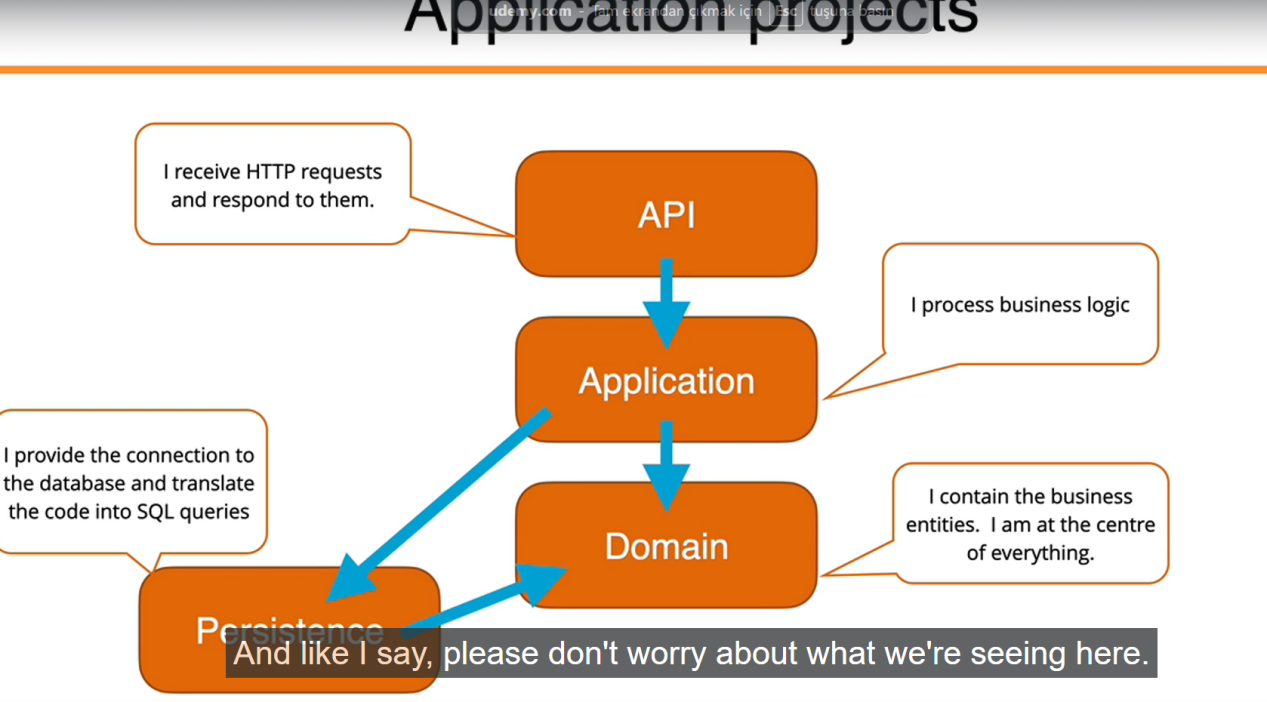
.NET

Bu bir clean architecturedur



MicrosoftEntityFrameWorkCore.Design Apiya

Microsoft ENtity FrameWorkCore.Sqlite Persistencea

builder.Services.AddDbContext<DataContext>(opt =>{

    opt.UseSqlite(builder.Configuration.GetConnectionString("DefaultConnection"));

});

Eger baska database kullansaydik opt.usesqlite degisirdi (opt.useSqlServer falan olurdu.)

>Sqliteopendatabase

Sonra databae ac

\*\* dotnet watchdan sonra dotnet ef database update yaptik (/API)

\*\* migrationsu da /persistence ta yaptik



Seed data olmadan

Seed varsa ise

using var scope = app.Services.CreateScope();

var services = scope.ServiceProvider;

try

{

    var context = services.GetRequiredService<DataContext>();

    var logger = services.GetRequiredService<ILogger<Program>>();

    logger.LogInformation("Dropping the database...");

    await context.Database.EnsureDeletedAsync();

    logger.LogInformation("Database dropped.");

    logger.LogInformation("Starting database migration...");

    await context.Database.MigrateAsync();

    logger.LogInformation("Database migration completed.");

    logger.LogInformation("Starting data seeding...");

    await Seed.SeedData(context);

    logger.LogInformation("Data seeding completed.");

}

catch (Exception ex)

{

    var logger = services.GetRequiredService<ILogger<Program>>();

    logger.LogError(ex, "An error occurred during migration or seeding data");

}

app.Run();

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.defer-datasource-initialization=true

Springde seed data boyle calisir

Infrastructure dis servis entegrasyonlari yapilir

Entıtyde constructorlar get setler yaptık

Cqrs bir yaklasimdir

Applications sag tik manage nugets

Mediatr commandi alir handle eder

CQRSTE

Requestin commandi

RequestinREsponseu

Controllerda BaseController

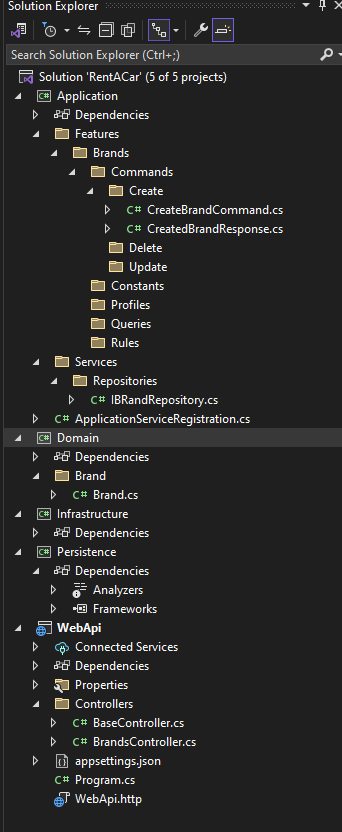
BrandController

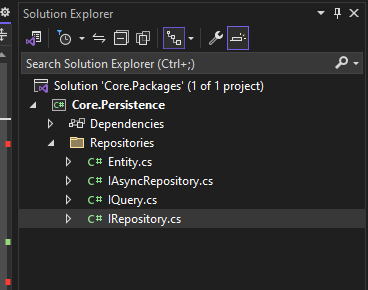
ApplicationServiceRegistration

Program.cse

builder.Services.AddApplicationServices(); // \*\*

eklenir





using Core.Persistence.Repositories;

using Domain.Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Application.Services.Repositories

{

public interface IBrandRepository: IAsyncRepository<Brand>, IRepository<Brand>

{

}

}

Core.persistence.repositories ile goruyor

Mapperi Querylerde kullaniriz

Mapper dtoya donusturmeyi saglar.

using MediatR;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Application.Features.Brands.Commands.Create

{

public class CreateBrandCommand:IRequest<CreatedBrandResponse> // apiden bir komut gelicek

{

public string Name { get; set; }

public class CreateBrandCommandHandler : IRequestHandler<CreateBrandCommand, CreatedBrandResponse>

{

public Task<CreatedBrandResponse> Handle(CreateBrandCommand request, CancellationToken cancellationToken)

{

CreatedBrandResponse createdBrandResponse = new CreatedBrandResponse();

createdBrandResponse.Name = request.Name;

createdBrandResponse.Id = new Guid();

return null;

}

}

}

}

Simdilik bunu gorduk AMA

namespace Application.Features.Brands.Commands.CreateBrand

{

public partial class CreateBrandCommand:IRequest<CreatedBrandDto>

{

public string Name { get; set; }

public class CreateBrandCommandHandler : IRequestHandler<CreateBrandCommand, CreatedBrandDto>

{

private readonly IBrandRepository \_brandRepository;

private readonly IMapper \_mapper;

private readonly BrandBusinessRules \_brandBusinessRules;

public CreateBrandCommandHandler(IBrandRepository brandRepository, IMapper mapper, BrandBusinessRules brandBusinessRules)

{

\_brandRepository = brandRepository;

\_mapper = mapper;

\_brandBusinessRules = brandBusinessRules;

}

public async Task<CreatedBrandDto> Handle(CreateBrandCommand request, CancellationToken cancellationToken)

{

await \_brandBusinessRules.BrandNameCanNotBeDuplicatedWhenInserted(request.Name);

Brand mappedBrand = \_mapper.Map<Brand>(request);

Brand createdBrand = await \_brandRepository.AddAsync(mappedBrand);

CreatedBrandDto createdBrandDto = \_mapper.Map<CreatedBrandDto>(createdBrand);

return createdBrandDto;

}

}

}

}

Burada handlelayarak repositoryle iliskilendirioyurz

Task<T> AddAsync(T entity);

Metodu IASYNCREPO interfaceinde yazili

# **DbContext.SaveChangesAsync Yöntem**

Bu ef core yontemidir

DynamicQuery arama filtreleme gibi bir sey

--------neil cummingse gectik oburu agir geldiii

dotnet new list

webapi 3 class library sln , solution file create yapacagiz

Write-Host "About to Create the directory" -ForegroundColor Green

mkdir Reactivities

cd Reactivities

Write-Host "About to create the solution and projects" -ForegroundColor Green

dotnet new sln

dotnet new webapi -n API --use-controllers

dotnet new classlib -n Application

dotnet new classlib -n Domain

dotnet new classlib -n Persistence

Write-Host "Adding projects to the solution" -ForegroundColor Green

dotnet sln add API/API.csproj

dotnet sln add Application/Application.csproj

dotnet sln add Domain/Domain.csproj

dotnet sln add Persistence/Persistence.csproj

Write-Host "Adding project references" -ForegroundColor Green

cd API

dotnet add reference ../Application/Application.csproj

cd ../Application

dotnet add reference ../Domain/Domain.csproj

dotnet add reference ../Persistence/Persistence.csproj

cd ../Persistence

dotnet add reference ../Domain/Domain.csproj

cd ..

Write-Host "Executing dotnet restore" -ForegroundColor Green

dotnet restore

Write-Host "Finished!" -ForegroundColor Green

Bunu hep yap her proje yaptiktan once

Weather falan ornek amacli veriliyor

using MediatR;

using Microsoft.AspNetCore.Mvc;

namespace API.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class BaseApiController : ControllerBase

    {

     private IMediator \_mediator;

     protected IMediator Mediator => \_mediator ??= HttpContext.RequestServices.GetService<IMediator>();

    }

}

Boylce thu n controllers in api yapiyoruz

Get details

using Domain;

using MediatR;

using Persistence;

namespace Application.Activities

{

    public class Details

    {

        public class Query : IRequest<Activity>

        {

            public Guid Id { get; set; }

        }

        public class Handler : IRequestHandler<Query, Activity>

        {

            private readonly DataContext \_context;

            public Handler(DataContext context)

            {

                \_context = context;

            }

            public async Task<Activity> Handle(Query request, CancellationToken cancellationToken)

            {

                return await \_context.Activities.FindAsync(request.Id);

            }

        }

    }

}

Controller

using Application.Activities;

using Domain;

using MediatR;

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using Persistence;

namespace API.Controllers

{

    public class ActivitiesController : BaseApiController

    {

        //dependency injection

       // mediatr cqrsten sonra kaldirdik private readonly DataContext \_context;

        /\*private readonly IMediator \_mediator;

        public ActivitiesController(IMediator mediator)

        {

            \_mediator = mediator;

        }\*/

        // base controllerda tanimladik

        [HttpGet] //api/activities

        public async Task<ActionResult<List<Activity>>> GetActivities()

        {

            // return await \_context.Activities.ToListAsync();

            return await Mediator.Send(new List.Query());

        }

        [HttpGet("{id}")] //api/activities/4

        public async Task<ActionResult<Activity>> GetActivity(Guid id)

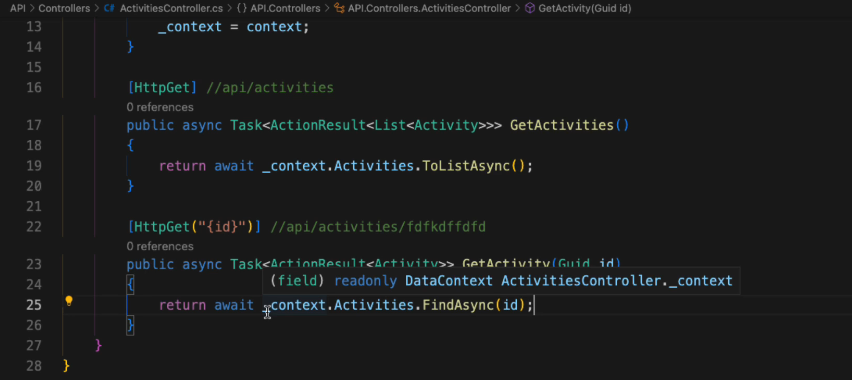
        {

            //return await \_context.Activities.FindAsync(id);

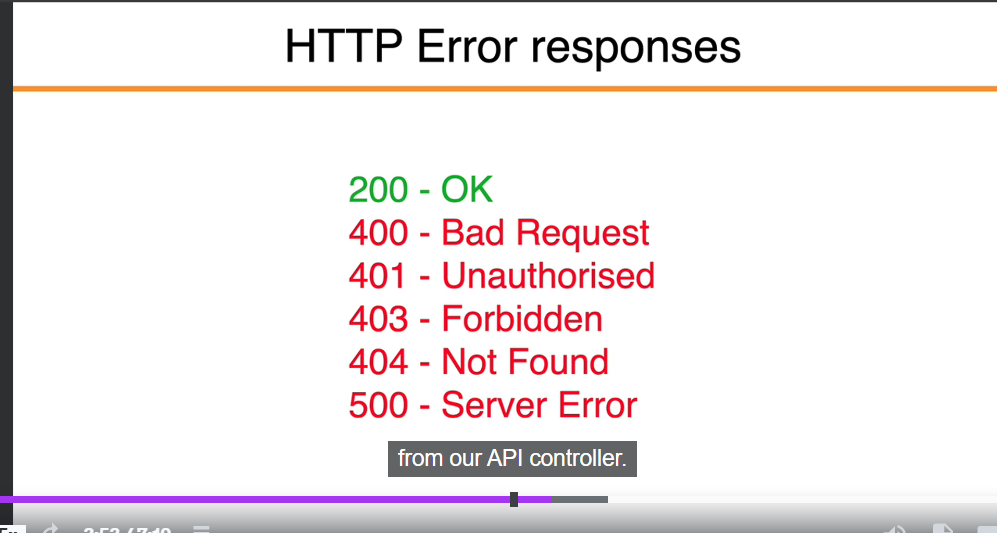
        return  await Mediator.Send(new Details.Query{Id = id});

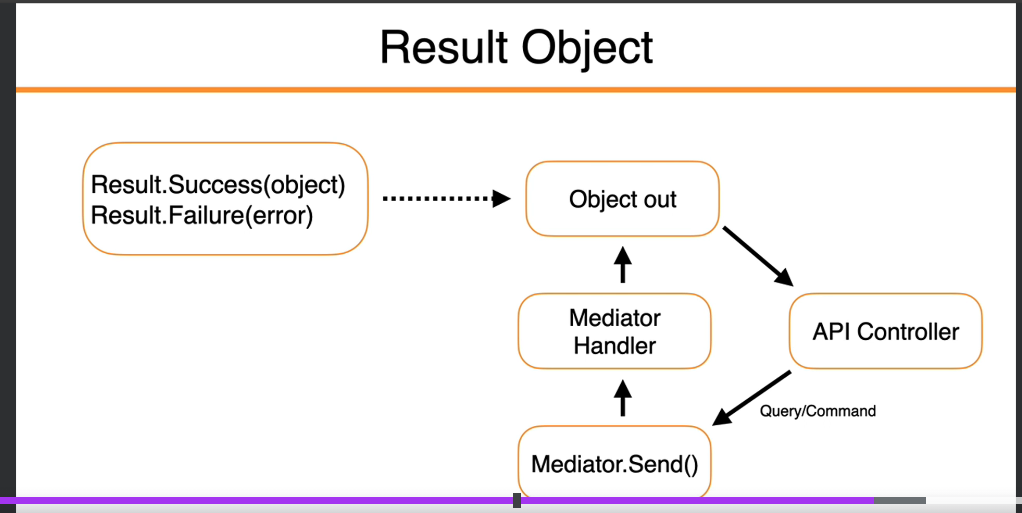
        }

Cqrs olmasa boyle olurdu :



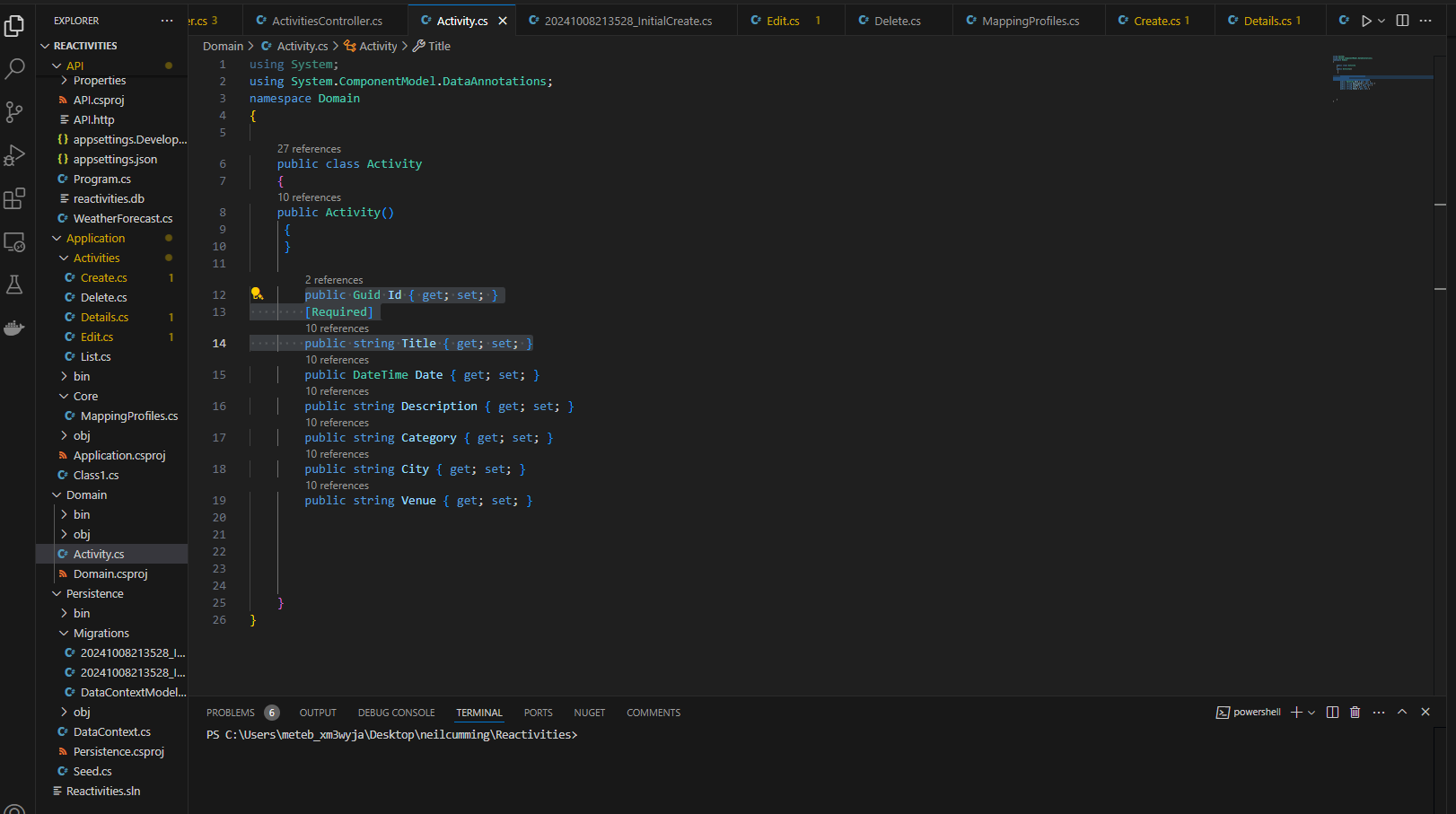
Error handling





Degerleri yazmadan da post yapabiliyoruz

Validate with data annotatiton



Required yazdik ekledik

Fluent annotation

using Domain;

using FluentValidation;

using MediatR;

using Persistence;

namespace Application.Activities

{

    public class Create

    {

   // hicbir sey dondurmuyor o  yuzzden <Activity yok>

   public class  Command: IRequest

   {

    public Activity Activity { get; set; }

   }

public class CommandValidator : AbstractValidator <Activity>

{

    public CommandValidator()

    {

        RuleFor(x => x.Title).NotEmpty();   // title bos olamaz activityden

    }

}

public class Handler : IRequestHandler<Command>

{

    private readonly DataContext \_context;

public Handler(DataContext context)

{

    \_context = context;

}

public async Task<Unit> Handle(Command request, CancellationToken cancellationToken)

{

    \_context.Activities.Add(request.Activity);

    await \_context.SaveChangesAsync();

    return Unit.Value;

}

List<int> intList = new List<int>(); // Sadece int türünden değerler kabul eder

intList.Add(1); // Geçerli

intList.Add(2);

List<string> stringList = new List<string>(); // Sadece string türünden değerler kabul eder

stringList.Add("hello"); // Geçerli

stringList.Add("world");

Acitivityvalidator ile :

public class CommandValidator : AbstractValidator <Command>   //

{

    public CommandValidator()

    {

        // RuleFor(x => x.Title).NotEmpty();   // title bos olamaz activityden

              RuleFor(x =>x.Activity).SetValidator(new ActivityValidator());

    }

    /\*

    var command = new Command { Activity = new Activity(), Title = "Example Title" };

var validator = new CommandValidator();

var result = validator.Validate(command);

if (!result.IsValid)

{

    // Hatalar var, burada işlenir.

}

    \*/

}

ActivityValidator

using Domain;

using FluentValidation;

namespace Application.Activities

{

    public class ActivityValidator : AbstractValidator<Activity>

    {

        public ActivityValidator()

        {

            RuleFor(x => x.Title).NotEmpty();

            RuleFor(x => x.Description).NotEmpty();

            RuleFor(x => x.Date).NotEmpty();

            RuleFor(x => x.Category).NotEmpty();

            RuleFor(x => x.City).NotEmpty();

            RuleFor(x => x.Venue).NotEmpty();

        }

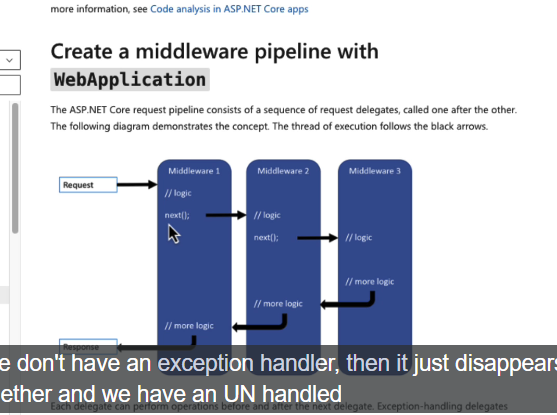
Exception

 if (activity == null)

                {

                    throw new Exception("Could not find activity");

                }



app.UseMiddleware<ExceptionMiddleware>();

Program.cseye yazdik bunu

Middleware.

using System.Net;

using System.Text.Json;

using Application.Core;

namespace API.Middleware

{

    public class ExceptionMiddleware

    {

        private readonly RequestDelegate \_next;

        private readonly ILogger<ExceptionMiddleware> \_logger;

        private readonly IHostEnvironment \_env;

        public ExceptionMiddleware(RequestDelegate next, ILogger<ExceptionMiddleware> logger,IHostEnvironment env)

        {

            \_next = next;

            \_logger = logger;

            \_env = env;

        }

        public async Task InvokeAsync(HttpContext context)

        {

            try

            {

                await \_next(context);

            }

            catch (Exception ex)

            {

                \_logger.LogError(ex, ex.Message);

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

                context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;

                var response = \_env.IsDevelopment()

                    ? new AppException(context.Response.StatusCode, ex.Message, ex.StackTrace?.ToString())

                    : new AppException(context.Response.StatusCode, "Internal Server Error");

                var options = new JsonSerializerOptions { PropertyNamingPolicy = JsonNamingPolicy.CamelCase };

                var json = JsonSerializer.Serialize(response, options);

                await context.Response.WriteAsync(json);

            }

        }

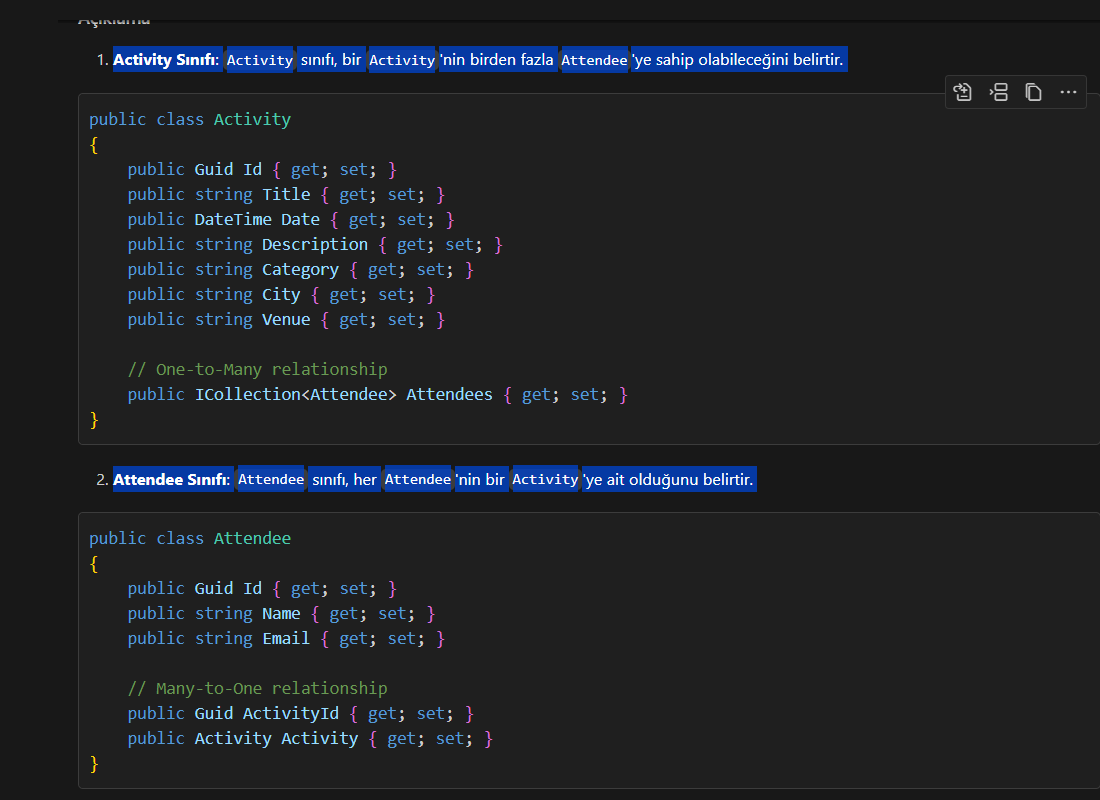
    }

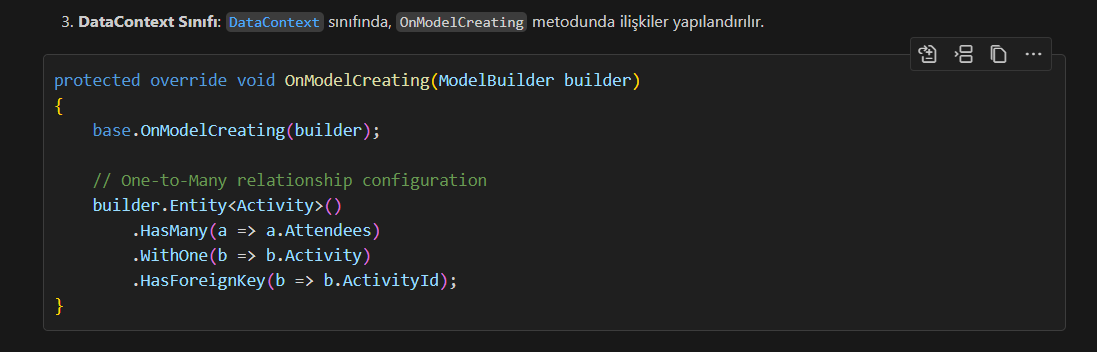
}

---------------------------------------------------

FindAsync metodu, birincil anahtar (primary key) ile bir varlığı veritabanında aramak için kullanılır. Ancak, FindByTitle gibi bir sorguda, başlık (title) gibi bir alan üzerinden arama yapmanız gerektiğinde, FindAsync yerine FirstOrDefaultAsync veya SingleOrDefaultAsync gibi metodları kullanmanız gerekir. Bu metodlar, belirli bir koşulu sağlayan ilk veya tek varlığı döndürür.

OnetomanyManytoOne





Hayır, OnModelCreating metodunda one-to-many ilişkiyi yapılandırırken, many-to-one ilişkiyi ayrıca yapılandırmanıza gerek yoktur

ProjectTo metodu, AutoMapper kullanarak bir sorgunun sonuçlarını doğrudan bir DTO'ya projekte etmek için kullanılır. Bu, veritabanından çekilen verilerin doğrudan DTO'ya dönüştürülmesini sağlar ve performans açısından daha verimli olabilir. Ancak, ProjectTo metodu genellikle bir liste veya koleksiyon döndüren sorgular için kullanılır.

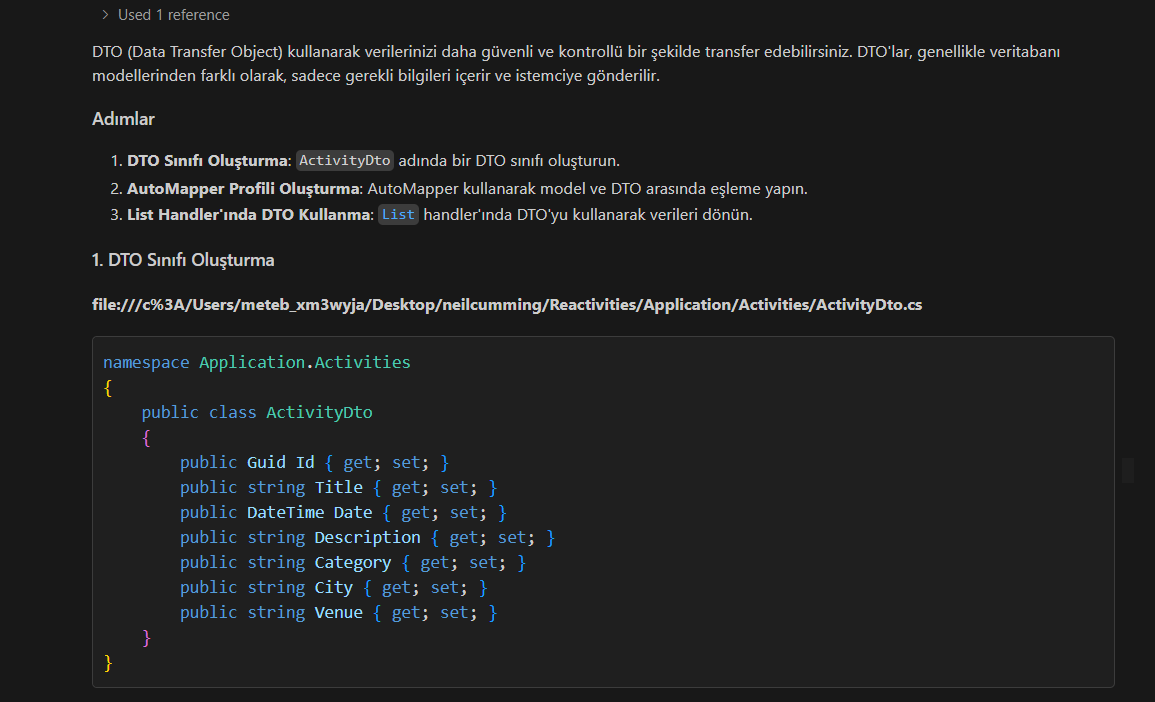
Verilen kod parçasında, Edit işlemi için bir [Handler](vscode-file://vscode-app/c:/Users/meteb_xm3wyja/AppData/Local/Programs/Microsoft VS Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.esm.html" \o ") sınıfı tanımlanmış. Bu sınıf, bir [Command](vscode-file://vscode-app/c:/Users/meteb_xm3wyja/AppData/Local/Programs/Microsoft VS Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.esm.html" \o ") nesnesini alır ve veritabanında ilgili [Activity](vscode-file://vscode-app/c:/Users/meteb_xm3wyja/AppData/Local/Programs/Microsoft VS Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.esm.html" \o ") nesnesini günceller. Bu durumda, ProjectTo kullanımı gerekli değildir çünkü tek bir nesne üzerinde işlem yapıyorsunuz ve bu nesneyi doğrudan güncelleyebilirsiniz.

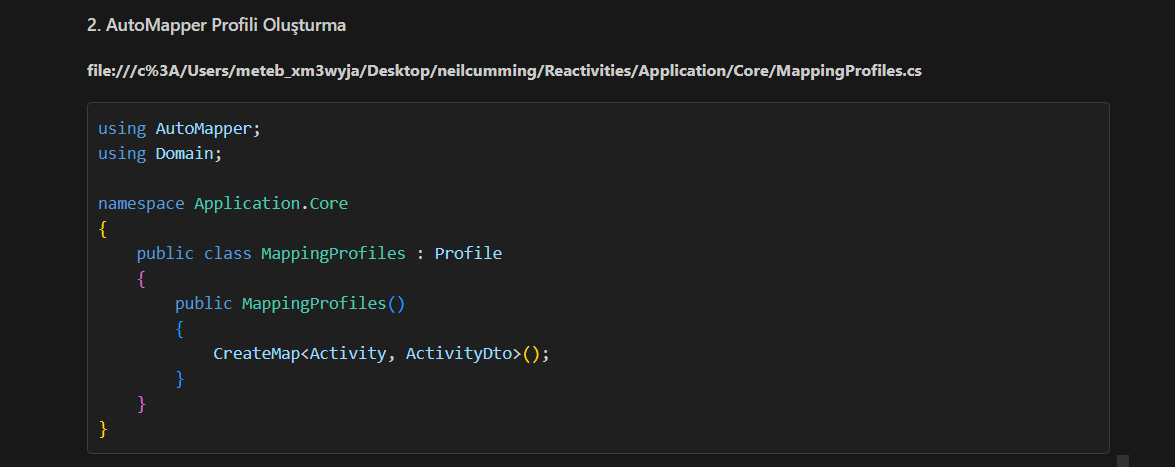
Commandde unit.value kullanilir queryde olmaz

**Komutlar (Commands)**: Sistem durumunu değiştiren işlemleri temsil eder. Bu işlemler genellikle veri ekleme, güncelleme veya silme gibi yan etkileri olan işlemlerdir. Komutlar, başarılı bir şekilde tamamlandıklarını belirtmek için genellikle [Unit](vscode-file://vscode-app/c:/Users/meteb_xm3wyja/AppData/Local/Programs/Microsoft VS Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.esm.html" \o ") türünde bir yanıt döndürürler. [Unit](vscode-file://vscode-app/c:/Users/meteb_xm3wyja/AppData/Local/Programs/Microsoft VS Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.esm.html" \o "), MediatR kütüphanesinde bir komutun başarılı bir şekilde tamamlandığını belirtmek için kullanılan bir türdür ve herhangi bir veri içermez.

**Sorgular (Queries)**: Sistemden veri okuma işlemlerini temsil eder. Bu işlemler, veri döndürmek için kullanılır ve genellikle belirli bir veri türünde yanıt döndürürler. Sorgular, veri döndürdükleri için [Unit](vscode-file://vscode-app/c:/Users/meteb_xm3wyja/AppData/Local/Programs/Microsoft VS Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.esm.html" \o ") türünde bir yanıt döndürmezler.

DTO





### **3. List Handler'ında DTO Kullanma**

#### file:///c%3A/Users/meteb\_xm3wyja/Desktop/neilcumming/Reactivities/Application/Activities/List.cs

using AutoMapper;

using AutoMapper.QueryableExtensions;

using Domain;

using MediatR;

using Microsoft.EntityFrameworkCore;

using Persistence;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

namespace Application.Activities

{

    public class List

    {

        public class Query : IRequest<List<ActivityDto>> { }

        public class Handler : IRequestHandler<Query, List<ActivityDto>>

        {

            private readonly DataContext \_context;

            private readonly IMapper \_mapper;

            public Handler(DataContext context, IMapper mapper)

            {

                \_context = context;

                \_mapper = mapper;

            }

            public async Task<List<ActivityDto>> Handle(Query request, CancellationToken cancellationToken)

            {

                return await \_context.Activities

                    .ProjectTo<ActivityDto>(\_mapper.ConfigurationProvider)

                    .ToListAsync(cancellationToken);

            }

        }

    }

}

1.adım

dotnet ef migrations add AddFullTextIndexToActivity

### **Adım 2: Migration'da Full-Text Index Eklemek**

Oluşturulan migration dosyasını açın ve Up metoduna full-text index eklemek için gerekli SQL komutlarını ekleyin.

#### Migrations/AddFullTextIndexToActivity.cs

using Microsoft.EntityFrameworkCore.Migrations;

namespace Persistence.Migrations

{

public partial class AddFullTextIndexToActivity : Migration

{

protected override void Up(MigrationBuilder migrationBuilder)

{

migrationBuilder.Sql(

sql: "CREATE FULLTEXT CATALOG ftCatalog AS DEFAULT;",

suppressTransaction: true);

migrationBuilder.Sql(

sql: "CREATE FULLTEXT INDEX ON Activities(Title, Description) KEY INDEX PK\_Activities;",

suppressTransaction: true);

}

protected override void Down(MigrationBuilder migrationBuilder)

{

migrationBuilder.Sql(

sql: "DROP FULLTEXT INDEX ON Activities;",

suppressTransaction: true);

migrationBuilder.Sql(

sql: "DROP FULLTEXT CATALOG ftCatalog;",

suppressTransaction: true);

}

}

}

Adım3

EF.Functions.FreeText veya EF.Functions.Contains fonksiyonlarını kullanarak full-text search sorgusu yazın.

using AutoMapper;

using AutoMapper.QueryableExtensions;

using MediatR;

using Microsoft.EntityFrameworkCore;

using Persistence;

using System.Collections.Generic;

using System.Threading;

using System.Threading.Tasks;

namespace Application.Activities

{

    public class Search

    {

        public class Query : IRequest<List<ActivityDto>>

        {

            public string SearchTerm { get; set; }

        }

        public class Handler : IRequestHandler<Query, List<ActivityDto>>

        {

            private readonly DataContext \_context;

            private readonly IMapper \_mapper;

            public Handler(DataContext context, IMapper mapper)

            {

                \_context = context;

                \_mapper = mapper;

            }

            public async Task<List<ActivityDto>> Handle(Query request, CancellationToken cancellationToken)

            {

                var activities = await \_context.Activities

                    .Where(a => EF.Functions.Contains(a.Title, request.SearchTerm) || EF.Functions.Contains(a.Description, request.SearchTerm))

                    .ProjectTo<ActivityDto>(\_mapper.ConfigurationProvider)

                    .ToListAsync(cancellationToken);

                return activities;

            }

        }

    }

}

### **Adım 4: Controller'da Full-Text Search Kullanım**

using Application.Activities;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace API.Controllers

{

[ApiController]

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

public class ActivitiesController : ControllerBase

{

private readonly IMediator \_mediator;

public ActivitiesController(IMediator mediator)

{

\_mediator = mediator;

}

[HttpGet("search")]

public async Task<ActionResult<List<ActivityDto>>> SearchActivities(string searchTerm)

{

return await \_mediator.Send(new Search.Query { SearchTerm = searchTerm });

}

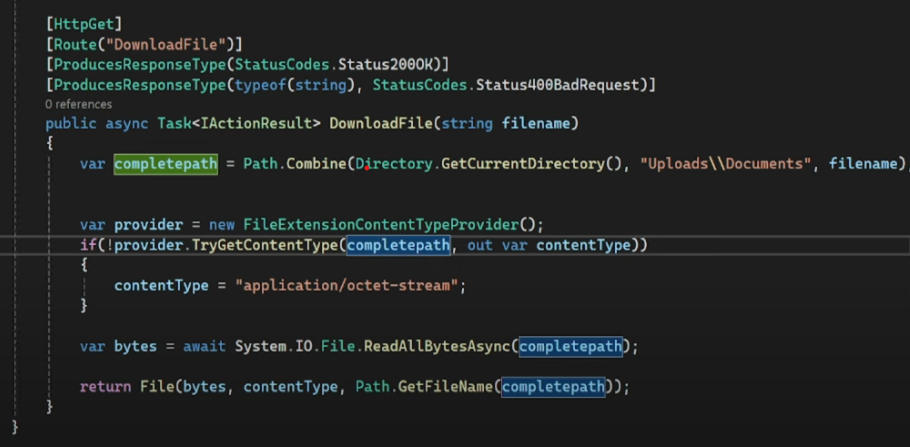
}

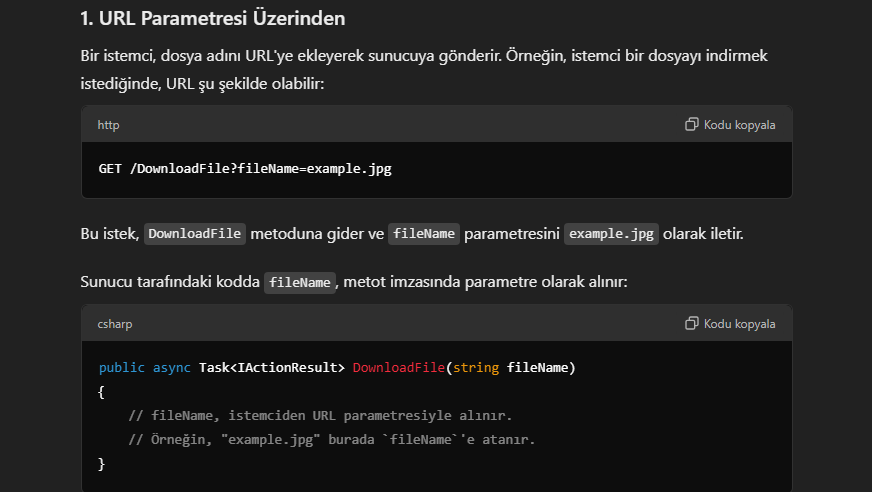
}

Dosya upload

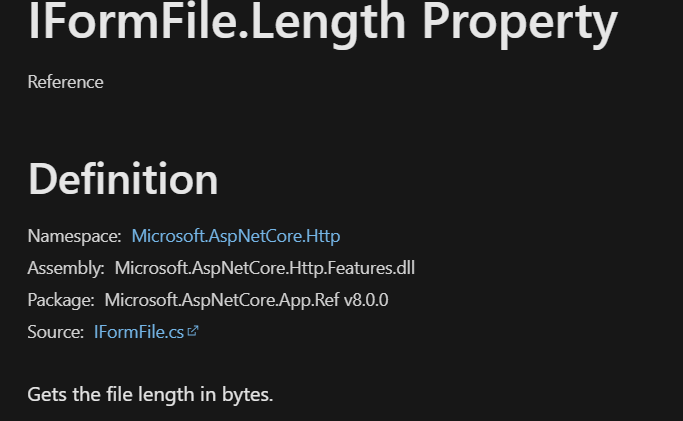
|  |
| --- |
| [HttpPost] |
|  | public async Task<IActionResult> FileUpload(IFormFile formFile) |
|  | { |
|  | if (formFile!=null) |
|  | { |
|  | var extent = Path.GetExtension(formFile.FileName); |
|  | var randomName = ($"{Guid.NewGuid()}{extent}"); |
|  | var path = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot\\img", randomName); |
|  |  |
|  | using(var stream=new FileStream(path, FileMode.Create)) |
|  | { |
|  | await formFile.CopyToAsync(stream); |
|  | } |
|  | } |
|  | return View(); |
|  | } |

Dosya Download





Bu baska file ona gore FileInfo.Length will return the length of file, in bytes (not size on disk), so this is what you are looking for, I think.



Springden bilgiler

### @RestController

@RestController, bir denetleyicinin (controller) RESTful bir denetleyici olduğunu belirtir.

* Bu notasyon, sınıfın tüm yöntemlerinde otomatik olarak @ResponseBody işlevselliğini ekler, böylece her metodun dönüş değeri doğrudan yanıt gövdesine yazılır.

Postta updatte @RequestBody kullanilir c# ta FromBody olabilir

Response Entity kullanilir.

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

private final EmployeeService employeeService;

public EmployeeController(EmployeeService employeeService) {

this.employeeService = employeeService;

}

// Create

@PostMapping

public ResponseEntity<Employee> addEmployee(@RequestBody Employee employee) {

Employee savedEmployee = employeeService.save(employee);

return new ResponseEntity<>(savedEmployee, HttpStatus.CREATED);

}

// Read all

@GetMapping

public ResponseEntity<List<Employee>> getAllEmployees() {

List<Employee> employees = employeeService.findAll();

return new ResponseEntity<>(employees, HttpStatus.OK);

}

// Read by ID

@GetMapping("/{id}")

public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id) {

return employeeService.findById(id)

.map(employee -> new ResponseEntity<>(employee, HttpStatus.OK))

.orElse(new ResponseEntity<>(HttpStatus.NOT\_FOUND));

}

// Update

@PutMapping("/{id}")

public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody Employee employee) {

Employee updatedEmployee = employeeService.update(id, employee);

return updatedEmployee != null

? new ResponseEntity<>(updatedEmployee, HttpStatus.OK)

: new ResponseEntity<>(HttpStatus.NOT\_FOUND);

}

// Delete

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {

return employeeService.delete(id)

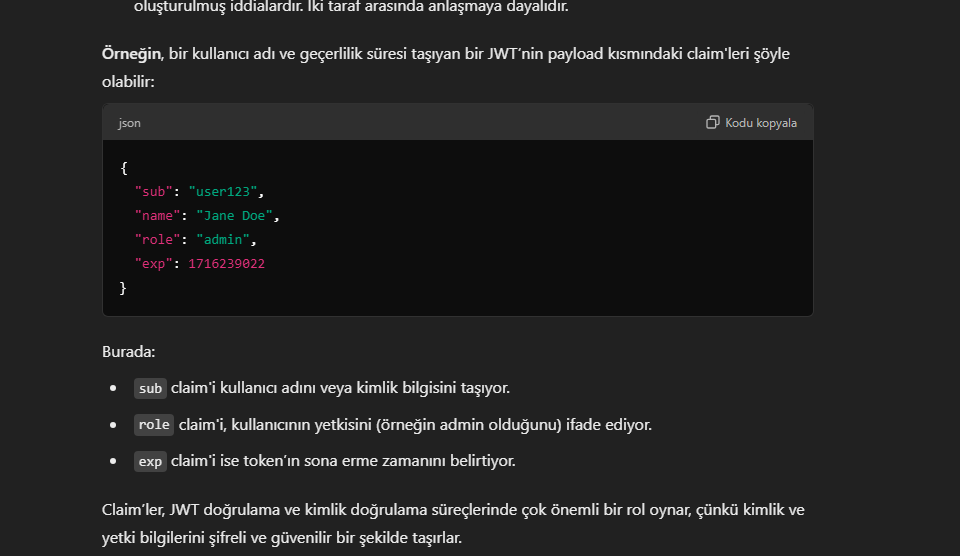
? new ResponseEntity<>(HttpStatus.NO\_CONTENT)

: new ResponseEntity<>(HttpStatus.NOT\_FOUND);

}

}

Springde Jwt mantigi



JWT'de, claim terimi, bir token içinde saklanan veri veya iddialar anlamına gelir. Claim'ler, token’ın taşıdığı bilgileri belirtir ve bu bilgiler, alıcı taraf için önemlidir. JWT, üç ana bölümden oluşur: header (başlık), payload (taşıyıcı veri), ve signature (imza). Claim’ler, token’ın payload kısmında saklanır.

UserDetails, kullanıcı kimliği ve yetkilendirmeleriyle ilgili bilgileri tutarak, güvenli bir kimlik doğrulama ve yetkilendirme süreci sağlar. Bu nedenle, JWT doğrulaması yapıldığında kullanıcının bilgilerini yüklemek için UserDetails'ı kullanmak önemlidir.

### 1. loadUserByUsername(String username)

* **Açıklama**: Verilen kullanıcı adını (username) alır ve bu kullanıcıya ait UserDetails nesnesini döner.
* **Kullanım**: Bu metod, genellikle bir kullanıcı giriş işlemi sırasında çağrılır. Kullanıcının kimliğini doğrulamak için, kullanıcı adıyla birlikte veritabanından kullanıcı bilgileri alınır.\

Bu durumda, eğer getAuthentication() null dönerse, uygulama kullanıcının kimlik bilgilerini yüklemeye ve doğrulamaya çalışacaktır. Bu, JWT'nin geçerli olup olmadığını kontrol etmek için önemlidir.

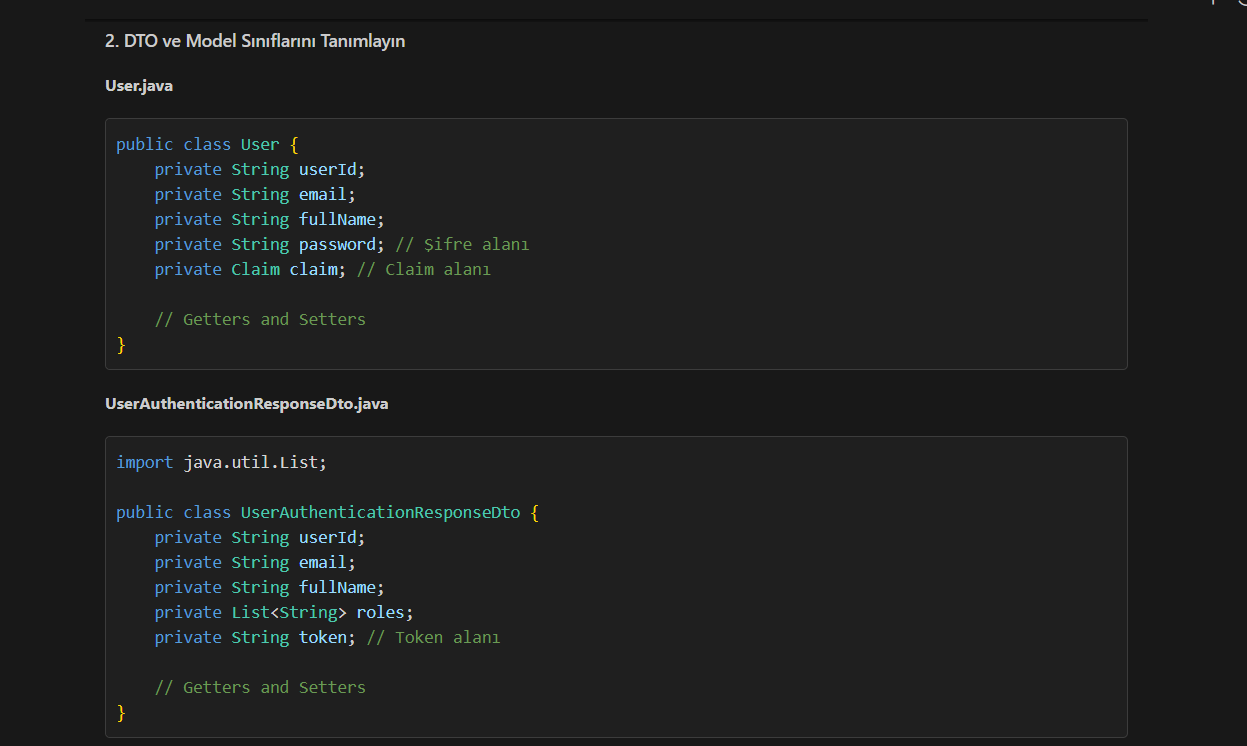
SecurityContextHolder.getContext().getAuthentication() == null:

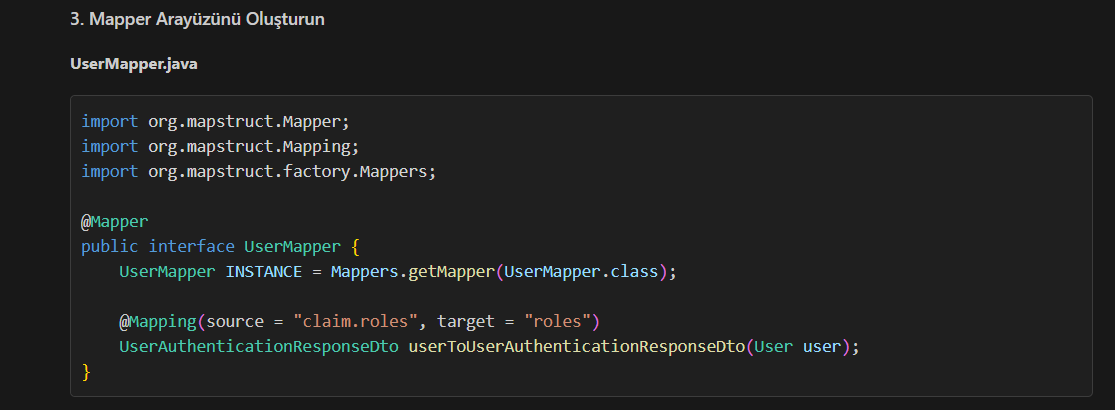
Bu koşul, mevcut güvenlik bağlamında (SecurityContext) herhangi bir kimlik doğrulama nesnesinin (Authentication) mevcut olup olmadığını kontrol eder. Eğer bu değer null ise, o anki istekte herhangi bir kullanıcı kimliği doğrulaması yapılmamış demektir.

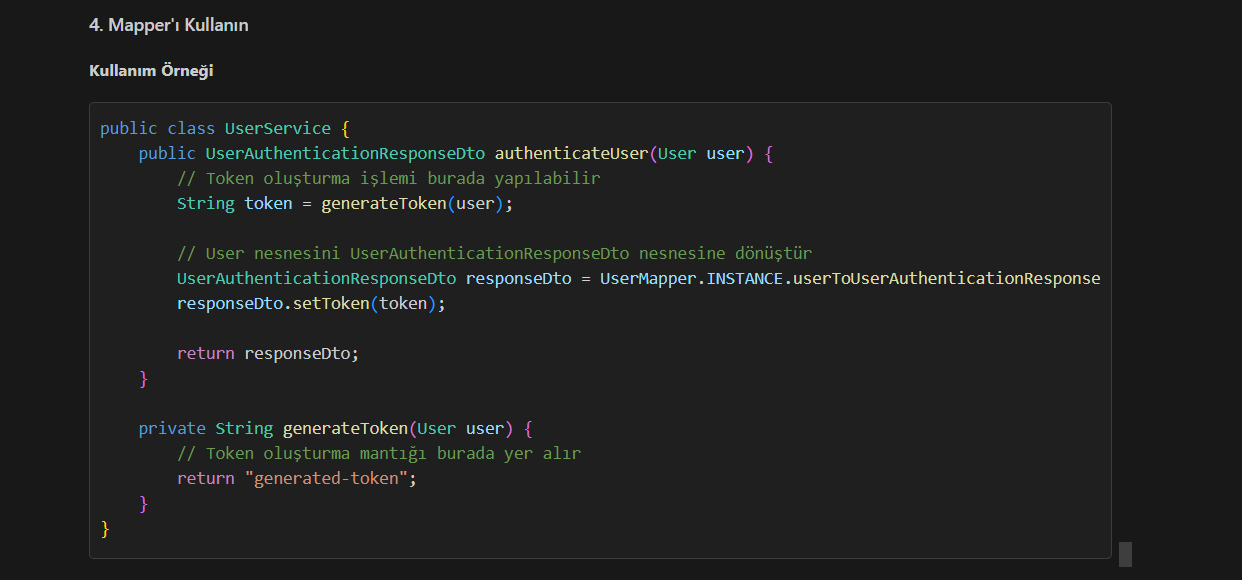
Bu, bir kullanıcının zaten kimlik doğrulamasını yapıp yapmadığını kontrol etmek içindir. Eğer kullanıcı daha önce doğrulanmışsa, tekrar doğrulama yapmaya gerek yoktur

\*\* Login yaparken authresponse yap(Auth requestYolla AuthResponse yap.)

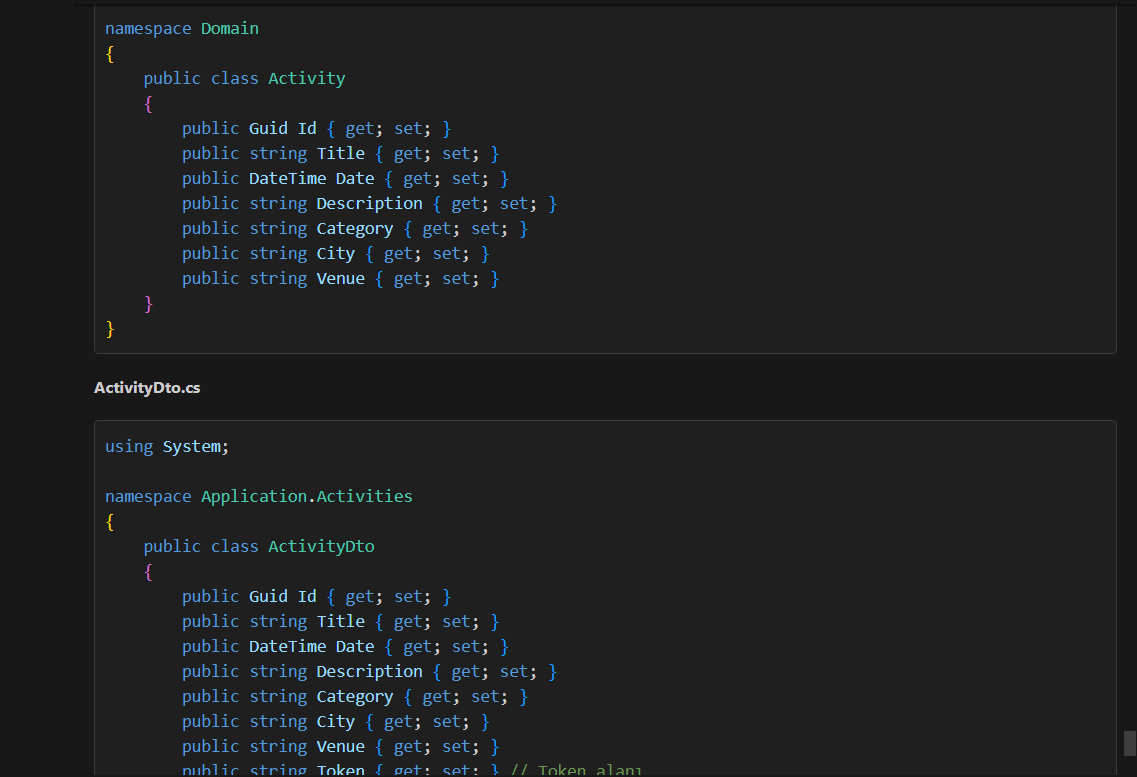
Eğer User sınıfında token alanı yoksa ve sadece dışarıdan geçen bir token varsa, yukarıdaki örnekteki @Mapping(target = "token", source = "token") kısmı, dışarıdan alınan token'ı UserAuthenticationResponseDto'ya aktaracaktır.

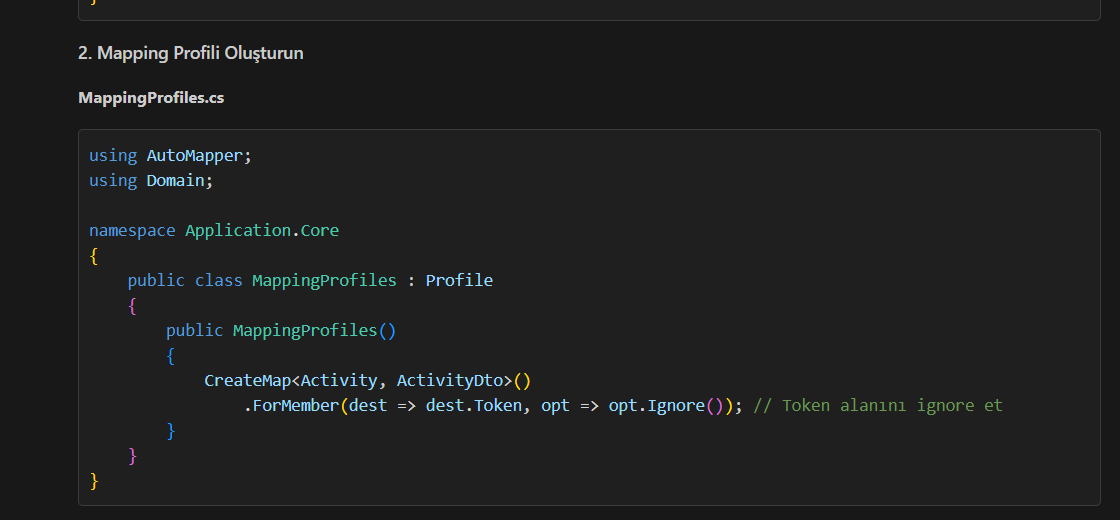


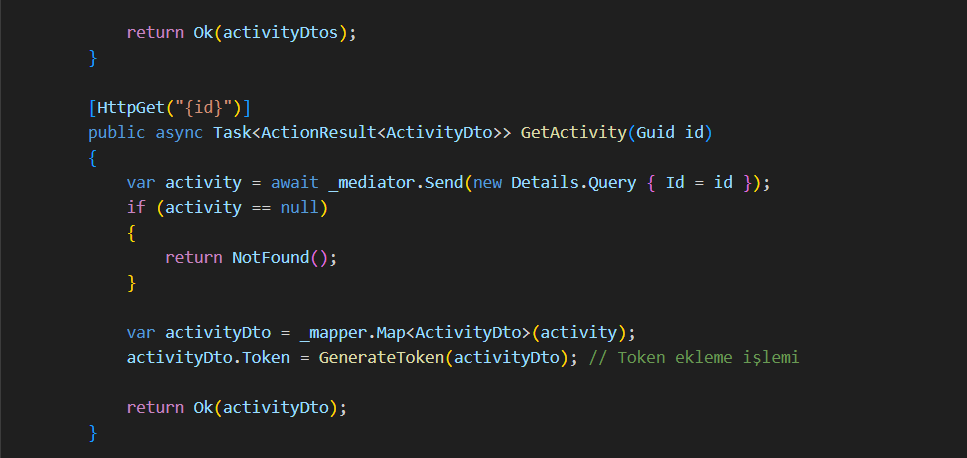




C# versiyonu

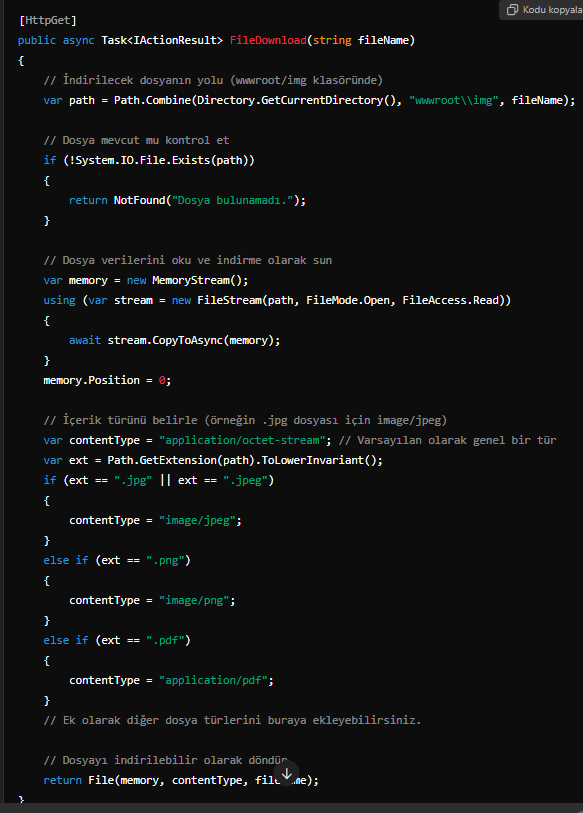






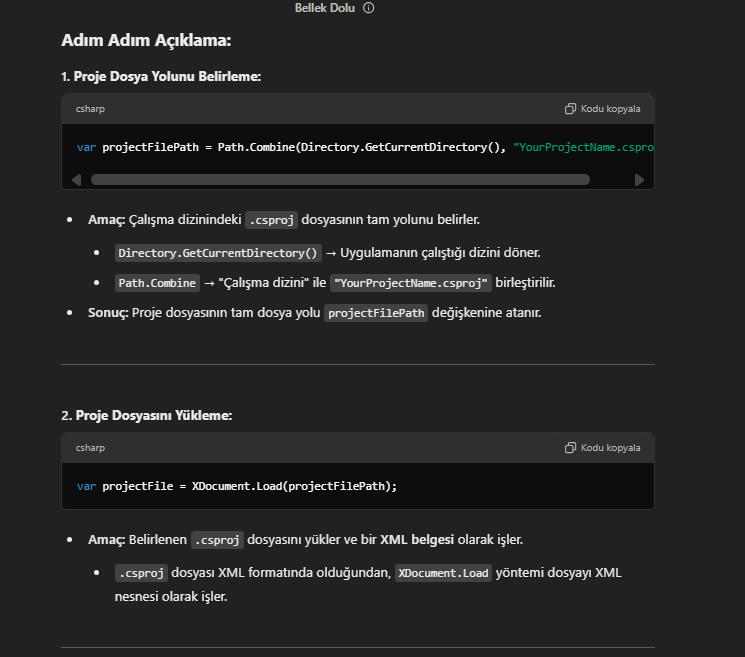
Son ustteki resim ornek generate token yanlis olabilir mesela.

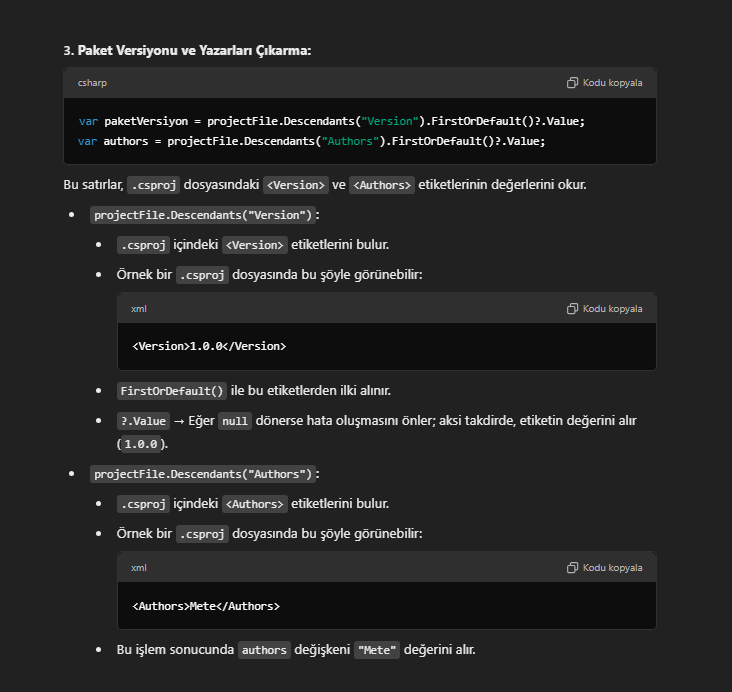
Dosya download2



· **Dosya istemciden alınır:** IFormFile kullanılarak.

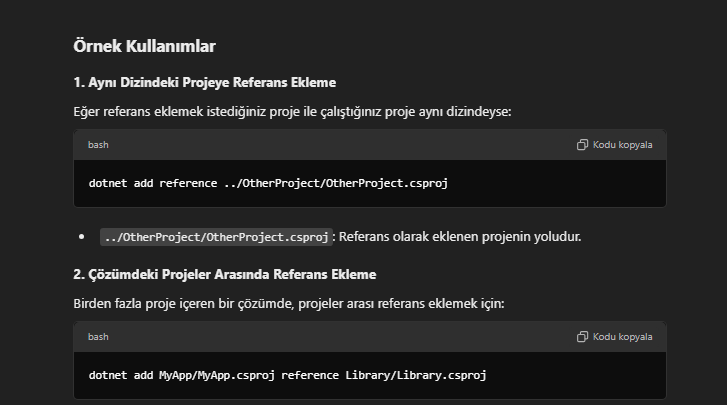
· **Sunucuda kaydedilir:** CopyToAsync ile belirlenen bir hedefe fiziksel olarak yazılır.





Unitvalue deger dondurmezse kullanilir (get haric)

Projemizde kisi manuel olarak indirecek ve dotnet reference yapacak



AMA PROJEDE .npgk yapacagiz o yuzden boyle olucak

