Лабораторна робота №4

Мета:Ознайомитися з процесом створення клієнт-серверної архітектури. Навчитися створювати RESTful API для взаємодії між клієнтом і сервером. Розвинути вміння проєктувати та реалізовувати ендпойнти для типових CRUD-операцій.

Завдання 1

Ознайомитися з кроками створення **ASP.NET Core Minimal API** | Check out the.

Створити **ASP.NET Core Minimal API**:

1. Створити **3–4 групи маршрутів** відповідно до Додатку А(наприклад: /todos, /users, /categories).
2. Використати **in-memory "сховище"** (простий список List в коді) для зберігання даних.
3. Реалізувати **CRUD** для кожної групи маршрутів:
   - **Create** (POST) — створення нового об'єкта.
   - **Read** (GET) — отримання списку об'єктів або одного об'єкта за ID.
   - **Update** (PUT/PATCH) — оновлення існуючого об'єкта.
   - **Delete** (DELETE) — видалення об'єкта за ID.
4. Виконати **валідацію даних** під час створення нових об'єктів (наприклад, перевірка порожніх полів, мінімальної довжини рядка, формату email тощо).
5. Повертає стандартні HTTP-відповіді:
   - 200 OK — успішний запит,
   - 201 Created — створення нового ресурсу,
   - 400 Bad Request — помилка валідації,
   - 404 Not Found — об'єкт не знайдено.
6. Для кожної групи створити **окремий набір ендпойнтів** у коді.
7. **Винести ендпойнти в окремі файли** (наприклад: Endpoints/TodoEndpoints.cs, Endpoints/UserEndpoints .cs) і підключити їх у Program.cs.
8. **Організувати структуру проєкту** за папками:

/Models
/Endpoints
/Data (за потреби)
Program.cs

9. **Додати Swagger (OpenAPI)** для зручного тестування API через браузер:

					ЛР.ОК24.ПІ231.13.04		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Євчук М.В.			Створення RESTful API	Літ.	Арк. 1 / Акрушів 35
Перевір.		Жереб Д.В.					
Н. Контр.						ХПК	
Затверд.							

10. Перевірити роботу API через **Swagger** або **Postman**.

Завдання 2

Ознайомитися з кроками створення **ASP.NET Core Web API** | Check out the.

Створити **ASP.NET Core Web API**:

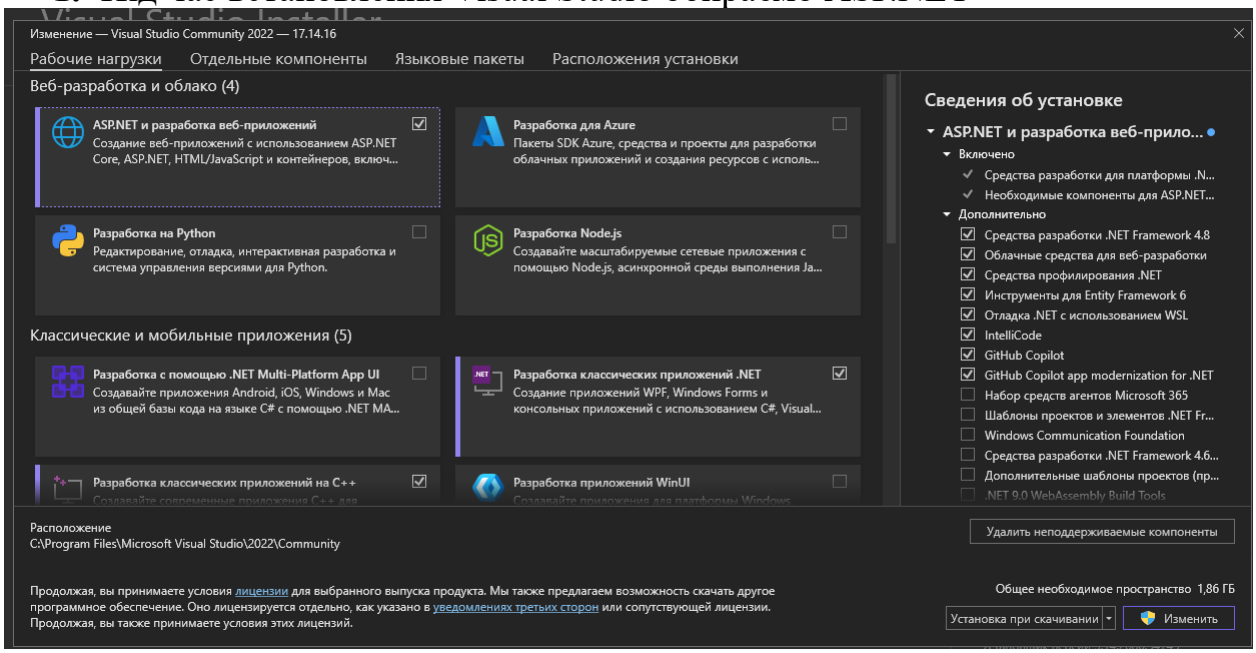
1. Створити **3–4 контролера** відповідно до Додатку А(наприклад: TodoController, UserController, CategoryController).
2. Створити відповідні моделі даних. Використати Enum, як поле класу. Про Enum тут
3. Використати **in-memory "сховище"** (простий список List в коді) для зберігання даних.
4. Реалізувати **CRUD** для кожної моделі в контролерах:
   - **Create** (POST) — створення нового об'єкта.
   - **Read** (GET) — отримання списку об'єктів або одного об'єкта за ID.
   - **Update** (PUT/PATCH) — оновлення існуючого об'єкта.
   - **Delete** (DELETE) — видалення об'єкта за ID.
5. Виконати **валідацію даних** під час створення нових об'єктів (наприклад, перевірка порожніх полів, мінімальної довжини рядка, формату email тощо). Для однієї моделі (самої меншої за к-тю полів) виконати валідацію через DataAnnotations. Для інших моделей виконати валідація через FluentValidation. Для одного з полів використати Regular Expressions
6. Повертає стандартні HTTP-відповіді:
   - 200 OK — успішний запит,
   - 201 Created — створення нового ресурсу,
   - 400 Bad Request — помилка валідації,
   - 404 Not Found — об'єкт не знайдено.
7. **Організувати структуру проєкту** за папками:
   /Models
   /Controllers
   /Data (за потреби)
   /Validators
   Program.cs

8. Перевірити роботу API через **Swagger** або **Postman**.
9. Підготувати **звіт**, який містить:
   - короткий опис проєкту;
   - код Program.cs, моделей та ендпойнтів;
   - скріншоти виконання запитів (GET, POST, помилки валідації);

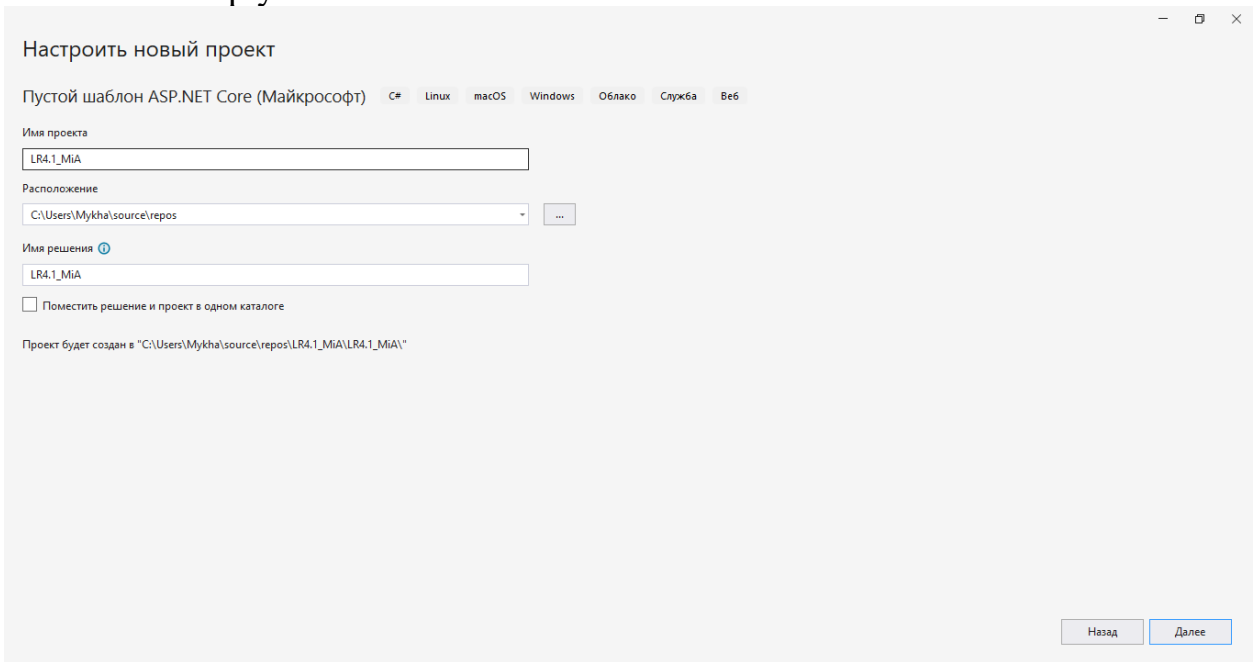
Хід роботи

40.Онлайн-сервіс для обміну кулінарними рецептами з рейтингами

1. Під час встановлення Visual Studio обираємо ASP.NET



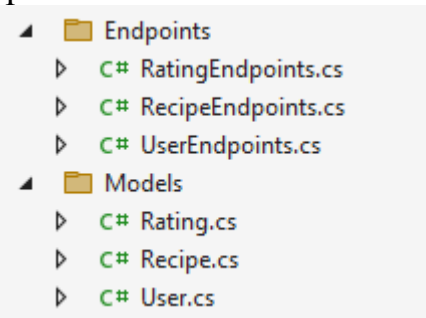
2. Перейдемо до створення проєкту. Для цього знаходимо проєкт ASP.NET Core empty.



Перейдемо до написання програми.

Створимо папки Models, Endpoints. В папці Models створюємо 3 класа, а саме User.cs, Recipe.cs, Rating.cs. В папці Endpoints створюємо також 3 класа, а саме UserEndpoints.cs, RecipeEndpoints.cs, RatingEndpoints.cs.

Ось як виглядає наша ієрархія.



Тепер перейдемо до написання коду.

Models

User.cs

```
namespace LW4.Task1_MiA.Models
{
    public class User
    {
        public int Id { get; set; }
        public string Username { get; set; }
        public string Email { get; set; }
    }
}
```

Recipe.cs

```
namespace LW4.Task1_MiA.Models
{
    public class Recipe
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Instructions { get; set; }
        public List<Rating> Ratings { get; set; }
    }
}
```

Rating.cs

```
namespace LW4.Task1_MiA.Models
{
    public class Rating
    {
        public int Id { get; set; }
        public int RecipeId { get; set; }
        public int UserId { get; set; }
        public int Score { get; set; }
        public string Review { get; set; }
    }
}
```

Endpoints

UserEndpoints.cs

```
using LW4.Task1_MiA.Models;
namespace LW4.Task1_MiA.Endpoints
{
    public static class UserEndpoints
    {
        public static void MapUserEndpoints(this WebApplication app)
        {
            var users = new List<User>
            {
                new User { Id = 1, Username = "john_doe", Email = "john@example.com"
},
                new User { Id = 2, Username = "jane_smith", Email =
"jane@example.com" },
```

```
                new User { Id = 3, Username = "alice_wonder", Email =
"alice@example.com" }
            };
            app.MapGet("/users", () => Results.Ok(users));
            app.MapGet("/users/{id:int}", (int id) =>
            {
                var user = users.FirstOrDefault(u => u.Id == id);
                return user is not null ? Results.Ok(user) : Results.NotFound();
            });

            app.MapPost("/users", (User newUser) =>
            {
                newUser.Id = users.Count + 1;
                users.Add(newUser);
                return Results.Created($"/users/{newUser.Id}", newUser);
            });

            app.MapPut("/users/{id:int}", (int id, User updatedUser) =>
            {
                var user = users.FirstOrDefault(u => u.Id == id);
                if (user is null) return Results.NotFound();
                user.Username = updatedUser.Username;
                user.Email = updatedUser.Email;
                return Results.Ok(user);
            });

            app.MapDelete("/users/{id:int}", (int id) =>
            {
                var user = users.FirstOrDefault(u => u.Id == id);
                if (user is null) return Results.NotFound();
                users.Remove(user);
                return Results.NoContent();
            });
        }
    }
}
```

RecipeEndpoints.cs

```
using LW4.Task1_MiA.Models;
namespace LW4.Task1_MiA.Endpoints
    public static class RecipeEndpoints
    {
        public static void MapRecipeEndpoints(this WebApplication app)
        {
            var recipes = new List<Recipe>
            {
                new Recipe { Id = 1, Name = "Pasta", Instructions = "Boil water, add
pasta" },
                new Recipe { Id = 2, Name = "Salad", Instructions = "Mix vegetables"
},
                new Recipe { Id = 3, Name = "Sandwich", Instructions = "Put
ingredients between bread slices" },
                new Recipe { Id = 4, Name = "Omelette", Instructions = "Beat eggs,
cook in a pan" },
                new Recipe { Id = 5, Name = "Soup", Instructions = "Boil broth, add
ingredients" },
                new Recipe { Id = 6, Name = "Steak", Instructions = "Season meat,
cook to desired doneness" },
                new Recipe { Id = 7, Name = "Pancakes", Instructions = "Mix batter,
cook on griddle" },
                new Recipe { Id = 8, Name = "Tacos", Instructions = "Fill tortillas
with ingredients" },
                new Recipe { Id = 9, Name = "Pizza", Instructions = "Prepare dough,
add toppings, bake" },
                new Recipe { Id = 10, Name = "Curry", Instructions = "Cook spices,
add meat/vegetables, simmer" },
```

```
                new Recipe { Id = 11, Name = "Grilled Cheese", Instructions =
"Butter bread, add cheese, grill" },
                new Recipe { Id = 12, Name = "Fruit Smoothie", Instructions = "Blend
fruits with yogurt or milk" }
                new Recipe { Id = 13, Name = "Roast Chicken", Instructions = "Season
chicken, roast in oven" },
            };
            app.MapGet("/recipes", () => Results.Ok(recipes));
            app.MapGet("/recipes/{id:int}", (int id) =>
            {
                var recipe = recipes.FirstOrDefault(r => r.Id == id);
                return recipe is not null ? Results.Ok(recipe) : Results.NotFound();
            });
            app.MapPost("/recipes", (Recipe newRecipe) =>
            {
                // Перевірка: чи заповнене поле Name
                if (string.IsNullOrWhiteSpace(newRecipe.Name))
                    return Results.BadRequest("Recipe name cannot be empty.");
                // Додавання нового рецепту
                newRecipe.Id = recipes.Count + 1;
                recipes.Add(newRecipe);
                return Results.Created($"/recipes/{newRecipe.Id}", newRecipe);
            });
            app.MapPut("/recipes/{id:int}", (int id, Recipe updatedRecipe) =>
            {
                var recipe = recipes.FirstOrDefault(r => r.Id == id);
                if (recipe is null) return Results.NotFound();
                recipe.Name = updatedRecipe.Name;
                recipe.Instructions = updatedRecipe.Instructions;
                return Results.Ok(recipe);
            });
            app.MapDelete("/recipes/{id:int}", (int id) =>
            {
                var recipe = recipes.FirstOrDefault(r => r.Id == id);
                if (recipe is null) return Results.NotFound();
                recipes.Remove(recipe);
                return Results.NoContent();
            });
        }
    }
}
```

RatingEndpoints.cs

```
using LW4.Task1_MiA.Models;
namespace LW4.Task1_MiA.Endpoints {
    public static class RatingEndpoints
    {
        public static void MapRatingEndpoints(this WebApplication app)
        {
            var ratings = new List<Rating>
            {
                new Rating { Id = 1, RecipeId = 1, UserId = 1, Score = 5, Review =
"Delicious!" },
                new Rating { Id = 2, RecipeId = 2, UserId = 2, Score = 4, Review =
"Tasty, but could use more dressing." },
                new Rating { Id = 3, RecipeId = 3, UserId = 3, Score = 3, Review =
"Average." },
                new Rating { Id = 4, RecipeId = 1, UserId = 2, Score = 4, Review =
"Pretty good!" },
                new Rating { Id = 5, RecipeId = 2, UserId = 3, Score = 5, Review =
"Loved it!" },
                new Rating { Id = 6, RecipeId = 3, UserId = 1, Score = 2, Review =
"Not my favorite." },
                new Rating { Id = 7, RecipeId = 1, UserId = 3, Score = 5, Review =
"Absolutely fantastic!" },
            };
```

```
            app.MapGet("/ratings", () => Results.Ok(ratings));
            app.MapGet("/ratings/{id:int}", (int id) =>
            {
                var rating = ratings.FirstOrDefault(r => r.Id == id);
                return rating is not null ? Results.Ok(rating) : Results.NotFound();
            });
            app.MapPost("/ratings", (Rating newRating) =>
            {
                newRating.Id = ratings.Count + 1;
                ratings.Add(newRating);
                return Results.Created($"/ratings/{newRating.Id}", newRating);
            });
            app.MapPut("/ratings/{id:int}", (int id, Rating updatedRating) =>
            {
                var rating = ratings.FirstOrDefault(r => r.Id == id);
                if (rating is null) return Results.NotFound();
                rating.Score = updatedRating.Score;
                rating.Review = updatedRating.Review;
                return Results.Ok(rating);
            });
            app.MapDelete("/ratings/{id:int}", (int id) =>
            {
                var rating = ratings.FirstOrDefault(r => r.Id == id);
                if (rating is null) return Results.NotFound();
                ratings.Remove(rating);
                return Results.NoContent();
            });
        }
    }
}
```

Program.cs

```
using LW4.Task1_MiA.Endpoints;

var builder = WebApplication.CreateBuilder(args);

// Swagger
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
var app = builder.Build();
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(opt =>
    {
        opt.SwaggerEndpoint("/swagger/v1/swagger.json", "Recipes API v1");
        opt.RoutePrefix = "swagger";
    });
}
app.MapRecipeEndpoints();
app.MapUserEndpoints();
app.MapRatingEndpoints();

app.Run();
```

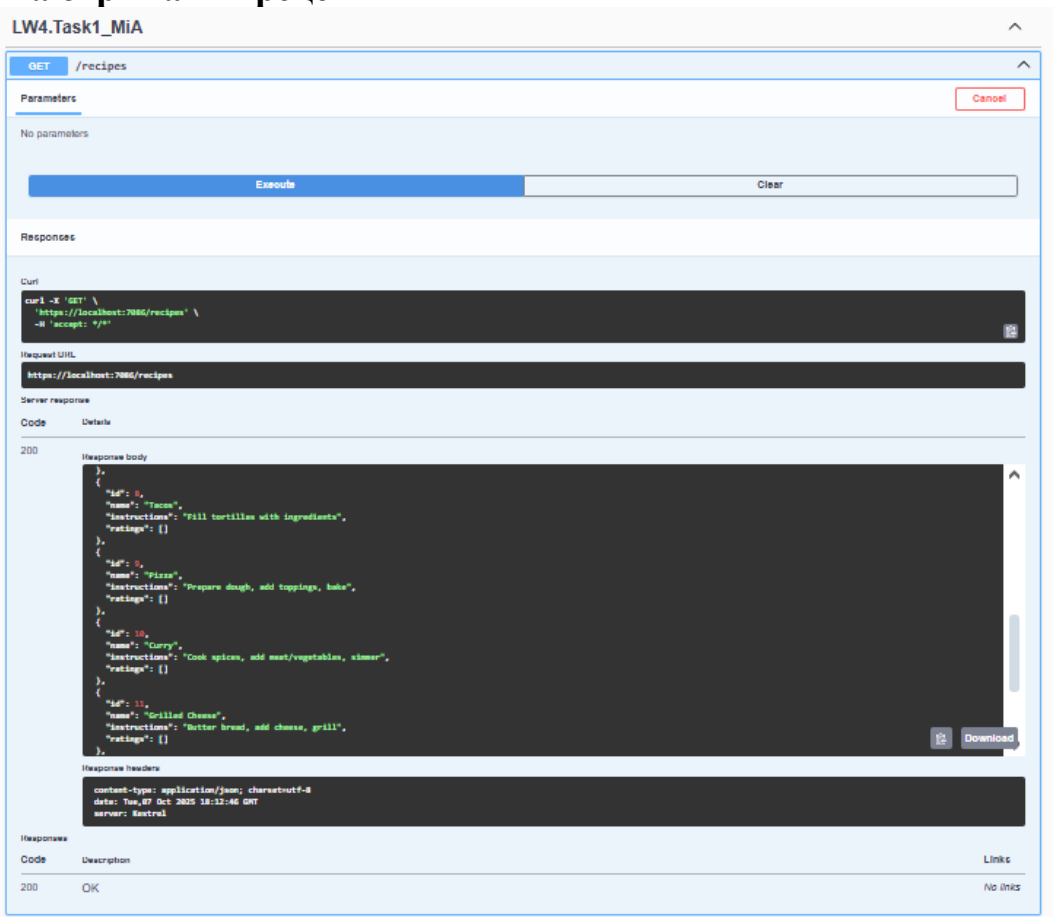
Результат роботи:

```
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7086
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5128
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\Mykha\source\repos\LW4.Task1_MiA\LW4.Task1_MiA
```

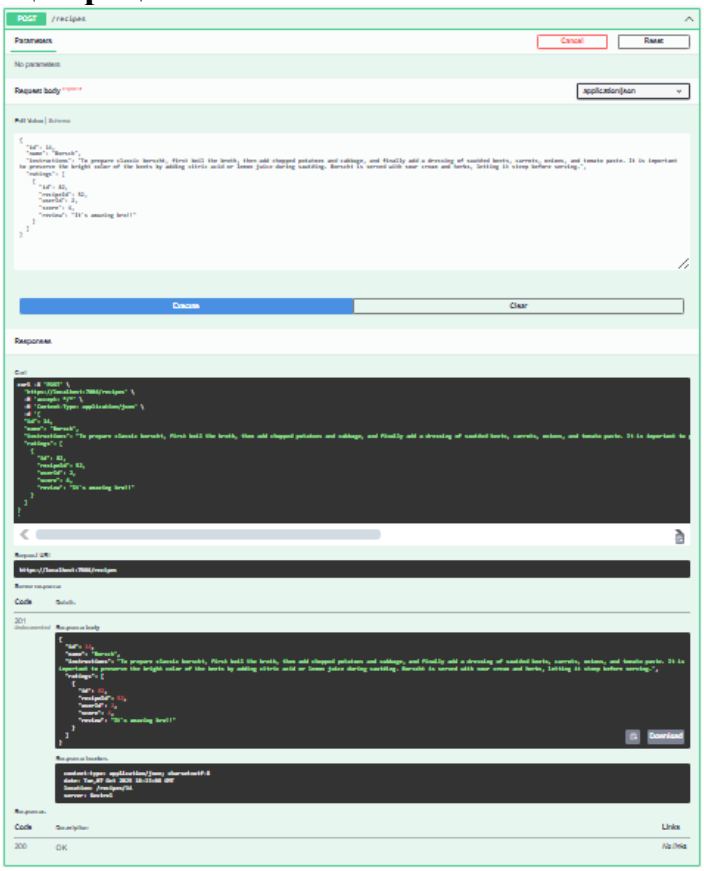
Програма показує те, що нам доступна вся інформація по посиланню.

Запити

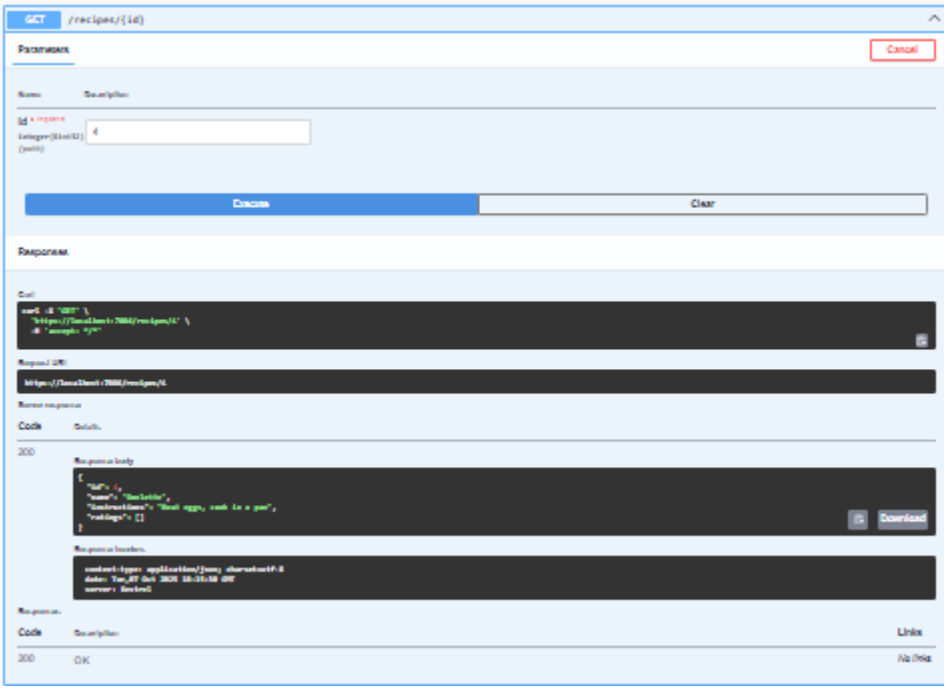
Запит на отримання рецептів



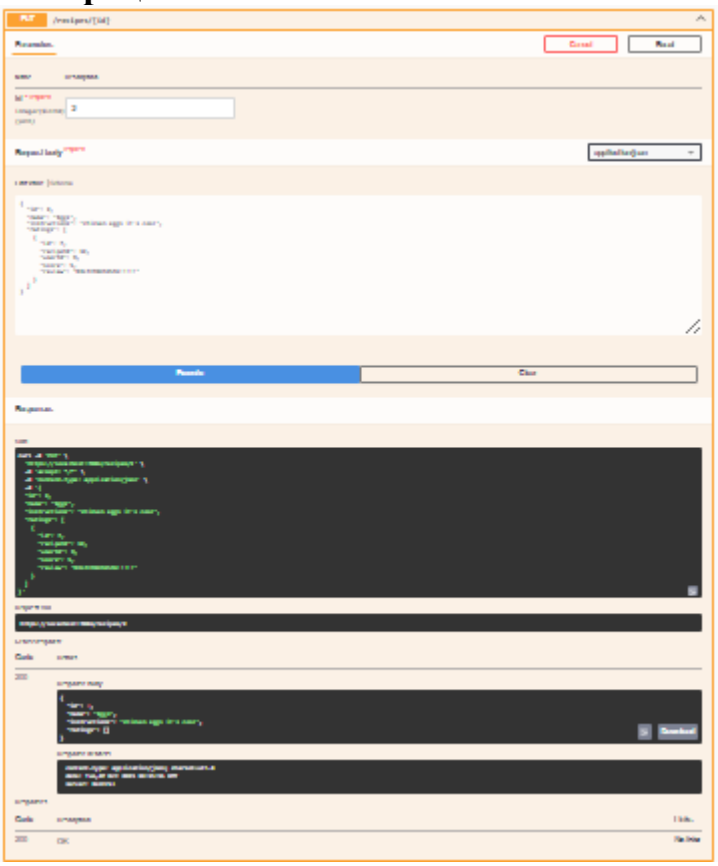
Запит на публікацію рецепта



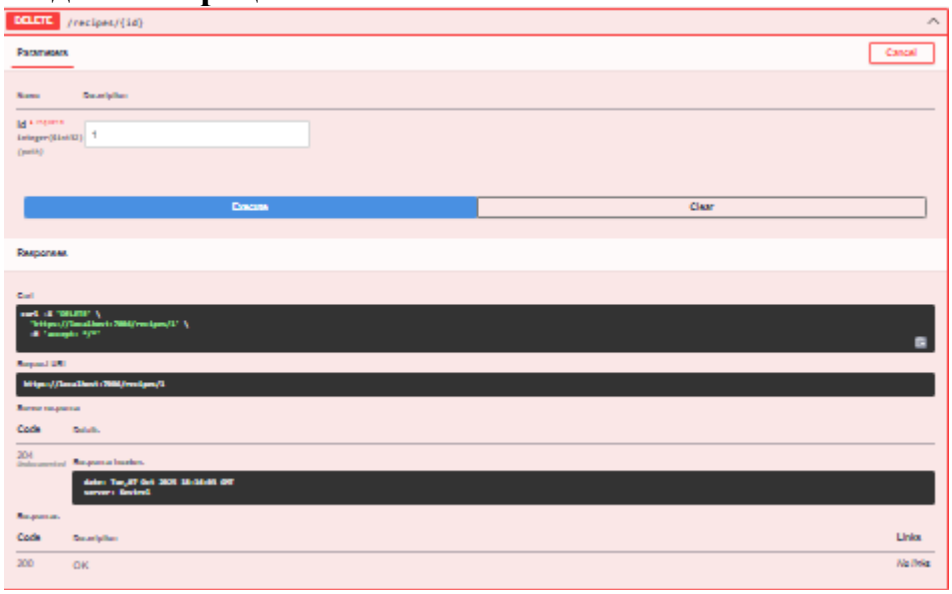
Запит на отримання конкретного рецепта



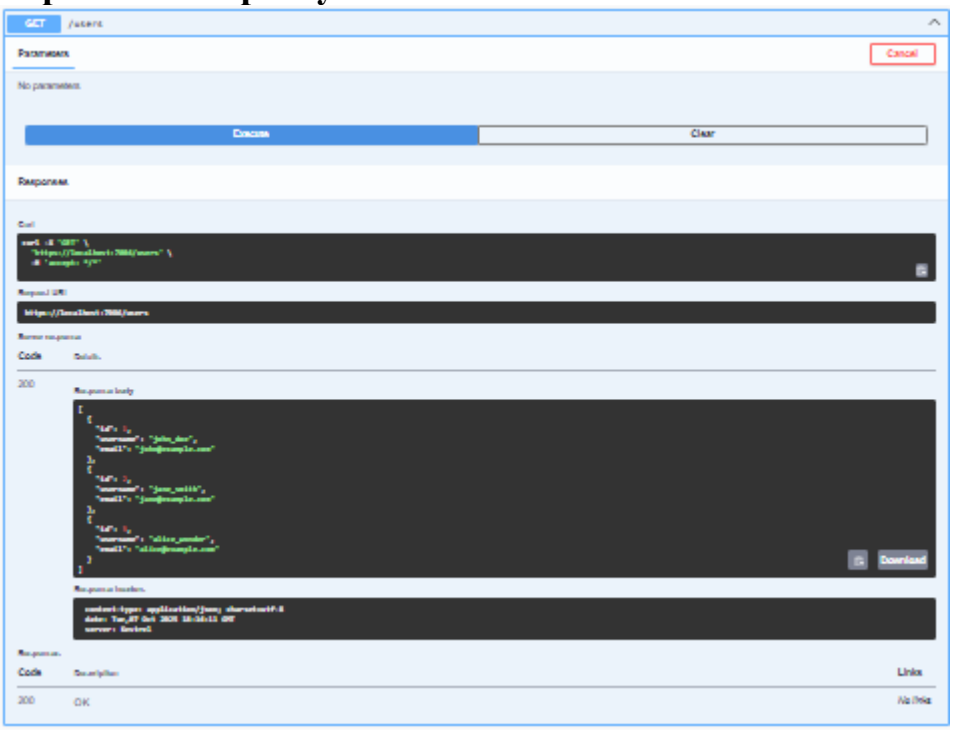
Запит на оновлення рецепта



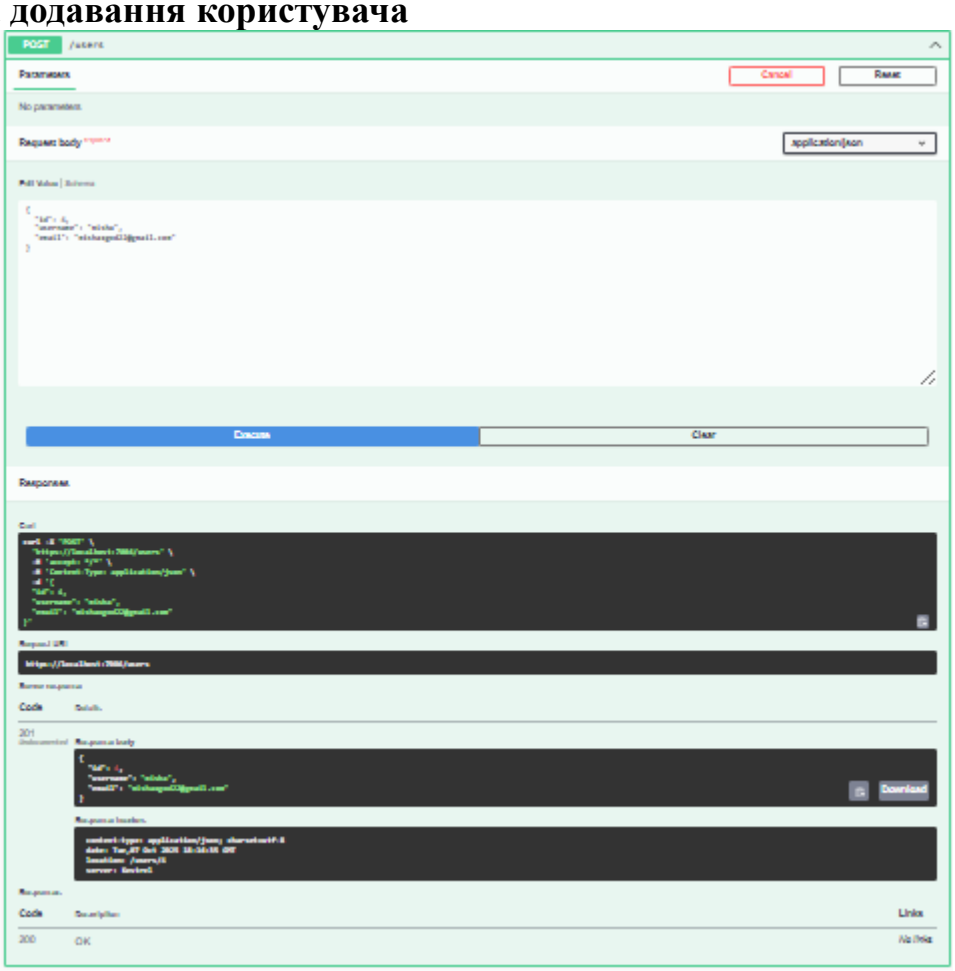
Запит на видалення рецепта



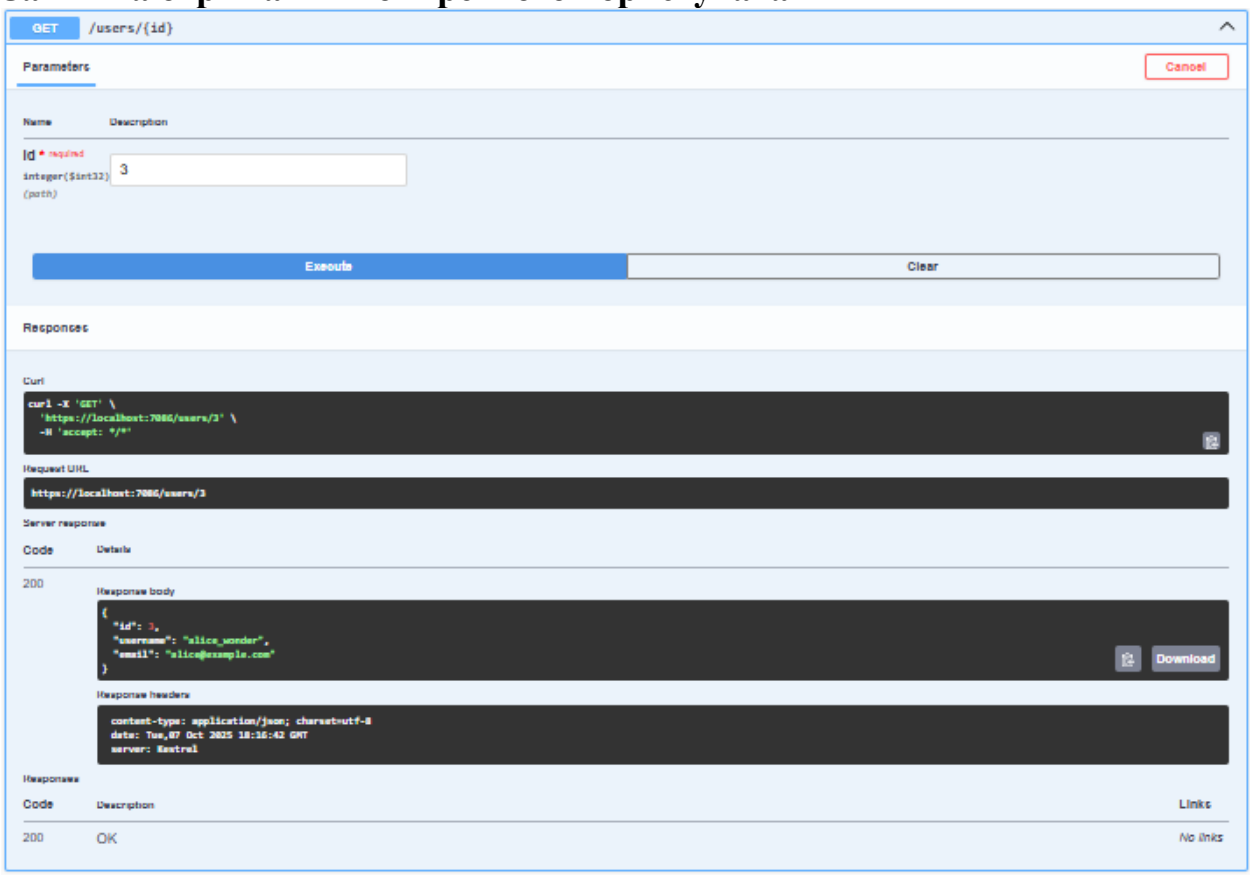
Запит на отримання користувачів



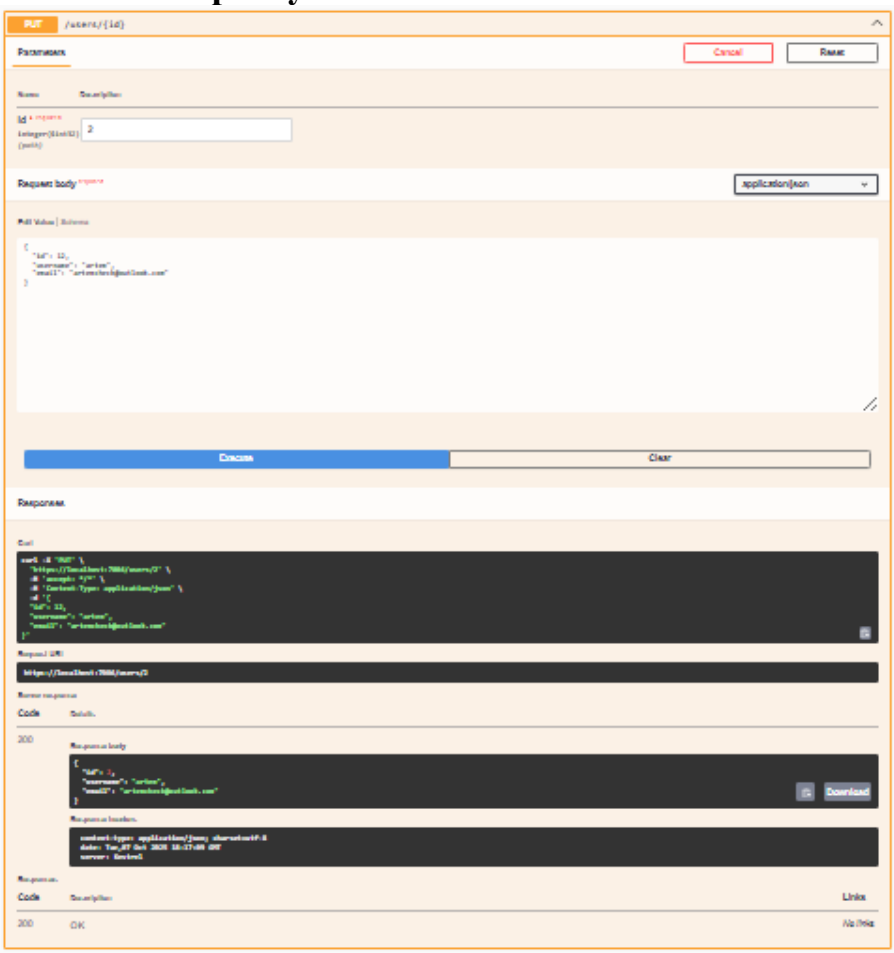
Запит на додавання користувача



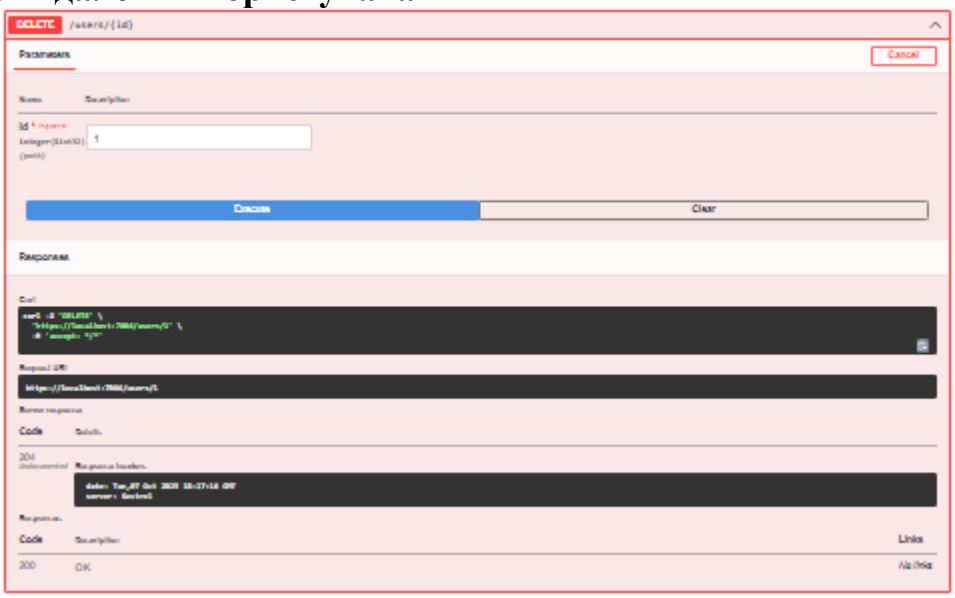
Запит на отримання конкретного користувача



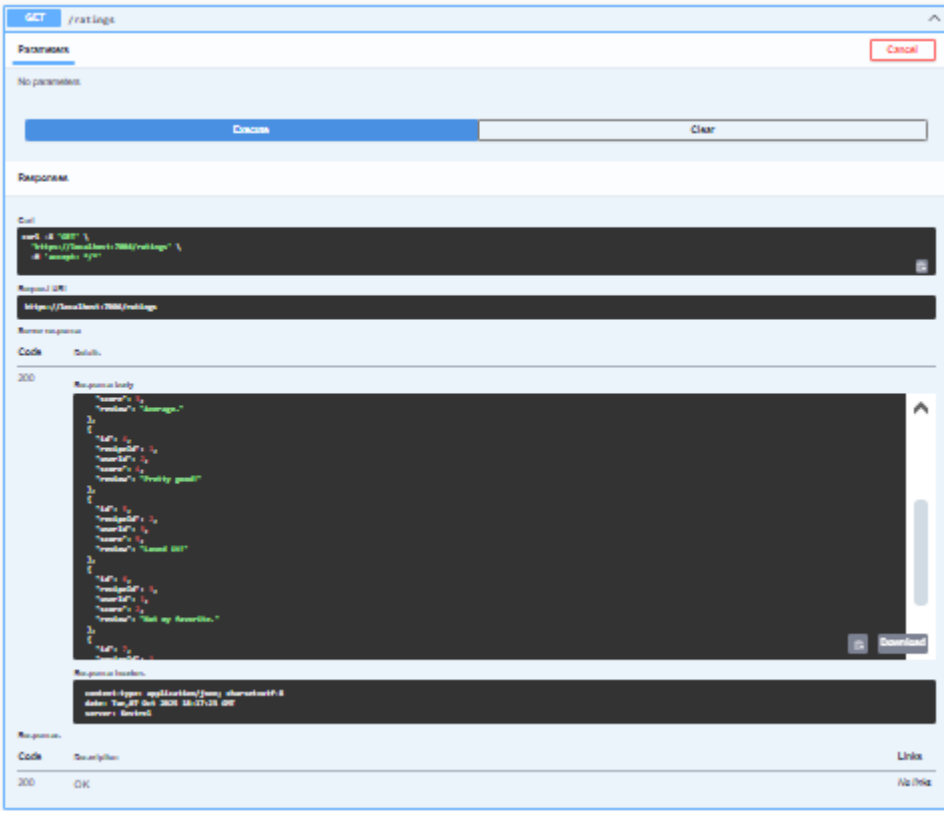
Запит на оновлення користувача



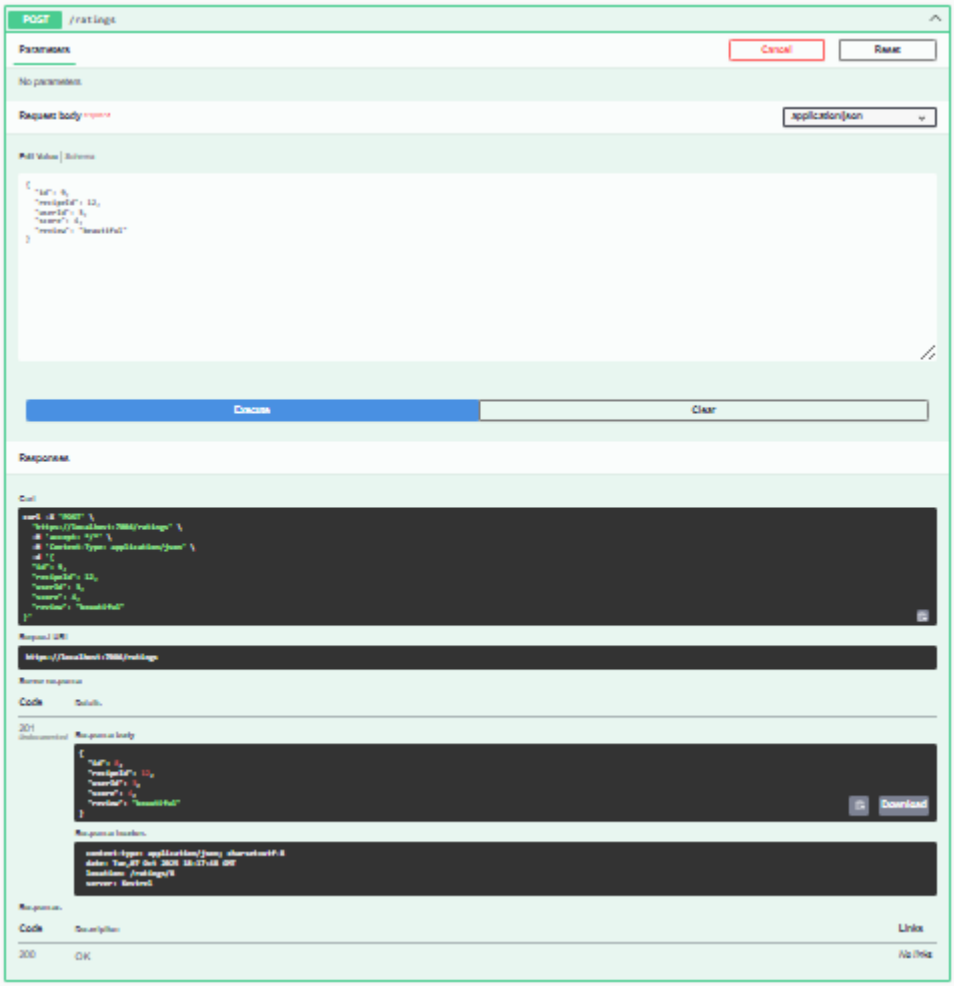
Запит на видалення користувача



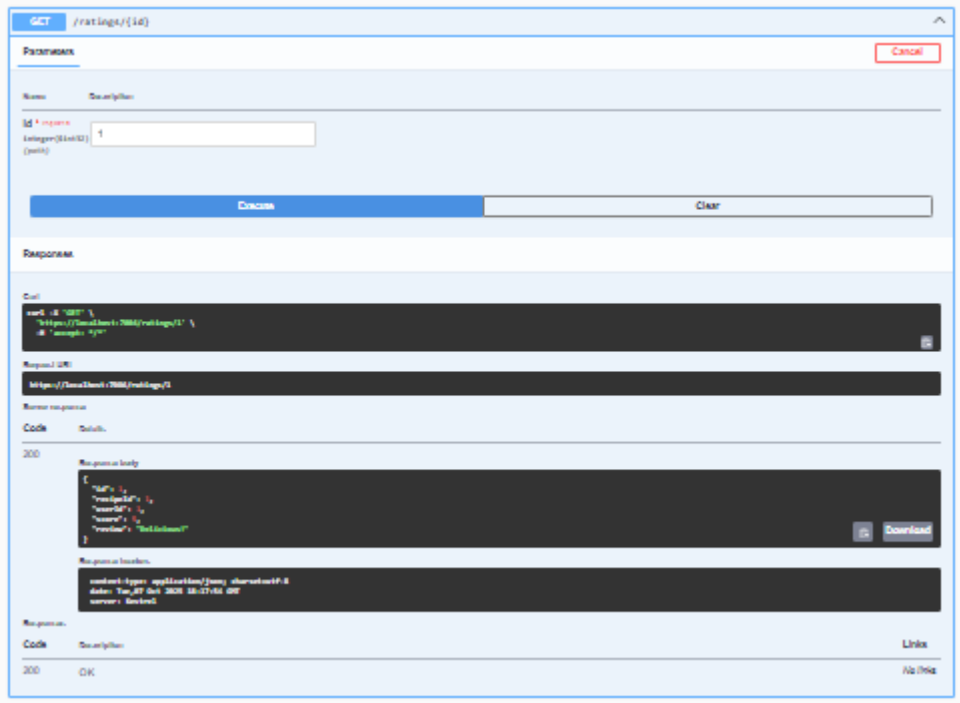
Запит на отримання оцінок



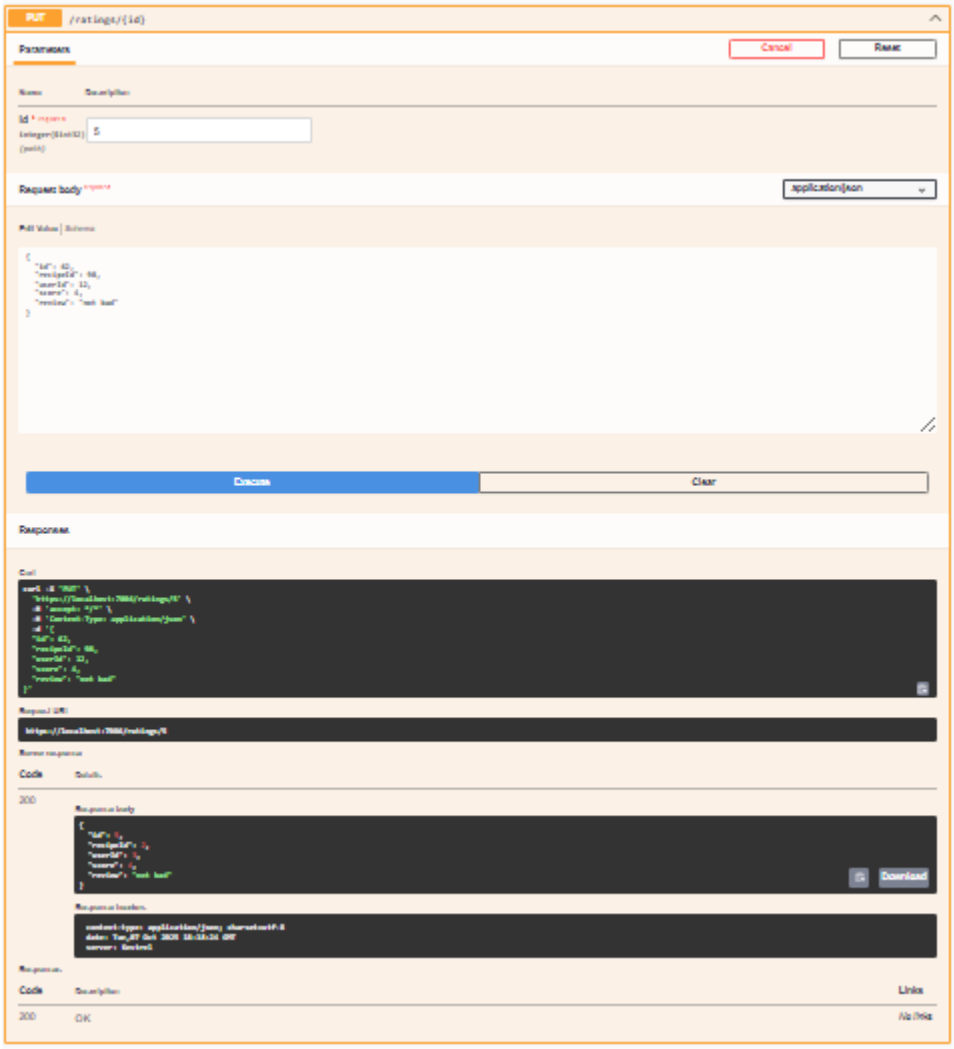
Запит на додавання оцінки



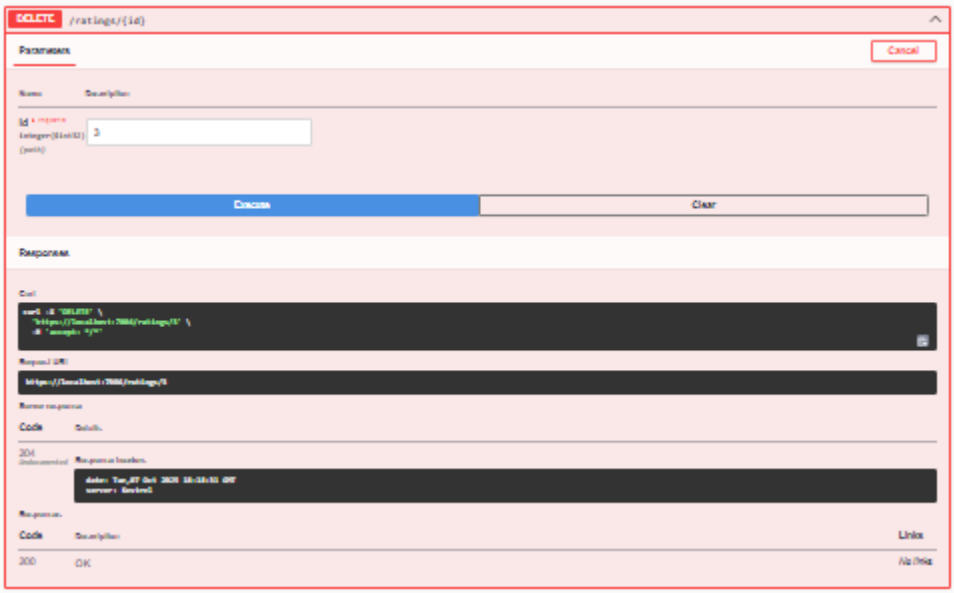
Запит на отримання конкретної оцінки



Запит на оновлення оцінки



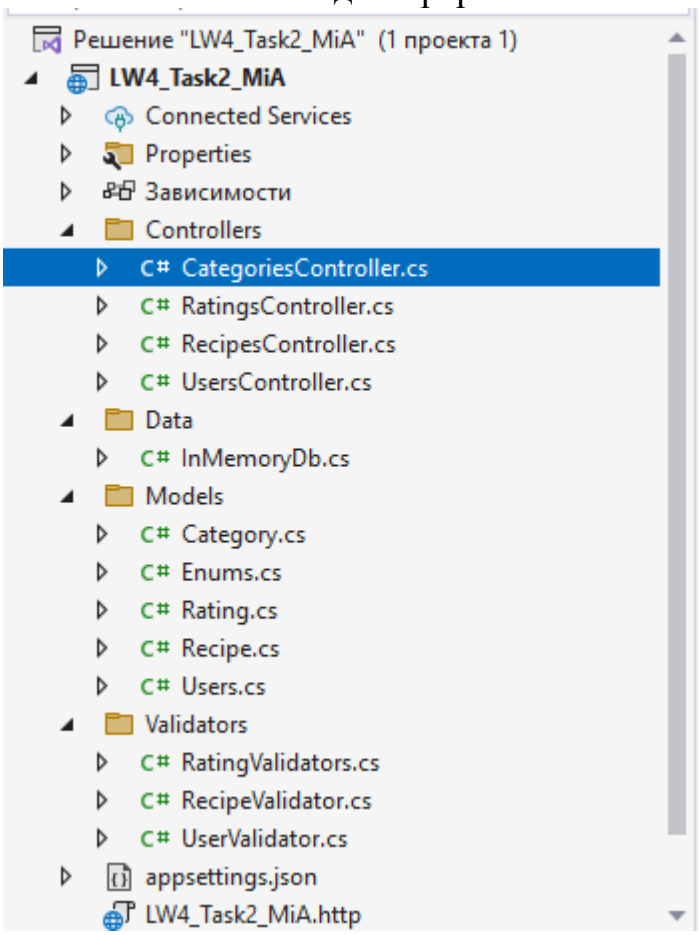
Запит на видалення оцінки



Завдання №2

Створимо проєкт з назвою LW4_Task2_MiA.

Так виглядає ієрархія



Тепер перейдемо до написання коду

Controllers

CategoriesControllers.cs:

```
using LW4_Task2_MiA.Data;
using LW4_Task2_MiA.Models;
using Microsoft.AspNetCore.Mvc;

namespace LW4_Task2_MiA.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class CategoriesController : ControllerBase
    {
        private readonly InMemoryDb _db;
        public CategoriesController(InMemoryDb db) => _db = db;

        [HttpGet]
        public ActionResult<IEnumerable<Category>> GetAll() => Ok(_db.Categories);

        [HttpGet("{id:int}")]
        public ActionResult<Category> GetById(int id)
        {
            var item = _db.Categories.FirstOrDefault(x => x.Id == id);
            return item is null ? NotFound() : Ok(item);
        }

        [HttpPost]
        public ActionResult<Category> Create([FromBody] Category model)
```

```
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);
            model.Id = _db.NextCategoryId();
            _db.Categories.Add(model);
            return CreatedAtAction(nameof(GetById), new { id = model.Id }, model);
        }

        [HttpPut("{id:int}")]
        public ActionResult<Category> Update(int id, [FromBody] Category model)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);
            var existing = _db.Categories.FirstOrDefault(x => x.Id == id);
            if (existing is null) return NotFound();
            existing.Name = model.Name;
            existing.Type = model.Type;
            return Ok(existing);
        }

        [HttpDelete("{id:int}")]
        public IActionResult Delete(int id)
        {
            var existing = _db.Categories.FirstOrDefault(x => x.Id == id);
            if (existing is null) return NotFound();
            _db.Categories.Remove(existing);
            return Ok();
        }
    }
}
```

RatingsControllers.cs:

```
using LW4_Task2_MiA.Data;
using LW4_Task2_MiA.Models;
using Microsoft.AspNetCore.Mvc;

namespace LW4_Task2_MiA.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class RatingsController : ControllerBase
    {
        private readonly InMemoryDb _db;
        public RatingsController(InMemoryDb db) => _db = db;

        [HttpGet]
        public ActionResult<IEnumerable<Rating>> GetAll() => Ok(_db.Ratings);

        [HttpGet("{id:int}")]
        public ActionResult<Rating> GetById(int id)
        {
            var r = _db.Ratings.FirstOrDefault(x => x.Id == id);
            return r is null ? NotFound() : Ok(r);
        }

        [HttpPost("/api/recipes/{recipeId:int}/ratings")]
        public ActionResult<Rating> CreateForRecipe(int recipeId, [FromBody] Rating
model)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);

            var recipe = _db.Recipes.FirstOrDefault(r => r.Id == recipeId);
            if (recipe is null) return NotFound("Recipe не знайдено.");

            // Перевіряємо існування користувача
            var user = _db.Users.FirstOrDefault(u => u.Id == model.UserId);
            if (user is null) return BadRequest("UserId не існує.");
```

```
            // Форсуємо правильні зв'язки
            var rating = new Rating
            {
                Id = _db.NextRatingId(),
                RecipeId = recipeId,           // ігноруємо model.RecipeId
                UserId = user.Id,
                Value = model.Value,
                Comment = model.Comment
            };

            _db.Ratings.Add(rating);
            return CreatedAtAction(nameof(GetById), new { id = rating.Id }, rating);
        }

        [HttpPut("{id:int}")]
        public ActionResult<Rating> Update(int id, [FromBody] Rating model)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);
            if (model.Id != 0 && model.Id != id)
                return BadRequest("Змінювати Id рейтингу заборонено.");

            var r = _db.Ratings.FirstOrDefault(x => x.Id == id);
            if (r is null) return NotFound();

            if (_db.Recipes.All(rc => rc.Id != model.RecipeId))
                return BadRequest("RecipeId не існує.");
            if (_db.Users.All(u => u.Id != model.UserId))
                return BadRequest("UserId не існує.");

            r.RecipeId = model.RecipeId;
            r.UserId = model.UserId;
            r.Value = model.Value;
            r.Comment = model.Comment;
            return Ok(r);
        }


        [HttpDelete("{id:int}")]
        public IActionResult Delete(int id)
        {
            var r = _db.Ratings.FirstOrDefault(x => x.Id == id);
            if (r is null) return NotFound();
            _db.Ratings.Remove(r);
            return Ok();
        }
    }
}
```

RecipesControllers.cs:

```
using LW4_Task2_MiA.Data;
using LW4_Task2_MiA.Models;
using Microsoft.AspNetCore.Mvc;

namespace LW4_Task2_MiA.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class RecipesController : ControllerBase
    {
        private readonly InMemoryDb _db;
        public RecipesController(InMemoryDb db) => _db = db;

        [HttpGet]
        public ActionResult<IEnumerable<Recipe>> GetAll() => Ok(_db.Recipes);

        [HttpGet("{id:int}")]
```

```
        public ActionResult<Recipe> GetById(int id)
        {
            var r = _db.Recipes.FirstOrDefault(x => x.Id == id);
            return r is null ? NotFound() : Ok(r);
        }

        [HttpPost]
        public ActionResult<Recipe> Create([FromBody] Recipe model)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);
            if (_db.Categories.All(c => c.Id != model.CategoryId)) return
BadRequest("CategoryId не існує.");
            if (_db.Users.All(u => u.Id != model.AuthorUserId)) return
BadRequest("AuthorUserId не існує.");

            model.Id = _db.NextRecipeId();
            _db.Recipes.Add(model);
            return CreatedAtAction(nameof(GetById), new { id = model.Id }, model);
        }

        [HttpPut("{id:int}")]
        public ActionResult<Recipe> Update(int id, [FromBody] Recipe model)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);
            var r = _db.Recipes.FirstOrDefault(x => x.Id == id);
            if (r is null) return NotFound();

            if (_db.Categories.All(c => c.Id != model.CategoryId)) return
BadRequest("CategoryId не існує.");
            if (_db.Users.All(u => u.Id != model.AuthorUserId)) return
BadRequest("AuthorUserId не існує.");

            r.Title = model.Title;
            r.Slug = model.Slug;
            r.Description = model.Description;
            r.Difficulty = model.Difficulty;
            r.CategoryId = model.CategoryId;
            r.AuthorUserId = model.AuthorUserId;
            return Ok(r);
        }

        [HttpDelete("{id:int}")]
        public IActionResult Delete(int id)
        {
            var r = _db.Recipes.FirstOrDefault(x => x.Id == id);
            if (r is null) return NotFound();
            _db.Recipes.Remove(r);
            _db.Ratings.RemoveAll(rt => rt.RecipeId == id); // каскадне очищення
рейтингів
            return Ok();
        }
    }
}
```

UsersControllers.cs:

```
using LW4_Task2_MiA.Data;
using LW4_Task2_MiA.Models;
using Microsoft.AspNetCore.Mvc;

namespace LW4_Task2_MiA.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    public class UsersController : ControllerBase
    {
        private readonly InMemoryDb _db;
```

```
        public UsersController(InMemoryDb db) => _db = db;

        [HttpGet]
        public ActionResult<IEnumerable<User>> GetAll() => Ok(_db.Users);

        [HttpGet("{id:int}")]
        public ActionResult<User> GetById(int id)
        {
            var u = _db.Users.FirstOrDefault(x => x.Id == id);
            return u is null ? NotFound() : Ok(u);
        }

        [HttpPost]
        public ActionResult<User> Create([FromBody] User model)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);
            model.Id = _db.NextUserId();
            _db.Users.Add(model);
            return CreatedAtAction(nameof(GetById), new { id = model.Id }, model);
        }

        [HttpPut("{id:int}")]
        public ActionResult<User> Update(int id, [FromBody] User model)
        {
            if (!ModelState.IsValid) return BadRequest(ModelState);
            var u = _db.Users.FirstOrDefault(x => x.Id == id);
            if (u is null) return NotFound();
            u.DisplayName = model.DisplayName;
            u.Email = model.Email;
            u.Role = model.Role;
            return Ok(u);
        }

        [HttpDelete("{id:int}")]
        public IActionResult Delete(int id)
        {
            var u = _db.Users.FirstOrDefault(x => x.Id == id);
            if (u is null) return NotFound();
            _db.Users.Remove(u);
            return Ok();
        }
    }
}
```

Data

InMemoryDb.cs:

```
using LW4_Task2_MiA.Models;

namespace LW4_Task2_MiA.Data
{
    public class InMemoryDb
    {
        public List<Category> Categories { get; } = new();
        public List<User> Users { get; } = new();
        public List<Recipe> Recipes { get; } = new();
        public List<Rating> Ratings { get; } = new();

        private int _catId = 1, _userId = 1, _recipeId = 1, _ratingId = 1;

        public InMemoryDb()
        {
            // Початкові дані
            Categories.AddRange(new[]
            {
                new Category{ Id = _catId++, Name="Десерти",
Type=CategoryType.Dessert },
```

```
                new Category{ Id = _catId++, Name="Сніданки",
Type=CategoryType.Breakfast },
                new Category{ Id = _catId++, Name="Обіди", Type=CategoryType.Lunch
},
                new Category{ Id = _catId++, Name="Вечері", Type=CategoryType.Dinner
},
                new Category{ Id = _catId++, Name="Інше", Type=CategoryType.Unknown
},
            });
            Users.Add(new User { Id = _userId++, DisplayName = "Alice", Email =
"alice@example.com", Role = UserRole.Regular });
            Users.Add(new User { Id = _userId++, DisplayName = "Bob", Email =
"bob@gmail.com", Role = UserRole.Admin });
            Users.Add(new User { Id = _userId++, DisplayName = "Charlie", Email =
"charlie@yahoo.com", Role = UserRole.Regular });
            Users.Add(new User { Id = _userId++, DisplayName = "Diana", Email =
"diana@duckduckgo.com", Role = UserRole.Regular });
            Recipes.Add(new Recipe
            {
                Id = _recipeId++,
                Title = "Яблучний пиріг",
                Slug = "yabluchnyi-pyrig",
                Description = "Класичний рецепт.",
                Difficulty = RecipeDifficulty.Medium,
                CategoryId = 1,
                AuthorUserId = 1
            });
            Recipes.Add(new Recipe
            {
                Id = _recipeId++,
                Title = "Омлет",
                Slug = "omlet",
                Description = "Швидкий сніданок.",
                Difficulty = RecipeDifficulty.Easy,
                CategoryId = 2,
                AuthorUserId = 1
            });
            Recipes.Add(new Recipe
            {
                Id = _recipeId++,
                Title = "Млинці",
                Slug = "mlynci",
                Description = "Тонкі млинці з начинкою.",
                Difficulty = RecipeDifficulty.Easy,
                CategoryId = 2,
                AuthorUserId = 1
            });
            Recipes.Add(new Recipe
            {
                Id = _recipeId++,
                Title = "Тірамісу",
                Slug = "tiramisu",
                Description = "Італійський десерт.",
                Difficulty = RecipeDifficulty.Hard,
                CategoryId = 1,
                AuthorUserId = 1
            });
            Ratings.Add(new Rating { Id = _ratingId++, RecipeId = 1, UserId = 1,
Value = 5, Comment = "Смачно!" });
            Ratings.Add(new Rating { Id = _ratingId++, RecipeId = 1, UserId = 2,
Value = 4, Comment = "Добре, але можна краще." });
            Ratings.Add(new Rating { Id = _ratingId++, RecipeId = 2, UserId = 3,
Value = 5, Comment = "Просто і смачно." });
            Ratings.Add(new Rating { Id = _ratingId++, RecipeId = 3, UserId = 4,
Value = 3, Comment = "Млинці вийшли трохи товсті." });
```

```
            Ratings.Add(new Rating { RecipeId = _ratingId++, UserId = 2, Value = 5,
Comment = "Обожнюю тірамісу!" });
            Ratings.Add(new Rating { Id = _ratingId++, RecipeId = 4, UserId = 3,
Value = 4, Comment = "Дуже смачно, але складно готувати." });
            Ratings.Add(new Rating { Id = _ratingId++, RecipeId = 2, UserId = 4,
Value = 4, Comment = "Хороший рецепт для швидкого сніданку." });
        }

        public int NextCategoryId() => _catId++;
        public int NextUserId() => _userId++;
        public int NextRecipeId() => _recipeId++;
        public int NextRatingId() => _ratingId++;
    }
}
```

Models

Category.cs:

```
using System.ComponentModel.DataAnnotations;

namespace LW4_Task2_MiA.Models
{
    public class Category
    {
        public int Id { get; set; }

        [Required, StringLength(40, MinimumLength = 2)]
        public string Name { get; set; } = string.Empty;

        [Required]
        public CategoryType Type { get; set; } = CategoryType.Unknown;
    }
}
```

Enums.cs:

```
namespace LW4_Task2_MiA.Models
{
        public enum RecipeDifficulty { Easy = 1, Medium = 2, Hard = 3 }
        public enum UserRole { Regular = 1, Moderator = 2, Admin = 3 }
        public enum CategoryType { Unknown = 0, Breakfast = 1, Lunch = 2, Dinner =
3, Dessert = 4 }
}
```

Rating.cs:

```
namespace LW4_Task2_MiA.Models
{
    public class Rating
    {
        public int Id { get; set; }
        public int RecipeId { get; set; }
        public int UserId { get; set; }
        public int Value { get; set; }
        public string? Comment { get; set; }
    }
}
```

Recipe.cs:

```
namespace LW4_Task2_MiA.Models
{
    public class Recipe
    {
        public int Id { get; set; }
        public string Title { get; set; } = string.Empty;
        public string Slug { get; set; } = string.Empty;
        public string Description { get; set; } = string.Empty;
        public RecipeDifficulty Difficulty { get; set; } = RecipeDifficulty.Easy;
        public int CategoryId { get; set; }
        public int AuthorUserId { get; set; }
```

```
    }
}
```

Users.cs:

```
namespace LW4_Task2_MiA.Models
{
    public class User
    {
        public int Id { get; set; }
        public string DisplayName { get; set; } = string.Empty;
        public string Email { get; set; } = string.Empty;
        public UserRole Role { get; set; } = UserRole.Regular;
    }
}
```

Validators

RatingValidator.cs:

```
using LW4_Task2_MiA.Models;
using FluentValidation;
namespace LW4_Task2_MiA.Validators
{
    public class RatingValidator : AbstractValidator<Rating>
    {
        public RatingValidator()
        {
            RuleFor(x => x.RecipeId).GreaterThan(0);
            RuleFor(x => x.UserId).GreaterThan(0);
            RuleFor(x => x.Value).InclusiveBetween(1, 5);
            RuleFor(x => x.Comment).MaximumLength(500);
        }
    }
}
```

RecipeValidator.cs:

```
using LW4_Task2_MiA.Models;
using FluentValidation;
using System.Text.RegularExpressions;
namespace LW4_Task2_MiA.Validators
{
    public class RecipeValidator : AbstractValidator<Recipe>
    {
        public RecipeValidator()
        {
            RuleFor(x => x.Title).NotEmpty().MinimumLength(3).MaximumLength(80);

            RuleFor(x => x.Slug).NotEmpty().Must(s => Regex.IsMatch(s, "^[a-z0-9-
]+$")).WithMessage("Slug має містити лише малі латиницю, цифри та тире.");

            RuleFor(x =>
x.Description).NotEmpty().MinimumLength(10).MaximumLength(1000);

            RuleFor(x => x.Difficulty).IsInEnum();

            RuleFor(x => x.CategoryId).GreaterThan(0);
            RuleFor(x => x.AuthorUserId).GreaterThan(0);
        }
    }
}
```

UserValidator.cs:

```
using LW4_Task2_MiA.Models;
using FluentValidation;
using System.Text.RegularExpressions;
namespace LW4_Task2_MiA.Validators
{
    public class UserValidator : AbstractValidator<User>
    {
```

```
        private const string EmailRegex =
            @"^[A-Za-z0-9._%+\-]+@[A-Za-z0-9.\-]+\.[A-Za-z]{2,}$";

        public UserValidator()
        {
            RuleFor(x =>
x.DisplayName).NotEmpty().MinimumLength(2).MaximumLength(50);

            RuleFor(x => x.Email).NotEmpty().Must(v => Regex.IsMatch(v,
EmailRegex)).WithMessage("Невірний формат email.");

            RuleFor(x => x.Role).IsInEnum();
        }
    }
}
```

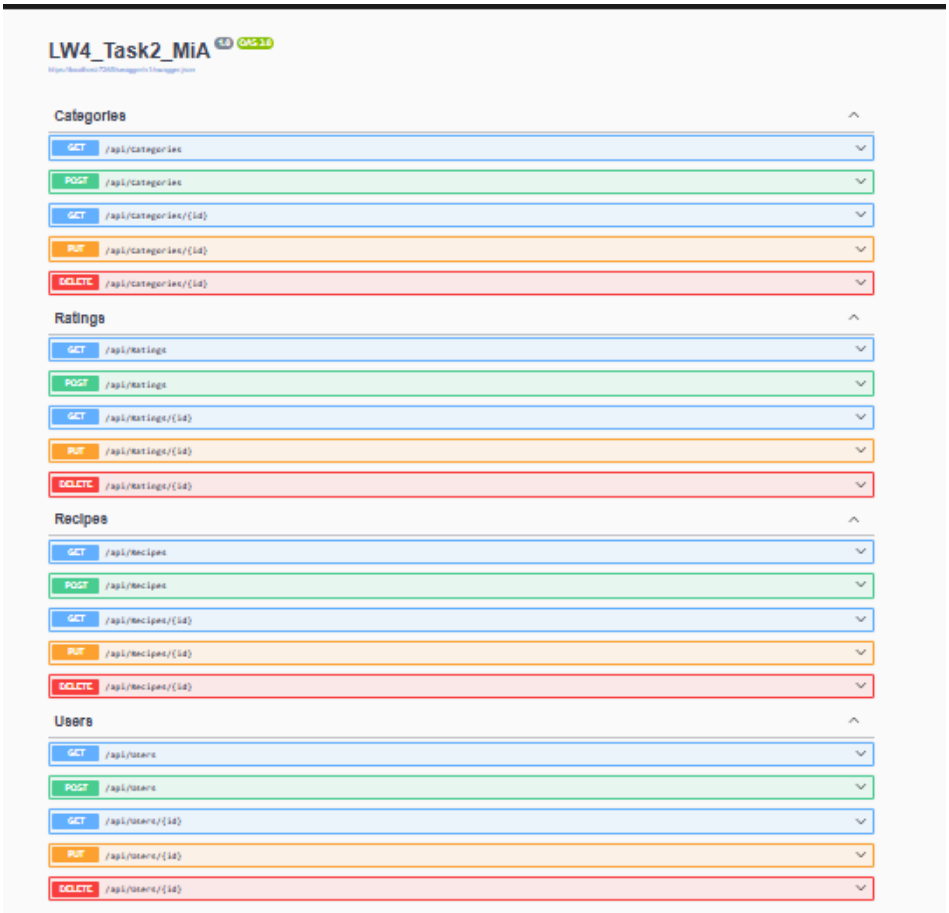
Program.cs:

```
using FluentValidation;
using FluentValidation.AspNetCore;
using LW4_Task2_MiA.Data;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddSingleton<InMemoryDb>();
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

// FluentValidation
builder.Services.AddFluentValidationAutoValidation();
builder.Services.AddValidatorsFromAssemblyContaining<LW4_Task2_MiA.Validators.Recipe
Validator>();

var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();
app.MapControllers();

app.Run();
```

Результат програми

```
C:\Users\Mykha\source\repos'
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7265
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5141
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\Mykha\source\repos\LW4_Task2_MiA\LW4_Task2_MiA
```

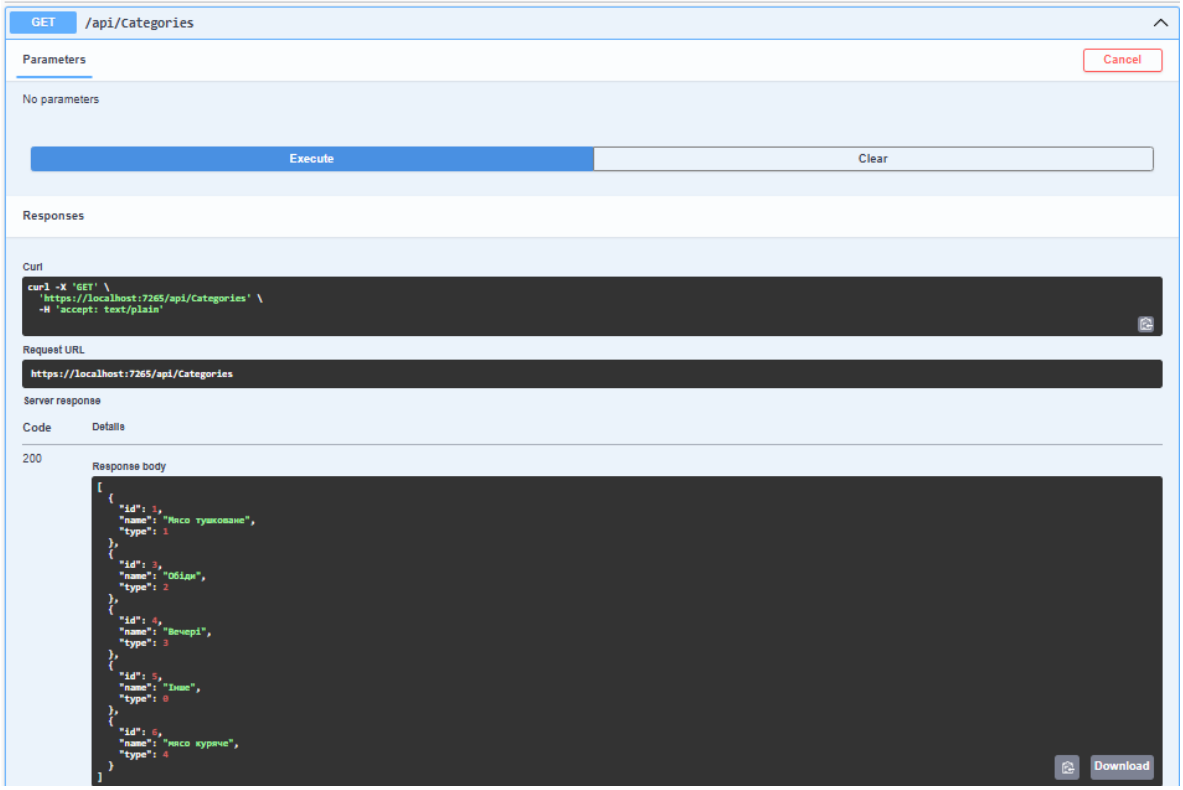
Програма працює. Переходимо до Swagger.

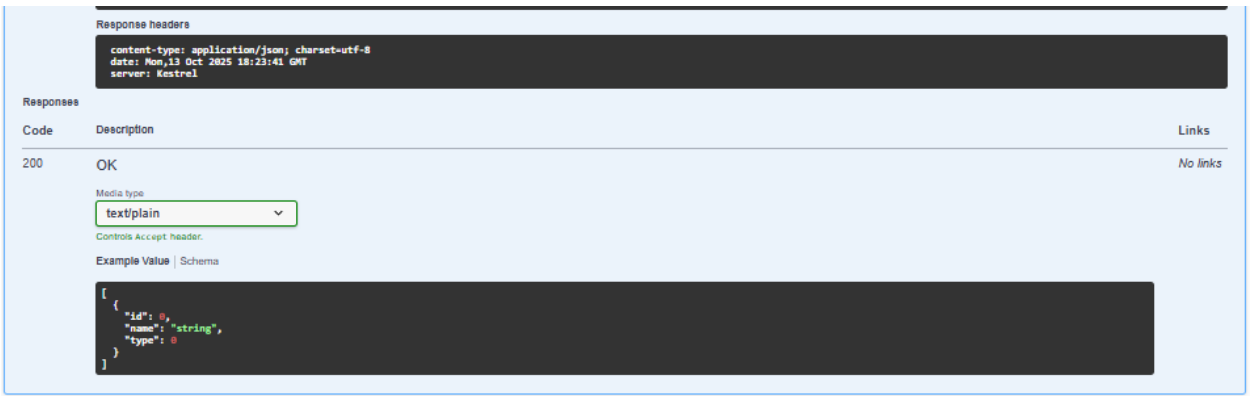


В програмі є 4 контролера: Categories, Ratings, Recipes, Users.
Для кожного з них є CRUD (C - Create, R - Read, U - Update, D – Delete).

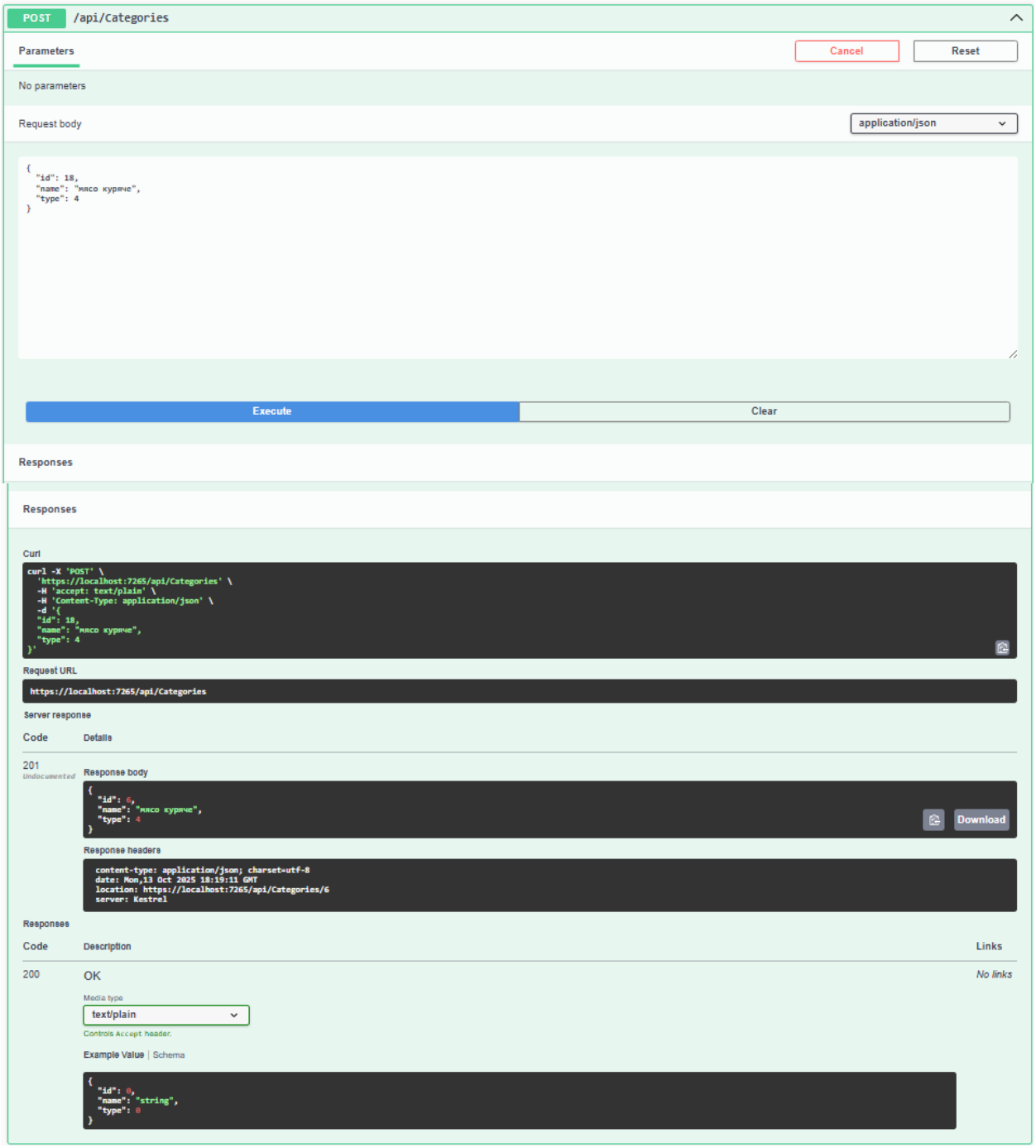
Почнемо з Categories

Виконаємо запит на категорії.

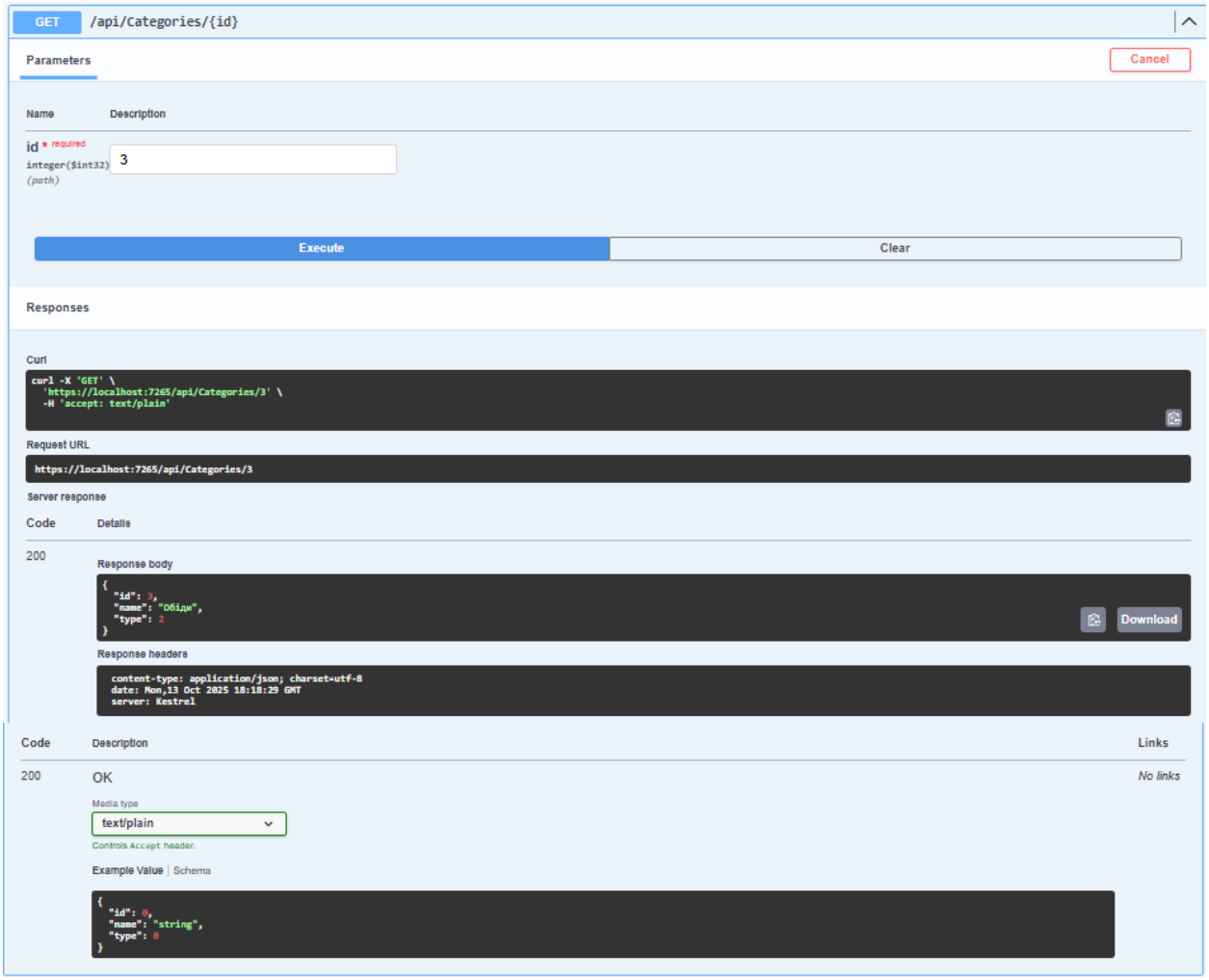




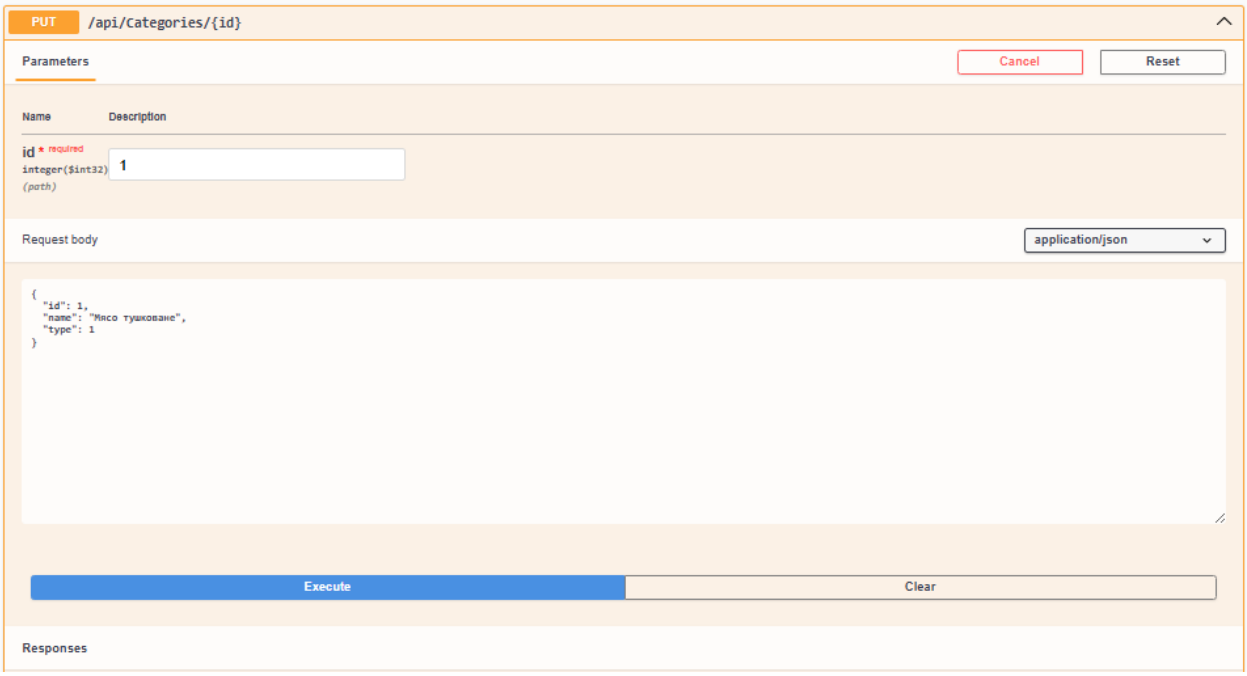
Виконаємо запит на додавання категорії.

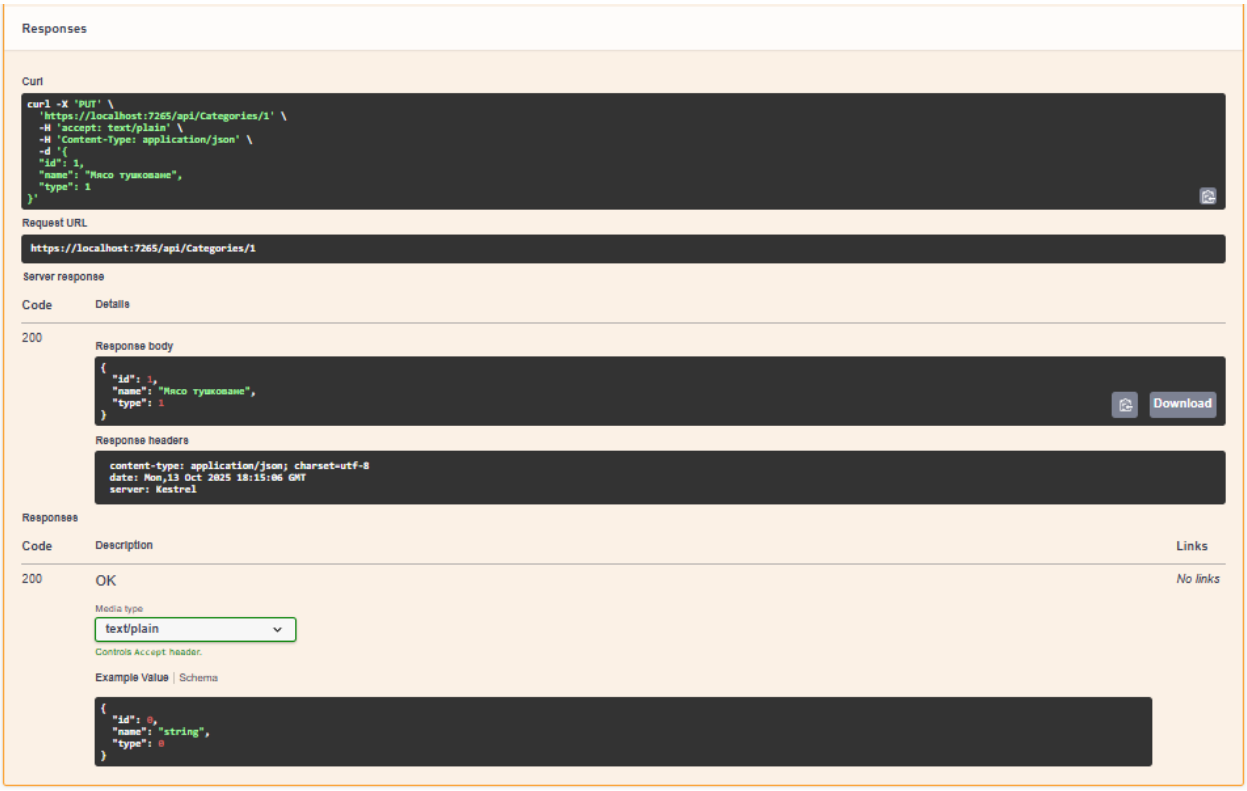


Виконаємо запит на отримання конкретної категорії.

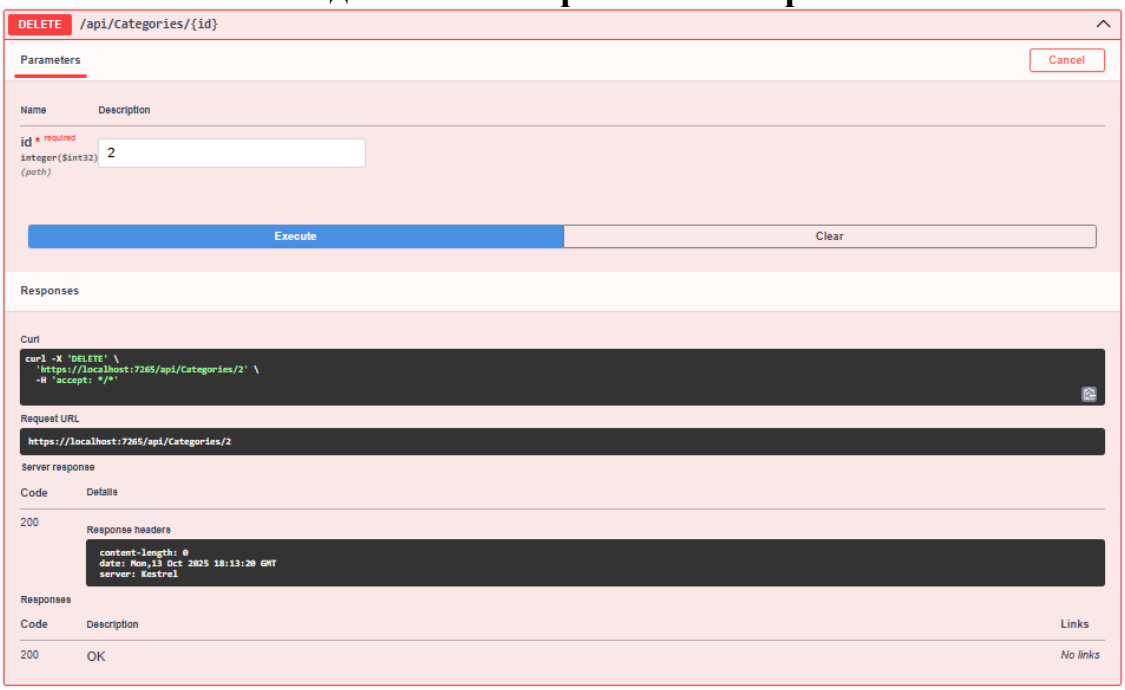


Виконаємо запит на оновлення конкретної категорії.



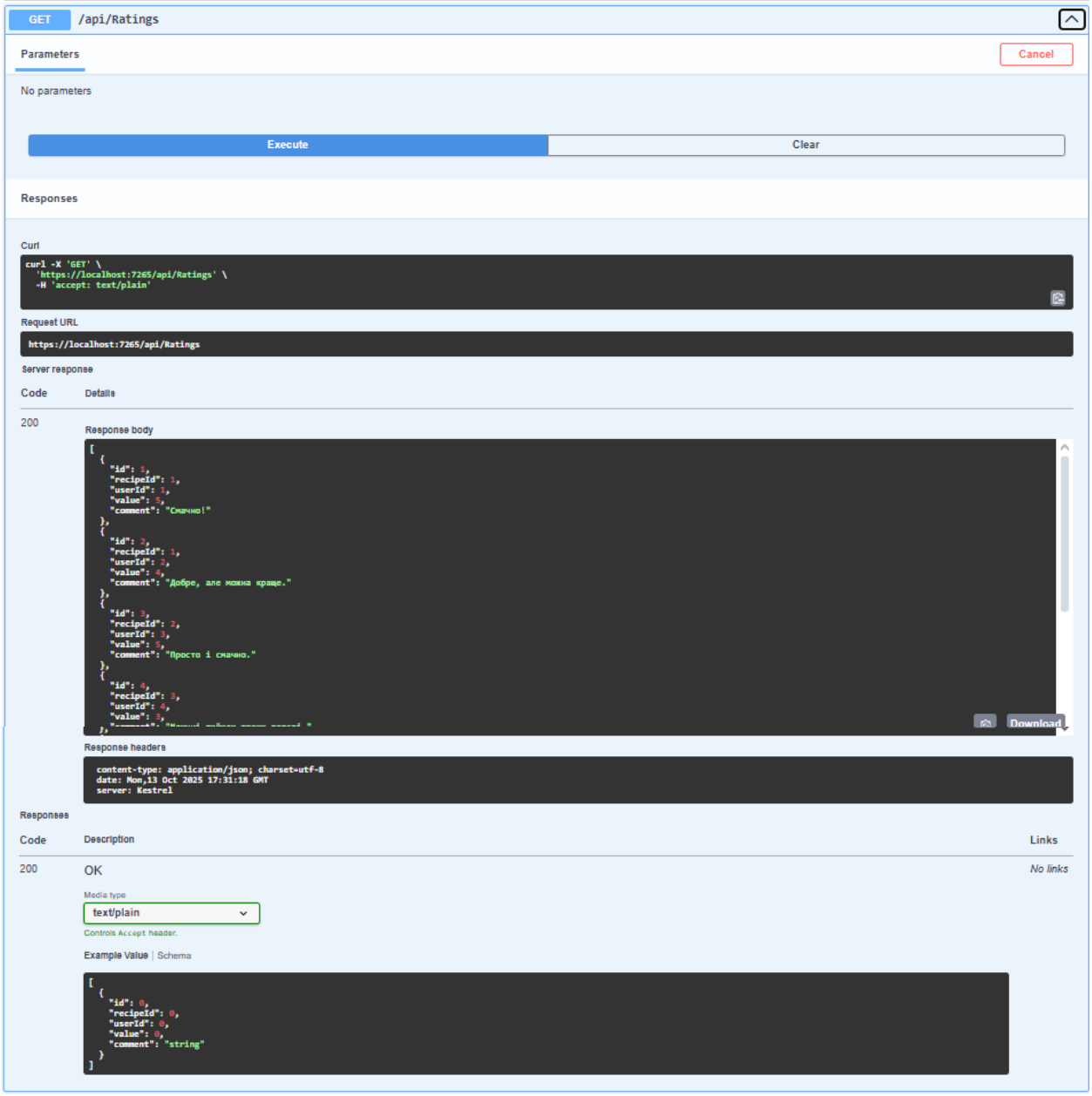


Виконаємо запит на видалення конкретної категорії.

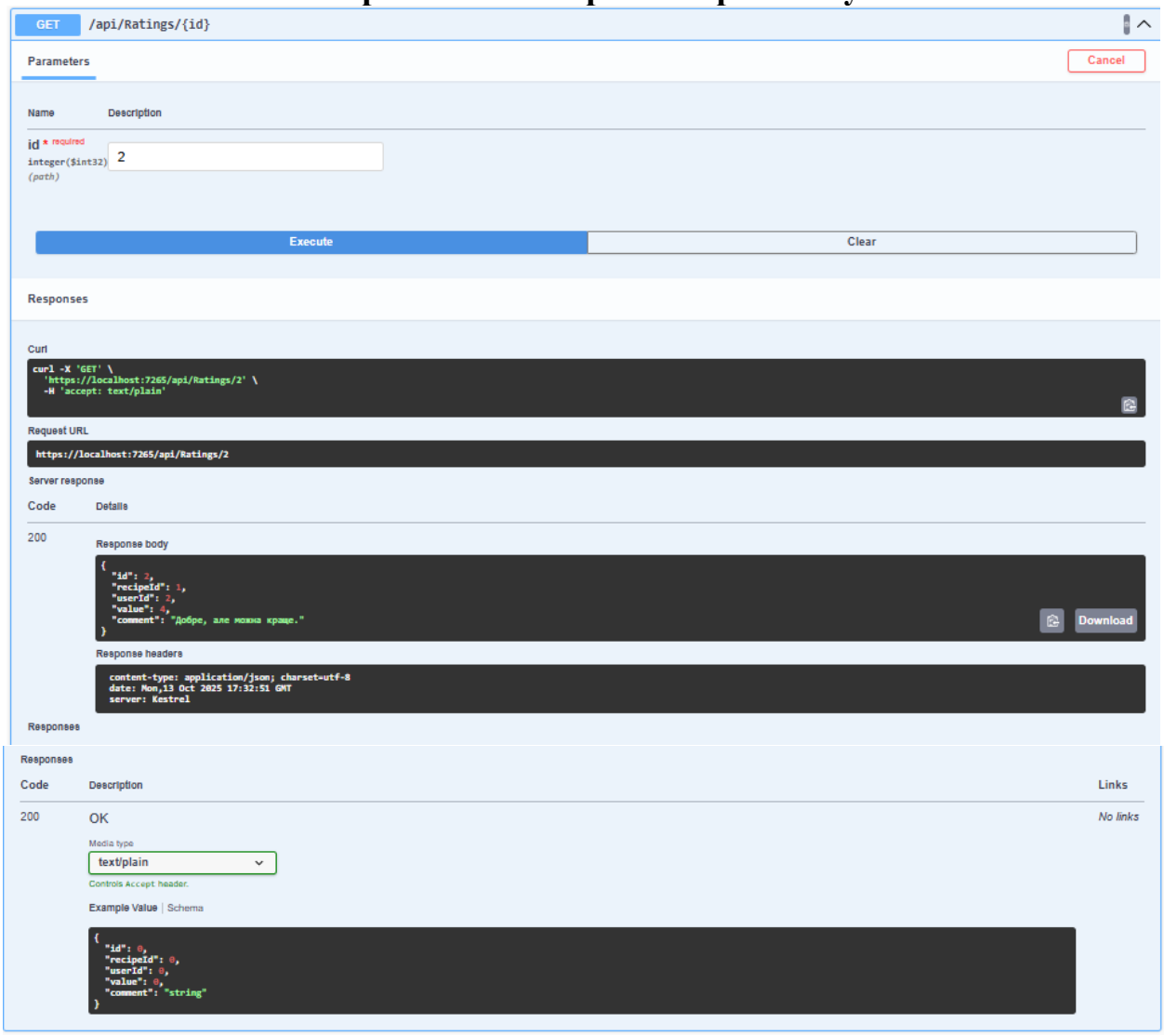


Ratings

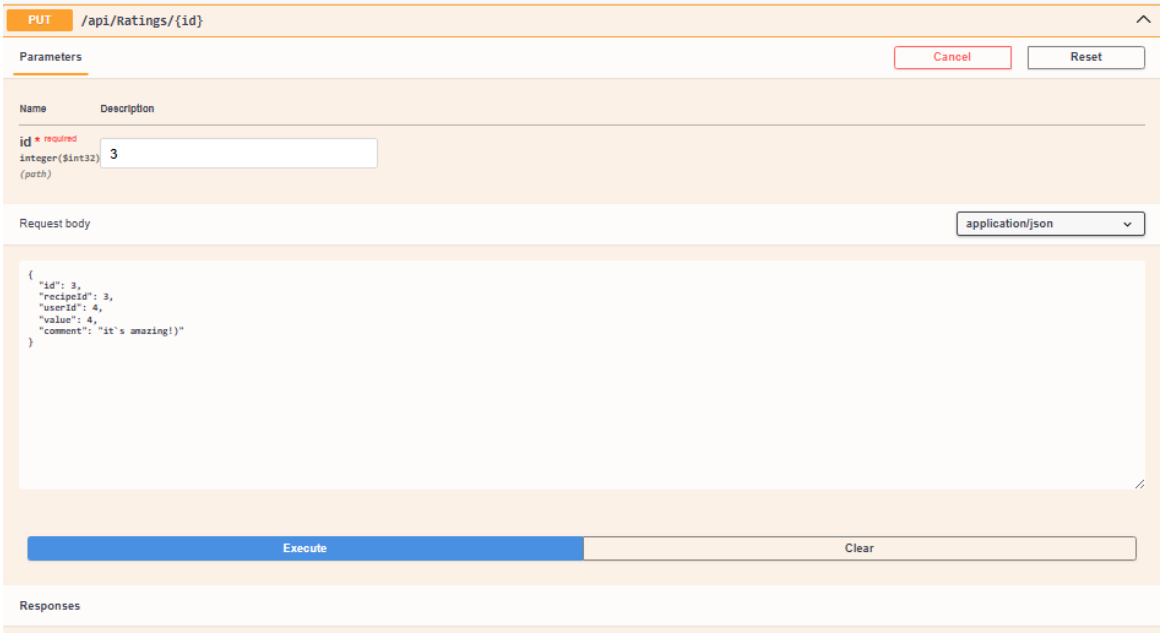
Виконаємо запит на отримання рейтингів.

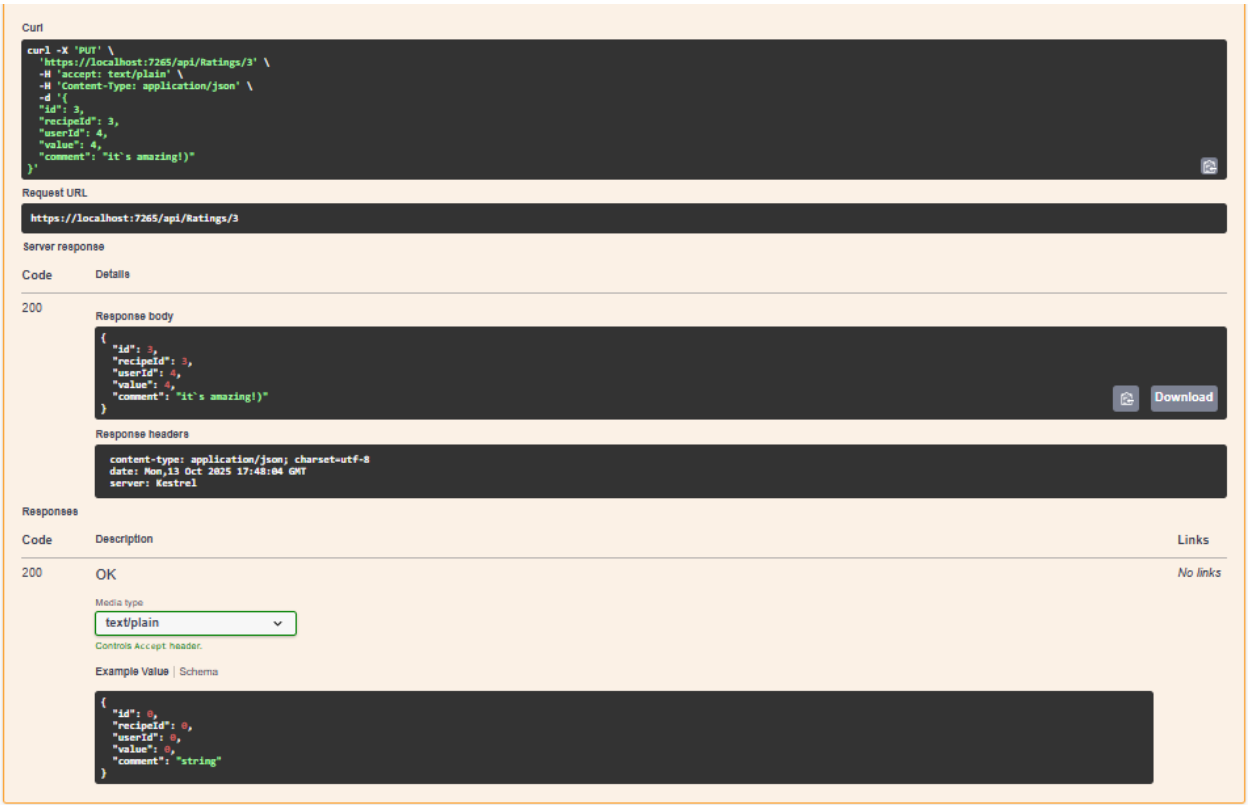


Виконаємо запит на отримання конкретного рейтингу.

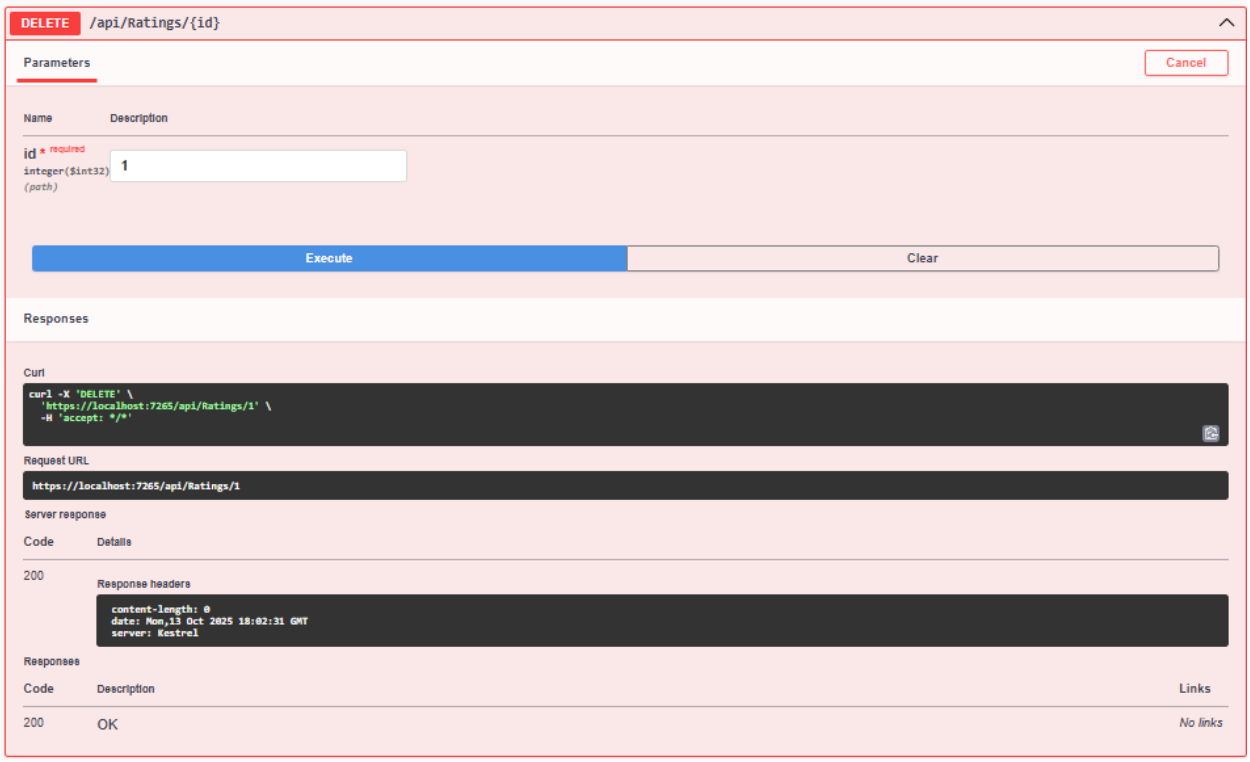


Виконаємо запит на оновлення конкретного рейтингу.

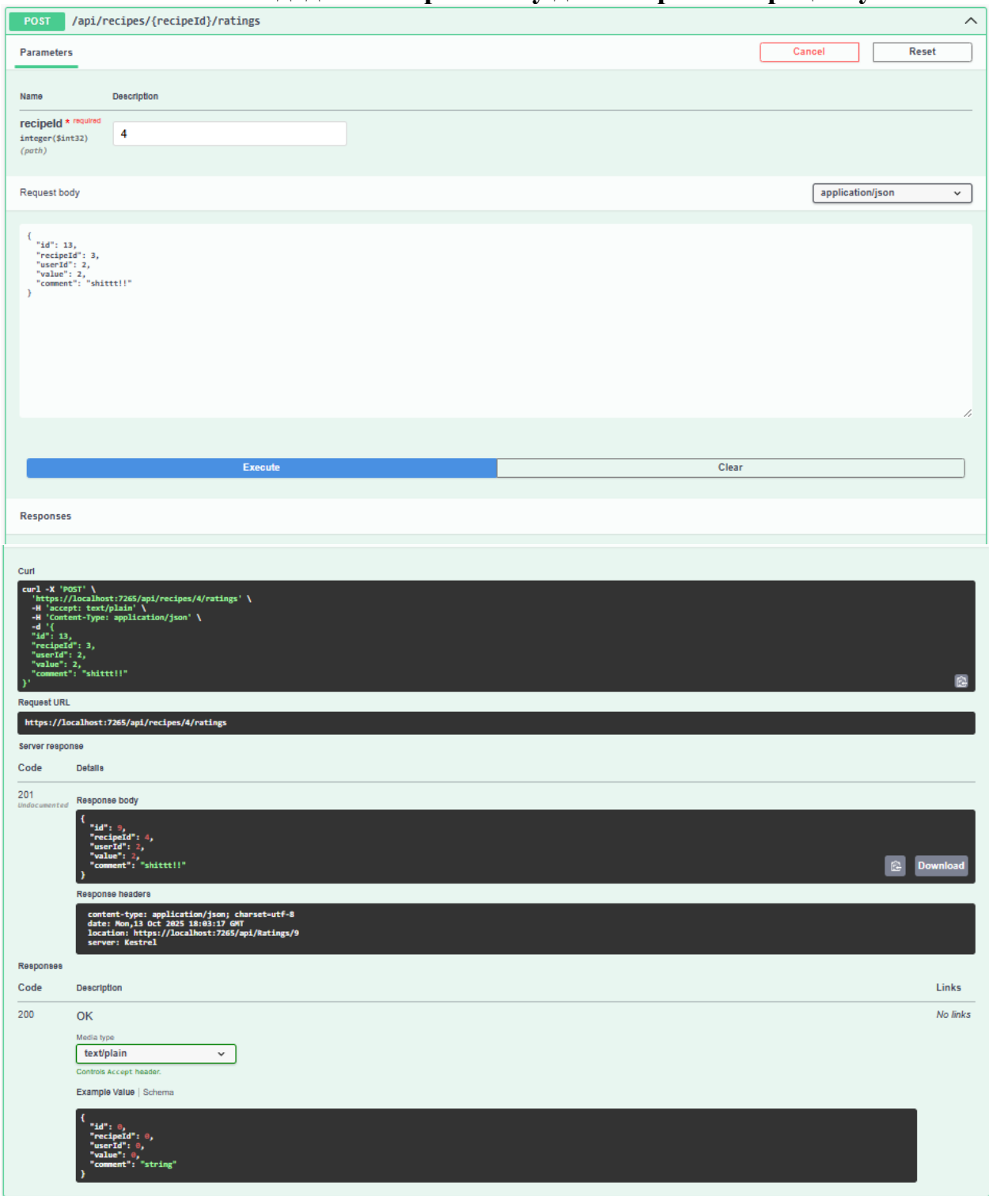




Виконаємо запит на видалення конкретного рейтингу.

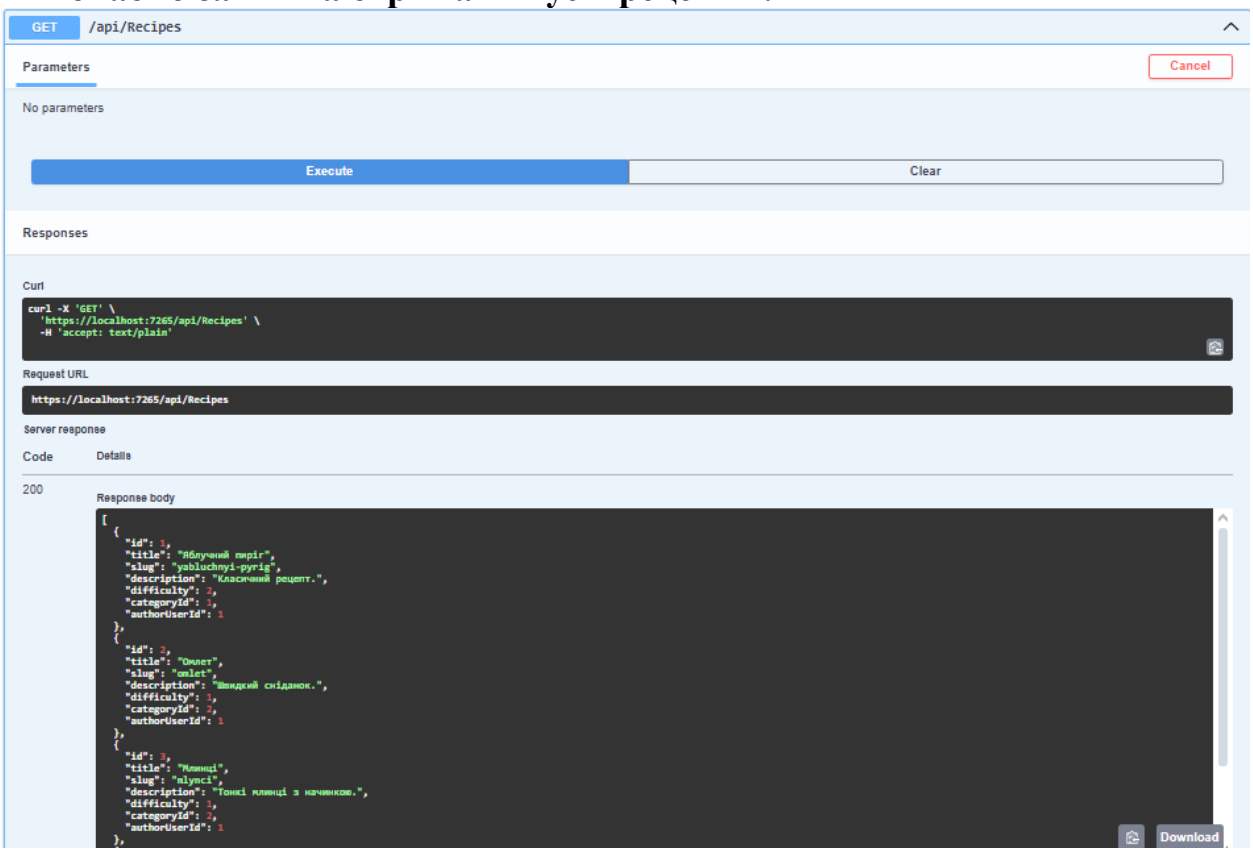


Виконаємо запит на додавання рейтингу до конкретного рецепту.

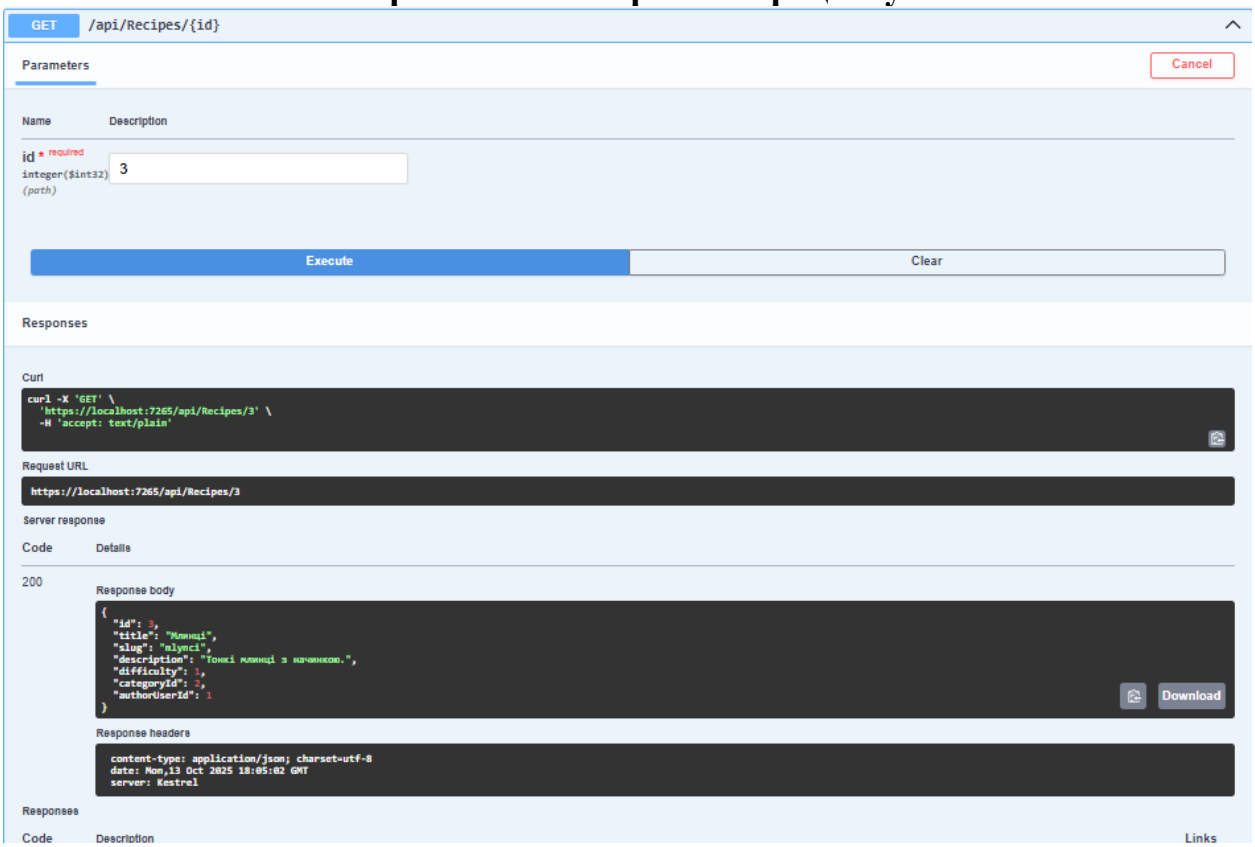


Recipes

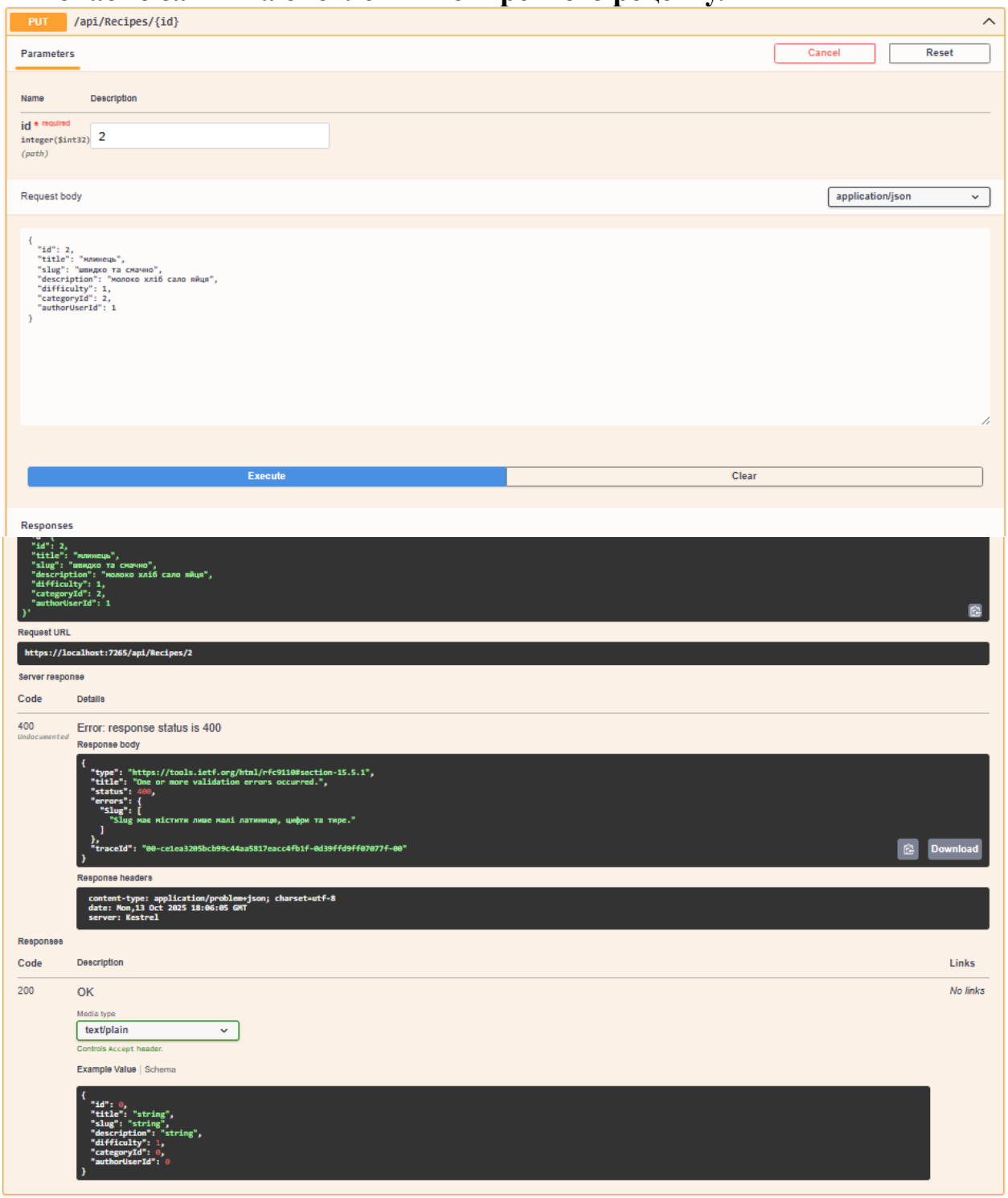
Виконаємо запит на отримання усіх рецептів.



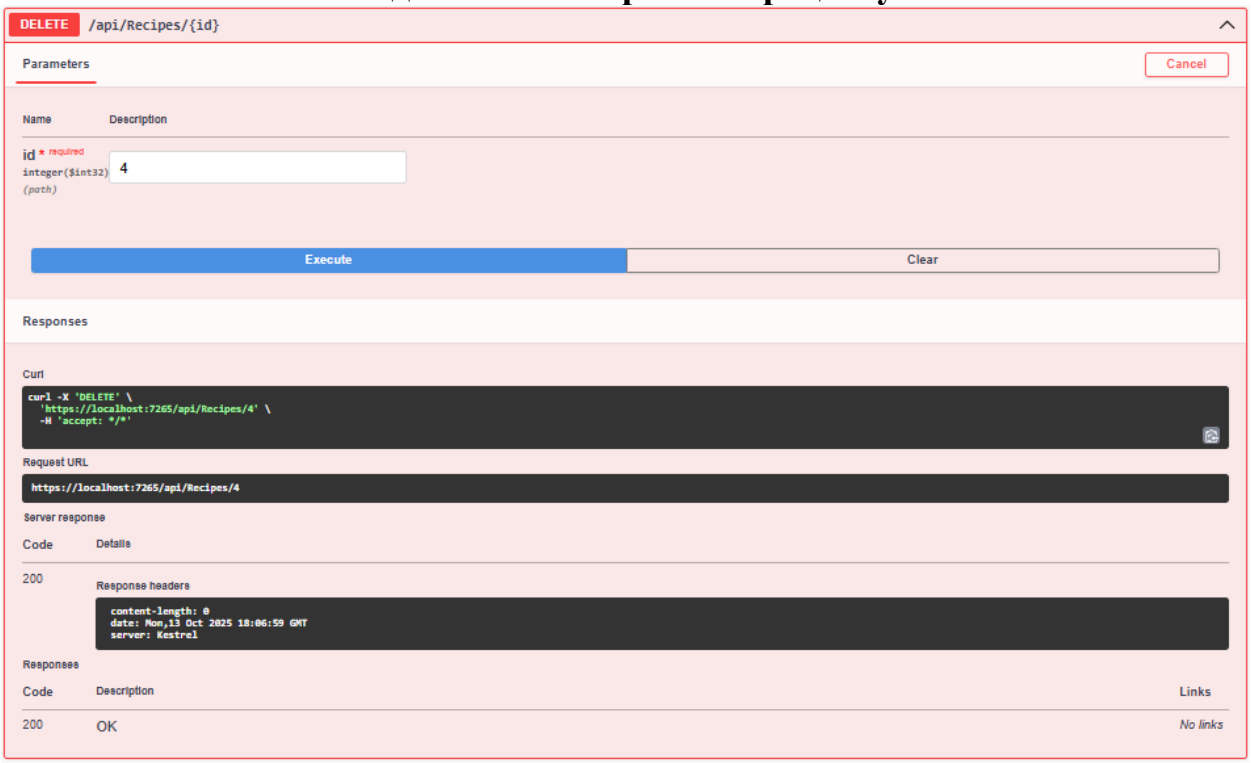
Виконаємо запит на отримання конкретного рецепту.



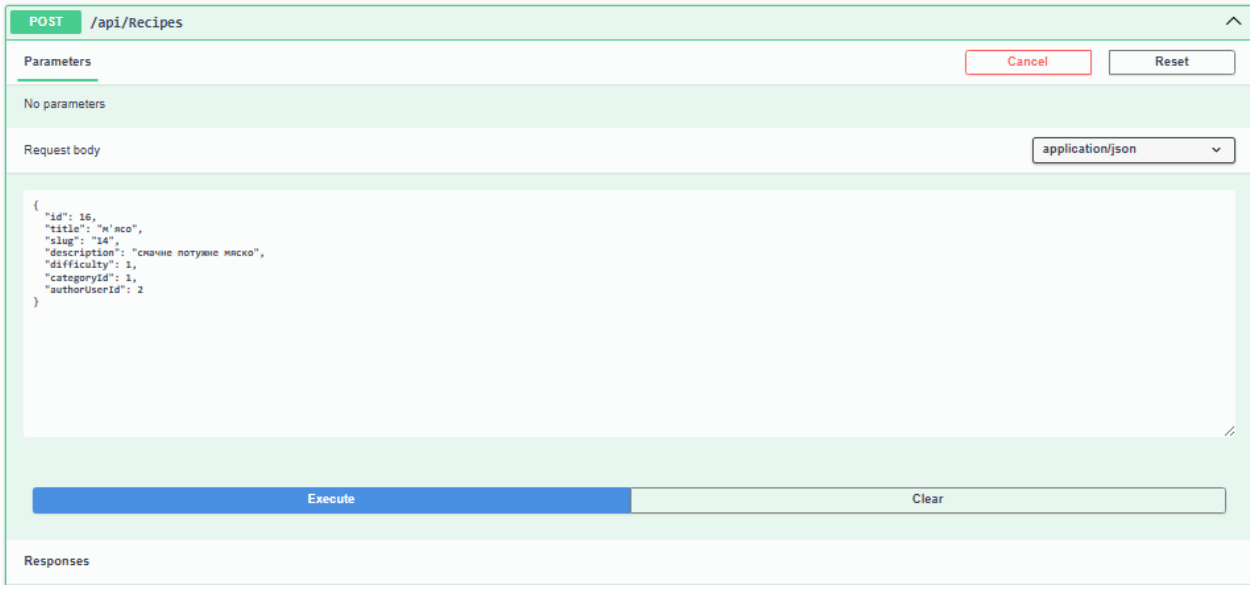
Виконаємо запит на оновлення конкретного рецепту.

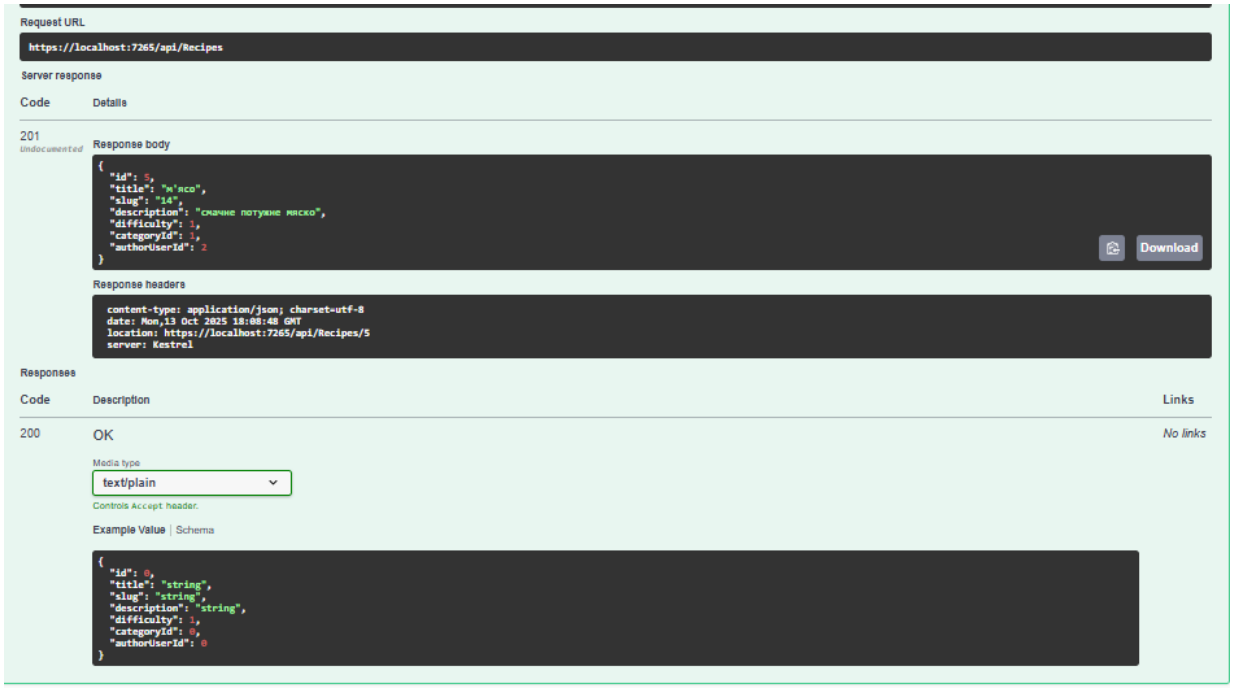


Виконаємо запит на видалення конкретного рецепту.



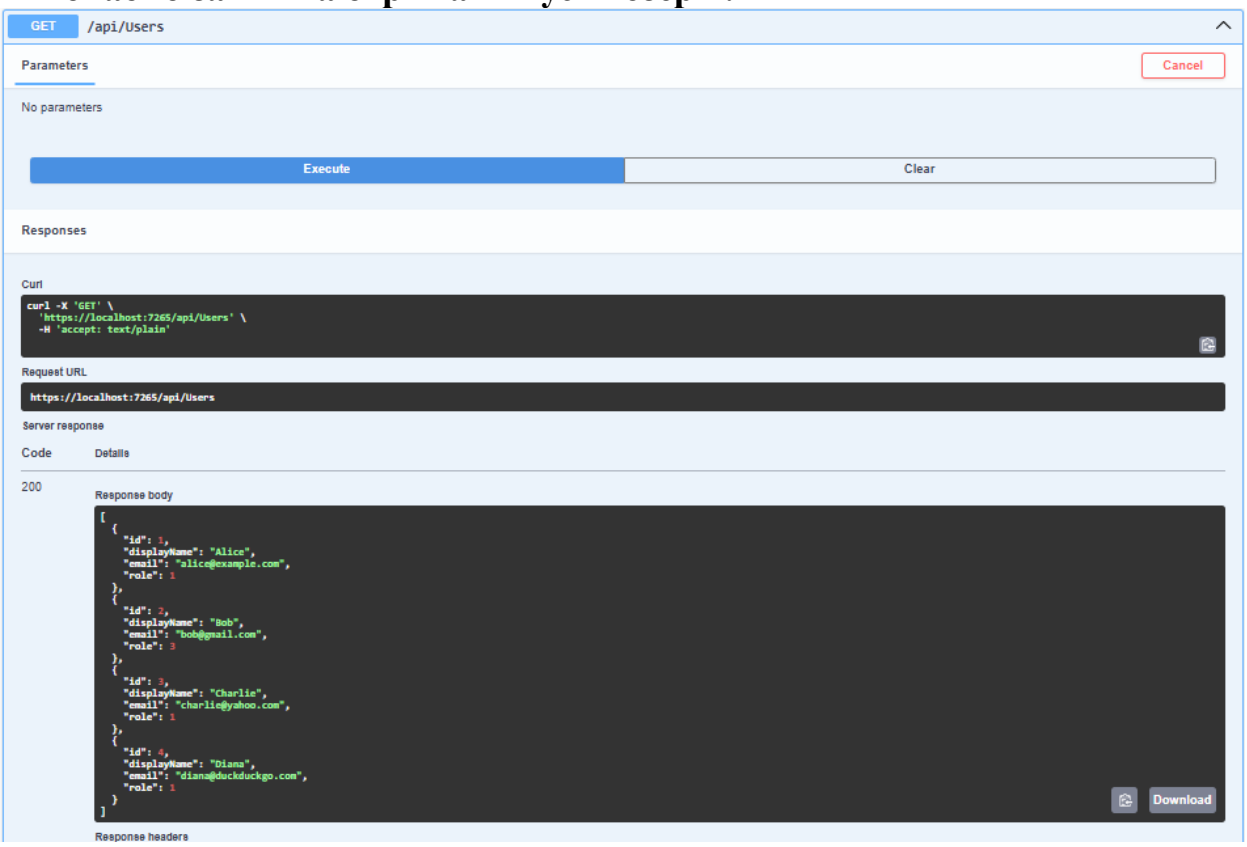
Виконаємо запит на публікацію рецепта.

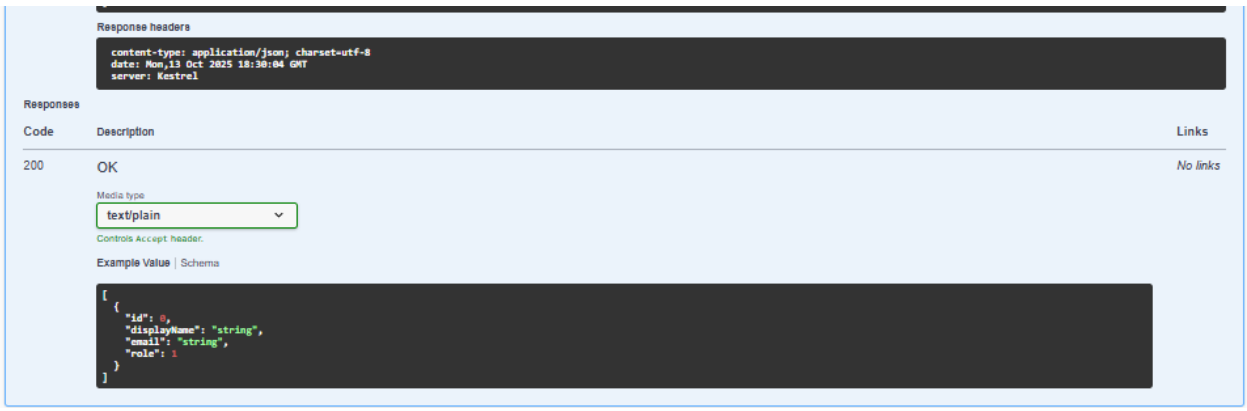




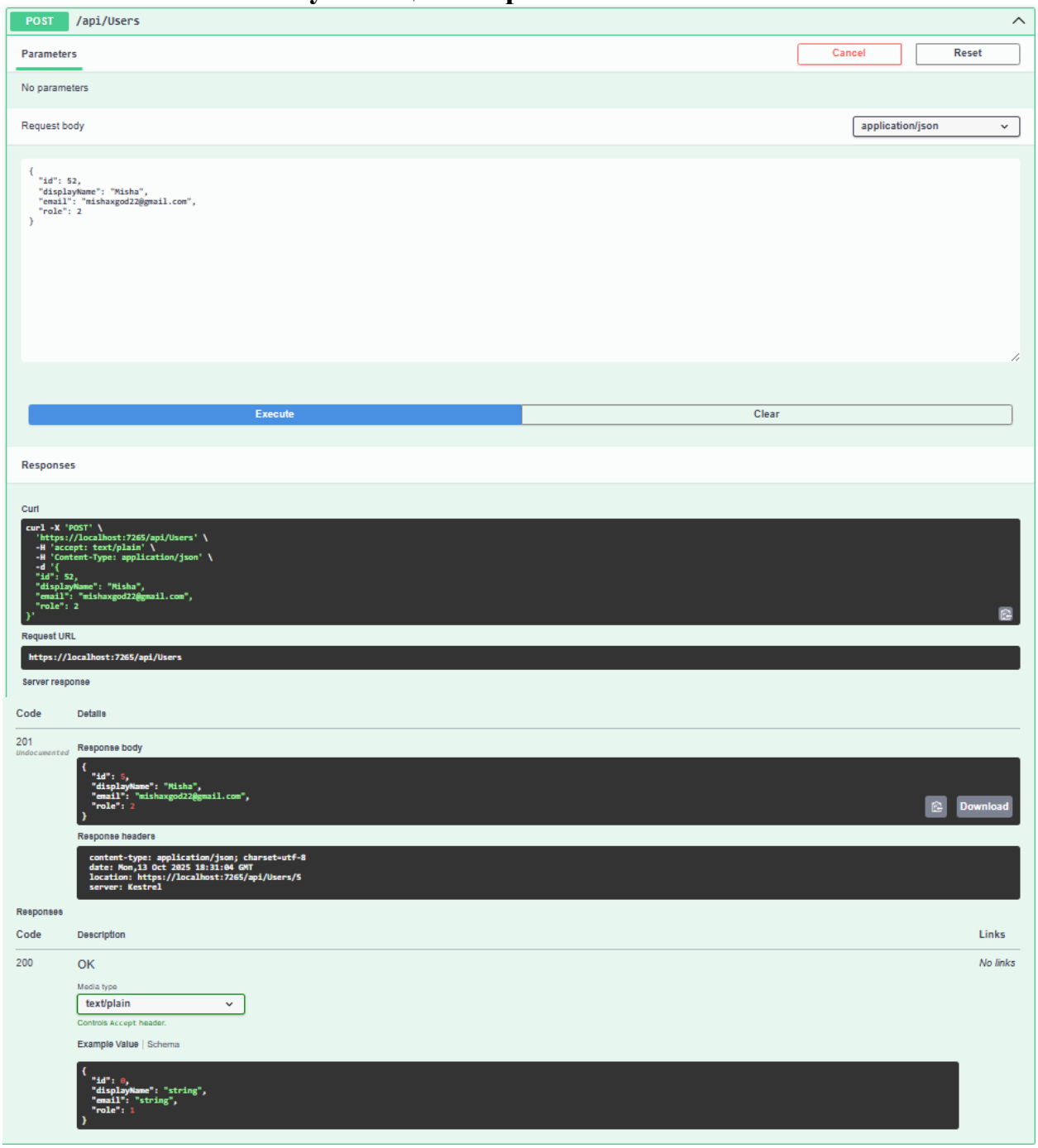
Users

Виконаємо запит на отримання усіх юзерів.

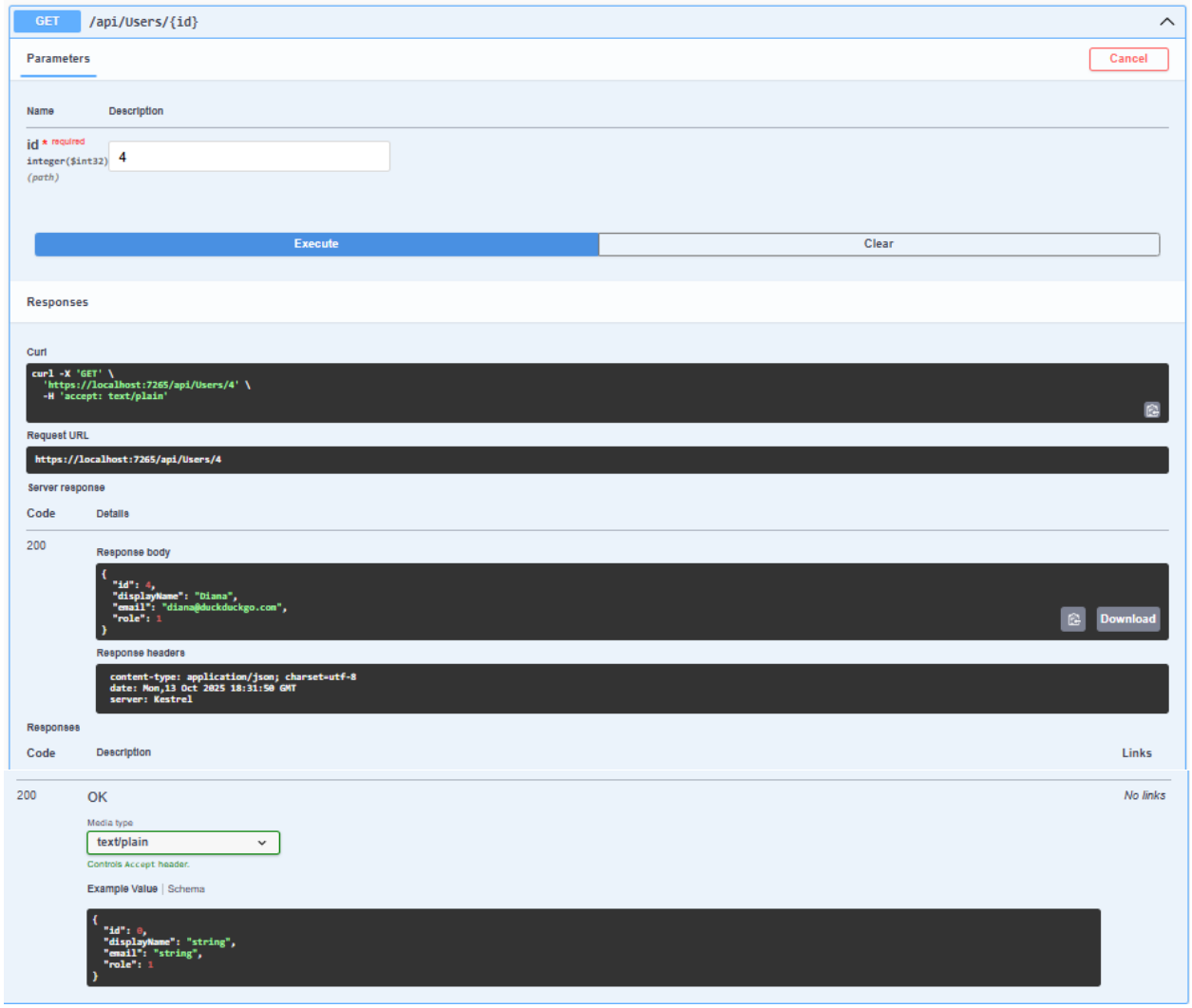




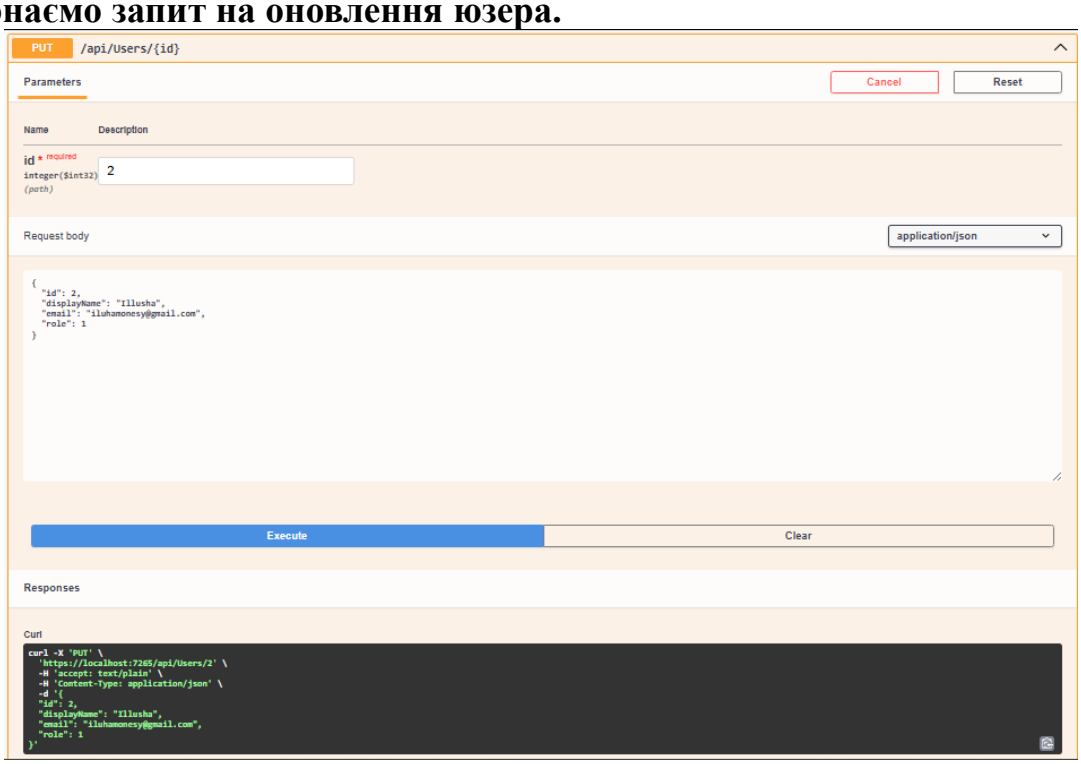
Виконаємо запит на публікацію юзера.



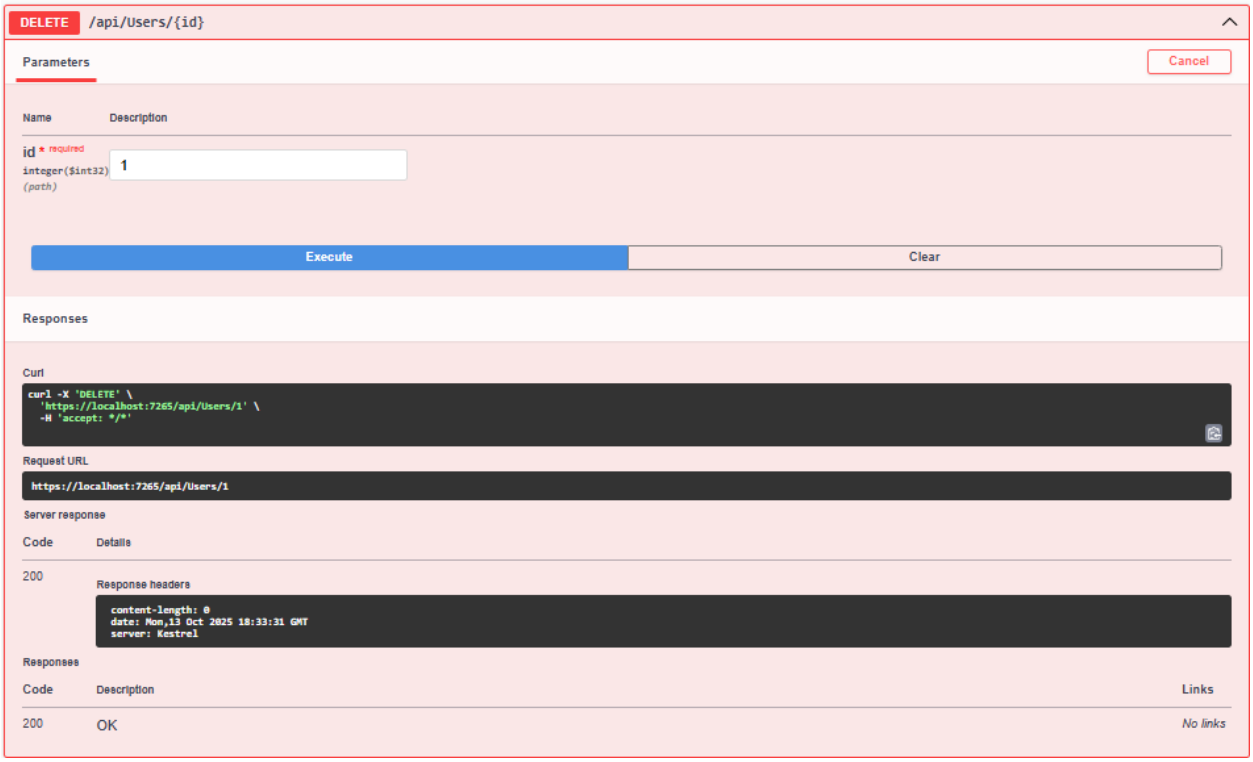
Виконаємо запит на отримання конкретного юзера.



Виконаємо запит на оновлення юзера.



Виконаємо запит на видалення юзера.



Висновок: Під час виконання лабораторної роботи, я ознайомився з процесом створення клієнт-серверної архітектури. Навчився створювати RESTful API для взаємодії між клієнтом і сервером.