**Лабораторна робота №3**

**Мета:** Ознайомитися з принципами клієнт-серверної архітектури та зрозуміти її роль у сучасних програмних системах. Навчитися створювати RESTful API для взаємодії між клієнтом і сервером. Закріпити практичні навички роботи з HTTP-запитами та відповідями. Розвинути вміння проєктувати та реалізовувати ендпойнти для типових CRUD-операцій.

**Завдання**

Завдання 1 — Створити свій Fake Online REST Server і виконати базові HTTP-запити

1. Підготовка репозиторію
   * Створіть на GitHub публічний репозиторій, напр.: your-username/fake-api.
   * Додайте файл db.json у корінь репозиторію з власними сутностями. Приклад:

{  
 "users": [  
 { "id": 1, "name": "Alice", "email": "alice@example.com", "role": "admin" },  
 { "id": 2, "name": "Bob", "email": "bob@example.com", "role": "user" }  
 ],  
 "posts": [  
 { "id": 1, "title": "Hello REST", "userId": 1, "tags": ["intro","rest"] },  
 { "id": 2, "title": "JSON Tips", "userId": 2, "tags": ["json"] }  
 ]  
}

* + Переконайтесь, що id унікальні та цілі числа.

1. Запуск Fake Online REST
   * Базова URL-адреса після публікації:

https://my-json-server.typicode.com/<your-username>/<repo-name>

your-username - GitHub nickname

repo-name - repository name

* + Приклади ресурсів:
    - GET /users
    - GET /users/1
    - GET /posts?userId=1
    - GET /posts?q=aaaa (повнотекстовий пошук)

1. Перевірка запитів у Postman (або аналогах)
   * Створіть колекцію LW3\_Fake\_REST.
   * Додайте та виконайте мінімальний набір запитів:

GET /users — отримати список користувачів.

GET /users/1 — отримати одного користувача.

GET /posts?\_sort=title&\_order=asc&\_limit=1 — сортування/ліміт.

GET /posts?userId=2 — фільтрація за полем.

GET /posts?q=REST — пошук (працює по кількох полях).

У my-json-server записи не зберігаються після POST/PUT/PATCH/DELETE (це read-only), але ви можете відпрацювати формування тіла запитів у Postman (Body → raw → JSON) і перевірити, що сервер відповідає коректними статусами або повідомленнями (може повертати 404/405). Для реальної CRUD-практики — локально підняти json-server або використати інший мок-сервіс. Тут важливо навчитися формувати правильні запити.

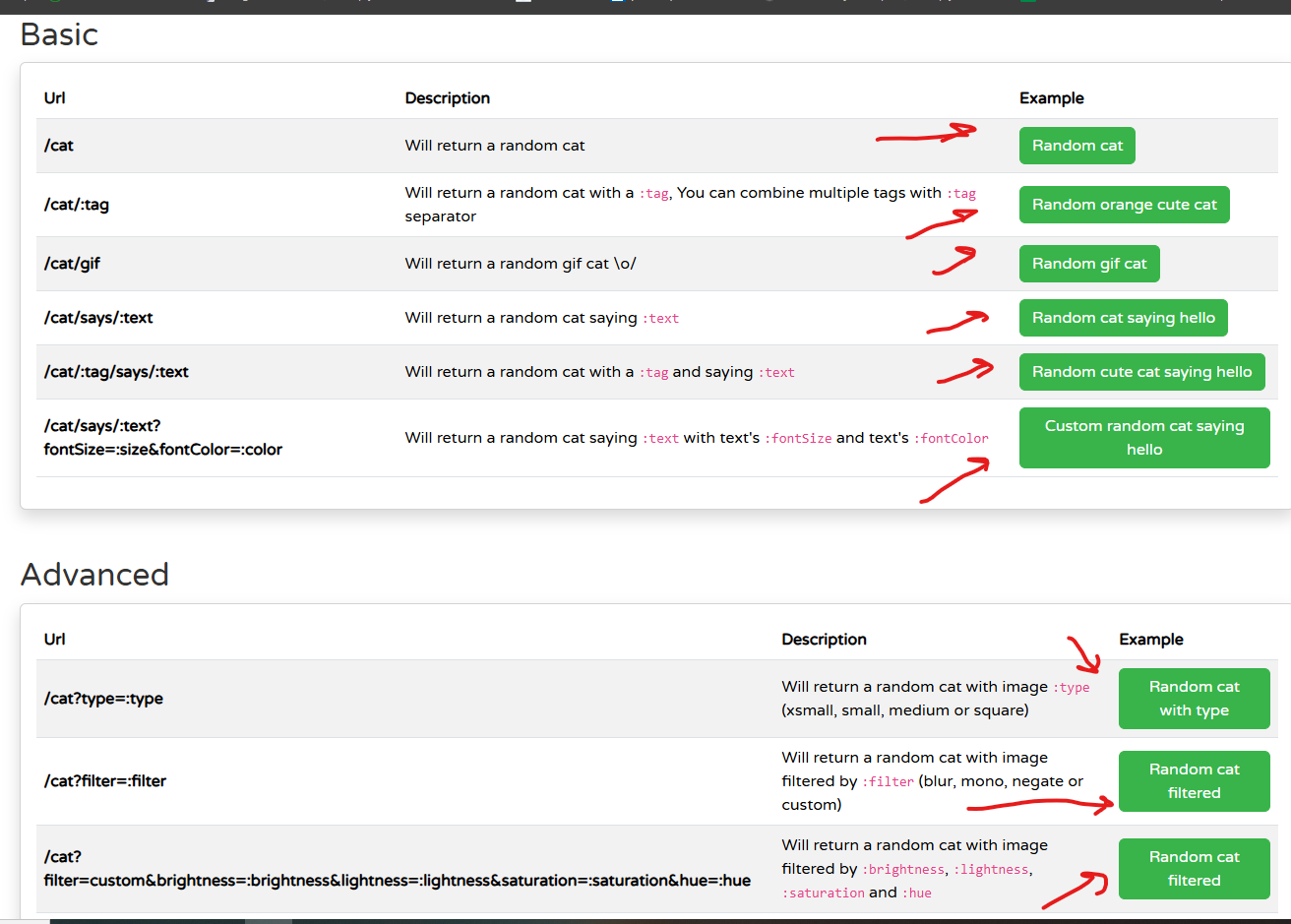
* + Зафіксуйте у звіті Status Code, Headers, Body. Зверніть увагу на Content-Type: application/json; charset=utf-8.

1. Приклади через curl (додатково до Postman)

# 1) Усі користувачі  
curl -i https://my-json-server.typicode.com/<user>/<repo>/users  
  
# 2) Один користувач  
curl -i https://my-json-server.typicode.com/<user>/<repo>/users/1  
  
# 3) Фільтр  
curl -i "https://my-json-server.typicode.com/<user>/<repo>/posts?userId=2"  
  
# 4) Сортування + ліміт  
curl -i "https://my-json-server.typicode.com/<user>/<repo>/posts?\_sort=title&\_order=asc&\_limit=1"  
  
# 5) Пошук  
curl -i "https://my-json-server.typicode.com/<user>/<repo>/posts?q=json"

Завдання 2 — Робота з API сервісу CATAAS (Cat as a Service) та SwaggerAPI

1. Перейдіть за посиланням <https://cataas.com/> та спробуйте виконати різні запити (натисніть відповідні кнопки)



1. Створіть колекцію в Postman

Назвіть її: LW3\_CATAAS\_API.

1. Додайте та виконайте запити:

Базова адреса API: [https://cataas.com](https://cataas.com/)

Основні ресурси:

* + GET /cat — отримати випадкове фото кота.
  + GET /cat/says/{text} — кіт із підписом.
  + GET /api/cats — список котів (JSON).
  + GET /cat/gif — GIF-анімація кота.

Зверніть увагу на параметри ?filter=mono, ?type=gif, ?json=true, ?width=300&height=200 тощо.

Виконайте:

* + GET /cat — отримати випадкове зображення кота.
  + GET /cat/says/HelloWorld — кіт із підписом вашого mood (картинка з текстом яка описує тебе).
  + GET /cat/says/API?filter=mono&width=400&height=200 — кіт із чорно-білим підписом. Спробуйте використати різні параметри запиту
  + GET /cat/gif— отримати випадкове gif-кота.
  + GET /api/cats?limit=5 — отримати JSON-список перших 5 котів.
  + GET /cat/gif?json=true — отримати метадані кото-GIF у форматі JSON.

Для кожного запиту зафіксуйте у звіті:

* + Status Code (200, 404, інші).
  + Response Headers (зверніть увагу на Content-Type).
  + Response Body (зображення або JSON).

1. Ознайомлення з документацією. Виконання запитів через Swagger UI

Ознайомитися з матеріалом по SwaggerAPI:

[Тестування API з Swagger: як перевірити функціональність API](https://blog.ithillel.ua/articles/api-testing-with-swagger)

У Swagger знайдіть розділ Endpoints. Виконайте мінімум 3 запити напряму через вебінтерфейс Swagger (наприклад: GET /cat, GET /cat/says/{text}, GET /api/cats).

Для кожного запиту зафіксуйте у звіті:

* + Request URL
  + Status Code
  + Response Headers
  + Response Body (JSON чи зображення).

Завдання 3 — Console-клієнт до публічного REST API

1. Створіть Console App (C# або будь-яку іншу мову програмування) і за допомогою бібліотеки HttpClient, надішліть GET-запит до сервісу відкритого REST API ([Додаток А](https://docs-mai.vercel.app/docs/labs/lw3#A))
2. Отримайте відповідь від сервера у форматі JSON
3. Створіть у проєкті DTO-клас із відповідними властивостями
4. Розпарсьте JSON-відповідь у C#-об’єкт за допомогою System.Text.Json.
5. Виведіть у консоль отримані дані у читабельному форматі.

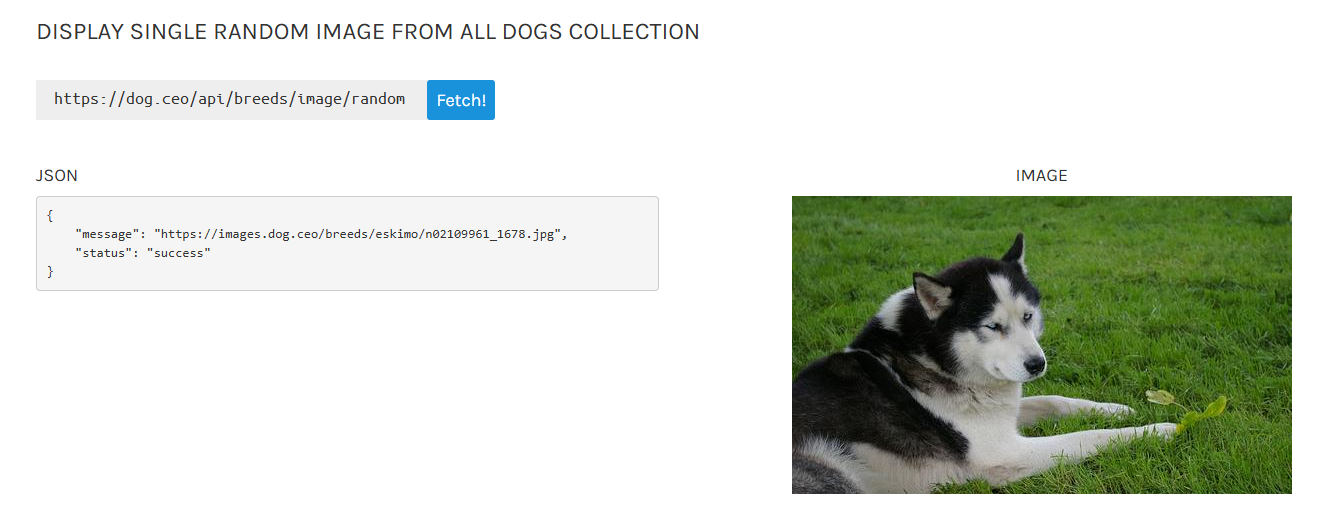
Приклад виконання у [Додатк B](https://docs-mai.vercel.app/docs/labs/lw3#B).

Завдання 4 — Інтеграція з Dog.CEO API у WinForms

Документація <https://dog.ceo/dog-api/documentation/random>

4.1. Витягнути рандомне зображення собаки і відобразити у WinForms

1. Створіть WinForms-проєкт Template: *Windows Forms App* (.NET 6/7/8). Наближений макет форми:



1. Використайте ендпойнт GET https://dog.ceo/api/breeds/image/random і отримайте рандомне зображення собаки і відбражіть його в PictureBox, при натискані кнопки отрмати нове зображення

4.2. Створити галерею світлин собак

1. За допомогою GET https://dog.ceo/api/breeds/image/random/{n} ендпойнта завантажити N зображень собак
2. Додайте перемекач кнопками < та > щоб змінювати світлину

Завдання 5 — WinForms-клієнт до публічного REST API

Створити простий клієнт на C# WinForms, який зчитує дані з відкритого REST API, відображає список ресурсів і деталі одного ресурсу, підтримує мінімальні фільтри/пошук, та коректно обробляє помилки (якщо АPI дозволяє). API можна обрати зі списку публічних API або власне.

Заборонено копіювати чужі проєкти. Якщо буде виявлено плагіат — “гайки” (нуль за роботу й інший варіант).

Що необхідно зробити:

1. Отримати колекцію (масив) JSON-об’єктів із відкритого API ([Додаток А](https://docs-mai.vercel.app/docs/labs/lw3#A)).
2. Відобразити в DataGridView (або ListView) з можливістю прокрутки.
3. Якщо можливості АРІ дозволяють (більшіть так) виконати мінімальний пошук/фільтр (по одному полю або query-параметру API).
4. При виборі рядка зі списку — виконати другий запит (наприклад, GET /resource/{id} або аналог) і показати деталі у правій панелі (Labels/TextBoxes/PictureBox).
5. Кнопка Refresh для повторного завантаження списку.
6. Відображати користувачу читабельні повідомлення при: відсутності інтернету, 4xx/5xx, таймауті, невалідному JSON.
7. Блокувати кнопки під час запиту, показувати індикатор стану (StatusStrip/Label).

Нефункціональні вимоги

* .NET 6/7/8, WinForms, C#, за бажанням інша технологія, інші технології до написання UI. It’s up to you!
* Структура проєкту:

Forms/, Services/ApiClient.cs, Models/Dto.cs, Utils/.

* Код-стайл: читабельні назви, мінімальні коментарі, винесення констант (базова адреса API тощо).

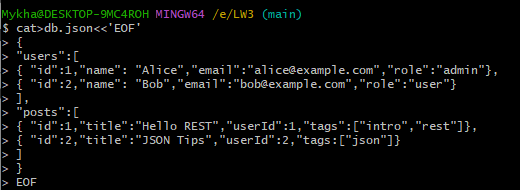
**Хід роботи**

**Завдання №1**

1. Створимо репозиторій та назвемо LW3\_MiA\_Yevchuk

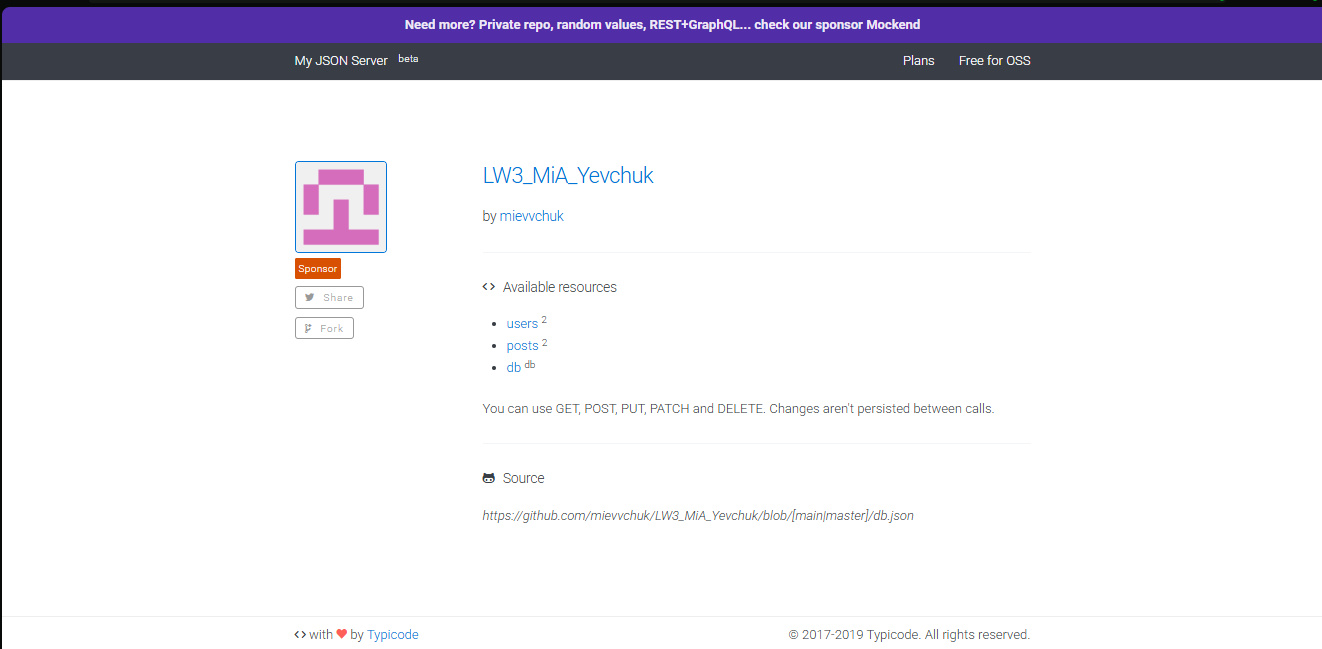


1. Додамо туди файл **db.json** у корінь репозиторію. Додаємо через **Git Bash**



1. Запускаємо **Fake Online REST**

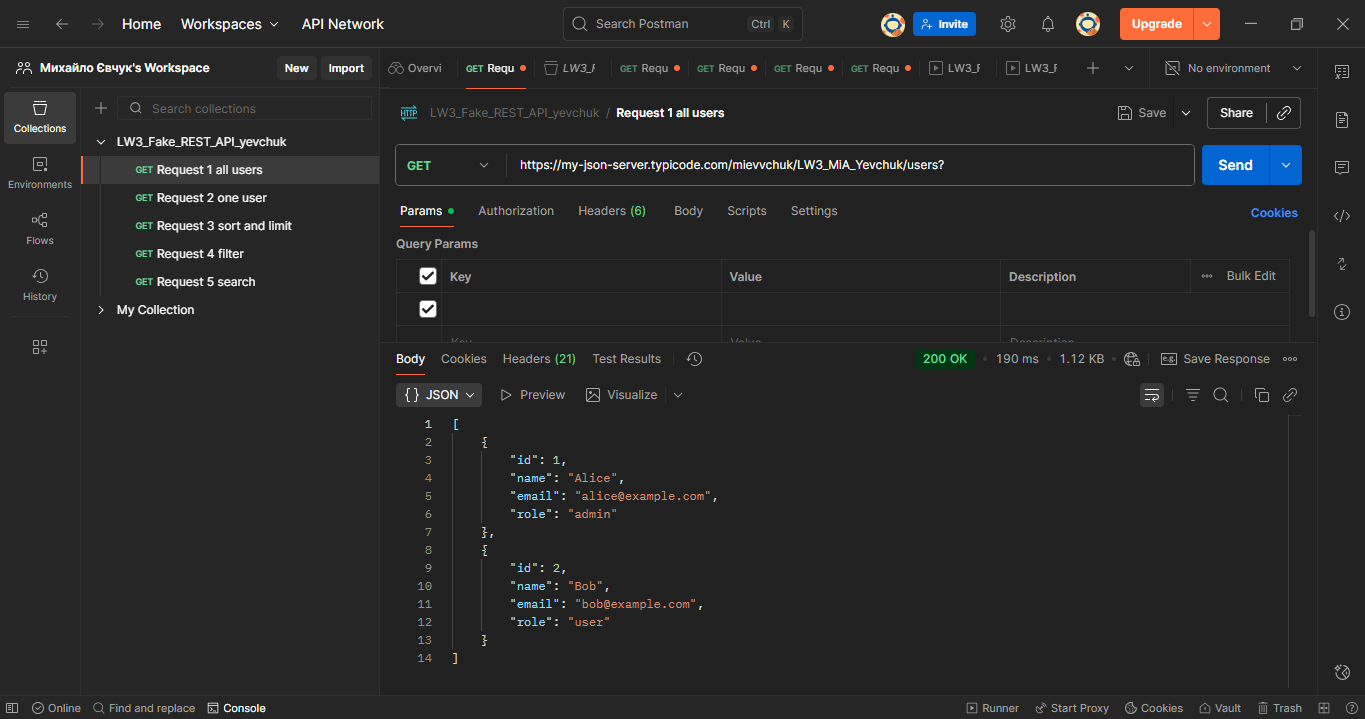
Базова URL-адреса після публікації: <https://my-json-server.typicode.com/mievvchuk/LW3_MiA_Yevchuk>

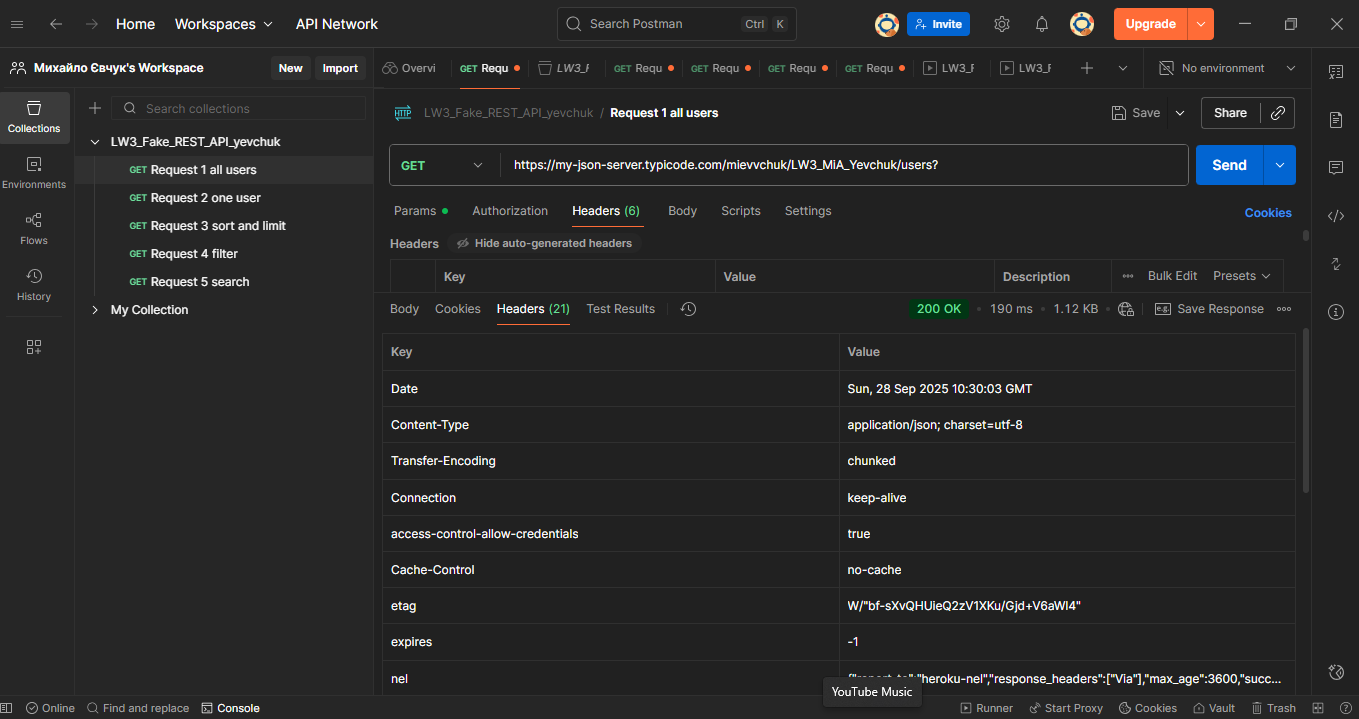


1. Перевірка запитів через PostMan

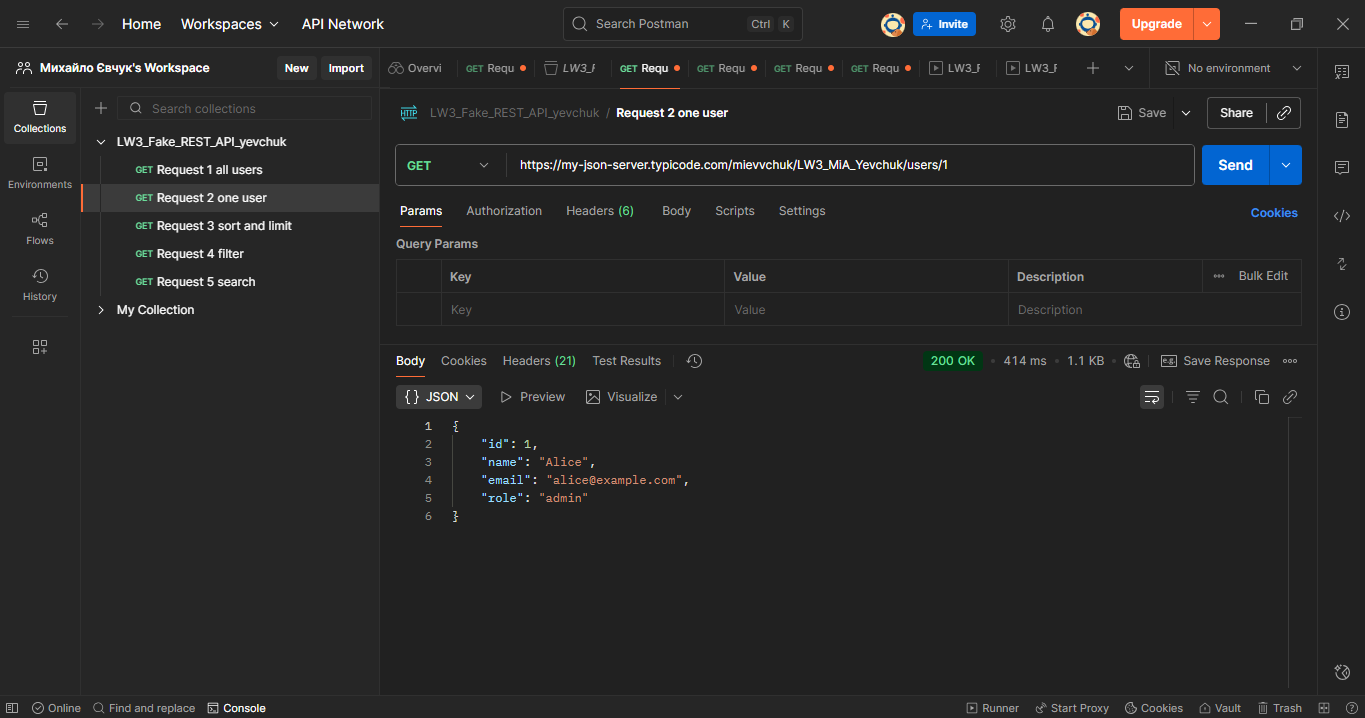
Створюємо колекцію і називаємо її **“LW3\_Fake\_REST\_API\_yevchuk”**

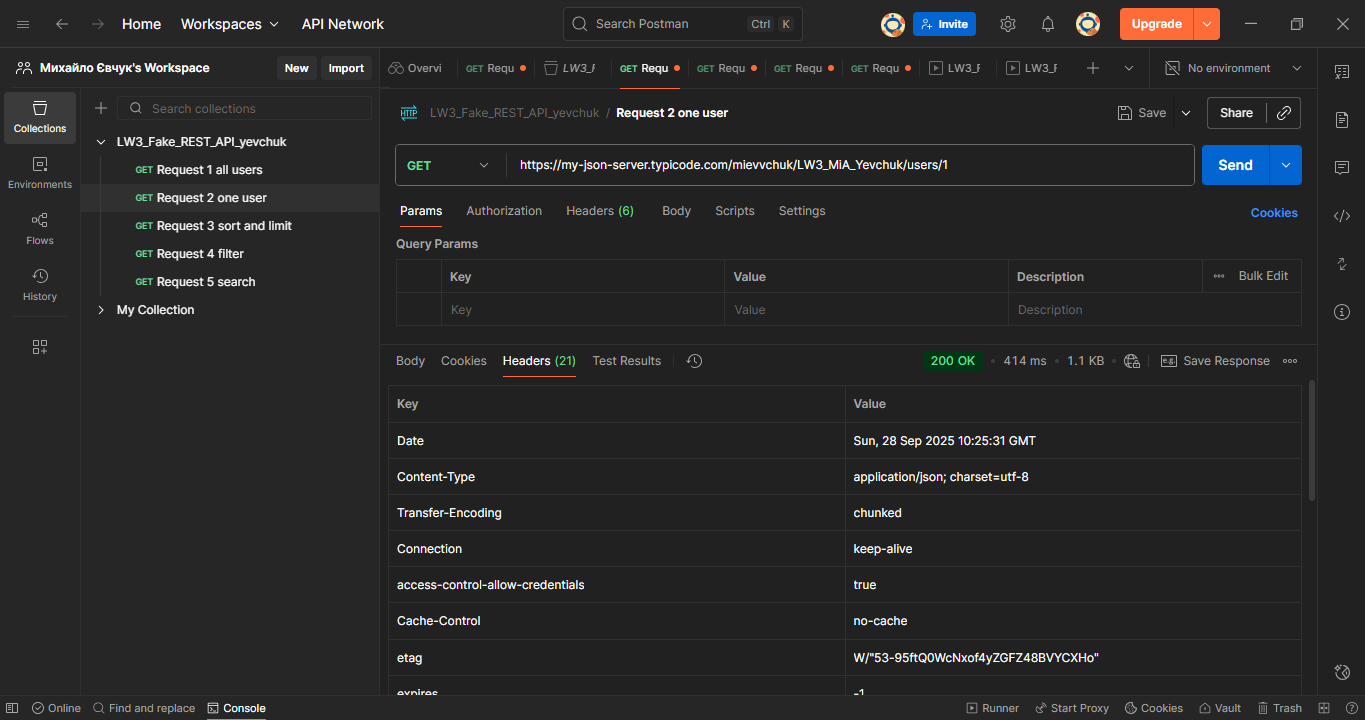
* 1. Виконаємо запит на усіх користувачів



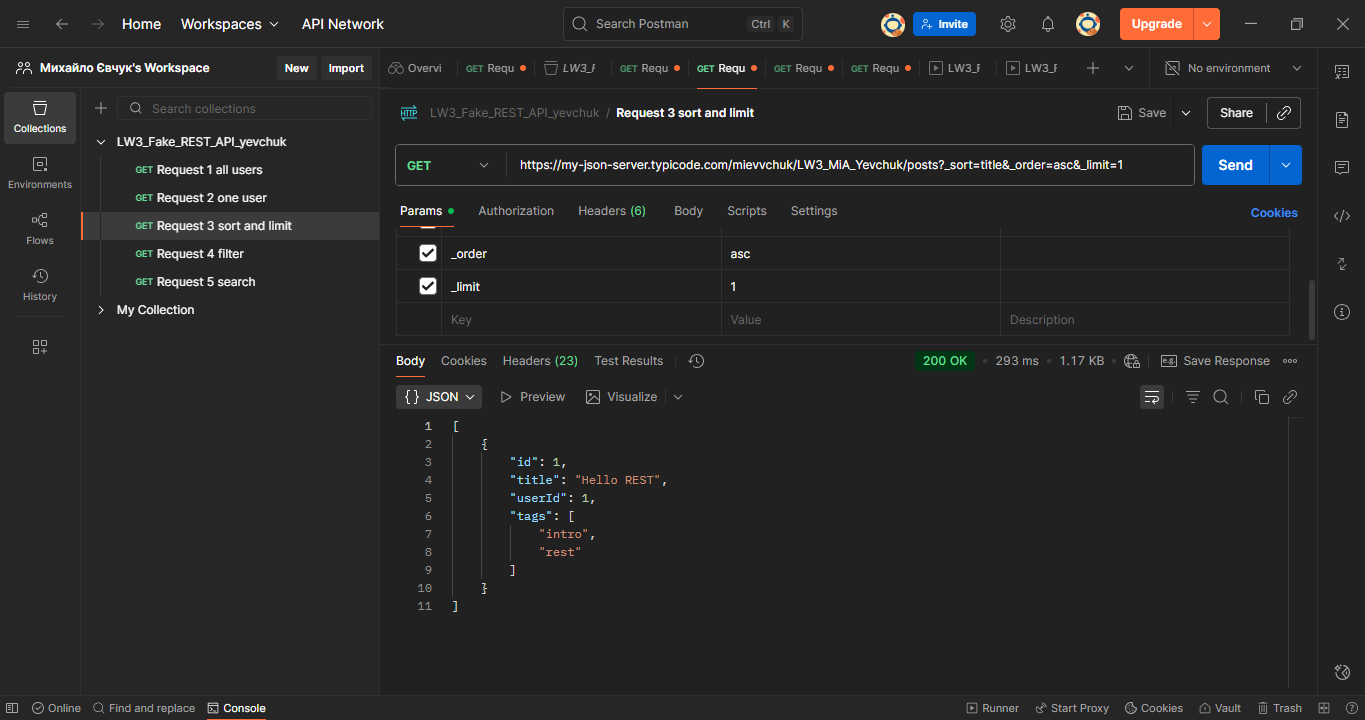


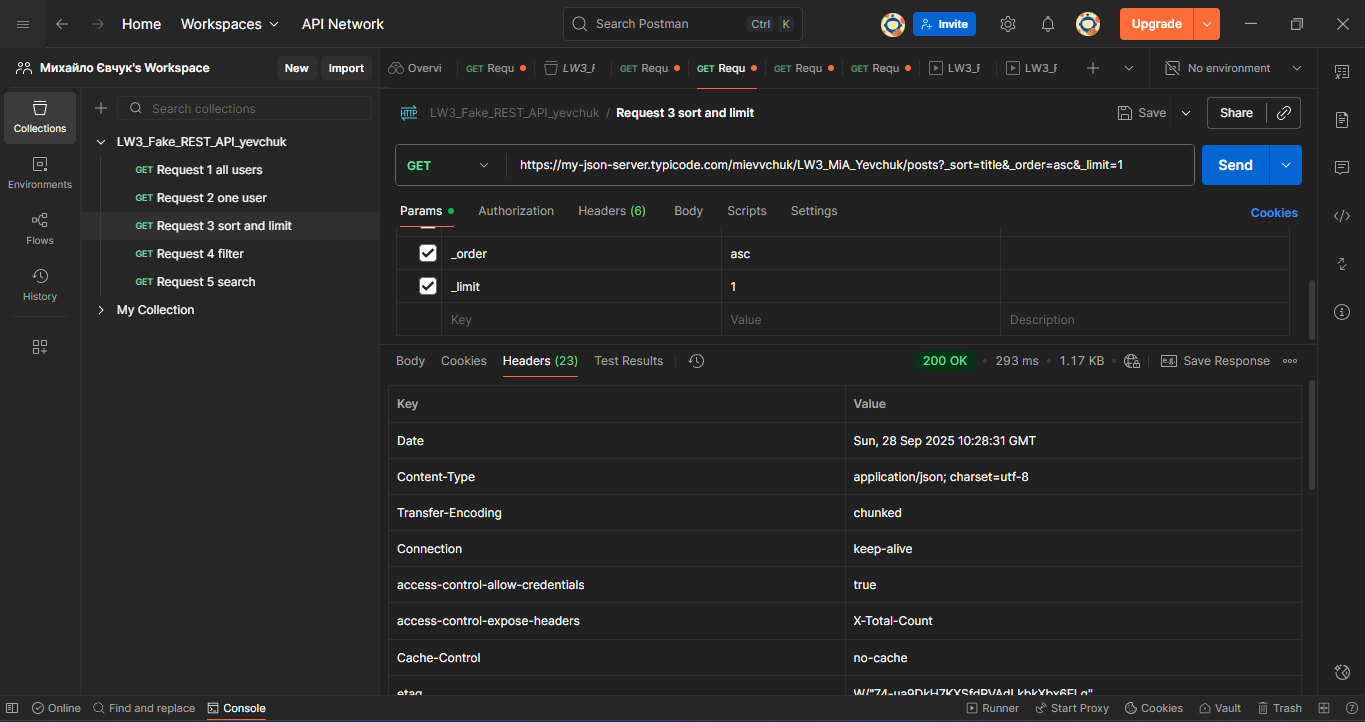
* 1. Виконаємо запит на одного користувача



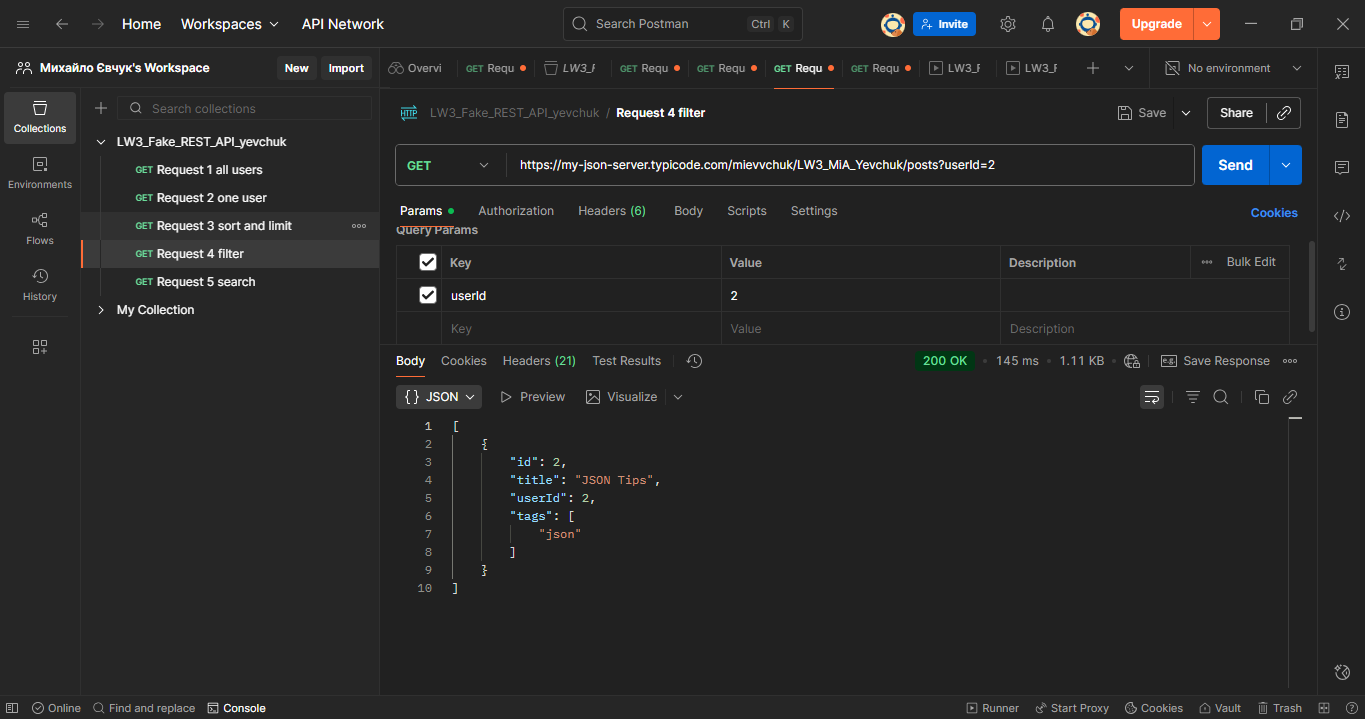


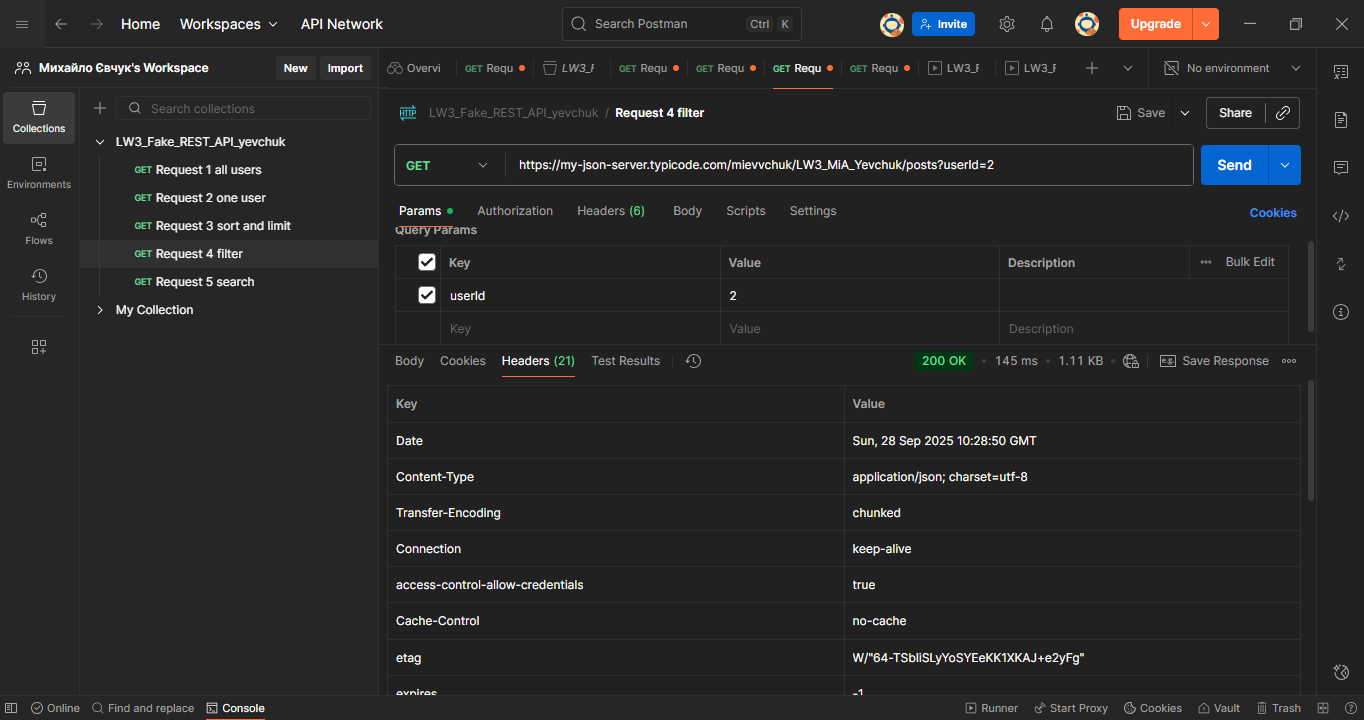
* 1. Виконаємо запит на сортування/ліміт



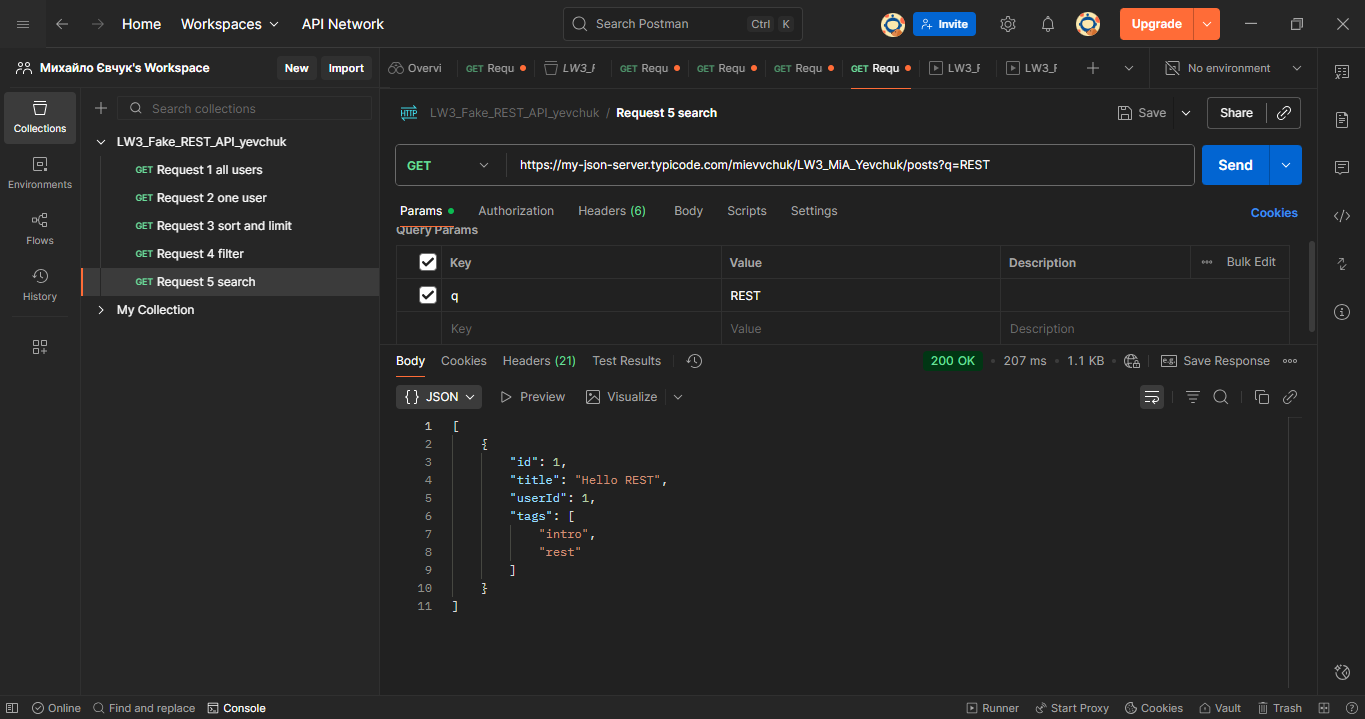


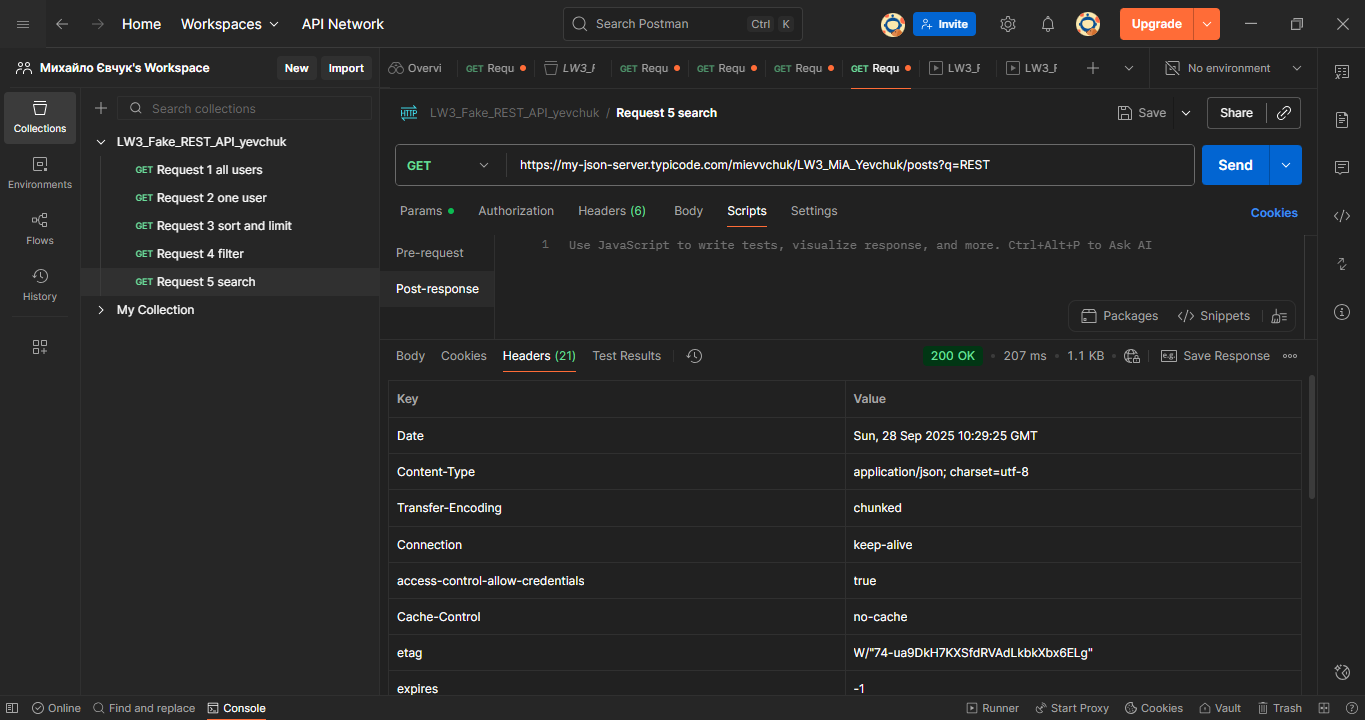
* 1. Виконаємо запит на пошук по фільтру





* 1. Виконаємо запит на пошук





1. Приклади через curl -i:
   1. Усі користувачі



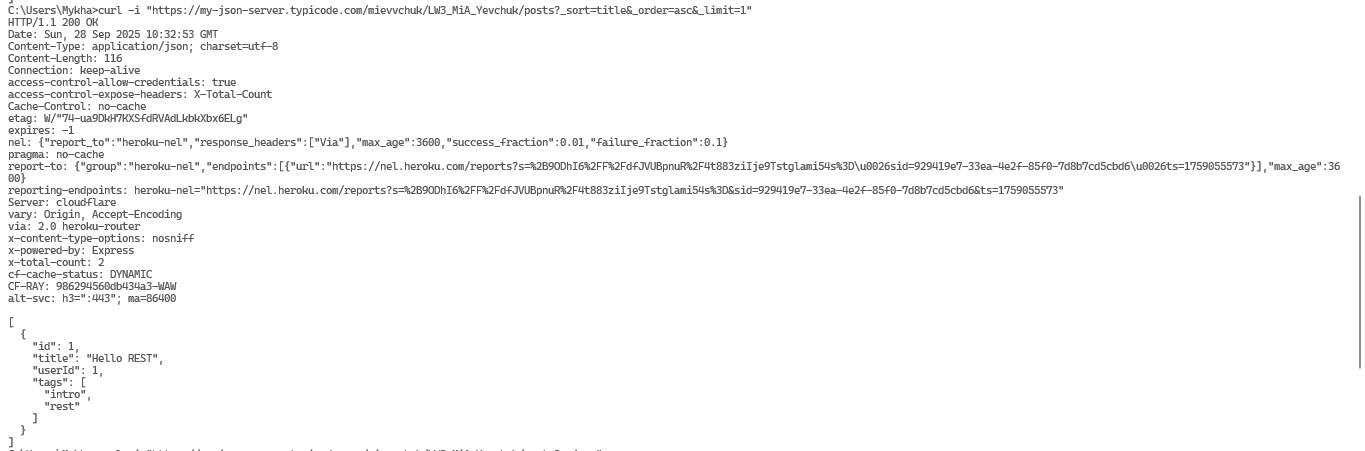
* 1. Один користувач



* 1. Фільтрація



* 1. Сортування + ліміт



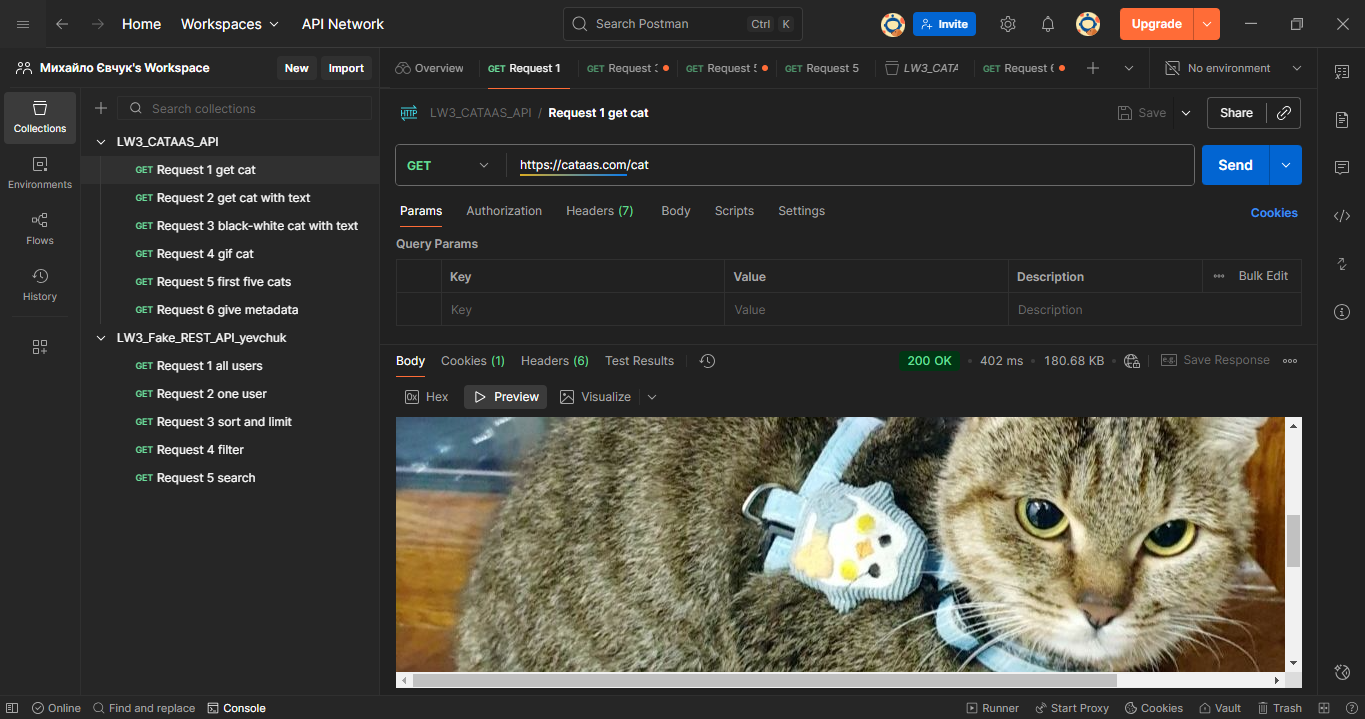
* 1. Пошук

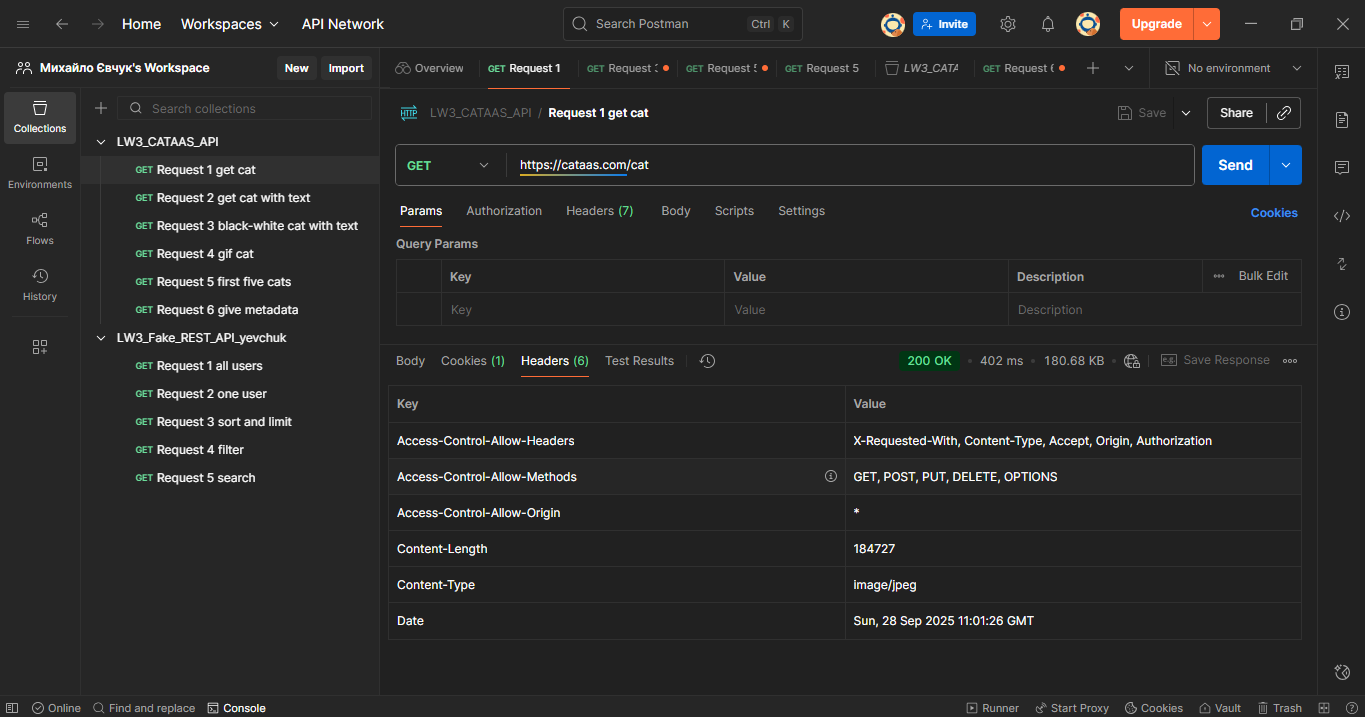


**Завдання №2**

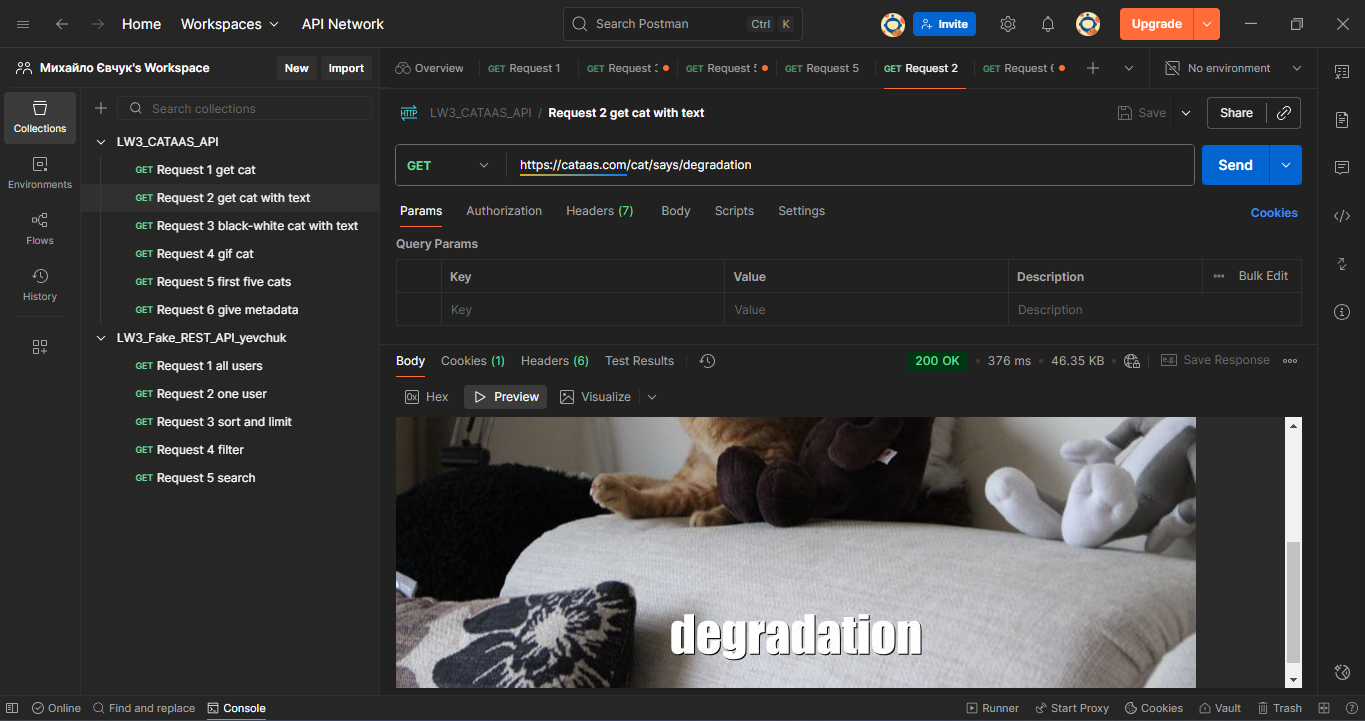
**Робота з API сервісу CATAAS (Cat as a Service) та SwaggerAPI**

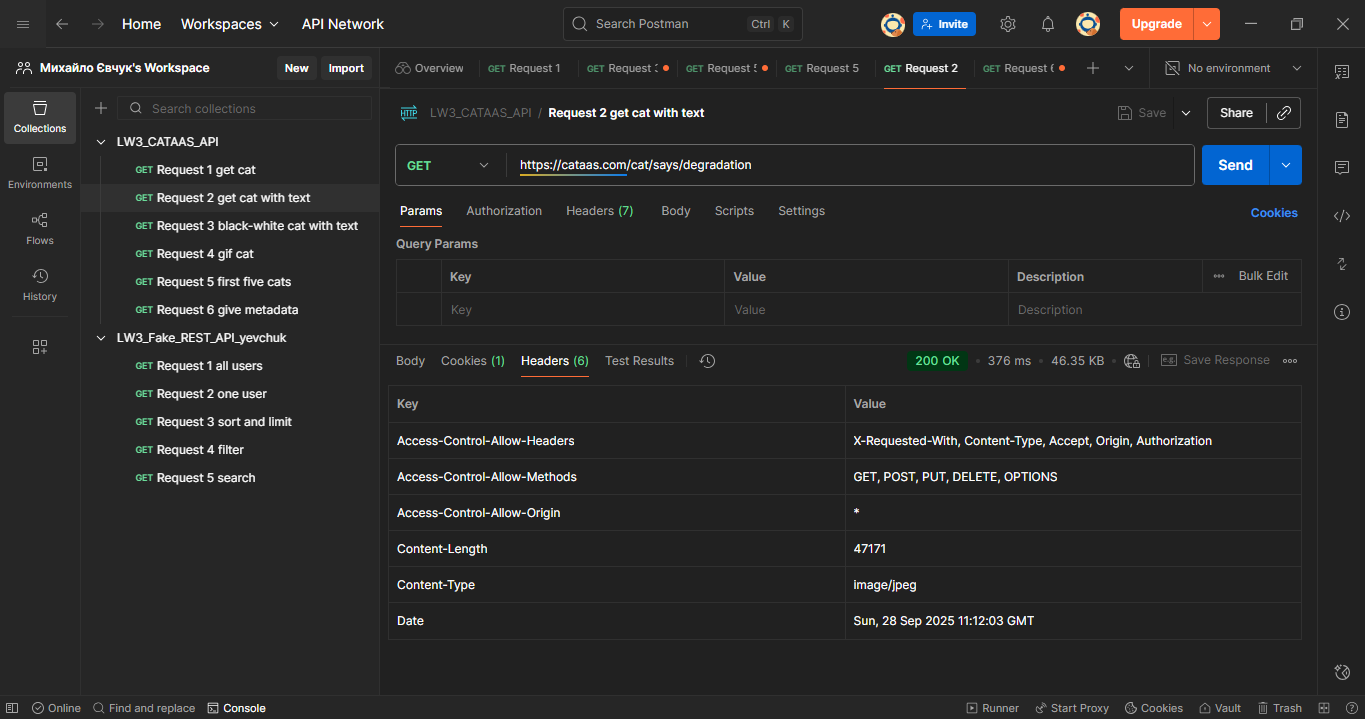
1. Створюємо колекцію **LW3\_CATAAS\_API**
2. Виконуємо запити
   * Отримати випадкове зображення кота



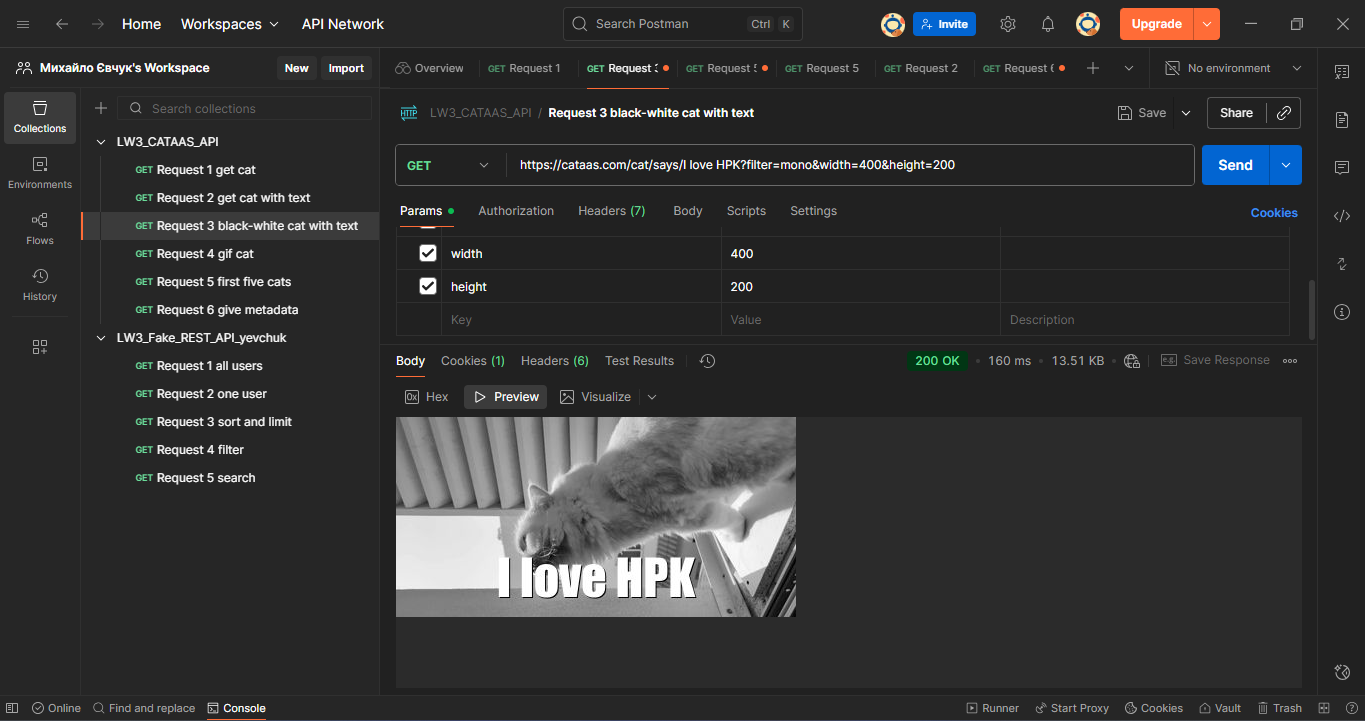


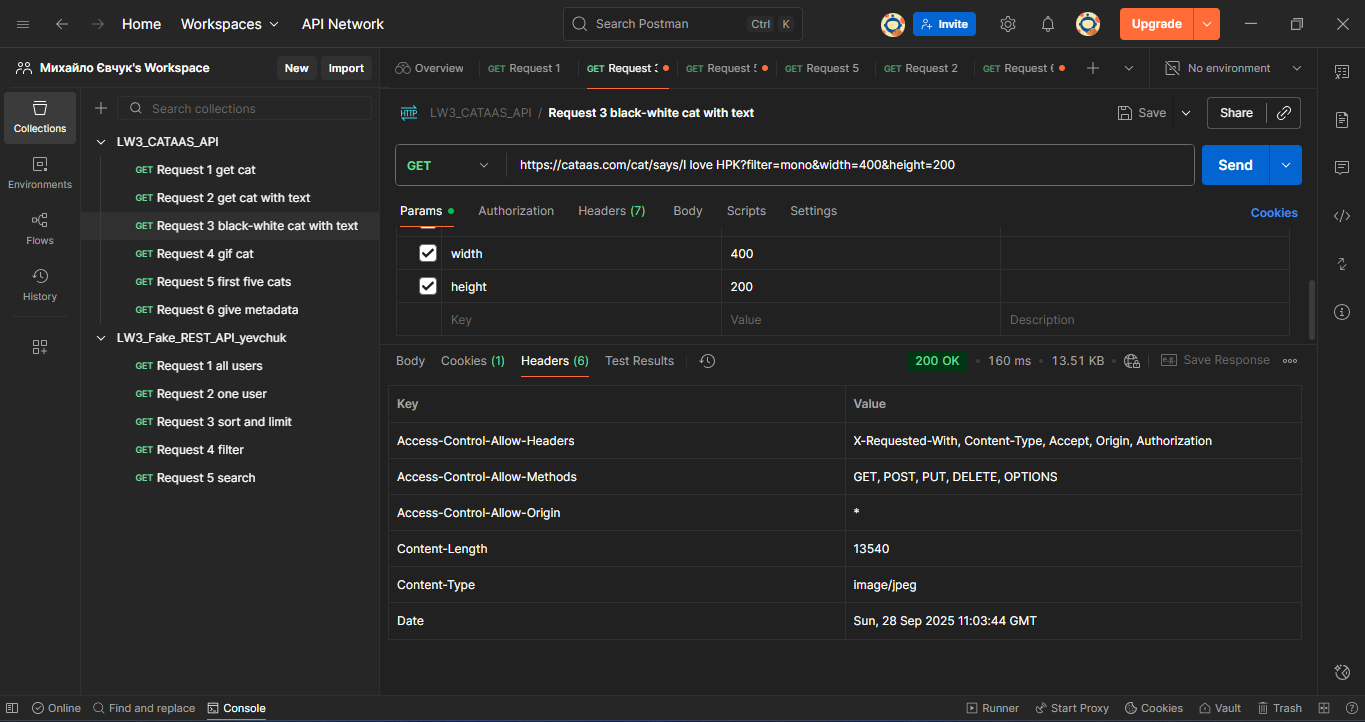
* Кіт із підписом вашого mood (картинка з текстом яка описує тебе).



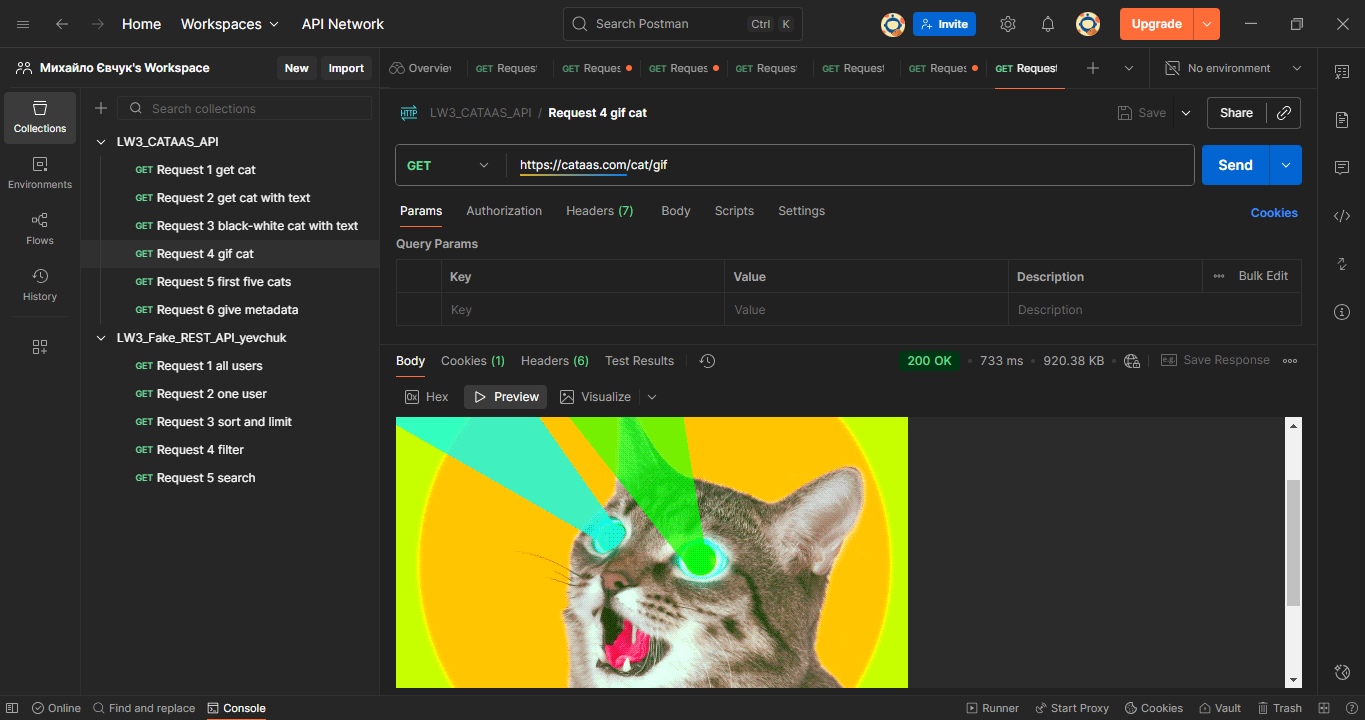


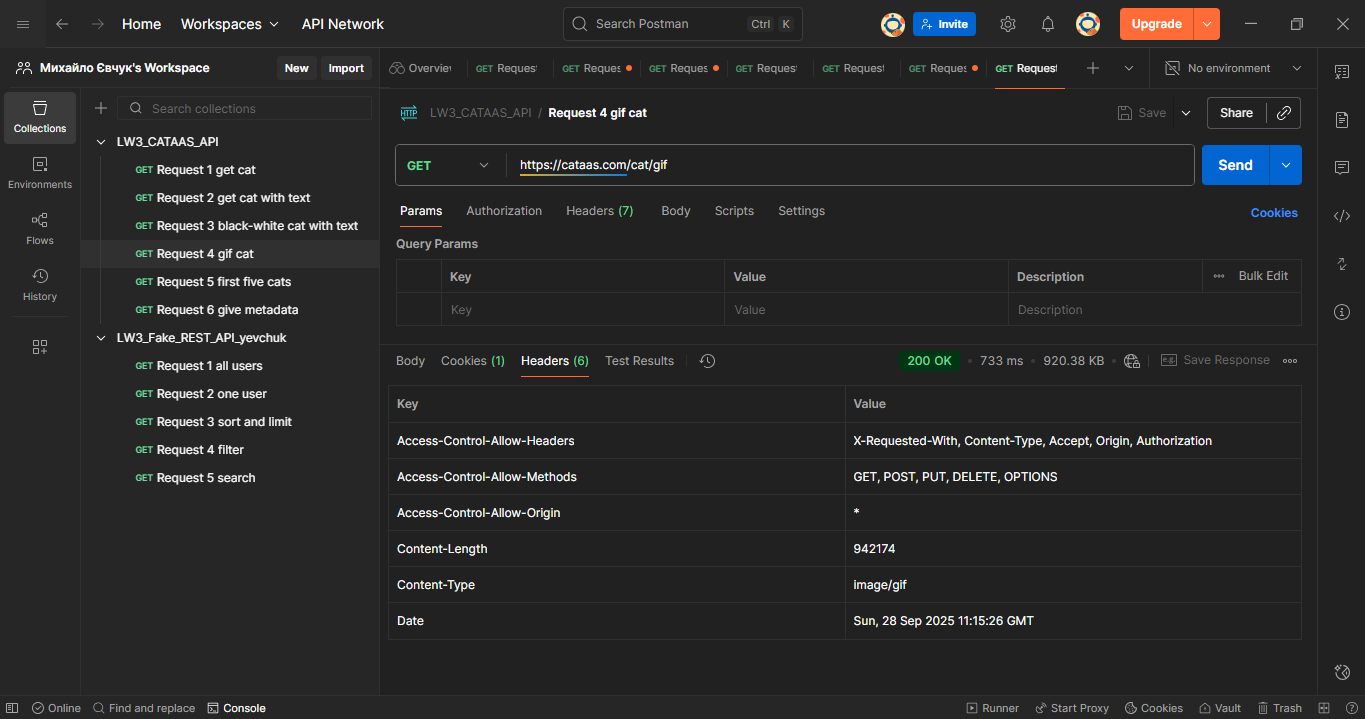
* Кіт із чорно-білим підписом. Спробуйте використати різні параметри запиту.



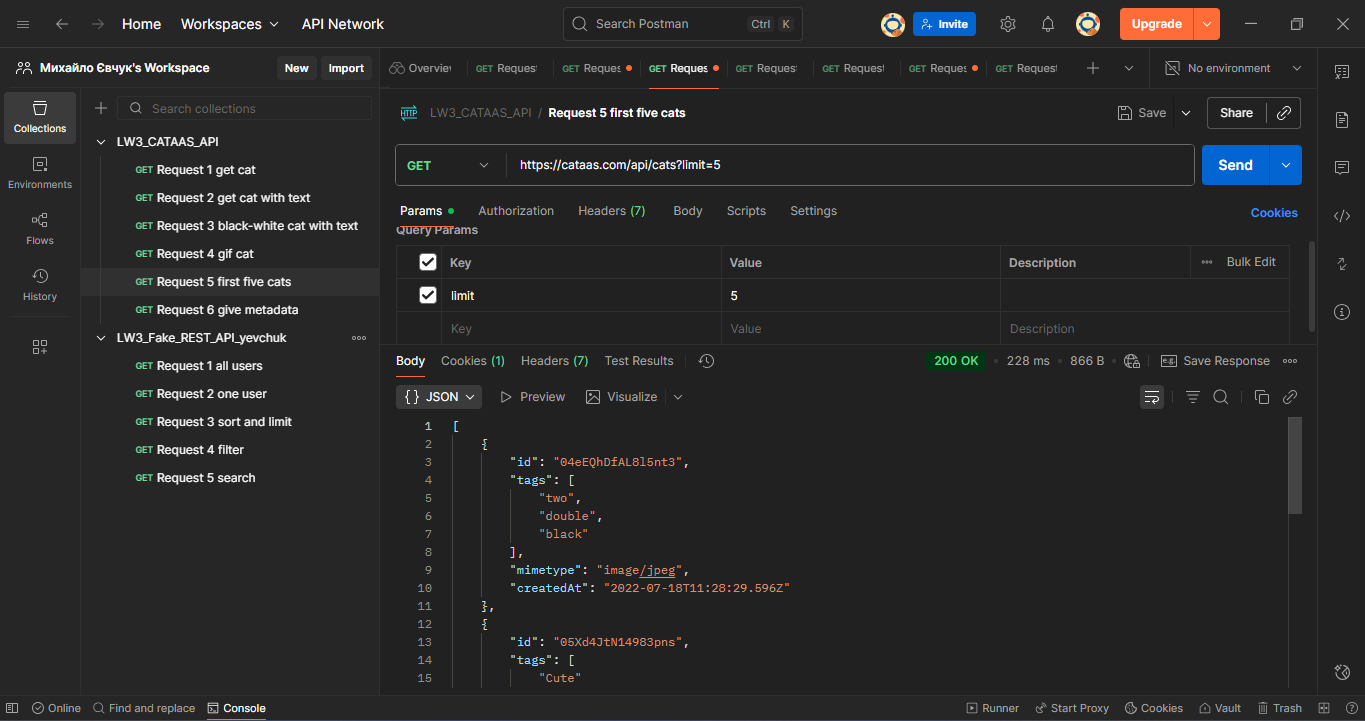


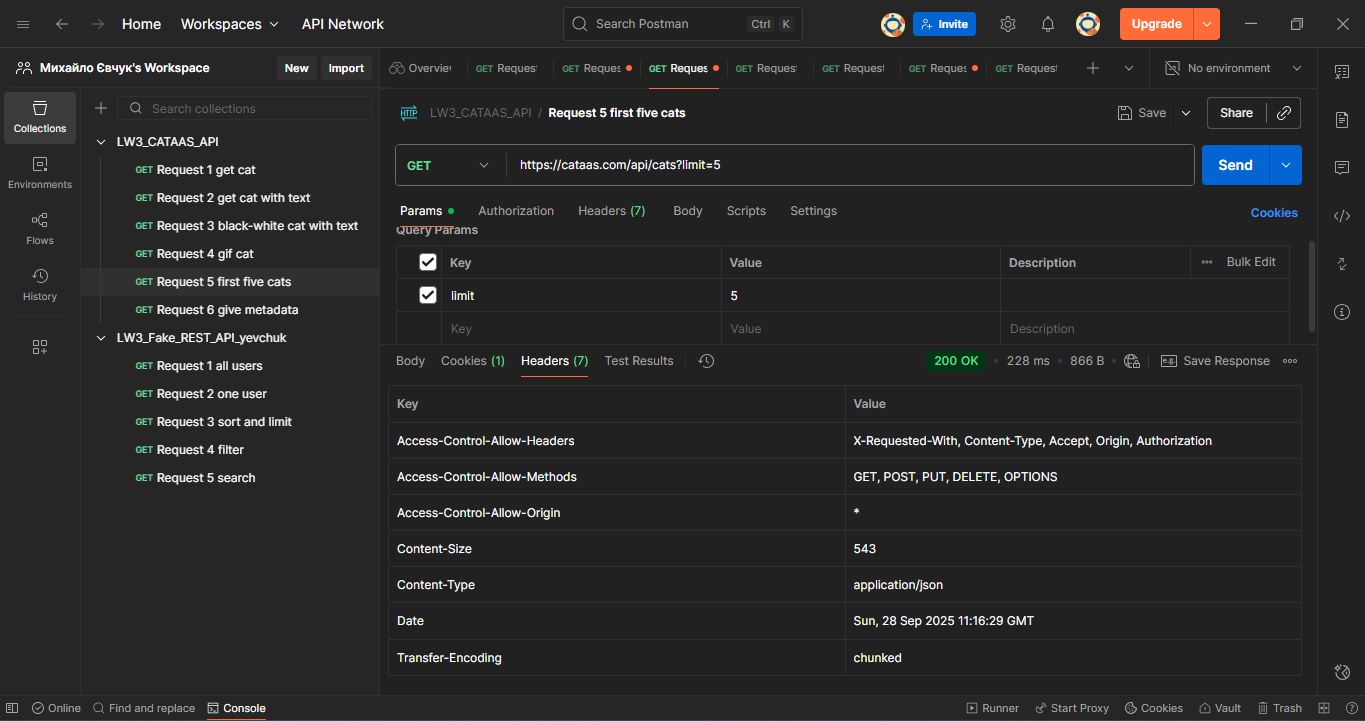
* Отримати випадкове gif-кота.



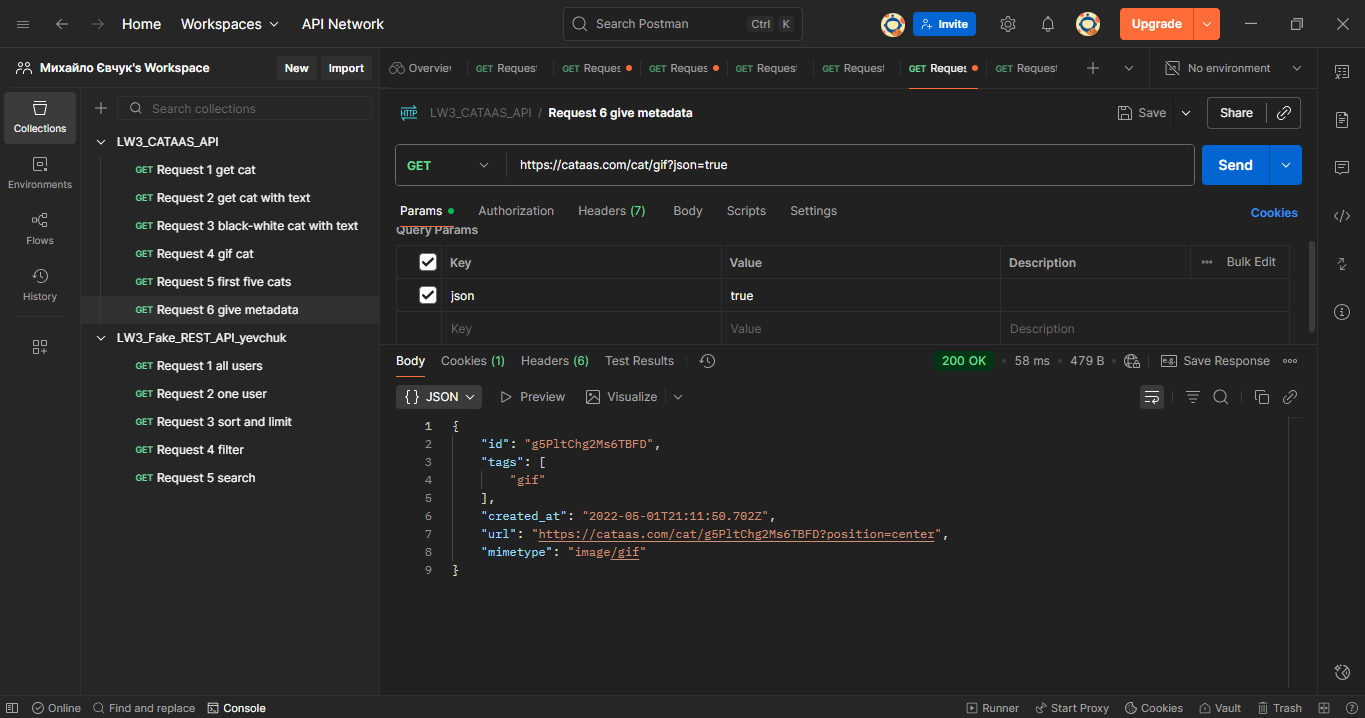


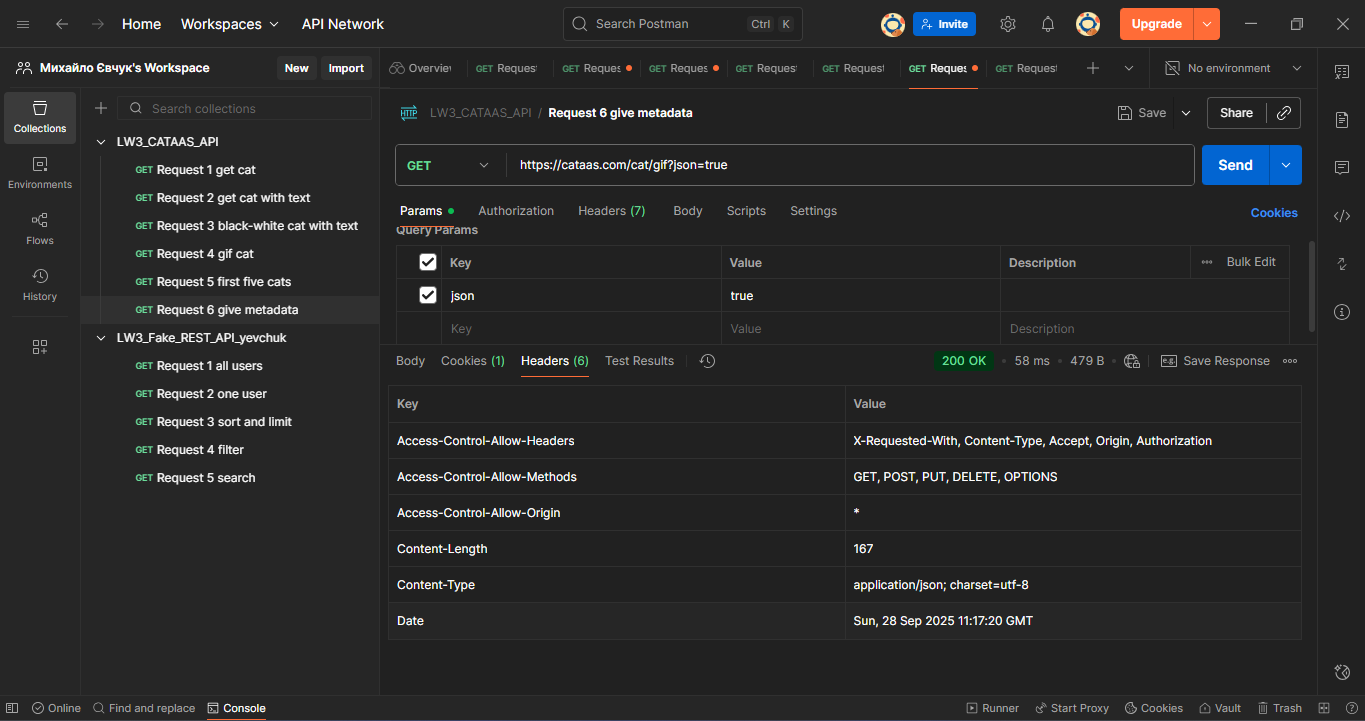
* Отримати JSON-список перших 5 котів.





* Отримати метадані кото-GIF у форматі JSON.

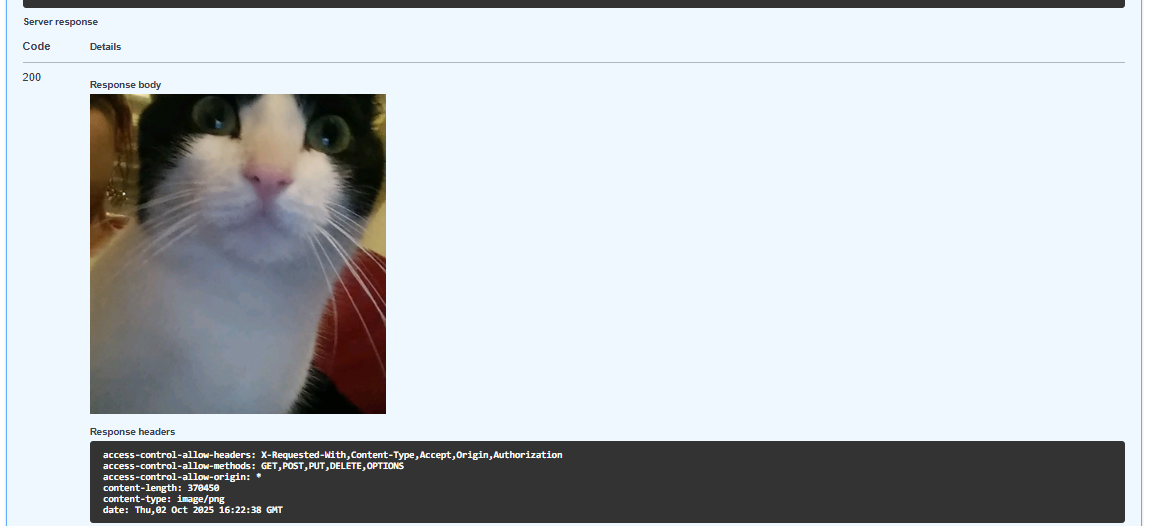




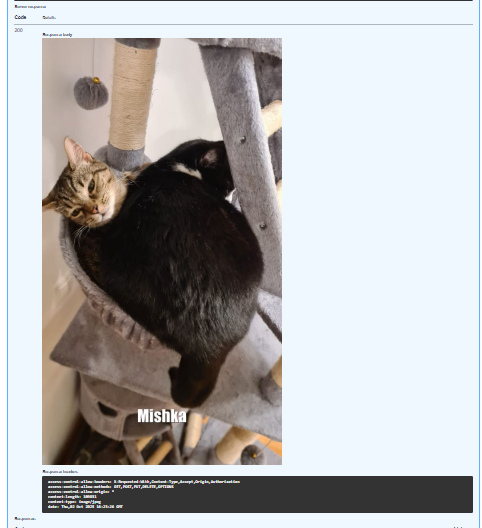
1. **Ознайомлення з документацією. Виконання запитів через Swagger UI**

**Виконаємо запити**

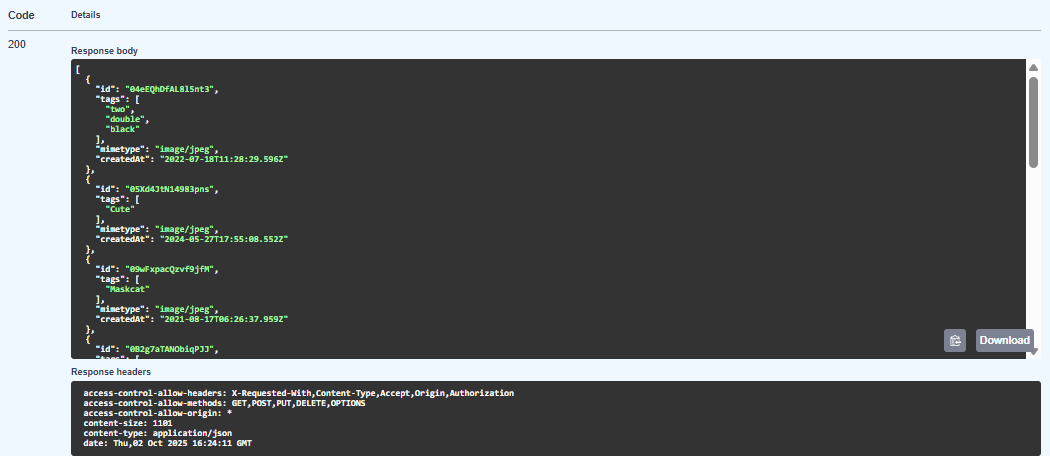
* 1. Виконаємо запит кота



* 1. Виконаємо запит на кота з текстом



* 1. Виконаємо запит на усіх котів

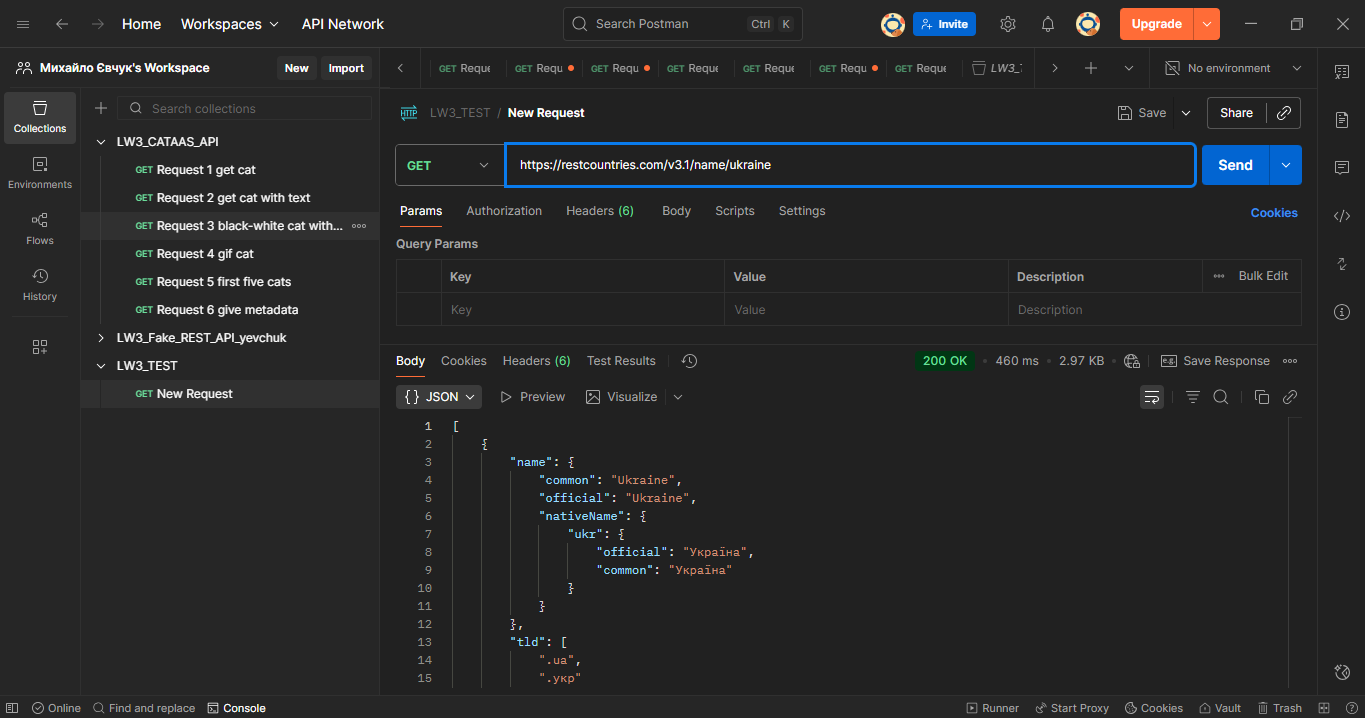


**Завдання №3**

**Console-клієнт до публічного REST API**

1. Створюємо консольний проєкт С# з назвою LW3\_MiA.
2. Крок 1

Виконаємо тестовий запит через PostMan



1. Крок 2

Створимо класи:

public class CountryName

{

public string common { get; set; } = string.Empty;

public string official { get; set; } = string.Empty;

}

public class Flags

{

public string png { get; set; } = string.Empty;

public string svg { get; set; } = string.Empty;

public string alt { get; set; } = string.Empty;

}

public class Country

{

public CountryName name { get; set; } = new CountryName();

public List<string> capital { get; set; } = new List<string>();

public string region { get; set; } = string.Empty;

public string subregion { get; set; } = string.Empty;

public long population { get; set; }

public Dictionary<string, string> languages { get; set; } = new Dictionary<string, string>();

public string cca2 { get; set; } = string.Empty;

public Flags flags { get; set; } = new Flags();

}

1. Крок 3

Налаштуємо HttpClient.

private static readonly HttpClient \_http = new HttpClient

{

BaseAddress = new Uri("https://restcountries.com"),

Timeout = TimeSpan.FromSeconds(15)

};

1. Крок 4

Виконаємо GET запит і запитуємо назву, столицю, регіон, підрегіон, населення, мова, прапор

var response = await \_http.GetAsync("/v3.1/name/ukraine?fields=name,capital,region,subregion,population,languages,cca2,flags");

response.EnsureSuccessStatusCode();

1. Крок 5

Отримаємо JSON як рядок

var json = await response.Content.ReadAsStringAsync();

Console.WriteLine("Raw JSON:");

Console.WriteLine(json);

1. Крок 6

Розпарсимо JSON

var countries = JsonSerializer.Deserialize<List<Country>>(json, new JsonSerializerOptions

{

PropertyNameCaseInsensitive = true

});

1. Крок 7

Використаємо дані з об'єкта

if (countries != null && countries.Count > 0)

{

var c = countries[0];

Console.WriteLine("\n=== Результат парсингу ===");

Console.WriteLine($"Назва: {c.name.common} ({c.name.official}) [{c.cca2}]");

Console.WriteLine($"Столиця: {(c.capital.Count > 0 ? string.Join(", ", c.capital) : "N/A")}");

Console.WriteLine($"Регіон: {c.region}");

Console.WriteLine($"Підрегіон: {c.subregion}");

Console.WriteLine($"Населення: {c.population:N0}");

Console.WriteLine($"Мови: {(c.languages.Count > 0 ? string.Join(", ", c.languages.Values) : "N/A")}");

Console.WriteLine($"Прапор (PNG): {c.flags.png}");

Console.WriteLine($"Прапор (SVG): {c.flags.svg}");

}

1. Повний код програми:

using System;

using System.Net.Http;

using System.Text.Json;

using System.Threading.Tasks;

using System.Collections.Generic;

public class CountryName

{

public string common { get; set; } = string.Empty;

public string official { get; set; } = string.Empty;

}

public class Flags

{

public string png { get; set; } = string.Empty;

public string svg { get; set; } = string.Empty;

public string alt { get; set; } = string.Empty;

}

public class Country

{

public CountryName name { get; set; } = new CountryName();

public List<string> capital { get; set; } = new List<string>();

public string region { get; set; } = string.Empty;

public string subregion { get; set; } = string.Empty;

public long population { get; set; }

public Dictionary<string, string> languages { get; set; } = new Dictionary<string, string>();

public string cca2 { get; set; } = string.Empty;

public Flags flags { get; set; } = new Flags();

}

class Program

{

private static readonly HttpClient \_http = new HttpClient

{

BaseAddress = new Uri("https://restcountries.com"),

Timeout = TimeSpan.FromSeconds(15)

};

static async Task Main()

{

try

{

Console.WriteLine("Запит до REST Countries...");

var url = "/v3.1/name/ukraine?fields=name,capital,region,subregion,population,languages,cca2,flags";

var response = await \_http.GetAsync(url);

Console.WriteLine($"Request URL: {(\_http.BaseAddress + url)}");

Console.WriteLine($"Status: {(int)response.StatusCode} {response.ReasonPhrase}");

Console.WriteLine($"Content-Type: {response.Content.Headers.ContentType}");

response.EnsureSuccessStatusCode();

var json = await response.Content.ReadAsStringAsync();

Console.WriteLine("Raw JSON:");

Console.WriteLine(json);

var countries = JsonSerializer.Deserialize<List<Country>>(json, new JsonSerializerOptions

{

PropertyNameCaseInsensitive = true

});

if (countries != null && countries.Count > 0)

{

var c = countries[0];

Console.WriteLine("\n=== Результат парсингу ===");

Console.WriteLine($"Назва: {c.name.common} ({c.name.official}) [{c.cca2}]");

Console.WriteLine($"Столиця: {(c.capital.Count > 0 ? string.Join(", ", c.capital) : "N/A")}");

Console.WriteLine($"Регіон: {c.region}");

Console.WriteLine($"Підрегіон: {c.subregion}");

Console.WriteLine($"Населення: {c.population:N0}");

Console.WriteLine($"Мови: {(c.languages.Count > 0 ? string.Join(", ", c.languages.Values) : "N/A")}");

Console.WriteLine($"Прапор (PNG): {c.flags.png}");

Console.WriteLine($"Прапор (SVG): {c.flags.svg}");

}

}

catch (Exception ex)

{

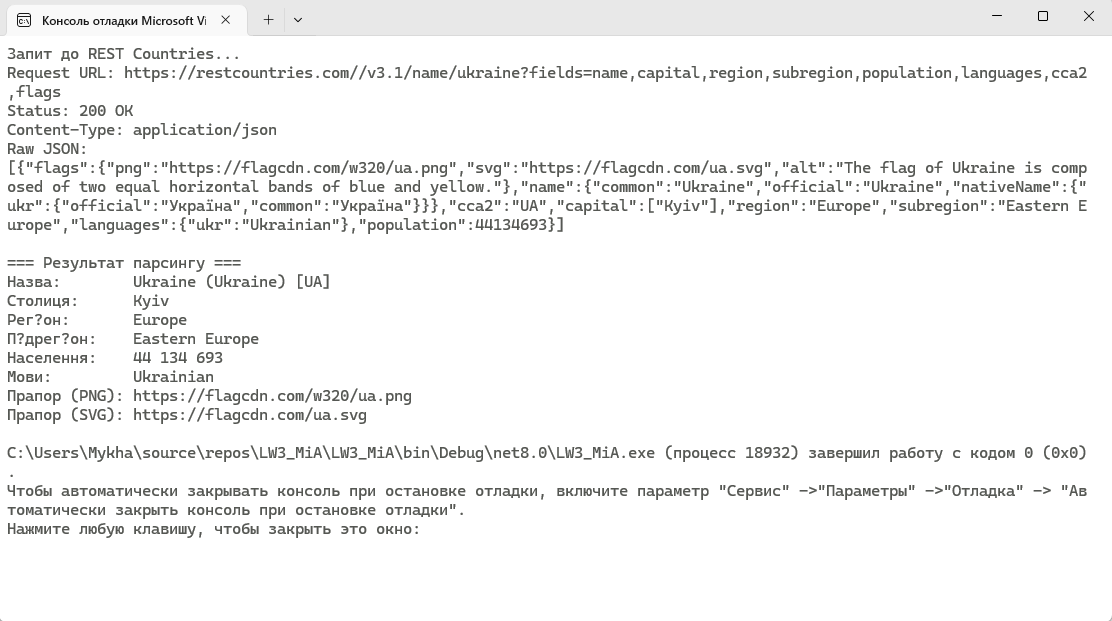
Console.WriteLine("Помилка: " + ex.Message);

}

}

}

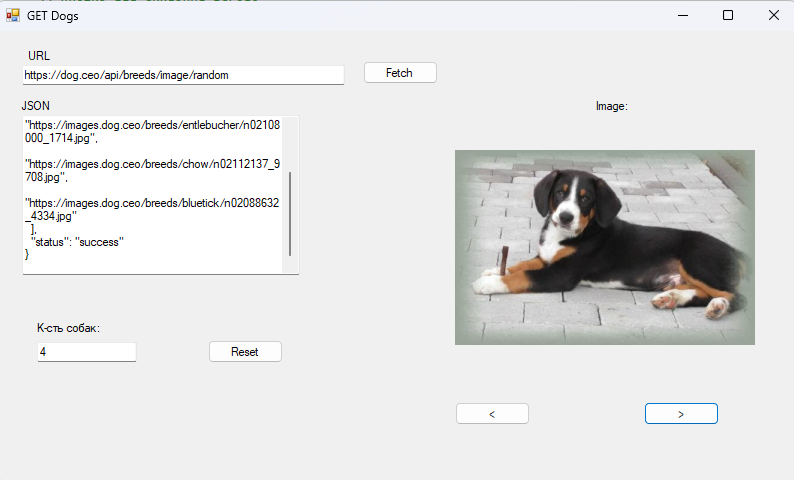
Результат програми:



**Завдання №4**

**Інтеграція з Dog.CEO API у WinForms**

1. **Створити галерею світлин собак**

****

**Код програми:**

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Net.Http;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace DogGalleryApp

{

public partial class Form1 : Form

{

private static readonly HttpClient \_http = new HttpClient();

private List<string> dogImages = new List<string>();

private int currentIndex = 0;

public Form1()

{

InitializeComponent();

pictureBox1.SizeMode = PictureBoxSizeMode.Zoom; // Переконайся, що pictureBox1 — це правильне ім'я елемента

// Задаємо початковий URL

textBox1.Text = "https://dog.ceo/api/breeds/image/random";

}

// Метод для завантаження зображень

private async Task LoadImages(int count)

{

try

{

string url = $"https://dog.ceo/api/breeds/image/random/{count}";

var json = await \_http.GetStringAsync(url);

var dogResponse = JsonConvert.DeserializeObject<DogResponse>(json);

// Вивести отриманий JSON у textBox2

textBox2.Text = JsonConvert.SerializeObject(dogResponse, Formatting.Indented); // Форматуємо JSON для читабельності

if (dogResponse != null && dogResponse.message.Count > 0)

{

dogImages = dogResponse.message;

currentIndex = 0; // Починаємо з першого зображення

DisplayImage();

}

else

{

MessageBox.Show("Не вдалося отримати зображення.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

catch (Exception ex)

{

MessageBox.Show("Помилка: " + ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

// Метод для відображення зображення

private void DisplayImage()

{

if (dogImages.Count > 0 && currentIndex >= 0 && currentIndex < dogImages.Count)

{

var imageUrl = dogImages[currentIndex];

var imageBytes = \_http.GetByteArrayAsync(imageUrl).Result;

using (var ms = new MemoryStream(imageBytes))

{

pictureBox1.Image = Image.FromStream(ms);

}

}

}

// Кнопка для завантаження N зображень

private async void buttonLoad\_Click(object sender, EventArgs e)

{

int count;

// Перевірка, чи введене значення є числом

if (int.TryParse(textBox3.Text, out count) && count >= 1)

{

await LoadImages(count); // Завантажуємо зображення

}

else

{

MessageBox.Show("Будь ласка, введіть коректне число більше 0.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

// Кнопка для перегляду попереднього зображення

private void buttonPrev\_Click(object sender, EventArgs e)

{

if (dogImages.Count > 0 && currentIndex > 0)

{

currentIndex--;

DisplayImage();

}

}

// Кнопка для перегляду наступного зображення

private void buttonNext\_Click(object sender, EventArgs e)

{

if (dogImages.Count > 0 && currentIndex < dogImages.Count - 1)

{

currentIndex++;

DisplayImage();

}

}

// Кнопка для скидання всього

private void buttonReset\_Click(object sender, EventArgs e)

{

textBox2.Clear();

pictureBox1.Image = null;

dogImages.Clear();

currentIndex = 0; // Скидаємо індекс

textBox3.Clear(); // Очищаємо TextBox

}

}

// Клас для парсингу відповіді з API

public class DogResponse

{

public List<string> message { get; set; } = new List<string>();

public string status { get; set; } = string.Empty;

}

}

**Завдання №5**

**WinForms-клієнт до публічного REST API**

Створити простий клієнт на C# WinForms, який зчитує дані з відкритого REST API, відображає список ресурсів і деталі одного ресурсу, підтримує мінімальні фільтри/пошук, та коректно обробляє помилки (якщо АPI дозволяє). API можна обрати зі списку публічних API або власне.

**Що необхідно зробити:**

1. Отримати колекцію (масив) JSON-об’єктів із відкритого API (Додаток А).
2. Відобразити в **DataGridView** (або ListView) з можливістю прокрутки.
3. Якщо можливості АРІ дозволяють (більшіть так) виконати **мінімальний пошук/фільтр** (по одному полю або query-параметру API).
4. При виборі рядка зі списку — виконати **другий запит** (наприклад, GET /resource/{id} або аналог) і показати деталі у правій панелі (Labels/TextBoxes/PictureBox).
5. Кнопка **Refresh** для повторного завантаження списку.
6. Відображати користувачу читабельні повідомлення при: відсутності інтернету, 4xx/5xx, таймауті, невалідному JSON.
7. Блокувати кнопки під час запиту, показувати індикатор стану (**StatusStrip/Label**).

**Нефункціональні вимоги**

* .NET 6/7/8, WinForms, C#, за бажанням інша технологія, інші технології до написання UI. It’s up to you!
* Структура проєкту:

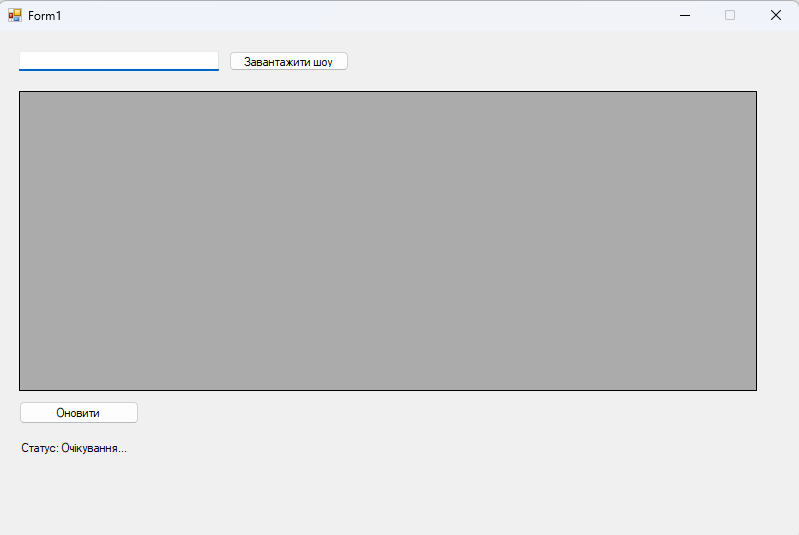
Forms/, Services/ApiClient.cs, Models/Dto.cs, Utils/.

* Код-стайл: читабельні назви, мінімальні коментарі, винесення констант (базова адреса API тощо).

**Оберемо 13 варіант**

|  |  |  |
| --- | --- | --- |
| 13 | TVMaze | [https://api.tvmaze.com](https://api.tvmaze.com/) |

**Макет програми:**

****

**Show.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TVMazeClient.Models

{

public class Show

{

public int Id { get; set; }

public string Name { get; set; }

public string Language { get; set; }

public string Premiered { get; set; }

public string Summary { get; set; }

}

}

**ApiClient.cs**

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Net.Http;

using System.Threading.Tasks;

using TVMazeClient.Models;

namespace TVMazeClient.Services

{

public class ApiClient

{

private static readonly HttpClient client = new HttpClient();

private const string BaseUrl = "https://api.tvmaze.com";

// Отримання списку шоу

public async Task<List<Show>> GetShowsAsync(string search = "")

{

try

{

var url = $"{BaseUrl}/search/shows?q={search}";

var response = await client.GetStringAsync(url);

var shows = JsonConvert.DeserializeObject<List<ApiShowResponse>>(response);

return shows?.ConvertAll(s => new Show

{

Id = s.Show.Id,

Name = s.Show.Name,

Language = s.Show.Language,

Premiered = s.Show.Premiered,

Summary = s.Show.Summary

});

}

catch (Exception ex)

{

throw new Exception("Error fetching shows: " + ex.Message);

}

}

// Отримання деталей шоу

public async Task<Show> GetShowDetailsAsync(int id)

{

try

{

var url = $"{BaseUrl}/shows/{id}";

var response = await client.GetStringAsync(url);

var show = JsonConvert.DeserializeObject<Show>(response);

return show;

}

catch (Exception ex)

{

throw new Exception("Error fetching show details: " + ex.Message);

}

}

// Внутрішній клас для парсингу відповіді API

private class ApiShowResponse

{

public Show Show { get; set; }

}

}

}

**Form1.cs**

using System;

using System.Collections.Generic;

using System.Windows.Forms;

using TVMazeClient.Models;

using TVMazeClient.Services;

namespace LW3\_Task5\_MiA

{

public partial class Form1 : Form

{

private readonly ApiClient \_apiClient;

private List<Show> \_shows;

public Form1()

{

InitializeComponent();

\_apiClient = new ApiClient();

}

// Завантаження шоу по запиту

private async void btnLoad\_Click(object sender, EventArgs e)

{

try

{

button1.Enabled = false;

label1.Text = "Loading...";

// Отримуємо шоу за допомогою API

\_shows = await \_apiClient.GetShowsAsync(textBox1.Text);

// Відображення шоу в DataGridView

dataGridView1.DataSource = \_shows;

// Налаштування розтягування колонок

dataGridView1.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells);

// Якщо хочете вручну налаштувати ширину колонок, можна зробити так:

dataGridView1.Columns["Id"].Width = 50;

dataGridView1.Columns["Name"].Width = 200;

dataGridView1.Columns["Language"].Width = 100;

dataGridView1.Columns["Premiered"].Width = 120;

dataGridView1.Columns["Summary"].Width = 270;

// Для забезпечення гарного вигляду колонок можна додати мінімальну ширину

dataGridView1.Columns["Id"].MinimumWidth = 50;

dataGridView1.Columns["Name"].MinimumWidth = 100;

dataGridView1.Columns["Language"].MinimumWidth = 80;

dataGridView1.Columns["Premiered"].MinimumWidth = 100;

dataGridView1.Columns["Summary"].MinimumWidth = 150;

label1.Text = $"{\_shows.Count} shows loaded.";

}

catch (Exception ex)

{

MessageBox.Show($"Error: {ex.Message}", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

finally

{

button1.Enabled = true;

}

}

// Обробка кліку по шоу для отримання детальної інформації

private async void dataGridView\_CellClick(object sender, DataGridViewCellEventArgs e)

{

if (e.RowIndex >= 0)

{

int id = (int)dataGridView1.Rows[e.RowIndex].Cells["Id"].Value;

var showDetails = await \_apiClient.GetShowDetailsAsync(id);

// Відображення деталей шоу

label2.Text = showDetails.Name;

label3.Text = showDetails.Summary;

label4.Text = showDetails.Language;

label5.Text = showDetails.Premiered;

}

}

// Оновлення списку шоу

private void btnRefresh\_Click(object sender, EventArgs e)

{

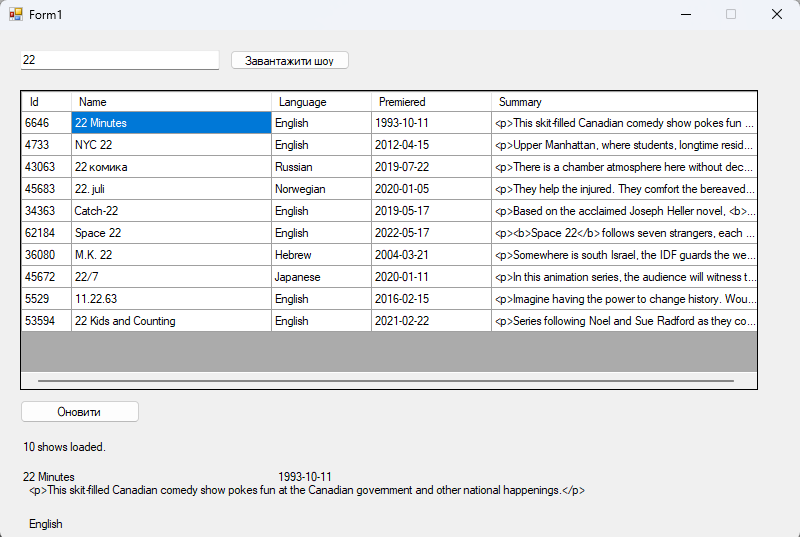
btnLoad\_Click(sender, e);

}

}

}

**Результат програми:**

****