архітектурний стиль взаємодії систем. Проаналізовані теоретичні відомості стали фундаментом для побудови детальної архітектури системи та бази даних, на основі яких було реалізовано ряд сервісів, які разом складають сервіс для розробки та впровадження програмних інтерфейсів.

Розроблене програмне забезпечення передбачає широкий спектр потенційних розширень та удосконалень. Система, будучи динамічною та гнучкою, може ефективно інтегрувати нові функції та властивості, що зможуть значно підвищити її продуктивність, надійність або корисність для

кінцевого користувача. Завдяки архітектурному підходу та технологічним рішенням, що були використані під час розробки, можливість такого розширення не тільки існує, але й передбачається як інтегральний елемент стратегії продукту. Серед потенційних розширень та удосконалень, які можуть бути впроваджені в майбутньому:

- можливість об’єднувати вже існуючу зовнішню систему із розробленим сервісом шляхом інтеграції баз даних – це дозволило б поширити вже існуючий у зовнішній системі набір користувачів та інших сутностей до сервісу та навпаки;
- можливість обмежувати доступ до програмних інтерфейсів нативно, без використання обробників даних, що дозволило б зменшити навантаження на систему та збільшити швидкість обробки запитів;
- створення публічної бібліотеки обробників даних з можливістю їх перевикористання різними проєктами, що дозволило б користувачам використовувати вже готові рішення, розроблені іншими;
- розширення можливостей обсягів даних, наближення їх до звичайних таблиць баз даних з можливістю визначати обмеження унікальності, обов’язковості даних, та зв’язки між обсягами даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fielding R. Architectural Styles and the Design of Network-based Software Architectures : дис. докт. філос. наук / Fielding Roy Thomas – Ірвайн, 2000. – 162 с.
2. Fielding R. HTTP Semantics [Електронний ресурс] / R. Fielding, M. Nottingham, J. Reschke. – 2022. – Режим доступу до ресурсу: https://www.rfc-editor.org/rfc/rfc9110.html.
3. Use containers to Build, Share and Run your applications [Електронний ресурс] – Режим доступу до ресурсу: https://www.docker.com/resources/what-container/.
4. Scarfone K. Guide to Security for Full Virtualization Technologies / K. Scarfone, M. Souppaya, P. Hoffman. – Гейтерсбург: National Institute of Standards and Technology, 2011. – 35 с.
5. What is Amazon API Gateway? [Електронний ресурс] – Режим доступу д ресурсу: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html.
6. UML 2 Toolkit / H.Eriksson, M. Penker, B. Lyons, D. Fado. – Індіанаполіс: Wiley Publishing, 2004. – 515 с.
7. Vase T. Advantages of Docker [Електронний ресурс] / Tuomas Vase. – 2015. – Режим доступу до ресурсу: https://jyx.jyu.fi/bitstream/handle/123456789/48029/URN%3aNBN%3afi%3ajyu-201512093942.pdf.
8. Secure and fast microVMs for serverless computing [Електронний ресурс] – Режим доступу до ресурсу: https://firecracker-microvm.github.io/.

					ДТЕУ 121 02-1.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		01.11.23	*Програмний комплекс для розробки API*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Котенко Н.О.		01.11.23		*СВД*	*50*	*50*
Гарант		Котенко Н.О.		01.11.23		Факультет інформаційних технологій 2м курс, 2 група		
Розробив		Александров А. Ю.		01.11.23	*Список використаних джерел*			

ТЕХНІЧНЕ ЗАВДАННЯ

1. Мета та призначення системи:
    a. Створення онлайн-сервісу, який дозволить користувачам легко проєктувати та розробляти API без потреби у створенні та підтримці власної серверної інфраструктури.
2. Основні функції:
    a. Створення програмних інтерфейсів:
        - можливість налаштування базових характеристик програмних інтерфейсів: шлях доступу, таблицю в базі даних, тип доступу, список полів, обмеження на поля;
        - автоматична генерація документації для кожного програмного інтерфейсу;
        - можливість активації та деактивації програмних інтерфейсів.
    b. Робота з базою даних:
        - створення та керування таблицями, можливість налаштування наступних характеристик: назви таблиці, списку полів з типами даних.
    c. Розширені можливості програмних інтерфейсів:
        - використання Python для створення обробників даних, які будуть виконуватися при запитах та відповідях.
3. Технологічні вимоги:
    a. Контейнеризація – використання контейнерів для окремих компонентів системи.

					ДТЕУ 121 02-1.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		06.09.23	*Програмний комплекс для розробки API*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Котенко Н.О.		06.09.23		*ТЗ*	*51*	*50*
Гарант		Котенко Н.О.		06.09.23		*Факультет інформаційних технологій 2м курс, 2 група*		
Розробив		Александров А. Ю.		06.09.23	*Технічне завдання*			

ДОДАТКИ

Додаток А

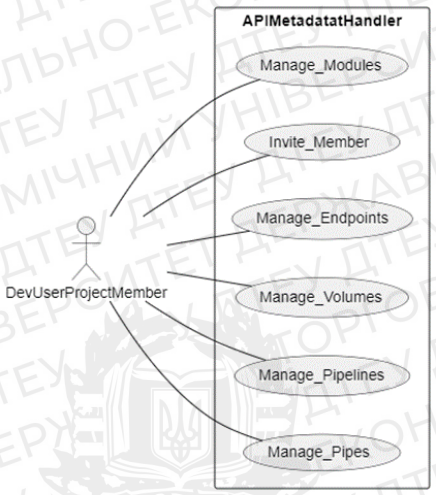


Рис. А.1. Діаграма сценаріїв використання користувача-розробника

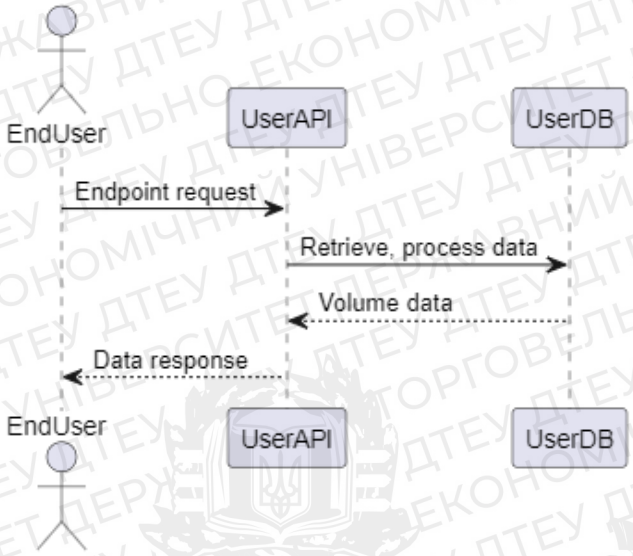


Рис. Б.1. Діаграма послідовності використання програмного інтерфейсу

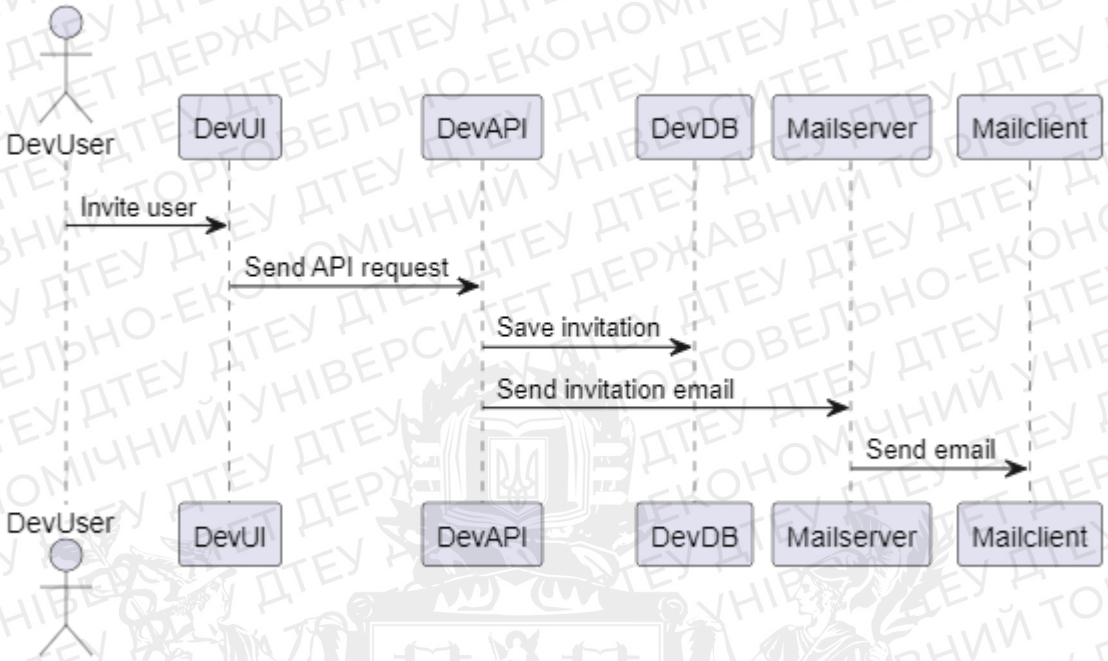


Рис. В.1. Діаграма запрошення нового користувача до проєкту

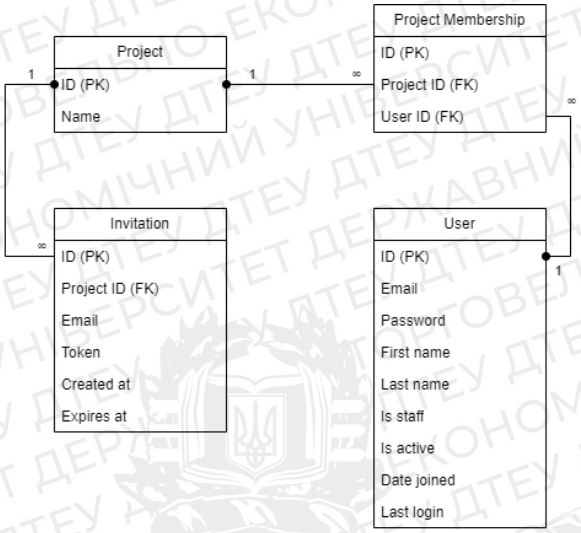


Рис. Г.1. Логічна модель даних взаємодії проєктів і користувачів

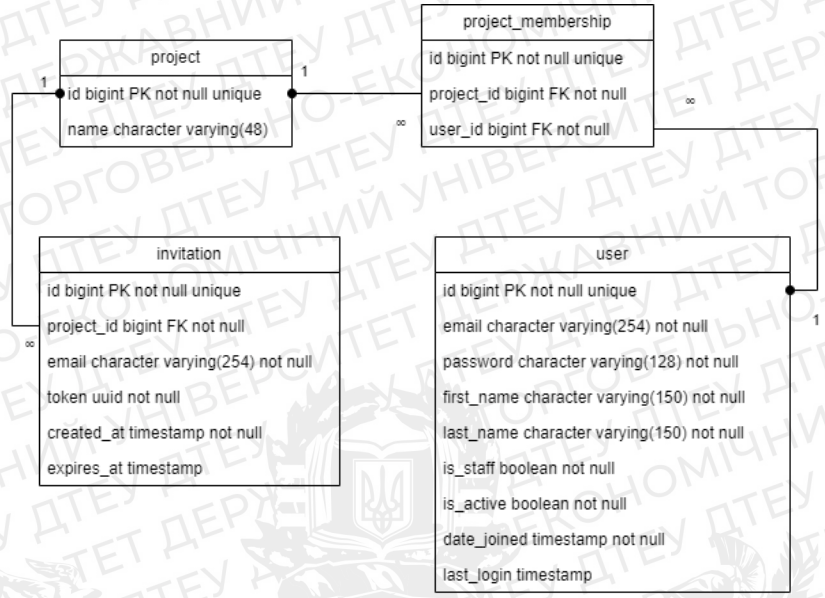
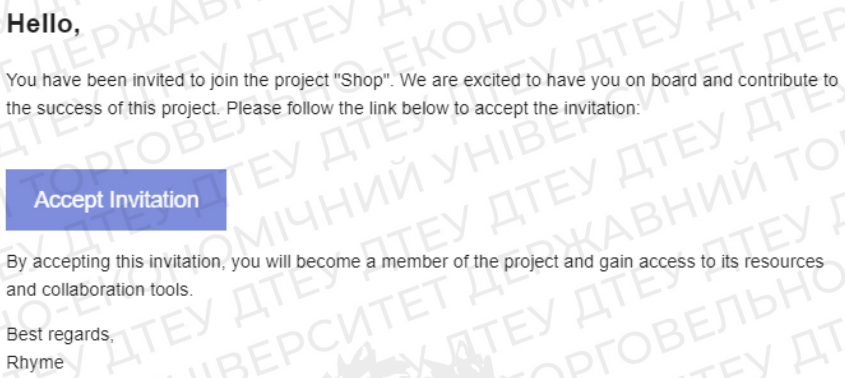


Рис. Д.1. Фізична модель даних взаємодії проєктів і користувачів



Hello,

You have been invited to join the project "Shop". We are excited to have you on board and contribute to the success of this project. Please follow the link below to accept the invitation:

Accept Invitation

By accepting this invitation, you will become a member of the project and gain access to its resources and collaboration tools.

Best regards,
Rhyme

Рис. Е.1. Наповнення електронного листа із запрошенням до проєкту

Лістинг програмного коду застосування запрошення до проєкту

```
@transaction.atomic
def accept_invitation_to_project(token: str, user: User) -> None:
    try:
        invitation = Invitation.objects.exclude(
            expires_at__lt=datetime.now(timezone.utc)
        ).get(token=token, email=user.email)
    except Invitation.DoesNotExist:
        raise ProjectInvitationInvalidError
    invitation.project.members.add(user)
    invitation.delete()
```

Рис. И.1. Сторінка створення та списку проєктів

Додаток К

Лістинг програмного коду основного методу ASGI-додатку

```
async def __call__(self, scope, receive, send) -> None:
    response: list[dict] | dict[str, str] | None
    if scope["type"] == "lifespan":
        response = await self._resolve_lifespan(await receive())
    elif scope["type"] == "http":
        response = await self._resolve_http(await receive(), scope)
    else:
        raise ValueError("Unknown request type")
    if isinstance(response, list):
        for _response in response:
            await send(_response)
    else:
        await send(response)
```