RISETECHNOLOGY

API Entegrasyonu & Dokümantasyon

(*REST API – Telefon Rehberi*)

TELEFON REHBERİ MİKROSERVİS KULLANIMI

VERSION 1.0
23/01/2023

Hazırlayan: MUHAMMET ESER

Sayfalama

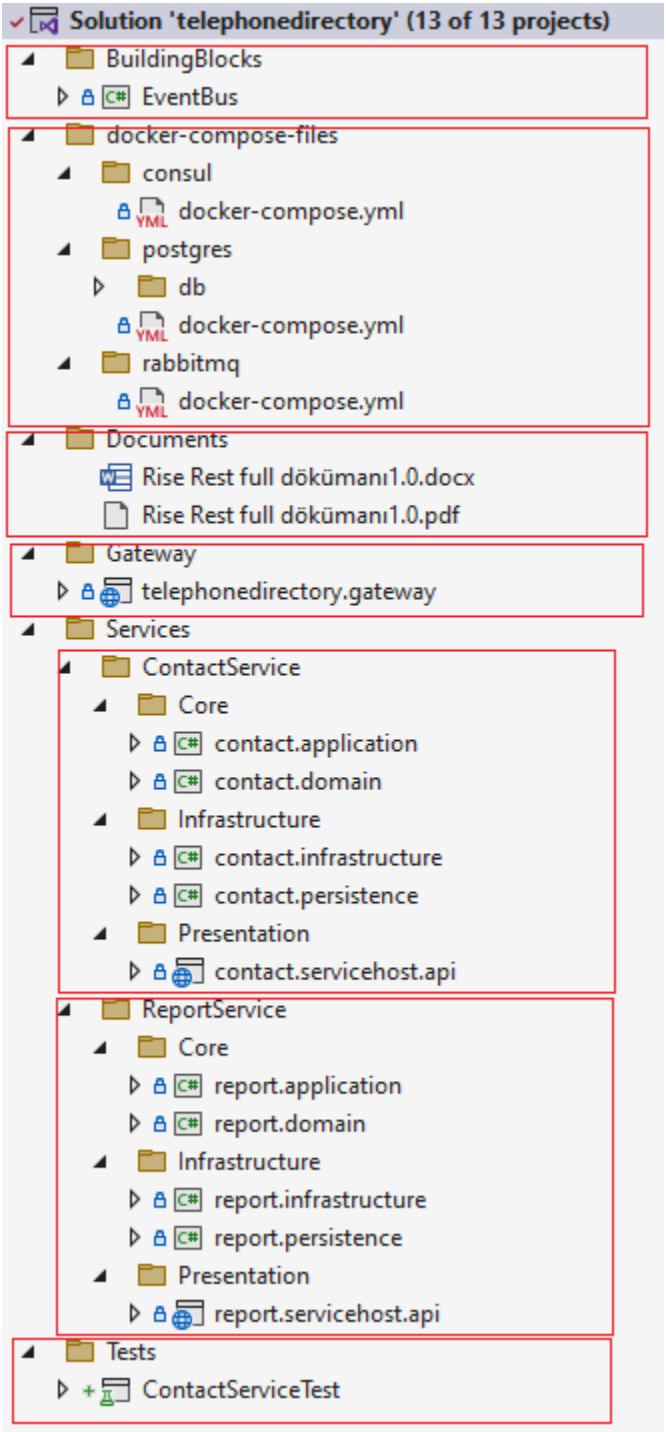
İçerik

Mikroservis Mimarisinin Yapısı

Bu dökümanın amacı, Telefon Rehberi RestAPI'leri (Uygulama Arayüzü) için yüksek düzeyde bir özellik sağlamaktır. Bu dökümantasyonun okuyucularının zaten temel API/Web Hizmetleri kavramlarını anlamış olmasını bekliyoruz.

Telefon Rehberi Web API hizmeti, Client tarafından, istedikleri zaman APİ ye ait detayları almak için kullanılacaktır. API , Client tarafından çağrıldığında; doğrulatıp yetkilendireceğiz ve sahip oldukları erişim düzeyine ve ihtiyaç duydukları bilgilere göre iletişim bilgilerini gösterecektir. **Cevap, varsayılan olarak JSON biçiminde olacaktır.**

Aşağıda açıklanan endpointler, API çağrılarıdır ve client ihtiyaçlarına hizmet etmek için gerektiği kadar endpoint hazırlanabilir.



Teknoloji ve Mimari

Proje ,Asp .Net 6.0 ile yazılarak mikroservis mimarisi hazırlanmıştır.

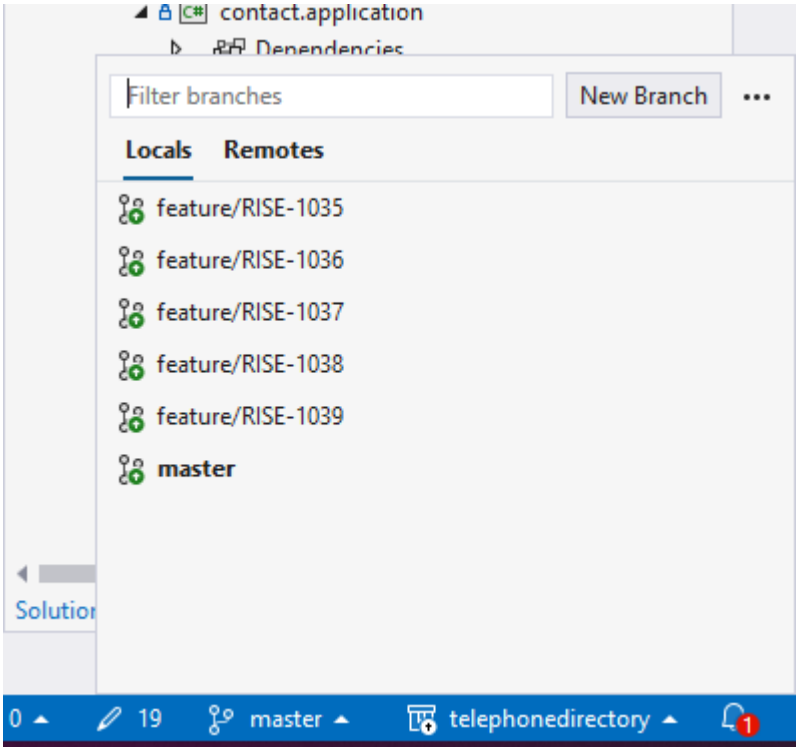
Yapısal olarak ;

- ✓ Veritabanı kısmında Postgresql
- ✓ Kuyruk sisteminde RabbitMQ
- ✓ Mikroservisler arası iletişimde EventBus(Mass Transit)
- ✓ Service Discover için Consul

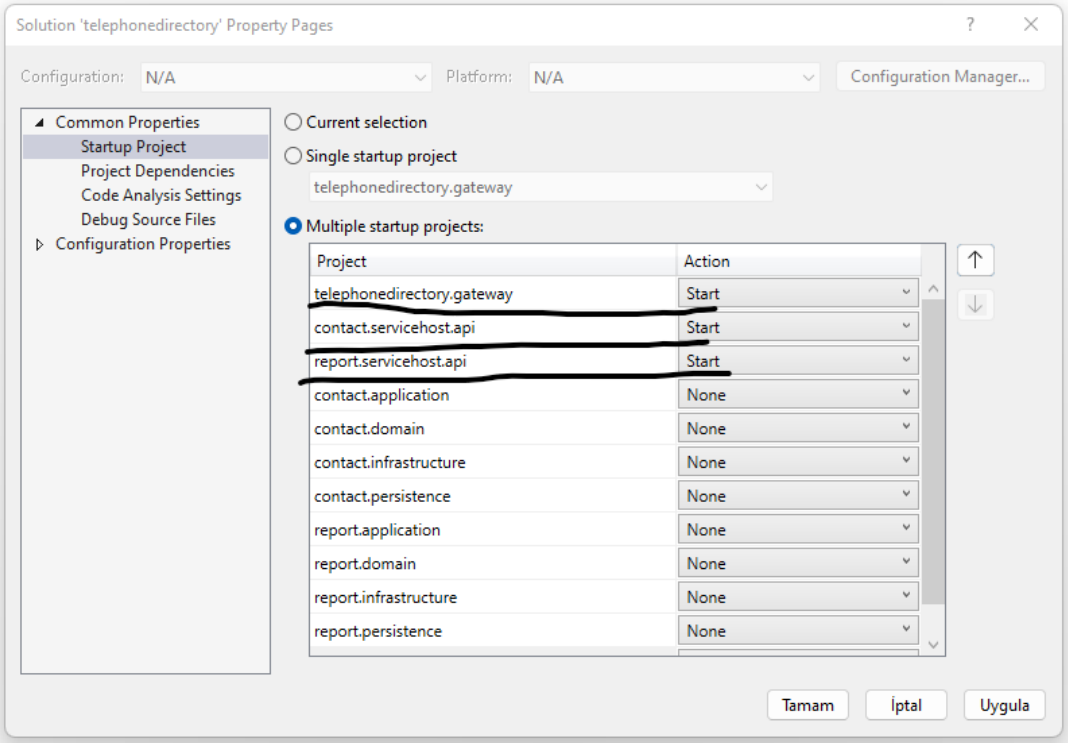
kullanılmıştır. Postgresql, RabbitMQ ve Consul docker üzerinden çalışmaktadır. Docker üzerinden çalışan uygulamalar için proje içerisinde bulunan docker-compose-files klasöründe .yml dosyaları bulunmaktadır .yml dosyasının bulunduğu dizinde power-shell i çalıştırıp docker-compose up komutu yazarak kurabilirsiniz .Uluslararası standartlara uygun patternlerle proje geliştirilmiştir. Onion Architecture, CQRS, Domain-Driven-Design kullanılmıştır.

Teknolojiler ve Kullanım Amaçları

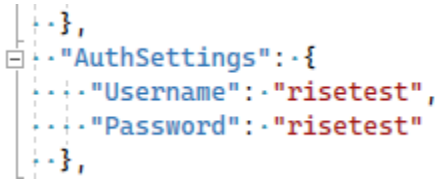
Projede versiyonlama kullanılarak çalışılmıştır ve git flow kullanıldı,farklı branch lerde hazırlanan modüller daha sonra master branch ine yıkılarak birleştirildi.

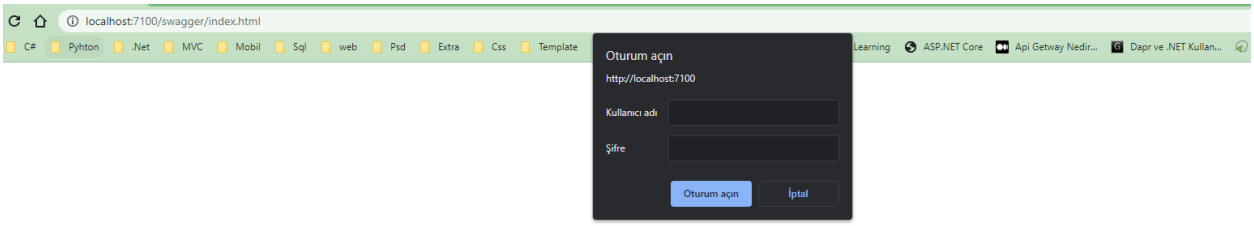


Projeyı çalıştırmak için aşağıdaki gibi ayarlanmalıdır.

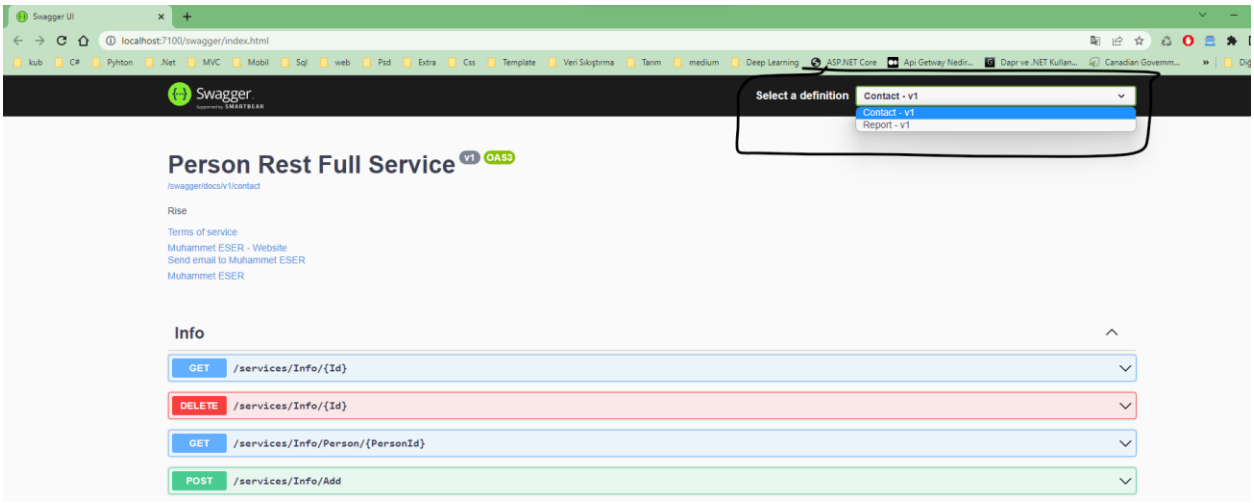


Proje çalıştıgında apiye istek atabilmek için kullanıcı adı ve şifre girilmedir.





Giriş yaptıkdan sonar mikroservisler burada listelenmektedir . Aşağıdaki gibi;



Ocelot

API-GATEWAY desteği sağlamaktadır. Tek bir domain üzerinden bütün mikroservislere istek atılmasını sağlar.Gateway projesi içinde bulunan json dosyasında konfigürasyon ayarları yapılmaktadır. Ayrı bir sunucuya kurulmalıdır.

Nuget paketi olarak ;

- MMLib.SwaggerForOcelot
- Ocelot

Farklı projelerde bulunan controller burada tanımlıyoruz.

Swaggerde görünmesini ayarlıyoruz.

Yayın yapacağımız apigateway adresimiz.

Consul

ServiceDiscovery olmasını sağlar.Mikroservislerin bulundugu cihaz ip sini otomatik olarak alıp Gateway kısmına kaydeder.Çalışan veya çalışmayan mikroservisleri monitoring olarak izlemektedir. Ayrı bir sunucuya kurulabilir.

Nuget paketi olarak

- Ocelot.Provider.Consul

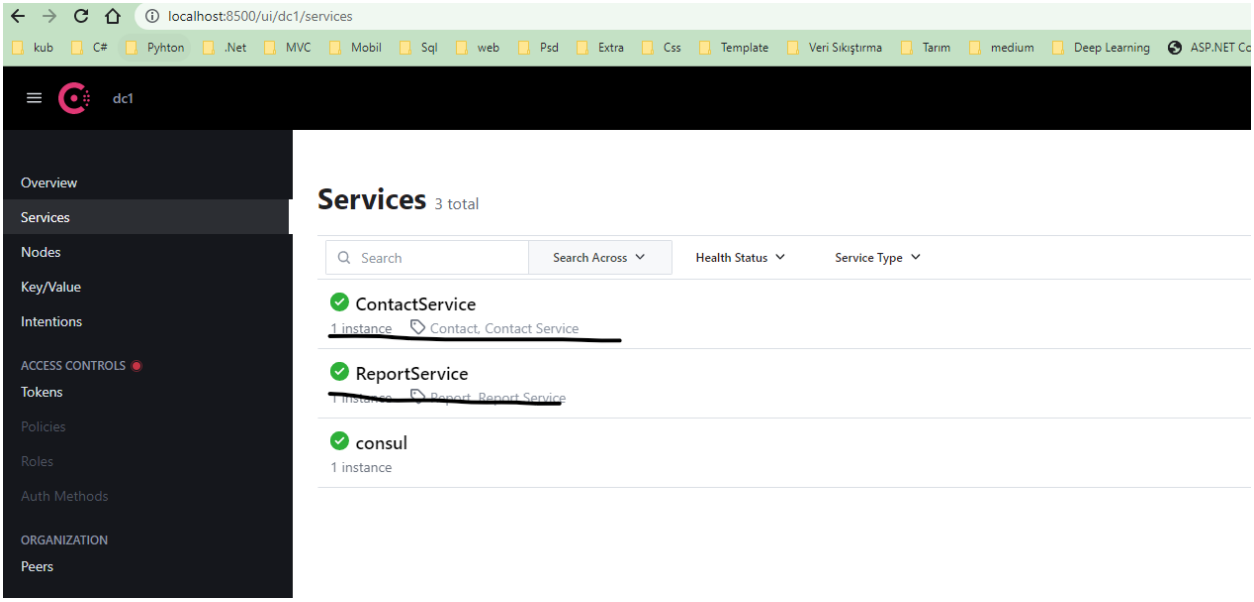
Apigatewayde ocelot konfigürasyonunda discovery true yapmamız gerekiyor.

```
  "GlobalConfiguration": {
    "BaseUrl": "http://localhost:7100",
    "ServiceDiscoveryProvider": {
      "Host": "localhost",
      "Port": 8500,
      "Type": "Consul"
    }
  }
```

Apigateway kısmında ocelot için ikinci konfigürasyon tanımları

```
using Microsoft.Extensions.Logging;
namespace report.servicehost.api.Extensions
{
    0 references
    public static class ConsulRegistration
    {
        1 reference
        public static IServiceCollection ConfigureConsul(this IServiceCollection services, IConfiguration configuration)
        {
            services.AddSingleton<IConsulClient, ConsulClient>(p => new ConsulClient(consulConfig =>
            {
                var address = configuration["ConsulConfig:Address"];
                consulConfig.Address = new Uri(address);
            }));

            return services;
        }

        1 reference
        public static IApplicationBuilder RegisterWithConsul(this IApplicationBuilder app, IHostApplicationLifetime lifetime)
        {
            var consulClient = app.ApplicationServices.GetRequiredService<IConsulClient>();

            var loggingFactory = app.ApplicationServices.GetRequiredService<ILoggerFactory>();

            var logger = loggingFactory.CreateLogger<IApplicationBuilder>();

            //Get server IP address
            var features = app.Properties["server.Features"] as FeatureCollection;
            var addresses = features.Get<IServerAddressesFeature>();
            var address = addresses.Addresses.Count() == 0 ? "http://localhost:7102" : addresses.Addresses.First();

            //Register service with consul
            var uri = new Uri(address);
            var registration = new AgentServiceRegistration()
            {
                ID = $"ReportService",
                Name = "ReportService",
                Address = $"{uri.Host}",
                Port = uri.Port,
                Tags = new[] { "Report Service", "Report" },
            };

            logger.LogInformation("Registering with Consul");
            consulClient.Agent.ServiceDeregister(registration.ID).Wait();
            consulClient.Agent.ServiceRegister(registration).Wait();

            lifetime.ApplicationStopping.Register(() =>
            {
```

Her bir mikroservisde yapılandırılması için , konfigürasyon ayaları



Çalışan mikroservislerin monitoring yaparak izlenmesi.

CQRS (MediatR)

Proje kapsamında cqrs patterni aktif olarak kullanılmaktadır. Mikroservislerin hepsinde Mediatr ile birlikte kullanılarak handler edildi.

Commands ve Query olarak ayrı ayrı modeller tanımlanıp uygun olarak geliştirime yapıldi. İlk olarak gelen istek Mediatr ile send edilip sonrada handler e düşmektedir.

Nuger paketi olarak ;

- MediatR
- MediatR.Extensions.Microsoft.DependencyInjection

```
namespace report.servicehost.api.Controllers
{
    [ApiController]
    [Route("api/v1/[controller]")]
    [EnableCors("AllowOrigin")]
    [Produces("application/json")]

    0 references
    public class ReportController : ControllerBase
    {
        private IMediator _mediator;
        1 reference
        protected IMediator Mediator => _mediator ??= HttpContext.RequestServices.GetService<IMediator>();

        [Route("Get")]
        [HttpGet]
        0 references
        public async Task<IActionResult> Get()
        {
            var response = await Mediator.Send(new GetAllReportsQuery());
            return Ok(response);
        }

    }
}
```

Domain-Driver-Desgin (DDD)

Mikroservis projemizde DDD tam olarak uygulanmıştır,ve Proje genelinde farklı kalıplara ayrılarak hazırlanmıştır.

Message Brokers (RabbitMQ & MassTransit)

Kuyruk sistemi olarak Rabbitmq kullanıldı ve event bus içinde mass transit tercih edildi.Farklı mikroservisler birbirleri arasında iletişim kurarak toplu işlemlerin yapılmasını sağlamaktadır. Ayrı br sunucuya kurulmalıdır.

Nuget paketi olarak ;

- MassTransit
- MassTransit.AspNetCore
- MassTransit.RabbitMQ

```
"RabbitMQ": {
    "baseuri": "rabbitmq://localhost:5672",
    "username": "guest",
    "password": "guest",
    "personidqueue": "/personIdQueue",
    "personData": "personData"
}
```

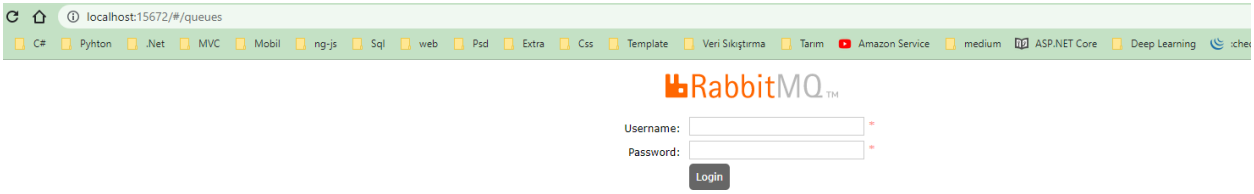
Kullanılacak mikroserviste konfigürasyon işlemleri

```
0 references
public static class BusConfiguratorRegistration
{
    1 reference
    public static void ConfigureBus(this IServiceCollection services, IConfiguration configuration)
    {
        services.AddMassTransit(x =>
        {
            x.AddBus(provider => Bus.Factory.CreateUsingRabbitMq(config =>
            {
                config.Host(new Uri(configuration["RabbitMQ:baseuri"]), h =>
                {
                    h.Username(configuration["RabbitMQ:username"]);
                    h.Password(configuration["RabbitMQ:password"]);
                });
            }));
        });
        services.AddMassTransitHostedService();
    }
}
```

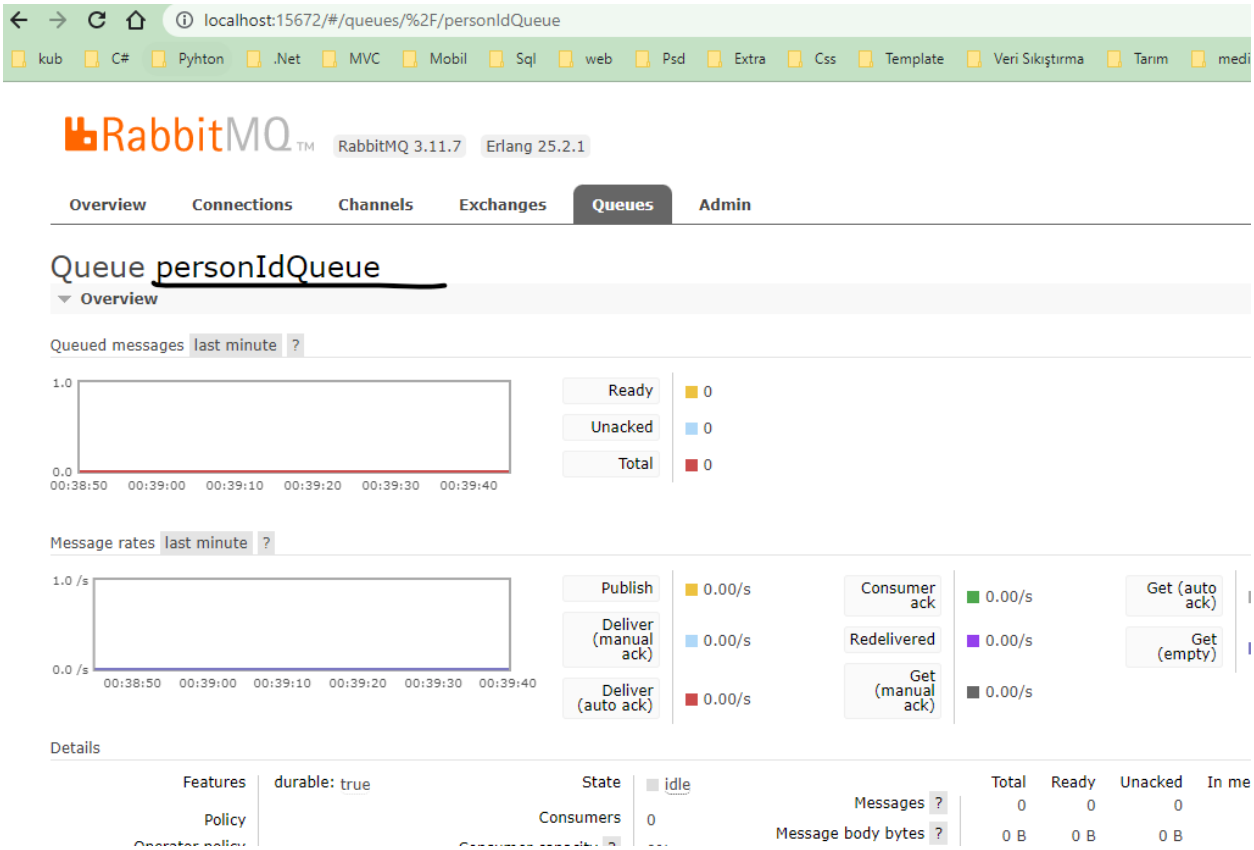
Kuyruk sisteminin publisher kısmı için konfigürasyon ayaları

```
0 references
public static class BusConfiguratorRegistration
{
    1 reference
    public static void ConfigureBus(this IServiceCollection services, IConfiguration configuration)
    {
        services.AddMassTransit(x =>
        {
            x.AddConsumer<PersonsConsumer>();
            x.AddBus(provider => Bus.Factory.CreateUsingRabbitMq(config =>
            {
                config.Host(new Uri(configuration["RabbitMQ:baseuri"]), h =>
                {
                    h.Username(configuration["RabbitMQ:username"]);
                    h.Password(configuration["RabbitMQ:password"]);
                });
                config.ReceiveEndpoint(configuration["RabbitMQ:personidqueue"], ep =>
                {
                    ep.PrefetchCount = 16;
                    ep.UseMessageRetry(r => r.Interval(2, 100));
                    ep.ConfigureConsumer<PersonsConsumer>(provider);
                });
            }));
        });
        services.AddMassTransitHostedService();
    }
}
```

Kuyruk sisteminin Consumer kısmı için konfigürasyon ayarları



Login olarak kuyruktaki işlemleri görebilirsiniz



Kuyruklar buradan detaylı incelenebilir.

AutoMapper (Mapping)

Client tarafına gönderilecek olan verilerin ve propertlerin haritalandıgı yerdir.En önemli kısım ise veritabanında bulunan property name lerin client tarafında görünmemesinidr. Gösterilmek istenmeyen property ler içinde **Ignore()** yapabilmekteyiz.

Nuget paketi olarak ;

- AutoMapper
- AutoMapper.Extensions.Microsoft.DependencyInjection

```
1 reference
public class PersonsMapping : Profile
{
    0 references
    public PersonsMapping()
    {
        CreateMap<CreatePersonsCommands, contact.domain.Entities.Persons>()
            .ReverseMap();
        CreateMap<contact.domain.Entities.Persons, PersonsResponse>()
            .ReverseMap();
    }
}
```

Mapping kullanımı ReverseMap tersinide almak anlamındadır.

```
0 references
static public class ServiceRegistration
{
    1 reference
    public static void AddApplicationServices(this IServiceCollection serviceCollection)
    {
        serviceCollection.AddMediatR(Assembly.GetExecutingAssembly());
        serviceCollection.AddAutoMapper(Assembly.GetExecutingAssembly());
    }
}
```

Buradaki konfigürasyon ayarındaki en önemli konu mapplemenin yapıldığı projede eklenmelidir. Daha sonra Program.cs de eklenmeli buradaki fonksiyon

FluentValidation (Property Null kontrolleri)

Propertylerin null ,check, if gibi kontrolleri yapmak için kulanılmatadır ve property üzerine attribute olarak eklememek için kullanılır.

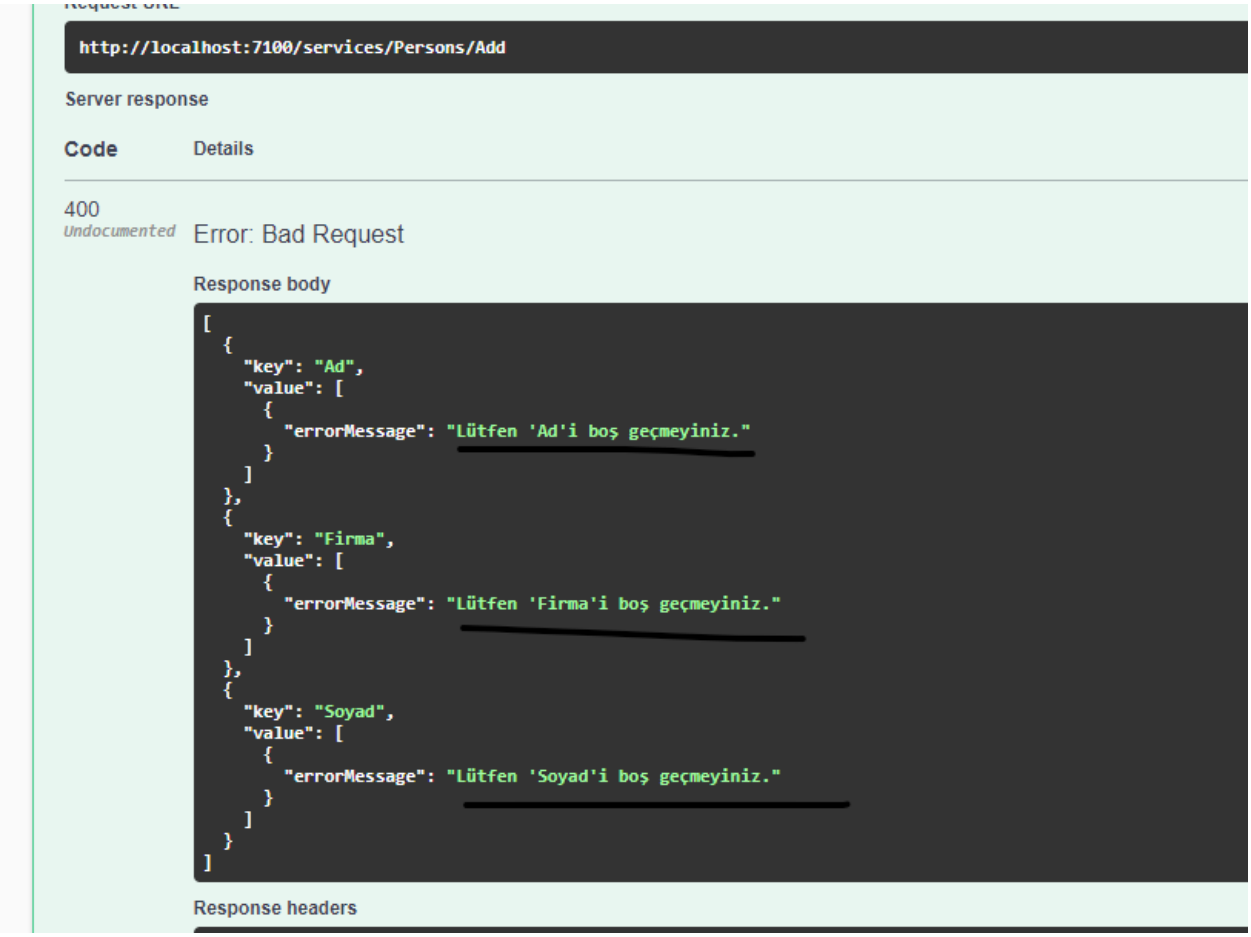
Nuget paketi olarak ;

- FluentValidation.DependencyInjectionExtensions

```
2 references
public class CreatePersonsCommandsValidator : AbstractValidator<CreatePersonsCommands>
{
    0 references
    public CreatePersonsCommandsValidator()
    {
        RuleFor(p => p.Ad)
            .NotNull()
            .NotEmpty()
            .WithMessage("Lütfen 'Ad'i boş geçmeyiniz.");
        RuleFor(p => p.Soyad)
            .NotNull()
            .NotEmpty()
            .WithMessage("Lütfen 'Soyad'i boş geçmeyiniz.");
        RuleFor(p => p.Firma)
            .NotNull()
            .NotEmpty()
            .WithMessage("Lütfen 'Firma'i boş geçmeyiniz.");
    }
}
```

Burada zorunlu alanları RoleFor olarak ekleyip acıklamasını belirtiğimizde .bize response olarak mesaj vermektedir.

Burada doş gönderdiğimde paramerteleri



Burada boş bırakılan ve kurallara uymayan propertyler detaylı bir şekilde açıklanmaktadır.