Міністерство освіти і науки України
Державний університет «Одеська Політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра системного програмного забезпечення

Ткаченко Олександр Дмитрович
студент групи АС-161

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Приватна соціальна мережа для гейміфікації процесу навчання співробітників компаній

Підтема 2. *Серверна частина*

Спеціальність:
121 Інженерія програмного забезпечення

Освітня програма:
Інженерія програмного забезпечення

Керівник:
Писаренко Катерина Олександрівна,
канд. техн. наук, доцент

Одеса – 2021

ЗМІСТ

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ 4

ВСТУП 7

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ 9

1.1 Аналіз предметної області 9

1.2 Аналіз існуючих аналогів 10

Висновок 11

2 ОПИС АЛГОРИТМУ РОБОТИ СЕРВЕРНОЇ ЧАСТИНИ 12

2.1 Архітектурний стиль 12

2.2 Взаємодія клієнта та сервера 13

2.3 Діаграма станів 14

2.4 Рівень та очки навичок користувача 15

3 СПЕЦИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ 17

3.1 Діаграма варіантів використання 17

3.2 Сценарії варіантів використання 18

3.3 Нефункціональні вимоги 27

Висновок 28

4 ПРОЕКТУВАННЯ СИСТЕМИ 29

4.1 Архітектура програмного продукту 29

4.2 Реляційна модель бази даних 30

4.3 Діаграма програмних класів 34

4.4 Діаграми послідовності 35

Висновок 37

5 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ 38

5.1 Огляд використовуваних технологій 38

5.5 Розгортання проекту 38

5.6 Огляд структури проекту 43

Висновок 46

6 ТЕСТУВАННЯ 47

6.1 Тестування методу авторізації 49

6.2 Тестування методу отримання персони за ідентифікатором 51

6.3 Тестування методу додавання нової персони 53

6.4 Тестування методу видалення персони 57

6.5 Тестування методів створення, видалення та отримання навичок 59

6.6 Тестування методів додавання тестів 64
6.7 Тестування методів перегляду та видалення тестів 68
6.8 Тестування методу проходження тесту 71
Висновок 73
ВИСНОВОК 74
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 76
ДОДАТОК А. ЛІСТІНГ 77

Міністерство освіти і науки України

Державний університет «Одеська Політехніка»

Навчально-науковий інститут комп'ютерних систем

Кафедра системного програмного забезпечення

Рівень вищої освіти: другий (магістерський)

Спеціальність: 121 – Інженерія програмного забезпечення

Спеціалізація: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Любченко В.В.

«___» _____ 20___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ткаченка Олександра Дмитровича, група АС-161

1. Тема роботи: *Приватна соціальна мережа для гейміфікації процесу навчання співробітників компаній.*

Підтема: *Серверна частина*

Керівник роботи: *Писаренко Катерина Олександрівна, канд. техн. наук, доцент*

Затверджені наказом ректора від «25» жовтня 2021 р. № 374-в

2. Зміст роботи: завдання на кваліфікаційну роботу, аналіз предметної області та існуючих рішень, опис алгоритму роботи серверної частини, специфікація вимог до системи, проектування системи, програмна реалізація системи, тестування.

3. Перелік ілюстративного матеріалу:

Згідно зі слайдами презентації

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

5. Дата видачі завдання «___» _____ 20___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	Аналіз вимог до веб-застосування	1.09.2021-02.09.2021	Виконано
2	План виконання проекту	03.09.2021-07.09.2021	Виконано
3	Аналіз існуючих аналогів	08.09.2021-09.09.2021	Виконано
4	Проектування веб-застосування	10.09.2021-24.09.2021	Виконано
5	Програмна реалізація веб-застосування	24.09.2021- 29.10.2021	Виконано
6	Тестування веб-застосування	01.10.2021-15.11.2021	Виконано
7	Написання пояснювальної записки	15.11.2021-27.11.2021	Виконано

Здобувач вищої освіти _____ *О. Д. Ткаченко*

Керівник роботи _____ *К. О. Писаренко*

АНОТАЦІЯ

Робота присвячена розробці серверної частини приватної соціальної мережі для гейміфікації процесу розвитку співробітників компанії.

Метою роботи є підвищення кваліфікації співробітників шляхом додавання до процесу навчання формату гри.

Як результат розроблена серверна частина для соціальної мережі з такими елементами гри як рівень користувача, очки навичок, проходження тестів.

У роботі використовується мова програмування Python, фреймворк для розробки веб-застосувань Flask, система управління базами даних PostgreSQL.

Ключові слова: соціальна мережа, Python, веб-застосування, HTTP, сервер.

ABSTRACT

The work is devoted to the development of the server part of a private social network for gamification of the development process of the company's employees.

The purpose of the work is to improve the skills of employees by adding a game format to the learning process.

As a result, a server part was developed for a social network with such elements of the game as user level, skill points, passing tests.

The work uses the Python programming language, a framework for developing Flask web applications, and the PostgreSQL database management system.

Keywords: social network, Python, web application, HTTP, server.

ВСТУП

В березні 2020 року був введений карантин на всій території України. Карантин передбачає закриття навчальних закладів, заборону масових зборів понад 200 осіб та закриття авіасполучення з деякими країнами. Було повідомлено, що буде закрито більшу частину прикордонних переходів - 170 із 219. Тому більшість підприємств були змушені на перехід до дистанційного режиму роботи.

В дистанційному режимі роботи є свої переваги та недоліки. Наприклад, до переваг можна віднести:

- Робота у своєму ритмі.
- Економія часу та грошей. Не треба витрачати час для підготовки к роботі та гроші на шлях до роботи.
- Для бізнесу це зниження затрат.

До недоліків слід віднести наступни пункти:

- Проблеми самоконтролю.
- Відсутність кар’єрного зростання.

Також роботодавцям складно контролювати процеси, які пов’язані з розвитком своїх співробітників. Працюючи з дому неможливо показати свій розвиток у тій чи іншій сфері роботи.

Для вирішення виявлених недоліків, які були описані вище, було прийнято рішення розробити соціальну мережу для компанії, за допомогою якої можна буде відстежити процеси розвитку, для підвищення процесу мобільності співробітників між проектами в рамках компанії. Для того, щоб процес навчання був більш цікавий, треба гейміфікавати його, шляхом додавання механік гри до соціальної мережі [1].

Метою роботи є підвищення кваліфікації співробітників шляхом додавання до процесу навчання формату гри.

У першому розділі роботи проводиться аналіз предметної області та існуючих аналогів.

Другий розділ містить опис алгоритму роботи серверної частини, а також опис зв’язку серверної та клієнтської частини.

У третьому розділі була сформульована специфікація функціональних та нефункціональних вимог до розробляємої системи.

Четвертий розділ містить проектування серверної частини веб-застосування соціальної мережі: розглянута архітектура системи, побудована реляційна модель бази даних, продемонстровані програмні класи та діаграми послідовності.

У п’ятому розділі наведена програмна реалізація системи та приклад розгортання проекту.

Шостий розділ містить тестування розробленої системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ

В рамках даної роботи поставлено наступне завдання: користувачі зможуть додавати в свою анкету навички (наприклад, коли освоїли щось нове по своїй спеціальності або іншій області) і ділитися ними з колегами, проходити тести, отримувати за це очки досвіду, піднімати рівні; також один користувач зможе викликати іншого на поєдинок, переможцем якого буде той, хто дасть більше правильних відповідей.

1.1 Аналіз предметної області

Дане веб-застосування розробляється для компаній, які зацікавлені в моніторингу і мотивації рівня розвитку своїх співробітників. Для великих компаній, де працює велика кількість людей, в епоху пандемії і віддаленої роботи стало важко перевіряти якість кваліфікації своїх співробітників.

Для того, щоб співробітники швидше освоювали цю соціальну мережу, було прийнято рішення гейміфіцировать деякою мірою використовувати даний веб-ресурсу. За кожен успішно освоєний навичок, співробітник отримає очки досвіду, кількість яких залежить від складності та ступеня освоєння того чи іншого навичка. При певній кількості очок навичок, користувач отримує новий рівень. Чим вище рівень, тим більше очок потрібно буде добути для переходу на наступний.

Два користувача можуть пройти спільно тест-вікторину, за підсумками якого переможе той, хто дасть більше правильних відповідей. Переможець отримує додаткові очки навичок.

Буде доступна рейтингова система по навичкам. Сортуватися вона буде за якістю освоєння того чи іншого навичка. Робиться це для того, щоб можна було оперативно замінити співробітника, що звільнено.

1.2 Аналіз існуючих аналогів

Було проведено аналіз аналогів існуючих рішень для корпоративних соціальних мереж.

Convo

Convo - сервіс для організації централізованої соціальної мережі.

Переваги:

- обмін повідомленнями;
- профілі співробітників;
- обмін файлами;
- фільтри та пошук.

Недоліки:

- немає повідомлень;
- немає можливості керування групами;
- немає системи мотивації співробітників.

DaOffice

DaOffice - корпоративна соціальна мережа, яку можна розгорнути в будь-якій компанії для збору всієї необхідної інформації про співробітників, їх спільної роботи, а також швидкого пошуку контактів колег, отримання відповідей на питання, експертних даниху.

Переваги:

- повідомлення;
- профілі співробітників;
- обмін файлами;
- фільтрі та пошук;
- система мотивації співробітників.

Недоліки:

- немає можливості керування групами.

Yammer

Yammer - це корпоративна соціальна мережа, яка допомагає співробітникам вашої компанії відкрито співпрацювати і взаємодіяти, в тому числі з партнерами.

Переваги:

- обмін повідомленнями;
- профілі співробітників;
- обмін файлами;
- фільтри та пошук.

Недоліки:

- немає системи мотивації співробітників.

Висновок

Базуючись на аналізі предметної області та можливостях аналогів, було вирішено розробити систему, яка буде вирішувати тільки необхідні проблеми, не буде містити зайвих функцій, та простий інтерфейс, а також наступні функції:

- обмін повідомленнями;
- профілі співробітників;
- обмін файлами;
- фільтрі та пошук;
- система мотивації співробітників.

2 ОПИС АЛГОРИТМУ РОБОТИ СЕРВЕРНОЇ ЧАСТИНИ

2.1 Архітектурний стиль

Для реалізації цього проекту був обран архітектурний стиль REST [2].

REST - це архітектурний стиль взаємодії компонентів розподілених в мережі. Якщо сказати іншими словами - це набір правил про те, як программисту краще організувати написання серверної частини, щоб усі системи легко обмінювались даними та прилад можна було масштабувати. Вперше цей термін був введений Роєм Філдінгом в 2000 році.

На рисунку 2.1 продемонстрована схема роботи цього стилю: клієнт запит серверу, сервер його обробляє та повертає відповідь клієнту.

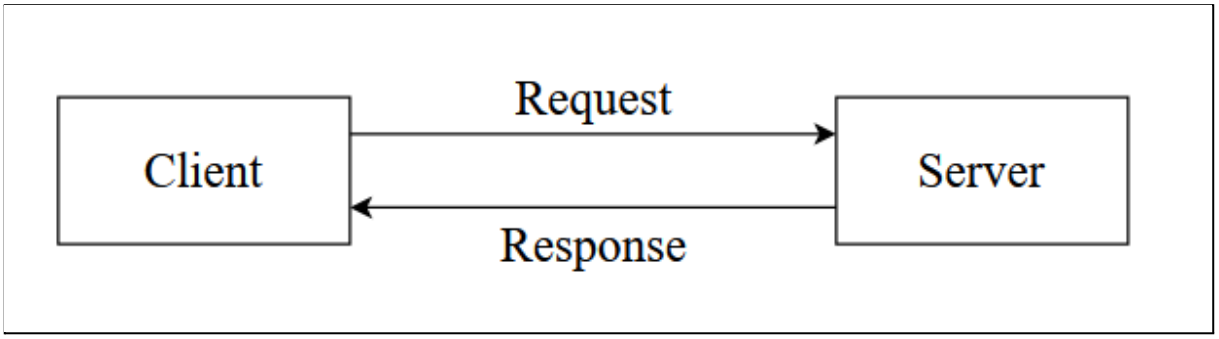


Рисунок 2.1 - схема роботи REST

Для обміну даних в цьому проєкті був обран JSON-формат.

JSON - текстовий формат обміну даними. Як і багато інших текстових форматів, JSON легко читається людьми. JSON-текст є однією з двох структур:

- набір пар “ключ: значення” (структура словник);
- упорядкований набір значень (структура лист або масив).

Для зручного написання клієнтської частини, будуть використовуватись наступні HTTP-методи:

- GET - для отримання інформації;
- POST - для додавання нової інформації;
- PUT - для оновлення існуючий інформації;

- DELETE - для видалення інформації.

Використання цих методів дозволяє для одної URL-адреси виконувати різні операції.

Щоб повідомити користувача про успішність або безуспішність операції будут використані HTTP коди статусу:

- 200 - успіх;
- 204 - успіх без відповіді;
- 400 - некоректний запит;
- 401 - не авторизований;
- 404 - сторінку не знайдено;
- 500 - внутрішня помилка сервера.

Наприклад, якщо користувач хоче отримати дані неіснуючий або видалений персони, відповідь від сервера буде з кодом 404 - сторінку не знайдено. Або якщо оновить якусь інформацію профилю або тесту, отримує відповідь 204 без тіла відповіді. Якщо користувач, який не пройшов авторизацію та не отримав токен або токен вже не валідний, він отримує відповідь з 401 статус кодом.

2.2 Взаємодія клієнта та сервера

Після того, як був описаний архітектурний стиль, треба описати алгоритм взаємодії клієнтської частини та серверної. Зв’язок між двома частинами буде реалізовано через HTTP-запити з клієнтської частини на певні API серверної.

API - опис того, як комп’ютерна програма може бути пов’язана з іншою програмою. Зазвичай він включається в деякі описи протоколів Інтернету (наприклад, RFC), програмного забезпечення (фреймворків) або виклики функцій для стандартних операційних систем. Часто реалізується як окрема бібліотека програмного забезпечення або сервіс операційної системи. Використовується програмістами при написанні різних приладів.

Створення API спрощує процес програмування, абстрагуючи основну реалізацію та надаючи лише ті об'єкти чи завдання, які потрібні розробнику.

На рисунку 2.2 продемонстрована схема взаємодії клієнтської частини та серверної.

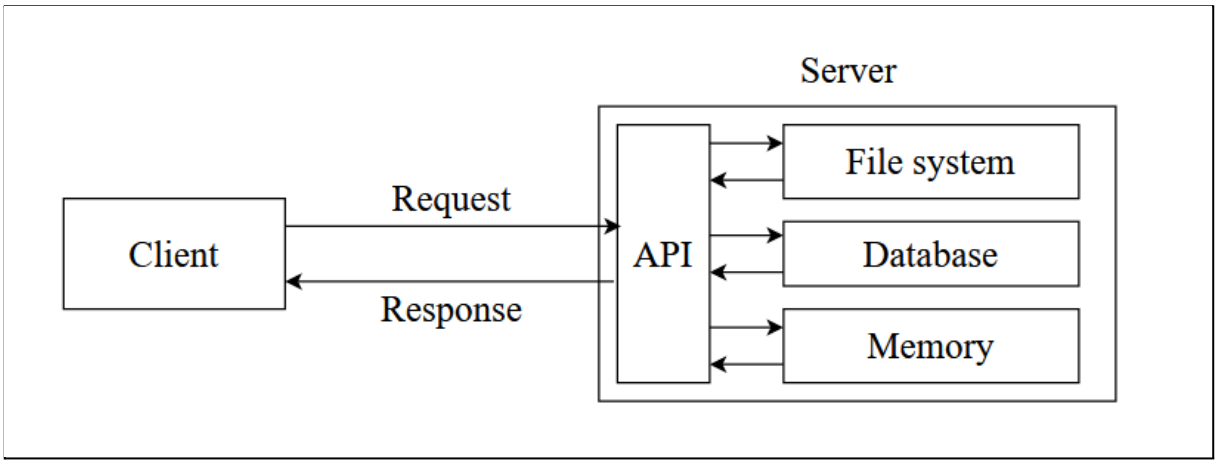


Рисунок 2.2 - Схема взаємодії клієнтської та серверної частин

В цьому випадку API - метод контролера з певною URL-адресою, на яку з клієнтської частини будуть приходити запити. Після цього будуть виконуватись запити в базу даних, файлову систему або в пам’ять компьютера.

2.3 Діаграма станів

Діаграма станів показує динамічну поведінку сутностей на основі специфікації реакції на отримання певних подій. Основна мета цієї діаграми — описати послідовність станів і переходів, які в сукупності характеризують поведінку елемента моделі протягом його життєвого циклу.

На рисунку 2.3 продемонстрована робота серверної частини.

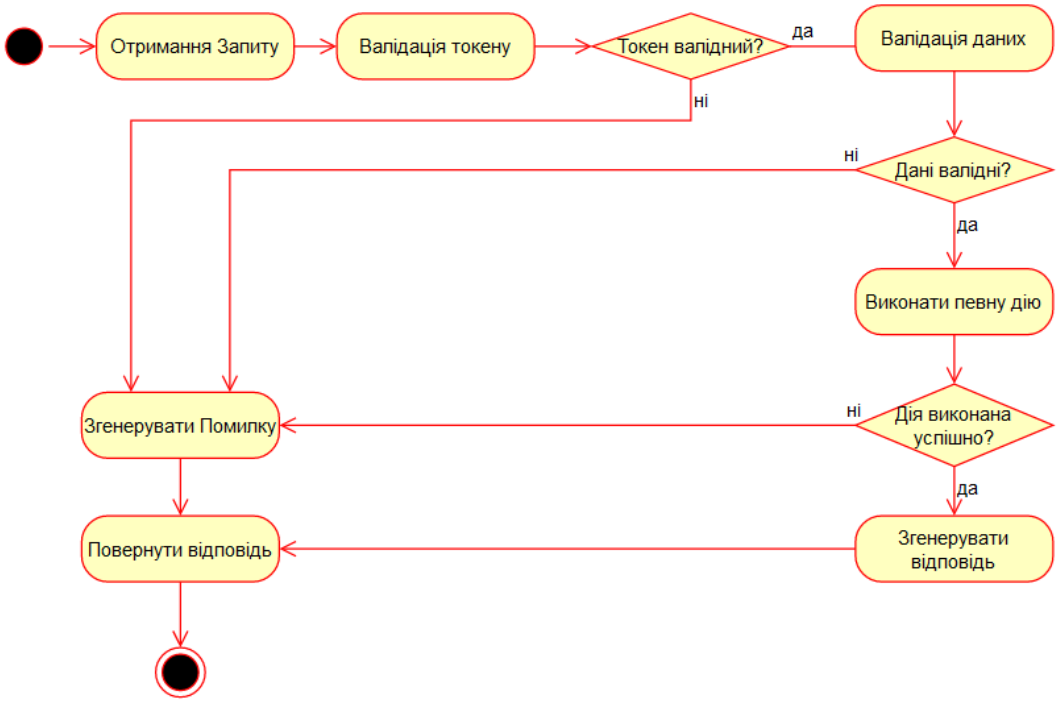


Рисунок 2.3 - Діаграма станів серверної частини

Усі запити будуть оброблятися по одному сценарію:

1. Валідація токену - якщо персона, яка надіслала запит, має не валідний токен або ця персона не існує в базі - токен не пройде валідацію. Клієнт отримує помилку.
2. Після валідації токену буде проведена валідація надісланих даних. Якщо не всі обов’язкові поля запиту були заповнені, клієнт отримує помилку.
3. Після успішних валідацій буде виконано певну даю. Якщо дія закінчилась помилкою, користувач отримує помилку.
4. Далі генерується відповідь, ті відправляється користувачу.

2.4 Рівень та очки навичок користувача

Один з засобів гейміфікувати якийсь процес - введення рейтингової системи.

Рейтинг — числовий чи порядковий захід, що вказує на значущість чи важливість певного об'єкта чи явища. Механіка рейтингу пов'язана з механікою

очок і часто з механікою рівнів користувача. Рейтинг без очок неможливий – система не зрозуміє порядок, у якому користувачі мають з'являтися у рейтингу без рівнів.

Однак треба пам'ятати, що користувачу легше не приймати участь в гейміфіцірованної системі спочатку, ніж після того як він провів в системі багато часу.

Соціальна мережа буде гейміфіцірувати процес розвитку завдяки рейтингової системи. В кожного користувача буде свій рівень та очки, яки він зможе отримати при проходженні тестів. Треба використати формулу підрахунку кількості очок за рівень, яка буде давати низькорівневим користувачам здобувати новий рівень швидше, ніж вискорівневим:

$$\frac{(lvl*(lvl+1))}{2} * const, \quad (2.1)$$

де lvl - рівень користувача,

const - коефіцієнт для підвищення кількості очок.

В таблиці 2.1 наведені результати розрахунку за формулою, з const = 20.

Таблиця 2.1 - Результати розрахунку за формулою

Рівень	Кількість очок (КО)	Разність ($КО_n$ - $КО_{n-1}$)
2	60	60
3	120	80
4	200	100
5	300	120

С результатів розрахунку бачимо, що для досягнення слідуючого рівня, треба отримати більше очок, ніж для попереднього.

3 СПЕЦИФІКАЦІЯ ВИМОГ ДО СИСТЕМИ

3.1 Діаграма варіантів використання

Один із способів опису вимог до розроблюваної системі - складання діаграми варіантів використання.

Варіанти використання є описом того, що повинна робити система. На рисунку 3.1 представлена діаграма варіантів використання, яка описує основні функції розроблюваного веб-застосування.

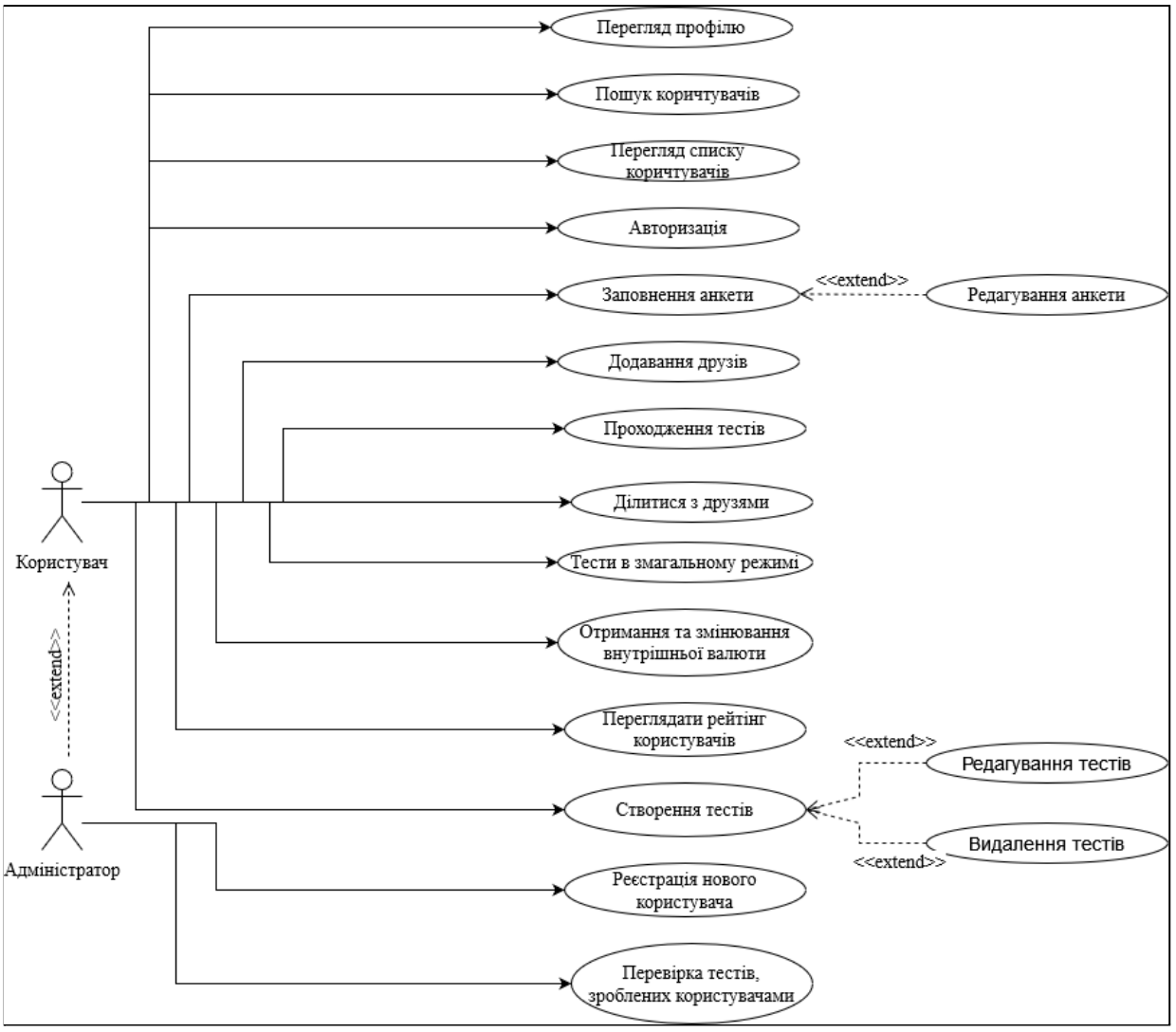


Рисунок 3.1 - Діаграма варіантів використання

Актор з роллю Користувач має наступні можливості:

- Авторизуватись
- Заповнити анкету
- Редагувати анкету
- Додавати друзів
- Проходити тести
- Ділитися результатами
- Проходити тест в змагальному режимі
- Отримувати і обмінювати внутрішню валюту сервісу

Актор з роллю Адміністратор має всі ті ж можливості, але ще і додаткові:

- Реєстрація нового користувача
- Перевірка тестів, зроблених користувачами
- Створення тестів

3.2 Сценарії варіантів використання

На діаграмі варіантів використання (див. рис. 3.1) соціальна мережа має 12 варіантів використання. Нижче представлені опис кожного з них.

Опис варіанту “Авторизація”

Опис: користувач відправив запит на авторизацію в соціальній мережі.

Актори: користувач, серверна частина, база даних.

Основний сценарій:

1. Сервер отримав запит на авторизацію.
2. Сервер перевіряє дані.
3. Сервер перевіряє дані в базі даних.
4. Сервер повертає користувачу інформацію про успіх.

Альтернативний сценарій:

1а. Сервер не отримав запит.

2а. Дані прийшли не валідні. Користувач отримує відповідне повідомлення.

2b. Токен не пройшов валідацію. Користувач отримує відповідне повідомлення.

3a. В базі не знайшли відповідного користувача. Користувач отримує відповідне повідомлення.

Опис варіанту “Заповнення анкети”

Опис: Користувач додав нову інформацію до свого профілю.

Актори: користувач, серверна частина, база даних.

Основний сценарій:

1. Сервер отримав HTTP-запит на додавання певної інформації до свого профілю в соціальної мережі.
2. Сервер перевіряє дані.
3. Сервер відправляє запит до бази даних.
4. Дані додається до бази даних.
5. Сервер повертає користувачу повідомлення про успіх.

Альтернативний сценарій:

1a. Сервер не отримав HTTP-запит.

2a. Дані не пройшли перевірку. Користувач отримує негативний результат.

2b. Токен не пройшов валідацію. Користувач отримує відповідне повідомлення.

3a. База даних не відповідає на запити серверу. Користувач отримує негативний результат.

4a. База даних не може додати користувачу нови дані. Користувач отримує негативний результат.

Опис варіанту “Редагування анкети”

Опис: Користувач намагається відновити свій профіль.

Актори: користувач, серверна частина, база даних.

Основний сценарій:

1. Сервер отримав HTTP-запит на оновлення певної інформації до свого профілю в соціальної мережі.
2. Сервер перевіряє дані.
3. Сервер відправляє запит до бази даних.
4. Дані додається до бази даних.
5. Сервер повертає користувачу повідомлення про успіх.

Альтернативний сценарій:

1a. Сервер не отримав HTTP-запит.

2a. Дані не пройшли перевірку. Користувач отримує негативний результат.

2b. Токен не пройшов валідацію. Користувач отримує відповідне повідомлення.

3a. База даних не відповідає на запити серверу. Користувач отримує негативний результат.

4a. База даних не може додати користувачу нови дані. Користувач отримує негативний результат.

Опис варіанту “Додавання друзів”

Опис: Користувач намагається додати нового друга

Актори: користувач, серверна частина, база даних.

Основний сценарій:

1. Сервер отримав HTTP-запит на додавання користувачем нового друга.
2. Сервер перевіряє дані з запиту.
3. Сервер звертається в базу даних для перевірки можливості додати нового друга.
4. База даних повертає результат додавання.
5. Користувач отримує позитивну відповідь.

Альтернативний сценарій:

1a. Сервер не отримав HTTP-запит.

2a. Дані не пройшли перевірку. Користувач отримує негативний результат.

2b. Токен не пройшов валідацію. Користувач отримує відповідне повідомлення.

3a. База даних не відповідає на запити серверу. Користувач отримує негативний результат.

3b. Користувач вже має такого друга. Користувач отримує певне повідомлення с негативним статусом.

4a. База даних не може додати користувачу нови дані. Користувач отримує негативний результат.

Опис варіанту “Проходження тестів”

Опис: Користувач проходить тест за навичками.

Актори: користувач, серверна частина, база даних.

Основний сценарій:

1. Сервер отримав HTTP-запит на проходження користувачем теста.
2. Сервер перевіряє користувачеві дані з запиту.
3. Сервер перевіряє дані про проходження тестів в базі даних.
4. Сервер повертає відповідь у вигляді json-структури з питаннями і варіантами відповідей для тесту.
5. Сервер отримує відповіді на питання та перевіряє правильність відповідей.
6. Сервер надсилає результат в базу даних для фіксування результату та обчислює кількість очок досвіду користувача та додає їх.
7. Сервер повертає користувачу результат проходження тесту.

Альтернативний сценарій:

1a. Сервер не отримав HTTP-запит.

2a. Дані не пройшли перевірку. Користувач отримує негативний результат.

2b. Токен не пройшов валідацію. Користувач отримує відповідне повідомлення.

3a. База даних не відповідає на запити серверу. Користувач отримує негативний результат.

4a. json-структура тесту не було знайдено.

5a. Користувач вирішив не проходити цей тест. В базу зафіксується цей результат.

6a. База даних не відповідає на запити серверу. Користувач отримує негативний результат. Дані про проходження будуть зафіксовані в кеш.

Опис варіанту “Ділитися з друзями”

Опис: Користувач вирішив поділитися результатами тесту зі свома друзями.

Актори: користувач, серверна частина, база даних.

Основний сценарій:

1. Сервер отримав HTTP-запит на додавання користувачем нового запису з результатами тесту.
2. Сервер перевіряє дані з запиту.
3. Сервер звертається в базу даних для перевірки існування такого результату та можливості додати на сторінку цей результат.
4. База даних повертає результат додавання.
5. Сервер вертає позитивний статус.

Альтернативний сценарій:

1a. Сервер не отримав HTTP-запит.

2a. Дані не пройшли перевірку. Користувач отримує негативний результат.

2b. Токен не пройшов валідацію. Користувач отримує відповідне повідомлення.

3a. База даних не відповідає на запити серверу. Користувач отримує негативний результат.

3b. Користувач вже додав на сторінку такий запис. Користувач отримує певне повідомлення с негативним статусом.

4a. База даних не може додати користувачу нови дані. Користувач отримує негативний результат.

Опис варіанту “Тести в змагальному режимі”

Опис: Два користувача вирішили пройти один тест та вирішити хто краще володіє темою.

Актори: користувач, серверна частина, база даних.

Основний сценарій:

1. Сервер отримав HTTP-запит на проходження користувачамі теста.
2. Сервер перевіряє дані користувачів з запиту.
3. Сервер повертає відповідь у вигляді json-структури з питаннями і варіантами відповідей для тесту.
4. Сервер отримує відповіді на питання та перевіряє правильність відповідей.
5. Сервер надсилає результат в базу даних для фіксування результату та обчислює кількість очок досвіду користувача та додає їх.
6. Сервер повертає користувачам результати проходження тесту.

Альтернативний сценарій:

1a. Сервер не отримав HTTP-запит.

2a. Дані не пройшли перевірку. Користувач отримує негативний результат.

2b. Токен не пройшов валідацію. Користувач отримує відповідне повідомлення.

3a. База даних не відповідає на запити серверу. Користувач отримує негативний результат.

4a. json-структура тесту не було знайдено.

5a. Один з користувачів вирішив не проходити цей тест. В базу зафіксується цей результат.

6a. База даних не відповідає на запити серверу. Користувачі отримують негативний результат. Дані про проходження будуть зафіксовані в кеш.

Опис варіанту “Отримання та змінювання внутрішньої валюти”

Опис: Користувач отримав новий уровень та вирішив змінити внутрішню валюту на бонус.

Актори: користувач, серверна частина, база даних, представник адміністрації команії.

Основний сценарій:

1. Сервер отримав HTTP-запит на зміну внутрішньої валюти на бонус.
2. Сервер перевіряє дані з запиту.
3. Сервер звертається в базу даних для перевірки кількості валюти.
4. Сервер направляє повідомлення адміністрації компанії, що користувач хоче змінити валюту на бонус.
5. Після того, як адміністрація розглянула запит, з рахунку абонента буде списана валюта.
6. Сервер надішле повідомлення.

Альтернативний сценарій:

1a. Сервер не отримав HTTP-запит.

2a. Дані не пройшли перевірку. Користувач отримує негативний результат.

2b. Токен не пройшов валідацію. Користувач отримує відповідне повідомлення.

3a. База даних не відповідає на запити серверу. Користувач отримує негативний результат.

3b. У користувача недостатньо валюти. Користувач отримує повідомлення з негативним статусом.

5a. Адміністрація не схвалила запит. Користувач отримує повідомлення з негативним статусом.

Опис варіанту “Переглядати рейтинг користувачів”

Опис: Користувач соціальної мережі вирішив переглянути рейтінг володіння навиком.

Актори: користувач, серверна частина, база даних.

Основний сценарій:

1. Сервер отримав HTTP-запит на перегляд рейтингу.

2. Сервер перевіряє дані запиту.
3. Направляє запит в базу даних для сортування користувачів по навику.
4. Сервер повертає користувачу відсортований список користувачів з очками володіння (очки, які користувачі отримали за проходження тестів).

Альтернативний сценарій:

1a. Сервер не отримав HTTP-запит.

2a. Дані не пройшли перевірку. Користувач отримує негативний результат.

2b. Токен не пройшов валідацію. Користувач отримує відповідне повідомлення.

3a. База даних не відповідає на запити серверу. Користувач отримує негативний результат.

3b. Навик не було знайдено в базі даних. Користувач отримує негативний результат.

3c. Користувачів, які володіють цім навиком, не було знайдено.

Опис варіанту “Створення тестів”

Опис: Користувач додає тести для певного навика.

Актори: користувач, серверна частина, база даних.

Основний сценарій:

1. Сервер отримав HTTP-запит на перегляд рейтингу.
2. Сервер перевіряє дані запиту та валідність json-структури з питаннями.
3. Направляє запит в базу даних для додавання нового тесту для навика.
4. Сервер повертає користувачу повідомлення з успішним статусом.

Альтернативний сценарій:

1a. Сервер не отримав HTTP-запит.

2a. Дані не пройшли перевірку. Користувач отримує негативний результат.

2b. Токен не пройшов валідацію. Користувач отримує відповідне повідомлення.

3a. База даних не відповідає на запити серверу. Користувач отримує негативний результат.

3b. Навик не було знайдено в базі даних. Користувач отримує негативний результат.

Опис варіанту “Реєстрація нового користувача”

Опис: Адміністратор додає нового працівника компанії.

Актори: користувач з роллю адміністратор, серверна частина, база даних.

Основний сценарій:

1. Сервер отримав HTTP-запит на реєстрацію працівника.
2. Сервер перевіряє дані запиту та валідність json-структури з питаннями.
3. Направляє запит в базу даних для додавання нового працівника для навика.
4. Сервер повертає користувачу повідомлення з успішним статусом.

Альтернативний сценарій:

1a. Сервер не отримав HTTP-запит.

2a. Дані не пройшли перевірку. Користувач отримує негативний результат.

2b. Токен не пройшов валідацію. Користувач отримує відповідне повідомлення.

3a. База даних не відповідає на запити серверу. Користувач отримує негативний результат.

3b. Працівник з такими даними вже існує в базі.Користувач отримує негативний результат.

Опис варіанту “Перевірка тестів, зроблених користувачами”

Опис: Користувач з роллю адміністратор схвалив тест, який було створено користувачем.

Актори: користувач з роллю адміністратор, серверна частина, база даних.

Основний сценарій:

1. Сервер отримав HTTP-запит з повідомленням про схвалення тесту.

2. Сервер перевіряє дані запиту.
3. Сервер напрялає запит в базу даних на перевірку чи існує такий тест.
4. Направляє запит в базу даних для схвалення тесту.
5. Сервер повертає адміністратору повідомлення з успішним статусом.
6. Сервер повідомляє автора тесту про схвалення

Альтернативний сценарій:

1a. Сервер не отримав HTTP-запит.

2a. Дані не пройшли перевірку. Користувач отримує негативний результат.

2b. Токен не пройшов валідацію. Користувач отримує відповідне повідомлення.

3a. База даних не відповідає на запити серверу. Користувач отримує негативний результат.

3b. Тест не існує в базі. Користувач отримує негативний результат.

3.3 Нефункціональні вимоги

Нефункціональні вимоги визначають властивості та обмеження до розроблюваного ПЗ.

Під час формування нефункціональних вимог були визначені наступні сценарії якості розроблювальної системи.

Надійність:

1. Система зберігає 95% введених користувачем даних.

3. При неможливості зберегти дані в БД система зберігає дані в кеш після чого робить додаткові запити.

Зручність використання:

1. Зрозумілі повідомлення при успішному виконанні операції або невірно введеним даним.

Продуктивність:

1. Час додавання нових даних складає не більш 2 сек.

3. Час оновлення даних складає не більше 3 сек.

Безпека:

1. Система робить резервне копіювання після кожного запиту додавання або редагування рядків.

Вимоги до реалізації продукту:

- серверна частина повинна бути реалізована мовою Python.
- база даних повинна бути легкою та простою.
- програмне забезпечення повинно працювати на актуальних версіях браузерів або смартфонів.

Висновок

В даному розділі була побудована діаграма варіантів використання, де показано основні варіанти використання системи користувачів з різними ролями. Далі – описи прецедентів. На даному етапі був описаний алгоритм дій для кожного варіанту використання. Останнім етапом став опис нефункціональних вимог до системи, в яких були описані вимоги системи для забезпечення надійності, зручності використання, продуктивності та безпеки системи.

4 ПРОЕКТУВАННЯ СИСТЕМИ

4.1 Архітектура програмного продукту

Після аналізу предметної області та визначення вимог до системи, було вирішено використати багаторівневу архітектуру. Така архітектура дозволяє розподілити систему на рівні, кожен з якого може звертатися тільки на один рівень нижче. Таким чином розробку кожного рівня можна вести незалежно від інших, що дозволяє модифікувати систему. Однак головним недоліком цього підходу є ускладнення системи.

На рисунку 4.1 продемонстровано архитектуру серверної частини соціальної мережі у вигляді діаграми компонентів.

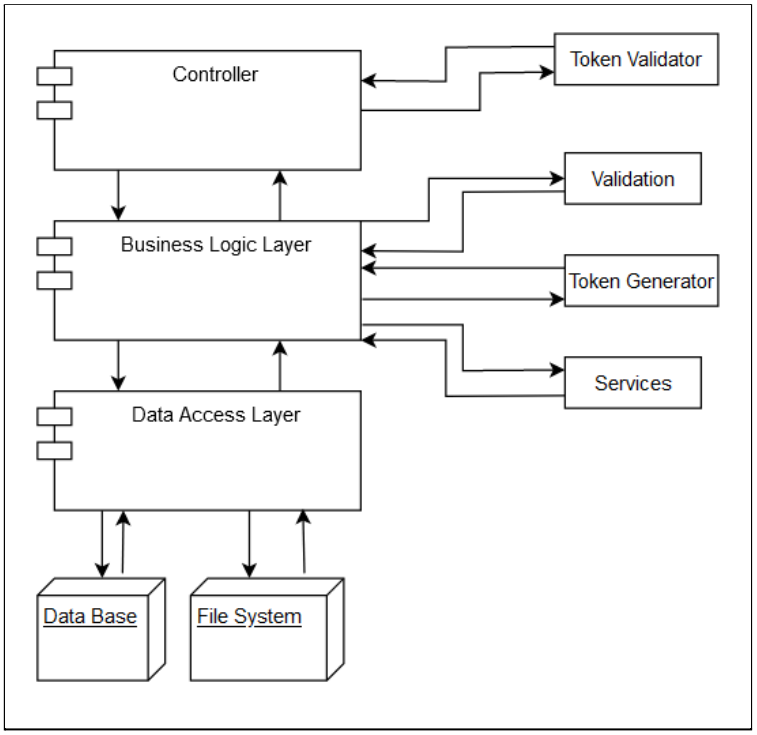


Рисунок 4.1 - Діаграма компонентів серверної частини соціальної мережі

На діаграмі представлено взаємодію компонентів системи. Між всіма рівнями спілкування буде за допомогою DTO (Data Transfer Object). Кожен компонент має свій рівень відповідальності. Контролер отримує запити користувача та повертає

інформацію про виконання запиту. Сервіс отримує дані з запиту, валідує їх, виконує операції та звертається, за допомогою моделі, в базу даних.

4.2 Реляційна модель бази даних

Потрібно створити модель бази даних для зберігання рядків даних, які будуть містить інформацію про користувачів соціальної мережі, про тести та додаткову інформацію для функціонування серверної частини. На рисунку 4.2 приведена реляційна модель бази даних.

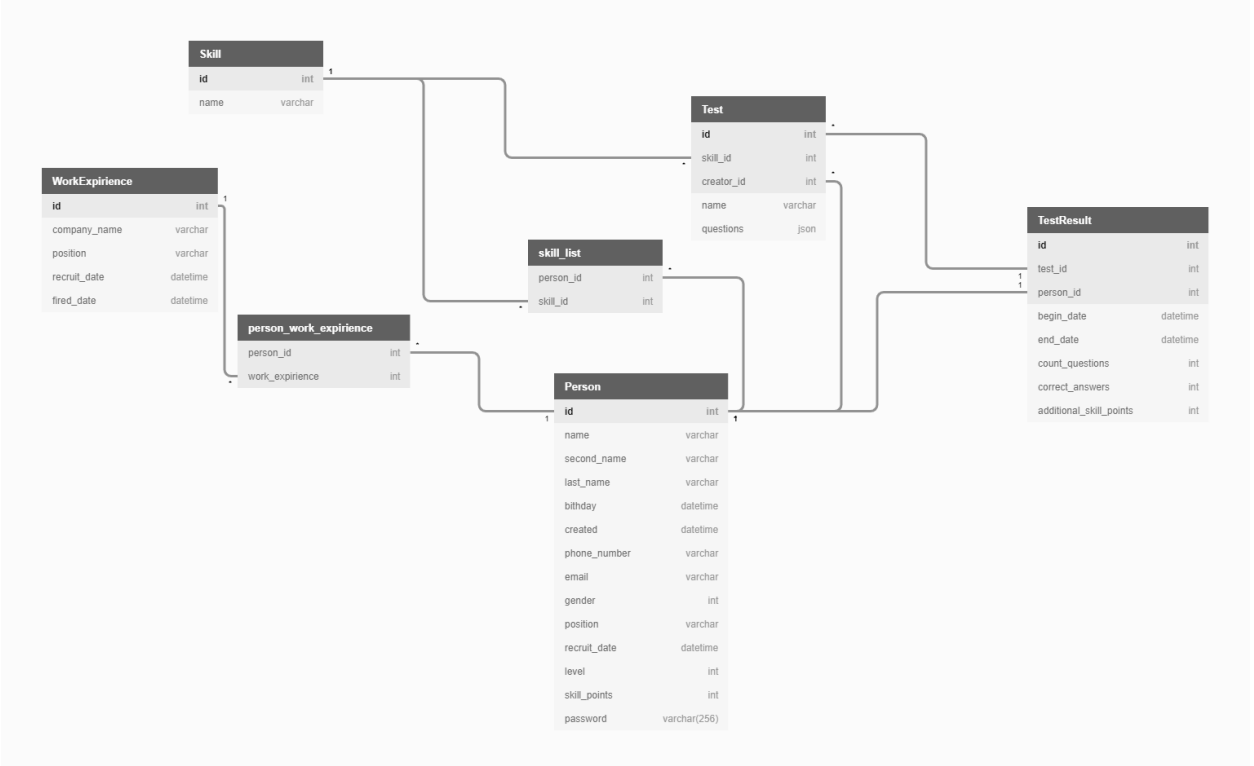


Рисунок 4.2 - Архітектура бази даних

Розглянемо опис кожної таблиці.

Person - інформація про людину (табл. 4.1).

Таблиця 4.1 - Опис таблиці Person

Назва	Тип	Ключ	Пояснення
id	int	Primary Key	Ідентифікатор запису
name	varchar	-	Им’я
second_name	varchar	-	По-батькові
last_name	varchar	-	Фамілія
birthday	datetime	-	Дата народження
created	datetime	-	Дата створення запису
phone_number	varchar	-	Номер телефона
email	varchar	-	Електрона пошта
gender	int	-	Пол
level	int	-	Рівень користувача
skill_points	int	-	Кількість очок
position	varchar	-	Посада
recruit_date	datetime	-	Дата найму
password	string	-	Пароль користувача

person_work_expirience - додаткова таблиця для досвіду роботи (табл. 4.2).

Таблиця 4.2 - Опис таблиці person_work_expirience

Назва	Тип	Ключ	Пояснення
person_id	int	Foreign Key	Ідентифікатор запису
work_expirience	int	Foreign Key	Ідентифікатор запису досвіду роботи

WorkExpirience - таблиця для опису досвіду роботи (табл. 4.3).

Таблиця 4.3 - Опис таблиці WorkExpirience

Назва	Тип	Ключ	Пояснення
id	int	Primary Key	Ідентифікатор запису
company_name	varchar	-	Назва попередньо міста роботи
position	varchar	-	Попередня посада
recruit_date	datetime	-	Дата початку роботи
fired_date	datetime	-	Дата закінчення

Test - таблиця для зберігання питань та відповідей для тестування (табл. 4.4)

Таблиця 4.4 - Опис таблиці Test

Назва	Тип	Ключ	Пояснення
id	int	Primary Key	Ідентифікатор запису
skill_id	int	Foreign Key	Ідентифікатор навика

Продовження таблиці 4.4

Назва	Тип	Ключ	Пояснення
creator_id	int	Foreign Key	Ідентифікатор творця
name	varchar	-	Назва тесту
questions	json	-	json-структура з питаннями та відповідями

Skill - таблиця для зберігання навиків (табл. 4.5)

Таблиця 4.5 - Опис таблиці Skill

Назва	Тип	Ключ	Пояснення
id	int	Primary Key	Ідентифікатор запису
name	varchar	-	Назва навика

skill_list - додаткова таблиця для перегляду тестів одного співробітника (табл. 4.6)

Таблиця 4.6 - Опис таблиці skill_list

Назва	Тип	Ключ	Пояснення
person_id	int	Foreign Key	Ідентифікатор людини
skill_id	int	Foreign Key	Ідентифікатор навика

TetsResult - таблиця для зберігання результатів тестів користувачів (табл. 4.7).

Таблиця 4.7 - Опис таблиці TestResult

Назва	Тип	Ключ	Пояснення
id	int	Primary Key	Ідентифікатор запису
test_id	int	Foreign Key	Ідентифікатор тесту
person_id	int	Foreign Key	Ідентифікатор користувача
begin_date	datetime	-	Дата початку проходження тесту
end_date	datetime	-	Дата кінця проходження тесту
count_questions	int	-	Кількість відповідей
correct_answers	int	-	Кількість правильних відповідей
additional_skill_points	int	-	Кількість очок, які отримав користувач

4.3 Діаграма програмних класів

Після формування архітектури системи та формування архітектури бази даних, слід перейти до формування статичної структури у вигляді класів програми. Цю задачу виконує діаграма програмних класів (рис. 4.3).

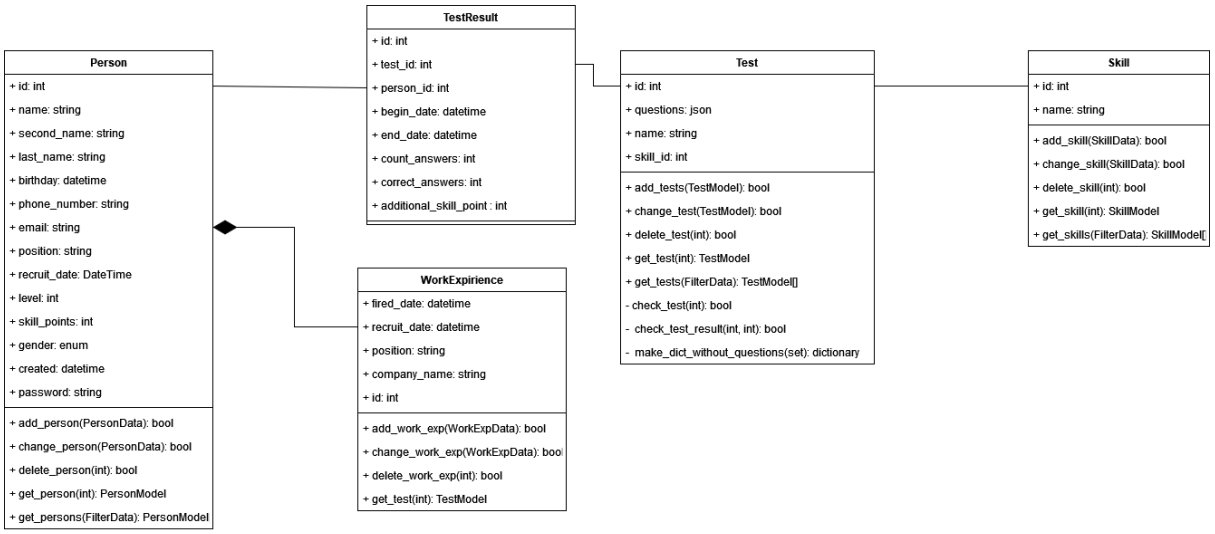


Рисунок 4.3 - Діаграма классів

Клас Person містить інформацію та функції про кандидата на працевлаштування.

Клас Employee містить інформацію та функції для обробки даних про співробітника, який був кандидатом на працевлаштування в компанії.

Клас WorkExperience містить інформацію та функції для обробки інформації про попереднє місце роботи людини.

Клас Test містить інформацію та функції для обробки даних про тести.

Клас Skill містить інформацію та функції для обробки даних про навички.

4.4 Діаграми послідовності

В даному підрозділі було побудовано діаграми послідовності для методів контролеру, які були побудовані на протязі практики.

Діаграма послідовності відображає поведінку об’єктів протягом часу виклика функції.

На рисунку 4.4 відображена діаграма послідовності прецеденту “Авторизация”.

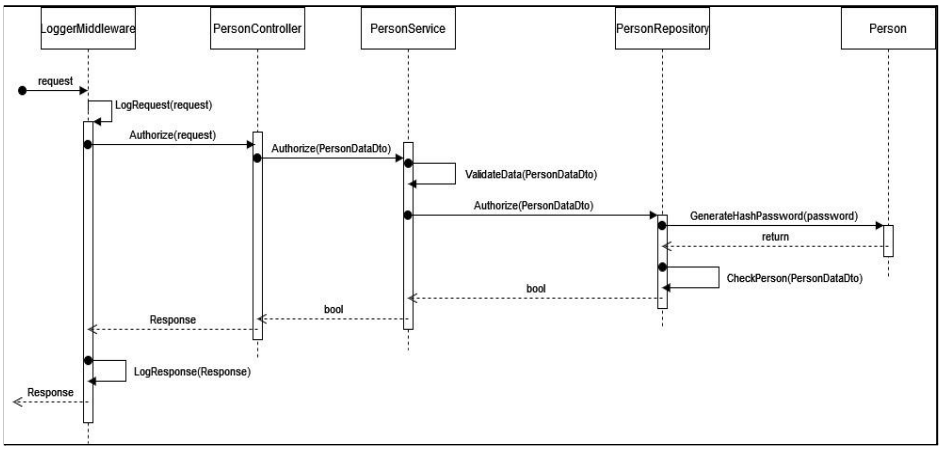


Рисунок 4.4 - Діаграма послідовності прецеденту “Авторизація”

Коли сервер отримує запит на авторизацію, запит додається в лог. Після цього контролер отримує запит та формує PersonDataDto - об’єкт, який містить інформацію про персону. Далі викликається метод авторизації в класі PersonService. Цей клас перевіряє дані та викликає метод в класі PersonRepository, який має доступ до бази даних. В цьому класі проходить перевірка існування персони в базі даних. Далі повертається результат перевірки. В контролері формується відповідь. Потім відповідь також додається в лог та повертається користувачу.

Розглянемо діаграму послідовності прецеденту “Додавання тесту” на рисунку 4.5.

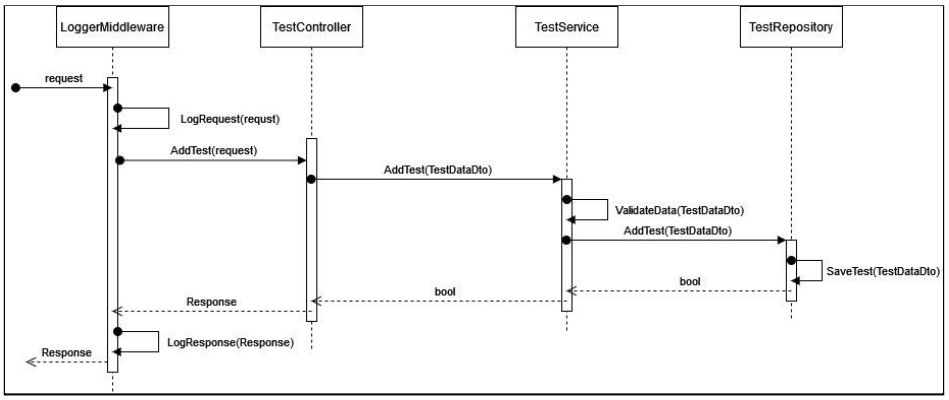


Рисунок 4.5 - Діаграма послідовності прецеденту “Додавання тесту”

Коли сервер отримує запит на додавання нового тесту, запит додається до файлу з логами. Після цього запит передається в контролер, де формується TestDataDto. Далі цей об’єкт передається в сервіс тестов, де перевіряється, та йде до репозиторію, де він зберігається в базі даних. Потім результат повертається та на основі цього результата формується відповідь, яка також додається до файлу логу, та вертається користувачу.

Висновок

В даному розділі було спроектовано веб-застосування, враховуючи прецеденти, які були створені в попередньому розділі.

По-перше було створено архітектуру веб-застосування, яку потрібно зробити. Це буде слоїста архитектура, тому що в каждого рівня своя ступінь відповідальності.

Також спроектована модель бази даних та діаграма програмних класів для веб-застосування.

Останнім кроком було створення діаграми послідовностей для двох прецедентів - авторизації та додавання тесту.

5 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Для реалізації серверної частини веб-застосування було вирішено використовувати мову програмування Python. Розробка API буде зроблено на мікрофреймворкі Flask. Як СУБД було обрано PostgreSQL.

5.1 Огляд використовуваних технологій

Python - високорівнева об’єктно-орієнтована мова програмування з динамічною типізацією. Використовується для написання веб-застосування, машинного навчання, написання простих скриптів [5].

Flask - фреймворк для розробки веб-застосунків на мові програмування Python. Має багато модулів, які за необхідністю можна встановити та додати до проекту окремо [3][6].

PostgreSQL - об'єктно-реляційна система керування базами даних [7].

SQLAlchemy - це інструментарій SQL, який дозволяє переносити моделі описані мовою Python, в таблиці бази даних за допомогою міграцій [4].

5.5 Розгортання проекту

Для розгортання проекту потрібно мати на комп’ютері встановлений Python 3.8+. Треба зробити нове віртуальне оточення. Для цього потрібно в консолі викликати команду python -m virtualenv example (рис. 5.1).

```
E:\>python -m virtualenv example
created virtual environment CPython3.8.1.final.0-64 in 12398ms
  creator CPython3Windows(dest=E:\example, clear=False, no_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, via=copy, app_data_dir=C:\Users\tkachn
ekox\AppData\Local\pypa\virtualenv)
    added seed packages: pip==21.3, setuptools==58.2.0, wheel==0.37.0
  activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerShellActivator,PythonActivator
```

Рисунок 5.1 - Створення віртуального оточення

В папку треба клонувати проект з git-репозиторію за допомогою команди git clone https://github.com/tkoldm/SocNet.git (рис. 5.2).

```
E:\example>git clone https://github.com/tkoldm/SocNet.git
Cloning into 'SocNet'...
remote: Enumerating objects: 177, done.
remote: Counting objects: 100% (177/177), done.
remote: Compressing objects: 100% (119/119), done.

Receiving objects: 100% (177/177), 35.29 KiB | 516.00 KiB/s, done.
Resolving deltas: 100% (71/71), done.

E:\example>
```

Рисунок 5.2 - Клонування проекту

Далі необхідно активувати віртуальне оточення командою Scripts\activate та перейти в папку з тільки що склонованим проектом. Потім встановити необхідні залежності за допомогою пакетного менеджеру Python та файла залежностей командою pip install -r requirements.txt (рис. 5.3).

```
(example) E:\example\SocNet>pip install -r requirements.txt
Collecting alembic==1.5.8
  Using cached alembic-1.5.8-py2.py3-none-any.whl (159 kB)
Collecting aniso8601==9.0.1
  Using cached aniso8601-9.0.1-py2.py3-none-any.whl (52 kB)
Collecting astroid==2.5.1
  Using cached astroid-2.5.1-py3-none-any.whl (222 kB)
Collecting attrs==20.3.0
  Using cached attrs-20.3.0-py2.py3-none-any.whl (49 kB)
Collecting certifi==2020.12.5
  Using cached certifi-2020.12.5-py2.py3-none-any.whl (147 kB)
Collecting cffi==1.15.0
  Using cached cffi-1.15.0-cp38-cp38-win_amd64.whl (179 kB)
Collecting chardet==4.0.0
  Using cached chardet-4.0.0-py2.py3-none-any.whl (178 kB)
Collecting click==7.1.2
  Using cached click-7.1.2-py2.py3-none-any.whl (82 kB)
Collecting colorama==0.4.4
  Using cached colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Collecting cryptography==35.0.0
  Using cached cryptography-35.0.0-cp36-abi3-win_amd64.whl (2.1 MB)
Collecting debugpy==1.5.1
  Using cached debugpy-1.5.1-cp38-cp38-win_amd64.whl (4.4 MB)
Collecting dnspython==2.1.0
  Using cached dnspython-2.1.0-py3-none-any.whl (241 kB)
Collecting email-validator==1.1.3
  Using cached email_validator-1.1.3-py2.py3-none-any.whl (18 kB)
Collecting Faker==6.6.2
  Using cached Faker-6.6.2-py3-none-any.whl (1.2 MB)
```

Рисунок 5.3 - Встановлення залежностей

Наступний крок - створення файлів оточення. Для того, щоб запустити проект потрібно створити в кореневому каталозі файл з розширенням .env зі структурою, яка продемонстрована на рисунку 5.4.

```
SECRET_KEY="SecretKey"
SQLALCHEMY_TRACK_MODIFICATIONS=False
DB_DRIVER="postgresql_or_another"
POSTGRES_USER=postgres_user
POSTGRES_PASSWORD=postgres_user_password
POSTGRES_HOST=host
POSTGRES_DB=name_of_db
PERSONS_PER_PAGE=5
LOGGER_FROMATTER=%(asctime)s:%(name)s:%(levelname)s:%(message)s
LOG_LEVEL=INFO
LOG_PATH=app.log
LEVEL_CONSTANT = 5
```

Рисунок 5.4 - Приклад файлу оточення

Якщо є необхідність використати базу даних в контейнері Docker, потрібно зібрати docker-compose.yaml з папки docker як продемонтровано на рисунку 5.5.

```
PS E:\example\SocNet\docker> docker-compose up -d
Creating network "docker_default" with the default driver
Creating volume "docker_postgres" with default driver
Pulling database (postgres:)...
latest: Pulling from library/postgres
eff15d958d66: Pull complete
de2b4ab3ade5: Pull complete
108afa831d95: Pull complete
e5821d5963ce: Pull complete
5be06220aa99: Pull complete
c877668f09b8: Pull complete
7037ade1772d: Pull complete
d46bc4e17ddc: Pull complete
a60906bcf87f: Pull complete
a1c85f71c941: Pull complete
69f50e484cab: Pull complete
2f8a286a55a4: Pull complete
8d590b0d720c: Pull complete
Digest: sha256:1fe27b334443793af98d7eb320ad6f9f30fcc9bc068f545cb46ec01cefe9c8ee
Status: Downloaded newer image for postgres:latest
```

Рисунок 5.5 - Створення контейнеру Docker

Якщо потрібно змінити назву бази даних, користувача або пароль - відповідні зміни потрібно провести в файлі database.env. Приклад структури цього файлу на рисунку 5.6.

```
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
POSTGRES_DB=socnetdb
```

Рисунок 5.6 - Приклад файлу конфігурації бази даних

Наступний крок - застосування міграцій на базу даних. Для цього треба виконати в консолі команду set FLASK_APP=main:app для встановлення виконуваного файлу main.py як веб-застосування в віртуальному оточенні. Далі треба виконати команду flask db upgrade (рис. 5.7).

```
(example) E:\example\SocNet>set FLASK_APP=main:app

(example) E:\example\SocNet>flask db upgrade
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade  -> 9bcc75f3f61c, Created models
```

Рисунок 5.7 - Застосування міграцій

В базу додались потрібні таблиці. Це видно на рисунку 5.8.

alembic_version	16 385	postgres	pg_default
person	16 412	postgres	pg_default
person_skills	16 427	postgres	pg_default
role	16 391	postgres	pg_default
skill	16 398	postgres	pg_default
test	16 443	postgres	pg_default
test_result	16 464	postgres	pg_default
work_exp	16 405	postgres	pg_default

Рисунок 5.8 - Створені після міграції таблиці

Якщо потрібно додати тестові дані в таблиці з користувачами, ролями користувачів та досвідом роботи, треба виконати команду python insertion.py up (рис. 5.9)

```
(example) E:\example\SocNet>python insertion.py up

(example) E:\example\SocNet>
```

Рисунок 5.9 - Створення тестових користувачів

Коли створення даних завершено, можна переглянути нові дані (рис. 5.10).

create_date	remove_date	id	first_name	second_name	last_name	birthday	phone_number	email	position
2018-03-07 03:00:00.000 +0300		1	Lisa	Brewer	MD	1987-09-06	2223207334994	fneal@gmail.com	Technical author
2018-12-09 03:00:00.000 +0300		2	Mary	Wilson	II	1988-03-11	2172058828431	stephanieweeks@yahoo.com	Energy engineer
2018-01-03 03:00:00.000 +0300		3	James	Wyatt	MD	2000-06-17	5202767773825	housenicholas@boyd-wiley.net	Museum/gallery exhibitions offic
2018-02-12 03:00:00.000 +0300		4	John	Evans	DDS	1978-01-07	3130370706842	janetmoore@gmail.com	Sound technician, broadcasting/f
2019-10-22 03:00:00.000 +0300		5	Kristen	Smith	Jr.	1987-03-30	2970734099676	adrian67@hotmail.com	Psychiatrist
2017-01-04 03:00:00.000 +0300		6	Donna	Barber	MD	1979-09-30	8576941533552	elizabethbrooks@hotmail.com	Accountant, chartered
2018-08-13 03:00:00.000 +0300		7	Tammy	Williams	DDS	1999-11-04	1456979444137	laura91@harris.com	Psychotherapist, dance movement
2019-03-18 03:00:00.000 +0300		8	Ashley	Reid	DDS	1996-12-23	7564735286241	joanroberts@yahoo.com	Audiological scientist
2018-10-07 03:00:00.000 +0300		9	Andrew	Day	DDS	1993-09-07	8014101451955	lewisronald@acevedo.biz	Radiographer, therapeutic
2018-10-02 03:00:00.000 +0300		10	Philip	Gonzalez	Jr.	2002-08-13	7307546422867	christinecastillo@gmail.com	Nurse, mental health

Рисунок 5.10 - Перегляд тестових користувачів

Останній крок - запуск в тестовому режимі веб-застосування. Для цього необхідно виконати команду flask run як зображано на рисунку 5.11.

```
(example) E:\example\SocNet>flask run
 * Serving Flask app "main:app"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Рисунок 5.11 - Запуск веб-застосування

5.6 Огляд структури проекту

Виходячи з усіх вимог та обмежень, які були виявлені на етапі проектування системи, а також враховуючи архітектуру проекту, структура готового проекту має такий вигляд:

```
\---SocNet
 |   .gitignore
 |   config.py
 |   insertion.py
 |   main.py
 |   README.md
 |   requirements.txt
 |
 +---app
 |   |   app.py
 |   |   BlueprintGroup.py
 |   |   __init__.py
 |   |
 |   +---exceptions
 |   |       PersonException.py
 |   |       SkillException.py
 |   |       TestException.py
 |   |       TokenException.py
 |   |       __init__.py
 |   |
 |   +---middleware
 |   |       LoggerMiddleware.py
 |   |       __init__.py
 |   |
 |   \---validation
 |           BaseValidation.py
 |           PersonValidation.py
 |           TestValidation.py
 |           __init__.py
 |
 +---bll
 |   |   PersonService.py
 |   |   SkillService.py
```

```
|   |   TestService.py
|   |   TokenHelper.py
|   |   __init__.py
|   |
|   \---dto
|           PersonSerialized.py
|           TestSerialized.py
|           TestSerializedView.py
|           __init__.py
|
+---controller
|   |   PersonController.py
|   |   SkillController.py
|   |   TestController.py
|   |   TokenHelper.py
|   |   __init__.py
|   |
|   \---Dto
|           PersonAuthDataDto.py
|           PersonDataDto.py
|           PersonUpdateDto.py
|           SkillCreateDto.py
|           TestUpdateDto.py
|           TetsCreateDto.py
|           __init__.py
|
+---dal
|   |   extensions.py
|   |   __init__.py
|   |
|   \---models
|           BaseMixin.py
|           PersonModel.py
|           RoleModel.py
|           SkillModel.py
|           TestModel.py
|           TestResult.py
|           WorkExpirienceModel.py
|           __init__.py
|
+---docker
|     database.env
```

```
|     docker-compose.yaml
|
\---migrations
      |   alembic.ini
      |   env.py
      |   README
      |   script.py.mako
      |
      \---versions
            9bcc75f3f61c_created_models.py
```

У кожному модулі є файл __init__.py, це необхідно для того, щоб при імпорті будь-якого модуля всі необхідні класи та структури з модуля безумовно імпортувалися.

Наприклад, у папці app __init__.py містить наступний код:

```
from .app import create_app
```

це означає, що при імпорті модуля app у будь-який модуль за замовчуванням імпортуватиметься функція create_app.

На рівнях controller і bll (business logic layer) оголошено об'єкти передачі інформації на різні рівні (data transfer object). Це зроблено для того, щоб передавати дані між шарами одним параметром, а не безліччю різнорідних значень.

У модулі middleware оголошено клас для логування запитів, що надходять від клієнта, та відповідей, які надсилаються клієнту. Для того, щоб була можливість відстежити ланцюжок запит-відповідь, було вирішено вказувати унікальний ідентифікатор типу Guid, який при логуванні до файлу додається до лога запиту та відповіді. У зв'язаних між собою запитів та відповідей цей ідентифікатор буде свій.

У файлі README.md також є інструкція з розгортання, налаштування та запуску проекту.

Висновок

В цьому розділі було розглянуто технології, які було використано під час розробки проекту: мова програмування Python, микрофреймворк Flask, система управління базами даних PostgreSQL та об'єктно-реляційне відображення для мови програмування Python SQLAlchemy. А також приклад розгортання проекту в тестовому середовищі та застосування зроблених міграцій на базу даних в Docker-контейнері. Також було розглянуто структуру готового проекту.

6 ТЕСТУВАННЯ

Тестування - це одна з технік контролю якості, яка включає планування робіт, проектування та написання тестів для певних компонентів, виконання тестування та аналізу вихідних результатів [8].

Мета тестування - перевірити наскільки готове програмне рішення відповідає вимогам, та знайти помилки в роботі програми до того, як їх знайдуть цільови користувачи.

Модульне тестування - це тип тестування програмного забезпечення, яке перевіряє окремі програмні модулі або компоненти. Його мета полягає в тому, щоб переконатися, що кожна одиниця коду працює належним чином. Цей тип тестування проводиться розробниками на етапі кодування програми. Модульні тести виділяють фрагмент коду та перевіряють його роботу. Одиницею вимірювання може бути одна функція, метод, процедура, модуль або об’єкт [9].

Інтеграційне тестування - це вид тестування, при якому перевіряється відповідність вимогам інтеграція модулів, їх взаємодія один з одним, а також інтеграція підсистем у загальну систему. Інтеграційні тести використовують компоненти, які вже були перевірені за допомогою модульних тестів і згруповані в набори. Ці збірки перевіряються відповідно до встановленого для них плану випробувань і об’єднуються через їхні інтерфейси [11].

В цьому розділі буде проведено огляд процесу інтеграційного тестування серверної частини соціальної мережі за допомогою програмного забезпечення Postman.

Postman - це набір інструментів тестування API. Це середовище розробки для створення, тестування, моніторингу та публікації документації API. Метою Postman є перевірка надсилання запитів від клієнта до сервера та отримання відповіді від сервера [11].

Етапи використання запитів у Postman:

- Додавання запиту.

- Налаштування запиту.
- Виконання запиту.
- Реєстрація запиту.

На початку тестування було створено колекції запитів для кожного контролеру (рис. 6.1).

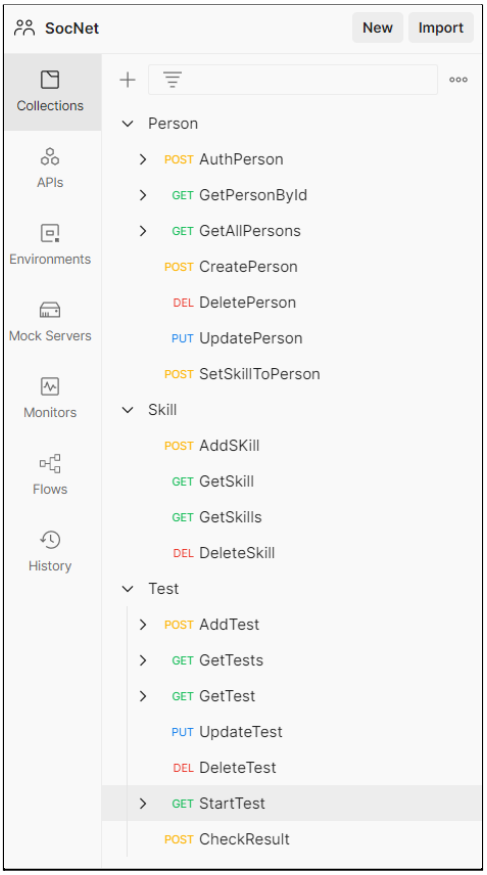


Рисунок 6.1 - Колекції запитів для кожного контролеру

Кожен контролер отримує запити з певним методом та виконує потрібну операцію за даними.

Далі буде розглянуто тестування з валідними та не валідними даними для запитів до серверної частини.

6.1 Тестування методу авторізації

Для авторизації в соціальної мережі потрібно ввести електронну пошту та пароль персони. Сервер отримує HTTP-запит з методом POST по маршруту “/api/person/login”. Якщо персона не знайдена або пароль не співпадає, буде повернено помилку. В позитивному сценарії - користувач отримує токен, який підтверджує його авторізацію. Перевіримо як пройде авторіація з коректними даними.

На рисунку 6.2 ми бачимо, що такий користувач був знайден в таблиці користувачів, та хеш його паролю співпадає з хешем в базі.

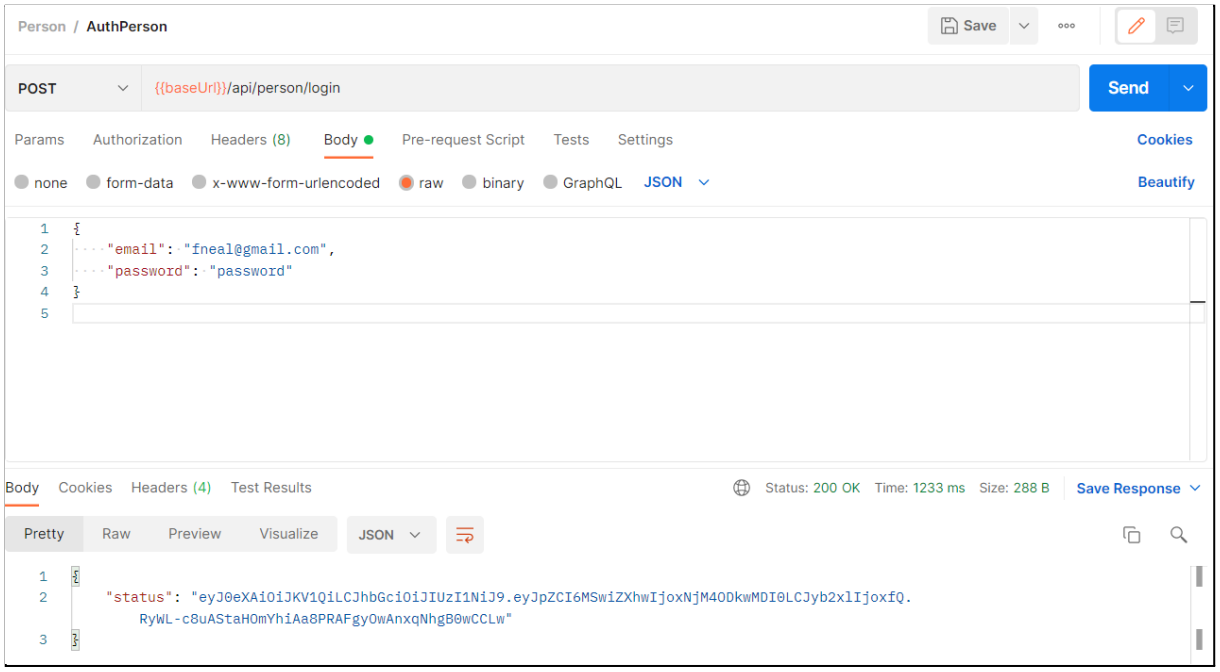


Рисунок 6.2 - Тестування методу авторизації з правильними даними

Далі проведемо тест з неправильним паролем. На рисунку 6.3 продемонтрован запит та відповідь сервера, якщо пароль існуючої персони не співпадає.

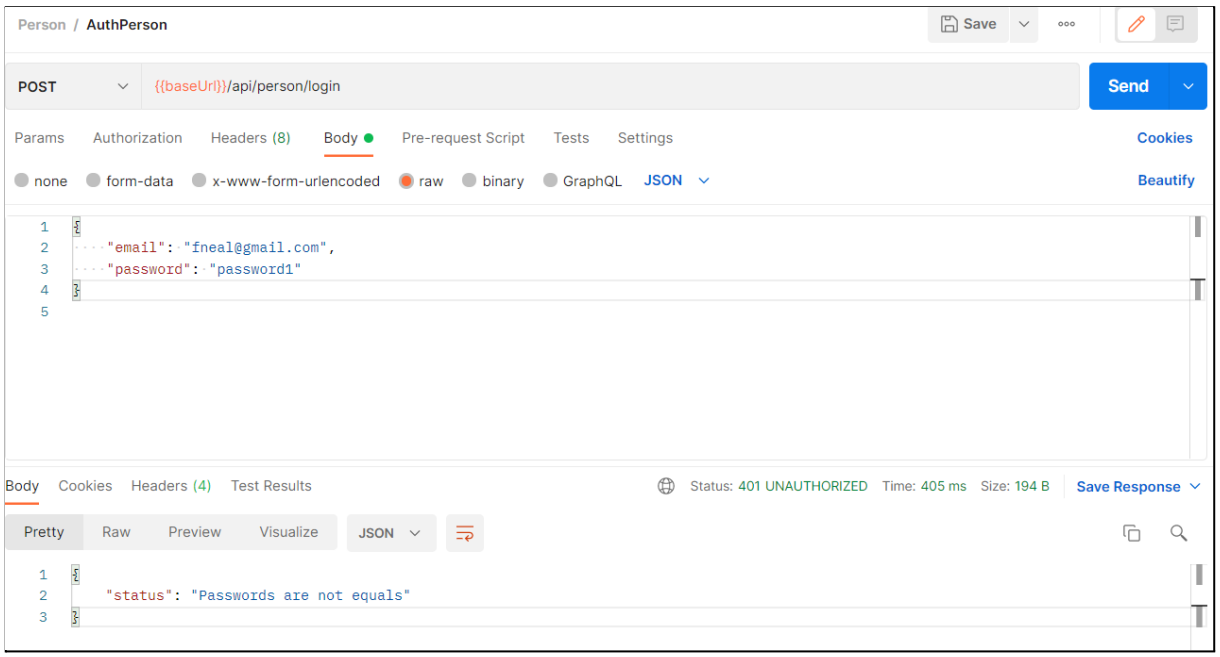


Рисунок 6.3 - Тестування методу авторизації з неправильним паролем

На рисунку 6.4 продемонтрован результат тесту з електронною поштою, яку не було знайдено в базі даних, тобто жодна персона не має таку пошту.

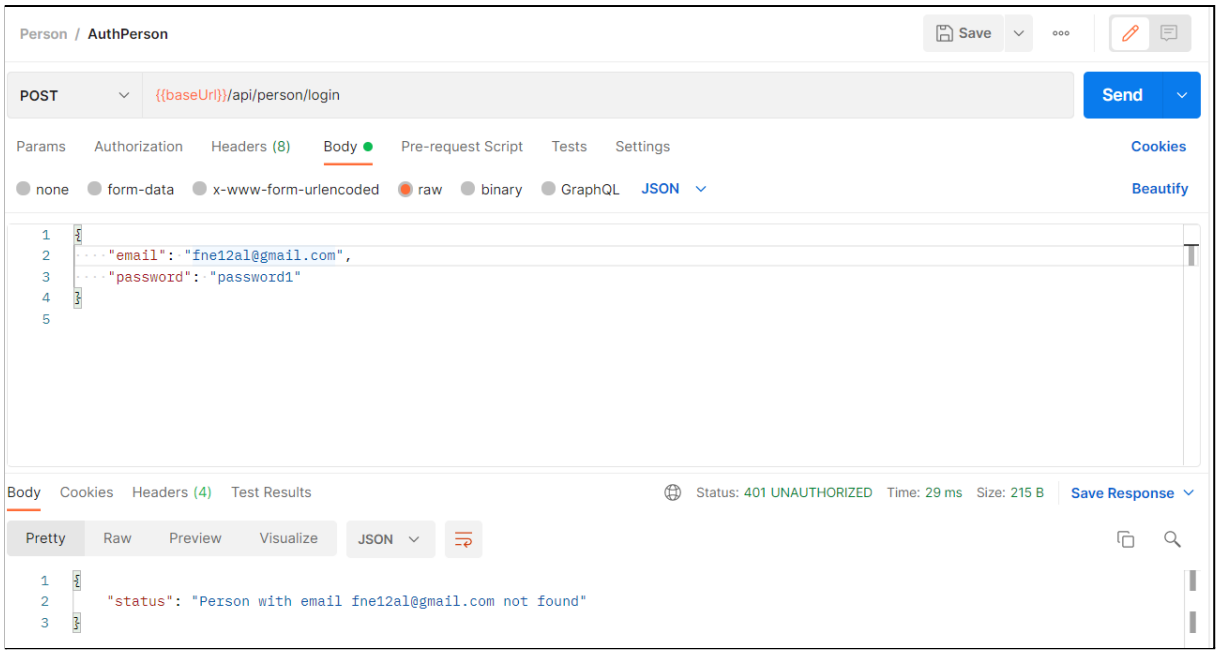


Рисунок 6.4 - Тестування методу авторизації з неіснуючою поштою

Під час тестування методу авторизації не було виявлено непередбаченої поведінки системи.

6.2 Тестування методу отримання персони за ідентифікатором

Для тестування цього методу не потрібно бути авторизованим у системі, однак треба знати хоча б ідентифікатор персони. Якщо був введен правильний ідентифікатор, дані про персону повернуться користувачу, як продемонтровано на рисунку 6.5.

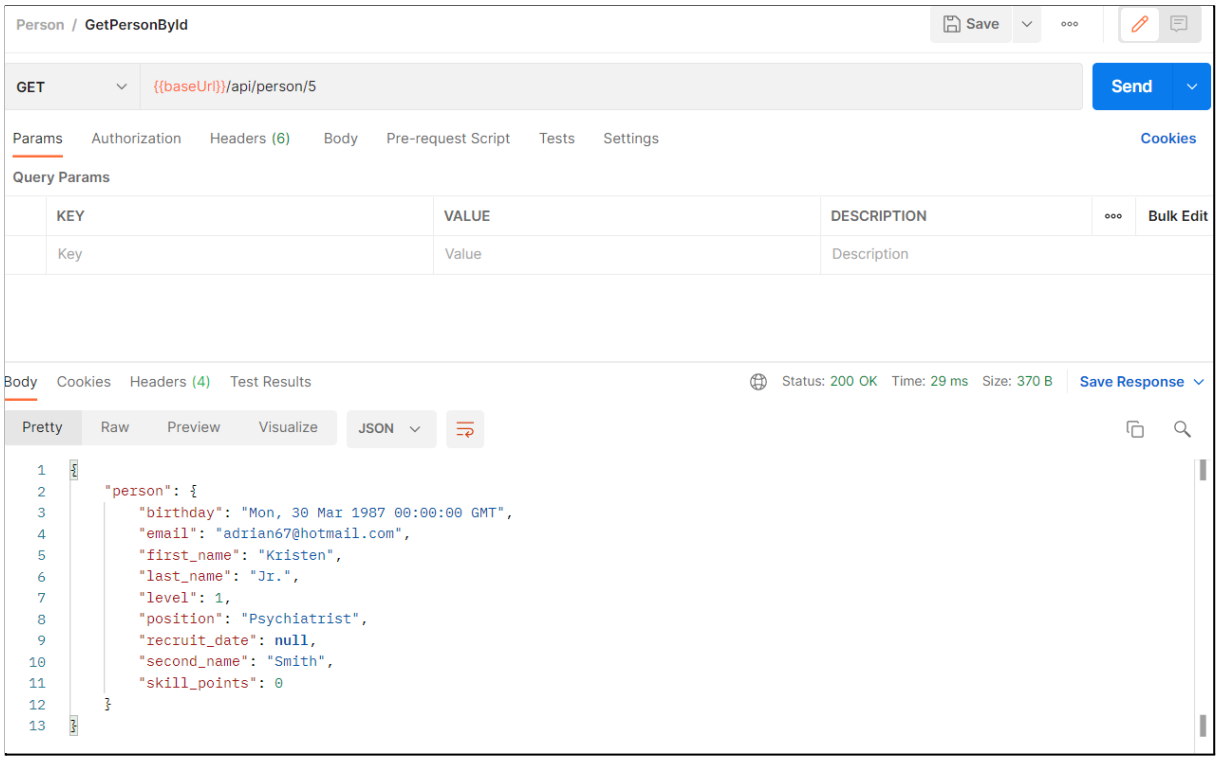


Рисунок 6.5 - Результат отримання даних персони з існуючим ідентифікатором

Якщо намагаються отримати персону з неіснуючим ідентифікатором або персону, яку видалили, користувач отримує відповідне повідомлення.

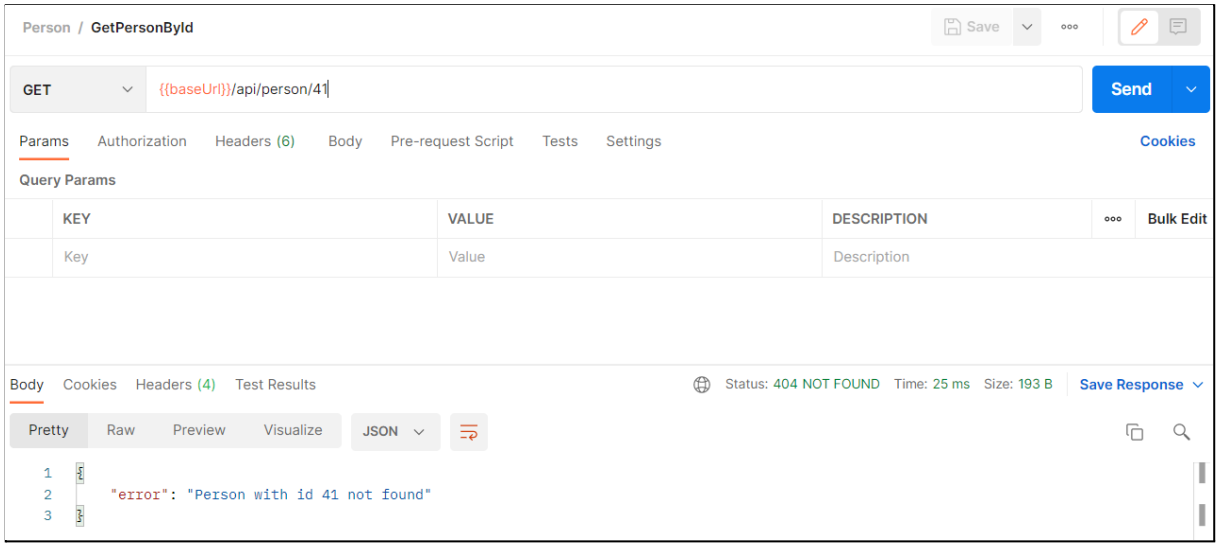


Рисунок 6.6 - Тестування методу отримання персони з неіснуючим ідентифікатором

На рисунку 6.6 продемонстровано негативний сценарій тесту отримання персони, тому що в базі даних зараз тільки 10 персон. Тобто Персона з ідентифікатором 41 ще не була створена.

За допомогою SQL-скріпта:

```
UPDATE public.person
SET remove_date=now()
WHERE id=3;
```

встановимо персоні з ідентифікатором 3 дату видалення (рис. 6.7).

	create_date	remove_date	id	first_name	second_name	last_name	birthday	phone_number	email
1	2018-03-07 03:00:00.000 +0300	[NULL]	1	Lisa	Brewer	MD	1987-09-06	2223207334994	fneal@gmail.com
2	2018-12-09 03:00:00.000 +0300	[NULL]	2	Mary	Wilson	II	1988-03-11	2172058828431	stephanieweeks@yahoo.com
3	2018-02-12 03:00:00.000 +0300	[NULL]	4	John	Evans	DDS	1978-01-07	3130370706842	janetmoore@gmail.com
4	2019-10-22 03:00:00.000 +0300	[NULL]	5	Kristen	Smith	Jr.	1987-03-30	2970734099676	adrian67@hotmail.com
5	2017-01-04 03:00:00.000 +0300	[NULL]	6	Donna	Barber	MD	1979-09-30	8576941533552	elizabethbrooks@hotmail.com
6	2018-08-13 03:00:00.000 +0300	[NULL]	7	Tammy	Williams	DDS	1999-11-04	1456979444137	laura91@harris.com
7	2019-03-18 03:00:00.000 +0300	[NULL]	8	Ashley	Reid	DDS	1996-12-23	7564735286241	joanroberts@yahoo.com
8	2018-10-07 03:00:00.000 +0300	[NULL]	9	Andrew	Day	DDS	1993-09-07	8014101451955	lewisronald@acevedo.biz
9	2018-10-02 03:00:00.000 +0300	[NULL]	10	Philip	Gonzalez	Jr.	2002-08-13	7307546422867	christinecastillo@gmail.com
10	2018-01-03 03:00:00.000 +0300	2021-11-15 18:43:10.570 +0300	3	James	Wyatt	MD	2000-06-17	5202767773825	housenicholas@boyd-wiley.net

Рисунок 6.7 - Встановлення дати видалення для персони з ідентифікатором 3

Тепер протестуємо можливість отримати інформацію про персону з ідентифікатором 3 (рис. 6.8).

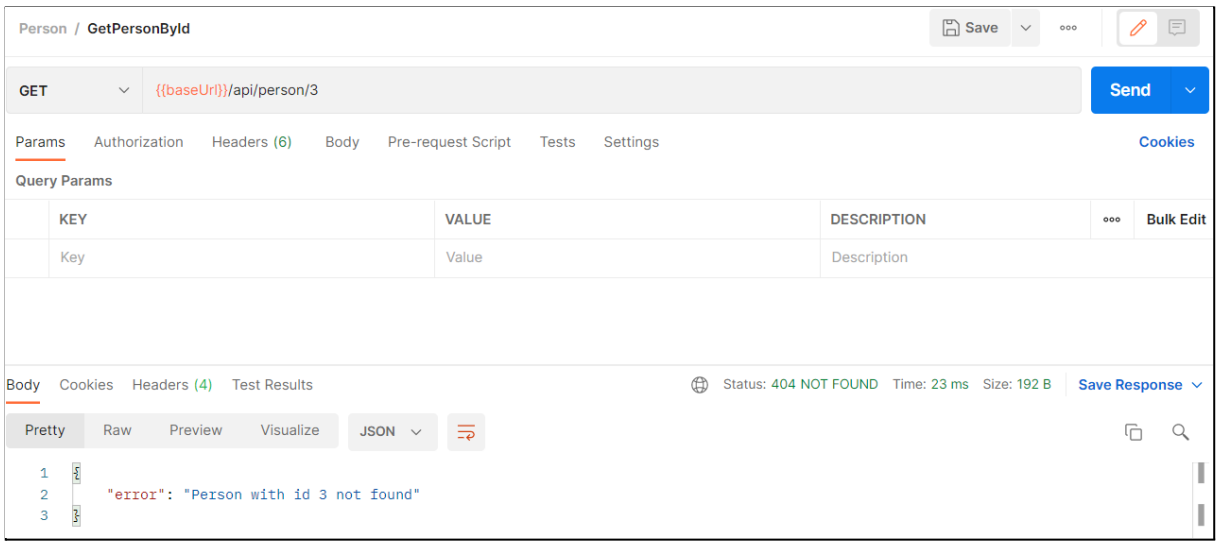


Рисунок 6.8 - Тестування отримання видаленної персони

Хоча персона існує в базі, але в неї є дата видалення, тому буде повернено, що персону не знайдено.

Під час тестування методу отримання персони за идентификатором не було виявлено непередбаченої поведінки серверу.

6.3 Тестування методу додавання нової персони

Для того, щоб додати персону, необхідно мати права адміністратора та бути авторизованим в системі. Якщо не була пройдена хоча б одна перевірка, користувач отримує певну помилку.

Для того, щоб пройти перевірку, потрібно знайти персону з ролю, яка має права адміністратора та отримати для цієї персони токен. Після отримання токену, його необхідно внести в запиту з ключем Authorization. На рисунку 6.9 продемонстровано тестування позитивного сценарію.

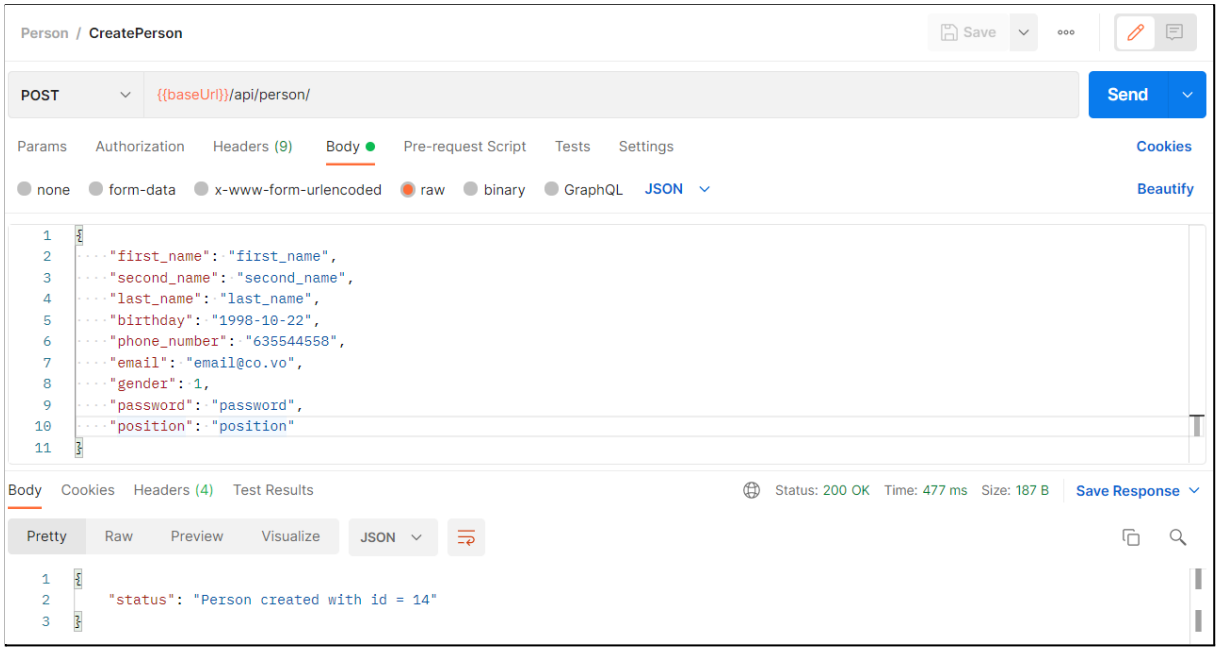


Рисунок 6.9 - Тестування додавання персони з валідними даними

Якщо ще раз спробувати додати персону з такими даними, то буде отримано повідомлення про те, що користувач з такою поштою або номером телефону вже був доданий до бази даних (рис. 6.10).

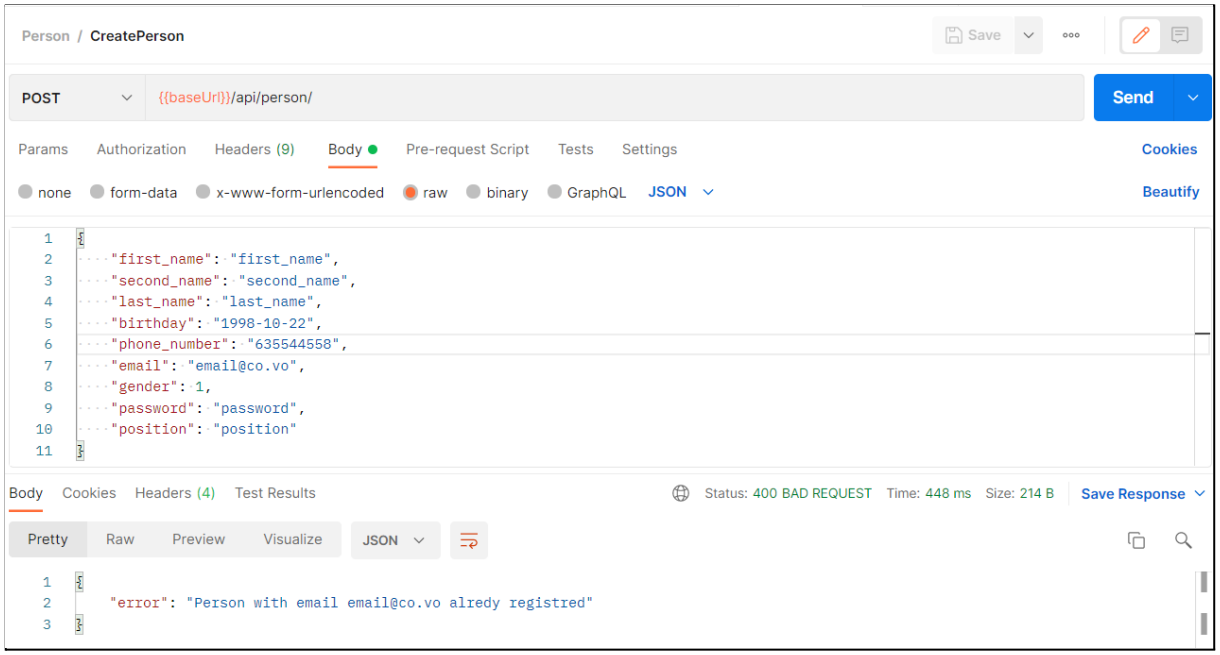


Рисунок 6.10 - Тестування додавання персони, яка дублює електронну пошту

Якщо змінити пошту, та надіслати запит ще раз, буде отримано повідомлення з помилкою, що в базі вже існує такий номер телефону (рис. 6.11).

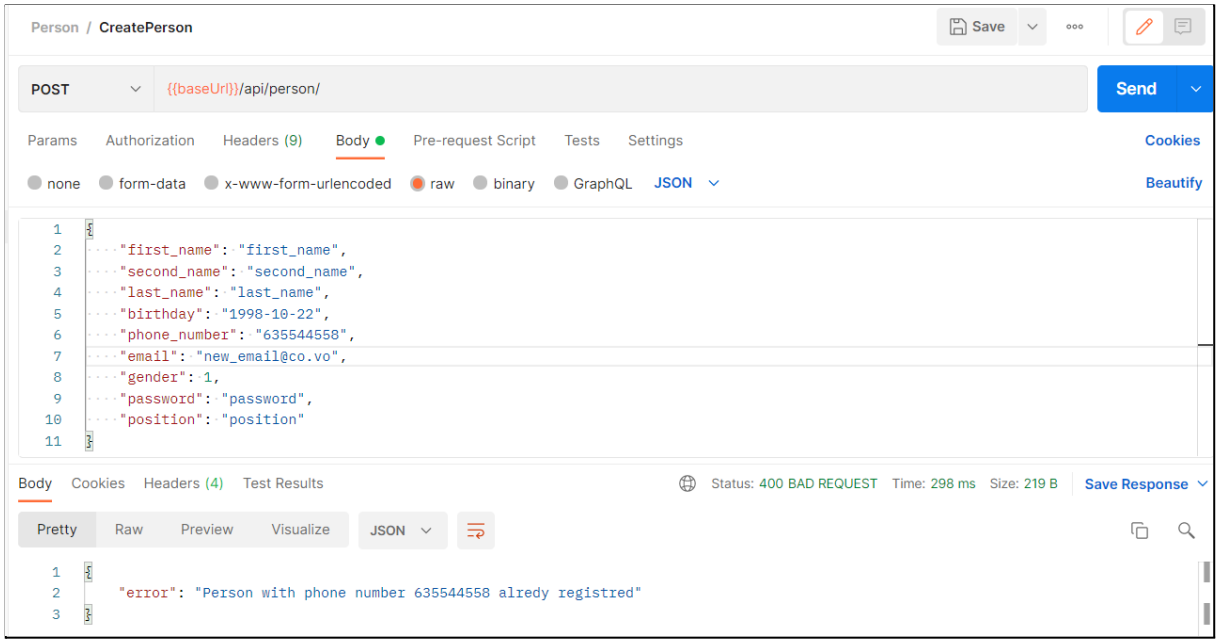


Рисунок 6.11 - Тестування додавання персони, яка дублює номер телефону

Також треба протестувати додавання персони з токеном без прав адміністратора. На рисунку 6.12 продемонтрован результат додавання нової персони від персони, яка не адміністратор.

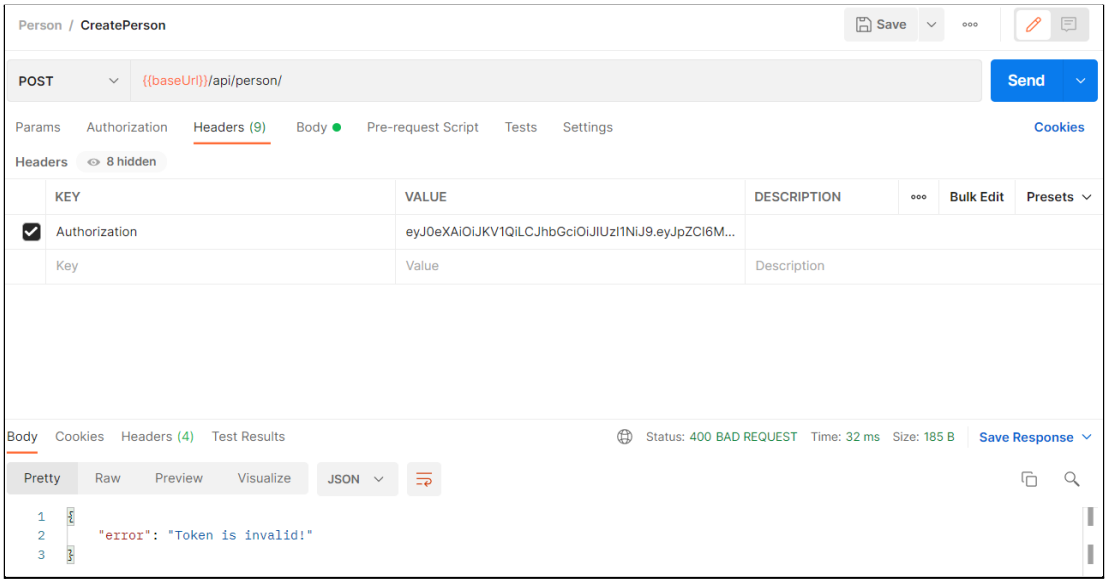


Рисунок 6.12 - Результат тесту додавання персони без прав адміністратора

Також треба протестувати випадок додавання персони без токену авторизації. На рисунку 6.13 продемонтрован тестовий випадок, коли не передається токен.

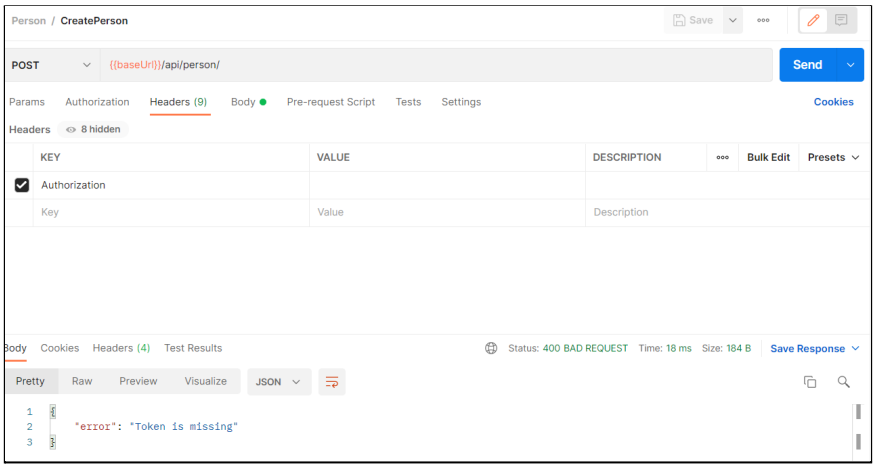


Рисунок 6.13 - Тестування методу додавання персони від користувача без токену

Також для додавання персони був створений валідатор, в якому вказано які параметри персони є обов’язковими, а які - опціональними. Для тестування валідації даних будемо передавати всі дані пустими. Сервер повинен повернути помилки валідації, як зображено на рисунку 6.14.

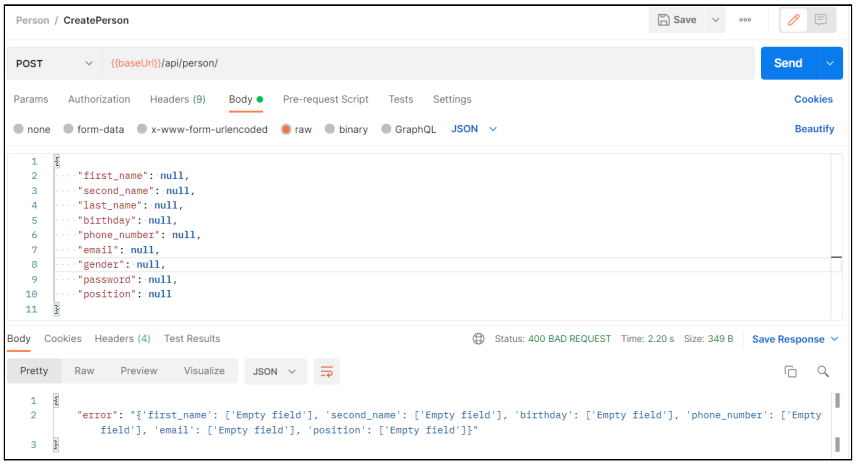


Рисунок 6.14 - Тестування додавання персони з пустими параметрами

Валідатор повернув помилки, що поля з ім’ям, прізвищем, днем народження, номером телефона, електронною поштою та посадою нового користувача повинні бути обов’язково заповнені.

Під час тестування методу додавання нової персони, непередбаченої поведінки серверу не було виявлено.

6.4 Тестування методу видалення персони

Поточна реалізація системи передбачає видалення персони тільки тоді, коли ідентифікатор користувача, який робить запрос, співпадає з персоной, яку потрібно видалити. Також користувач потрібен бути авторизованим у системі.

Для тесту видалемо користувача, якого додали у попередньому тесті. Для цього надішлемо запит як продемонтровано на рисунку 6.15.

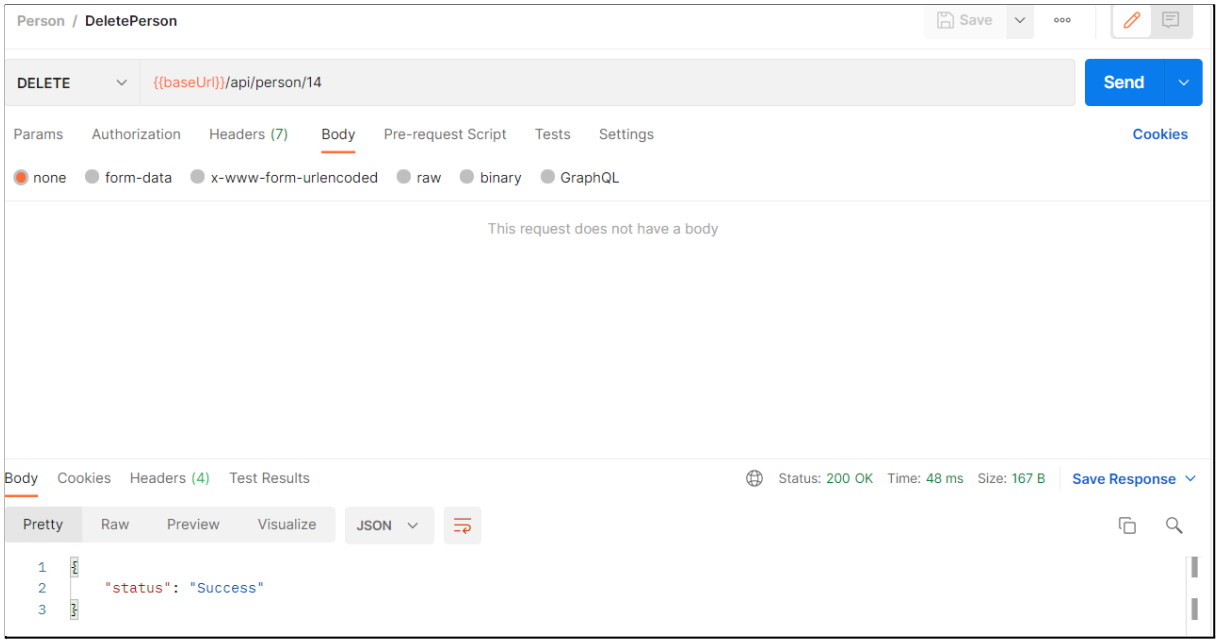
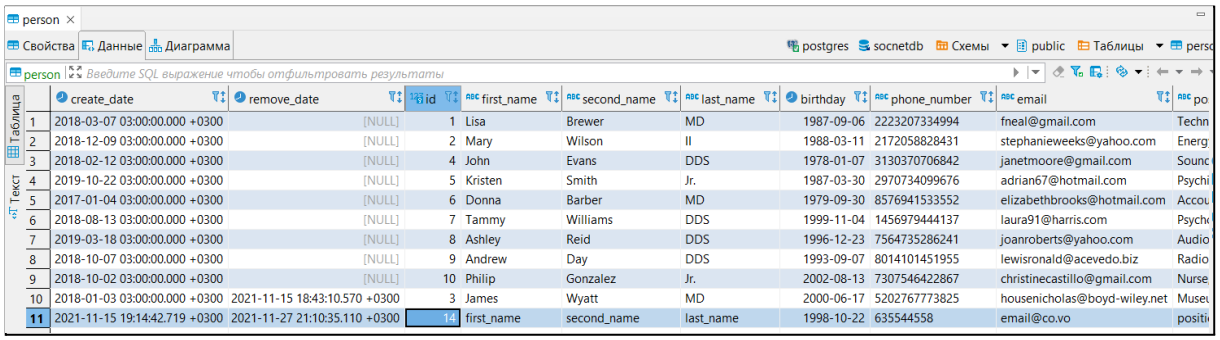


Рисунок 6.15 - Тестування методу видалення персони

Для перевірки того, що персону було видалено, подивимось таблицю person в базі даних (рис. 6.16).



person

Свойства | Данные | Диаграмма | postgres | socnetdb | Схемы | public | Таблицы

person — Введите SQL выражение чтобы отфильтровать результаты

	create_date	remove_date	id	first_name	second_name	last_name	birthday	phone_number	email	po
1	2018-03-07 03:00:00.000 +0300	[NULL]	1	Lisa	Brewer	MD	1987-09-06	2223207334994	fneal@gmail.com	Techn
2	2018-12-09 03:00:00.000 +0300	[NULL]	2	Mary	Wilson	II	1988-03-11	2172058828431	stephanieweeks@yahoo.com	Energ
3	2018-02-12 03:00:00.000 +0300	[NULL]	4	John	Evans	DDS	1978-01-07	3130370706842	janetmoore@gmail.com	Sounc
4	2019-10-22 03:00:00.000 +0300	[NULL]	5	Kristen	Smith	Jr.	1987-03-30	2970734099676	adrian67@hotmail.com	Psychi
5	2017-01-04 03:00:00.000 +0300	[NULL]	6	Donna	Barber	MD	1979-09-30	8576941533552	elizabethbrooks@hotmail.com	Accou
6	2018-08-13 03:00:00.000 +0300	[NULL]	7	Tammy	Williams	DDS	1999-11-04	1456979444137	laura91@harris.com	Psych
7	2019-03-18 03:00:00.000 +0300	[NULL]	8	Ashley	Reid	DDS	1996-12-23	7564735286241	joanroberts@yahoo.com	Audio
8	2018-10-07 03:00:00.000 +0300	[NULL]	9	Andrew	Day	DDS	1993-09-07	8014101451955	lewisronald@acevedo.biz	Radio
9	2018-10-02 03:00:00.000 +0300	[NULL]	10	Philip	Gonzalez	Jr.	2002-08-13	7307546422867	christinecastillo@gmail.com	Nurse
10	2018-01-03 03:00:00.000 +0300	2021-11-15 18:43:10.570 +0300	3	James	Wyatt	MD	2000-06-17	5202767773825	housenicholas@boyd-wiley.net	Muse
11	2021-11-15 19:14:42.719 +0300	2021-11-27 21:10:35.110 +0300	14	first_name	second_name	last_name	1998-10-22	635544558	email@co.vo	positi

Рисунок 6.16 - Таблиця person в базі даних

В таблиці ми бачимо, що для персони с ідентифікатором 14 було встановлено дату видалення, тобто вона враховується видаленою для всіх запитів. Для демонстрації, що персона дійсно була видалена, спробуємо авторизуватись з її електронною поштою (рис. 6.17).

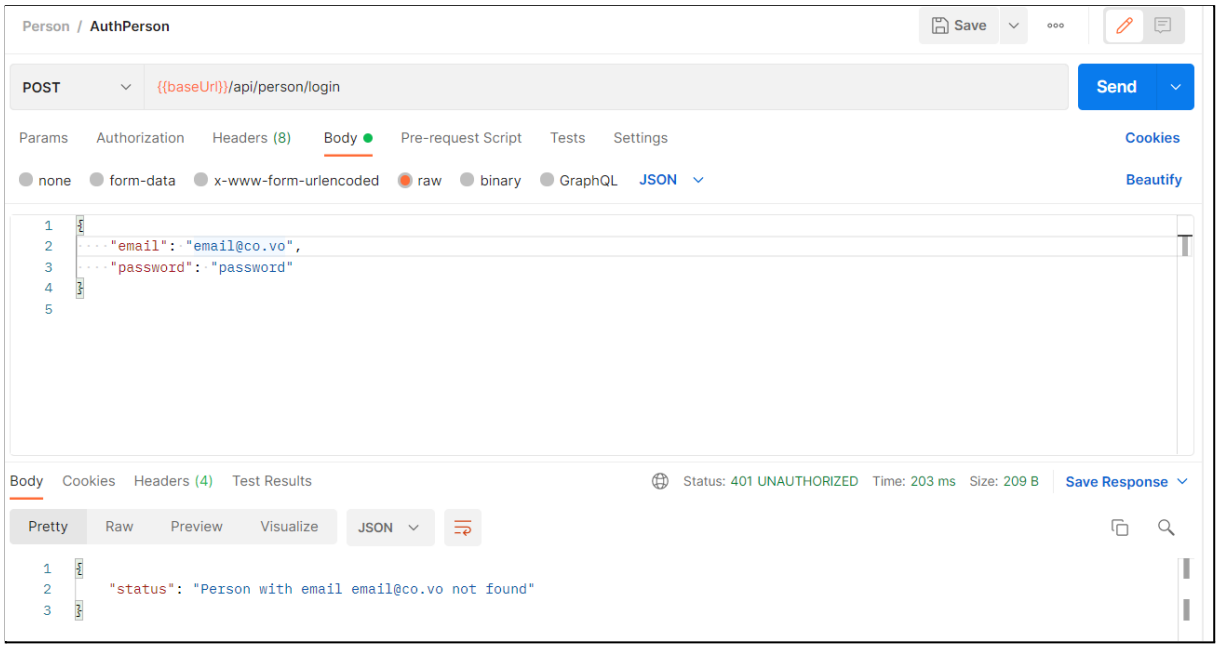


Рисунок 6.17 - Спроба авторизації з поштою видаленної персони

Зараз потрібно перевірити можливість видалення персони, чий ідентифікатор відрізняється від ідентифікатору поточної персони. Для цього авторізуємость як

персона з ідентифікатором 2, та спробуємо видалити персону с ідентифікатором 1 (рис. 6.18).

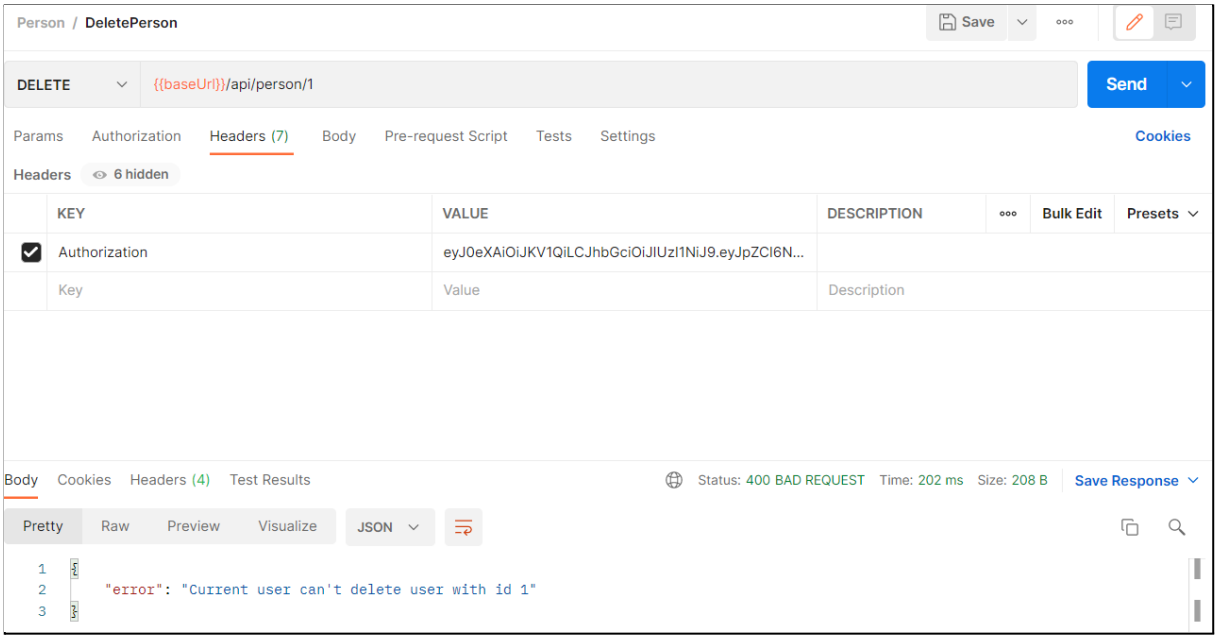


Рисунок 6.18 - Спроба видалення персони від користувача чиї ідентифікатори відрізняються

Під час тестування методу видалення персони непередбаченої поведінки серверу помічено не було.

6.5 Тестування методів створення, видалення та отримання навичок

Для використання методів навичок, користувач потрібен бути авторизованим. Для додавання навички потрібно в тілі запиту вказати її ім’я, як це продемонстровано на рисунку 6.19.

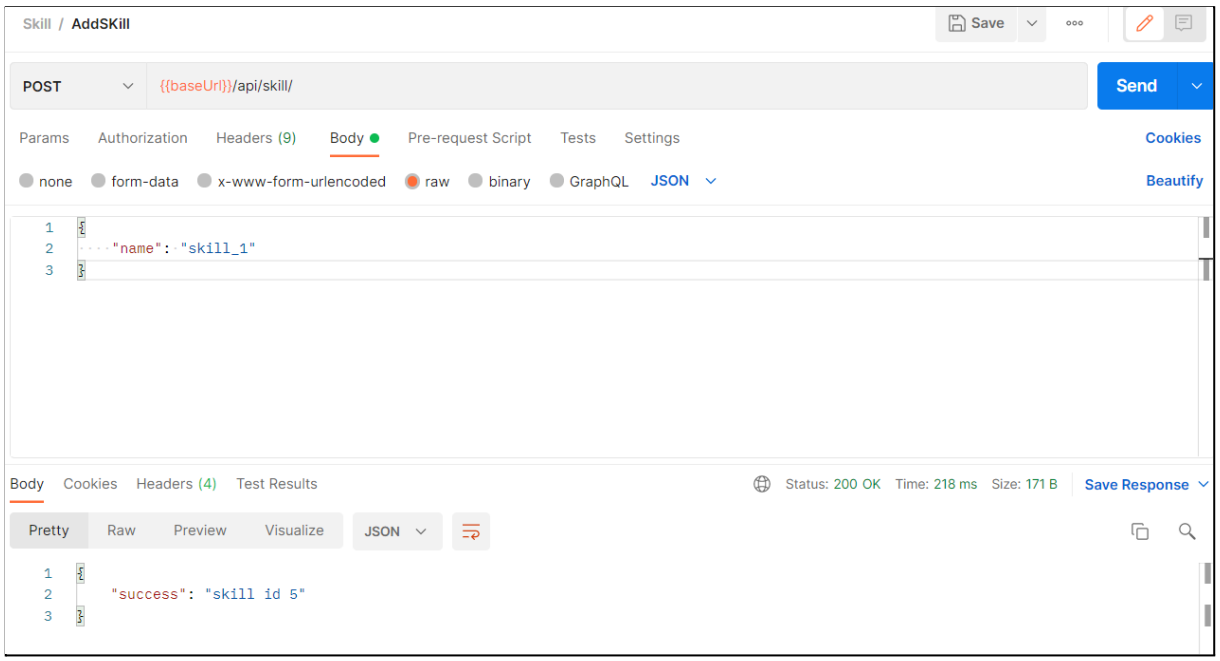


Рисунок 6.19 - Тестування метода додавання навика

Тепер розглянемо випадок, коли неавторизований користувач намагається додати новий навик. Сервер повинен повернути помилку перевірки токену (рис. 6.20).

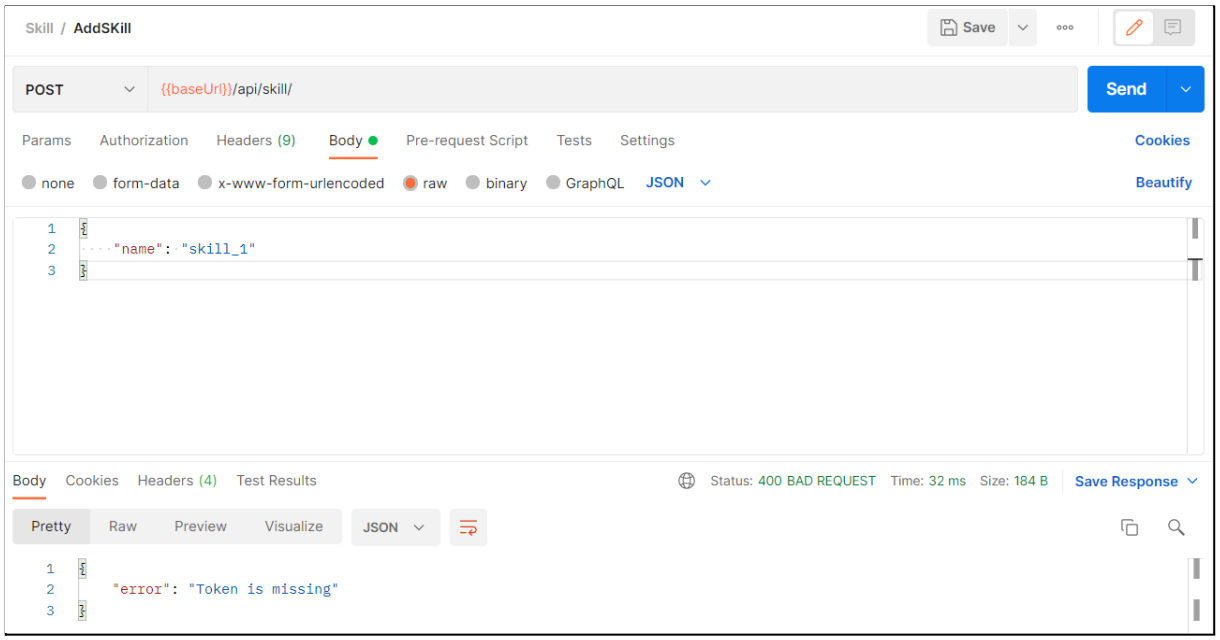


Рисунок 6.20 - Тестування додавання навику без токену авторизації

Тепер потрібно протестувати метод отримання новички за ідентифікатором. Якщо користувач має токен авторизации, він отримує назву навички та її ідентифікатор (рис. 6.21).

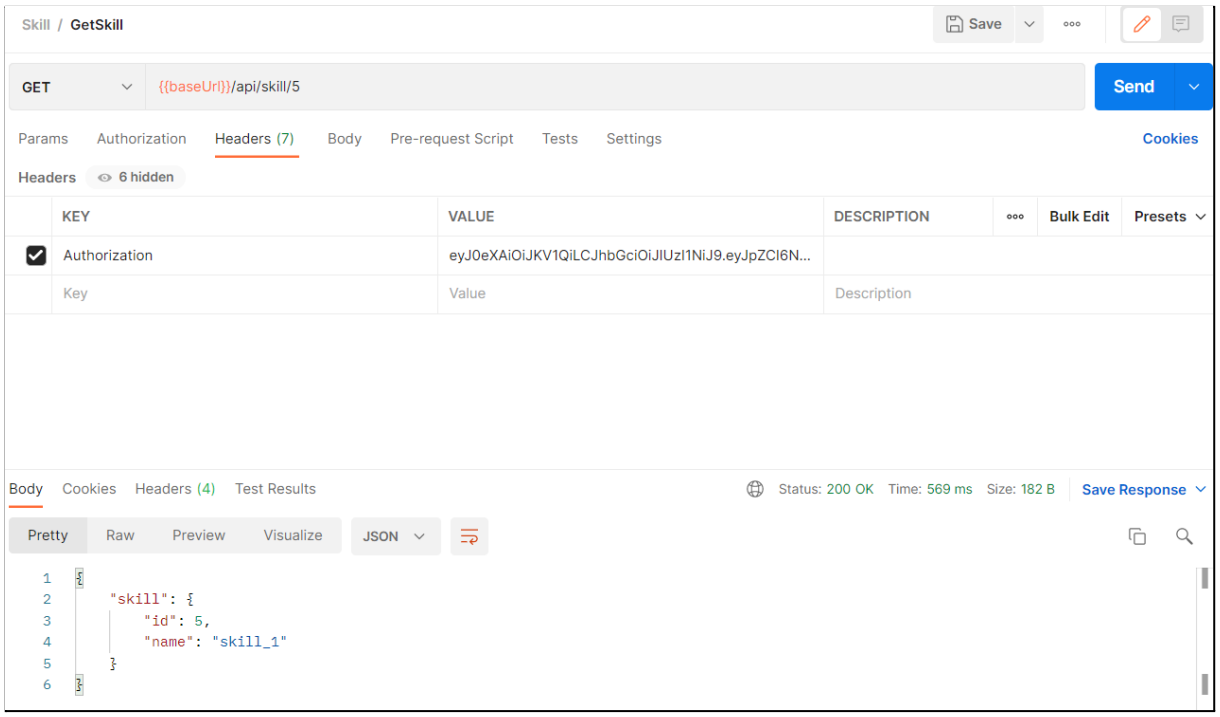


Рисунок 6.21 - Перегляд навику за ідентифікатором

Далі розглянемо метод отримання навичок. Також можна в запиті вказати яку сторінку відобразити, скільки записів потрібно бути на сторінці та також поле сортування (рис. 6.22).

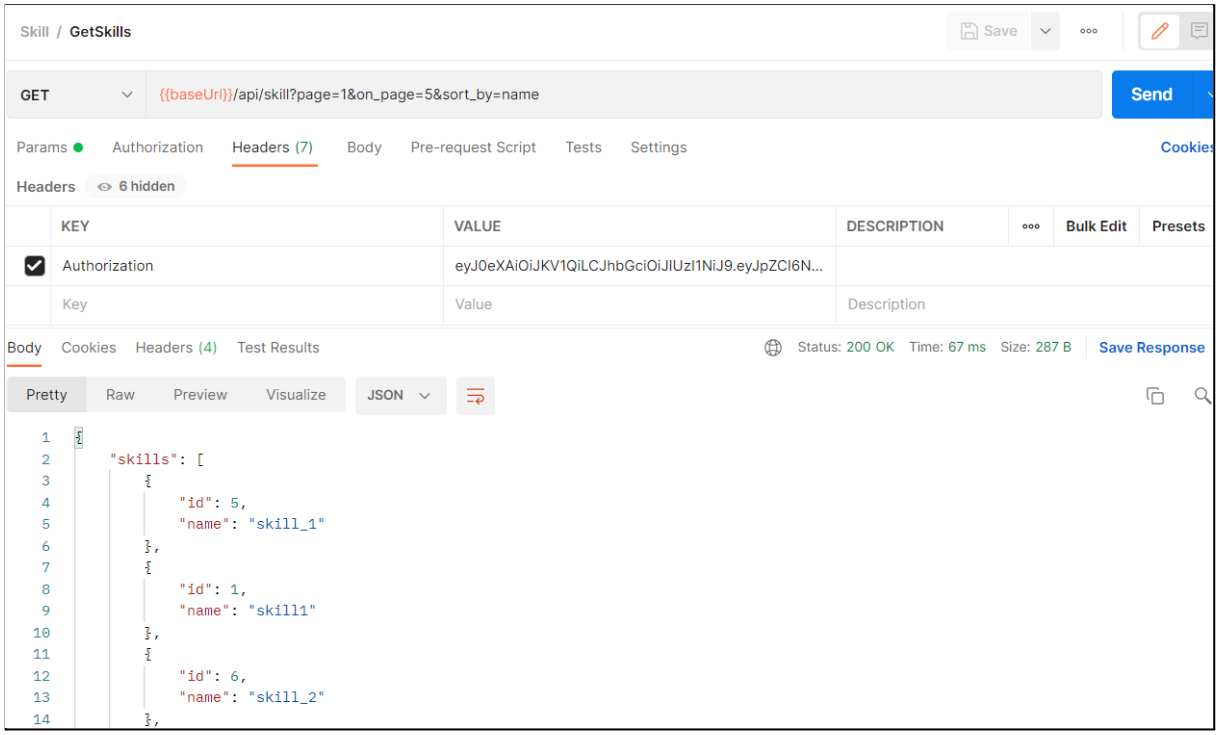


Рисунок 6.22 - Перегляд навичок з параметрами відображення

Зараз треба протестувати метод видалення запису. Якщо користувач має права адміністратора, тест буде видалено (рис. 6.23).

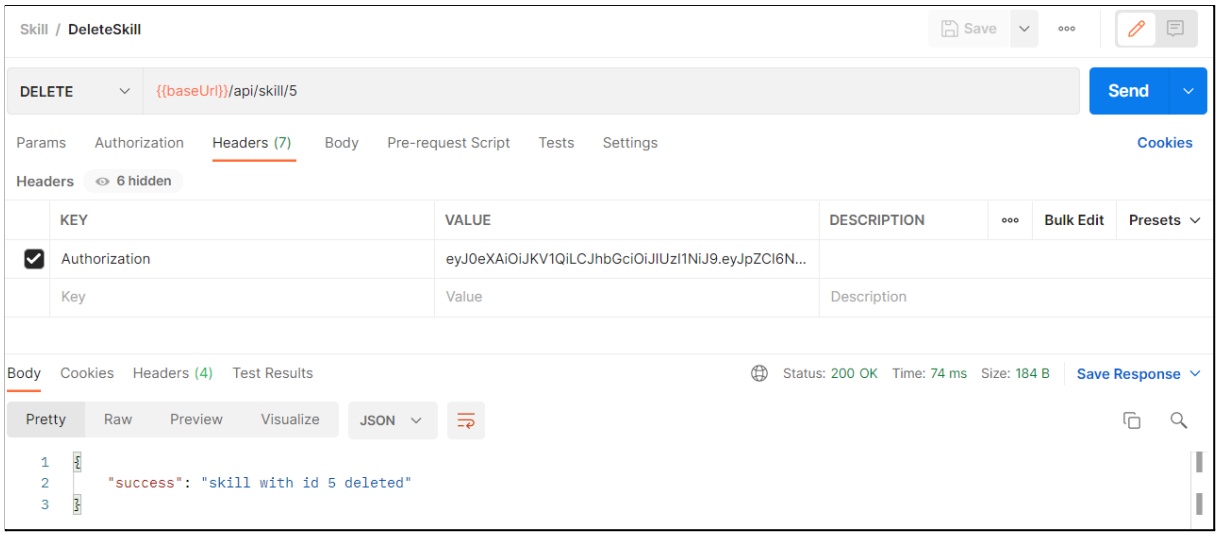


Рисунок 6.23 - Видалення навику користувачем з правами адміністратора

Якщо подивитись на рисунок 6.24 можна помітити, що навик має дату видалення, тому він більше не буде відображатись в списку навичок, а також його неможливо отримати за ідентифікатором.

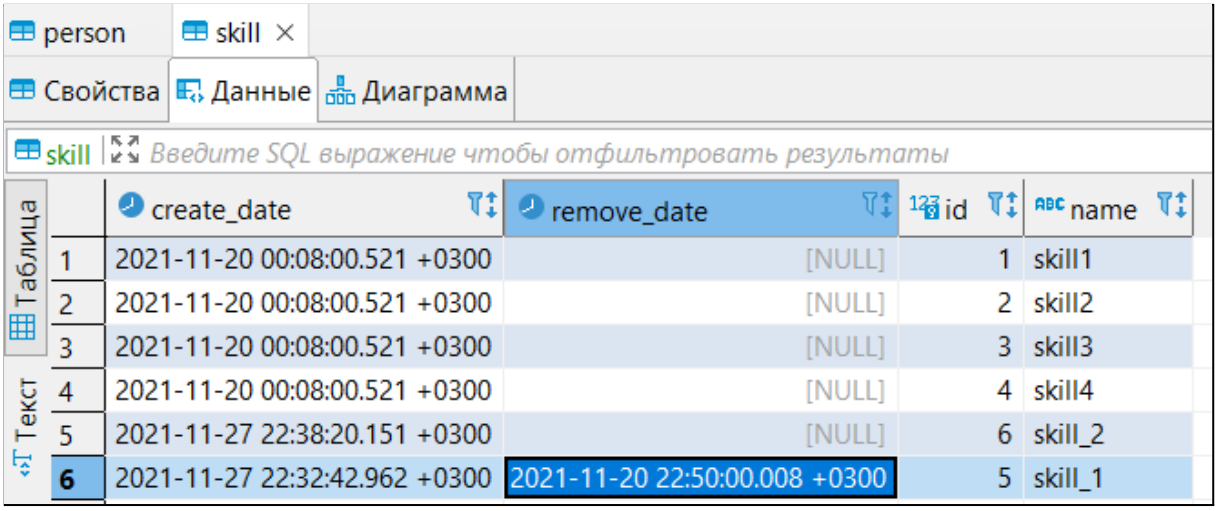


Рисунок 6.24 - Таблиця навичок

Але якщо користувач не має прав на видалення, він отримує певне повідомлення (рис. 6.25).

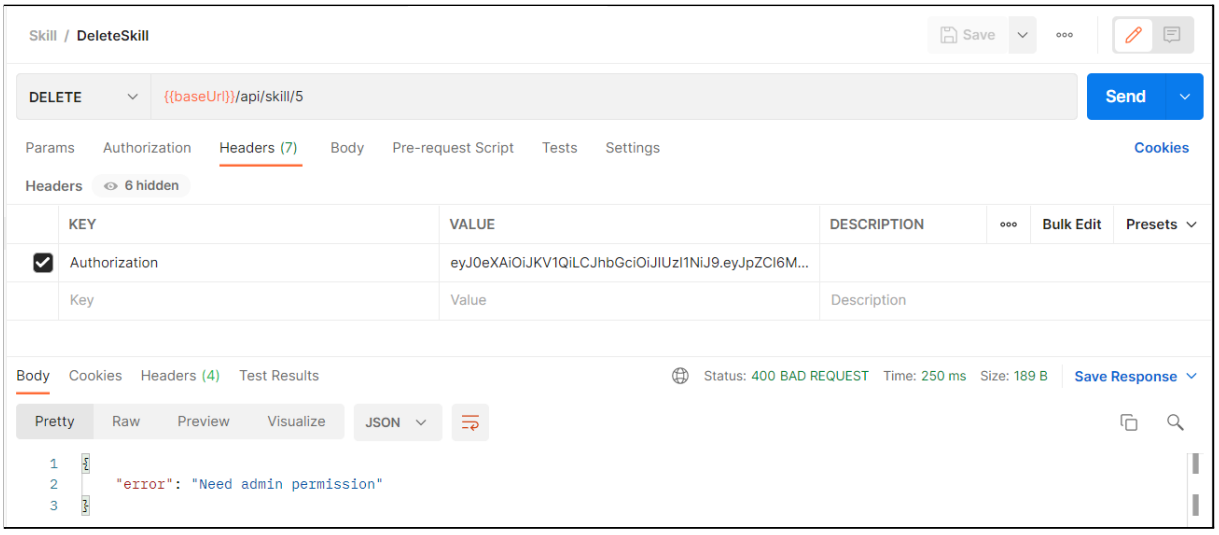


Рисунок 6.25 - Повернення помилки перевірки токену

Під час тестування методів перегляду, додавання та видалення навичок непередбаченої поведінки серверу не було виявлено.

6.6 Тестування методів додавання тестів

Для додавання тестів користувач повинен бути авторизован, також для тесту є валідатор, який перевіряє які дані є обов’язковими - ідентифікатор навику, ідентифікатор творця, назву, максимальну кількість очок, та перевіряє щоб для кожного блоку питань була правильна відповідь.

Для початку треба додати тест, який має правильні дані:

```
{
    "skill_id": 1,
    "name": "TestTest",
    "max_point": 10,
    "questions": [
        {
            "question": "Question 1",
            "answer_1": "7",
            "answer_2": "2",
            "answer_3": "9",
            "answer_4": "4",
            "correct_answer": "answer_4"
        },
        {
            "question": "Question 2",
            "answer_1": "2",
            "answer_2": "4",
            "answer_3": "6",
            "answer_4": "9",
            "correct_answer": "answer_3"
        }
    ]
}
```

та отримуємо результат, як зображено на рисунку 6.26.

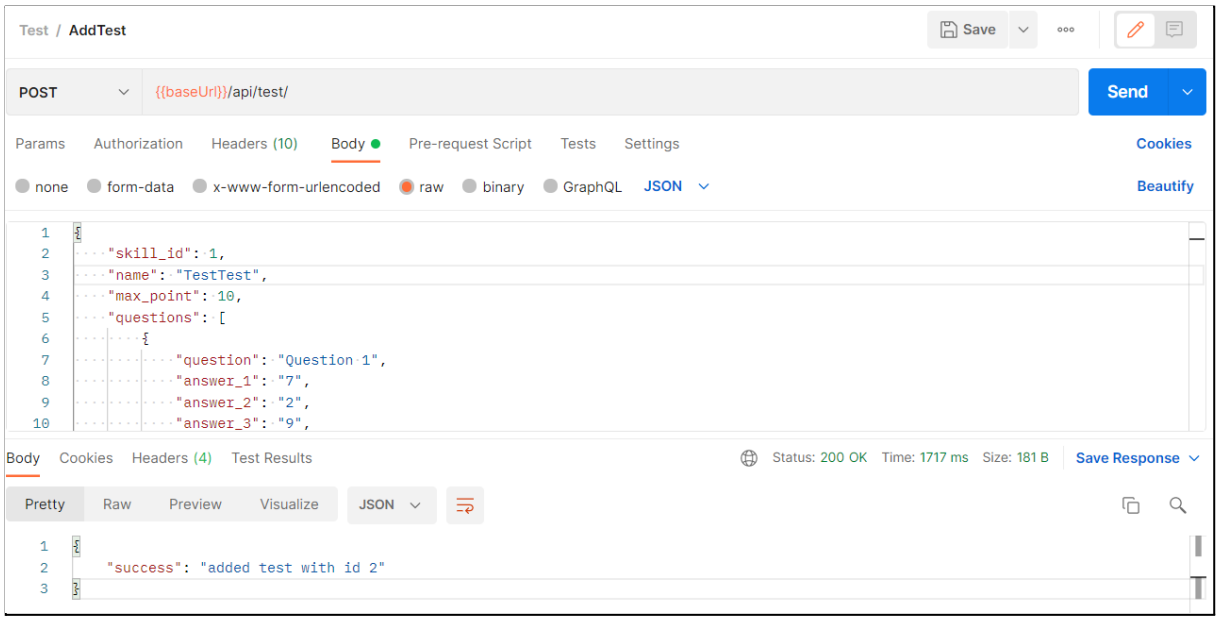


Рисунок 6.26 - Тестування методу додавання тесту

Тепер проведемо тестування - спробуємо додати тест без правильних відповідей:

```
{
    "skill_id": 1,
    "name": "UnCorrect",
    "max_point": 10,
    "questions": [
        {
            "question": "Question 1",
            "answer_1": "7",
            "answer_2": "2",
            "answer_3": "9",
            "answer_4": "4"
        },
        {
            "question": "Question 2",
            "answer_1": "2",
            "answer_2": "4",
            "answer_3": "6",
            "answer_4": "9"
        }
```

```
    ]
}
```

та був отриманий результат, який продемонтрован на рисунку 6.27.

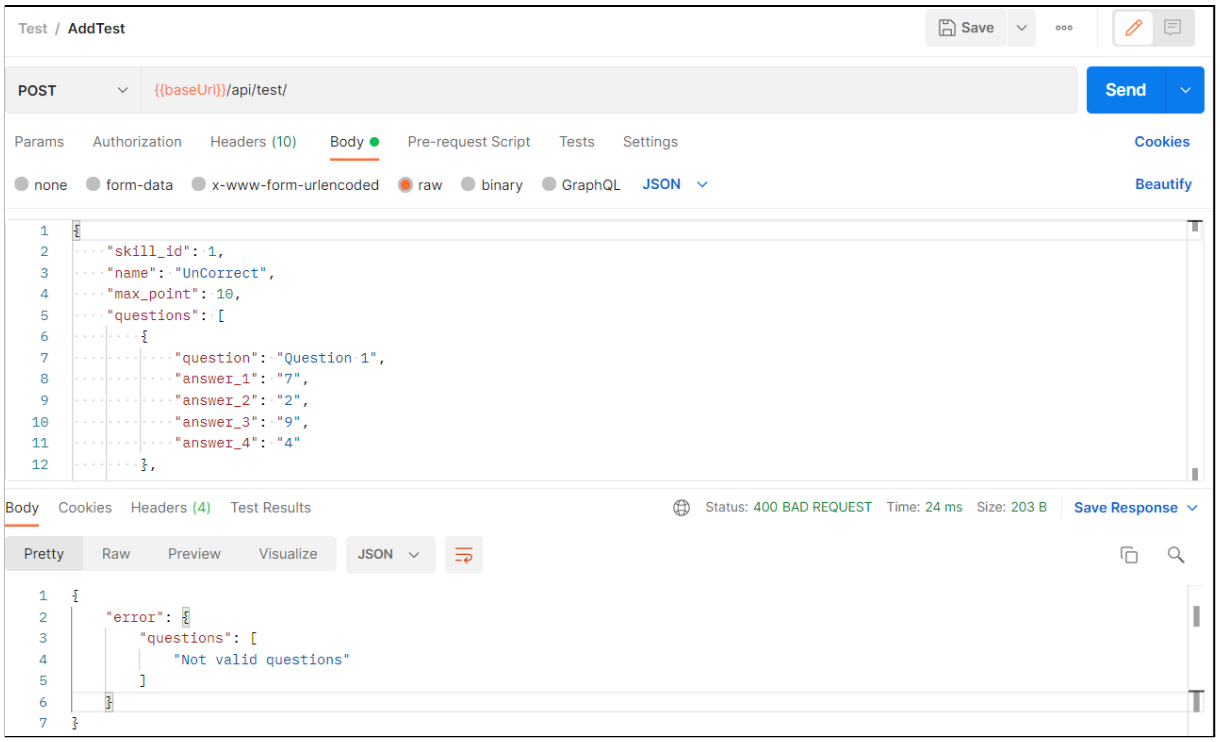


Рисунок 6.27 - Негативна відповідь на спробу додавання некоректної структури тесту

Також треба провести тестування методу додавання тесту з пустими значеннями для всіх полей, крім питань:

```
{
    "skill_id": null,
    "name": null,
    "max_point": null,
    "questions": [
        {
            "question": "Question 1",
            "answer_1": "7",
            "answer_2": "2",
            "answer_3": "9",
            "answer_4": "4",
```

```
            "correct_answer": "answer_4"
        },
        {
            "question": "Question 2",
            "answer_1": "2",
            "answer_2": "4",
            "answer_3": "6",
            "answer_4": "9",
            "correct_answer": "answer_3"
        }
    ]
}
```

Після того, як цей запит був оброблен, сервер повернув відповідь, котру продемонтровано на рисунку 6.28.

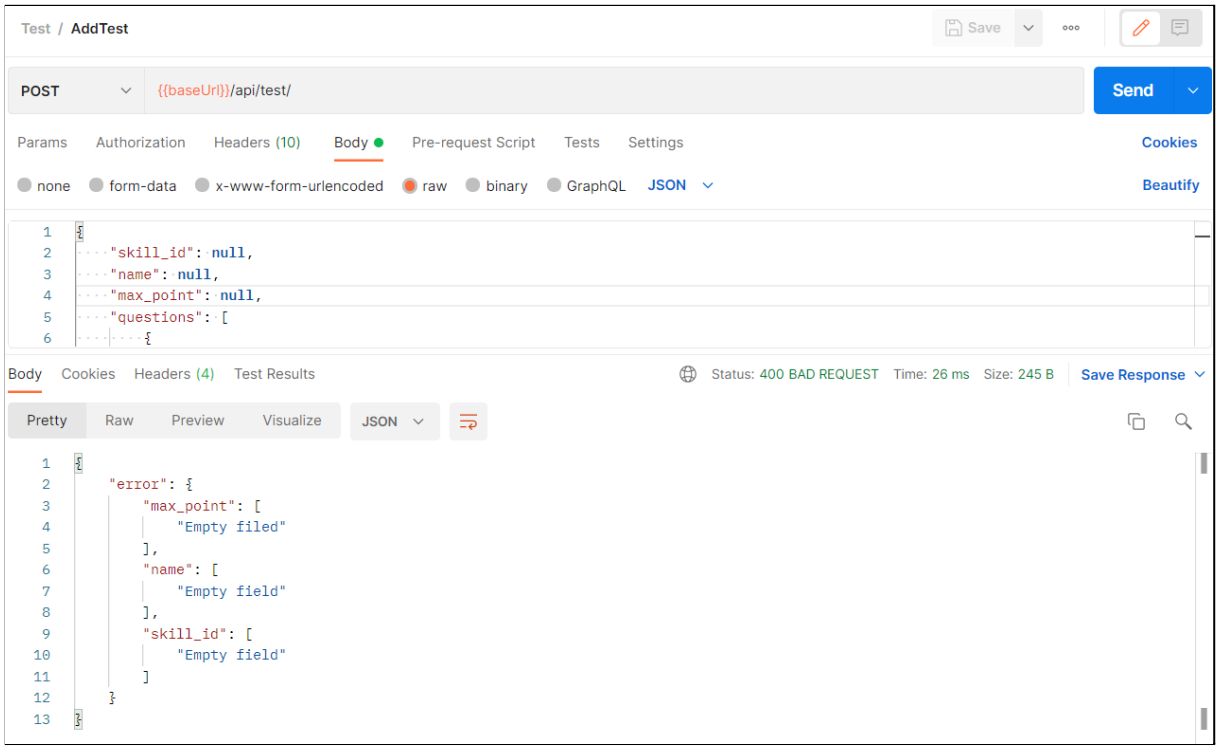


Рисунок 6.28 - Повернення помилок валідації

Під час тестування не було виявлено непередбаченої поведінки серверу.

6.7 Тестування методів перегляду та видалення тестів

Для перегляду тестів потрібен токен. А видалити тест може тільки користувач, який його створив.

Спочатку треба провести тест видалення запису. Спеціально для цього був доданий ще один тест. На рисунку 6.29 зображено таблиця тестів.

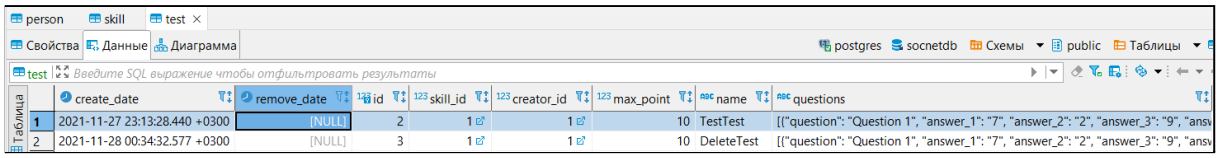


Рисунок 6.29 - Перелік існуючих тестів

Для того, щоб видалити тест, потрібно виконати запит, як зображано на рисунку 6.30.

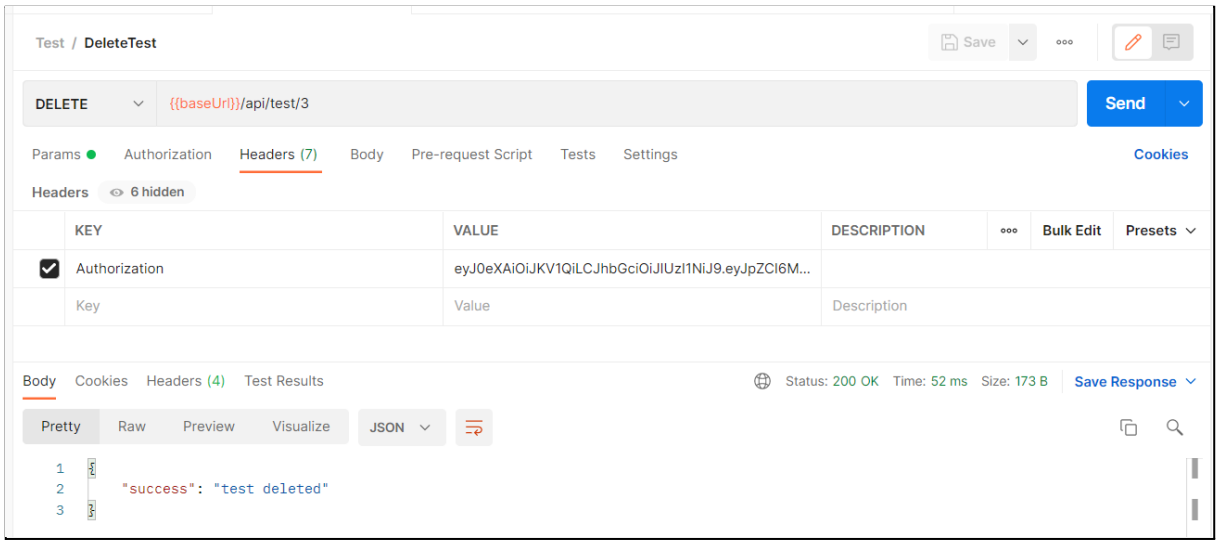
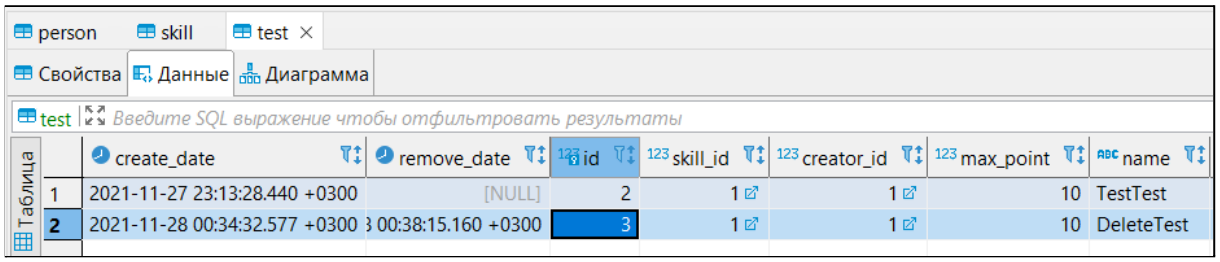


Рисунок 6.30 - Видалення тесту

Для перевірки видалення тесту звернемось до бази даних. На рисунку 6.31 ми бачимо, що тест з ідентифікатором 3 має дату видалення.



6.31 - Перелік тестів в таблиці

Тепер протестуємо запит для отримання тестів. Головне - коли сервер відправляє дані про тест користувачу, потрібно відправляти тільки питання без правильних відповідей. На рисунку 6.32 зображено отримання тесту за ідентифікатором.

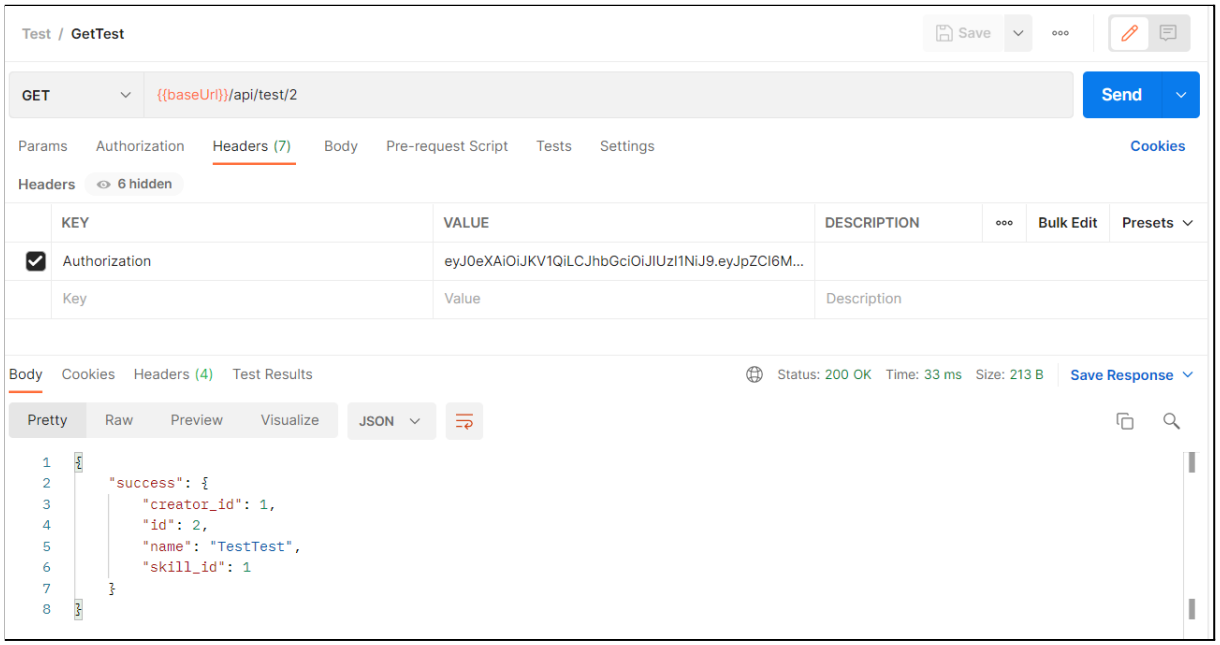


Рисунок 6.32 - Отримання тесту за ідентифікатором

На рисунку 6.33 зображено результат запиту, коли користувач намагається отримати тест, який помічений як видалений.

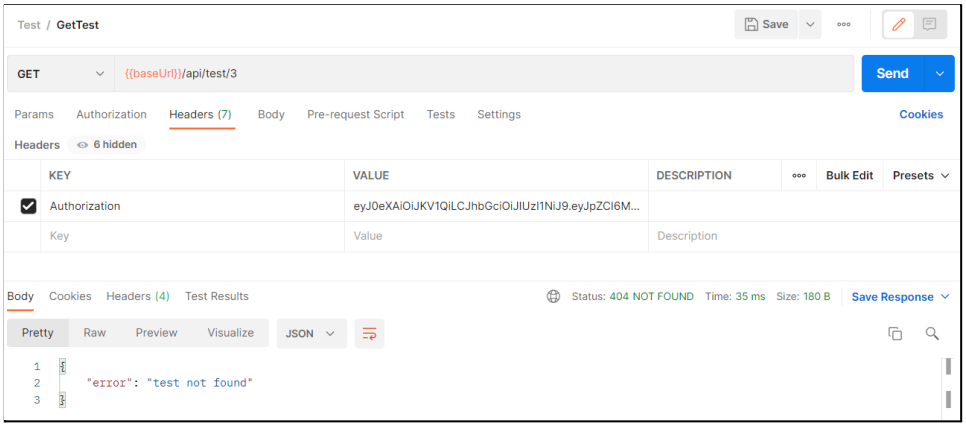


Рисунок 6.33 - Спроба отримати видалений тест

Для перегляду тестів потрібно вказати яку сторінку показати та скільки записів потрібно бути на сторінці. Результат демонстрації першої сторінки з двома записами продемонстровано на рисунку 6.34.

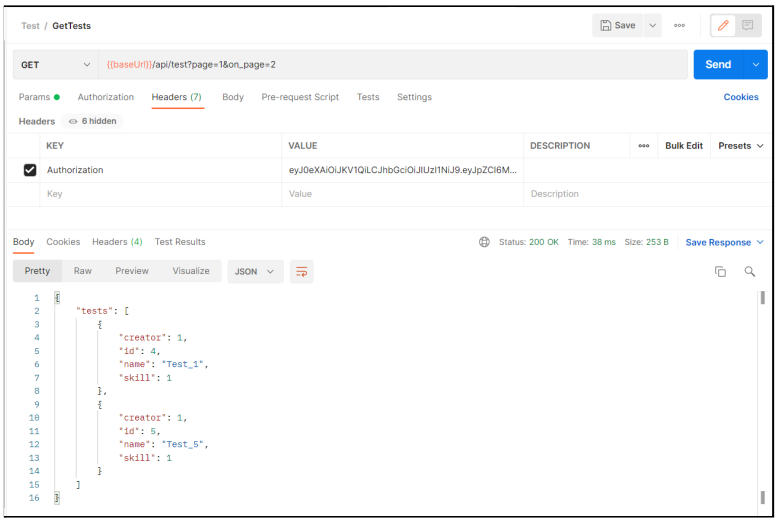


Рисунок 6.34 - Перегляд двох тестів

А якщо в параметрі on_page передати 5, то результат буде таким:

```
{
    "tests": [
        {
            "creator": 1,
            "id": 4,
            "name": "Test_1",
```

```
            "skill": 1
        },
        {
            "creator": 1,
            "id": 5,
            "name": "Test_5",
            "skill": 1
        },
        {
            "creator": 1,
            "id": 6,
            "name": "Test_6",
            "skill": 1
        },
        {
            "creator": 1,
            "id": 7,
            "name": "Test_7",
            "skill": 1
        },
        {
            "creator": 1,
            "id": 2,
            "name": "TestTest",
            "skill": 1
        }
    ]
}
```


Під час тестування не було виявлено непередбаченої поведінки серверу.

6.8 Тестування методу проходження тесту

Для проходження тесту користувач повинен бути авторизованим. Для початку проходження тесту слід надіслати запит, котрий повертає тест у вигляді питань. Після завершення надіслати ще один запит, в якому треба надіслати відповіді. Далі сервер опрацює запит, перевірить відповіді та поверне скільки очків набрав користувач. На початку викликаємо запит для старту теста (рис. 6.35).

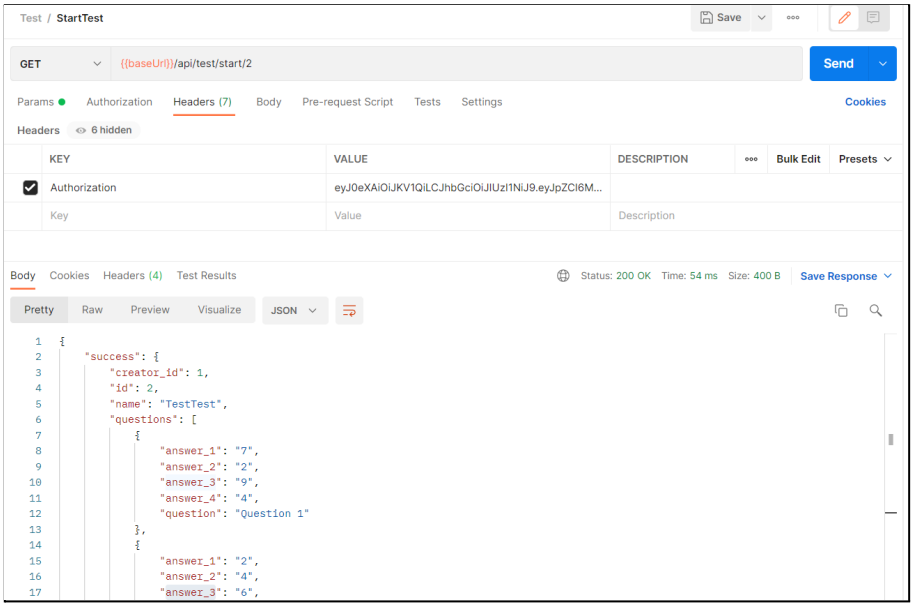


Рисунок 6.35 - Початок тесту

Після цього слід відправити ще один запит за відповідями. Після перевірки сервером правильності відповідей користувачу повинен повернутись результат тесту. Якщо він отримує очки, йому повертається кількість додаткових, якщо він отримує новий рівень, він отримує певне повідомлення.

Результат перевірки відповідей наведено на рисунку 6.36.

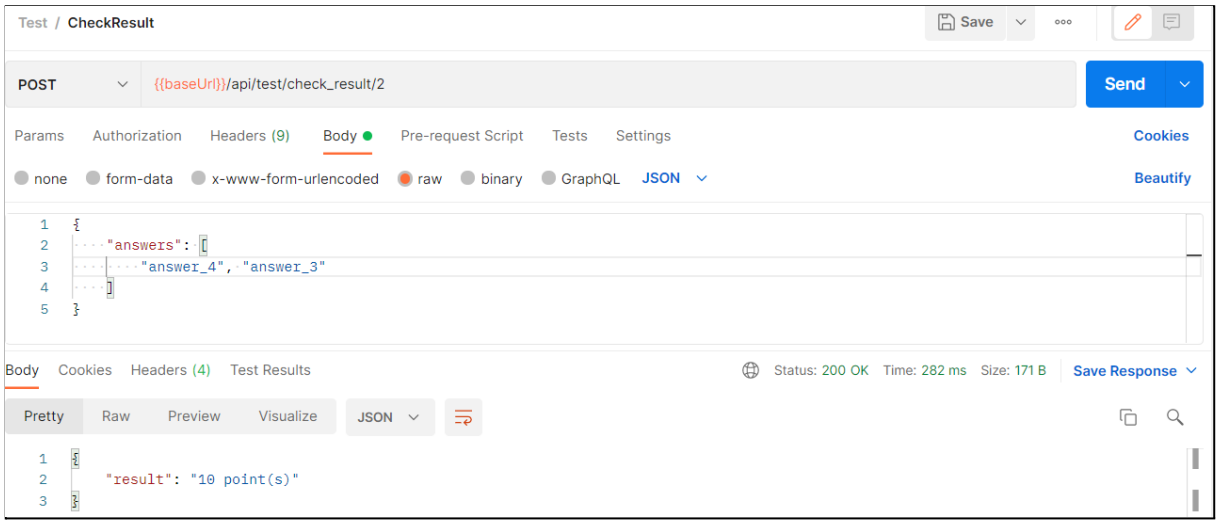


Рисунок 6.36 - Результат перевірки відповідей

Якщо спробувати відправити відповіді на тест, який ще не почали проходити, повернеться певне повідомлення (рис. 6.37).

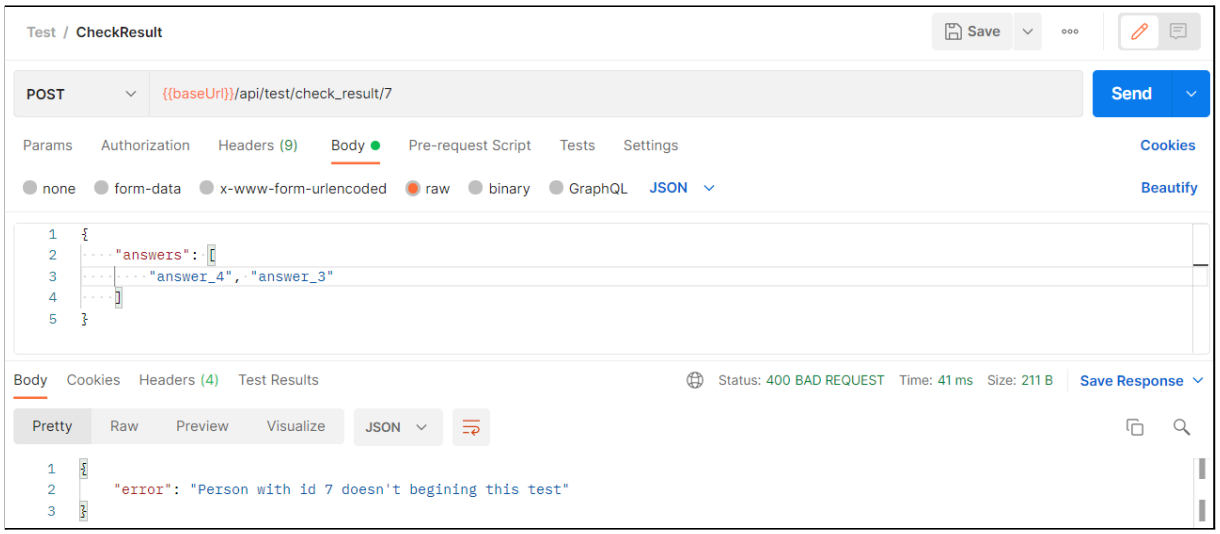


Рисунок 6.37 - Помилка при спробі відповісти на тест, який не починали проходити

При тестуванні методів проходження тестів не було виявлено непередбаченої поведінки серверу.

Висновок

В даному розділі був описаний процес інтеграційного тестування розробленої соціальної мережі. Було проведено тестування більшості прецедентів системи - авторизація, додавання нового користувача, перегляд та видалення користувачів, додавання, перегляд, видалення навичок, додавання, перегляд та видалення тестів, а також процес проходження тестів - старт проходження тесту, та перевірка результату тесту.

ВИСНОВОК

Ця робота була присвячена розробці серверної частини для приватний соціальної мережі для гейміфікації процессу професіонального розвитку співробітників компанії.

Метою роботи є підвищення кваліфікації співробітників шляхом додавання до процесу навчання формату гри. Цю мету дипломної роботи можно вважати досягненою так як при користуванні системою більш ніж 20 особами їх опитування показало, що 97% користувачів сподобався інтерфейс, та близько 93% сподобалось, методика навчання за допомогою гейміфікації.

Під час написання роботи було проведено аналіз предметної області та аналіз існуючих аналогів. На основі даних, отриманих з аналізу, були сформовані функціональни вимоги до системи. Після цього був описан кожен прецедент з функціональних вимог, сформовані нефункціональні вимоги.

Також був проведений опис взаємодії серверної частини та клієнтською, та наведена діаграма станів для серверної частини.

Після цього було виконано проектування, під час якого була створена схема архітектура серверної частини, діаграма програмних класів та діаграми послідовності, прототипування зовнішнього вигляду системи та її імплементація. Виконано проектування структури бази даних.

На етапі програмної реалізації системи були описані технології, які використовуються в написанні веб-застосування - мова програмування Python, фреймворк для веб-застосувань Flask, система управління базами даних PostgreSQL. Був наведен пример розгортання проекту за допомогою віртуального оточення Python та за допомогою Docker-контейнерів з базой даних.

На останньому етапі було проведено тестування наступних методів - авторизація, додавання нової персони, отримання персони за ідентифікатором, видалення персони; додавання, отримання та видалення навичок; додавання,

отримання, видалення тестів, а також тестування методу проходження тестів. Під час тестування непередбаченої повединки системи помічено не було.

Розроблене веб-застосування можна запустити як на ОС Windows, так і на дістрибутивах Linux завдяки крос-платформності мови програмування Python.

У майбутньому функціонал системи можна розширити шляхом додавання нових видів тестів, дороботкою існуючий системи рейтингу на більш гибку, додати нові ролі користувачів та змінити алгоритм додавання персони.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гуменникова Т. Р., Лугова, Т. А., Рященко, О. І., і Трояновська, Ю. Л. (2018). Інтеграція процесу розробки комп`ютерних ігор з доповненою реальністю у компоненти STREAM-освіти. Вісник сучасних інформаційних технологій, 49-61. вид. 2018. Вип 1(1). URL: http://hait.ccs.od.ua/index.php/journal/article/view/86 (дата звернення 19.10.2021).
2. В. А. Крісілов, К. О. Писаренко, і В. Н. Хуї, «Метод динамічного формування контенту в умовах обмежених ресурсів», вид. 2019, вип. 2, вип. 2, с. 89-105. URL: https://aait.opu.ua (дата звернення 18.10.2021).
3. Flask’s documentation. URL: https://flask.palletsprojects.com/en/2.0.x/ (дата звернення 16.09.2021).
4. SQLAlchemy 1.4 Documentation. URL: https://docs.sqlalchemy.org/en/14/ (дата звернення 18.09.2021)
5. Python URL: https://ru.wikipedia.org/wiki/Python (дата звернення 07.11.2021)
6. Flask URL: https://uk.wikipedia.org/wiki/Flask (дата звернення 07.11.2021)
7. PostgreSQL URL: https://uk.wikipedia.org/wiki/PostgreSQL (дата звернення 07.11.2021)
8. Тестування програмного забезпечення: https://uk.wikipedia.org/ wiki/Тестування_програмного_забезпечення (дата звернення 15.11.2021)
9. Модульне тестування: https://uk.wikipedia.org/wiki/ Тестування_програмного_забезпечення#Модульне_тестування (дата звернення 15.11.2021)
10. Інтеграційне тестування: https://uk.wikipedia.org/wiki/ Тестування_програмного_забезпечення#Інтеграційне_тестування (дата звернення 15.11.2021)
11. Postman Documentation: https://learning.postman.com/docs/ (дата звернення 16.11.2021)