**Directory Structure**

MyOnionApp

├── MyOnionApp.API

│ ├── Controllers

│ │ └── ExampleController.cs

│ ├── Middleware

│ │ └── ExceptionHandlingMiddleware.cs

│ ├── Startup.cs

├── MyOnionApp.Application

│ ├── DTOs

│ │ ├── ApiResponse.cs

│ │ └── ExampleDto.cs

│ ├── Exceptions

│ │ ├── NotFoundException.cs

│ │ └── ValidationException.cs

│ ├── Mappings

│ │ └── MappingProfile.cs

│ ├── Services

│ │ └── ExampleService.cs

│ ├── Validators

│ │ └── ExampleValidator.cs

├── MyOnionApp.Domain

│ ├── Entities

│ │ └── ExampleEntity.cs

│ ├── Interfaces

│ │ └── IExampleRepository.cs

├── MyOnionApp.Infrastructure

│ ├── Data

│ │ ├── ApplicationDbContext.cs

│ │ └── ExampleRepository.cs

**1. MyOnionApp.API/Controllers/ExampleController.cs**

using Microsoft.AspNetCore.Mvc;

using MyOnionApp.Application.DTOs;

using MyOnionApp.Application.Services;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace MyOnionApp.API.Controllers

{

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

[ApiController]

public class ExampleController : ControllerBase

{

private readonly ExampleService \_service;

public ExampleController(ExampleService service)

{

\_service = service;

}

[HttpGet]

public async Task<ActionResult<ApiResponse<IEnumerable<ExampleDto>>>> GetAll()

{

var dtos = await \_service.GetAllAsync();

return Ok(ApiResponse<IEnumerable<ExampleDto>>.SuccessResponse(dtos, "Data retrieved successfully"));

}

[HttpGet("{id}")]

public async Task<ActionResult<ApiResponse<ExampleDto>>> GetById(int id)

{

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

if (dto == null)

{

return NotFound(ApiResponse<ExampleDto>.ErrorResponse(404, new List<string> { "Data not found" }));

}

return Ok(ApiResponse<ExampleDto>.SuccessResponse(dto, "Data retrieved successfully"));

}

[HttpPost]

public async Task<ActionResult<ApiResponse<ExampleDto>>> Add([FromBody] ExampleDto dto)

{

if (!ModelState.IsValid)

{

var errors = new List<string>();

foreach (var error in ModelState.Values)

{

foreach (var subError in error.Errors)

{

errors.Add(subError.ErrorMessage);

}

}

return BadRequest(ApiResponse<ExampleDto>.ErrorResponse(400, errors, "Validation errors occurred"));

}

await \_service.AddAsync(dto);

return CreatedAtAction(nameof(GetById), new { id = dto.Id }, ApiResponse<ExampleDto>.SuccessResponse(dto, "Data added successfully"));

}

}

}

**2. MyOnionApp.API/Middleware/ExceptionHandlingMiddleware.cs**

using Microsoft.AspNetCore.Http;

using MyOnionApp.Application.DTOs;

using MyOnionApp.Application.Exceptions;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Net;

using System.Threading.Tasks;

namespace MyOnionApp.API.Middleware

{

public class ExceptionHandlingMiddleware

{

private readonly RequestDelegate \_next;

public ExceptionHandlingMiddleware(RequestDelegate next)

{

\_next = next;

}

public async Task InvokeAsync(HttpContext context)

{

try

{

await \_next(context);

}

catch (Exception ex)

{

await HandleExceptionAsync(context, ex);

}

}

private static Task HandleExceptionAsync(HttpContext context, Exception exception)

{

HttpStatusCode statusCode;

string message;

var response = new ApiResponse<object> { Success = false };

switch (exception)

{

case NotFoundException notFoundException:

statusCode = HttpStatusCode.NotFound;

message = notFoundException.Message;

break;

case ValidationException validationException:

statusCode = HttpStatusCode.BadRequest;

response.Errors = validationException.Errors;

message = validationException.Message;

break;

default:

statusCode = HttpStatusCode.InternalServerError;

message = "An unexpected error occurred. Please try again later.";

break;

}

response.StatusCode = (int)statusCode;

response.Message = message;

var result = JsonConvert.SerializeObject(response);

context.Response.ContentType = "application/json";

context.Response.StatusCode = (int)statusCode;

return context.Response.WriteAsync(result);

}

}

}

**3. MyOnionApp.API/Startup.cs**

using AutoMapper;

using FluentValidation.AspNetCore;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using MyOnionApp.Application.Mappings;

using MyOnionApp.Application.Services;

using MyOnionApp.Application.Validators;

using MyOnionApp.Domain.Interfaces;

using MyOnionApp.Infrastructure.Data;

using MyOnionApp.API.Middleware;

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers()

.AddFluentValidation(fv => fv.RegisterValidatorsFromAssemblyContaining<ExampleValidator>());

services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer("YourConnectionStringHere"));

services.AddScoped<IExampleRepository, ExampleRepository>();

services.AddScoped<ExampleService>();

services.AddAutoMapper(typeof(MappingProfile));

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment())

{

app.UseDeveloperExceptionPage();

}

else

{

app.UseExceptionHandler("/error");

app.UseHsts();

}

app.UseMiddleware<ExceptionHandlingMiddleware>();

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthorization();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllers();

});

}

}

**4. MyOnionApp.Application/DTOs/ApiResponse.cs**

namespace MyOnionApp.Application.DTOs

{

public class ApiResponse<T>

{

public bool Success { get; set; }

public int StatusCode { get; set; }

public T Data { get; set; }

public string Message { get; set; }

public List<string> Errors { get; set; }

public ApiResponse()

{

Errors = new List<string>();

}

public static ApiResponse<T> SuccessResponse(T data, string message = null)

{

return new ApiResponse<T>

{

Success = true,

StatusCode = 200,

Data = data,

Message = message

};

}

public static ApiResponse<T> ErrorResponse(int statusCode, List<string> errors, string message = null)

{

return new ApiResponse<T>

{

Success = false,

StatusCode = statusCode,

Errors = errors,

Message = message

};

}

}

}

**5. MyOnionApp.Application/DTOs/ExampleDto.cs**

namespace MyOnionApp.Application.DTOs

{

public class ExampleDto

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**6. MyOnionApp.Application/Exceptions/NotFoundException.cs**

namespace MyOnionApp.Application.Exceptions

{

public class NotFoundException : Exception

{

public NotFoundException(string message) : base(message) { }

}

}

**7. MyOnionApp.Application/Exceptions/ValidationException.cs**

namespace MyOnionApp.Application.Exceptions

{

public class ValidationException : Exception

{

public List<string> Errors { get; }

public ValidationException(List<string> errors)

{

Errors = errors;

}

public ValidationException(string message, List<string> errors) : base(message)

{

Errors = errors;

}

}

}

**8. MyOnionApp.Application/Mappings/MappingProfile.cs**

using AutoMapper;

using MyOnionApp.Application.DTOs;

using MyOnionApp.Domain.Entities;

namespace MyOnionApp.Application.Mappings

{

public class MappingProfile : Profile

{

public MappingProfile()

{

CreateMap<ExampleEntity, ExampleDto>().ReverseMap();

}

}

}

**9. MyOnionApp.Application/Services/ExampleService.cs**

using AutoMapper;

using MyOnionApp.Application.DTOs;

using MyOnionApp.Application.Exceptions;

using MyOnionApp.Domain.Interfaces;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace MyOnionApp.Application.Services

{

public class ExampleService

{

private readonly IExampleRepository \_repository;

private readonly IMapper \_mapper;

public ExampleService(IExampleRepository repository, IMapper mapper)

{

\_repository = repository;

\_mapper = mapper;

}

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

{

var entities = await \_repository.GetAllAsync();

return \_mapper.Map<IEnumerable<ExampleDto>>(entities);

}

public async Task<ExampleDto> GetByIdAsync(int id)

{

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

if (entity == null)

{

throw new NotFoundException($"Entity with id {id} not found.");

}

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

}

public async Task AddAsync(ExampleDto dto)

{

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

await \_repository.AddAsync(entity);

}

}

}

**10. MyOnionApp.Application/Validators/ExampleValidator.cs**

using FluentValidation;

using MyOnionApp.Application.DTOs;

namespace MyOnionApp.Application.Validators

{

public class ExampleValidator : AbstractValidator<ExampleDto>

{

public ExampleValidator()

{

RuleFor(x => x.Name).NotEmpty().WithMessage("Name is required.");

}

}

}

**11. MyOnionApp.Domain/Entities/ExampleEntity.cs**

namespace MyOnionApp.Domain.Entities

{

public class ExampleEntity

{

public int Id { get; set; }

public string Name { get; set; }

}

}

**12. MyOnionApp.Domain/Interfaces/IExampleRepository.cs**

using MyOnionApp.Domain.Entities;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace MyOnionApp.Domain.Interfaces

{

public interface IExampleRepository

{

Task<IEnumerable<ExampleEntity>> GetAllAsync();

Task<ExampleEntity> GetByIdAsync(int id);

Task AddAsync(ExampleEntity entity);

}

}

**13. MyOnionApp.Infrastructure/Data/ApplicationDbContext.cs**

using Microsoft.EntityFrameworkCore;

using MyOnionApp.Domain.Entities;

namespace MyOnionApp.Infrastructure.Data

{

public class ApplicationDbContext : DbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options) { }

public DbSet<ExampleEntity> Examples { get; set; }

}

}

**14. MyOnionApp.Infrastructure/Data/ExampleRepository.cs**

using Microsoft.EntityFrameworkCore;

using MyOnionApp.Domain.Entities;

using MyOnionApp.Domain.Interfaces;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace MyOnionApp.Infrastructure.Data

{

public class ExampleRepository : IExampleRepository

{

private readonly ApplicationDbContext \_context;

public ExampleRepository(ApplicationDbContext context)

{

\_context = context;

}

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

{

return await \_context.Examples.ToListAsync();

}

public async Task<ExampleEntity> GetByIdAsync(int id)

{

return await \_context.Examples.FindAsync(id);

}

public async Task AddAsync(ExampleEntity entity)

{

await \_context.Examples.AddAsync(entity);

await \_context.SaveChangesAsync();

}

}

}

This setup includes a custom exception handling middleware, a standardized API response class, service and controller updates to handle exceptions, and a centralized way to manage validation and other custom exceptions. This structure provides a clean, maintainable, and scalable way to handle API responses and errors in your ASP.NET Core application.

Bottom of Form