Національний технічний університет України «КПІ»

Факультет інформатики та обчислювальної техніки

Кафедра Інформаційних систем та технологій

Лабораторна робота №4

з дисципліни « Сучасні технології розробки WEB-застосувань на платформі Microsoft.NET»

на тему: «Імплементація REST API»

Виконав:

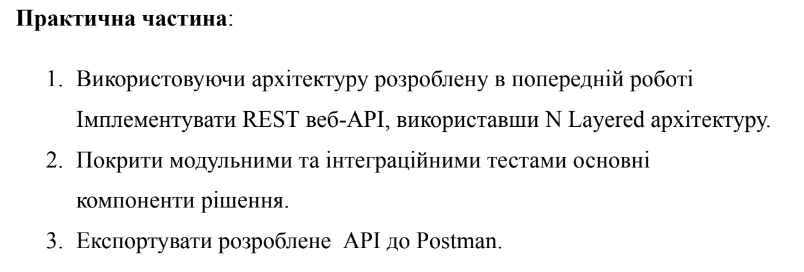
студент гр. ІС-11

Іваніцький М.

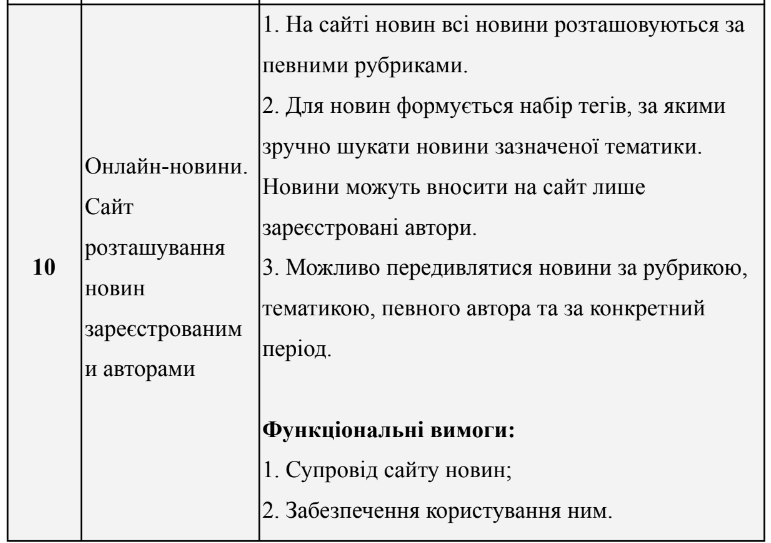
Викладач:

Бардін В.

2023 рік

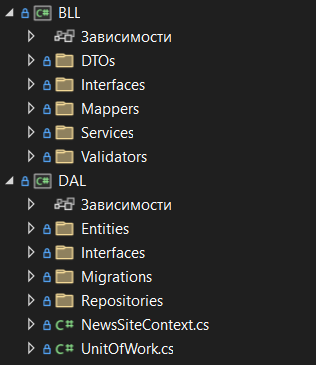
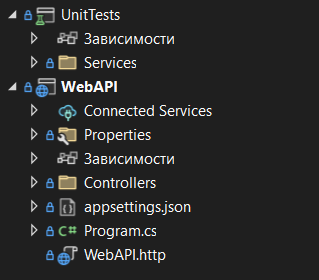
**Завдання:**

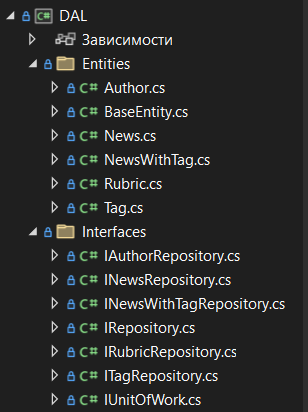
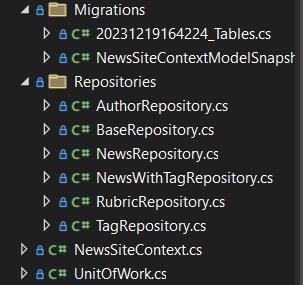
**Варіант:**

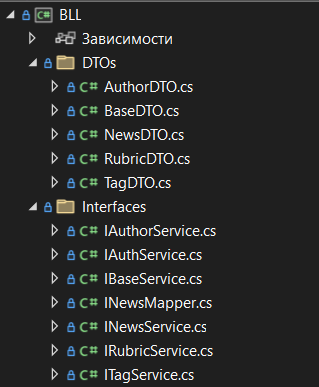
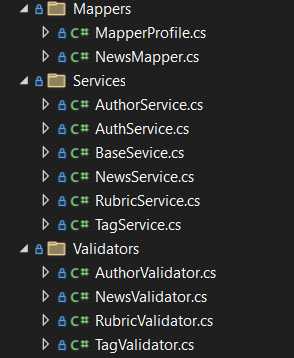
****

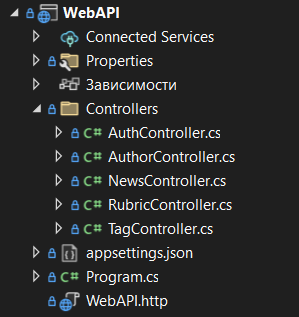
**Посилання на репозиторій:**

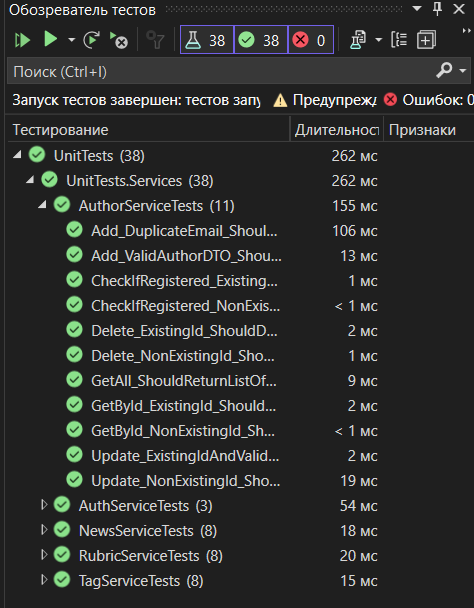
<https://github.com/Maksvell/WebApi_NewsSite>

**Огляд проекту:**

Data Access Layer:

Business Logic Layer:

Presentation Layer:

**Тестування:**

**Контролери:**

**AuthController:**

* (POST) api/register – Реєстрація автора;
* (POST) api/login – Вхід зареєестрованого автора в систему (генерація токена за допомогою якого можна в подальшому використовувати дії створення, редагування і тп);

**AuthorController:**

* (GET) api/authors – Отримати усіх авторів;
* (GET) api/authors/{id} – Отримати інформацію про конкретного автора;
* (DELETE) api/auhtors/{id} – Видалити конкретного автора;
* (PUT) api/authors/{id} – Оновити інформацію про конкртного автора;

**NewsController:**

* (GET) api/news – Отримати усі новини;
* (GET) api/news/{id} – Отримати конкретну новину;
* (GET) api/news/by-author/{id} – Отримати новини за конкретним автором;
* (GET) api/news/by-rubric/{id} – Отримати новини за конкретною рубрикою;
* (GET) api/news/by-tag/{id} – Отримати новини за конкретним тегом;
* (POST) api/news – Створити новину;
* (PUT) api/news/{id} – Оновити інформацію про конкретну новину;
* (DELETE) api/news/{id} – Видалити конкретну новину;

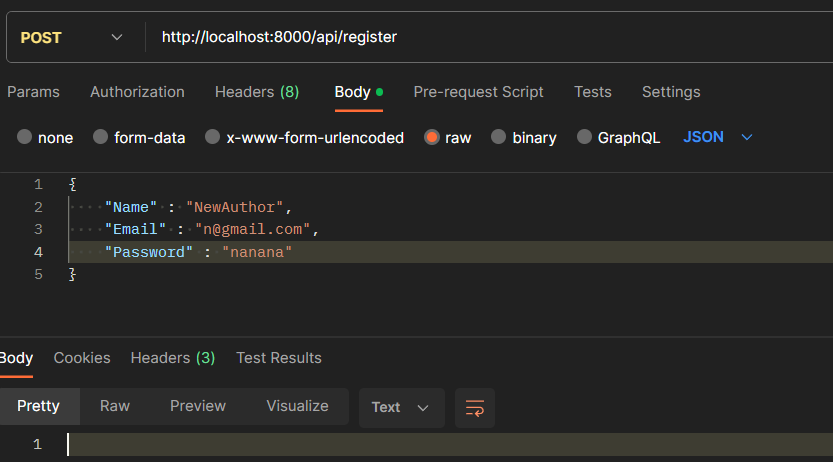
**RubricController:**

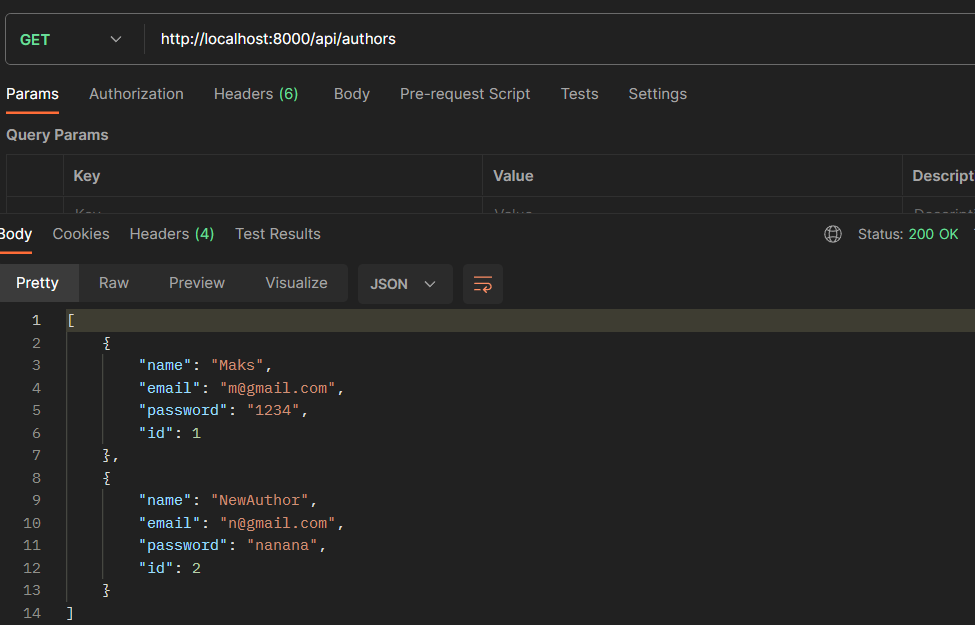
* (GET) api/rubrics – Отримати усіх авторів;
* (GET) api/rubrics/{id} – Отримати інформацію про конкретну рубрику;
* (POST) api/rubrics – Створити нову рубрику;
* (DELETE) api/rubrics/{id} – Видалити конкретну рубрику;
* (PUT) api/rubrics/{id} – Оновити інформацію про конкретну рубрику;

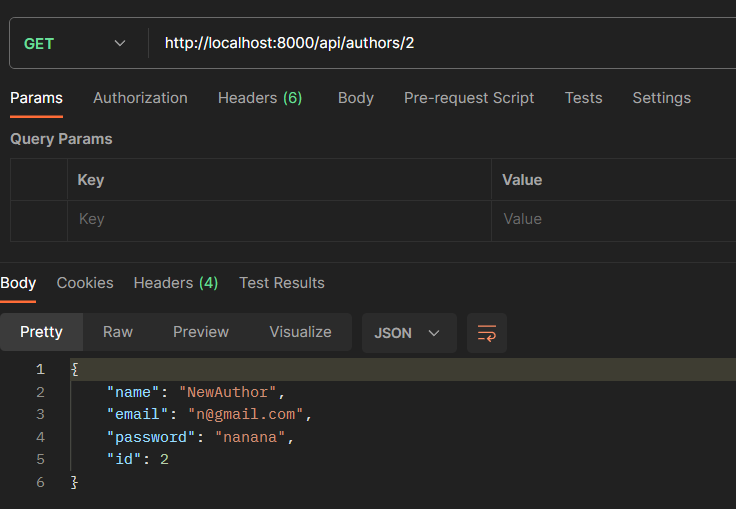
**TagController:**

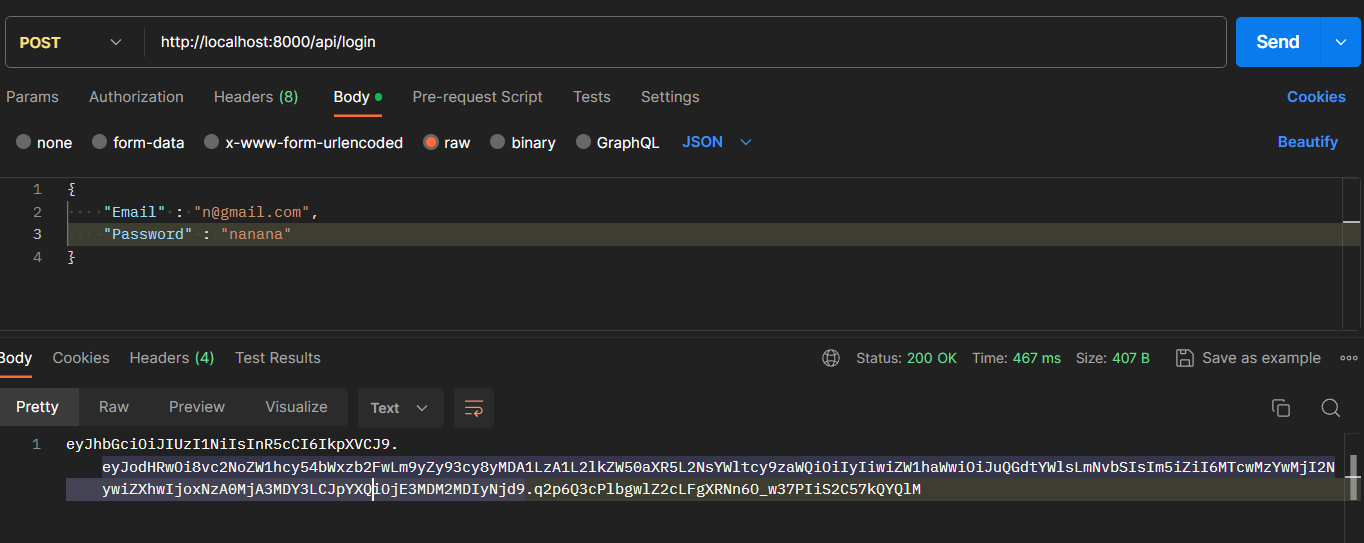
* (GET) api/tags – Отримати усіх авторів;
* (GET) api/tags/{id} – Отримати інформацію про конкретний тег;
* (POST) api/tags – Створити новий тег;
* (DELETE) api/tags/{id} – Видалити конкретний тег;
* (PUT) api/tags/{id} – Оновити інформацію про конкретний тег;

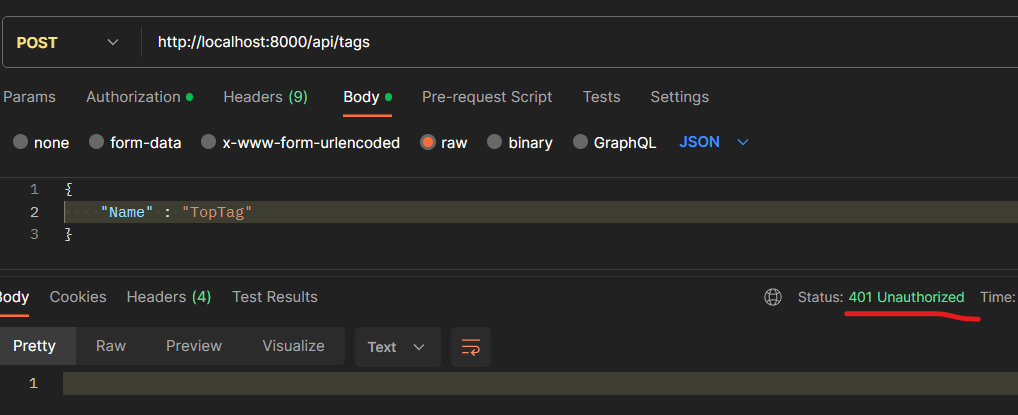
**Інтеграційне тестування:**

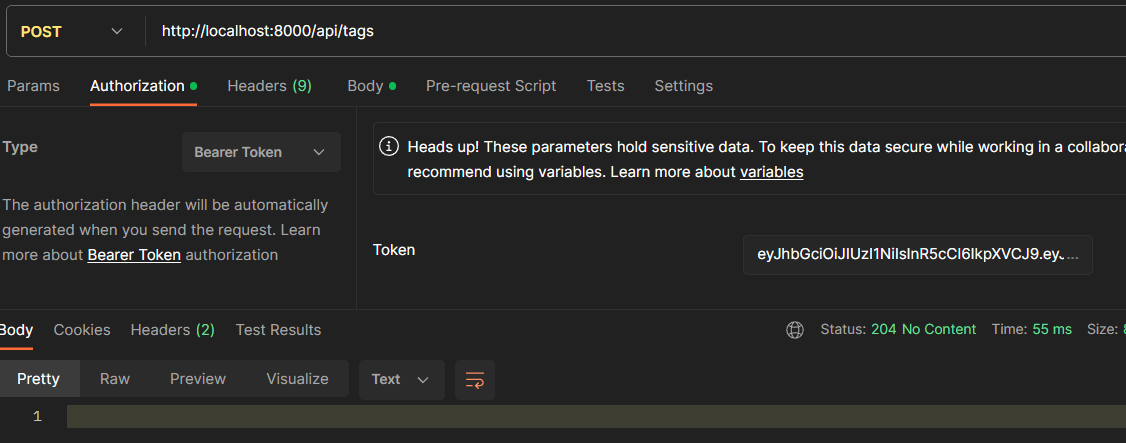
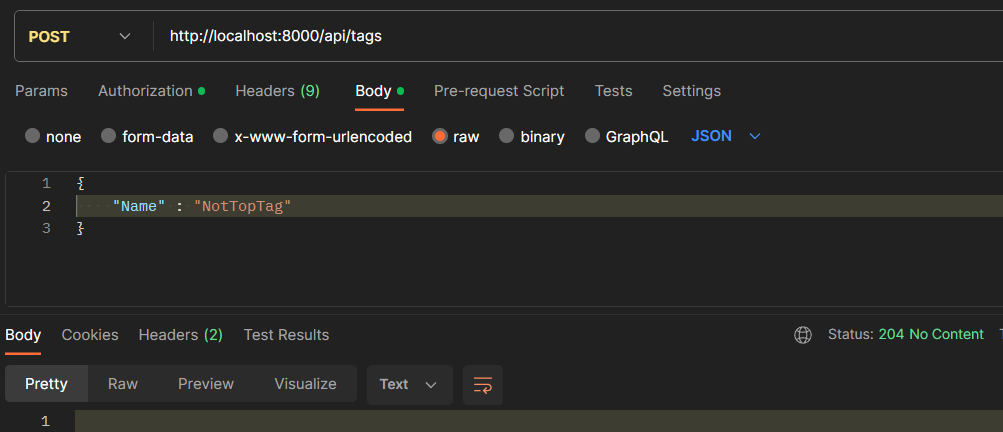
Створимо нового автора:

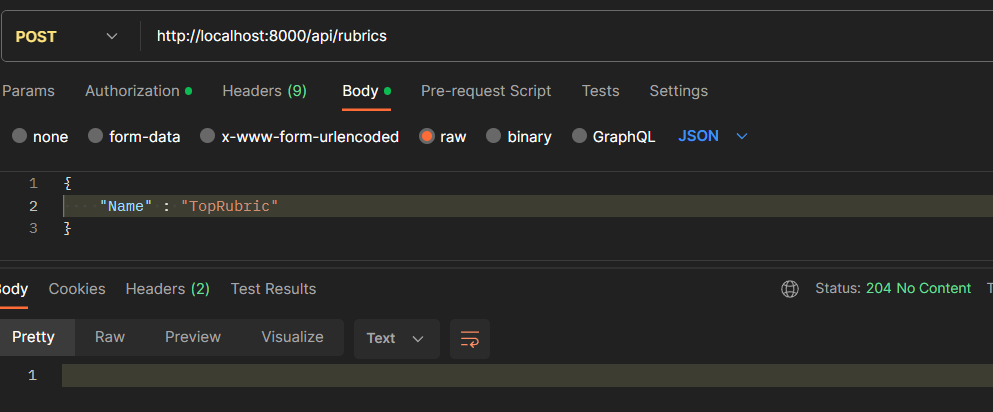
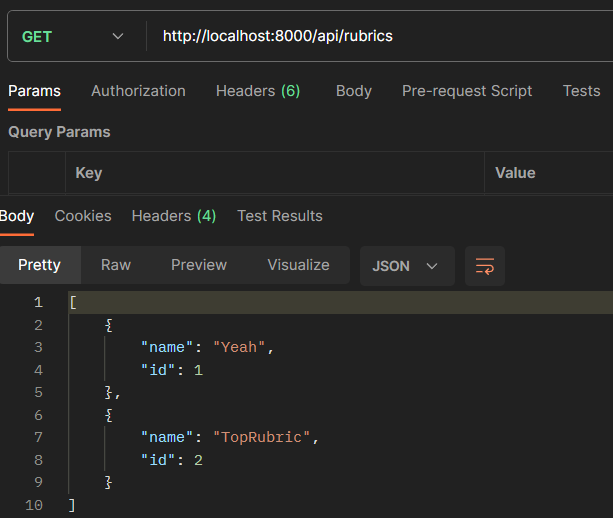
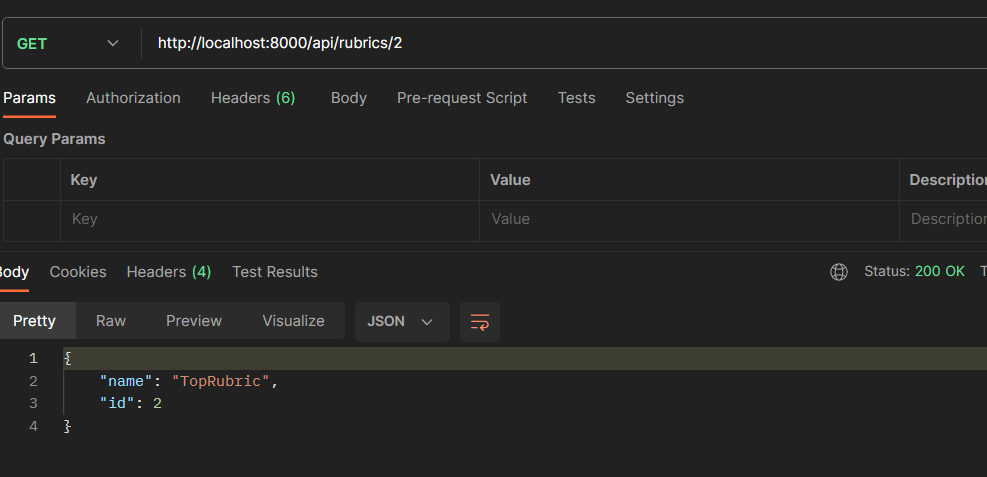
 Перевіримо чи з’явився він серед авторів:

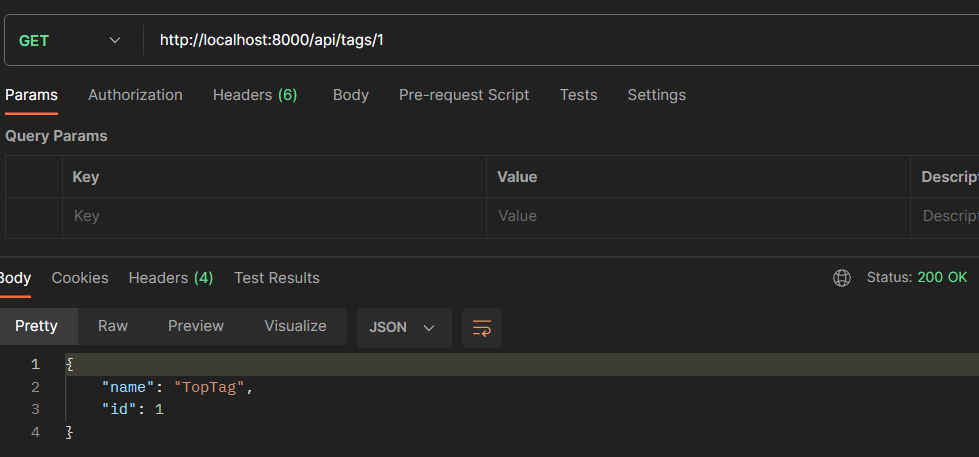
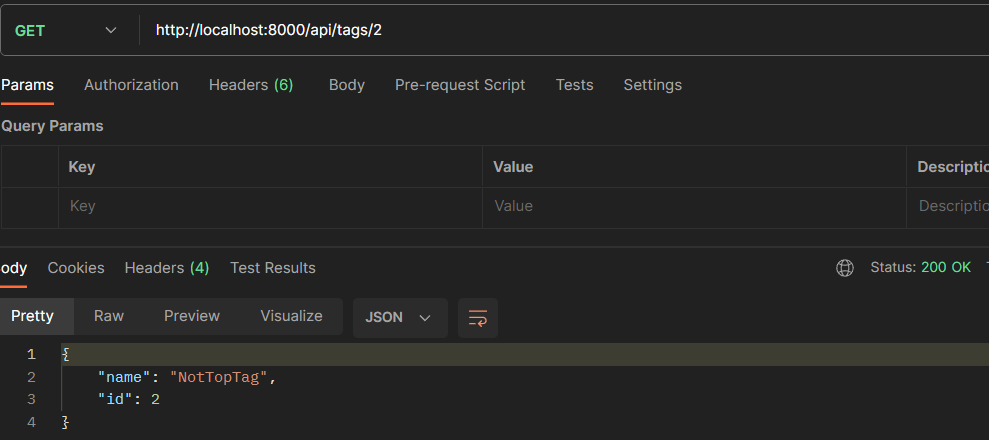
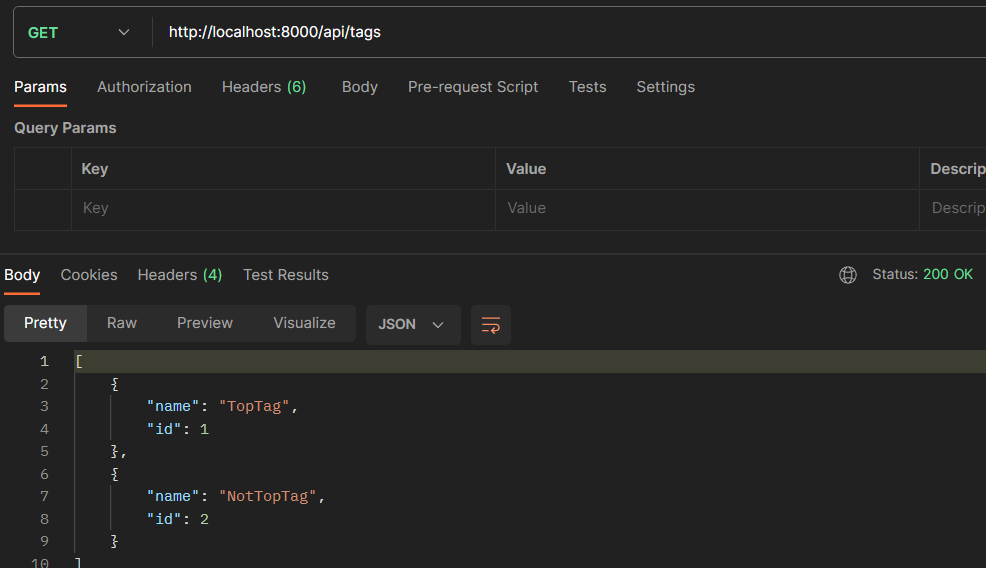
Перевіримо чи зможемо отримато його за Id:

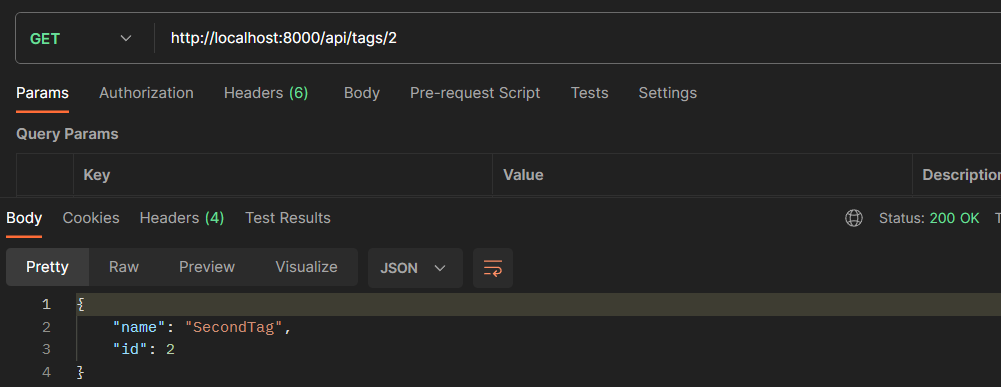
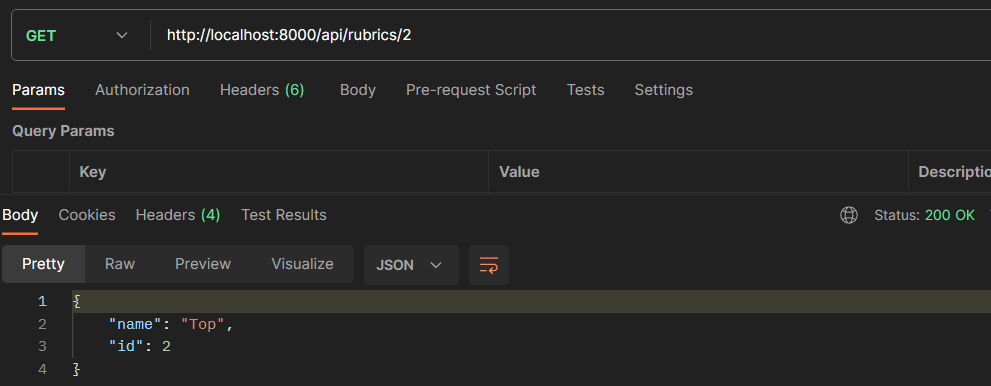
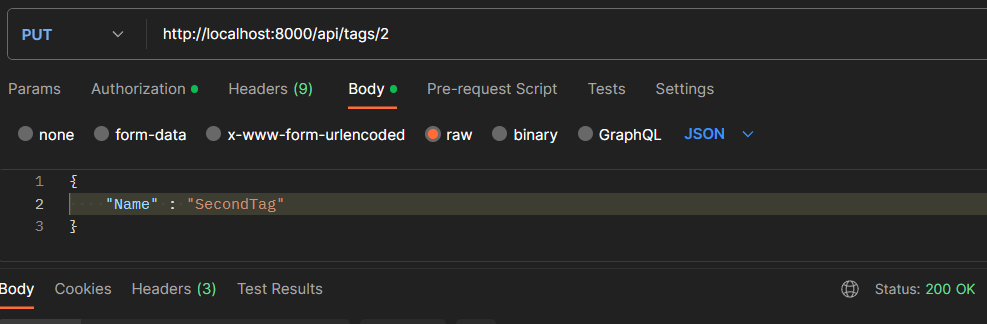
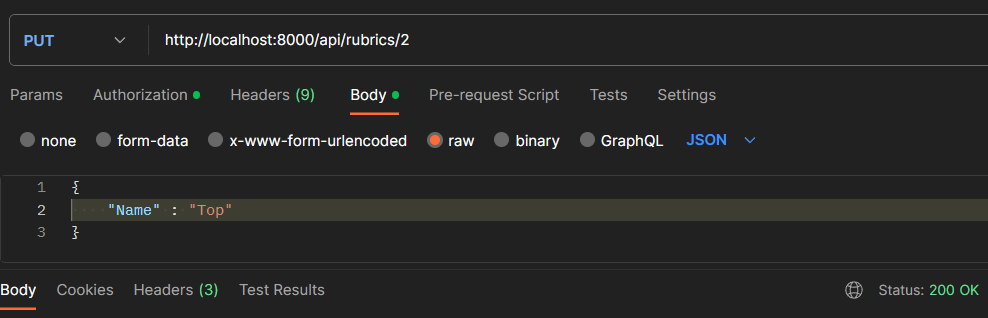
 Спробуємо залогінитись та отримати токен:

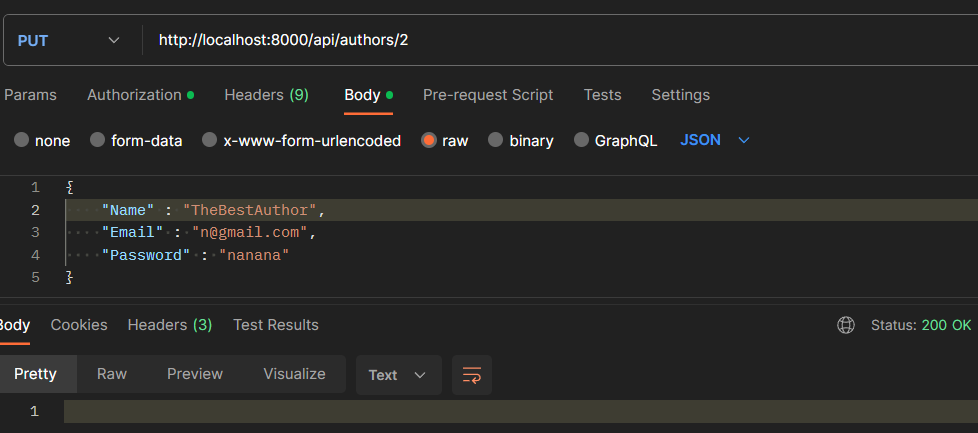
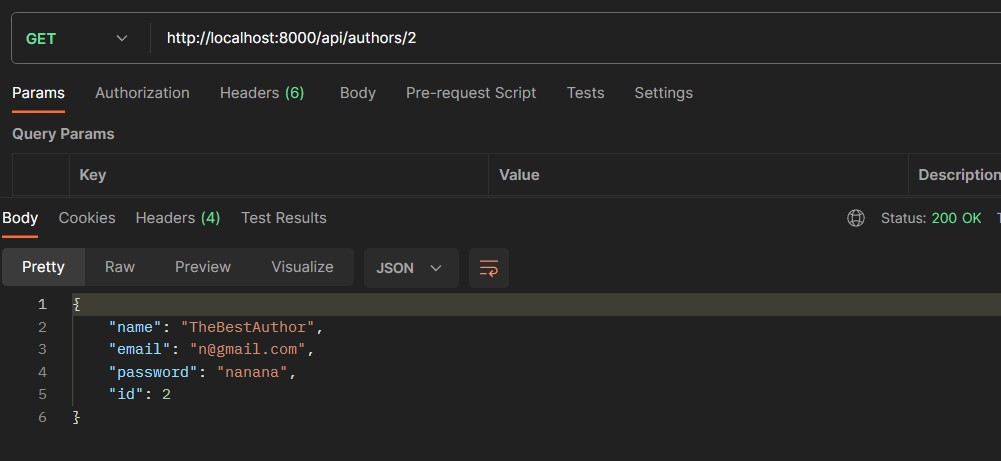
 Додамо рубрику та декілька тегів:

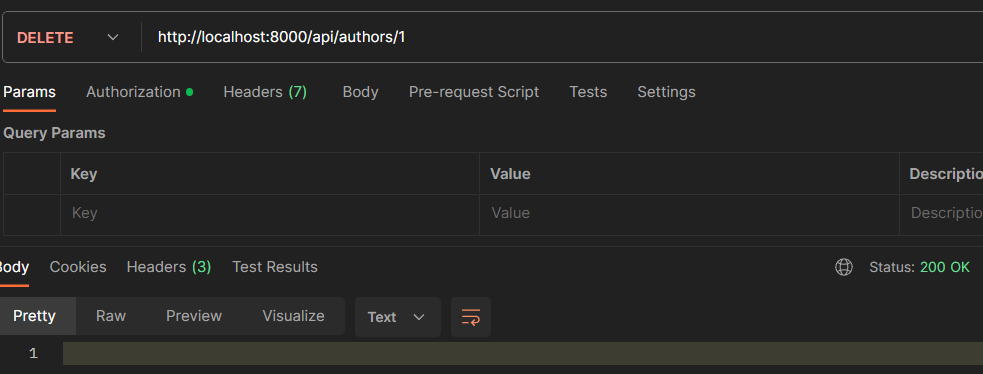
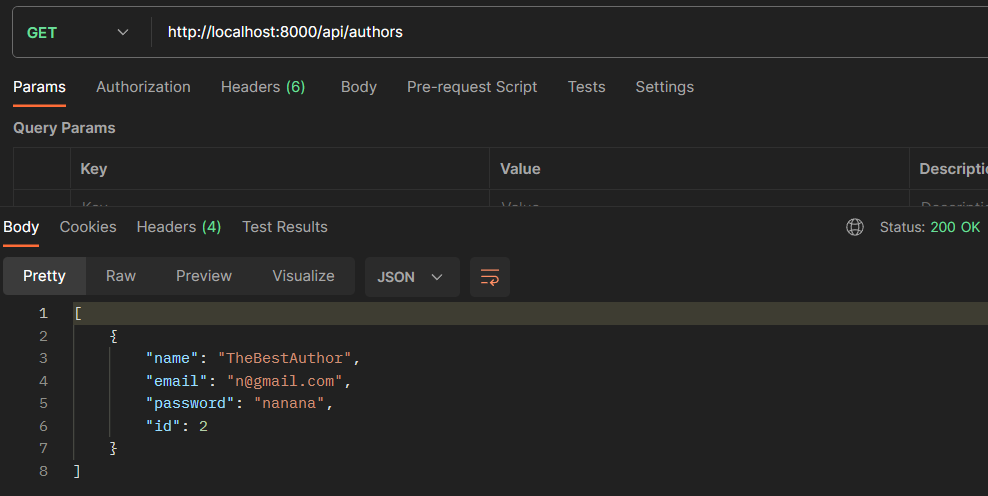
 Не дозволяє, оскільки ми не авторизовані, використаємо токен, який отримали раніше та таки створимо рубрику та декілька тегів:

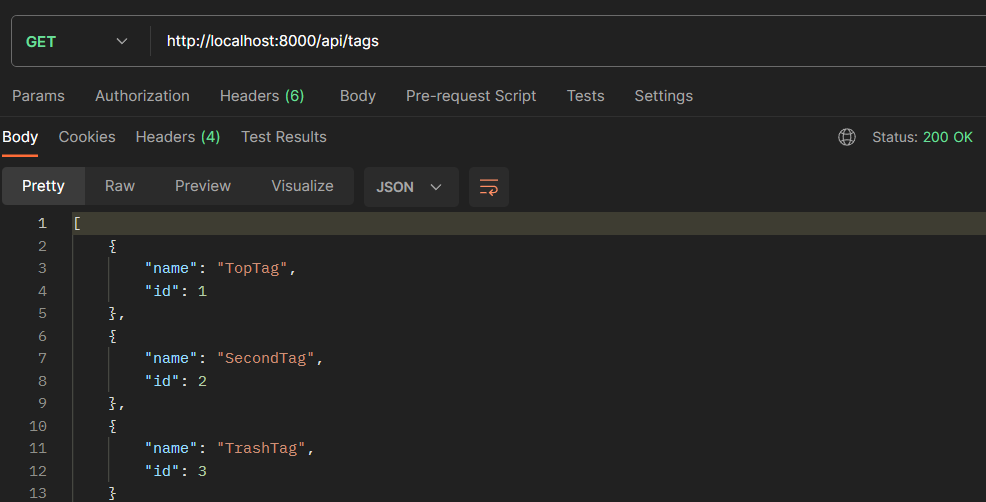
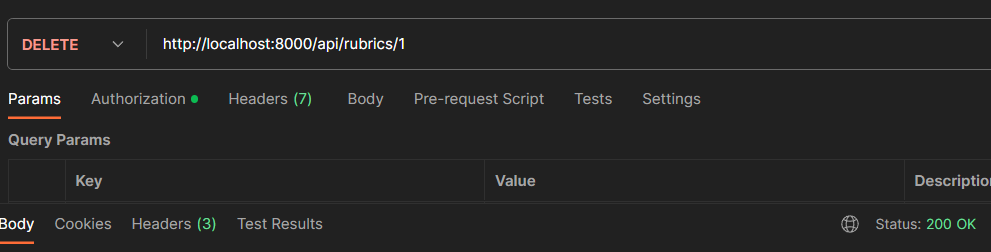
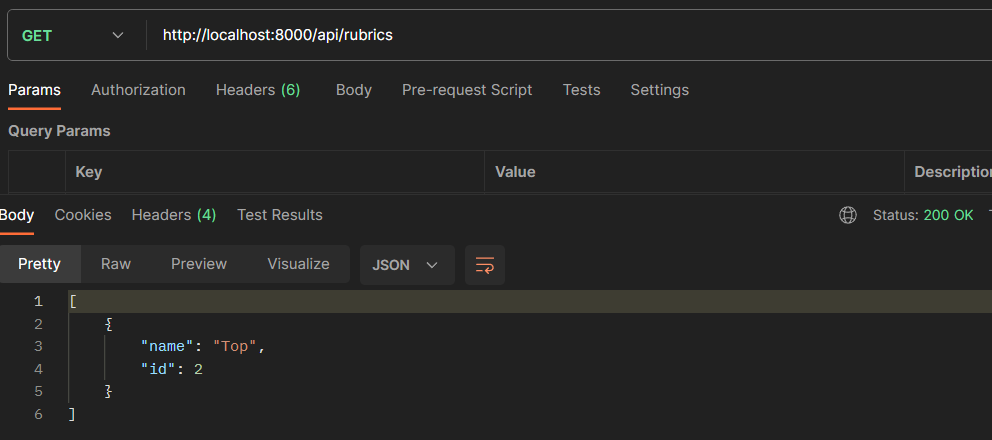
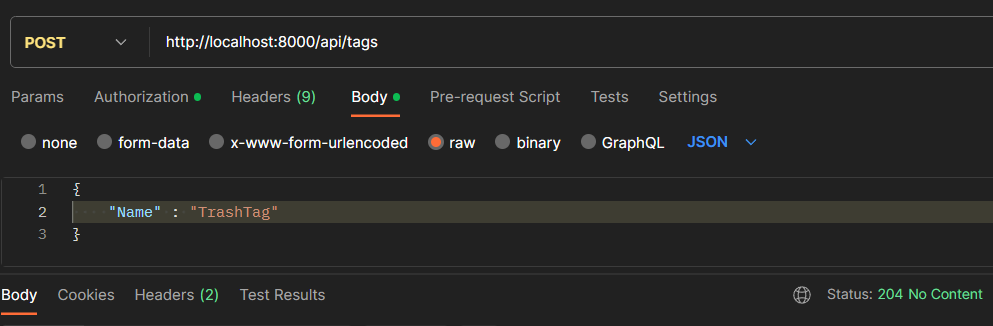
 Перевіримо чи наявні вони у загальних списках та чи можемо ми їх отримати по Id:

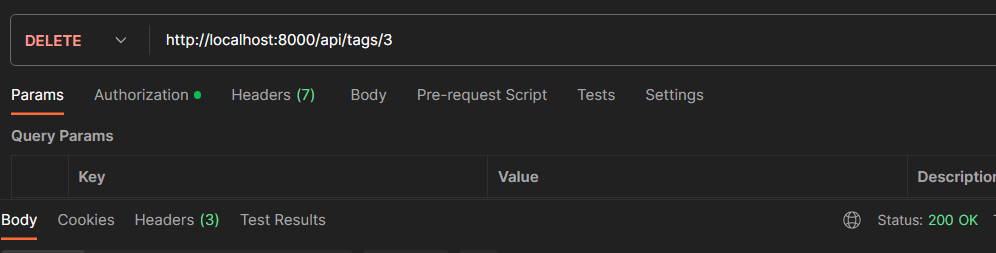
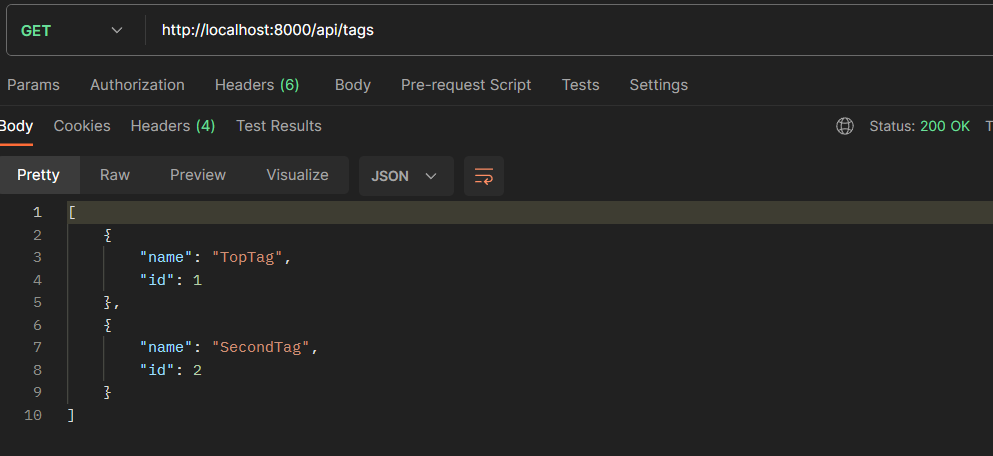
 

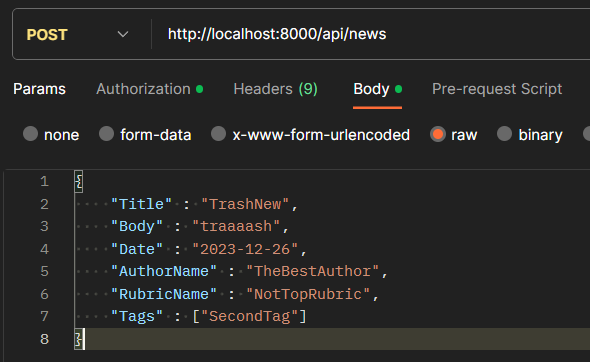
   Відредагуємо рубрику з тегом та перевіримо чи змінились вони:

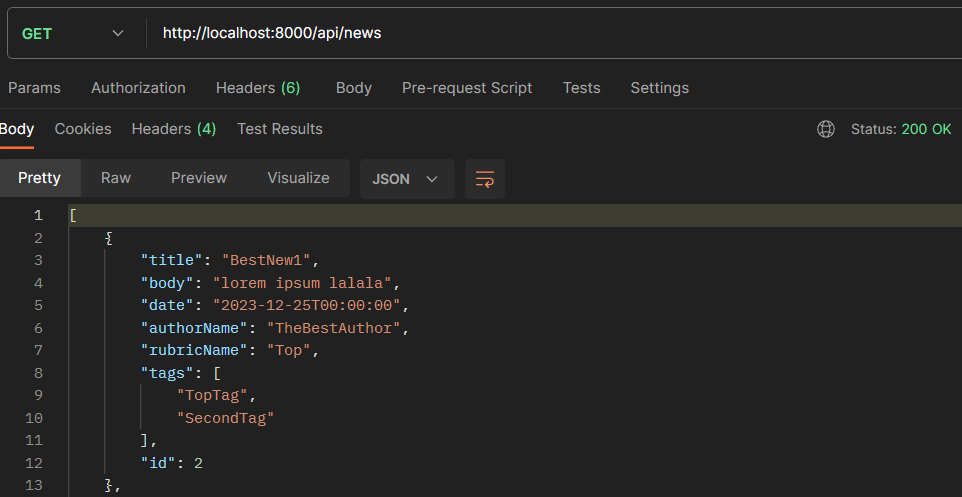
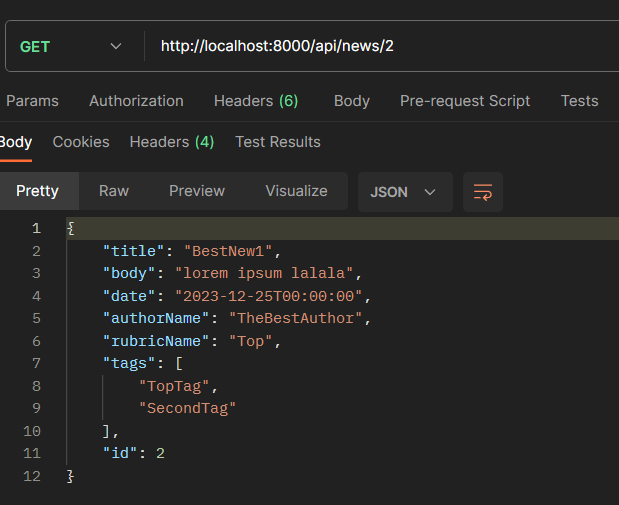
 Також відредагуємо нашого автора:

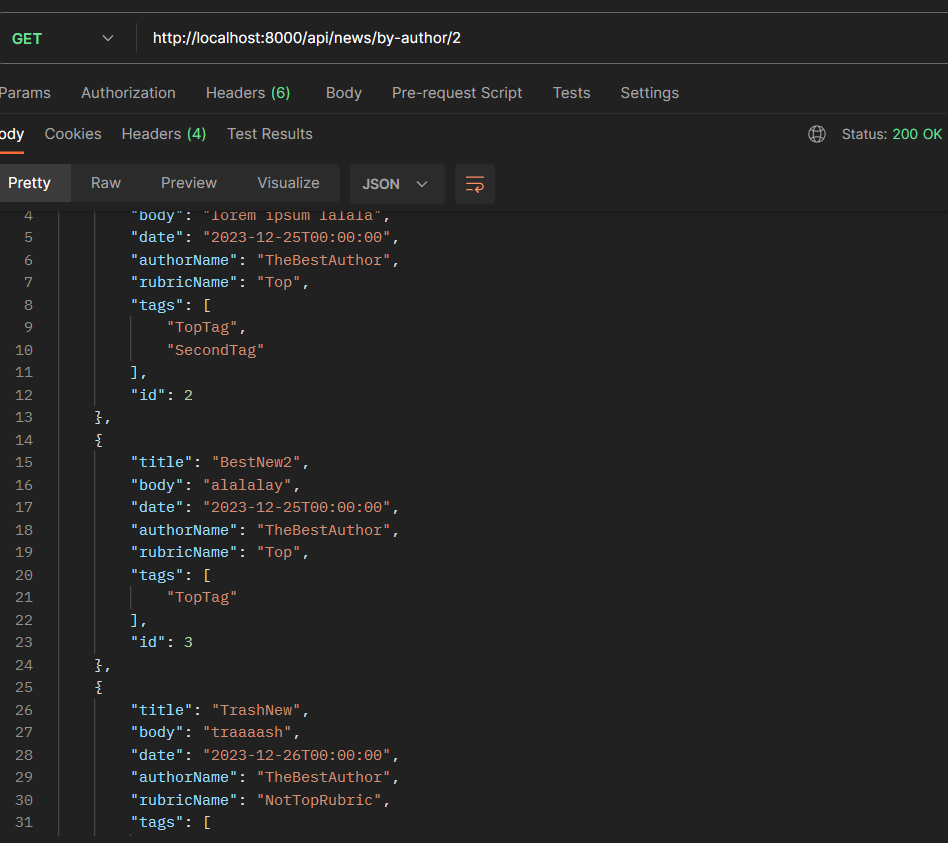
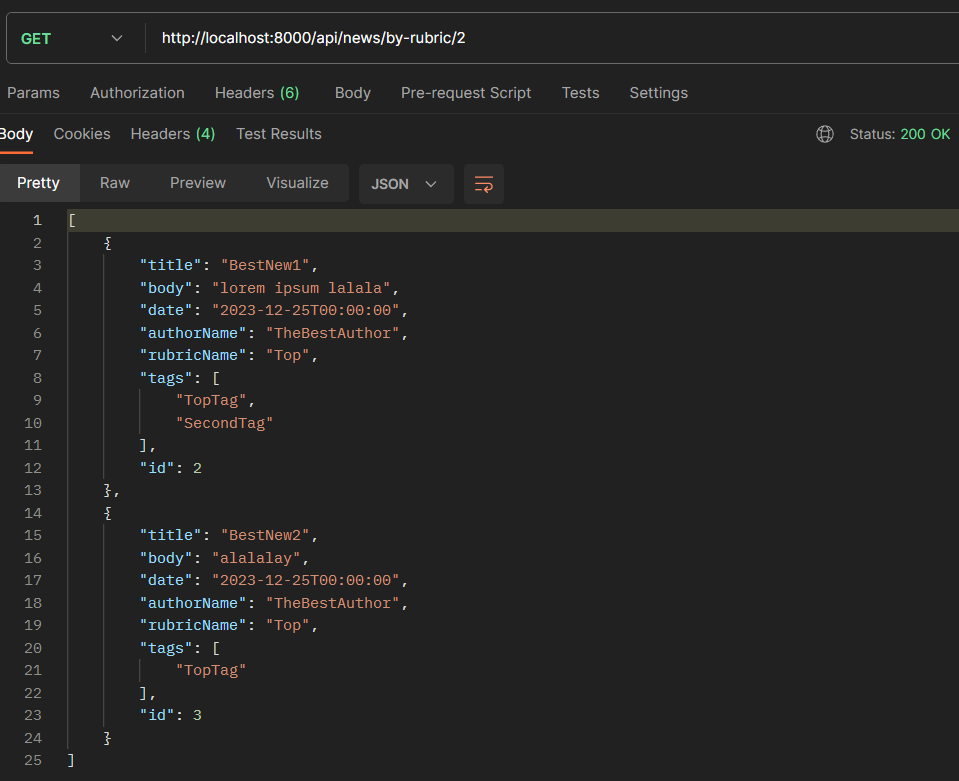
 І видалимо іншого автора:

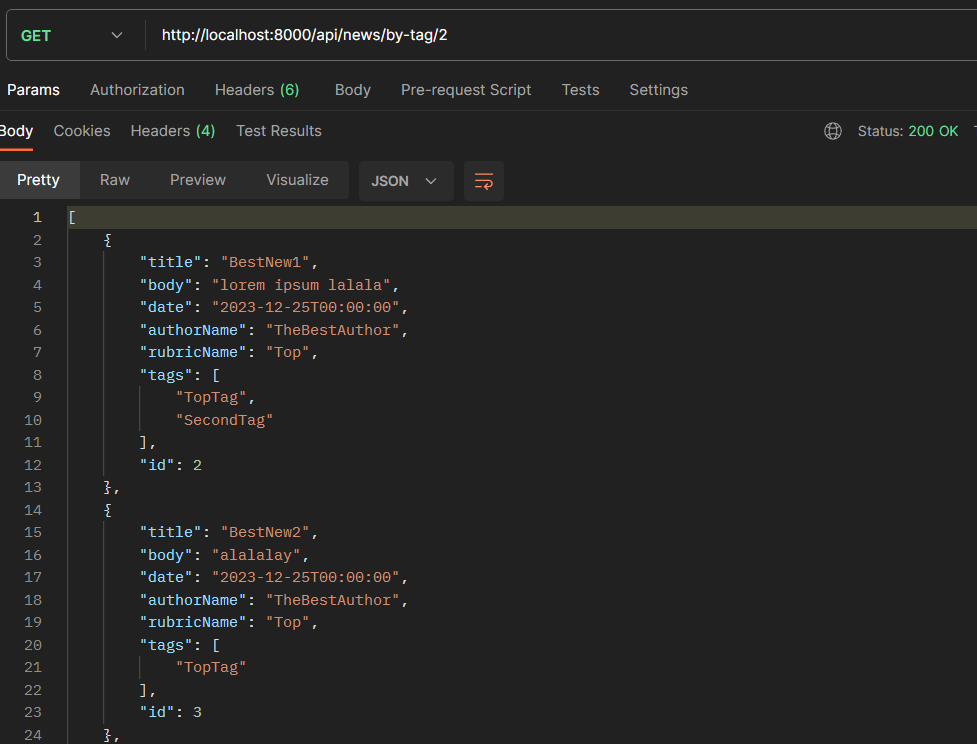
  Також видалимо од ну рубрику, т а створимо, а потім видалимо тег:

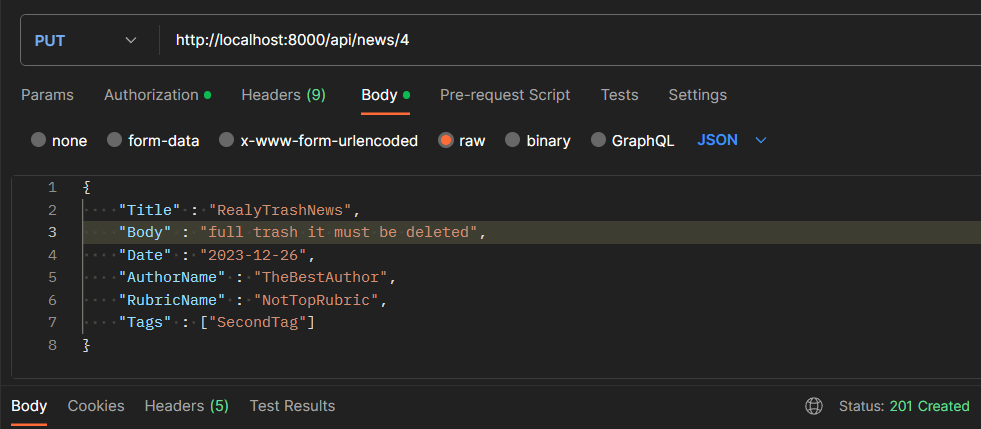
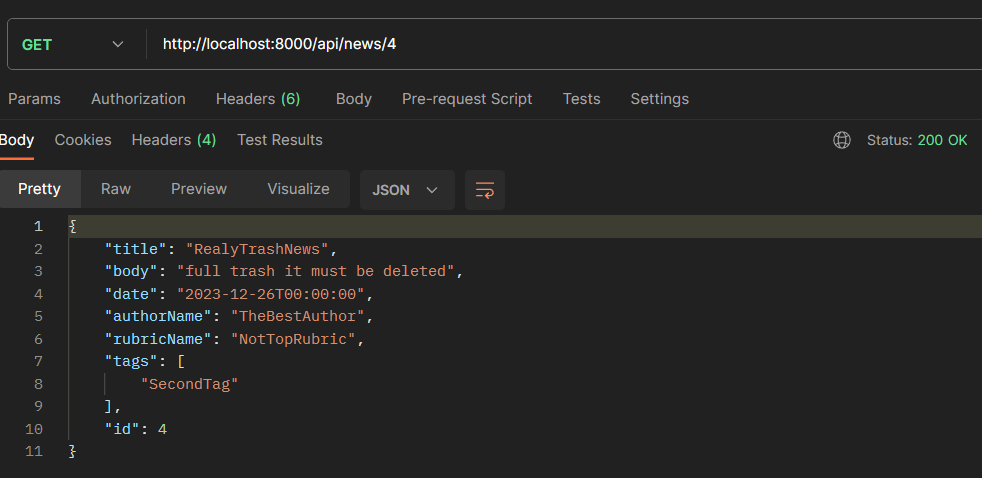
 Тепер створимо декілька новин:

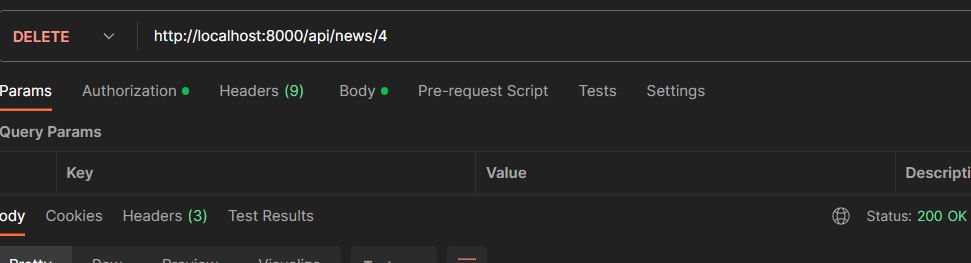
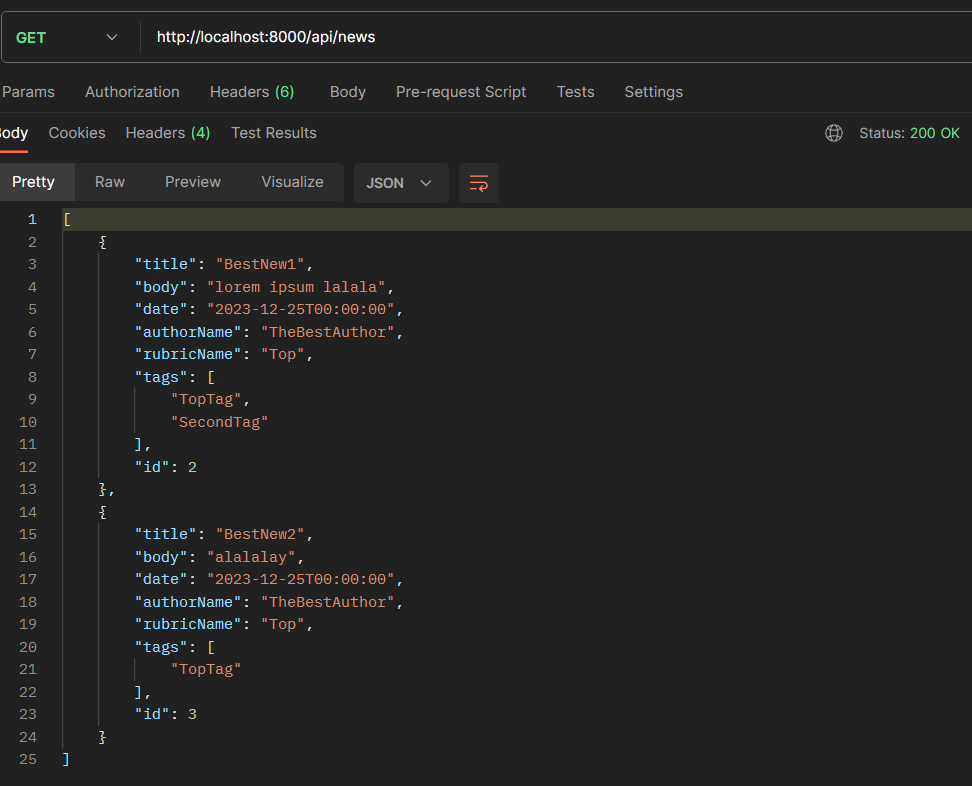


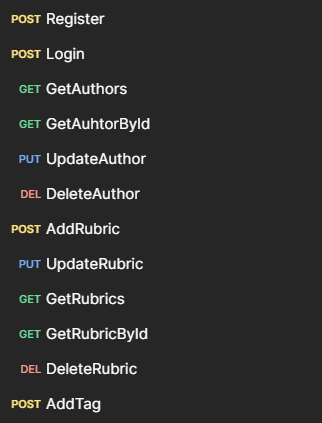
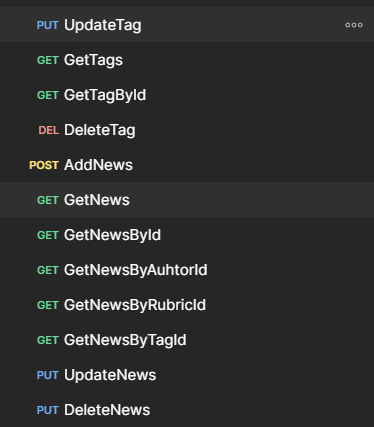


Спробуємо отримати всі новини за автором, рубрикою та тегом:



 Відредагуємо якусь новину:

 І видалимо її:

Усі запити в Postman:

**Відповіді на питання:**

1. N Layered та K Tiered архітектура:

N Layered або K Tiered архітектури - це концепції організації коду в програмах. Вони визначають, як програмна функціональність повинна бути розподілена між різними рівнями або тирами. Наприклад, в N Layered архітектурі можуть бути рівні, такі як Presentation Layer, Business Logic Layer та Data Access Layer. В K Tiered архітектурі можуть бути тири, такі як Presentation Tier, Business Tier та Data Tier. Кожен рівень або тир відповідає за конкретний аспект функціональності програми, що дозволяє підтримувати чистоту коду та легше виявляти помилки.

1. Data Access Layer (DAL) в ASP.NET Core з EF Core:

Data Access Layer в ASP.NET Core зазвичай використовує Entity Framework Core (EF Core) для забезпечення спрощеного доступу до бази даних. Основні компоненти DAL включають DbContext, який визначає сесію з базою даних, а також сутності (Entity Classes), які відображають таблиці бази даних. Використання LINQ дозволяє складати запити до бази даних, а атрибути в EF Core дозволяють визначати відносини між об'єктами та таблицями бази даних.

1. Business Logic Layer (BLL) в архітектурі N Layered:

Business Logic Layer (BLL) в N Layered архітектурі включає в себе бізнес-логіку, що визначає, як система повинна поводитися у визначених сценаріях. Основні принципи роботи включають в себе валідацію даних перед їх передачею до Data Access Layer (DAL), управління бізнес-правилами та інші аспекти логіки, що не пов'язані з безпосереднім доступом до даних чи інтерфейсом користувача.

1. REST API в ASP.NET Core:

REST API в ASP.NET Core реалізується за допомогою контролерів, які містять дії, доступні для виклику з клієнтських додатків. Контролери визначають різні HTTP-методи (GET, POST, PUT, DELETE) за допомогою атрибутів, таких як `[HttpGet]`, `[HttpPost]`. Маршрутизація визначає, які URL відповідають конкретним діям контролера.

1. Specification Pattern:

Specification Pattern є шаблоном проектування, який дозволяє визначити критерії для вибору об'єктів в системі. У контексті баз даних він може бути використаний для формулювання складних запитів. Цей паттерн інтегрується в архітектуру рішення, дозволяючи визначати умови запиту як об'єкти, що можуть бути повторно використані та легко комбіновані для складних вибірок.

1. NSubstitute для модульного тестування в .NET:

NSubstitute - це бібліотека для створення фейкових об'єктів у тестах. Вона дозволяє створювати замінники реальних об'єктів, спрощуючи тестування та дозволяючи контролювати поведінку об'єктів.

1. AutoBogus для генерації тестових даних:

AutoBogus - це бібліотека, яка автоматично генерує фейкові дані для тестування. Вона дозволяє швидко створювати тестові об'єкти з мінімальними зусиллями, що сприяє ефективному тестуванню.

1. Microsoft.DependencyInjection в ASP.NET Core:

Microsoft.DependencyInjection - це вбудований контейнер внедрення залежностей в ASP.NET Core. Він спрощує роботу з залежностями, дозволяючи зареєструвати служби та їх залежності в сервісному контейнері. Це полегшує ін'єкцію залежностей в класи та додає гнучкість до конфігурації служб.

1. Використання Specification Pattern для гнучких запитів до бази даних:

Specification Pattern у контексті баз даних дозволяє створювати гнучкі та повторно використовувані умови запитів. Об'єкти-специфікації описують критерії вибору записів з бази даних, і ці об'єкти можуть бути комбіновані для складних запитів.

1. Dependency Injection в ASP.NET Core:

Dependency Injection (DI) - це паттерн, який дозволяє внедрювати залежності в класи. В ASP.NET Core використовується Microsoft.Extensions.DependencyInjection для реалізації DI. Це спрощує управління залежностями та робить код більш тестовим та модульним.

1. Фейкові об'єкти з NSubstitute для модульного тестування в ASP.NET Core:

При модульному тестуванні в ASP.NET Core, NSubstitute може бути використаний для створення фейкових об'єктів, щоб замінити реальні залежності. Це дозволяє тестувати окремі компоненти системи незалежно від решти.

1. Експорт API до Postman:

Експорт API до Postman дозволяє зберігати та документувати API. Postman генерує колекції запитів, які можна легко імпортувати та використовувати для тестування різних ендпоінтів. Це полегшує співпрацю між розробниками та тестувальниками та дозволяє ефективно тестувати та документувати API.

1. Міграції в EF Core:

Міграції в Entity Framework Core (EF Core) дозволяють автоматизовано оновлювати схему бази даних при зміні моделі даних. Це спрощує процес управління базою даних та зберігання структури сумісною з оновленнями програми.

1. Аутентифікація та авторизація в ASP.NET Core REST API:

Аутентифікація та авторизація в ASP.NET Core REST API використовують системи ідентифікації, такі як JWT. Аутентифікація підтверджує ідентичність користувача, а авторизація визначає його доступ до ресурсів на основі визначених ролей чи політик.

1. Оптимізація масштабованості REST API в ASP.NET Core:

Для оптимізації продуктивності та масштабованості REST API в ASP.NET Core можна використовувати різні техніки. Це включає в себе використання кешування для зменшення завантаження сервера, використання CDN для швидкої доставки ресурсів, горизонтальне масштабування за допомогою балансувальників навантаження та оптимізацію запитів та відповідей за допомогою асинхронності та інших технік оптимізації коду.