Restorasyon Otomasyonu SignalR

✅ **Konu Tanımı**  
→ *SignalR*, ASP.NET dünyasında **gerçek zamanlı (real-time)** web uygulamaları geliştirmek için kullanılan bir kütüphanedir.

🧠 **Mantığı**  
→ Normalde istemci (client), sunucudan veri almak için sürekli "istek" atmak zorundadır. *SignalR* sayesinde sunucu, bir değişiklik olduğunda anında istemciye haber gönderir. Bu da canlı bildirimler, sohbet uygulamaları, anlık güncellemeler gibi senaryoları kolaylaştırır.

📍 **Nerede ve Nasıl Kullanılır?**  
→ Canlı veri gerektiren uygulamalarda tercih edilir. Örneğin:

* Chat (sohbet) uygulamaları
* Anlık bildirim sistemleri
* Canlı skor takibi
* Dashboard üzerinde gerçek zamanlı veri güncelleme

Alternatif olarak *WebSocket*, *gRPC* veya *long polling* kullanılabilir; fakat *SignalR* bunların yönetimini otomatik yapar.

📌 **Günlük Hayattan Basit Bir Örnek**  
Bir futbol maçı izlediğini düşün. Skor değiştiğinde spikerin beklemeden anında sana sonucu bildirmesi gibi, *SignalR* da sunucudan istemciye anlık bilgi aktarır.

💻 **C# Kod Örneği**



🔁 **Alternatifler ve Kıyaslama**

* **WebSocket**: Daha düşük seviye, manuel yönetim gerekiyor.
* **gRPC streaming**: Performanslı ama daha teknik ayar gerektirir.
* **SignalR**: Kolay kurulum, farklı bağlantı yöntemlerini (WebSocket, Server-Sent Events, Long Polling) otomatik seçer.

📚 **Terim Sözlüğü**

| **İngilizce Terim** | **Türkçe Açıklama** |
| --- | --- |
| SignalR | Gerçek zamanlı iletişim kütüphanesi |
| Hub | İstemci ve sunucu arasındaki iletişim merkezi |
| Client | İstemci (tarayıcı, mobil uygulama vb.) |
| Server | Sunucu, veriyi gönderen taraf |
| WebSocket | Çift yönlü sürekli bağlantı protokolü |
| Long Polling | İstemcinin belirli aralıklarla sunucudan bilgi isteme yöntemi |

Mini bilgi

\_aboutService.TDelete(detectedValues!);//bu ! ile null değil null olursa hata fırlatır ama zaten silme yapacağımız için sorun yok

builder.Services.AddDbContext<SignalRContext>();

builder.Services.AddScoped<IAboutService, AboutManager>(); //ıaboutservice gördüğün yerde aboutmanagerı çağır

builder.Services.AddScoped<IAboutDal, EfAboutDal>(); //ıaboutdal gördüğün yerde efaboutdalı çağır

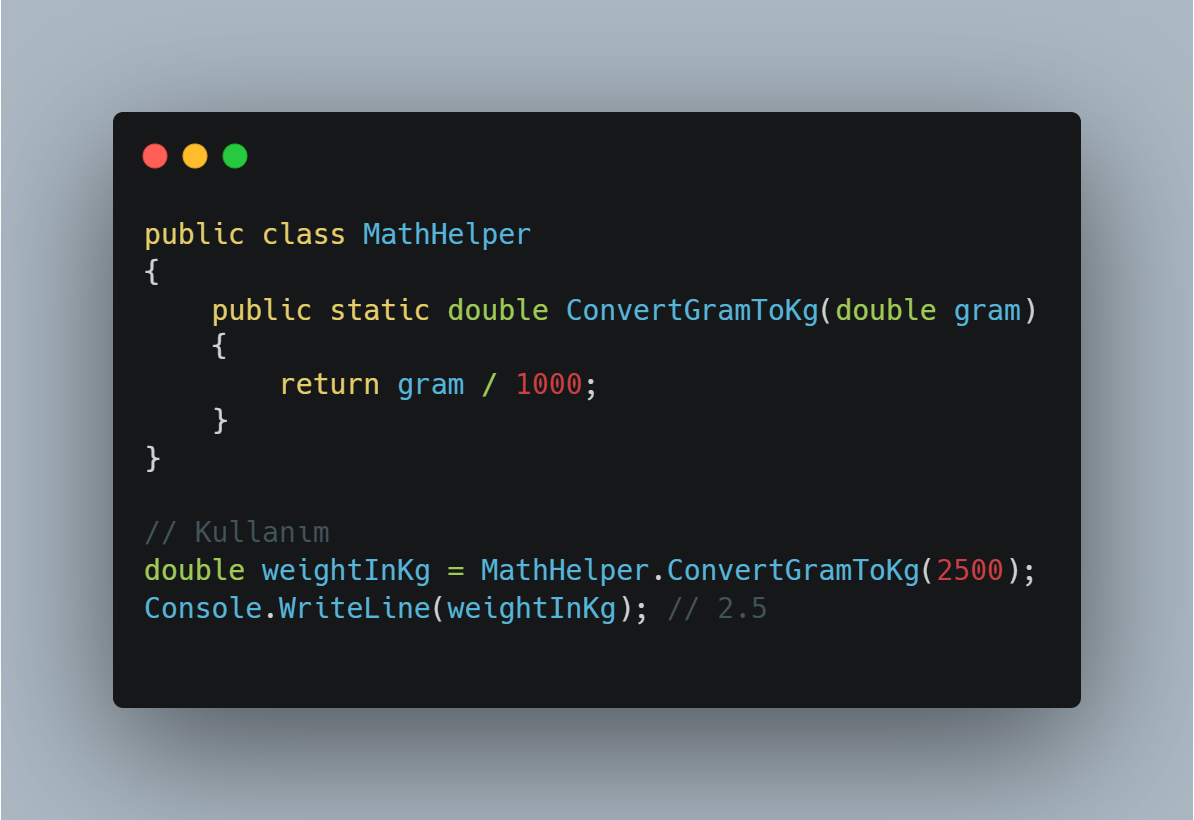
✅ **Konu Tanımı**  
→ *static method*, bir **class’a bağlı**, ancak **o class’tan oluşturulacak nesneye (instance) bağlı olmayan** metottur. Nesne oluşturmadan direkt class üzerinden çağrılır.

🧠 **Mantığı**  
→ Bazı işlemler, nesneye özgü veri tutmayı gerektirmez. Bu durumda her nesne için metodu tekrar tanımlamak yerine, tek bir *static method* tanımlayarak kaynak kullanımını ve kod sadeliğini artırır.

📍 **Nerede ve Nasıl Kullanılır?**  
→ Genel yardımcı işlemler (utility/helper), matematiksel hesaplamalar veya sabit verilere erişim gereken durumlarda tercih edilir.  
Alternatif olarak normal (instance) method kullanılır; fakat o zaman **nesne oluşturmak zorunludur**.

📌 **Günlük Hayattan Basit Bir Örnek**  
Bir mutfak tartısı düşün. Her tartıda “gramı kiloya çevir” fonksiyonu vardır. Tartının kendisine özel bir veri tutmaya gerek yok, tek bir yöntem yeter. İşte bu fonksiyon *static method* olarak tanımlanabilir.

💻 **C# Kod Örneği**

🔁 **Alternatifler ve Kıyaslama**

* **Instance method**: Nesneye özgü veri gerektiren işlemler için kullanılır.
* **Static method**: Nesne oluşturmadan direkt erişim sağlar, performans ve kullanım kolaylığı sunar.
* **Static class**: Sadece static method ve property barındırabilir, nesne oluşturulamaz.

📚 **Terim Sözlüğü**

| **İngilizce Terim** | **Türkçe Açıklama** |
| --- | --- |
| static | Nesneye değil class’a bağlı |
| method | İşlev, davranışı tanımlar |
| instance | Nesne örneği, class’tan oluşturulan obje |
| static class | Sadece static üye içerebilen class |

Program.cs de biriktirmek yerine

Bunu program.cs e yaz

builder.Services.ContainerDependencies();//böylelikle container oluyor

business layerda bir container klasörü aç ve extension adında bir class aç ve gerekeni yap

public static class Extension //api katmanından erişmek için static yapıyoruz

{

public static void ContainerDependencies(this IServiceCollection services)

{

services.AddScoped<IAboutService, AboutManager>(); //ıaboutservice gördüğün yerde aboutmanagerı çağır

services.AddScoped<IAboutDal, EfAboutDal>(); //ıaboutdal gördüğün yerde efaboutdalı çağır

}

}

Örnek bir api kodu yazılışı

using AutoMapper;

using Microsoft.AspNetCore.Http;

using Microsoft.AspNetCore.Mvc;

using Project.BusinessLayer.Abstract;

using Project.DtoLayer.AboutDto;

using Project.EntityLayer.Concrete;

namespace Project.Api.Controllers

{

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

[ApiController]

public class AboutController : ControllerBase

{

private readonly IAboutService \_aboutService;

private readonly IMapper \_mapper;

public AboutController(IAboutService aboutService, IMapper mapper)

{

\_aboutService = aboutService;

\_mapper = mapper;

}

[HttpGet]

public IActionResult AboutList()

{

var detectedValues = \_aboutService.TGetListAll();

return Ok(detectedValues);

}

[HttpPost]

public IActionResult AboutInsert(CreateAboutDto createAboutDto)

{

var detectedValues = \_mapper.Map<About>(createAboutDto);

\_aboutService.TInsert(detectedValues);

return Ok("Hakkımda Kısmı Başarılı Bir Şekilde Eklendi");

}

[HttpDelete]

public IActionResult AboutDelete(int id)

{

var detectedValues = \_aboutService.TGetByID(id);

\_aboutService.TDelete(detectedValues!);//bu ! ile null değil null olursa hata fırlatır ama zaten silme yapacağımız için sorun yok

return Ok("Hakkımda Kısmı Başarılı Bir Şekilde Silindi");

}

[HttpPut]

public IActionResult AboutUpdate(UpdateAboutDto updateAboutDto)

{

var detectedValues = \_mapper.Map<About>(updateAboutDto);

\_aboutService.TUpdate(detectedValues);

return Ok("Hakkımda Kısmı Başarılı Bir Şekilde Güncellendi");

}

[HttpGet("AboutGetByID")]

public IActionResult AboutGetByID(int id)

{

var detectedValues = \_aboutService.TGetByID(id);

return Ok(detectedValues);

}

}

}

Entitye Özgü method yazımı

Önce dal kısmı ardından business

İnterface kısmı

public interface IProductDal:IGenericDal<Product>

{

List<Product> GetProductsWithCategories();

}

Ve entityframework kısmı

public List<Product> GetProductsWithCategories()

{

var context = new SignalRContext();

return context.Products.Include(x=>x.Category).ToList();

}

Sonra business katmanına geç ve

Abstract kısmı service kısmı yani

public interface IProductService : IGenericService<Product>

{

List<Product> TGetProductsWithCategories();

}

Manager kısmı

public List<Product> TGetProductsWithCategories()

{

return \_productDal.GetProductsWithCategories();

}

Ardından dto kısmını oluşturuyoruz

public class ResultProductWithCategory

{

public int ProductID { get; set; }

public string? ProductName { get; set; }

public string? ProductDescription { get; set; }

public decimal ProductPrice { get; set; }

public string? ProductImageUrl { get; set; }

public bool ProductStatus { get; set; }

public string? CategoryCategoryName { get; set; }// işte bu kısmı sonradan ekledik çünkü include ile birbirne bağladık ve bu bilgilerin yanında bu da lazım oldu

}

AutoMapper’ın **Flattening** özelliğini kullanıyoruz.

AutoMapper şunu yapıyor:

Eğer DTO’da property adı **NavigationPropertyName.PropertyName** formatındaysa, entity içindeki navigation property’den otomatik eşleştirme yap.

Mappinge ekledik

public class ProductMapping:Profile

{

public ProductMapping()

{

CreateMap<Product, ResultProductDto>().ReverseMap();

CreateMap<Product, GetProductDto>().ReverseMap();

CreateMap<Product, CreateProductDto>().ReverseMap();

CreateMap<Product, UpdateProductDto>().ReverseMap();

CreateMap<Product, ResultProductWithCategory>().ReverseMap();//mapping e ekledik

}

}

Ve son olarak api kısmı

[HttpGet("ProductListWithCategory")]

public IActionResult ProductListWithCategory()

{

var detectedValues = \_productService.TGetProductsWithCategories();

return Ok(\_mapper.Map<List<ResultProductWithCategory>>(detectedValues));

}

APİ TÜKETME

**🟢 Konu Tanımı**  
→ JSON (JavaScript Object Notation), veri değişimi için kullanılan hafif ve okunabilir bir veri formatıdır. İnsan tarafından anlaşılır, programlar tarafından kolayca işlenebilir.

**🧠 Mantığı**  
→ Farklı uygulamalar veya sistemler arasında veri paylaşımını standart ve anlaşılır bir şekilde yapmak için vardır. XML gibi ağır yapılar yerine daha basit ve hızlı bir çözüm sunar.

**📍 Nerede ve Nasıl Kullanılır?**  
→ Web servislerinde (API), konfigürasyon dosyalarında ve veri depolamada tercih edilir. Alternatifleri: XML, YAML. JSON daha sade ve performanslıdır.

**📌 Günlük Hayattan Basit Bir Örnek**  
→ Marketten alışveriş listesi paylaşmak gibi düşün:

{

"isim": "Elma",

"miktar": 5,

"birim": "kg"

}

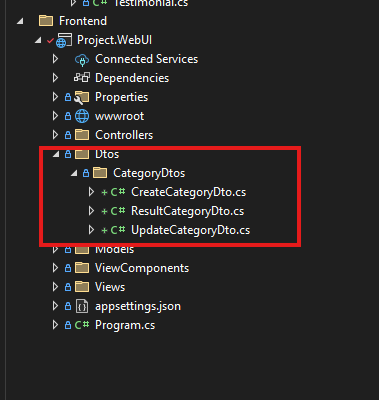
Bu listeyi başka biri kolayca okuyup kullanabilir.

**💻 C# Kod Örneği**



**🔁 Alternatifler ve Kıyaslama**

* **XML:** Daha karmaşık, daha ağır, fakat bazı eski sistemlerde tercih edilir.
* **YAML:** İnsan okunabilirliği yüksek, ama C#’ta JSON kadar desteklenmez.



Ek olarak buradaki dto katmanı ise bu sırf UI katmanı için dto diğer api içindi bu da sırf UI için bunu unutma

builder.Services.AddHttpClient();

ekle program cs

using Microsoft.AspNetCore.Mvc;

using Newtonsoft.Json;

using Project.WebUI.Dtos.CategoryDtos;

using System.Net.Http;

using System.Text;

using System.Threading.Tasks;

namespace Project.WebUI.Controllers

{

public class CategoryController : Controller

{

private readonly IHttpClientFactory \_httpClientFactory;

public CategoryController(IHttpClientFactory httpClientFactory)

{

\_httpClientFactory = httpClientFactory;

}

// apiyi çağıran oradaki şeyleri kullanmamıza yarayan yöntem budur

public async Task<IActionResult> Index()

{

var client = \_httpClientFactory.CreateClient(); //burada bir bağlantı oluşturtuyoruz

var responseMessage = await client.GetAsync("https://localhost:7240/api/Category");//ulaşacağımız yani buradaki konuma gitsin oradan çeksin veriyi

if (responseMessage.IsSuccessStatusCode) //yani verim tamam 200 döndü her şey iyiyse

{

var jsonData = await responseMessage.Content.ReadAsStringAsync();//jsondan gelen içeriği string formatında oku

var values = JsonConvert.DeserializeObject<List<ResultCategoryDto>>(jsonData);

//listlerken deserelize ekleme güncelleme serialize neden deserelize ile bu json verisini okunabilir hale getiriyoruz nereden jsondan gelen veriyi

return View(values); //gelen veriyi düzledik

}

return View();

}

[HttpGet]

public IActionResult CategoryCreate()

{

return View();

}

[HttpPost]

public async Task<IActionResult> CategoryCreate(CreateCategoryDto createCategoryDto)

{

createCategoryDto.CategoryStatus = true;

var client = \_httpClientFactory.CreateClient(); //burada bir bağlantı oluşturtuyoruz

var jsonData = JsonConvert.SerializeObject(createCategoryDto);//jsona dönüştür

StringContent stringContent = new StringContent(jsonData, Encoding.UTF8, "application/json");

//jsondatadan gelen veri encodint utf 8 yap ve json formatında ver

var responseMessage = await client.PostAsync("https://localhost:7240/api/Category", stringContent);//yine ama bu sefer get yerine post yapıp veriyi gönderiyoruz

if (responseMessage.IsSuccessStatusCode)//yani verim tamam 200 döndü her şey iyiyse

{

return RedirectToAction("Index");//indexe beni at

}

return View();

}

public async Task<IActionResult> CategoryDelete(int id)

{

var client = \_httpClientFactory.CreateClient();

await client.DeleteAsync($"https://localhost:7240/api/Category/{id}");

return RedirectToAction("Index"); // artık view aramıyor

}

[HttpGet]

public async Task<IActionResult> CategoryUpdate(int id)

{

//updateCategoryDto.CategoryStatus = true;

var client = \_httpClientFactory.CreateClient(); //burada bir bağlantı oluşturtuyoruz

var responseMessage = await client.GetAsync($"https://localhost:7240/api/Category/{id}");//yine ama bu sefer get yerine post yapıp veriyi gönderiyoruz

if (responseMessage.IsSuccessStatusCode)//yani verim tamam 200 döndü her şey iyiyse

{

var jsonData = await responseMessage.Content.ReadAsStringAsync();

var values = JsonConvert.DeserializeObject<UpdateCategoryDto>(jsonData);

return View(values);

}

return View();

}

[HttpPost]

public async Task<IActionResult> CategoryUpdate(UpdateCategoryDto updateCategoryDto)

{

//createCategoryDto.CategoryStatus = true;

var client = \_httpClientFactory.CreateClient(); //burada bir bağlantı oluşturtuyoruz

var jsonData = JsonConvert.SerializeObject(updateCategoryDto);//jsona dönüştür

StringContent stringContent = new StringContent(jsonData, Encoding.UTF8, "application/json");

//jsondatadan gelen veri encodint utf 8 yap ve json formatında ver

var responseMessage = await client.PutAsync("https://localhost:7240/api/Category", stringContent);//yine ama bu sefer get yerine put yapıp veriyi gönderiyoruz

if (responseMessage.IsSuccessStatusCode)//yani verim tamam 200 döndü her şey iyiyse

{

return RedirectToAction("Index");//indexe beni at

}

return View();

}

}

}

Örnek ekle sil güncelle ve listele

using Microsoft.AspNetCore.Mvc;

using Newtonsoft.Json;

using Project.WebUI.Dtos.ProductDtos;

using System.Text;

using System.Threading.Tasks;

namespace Project.WebUI.Controllers

{

public class ProductController : Controller

{

private readonly IHttpClientFactory \_httpClientFactory;

public ProductController(IHttpClientFactory httpClientFactory)

{

\_httpClientFactory = httpClientFactory;

}

public async Task<IActionResult> Index()

{

var client = \_httpClientFactory.CreateClient(); //bunu her zaman açmak zorundasın

var responseMessage = await client.GetAsync("https://localhost:7240/api/Product"); //ilk adım veriyi çek

if (responseMessage.IsSuccessStatusCode) // eğer veri çekildiyse

{

var jsonData = await responseMessage.Content.ReadAsStringAsync(); //veriyi oku string formatta

var values = JsonConvert.DeserializeObject<List<ResultProductDto>>(jsonData); //veriyi çevir

return View(values); //veriyi ver

}

return View();

}

[HttpGet]

public IActionResult ProductCreate()

{

return View();

}

[HttpGet]

public async Task<IActionResult> ProductCreate(CreateProductDto createProductDto)

{

createProductDto.ProductStatus = true; //default olarak true döndürmek için

var client = \_httpClientFactory.CreateClient(); //klasik açıyotuz

var jsonData = JsonConvert.SerializeObject(createProductDto); //verimizi önce json formata döndürüyoruz

StringContent content = new StringContent(jsonData, Encoding.UTF8, "application/json"); //sonra nasıl göndereceğimize söylüyoruz json datadan gfelen veri encoding olarak utf8 ve json veri diyoruz

var responseMessage = await client.PostAsync("https://localhost:7240/api/Category", content); //sonra veriyi gönderiyoruz

if (responseMessage.IsSuccessStatusCode)

{

return RedirectToAction("Index");//başarılıysa indxe dön

}

return View();

}

public async Task<IActionResult> ProductDelete(int id)

{

var client = \_httpClientFactory.CreateClient(); //klasik kapıyı aç

await client.DeleteAsync($"https://localhost:7240/api/Category{id}"); //veriyi bul ve sil

return RedirectToAction("Index");

}

public async Task<IActionResult> ProductUpdate(int id)

{

var client = \_httpClientFactory.CreateClient();

var responseMessage = await client.GetAsync($"https://localhost:7240/api/Category{id}"); // veriyi getir

if (responseMessage.IsSuccessStatusCode) // veri başarılı gelirse

{

var jsonData = await responseMessage.Content.ReadAsStringAsync(); // veriyi string formatında oku

var values = JsonConvert.DeserializeObject<UpdateProductDto>(jsonData); // veriyi normal formata getr

return View(values); //bana sun

}

return View();

}

public async Task<IActionResult> ProductUpdate(UpdateProductDto updateProductDto)

{

var client = \_httpClientFactory.CreateClient();

var jsonData = JsonConvert.SerializeObject(updateProductDto);//giden veriyi bu formata sok

StringContent stringContent = new StringContent(jsonData, Encoding.UTF8, "application/json");//nasıl gideceğini söyle

var responseMessage = await client.PutAsync("https://localhost:7240/api/Category", stringContent); //ardından gönder

if (responseMessage.IsSuccessStatusCode)

{

return RedirectToAction("Index"); //başarılı olursa gönder

}

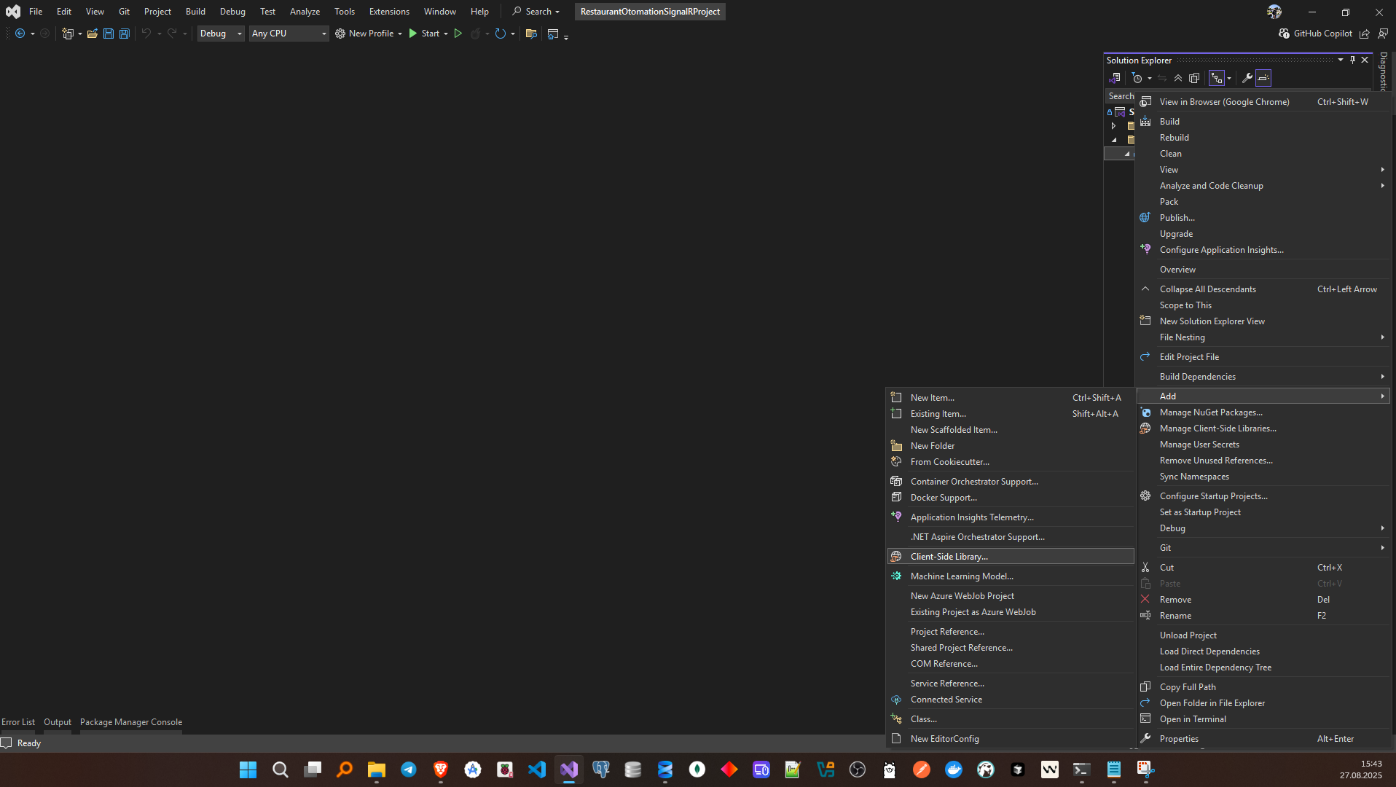
return View();

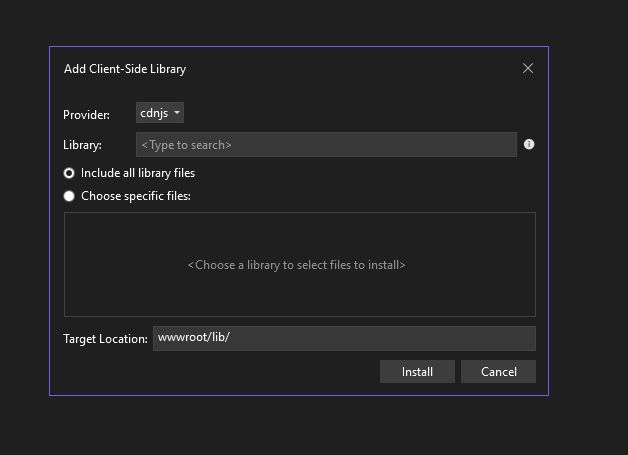
}

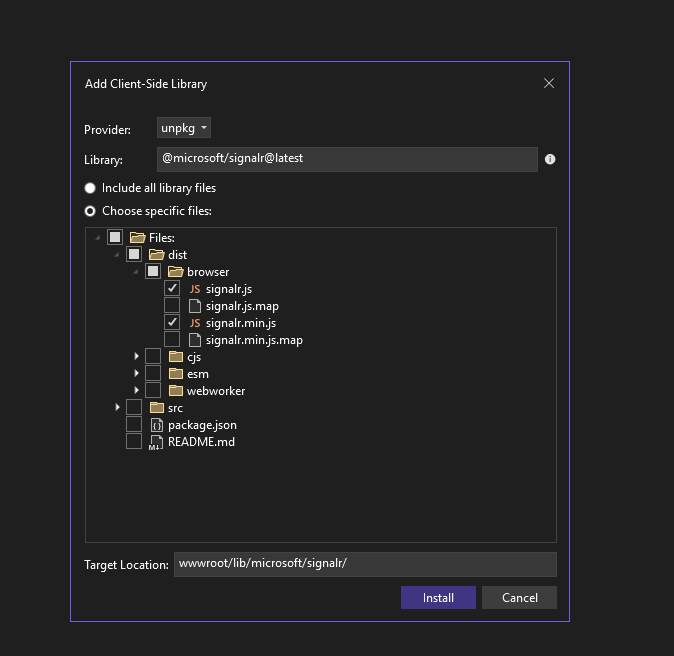
}

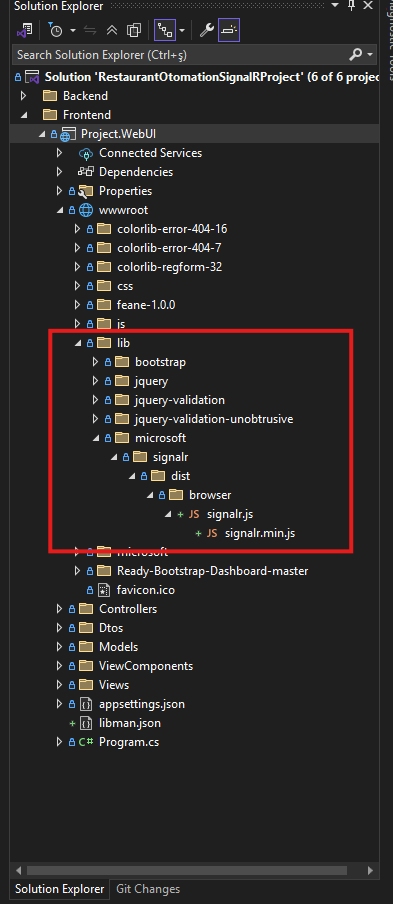
}

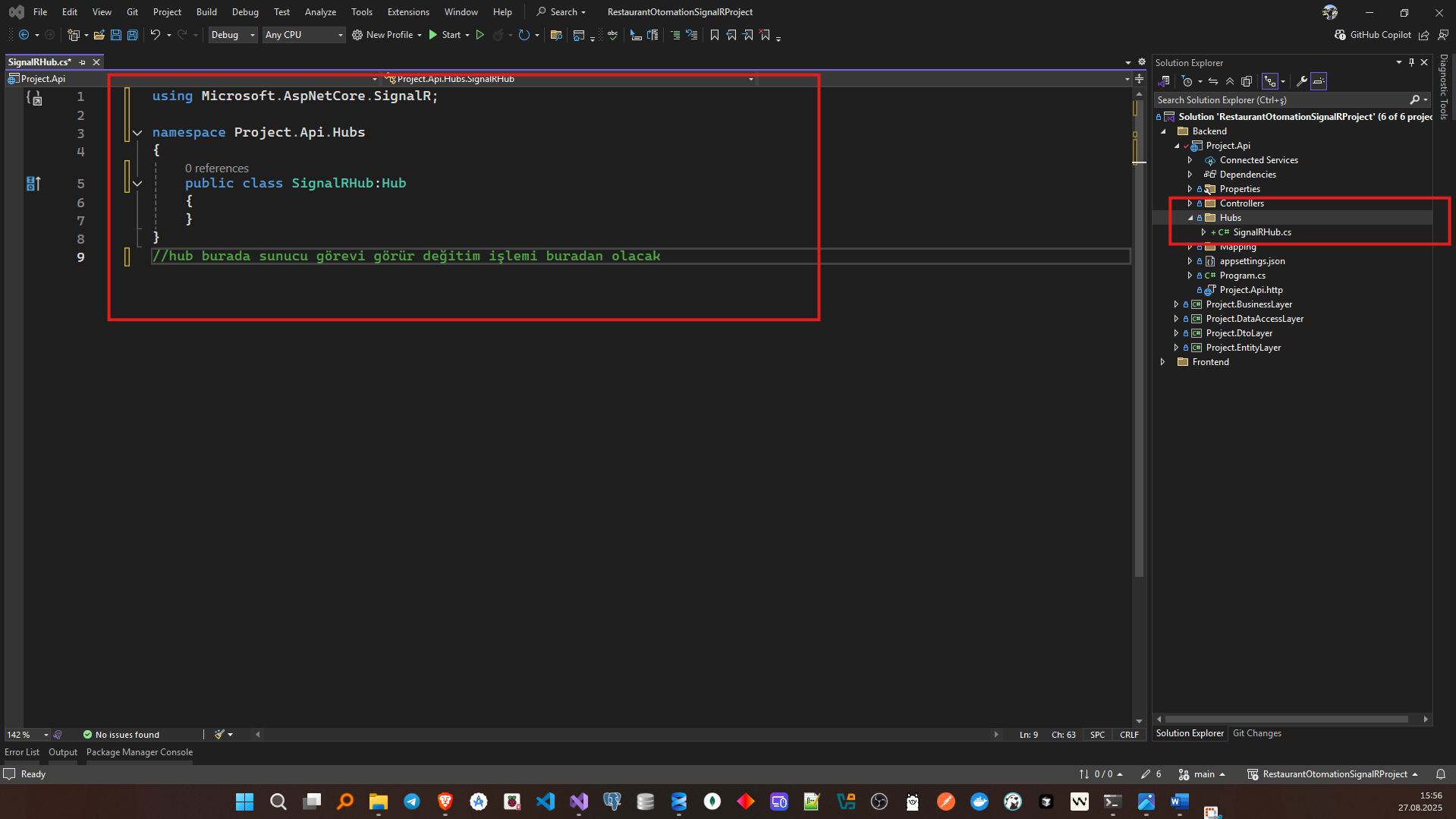
SignalR kurulumu,











Api da hub klasörü aç

Sonra bu işlemleri yap

Cors politikasına geç peki nedir bu cors politikası

✅ **Konu Tanımı**  
→ **CORS Policy**, bir web uygulamasının başka bir domain (kaynak) üzerindeki verilere erişimini kontrol eden güvenlik mekanizmasıdır.

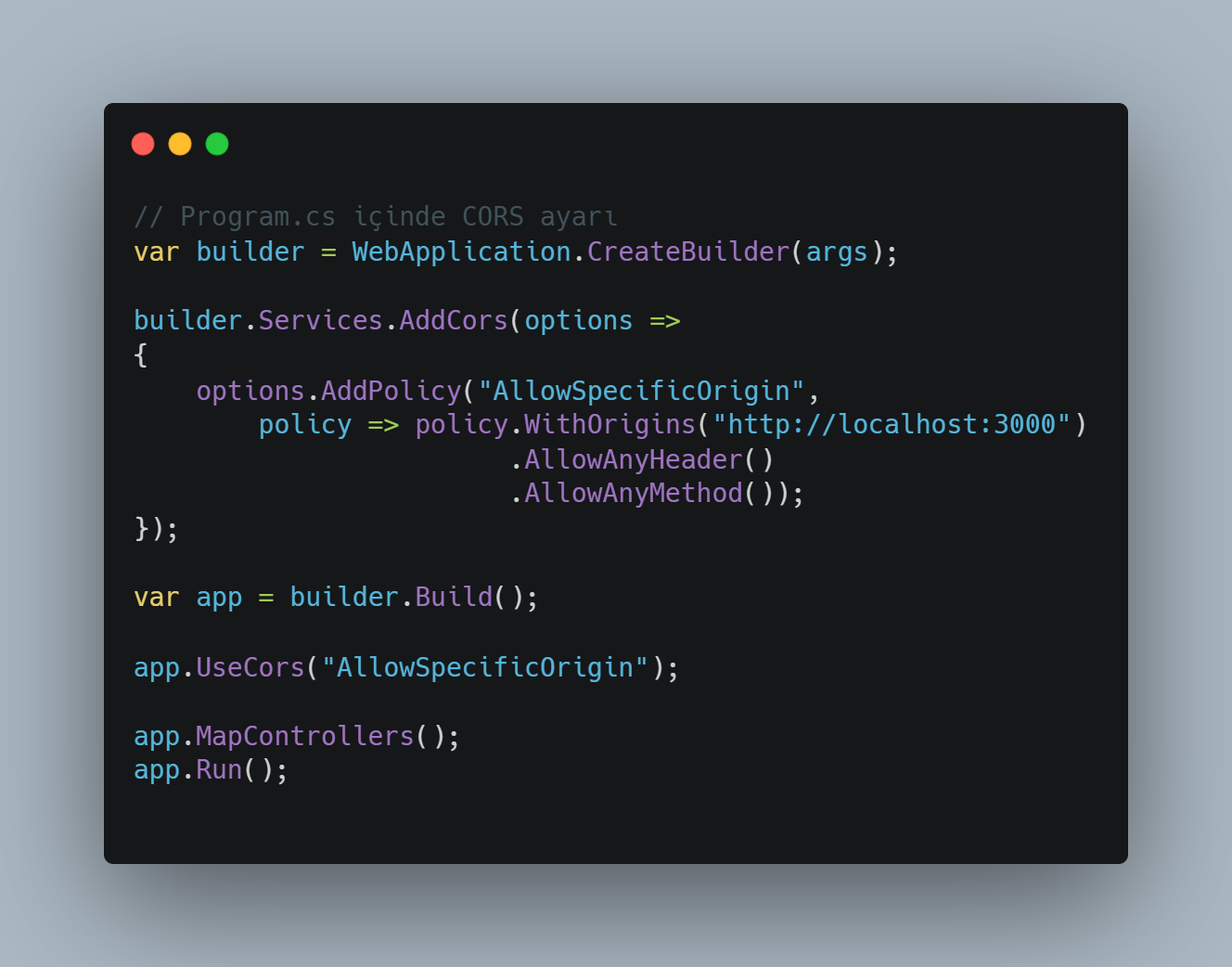
🧠 **Mantığı**  
→ Tarayıcılar, güvenlik için **aynı origin (kaynak)** dışındaki isteklere kısıtlama getirir.  
CORS bu kısıtlamayı esnetmek için vardır: Sunucu, hangi domain’lere erişim izni verdiğini belirtir.

📍 **Nerede ve Nasıl Kullanılır?**  
→ Örneğin:

* Frontend (http://localhost:3000) ile Backend (http://localhost:5000) farklı portlarda çalışıyorsa.
* API başka bir domain’den çağrılıyorsa.  
  Alternatif: Proxy kullanmak veya API Gateway ile istek yönlendirmek.

📌 **Günlük Hayattan Basit Bir Örnek**  
→ Bir binaya (server) girmek isteyen yabancı (farklı domain) için kapıda güvenlik görevlisi (CORS Policy) var. Güvenlik görevlisi sadece **izin verilen misafirlere** giriş hakkı verir.

💻 **C# Kod Örneği** (ASP.NET Core’da)



🔁 **Alternatifler ve Kıyaslama**

* **CORS olmadan**: Tarayıcı güvenlik nedeniyle isteği engeller.
* **Proxy kullanımı**: İstekler backend’e yönlendirilerek CORS ihtiyacı azaltılabilir.
* **API Gateway**: Trafiği merkezi noktadan yönetir, CORS ayarlarını daha kolay yapar.

📚 **Terim Sözlüğü**

| **İngilizce Terim** | **Türkçe Açıklama** |
| --- | --- |
| CORS (Cross-Origin Resource Sharing) | Farklı domain’ler arası kaynak paylaşımı |
| origin | Kaynak (protocol + domain + port kombinasyonu) |
| policy | Politika, kurallar bütünü |
| header | HTTP isteğinde/cevabında ek bilgi alanı |
| proxy | İstekleri yönlendiren ara sunucu |
| API Gateway | API trafiğini merkezi yönetim noktası |

using Project.Api.Hubs;

using Project.BusinessLayer.Abstract;

using Project.BusinessLayer.Concrete;

using Project.BusinessLayer.Container;

using Project.DataAccessLayer.Abstract;

using Project.DataAccessLayer.Concrete;

using Project.DataAccessLayer.EntityFramework;

using System.Reflection;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<SignalRContext>();

builder.Services.AddAutoMapper(Assembly.GetExecutingAssembly());

builder.Services.ContainerDependencies();//böylelikle container oluyor

//cors politikası ve signal r kütüphanesi

builder.Services.AddCors(opt =>

{

opt.AddPolicy("CorsPolicy", builder =>

{

builder.AllowAnyHeader().AllowAnyMethod().SetIsOriginAllowed((host) => true).AllowCredentials();

});

});

builder.Services.AddSignalR();

// Add services to the container.

builder.Services.AddControllers();

// Learn more about configuring OpenAPI at https://aka.ms/aspnet/openapi

builder.Services.AddOpenApi();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.MapOpenApi();

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseCors("CorsPolicy");//corsu buradan çağırdık

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.MapHub<SignalRHub>("/signalrhub");//burası bir yere istek gönderdiğimiz yer category/ındex yerine signalrhub gönder gibi

app.Run();

Örnek

SignalRContext context = new SignalRContext();

public async Task SendCategoryCount()

{

var values = context.Categories.Count();

await Clients.All.SendAsync("ReceiveCategoryCount", values);

}

@{

ViewData["Title"] = "Index2";

Layout = "~/Views/AdminLayout/Index.cshtml";

}

<**script** src="~/lib/jquery/dist/jquery.min.js"></**script**>

<**script** src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></**script**>

<**script** **src**="~/js/site.js" **asp-append-version**="true"></**script**>

<**script** src="~/microsoft/signalr/dist/browser/signalr.min.js"></**script**>

<div class="content">

<div class="container-fluid">

<div class="row">

<div class="col-md-12">

<div class="card">

<**script** **type**="text/javascript">

$(document).ready(() => {

var connection = new signalR.HubConnectionBuilder().withUrl("https://localhost:7240/SignalRHub").build();

$("#connstatus").text(connection.state);

connection.start().then(() => {

$("#connstatus").text(connection.state);

setInterval(() => {

connection.invoke("SendCategoryCount");

}, 1000);

}).catch((err) => { console.log(err) });

connection.on("ReceiveCategoryCount", (value) => {

$("#categorycount").text(value);

});

});

</**script**>

</div>

</div>

</div>

</div>

</div>

<div class="row">

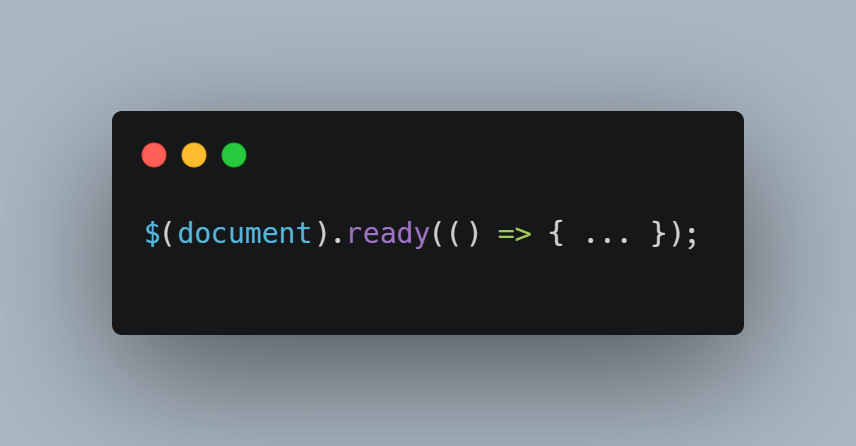
<div class="col-md-8 offset-2">

<div class="alert alert-success">Bağlantının Durumu:<strong id="connstatus"></strong></div>

<div class="alert alert-info">Kategori Sayısı:<strong id="categorycount"></strong></div>

</div>

</div>



 Bu, **jQuery** fonksiyonu.

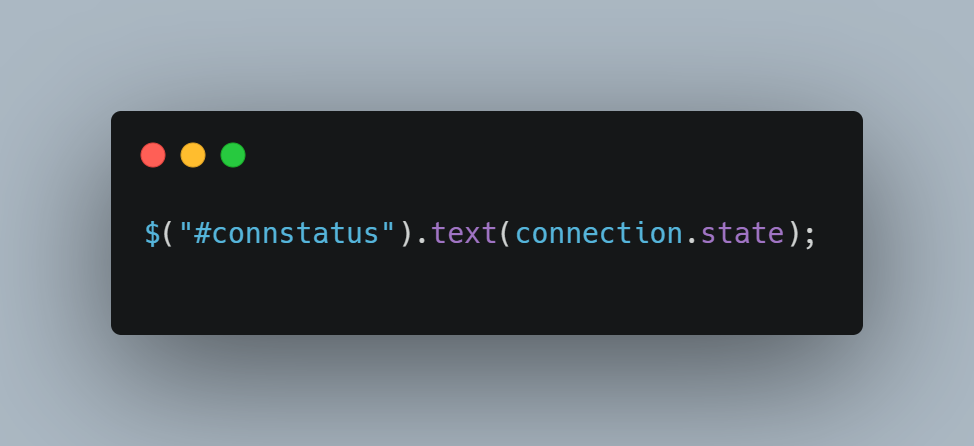
 Sayfa tamamen yüklendikten sonra içindeki kodun çalışmasını sağlar.

 Yani HTML ve DOM elementleri hazır olmadan kod çalışmaz, böylece #connstatus veya #categorycount bulunamaz hatası almazsın.



* **signalR.HubConnectionBuilder()** → SignalR hub bağlantısı oluşturmak için kullanılan builder objesi.
* **withUrl("https://localhost:7240/SignalRHub")** → Server tarafında tanımladığın Hub’a bağlanacak URL.
* **build()** → Builder’ı tamamlar ve connection objesini döndürür.

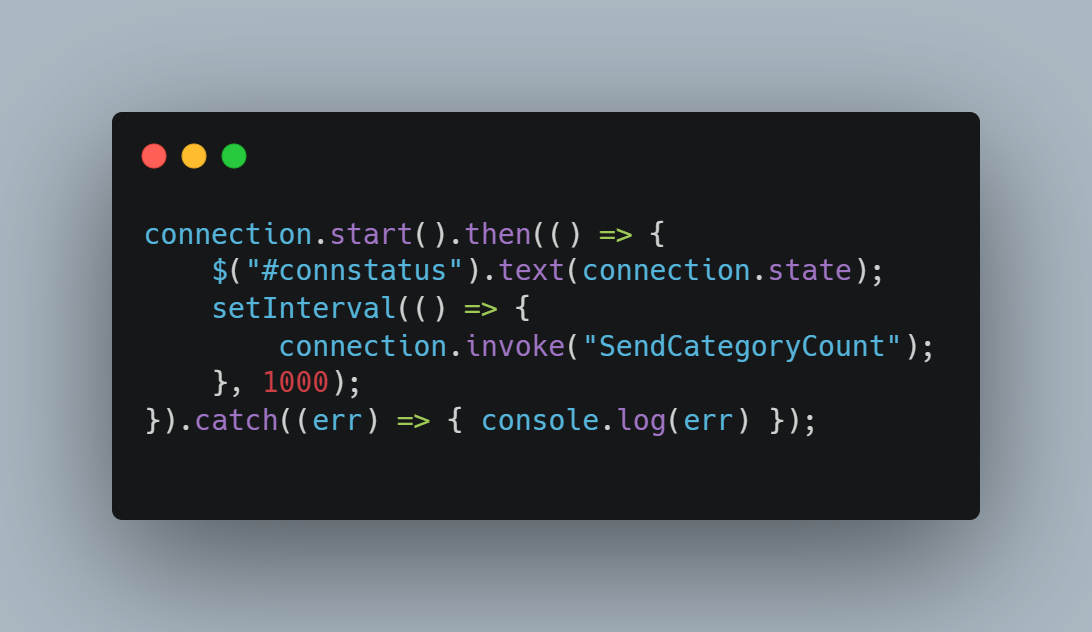
Bu objeyi kullanarak server’a veri gönderir veya server’dan veri alırsın.



 connection.state → SignalR bağlantısının durumunu gösterir.

 Başlangıçta Disconnected olur.

 #connstatus → HTML sayfanda bir element (örn: <span id="connstatus"></span>) olmalı.

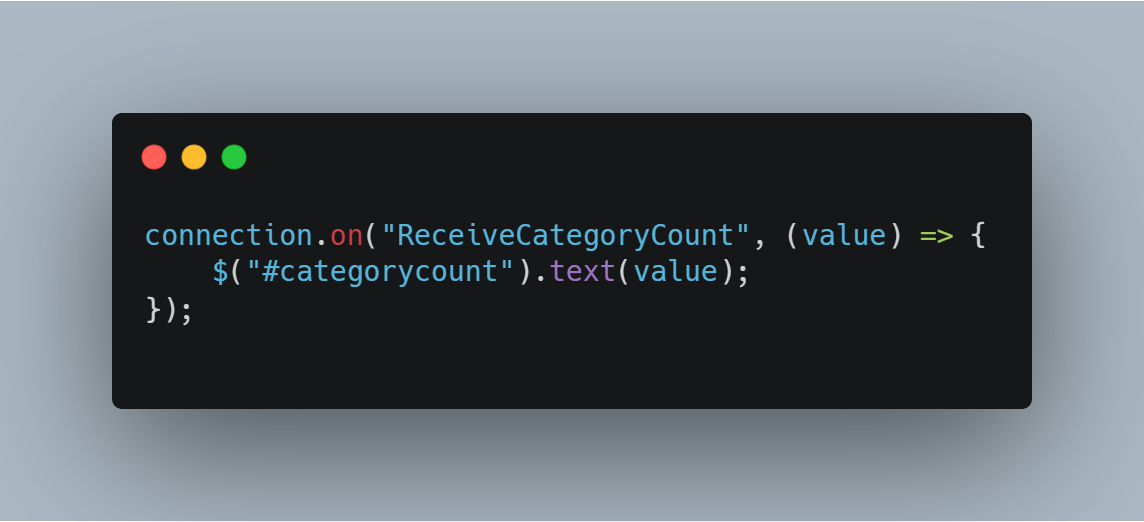


 **connection.start()** → SignalR server’a bağlanmayı başlatır.

 **.then()** → Bağlantı başarılı olursa yapılacak işlemler.

* Durumu günceller (Connected).
* setInterval ile **her 1 saniyede server’a "SendCategoryCount" metodunu çağırır**.

 **.catch(err => console.log(err))** → Bağlantı sırasında bir hata olursa konsola yazar.



* connection.on(eventName, callback) → Server’dan gelen belirli bir event’i dinler.
* "ReceiveCategoryCount" → Server tarafında bu isimle tanımlı metod çağrıldığında tetiklenir.
* value → Server’dan gelen veri (örneğin kategori sayısı).
* $("#categorycount").text(value); → Gelen değeri sayfadaki elemente yazdırır.

**Özet Mantık**

1. Sayfa yüklendiğinde SignalR bağlantısı kurulur.
2. Bağlantı durumu gösterilir (Disconnected → Connected).
3. Her 1 saniye server’a SendCategoryCount mesajı gönderilir.
4. Server, ReceiveCategoryCount eventi ile kategori sayısını gönderir.
5. JavaScript, bu veriyi sayfadaki #categorycount elementine yazdırır.
6. Böylece **gerçek zamanlı kategori sayısı güncellemesi** yapılmış olur.

* invoke("SendCategoryCount") server metodunu tetikler, server metodunda **Clients.All.SendAsync("ReceiveCategoryCount", value)** ile veriyi tüm client’lara gönderir.
* Böylece her kullanıcı sayfa yenilemeden güncel veri görür.

Ek bilgi

using var context = new SignalRContext(); //--> kod bittiğinde yani scope dışına çıktısında dispose yapılır ve sistem yorumaz

//var context = new SignalRContext(); --> bunu kullanmamamızın sebebi dispose etmezsek hala enerji tükertir

Fluent validation İŞLEMLERİ

using FluentValidation;

using Project.DtoLayer.BookingDto;

using Project.EntityLayer.Concrete;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Project.BusinessLayer.ValidationRules.BookingValidations

{

public class CreateBookingValidation:AbstractValidator<CreateBookingDto>

{

public CreateBookingValidation()

{

RuleFor(x=>x.BookingName).NotEmpty().WithMessage("Bu alan boş geçilemez.");

RuleFor(x=>x.BookingName).MinimumLength(5).WithMessage("Bu alan 5 Karakterden az olamaz.");

RuleFor(x=>x.BookingName).MaximumLength(50).WithMessage("Bu alan 50 Karakterden fazla olamaz.");

RuleFor(x=>x.BookingPhone).NotEmpty().WithMessage("Bu alan boş geçilemez.");

RuleFor(x=>x.BookingEmail).NotEmpty().WithMessage("Bu alan boş geçilemez.");

RuleFor(x=>x.BookingEmail).EmailAddress().WithMessage("Geçerli bir mail adresi giriniz.");

RuleFor(x=>x.BookingDate).NotEmpty().WithMessage("Bu alan boş geçilemez.");

RuleFor(x=>x.BookingPersonCount).NotEmpty().WithMessage("Bu alan boş geçilemez.");

RuleFor(x=>x.BookingDescription).MaximumLength(500).WithMessage("Bu alan 500 karakterden fazla geçilemez.");

}

}

}

private readonly IValidator<CreateBookingDto> \_validator;

public BookingController(IMapper mapper, IBookingService bookingService, IValidator<CreateBookingDto> validator)

{

\_mapper = mapper;

\_bookingService = bookingService;

\_validator = validator;

}

[HttpPost]

public IActionResult BookingInsert(CreateBookingDto createBookingDto)

{

var validationResult = \_validator.Validate(createBookingDto);

if (!validationResult.IsValid)

{

return BadRequest(validationResult.Errors);

}

var detectedValues = \_mapper.Map<Booking>(createBookingDto);

\_bookingService.TInsert(detectedValues);

return Ok("Rezervasyon Başarıyla eklendi");

}

builder.Services.AddValidatorsFromAssemblyContaining<CreateBookingValidation>();

[HttpPost]

public async Task<IActionResult> Index(CreateBookingDto createBookingDto)

{

var client = \_httpClientFactory.CreateClient();

var makeSerialize = JsonConvert.SerializeObject(createBookingDto);

StringContent content = new StringContent(makeSerialize, Encoding.UTF8, "application/json");

var sendData = await client.PostAsync("https://localhost:7240/api/Booking", content);

if (sendData.IsSuccessStatusCode)

{

return RedirectToAction("Index");

}

else

{

var errorContent = await sendData.Content.ReadAsStringAsync();

ModelState.AddModelError(string.Empty, errorContent);

}

return View();

}

**span** **asp-validation-for**="BookingPersonCount" class="text-danger"></**span**>

bunun gibi olacak

@model CreateBookingDto

@{

Layout = null;

}

<!DOCTYPE html>

<html>

@await Component.InvokeAsync("\_UILayoutHeadComponentPartials")

@await Component.InvokeAsync("\_UILayoutNavbarComponentPartials")

<body>

<section class="book\_section layout\_padding">

<div class="container">

<div class="heading\_container">

<h2>

Misafirimiz Olmak İstermisiniz

</h2>

</div>

<div class="row">

<div class="col-md-6">

<div class="form\_container">

<**form** method="post">

<div>

<**input** **type**="text" class="form-control" placeholder="İsim Ve Soyisim" **asp-for**="BookingName" />

<**span** **asp-validation-for**="BookingName" class="text-danger"></**span**>

</div>

<div>

<**input** **type**="text" class="form-control" placeholder="Telefon Numaranız" **asp-for**="BookingPhone" />

<**span** **asp-validation-for**="BookingPhone" class="text-danger"></**span**>

</div>

<div>

<**input** **type**="email" class="form-control" placeholder="Email Adresiniz" **asp-for**="BookingEmail" />

<**span** **asp-validation-for**="BookingEmail" class="text-danger"></**span**>

</div>

<div>

<**select** class="form-control nice-select wide" **asp-for**="BookingPersonCount">

<**span** **asp-validation-for**="BookingPersonCount" class="text-danger"></**span**>

<**option** **value**="" disabled selected>

Kaç Kişisiniz?

</**option**>

<**option** **value**="1">

1

</**option**>

<**option** **value**="2">

2

</**option**>

<**option** **value**="3">

3

</**option**>

<**option** **value**="4">

4

</**option**>

<**option** **value**="5">

5

</**option**>

<**option** **value**="6">

6 kişi ve fazlası

</**option**>

</**select**>

</div>

<div>

<**input** **type**="date" class="form-control" **asp-for**="BookingDate">

<**span** **asp-validation-for**="BookingDate" class="text-danger"></**span**>

</div>

<div class="btn\_box">

<button>

Rezervasyonu Yap

</button>

</div>

</**form**>

</div>

</div>

<div class="col-md-6">

<div class="map\_container ">

<div id="googleMap"></div>

</div>

</div>

</div>

</div>

</section>

@await Component.InvokeAsync("\_UILayoutFooterComponentPartials")

@await Component.InvokeAsync("\_UILayoutScriptComponentPartials")

</body>

</html>

QR İŞLEMLERİ

Dlli dependencis de browserden arat uı kısmına koy

Ardından kodlar