# **Real-Time Chat Backend Servisi**

Bu proje, **Bluesense Backend Engineer Assignment** görevi kapsamında geliştirilmiş, ASP.NET Core tabanlı, ölçeklenebilir ve gerçek zamanlı bir sohbet uygulaması için backend servisidir.

Proje, modern .NET teknolojileri kullanılarak, temiz kod ve sağlam mimari prensipleriyle geliştirilmiştir. Kimlik doğrulama, grup yönetimi, gerçek zamanlı mesajlaşma ve dosya yükleme gibi temel özellikleri içerir. Ayrıca Docker ile konteynerleştirilmiş ve GitHub Actions ile CI/CD otomasyonu sağlanmıştır.

## **Ana Özellikler (Key Features)**

* **Auth (Kimlik Doğrulama):** JWT tabanlı güvenli kullanıcı kaydı (/register) ve girişi (/login).
* **Groups (Gruplar):** Herkese açık/özel gruplar oluşturma, gruplara katılma ve ayrılma.
* **Messages (Mesajlar):** Grup içinde metin ve dosya (resim, belge vb.) gönderimi.
* **Real-time (Gerçek Zamanlı):** SignalR kullanılarak WebSocket üzerinden anlık mesaj güncellemeleri.
* **Listing & Search (Listeleme ve Arama):** Grup içi mesajlarda ve herkese açık gruplarda arama yapma.

## **Teknoloji Yığını (Tech Stack)**

* **Framework:** ASP.NET Core 8 (Web API)
* **Veritabanı:** Entity Framework Core & SQLite
* **Gerçek Zamanlı İletişim:** SignalR
* **Test:** xUnit
* **Konteynerleştirme:** Docker & Docker Compose
* **CI/CD:** GitHub Actions

## **Yerel Kurulum ve Çalıştırma (Local Setup)**

Projeyi yerel makinenizde çalıştırmak için aşağıdaki adımları izleyin:

1. **Depoyu Klonlayın:**  
   git clone <https://github.com/yldzxsvvl/RealtimeChatBackend.git>
2. **Proje Dizinine Gidin:**  
   cd YeniChatProjesi   
   # Not: Eğer klonladığınız klasörün adı farklıysa, o klasörün içine girin.
3. **Gerekli Paketleri Yükleyin:**  
   dotnet restore
4. **Uygulamayı Başlatın:**  
   dotnet run --project RealtimeChatBackend
5. Uygulama artık http://localhost:5222 adresinde çalışıyor olacaktır.

## **Docker ile Çalıştırma**

Projeyi Docker konteynerleri içinde çalıştırmak için ana dizinde aşağıdaki komutu kullanın. Docker'ın bilgisayarınızda kurulu ve çalışır durumda olması gerekmektedir.

docker-compose up --build

## **Testler**

Projenin birim ve entegrasyon testlerini çalıştırmak için ana dizinde aşağıdaki komutu kullanın:

dotnet test

## **API Kullanımı ve Dokümantasyonu**

Tüm API endpoint'leri, modelleri ve test arayüzü için Swagger dokümantasyonunu kullanabilirsiniz. Sunucu çalıştıktan sonra aşağıdaki adrese gidin:

* **Swagger UI:** http://localhost:5222/swagger

Gerçek zamanlı sohbet arayüzünü test etmek için hazırlanan basit HTML istemcisini kullanabilirsiniz:

* **SignalR Test İstemcisi:** <http://localhost:5222/SignalRTestClient.html>

**Mimari ve Teknik Kararlar**

### **Mimari**

Proje, **Onion Architecture (Soğan Mimarisi)** prensiplerine uygun olarak tasarlanmıştır. Bu mimari, projenin katmanlarını birbirinden bağımsız ve test edilebilir hale getirir.

.  
├── Core/ # İş mantığı ve varlıklar (Entities).  
├── Application/ # Arayüzler (Interfaces) ve servis kontratları.  
├── Infrastructure/ # Veritabanı, harici servisler gibi dış dünya implementasyonları.  
└── API/ # Controller'lar, Hub'lar ve dış dünyaya açılan kapı.

### **Teknik Kararlar ve Takaslar (Trade-offs)**

Bu bölümde, görev tanımındaki gereksinimlere karşılık geliştirme sürecini hızlandırmak ve basitleştirmek için alınan bilinçli kararlar açıklanmaktadır.

* **Veritabanı Seçimi:** Görev tanımında PostgreSQL istenmesine rağmen, geliştirme kolaylığı ve harici bir kurulum bağımlılığını ortadan kaldırmak için **SQLite** tercih edilmiştir. Entity Framework Core kullanıldığı için, Program.cs ve appsettings.json dosyalarındaki bağlantı dizesi değiştirilerek ve ilgili EF Core paketi eklenerek proje kolayca PostgreSQL'e geçirilebilir. Bu, projenin "polyglot persistence" yeteneğini gösterir.
* **Önbellekleme (Caching):** Görev tanımında Redis istenmesine rağmen, geliştirme ortamında harici bir Redis sunucusu kurma zorunluluğunu ortadan kaldırmak için basit bir **InMemoryCacheService** kullanılmıştır. RedisCacheService altyapısı ve arayüzü hazır olup, Program.cs dosyasında tek satırlık bir değişiklikle kolayca aktive edilebilir. Bu, projenin farklı ortamlara kolayca adapte olabilmesini sağlar.

## **CI/CD Pipeline**

Proje, GitHub Actions kullanılarak bir Sürekli Entegrasyon (CI) hattına sahiptir. main branch'ine yapılan her push işleminde aşağıdaki adımlar otomatik olarak çalıştırılır:

1. Kod indirilir.
2. .NET 8 SDK'sı kurulur.
3. Bağımlılıklar yüklenir (dotnet restore).
4. Proje derlenir (dotnet build).
5. Tüm testler çalıştırılır (dotnet test).