**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**

**ЗАВДАННЯ 1. Ознайомитися з кроками створення ASP.NET Core Minimal API**

**ЗАВДАННЯ 2. Ознайомитися з кроками створення ASP.NET Core Web API**

**ЗАВДАННЯ 3. Спроєктувати архітектури Controller – Service – Repository**

**ЗАВДАННЯ 4. Ознайомитися з кроками роботи MongoDB |**

**ЗАВДАННЯ 5. Ознайомитися з матеріалом Dependency Injection та AutoMapper**

**ЗАВДАННЯ 6. Оформити звіт та продемонструвати викладачу.**

**Хід роботи**

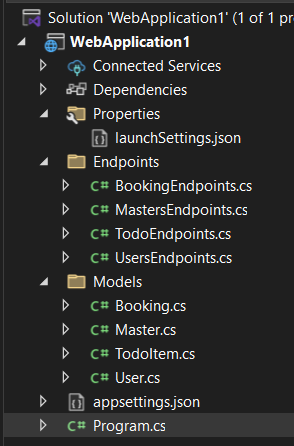
**Завдання 1. Ознайомитися з кроками створення ASP.NET Core Minimal API**

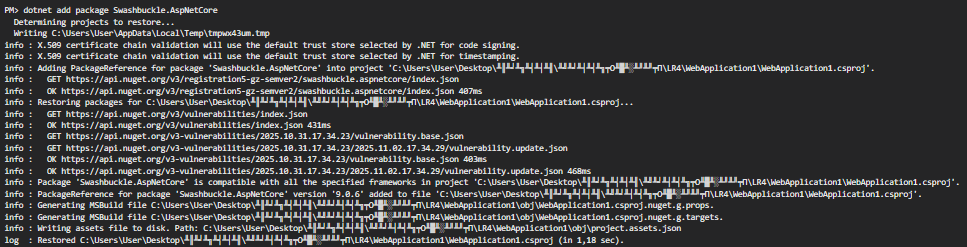
Створено один тестовий ендпоінт Todos (Endpoints/TodoEndpoints.cs) та три ендпоінти на тему «Платформа для бронювання майстрів (ремонт, клінінг, монтаж)».

Три ендпоінти по темі:

* Endpoints/UsersEndpoints.cs. Ендпоінт з користувачами, з полями Id, Name, Age, Email. При створені нового користувача йде валідація чи пусті поля Name та Email. Перевірка чи email не коротше 11 символів, адже “@gmail.com” займає 10 символів і принаймні один символ має бути в назві. Також додано валідацію із регулярним виразом (^[^@\s]+@[^@\s]+\.[^@\s]+$) для правильного введення електронної пошти. Також всі ці перевірки додані до PUT, адже тоді також змінюються всі поля і потрібні всі перевірки.
* Endpoints/MastersEndpoints.cs. Ендпоінт з майстрами, з полями Id, Name, Category, Ranking. При створені нового майстра йде валідація чи пусті поля Name та Category. Також всі ці перевірки додані до PUT, адже тоді також змінюються всі поля і потрібні всі перевірки.
* Endpoints/BookingEndpoints.cs. Ендпоінт з бронюваннями послуг, з полями Id, MasterId, Date, ServiceDetails, Status. При створені нового користувача йде валідація чи пусті поля Date, ServiceDate та Status. Є перевірка чи існує MasterId у списку masters, адже якщо не існує майстра з таким id, то ми не можемо додати таке бронювання. І є валідація довжини дати (не менше 10 символів) + перевірка через регулярний вираз (^(0[1-9]|[12][0-9]|3[01])\.(0[1-9]|1[0-2])\.\d{4}$) для введення як dd.mm.yyyy. Також всі ці перевірки додані до PUT, адже тоді також змінюються всі поля і потрібні всі перевірки.

***Структура***

******

***Підключення Swagger***

***Program.cs***

using WebApplication1.Endpoints;

using WebApplication1.Models;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var masters = new List<Master>();

var bookings = new List<Booking>();

builder.Services.AddSingleton(masters);

builder.Services.AddSingleton(bookings);

var app = builder.Build();

app.MapTodoEndpoints();

app.MapUsersEndpoints();

app.MapBookingEndpoints();

app.MapMastersEndpoints();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.Run();

***Models/Booking.cs***

namespace WebApplication1.Models

{

public class Booking

{

public int Id { get; set; }

public int MasterId { get; set; }

public string Date { get; set; }

public string ServiceDetails { get; set; }

public string Status { get; set; }

}

}

***Models/Users.cs***

namespace WebApplication1.Models

{

public class User

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public string Email { get; set; }

}

}

***Models/Master.cs***

namespace WebApplication1.Models

{

public class Master

{

public int Id { get; set; }

public string Name { get; set; }

public string Category { get; set; }

public double Ranking { get; set; }

}

}

***Models/TodoItem.cs***

namespace WebApplication1.Models

{

public class TodoItem

{

public int Id { get; set; } // Унікальний ідентифікатор

public string Name { get; set; } // Назва завдання

public bool IsComplete { get; set; } // Статус виконання

}

}

***Endpoints/BookingEndpoints.cs***

using System.Text.RegularExpressions;

using WebApplication1.Models;

namespace WebApplication1.Endpoints

{

public static class BookingEndpoints

{

public static void MapBookingEndpoints(this WebApplication app)

{

var bookings = new List<Booking>()

{

new Booking { Id = 1, MasterId = 2, Date = "11.12.2025", ServiceDetails = "socket changing", Status = "pending" },

new Booking { Id = 2, MasterId = 1, Date = "25.11.2025", ServiceDetails = "kitchen faucet repair", Status = "confirmed" }

};

app.MapGet("/bookings", () =>

{

return Results.Ok(bookings);

});

app.MapGet("/bookings/{id:int}", (int id) =>

{

var booking = bookings.FirstOrDefault(t => t.Id == id);

return booking is not null ? Results.Ok(booking) : Results.NotFound();

});

app.MapPost("/bookings", (Booking newBooking, List<Master> masters, List<Booking> bookings) =>

{

if (!masters.Any(m => m.Id == newBooking.MasterId))

return Results.BadRequest($"No master found with Id = {newBooking.MasterId}");

if (string.IsNullOrWhiteSpace(newBooking.ServiceDetails))

return Results.BadRequest("Поле 'Servicedetails' не може бути порожнім.");

if (string.IsNullOrWhiteSpace(newBooking.Date))

return Results.BadRequest("Поле 'Date' не може бути порожнім.");

if (newBooking.Date.Length < 10)

return Results.BadRequest("Дата має містити щонайменше 3 символи.");

var datePattern = @"^(0[1-9]|[12][0-9]|3[01])**\.**(0[1-9]|1[0-2])**\.**\d{4}$";

if (!Regex.IsMatch(newBooking.Date, datePattern))

return Results.BadRequest("Невірний формат дати. Використовуйте формат dd.MM.yyyy (наприклад, 05.11.2025).");

if (string.IsNullOrWhiteSpace(newBooking.Status))

return Results.BadRequest("Поле 'Status' не може бути порожнім.");

newBooking.Id = bookings.Max(t => t.Id) + 1;

bookings.Add(newBooking);

return Results.Created($"/bookings/{newBooking.Id}", newBooking);

});

app.MapPut("/bookings/{id:int}", (int id, Booking updatedBooking, List<Master> masters) =>

{

var booking = bookings.FirstOrDefault(t => t.Id == id);

if (booking is null) return Results.NotFound();

if (!masters.Any(m => m.Id == updatedBooking.MasterId))

return Results.BadRequest($"No master found with Id = {updatedBooking.MasterId}");

if (string.IsNullOrWhiteSpace(updatedBooking.ServiceDetails))

return Results.BadRequest("Поле 'Servicedetails' не може бути порожнім.");

if (string.IsNullOrWhiteSpace(updatedBooking.Date))

return Results.BadRequest("Поле 'Date' не може бути порожнім.");

if (updatedBooking.Date.Length < 10)

return Results.BadRequest("Дата має містити щонайменше 3 символи.");

var datePattern = @"^(0[1-9]|[12][0-9]|3[01])**\.**(0[1-9]|1[0-2])**\.**\d{4}$";

if (!Regex.IsMatch(updatedBooking.Date, datePattern))

return Results.BadRequest("Невірний формат дати. Використовуйте формат dd.MM.yyyy (наприклад, 05.11.2025).");

if (string.IsNullOrWhiteSpace(updatedBooking.Status))

return Results.BadRequest("Поле 'Status' не може бути порожнім.");

booking.MasterId = updatedBooking.MasterId;

booking.Date = updatedBooking.Date;

booking.ServiceDetails = updatedBooking.ServiceDetails;

booking.Status = updatedBooking.Status;

return Results.Ok(booking);

});

app.MapDelete("/bookings/{id:int}", (int id) =>

{

var booking = bookings.FirstOrDefault(t => t.Id == id);

if (booking is null) return Results.NotFound();

bookings.Remove(booking);

return Results.NoContent();

});

}

}

}

***Endpoints/UsersEndpoints.cs***

using System.Text.RegularExpressions;

using WebApplication1.Models;

namespace WebApplication1.Endpoints

{

public static class UsersEndpoints

{

public static void MapUsersEndpoints(this WebApplication app)

{

var users = new List<User>()

{

new User { Id = 1, Name = "Clara", Age = 21, Email = "clara123@gmail.com" },

new User { Id = 2, Name = "Thomas", Age = 34, Email = "thomasredish@gmail.com" }

};

app.MapGet("/users", () =>

{

return Results.Ok(users);

});

app.MapGet("/users/{id:int}", (int id) =>

{

var user = users.FirstOrDefault(t => t.Id == id);

return user is not null ? Results.Ok(user) : Results.NotFound();

});

app.MapPost("/users", (User newUser) =>

{

if (string.IsNullOrWhiteSpace(newUser.Name))

return Results.BadRequest("Поле 'Name' не може бути порожнім.");

if (string.IsNullOrWhiteSpace(newUser.Email))

return Results.BadRequest("Поле 'Email' не може бути порожнім.");

if (newUser.Email.Length < 11)

return Results.BadRequest("Email має містити щонайменше 3 символи.");

var emailPattern = @"^[^@\s]+@[^@\s]+**\.**[^@\s]+$";

if (!Regex.IsMatch(newUser.Email, emailPattern, RegexOptions.IgnoreCase))

return Results.BadRequest("Невірний формат Email.");

newUser.Id = users.Max(t => t.Id) + 1;

users.Add(newUser);

return Results.Created($"/todos/{newUser.Id}", newUser);

});

app.MapPut("/users/{id:int}", (int id, User updatedUser) =>

{

var user = users.FirstOrDefault(t => t.Id == id);

if (user is null) return Results.NotFound();

if (string.IsNullOrWhiteSpace(updatedUser.Name))

return Results.BadRequest("Поле 'Name' не може бути порожнім.");

if (string.IsNullOrWhiteSpace(updatedUser.Email))

return Results.BadRequest("Поле 'Email' не може бути порожнім.");

if (updatedUser.Email.Length < 11)

return Results.BadRequest("Email має містити щонайменше 3 символи.");

user.Name = updatedUser.Name;

user.Age = updatedUser.Age;

user.Email = updatedUser.Email;

return Results.Ok(user);

});

app.MapDelete("/users/{id:int}", (int id) =>

{

var user = users.FirstOrDefault(t => t.Id == id);

if (user is null) return Results.NotFound();

users.Remove(user);

return Results.NoContent();

});

}

}

}

***Endpoints/TodoEndpoints.cs***

using WebApplication1.Models;

namespace WebApplication1.Endpoints

{

public static class TodoEndpoints

{

public static void MapTodoEndpoints(this WebApplication app)

{

var todoItems = new List<TodoItem>()

{

new TodoItem { Id = 1, Name = "Learn .NET 8 Minimal APIs", IsComplete = true },

new TodoItem { Id = 2, Name = "Build a Simple To-Do API", IsComplete = false }

};

app.MapGet("/todos", () =>

{

return Results.Ok(todoItems);

});

app.MapGet("/todos/{id:int}", (int id) =>

{

var todo = todoItems.FirstOrDefault(t => t.Id == id);

return todo is not null ? Results.Ok(todo) : Results.NotFound();

});

app.MapPost("/todos", (TodoItem newTodo) =>

{

if (!ValidateTodo(newTodo, out var error))

return Results.BadRequest(error);

newTodo.Id = todoItems.Count + 1;

todoItems.Add(newTodo);

return Results.Created($"/todos/{newTodo.Id}", newTodo);

});

app.MapPut("/todos/{id:int}", (int id, TodoItem updatedTodo) =>

{

var todo = todoItems.FirstOrDefault(t => t.Id == id);

if (todo is null) return Results.NotFound();

todo.Name = updatedTodo.Name;

todo.IsComplete = updatedTodo.IsComplete;

return Results.Ok(todo);

});

app.MapDelete("/todos/{id:int}", (int id) =>

{

var todo = todoItems.FirstOrDefault(t => t.Id == id);

if (todo is null) return Results.NotFound();

todoItems.Remove(todo);

return Results.NoContent();

});

bool ValidateTodo(TodoItem todo, out string? error)

{

if (string.IsNullOrWhiteSpace(todo.Name))

{

error = "Назва не може бути порожньою.";

return false;

}

if (todo.Name.Length < 3)

{

error = "Назва має бути не коротша за 3 символи.";

return false;

}

error = null;

return true;

}

}

}

}

***Endpoints/MastersEndpoints.cs***

using WebApplication1.Models;

namespace WebApplication1.Endpoints

{

public static class MastersEndpoints

{

public static void MapMastersEndpoints(this WebApplication app)

{

var masters = new List<Master>()

{

new Master { Id = 1, Name = "Theodor", Category = "repair", Ranking = 4.5 },

new Master { Id = 2, Name = "Samanta", Category = "repair", Ranking = 4.8 }

};

app.MapGet("/masters", () =>

{

return Results.Ok(masters);

});

app.MapGet("/masters/{id:int}", (int id) =>

{

var master = masters.FirstOrDefault(t => t.Id == id);

return master is not null ? Results.Ok(master) : Results.NotFound();

});

app.MapPost("/masters", (Master newMaster) =>

{

if (string.IsNullOrWhiteSpace(newMaster.Name))

return Results.BadRequest("Поле 'Name' не може бути порожнім.");

if (string.IsNullOrWhiteSpace(newMaster.Category))

return Results.BadRequest("Поле 'Category' не може бути порожнім.");

newMaster.Id = masters.Max(t => t.Id) + 1;

masters.Add(newMaster);

return Results.Created($"/masters/{newMaster.Id}", newMaster);

});

app.MapPut("/masters/{id:int}", (int id, Master updatedMaster) =>

{

var master = masters.FirstOrDefault(t => t.Id == id);

if (master is null) return Results.NotFound();

if (string.IsNullOrWhiteSpace(updatedMaster.Name))

return Results.BadRequest("Поле 'Name' не може бути порожнім.");

if (string.IsNullOrWhiteSpace(updatedMaster.Category))

return Results.BadRequest("Поле 'Category' не може бути порожнім.");

master.Name = updatedMaster.Name;

master.Category = updatedMaster.Category;

master.Ranking = updatedMaster.Ranking;

return Results.Ok(master);

});

app.MapDelete("/masters/{id:int}", (int id) =>

{

var master = masters.FirstOrDefault(t => t.Id == id);

if (master is null) return Results.NotFound();

masters.Remove(master);

return Results.NoContent();

});

}

}

}

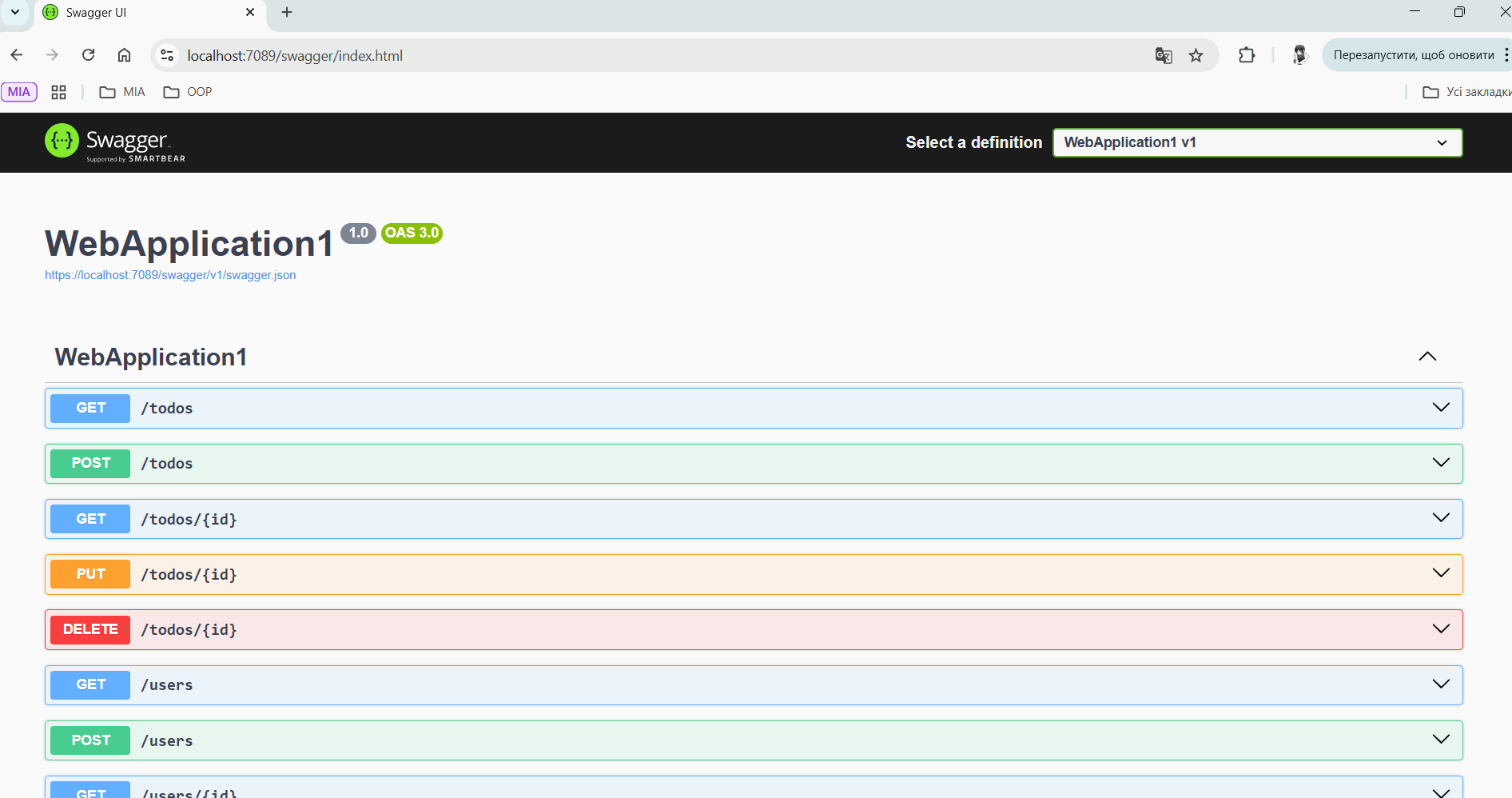
******

Рисунок 1 – Swagger UI з всіма ендпоінтами

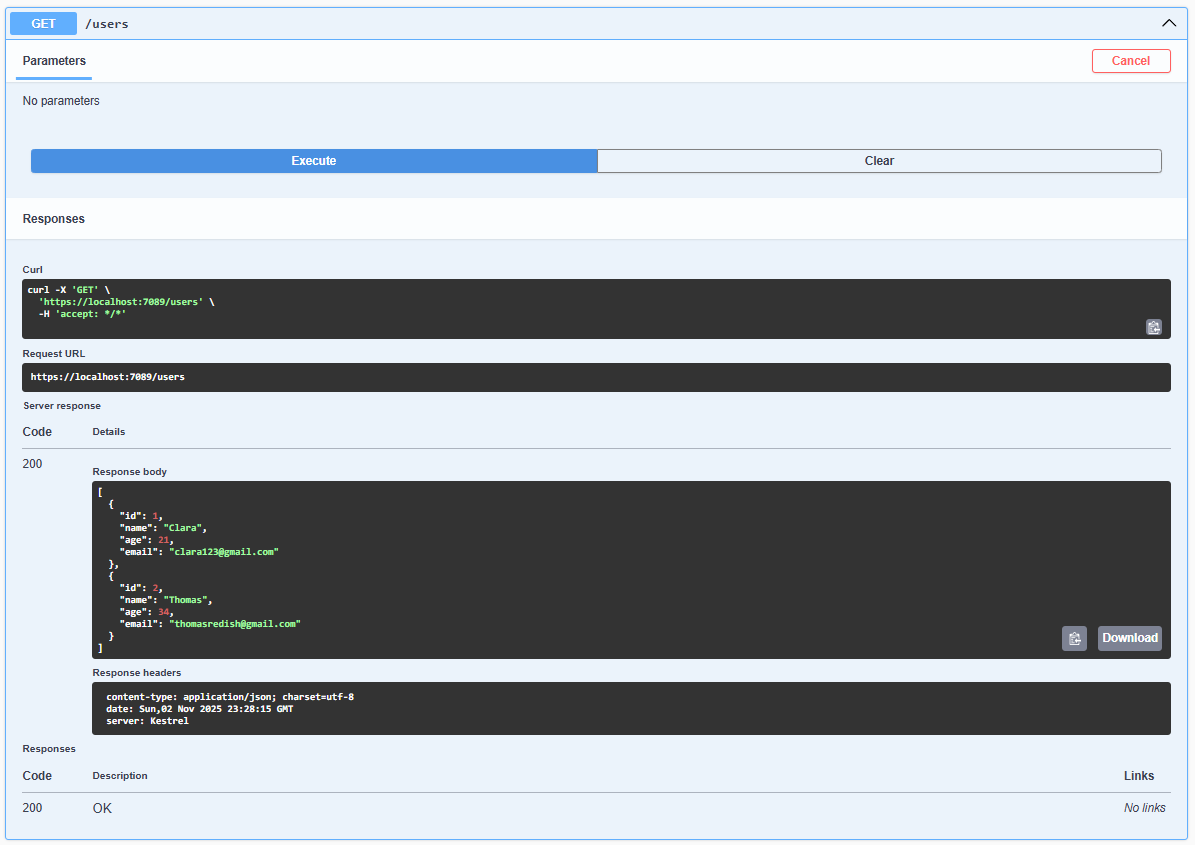


Рисунок 2 – GET для users для отримання списку користувачів

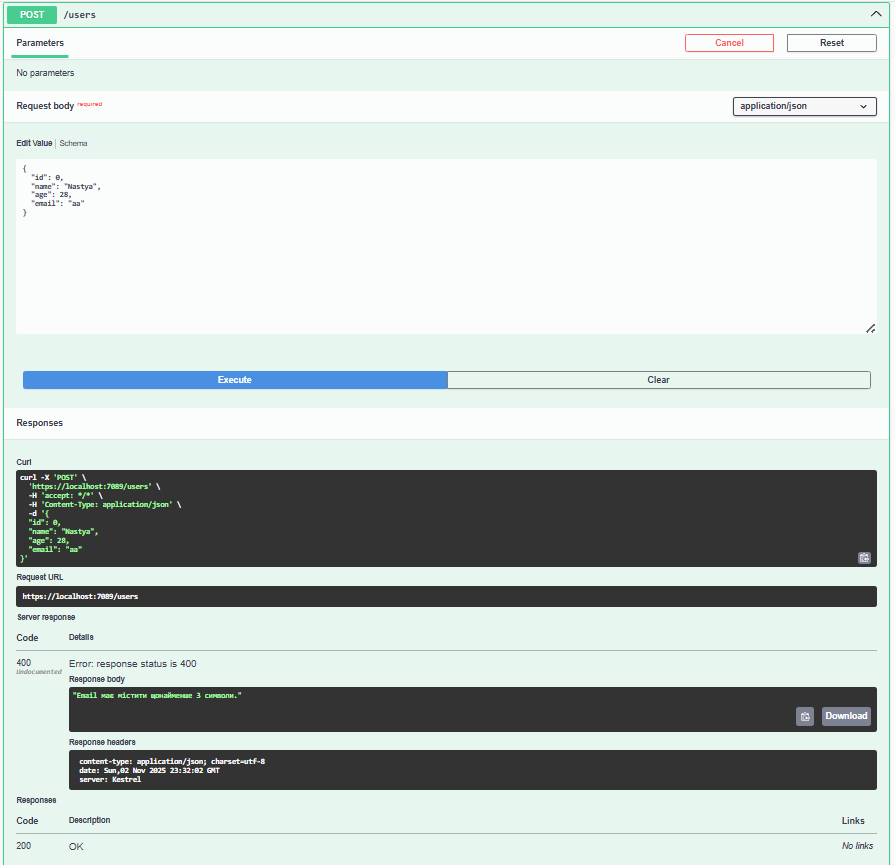


Рисунок 3 – POST для users. Показано валідацію даних: введено менше 11 символів для ел. пошти

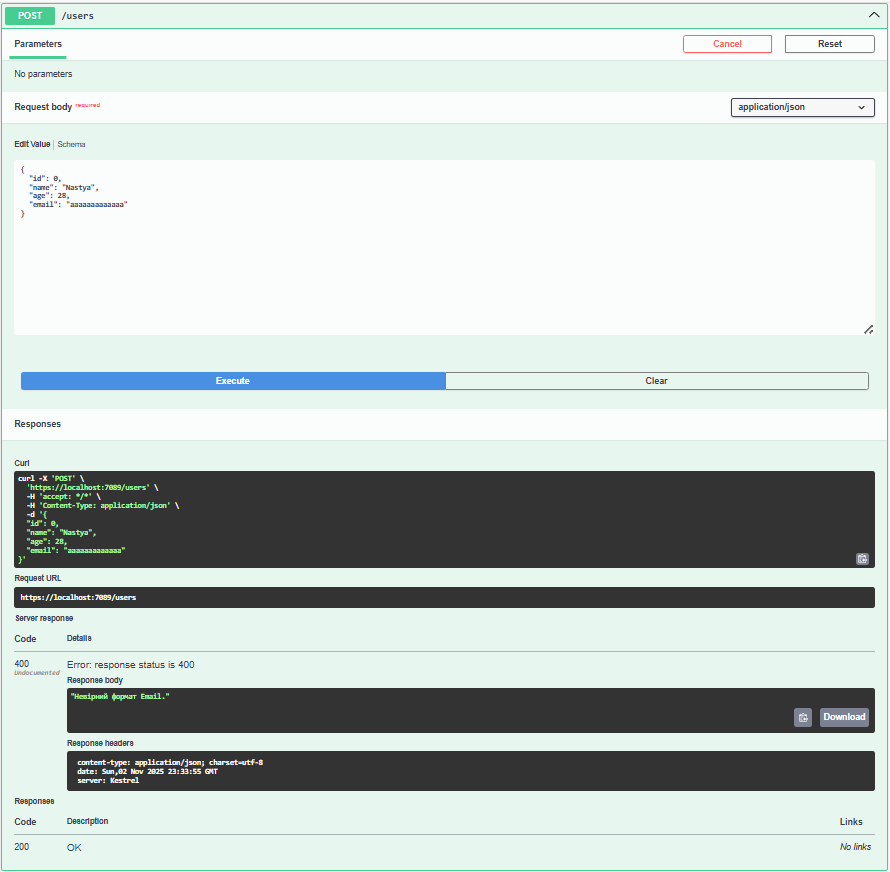


Рисунок 4 – POST для users. Валідація через регулярний вираз, введено не менше 11 символів, проте неправильний формат

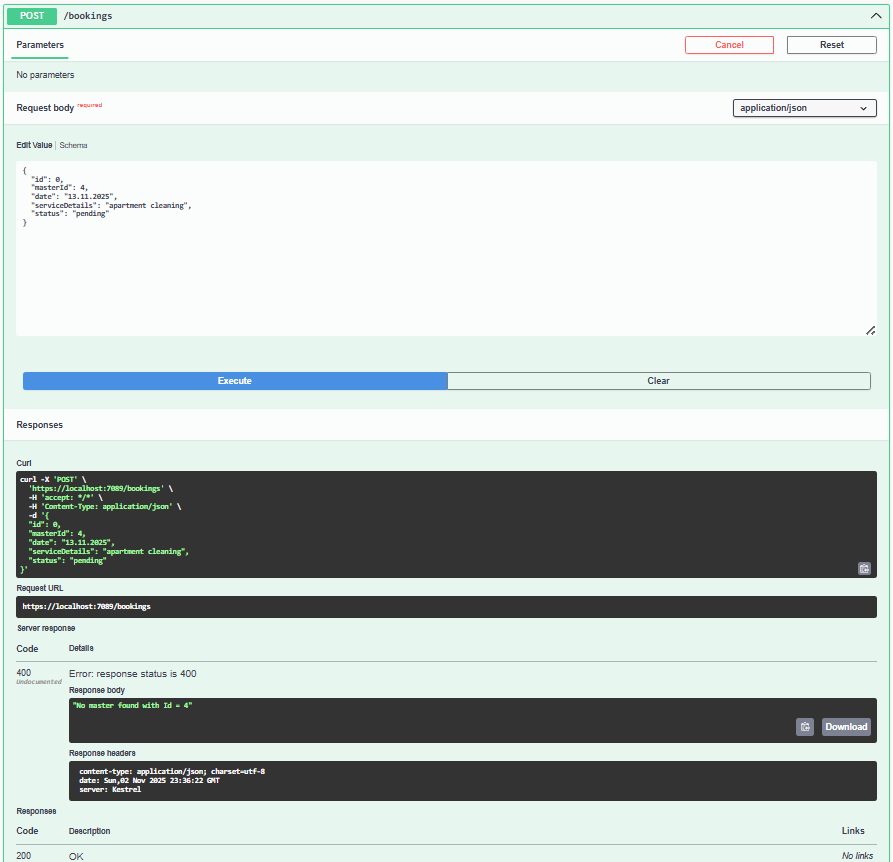


Рисунок 5 – POST для bookings. Приклад введення MasterId, якого не існує в списку masters

**Завдання 2. Ознайомитися з кроками створення ASP.NET Core Web API**

Створено один тестовий контролер Todos (Controllers/TodosController.cs) та три контролери на тему «Платформа для бронювання майстрів (ремонт, клінінг, монтаж)».

Три контролери по темі:

* Controllers/UserController.cs. Контролер з користувачами, з полями Id, Name, Age, Email. Додано валідація (Vallidators/UserValidator.cs) чи пусті поля Name та Email. Перевірка чи email не коротше 11 символів, адже “@gmail.com” займає 10 символів і принаймні один символ має бути в назві. Також додано валідацію завдяки методу .EnailAdress() для правильного введення електронної пошти.
* Contollers/MasterController.cs. Контролер з майстрами, з полями Id, Name, Category, Ranking. Додана валідація (Vallidators/MasterValidator.cs) чи пусті поля Name, Ranking та Category та чи існує введене значення в поле Category у Enum Category завдяки методу .IsInEnum().
* Contollers/BookingController.cs. Контрлер з бронюваннями послуг, з полями Id, MasterId, Date, ServiceDetails, Status. Додана валідація (Vallidators/BookingValidator.cs) чи пусті поля Date, MasterId, ServiceDate та Status та чи існує введене значення у поле Status у Enum Status завдяки методу .IsInEnum. І є валідація довжини дати (не менше 10 символів) + перевірка через регулярний вираз (^(0[1-9]|[12][0-9]|3[01])\.(0[1-9]|1[0-2])\.\d{4}$) для введення як dd.mm.yyyy завдяки методу .Matches(regex).

***Структура***



***Program.cs***

using FluentValidation;

using FluentValidation.AspNetCore;

using MyWebApi.Models;

using MyWebApi.Validators;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// Реєструємо контролери і FluentValidation

builder.Services.AddControllers().AddFluentValidation();

// Реєструємо всі валідатори автоматично

builder.Services.AddValidatorsFromAssemblyContaining<TodoItemValidator>();

builder.Services.AddValidatorsFromAssemblyContaining<UserValidator>();

builder.Services.AddValidatorsFromAssemblyContaining<BookingValidator>();

builder.Services.AddValidatorsFromAssemblyContaining<MasterValidator>();

builder.Services.AddControllers()

.AddJsonOptions(options =>

{

options.JsonSerializerOptions.Converters.Add(new System.Text.Json.Serialization.JsonStringEnumConverter());

});

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

***Models/User.cs***

namespace MyWebApi.Models

{

public class User

{

public int Id { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public string Email { get; set; }

}

}

***Models/Todos.cs***

namespace MyWebApi.Models

{

public class TodoItem

{

public int Id { get; set; }

public string Name { get; set; } = "";

public bool IsComplete { get; set; }

}

}

***Models/Master.cs***

namespace MyWebApi.Models

{

public enum Category

{

Plumbing,

Electrical,

Flooring,

Assembly

}

public class Master

{

public int Id { get; set; }

public string Name { get; set; }

public Category Category { get; set; }

public double Ranking { get; set; }

}

}

***Models/Booking.cs***

namespace MyWebApi.Models

{

public enum Status

{

Confirmed,

Pending,

Cancelled,

Competed

}

public class Booking

{

public int Id { get; set; }

public int MasterId { get; set; }

public string Date { get; set; }

public string ServiceDetails { get; set; }

public Status Status { get; set; }

}

}

***Controllers/MasterController.cs***

using Microsoft.AspNetCore.Mvc;

using MyWebApi.Models;

namespace MyWebApi.Controllers

{

[ApiController]

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

public class MasterController : ControllerBase

{

private static List<Master> masters = new()

{

new Master { Id = 1, Name = "Jackson", Category = Category.Electrical, Ranking = 6.7 },

new Master { Id = 2, Name = "Mickey", Category = Category.Flooring, Ranking = 8.9 }

};

[HttpGet]

public ActionResult<IEnumerable<Master>> GetAll()

{

return Ok(masters);

}

[HttpGet("{id}")]

public ActionResult<Master> GetById(int id)

{

var master = masters.FirstOrDefault(t => t.Id == id);

if (master == null)

return NotFound($"Майстра з ID={id} не знайдено.");

return Ok(master);

}

[HttpPost]

public IActionResult AddMaster([FromBody] Master newMaster)

{

newMaster.Id = masters.Count + 1;

if (!Enum.IsDefined(typeof(Category), newMaster.Category))

return BadRequest($"Invalid Difficulty value: {newMaster.Category}");

masters.Add(newMaster);

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

}

[HttpPut("{id}")]

public IActionResult Update(int id, Master updatedMaster)

{

var master = masters.FirstOrDefault(t => t.Id == id);

if (master == null)

return NotFound($"Майстра з ID={id} не знайдено.");

master.Name = updatedMaster.Name;

master.Category = updatedMaster.Category;

master.Ranking = updatedMaster.Ranking;

return NoContent();

}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

var master = masters.FirstOrDefault(t => t.Id == id);

if (master == null)

return NotFound($"Майстра з ID={id} не знайдено.");

masters.Remove(master);

return NoContent();

}

}

}

***Controllers/UserController.cs***

using Microsoft.AspNetCore.Mvc;

using MyWebApi.Models;

namespace MyWebApi.Controllers

{

[ApiController]

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

public class UserController : ControllerBase

{

private static List<User> users = new()

{

new User { Id = 1, Name = "Mary", Age = 32, Email = "mary33@gmail.com" },

new User { Id = 2, Name = "Dorothy", Age = 54, Email = "dorothy22@gmail.com" }

};

[HttpGet]

public ActionResult<IEnumerable<User>> GetAll()

{

return Ok(users);

}

[HttpGet("{id}")]

public ActionResult<User> GetById(int id)

{

var todo = users.FirstOrDefault(t => t.Id == id);

if (todo == null)

return NotFound($"Користувача з ID={id} не знайдено.");

return Ok(todo);

}

[HttpPost]

public IActionResult AddUser([FromBody] User newUser)

{

newUser.Id = users.Count + 1;

users.Add(newUser);

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

}

[HttpPut("{id}")]

public IActionResult Update(int id, User updatedUser)

{

var user = users.FirstOrDefault(t => t.Id == id);

if (user == null)

return NotFound($"Користувач з ID={id} не знайдено.");

user.Name = updatedUser.Name;

user.Age = updatedUser.Age;

user.Email = updatedUser.Email;

return NoContent();

}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

var user = users.FirstOrDefault(t => t.Id == id);

if (user == null)

return NotFound($"Користувач з ID={id} не знайдено.");

users.Remove(user);

return NoContent();

}

}

}

***Controllers/BookingController.cs***

using Microsoft.AspNetCore.Mvc;

using MyWebApi.Models;

namespace MyWebApi.Controllers

{

[ApiController]

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

public class BookingController : ControllerBase

{

private static List<Booking> bookings = new()

{

new Booking { Id = 1, MasterId = 2, Date = "22.11.2025", ServiceDetails = "Deck repairs", Status = Status.Confirmed },

new Booking { Id = 2, MasterId = 1, Date = "10.11.2025", ServiceDetails = "Socket repairs", Status = Status.Pending }

};

[HttpGet]

public ActionResult<IEnumerable<Booking>> GetAll()

{

return Ok(bookings);

}

[HttpGet("{id}")]

public ActionResult<Booking> GetById(int id)

{

var todo = bookings.FirstOrDefault(t => t.Id == id);

if (todo == null)

return NotFound($"Завдання з ID={id} не знайдено.");

return Ok(todo);

}

[HttpPost]

public IActionResult AddBooking([FromBody] Booking newBooking)

{

newBooking.Id = bookings.Count + 1;

if (!Enum.IsDefined(typeof(Status), newBooking.Status))

return BadRequest($"Invalid Difficulty value: {newBooking.Status}");

bookings.Add(newBooking);

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

}

[HttpPut("{id}")]

public IActionResult Update(int id, Booking updatedBooking)

{

var booking = bookings.FirstOrDefault(t => t.Id == id);

if (booking == null)

return NotFound($"Завдання з ID={id} не знайдено.");

//if (!masters.Any(m => m.Id == updatedBooking.MasterId))

// return Results.BadRequest($"No master found with Id = {updatedBooking.MasterId}");

booking.MasterId = updatedBooking.MasterId;

booking.Date = updatedBooking.Date;

booking.Status = updatedBooking.Status;

booking.Status = updatedBooking.Status;

return NoContent();

}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

var booking = bookings.FirstOrDefault(t => t.Id == id);

if (booking == null)

return NotFound($"Завдання з ID={id} не знайдено.");

bookings.Remove(booking);

return NoContent();

}

}

}

***Controllers/TodosController.cs***

using Microsoft.AspNetCore.Mvc;

using MyWebApi.Models;

namespace MyWebApi.Controllers

{

[ApiController]

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

public class TodoController : ControllerBase

{

private static List<TodoItem> todos = new()

{

new TodoItem { Id = 1, Name = "Вивчити ASP.NET Core Web API", IsComplete = true },

new TodoItem { Id = 2, Name = "Створити власний ToDo контролер", IsComplete = false }

};

[HttpGet]

public ActionResult<IEnumerable<TodoItem>> GetAll()

{

return Ok(todos);

}

[HttpGet("{id}")]

public ActionResult<TodoItem> GetById(int id)

{

var todo = todos.FirstOrDefault(t => t.Id == id);

if (todo == null)

return NotFound($"Завдання з ID={id} не знайдено.");

return Ok(todo);

}

[HttpPost]

public ActionResult<TodoItem> Create(TodoItem newTodo)

{

newTodo.Id = todos.Count + 1;

todos.Add(newTodo);

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

}

[HttpPut("{id}")]

public IActionResult Update(int id, TodoItem updatedTodo)

{

var todo = todos.FirstOrDefault(t => t.Id == id);

if (todo == null)

return NotFound($"Завдання з ID={id} не знайдено.");

todo.Name = updatedTodo.Name;

todo.IsComplete = updatedTodo.IsComplete;

return NoContent();

}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

var todo = todos.FirstOrDefault(t => t.Id == id);

if (todo == null)

return NotFound($"Завдання з ID={id} не знайдено.");

todos.Remove(todo);

return NoContent();

}

}

}

***Validators/UserValidator.cs***

using FluentValidation;

using MyWebApi.Models;

namespace MyWebApi.Validators

{

public class UserValidator : AbstractValidator<User>

{

public UserValidator()

{

RuleFor(x => x.Name)

.NotEmpty().WithMessage("Поле 'Name' є обов'язковим.")

.MinimumLength(3).WithMessage("Назва має містити щонайменше 3 символи.")

.MaximumLength(100).WithMessage("Назва не може перевищувати 100 символів.");

RuleFor(x => x.Age)

.NotEmpty().WithMessage("Поле 'Age' є обов'язковим.");

RuleFor(x => x.Email)

.NotEmpty().WithMessage("Поле 'Email' є обов'язковим.")

.MinimumLength(11).WithMessage("Ел. пошта має містити щонайменше 11 символів.")

.EmailAddress().WithMessage("Ел. пошта не того формату");

}

}

}

***Validators/MasterValidator.cs***

using FluentValidation;

using MyWebApi.Models;

namespace MyWebApi.Validators

{

public class MasterValidator : AbstractValidator<Master>

{

public MasterValidator()

{

RuleFor(x => x.Name)

.NotEmpty().WithMessage("Поле 'Name' є обов'язковим.")

.MinimumLength(3).WithMessage("Назва має містити щонайменше 3 символи.")

.MaximumLength(100).WithMessage("Назва не може перевищувати 100 символів.");

RuleFor(x => x.Category)

.NotNull().WithMessage("Поле 'Category' має бути вказано.")

.IsInEnum();

RuleFor(x => x.Ranking)

.NotNull().WithMessage("Поле 'ranking' має бути вказано.");

}

}

}

***Validators/BookingValidator.cs***

using FluentValidation;

using MyWebApi.Models;

namespace MyWebApi.Validators

{

public class BookingValidator : AbstractValidator<Booking>

{

public BookingValidator()

{

RuleFor(x => x.Date)

.NotEmpty().WithMessage("Поле 'Name' є обов'язковим.")

.MinimumLength(10).WithMessage("Назва має містити щонайменше 10 символів.")

.Matches(@"^(0[1-9]|[12][0-9]|3[01])**\.**(0[1-9]|1[0-2])**\.**\d{4}$");

RuleFor(x => x.MasterId)

.NotEmpty().WithMessage("Поле 'MasterId' є обов'язковим.");

RuleFor(x => x.Status)

.NotEmpty().WithMessage("Поле 'Status' є обов'язковим.")

.IsInEnum();

RuleFor(x => x.ServiceDetails)

.NotNull().WithMessage("Поле 'ServiceDetails' має бути вказано.");

}

}

}

***Validators/TodoItemValidator.cs***

using FluentValidation;

using MyWebApi.Models;

namespace MyWebApi.Validators

{

public class TodoItemValidator : AbstractValidator<TodoItem>

{

public TodoItemValidator()

{

RuleFor(x => x.Name)

.NotEmpty().WithMessage("Поле 'Name' є обов'язковим.")

.MinimumLength(3).WithMessage("Назва має містити щонайменше 3 символи.")

.MaximumLength(100).WithMessage("Назва не може перевищувати 100 символів.");

RuleFor(x => x.IsComplete)

.NotNull().WithMessage("Поле 'IsComplete' має бути вказано.");

}

}

}

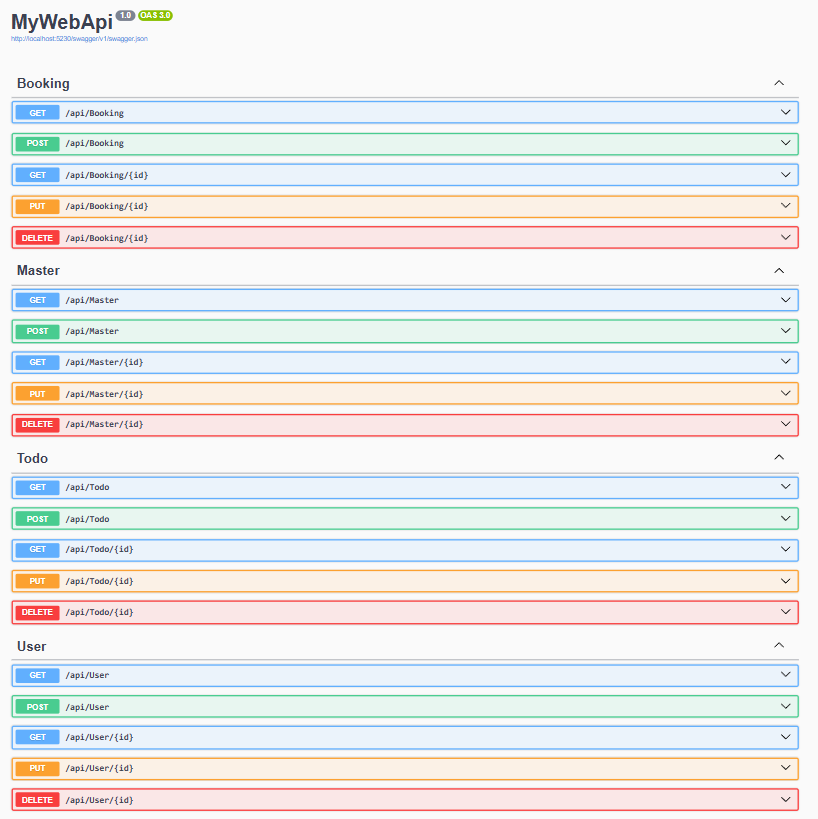


Рисунок 6 – Swagger UI з всіма контролерами



Рисунок 7 – метод POST для booking

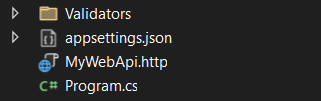


Рисунок 8 – метод POST для booking (валідація введення дати)

**Завдання 3.**

***Струкутра***





***Program.cs***

using FluentValidation;

using FluentValidation.AspNetCore;

using MyWebApi.Models;

using MyWebApi.Validators;

using MyWebApi.Repositories.Interfaces;

using MyWebApi.Repositories.Implementations;

using MyWebApi.Services.Interfaces;

using MyWebApi.Services.Implementations;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers()

.AddFluentValidation()

.AddJsonOptions(options =>

{

options.JsonSerializerOptions.Converters.Add(new System.Text.Json.Serialization.JsonStringEnumConverter());

});

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

builder.Services.AddValidatorsFromAssemblyContaining<UserValidator>();

builder.Services.AddValidatorsFromAssemblyContaining<MasterValidator>();

builder.Services.AddValidatorsFromAssemblyContaining<BookingValidator>();

builder.Services.AddValidatorsFromAssemblyContaining<TodoItemValidator>();

builder.Services.AddScoped<IMasterRepository, MasterRepository> ();

builder.Services.AddScoped<IBookingRepository, BookingRepository>();

builder.Services.AddScoped<ITodoItemRepository, TodoItemRepository>();

builder.Services.AddScoped<IUserRepository, UserRepository>();

builder.Services.AddScoped<IMasterService, MasterService>();

builder.Services.AddScoped<IBookingService, BookingService>();

builder.Services.AddScoped<ITodoItemService, TodoItemService>();

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

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

***Services/Interfaces/IBookingService.cs***

using MyWebApi.Models;

namespace MyWebApi.Services.Interfaces

{

public interface IBookingService

{

Task<IEnumerable<Booking>> GetAllAsync();

Task<Booking?> GetByIdAsync(int id);

Task AddAsync(Booking booking);

Task UpdateAsync(Booking booking);

Task DeleteAsync(int id);

}

}

***Services/Interfaces/IMasterService.cs***

using MyWebApi.Models;

namespace MyWebApi.Services.Interfaces

{

public interface IMasterService

{

Task<IEnumerable<Master>> GetAllAsync();

Task<Master?> GetByIdAsync(int id);

Task AddAsync(Master master);

Task UpdateAsync(Master master);

Task DeleteAsync(int id);

}

}

***Services/Interfaces/ITodoService.cs***

using MyWebApi.Models;

namespace MyWebApi.Services.Interfaces

{

public interface ITodoItemService

{

Task<IEnumerable<TodoItem>> GetAllAsync();

Task<TodoItem?> GetByIdAsync(int id);

Task AddAsync(TodoItem item);

Task UpdateAsync(TodoItem item);

Task DeleteAsync(int id);

}

}

***Services/Interfaces/IUserService.cs***

using MyWebApi.Models;

namespace MyWebApi.Services.Interfaces

{

public interface IUserService

{

Task<IEnumerable<User>> GetAllAsync();

Task<User?> GetByIdAsync(int id);

Task AddAsync(User user);

Task UpdateAsync(User user);

Task DeleteAsync(int id);

}

}

***Services/Implementation/BookingService.cs***

using MyWebApi.Models;

using MyWebApi.Repositories.Interfaces;

using MyWebApi.Services.Interfaces;

namespace MyWebApi.Services.Implementations

{

public class BookingService : IBookingService

{

private readonly IBookingRepository \_repository;

public BookingService(IBookingRepository repository) => \_repository = repository;

public Task<IEnumerable<Booking>> GetAllAsync() => \_repository.GetAllAsync();

public async Task<Booking?> GetByIdAsync(int id)

{

var booking = await \_repository.GetByIdAsync(id);

if (booking == null)

throw new KeyNotFoundException($"Booking with id {id} not found");

return booking;

}

public async Task AddAsync(Booking booking)

{

if (string.IsNullOrWhiteSpace(booking.Date))

throw new ArgumentException("Date is required");

if (string.IsNullOrWhiteSpace(booking.ServiceDetails))

throw new ArgumentException("ServiceDetails is required");

await \_repository.AddAsync(booking);

}

public Task UpdateAsync(Booking booking) => \_repository.UpdateAsync(booking);

public Task DeleteAsync(int id) => \_repository.DeleteAsync(id);

}

}

***Services/Implementation/MasterService.cs***

using MyWebApi.Models;

using MyWebApi.Repositories.Interfaces;

using MyWebApi.Services.Interfaces;

namespace MyWebApi.Services.Implementations

{

public class MasterService : IMasterService

{

private readonly IMasterRepository \_repository;

public MasterService(IMasterRepository repository) => \_repository = repository;

public Task<IEnumerable<Master>> GetAllAsync() => \_repository.GetAllAsync();

public async Task<Master?> GetByIdAsync(int id)

{

var master = await \_repository.GetByIdAsync(id);

if (master == null)

throw new KeyNotFoundException($"Master with id {id} not found");

return master;

}

public async Task AddAsync(Master master)

{

if (string.IsNullOrWhiteSpace(master.Name))

throw new ArgumentException("Name is required");

if (master.Ranking < 0 || master.Ranking > 5)

throw new ArgumentException("Ranking must be between 0 and 5");

await \_repository.AddAsync(master);

}

public Task UpdateAsync(Master master) => \_repository.UpdateAsync(master);

public Task DeleteAsync(int id) => \_repository.DeleteAsync(id);

}

}

***Services/Implementation/TodoService.cs***

using MyWebApi.Models;

using MyWebApi.Repositories.Interfaces;

using MyWebApi.Services.Interfaces;

namespace MyWebApi.Services.Implementations

{

public class TodoItemService : ITodoItemService

{

private readonly ITodoItemRepository \_repository;

public TodoItemService(ITodoItemRepository repository) => \_repository = repository;

public Task<IEnumerable<TodoItem>> GetAllAsync() => \_repository.GetAllAsync();

public async Task<TodoItem?> GetByIdAsync(int id)

{

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

if (item == null)

throw new KeyNotFoundException($"TodoItem with id {id} not found");

return item;

}

public async Task AddAsync(TodoItem item)

{

if (string.IsNullOrWhiteSpace(item.Name))

throw new ArgumentException("Name is required");

await \_repository.AddAsync(item);

}

public Task UpdateAsync(TodoItem item) => \_repository.UpdateAsync(item);

public Task DeleteAsync(int id) => \_repository.DeleteAsync(id);

}

}

***Services/Implementation/UserService.cs***

using MyWebApi.Models;

using MyWebApi.Repositories.Interfaces;

using MyWebApi.Services.Interfaces;

namespace MyWebApi.Services.Implementations

{

public class UserService : IUserService

{

private readonly IUserRepository \_repository;

public UserService(IUserRepository repository) => \_repository = repository;

public Task<IEnumerable<User>> GetAllAsync() => \_repository.GetAllAsync();

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

{

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

if (user == null)

throw new KeyNotFoundException($"User with id {id} not found");

return user;

}

public async Task AddAsync(User user)

{

if (string.IsNullOrWhiteSpace(user.Name))

throw new ArgumentException("Name is required");

if (user.Age < 18)

throw new ArgumentException("Age must be at least 18");

if (string.IsNullOrWhiteSpace(user.Email) || !user.Email.Contains("@"))

throw new ArgumentException("Valid Email is required");

await \_repository.AddAsync(user);

}

public Task UpdateAsync(User user) => \_repository.UpdateAsync(user);

public Task DeleteAsync(int id) => \_repository.DeleteAsync(id);

}

}

***Conrollers/BookingController.cs***

using Microsoft.AspNetCore.Mvc;

using MyWebApi.Models;

using MyWebApi.Services.Interfaces;

namespace MyWebApi.Controllers

{

[ApiController]

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

public class BookingsController : ControllerBase

{

private readonly IBookingService \_service;

public BookingsController(IBookingService service) => \_service = service;

[HttpGet]

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

=> Ok(await \_service.GetAllAsync());

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

public async Task<ActionResult<Booking>> Get(int id)

{

try

{

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

return Ok(booking);

}

catch (KeyNotFoundException ex)

{

return NotFound(ex.Message);

}

}

[HttpPost]

public async Task<ActionResult<Booking>> Create(Booking booking)

{

try

{

await \_service.AddAsync(booking);

return CreatedAtAction(nameof(Get), new { id = booking.Id }, booking);

}

catch (ArgumentException ex)

{

return BadRequest(ex.Message);

}

}

[HttpPut("{id:int}")]

public async Task<IActionResult> Update(int id, Booking booking)

{

if (id != booking.Id) return BadRequest("ID mismatch");

await \_service.UpdateAsync(booking);

return NoContent();

}

[HttpDelete("{id:int}")]

public async Task<IActionResult> Delete(int id)

{

await \_service.DeleteAsync(id);

return NoContent();

}

}

}

***Controllers/MasterController.cs***

using Microsoft.AspNetCore.Mvc;

using MyWebApi.Models;

using MyWebApi.Services.Interfaces;

namespace MyWebApi.Controllers

{

[ApiController]

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

public class MastersController : ControllerBase

{

private readonly IMasterService \_service;

public MastersController(IMasterService service) => \_service = service;

[HttpGet]

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

=> Ok(await \_service.GetAllAsync());

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

public async Task<ActionResult<Master>> Get(int id)

{

try

{

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

return Ok(master);

}

catch (KeyNotFoundException ex)

{

return NotFound(ex.Message);

}

}

[HttpPost]

public async Task<ActionResult<Master>> Create(Master master)

{

try

{

await \_service.AddAsync(master);

return CreatedAtAction(nameof(Get), new { id = master.Id }, master);

}

catch (ArgumentException ex)

{

return BadRequest(ex.Message);

}

}

[HttpPut("{id:int}")]

public async Task<IActionResult> Update(int id, Master master)

{

if (id != master.Id) return BadRequest("ID mismatch");

await \_service.UpdateAsync(master);

return NoContent();

}

[HttpDelete("{id:int}")]

public async Task<IActionResult> Delete(int id)

{

await \_service.DeleteAsync(id);

return NoContent();

}

}

}

***Controllers/TodosController.cs***

using Microsoft.AspNetCore.Mvc;

using MyWebApi.Models;

using MyWebApi.Services.Interfaces;

namespace MyWebApi.Controllers

{

[ApiController]

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

public class TodoItemsController : ControllerBase

{

private readonly ITodoItemService \_service;

public TodoItemsController(ITodoItemService service) => \_service = service;

[HttpGet]

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

=> Ok(await \_service.GetAllAsync());

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

public async Task<ActionResult<TodoItem>> Get(int id)

{

try

{

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

return Ok(item);

}

catch (KeyNotFoundException ex)

{

return NotFound(ex.Message);

}

}

[HttpPost]

public async Task<ActionResult<TodoItem>> Create(TodoItem item)

{

try

{

await \_service.AddAsync(item);

return CreatedAtAction(nameof(Get), new { id = item.Id }, item);

}

catch (ArgumentException ex)

{

return BadRequest(ex.Message);

}

}

[HttpPut("{id:int}")]

public async Task<IActionResult> Update(int id, TodoItem item)

{

if (id != item.Id) return BadRequest("ID mismatch");

await \_service.UpdateAsync(item);

return NoContent();

}

[HttpDelete("{id:int}")]

public async Task<IActionResult> Delete(int id)

{

await \_service.DeleteAsync(id);

return NoContent();

}

}

}

***Controllers/UserController.cs***

using Microsoft.AspNetCore.Mvc;

using MyWebApi.Models;

using MyWebApi.Services.Interfaces;

namespace MyWebApi.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()

=> Ok(await \_service.GetAllAsync());

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

public async Task<ActionResult<User>> Get(int id)

{

try

{

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

return Ok(user);

}

catch (KeyNotFoundException ex)

{

return NotFound(ex.Message);

}

}

[HttpPost]

public async Task<ActionResult<User>> Create(User user)

{

try

{

await \_service.AddAsync(user);

return CreatedAtAction(nameof(Get), new { id = user.Id }, user);

}

catch (ArgumentException ex)

{

return BadRequest(ex.Message);

}

}

[HttpPut("{id:int}")]

public async Task<IActionResult> Update(int id, User user)

{

if (id != user.Id) return BadRequest("ID mismatch");

await \_service.UpdateAsync(user);

return NoContent();

}

[HttpDelete("{id:int}")]

public async Task<IActionResult> Delete(int id)

{

await \_service.DeleteAsync(id);

return NoContent();

}

}

}

***Repositories/Interfaces/IBookingRepository.cs***

using MyWebApi.Models;

namespace MyWebApi.Repositories.Interfaces

{

public interface IBookingRepository

{

Task<IEnumerable<Booking>> GetAllAsync();

Task<Booking?> GetByIdAsync(int id);

Task AddAsync(Booking booking);

Task UpdateAsync(Booking booking);

Task DeleteAsync(int id);

}

}

***Repositories/Interfaces/IMasterRepository.cs***

using MyWebApi.Models;

namespace MyWebApi.Repositories.Interfaces

{

public interface IMasterRepository

{

Task<IEnumerable<Master>> GetAllAsync();

Task<Master?> GetByIdAsync(int id);

Task AddAsync(Master master);

Task UpdateAsync(Master master);

Task DeleteAsync(int id);

}

}

***Repositories/Interfaces/ITodoRepository.cs***

using MyWebApi.Models;

namespace MyWebApi.Repositories.Interfaces

{

public interface ITodoItemRepository

{

Task<IEnumerable<TodoItem>> GetAllAsync();

Task<TodoItem?> GetByIdAsync(int id);

Task AddAsync(TodoItem item);

Task UpdateAsync(TodoItem item);

Task DeleteAsync(int id);

}

}

***Repositories/Interfaces/IUserRepository.cs***

using MyWebApi.Models;

namespace MyWebApi.Repositories.Interfaces

{

public interface IUserRepository

{

Task<IEnumerable<User>> GetAllAsync();

Task<User?> GetByIdAsync(int id);

Task AddAsync(User user);

Task UpdateAsync(User user);

Task DeleteAsync(int id);

}

}

***Repositories/Implementation/BookingRepository.cs***

using MyWebApi.Models;

using MyWebApi.Repositories.Interfaces;

namespace MyWebApi.Repositories.Implementations

{

public class BookingRepository : IBookingRepository

{

private readonly List<Booking> \_bookings = new()

{

new() { Id = 1, MasterId = 1, Date = "2025-11-15", ServiceDetails = "Ремонт крану", Status = Status.Pending },

new() { Id = 2, MasterId = 2, Date = "2025-11-16", ServiceDetails = "Заміна проводки", Status = Status.Confirmed }

};

private int \_nextId = 3;

public Task<IEnumerable<Booking>> GetAllAsync() => Task.FromResult(\_bookings.AsEnumerable());

public Task<Booking?> GetByIdAsync(int id) =>

Task.FromResult(\_bookings.FirstOrDefault(b => b.Id == id));

public Task AddAsync(Booking booking)

{

booking.Id = \_nextId++;

\_bookings.Add(booking);

return Task.CompletedTask;

}

public Task UpdateAsync(Booking booking)

{

var existing = \_bookings.FirstOrDefault(b => b.Id == booking.Id);

if (existing != null)

{

existing.MasterId = booking.MasterId;

existing.Date = booking.Date;

existing.ServiceDetails = booking.ServiceDetails;

existing.Status = booking.Status;

}

return Task.CompletedTask;

}

public Task DeleteAsync(int id)

{

var booking = \_bookings.FirstOrDefault(b => b.Id == id);

if (booking != null) \_bookings.Remove(booking);

return Task.CompletedTask;

}

}

}

***Repositories/Implementation/MasterRepository.cs***

using MyWebApi.Models;

using MyWebApi.Repositories.Interfaces;

namespace MyWebApi.Repositories.Implementations

{

public class MasterRepository : IMasterRepository

{

private readonly List<Master> \_masters = new()

{

new() { Id = 1, Name = "Іван Петренко", Category = Category.Plumbing, Ranking = 4.8 },

new() { Id = 2, Name = "Олег Сидоренко", Category = Category.Electrical, Ranking = 4.9 },

new() { Id = 3, Name = "Микола Коваль", Category = Category.Assembly, Ranking = 4.5 }

};

private int \_nextId = 4;

public Task<IEnumerable<Master>> GetAllAsync() => Task.FromResult(\_masters.AsEnumerable());

public Task<Master?> GetByIdAsync(int id) =>

Task.FromResult(\_masters.FirstOrDefault(m => m.Id == id));

public Task AddAsync(Master master)

{

master.Id = \_nextId++;

\_masters.Add(master);

return Task.CompletedTask;

}

public Task UpdateAsync(Master master)

{

var existing = \_masters.FirstOrDefault(m => m.Id == master.Id);

if (existing != null)

{

existing.Name = master.Name;

existing.Category = master.Category;

existing.Ranking = master.Ranking;

}

return Task.CompletedTask;

}

public Task DeleteAsync(int id)

{

var master = \_masters.FirstOrDefault(m => m.Id == id);

if (master != null) \_masters.Remove(master);

return Task.CompletedTask;

}

}

}

***Repositories/Implementation/TodoRepository.cs***

using MyWebApi.Models;

using MyWebApi.Repositories.Interfaces;

namespace MyWebApi.Repositories.Implementations

{

public class TodoItemRepository : ITodoItemRepository

{

private readonly List<TodoItem> \_items = new()

{

new() { Id = 1, Name = "Купити продукти", IsComplete = false },

new() { Id = 2, Name = "Зателефонувати клієнту", IsComplete = true }

};

private int \_nextId = 3;

public Task<IEnumerable<TodoItem>> GetAllAsync() => Task.FromResult(\_items.AsEnumerable());

public Task<TodoItem?> GetByIdAsync(int id) =>

Task.FromResult(\_items.FirstOrDefault(i => i.Id == id));

public Task AddAsync(TodoItem item)

{

item.Id = \_nextId++;

\_items.Add(item);

return Task.CompletedTask;

}

public Task UpdateAsync(TodoItem item)

{

var existing = \_items.FirstOrDefault(i => i.Id == item.Id);

if (existing != null)

{

existing.Name = item.Name;

existing.IsComplete = item.IsComplete;

}

return Task.CompletedTask;

}

public Task DeleteAsync(int id)

{

var item = \_items.FirstOrDefault(i => i.Id == id);

if (item != null) \_items.Remove(item);

return Task.CompletedTask;

}

}

}

***Repositories/Implementation/UserRepository.cs***

using MyWebApi.Models;

using MyWebApi.Repositories.Interfaces;

namespace MyWebApi.Repositories.Implementations

{

public class UserRepository : IUserRepository

{

private readonly List<User> \_users = new()

{

new() { Id = 1, Name = "Олена Ковальчук", Age = 28, Email = "olena@example.com" },

new() { Id = 2, Name = "Сергій Іванов", Age = 35, Email = "sergiy@example.com" }

};

private int \_nextId = 3;

public Task<IEnumerable<User>> GetAllAsync() => Task.FromResult(\_users.AsEnumerable());

public Task<User?> GetByIdAsync(int id) =>

Task.FromResult(\_users.FirstOrDefault(u => u.Id == id));

public Task AddAsync(User user)

{

user.Id = \_nextId++;

\_users.Add(user);

return Task.CompletedTask;

}

public Task UpdateAsync(User user)

{

var existing = \_users.FirstOrDefault(u => u.Id == user.Id);

if (existing != null)

{

existing.Name = user.Name;

existing.Age = user.Age;

existing.Email = user.Email;

}

return Task.CompletedTask;

}

public Task DeleteAsync(int id)

{

var user = \_users.FirstOrDefault(u => u.Id == id);

if (user != null) \_users.Remove(user);

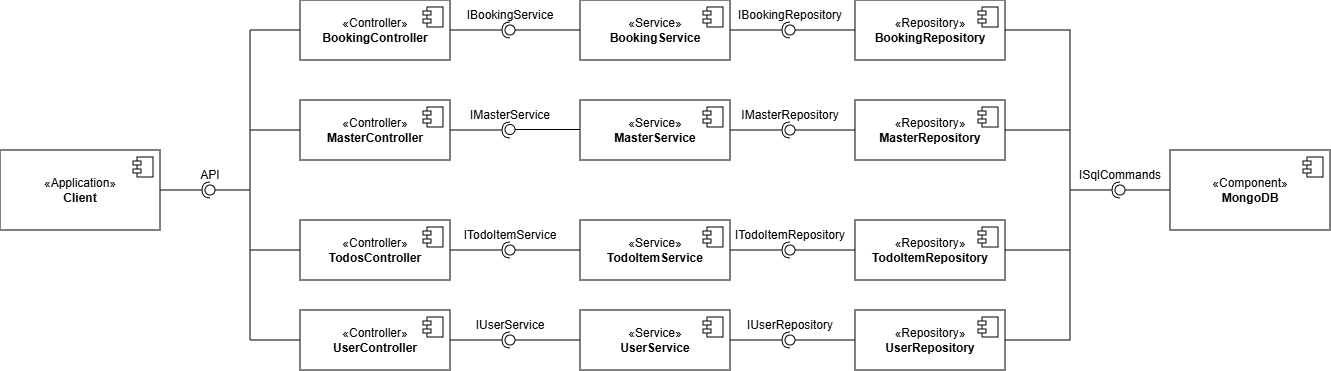
return Task.CompletedTask;

}

}

}

COMPONENT DIAGRAM



ОПИС КОЖНОГО РІВНЯ

***Controller Layer:***

Контролери виконують роль інтерфейсу між клієнтом і системою, обробляючи HTTP-запити та повертаючи відпо віді, забезпечуючи взаємодію з API. Вони реалізують ендпоінти для CRUD-операцій над сутностями, такими як бронювання, майстри, завдання та користувачі, інтегруючи сервіси для бізнес-логіки. Прикладами методів є GetAll() для отримання списку об'єктів, Create() для додавання нового запису з валідацією та обробкою помилок, а також Update() і Delete() для модифікації та видалення даних з перевіркою ID.

***Service Layer:***

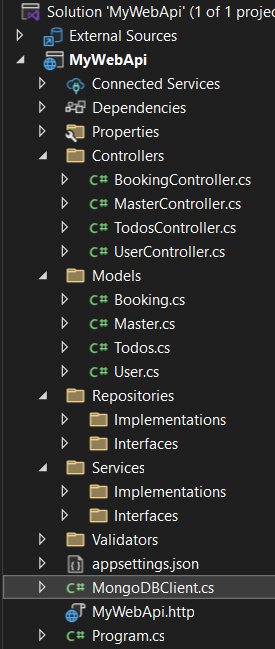
Сервіси відповідають за бізнес-логіку системи, забезпечуючи проміжний шар між контролерами та репозиторіями для застосування правил, валідації та обробки даних. Вони реалізують інтерфейси з методами для операцій над моделями, додаючи перевірки на помилки, такі як відсутність обов'язкових полів чи невалідні значення, перед взаємодією з даними. Прикладами обов'язків є GetByIdAsync() з викиданням винятку при відсутності запису, AddAsync() з валідацією полів (наприклад, перевірка віку користувача чи рейтингу майстра) та UpdateAsync() для оновлення сутностей без прямого доступу до сховища.

***Repository Layer:***

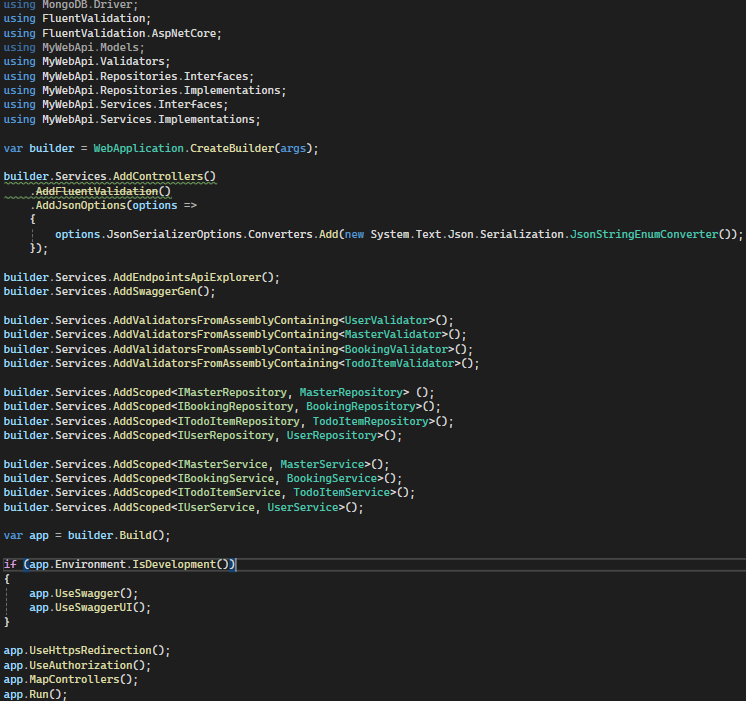
Репозиторії слугують шаром доступу до даних, абстрагуючи роботу зі сховищем (тут - в пам'яті) та забезпечуючи незалежність від конкретної бази даних. Вони реалізують інтерфейси для базових CRUD-операцій, керуючи колекціями об'єктів у списку з автоматичним присвоєнням ID. Прикладами методів є GetAllAsync() для повернення всіх записів, AddAsync() для додавання нового об'єкта з інкрементом ID, UpdateAsync() для модифікації існуючого запису та DeleteAsync() для видалення за ID без додаткової логіки.

**Завдання 4.**

***Структура програми***

******

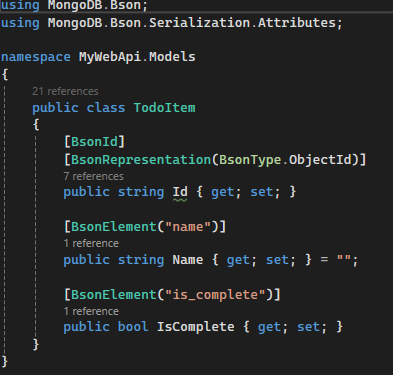
***Program.cs***

******

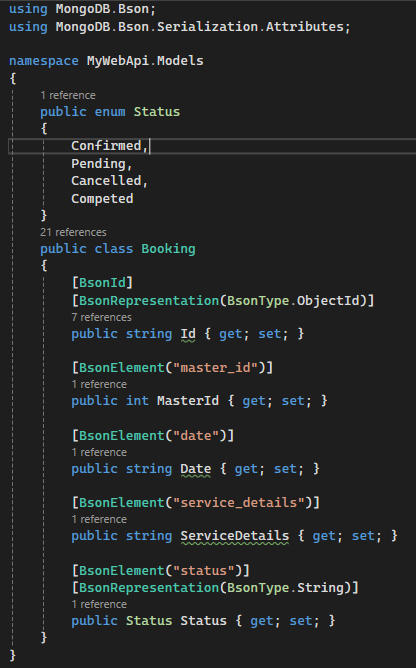
***MongoDBClient.cs***

******

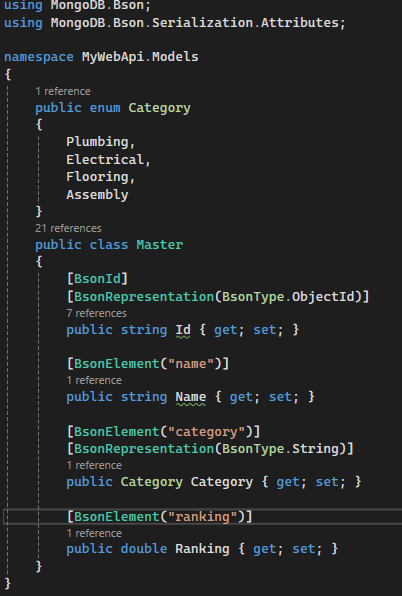
***Models/Todos.cs***

******

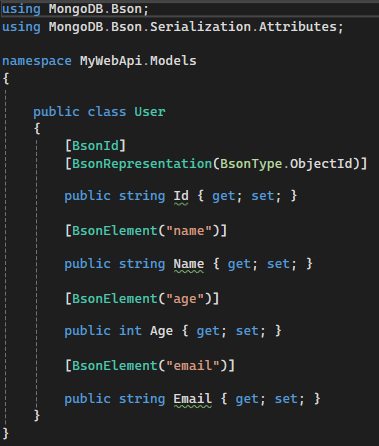
***Models/Booking.cs***

******

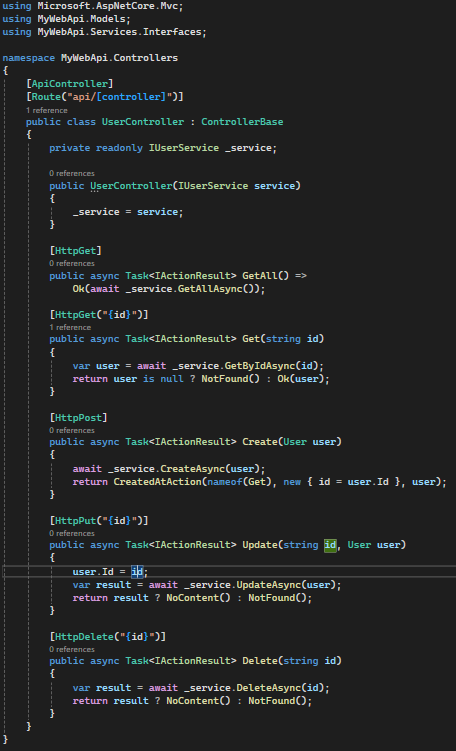
***Models/Master.cs***

******

***Models/User.cs***

******

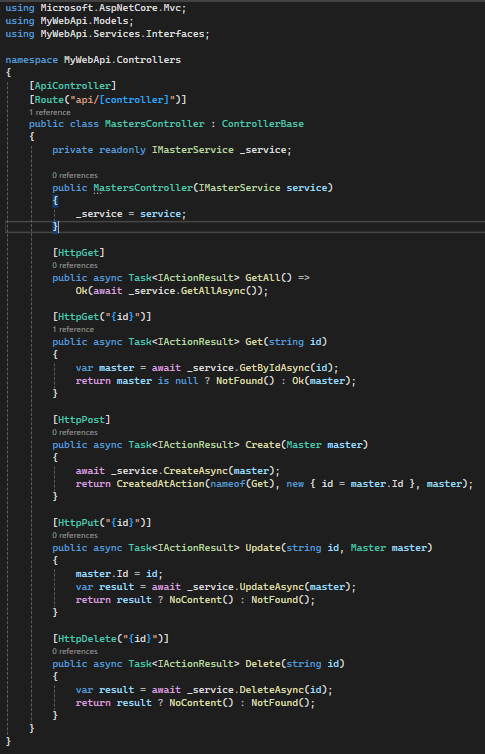
***Controllers/UserController.cs***

******

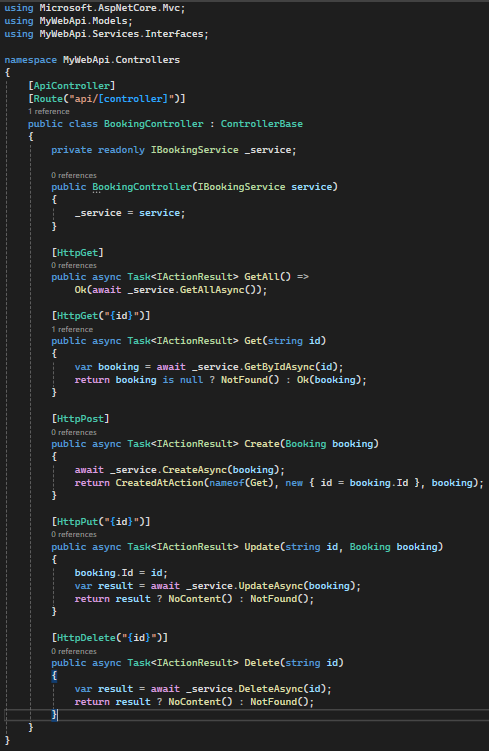
***Controllers/TodosController.cs***

******

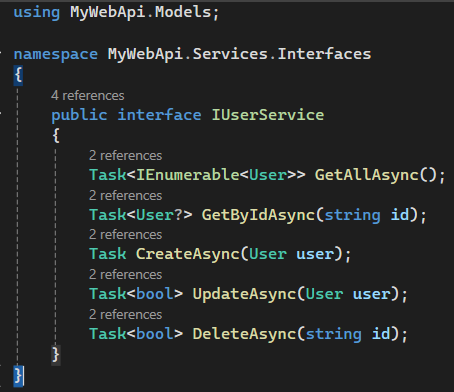
***Controllers/MasterController.cs***

******

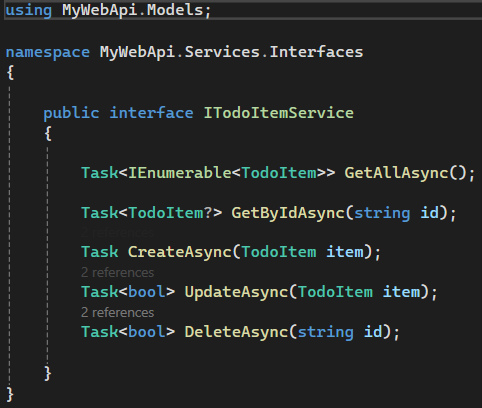
***Controllers/BookingController.cs***

******

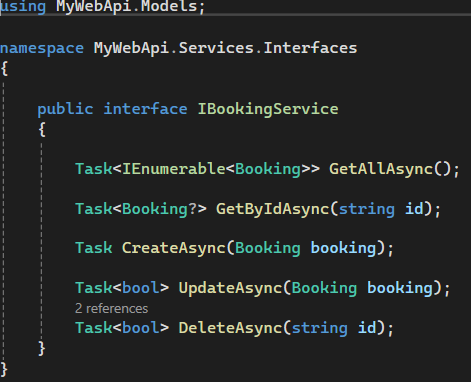
***Service/ Interfaces/IUserService.cs***

******

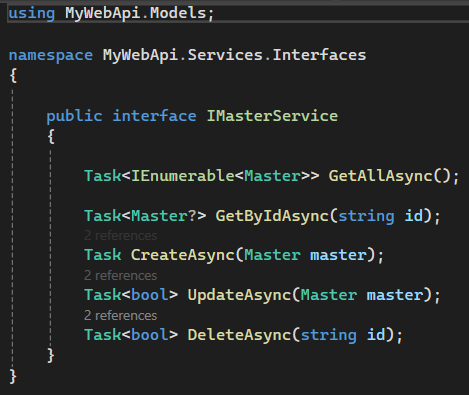
***Service/ Interfaces/ITodoItemService.cs***

******

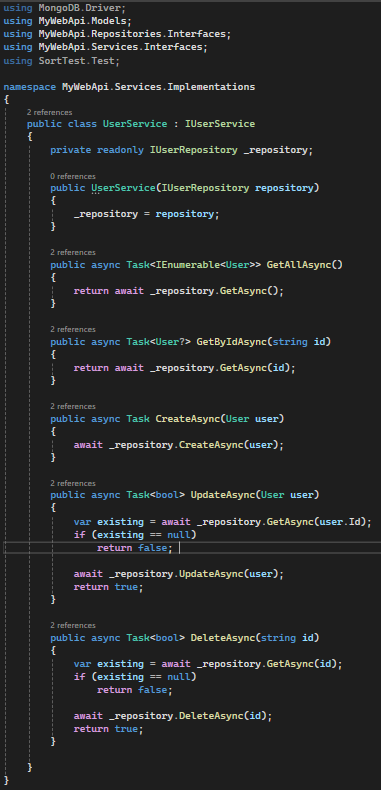
***Service/ Interfaces/IBookingService.cs***

******

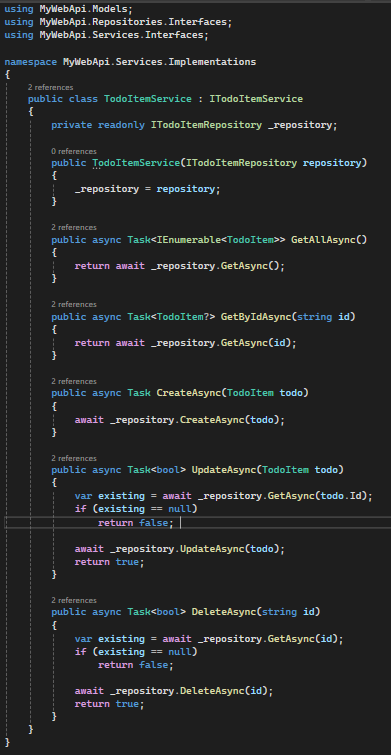
***Service/ Interfaces/IMasterService.cs***

******

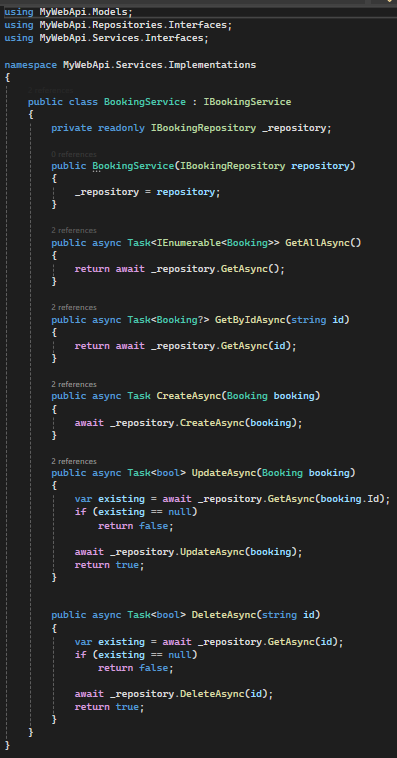
***Service/Implementation/UserService.cs***

******

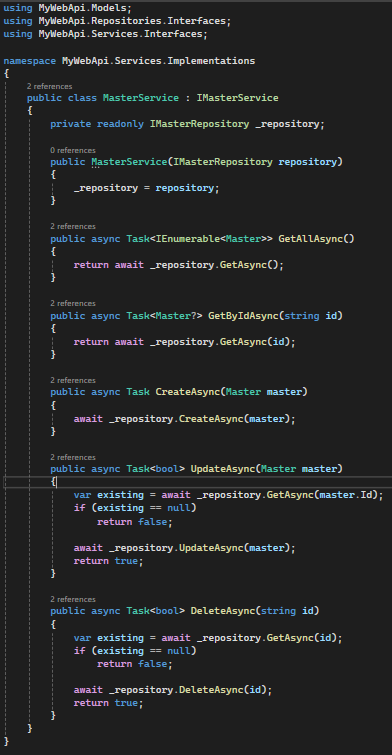
***Service/Implementation/TodoItemService.cs***

******

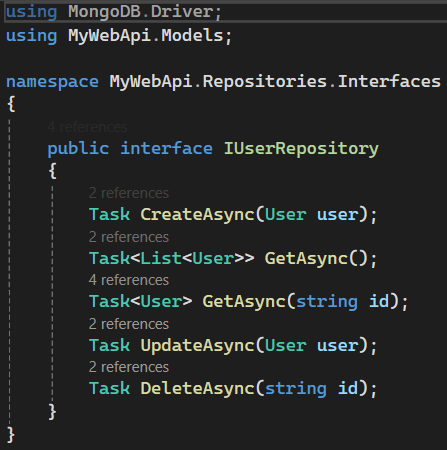
***Service/Implementation/BookingService.cs***

******

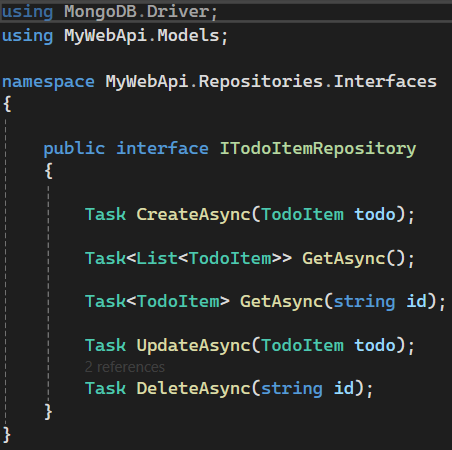
***Service/Implementation/MasterService.cs***

******

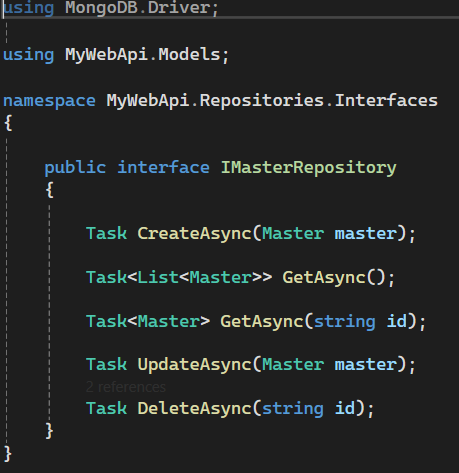
***Repositories/ Interfaces/IUserRepository.cs***

******

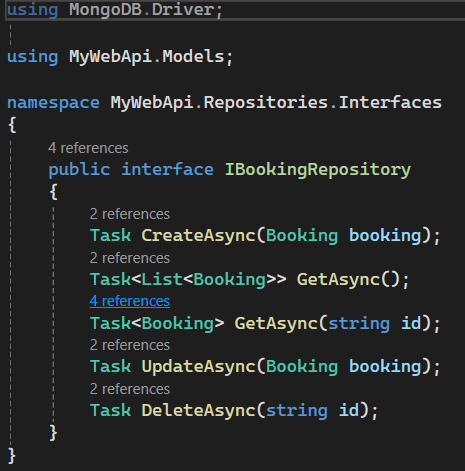
***Repositories/ Interfaces/ITodoItemRepository.cs***

******

***Repositories/ Interfaces/IMasterRepository.cs***

******

***Repositories/ Interfaces/IBookingRepository.cs***

******

***Repositories/Implementation/UserRepository.cs***

******

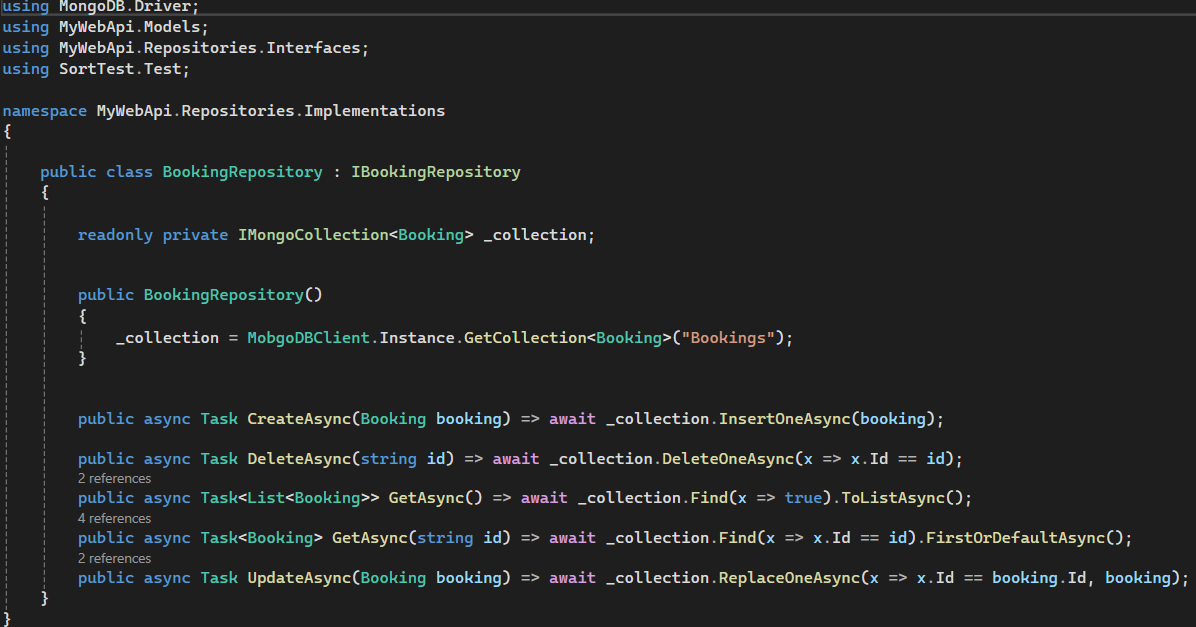
***Repositories/Implementation/TodoItemRepository.cs***

******

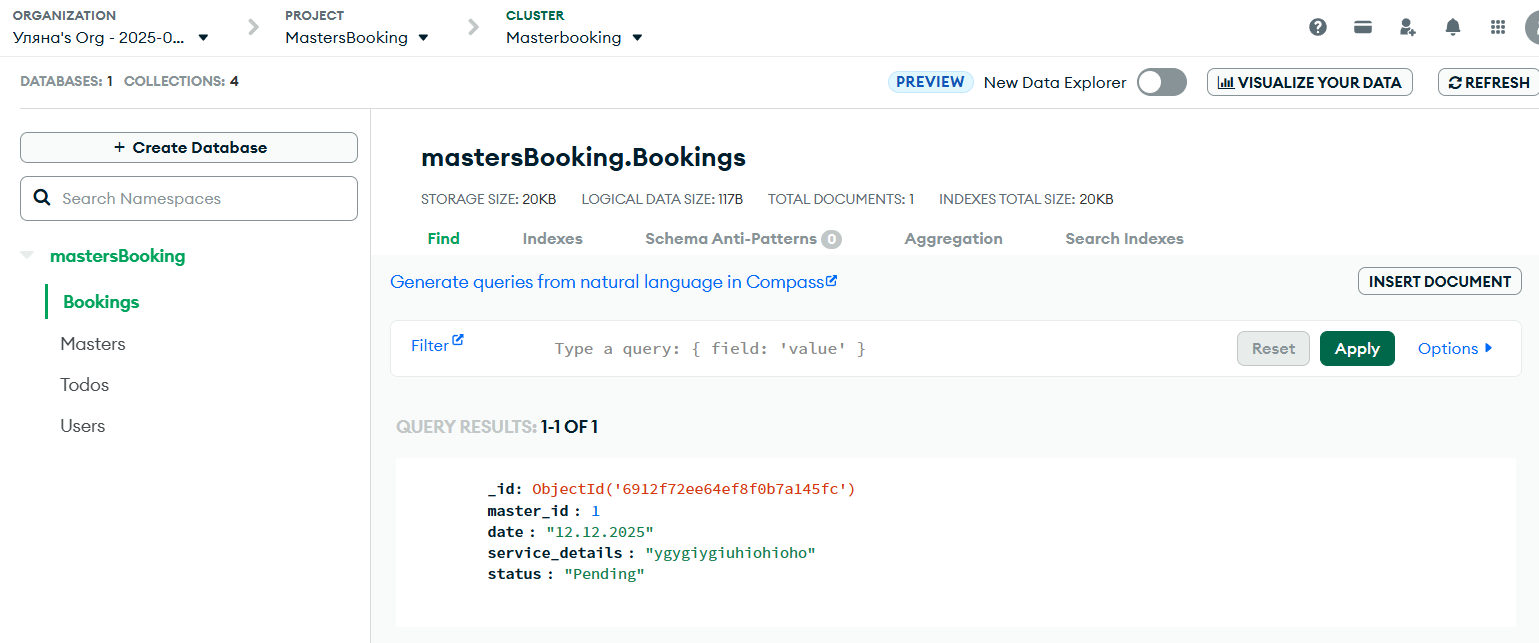
***Repositories/Implementation/MasterRepository.cs***

******

***Repositories/Implementation/BookingRepository.cs***

******

РЕЗУЛЬТАТИ (ТЕСТУВАННЯ CRUD-ОПЕРАЦІЙ ЧЕРЕЗ MONGODB)



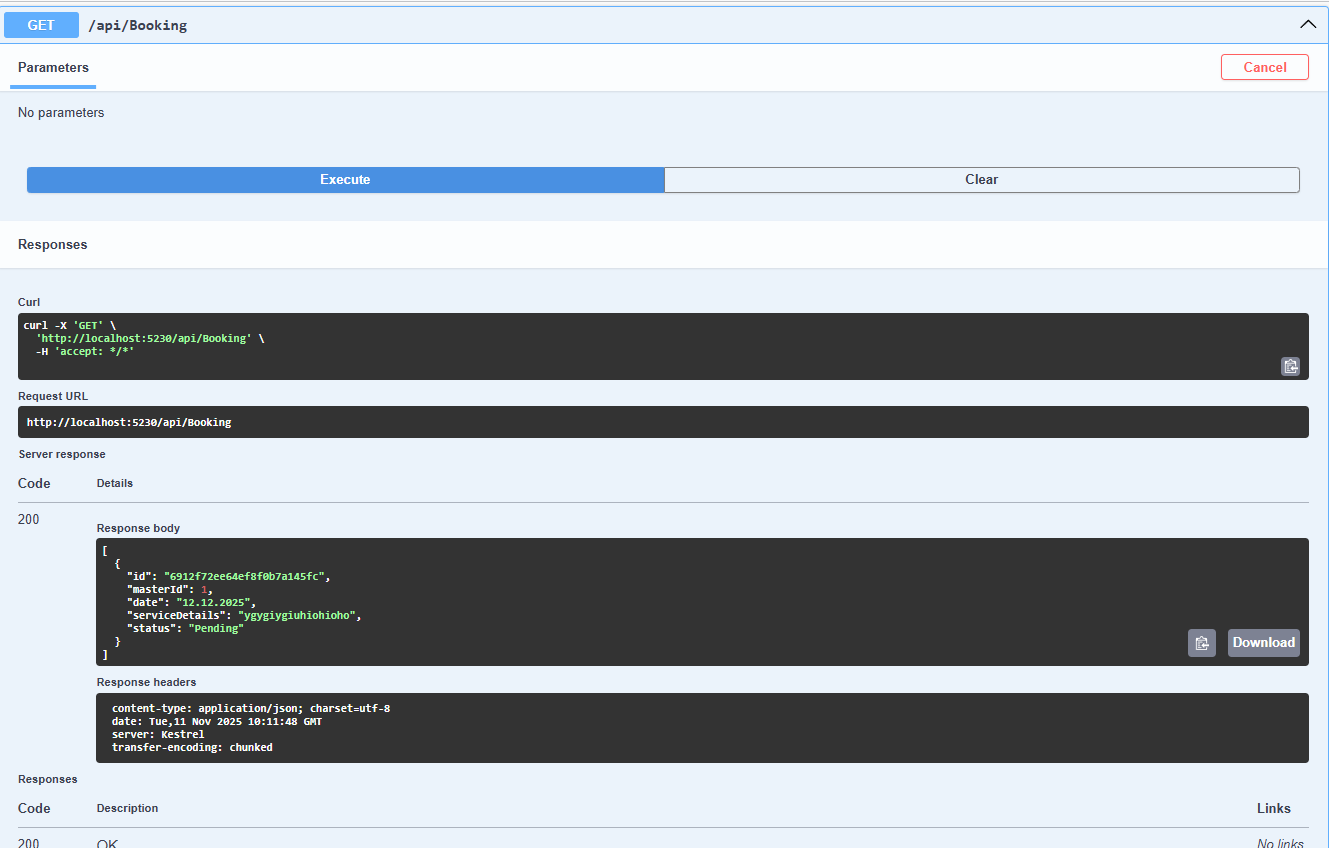
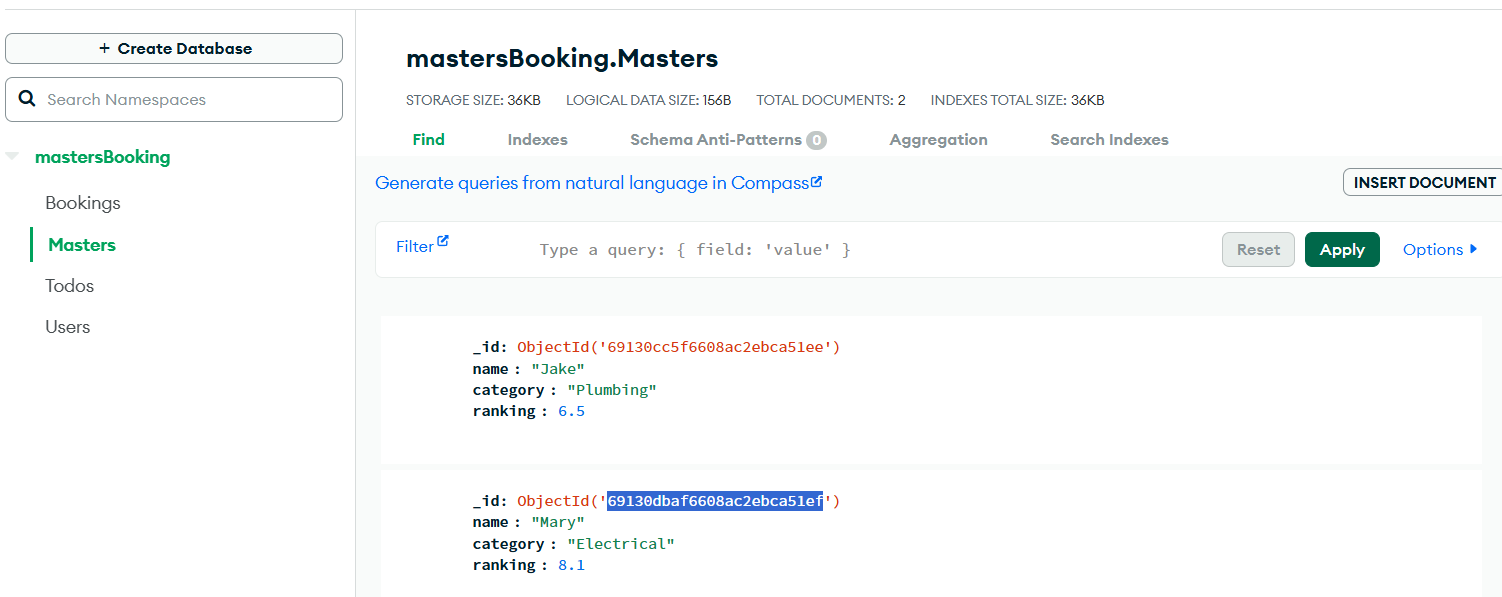


Рисунок 9 – GET() метод. Витягує список бронювання з колекції у MongoDB





Рисунок 10 – POST() метод. Створення нового документу у колекцію Masters



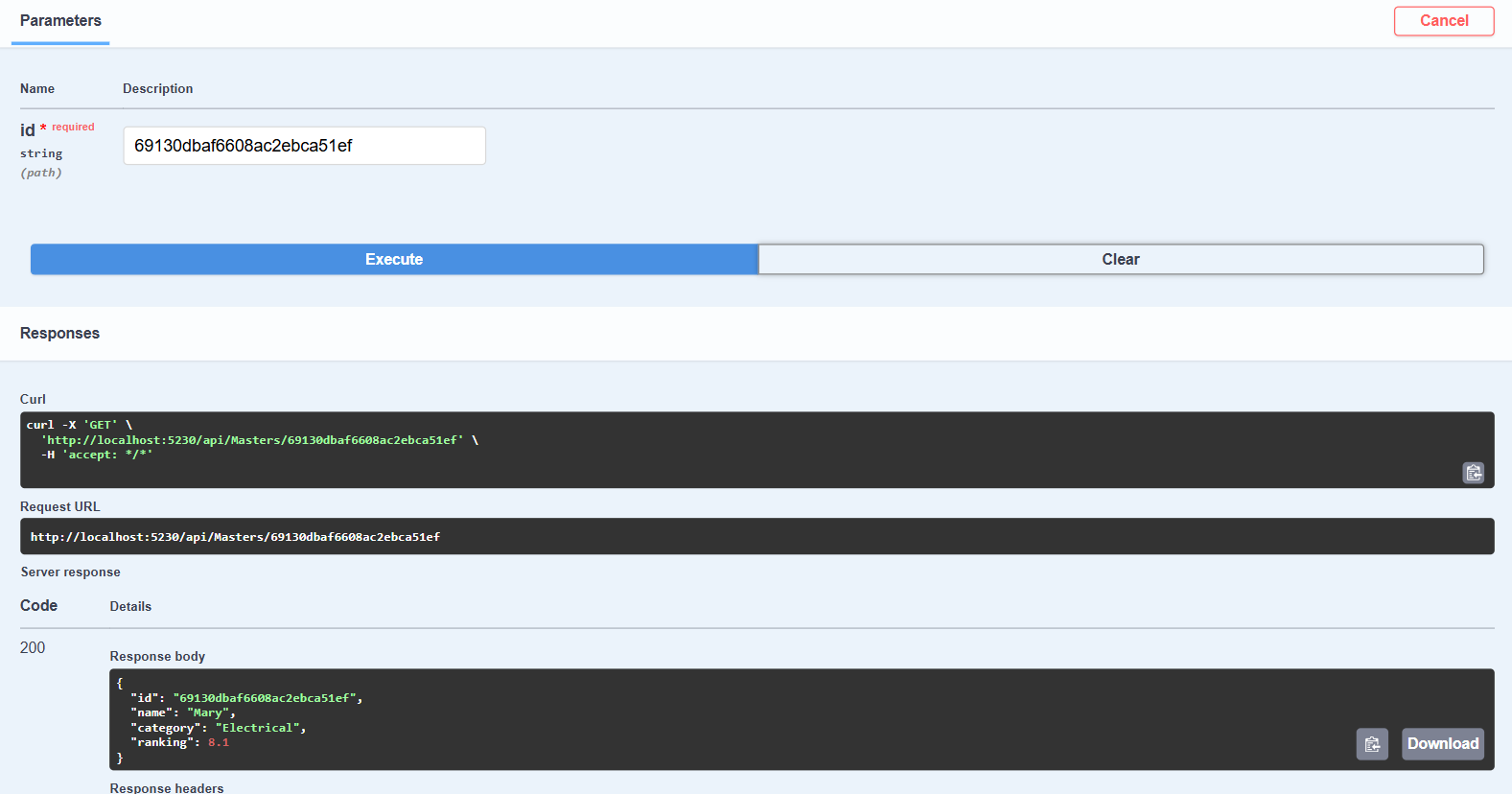
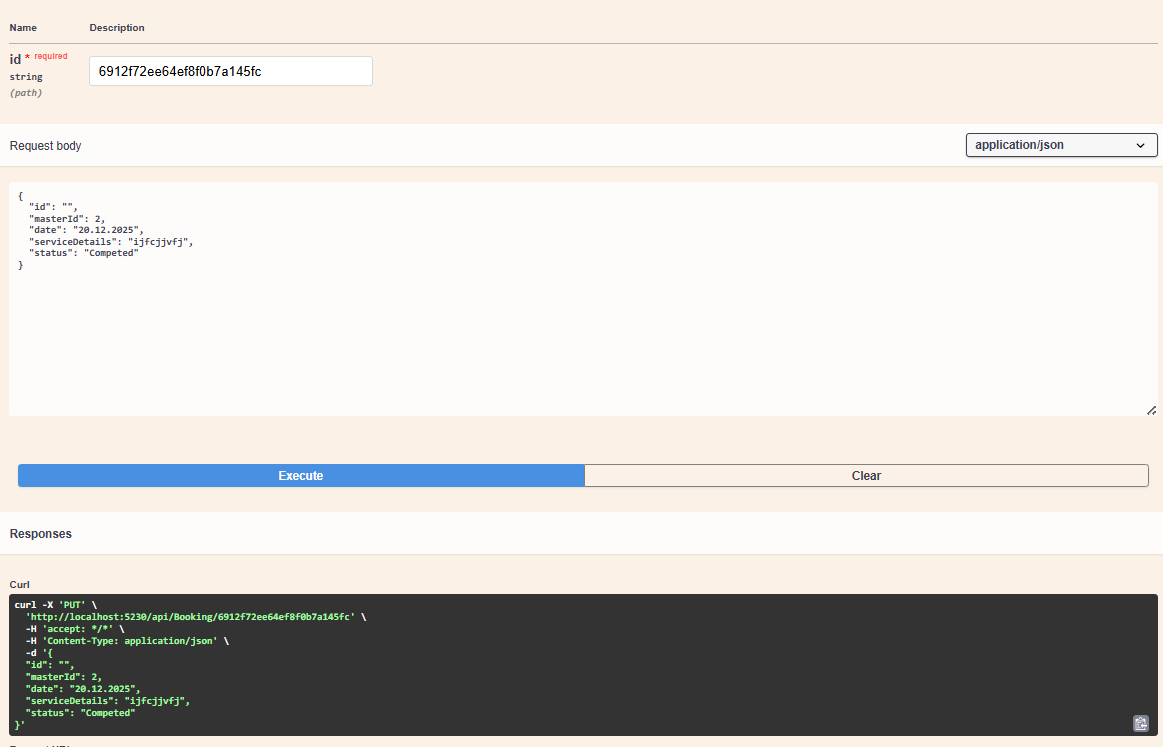


Рисунок 11 – GET(ID) метод. Пошук за id у колекції Masters



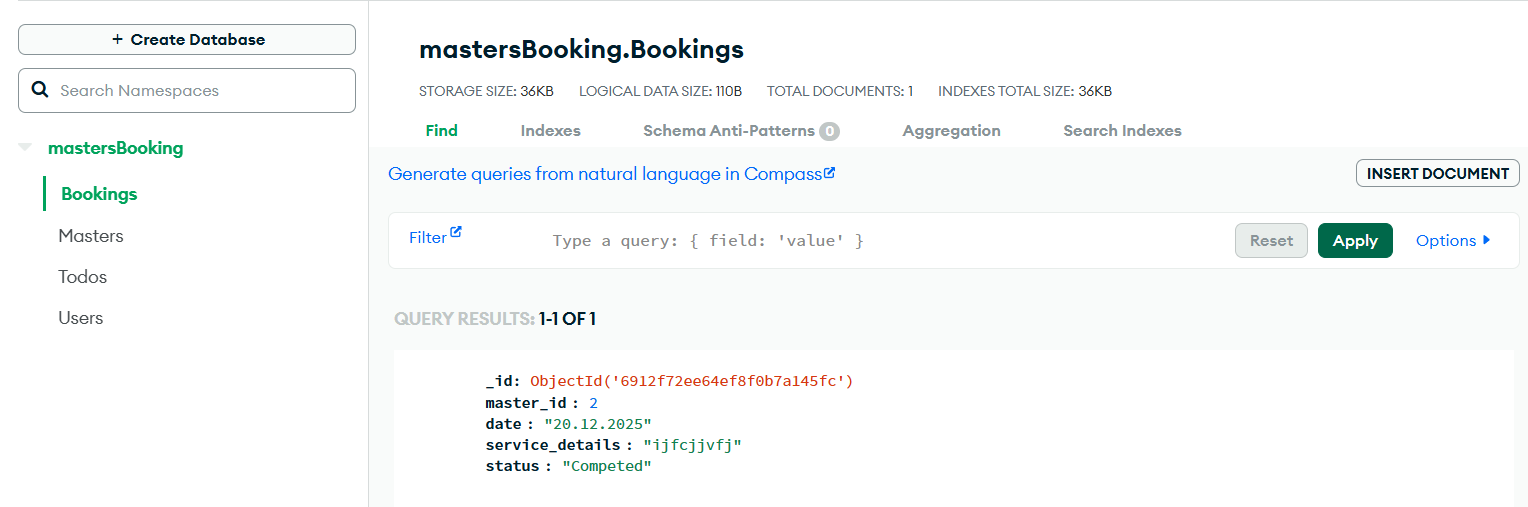
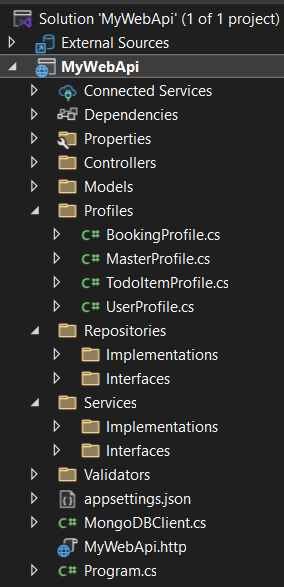


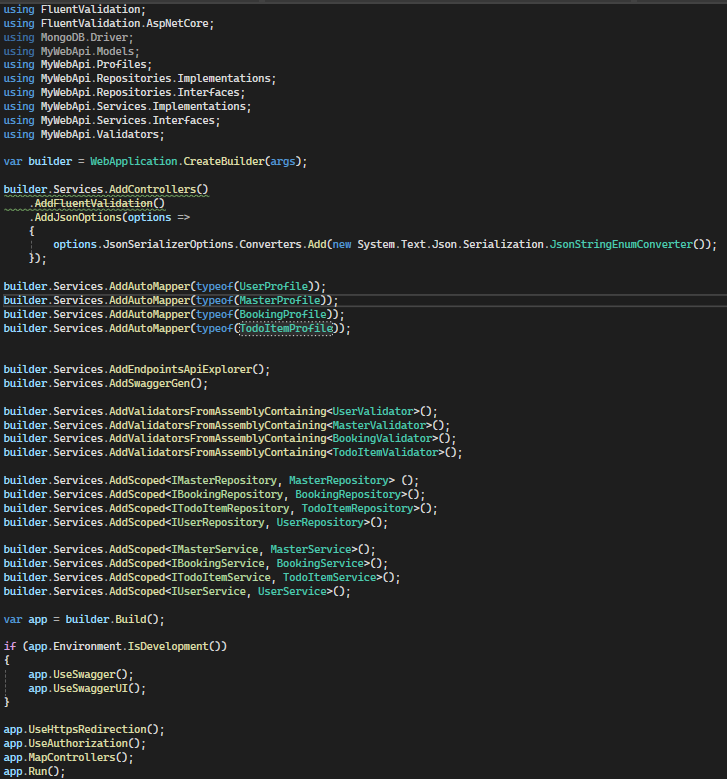
Рисунок 12 – PUT() метод. Оновлює дані документа, знайденого за id в колекції Booking

**Завдання 5.**

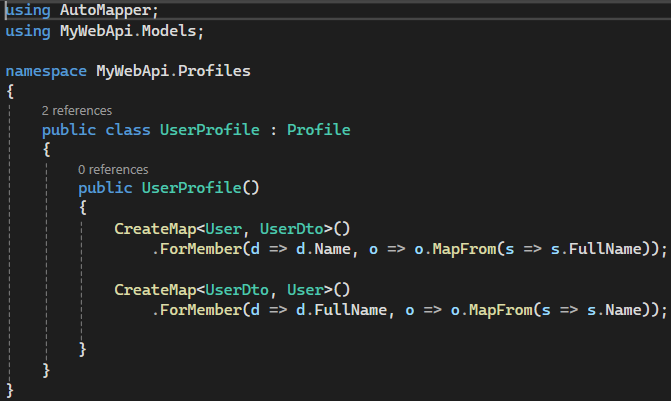
***Структура програми***



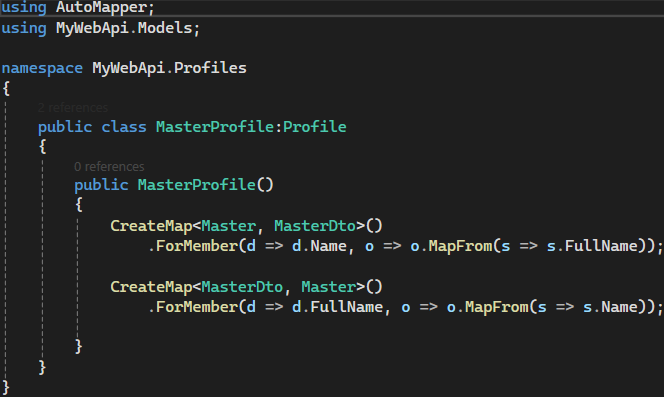
***Program.cs***

******

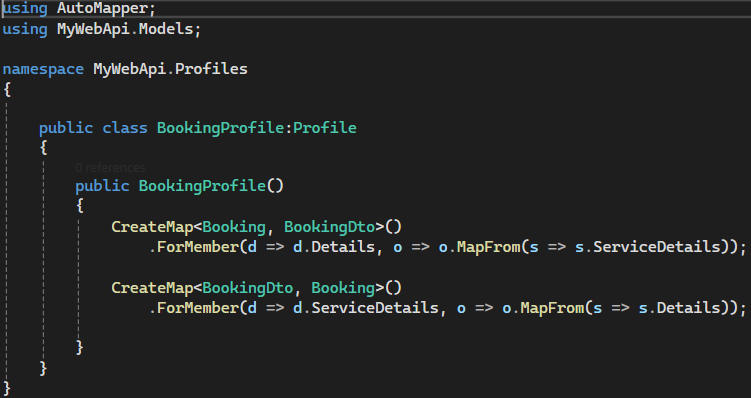
***Profiles/UserProfile.cs***

******

***Profiles/MasterProfile.cs***

******

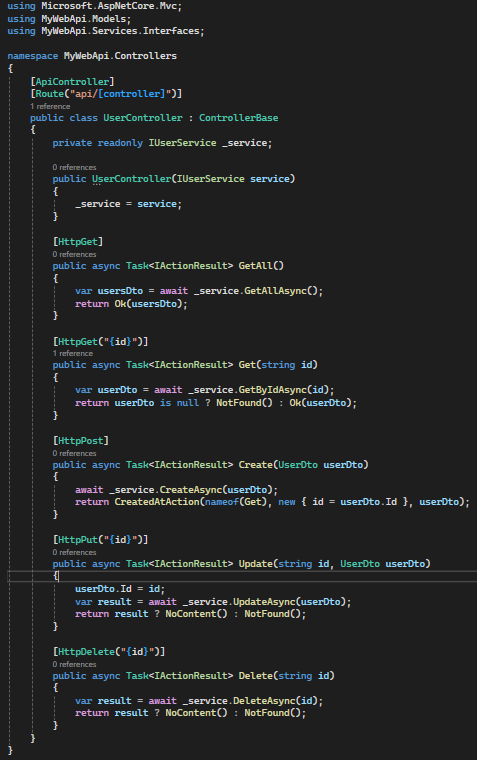
***Profiles/BookingProfile.cs***

******

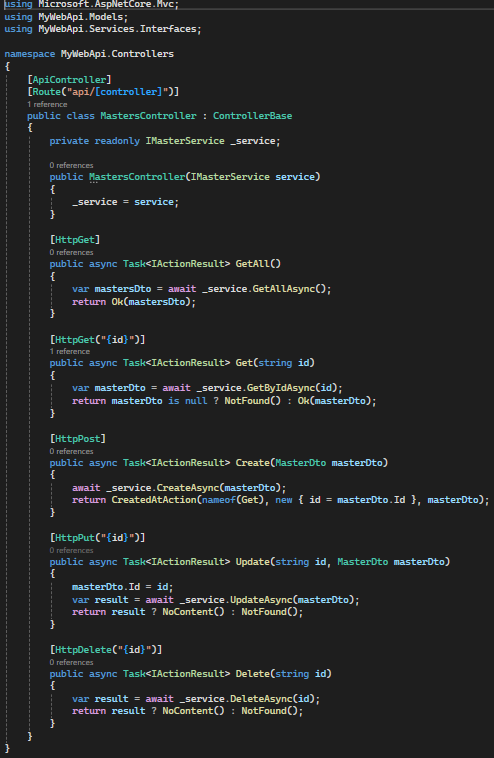
***Profiles/TodoItemProfile.cs***

******

***Controllers/UserController.cs***

******

***Controllers/MasterController.cs***

******

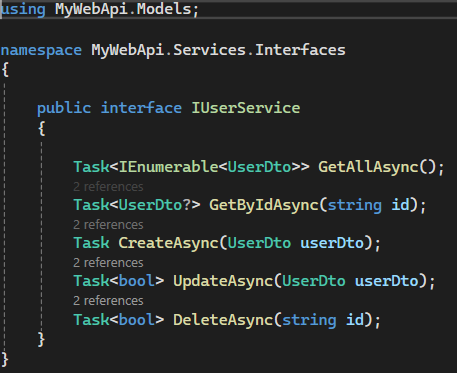
***Controllers/BookingController.cs***

******

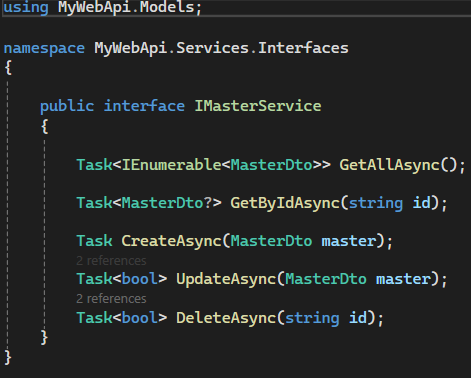
***Controllers/TodosController.cs***

******

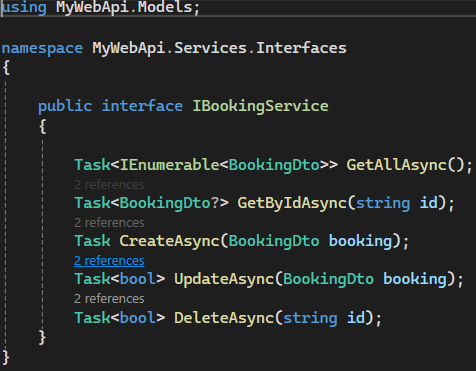
***Service/Interfaces/IUserService.cs***

******

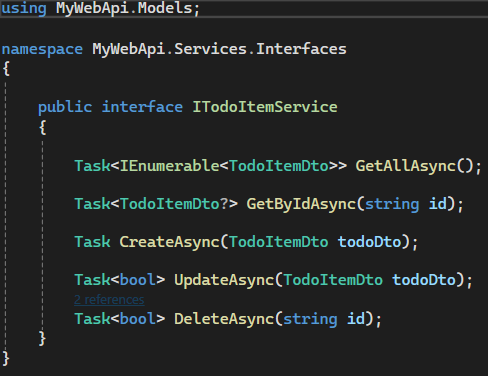
***Service/Interfaces/IMasterService.cs***

******

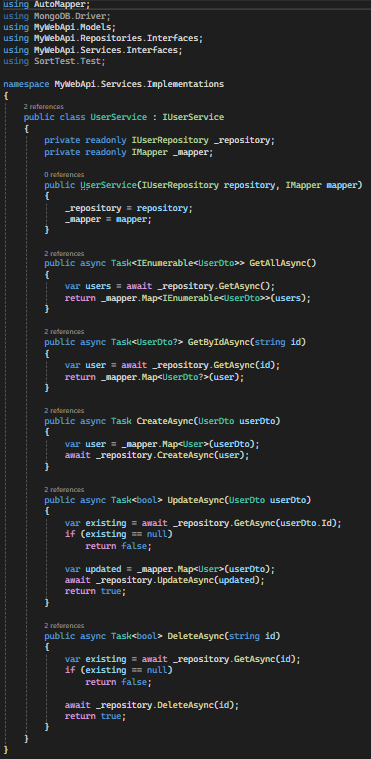
***Service/Interfaces/IBookingService.cs***

******

***Service/Interfaces/ITodoItemService.cs***

******

***Service/Implementation/UserService.cs***

******

***Service/Implementation/MasterService.cs***

******

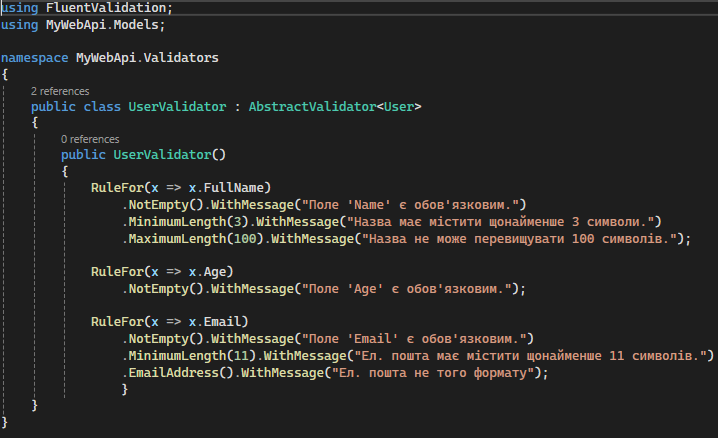
***Service/Implementation/BookingService.cs***

******

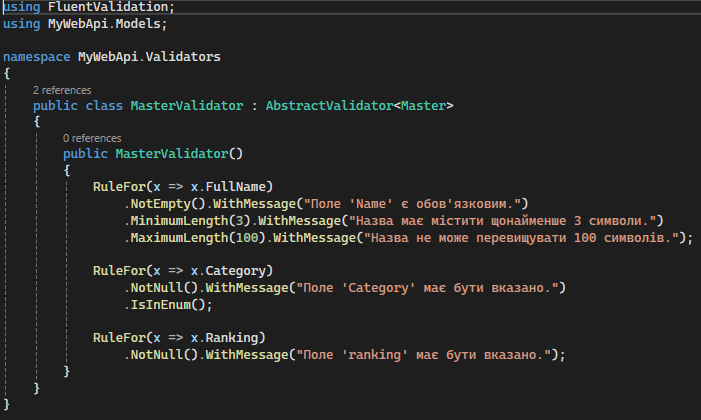
***Service/Implementation/TodoItemService.cs***

******

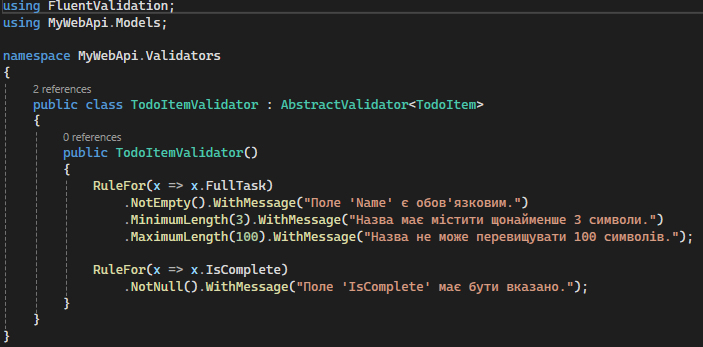
***Validators/UserValidator.cs***

******

***Validators/MasterValidator.cs***

******

***Validators/TodoItemValidator.cs***

******

ПОКАЗАТИ ПРИКЛАД МАПІНГУ (DTO ↔ MODEL) У КОДІ

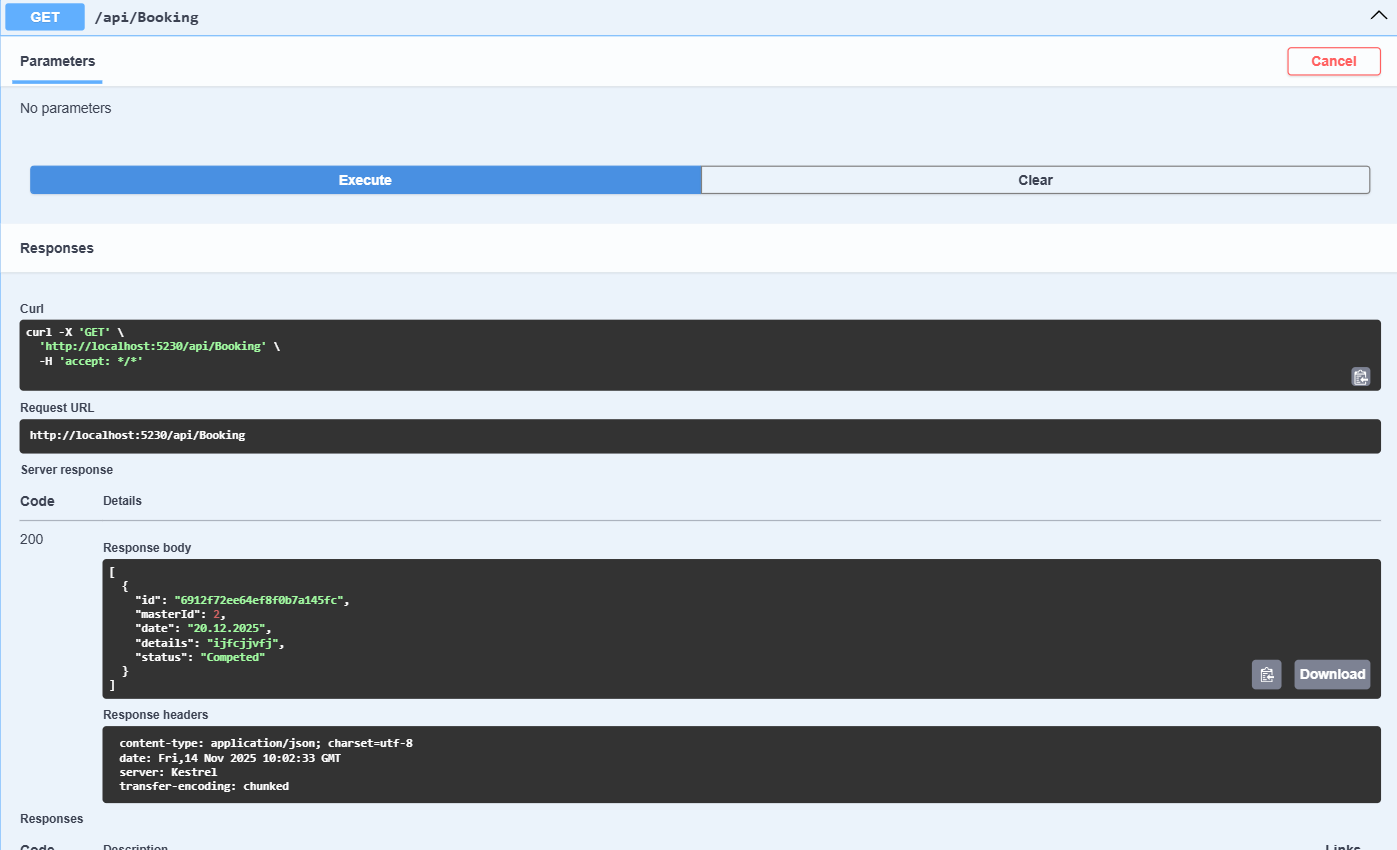
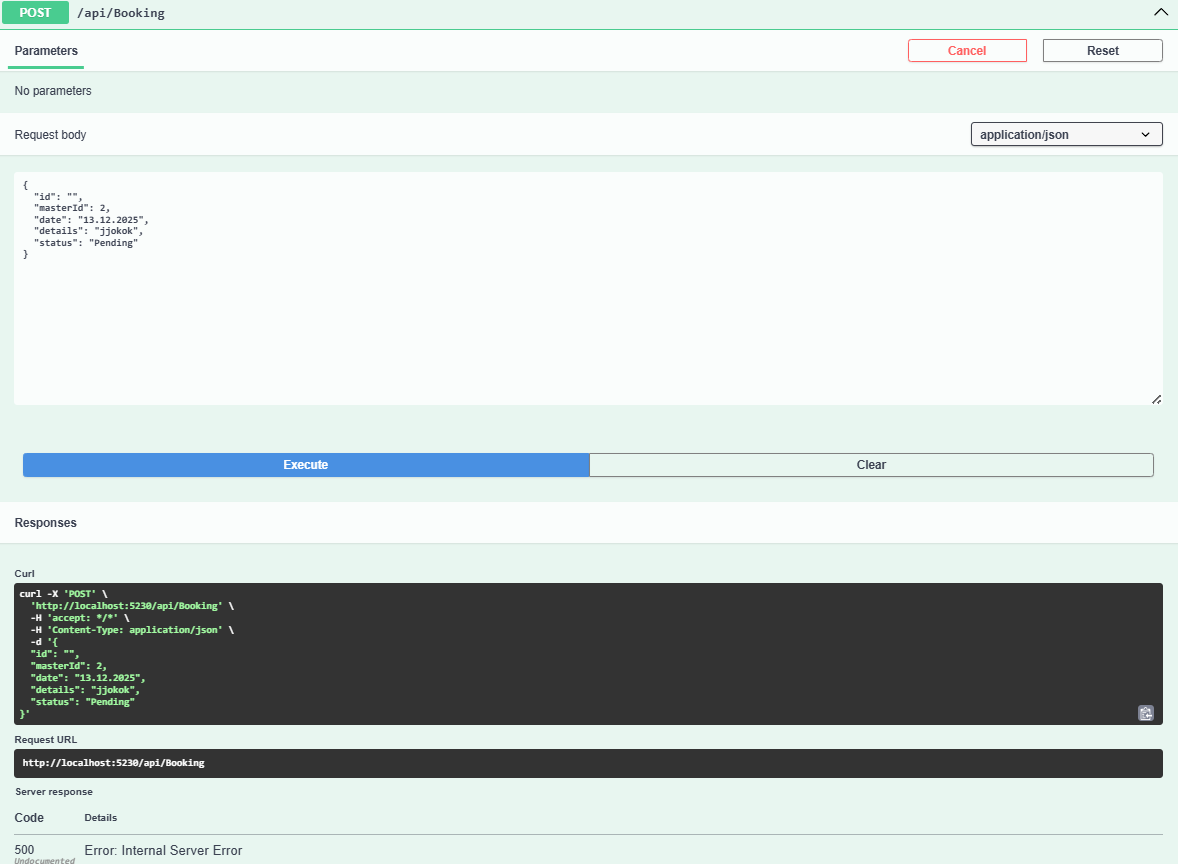
******

Рисунок 13 – GET() метод. Витягує список з колекції Bookings у MongoDB



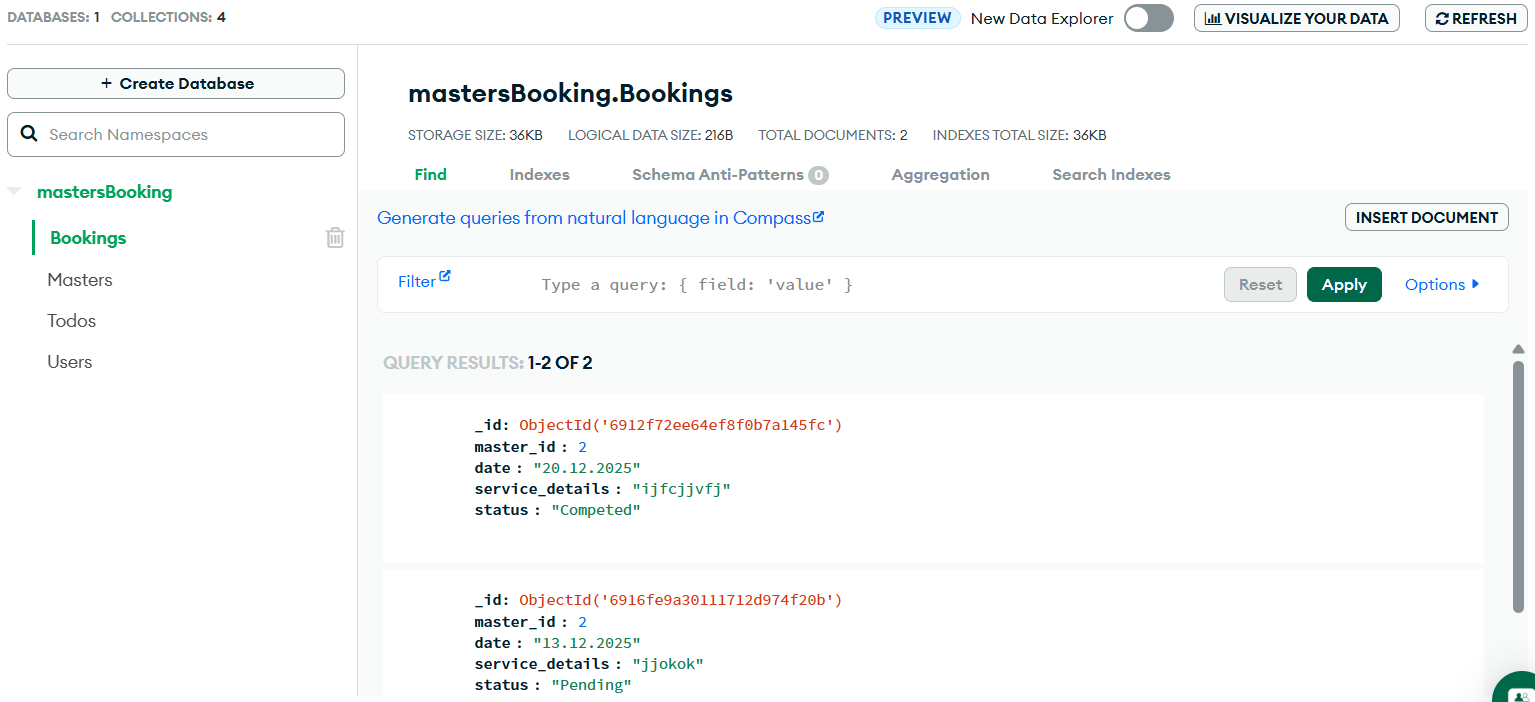
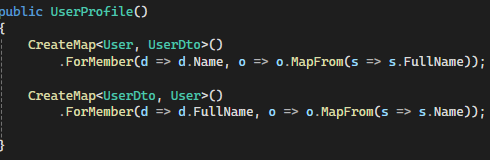


Рисунок 14 – POST() метод. Створює новий документ в колекції Bookings у MongoDB

ПОКАЗАТИ ПРИКЛАД МАПІНГУ (DTO ↔ MODEL) У КОДІ

Знаходить у Profile модулі



Перше переводить User -> UserTDO;

Друге переводить UserDTO -> User;

КОРОТКЕ ПОЯСНЕННЯ, ЯК ПРАЦЮЄ DI І AUTOMAPPER У ВАШОМУ ПРОЄКТІ.

DI дозволяє передавати залежності у клас замість створення їх всередині.

У проєкті контролери не створюють самостійно сервіси (ITodoItemService, IUserService, IMasterService). Їх постачає контейнер DI при створенні контролера.

AutoMapper використовується для конвертації між моделями бази даних (Model) і DTO, щоб не писати вручну new DTO { ... }.

Сервіси приймають DTO від контролера і конвертують їх у Model перед тим, як викликати репозиторій.

А коли репозиторій повертає Model, сервіс конвертує її в DTO перед тим, як віддати контролеру.

**Завдання 6.**