**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.

**Завдання №6**

**Завдання 6**

**Ознайомитися з кроками виконання роботи |**[**Check out**](https://docs-mai.vercel.app/docs/developmement/auth_impl)**.**

1. Додати аутентифікацію через JWT-токен
   * Встановити необхідні пакети: Microsoft.AspNetCore.Authentication.JwtBearer, System.IdentityModel.Tokens.Jwt
   * Додати у Program.cs налаштування JWT-аутентифікації. Визначити JwtSettings (секрет, issuer, audience, час життя токену) та додати AddAuthentication().AddJwtBearer(...).
   * Створити сервіс для генерації токенів (JwtService або TokenService), який створює токен із набором Claims (Id, Email, Role тощо) та повертає JWT-рядок клієнту.
   * Додати схему авторизації у Swagger, щоб можна було вводити токен для захищених ендпойнтів.
2. **Створити контролер для перевірки доступу**
   * Створити контролер, наприклад TestController.
   * Додати метод, захищений атрибутом [Authorize].
   * Усередині методу зчитати дані користувача з Claims
   * Повернути з контролера інформацію про поточного користувача (Id, Email).

**На оцінку 4:**

1. **Реалізувати механізм оновлення токену (Refresh Token)**

* Додати до моделі користувача поле RefreshToken та RefreshTokenExpiryTime.
* Після логіну згенерувати AccessToken і RefreshToken та зберегти у базі даних.
* Додати ендпоінт POST /api/auth/refresh
* Забезпечити можливість відкликання старих токенів при повторній автентифікації.

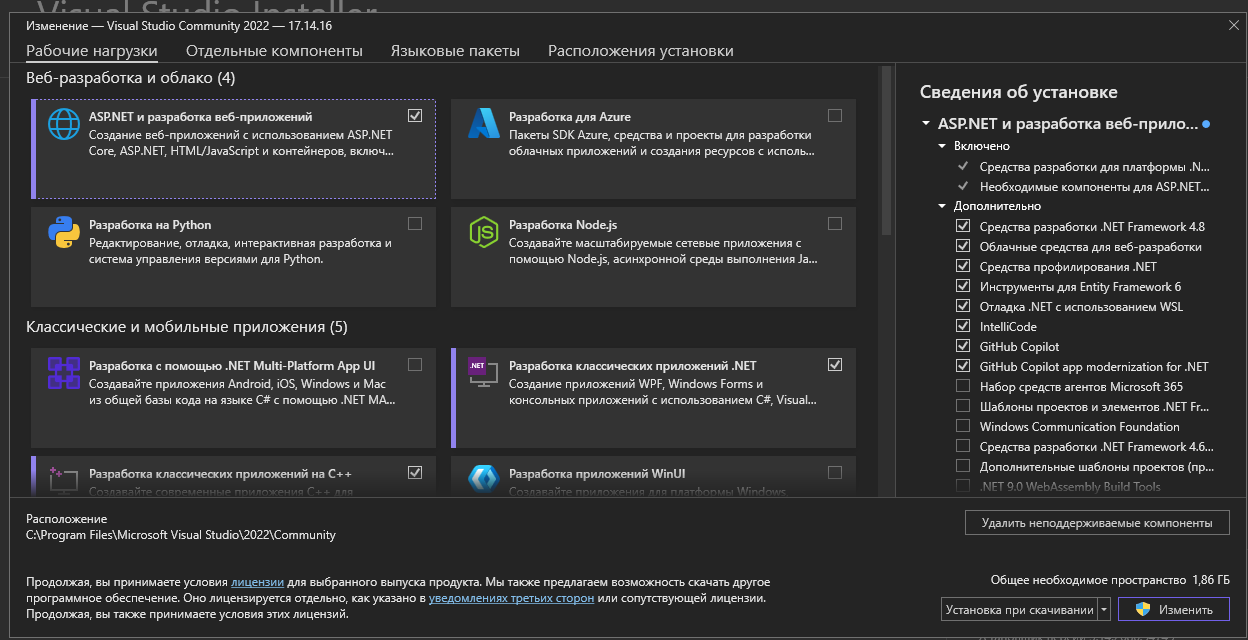
**На оцінку 5:**

1. **Додати авторизацію користувачів із ролями**
   * Створити enum UserRoles (наприклад Admin = 1, Manager = 2, User = 4)
   * Додати поле Role (int) до моделі користувача.
   * Додати атрибути [Authorize(Roles = "Admin")], [Authorize(Roles = "Manager,Admin")] у контролерах або діях.
   * Перевірити доступи до різних маршрутів у Swagger — вхід під різними користувачами.

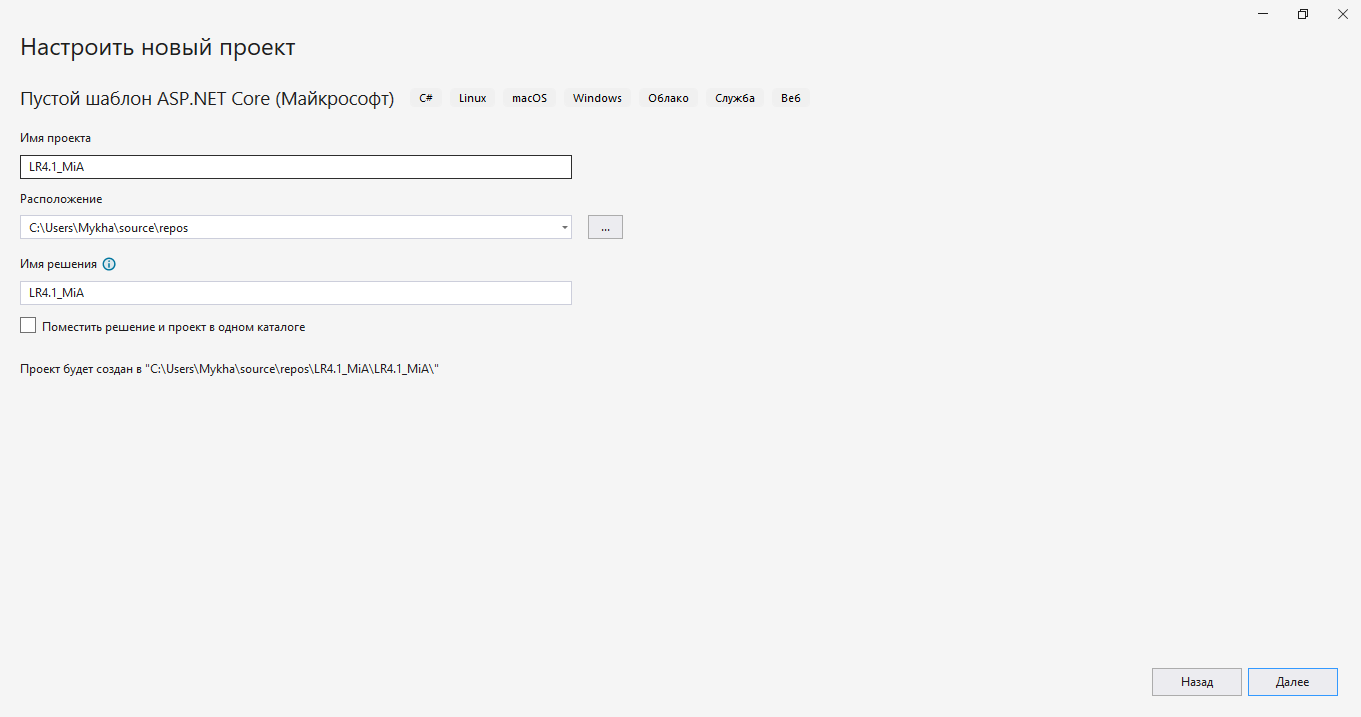
**Хід роботи**

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

1. Під час встановлення Visual Studio обираємо ASP.NET



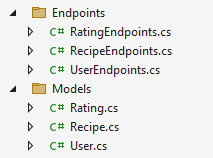
1. Перейдемо до створення проєкту. Для цього знаходимо проєкт ASP.NET Core empty.



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";

});

}

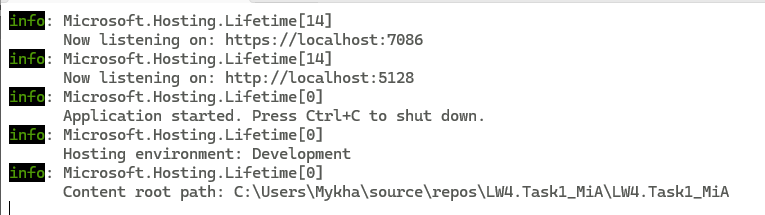
app.MapRecipeEndpoints();

app.MapUserEndpoints();

app.MapRatingEndpoints();

app.Run();

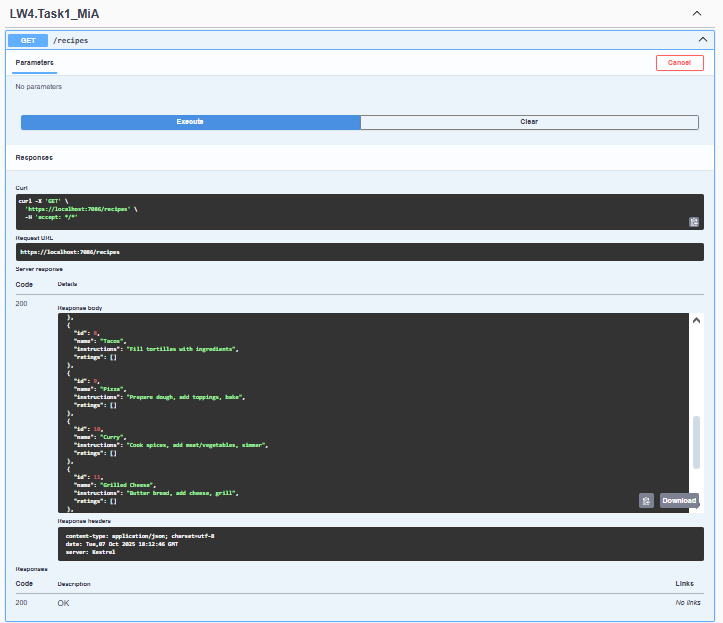
**Результат роботи:**

****

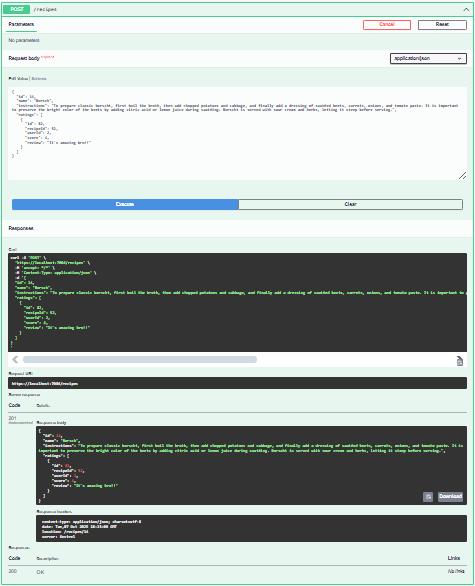
Програма показує те, що нам доступна вся інформація по посиланню.

**Запити**

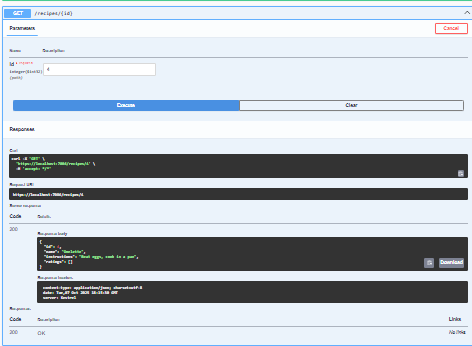
**Запит на отримання рецептів**



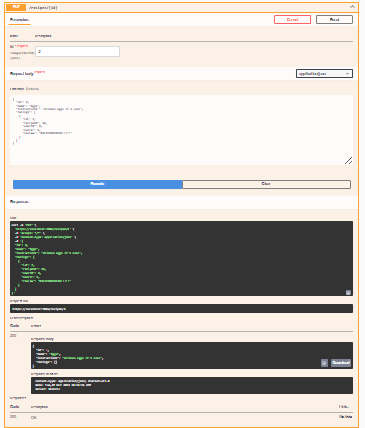
**Запит на публікацію рецепта**



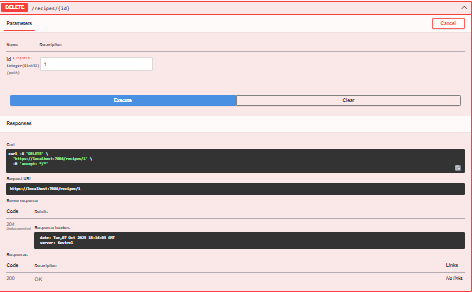
**Запит на отримання конкретного рецепта**

****

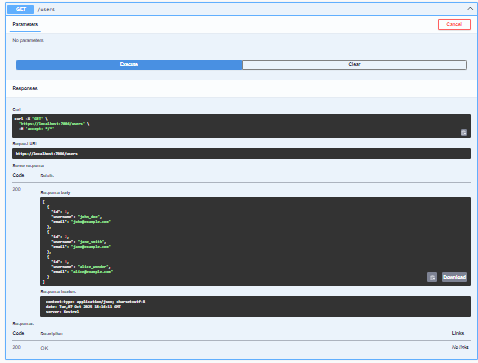
**Запит на оновлення рецепта**

****

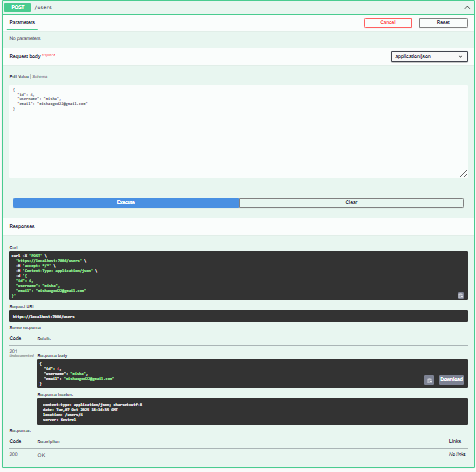
**Запит на видалення рецепта**

****

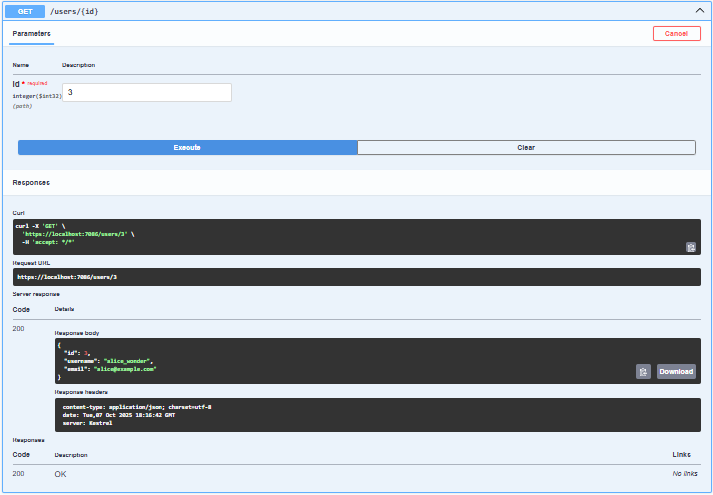
**Запит на отримання користувачів**

****

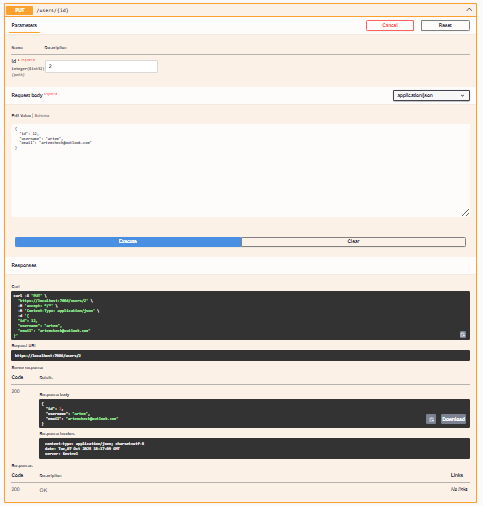
**Запит на додавання користувача**

****

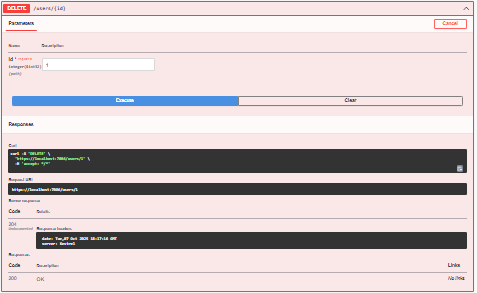
**Запит на отримання конкретного користувача**

****

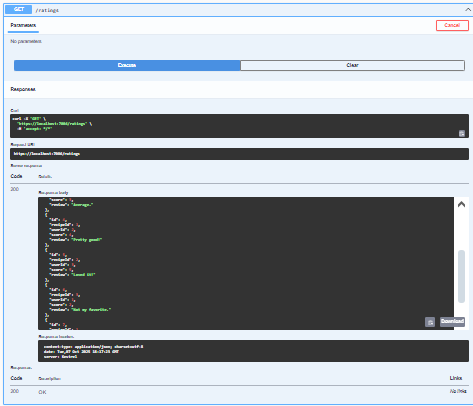
**Запит на оновлення користувача**

****

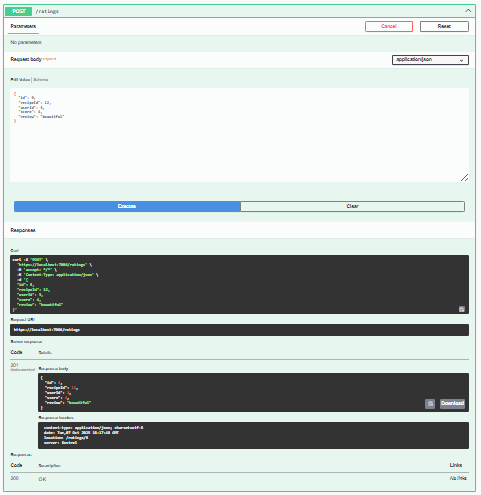
**Запит на видалення користувача**

****

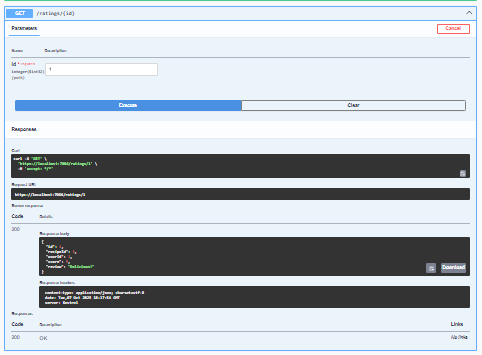
**Запит на отримання оцінок**

****

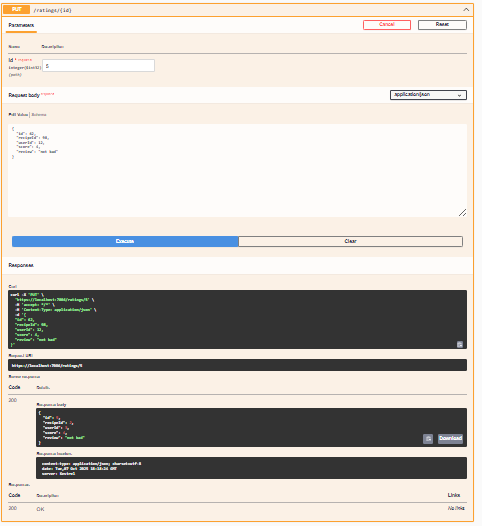
**Запит на додавання оцінки**

****

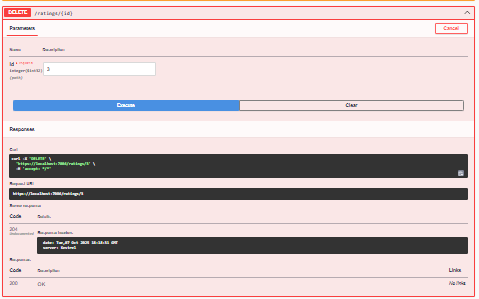
**Запит на отримання конкретної оцінки**

****

**Запит на оновлення оцінки**



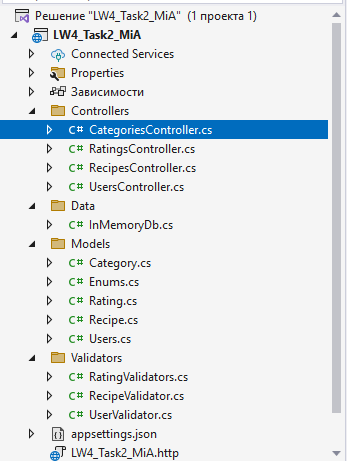
**Запит на видалення оцінки**



**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();

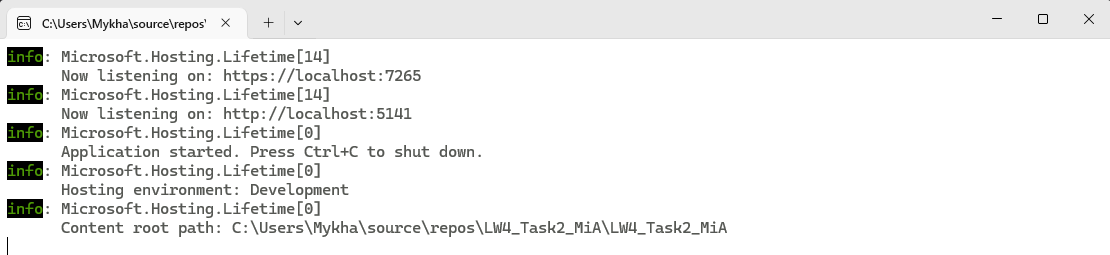
}

app.UseHttpsRedirection();

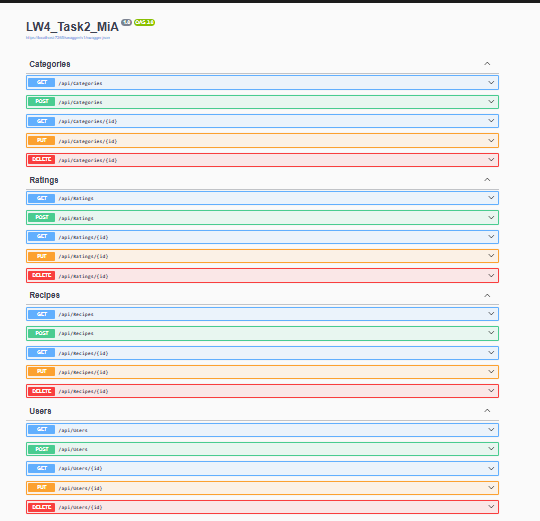
app.MapControllers();

app.Run();

**Результат програми**



Програма працює. Переходимо до Swagger.

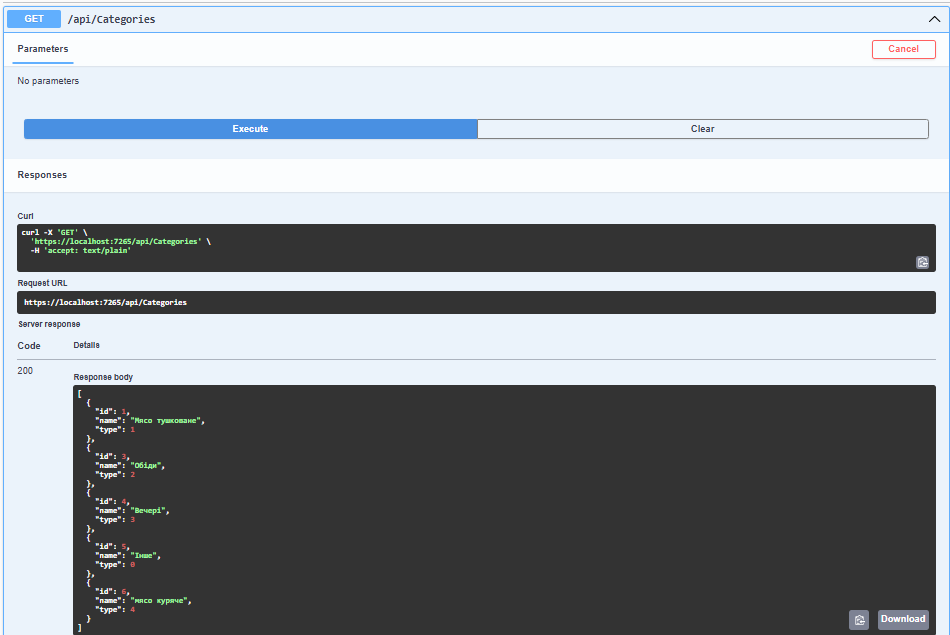


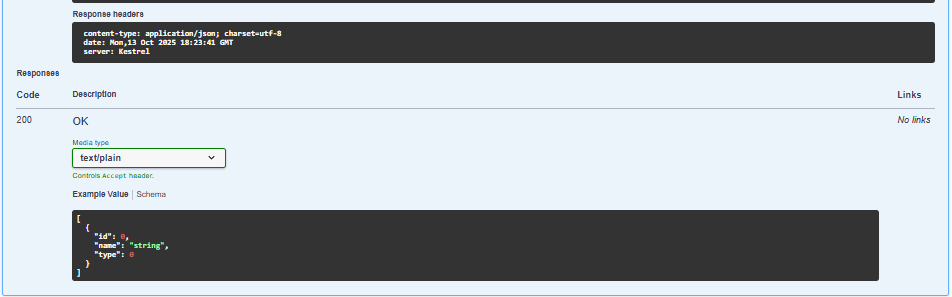
В програмі є 4 контролера: Categories, Ratings, Recipes, Users.

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

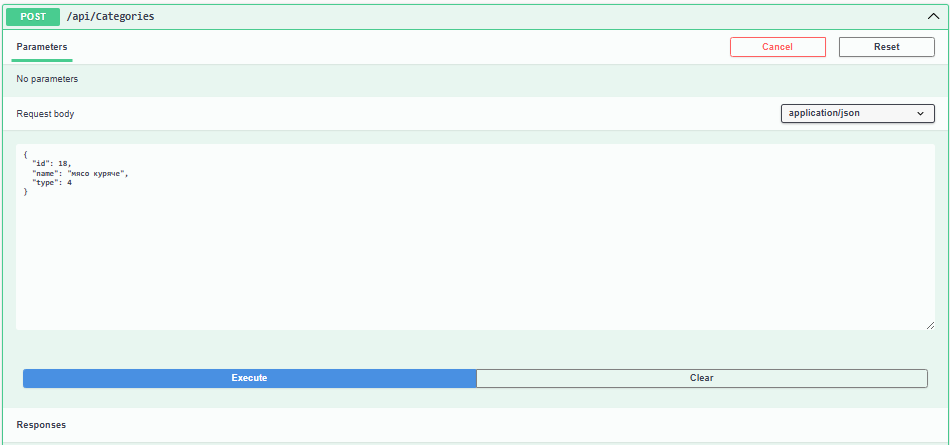
**Почнемо з Categories**

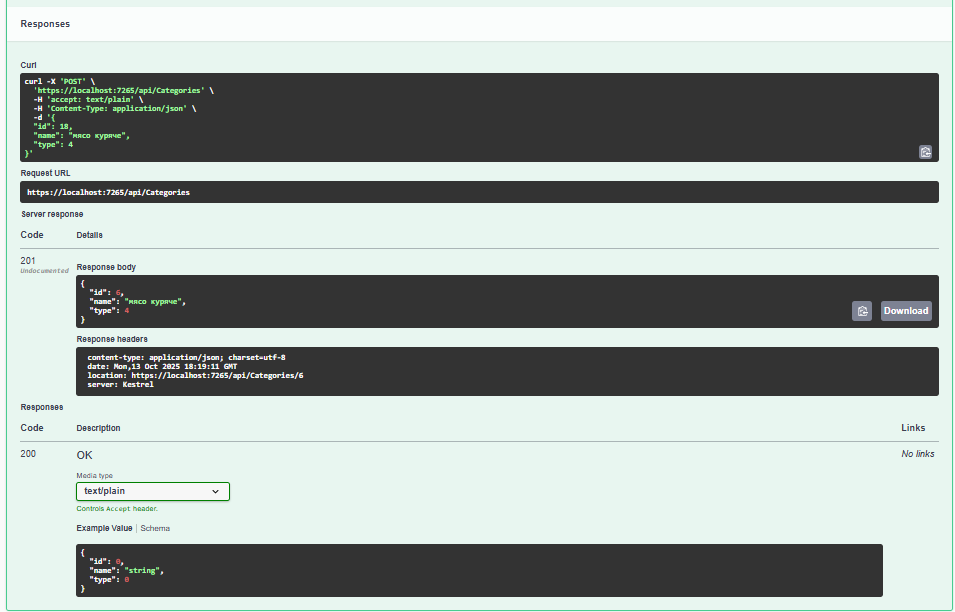
**Виконаємо запит на категорії.**

****

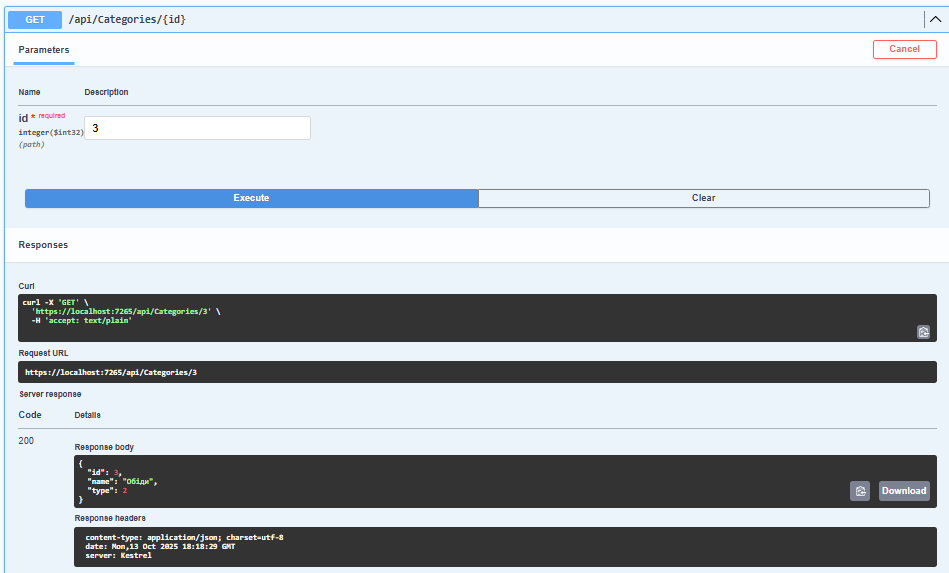
****

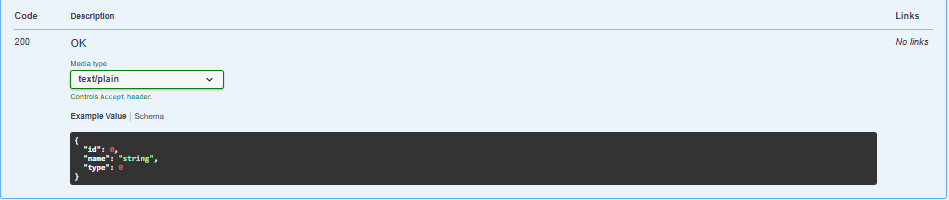
**Виконаємо запит на додавання категорії.**

****

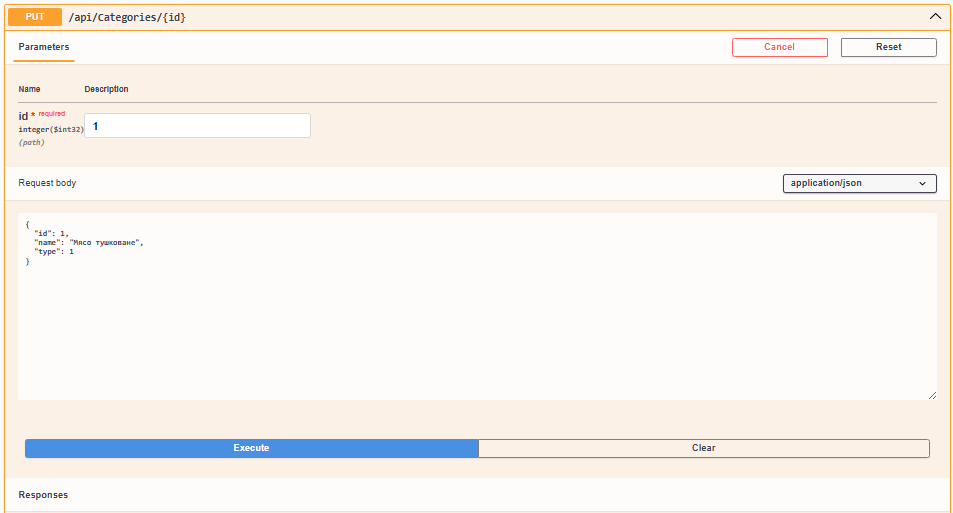
****

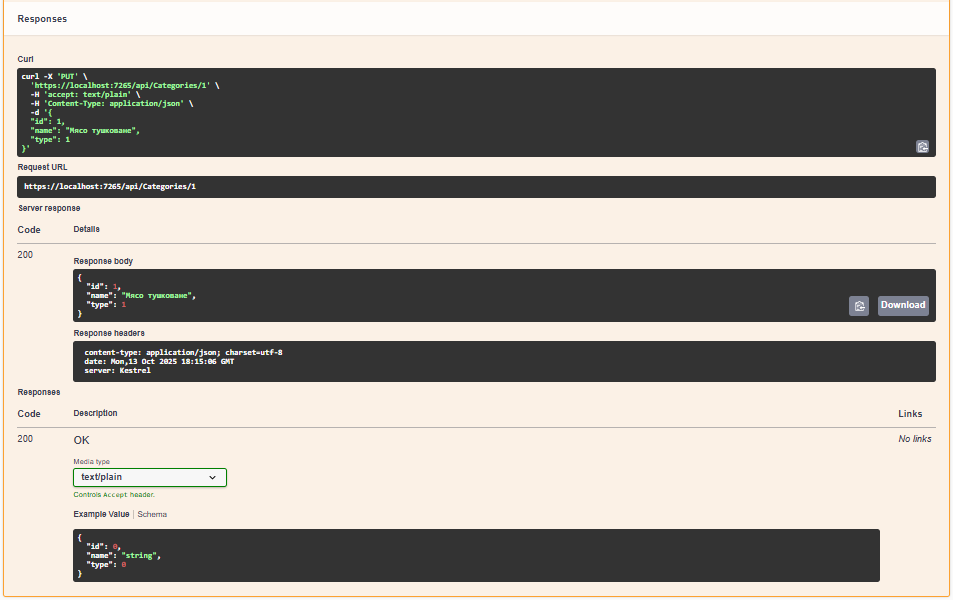
**Виконаємо запит на отримання конкретної категорії.**



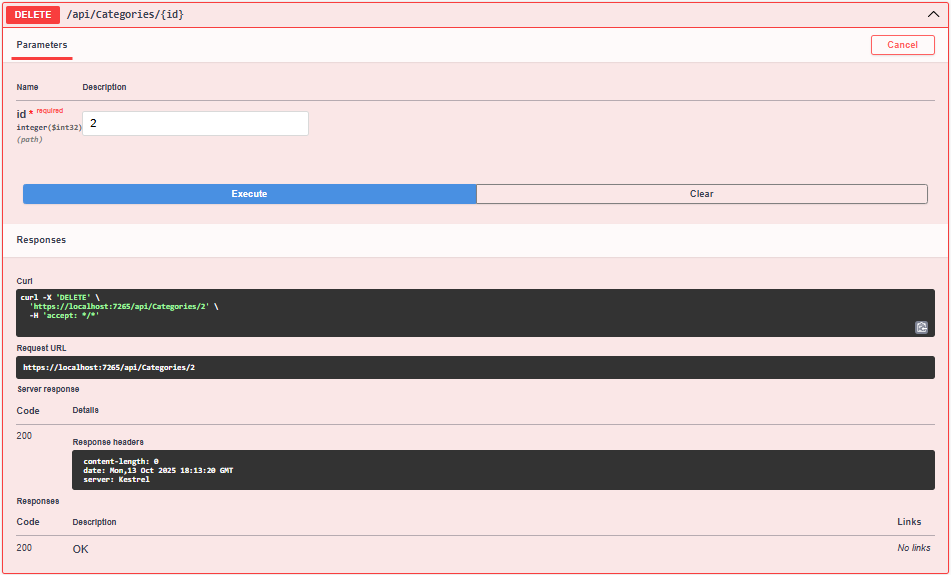


**Виконаємо запит на оновлення конкретної категорії.**





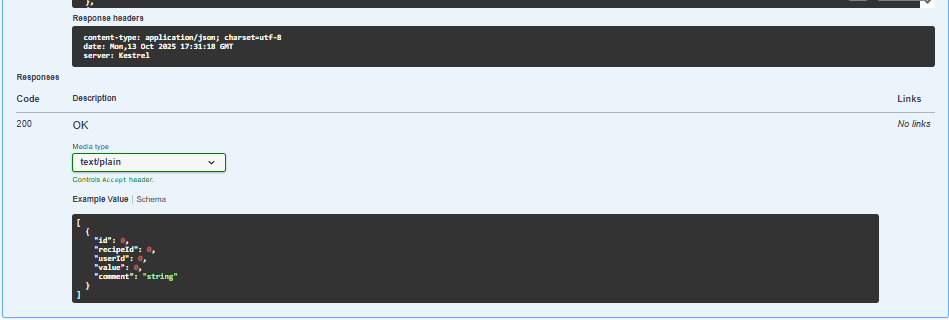
**Виконаємо запит на видалення конкретної категорії.**

****

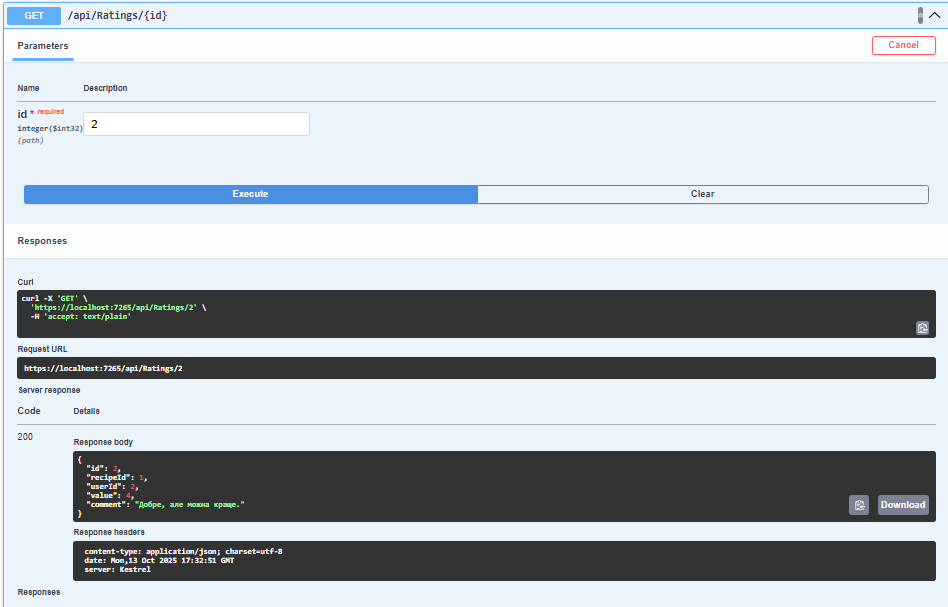
**Ratings**

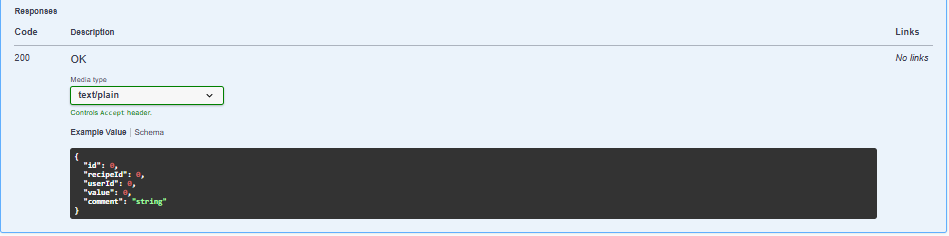
**Виконаємо запит на отримання рейтингів.**

****

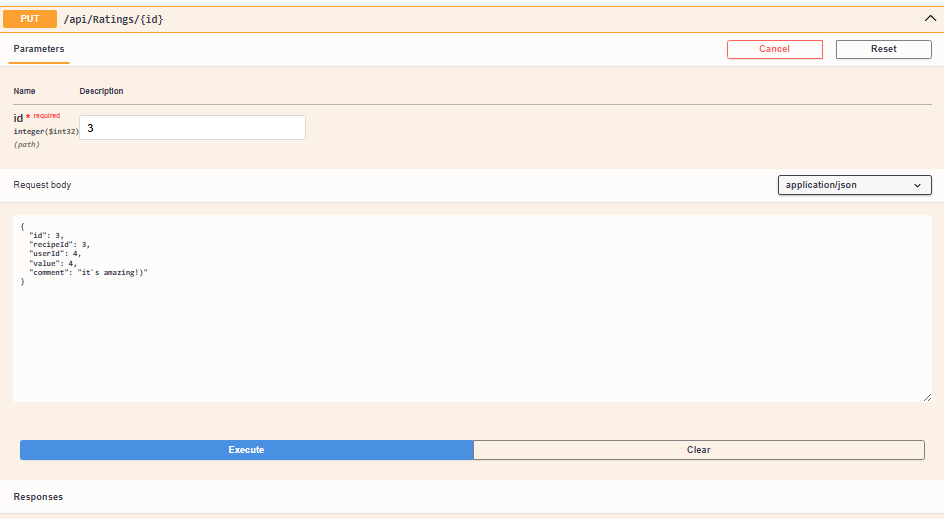
****

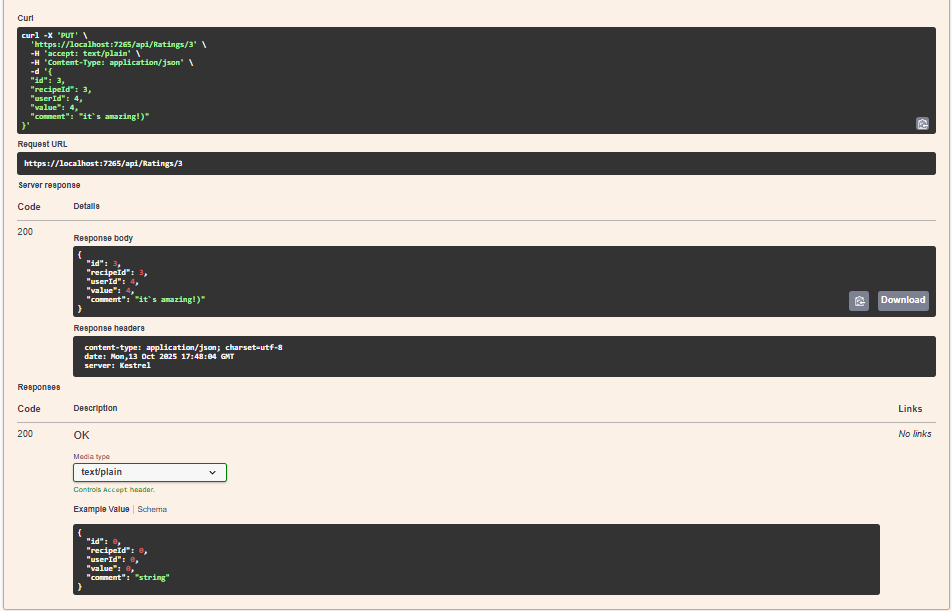
**Виконаємо запит на отримання конкретного рейтингу.**

****

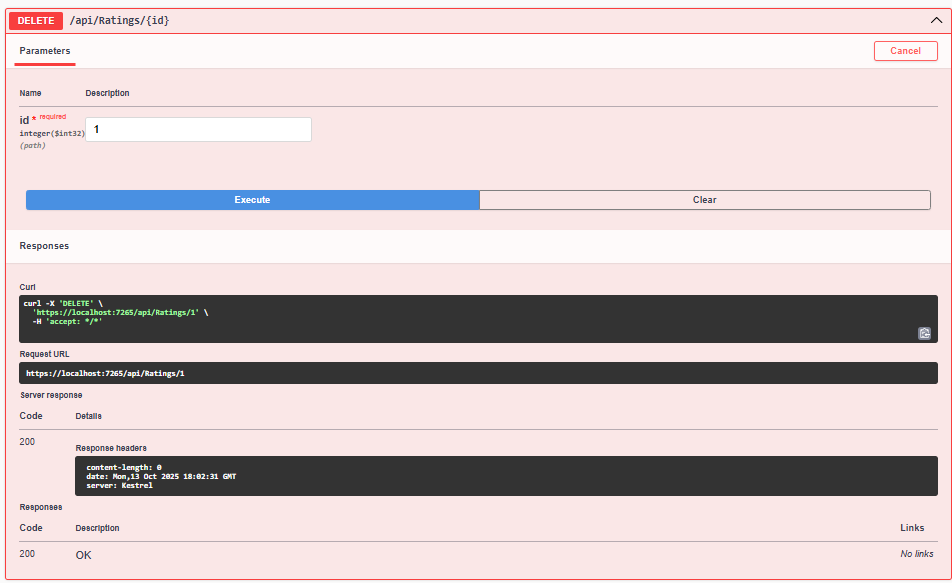
****

**Виконаємо запит на оновлення конкретного рейтингу.**

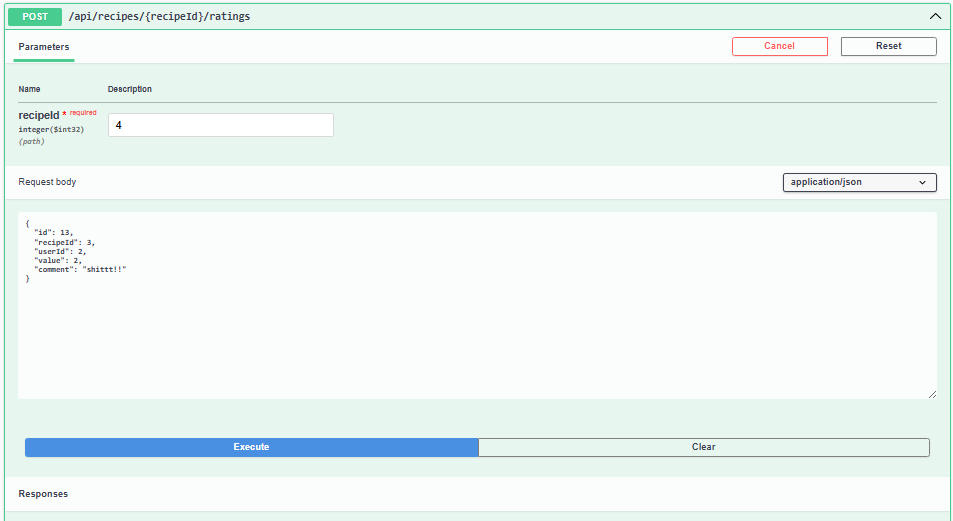
****

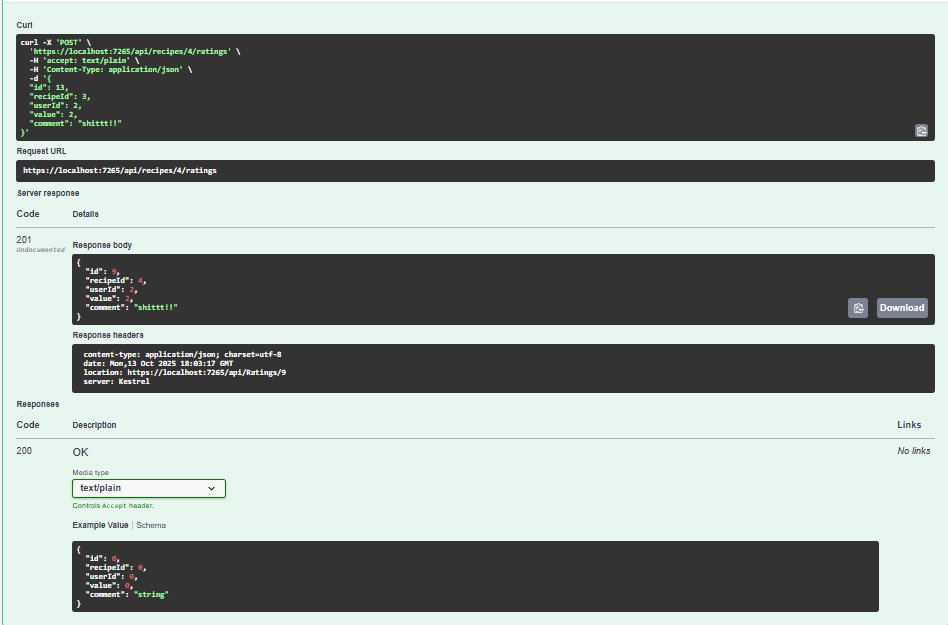
****

**Виконаємо запит на видалення конкретного рейтингу.**

****

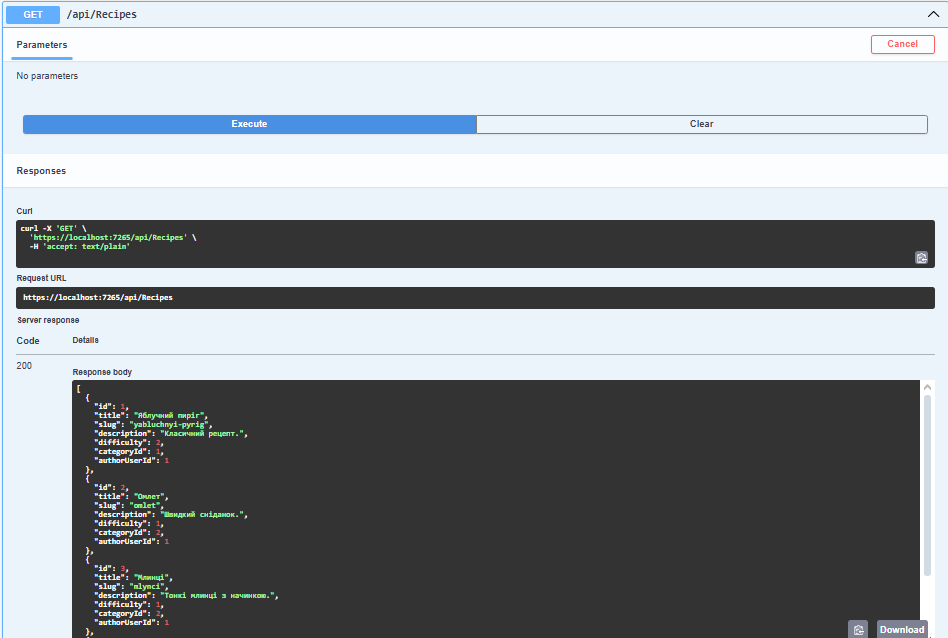
**Виконаємо запит на додавання рейтингу до конкретного рецепту.**

****

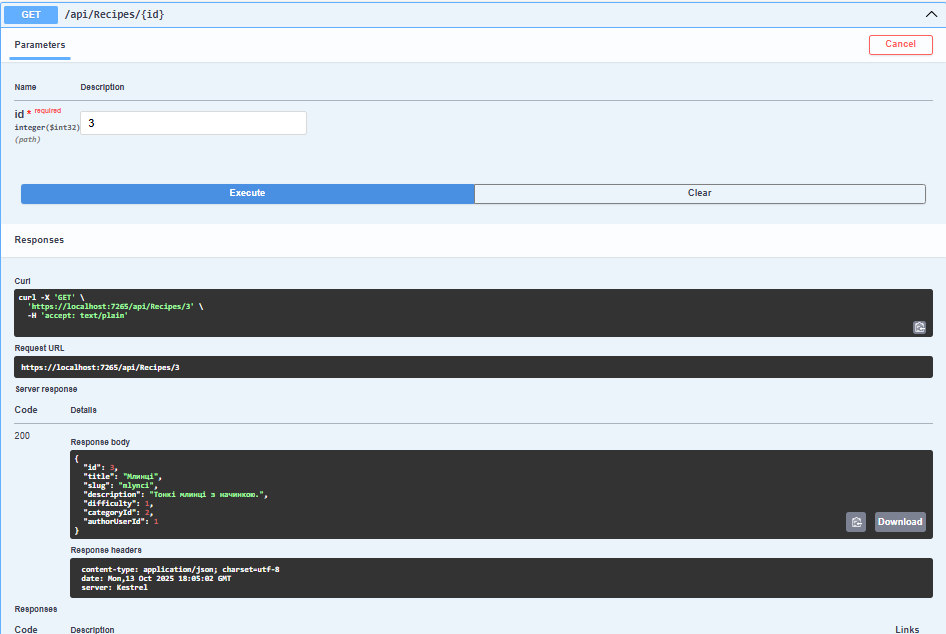
****

**Recipes**

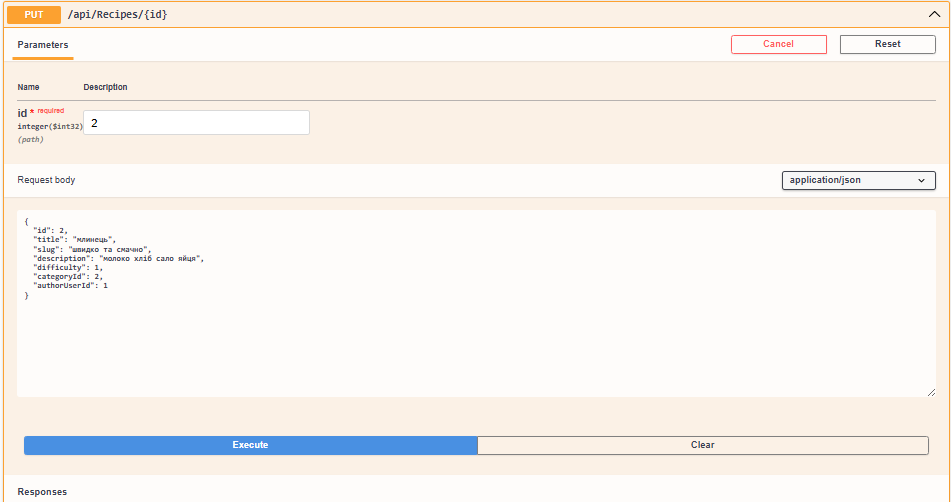
**Виконаємо запит на отримання усіх рецептів.**

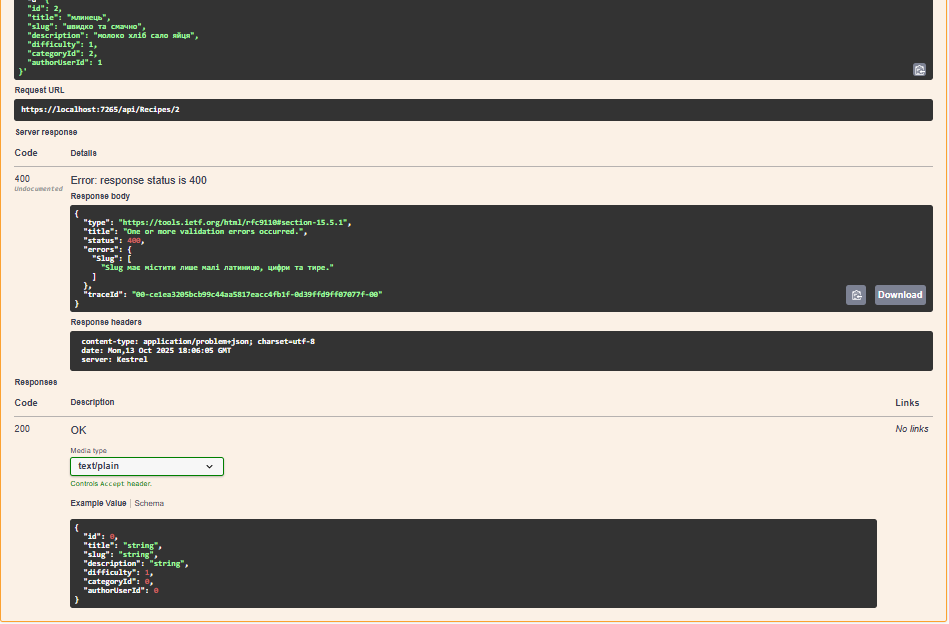
****

**Виконаємо запит на отримання конкретного рецепту.**

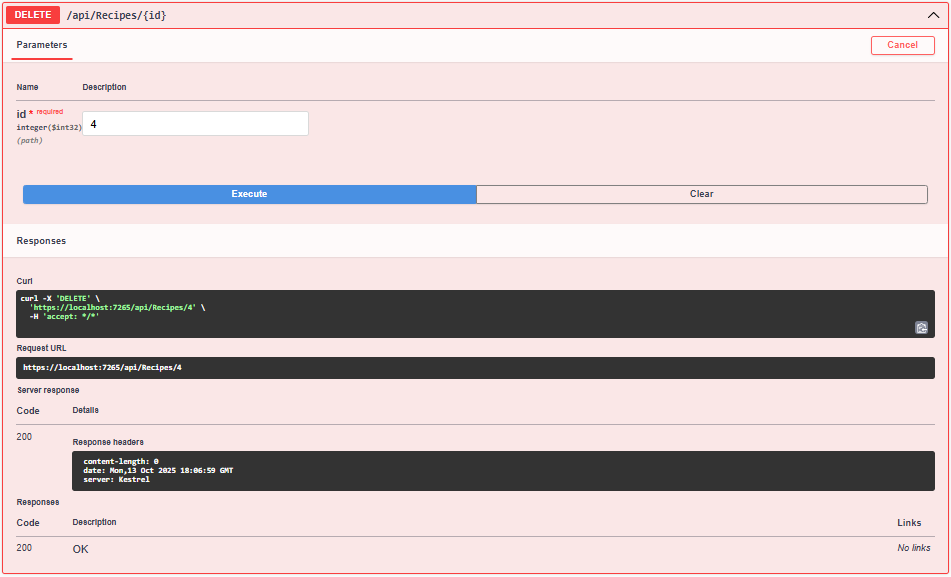
****

**Виконаємо запит на оновлення конкретного рецепту.**

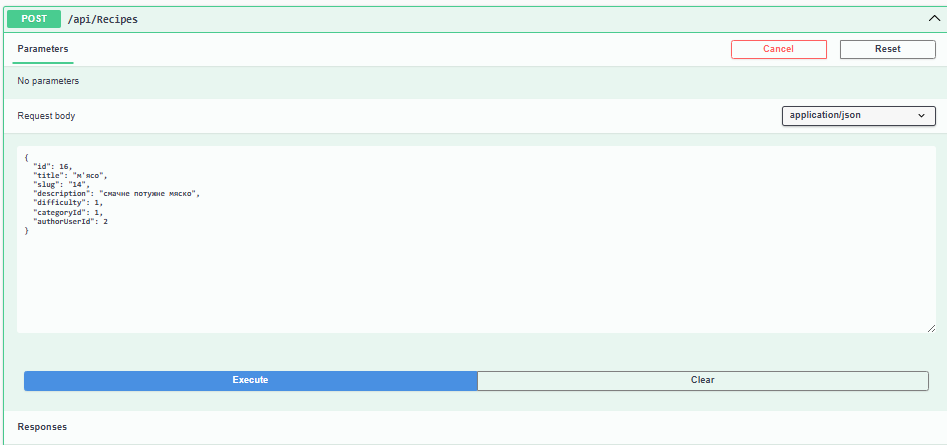
****

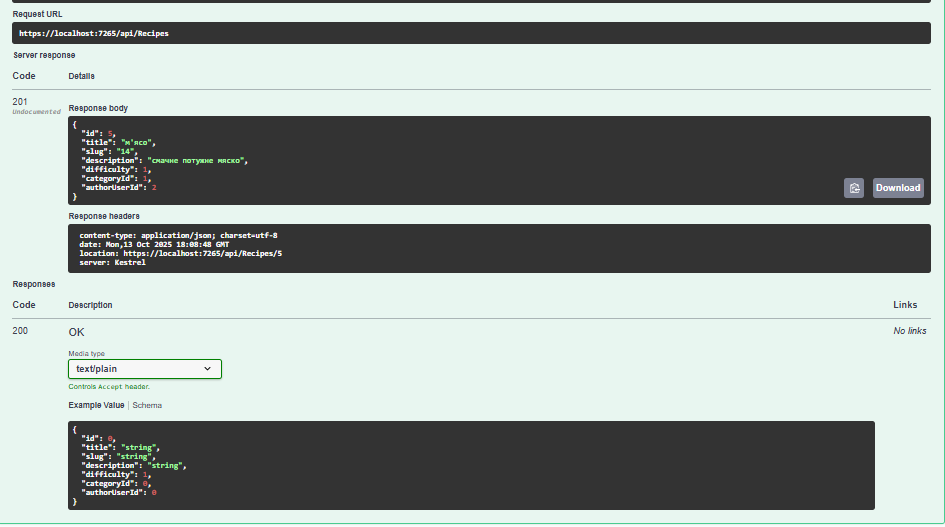
****

**Виконаємо запит на видалення конкретного рецепту.**

****

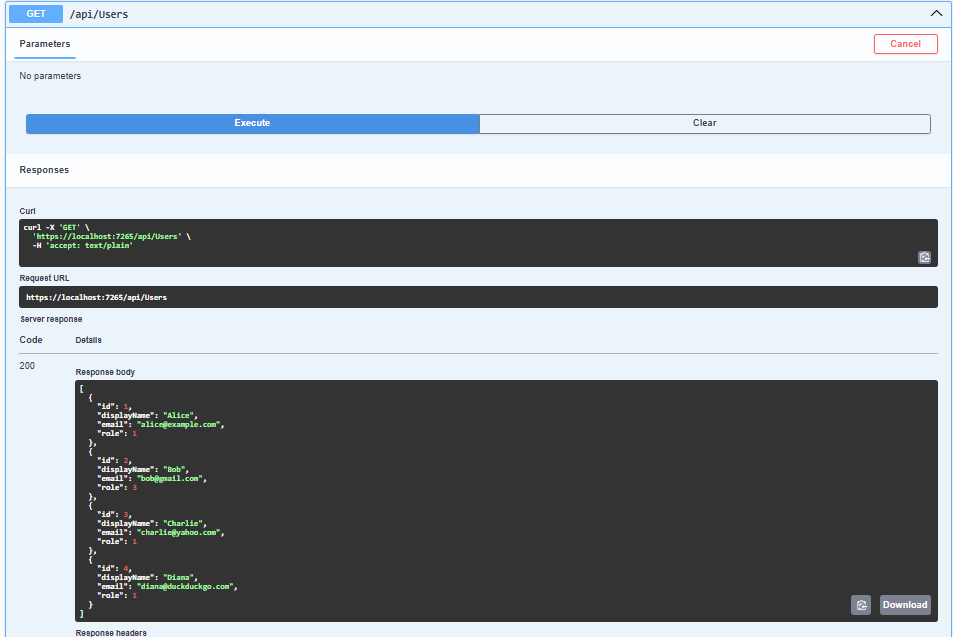
**Виконаємо запит на публікацію рецепта.**

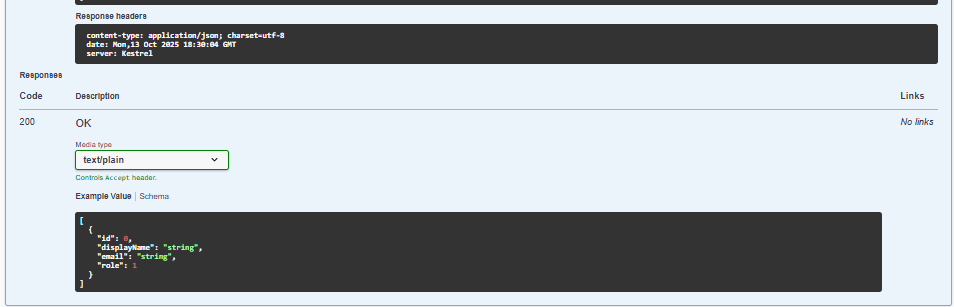
****

****

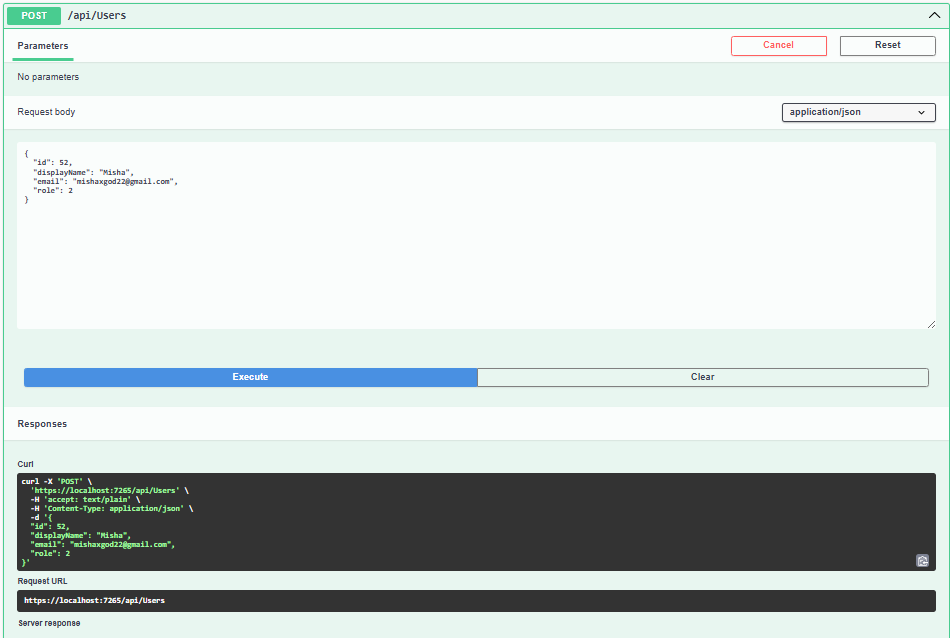
**Users**

**Виконаємо запит на отримання усіх юзерів.**

****

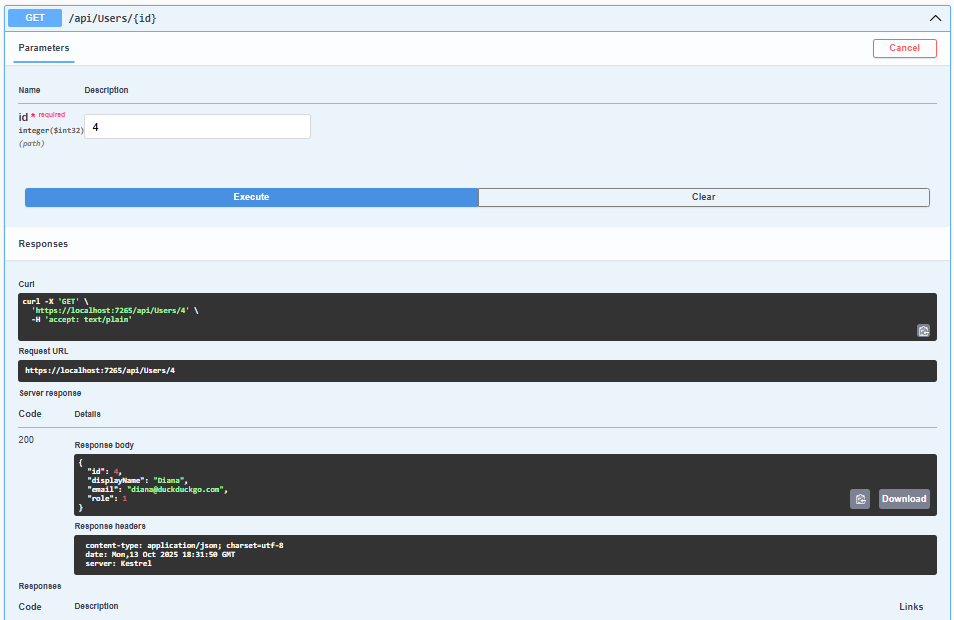
****

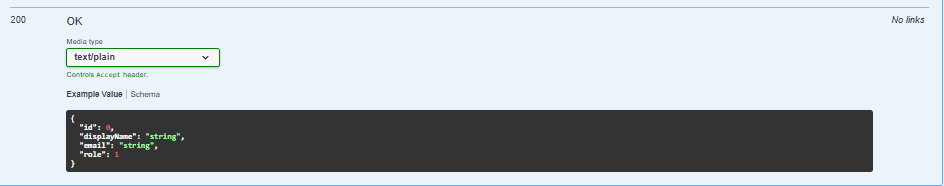
**Виконаємо запит на публікацію юзера.**

****

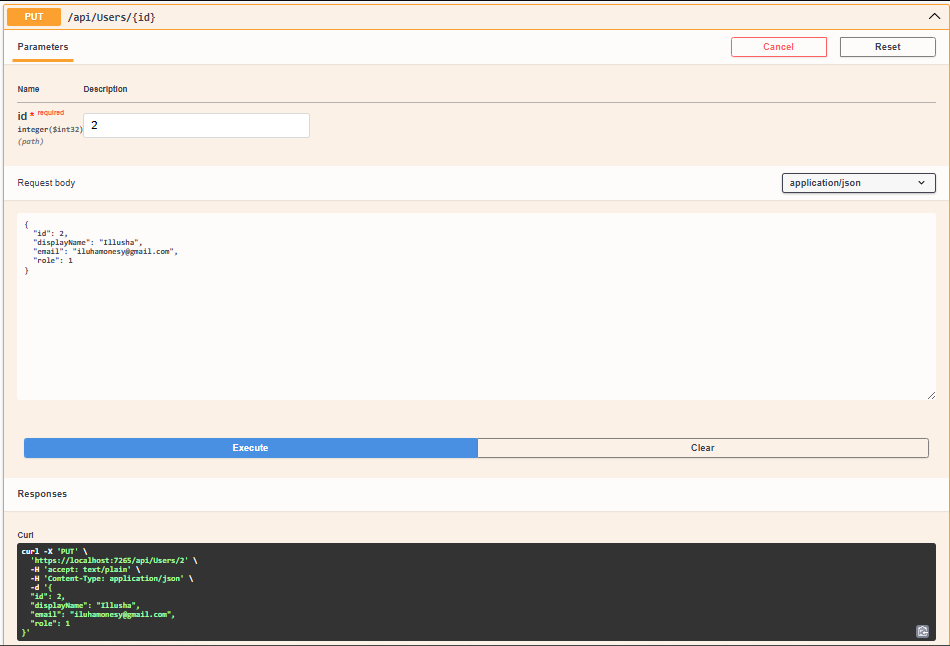
****

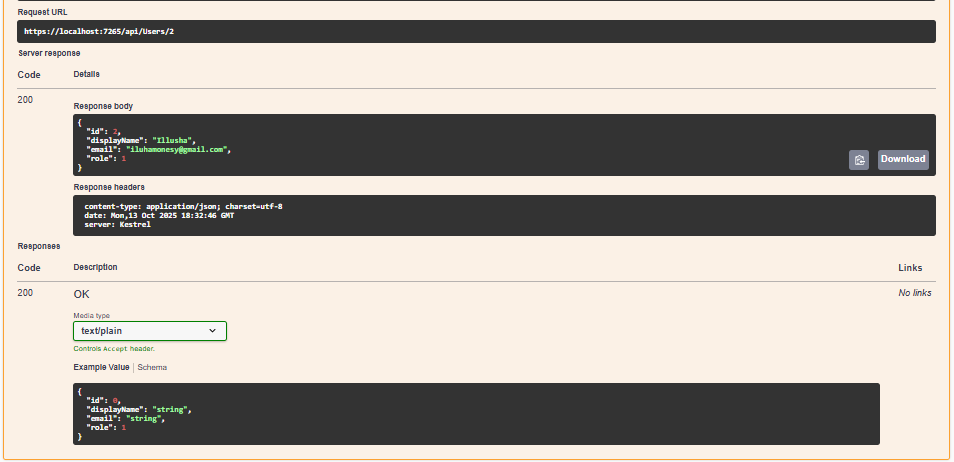
**Виконаємо запит на отримання конкретного юзера.**

****

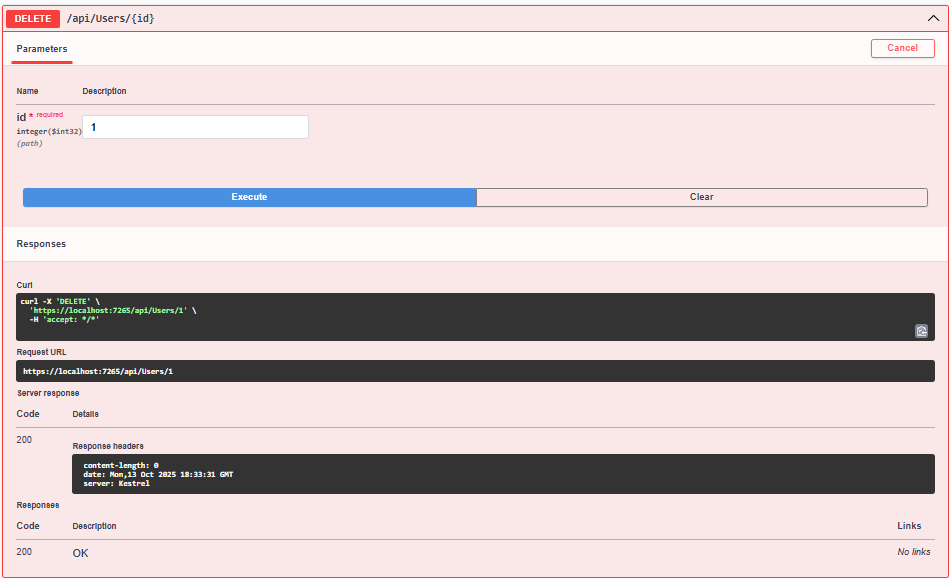
****

**Виконаємо запит на оновлення юзера.**

****

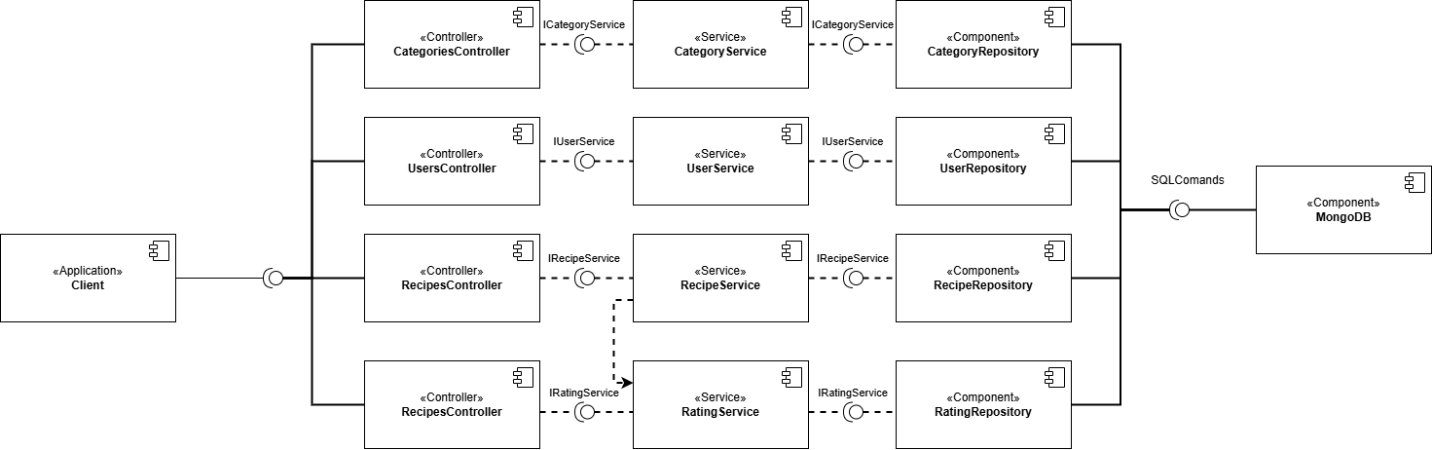
****

**Виконаємо запит на видалення юзера.**

****

**Завдання №3**

**Створити діаграму компонентів.**

****

**Controller layer:** Служить вхідною точкою для клієнтів: приймає HTTP-запити, перевіряє модель і викликає відповідний сервіс. Реалізує маршрути типу GET, POST та формує HTTP-відповідь.

Приклади методів: *GetAll(), GetById(id), Create(dto), Delete(id).*

**Service layer:** Відповідає за бізнес-логіку та правила роботи з даними: перевіряє існування сутності, готує її до збереження, може викликати кілька репозиторіїв за одну операцію. Реалізує сценарії типу “створити рецепт і потім видалити пов’язані рейтинги при видаленні”.

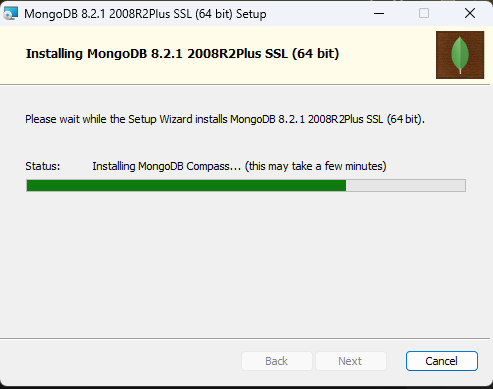
Приклади методів: *CreateAsync(entity), UpdateAsync(id, entity), DeleteAsync(id), GetAllAsync().*

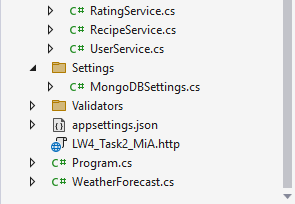
**Repository layer:** Забезпечує доступ до джерела даних і приховує деталі БД від сервісів. Реалізує базові CRUD-операції над конкретними колекціями/таблицями.

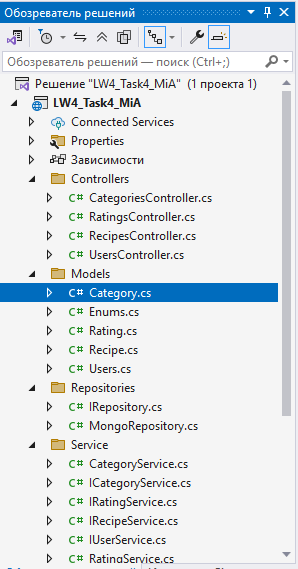
Приклади методів: *GetAllAsync(), GetByIdAsync(id), InsertAsync(entity) / CreateAsync(entity), UpdateAsync(id, entity), DeleteAsync(id).*

**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);

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)

{

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;

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)

{

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()

{

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

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

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();

//після переходу на Mongo

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

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

}

}

}

**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);

// 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);

});

// 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"));

// 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();

}

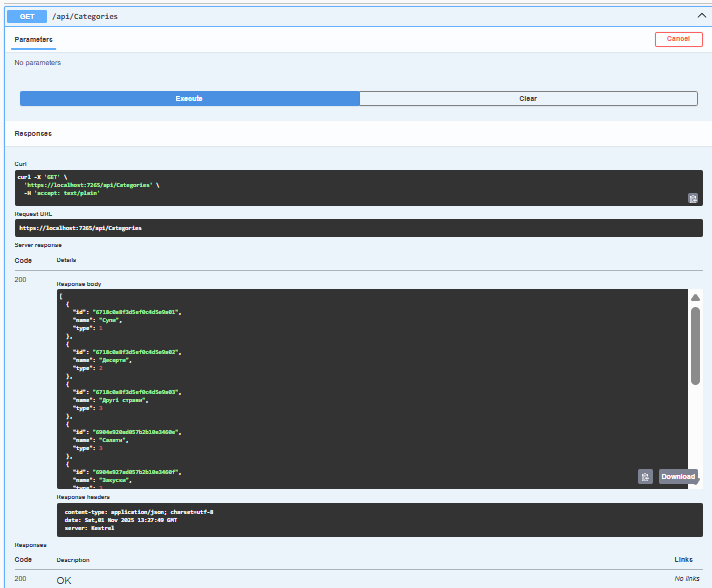
app.UseHttpsRedirection();

app.MapControllers();

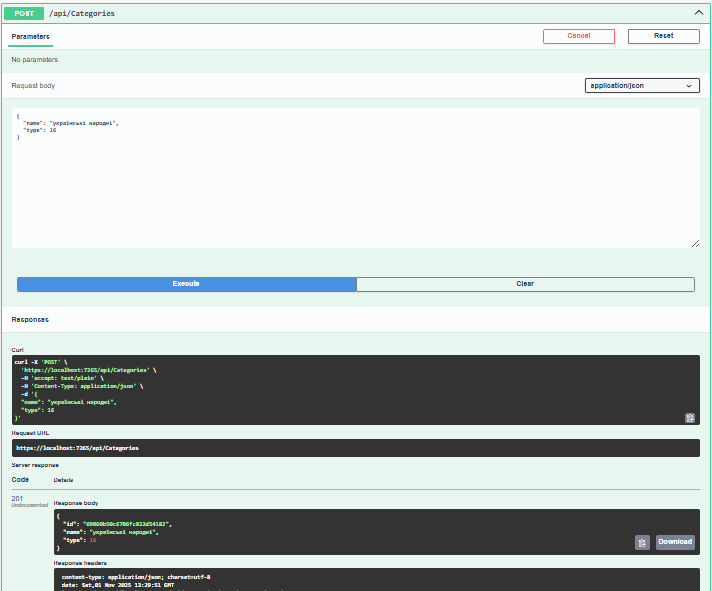
app.Run();

**Результат програми:**

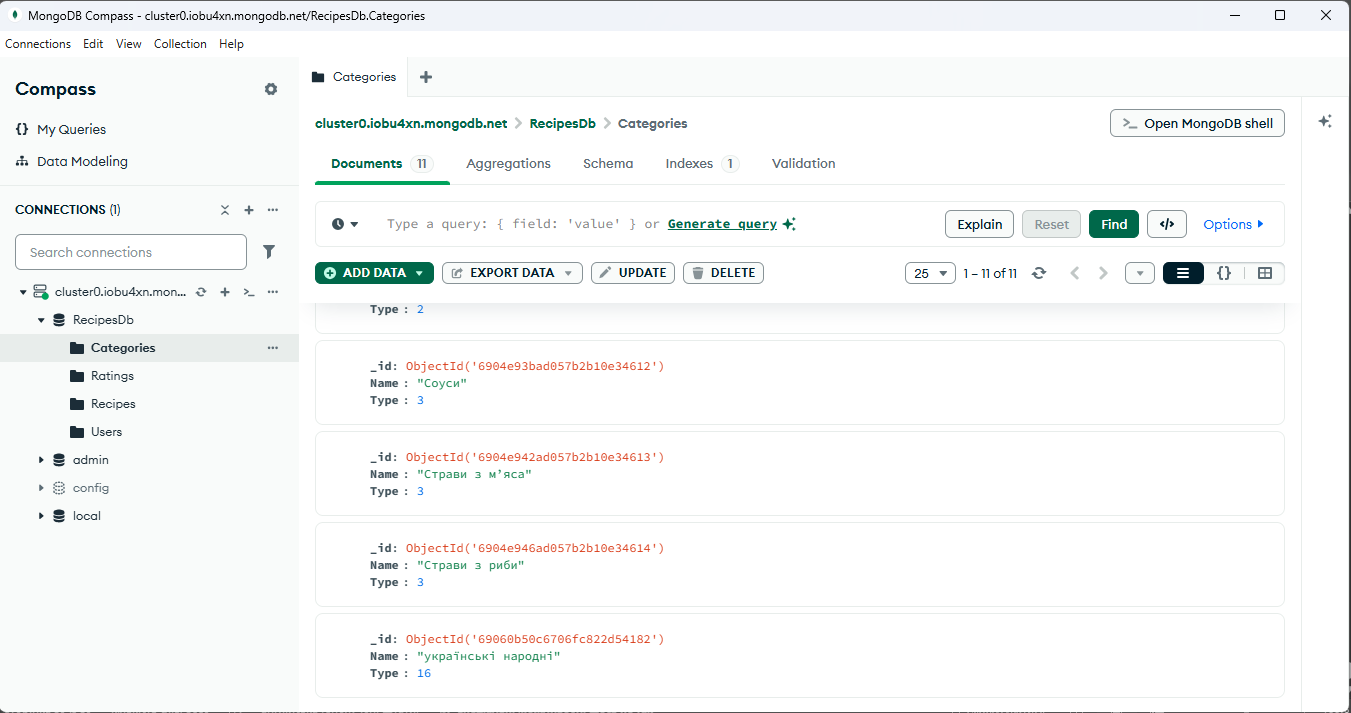
Виконаємо запит на отримання категорій, програма видасть ті категорії які записані були в бд.

****

Запишемо категорію та переглянемо в MongoDB. Записуємо без ID (MongoDB генерує сама).

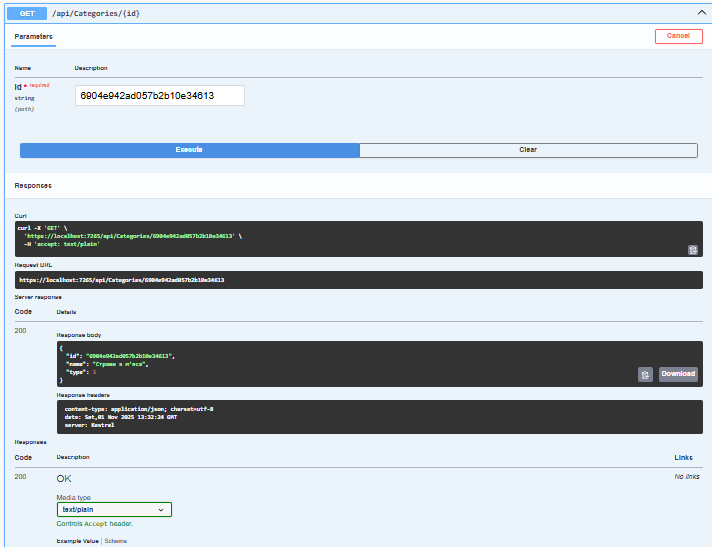
****

Переглянемо в MongoDB

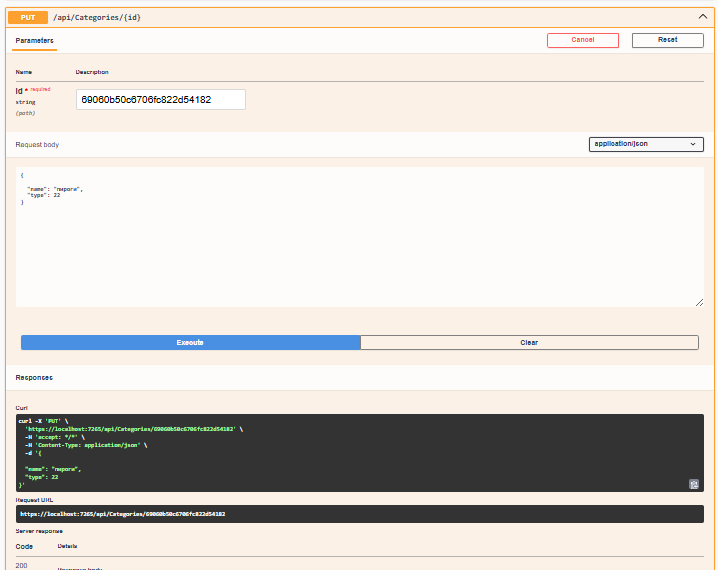
****

Дані були записані в БД.

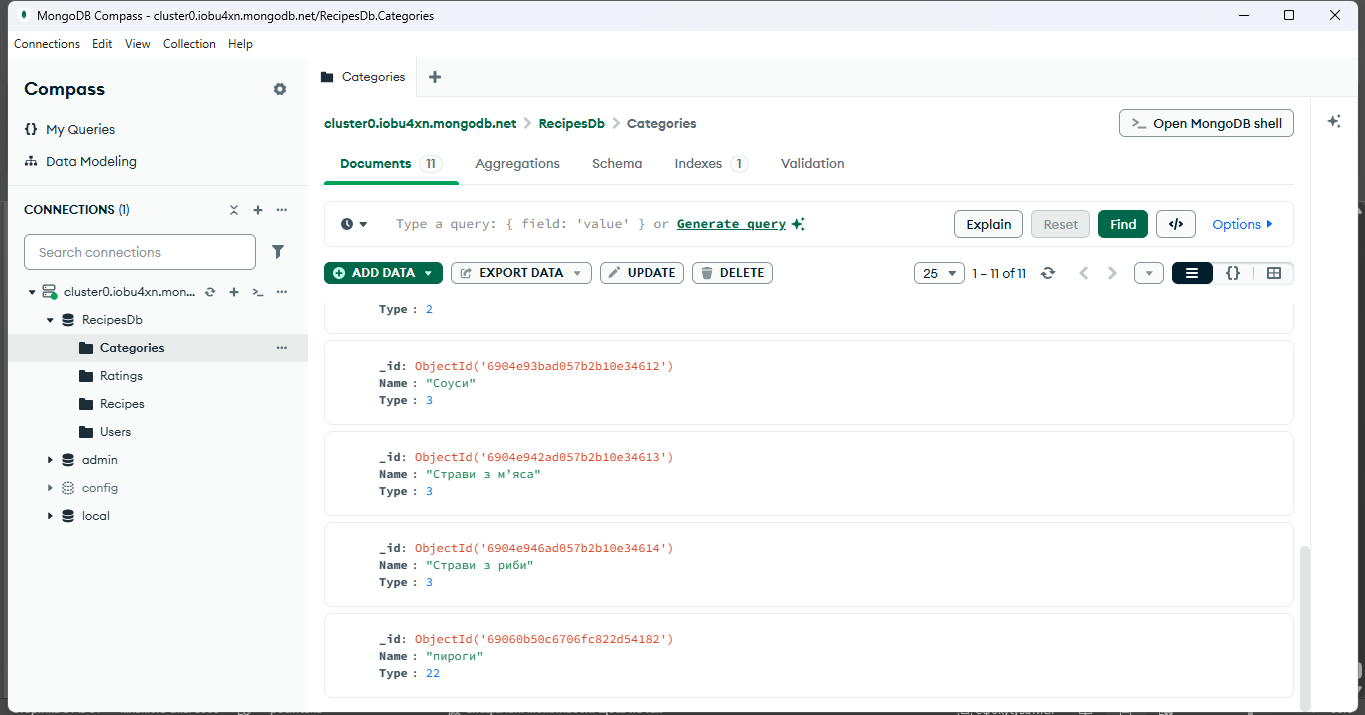
Отримуємо по конкретному ID

****

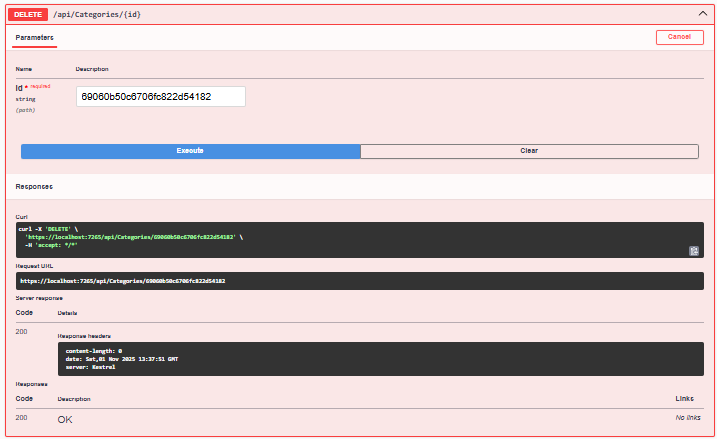
Виконаємо оновлення по конкретному ID



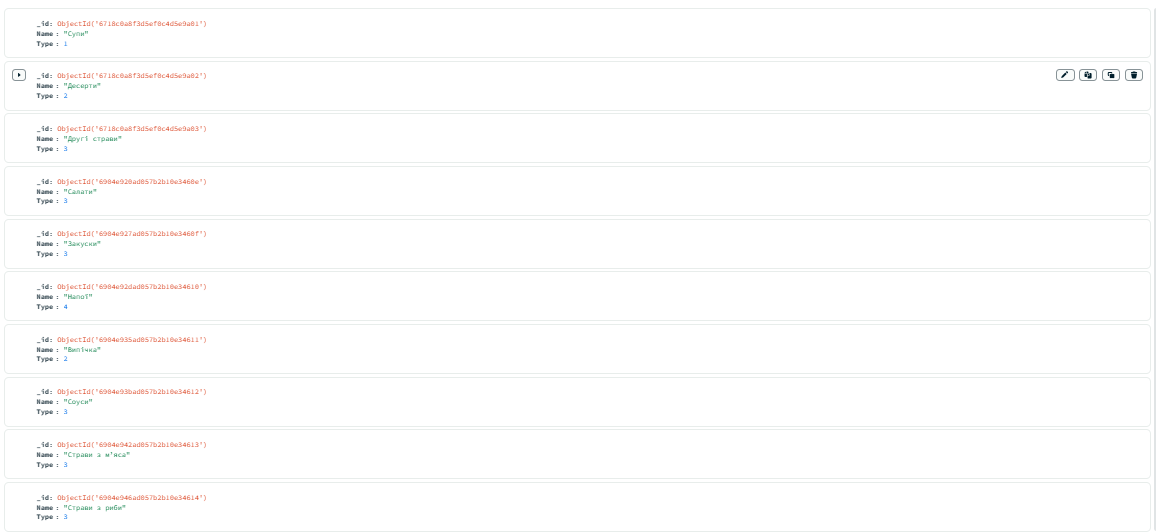
Перевіримо в БД



Видалимо категорію по конкретному ID

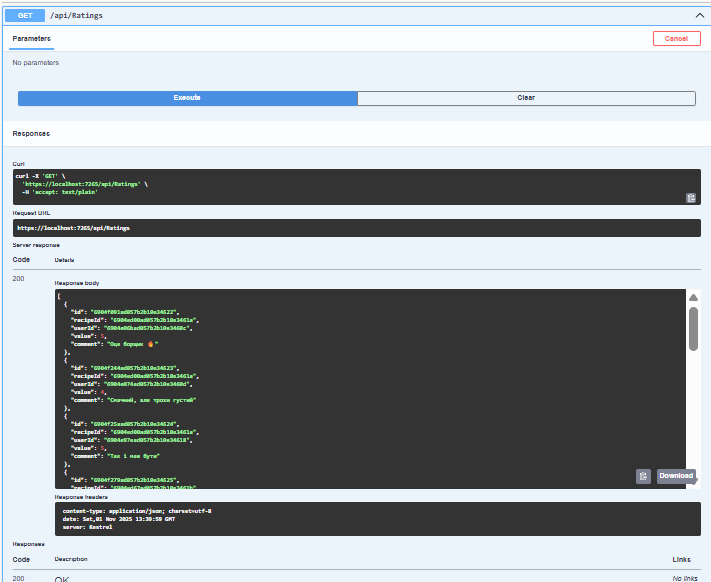


Перевіримо в БД чи вона є. Якщо немає значить успішно видалена.

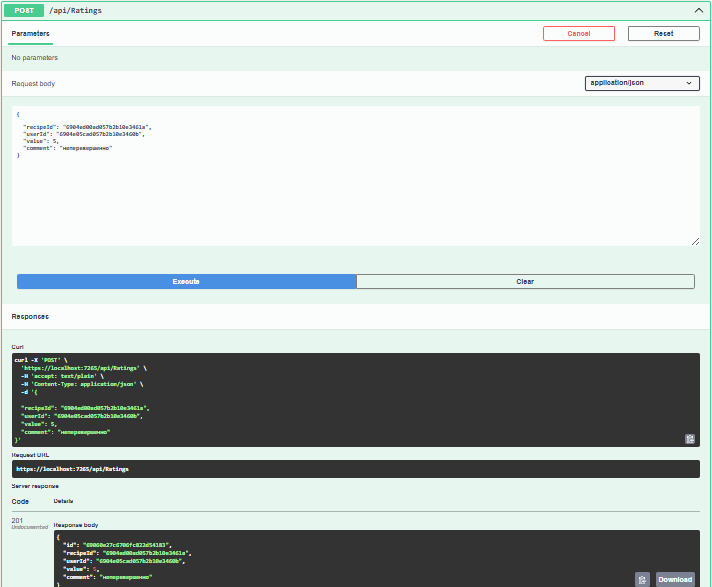


**Рейтинги**

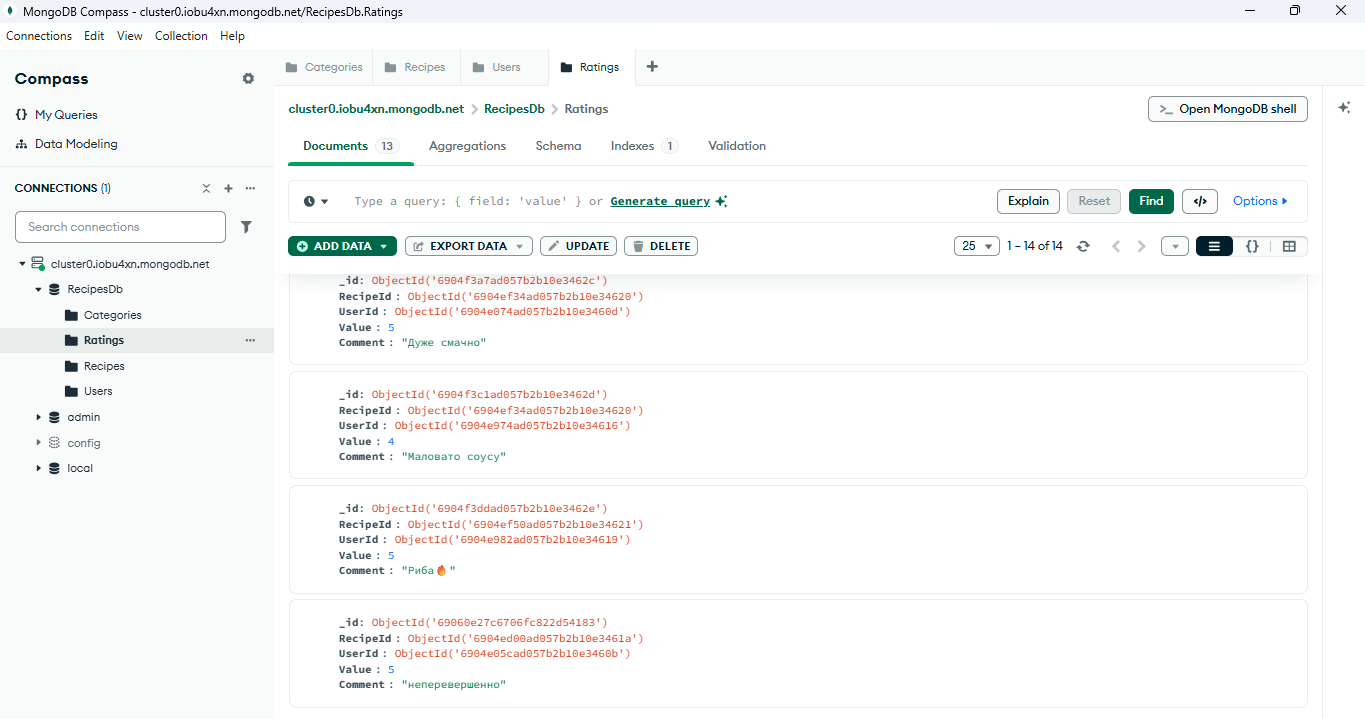
Запит на отримання усіх рейтингів

****

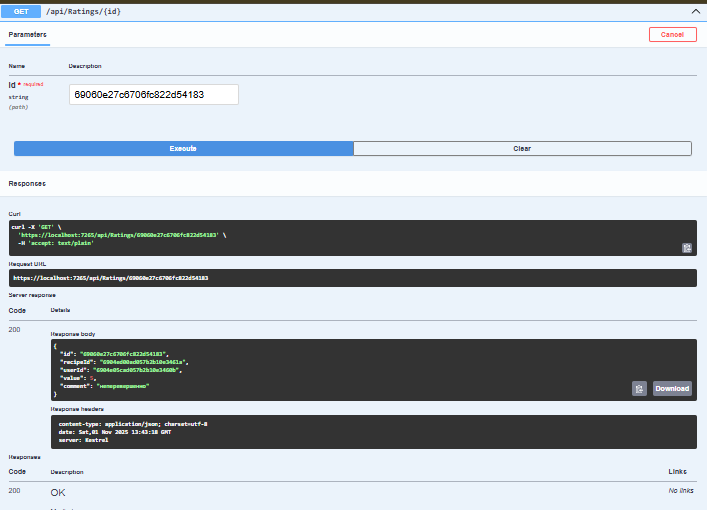
Запит на публікування рейтингу (без ID, MongoDB робить генерацію сама).

****

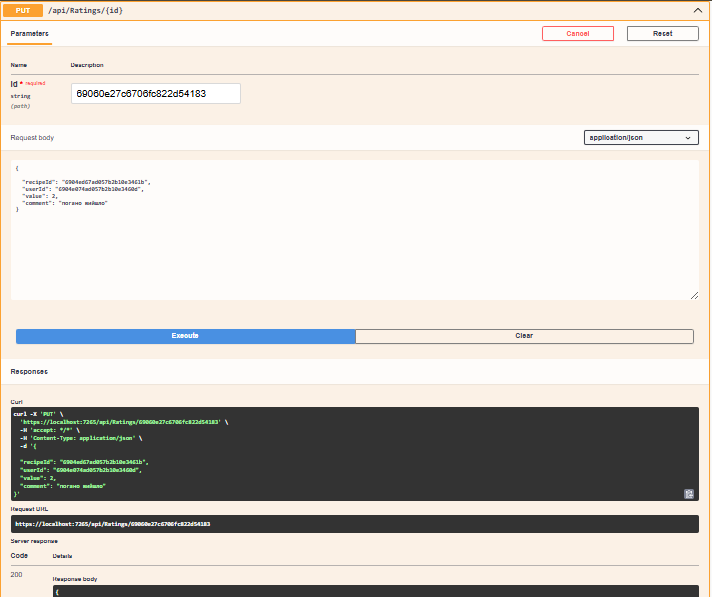
Переглянемо в БД.

****

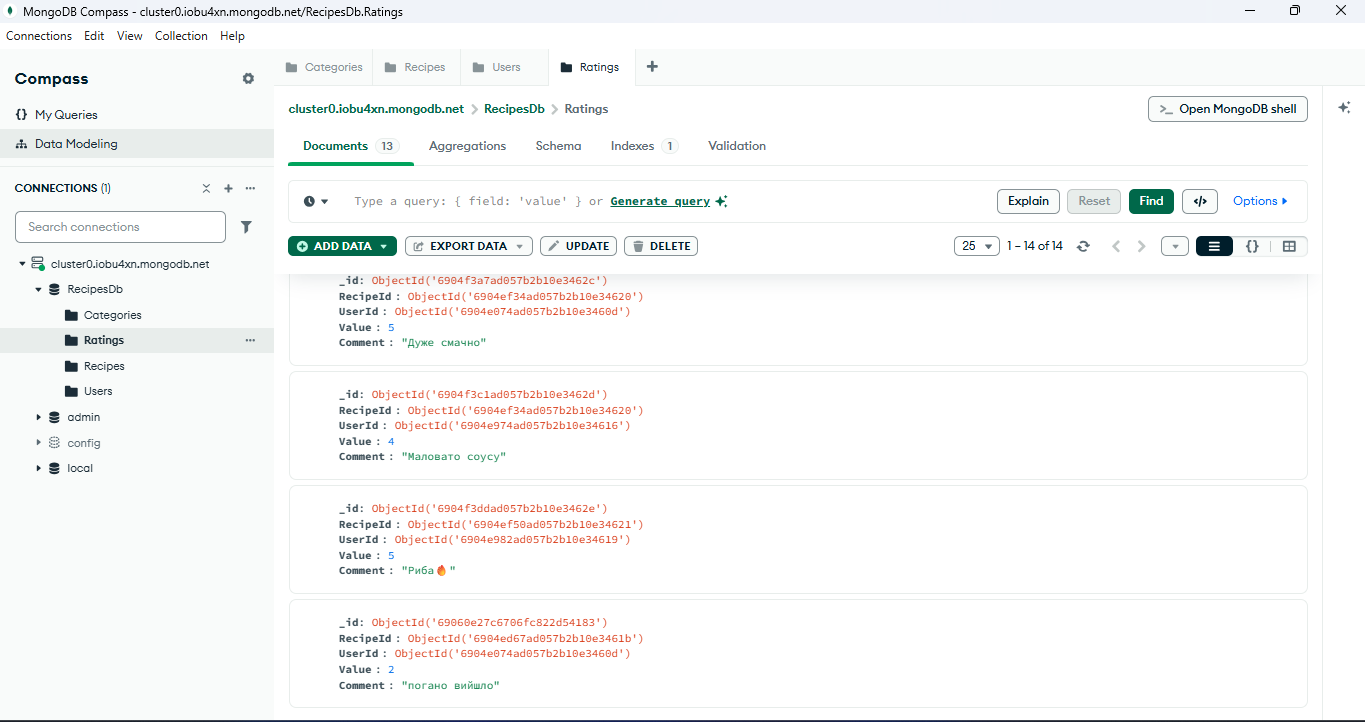
Отримаємо рейтинг по конкретному ID.

****

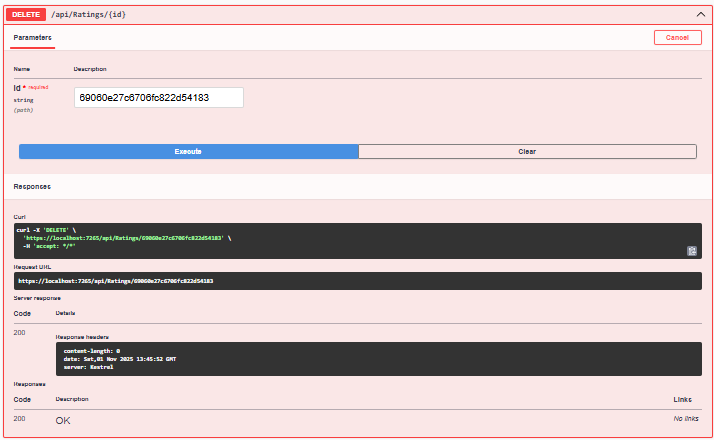
Оновлюємо рейтинг по конкретному ID.

****

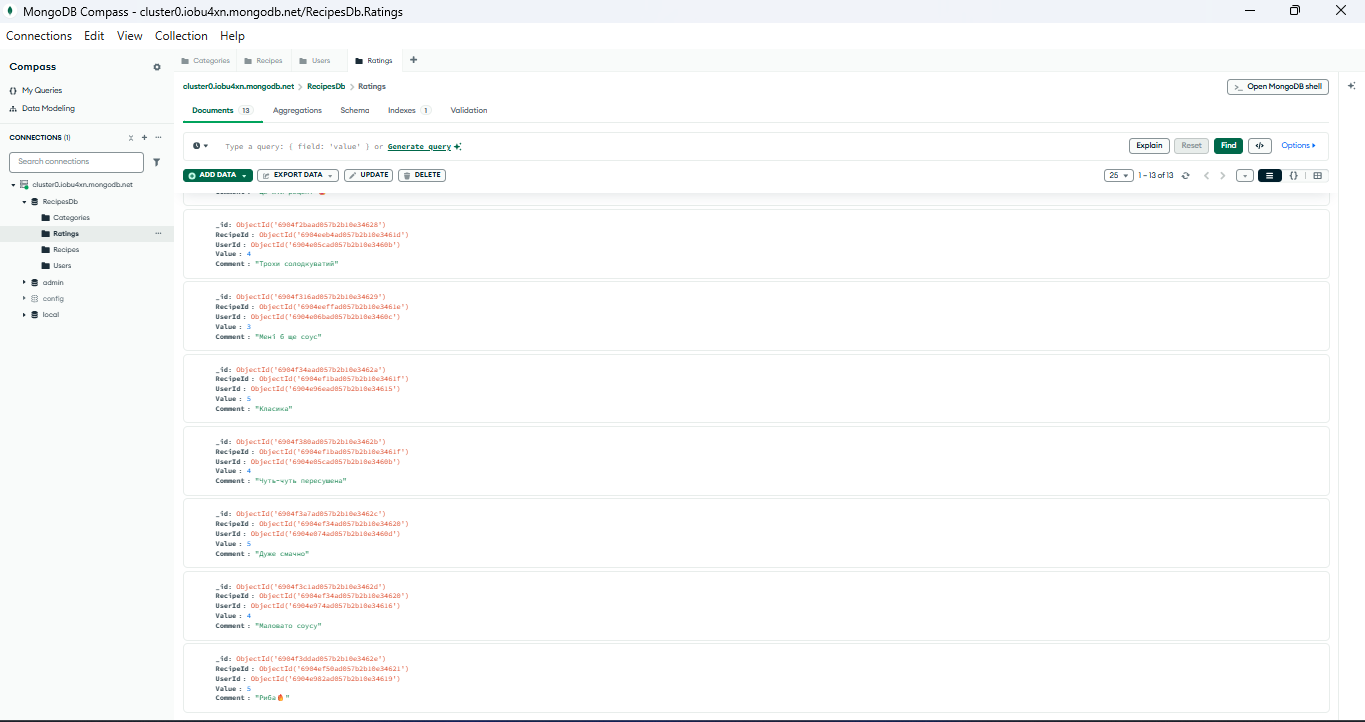
Перевіримо в БД.

****

Видалимо рейтинг по конкретному ID.

****

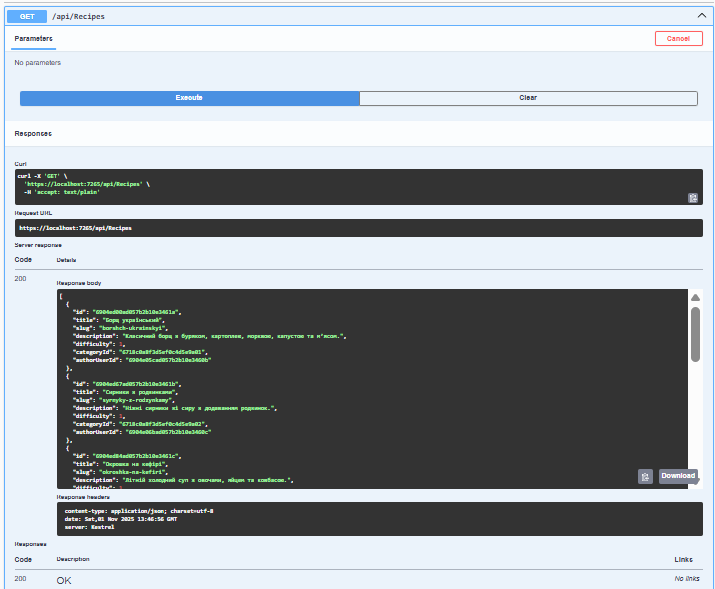
Перевіримо в БД.

****

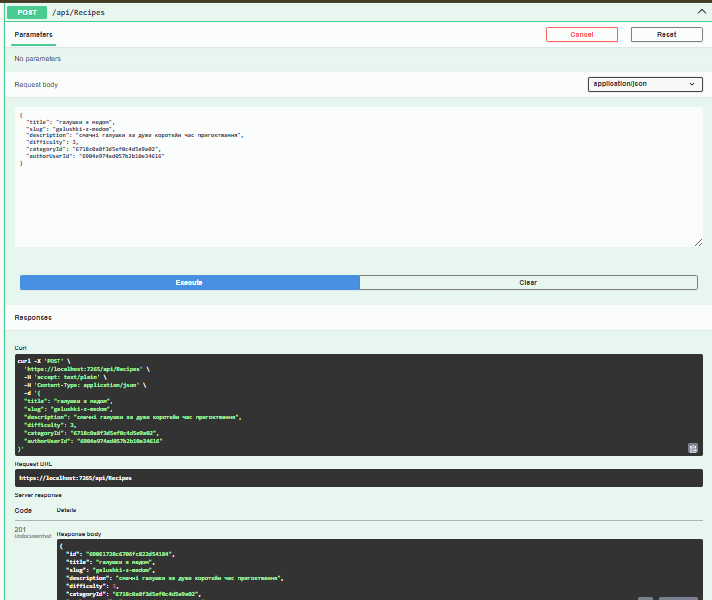
Оцінка була успішно видалена.

**Рецепти**

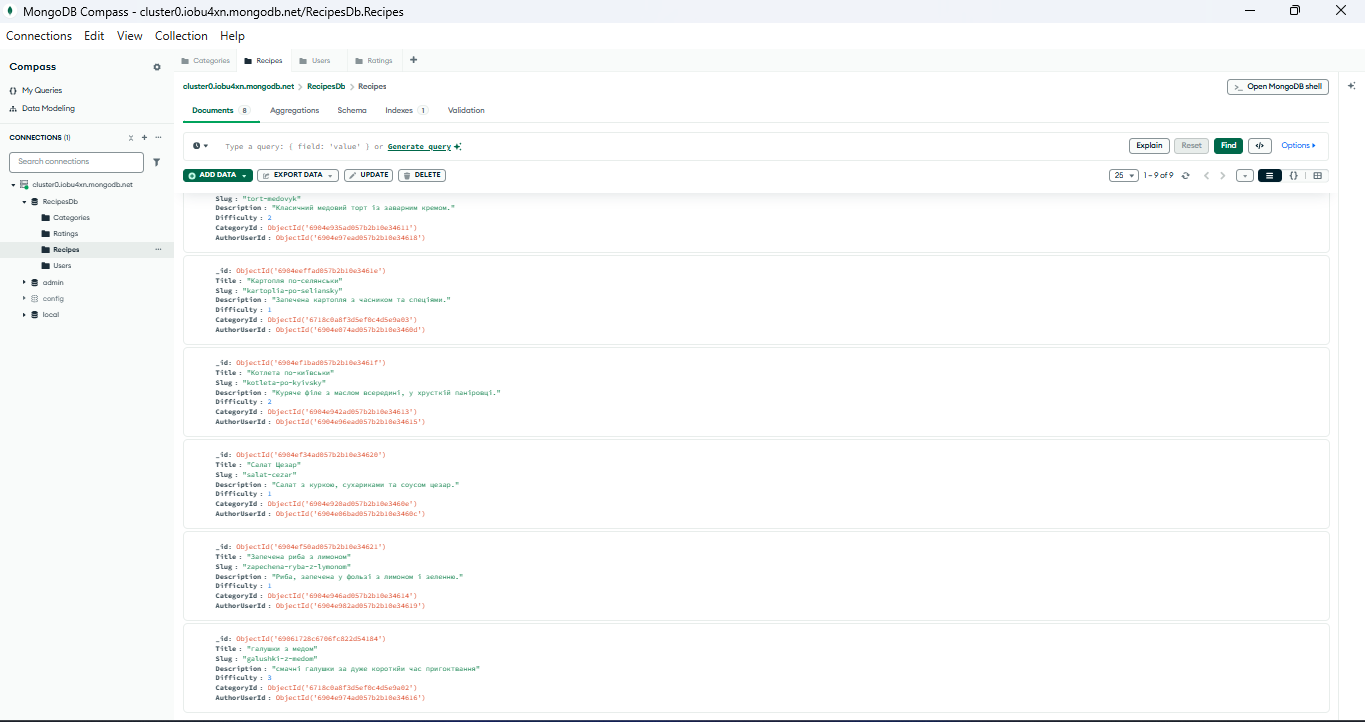
Отримаємо усі рецепти

****

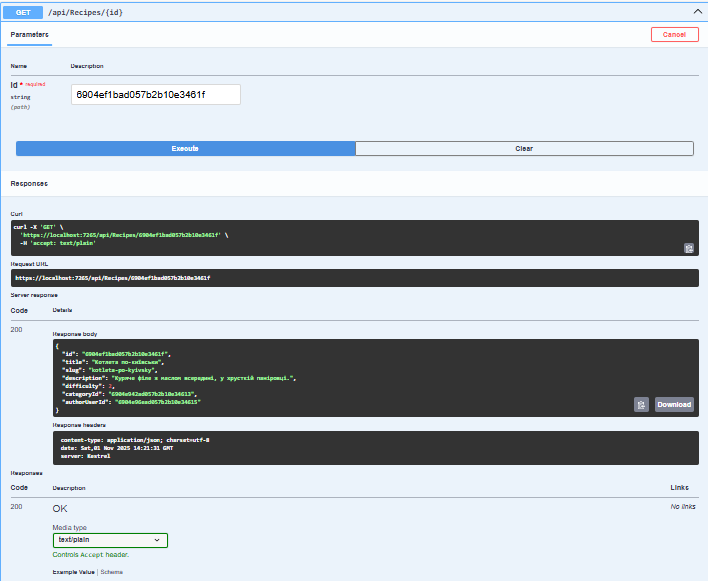
Публікуємо рецепт в БД (без ID, MongoDB генерує сама).

****

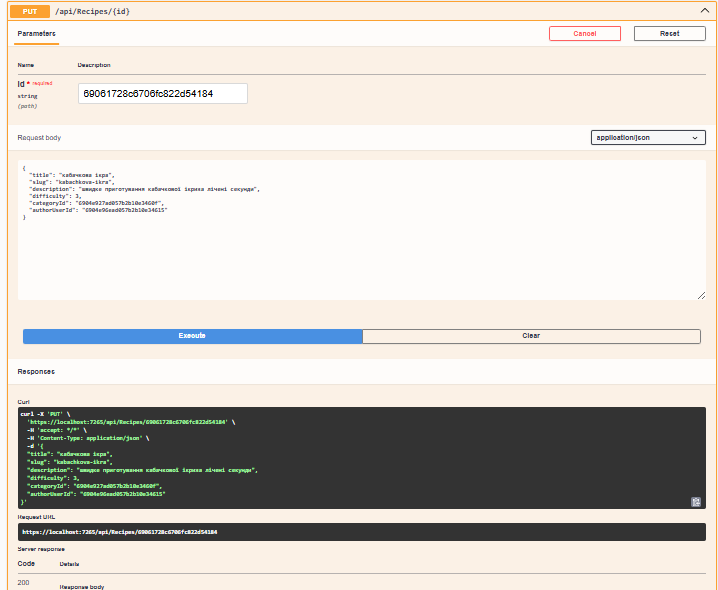
Перевіряємо в БД.

****

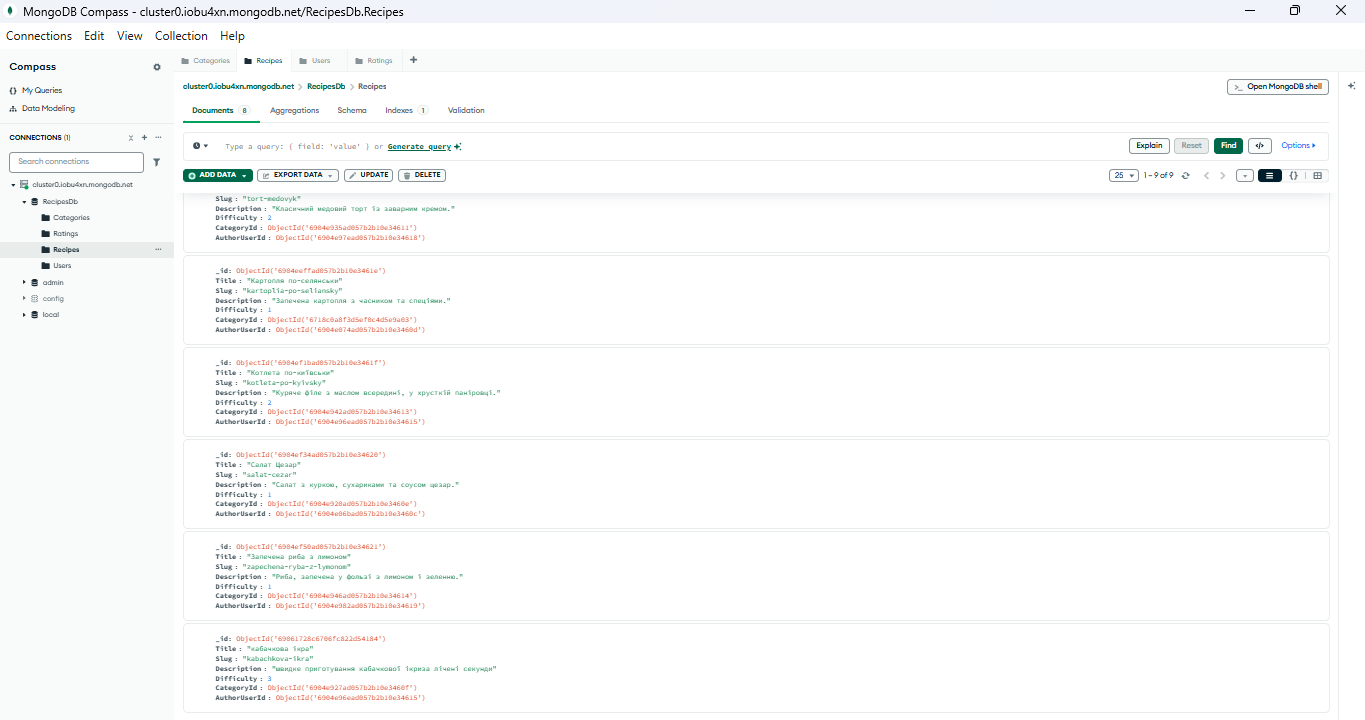
Отримаємо рецепт по конкретному ID.

****

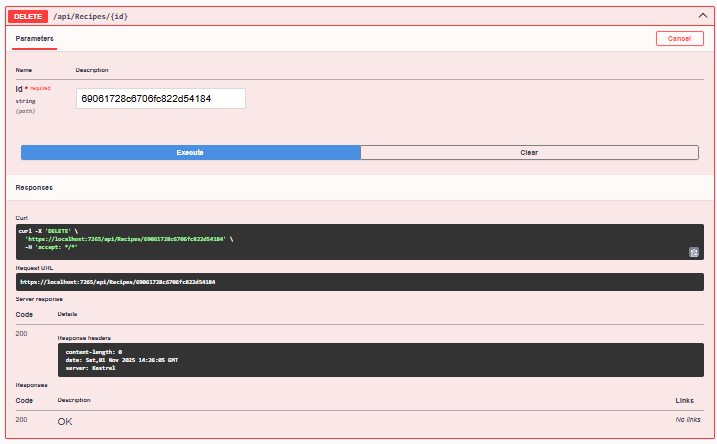
Оновлюємо рецепт по конкретному ID.

****

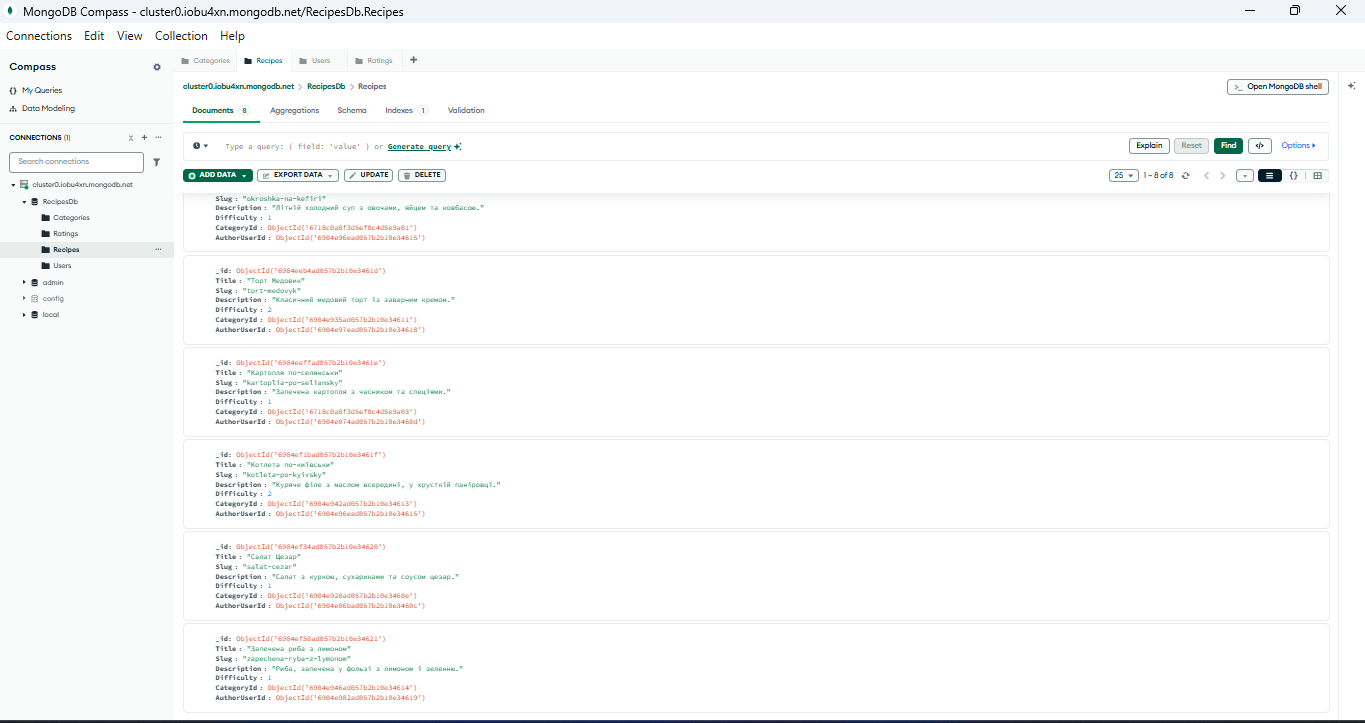
Перевіряємо в БД.

****

Видалимо рецепт по конкретному ID.

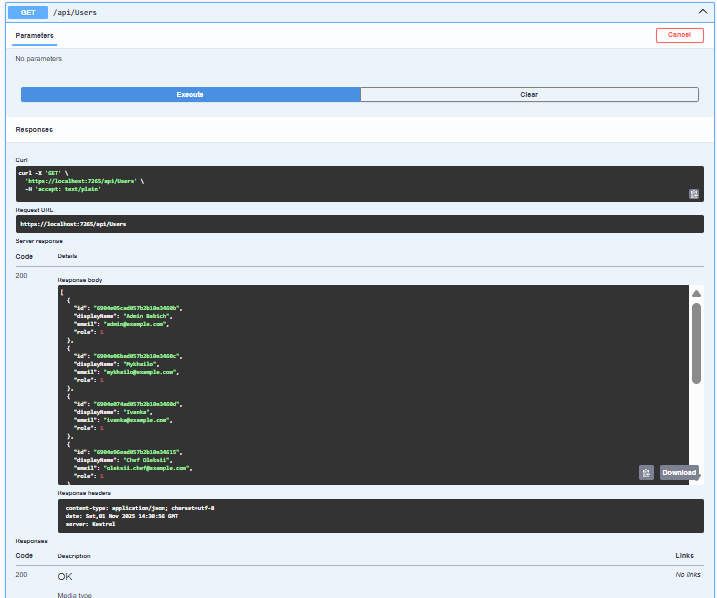
****

Перевіримо в БД.

****

**Користувачі**

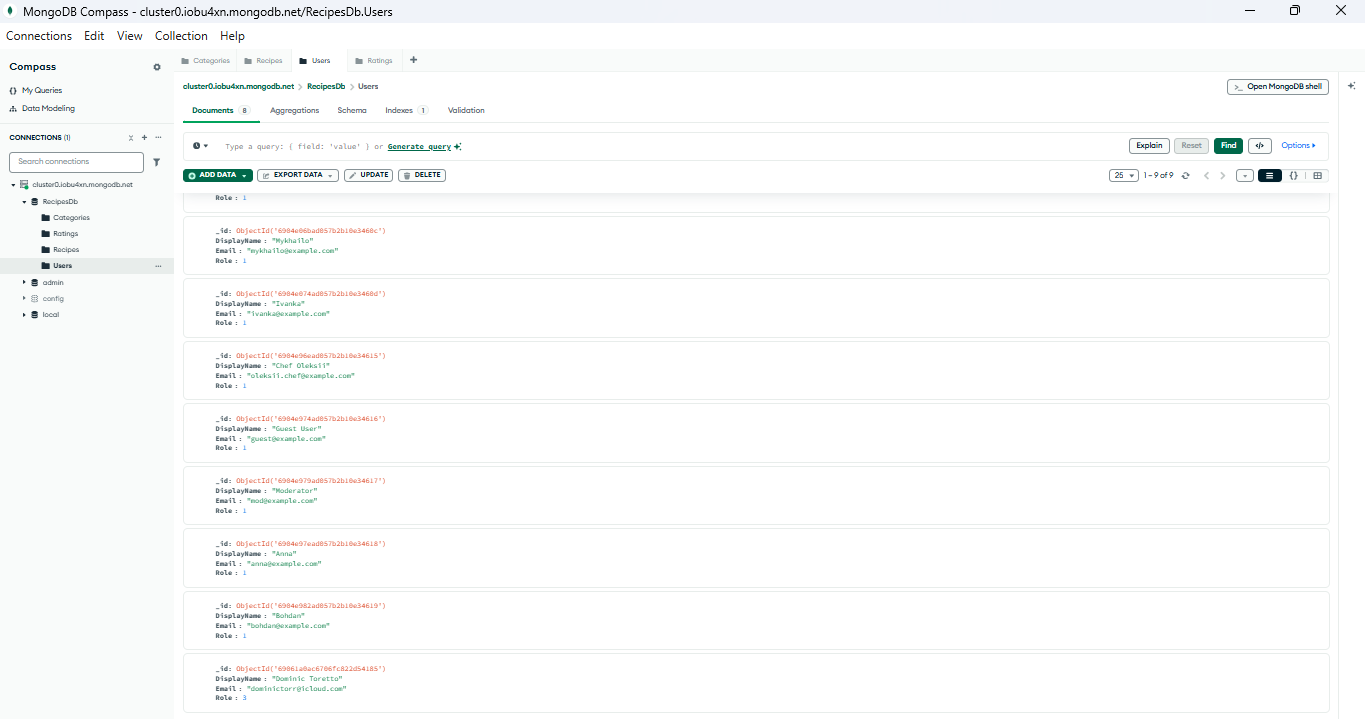
Отримаємо усіх користувачів.

****

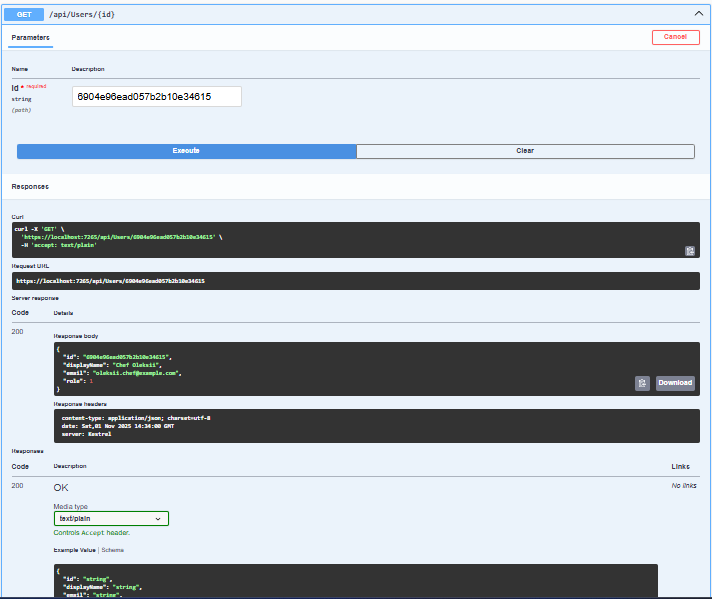
Публікуємо користувача (без ID, MongoDB згенерує сама).

****

Перевіримо в БД.

****

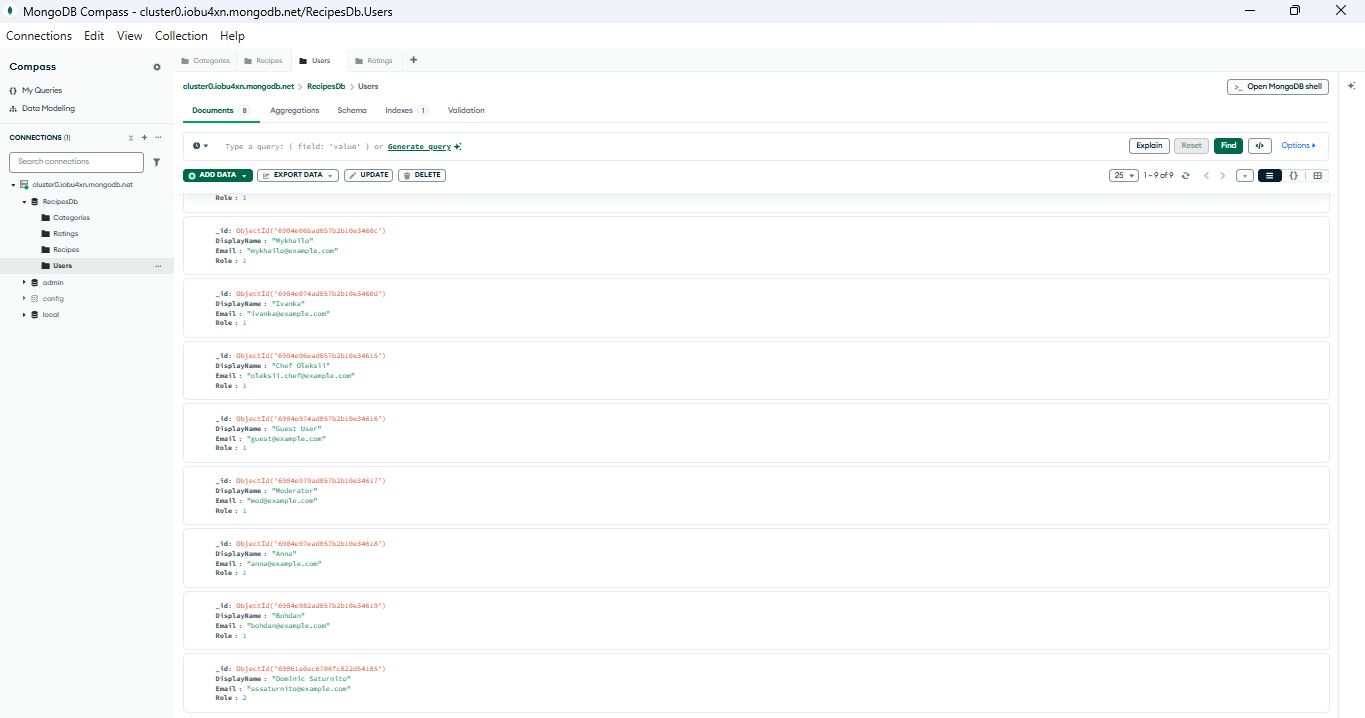
Отримаємо користувача по конкретному ID.

****

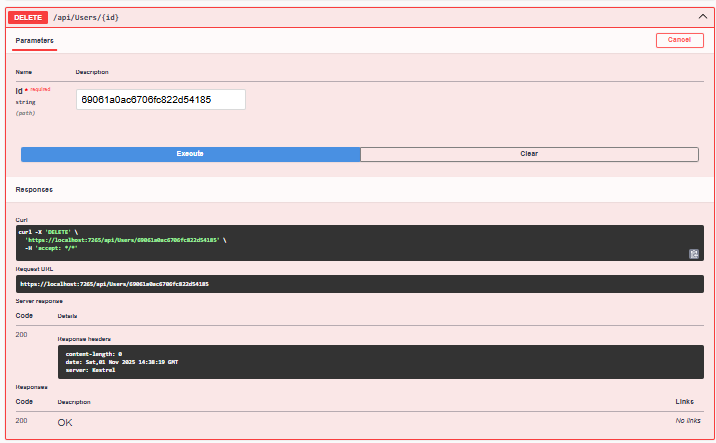
Оновлюємо користувача по конкретному ID.

****

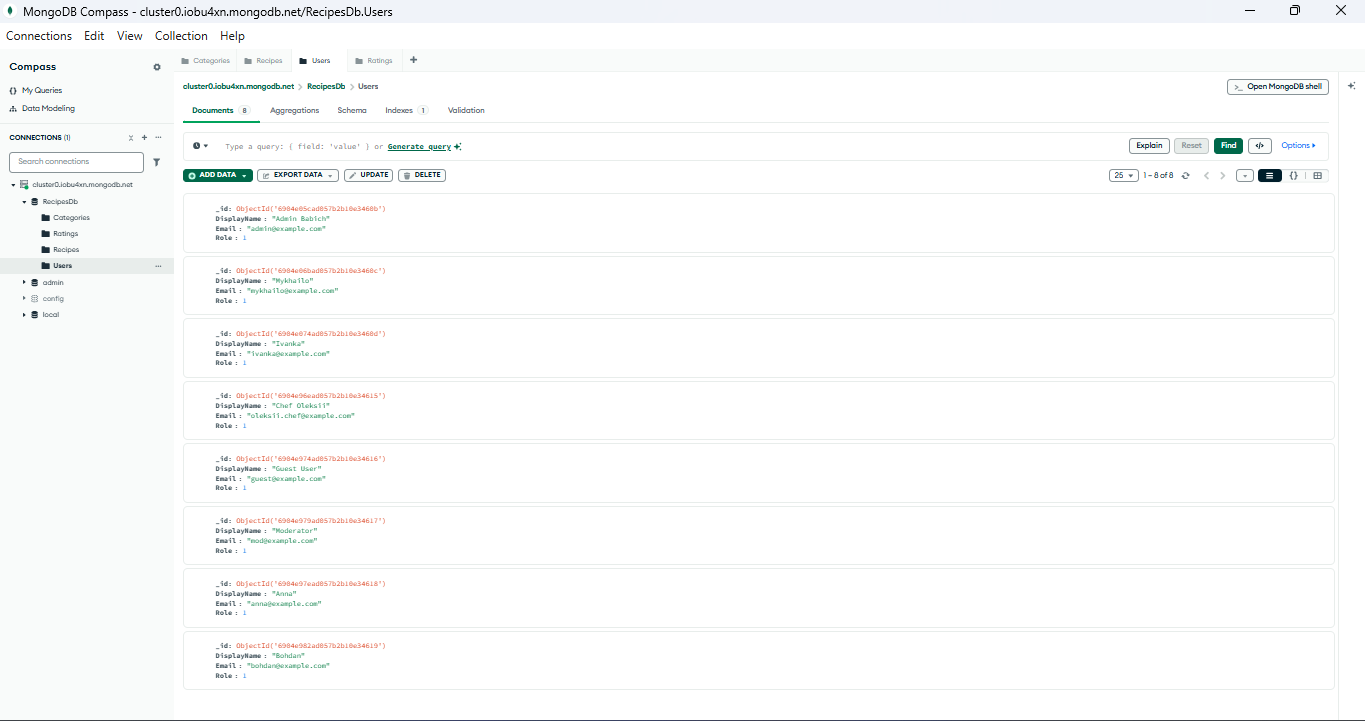
**Перевіримо в БД.**

****

Видалимо користувача по конкретному ID.

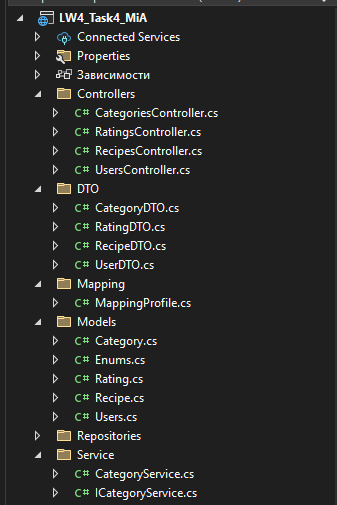
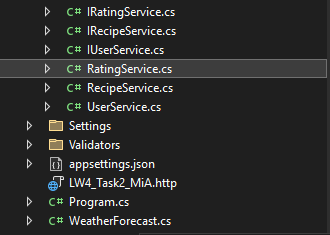
****

Перевіримо в БД.

****

**Завдання №5**

**Структура проєкту**

****

**Код програми:**

**Controllers**

**CategoriesController.cs:**

using AutoMapper;

using LW4\_Task4\_MiA.DTO;

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;

private readonly IMapper \_mapper;

public CategoriesController(ICategoryService service, IMapper mapper)

{

\_service = service;

\_mapper = mapper;

}

[HttpGet]

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

{

var items = await \_service.GetAllAsync();

return Ok(items);

}

[HttpGet("{id}")]

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

{

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

if (item is null) return NotFound();

return Ok(item);

}

[HttpPost]

public async Task<ActionResult<CategoryDto>> Create([FromBody] CategoryDto dto)

{

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

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

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

}

[HttpPut("{id}")]

public async Task<IActionResult> Update(string id, [FromBody] CategoryDto dto)

{

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

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

return ok ? Ok(dto) : NotFound();

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(string id)

{

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

return ok ? Ok() : NotFound();

}

}

}

**RatingsController.cs:**

using AutoMapper;

using LW4\_Task4\_MiA.DTO;

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;

private readonly IMapper \_mapper;

public RatingsController(IRatingService service, IMapper mapper)

{

\_service = service;

\_mapper = mapper;

}

[HttpGet]

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

{

var items = await \_service.GetAllAsync();

return Ok(items);

}

[HttpGet("{id}")]

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

{

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

return item is null ? NotFound() : Ok(item);

}

[HttpPost]

public async Task<ActionResult<RatingDto>> Create([FromBody] RatingDto dto)

{

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

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

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

}

[HttpPut("{id}")]

public async Task<IActionResult> Update(string id, [FromBody] RatingDto dto)

{

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

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

return ok ? Ok(dto) : NotFound();

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(string id)

{

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

return ok ? Ok() : NotFound();

}

}

}

**RecipesController.cs:**

using AutoMapper;

using LW4\_Task4\_MiA.DTO;

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;

private readonly IMapper \_mapper;

public RecipesController(IRecipeService service, IMapper mapper)

{

\_service = service;

\_mapper = mapper;

}

[HttpGet]

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

{

var items = await \_service.GetAllAsync();

return Ok(items);

}

[HttpGet("{id}")]

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

{

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

return item is null ? NotFound() : Ok(item);

}

[HttpPost]

public async Task<ActionResult<RecipeDto>> Create([FromBody] RecipeDto dto)

{

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

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

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

}

[HttpPut("{id}")]

public async Task<IActionResult> Update(string id, [FromBody] RecipeDto dto)

{

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

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

return ok ? Ok(dto) : NotFound();

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(string id)

{

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

return ok ? Ok() : NotFound();

}

}

}

**UsersController.cs:**

using AutoMapper;

using LW4\_Task4\_MiA.DTO;

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;

private readonly IMapper \_mapper;

public UsersController(IUserService service, IMapper mapper)

{

\_service = service;

\_mapper = mapper;

}

[HttpGet]

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

{

var users = await \_service.GetAllAsync();

return Ok(users);

}

[HttpGet("{id}")]

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

{

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

return user is null ? NotFound() : Ok(user);

}

[HttpPost]

public async Task<ActionResult<UserDto>> Create([FromBody] UserDto dto)

{

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

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

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

}

[HttpPut("{id}")]

public async Task<IActionResult> Update(string id, [FromBody] UserDto dto)

{

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

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

return ok ? Ok(dto) : NotFound();

}

[HttpDelete("{id}")]

public async Task<IActionResult> Delete(string id)

{

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

return ok ? Ok() : NotFound();

}

}

}

**DTO**

**CategoryDTO.cs:**

namespace LW4\_Task4\_MiA.DTO

{

public class CategoryDto

{

public string? Id { get; set; }

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

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

}

}

**RatingDTO.cs:**

namespace LW4\_Task4\_MiA.DTO

{

public class RatingDto

{

public string? Id { get; set; }

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

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

public int Value { get; set; }

public string? Comment { get; set; }

public string? UserName { get; set; }

public string? RecipeTitle { get; set; }

}

}

**RecipeDTO.cs:**

namespace LW4\_Task4\_MiA.DTO

{

public class RecipeDto

{

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 string Difficulty { get; set; } = string.Empty;

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

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

}

}

**UserDTO.cs:**

namespace LW4\_Task4\_MiA.DTO

{

public class UserDto

{

public string? Id { get; set; }

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

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

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

}

}

**Mapping**

**MappingProfile.cs:**

using AutoMapper;

using LW4\_Task2\_MiA.Models;

using LW4\_Task4\_MiA.DTO;

namespace LW4\_Task4\_MiA.Mapping

{

public class MappingProfile : Profile

{

public MappingProfile()

{

CreateMap<Category, CategoryDto>().ForMember(d => d.Type, opt => opt.MapFrom(s => s.Type.ToString())).ReverseMap();

CreateMap<Recipe, RecipeDto>().ForMember(d => d.Difficulty, opt => opt.MapFrom(s => s.Difficulty.ToString())).ReverseMap();

CreateMap<Rating, RatingDto>().ReverseMap();

CreateMap<User, UserDto>().ForMember(d => d.Role, opt => opt.MapFrom(s => s.Role.ToString())).ReverseMap();

}

}

}

**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;

}

}

**User.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 AutoMapper;

using LW4\_Task2\_MiA.Models;

using LW4\_Task2\_MiA.Repositories;

using LW4\_Task4\_MiA.DTO;

namespace LW4\_Task4\_MiA.Service

{

public class CategoryService : ICategoryService

{

private readonly IRepository<Category> \_repo;

private readonly IMapper \_mapper;

public CategoryService(IRepository<Category> repo, IMapper mapper)

{

\_repo = repo;

\_mapper = mapper;

}

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

=> \_mapper.Map<IEnumerable<CategoryDto>>(await \_repo.GetAllAsync());

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

=> \_mapper.Map<CategoryDto?>(await \_repo.GetByIdAsync(id));

public async Task<CategoryDto> CreateAsync(CategoryDto dto)

{

var entity = \_mapper.Map<Category>(dto);

await \_repo.CreateAsync(entity);

return \_mapper.Map<CategoryDto>(entity);

}

public async Task<bool> UpdateAsync(string id, CategoryDto dto)

{

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

if (entity is null) return false;

\_mapper.Map(dto, entity);

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

}

public async Task<bool> DeleteAsync(string id)

=> await \_repo.DeleteAsync(id);

}

}

**ICategoryService.cs:**

using LW4\_Task4\_MiA.DTO;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace LW4\_Task4\_MiA.Service

{

public interface ICategoryService

{

Task<IEnumerable<CategoryDto>> GetAllAsync();

Task<CategoryDto?> GetByIdAsync(string id);

Task<CategoryDto> CreateAsync(CategoryDto category);

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

Task<bool> DeleteAsync(string id);

}

}

**IRatingService.cs:**

using LW4\_Task4\_MiA.DTO;

namespace LW4\_Task4\_MiA.Service

{

public interface IRatingService

{

Task<IEnumerable<RatingDto>> GetAllAsync();

Task<RatingDto?> GetByIdAsync(string id);

Task<RatingDto> CreateAsync(RatingDto rating);

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

Task<bool> DeleteAsync(string id);

Task DeleteByRecipeIdAsync(string recipeId);

}

}

**IRecipeService.cs:**

using LW4\_Task4\_MiA.DTO;

namespace LW4\_Task4\_MiA.Service

{

public interface IRecipeService

{

Task<IEnumerable<RecipeDto>> GetAllAsync();

Task<RecipeDto?> GetByIdAsync(string id);

Task<RecipeDto> CreateAsync(RecipeDto recipe);

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

Task<bool> DeleteAsync(string id);

}

}

**IUserService.cs:**

using LW4\_Task4\_MiA.DTO;

namespace LW4\_Task4\_MiA.Service

{

public interface IUserService

{

Task<IEnumerable<UserDto>> GetAllAsync();

Task<UserDto?> GetByIdAsync(string id);

Task<UserDto> CreateAsync(UserDto user);

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

Task<bool> DeleteAsync(string id);

Task<UserDto?> GetByEmailAsync(string email);

}

}

**RatingService.cs:**

using AutoMapper;

using LW4\_Task2\_MiA.Models;

using LW4\_Task2\_MiA.Repositories;

using LW4\_Task4\_MiA.DTO;

using MongoDB.Driver;

namespace LW4\_Task4\_MiA.Service

{

public class RatingService : IRatingService

{

private readonly IRepository<Rating> \_repo;

private readonly IMongoCollection<User> \_users;

private readonly IMongoCollection<Recipe> \_recipes;

private readonly IMapper \_mapper;

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

{

\_repo = repo;

\_mapper = mapper;

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

\_recipes = db.GetCollection<Recipe>("Recipes");

}

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

{

var ratings = await \_repo.GetAllAsync();

var result = new List<RatingDto>();

foreach (var r in ratings)

{

var dto = \_mapper.Map<RatingDto>(r);

var user = await \_users.Find(u => u.Id == r.UserId).FirstOrDefaultAsync();

var recipe = await \_recipes.Find(rec => rec.Id == r.RecipeId).FirstOrDefaultAsync();

dto.UserName = user?.DisplayName ?? "(невідомий користувач)";

dto.RecipeTitle = recipe?.Title ?? "(невідомий рецепт)";

result.Add(dto);

}

return result;

}

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

{

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

if (rating is null) return null;

var dto = \_mapper.Map<RatingDto>(rating);

var user = await \_users.Find(u => u.Id == rating.UserId).FirstOrDefaultAsync();

var recipe = await \_recipes.Find(r => r.Id == rating.RecipeId).FirstOrDefaultAsync();

dto.UserName = user?.DisplayName ?? "(невідомий користувач)";

dto.RecipeTitle = recipe?.Title ?? "(невідомий рецепт)";

return dto;

}

public async Task<RatingDto> CreateAsync(RatingDto dto)

{

var rating = \_mapper.Map<Rating>(dto);

rating.Id = null;

await \_repo.CreateAsync(rating);

return \_mapper.Map<RatingDto>(rating);

}

public async Task<bool> UpdateAsync(string id, RatingDto dto)

{

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

if (existing is null) return false;

var updated = \_mapper.Map<Rating>(dto);

updated.Id = id;

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

}

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 \_recipes.Database.GetCollection<Rating>("Ratings").DeleteManyAsync(filter);

}

}

}

**RecipeService.cs:**

using AutoMapper;

using LW4\_Task2\_MiA.Models;

using LW4\_Task2\_MiA.Repositories;

using LW4\_Task4\_MiA.DTO;

using MongoDB.Driver;

namespace LW4\_Task4\_MiA.Service

{

public class RecipeService : IRecipeService

{

private readonly IRepository<Recipe> \_repo;

private readonly IMongoCollection<Category> \_categories;

private readonly IMongoCollection<User> \_users;

private readonly IRatingService \_ratings;

private readonly IMapper \_mapper;

public RecipeService(IRepository<Recipe> repo, IRatingService ratings, IMongoDatabase db, IMapper mapper)

{

\_repo = repo;

\_ratings = ratings;

\_mapper = mapper;

\_categories = db.GetCollection<Category>("Categories");

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

}

// Всі рецепти як RecipeDto

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

{

var recipes = await \_repo.GetAllAsync();

var result = new List<RecipeDto>();

foreach (var r in recipes)

{

var dto = \_mapper.Map<RecipeDto>(r);

var category = await \_categories.Find(c => c.Id == r.CategoryId).FirstOrDefaultAsync();

var author = await \_users.Find(u => u.Id == r.AuthorUserId).FirstOrDefaultAsync();

dto.CategoryId = category?.Name ?? "(невідома категорія)";

dto.AuthorUserId = author?.DisplayName ?? "(невідомий автор)";

result.Add(dto);

}

return result;

}

// Один рецепт як RecipeDto

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

{

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

if (recipe is null) return null;

var dto = \_mapper.Map<RecipeDto>(recipe);

var category = await \_categories.Find(c => c.Id == recipe.CategoryId).FirstOrDefaultAsync();

var author = await \_users.Find(u => u.Id == recipe.AuthorUserId).FirstOrDefaultAsync();

dto.CategoryId = category?.Name ?? "(невідома категорія)";

dto.AuthorUserId = author?.DisplayName ?? "(невідомий автор)";

return dto;

}

public async Task<RecipeDto> CreateAsync(RecipeDto recipeDto)

{

var recipe = \_mapper.Map<Recipe>(recipeDto);

recipe.Id = null;

await \_repo.CreateAsync(recipe);

return \_mapper.Map<RecipeDto>(recipe);

}

public async Task<bool> UpdateAsync(string id, RecipeDto recipeDto)

{

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

if (existing is null) return false;

var updated = \_mapper.Map<Recipe>(recipeDto);

updated.Id = id;

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

}

public async Task<bool> DeleteAsync(string id)

{

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

if (existing is null) return false;

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

if (ok)

await \_ratings.DeleteByRecipeIdAsync(id);

return ok;

}

}

}

**UserService.cs:**

using AutoMapper;

using LW4\_Task2\_MiA.Models;

using LW4\_Task2\_MiA.Repositories;

using LW4\_Task4\_MiA.DTO;

using MongoDB.Driver;

namespace LW4\_Task4\_MiA.Service

{

public class UserService : IUserService

{

private readonly Irepository<User> \_repo;

private readonly ImongoCollection<User> \_collection;

private readonly Imapper \_mapper;

public UserService(Irepository<User> repo, ImongoDatabase db, Imapper mapper)

{

\_repo = repo;

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

\_mapper = mapper;

}

public async Task<Ienumerable<UserDto>> GetAllAsync()

=> \_mapper.Map<Ienumerable<UserDto>>(await \_repo.GetAllAsync());

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

=> \_mapper.Map<UserDto?>(await \_repo.GetByIdAsync(id));

public async Task<UserDto> CreateAsync(UserDto dto)

{

var entity = \_mapper.Map<User>(dto);

entity.Id = null;

await \_repo.CreateAsync(entity);

return \_mapper.Map<UserDto>(entity);

}

public async Task<bool> UpdateAsync(string id, UserDto dto)

{

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

if (entity is null) return false;

\_mapper.Map(dto, entity);

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

}

public async Task<bool> DeleteAsync(string id)

=> await \_repo.DeleteAsync(id);

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

{

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

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

return \_mapper.Map<UserDto?>(user);

}

}

}

**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**

**RatingValidator.cs:**

using LW4\_Task2\_MiA.Models;

using FluentValidation;

using System;

namespace LW4\_Task2\_MiA.Validators

{

public class RatingValidator : AbstractValidator<Rating>

{

public RatingValidator()

{

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

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

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();

//рядкові ObjectId

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

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

}

}

}

**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.Mapping;

using LW4\_Task4\_MiA.Service;

using LW4\_Task4\_MiA.Settings;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Options;

using MongoDB.Driver;

var builder = WebApplication.CreateBuilder(args);

// 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);

});

// 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”));

// Сервіси

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

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

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

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

//AutoMapper

builder.Services.AddAutoMapper(cfg =>

{

cfg.AddProfile<MappingProfile>();

});

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();

}

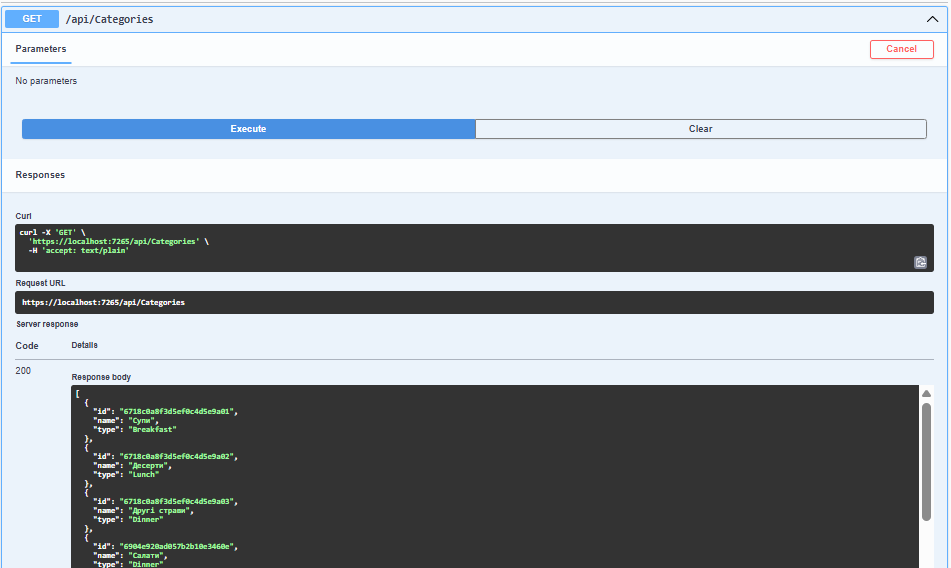
app.UseHttpsRedirection();

app.MapControllers();

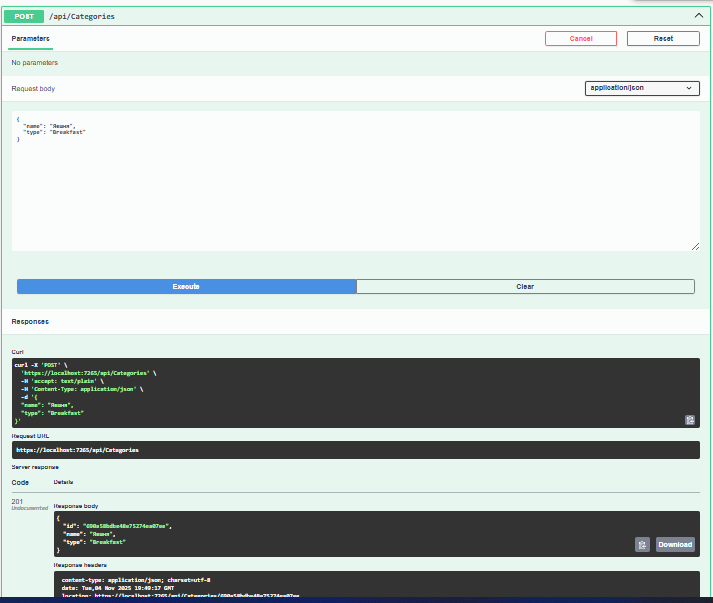
app.Run();

**Результати програми:**

Отримаємо усі категорії. Як бачимо поле Type тепер відображається текстом а не цифрою, отже підхід (патерн) працює.

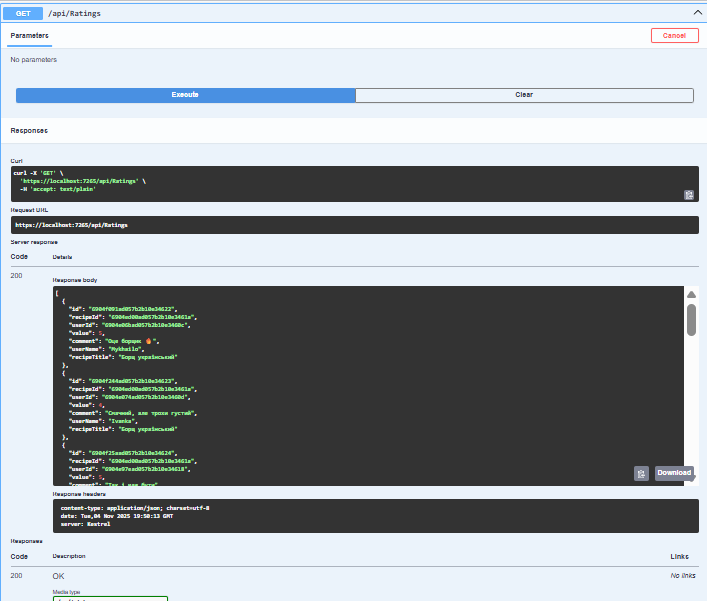


Запостимо нову категорію



Рейтинги

Отримаємо всі рейтинги. Патерн працює успішно.

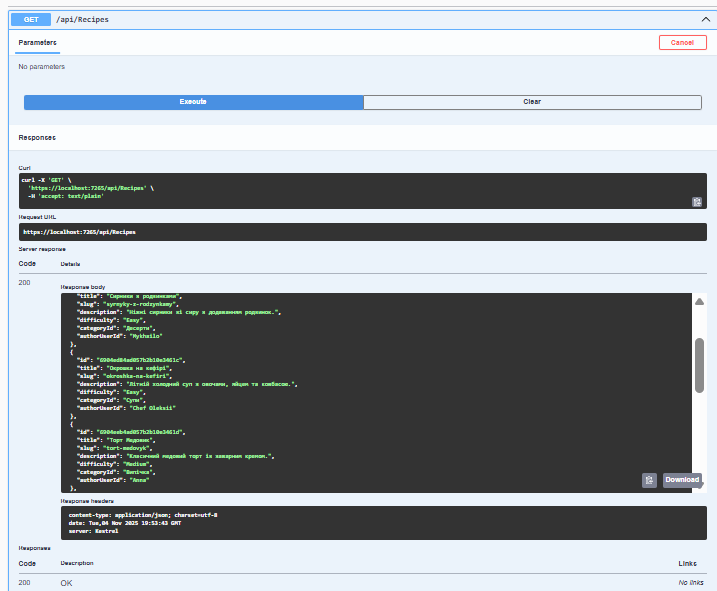


Запостимо рейтинг.

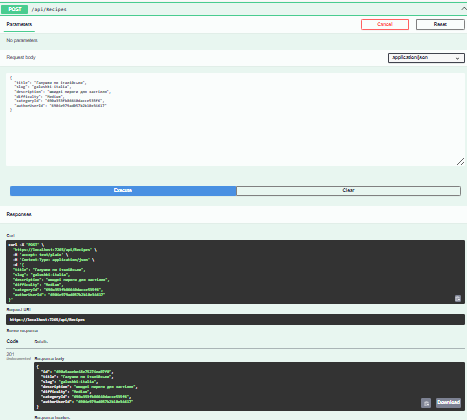


Рецепти

Отримаємо усі рецепти. Патерн працює (підтягує дані в текстовому форматі).

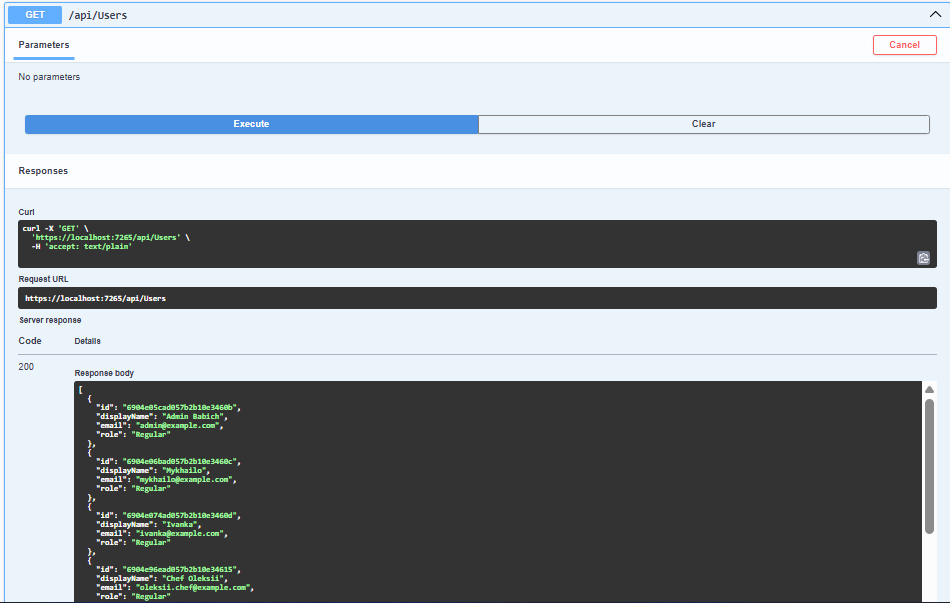


Запостимо новий рецепт.

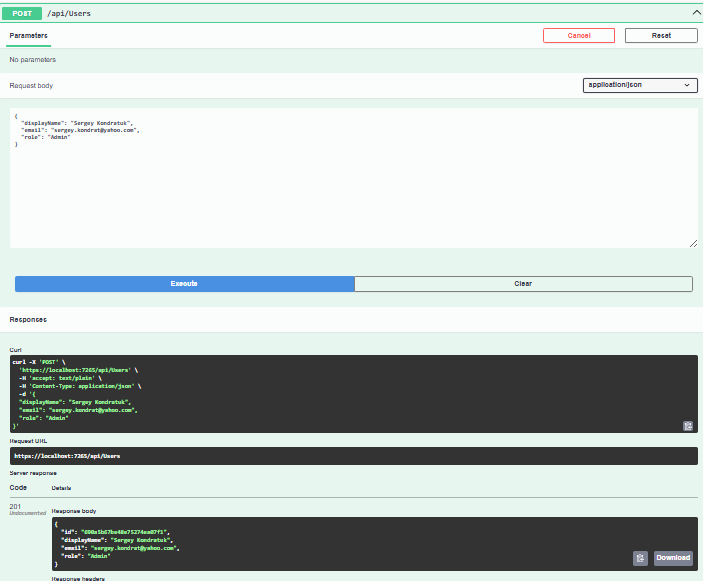


Користувачі

Отримаємо усіх користувачів. Патерн працює (показує роль в текстовому форматі).

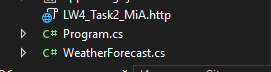
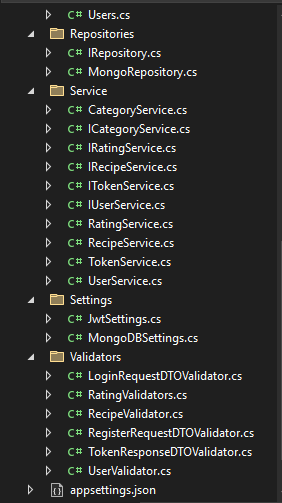
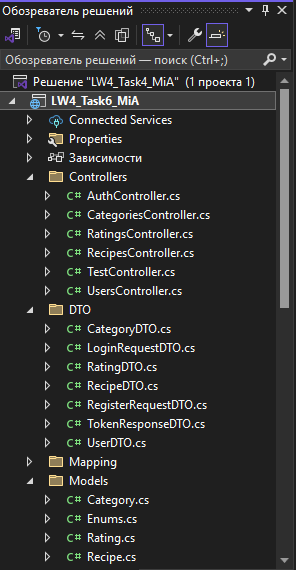


Запостимо нового користувача.



**Завдання №6**

**Структура проєкту**

****

**Controllers**

**AuthController.cs**

using AutoMapper;

using LW4\_Task2\_MiA.Controllers;

using LW4\_Task2\_MiA.Models;

using LW4\_Task4\_MiA.DTO;

using LW4\_Task4\_MiA.Service;

using LW4\_Task6\_MiA.DTO;

using LW4\_Task6\_MiA.Service;

using LW4\_Task6\_MiA.Settings;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.Extensions.Options;

using System.Security.Claims;

namespace LW4\_Task6\_MiA.Controllers

{

[ApiController]

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

public class AuthController : ControllerBase

{

private readonly IUserService \_userService;

private readonly ITokenService \_tokenService;

private readonly IMapper \_mapper;

private readonly int \_refreshTokenExpirationDays;

public AuthController(IUserService userService, ITokenService tokenService, IMapper mapper, IOptions<JwtSettings> jwtSettings)

{

\_userService = userService;

\_tokenService = tokenService;

\_mapper = mapper;

\_refreshTokenExpirationDays = jwtSettings.Value.RefreshTokenExpirationDays;

}

[HttpGet("me")]

[Authorize]

public async Task<ActionResult<UserDto>> GetCurrentUser()

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

if (string.IsNullOrEmpty(userId))

{

return Unauthorized();

}

var userDto = await \_userService.GetByIdAsync(userId);

if (userDto == null)

{

return NotFound("Користувача, пов'язаного з цим токеном, не знайдено.");

}

return Ok(userDto);

}

[HttpPost("register")]

[AllowAnonymous]

public async Task<ActionResult<UserDto>> Register([FromBody] RegisterRequestDto registerDto)

{

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

var createdUser = await \_userService.RegisterAsync(registerDto);

if (createdUser == null)

{

return BadRequest(new { message = "Цей email вже використовується." });

}

return CreatedAtAction(

nameof(UsersController.GetById),

"Users",

new { id = createdUser.Id },

createdUser);

}

[HttpPost("login")]

[AllowAnonymous]

public async Task<ActionResult<TokenResponseDto>> Login([FromBody] LoginRequestDto loginRequest)

{

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

var user = await \_userService.ValidateAndGetUserAsync(loginRequest.Email, loginRequest.Password);

if (user == null)

{

return Unauthorized("Невірний email або пароль.");

}

var accessToken = \_tokenService.CreateAccessToken(user);

var refreshToken = \_tokenService.CreateRefreshToken();

user.RefreshToken = refreshToken;

user.RefreshTokenExpiryTime = DateTime.UtcNow.AddDays(\_refreshTokenExpirationDays);

await \_userService.UpdateAsync(user.Id!, \_mapper.Map<UserDto>(user));

return Ok(new TokenResponseDto

{

AccessToken = accessToken,

RefreshToken = refreshToken

});

}

[HttpPost("refresh")]

[AllowAnonymous]

public async Task<ActionResult<TokenResponseDto>> Refresh([FromBody] TokenResponseDto tokenDto)

{

var principal = \_tokenService.GetPrincipalFromExpiredToken(tokenDto.AccessToken);

if (principal?.Identity?.Name is null)

{

return Unauthorized("Недійсний access token.");

}

var userId = principal.Identity.Name;

var userDto = await \_userService.GetByIdAsync(userId);

var user = \_mapper.Map<User>(userDto);

if (user == null || user.RefreshToken != tokenDto.RefreshToken || user.RefreshTokenExpiryTime <= DateTime.UtcNow)

{

return Unauthorized("Недійсний refresh token.");

}

var newAccessToken = \_tokenService.CreateAccessToken(user);

var newRefreshToken = \_tokenService.CreateRefreshToken();

user.RefreshToken = newRefreshToken;

user.RefreshTokenExpiryTime = DateTime.UtcNow.AddDays(\_refreshTokenExpirationDays);

await \_userService.UpdateAsync(user.Id!, \_mapper.Map<UserDto>(user));

return Ok(new TokenResponseDto

{

AccessToken = newAccessToken,

RefreshToken = newRefreshToken

});

}

}

}

**CategoriesController.cs:**

using AutoMapper;

using LW4\_Task4\_MiA.DTO;

using LW4\_Task4\_MiA.Service;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

namespace LW4\_Task2\_MiA.Controllers

{

[ApiController]

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

[Authorize]

public class CategoriesController : ControllerBase

{

private readonly ICategoryService \_service;

private readonly IMapper \_mapper;

public CategoriesController(ICategoryService service, IMapper mapper)

{

\_service = service;

\_mapper = mapper;

}

[HttpGet]

[AllowAnonymous]

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

{

var items = await \_service.GetAllAsync();

return Ok(items);

}

[HttpGet("{id}")]

[AllowAnonymous]

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

{

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

if (item is null) return NotFound();

return Ok(item);

}

[HttpPost]

[Authorize(Roles = "Admin,Moderator")]

public async Task<ActionResult<CategoryDto>> Create([FromBody] CategoryDto dto)

{

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

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

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

}

[HttpPut("{id}")]

[Authorize(Roles = "Admin,Moderator")]

public async Task<IActionResult> Update(string id, [FromBody] CategoryDto dto)

{

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

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

return ok ? Ok(dto) : NotFound();

}

[HttpDelete("{id}")]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Delete(string id)

{

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

return ok ? Ok() : NotFound();

}

}

}

**RatingsController.cs:**

using AutoMapper;

using LW4\_Task4\_MiA.DTO;

using LW4\_Task4\_MiA.Service;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

namespace LW4\_Task2\_MiA.Controllers

{

[ApiController]

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

public class RatingsController : ControllerBase

{

private readonly IRatingService \_service;

private readonly IMapper \_mapper;

public RatingsController(IRatingService service, IMapper mapper)

{

\_service = service;

\_mapper = mapper;

}

[HttpGet]

[AllowAnonymous]

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

{

var items = await \_service.GetAllAsync();

return Ok(items);

}

[HttpGet("{id}")]

[AllowAnonymous]

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

{

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

return item is null ? NotFound() : Ok(item);

}

[HttpPost]

[Authorize]

public async Task<ActionResult<RatingDto>> Create([FromBody] RatingDto dto)

{

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

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

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

}

[HttpPut("{id}")]

[Authorize(Roles = "Admin,Moderator")]

public async Task<IActionResult> Update(string id, [FromBody] RatingDto dto)

{

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

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

return ok ? Ok(dto) : NotFound();

}

[HttpDelete("{id}")]

[Authorize(Roles = "Admin,Moderator")]

public async Task<IActionResult> Delete(string id)

{

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

return ok ? Ok() : NotFound();

}

}

}

**RecipesController.cs:**

using AutoMapper;

using LW4\_Task4\_MiA.DTO;

using LW4\_Task4\_MiA.Service;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

namespace LW4\_Task2\_MiA.Controllers

{

[ApiController]

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

public class RecipesController : ControllerBase

{

private readonly IRecipeService \_service;

private readonly IMapper \_mapper;

public RecipesController(IRecipeService service, IMapper mapper)

{

\_service = service;

\_mapper = mapper;

}

[HttpGet]

[AllowAnonymous]

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

{

var items = await \_service.GetAllAsync();

return Ok(items);

}

[HttpGet("{id}")]

[AllowAnonymous]

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

{

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

return item is null ? NotFound() : Ok(item);

}

[HttpPost]

[Authorize]

public async Task<ActionResult<RecipeDto>> Create([FromBody] RecipeDto dto)

{

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

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

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

}

[HttpPut("{id}")]

[Authorize(Roles = "Admin,Moderator")]

public async Task<IActionResult> Update(string id, [FromBody] RecipeDto dto)

{

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

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

return ok ? Ok(dto) : NotFound();

}

[HttpDelete("{id}")]

[Authorize(Roles = "Admin,Moderator")]

public async Task<IActionResult> Delete(string id)

{

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

return ok ? Ok() : NotFound();

}

}

}

**TestController.cs:**

using LW4\_Task2\_MiA.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System.Security.Claims;

namespace LW4\_Task6\_MiA.Controllers

{

[ApiController]

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

public class TestController : ControllerBase

{

[HttpGet("protected")]

[Authorize]

public IActionResult GetProtectedData()

{

var userId = User.FindFirstValue(ClaimTypes.NameIdentifier);

var email = User.FindFirstValue(ClaimTypes.Email);

var role = User.FindFirstValue(ClaimTypes.Role);

return Ok(new

{

Message = "Це захищені дані.",

UserId = userId,

Email = email,

Role = role

});

}

}

}

**UsersController.cs:**

using AutoMapper;

using LW4\_Task4\_MiA.DTO;

using LW4\_Task4\_MiA.Service;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.AspNetCore.Authorization;

namespace LW4\_Task2\_MiA.Controllers

{

[ApiController]

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

[Authorize(Roles = "Admin,Moderator")]

public class UsersController : ControllerBase

{

private readonly IUserService \_service;

private readonly IMapper \_mapper;

public UsersController(IUserService service, IMapper mapper)

{

\_service = service;

\_mapper = mapper;

}

[HttpGet]

[Authorize(Roles = "Admin,Moderator")]

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

{

var users = await \_service.GetAllAsync();

return Ok(users);

}

[HttpGet("{id}")]

[Authorize(Roles = "Admin,Moderator")]

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

{

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

return user is null ? NotFound() : Ok(user);

}

[HttpPost]

[Authorize(Roles = "Admin")]

public async Task<ActionResult<UserDto>> Create([FromBody] UserDto dto)

{

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

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

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

}

[HttpPut("{id}")]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Update(string id, [FromBody] UserDto dto)

{

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

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

return ok ? Ok(dto) : NotFound();

}

[HttpDelete("{id}")]

[Authorize(Roles = "Admin")]

public async Task<IActionResult> Delete(string id)

{

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

return ok ? Ok() : NotFound();

}

}

}

**DTO**

**CategoryDTO.cs:**

namespace LW4\_Task4\_MiA.DTO

{

public class CategoryDto

{

public string? Id { get; set; }

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

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

}

}

**LoginRequestDTO.cs:**

using System.ComponentModel.DataAnnotations;

namespace LW4\_Task6\_MiA.DTO

{

public class LoginRequestDto

{

[Required]

[EmailAddress]

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

[Required]

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

}

}

**RatingDTO.cs:**

namespace LW4\_Task4\_MiA.DTO

{

public class RatingDto

{

public string? Id { get; set; }

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

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

public int Value { get; set; }

public string? Comment { get; set; }

public string? UserName { get; set; }

public string? RecipeTitle { get; set; }

}

}

**RecipeDTO.cs:**

namespace LW4\_Task4\_MiA.DTO

{

public class RecipeDto

{

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 string Difficulty { get; set; } = string.Empty;

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

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

}

}

**RegisterRequestDTO.cs:**

namespace LW4\_Task6\_MiA.DTO

{

public class RegisterRequestDto

{

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

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

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

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

}

}

**TokenResponseDTO.cs:**

namespace LW4\_Task6\_MiA.DTO

{

public class TokenResponseDto

{

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

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

}

}

**UserDTO.cs:**

namespace LW4\_Task4\_MiA.DTO

{

public class UserDto

{

public string? Id { get; set; }

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

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

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

}

}

**Mapping**

**MapingProfile.cs:**

using AutoMapper;

using LW4\_Task2\_MiA.Models;

using LW4\_Task4\_MiA.DTO;

namespace LW4\_Task4\_MiA.Mapping

{

public class MappingProfile : Profile

{

public MappingProfile()

{

CreateMap<Category, CategoryDto>().ForMember(d => d.Type, opt => opt.MapFrom(s => s.Type.ToString())).ReverseMap();

CreateMap<Recipe, RecipeDto>().ForMember(d => d.Difficulty, opt => opt.MapFrom(s => s.Difficulty.ToString())).ReverseMap();

CreateMap<Rating, RatingDto>().ReverseMap();

CreateMap<User, UserDto>().ForMember(d => d.Role, opt => opt.MapFrom(s => s.Role.ToString())).ReverseMap();

}

}

}

**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;

public string? PasswordHash { get; set; }

public string? RefreshToken { get; set; }

public DateTime? RefreshTokenExpiryTime { get; set; }

}

}

**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 AutoMapper;

using LW4\_Task2\_MiA.Models;

using LW4\_Task2\_MiA.Repositories;

using LW4\_Task4\_MiA.DTO;

namespace LW4\_Task4\_MiA.Service

{

public class CategoryService : ICategoryService

{

private readonly IRepository<Category> \_repo;

private readonly IMapper \_mapper;

public CategoryService(IRepository<Category> repo, IMapper mapper)

{

\_repo = repo;

\_mapper = mapper;

}

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

=> \_mapper.Map<IEnumerable<CategoryDto>>(await \_repo.GetAllAsync());

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

=> \_mapper.Map<CategoryDto?>(await \_repo.GetByIdAsync(id));

public async Task<CategoryDto> CreateAsync(CategoryDto dto)

{

var entity = \_mapper.Map<Category>(dto);

await \_repo.CreateAsync(entity);

return \_mapper.Map<CategoryDto>(entity);

}

public async Task<bool> UpdateAsync(string id, CategoryDto dto)

{

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

if (entity is null) return false;

\_mapper.Map(dto, entity);

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

}

public async Task<bool> DeleteAsync(string id)

=> await \_repo.DeleteAsync(id);

}

}

**ICategoryService.cs:**

using LW4\_Task4\_MiA.DTO;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace LW4\_Task4\_MiA.Service

{

public interface ICategoryService

{

Task<IEnumerable<CategoryDto>> GetAllAsync();

Task<CategoryDto?> GetByIdAsync(string id);

Task<CategoryDto> CreateAsync(CategoryDto category);

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

Task<bool> DeleteAsync(string id);

}

}

**IRatingService.cs:**

using LW4\_Task4\_MiA.DTO;

namespace LW4\_Task4\_MiA.Service

{

public interface IRatingService

{

Task<IEnumerable<RatingDto>> GetAllAsync();

Task<RatingDto?> GetByIdAsync(string id);

Task<RatingDto> CreateAsync(RatingDto rating);

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

Task<bool> DeleteAsync(string id);

Task DeleteByRecipeIdAsync(string recipeId);

}

}

**IRecipeService.cs:**

using LW4\_Task4\_MiA.DTO;

namespace LW4\_Task4\_MiA.Service

{

public interface IRecipeService

{

Task<IEnumerable<RecipeDto>> GetAllAsync();

Task<RecipeDto?> GetByIdAsync(string id);

Task<RecipeDto> CreateAsync(RecipeDto recipe);

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

Task<bool> DeleteAsync(string id);

}

}

**ITokenService.cs:**

using LW4\_Task2\_MiA.Models;

using System.Security.Claims;

namespace LW4\_Task6\_MiA.Service

{

public interface ITokenService

{

string CreateAccessToken(User user);

string CreateRefreshToken();

ClaimsPrincipal? GetPrincipalFromExpiredToken(string token);

}

}

**IUserService.cs:**

using LW4\_Task2\_MiA.Models;

using LW4\_Task4\_MiA.DTO;

using LW4\_Task6\_MiA.DTO;

namespace LW4\_Task4\_MiA.Service

{

public interface IUserService

{

Task<IEnumerable<UserDto>> GetAllAsync();

Task<UserDto?> GetByIdAsync(string id);

Task<UserDto> CreateAsync(UserDto user);

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

Task<bool> DeleteAsync(string id);

Task<UserDto?> GetByEmailAsync(string email);

Task<UserDto?> RegisterAsync(RegisterRequestDto dto);

Task<User?> ValidateAndGetUserAsync(string email, string password);

}

}

**RatingService.cs:**

using AutoMapper;

using LW4\_Task2\_MiA.Models;

using LW4\_Task2\_MiA.Repositories;

using LW4\_Task4\_MiA.DTO;

using MongoDB.Driver;

namespace LW4\_Task4\_MiA.Service

{

public class RatingService : IRatingService

{

private readonly IRepository<Rating> \_repo;

private readonly IMongoCollection<User> \_users;

private readonly IMongoCollection<Recipe> \_recipes;

private readonly IMapper \_mapper;

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

{

\_repo = repo;

\_mapper = mapper;

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

\_recipes = db.GetCollection<Recipe>("Recipes");

}

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

{

var ratings = await \_repo.GetAllAsync();

var result = new List<RatingDto>();

foreach (var r in ratings)

{

var dto = \_mapper.Map<RatingDto>(r);

var user = await \_users.Find(u => u.Id == r.UserId).FirstOrDefaultAsync();

var recipe = await \_recipes.Find(rec => rec.Id == r.RecipeId).FirstOrDefaultAsync();

dto.UserName = user?.DisplayName ?? "(невідомий користувач)";

dto.RecipeTitle = recipe?.Title ?? "(невідомий рецепт)";

result.Add(dto);

}

return result;

}

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

{

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

if (rating is null) return null;

var dto = \_mapper.Map<RatingDto>(rating);

var user = await \_users.Find(u => u.Id == rating.UserId).FirstOrDefaultAsync();

var recipe = await \_recipes.Find(r => r.Id == rating.RecipeId).FirstOrDefaultAsync();

dto.UserName = user?.DisplayName ?? "(невідомий користувач)";

dto.RecipeTitle = recipe?.Title ?? "(невідомий рецепт)";

return dto;

}

public async Task<RatingDto> CreateAsync(RatingDto dto)

{

var rating = \_mapper.Map<Rating>(dto);

rating.Id = null;

await \_repo.CreateAsync(rating);

return \_mapper.Map<RatingDto>(rating);

}

public async Task<bool> UpdateAsync(string id, RatingDto dto)

{

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

if (existing is null) return false;

var updated = \_mapper.Map<Rating>(dto);

updated.Id = id;

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

}

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 \_recipes.Database.GetCollection<Rating>("Ratings").DeleteManyAsync(filter);

}

}

}

**RecipeService.cs:**

using AutoMapper;

using LW4\_Task2\_MiA.Models;

using LW4\_Task2\_MiA.Repositories;

using LW4\_Task4\_MiA.DTO;

using MongoDB.Driver;

namespace LW4\_Task4\_MiA.Service

{

public class RecipeService : IRecipeService

{

private readonly IRepository<Recipe> \_repo;

private readonly IMongoCollection<Category> \_categories;

private readonly IMongoCollection<User> \_users;

private readonly IRatingService \_ratings;

private readonly IMapper \_mapper;

public RecipeService(IRepository<Recipe> repo, IRatingService ratings, IMongoDatabase db, IMapper mapper)

{

\_repo = repo;

\_ratings = ratings;

\_mapper = mapper;

\_categories = db.GetCollection<Category>("Categories");

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

}

// Всі рецепти як RecipeDto

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

{

var recipes = await \_repo.GetAllAsync();

var result = new List<RecipeDto>();

foreach (var r in recipes)

{

var dto = \_mapper.Map<RecipeDto>(r);

var category = await \_categories.Find(c => c.Id == r.CategoryId).FirstOrDefaultAsync();

var author = await \_users.Find(u => u.Id == r.AuthorUserId).FirstOrDefaultAsync();

dto.CategoryId = category?.Name ?? "(невідома категорія)";

dto.AuthorUserId = author?.DisplayName ?? "(невідомий автор)";

result.Add(dto);

}

return result;

}

// Один рецепт як RecipeDto

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

{

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

if (recipe is null) return null;

var dto = \_mapper.Map<RecipeDto>(recipe);

var category = await \_categories.Find(c => c.Id == recipe.CategoryId).FirstOrDefaultAsync();

var author = await \_users.Find(u => u.Id == recipe.AuthorUserId).FirstOrDefaultAsync();

dto.CategoryId = category?.Name ?? "(невідома категорія)";

dto.AuthorUserId = author?.DisplayName ?? "(невідомий автор)";

return dto;

}

public async Task<RecipeDto> CreateAsync(RecipeDto recipeDto)

{

var recipe = \_mapper.Map<Recipe>(recipeDto);

recipe.Id = null;

await \_repo.CreateAsync(recipe);

return \_mapper.Map<RecipeDto>(recipe);

}

public async Task<bool> UpdateAsync(string id, RecipeDto recipeDto)

{

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

if (existing is null) return false;

var updated = \_mapper.Map<Recipe>(recipeDto);

updated.Id = id;

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

}

public async Task<bool> DeleteAsync(string id)

{

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

if (existing is null) return false;

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

if (ok)

await \_ratings.DeleteByRecipeIdAsync(id);

return ok;

}

}

}

**TokenService.cs:**

using LW4\_Task2\_MiA.Models;

using LW4\_Task6\_MiA.Settings;

using Microsoft.Extensions.Options;

using System.IdentityModel.Tokens.Jwt;

using Microsoft.IdentityModel.Tokens;

using System.Security.Claims;

using System.Security.Cryptography;

using System.Text;

namespace LW4\_Task6\_MiA.Service

{

public class TokenService : ITokenService

{

private readonly JwtSettings \_jwtSettings;

private readonly SymmetricSecurityKey \_key;

public TokenService(IOptions<JwtSettings> jwtSettings)

{

\_jwtSettings = jwtSettings.Value;

\_key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_jwtSettings.Secret));

}

public string CreateAccessToken(User user)

{

var claims = new List<Claim>

{

new(JwtRegisteredClaimNames.Sub, user.Id ?? string.Empty),

new(JwtRegisteredClaimNames.Email, user.Email),

new(JwtRegisteredClaimNames.Name, user.DisplayName),

new(ClaimTypes.Role, user.Role.ToString())

};

var creds = new SigningCredentials(\_key, SecurityAlgorithms.HmacSha512Signature);

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(claims),

Expires = DateTime.UtcNow.AddMinutes(\_jwtSettings.AccessTokenExpirationMinutes),

Issuer = \_jwtSettings.Issuer,

Audience = \_jwtSettings.Audience,

SigningCredentials = creds

};

var tokenHandler = new JwtSecurityTokenHandler();

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

}

public string CreateRefreshToken()

{

var randomNumber = new byte[64];

using var rng = RandomNumberGenerator.Create();

rng.GetBytes(randomNumber);

return Convert.ToBase64String(randomNumber);

}

public ClaimsPrincipal? GetPrincipalFromExpiredToken(string token)

{

var tokenValidationParameters = new TokenValidationParameters

{

ValidateAudience = true,

ValidateIssuer = true,

ValidateIssuerSigningKey = true,

IssuerSigningKey = \_key,

ValidateLifetime = false,

ValidIssuer = \_jwtSettings.Issuer,

ValidAudience = \_jwtSettings.Audience,

RoleClaimType = ClaimTypes.Role

};

var tokenHandler = new JwtSecurityTokenHandler();

try

{

var principal = tokenHandler.ValidateToken(token, tokenValidationParameters, out var securityToken);

if (securityToken is not JwtSecurityToken jwtSecurityToken ||

!jwtSecurityToken.Header.Alg.Equals(SecurityAlgorithms.HmacSha512Signature, StringComparison.InvariantCultureIgnoreCase))

{

return null;

}

return principal;

}

catch

{

return null;

}

}

}

}

**UserService.cs:**

using AutoMapper;

using LW4\_Task2\_MiA.Models;

using LW4\_Task2\_MiA.Repositories;

using LW4\_Task4\_MiA.DTO;

using LW4\_Task6\_MiA.DTO;

using MongoDB.Driver;

namespace LW4\_Task4\_MiA.Service

{

public class UserService : IUserService

{

private readonly IRepository<User> \_repo;

private readonly IMongoCollection<User> \_collection;

private readonly IMapper \_mapper;

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

{

\_repo = repo;

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

\_mapper = mapper;

}

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

=> \_mapper.Map<IEnumerable<UserDto>>(await \_repo.GetAllAsync());

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

=> \_mapper.Map<UserDto?>(await \_repo.GetByIdAsync(id));

public async Task<UserDto> CreateAsync(UserDto dto)

{

var entity = \_mapper.Map<User>(dto);

entity.Id = null;

await \_repo.CreateAsync(entity);

return \_mapper.Map<UserDto>(entity);

}

public async Task<bool> UpdateAsync(string id, UserDto dto)

{

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

if (entity is null) return false;

\_mapper.Map(dto, entity);

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

}

public async Task<bool> DeleteAsync(string id)

=> await \_repo.DeleteAsync(id);

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

{

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

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

return \_mapper.Map<UserDto?>(user);

}

public async Task<UserDto?> RegisterAsync(RegisterRequestDto dto)

{

var existingUser = await GetByEmailAsync(dto.Email);

if (existingUser != null)

{

return null;

}

string passwordHash = BCrypt.Net.BCrypt.HashPassword(dto.Password);

var user = new User

{

Id = null,

DisplayName = dto.DisplayName,

Email = dto.Email,

PasswordHash = passwordHash,

Role = UserRole.Regular

};

await \_repo.CreateAsync(user);

return \_mapper.Map<UserDto>(user);

}

public async Task<User?> ValidateAndGetUserAsync(string email, string password)

{

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

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

if (user == null || user.PasswordHash == null)

{

return null;

}

bool isPasswordValid = BCrypt.Net.BCrypt.Verify(password, user.PasswordHash);

return isPasswordValid ? user : null;

}

}

}

**Settings**

**JwtSettings.cs:**

namespace LW4\_Task6\_MiA.Settings

{

public class JwtSettings

{

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

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

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

public int AccessTokenExpirationMinutes { get; set; }

public int RefreshTokenExpirationDays { get; set; }

}

}

**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**

**LoginRequestDTOValidator.cs:**

using FluentValidation;

using LW4\_Task6\_MiA.DTO;

namespace LW4\_Task6\_MiA.Validators

{

public class LoginRequestDto

{

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

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

}

}

**RatingValidator.cs:**

using LW4\_Task2\_MiA.Models;

using FluentValidation;

using System;

namespace LW4\_Task2\_MiA.Validators

{

public class RatingValidator : AbstractValidator<Rating>

{

public RatingValidator()

{

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

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

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();

//рядкові ObjectId

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

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

}

}

}

**RegisterRequestDTOValidator.cs:**

using FluentValidation;

using LW4\_Task6\_MiA.DTO;

namespace LW4\_Task6\_MiA.Validators

{

public class RegisterRequestDtoValidator : AbstractValidator<RegisterRequestDto>

{

public RegisterRequestDtoValidator()

{

RuleFor(x => x.DisplayName)

.NotEmpty().WithMessage("Display Name не може бути порожнім.")

.Length(2, 50).WithMessage("Display Name має бути від 2 до 50 символів.");

RuleFor(x => x.Email)

.NotEmpty().WithMessage("Email не може бути порожнім.")

.EmailAddress().WithMessage("Введіть коректний email.");

RuleFor(x => x.Password)

.NotEmpty().WithMessage("Пароль не може бути порожнім.")

.MinimumLength(8).WithMessage("Пароль має бути не менше 8 символів.");

RuleFor(x => x.ConfirmPassword)

.Equal(x => x.Password).WithMessage("Паролі не співпадають.");

}

}

}

**TokenResponseDTOValidator.cs:**

using FluentValidation;

using LW4\_Task6\_MiA.DTO;

namespace LW4\_Task6\_MiA.Validators

{

public class TokenResponseDTOValidator : AbstractValidator<TokenResponseDto>

{

public TokenResponseDTOValidator()

{

RuleFor(x => x.AccessToken)

.NotEmpty().WithMessage("AccessToken не може бути порожнім.");

RuleFor(x => x.RefreshToken)

.NotEmpty().WithMessage("RefreshToken не може бути порожнім.");

}

}

}

**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 System.Text;

using System.Security.Claims;

using Microsoft.IdentityModel.Tokens;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using FluentValidation;

using FluentValidation.AspNetCore;

using LW4\_Task2\_MiA.Repositories;

using LW4\_Task4\_MiA.Mapping;

using LW4\_Task4\_MiA.Service;

using LW4\_Task4\_MiA.Settings;

using LW4\_Task6\_MiA.Service;

using LW4\_Task6\_MiA.Settings;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Options;

using Microsoft.OpenApi.Models;

using MongoDB.Driver;

var builder = WebApplication.CreateBuilder(args);

// 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);

});

// 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"));

// Сервіси

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

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

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

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

// Налаштування JWT

builder.Services.Configure<JwtSettings>(builder.Configuration.GetSection("Jwt"));

builder.Services.AddSingleton<ITokenService, TokenService>();

// Додавання Аутентифікації

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(options =>

{

var jwtSettings = builder.Configuration.GetSection("Jwt").Get<JwtSettings>();

if (jwtSettings == null) throw new InvalidOperationException("JwtSettings not configured.");

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = jwtSettings.Issuer,

ValidAudience = jwtSettings.Audience,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings.Secret)),

RoleClaimType = ClaimTypes.Role, // Вказуємо, що роль знаходиться в ClaimTypes.Role

ClockSkew = TimeSpan.Zero // Забираємо 5-хвилинний допуск за замовчуванням

};

});

// Додавання Авторизації

builder.Services.AddAuthorization();

//AutoMapper

builder.Services.AddAutoMapper(cfg =>

{

cfg.AddProfile<MappingProfile>();

});

builder.Services.AddControllers().AddJsonOptions(o => { }).ConfigureApiBehaviorOptions(o => { });

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

{

options.SuppressImplicitRequiredAttributeForNonNullableReferenceTypes = true;

});

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(options =>

{

options.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

In = ParameterLocation.Header,

Description = "Будь ласка, введіть 'Bearer ', а потім ваш токен.\n\nПриклад: 'Bearer 12345abcdef'",

Name = "Authorization",

Type = SecuritySchemeType.ApiKey,

Scheme = "Bearer"

});

options.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

Array.Empty<string>()

}

});

});

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();

// Додаємо аутентифікацію та авторизацію

app.UseAuthentication();

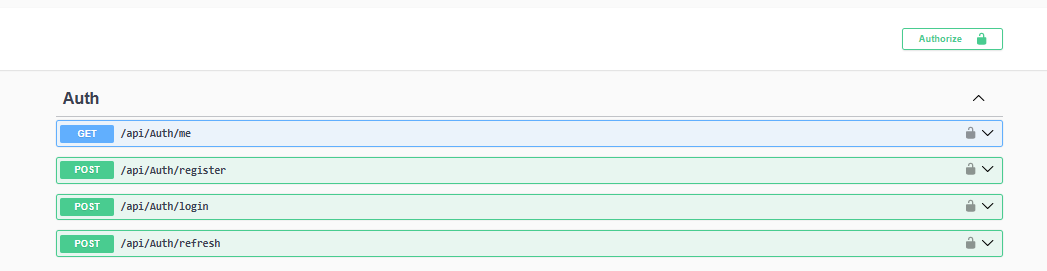
app.UseAuthorization();

app.MapControllers();

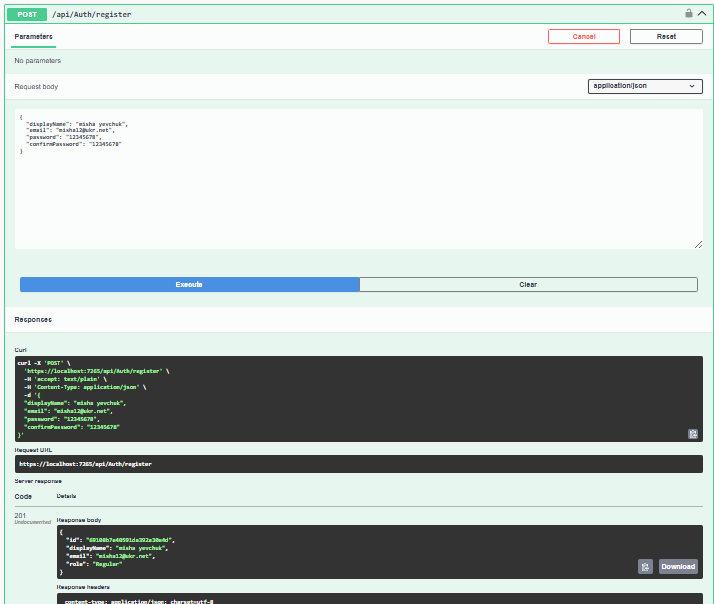
app.Run();

**Результат програми:**

В Swagger додались запити на авторизацію

****

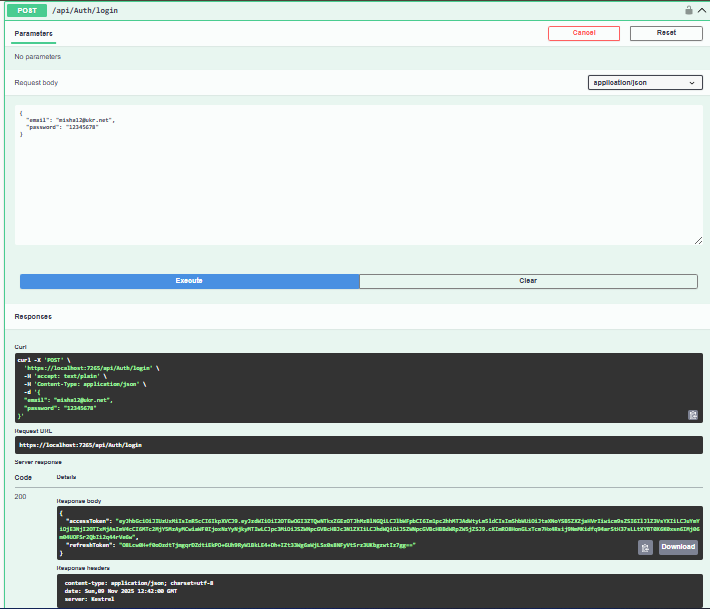
Виконаємо запит на реєстрацію

****

Користувач успішно зареєструвався, та йому видало accessToken, refreshToken

За допомогою accessToken в користувача є доступ до окремих запитів. В адміністратора є доступ на видалення та оновлення, так само в модератора.

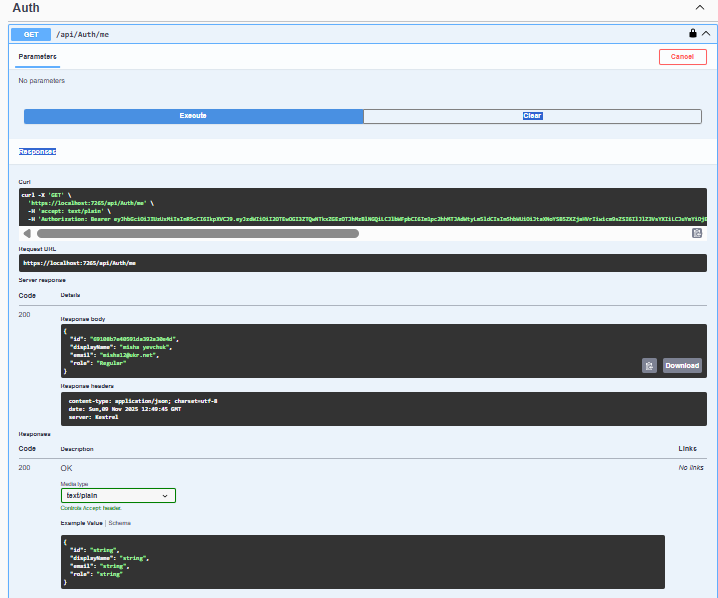
Виконаємо запит на логін.



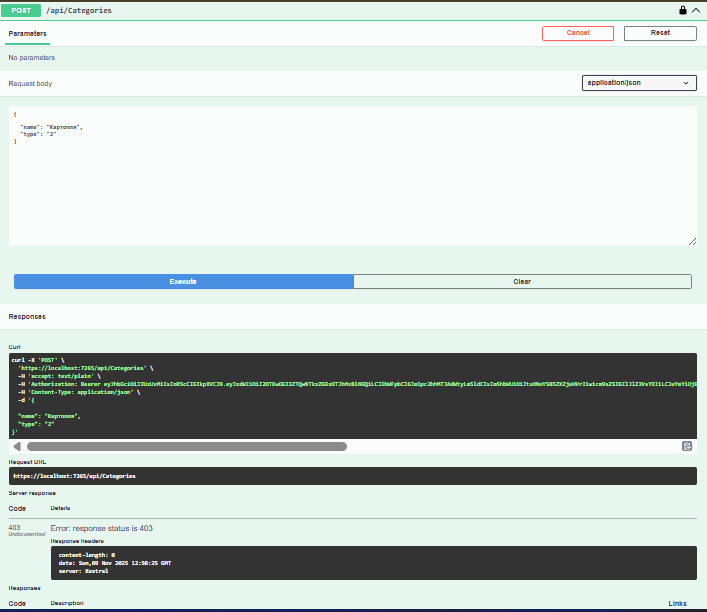
Візьмем цей токен та вставимо в Authorize.



Перевіримо чи дійсно увійшов в систему

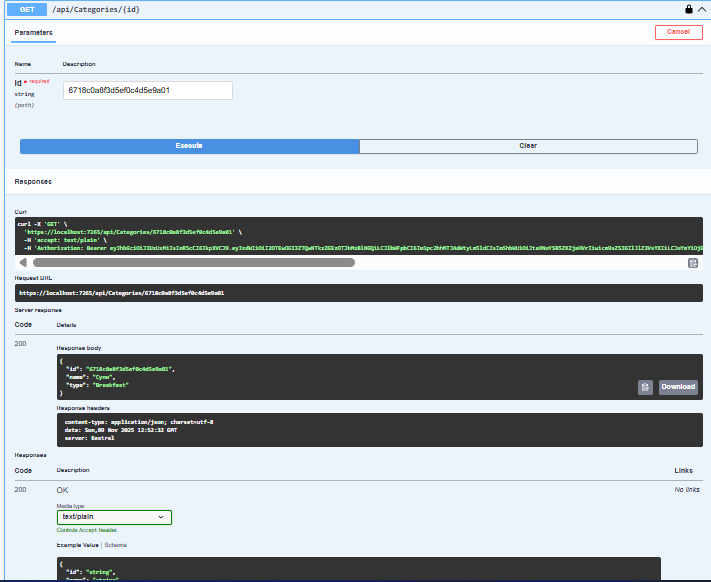


Користувач увійшов в систему.

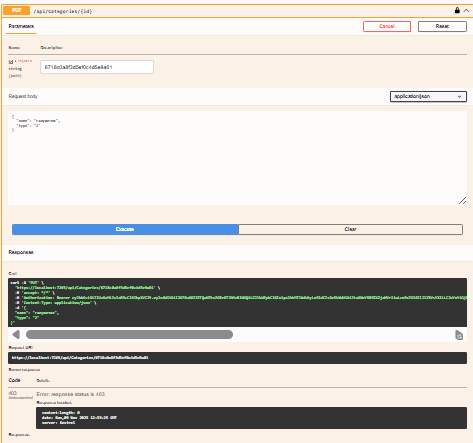


Звичайний користувач не може публікувати ані категорії, ані рецепти.

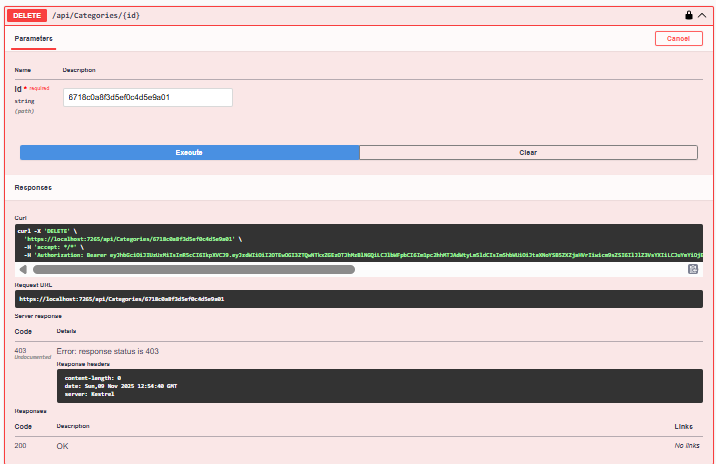
Запит на отримання по категорії айді.



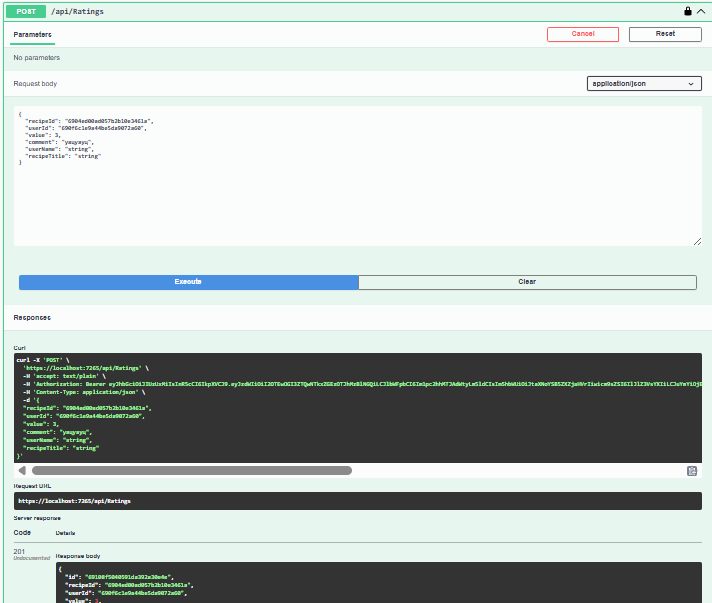
Доступу на оновлення категорії немає.



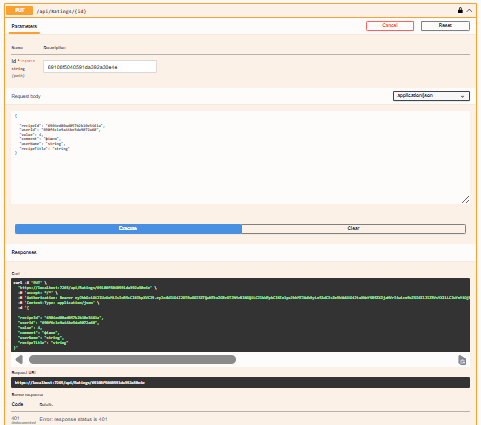
Запит на видалення, користувач не може.



Публікувати рейтинг користувач може, а от змінювати та видаляти, ні.



Також на оновлення рейтингу, користувач немає доступу.



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