МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
____ _____ 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 – Комп'ютерних наук,

освітньо-професійної програми «Інформатика»

на тему: Інформаційна технологія оптимізації веб-орієнтованих інформаційних систем

здобувача групи ІН.мз-21с Хоруженка Ярослава Віталійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Ярослав ХОРУЖЕНКО
_____
(підпис)

Керівник,
старший викладач,
кандидат технічних наук
Олег БЕРЕСТ

_____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук, освітньо-наукової програми «Інформатика»
здобувача групи ІН.мз-21с Хоруженка Ярослава Віталійовича

1. Тема роботи: «Інформаційна технологія оптимізації веб-орієнтованих інформаційних систем»
затверджую наказом по СумДУ від *«20» листопада 2023 р. наказ №1308-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до 14 грудня 2023 року*

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Дослідження підходів та технологій, що використовуються для розробки веб-орієнтованих інформаційних систем. 3) Розробка веб-орієнтованої інформаційної системи з урахуванням результатів проведених досліджень. 4) Аналіз результатів.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	Аналіз проблеми предметної області, постановка й формування завдань дослідження	17.10.23-20.10.23	
2	Огляд веб-орієнтованих інформаційних систем	23.10.23-25.10.23	
3	Аналіз існуючих мов програмування, що використовуються для розробки інформаційних систем	26.10.23-27.10.23	
4	Загальний огляд фреймворків, що використовуються для розробки інформаційних систем	30.10.23-01.11.23	
5	Порівняльний аналіз популярних фреймворків, що використовуються для розробки інформаційних систем	02.11.23-03.11.23	
6	Дослідження та розробка інформаційної моделі інформаційної системи	06.11.23-07.11.23	
7	Аналіз підходящих засобів програмної реалізації інформаційних систем, їх вибір та впровадження	08.11.23-10.11.23	
8	Опис програмної реалізації інформаційних систем	13.11.23-17.11.23	
9	Опис результатів тестування інформаційної системи	20.11.23-24.11.23	
10	Розробка та опис інструкції користувача інформаційної системи	27.11.23-29.11.23	
11	Оформлення пояснювальної записки до кваліфікаційної роботи	30.11.23-01.12.23	

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 88 сторінок, 2 таблиці, 25 рисунків, 62 літературних джерела, 2 додатки.

Об’єкт дослідження — процес проектування та реалізації веб-орієнтованих інформаційних систем, а також їх оптимізації з використанням сучасних фреймворків та інших інструментів розробки.

Мета роботи — аналіз сучасних підходів до розробки та оптимізації веб-орієнтованих інформаційних систем, а також демонстрація їх на конкретному прикладі.

Результати — проведено загальний огляд веб-орієнтованих інформаційних систем, та прийнято рішення про реалізацію бекенд веб-застосунку. Здійснено аналіз мов програмування, а також популярних фреймворків, що використовуються для розробки веб-орієнтованих інформаційних систем. Досліджено та розроблено інформаційну модель інформаційної системи та на її підставі підібрано підходящі засоби програмної реалізації. Розроблено інформаційну систему з урахуванням обраних засобів програмної реалізації, а також здійснено її опис. Також описано результати тестування інформаційної системи. Розроблено та описано інструкцію користувача інформаційної системи.

ІНФОРМАЦІЙНА СИСТЕМА, СУЧАСНІ ФРЕЙМВОРКИ, PYTHON, FASTAPI, ВЕБ-СЕРВІС, ВЕБ-ЗАСТОСУНОК, БЛОГ

ЗМІСТ

ВСТУП

Обґрунтування вибору теми роботи та її актуальність. У сучасному суспільстві важко уявити функціонування будь-якого бізнесу чи організації без належної інформаційної підтримки. Веб-орієнтовані інформаційні системи виступають ключовим елементом цього процесу, забезпечуючи зручний та ефективний доступ до інформації.

Використання інформаційних веб-орієнтованих систем для бізнесу дозволяє компаніям оптимізувати процеси управління, забезпечує зручний та швидкий доступ до важливої інформації, сприяє впровадженню ефективних стратегій маркетингу та взаємодії з клієнтами через онлайн-платформи.

У сфері державного управління веб-орієнтовані інформаційні системи сприяють ефективній комунікації між різними відділами та рівнями уряду, полегшують доступ громадян до різних господарських та соціальних послуг через електронні платформи.

В науковій діяльності веб-орієнтовані інформаційні системи дозволяють вченим спільно працювати над проектами, обмінюватися результатами досліджень, отримувати доступ до великих обсягів наукової інформації та баз даних.

Таким чином розробка та оптимізація цих систем стає важливим завданням в умовах стрімкого розвитку інформаційних технологій практично в усіх сферах життя людини.

Предмет дослідження. Веб-орієнтовані інформаційні системи, які використовуються для організації та обробки інформації через мережу Інтернет.

Об’єкт дослідження. Процес проектування та реалізації веб-орієнтованих інформаційних систем, а також їх оптимізації з використанням сучасних фреймворків та інших інструментів розробки.

Мета дослідження. Аналіз сучасних підходів до розробки та оптимізації

веб-орієнтованих інформаційних систем, а також демонстрація їх на конкретному прикладі.

Основними завданнями кваліфікаційної роботи є:

- проведення загального огляду веб-орієнтованих інформаційних систем
- здійснення аналізу існуючих мов програмування, що використовуються для розробки інформаційних систем
- проведення загального огляду фреймворків, що використовуються для розробки інформаційних систем
- проведення порівняльного аналізу популярних фреймворків, що використовуються для розробки інформаційних систем
- дослідження та розробка інформаційної моделі інформаційної системи
- проведення аналізу підходящих засобів програмної реалізації інформаційних систем, їх вибір та впровадження
- здійснення опису програмної реалізації інформаційних систем
- опис результатів тестування інформаційної системи
- розробка та опис інструкції користувача інформаційної системи

Методи дослідження. В рамках роботи використані: абстрагування, аналіз, порівняння, формалізація, синтез, експеримент та вимірювання.

Структура. Робота структурована відповідно до основних завдань.

Практичне значення. Застосування результатів досліджень, здійснених в межах цієї роботи, дозволить зробити процес розробки веб-орієнтованих інформаційних систем більш легким та швидким, а самі системи більш надійними, ефективними та продуктивними.

1 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЧНОГО СТАНУ У РОЗРОБЦІ ВЕБ-ОРІЄНТОВАНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Загальний огляд веб-орієнтованих інформаційних систем

Існує безліч видів веб-орієнтованих інформаційних систем, тобто, систем, які базуються на веб-технологіях для обробки, зберігання та передачі інформації через Інтернет. Вони можуть бути класифіковані за різними критеріями. Серед основних видів веб-орієнтованих інформаційних систем можна виділити наступні [1-2]:

Динамічні веб-сайти. Ці веб-сайти взаємодіють із користувачами та базами даних, генеруючи контент на льоту в залежності від конкретного запиту користувача чи інших факторів. Динамічні веб-сайти часто використовують мови програмування на сервері, такі як PHP, JavaScript, Ruby, Java або Python.

Статичні веб-сайти. Це основний тип веб-сайтів, які відображають фіксований контент, інформацію та графіку. Вони зазвичай використовуються для представлення компаній, продуктів чи послуг, і їхній контент залишається без змін до тих пір, поки адміністратор сайту не внесе оновлення.

Інтернет-магазини. Вони дозволяють користувачам переглядати та придбавати товари або послуги онлайн. Інтернет-магазини можуть бути як статичними, так і динамічними, залежно від складності та функціональності.

Веб-застосунки. Сучасні веб-додатки надають користувачам широкий спектр функціональності, включаючи обробку даних у реальному часі, інтерактивність та спеціалізовані функції. Google Docs, Slack або Jira – це приклади веб-застосунків.

Портфоліо та особисті веб-сайти. Це веб-сайти, які використовуються для представлення особистого або професійного портфоліо. Вони часто включають інформацію про роботи, досягнення, проекти тощо.

Блоги та форуми. Блоги надають платформу для ведення онлайн-журналу,

а форуми дозволяють користувачам обговорювати різні теми та ділитися думками та інформацією.

Соціальні мережі. Застосунки цього типу дозволяють користувачам спілкуватися, обмінюватися інформацією та взаємодіяти онлайн. Наприклад: Facebook, X (ex Twitter), LinkedIn тощо.

Електронна пошта та календарі. Веб-застосунки для роботи з електронною поштою, календарями та завданнями. Приклади включають Google Calendar, Gmail, Outlook.com та інші.

Це лише кілька загальних категорій, і веб-орієнтовані інформаційні системи можуть поєднувати різні функції та характеристики. Різноманітність цих систем відображає різні потреби користувачів та бізнес-сфер.

У подальшому ця робота буде розглядати веб-орієнтовані інформаційні системи більшою мірою в розрізі веб-застосунків та веб-сайтів з наявною backend компонентою.

Веб-застосунок (веб-додаток) [3] – це програмне забезпечення, розроблене для використання через Інтернет і може бути використаний на різних пристроях та платформах. Веб-застосунки зазвичай використовуються для виконання різних завдань, від ведення електронної пошти та обміну повідомленнями до роботи з електронними таблицями та ведення блогів.

Основні характеристики веб-застосунків включають [4]:

Доступ через браузер. Користувачі можуть отримати доступ до веб-застосунку через веб-браузер, такий як Google Chrome, Mozilla Firefox, або Safari. Немає необхідності встановлювати додаткове програмне забезпечення на пристрої користувача.

Взаємодія через Інтернет. Веб-застосунок взаємодіє з користувачем та сервером через Інтернет. Зазвичай, весь обмін даними відбувається по протоколу HTTP або HTTPS.

Множина функцій. Веб-застосунки можуть виконувати різноманітні

завдання, включаючи роботу з базами даних, обробку форм, відправлення та отримання даних у режимі реального часу, інтеграцію з іншими службами тощо.

Мультиплатформенність. Оскільки веб-застосунок використовується через браузер, він може працювати на різних операційних системах, таких як Windows, macOS, Linux, а також на різних пристроях, включаючи комп'ютери, планшети та смартфони.

Постійні оновлення. Веб-застосунки можуть бути легко оновлюваними, оскільки оновлення відбуваються на сервері, і користувачам не потрібно завантажувати або встановлювати нові версії.

Розробка веб-застосунку це складний процес, який включає роботу над різними етапами та використанням різноманітних технологій. Ось ключові елементи, які можна виділити в розробці веб-застосунку:

Вимоги. Визначення функціональних та технічних вимог веб-застосунку. Це включає визначення функцій, які повинен виконувати застосунок, а також технічних обмежень та вимог до продуктивності.

Дизайн інтерфейсу користувача (UI/UX). Створення інтерфейсу користувача, який буде ефективним, зручним та привабливим для використання. Дизайн також повинен враховувати вимоги щодо доступності та взаємодії з користувачем.

Бекенд (Backend). Серверна частина застосунку, яка відповідає за обробку логіки бізнес-процесів, взаємодію з базою даних, обробку запитів від клієнтської частини і забезпечення безпеки даних.

Фронтенд (Frontend). Клієнтська частина застосунку, яка відповідає за відображення інтерфейсу користувача та взаємодію з користувачем. Зазвичай включає в себе HTML, CSS та JavaScript.

Тестування. Використання різних методів тестування, таких як функціональне, інтеграційне, та прийомне тестування, для забезпечення якості та відповідності вимогам.

Безпека. Забезпечення безпеки застосунку, включаючи захист від атак, обробку аутентифікації та авторизації, а також шифрування даних.

Розгортання та моніторинг. Розгортання веб-застосунку на сервері та налаштування середовища для експлуатації. Моніторинг роботи застосунку для виявлення проблем та оптимізації продуктивності.

Супровід та підтримка. Підтримка та оновлення веб-застосунку після його розгортання. Це включає в себе виправлення помилок, вдосконалення функціональності та реагування на зміни вимог чи технічного оточення.

Ці елементи взаємодіють між собою для створення повноцінного та працюючого веб-застосунку. Успішна розробка вимагає ретельного управління всіма цими аспектами та врахування вимог та потреб користувачів.

В межах цієї роботи я хотів би розглянути можливості розробки та оптимізації веб-орієнтованих інформаційних систем – бекенд веб-застосунків шляхом вибору та використання оптимальних фреймворків та інших інструментів розробки.

Імплементація backend веб-застосунку — це процес створення серверної частини програми, яка обробляє запити від клієнтської частини та відповідає на них. Більш детально цей процес буде висвітлено в другому та третьому розділах цієї роботи.

1.2 Аналіз існуючих мов програмування, що використовуються для розробки інформаційних систем

Перш за все розглянемо мови програмування, які використовуються для розробки веб-застосунків [5-6].

JavaScript [7-8].

До переваг цієї мови програмування можна віднести:

- вважається важливим інструментом для фронтенд розробки

- менші вимоги до сервера
- гнучкість у взаємодії з іншими мовами програмування та інструментами
- широкий спектр застосування, включаючи розробку ігор і мобільних додатків

До недоліків:

- гірша безпека на стороні клієнта
- статична антація типів
- браузери можуть по-різному інтерпретувати код
- іноді може виникати проблема з управлінням асинхронністю

Python [9-10].

Переваги:

- зручний та легко зрозумілий синтаксис.
- використовується для створення бекенд-розробки
- універсальність застосування
- велика кількість бібліотек та інших готових інструментів, що дозволяють швидко розробляти застосунки
- широке застосування у сфері ШІ

Недоліки:

- не використовується для запуску на стороні клієнта, оскільки жоден із основних веб-браузерів не має вбудованої підтримки запуску Python скриптів
- не підходить для розробки на мобільних пристроях
- споживає велику кількість пам’яті
- не сама швидка мова програмування, але це більше залежить від конкретної імплементації

Java [9].

Переваги:

- використовується для створення бекенду великих підприємств через свою надійність та захищеність
- незалежний від платформи

Недоліки:

- складність великих монолітних проектів
- читабельність коду може визивати труднощі
- через інтенсивність використання ресурсів може мати складнощі з масштабуванням

PHP [10].

Переваги:

- широко використовується для веб-розробки, зокрема для створення динамічних веб-сайтів
- легко інтегрується з базами даних
- має простий синтаксис
- має гарну сумісність з різними операційними системами

Недоліки:

- низький рівень безпеки
- використовує “слабку” анотацію типів, що може призвести до неправильного розуміння і даних для користувачів
- значна кількість зворотно-несумісних змін в ядро мови програмування

Ruby [11].

Переваги:

- використовується з фреймворком Ruby on Rails для швидкої

розробки веб-додатків

- синтаксис простий та елегантний
- активна спільнота розробників

Недоліки:

- порівняно з іншими мовами, швидкість може бути меншою
- не найкращий процес відладки помилок

Кожна мова програмування має свої переваги та недоліки, і вибір залежить від конкретних вимог та цілей проекту. Часто великі веб-проекти використовують комбінацію мов для фронтенду та бекенду.

1.3 Загальний огляд фреймворків, що використовуються для розробки інформаційних систем

Веб-застосунки розвиваються настільки швидко, що їхні можливості використання та інтерактивність конкурують з можливостями нативних додатків. Технологія та експертиза, необхідні для створення індивідуальних рішень, які досягають цього рівня високої ефективності, є вимогливими. На щастя, існують інструменти, які полегшують розробку веб-застосунків, одним із яких є фреймворк для веб-застосунків.

Правила та архітектура серверних фреймворків дозволяють створювати прості сторінки, лендінги та форми різних видів. Для створення веб-застосунку із добре розробленим інтерфейсом вам потрібен ширший спектр функціональності. Серверні фреймворки обробляють HTTP-запити, управління та контроль бази даних, мапінг URL та інше. Ці фреймворки можуть покращити безпеку та формувати вихідні дані, спрощуючи процес розробки.

Таким чином, не буде перебільшенням зазначити, що в цей час вибір фреймворку (скоріш за все їх буде декілька) для реалізації веб-орієнтованої системи буде одночасно залежати та значною мірою визначати і архітектуру, і

набір технологій, і спосіб ініціалізації API (як для доступу до веб ресурсів серверу, так і для комунікації з базою даних), і навіть структуру проекту [12].

Веб-фреймворк – це набір вже написаного коду та структури, який допомагає розробникам виводити веб-застосунки швидше та ефективніше. Він забезпечує стандартну організацію коду, управління HTTP-запитами та відповідями, роботу з базою даних, шаблони для створення інтерфейсів користувача та інші корисні інструменти.

Фреймворки надають готовий набір інструментів та бібліотек, які значно спрощують розробку backend. Програмісти можуть використовувати готові рішення для таких завдань, як обробка маршрутів, взаємодія з базою даних, обробка запитів тощо, що пришвидшує процес розробки.

Також фреймворки зазвичай встановлюють конвенції та структуру проекту, що робить код більш організованим. Це полегшує розуміння та спільну роботу для команди розробників.

Багато веб-фреймворків мають вбудовані механізми для обробки безпеки, такі як обробка аутентифікації та авторизації, захист від атак, шифрування тощо. Це сприяє створенню безпечних backend-рішень.

Фреймворки сприяють стандартизації коду та структурі проекту, що дозволяє забезпечувати консистентність між різними частинами проекту. Це полегшує та прискорює розвиток та підтримку веб-застосунків.

Велика частина веб-фреймворків створена з урахуванням можливості розширення та масштабування. Вони можуть легко інтегруватися з новими функціональностями та масштабуватися для відповіді на зростаючі потреби в трафіку та обробці даних.

Популярні веб-фреймворки часто мають активні спільноти розробників і хорошу документацію. Це робить розробку більш ефективною і дозволяє швидше знаходити відповіді на технічні питання.

Таким чином, можна дійти до висновку, що розробка безпечного,

масштабованого та легкого у підтримці веб-застосунку без використання фреймворку скоріш за все буде марною тратою часу команди розробників, тому вибір підходящого під мету веб-застосунку фреймворку відіграє критичну роль [13].

Існує безліч веб-фреймворків, які можна класифікувати за такими критеріями [14]:

Мова програмування. Фреймворки можуть бути специфічними для певних мов програмування. Наприклад, Django, Flask та FastAPI використовуються для розробки веб-додатків на мові програмування Python, Ruby on Rails – на Ruby, Laravel – на PHP, Express.js – на JavaScript (Node.js).

Тип архітектури. Фреймворки можуть підтримувати різні архітектурні підходи, такі як MVC (Model-View-Controller), MVVM (Model-View-ViewModel), або інші. Наприклад, Angular та Ember.js – фреймворки з архітектурою MVVM, Django та Ruby on Rails – з архітектурою MVC [15].

Розташування коду. Є фреймворки, які дозволяють розробникам розміщати код в різних місцях. Деякі використовують підходи типу "батарейки включено" (batteries-included), де вони надають велику кількість вбудованих функцій, як приклади Django. Інші фреймворки, такі як Flask або Express.js, спрямовані на простоту та гнучкість, дозволяючи розробникам вибирати та додавати компоненти за потребою.

Вертикальні або горизонтальні. Вертикальні фреймворки фокусуються на розв'язанні конкретних завдань (наприклад, фреймворк для розробки блогу). Горизонтальні фреймворки, навпаки, призначені для розробки різноманітних видів веб-застосунків.

Frontend та backend фреймворки. Щодо backend-фреймворків, такі як наприклад Spring, Ruby on Rails чи FastAPI, це інструменти розробки, орієнтовані на створення та управління серверною частиною веб-застосунку. Ці фреймворки допомагають розробникам створювати логіку обробки даних,

взаємодію з базами даних, аутентифікацію, та інші серверні функції. Фреймворки для розробки клієнтської сторони (frontend), такі як наприклад, React, Angular, та Vue.js використовуються для створення інтерфейсу користувача та взаємодії з ним .

1.4 Порівняльний аналіз фреймворків, що використовуються для розробки інформаційних систем

Розглянемо більш детально популярні фреймворки для різних мов програмування у таблиці 1.1.

Таблиця 1.1 Порівняння фреймворків для різних мов програмування

JavaScript [16]		
Назва фреймворку	Тип	Особливості
React [21]	Бібліотека для розробки інтерфейсу користувача (UI)	Компонентний підхід. Центральним елементом React є компоненти, які спрощують структуру та управління UI.
		Віртуальний DOM. React використовує віртуальний DOM для оптимізації оновлень та покращення продуктивності.
		Гнучкість. Можливо використовувати разом з різними бібліотеками та фреймворками.

Angular [22]	Фреймворк для розробки frontend-додатків	Модульність. Angular пропонує модульну архітектуру для легшого управління та підтримки проектів.
		Двостороннє зв'язування. Забезпечує двостороннє зв'язування даних між моделлю та представленням.
		Система інструментів. Включає в себе широкий спектр інструментів для розробки, тестування та відлагодження.
Vue.js [23]	Прогресивний фреймворк для створення інтерфейсів	Легкість використання. Vue.js добре вписується в проекти будь-якої складності, і вивчення його є простим.
		Екосистема. Хоча Vue.js є меншим за розміром, він має добре організовану та розширювану екосистему.
		Реактивність. Вбудована система реактивності дозволяє легко відслідковувати та обробляти зміни в даних.
Java [16]		

Spring [24]	Фреймворк для розробки backend-додатків (інші модулі Spring також можуть використовуватися для frontend)	Широкий функціонал. Spring надає багатий функціонал для ведення бізнес-логіки, управління транзакціями, інтеграції з базами даних і іншими сервісами.
		Інверсія управління (IoC). Використовує принцип інверсії управління для полегшення взаємодії з об'єктами та управління залежностями.
		Spring Boot. Спрощує конфігурацію та розгортання додатків, що робить його ідеальним для швидкої розробки.
Ruby [18]		
Ruby on Rails [25]	Фреймворк для розробки повноцінних веб-додатків (backend)	Конвенція перед конфігурацією (Convention over Configuration). Використовує стандартизовані конвенції для автоматизації задач і полегшення розробки.
		Модель-Вигляд-Контролер (MVC) архітектура. Забезпечує відокремлення бізнес-логіки від представлення та обробки запитів.

		Широкий функціонал. Має вбудовані інструменти для роботи з базою даних, автентифікацією, тестуванням та іншими.
PHP [19]		
Laravel [26]	Фреймворк для розробки повноцінних веб-додатків (backend)	Eloquent ORM. Для роботи з базою даних, яка пропонує абстракцію бази даних та вирази для її запитів.
		Blade шаблони. Швидкі та прості шаблони для роботи з виглядом.
		Artisan Console. Консольний інструмент для автоматизації завдань та генерації коду.
Symfony [27]	Фреймворк для розробки повноцінних веб-додатків (backend)	Компонентна архітектура. Symfony побудований як набір відокремлених компонентів, що дозволяє використовувати їх незалежно.
		Doctrine ORM. Використовує абстракцію бази даних для спрощення взаємодії з базою даних.
		Twig шаблони. Шаблонний движок, який пропонує чистий та ефективний синтаксис.

Python [20]		
Django [28]	Фреймворк для повноцінних веб-додатків (backend)	ORM (Object-Relational Mapping). Використовує Django ORM для спрощення взаємодії з базою даних.
		Адміністративний панель. Надає вбудовану адміністративну панель для швидкого управління даними.
		Комплексний підхід. Включає в себе багато вбудованих функціональних можливостей, таких як аутентифікація, авторизація, форми та інше.
Flask [29]	Мінімалістичний фреймворк для швидкої розробки веб-додатків (backend)	Легкість використання. Flask призначений для простоти та гнучкості, надаючи базовий набір функцій.
		Розширюваність. Можливість розширення за допомогою різних розширень (extensions) згідно з потребами проекту.
		Необов'язкові компоненти. Дозволяє розробникам вибирати компоненти, які їм потрібні для конкретного застосунку.

FastAPI [30]	Сучасний фреймворк для створення високошвидкісних веб-додатків (backend) з підтримкою стандарту OpenAPI.	Автоматична документація: Генерує автоматичну документацію API на основі коду за допомогою стандарту OpenAPI.
		Асинхронність: Підтримка асинхронних запитів для оптимізації продуктивності.
		Валідація даних: Вбудована система валідації даних для безпеки та зручності.

Таким чином, можна зазначити, що кожен з перерахованих вище популярних фреймворків для веб-розробки має як свої плюси, так і мінуси, а також специфічну сферу застосування. Тобто вибір фреймворку буде залежати від наступних факторів:

- мова програмування
- тип застосунку
- вимоги щодо розміру та складності проекту
- наявності достатньої кількості документації та активної спільноти
- очікуваній швидкості та продуктивності
- вимог щодо безпеки продукту
- легкості інтеграції з іншими інструментами розробки
- наявності ліцензування

2 ПРОЕКТУВАННЯ ВЕБ-ОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Інформаційна модель інформаційної системи та її основні елементи

Інформаційна модель допомагає розробникам та аналітикам розуміти структуру та логіку веб-застосунку, що полегшує роботу з його розробкою, підтримкою та розширенням.

Інформаційна модель веб-застосунку – це абстракція, яка визначає, як дані обмінюються та обробляються в межах застосунку. Ця модель включає в себе структуру даних, їх типи, взаємодію компонентів системи та потоки даних.

Основні елементи інформаційної моделі веб-застосунку [31-32]:

Логіка бізнес-процесів. Опис та моделювання бізнес-логіки та процесів, які виконуються в межах веб-застосунку. Архітектура серверної частини та взаємодія між компонентами. Опис того, як серверна частина застосунку обробляє запити від клієнта, включаючи обробку HTTP-запитів, маршрутизацію, логіку бізнес-процесів та взаємодію з базою даних. А також, як компоненти веб-застосунку взаємодіють між собою. Це може включати взаємодію між серверною і клієнтською частинами, роботу з API, передачу та обробку даних.

Взаємодія з базою даних. Визначення, як дані зберігаються та отримуються з бази даних. Це може включати моделі даних, схему бази даних, запити до бази даних тощо.

Системні дані. Внутрішні дані, які обробляються та зберігаються в межах веб-застосунку. Це може включати дані про товари, послуги, налаштування системи тощо.

Забезпечення безпеки. Опис заходів забезпечення безпеки для захисту інформації та запобігання несанкціонованому доступу.

Користувацькі дані. Інформація, що вводиться та отримується від користувачів. Це може бути реєстраційна інформація, дані профілю, запити

користувачів тощо.

Відображення даних для користувача. Визначення того, як дані представляються та відображаються для користувача, включаючи дизайн інтерфейсу користувача.

2.2 Моделювання інформаційної системи

Спираючись на вищеперелічені елементи інформаційної моделі наш бекенд веб-застосунок може бути описаний наступним чином.

Логіка бізнес-процесів. На мою думку та виходячи з особистого досвіду цей пункт є найкритичнішим і таким, що має визначати всі інші елементи інформаційної моделі, або принаймні суттєво впливати на них. Саме тому останніми роками стає дуже поширеним Domain-Driven Design, або DDD — підхід до розробки програмного забезпечення, який реалізує конкретну модель предметної області та вирішує конкретну задачу бізнесу [33].

Перш за все визначимо який саме продукт ми хочемо створити. Це буде простий блог, а точніше його бекенд (серверна) частина. З точки зору бізнесу блог є продуктом, який надає платформу для публікації та обговорення контенту в Інтернеті. Безпосередньо логіка бізнес-процесів може бути описана (у спрощеному вигляді) у таблиці 2.1.

Таблиця 2.1 Опис логіки бізнес-процесів

Користувачі	Реєстрація	Користувач може створити обліковий запис на блозі, вводячи особисті дані
		Система перевіряє унікальність імені користувача та наявність обов'язкових полів

		Після успішної реєстрації користувачеві присвоюється унікальний ідентифікатор (ID)
	Аутентифікація	Зареєстровані користувачі можуть увійти до системи, використовуючи своє ім’я користувача та пароль
		Система перевіряє введені дані та, у випадку успіху, надає користувачеві токен для подальших запитів
Блоги	Створення блогів	Зареєстровані користувачі можуть створювати нові блоги, вказуючи заголовок та текст
		Система зберігає дані блога в базі даних, призначаючи йому унікальний ідентифікатор та пов'язуючи його із створеним користувачем
	Перегляд блогів	Користувачі можуть переглядати всі блоги або фільтрувати їх за унікальним ідентифікатором
	Редагування блогів	Користувачі можуть оновлювати власні блоги (заголовок та текст) фільтруючи їх за унікальним ідентифікатором
	Видалення	Користувачі можуть видаляти власні

	блогів	блоги фільтруючи їх за унікальним ідентифікатором
Коментарі	Додавання коментарів	Зареєстровані користувачі можуть додавати коментарі до існуючих блогів
		Система зберігає коментар у базі даних, пов'язуючи його із конкретним блогом та користувачем
	Відображення коментарів	Коментарі відображаються під відповідним блогом
Адміністративні функції	Управління користувачами	Адміністратори можуть переглядати та редагувати інформацію про користувачів, а також видаляти облікові записи.

Архітектура серверної частини та взаємодія між компонентами [34].

Застосунок буде побудований відповідно до принципів REST архітектури (Representational State Transfer, передача репрезентативного стану). Цей підхід має наступні елементи:

- Ресурси (Resources). Ресурси представляють об'єкти або послуги, які можуть бути ідентифіковані за унікальними URI (Uniform Resource Identifiers). Ресурси є центральним поняттям в архітектурі REST.
- Представлення (Representation). Ресурси можуть мати різні представлення. Наприклад, ресурс користувача може бути представлений у форматі JSON або XML. Клієнт може взаємодіяти з ресурсами, обмінюючи їхні представлення.
- Стан Ресурсів (State of Resources). REST передбачає, що стан

системи знаходиться в ресурсах і передається між клієнтом і сервером під час взаємодії [35]. Клієнт може переходити від одного стану до іншого, взаємодіючи з ресурсами.

- Уніфікований Інтерфейс (Uniform Interface). Уніфікований інтерфейс є ключовим принципом REST. Він визначає стандартний набір обміну повідомленнями, який включає HTTP методи (GET, POST, PUT, DELETE), URI для ідентифікації ресурсів, представлення для обміну даними та гіпермедіа як гіпертекстовий засіб для взаємодії.
- Stateless (Безстанційний принцип). Кожен запит від клієнта до сервера повинен містити всю необхідну інформацію для розуміння та виконання запиту. Сервер не повинен зберігати стан клієнта між запитами. Це полегшує масштабування та забезпечує простоту взаємодії.
- Кешування (Caching). REST підтримує можливість кешування, що дозволяє клієнту зберігати копії ресурсів та використовувати їх без повторного запиту до сервера. Це може покращити ефективність та знизити навантаження на сервер.
- Код на Вимогу (Code on Demand). Цей принцип не є обов'язковим для всіх RESTful систем, але передбачає, що сервер може відправити клієнту виконуваний код, наприклад, JavaScript. Це дозволяє розширювати функціональність клієнта без зміни самого клієнта.

Враховуючи, що веб-застосунок – блог буде мати виключно бекенд складову, то його дизайн має дещо спрощений вигляд у порівнянні із аналогічними застосунками, що мають також інтерфейс користувача.

Таким чином, система буде фактично розподілена на Модель (Model) та Контролер (Controller) [36].

Взаємодія між користувачем, а також адміністратором блогу, та самим застосунком буде відбуватися через контролер, який відповідатиме за обробку HTTP-запитів та взаємодію з моделлю та логікою додатку.

Модель представлятиме об'єкти (сутності), які зберігатимуться в базі даних.

Графічне зображення архітектури застосунку та взаємодії між його компонентами виглядає наступним чином (рис 2.1).

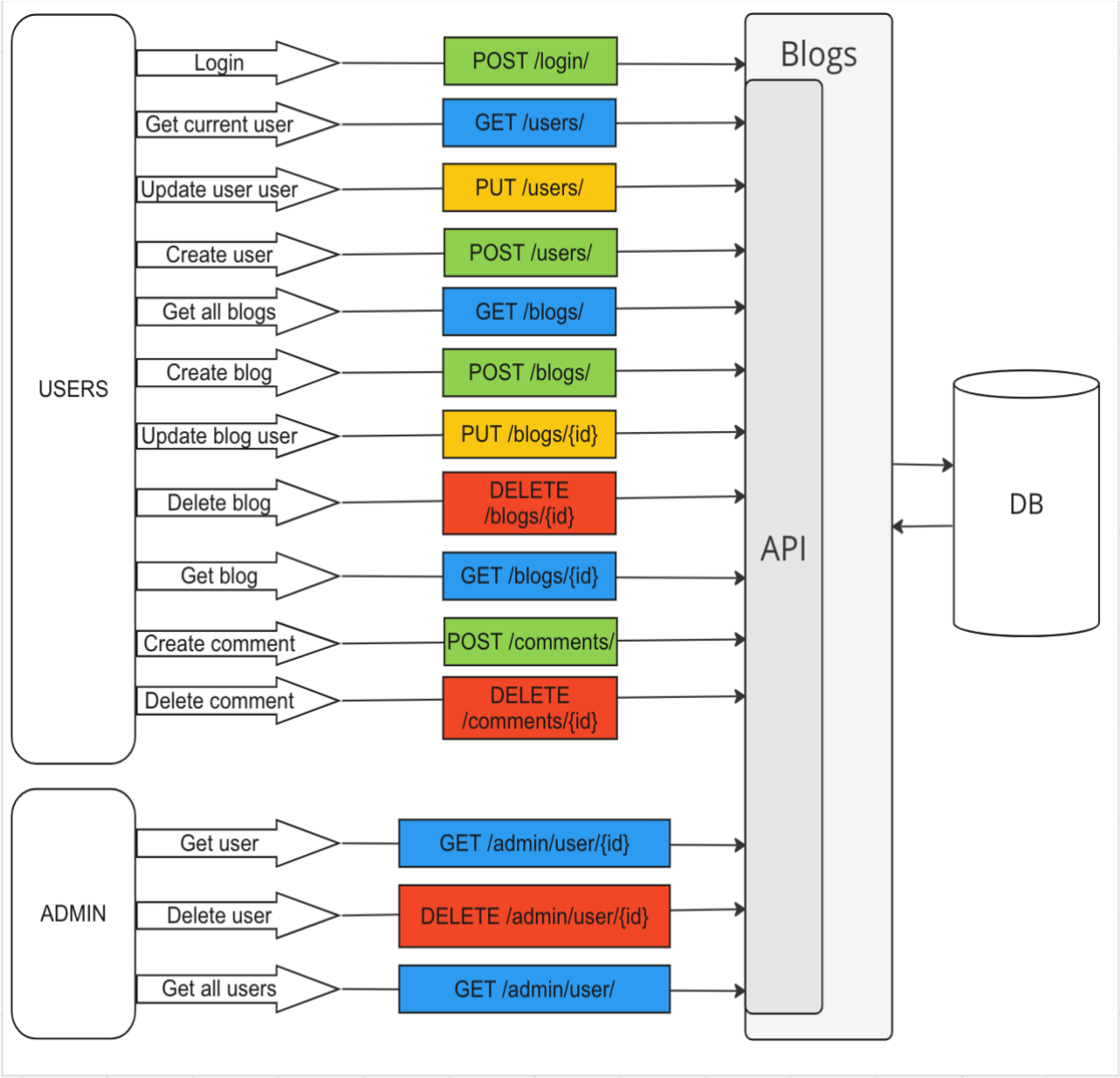


Рисунок 2.1 – Графічне зображення архітектури блогу

Таким чином, можна зазначити, що архітектура серверної частини та взаємодія між її компонентами виконана відповідно до вимог та є продовженням

бізнес логіки, що була описана раніше.

Взаємодія з базою даних. Більше детально про конкретну технологію, яка буде використана для збереження даних застосунку буде викладено у третьому розділі цієї роботи. В загальному вигляді ERD (Entity-relationship diagrams, графічне представлення структури бази даних, яке використовується для моделювання взаємозв'язків між сутностями та їх атрибутами) блогу виглядає наступним чином (рис. 2.2).

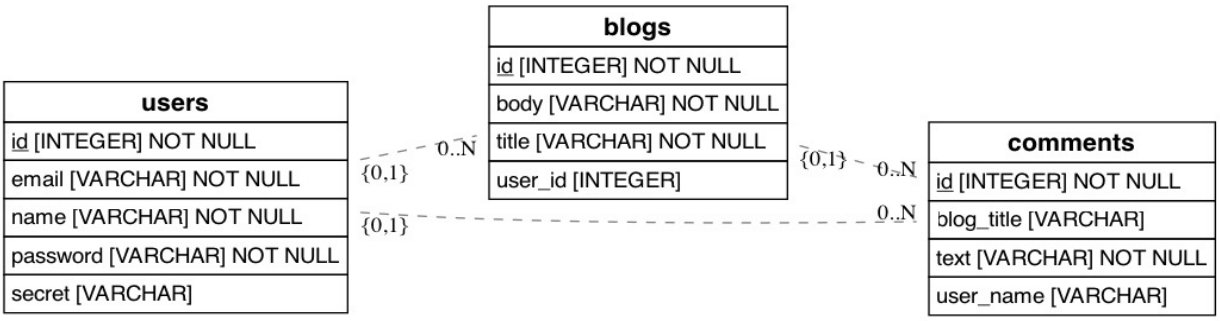


Рисунок 2.2 – ERD блогу

Застосунок має 3 таблиці: “users”, “blogs” та “comments”.

Атрибут “id” таблиці “users” виступає у ролі primary key власне для самої таблиці “users” і foreign key для таблиці “blogs”.

Атрибут “name” таблиці “users” є унікальним і виступає у ролі foreign key для таблиці “comments”.

Атрибут “id” таблиці “blogs” виступає у ролі primary key.

Атрибут “title” таблиці “blogs” є унікальним і виступає у ролі foreign key для таблиці “comments”.

Атрибут “id” таблиці “comments” виступає у ролі primary key.

Таким чином 0 або більше блогів може бути пов’язано не більше ніж з 1 користувачем. 0 або більше коментарів може бути пов’язано не більше ніж з 1 користувачем та блогом.

Усі дані отримуються від користувачів через API та відповідні шляхи

(endpoints) – пункти комунікації в мережі, зазвичай визначений за URL-адресою, які служать для доступу до конкретного ресурсу чи сервісу в мережі Інтернет. Дані отримані таким шляхом проходять валідацію та санітаризацію даних перед тим як бути збереженими в базу даних.

Системні дані. Враховуючи, що веб-застосунок розробляємий в рамках цієї роботи представляє собою міні блог, то значної кількості системних немає. Серед наявних можна виділити конфігураційні змінні, які генеруються у самому runtime необхідні для керування застосунком та його модулями. Наприклад, змінні, які визначають шлях до бази даних, токен адміністратора необхідний для авторизації його (її) дій, ін.

Забезпечення безпеки. Цей процес буде відбуватися за рахунок:

1) Шифрування паролей перед їхнім збереженням в базі даних, наприклад, шляхом використання для хешування паролів бібліотеки bcrypt, що забезпечує їхню безпеку та невідтворюваність навіть у випадку проникнення до бази даних.

2) Використання HTTPS та TLS для захищеного обміну інформацією між клієнтом і сервером. Це забезпечує шифрування даних під час їхньої передачі через мережу, запобігаючи можливим атакам на приватні дані. Встановлення сертифіката TLS дозволяє здійснювати безпечні HTTP-з'єднання, підвищуючи захищеність конфіденційності даних.

3) Включення валідації даних, генерації документації для API, а також використання вбудованих інструментів для аутентифікації та авторизації дозволяє покращити контроль та безпеку взаємодії з веб-застосунком, зменшуючи ризики зловживання та атак.

Користувацькі дані. Всі дані, які будуть циркулювати блогом та зберігатися, будуть отримуватися від користувачів (проходячи валідацію та санітаризацію). Серед них: ім’я та електронна пошта користувача, пароль та секрет для відновлення паролю. Крім того, заголовок та тіло блогу, а також коментарі до нього.

Відображення даних для користувача. Враховуючи, що наш блог буде мати лише повноцінну бекенд частину, планується використати такі засоби розробки програмного забезпечення, що в ідеалі дозволять згенерувати інтерфейс користувача. Крім того доступ до будь-якого шляху ресурсу має бути доступним користувачу шляхом взаємодії з API безпосередньо за допомогою HTTP-запитів. Тобто, розробник чи інша програма може використовувати HTTP-клієнт або інші засоби для надсилання запитів до сервера, взаємодіючи з ендпоінтами та отримуючи дані без використання графічного інтерфейсу користувача.

3. РЕАЛІЗАЦІЯ ВЕБ-ОРІЄНТОВАНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Вибір засобів програмної реалізації інформаційної системи

Вибір засобів програмної реалізації буде здійснюватися у відповідності до моделей, розроблених у попередньому розділі цієї роботи.

Перш за все потрібно визначитися з мовою програмування. Враховуючи, що розробляється бекенд компонента невеличкого веб-застосунку – блогу, а також спираючись на переваги і недоліки описані у першому розділі цієї роботи кожної з популярних мов програмування для веб-розробки, я хотів би реалізувати цей проект з використанням мови програмування Python. Вона гарно підходить для швидкої розробки, має велику кількість різноманітних фреймворків для розробки бекенд частини, а також я маю значний досвід програмування на цій мові, що має позивтивно вплинути на швидкість та якість розробки.

Наступним етапом буде вибір оптимального фреймворку для розробки веб-застосунків. При виборі фреймворку (одного з трьох наведених у 2 розділі цієї роботи) для розробки бекенд веб-застосунку я буду виходити з наступних факторів:

- вартість
- обсяг і складність проекту
- швидкість розробки
- кривої вивчення та наявних навичок розробників в команді
- наявність розвиненої спільноти та документації
- сучасність (future-proof)
- швидкість та продуктивність

Усі три фреймворки: Django, Flask та FastAPI є повністю безкоштовними.

Веб-застосунок, розробляємий в рамках цієї роботи буде відносно невеличким, тому за цим критерієм кращими будуть Flask (підійде для розробки

маленьких та середніх проектів) та FastAPI (дуже гнучкий, тому підійде для проектів будь-якої складності), в той час як Django більш націлений на середні та великі проекти.

Усі три перелічені фреймворки позиціонуються як такі, що забезпечують високу швидкість розробки продукту. А от крива вивчення фреймворку в Django трохи вища від Flask та FastAPI [37]. Особисто я маю досвід роботи з усіма трьома фреймворками, але комерційний досвід використання лише FastAPI.

Відповідно рейтингу 8 найкращих фреймворків Python для веб-розробки у 2022 році за версією Github Stars, Django — найпопулярніший веб-фреймворк із 65 687 зірками. На другому місці — Flask з 60 217, а на третьому — новачок FastAPI з 44 121 [20](рис 3.1.).

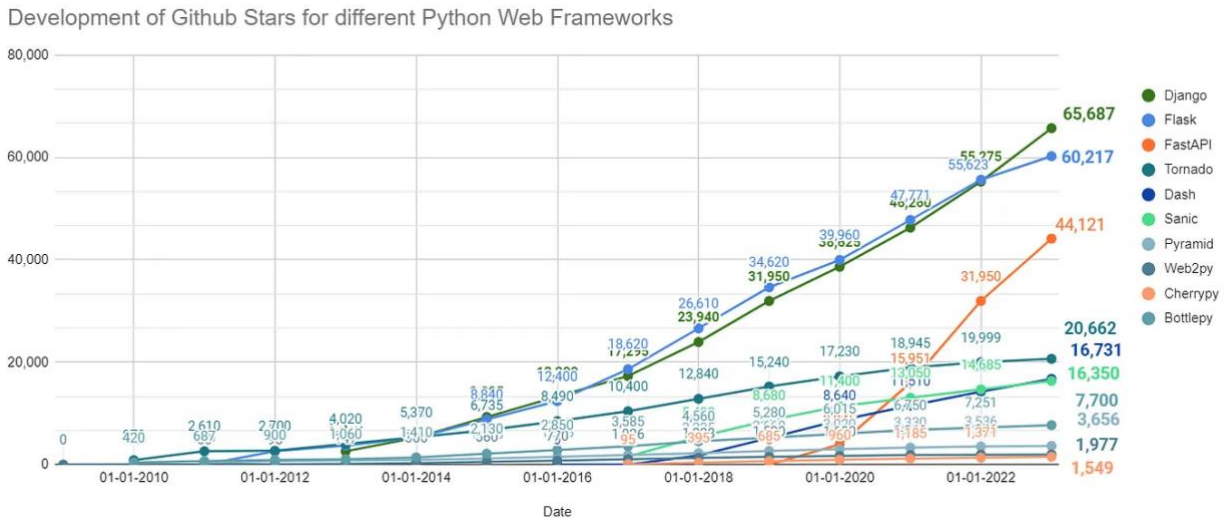


Рисунок 3.1 – Діаграма найбільш популярних фреймворків для мови програмування Python за версією Github Stars за 2022 рік

Наведена вище діаграма демонструє, що Django має перевагу з понад 5000 зірками, але Fastapi набирає силу. З 2021 по 2022 рік популярність Fastapi зросла на 38%, тоді як Django – лише на 18%. В цілому з точки зору наявності обільної документації та розвинутої спільноти усі три фреймворки досить близько, а незначне відставання FastAPI дуже стрімко скорочується.

Критерій сучасності та "захищений від майбутніх змін" для трьох фреймворків буде суттєво різнитися. Початкова дата випуску Django – 21 липня 2005, Flask – 1 квітня 2010, а FastAPI – 5 грудня 2018. Значна різниця відображається на технологіях які використовуються “під капотом” цих фреймворків. Старі версії Django (до 2019 року) та Flask (до 2021 року) не мали нативної підтримки асинхронного режиму виконання операцій на сервері, що значною мірою позначалося на швидкодії веб-застосунків реалізованих на базі цих фреймворків. Слід зазначити, що досі відповідно до офіційної документації Flask [38] “Підтримка асинхронності Flask є менш продуктивною, ніж фреймворки async-first через спосіб її реалізації”. Документація Django не декларує якихось “особливостей” у роботі в асинхронному режимі [39]. З іншої сторони FastAPI [40] був з самого початку побудований навколо підходу асинхронності, що дозволяє імплементувати асинхронний спосіб виклику ендпоінтів як з використанням async/await так і через threadpool. При цьому для цього не знадобиться допомога сторонніх бібліотек чи використання якихось хаків. FastAPI робить це в автоматичному режимі.

Асинхронний режим роботи API веб-застосунку має такі переваги [41]:

- вища продуктивність, дозволяючи одночасно обробляти кілька запитів, оскільки асинхронні API не блокують викликаючий потік під час обробки запиту;
- краща масштабуваність, дозволяючи обробляти більше запитів без збільшення кількості серверів;
- вищу надійність, зменшуючи ризик помилок, спричинених затримкою чи іншими проблемами мережі;
- поліпшує користувацький досвід, роблячи додатки більш відгукними (responsive), користувач може продовжувати користуватися додатком, поки запит обробляється в фоновому режимі.
- зменшення витрат за рахунок зменшення кількість ресурсів,

необхідних для виконання функцій веб-застосунку

Сучасність фреймворків визначає набір технологій, що використані “під їх капотом”, що в свою чергу позначається значною мірою на продуктивності та швидкості. Через те, що FastAPI є самим “молодим” фреймворком, а тому і з самого початку збудованому з більш сучасним “ядром”, він є і одним з найшвидших фреймворків для веб-розробки на мові програмування Python [42].

Враховуючи перераховані особливості кожного з трьох фреймворків, я вважаю що FastAPI є оптимальним вибором для імплементації бекенд частини блогу. Витримка з офіційної документації гарно підрезюмує його переваги [30,43]:

Швидкість роботи. Дуже висока продуктивність, на рівні з NodeJS і Go (завдяки Starlette і Pydantic). Один із найшвидших фреймворків Python.

Швидкість розробки. Покращує швидкість розробки функціоналу застосунку приблизно на 200–300%.

Менше помилок. Зменшує приблизно на 40% кількість помилок, спричинених людиною (розробником).

Інтуїтивно зрозумілий. Чудова підтримка IDE з автоматичним заповненням коду (auto-completion), що зменшує час відлагодження.

Простий. Розроблений, щоб бути простим у використанні та навчанні (менше часу на читання документації).

Лаконічність синтаксису. Мінімуму дублювання коду (DRY) за рахунок керування бажаним функціоналом на рівні оголошення параметра.

Надійний. Можливість легко отримати готовий для виводу продакшн код з автоматичною інтерактивною документацією.

Заснований на стандартах. Побудований на основі (і повністю сумісний з) відкритими стандартами для API: OpenAPI (раніше відомий як Swagger) і JSON Schema.

У якості серверу буде використано uvicorn – реалізація ASGI веб-сервера

для Python з потужним низько-рівневим інтерфейсом сервера/застосунку для асинхронних фреймворків [44]. Це дозволить в повній мірі розкрити асинхронний потенціал FastAPI. Крім того використання uvicorn рекомендовано власне самою документацію фреймворку, та й сам uvicorn має референс на FastAPI [45].

Розгортання веб-застосунку буде здійснюватися з використанням space (Deta) у якості хостингу, який є безкоштовним, а також рекомендується для використання самим FastAPI [46-47].

У якості СУБД (Database Management System, система управління базами даних) буде використано SQLite, яка гарно підходить для невеличкого проекту, а також не потребує окремого серверу для хостинга.

FastAPI пропонує використовувати SQLAlchemy у якості ORM (Object-Relational Mapping, об'єктно-реляційне відображення) – технологію програмування, яка надає зручний і вищий рівень абстракції для взаємодії з базами даних, перетворюючи об'єкти програмного коду на записи бази даних і навпаки [48].

Схема даних – класів-моделей для користувачів, блогів та коментарів, буде описана за допомогою SQLAlchemy та Pydantic.

Маршрути:

- /login/ – аутентифікація
- /users/ – реєстрація та управління користувачами
- /blogs/ – створення, читання, редагування та видаленням блогів
- /comments/ – створення та видалення коментарів
- /admin/ – адміністративні функції

Взаємодія з кожним типом ресурсу для обробки логіки бізнес-процесів буде здійснюватися через відповідні контролери.

Для аутентифікації користувачів та захисту API-шляхів буде використано JWT (JSON Web Token).

Для валідації вхідних даних використаємо Pydantic [49].

Безпека застосунку буде реалізована за рахунок хешування паролів перед їх збереженням з використанням бібліотеки passlib [50].

Також передача даних буде здійснюватися через HTTPS (Hypertext Transfer Protocol Secure) та SSL (Secure Sockets Layer, рівень захищених сокетів), що доступний out of box для хостингу на space (Deta) [51-52].

Логіка бізнес-процесів описана у розділі 2 цієї роботи.

Розробка буде здійснюватися з використанням IDE Visual Studio Code з використанням розподіленої системи керування версіями файлів та спільної роботи Git, а також GitHub у якості віддаленого сховища для Git-репозиторію.

3.2 Опис програмної реалізації інформаційної системи

Структура проекту (рис 3.2):

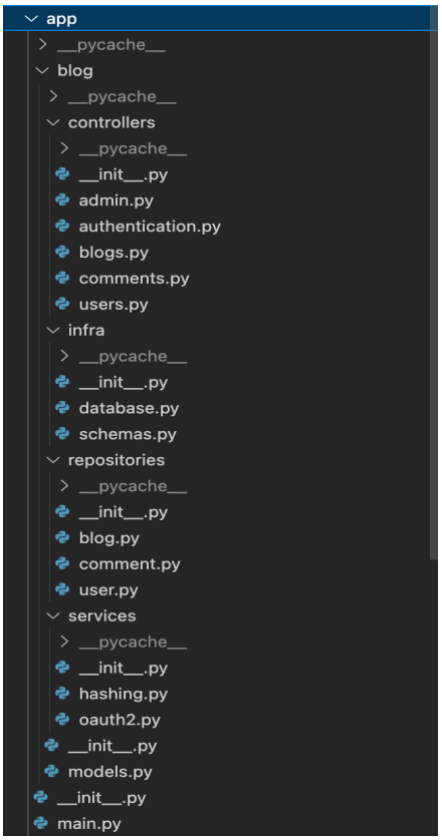


Рисунок 3.2 – Структура проекту блогу

Кореневим каталогом для блогу є директорія app. Всередині неї розташовані каталог blog та модуль main.py який є ентріпоінтом для запуску застосунку. Каталог blog містить підкаталоги з контролерами (каталог controllers), інфраструктурними модулями (каталог infra), репозиторіями (каталог repositories), сервісами (каталог services), а також модуль models.py, що описує моделі даних, з якими взаємодіють контролери (моделі, що відображають бізнес логіку застосунку).

Взаємодія користувача (адміністратора) з блогом здійснюється через 6 роутеров, які виконують функцію контроллеров [53]:

- admin
- authentication
- blogs
- comments
- users

Роутер admin групує в собі 2 шляхи (endpoints):

- /admin/user/{id} – доступний для виклику з HTTP методами GET та DELETE
- /admin/user/ – доступний для виклику з HTTP методом GET

Роутер authentication включає в себе 1 шлях:

- /login/ – доступний для виклику з HTTP методом POST

Роутер blogs групує в собі 2 шляхи:

- /blogs/– доступний для виклику з HTTP методами POST та GET
- /blogs/{id} – доступний для виклику з HTTP методами PUT, GET та DELETE

Роутер comments групує в собі 2 шляхи:

- /comments/ – доступний для виклику з HTTP методом POST
- /comments/{id} – доступний для виклику з HTTP методом DELETE

Роутер users включає в себе 1 шлях:

- /users/ – доступний для виклику з HTTP методами PUT, GET та POST

Усі шляхи описані відповідними функціями та декоровані екземплярами (instance) роутерів для їх агреграції.

Схема зберігаємих даних (внутрішні моделі, які не доступні користувачу чи адміністратору) описана у модулі schemas.py підкаталогу infra, та у точності відповідає раніше описаній ERD. Кожен клас моделі даних є підкласом екземпляру (instance) declarative_base об’єкту з бібліотеки SQLAlchemy – базового класу для визначення декларативних класів. Використання цього підходу значно спрощує отримання та запис даних у базу даних, а також їх валідацію на відповідність задекларованих типів даних.

Таким чином є три класи Blog, User та Comment.

Blog (таблиця blogs) має такі атрибути:

- id = Column(Integer, primary_key=True, index=True)
- title = Column(String, unique=True, nullable=False)
- body = Column(String, nullable=False)
- user_id = Column(Integer, ForeignKey('users.id'))
- creator = relationship('User', back_populates='blogs')
- comments = relationship('Comment')

User (таблиця users) має такі атрибути:

- id = Column(Integer, primary_key=True, index=True)
- name = Column(String, unique=True, nullable=False)
- email = Column(String, nullable=False)
- password = Column(String, nullable=False)
- secret = Column(String, nullable=True)
- blogs = relationship('Blog', back_populates='creator')
- comments = relationship('Comment')

Comment (таблиця comments) має такі атрибути:

- id = Column(Integer, primary_key=True, index=True)
- text = Column(String, nullable=False)
- user_name = Column(String, ForeignKey('users.name'))
- blog_title = Column(String, ForeignKey('blogs.title'))
- creator = relationship('User', back_populates='comments')
- blog = relationship('Blog', back_populates='comments')

Зазначу, що relationship – це спеціальний інструмент який дозволяє наповнювати моделі даних, даними з інших моделей, що з ними пов’язані таким чином [54].

Як зазначалося раніше, моделі, що відображають бізнес логіку застосунку розташовані в модулі models.py. Кожен клас, що їх описує є субкласом від BaseModel бібліотеки Pydantic (яка є однією з ключових бібліотек навколо яких побудовано FastAPI).

Такі моделі визначають поля як анотовані атрибути.

Моделі мають багато схожості з класами даних Python, але були розроблені з деякими невеличкими, але важливими відмінностями, які спрощують певні робочі процеси, пов’язані з перевіркою, серіалізацією та генерацією схем JSON.

Ненадійні дані можуть бути передані в модель, і після аналізу та перевірки Pydantic гарантує, що поля отриманого екземпляра моделі відповідатимуть типам полів, визначеним у моделі [55].

Таким чином моделі Pyndantic створюють певного рода контракт, який регламентує структуру даних запиту до шляху та відповіді (варіанти відповідей) на такий запит, у разі використання їх в імплементації шляху певного роутера FastAPI.

Присутні такі моделі:

BaseBlog (базовий клас для субкласів Blog та ShowBlog) з такими полями:

- title: str

- body: str

Blog (модель, що описує запит до деяких шляхів роутеру blogs):

- не має власних полей, але містить конфігураційне клас Config всередині себе

User (модель, що описує запит до деяких шляхів роутеру users):

- name: str
- email: str
- password: str
- secret: Optional[str] = None

ShowUserInBlog (базовий клас для субкласу ShowUser):

- id: int
- name: str
- email: str

Comment (модель, що описує запит до деяких шляхів роутеру comments):

- blog_title: str
- text: str

ShowComment (модель, що описує відповідь на запити до деяких шляхів роутеру comments):

- id: int

blog_title: str

- user_name: str
- text: str

ShowUser (модель, що описує відповідь на запити до деяких шляхів роутеру users):

- blogs: List[Blog] = []
- comments: List[ShowComment] = []

ShowBlog (модель, що описує відповідь на запити до деяких шляхів роутеру blogs):

- id: int
- creator: ShowUserInBlog
- comments: List[ShowComment] = []

За рахунок того, що FastAPI побудований на основі (і повністю сумісний з) OpenAPI і JSON Schema в ньому реалізований функціонал, що дозволяє автоматично генерувати документацію, що описує наявні роутери та шляхи, а також схеми даних.

Ця документація за замовчуванням доступна за шляхом /docs/ і для застосунку блог виглядає наступним чином (рис 3.3, 3.4):

Рисунок 3.3 – Загальний опис API-шляхів за стандартом OpenAPI

Рисунок 3.4 – Загальний вигляд JSON Schema для моделей, що використовуються для взаємодії з API-шляхами

Взаємодія контролера (функцій, що імплементують шляхи роутерів) з базою даних здійснюється через відповідні репозиторії (каталог repositories): blog.py, comment.py, user.py. Отримання, збереження та видалення даних у базі даних відбувається шляхом використання відповідних методів об’єкту Session з бібліотеки SQLAlchemy.

Також застосунок використовує сервіси, що організовані у модулях hashing.py та oauth2.py підкаталогу services.

Модуль hashing.py імплементує клас Hash який відповідає за створення хешовонаго значення з отриманого від користувача паролю, перед тим як його зберегти його у базу даних. Це забезпечує надійний захист чутливих даних користувача, навіть якщо база даних буде зламана. Для хешування використовують бібліотека passlib. Крім того цей клас має метод для подальшої верифікації паролю, введеному користувачем при спробі аутентифікації та зашифрованого паролю, що зберігається у базі даних.

Модуль oauth2.py відповідає за створення JWT токенів під час входу в систему користувача (login) необхідних для виконання дій, що потребують

авторизації. Це забезпечує необхідну сегментацію доступу до інформації користувачів підвищуючи безпеку застосунку.

Крім того в цьому модулі розташована функція, яка відповідає за перевірку токену адміністратора (admin_token), необхідного для здійснення адміністративних функцій.

Обробка помилок та виключень у застосунку здійснюється на різних рівнях і різними інструментами.

По-перше, у разі помилки пов’язаною з валідацію даних запиту, або некорекного запиту до шляху FastAPI відобразить (направить відповідну відповідь на запит) специфічну читабельну помилку.

Наприклад, у разі виклику шляху /users/ з методом PUT з хибними даними FastAPI автоматично згенерує відповідь з кодом 422 Validation Error та тілом, що може бути описано схемою:

```
{
  "detail": [
    {
      "loc": [
        "string",
        0
      ],
      "msg": "string",
      "type": "string"
    }
  ]
}
```

Крім тому, на рівні репозиторіїв також імплементовані засоби обробки

найбільш очікуваних помилок з використанням зарезервованих слів Python try/except.

Наприклад, у разі спроби створити блог, заголовок (title) якого вже існує в базі даних, що призведе до порушення Unique Constraint та помилки, передбачено оброблення помилки бази даних, та підняття (raise) помилки класу HTTPException, що вбудована в фреймворк FastAPI, з читабельним текстом відповіді та кодом 422 (UNPROCESSABLE ENTITY).

Приклад коду:

```
try:
    db.commit()
  except IntegrityError:
    db.rollback()
    raise HTTPException(
status_code=status.HTTP_422_UNPROCESSABLE_ENTITY,
detail=f'Blog with title {request.title} already exists')
```

Крім того, хоч і на цей час в застосунку не використовується жодна бібліотека для логіювання окремих частин коду (наприклад стандартна бібліотека logging), деякі логи, що відносяться до викликів шляхів FastAPI та роботи uvicorn можна побачити на хостингу space (Deta) через Runtime Logs панель [56](рис 3.5).

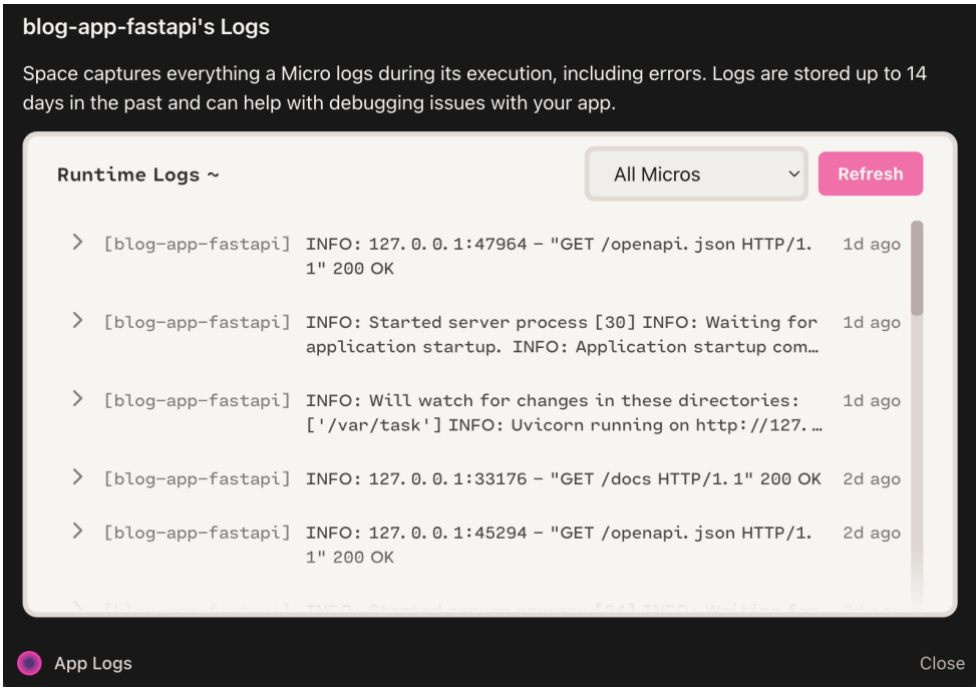


Рисунок 3.5 – Панель space (Deta) Runtime Logs

3.3 Тестування інформаційної системи та інструкція користувача

В рамках цієї роботи проведено наступні види тестування:

- unit-тестування
- функціональне тестування (API-тестування)
- навантажувальне (load) тестування
- secure code аналіз

Unit-тестування – перевірка окремих частин програмного коду (функцій, методів, класів) на коректність їх роботи, виконано з використанням бібліотеки Pytest [57].

Приклад unit-тесту (перевіряє роботу репозиторія, що відповідає за створення коментарів):

```
def test_create_comment() -> None:
    text = 'some text'
    db = next(get_db())
```

```
    new_user = user.create(models.User(name='john39', email='email',
password='password'), db)
    new_blog = blog.create(models.Blog(title='Cool title', body='blog body'),
new_user.id, db)
    new_comment = comment.create(models.Comment(
      text=text, blog_title=new_blog.title), user_name=new_user.name, db=db)
    next(get_db())
    assert type(new_comment) is schemas.Comment
    assert new_comment.text == text
    assert new_comment.blog_title == new_blog.title
    assert new_comment.user_name == new_user.name
```

Функціональне тестування – вид тестування, який перевіряє, чи відповідає програмне забезпечення визначеним вимогам та чи виконує воно очікувані функції. У випадку функціональне тестування також виконує роль API-тестування, т.я. усі функції застосунку реалізовано шляхом взаємодії користувача/адміністратора з застосунком через існуючі API-шляхи. Крім того функціональни тести також тестують API взаємодії застосунку за базою даних.

Функціональне тестування API-шляхів реалізоване з використанням об’єкту TestClient з бібліотеки FastAPI [58].

Приклад функціонального тесту (перевіряє позитивний сценарій створення коментаря через відповідний API-шлях):

```
def test_create_comment(test_client: TestClient) -> None:
    username = 'john'
    password = '123'
    blog_title = 'Cool title'
    comment_text = 'some text'
```

```
    with test_client:
        user_response = test_client.post(url='/users',
                                         json=dict(name=username,
                                                   password=password,
                                                   email='email@gmail.com'))
        assert user_response.status_code == 201
        auth_response = test_client.post(url='/login',
                                         data=dict(username=username,
                                                   password=password))

        assert auth_response.status_code == 200
        blog_response = test_client.post(
            url='/blogs', json=dict(title=blog_title, body='Some cool stuff'),
            headers={'Authorization': f'Bearer
{auth_response.json()["access_token"]}'})
        assert blog_response.status_code == 201
        assert blog_response.json()['title'] == blog_title
        comments_response = test_client.post(
            url='/comments', json=dict(blog_title=blog_title, text=comment_text),
            headers={'Authorization': f'Bearer
{auth_response.json()["access_token"]}'})
        assert comments_response.status_code == 201
        assert comments_response.json()['blog_title'] == blog_title
        assert comments_response.json()['text'] == comment_text
```

Рівень покриття коду тестами pytest (unit та функціональне тестування) в проекті – 100%.

Для аналізу покриття коду тестами я використав бібліотеку coverage

[59](рис 3.6, рис 3.7).

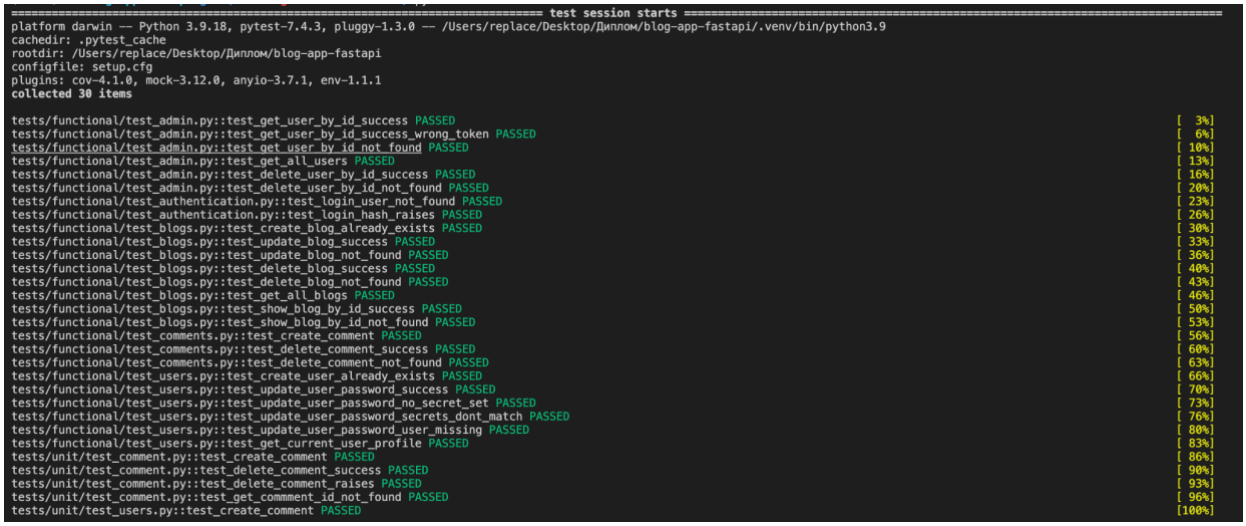


Рисунок 3.6 – Успішні результати тестування з використанням Pytest

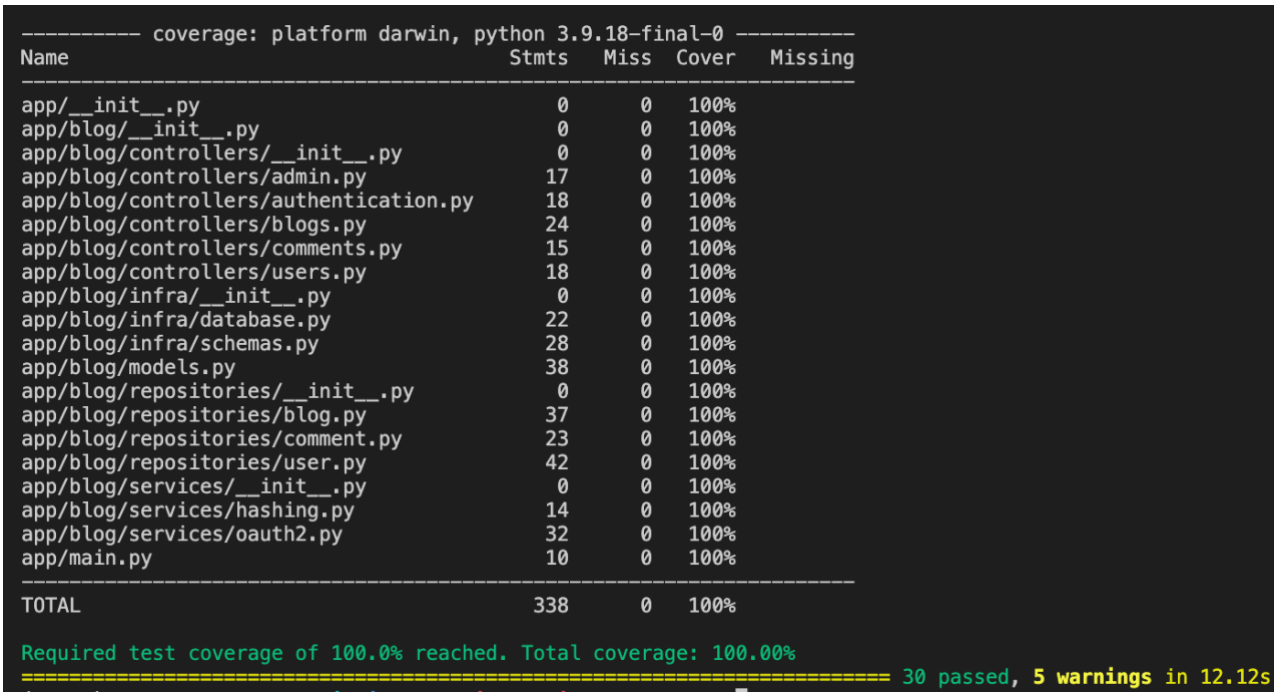


Рисунок 3.7 – Повне покриття тестами коду, відображене з використанням Coverage

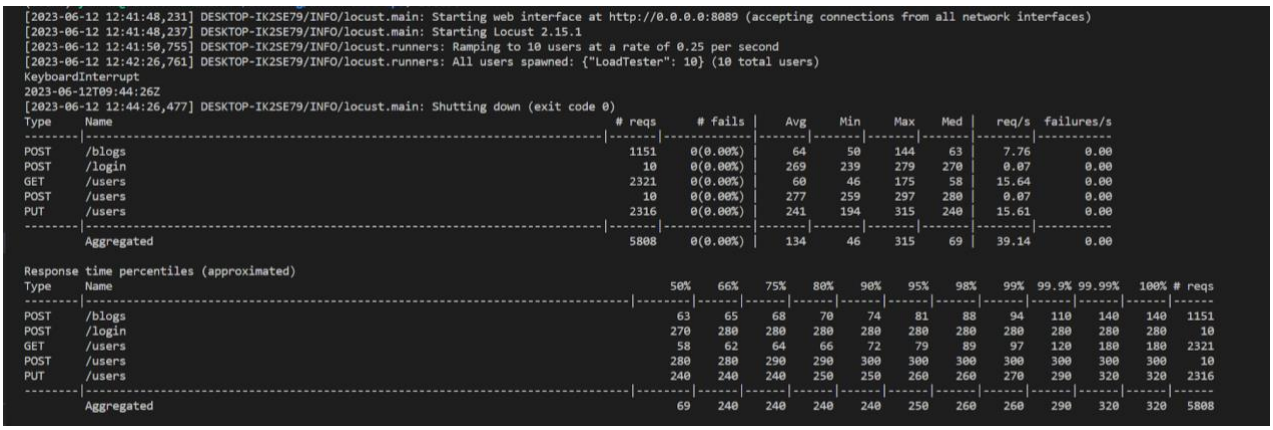
У зв’язку з тим, що хмарний хостинг space (Deta), на якому було

розгорнуто застосунок, не дозволяє здійснювати write операції з файлами (можливо через те, що він є лише надбудовою над AWS-lambda, стосовно цієї “особливості” інформація в офіційній документації відсутня), навантажувальне тестування – тестування програмного забезпечення, спрямоване на оцінку та перевірку працездатності системи при великому обсязі робочих навантажень, здійснювалося виключно локально.

В якості бібліотеки для проведення цього виду тестування я використав Locust [60].

Тестування проведено з імітацією використання 10 користувачами одночасно протягом 2 хвилин з поступовим додаванням 1 користувача один раз на 4 секунди для імітації "прогрівання" серверу.

Метрика що аналізується - Response Time, що відповідно до результатів тестування (рис 3.8) складає: 50th percentile Aggregated: 69 ms, 95th percentile Aggregated: 250 ms. Це означає, що в 50 відсотках усіх викликів ендпоінтів відповідь сервера була надена не більше ніж за 71 мс, а в 95 відсотках – не більше ніж за 250 мс. Крім того, при Aggregated кількості здійснених запитів у розмірі 5808 (за 2 хвилини) немає жодного відказу (Aggregated рядок стовпчику fails дорівнює значенню 0).



```
[2023-06-12 12:41:48,231] DESKTOP-IK2SE79/INFO/locust.main: Starting web interface at http://0.0.0.0:8089 (accepting connections from all network interfaces)
[2023-06-12 12:41:48,237] DESKTOP-IK2SE79/INFO/locust.main: Starting Locust 2.15.1
[2023-06-12 12:41:50,755] DESKTOP-IK2SE79/INFO/locust.runners: Ramping to 10 users at a rate of 0.25 per second
[2023-06-12 12:42:26,761] DESKTOP-IK2SE79/INFO/locust.runners: All users spawned: {"LoadTester": 10} (10 total users)
KeyboardInterrupt
2023-06-12T09:44:26Z
[2023-06-12 12:44:26,477] DESKTOP-IK2SE79/INFO/locust.main: Shutting down (exit code 0)
Type     Name                                                                          # reqs      # fails |    Avg     Min     Max    Med |   req/s  failures/s
--------|-----------------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
POST     /blogs                                                                         1151     0(0.00%) |     64      50     144     63 |    7.76        0.00
POST     /login                                                                           10     0(0.00%) |    269     239     279    270 |    0.07        0.00
GET      /users                                                                         2321     0(0.00%) |     60      46     175     58 |   15.64        0.00
POST     /users                                                                           10     0(0.00%) |    277     259     297    280 |    0.07        0.00
PUT      /users                                                                         2316     0(0.00%) |    241     194     315    240 |   15.61        0.00
--------|-----------------------------------------------------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
         Aggregated                                                                     5808     0(0.00%) |    134      46     315     69 |   39.14        0.00

Response time percentiles (approximated)
Type     Name                                                                             50%    66%    75%    80%    90%    95%    98%    99%  99.9% 99.99%   100% # reqs
--------|-----------------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|------|------|------
POST     /blogs                                                                            63     65     68     70     74     81     88     94    110    140    140   1151
POST     /login                                                                           270    280    280    280    280    280    280    280    280    280    280     10
GET      /users                                                                            58     62     64     66     72     79     89     97    120    180    180   2321
POST     /users                                                                           280    280    290    290    300    300    300    300    300    300    300     10
PUT      /users                                                                           240    240    240    250    250    260    260    270    290    320    320   2316
--------|-----------------------------------------------------------------------------|--------|------|------|------|------|------|------|------|------|------|------|------
         Aggregated                                                                        69    240    240    240    240    250    260    260    290    320    320   5808
```

Рисунок 3.8 – Результати навантажувального тестування з використанням Locust

Secure code аналіз – перевірка програмного коду з метою виявлення потенційних вразливостей виконано за допомогою бібліотеки Bandit.

Bandit — це інструмент, призначений для пошуку поширених проблем безпеки в коді Python (статистичний аналіз коду) [61]. Це інструмент є рекомендованим засобом для мови програмування Python [62].

Результати сканування не виявили наявних проблем (рис 3.9)

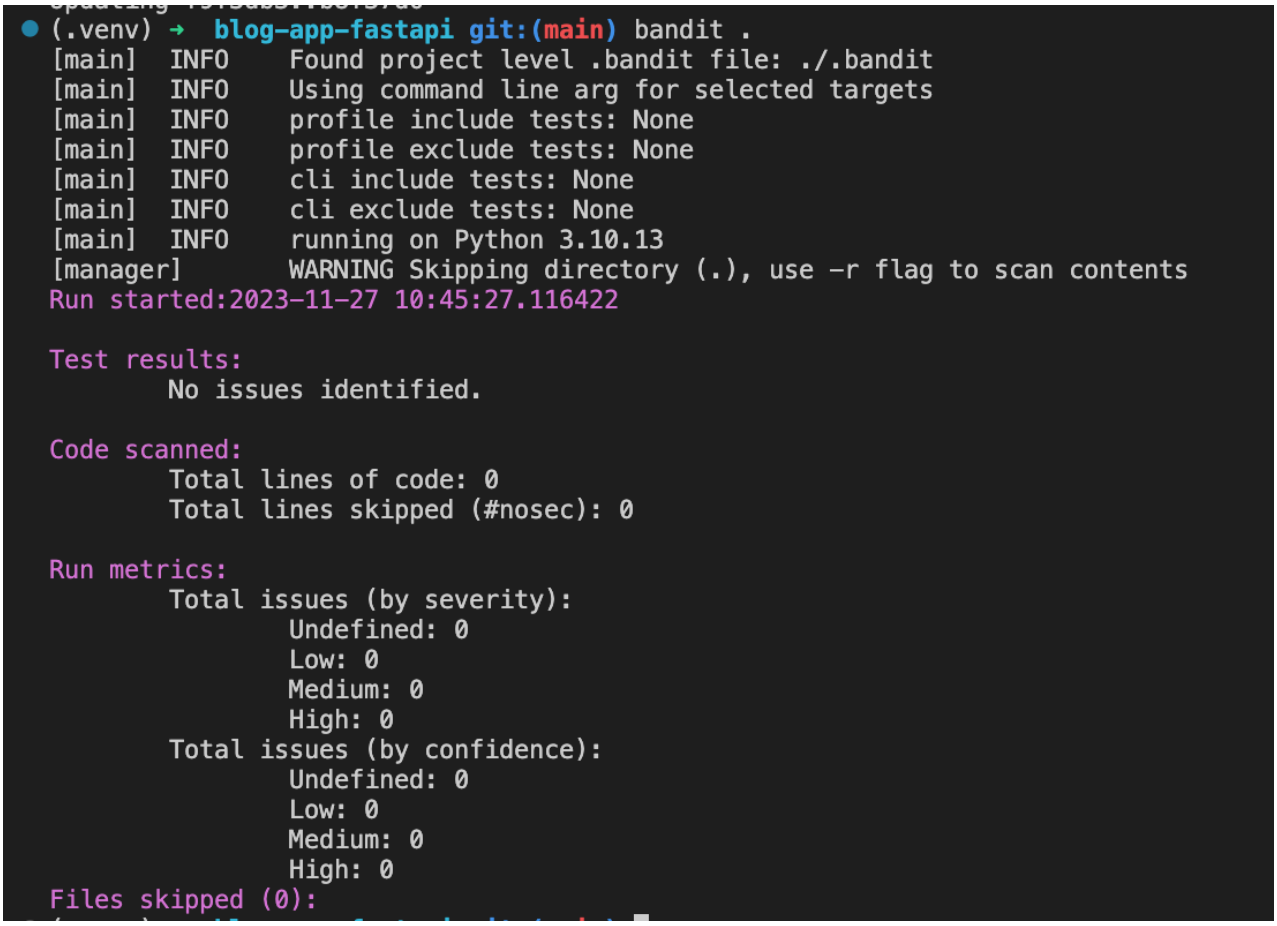


Рисунок 3.9 – Успішні результати статичного аналізу безпечності коду застосунку з використанням Bandit

Використання реалізованого веб-застосунку – блогу користувачу (та адміністратору) здійснюється відповідно до бізнес-логіки, що була описана у розділі 2 цієї роботи, а також згідно з документацією, що описує API-шляхи у попередньому підрозділі цього розділу роботи. Нижче будуть описані основні

операції доступні для виконання користувачем та адміністратором у блозі.

Більшість операцій з блогом потребує аутентифікації в системі, тому перш за все користувачу необхідно зареєструватися. Зробити це можливо шляхом направлення POST запиту на API-шлях ‘/users/’ з такою схемою даних для тіла запиту:

```
{
  "name": "string",
  "email": "string",
  "password": "string",
  "secret": "string"
}
```

Поле ‘secret’ є опціональним і використовується для відновлення паролю (якщо воно було присутнє під час створення користувача). Поле ‘name’ має бути унікальним. У разі наявності в базі даних користувача з таким іменем користувач отримає відповідь від сервера з кодом 422 та повідомленням про те, що такий користувач вже існує. У позитивному сценарії користувач отримає відповідь кодом 201.

Після реєстрації користувач може увійти в систему надіславши POST запит на API-шлях ‘/login/’ з такою схемою даних для тіла запиту:

```
{
  "username": "string",
  "password": "string"
}
```

У разі, якщо користувач ввів коректні дані, сервер надішле відповідь з

кодом 200 та полем ‘access_token’, значення якого в подальшому використовується для здійснення запитів, що потребують авторизації шляхом підстановки його в Authorization header.

У разі відсутності користувача з таким іменем, застосунок поверне код відповіді 404, а у разі неспівпадіння пароля (його хешу) з тим, що зберігається (зашифрованим) в базі даних – код 401.

Після того, як користувач успішно провів аутентифікацію, він (вона) може створювати блоги надсилаючи запити на API-шлях ‘/blogs/’ з такою схемою даних для тіла запиту:

```
{
  "title": "string",
  "body": "string"
}
```

Де поле ‘title’ – це заголовок блогу, який має бути унікальним, а ‘body’ – зміст.

У разі наявності в базі даних існуючого блогу з аналогічним заголовком сервер поверне користувачу відповідь з кодом 422 та відповідним повідомленням у тілі відповіді.

Крім того користувач може оновлювати ти видаляти власні блоги змінюючи заголовок та зміст блогу за допомогою виклику ‘/blogs/{id}’ з методами PUT та DELETE відповідно, де ‘id’ це ідентифікатор попередньо створеного блогу. Видалення блогу також призводить до видалення всіх коментарів пов'язаних з ним.

Також користувачам (навіть не авторизованим) доступні перегляд усіх наявних блогів, а також якогось конкретного за його ‘id’. Для цього необхідно викликати ‘/blogs/’ та ‘/blogs/{id}’ відповідно з методом GET.

У разі успішного створення блогу користувач отримає відповідь з кодом 201 та наступною схемою тіла відповіді:

```
{
	"title": "string",
	"body": "string",
	"id": 0,
	"creator": {
	 "id": 0,
	 "name": "string",
	 "email": "string"
	},
	"comments": []
}
```

Аналогічну відповідь отримає користувач і у разі виклику GET ‘/blogs/{id}’ або ‘/blogs/’, тільки з кодом відповіді 200. В останньому випадку відповідь містити масив з існуючими блогами описаними по такій же схемі.

Також користувачу доступна опція зі створення та видалення коментарів.

Створити коментар можливо шляхом виклику API-шляху ‘/comments/’ за допомогою методу POST та тілом запиту з такою схемою:

```
{
	"blog_title": "string",
	"text": "string"
}
```

Де ‘blog_title’ – це назва блогу, коментар до якого хоче залишити

користувач, а ‘text’ – це зміст коментаря.

У разі якщо блог з вказаним заголовком відсутній в базі даних, сервер надішле відповідь з кодом 404 та відповідним текстом.

Позитивний сценарій передбачає створення коментаря до блогу, повернення відповіді з кодом 201 та тілом відповіді з такою схемою:

```
{
        "id": 0,
        "blog_title": "string",
         "user_name": "string",
         "text": "string"
}
```

Також користувач може видаляти власні коментарі по їх ‘id’ викликаючи ‘/comments/{id}’ з методом DELETE.

В застосунку також присутня адміністративна частина. До неї відносяться 2 API-шляхи: ‘/admin/user/{id}’, який можна викликати з методами GET та DELETE, а також ‘/admin/user/’, який можна викликати з методом GET.

Для здійснення викликів адмін шляхів необхідно додавати до header виклику спеціальний ‘token’, який є секретним і має відповідати значенню, що зберігається в оточенні застосунку.

Якщо такий токен є некоректним, то той хто робив відповідний запит отримає від серверу відповідь з кодом 401 та тілом з відповідним повідомленням про некоректність токену.

GET ‘/admin/user/{id}’ дозволяє отримати інформацію по користувачу по його ‘id’. У разі наявності відповідного користувача у базі даних застосунку адміністратор отримає відповідь з кодом 200 та наступною схемою тіла відповіді:

```
{
  "id": 0,
  "name": "string",
  "email": "string",
  "blogs": [],
  "comments": []
}
```

Для GET ‘/admin/user/’ код буде також 200, а тіло відповіді буде майже таким самим, але це вже буде масив, що буде включати в себе набір користувачів, кожен з яких буде описаний за схемою наведеною вище.

Виклик DELETE ‘/admin/user/{id}’ призводить до отримання відповіді з кодом 200, але без тіла.

Видалення користувача також призводить до видалення всіх блогів та коментарів пов'язаних з ним.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було виконано ряд завдань.

Проведено загальний огляд веб-орієнтованих інформаційних систем, та прийнято рішення про реалізацію бекенд веб-застосунку.

Здійснено аналіз мов програмування, що використовуються для розробки інформаційних систем, серед яких: Java, JavaScript, PhP, Ruby та Python.

Проведено загальний огляд фреймворків, що використовуються для розробки інформаційних систем для вищезгаданих мов програмування.

Здійснено порівняльний аналіз популярних фреймворків, що використовуються для розробки інформаційних систем, а саме: React, Angular, Vue.js, Spring, Ruby on Rails, Laravel, Symfony, Django, Flask та FastAPI.

Досліджено та розроблено інформаційну модель інформаційної системи, яка складається з наступних елементів: логіка бізнес процесів, архітектура веб-застосунку, взаємодія з базою даних, системні дані, забезпечення безпеки, користувацькі дані, відображення даних для користувача.

Проведено аналіз підходящих засобів програмної реалізації інформаційних систем, та на підставі цього аналізу обрано такі, які оптимально підходять для цілей проекту, а саме: мова програмування – Python, фреймворк для реалізації API-шляхів – FastAPI, сервер – uvicorn, хостинг – space (Deta), СУБД – SQLite, ORM – SQLAlchemy, валідація даних – Pydantic, IDE – Visual Studio Code.

Розроблено інформаційну систему з урахуванням обраних засобів програмної реалізації, а також здійснено її опис.

Також описано результати тестування інформаційної системи, що свідчать про в її гарне функціонування, а саме: за рахунок повного тестового покриття функціоналу застосунку виключена більшість потенційних помилок під час використання, перевірка захищеності коду показала відсутність загальновідомих вразливостей, крім того застосунок продемонстрував здатність працювати під

навантаженням.

Розроблено та описано інструкцію користувача інформаційної системи.

Надалі планується здійснити міграцію застосунку з SQLite на PostgreSQL, змінити хостинг на той, що не буде створювати обмежень з операціями, доступними для виконання на ньому, розробити фронтенд частину для покращення досвіду користування застосунком. Крім того, планується протестувати поведінку застосунку під навантаженням з використанням інших фреймворків. Для цього потрібно буде створити застосунки з аналогічним набором функціоналу (або хоча б його елементів).

В цілому, за результатами досліджень здійснених в рамках цієї роботи, а також розробки інформаційної системи, проведеної на підставі зроблених досліджень, можна стверджувати, що використання оптимальних (доцільних) інструментів розробки веб-застосунків значною мірою полегшує безпосередньо процес розробки, покращує якість кінцевого продукту, підвищує стабільність, надійність та масштабованість системи, і захищає її від вразливостей пов’язаних, як з викликами, що присутні на момент проектування та розробки, так і з тими, що можуть виникнути в майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 10 types of web applications and how you can use them [Електронний ресурс] – Режим доступу до ресурсу: https://www.imaginarycloud.com/blog/10-types-of-web-applications-and-how-you-can-use-them/

2. 8 Types of Web Applications [Електронний ресурс] – Режим доступу до ресурсу: https://www.claysys.com/blog/types-of-web-applications/

3. What Is a Web Application? (With Benefits and Jobs) [Електронний ресурс] – Режим доступу до ресурсу: https://www.indeed.com/career-advice/career-development/what-is-web-application

4. What is a Web Application? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/web-application

5. The Top 6 Programming Languages in 2021 [Електронний ресурс] – Режим доступу до ресурсу: https://castillians.com/blogs/the-top-6-programming-languages-in-2021/203

6. 12 Best Languages for Web Development in 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://www.browserstack.com/guide/best-language-for-web-development

7. Advantages And Disadvantages Of JavaScript That You Need To Know! [Електронний ресурс] – Режим доступу до ресурсу: https://unstop.com/blog/advantages-and-disadvantages-of-javascript

8. 8 Async Javascript challenges for senior developers [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@nikita.malyschkin/8-async-javascript-challenges-for-senior-developers-b994f204d086

9. Java vs Python for Web Development: Comparison [Електронний ресурс] – Режим доступу до ресурсу: https://ncube.com/java-vs-python-for-web-development-the-ultimate-comparison/

10. Python vs. PHP - differences, pros, and cons [Електронний ресурс] – Режим доступу до ресурсу: https://sunscrapers.com/blog/python-vs-php-differences-

pros-and-cons/

11. Ruby on Rails: Pros & Cons. Things to Consider When Choosing the Technology [Електронний ресурс] – Режим доступу до ресурсу: https://www.netguru.com/blog/pros-cons-ruby-on-rails

12. The role of APIs in modern web development [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/pulse/role-apis-modern-web-development-erainterfaces/

13. Benefits Of Using Frameworks For Web Applications Development [Електронний ресурс] – Режим доступу до ресурсу: https://www.sencha.com/blog/benefits-of-using-frameworks-for-web-applications-development/

14. What are web frameworks and why you need them? [Електронний ресурс] – Режим доступу до ресурсу: https://intelegain-technologies.medium.com/what-are-web-frameworks-and-why-you-need-them-c4e8806bd0fb

15. Difference Between MVC vs MVP vs MVVM [Електронний ресурс] – Режим доступу до ресурсу: https://www.educba.com/mvc-vs-mvp-vs-mvvm/

16. 10 Best JavaScript Frameworks to Use in 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://hackr.io/blog/best-javascript-frameworks

17. 10 Best Java Frameworks for Web Development in 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://www.aimprosoft.com/blog/java-framework-for-web-development/

18. Top 10 Ruby Frameworks for Web Development [Електронний ресурс] – Режим доступу до ресурсу: https://techreviewer.co/blog/top-10-ruby-frameworks-for-web-development

19. 10 Best PHP Frameworks for Web Development in 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://www.cloudways.com/blog/best-php-frameworks/

20. Python Web development in 2022: Which web frameworks are the most popular by Github stars? [Електронний ресурс] – Режим доступу до ресурсу: https://gustavwillig.medium.com/python-web-development-in-2022-which-web-frameworks-are-the-most-popular-by-github-stars-598ba5a6d5ae

21. React. The library for web and native user interfaces [Електронний ресурс] – Режим доступу до ресурсу: https://react.dev/

22. ANGULAR FEATURES [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/features

23. The Progressive JavaScript Framework [Електронний ресурс] – Режим доступу до ресурсу: https://vuejs.org/

24. Why Spring? [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/why-spring

25. Ruby. Compress the complexity of modern web apps. [Електронний ресурс] – Режим доступу до ресурсу: https://rubyonrails.org/

26. The PHP Framework for Web Artisans [Електронний ресурс] – Режим доступу до ресурсу: https://laravel.com/

27. What is Symfony [Електронний ресурс] – Режим доступу до ресурсу: https://symfony.com/what-is-symfony

28. Why Django? [Електронний ресурс] – Режим доступу до ресурсу: https://www.djangoproject.com/start/overview/

29. Welcome to Flask [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/

30. FastAPI features [Електронний ресурс] – Режим доступу до ресурсу: https://fastapi.tiangolo.com/features/

31. The Pragmatic Programmer: your journey to mastery / Dave Thomas, Andy Hunt. – 2020

32. Modeling Web Applications [Електронний ресурс] – Режим доступу до ресурсу:

https://www.researchgate.net/publication/200827803_Modeling_Web_Applications

33. Що таке Domain-Driven Design та на якому етапі варто його впроваджувати в продукт [Електронний ресурс] – Режим доступу до ресурсу: https://dou.ua/forums/topic/39874/

34. Web Application Architecture: Working, Components, Types, Trends 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://thinksys.com/development/web-application-architecture-complete-guide/

35. What is REST [Електронний ресурс] – Режим доступу до ресурсу: https://restfulapi.net/

36. model-view-controller (MVC) [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/whatis/definition/model-view-controller-MVC

37. Best Python Frameworks for Web Development in 2023 [Електронний ресурс] – Режим доступу до ресурсу: https://www.softformance.com/blog/python-web-frameworks/

38. Flask. Using async and await [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/3.0.x/async-await/

39. Django. Asynchronous support [Електронний ресурс] – Режим доступу до ресурсу: https://docs.djangoproject.com/en/4.2/topics/async/

40. Do not (over)use async with FastAPI [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@rishibajargan/do-not-over-use-async-with-fastapi-fa70bed14d9c

41. Benefits of Asynchronous APIs [Електронний ресурс] – Режим доступу до ресурсу: https://www.linkedin.com/pulse/benefits-asynchronous-apis-apiwizio/

42. Which Python Framework is fastest? [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/dhruv_rajkotia/which-python-framework-is-fastest-2fgo

43. FastAPI: Everything you need to know about the most widely used Python web framework for Machine Learning [Електронний ресурс] – Режим доступу до ресурсу: https://datascientest.com/en/fastapi-everything-you-need-to-know-about-the-most-widely-used-python-web-framework-for-machine-learning

44. An ASGI web server, for Python. [Електронний ресурс] – Режим доступу до ресурсу: https://www.uvicorn.org/

45. Run a Server Manually - Uvicorn [Електронний ресурс] – Режим доступу до ресурсу: https://fastapi.tiangolo.com/deployment/manually/

46. Deploy FastAPI on Cloud Providers [Електронний ресурс] – Режим доступу до ресурсу: https://fastapi.tiangolo.com/deployment/cloud/

47. Welcome to Deta Space [Електронний ресурс] – Режим доступу до ресурсу: https://deta.space/docs/en/

48. SQL (Relational) Databases [Електронний ресурс] – Режим доступу до ресурсу: https://fastapi.tiangolo.com/tutorial/sql-databases/

49. Why use Pydantic? [Електронний ресурс] – Режим доступу до ресурсу: https://docs.pydantic.dev/latest/why/

50. Passlib documentation [Електронний ресурс] – Режим доступу до ресурсу: https://passlib.readthedocs.io/en/stable/

51. What is HTTPS? [Електронний ресурс] – Режим доступу до ресурсу: https://farzinpashaeee.medium.com/what-is-https-fd9c732ec950

52. Authentication. The Developer Perspective [Електронний ресурс] – Режим доступу до ресурсу: https://deta.space/docs/en/build/fundamentals/the-space-runtime/authentication#the-developer-perspective

53. APIRouter class - FastAPI [Електронний ресурс] – Режим доступу до ресурсу: https://fastapi.tiangolo.com/reference/apirouter/

54. Basic Relationship Patterns [Електронний ресурс] – Режим доступу до ресурсу: https://docs.sqlalchemy.org/en/20/orm/basic_relationships.html

55. Models - Pydantic [Електронний ресурс] – Режим доступу до ресурсу:

https://docs.pydantic.dev/latest/concepts/models/

56. Logging & Tracing in Python, FastApi, OpenCensus and Azure [Електронний ресурс] – Режим доступу до ресурсу: https://dev.to/tomas223/logging-tracing-in-python-fastapi-with-opencensus-a-azure-2jcm

57. pytest: helps you write better programs [Електронний ресурс] – Режим доступу до ресурсу: https://docs.pytest.org/en/7.4.x/

58. Testing - FastAPI [Електронний ресурс] – Режим доступу до ресурсу: https://fastapi.tiangolo.com/tutorial/testing/

59. Coverage documentation [Електронний ресурс] – Режим доступу до ресурсу: https://coverage.readthedocs.io/en/7.3.2/

60. Locust. An open source load testing tool. [Електронний ресурс] – Режим доступу до ресурсу: https://locust.io/

61. Welcome to Bandit [Електронний ресурс] – Режим доступу до ресурсу: https://bandit.readthedocs.io/en/latest/

62. Source Code Analysis Tools [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/www-community/Source_Code_Analysis_Tools

ДОДАТОК А. ПОВНИЙ ОПИС API-ШЛЯХІВ ЗА СТАНДАРТОМ OPENAPI

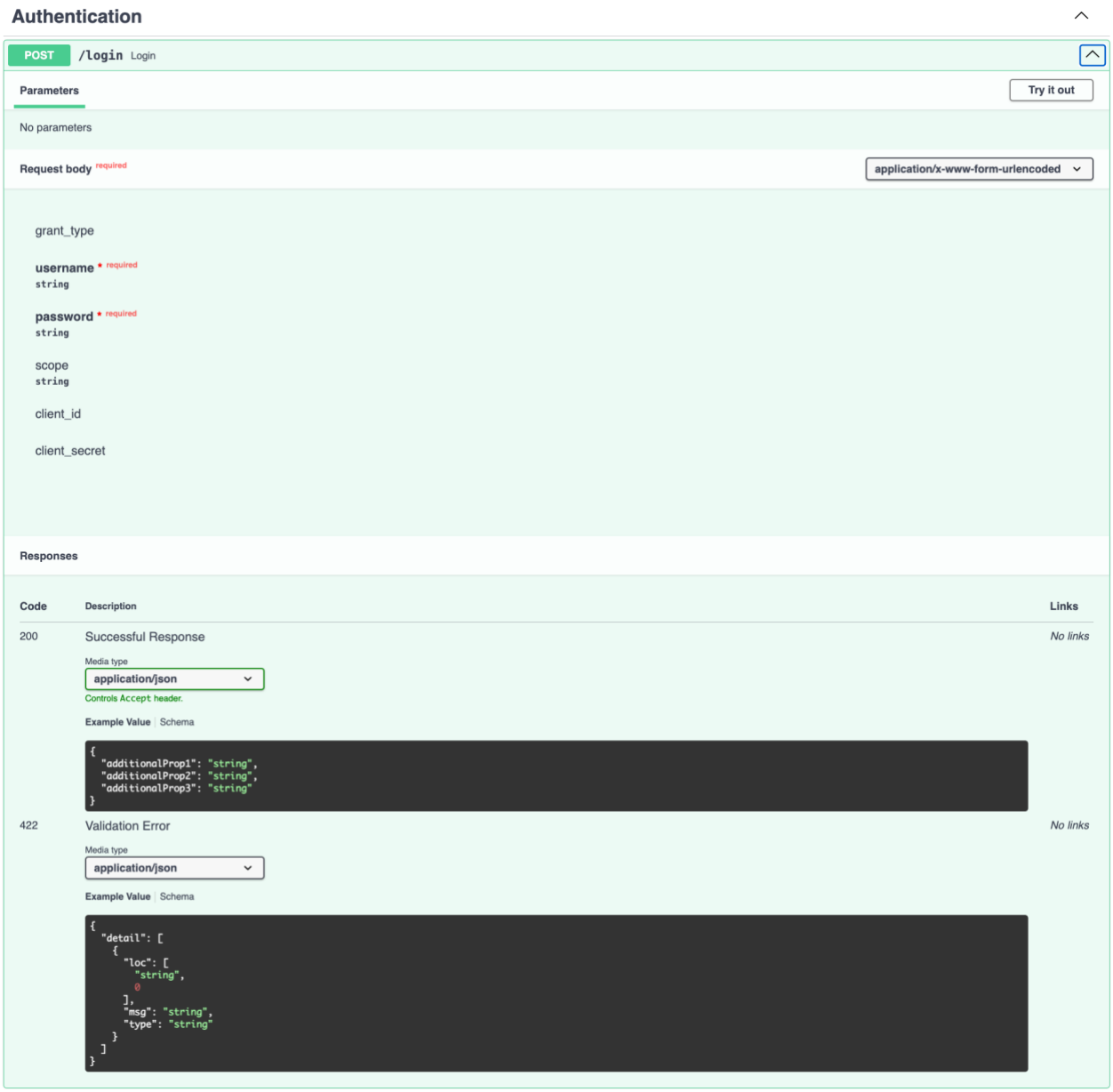


Рисунок А.1 – Authentication роутер. Шлях ‘/login/’, метод POST

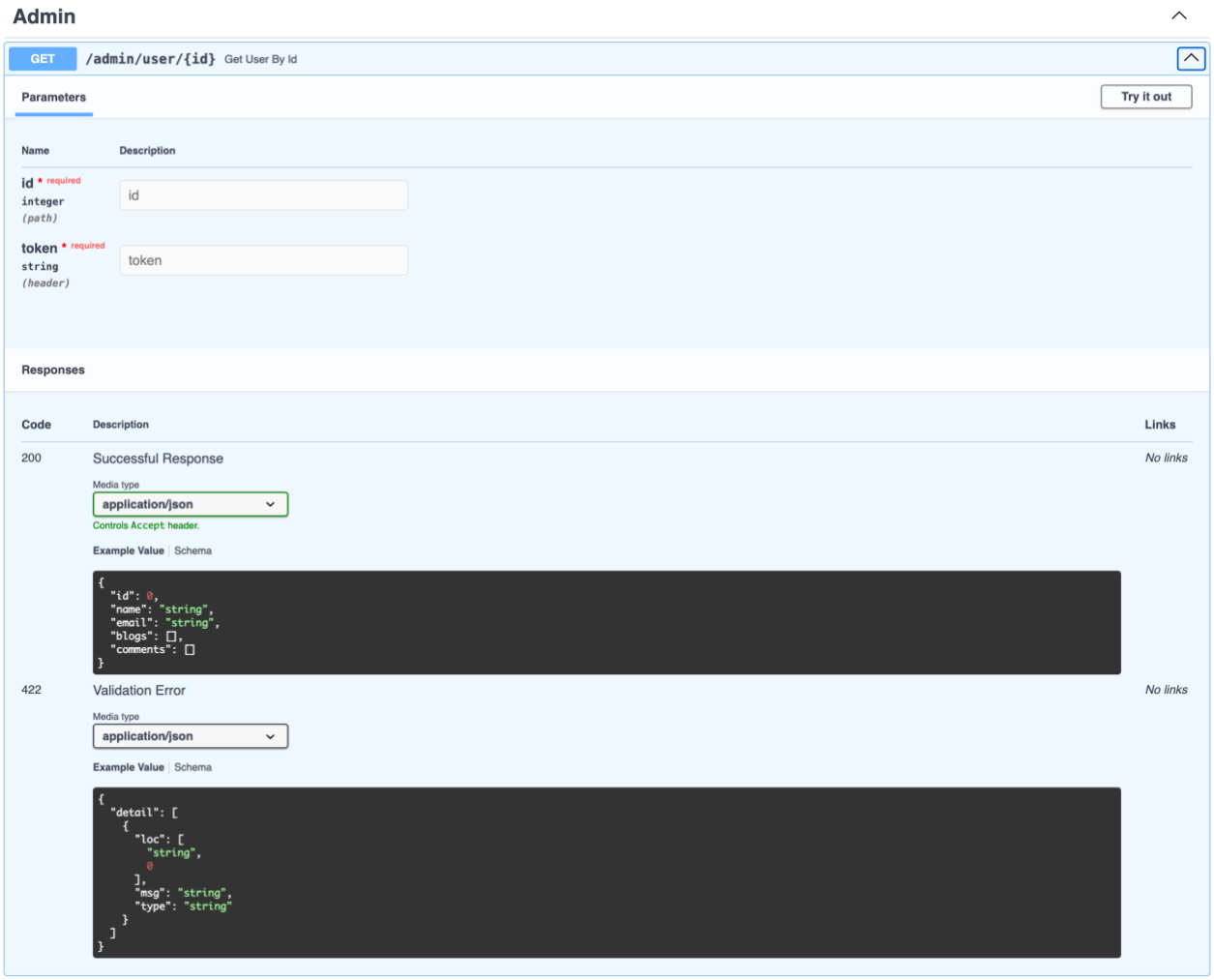


Рисунок А.2 – Admin роутер. Шлях ‘/admin/user/{id}’, метод GET

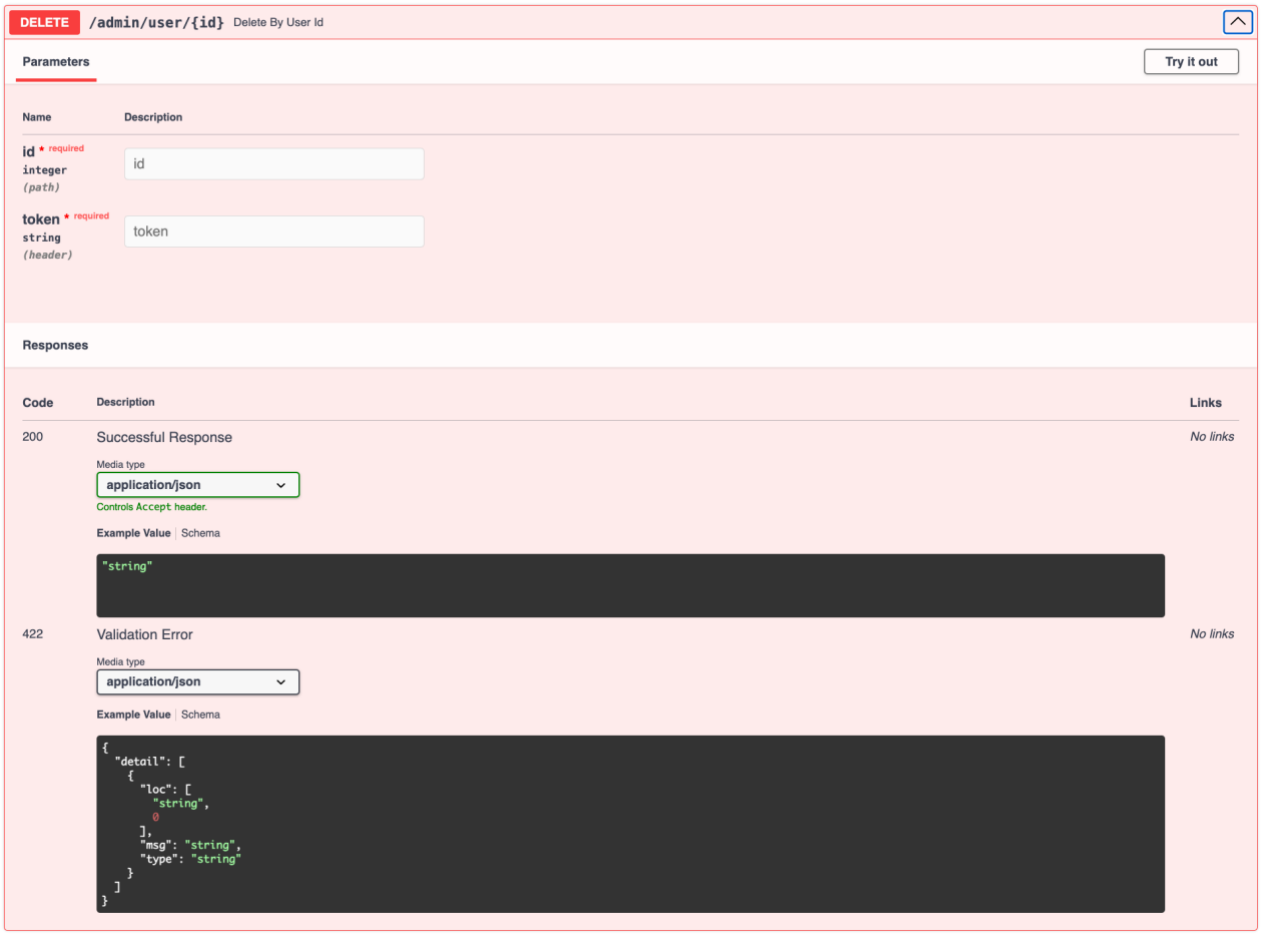


Рисунок А.3 – Admin роутер. Шлях ‘/admin/user/{id}’, метод DELETE

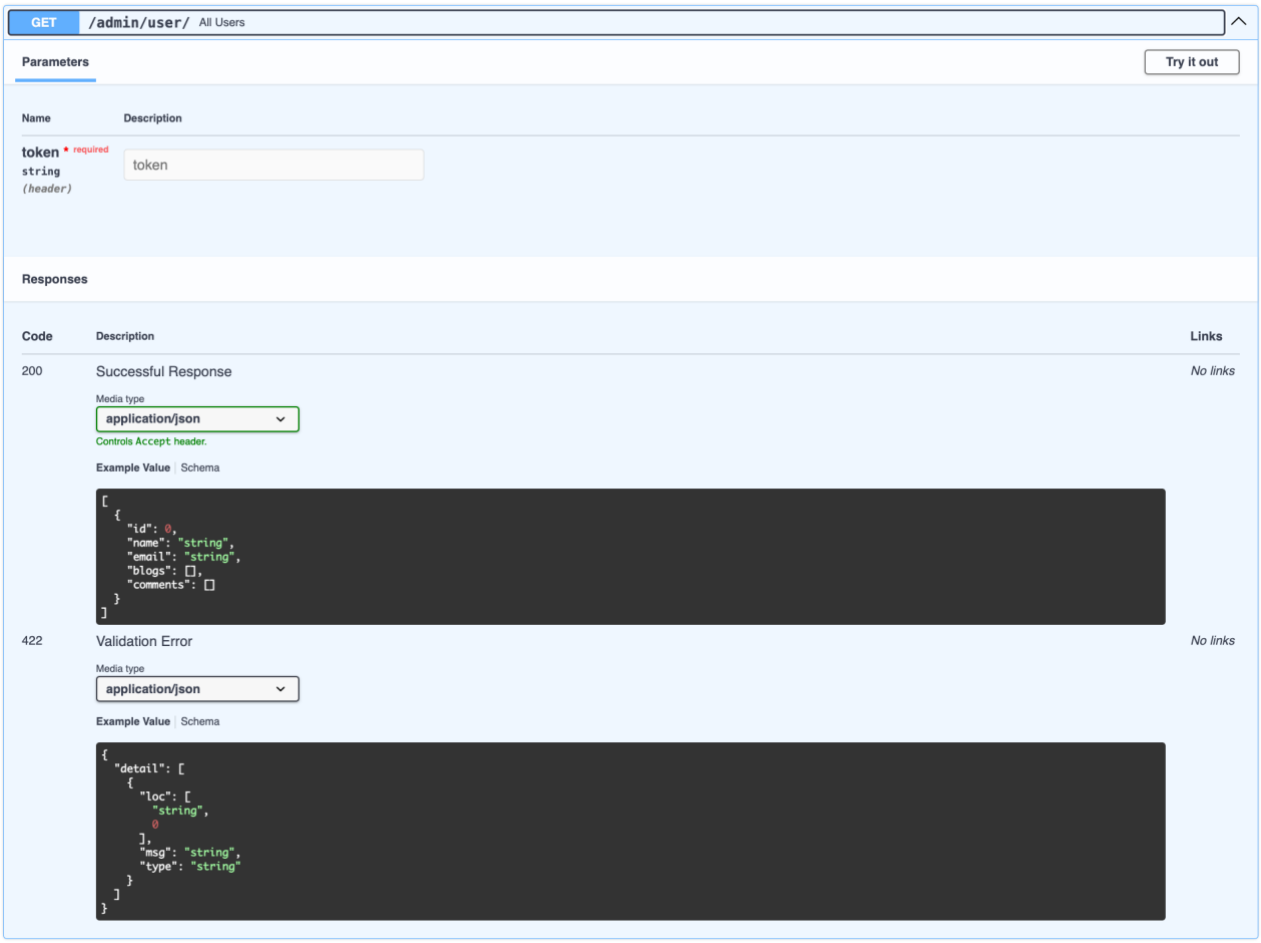


Рисунок А.4 – Admin роутер. Шлях ‘/admin/user/’, метод GET

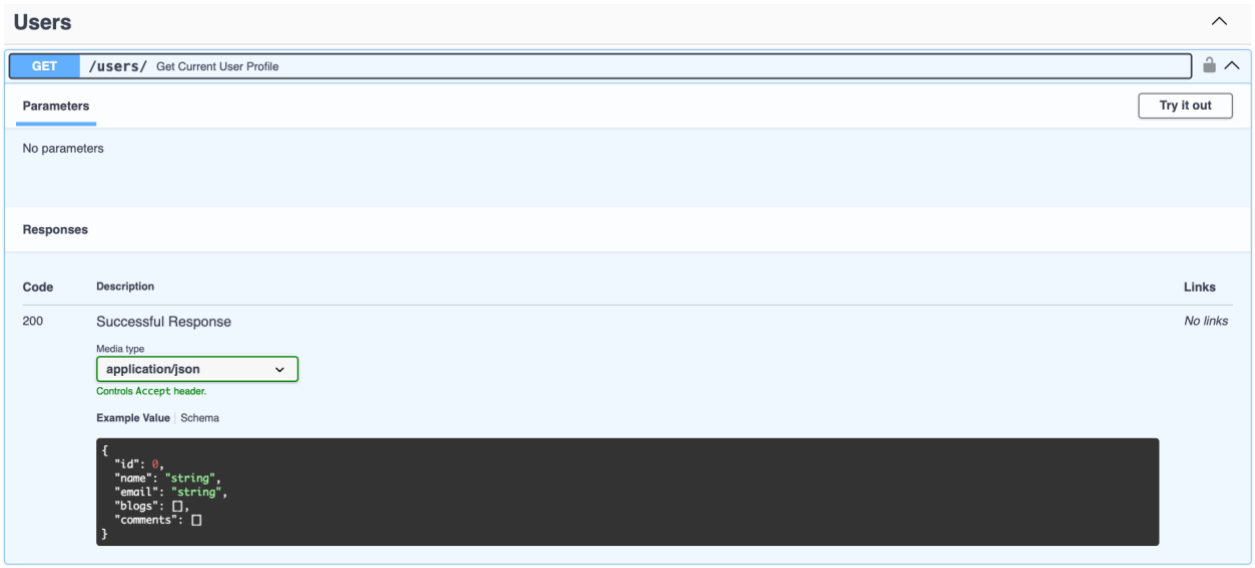


Рисунок А.5 – Users роутер. Шлях ‘/users/’, метод GET

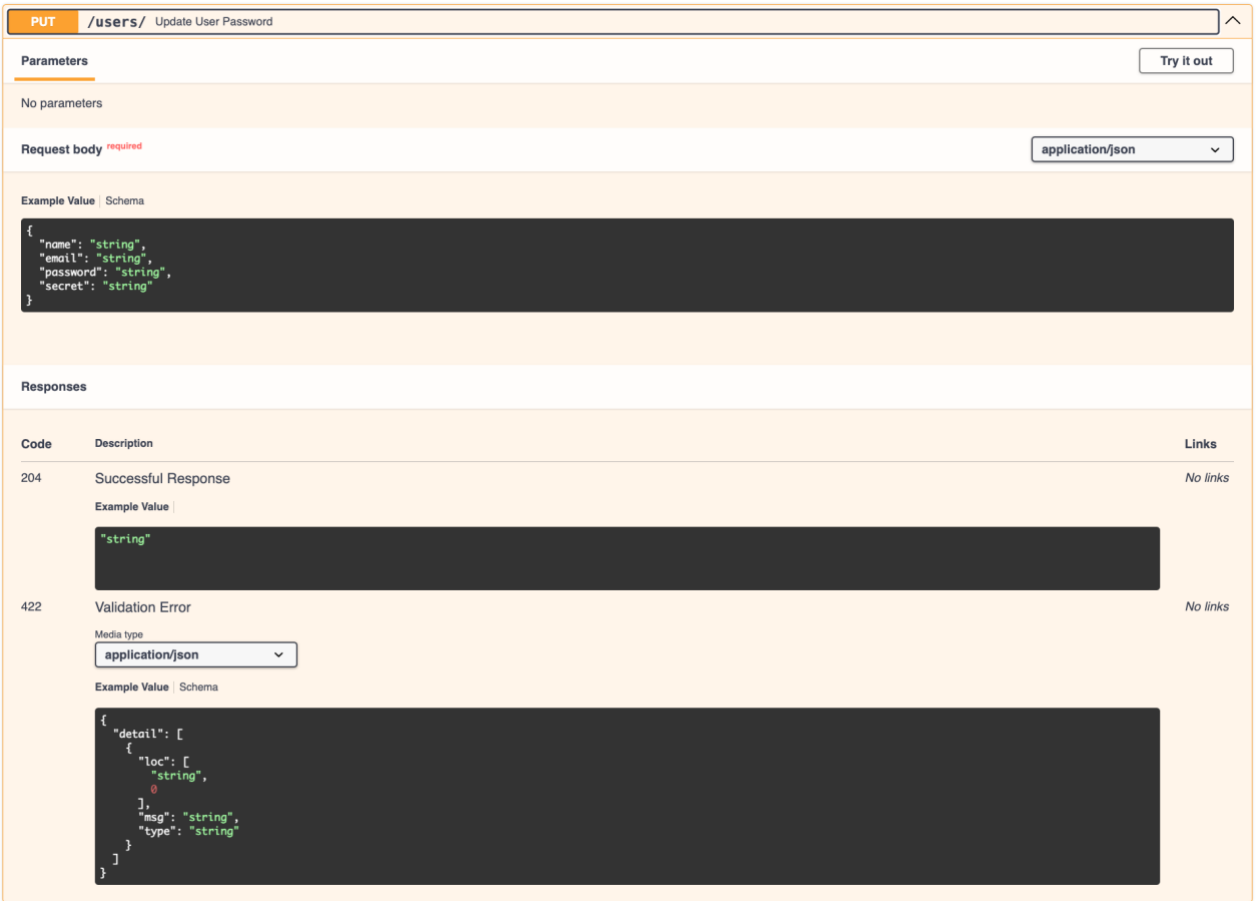


Рисунок А.6 – Users роутер. Шлях ‘/users/’, метод PUT

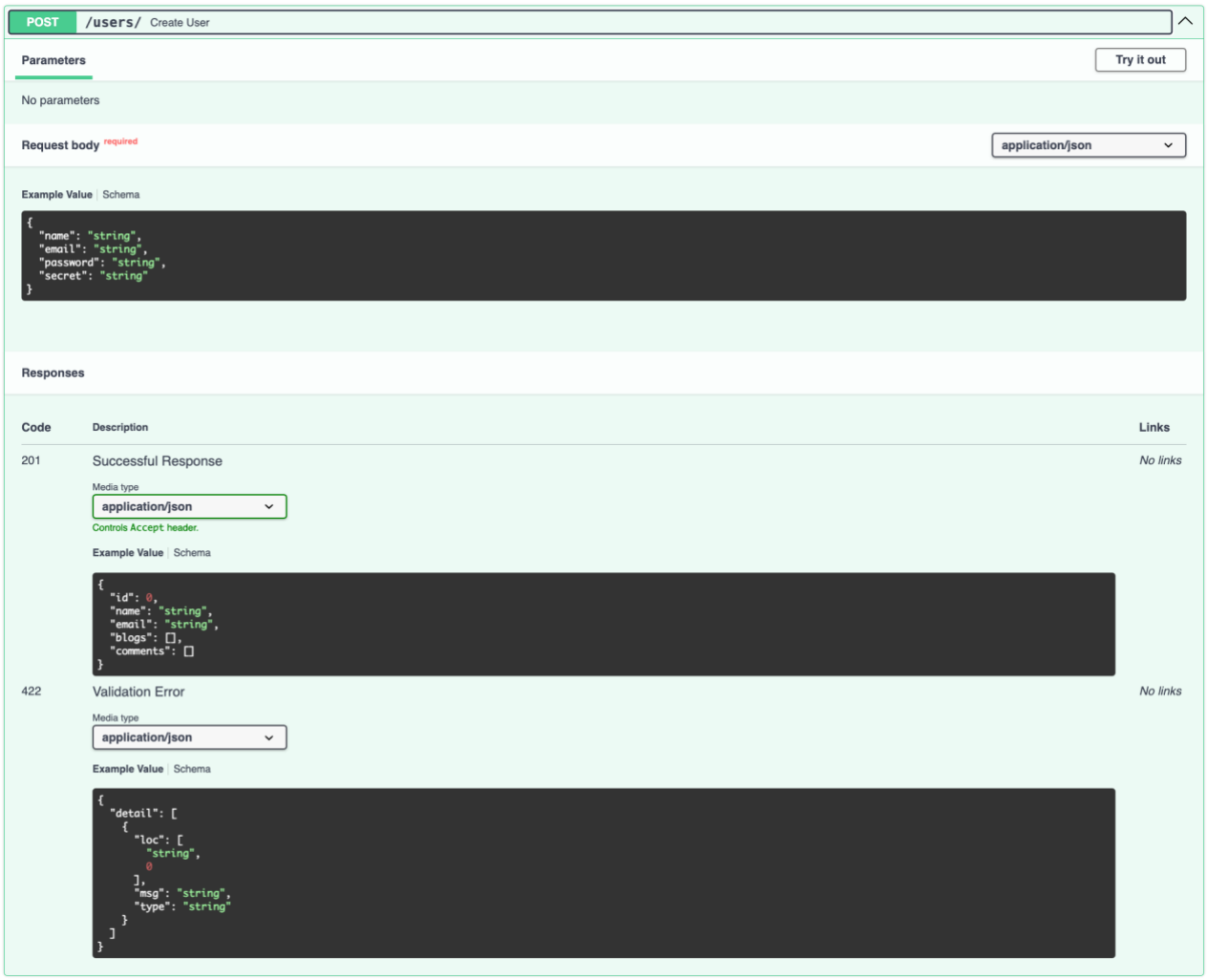


Рисунок А.7 – Users роутер. Шлях ‘/users/’, метод POST

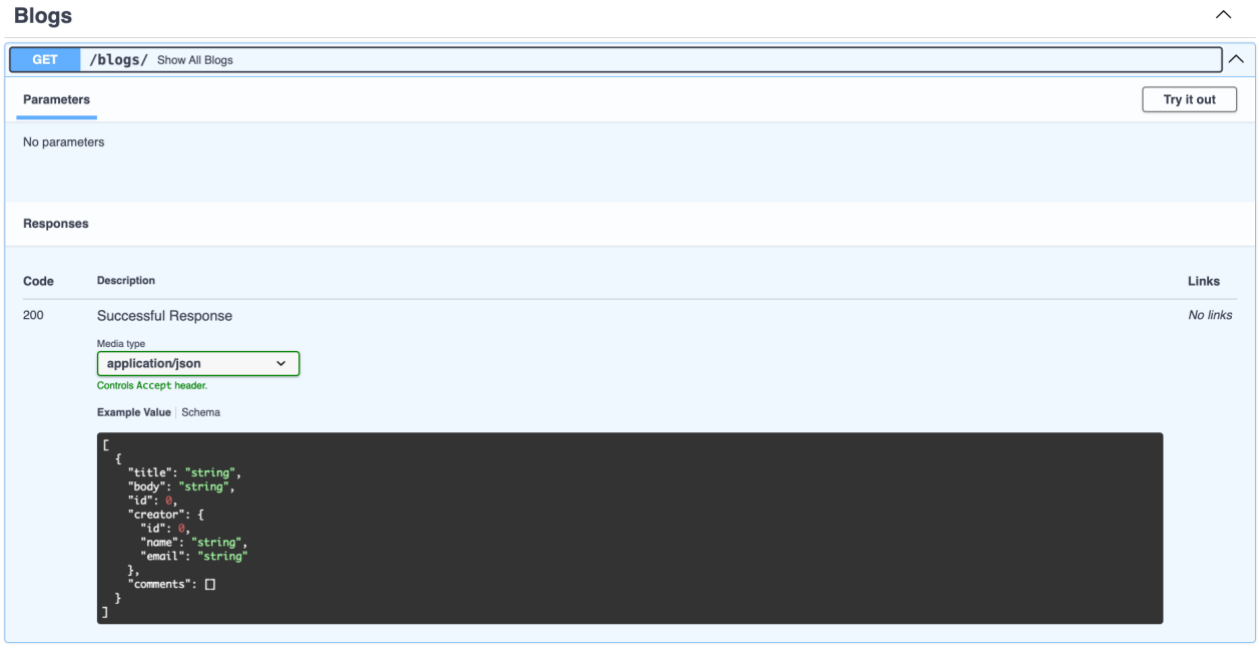


Рисунок А.8 – Blogs роутер. Шлях ‘/blogs/’, метод GET

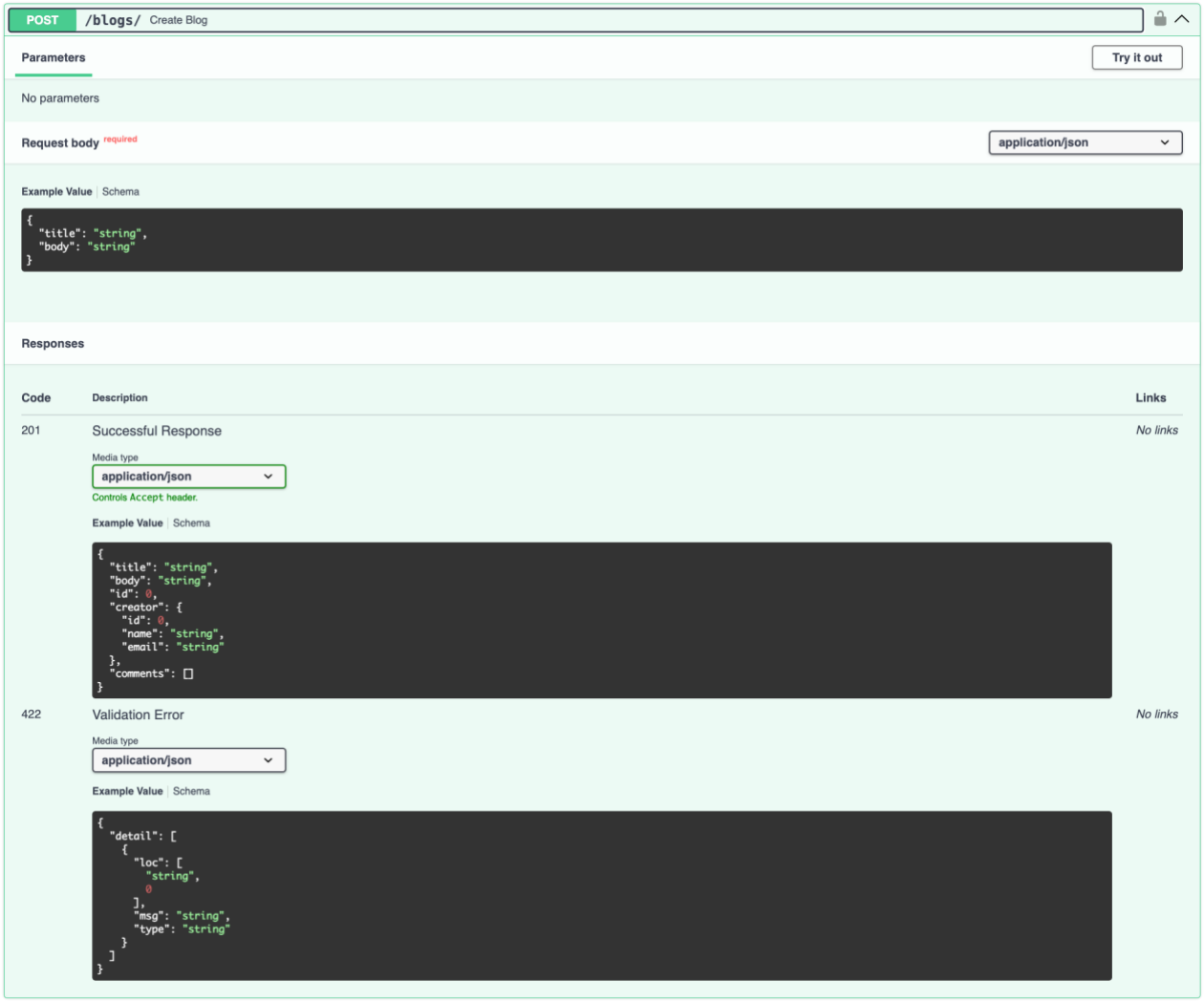


Рисунок А.9 – Blogs роутер. Шлях ‘/blogs/’, метод POST

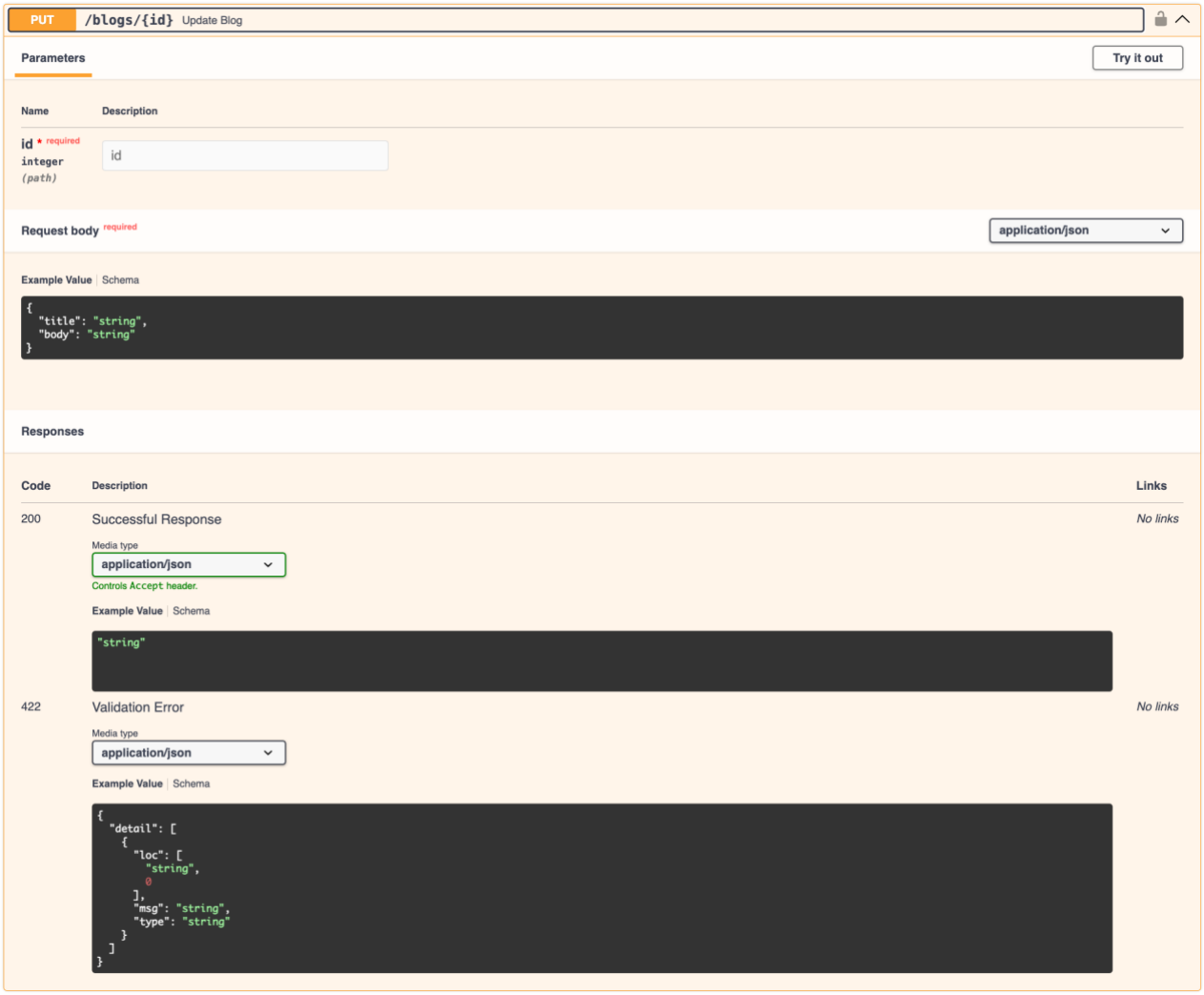


Рисунок А.10 – Blogs роутер. Шлях ‘/blogs/{id}’, метод PUT

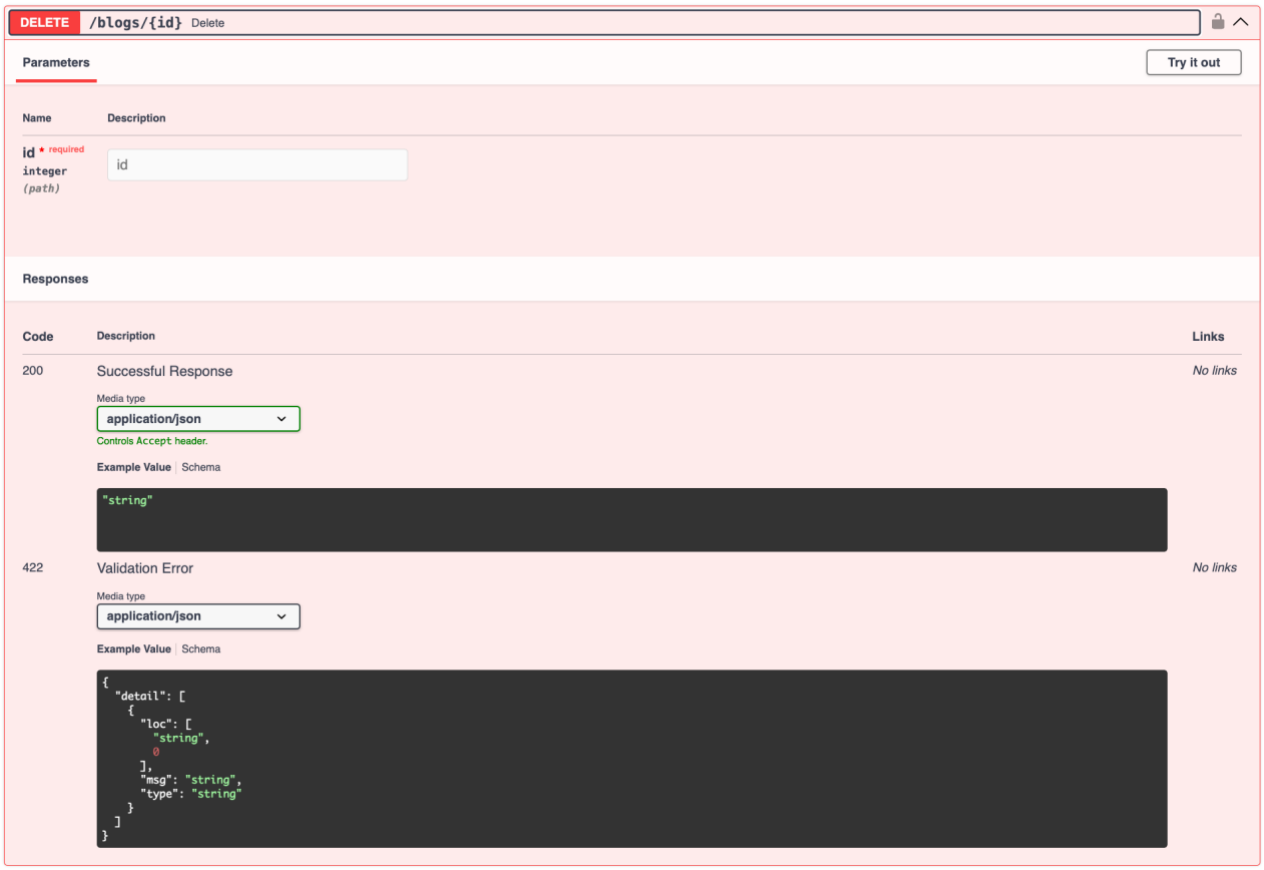


Рисунок А.11 – Blogs роутер. Шлях ‘/blogs/{id}’, метод DELETE

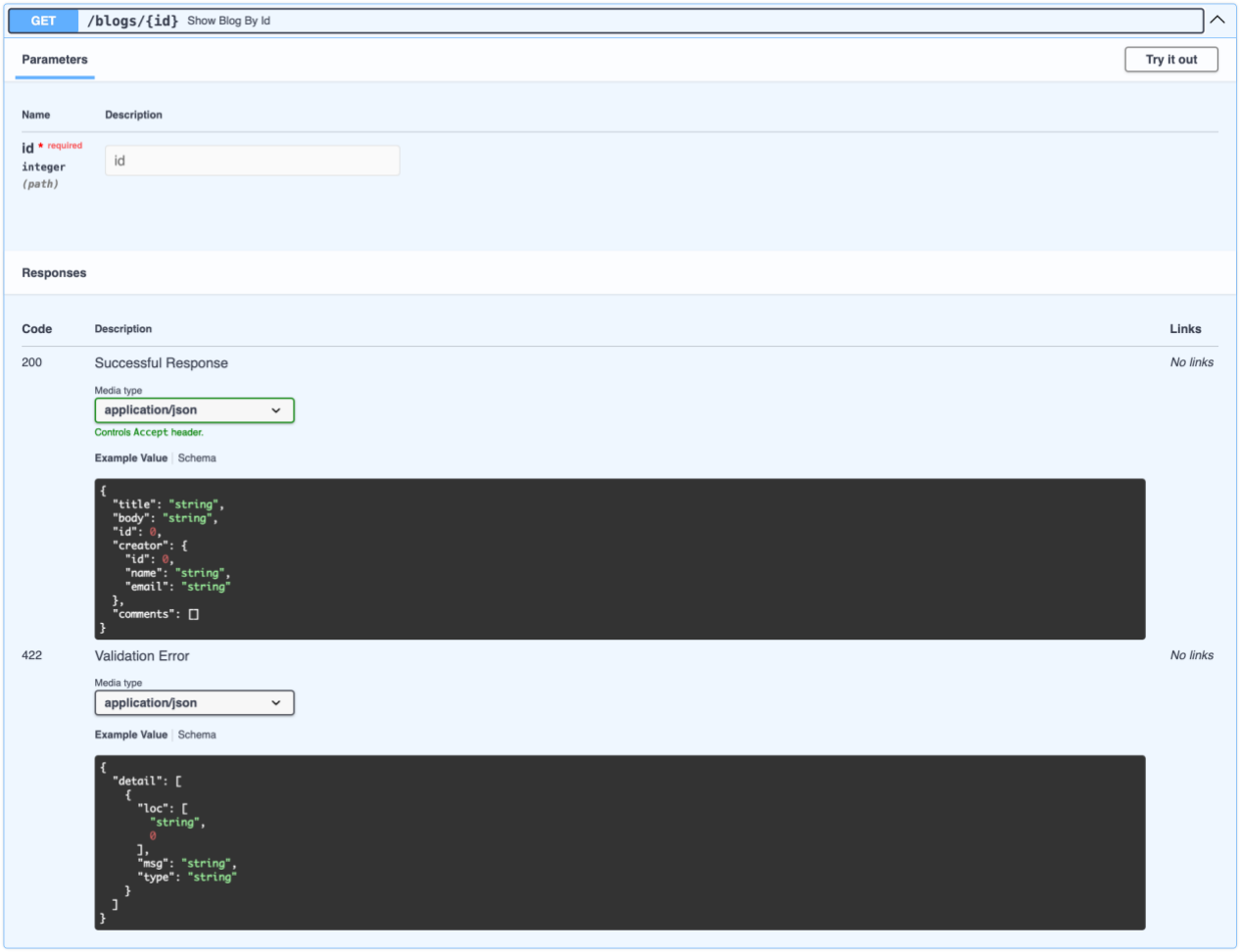


Рисунок А.12 – Blogs роутер. Шлях ‘/blogs/{id}’, метод GET

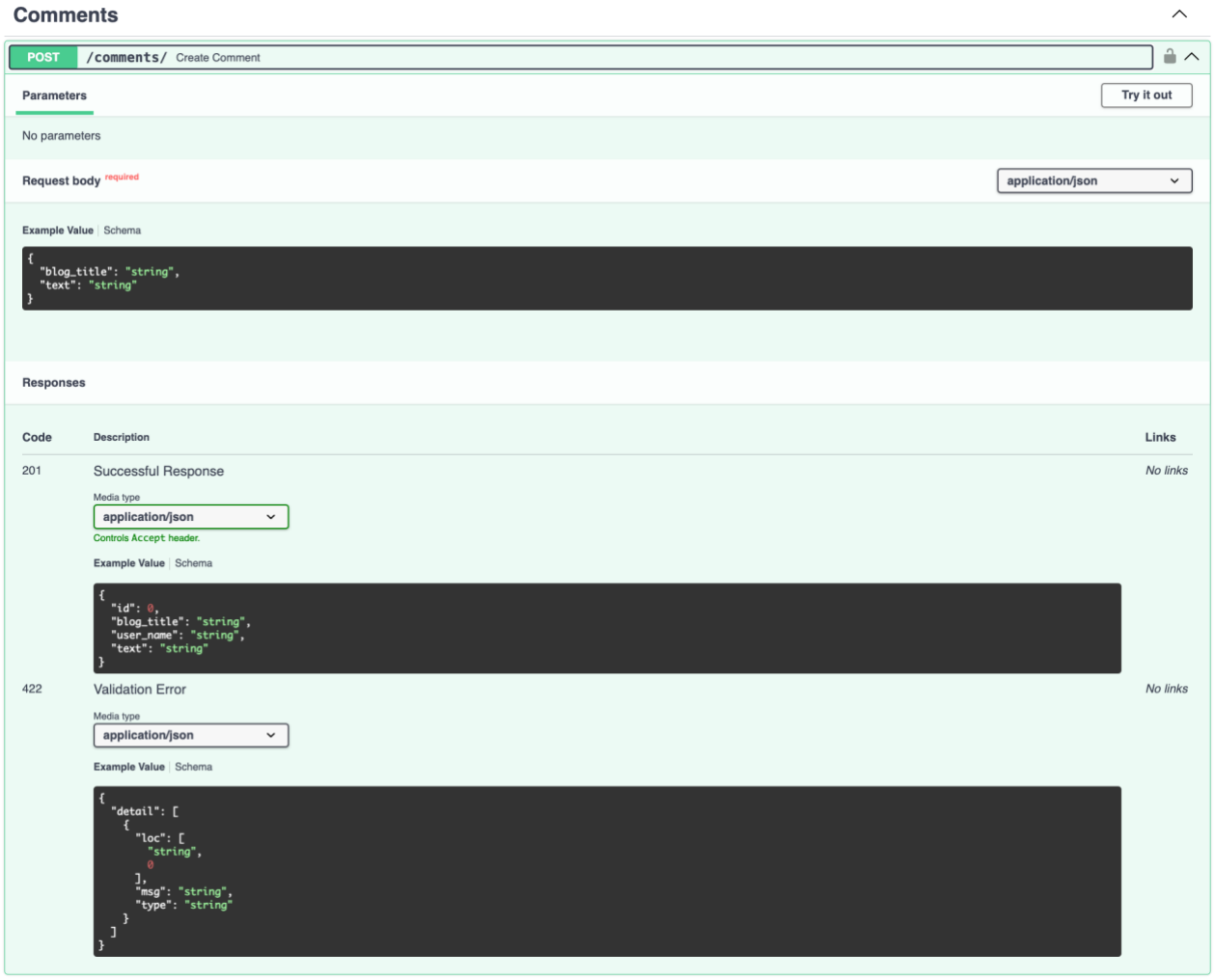


Рисунок А.13 – Comments роутер. Шлях ‘/comments/’, метод POST

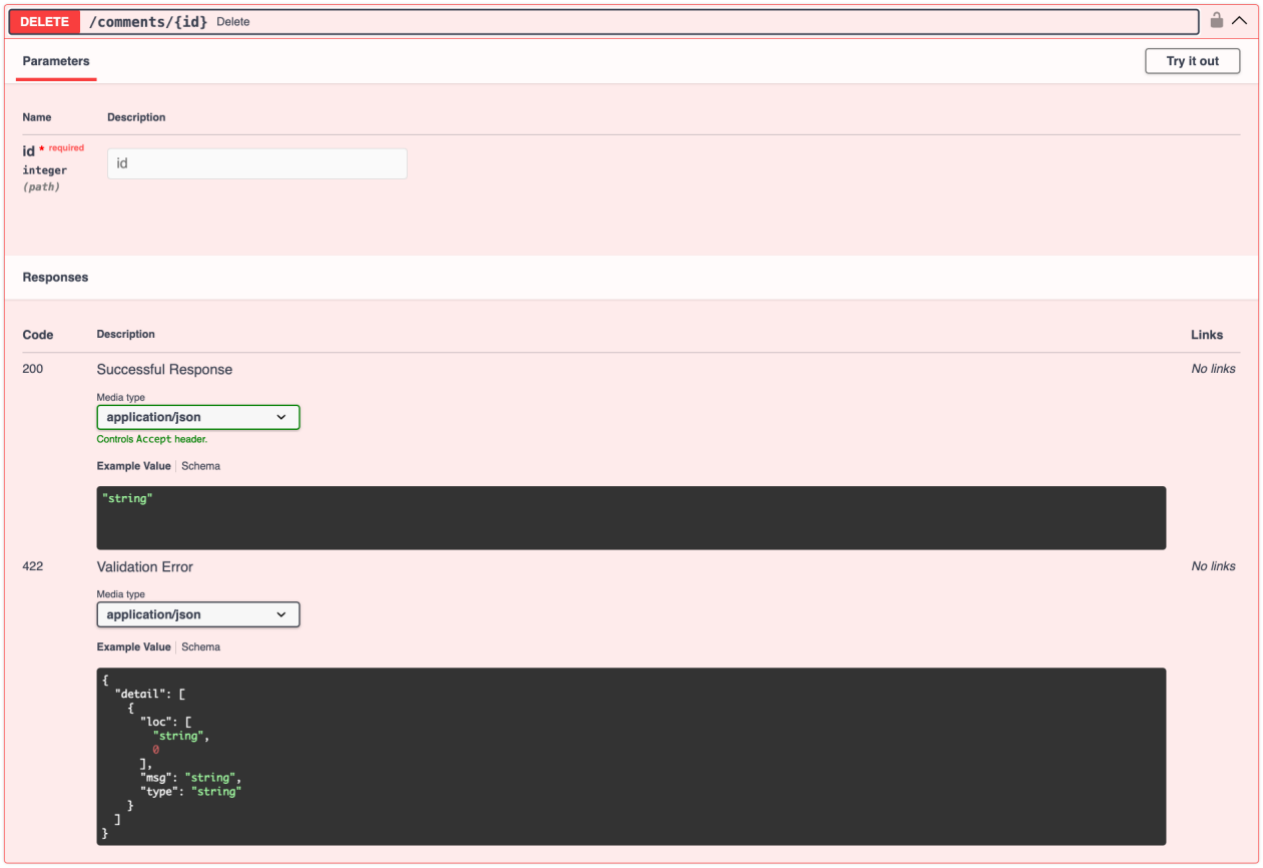


Рисунок А.14 – Comments роутер. Шлях ‘/comments/{id}’, метод DELETE

ДОДАТОК Б. КОД РЕАЛІЗАЦІЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

admin.py

Опис роутера ‘Admin’

```
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session
from starlette import status

from app.blog.infra.database import get_db
from app.blog.models import ShowUser
from app.blog.repositories import user
from app.blog.services.oauth2 import admin_token

router = APIRouter(
    prefix='/admin/user',
    tags=['Admin']
)


@router.get('/{id}', response_model=ShowUser, status_code=status.HTTP_200_OK)
def get_user_by_id(id: int, db: Session = Depends(get_db),
                   fake_valitaton: None = Depends(admin_token)) -> ShowUser:
    return ShowUser.model_validate(user.get_by_id(id, db))


@router.get('/', response_model=list[ShowUser], status_code=status.HTTP_200_OK)
def all_users(db: Session = Depends(get_db),
              fake_valitaton: None = Depends(admin_token)) -> list[ShowUser]:
```

```
    return [ShowUser.model_validate(user_db) for user_db in user.get_all(db)]


@router.delete('/{id}', status_code=status.HTTP_200_OK)
def delete_by_user_id(id: int, db: Session = Depends(get_db),
                      fake_valitaton: None = Depends(admin_token)) -> str:
    return user.delete(id, db)
```

authentication.py

Опис роутера ‘Authentication’

```
from fastapi import APIRouter, Depends, HTTPException
from fastapi.security import OAuth2PasswordRequestForm
from sqlalchemy.orm import Session
from starlette import status

from app.blog.infra import schemas
from app.blog.infra.database import get_db
from app.blog.services import oauth2
from app.blog.services.hashing import Hash

router = APIRouter(
    tags=['Authentication']
)


@router.post('/login')
def login(request: OAuth2PasswordRequestForm = Depends(), db: Session =
```

```
Depends(get_db)
        ) -> dict[str, str]:
    user = db.query(schemas.User).filter(schemas.User.name == request.username).first()
    if not user:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f'There is no username with such name: {request.username}')
    if not Hash.verify(user.password, request.password):
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED,
                            detail=f'Invalid password for {request.username}')

    access_token = oauth2.create_access_token(data={"name": user.name, "id": user.id})
    return {"access_token": access_token, "token_type": "bearer"}
```

blogs.py

Опис роутера ‘Blogs’

```
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session
from starlette import status

from app.blog import models
from app.blog.infra.database import get_db
from app.blog.infra.schemas import User
from app.blog.repositories import blog
from app.blog.services.oauth2 import get_current_user
```

```
router = APIRouter(
    prefix='/blogs',
    tags=['Blogs']
)


@router.post('/', status_code=status.HTTP_201_CREATED)
def create_blog(request: models.Blog, db: Session = Depends(get_db),
                current_user: User = Depends(get_current_user)) -> models.ShowBlog:
    return models.ShowBlog.model_validate(blog.create(request, current_user.id, db))


@router.put('/{id}', status_code=status.HTTP_200_OK)
def update_blog(id: int, request: models.Blog, db: Session = Depends(get_db),
                current_user: User = Depends(get_current_user)) -> str:
    return blog.update(id, request, current_user.id, db)


@router.delete('/{id}', status_code=status.HTTP_200_OK)
def delete(id: int, db: Session = Depends(get_db),
           current_user: User = Depends(get_current_user)) -> str:
    return blog.delete(id, current_user.id, db)


@router.get('/',                                    response_model=list[models.ShowBlog],
status_code=status.HTTP_200_OK)
def show_all_blogs(db: Session = Depends(get_db)) -> list[models.ShowBlog]:
```

```
    return [models.ShowBlog.model_validate(blog_db) for blog_db in blog.get_all(db)]


@router.get('/{id}', response_model=models.ShowBlog,
status_code=status.HTTP_200_OK)
def show_blog_by_id(id: int, db: Session = Depends(get_db)) -> models.ShowBlog:
    return models.ShowBlog.model_validate(blog.get_by_blog_id(id, db))
```

comments.py

Опис роутера ‘Comments’

```
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session
from starlette import status

from app.blog import models
from app.blog.infra.database import get_db
from app.blog.infra.schemas import User
from app.blog.repositories import comment
from app.blog.services.oauth2 import get_current_user

router = APIRouter(
    prefix='/comments',
    tags=['Comments']
)


@router.post('/', status_code=status.HTTP_201_CREATED)
```

```
def create_comment(request: models.Comment, db: Session = Depends(get_db),
                   current_user: User = Depends(get_current_user)) ->
models.ShowComment:
    return models.ShowComment.model_validate(comment.create(request,
current_user.name, db))


@router.delete('/{id}', status_code=status.HTTP_200_OK)
def delete(id: int, db: Session = Depends(get_db),
           current_user: User = Depends(get_current_user)) -> str:
    return comment.delete(id, db)
```

users.py

Опис роутера ‘Users’

```
from fastapi import APIRouter, Depends
from sqlalchemy.orm import Session
from starlette import status

from app.blog.infra.database import get_db
from app.blog.infra.schemas import User as UserDB
from app.blog.models import ShowUser, User
from app.blog.repositories import user
from app.blog.services.oauth2 import get_current_user

router = APIRouter(
    prefix='/users',
    tags=['Users']
```

```
)


@router.post('/', response_model=ShowUser,
status_code=status.HTTP_201_CREATED)
def create_user(request: User, db: Session = Depends(get_db)) -> ShowUser:
    return ShowUser.model_validate(user.create(request, db))


@router.put('/', status_code=status.HTTP_204_NO_CONTENT)
def update_user_password(request: User, db: Session = Depends(get_db)) -> None:
    user.reset_password(request, db)


@router.get('/', response_model=ShowUser, status_code=status.HTTP_200_OK)
def get_current_user_profile(db: Session = Depends(get_db),
                             current_user: UserDB = Depends(get_current_user)) -> ShowUser:
    return ShowUser.model_validate(user.get_by_id(current_user.id, db))
```

models.py

Опис опис моделей що використовуються для запитів до API-шляхів та відповідей на ці запити

```
from typing import List, Optional


from pydantic import BaseModel
```

```
class BaseBlog(BaseModel):
    title: str
    body: str


class Blog(BaseBlog):
    class Config:
        from_attributes = True


class User(BaseModel):
    name: str
    email: str
    password: str
    secret: Optional[str] = None


class ShowUserInBlog(BaseModel):
    id: int
    name: str
    email: str

    class Config:
        from_attributes = True


class Comment(BaseModel):
    blog_title: str
```

```
    text: str


class ShowComment(BaseModel):
    id: int
    blog_title: str
    user_name: str
    text: str

    class Config:
        from_attributes = True


class ShowUser(ShowUserInBlog):
    blogs: List[Blog] = []
    comments: List[ShowComment] = []


class ShowBlog(BaseBlog):
    id: int
    creator: ShowUserInBlog
    comments: List[ShowComment] = []

    class Config:
        from_attributes = True
```

schemas.py

Опис опис моделей що описують схему зберігання даних в базі даних

```
from sqlalchemy import Column, ForeignKey, Integer, String
from sqlalchemy.orm import relationship

from .database import Base


class Blog(Base):
    __tablename__ = 'blogs'
    id = Column(Integer, primary_key=True, index=True)
    title = Column(String, unique=True, nullable=False)
    body = Column(String, nullable=False)
    user_id = Column(Integer, ForeignKey('users.id', ondelete='CASCADE'))
    creator = relationship('User', back_populates='blogs')
    comments = relationship('Comment')


class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, unique=True, nullable=False)
    email = Column(String, nullable=False)
    password = Column(String, nullable=False)
    secret = Column(String, nullable=True)
    blogs = relationship('Blog', back_populates='creator')
    comments = relationship('Comment')
```

```
class Comment(Base):
    __tablename__ = 'comments'
    id = Column(Integer, primary_key=True, index=True)
    text = Column(String, nullable=False)
    user_name = Column(String, ForeignKey('users.name', ondelete='CASCADE'))
    blog_title = Column(String, ForeignKey('blogs.title', ondelete='CASCADE'))
    creator = relationship('User', back_populates='comments')
    blog = relationship('Blog', back_populates='comments')
```