**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](https://docs-mai.vercel.app/docs/developmement/minimal_api).

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

1. **Додати Swagger (OpenAPI)** для зручного тестування API через браузер:
2. Перевірити роботу API через **Swagger** або **Postman**.

**Завдання 2**

Ознайомитися з кроками створення **ASP.NET Core Web API** | [Check out the](https://docs-mai.vercel.app/docs/developmement/asp_net).

Створити **ASP.NET Core Web API**:

1. Створити **3–4 контролера** відповідно до Додатку А(наприклад: TodoController, UserController, CategoryController).
2. Створити відповідні моделі даних. Викорситати Enum, як поле класу. [Про Enum тут](https://docs-mai.vercel.app/docs/developmement/additionals/enum)
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](https://docs-mai.vercel.app/docs/developmement/additionals/regex)
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

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

**Завдання 3**

Спроєктувати архітектури Controller – Service – Repository

1. **Розробити логічну структуру Web API**, що складається з трьох рівнів:
   * **Controller Layer** — приймає HTTP-запити, валідує дані, викликає сервіси.
   * **Service Layer** — реалізує бізнес-логіку.
   * **Repository Layer** — відповідає за роботу з джерелом даних (спочатку in-memory).
2. **Створити UML Component Diagram**, на якій відобразити:
   * три основні компоненти: Controllers, Services, Repositories;
   * зв’язки між ними: Controller → Service → Repository;
   * залежності (інтерфейси IService, IRepository).
3. **Описати кожен рівень (2–3 речення)**:
   * його роль у системі;
   * що саме він реалізує;
   * приклади методів або обов’язків.
4. **Додати короткий опис до звіту.**

**Завдання 4**

Додати MongoDB та реалізація CRUD через Repository/Service. Підключити MongoDB замість in-memory списків і реалізувати CRUD-операції на рівні архітектури Controller–Service–Repository.

1. **Підключити MongoDB:**
   * додати пакет MongoDB.Driver;
   * створити MongoBdClient;
2. **Оновити моделі:**
   * додати атрибути [BsonId], [BsonRepresentation(BsonType.ObjectId)];
   * підготувати моделі документів для MongoDB.
3. **Створити Repository-шар:**
   * створити інтерфейси IRepository з методами: GetAllAsync(), GetByIdAsync(id), CreateAsync(entity), UpdateAsync(entity), DeleteAsync(id);
   * реалізувати їх за допомогою MongoDB драйвера.
4. **Створити Service-шар:**
   * створити інтерфейси IService і класи, які використовують репозиторії;
   * додати базову бізнес-логіку (наприклад, перевірку існування об’єкта перед оновленням чи видаленням).
5. **Оновити контролери:**
   * прибрати роботу зі списками List<T>;
   * підключити сервіси через конструктор (Dependency Injection), можна перейти відразу до завданян 5, де це виконується
   * CRUD-операції мають працювати з MongoDB.
6. **Перевірити роботу:**
   * виконати всі CRUD-запити через Swagger або Postman;
   * показати, що дані реально зберігаються у MongoDB (скрін із Mongo Compass).

**Завдання 5**

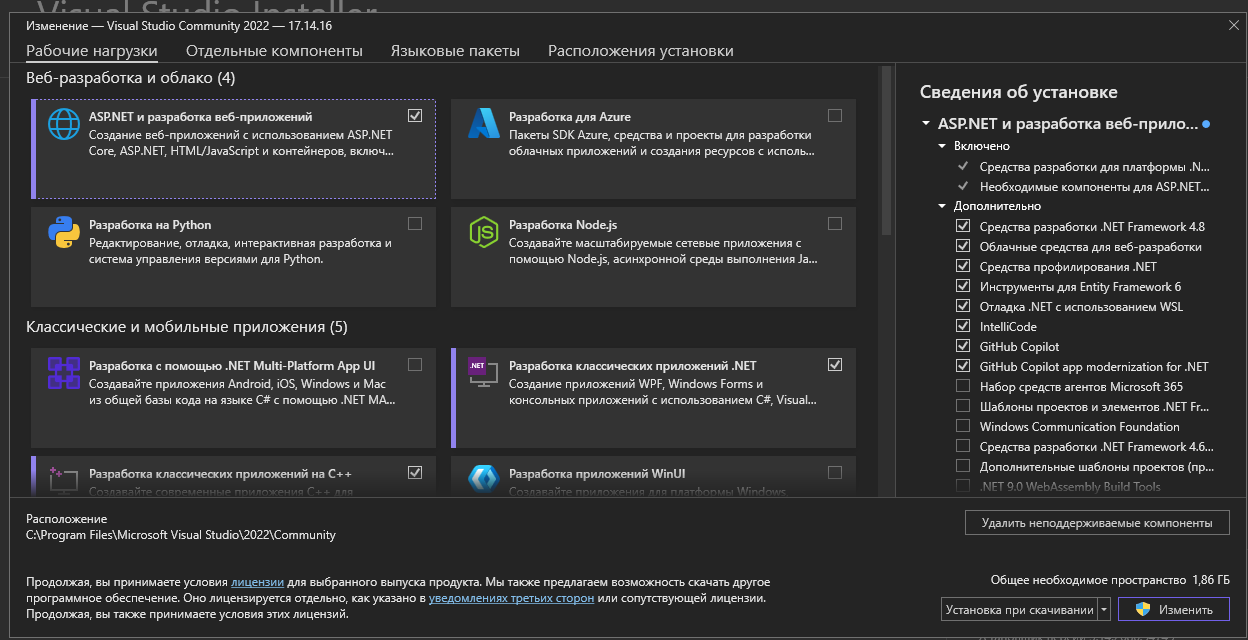
Впровадити Dependency Injection (DI) для всіх шарів і налаштувати AutoMapper для автоматичного мапінгу моделей.

1. **Налаштувати Dependency Injection:**
   * зареєструвати у Program.cs
   * отримувати залежності через конструктор у контролерах і сервісах (без new усередині)
2. **Додати AutoMapper:**
   * встановити пакет AutoMapper.Extensions.Microsoft.DependencyInjection
   * створити профіль у папці /Mapping
   * зареєструвати у Program.cs
   * використовувати IMapper у сервісах або контролерах для перетворення моделей
3. **Перевірити роботу:**
   * виконати CRUD-запити (щоб упевнитись, що DI і AutoMapper працюють)
   * показати приклад мапінгу (DTO ↔ Model) у коді
   * додати скріншоти успішних запитів
4. **Додати до звіту:**
   * фрагменти коду з DI-реєстрацією, приклад профілю AutoMapper;
   * коротке пояснення, як працює DI і AutoMapper у вашому проєкті.

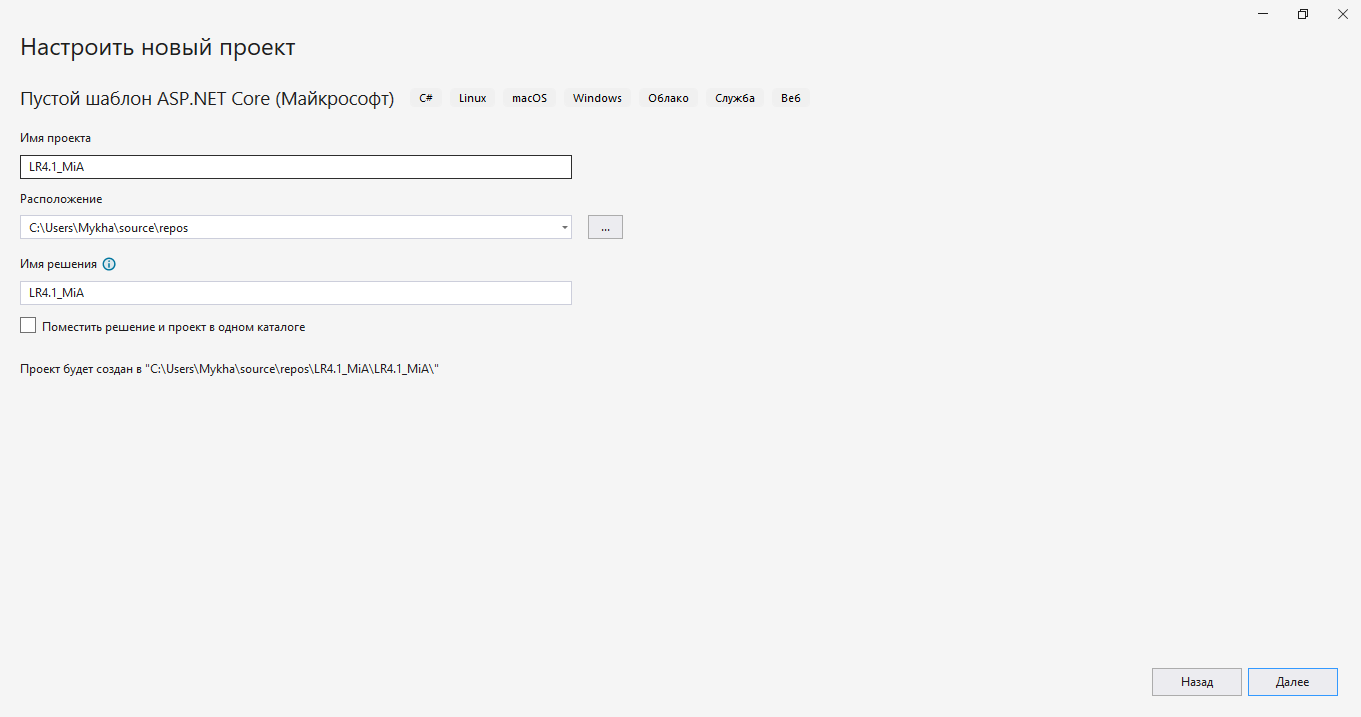
**Хід роботи**

**40.Онлайн-сервіс для обміну кулінарними рецептами з рейтингами**

1. Під час встановлення Visual Studio обираємо ASP.NET



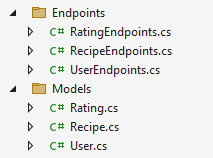
1. Перейдемо до створення проєкту. Для цього знаходимо проєкт ASP.NET Core empty.



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";

});

}

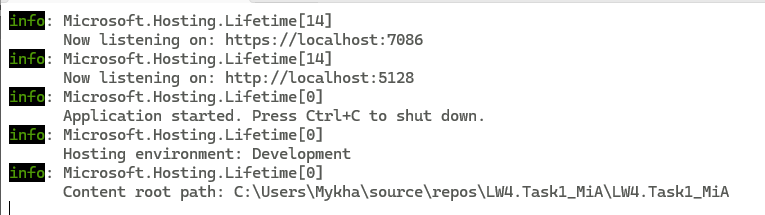
app.MapRecipeEndpoints();

app.MapUserEndpoints();

app.MapRatingEndpoints();

app.Run();

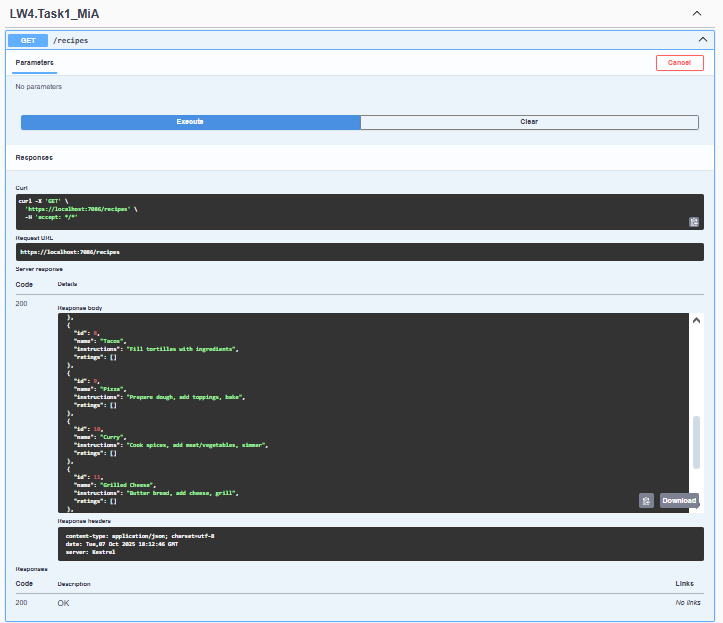
**Результат роботи:**

****

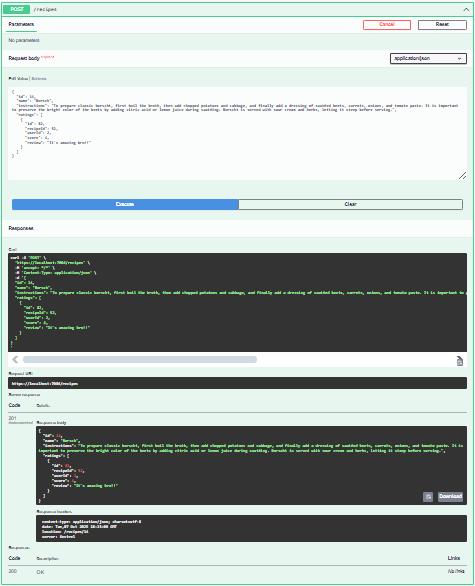
Програма показує те, що нам доступна вся інформація по посиланню.

**Запити**

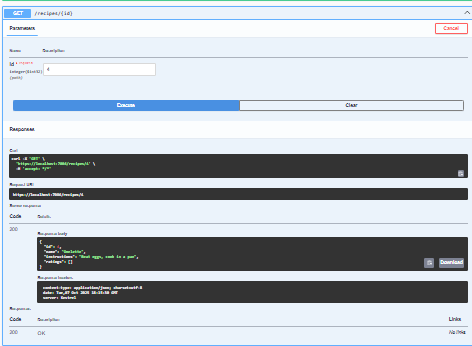
**Запит на отримання рецептів**



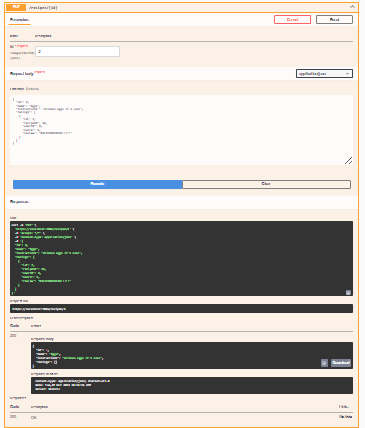
**Запит на публікацію рецепта**



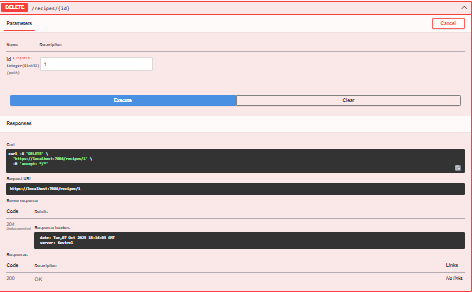
**Запит на отримання конкретного рецепта**

****

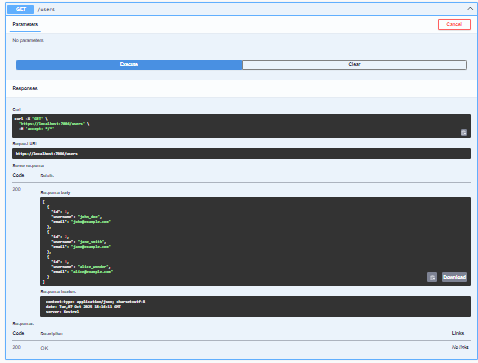
**Запит на оновлення рецепта**

****

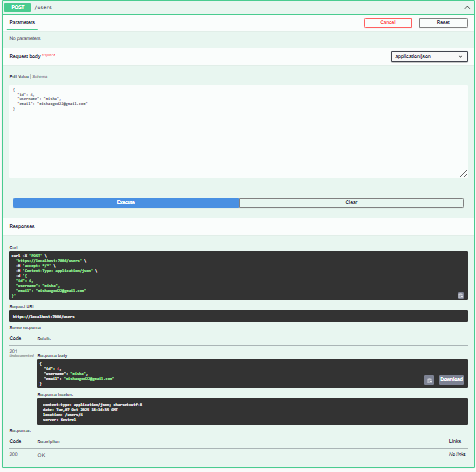
**Запит на видалення рецепта**

****

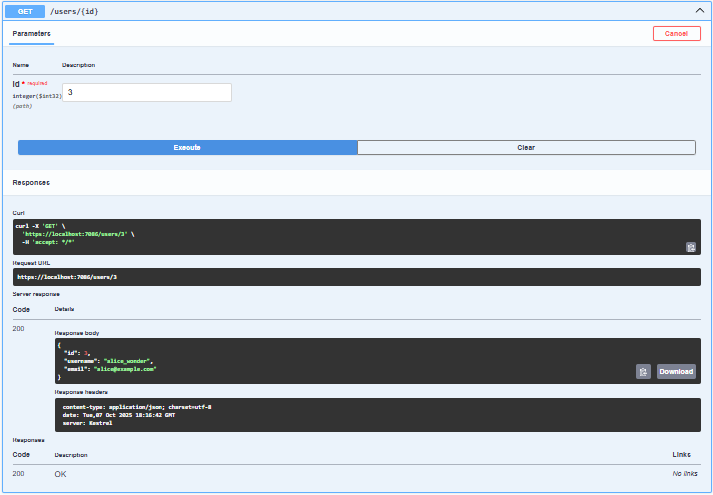
**Запит на отримання користувачів**

****

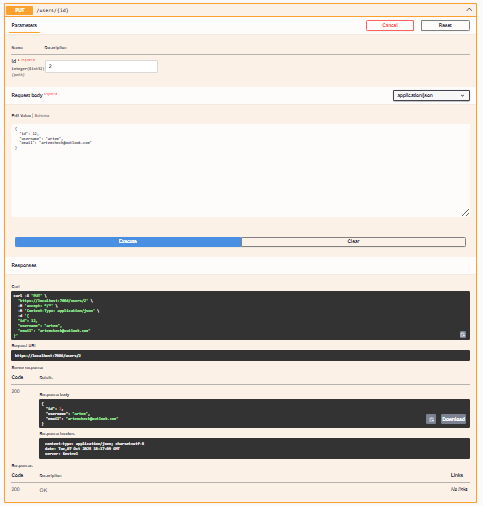
**Запит на додавання користувача**

****

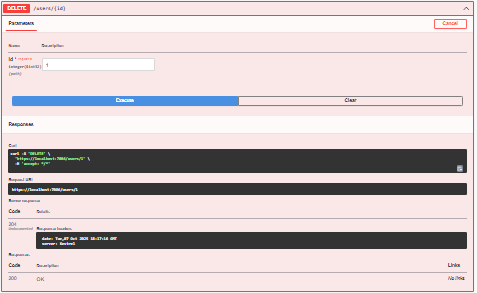
**Запит на отримання конкретного користувача**

****

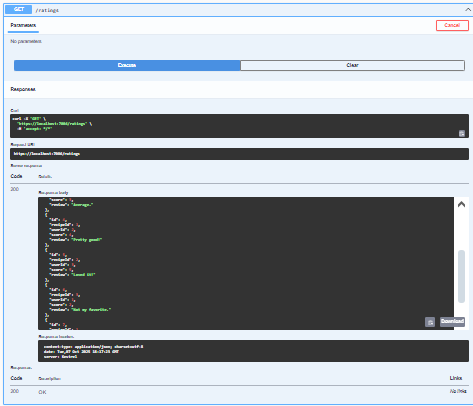
**Запит на оновлення користувача**

****

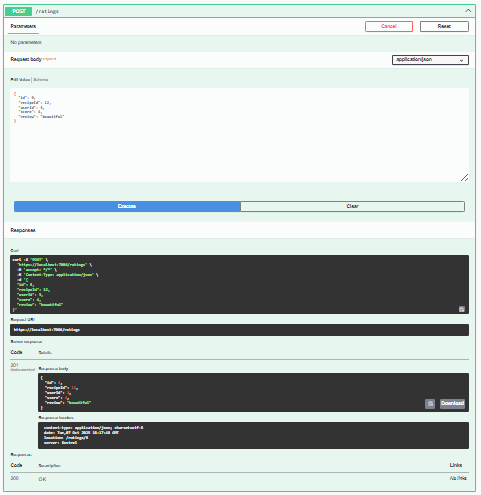
**Запит на видалення користувача**

****

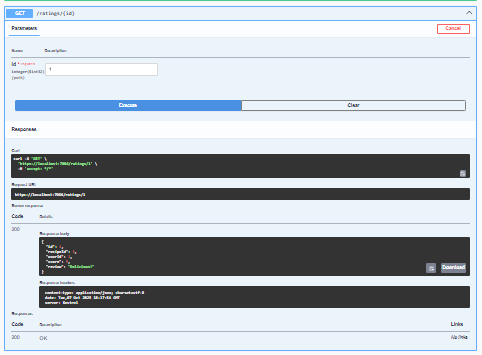
**Запит на отримання оцінок**

****

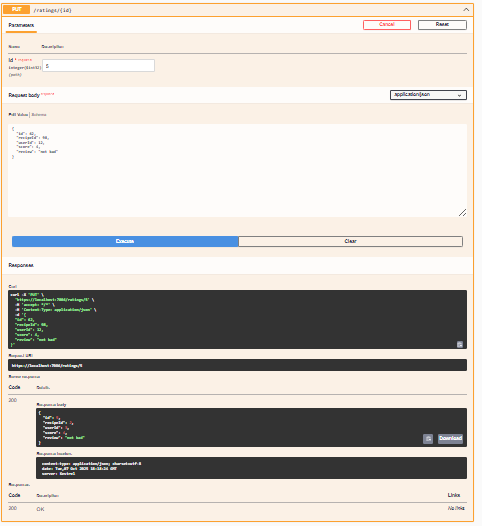
**Запит на додавання оцінки**

****

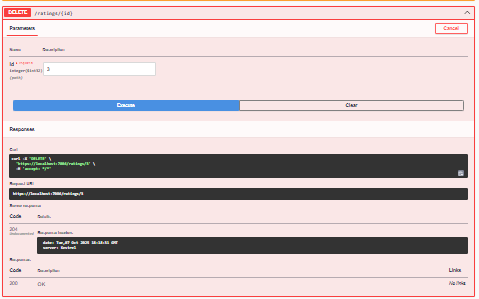
**Запит на отримання конкретної оцінки**

****

**Запит на оновлення оцінки**



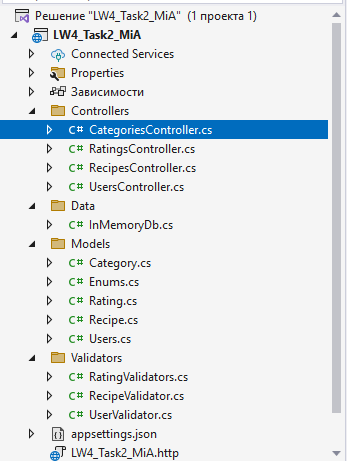
**Запит на видалення оцінки**



**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.RecipeValidator>();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

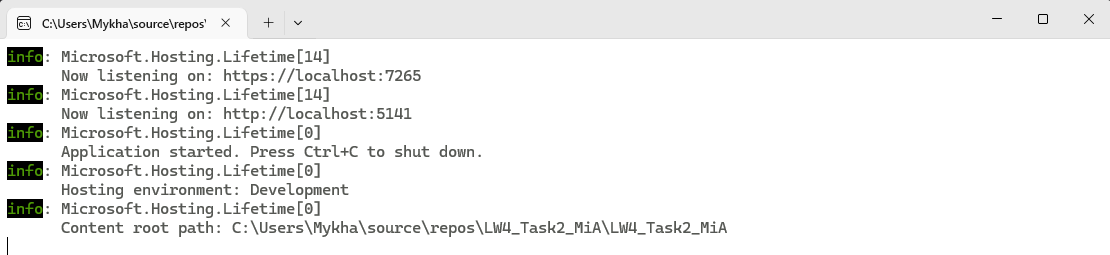
}

app.UseHttpsRedirection();

app.MapControllers();

app.Run();

**Результат програми**



Програма працює. Переходимо до Swagger.

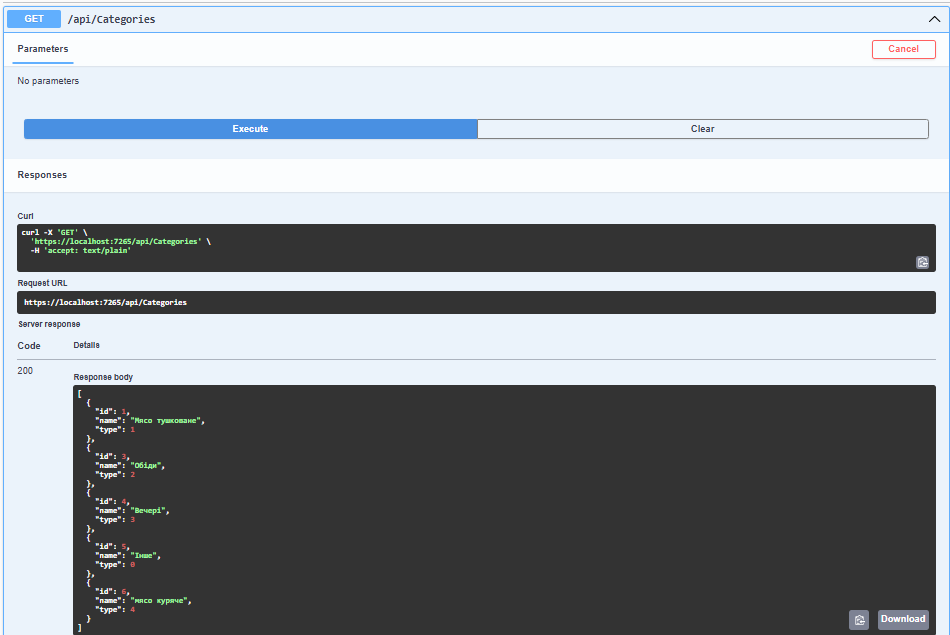


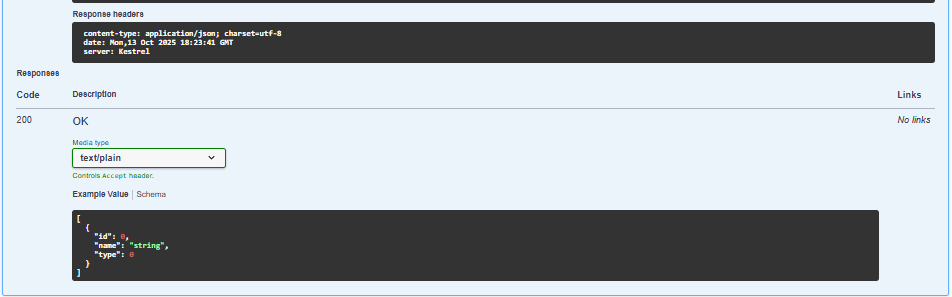
В програмі є 4 контролера: Categories, Ratings, Recipes, Users.

Для кожного з них є CRUD (C - Create, R - Read, U - Update, D – Delete).

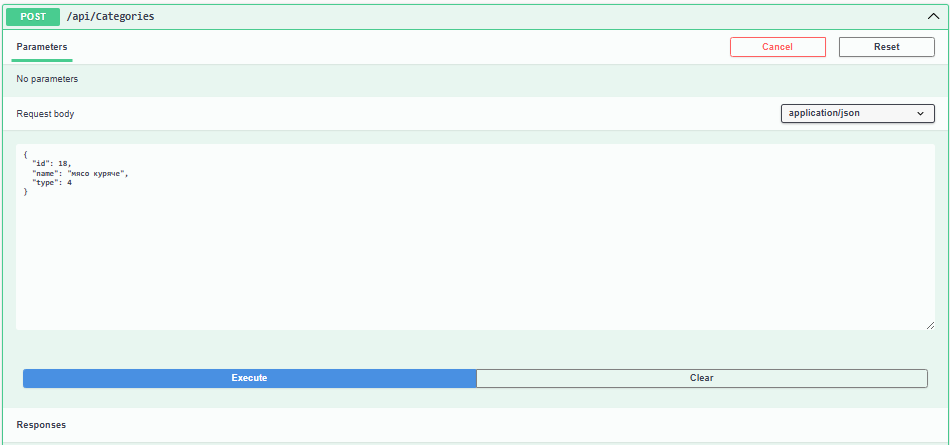
**Почнемо з Categories**

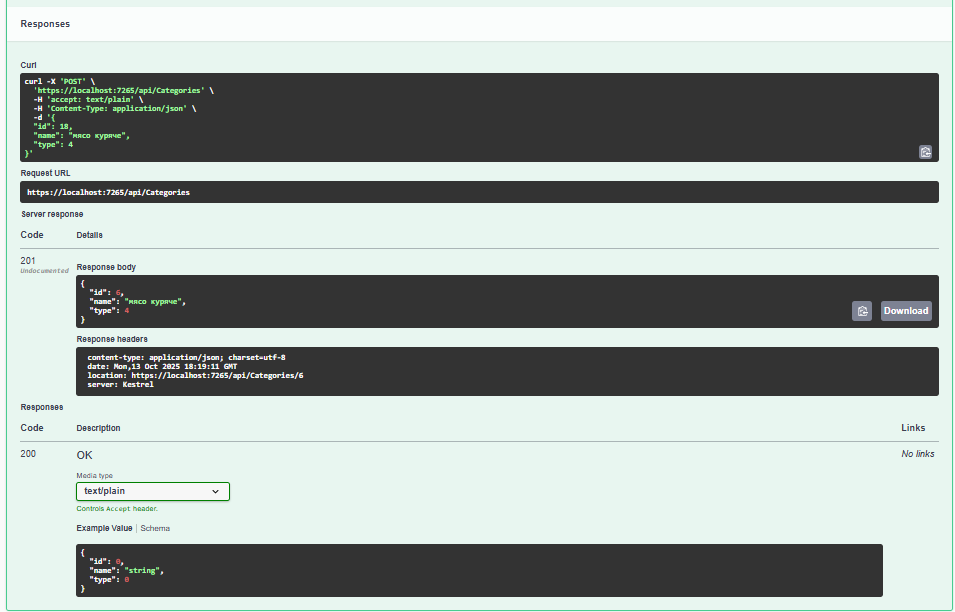
**Виконаємо запит на категорії.**

****

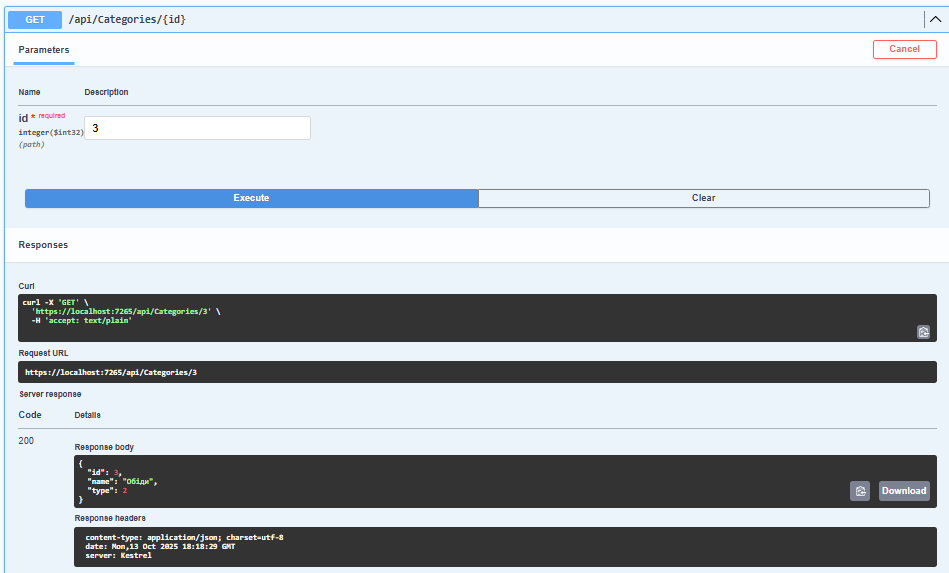
****

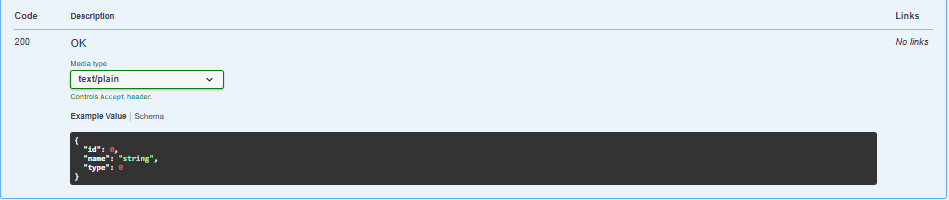
**Виконаємо запит на додавання категорії.**

****

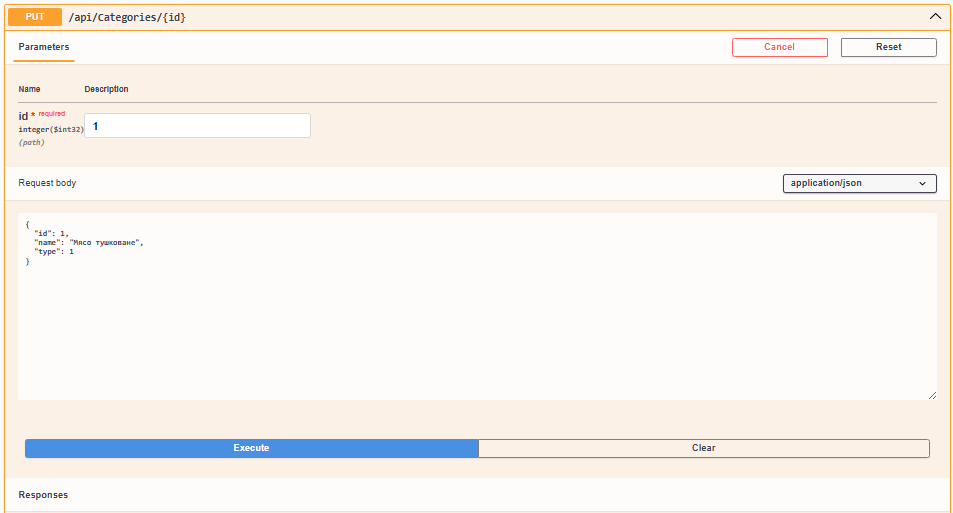
****

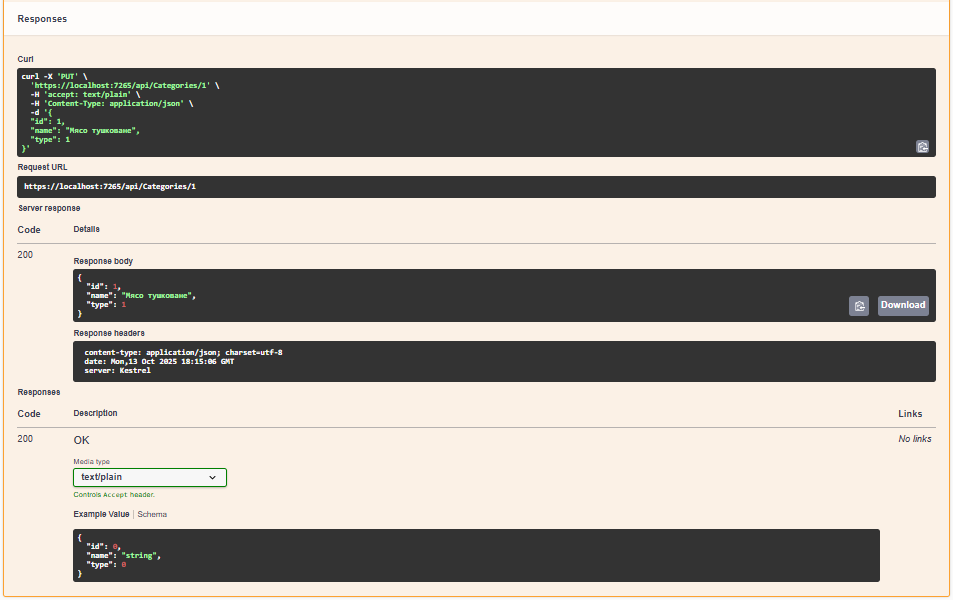
**Виконаємо запит на отримання конкретної категорії.**



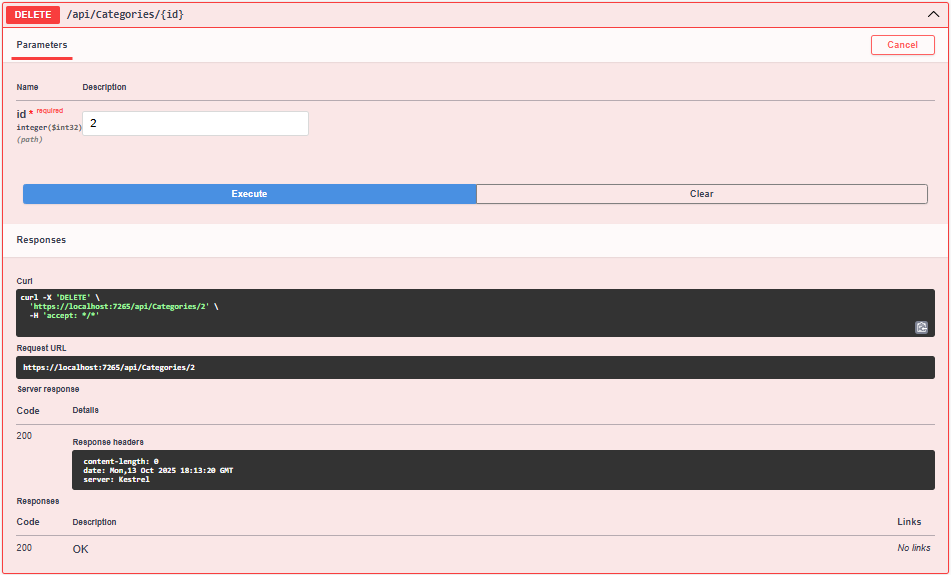


**Виконаємо запит на оновлення конкретної категорії.**



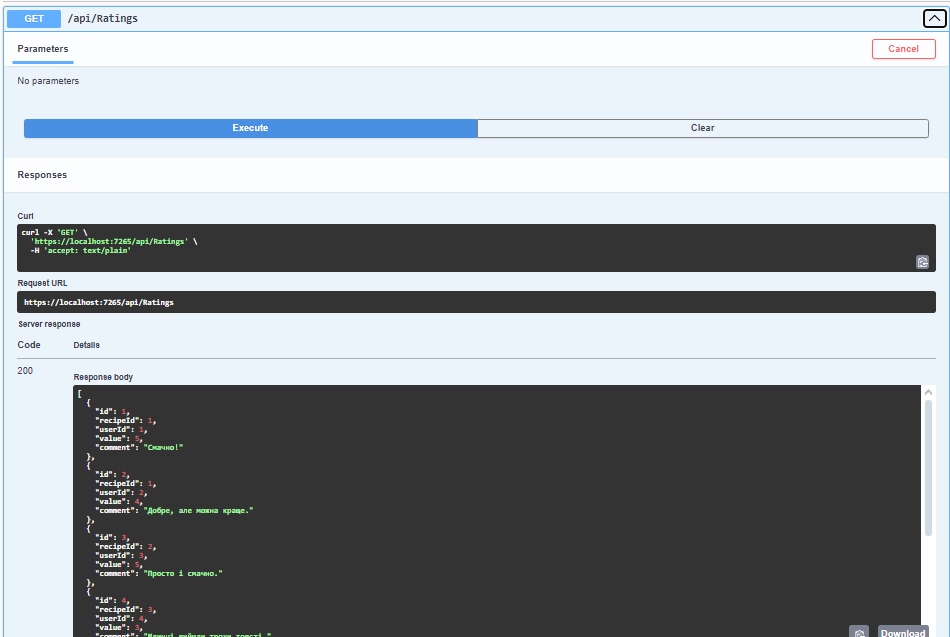


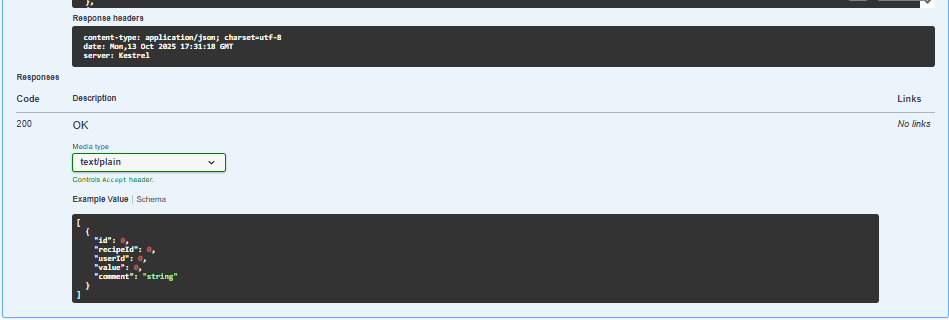
**Виконаємо запит на видалення конкретної категорії.**

****

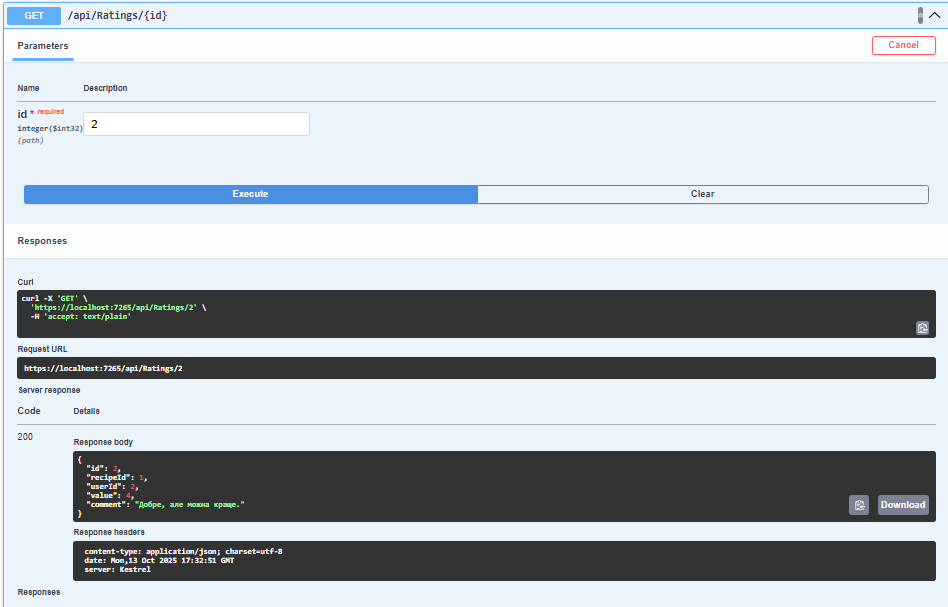
**Ratings**

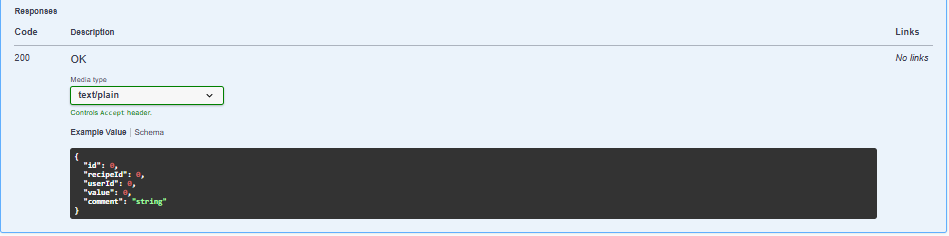
**Виконаємо запит на отримання рейтингів.**

****

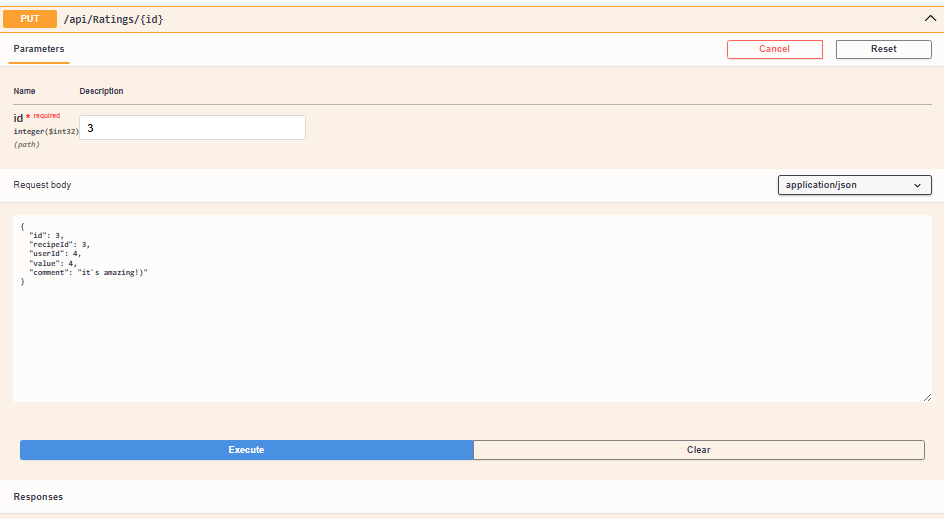
****

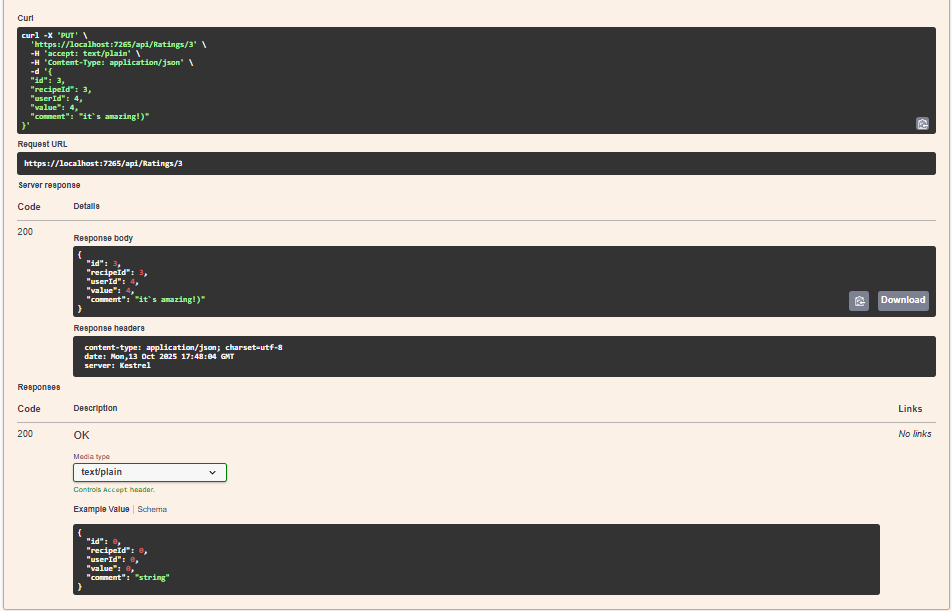
**Виконаємо запит на отримання конкретного рейтингу.**

****

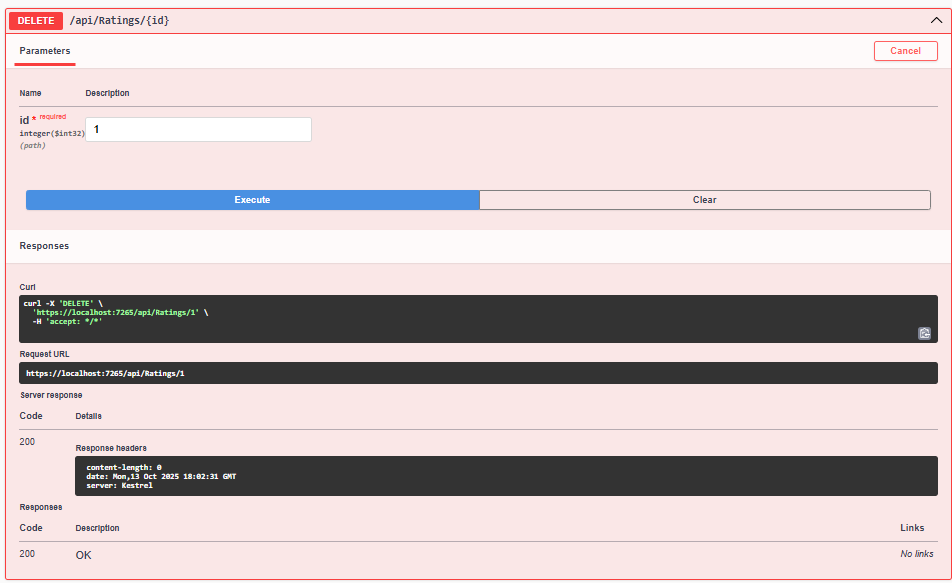
****

**Виконаємо запит на оновлення конкретного рейтингу.**

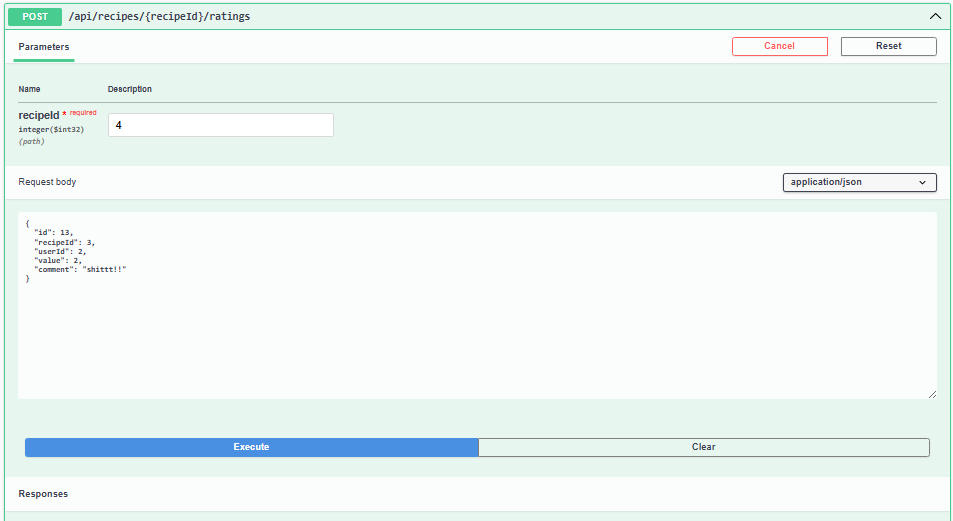
****

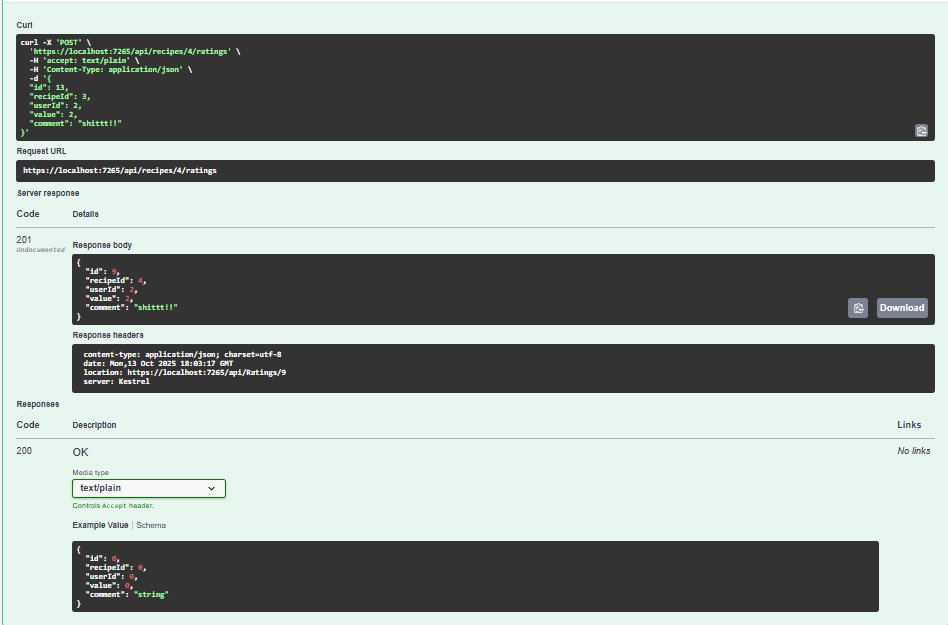
****

**Виконаємо запит на видалення конкретного рейтингу.**

****

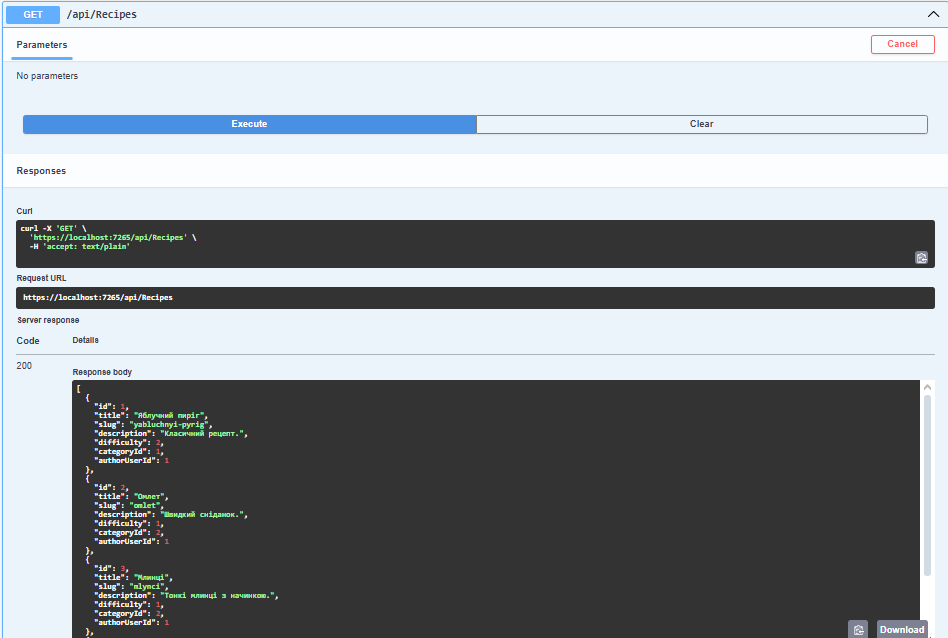
**Виконаємо запит на додавання рейтингу до конкретного рецепту.**

****

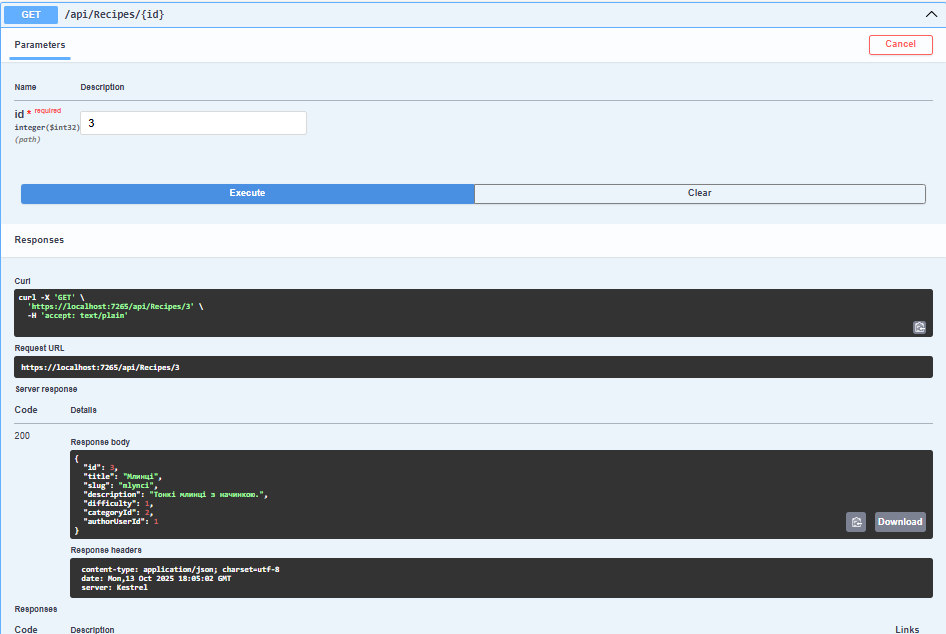
****

**Recipes**

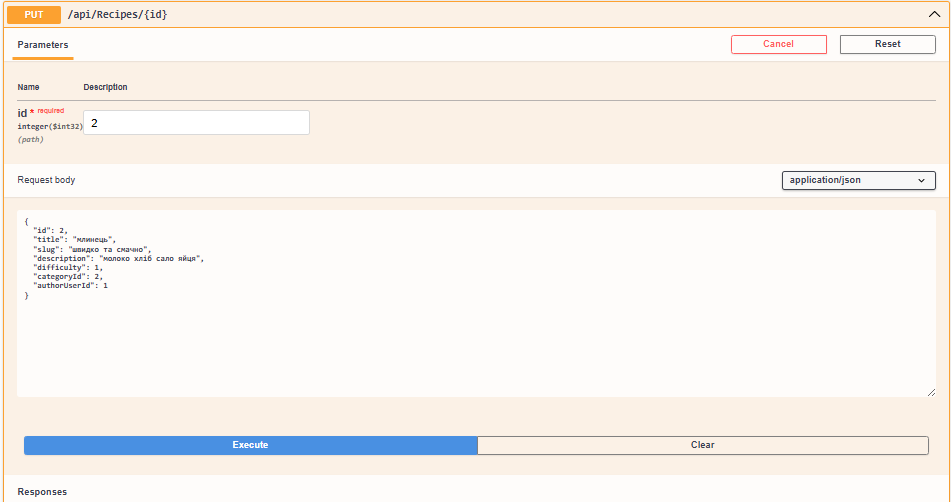
**Виконаємо запит на отримання усіх рецептів.**

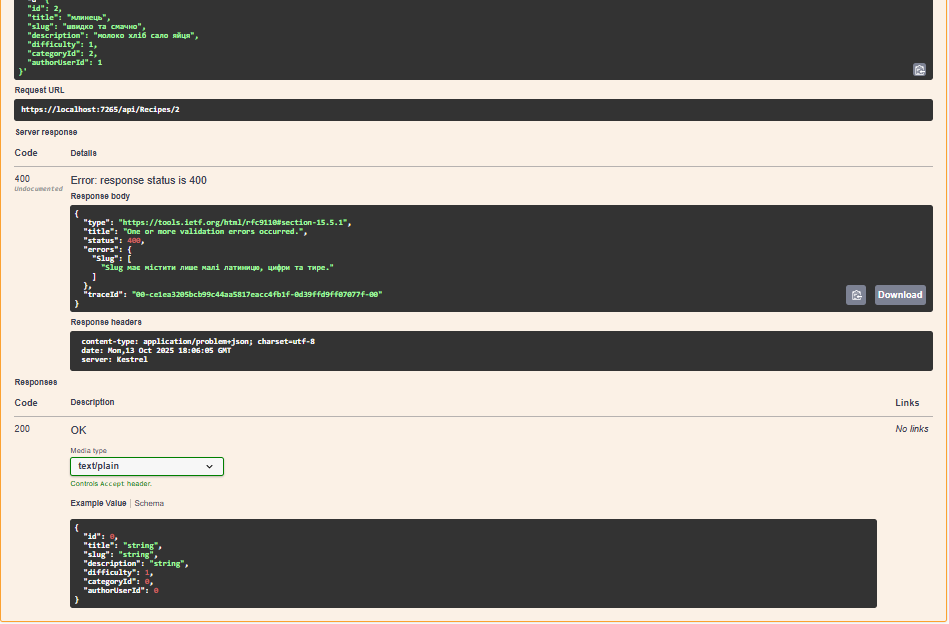
****

**Виконаємо запит на отримання конкретного рецепту.**

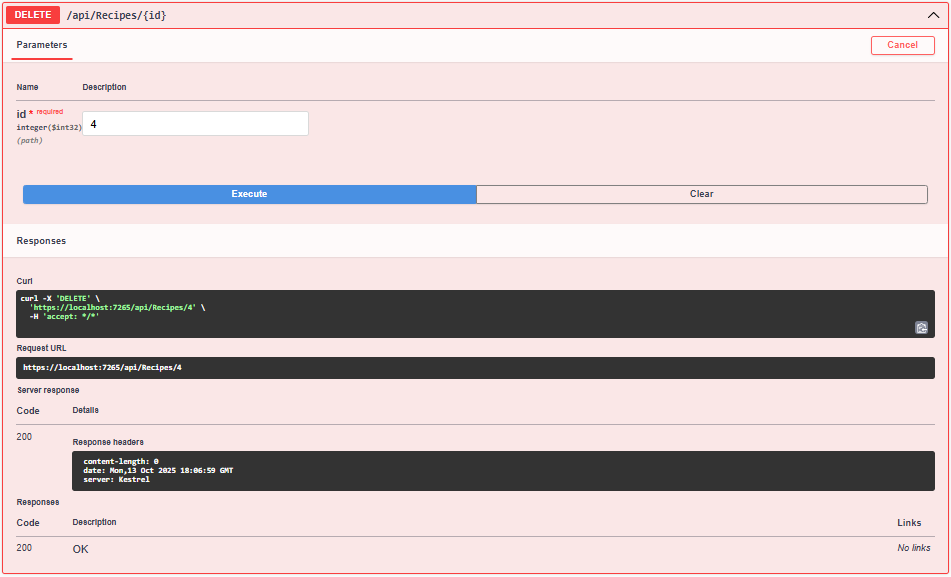
****

**Виконаємо запит на оновлення конкретного рецепту.**

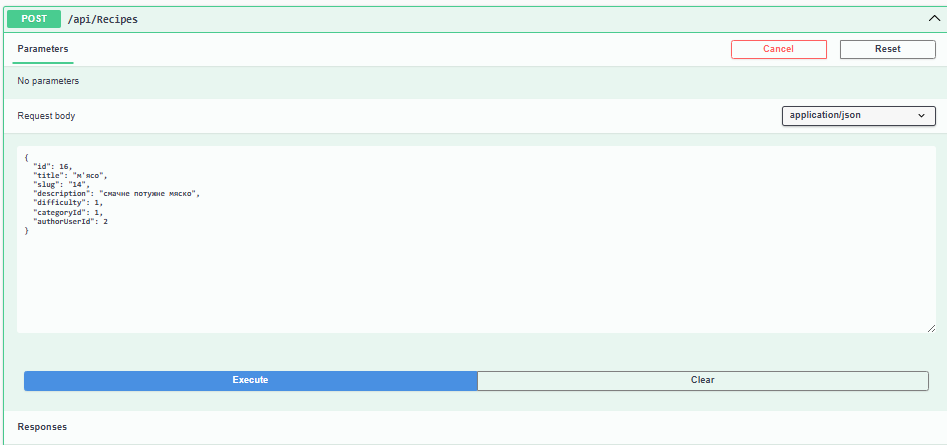
****

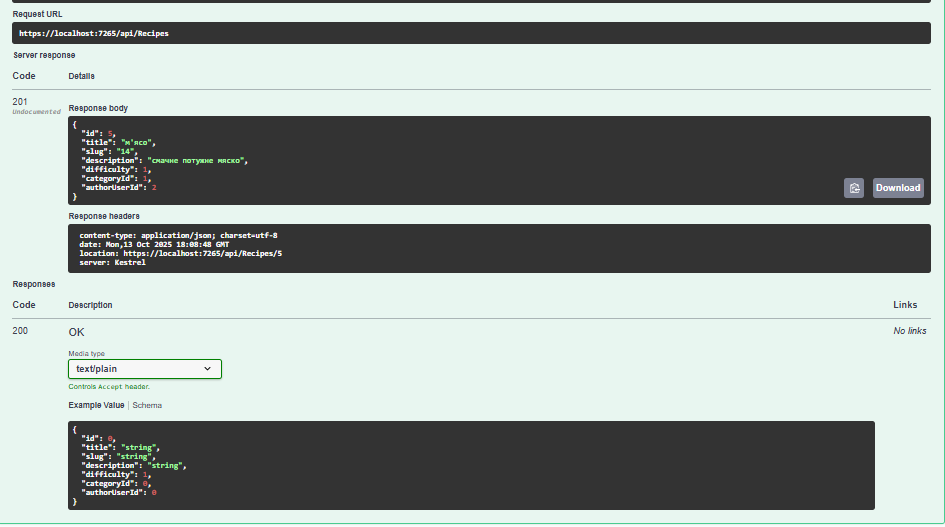
****

**Виконаємо запит на видалення конкретного рецепту.**

****

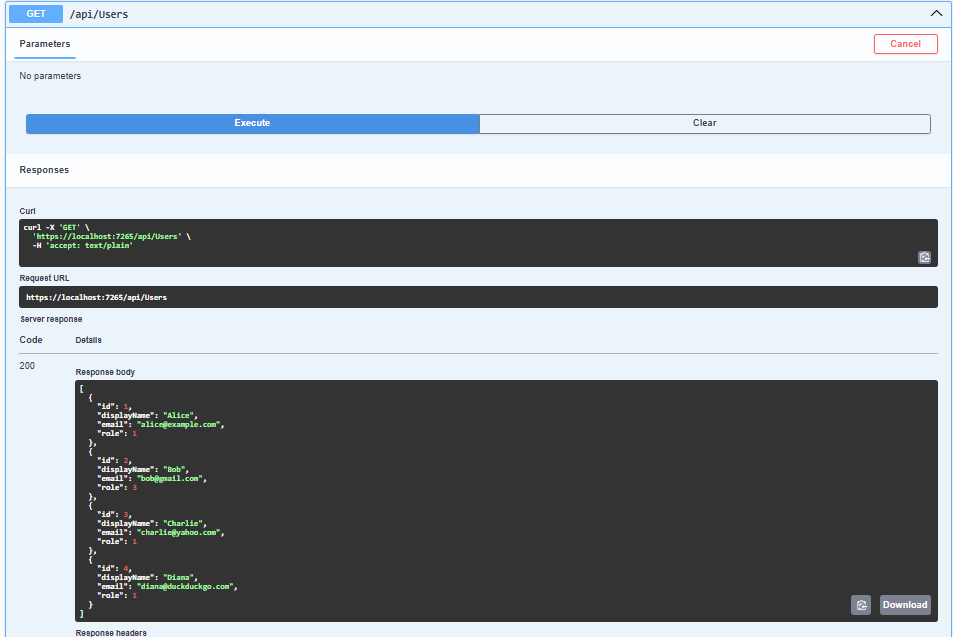
**Виконаємо запит на публікацію рецепта.**

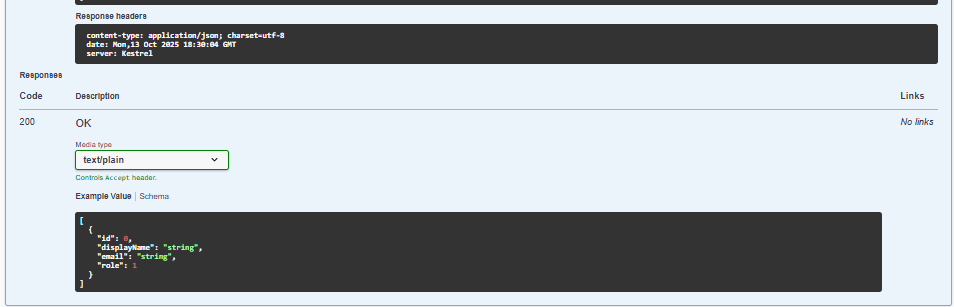
****

****

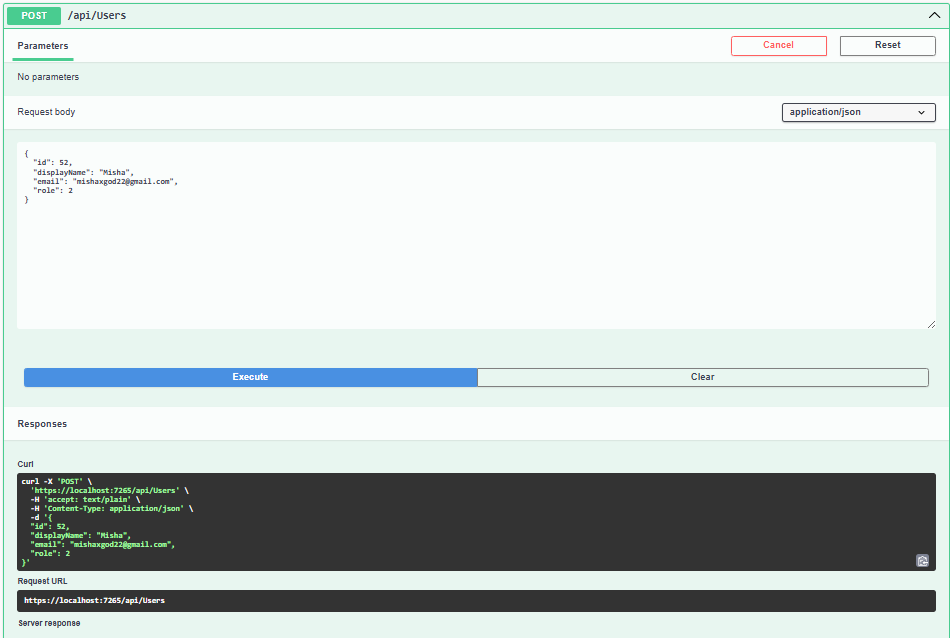
**Users**

**Виконаємо запит на отримання усіх юзерів.**

****

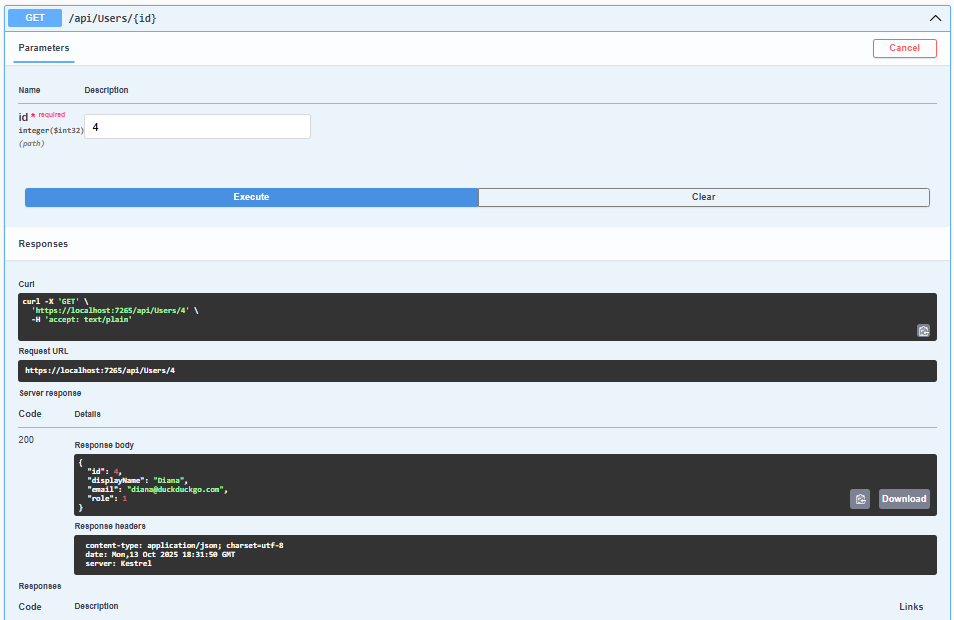
****

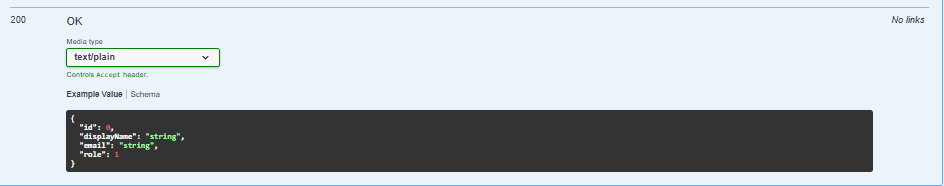
**Виконаємо запит на публікацію юзера.**

****

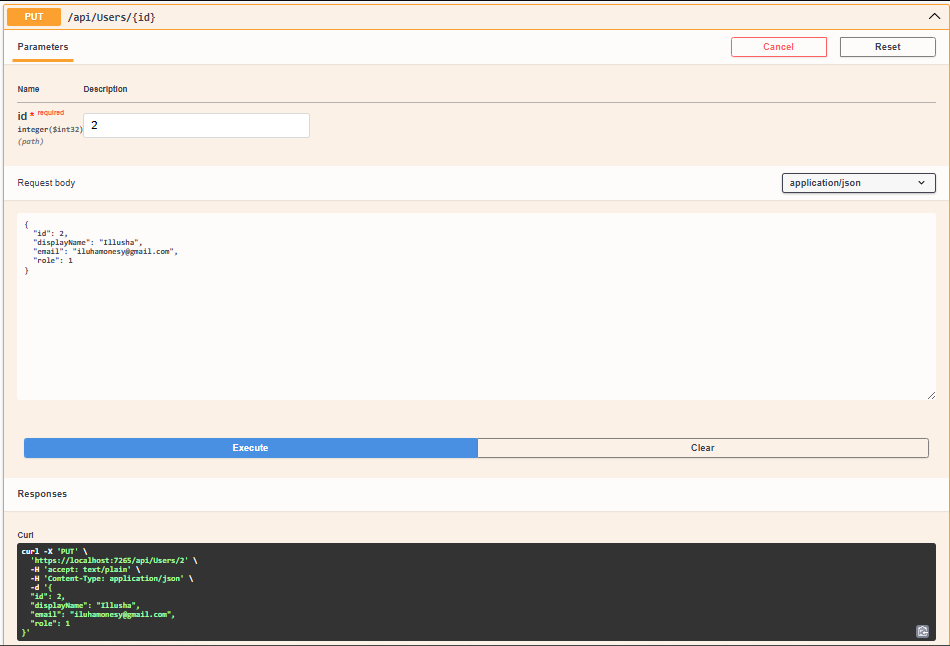
****

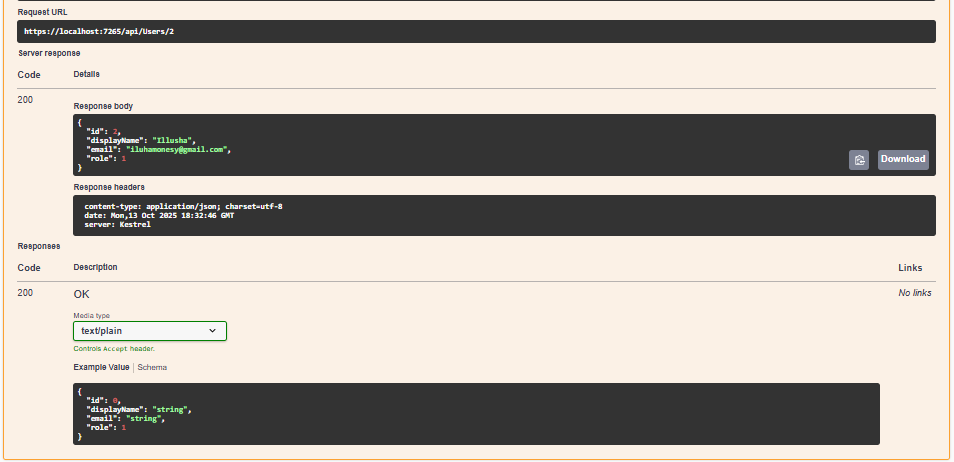
**Виконаємо запит на отримання конкретного юзера.**

****

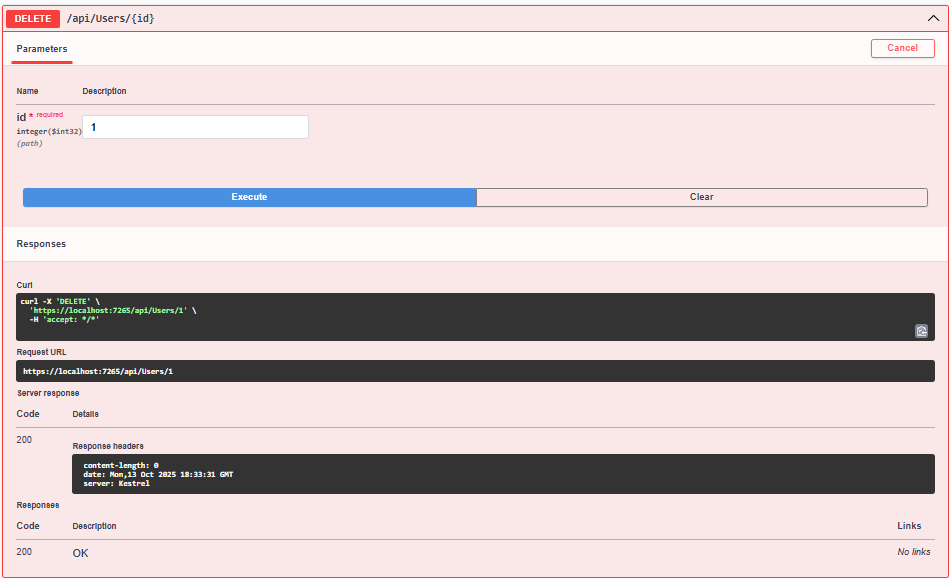
****

**Виконаємо запит на оновлення юзера.**

****

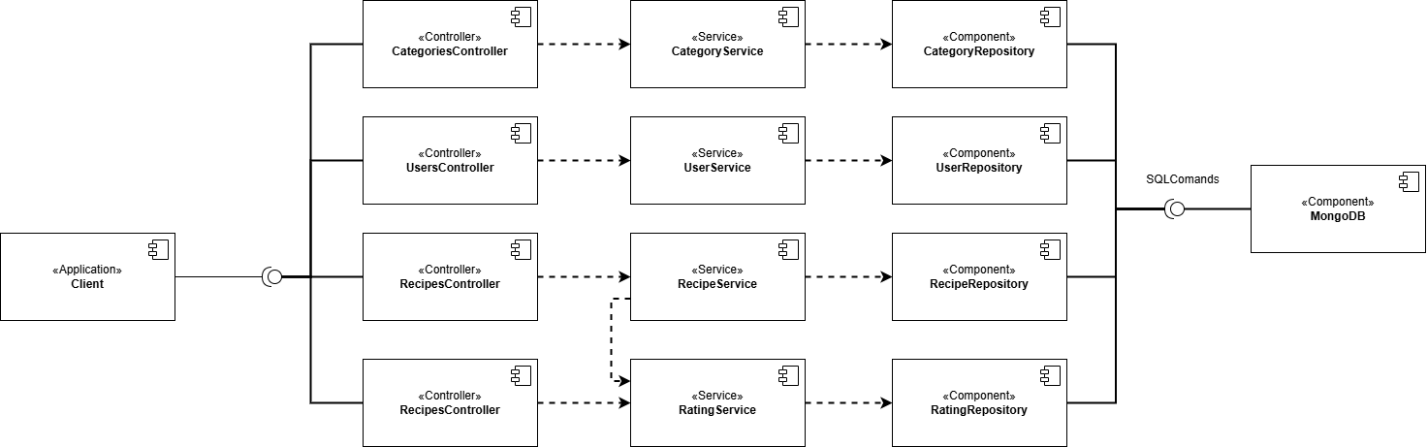
****

**Виконаємо запит на видалення юзера.**

****

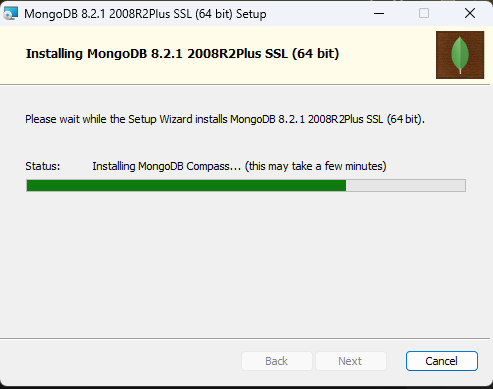
**Завдання №3**

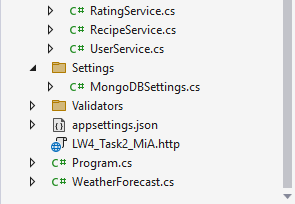
**Створити діаграму компонентів.**

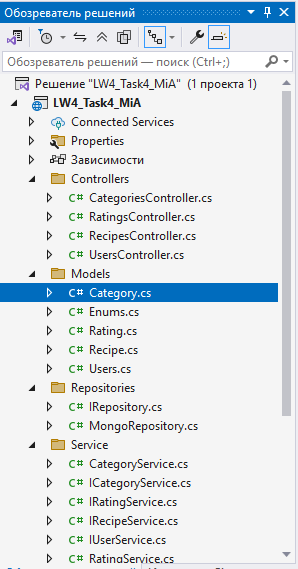
****

**Завдання №4**

Для виконання данного завдання, нам потрібно завантажити службу MongoDB Server. Встановлюємо.



Структура проєкту



**Код програми:**

**Controllers:**

**CategoriesController.cs:**

using LW4\_Task2\_MiA.Models;

using LW4\_Task4\_MiA.Service;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace LW4\_Task2\_MiA.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class CategoriesController : ControllerBase

{

private readonly ICategoryService \_service;

public CategoriesController(ICategoryService service)

{

\_service = service;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Category>>> GetAll()

{

var items = await \_service.GetAllAsync();

return Ok(items);

}

[HttpGet("{id}")]

public async Task<ActionResult<Category>> GetById(string id)

{

var item = await \_service.GetByIdAsync(id);

if (item is null) return NotFound();

return Ok(item);

}

[HttpPost]

public async Task<ActionResult<Category>> Create([FromBody] Category model)

{

if (!ModelState.IsValid) return BadRequest(ModelState);

var created = await \_service.CreateAsync(model);

return CreatedAtAction(nameof(GetById), new { id = created.Id }, created);

}

[HttpPut("{id}")]

public async Task<IActionResult> Update(string id, [FromBody] Category model)

{

if (!ModelState.IsValid) return BadRequest(ModelState);

var ok = await \_service.UpdateAsync(id, model);

if (!ok) return NotFound();

return Ok(model);

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(string id)

{

var ok = await \_service.DeleteAsync(id);

if (!ok) return NotFound();

return Ok();

}

}

}

**RatingsController.cs:**

using LW4\_Task2\_MiA.Models;

using LW4\_Task4\_MiA.Service;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace LW4\_Task2\_MiA.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class RatingsController : ControllerBase

{

private readonly IRatingService \_service;

public RatingsController(IRatingService service)

{

\_service = service;

}

// GET: api/ratings

[HttpGet]

public async Task<ActionResult<IEnumerable<Rating>>> GetAll()

{

var items = await \_service.GetAllAsync();

return Ok(items);

}

// GET: api/ratings/{id}

[HttpGet("{id}")]

public async Task<ActionResult<Rating>> GetById(string id)

{

var item = await \_service.GetByIdAsync(id);

if (item is null) return NotFound();

return Ok(item);

}

// POST: api/ratings

[HttpPost]

public async Task<ActionResult<Rating>> Create([FromBody] Rating model)

{

if (!ModelState.IsValid) return BadRequest(ModelState);

var created = await \_service.CreateAsync(model);

return CreatedAtAction(nameof(GetById), new { id = created.Id }, created);

}

[HttpPut("{id}")]

public async Task<IActionResult> Update(string id, [FromBody] Rating model)

{

if (!ModelState.IsValid) return BadRequest(ModelState);

var ok = await \_service.UpdateAsync(id, model);

if (!ok) return NotFound();

return Ok(model);

}

// DELETE: api/ratings/{id}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(string id)

{

var ok = await \_service.DeleteAsync(id);

if (!ok) return NotFound();

return Ok();

}

}

}

**RecipesController.cs:**

using LW4\_Task2\_MiA.Models;

using LW4\_Task4\_MiA.Service;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace LW4\_Task2\_MiA.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class RecipesController : ControllerBase

{

private readonly IRecipeService \_service;

public RecipesController(IRecipeService service)

{

\_service = service;

}

// GET: api/recipes

[HttpGet]

public async Task<ActionResult<IEnumerable<Recipe>>> GetAll()

{

var items = await \_service.GetAllAsync();

return Ok(items);

}

// GET: api/recipes/6750d4...

[HttpGet("{id}")]

public async Task<ActionResult<Recipe>> GetById(string id)

{

var item = await \_service.GetByIdAsync(id);

if (item is null) return NotFound();

return Ok(item);

}

// POST: api/recipes

[HttpPost]

public async Task<ActionResult<Recipe>> Create([FromBody] Recipe model)

{

if (!ModelState.IsValid) return BadRequest(ModelState);

var created = await \_service.CreateAsync(model);

return CreatedAtAction(nameof(GetById), new { id = created.Id }, created);

}

// PUT: api/recipes/{id}

[HttpPut("{id}")]

public async Task<IActionResult> Update(string id, [FromBody] Recipe model)

{

if (!ModelState.IsValid) return BadRequest(ModelState);

var ok = await \_service.UpdateAsync(id, model);

if (!ok) return NotFound();

return Ok(model);

}

// DELETE: api/recipes/{id}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(string id)

{

var ok = await \_service.DeleteAsync(id);

if (!ok) return NotFound();

return Ok();

}

}

}

**UsersController.cs:**

using LW4\_Task2\_MiA.Models;

using LW4\_Task4\_MiA.Service;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace LW4\_Task2\_MiA.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class UsersController : ControllerBase

{

private readonly IUserService \_service;

public UsersController(IUserService service)

{

\_service = service;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<User>>> GetAll()

{

var items = await \_service.GetAllAsync();

return Ok(items);

}

[HttpGet("{id}")]

public async Task<ActionResult<User>> GetById(string id)

{

var item = await \_service.GetByIdAsync(id);

if (item is null) return NotFound();

return Ok(item);

}

[HttpPost]

public async Task<ActionResult<User>> Create([FromBody] User model)

{

if (!ModelState.IsValid) return BadRequest(ModelState);

var created = await \_service.CreateAsync(model);

return CreatedAtAction(nameof(GetById), new { id = created.Id }, created);

}

[HttpPut("{id}")]

public async Task<IActionResult> Update(string id, [FromBody] User model)

{

if (!ModelState.IsValid) return BadRequest(ModelState);

var ok = await \_service.UpdateAsync(id, model);

if (!ok) return NotFound();

return Ok(model);

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(string id)

{

var ok = await \_service.DeleteAsync(id);

if (!ok) return NotFound();

return Ok();

}

}

}

**Models:**

**Category.cs:**

using System.ComponentModel.DataAnnotations;

using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

namespace LW4\_Task2\_MiA.Models

{

public class Category

{

[BsonId]

[BsonRepresentation(BsonType.ObjectId)]

public string Id { get; set; }

[Required, StringLength(40, MinimumLength = 2)]

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

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:**

using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

namespace LW4\_Task2\_MiA.Models

{

public class Rating

{

[BsonId]

[BsonRepresentation(BsonType.ObjectId)]

public string Id { get; set; }

[BsonRepresentation(BsonType.ObjectId)]

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

[BsonRepresentation(BsonType.ObjectId)]

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

public int Value { get; set; }

public string? Comment { get; set; }

}

}

**Recipe.cs:**

using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

namespace LW4\_Task2\_MiA.Models

{

public class Recipe

{

[BsonId]

[BsonRepresentation(BsonType.ObjectId)]

public string 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;

[BsonRepresentation(BsonType.ObjectId)]

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

[BsonRepresentation(BsonType.ObjectId)]

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

}

}

**Users.cs:**

using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

namespace LW4\_Task2\_MiA.Models

{

public class User

{

[BsonId]

[BsonRepresentation(BsonType.ObjectId)]

public string Id { get; set; }

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

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

public UserRole Role { get; set; } = UserRole.Regular;

}

}

**Repositories:**

**IRepository.cs:**

using System.Collections.Generic;

using System.Threading.Tasks;

namespace LW4\_Task2\_MiA.Repositories

{

public interface IRepository<T>

{

Task<IReadOnlyCollection<T>> GetAllAsync();

Task<T?> GetByIdAsync(string id);

Task CreateAsync(T entity);

Task<bool> UpdateAsync(string id, T entity);

Task<bool> DeleteAsync(string id);

}

}

**MongoRepository.cs:**

using MongoDB.Driver;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace LW4\_Task2\_MiA.Repositories

{

public class MongoRepository<T> : IRepository<T> where T : class

{

private readonly IMongoCollection<T> \_collection;

public MongoRepository(IMongoDatabase database, string collectionName)

{

\_collection = database.GetCollection<T>(collectionName);

}

public async Task<IReadOnlyCollection<T>> GetAllAsync()

{

var results = await \_collection.Find(\_ => true).ToListAsync();

return results;

}

public async Task<T?> GetByIdAsync(string id)

{

var filter = Builders<T>.Filter.Eq("Id", id);

return await \_collection.Find(filter).FirstOrDefaultAsync();

}

public async Task CreateAsync(T entity)

{

await \_collection.InsertOneAsync(entity);

}

public async Task<bool> UpdateAsync(string id, T entity)

{

var filter = Builders<T>.Filter.Eq("Id", id);

var result = await \_collection.ReplaceOneAsync(filter, entity);

return result.ModifiedCount > 0;

}

public async Task<bool> DeleteAsync(string id)

{

var filter = Builders<T>.Filter.Eq("Id", id);

var result = await \_collection.DeleteOneAsync(filter);

return result.DeletedCount > 0;

}

}

}

**Service:**

**CategoryService.cs:**

using LW4\_Task2\_MiA.Models;

using LW4\_Task2\_MiA.Repositories;

namespace LW4\_Task4\_MiA.Service

{

public class CategoryService : ICategoryService

{

private readonly IRepository<Category> \_repo;

public CategoryService(IRepository<Category> repo)

{

\_repo = repo;

}

public async Task<IEnumerable<Category>> GetAllAsync()

=> await \_repo.GetAllAsync();

public async Task<Category?> GetByIdAsync(string id)

=> await \_repo.GetByIdAsync(id);

public async Task<Category> CreateAsync(Category category)

{

category.Id = null;

await \_repo.CreateAsync(category);

return category;

}

public async Task<bool> UpdateAsync(string id, Category category)

{

var existing = await \_repo.GetByIdAsync(id);

if (existing is null) return false;

//фіксуємо id

category.Id = id;

return await \_repo.UpdateAsync(id, category);

}

public async Task<bool> DeleteAsync(string id)

{

var existing = await \_repo.GetByIdAsync(id);

if (existing is null) return false;

return await \_repo.DeleteAsync(id);

}

}

}

**ICategoryService.cs:**

using LW4\_Task2\_MiA.Models;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace LW4\_Task4\_MiA.Service

{

public interface ICategoryService

{

Task<IEnumerable<Category>> GetAllAsync();

Task<Category?> GetByIdAsync(string id);

Task<Category> CreateAsync(Category category);

Task<bool> UpdateAsync(string id, Category category);

Task<bool> DeleteAsync(string id);

}

}

**IRatingService.cs:**

using LW4\_Task2\_MiA.Models;

namespace LW4\_Task4\_MiA.Service

{

public interface IRatingService

{

Task<IEnumerable<Rating>> GetAllAsync();

Task<Rating?> GetByIdAsync(string id);

Task<Rating> CreateAsync(Rating rating);

Task<bool> DeleteAsync(string id);

Task<bool> UpdateAsync(string id, Rating rating);

//щоб при видаленні рецепта прибрати рейтинги

Task DeleteByRecipeIdAsync(string recipeId);

}

}

**IRecipeService.cs:**

using LW4\_Task2\_MiA.Models;

namespace LW4\_Task4\_MiA.Service

{

public interface IRecipeService

{

Task<IEnumerable<Recipe>> GetAllAsync();

Task<Recipe?> GetByIdAsync(string id);

Task<Recipe> CreateAsync(Recipe recipe);

Task<bool> UpdateAsync(string id, Recipe recipe);

Task<bool> DeleteAsync(string id);

}

}

**IUserService.cs:**

using LW4\_Task2\_MiA.Models;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace LW4\_Task4\_MiA.Service

{

public interface IUserService

{

Task<IEnumerable<User>> GetAllAsync();

Task<User?> GetByIdAsync(string id);

Task<User> CreateAsync(User user);

Task<bool> UpdateAsync(string id, User user);

Task<bool> DeleteAsync(string id);

Task<User?> GetByEmailAsync(string email);

}

}

**RatingService.cs:**

using LW4\_Task2\_MiA.Models;

using LW4\_Task2\_MiA.Repositories;

using MongoDB.Driver;

namespace LW4\_Task4\_MiA.Service

{

public class RatingService : IRatingService

{

private readonly IRepository<Rating> \_repo;

private readonly IMongoCollection<Rating> \_collection;

public RatingService(IMongoDatabase db, IRepository<Rating> repo)

{

\_repo = repo;

\_collection = db.GetCollection<Rating>("Ratings");

}

public async Task<IEnumerable<Rating>> GetAllAsync()

=> await \_repo.GetAllAsync();

public async Task<Rating?> GetByIdAsync(string id)

=> await \_repo.GetByIdAsync(id);

public async Task<Rating> CreateAsync(Rating rating)

{

// не приймаємо id від клієнта

rating.Id = null;

await \_repo.CreateAsync(rating);

return rating;

}

public async Task<bool> UpdateAsync(string id, Rating rating)

{

var existing = await \_repo.GetByIdAsync(id);

if (existing is null) return false;

rating.Id = id;

return await \_repo.UpdateAsync(id, rating);

}

public async Task<bool> DeleteAsync(string id)

=> await \_repo.DeleteAsync(id);

public async Task DeleteByRecipeIdAsync(string recipeId)

{

var filter = Builders<Rating>.Filter.Eq(r => r.RecipeId, recipeId);

await \_collection.DeleteManyAsync(filter);

}

}

}

**RecipeService.cs:**

using LW4\_Task2\_MiA.Models;

using LW4\_Task2\_MiA.Repositories;

namespace LW4\_Task4\_MiA.Service

{

public class RecipeService : IRecipeService

{

private readonly IRepository<Recipe> \_recipes;

private readonly IRatingService \_ratings;

public RecipeService(IRepository<Recipe> recipes, IRatingService ratings)

{

\_recipes = recipes;

\_ratings = ratings;

}

public async Task<IEnumerable<Recipe>> GetAllAsync()

=> await \_recipes.GetAllAsync();

public async Task<Recipe?> GetByIdAsync(string id)

=> await \_recipes.GetByIdAsync(id);

public async Task<Recipe> CreateAsync(Recipe recipe)

{

recipe.Id = null;

await \_recipes.CreateAsync(recipe);

return recipe;

}

public async Task<bool> UpdateAsync(string id, Recipe recipe)

{

var existing = await \_recipes.GetByIdAsync(id);

if (existing is null) return false;

// фіксуємо id

recipe.Id = id;

return await \_recipes.UpdateAsync(id, recipe);

}

public async Task<bool> DeleteAsync(string id)

{

var existing = await \_recipes.GetByIdAsync(id);

if (existing is null) return false;

var ok = await \_recipes.DeleteAsync(id);

if (ok)

{

// прибираємо рейтинги цього рецепта

await \_ratings.DeleteByRecipeIdAsync(id);

}

return ok;

}

}

}

**UserService.cs:**

using LW4\_Task2\_MiA.Models;

using LW4\_Task2\_MiA.Repositories;

using MongoDB.Driver;

namespace LW4\_Task4\_MiA.Service

{

public class UserService : IUserService

{

private readonly IRepository<User> \_repo;

private readonly IMongoCollection<User> \_collection;

public UserService(IRepository<User> repo, IMongoDatabase db)

{

\_repo = repo;

\_collection = db.GetCollection<User>("Users");

}

public async Task<IEnumerable<User>> GetAllAsync()

=> await \_repo.GetAllAsync();

public async Task<User?> GetByIdAsync(string id)

=> await \_repo.GetByIdAsync(id);

public async Task<User> CreateAsync(User user)

{

// не беремо id від клієнта

user.Id = null;

// (опціонально) перевірка емейлу

if (!string.IsNullOrWhiteSpace(user.Email))

{

var filter = Builders<User>.Filter.Eq(u => u.Email, user.Email);

var exists = await \_collection.Find(filter).FirstOrDefaultAsync();

if (exists is not null)

{

// можеш або кинути помилку, або повернути існуючого

return exists;

}

}

await \_repo.CreateAsync(user);

return user;

}

public async Task<bool> UpdateAsync(string id, User user)

{

var existing = await \_repo.GetByIdAsync(id);

if (existing is null) return false;

user.Id = id;

return await \_repo.UpdateAsync(id, user);

}

public async Task<bool> DeleteAsync(string id)

{

var existing = await \_repo.GetByIdAsync(id);

if (existing is null) return false;

return await \_repo.DeleteAsync(id);

}

public async Task<User?> GetByEmailAsync(string email)

{

var filter = Builders<User>.Filter.Eq(u => u.Email, email);

return await \_collection.Find(filter).FirstOrDefaultAsync();

}

}

}

**Settings:**

**MongoDBSettings.cs:**

using System;

namespace LW4\_Task4\_MiA.Settings

{

public class MongoDbSettings

{

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

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

}

}

**Validators:**

**RatingValidators.cs:**

using LW4\_Task2\_MiA.Models;

using FluentValidation;

using System;

namespace LW4\_Task2\_MiA.Validators

{

public class RatingValidator : AbstractValidator<Rating>

{

public RatingValidator()

{

// обов’язково вказати рецепт

RuleFor(x => x.RecipeId).NotEmpty().Length(24).WithMessage("RecipeId must be a valid Mongo ObjectId (24 chars).");

// обов’язково вказати користувача

RuleFor(x => x.UserId).NotEmpty().Length(24).WithMessage("UserId must be a valid Mongo ObjectId (24 chars).");

// рейтинг від 1 до 5

RuleFor(x => x.Value).InclusiveBetween(1, 5);

// коментар опціональний, але не довший

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()

{

// назва

RuleFor(x => x.Title).NotEmpty().MinimumLength(3).MaximumLength(80);

// slug тільки малі букви, цифри, тире

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

// опис

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

// складність

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

//після переходу на Mongo — це рядкові ObjectId

RuleFor(x => x.CategoryId).NotEmpty().Length(24).WithMessage("CategoryId має бути валідним Mongo ObjectId (24 символи).");

RuleFor(x => x.AuthorUserId).NotEmpty().Length(24).WithMessage("AuthorUserId має бути валідним Mongo ObjectId (24 символи).");

}

}

}

**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.Repositories;

using LW4\_Task4\_MiA.Settings;

using Microsoft.Extensions.Options;

using MongoDB.Driver;

using LW4\_Task4\_MiA.Service;

var builder = WebApplication.CreateBuilder(args);

// 1. Налаштування Mongo

builder.Services.Configure<MongoDbSettings>(

builder.Configuration.GetSection("MongoDb"));

builder.Services.AddSingleton<IMongoClient>(sp =>

{

var settings = sp.GetRequiredService<IOptions<MongoDbSettings>>().Value;

return new MongoClient(settings.ConnectionString);

});

builder.Services.AddSingleton<IMongoDatabase>(sp =>

{

var settings = sp.GetRequiredService<IOptions<MongoDbSettings>>().Value;

var client = sp.GetRequiredService<IMongoClient>();

return client.GetDatabase(settings.DatabaseName);

});

// 2. Репозиторії

builder.Services.AddScoped<IRepository<LW4\_Task2\_MiA.Models.Category>>(sp =>

new MongoRepository<LW4\_Task2\_MiA.Models.Category>(

sp.GetRequiredService<IMongoDatabase>(),

"Categories"));

builder.Services.AddScoped<IRepository<LW4\_Task2\_MiA.Models.User>>(sp =>

new MongoRepository<LW4\_Task2\_MiA.Models.User>(

sp.GetRequiredService<IMongoDatabase>(),

"Users"));

builder.Services.AddScoped<IRepository<LW4\_Task2\_MiA.Models.Recipe>>(sp =>

new MongoRepository<LW4\_Task2\_MiA.Models.Recipe>(

sp.GetRequiredService<IMongoDatabase>(),

"Recipes"));

builder.Services.AddScoped<IRepository<LW4\_Task2\_MiA.Models.Rating>>(sp =>

new MongoRepository<LW4\_Task2\_MiA.Models.Rating>(

sp.GetRequiredService<IMongoDatabase>(),

"Ratings"));

// 3. Сервіси

builder.Services.AddScoped<ICategoryService, CategoryService>();

builder.Services.AddScoped<IRatingService, RatingService>();

builder.Services.AddScoped<IRecipeService, RecipeService>();

builder.Services.AddScoped<IUserService, UserService>();

builder.Services.AddControllers()

.AddJsonOptions(o => { })

.ConfigureApiBehaviorOptions(o => { });

builder.Services.Configure<Microsoft.AspNetCore.Mvc.MvcOptions>(options =>

{

options.SuppressImplicitRequiredAttributeForNonNullableReferenceTypes = true;

});

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddFluentValidationAutoValidation();

builder.Services.AddValidatorsFromAssemblyContaining<LW4\_Task2\_MiA.Validators.RecipeValidator>();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.MapControllers();

app.Run();

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