# Backend Developer Code Case - Kod Açıklamalı Teknik Dökümantasyon

Hazırlayan: İshak Güzel

Tarih: 2025-10-17

Proje: Real-Time Configuration System (RabbitMQ + SignalR)

Teknolojiler: .NET 8, ASP.NET Core, RabbitMQ, SignalR, Docker, JSON

## İçindekiler

1. 1. Genel Bakış
2. 2. Klasör ve Dosya Yapısı
3. 3. Bileşen Bazlı Kod Açıklamaları
4. 4. Metot Bazlı Detaylar
5. 5. Akış Diyagramı ve Veri Yolu
6. 6. API Endpoint Dokümantasyonu
7. 7. Docker & Çalıştırma
8. 8. Hata Senaryoları ve Dayanıklılık
9. 9. Geliştirme Önerileri
10. 10. Ek: Örnek İstekler ve SignalR Eventleri

## 1. Genel Bakış

Bu belge projenin geliştiricilerinin kolayca anlayacağı şekilde, kod bazlı açıklamalarla hazırlanmıştır. Merkezi konfigürasyon yönetimi ve gerçek zamanlı bildirim amaçlanmıştır.

## 2. Klasör ve Dosya Yapısı

- backend\_solution.sln : Visual Studio çözüm dosyası  
- src/ConfigurationReaderLib : Ana kütüphane (ConfigurationReader, publisher, listener, hub)  
- src/ExampleApp : Minimal API + UI + SignalR host  
- storage/configs.json : Demo konfigürasyon verisi  
- docker-compose.yml : RabbitMQ ve app servisleri

## 3. Bileşen Bazlı Kod Açıklamaları

### ConfigurationReader.cs

Ana görevleri:  
- Başlangıçta storage'dan snapshot alıp cache'i oluşturur (LoadSnapshotAsync).  
- Periyodik olarak storage'ı kontrol eder (RefreshIfNeededAsync).  
- Değişiklik algılarsa cache'i günceller ve PublishChangeSafeAsync ile publish eder.  
- GetValue<T> ile tip güvenli geri dönüş sağlar; dönüşüm hatalarını fırlatır.

public ConfigurationReader(string applicationName, IConfigurationStore store, int refreshTimerIntervalInMs, IConfigurationChangePublisher? publisher = null)  
T GetValue<T>(string key)  
private Task LoadSnapshotAsync()  
private Task RefreshIfNeededAsync()

### RabbitMqPublisher.cs

Görevi: ConfigurationRecord objelerini 'config\_changes' exchange'ine JSON olarak publish eder. Fanout exchange kullanarak tüm bağlı queue'lara iletilmesini sağlar. Hata toleranslı kullanım PublishChangeSafeAsync tarafından ele alınır.

public Task PublishChangeAsync(ConfigurationRecord record)  
\_channel.BasicPublish(exchange: \_exchange, routingKey: "", basicProperties: null, body: bytes);

### ConfigurationChangeListener.cs

Görevi: 'config\_changes' exchange'ini dinleyip gelen mesajları SignalR Hub üzerinden istemcilere iletir. Queue bind ve consumer işlemleri burada gerçekleşir. Gelen mesaj JSON olarak deserialize edilir ve Clients.All.SendAsync('configChanged', rec) ile broadcast edilir.

\_channel.ExchangeDeclare(exchange: \_exchange, type: ExchangeType.Fanout, durable: false, autoDelete: true);  
\_channel.QueueBind(queue: \_queueName, exchange: \_exchange, routingKey: "");  
\_channel.BasicConsume(queue: \_queueName, autoAck: true, consumer: \_consumer);

### ConfigHub.cs

Basit SignalR Hub tanımı. Server tarafı IHubContext<ConfigHub> üzerinden broadcast yapar. Client bağlanıp 'configChanged' event'ini dinler.

public class ConfigHub : Hub { }

### JsonFileConfigurationStore.cs

Demo amaçlı JSON file store implementasyonu. GetAllAsync storage dosyasını okuyup ConfigurationRecord listesi döner. Üretim ortamında MongoDB/SQL/Redis gibi storagelar tercih edilmeli.

public Task<IEnumerable<ConfigurationRecord>> GetAllAsync(string applicationName)

### Program.cs (ExampleApp)

Minimal API yapılandırması. DI container'a ConfigurationReader ve store kayıtları yapılır. Endpoints:  
- GET /api/configs  
- POST /api/configs  
- SignalR hub: /confighub  
Ek: Uygulama başında ConfigurationChangeListener başlatılarak SignalR hub ile köprü kuruluyor.

text: app.MapGet(...); app.MapPost(...); app.MapHub<ConfigHub>("/confighub");

### index.html (Client)

HTML + JavaScript client. load() fonksiyonu ile /api/configs çekilip tablo güncellenir. add() ile POST atılır. SignalR client connection.on('configChanged', ...) ile gelen eventlerde load() tetiklenir. Automatic reconnect aktif.

const connection = new signalR.HubConnectionBuilder().withUrl('/confighub').withAutomaticReconnect().build();  
connection.on('configChanged', rec => { load(); });  
connection.start();

## 4. Metot Bazlı Detaylar

LoadSnapshotAsync(): Başlangıç snapshot'ı alır ve cache'i initialize eder. RefreshIfNeededAsync(): Periyodik kontrol ve fark tespiti yapar. PublishChangeSafeAsync(): Hata toleranslı yayın. GetValue<T>(): Cache'den değer okur ve type-conversion yapar.

## 5. Akış Diyagramı ve Veri Yolu

UI / API -> configs.json (storage) -> ConfigurationReader (detect) -> RabbitMqPublisher -> RabbitMQ Exchange -> ConfigurationChangeListener -> SignalR Hub -> Browser Clients

## 6. API Endpoint Dokümantasyonu

GET /api/configs -> Aktif konfigürasyonları listeler  
POST /api/configs -> Yeni config ekler (body örneği belgede). SignalR hub: /confighub (real-time updates).

## 7. Docker & Çalıştırma

docker-compose.yml ile RabbitMQ ve ExampleApp ayağa kalkar. Komut: docker-compose up --build. RabbitMQ yönetimi: http://localhost:15672 (guest/guest).

## 8. Hata Senaryoları ve Dayanıklılık

RabbitMQ kapalıysa publish çağrıları swallow edilir; sistem cache ile çalışmaya devam eder. JSON bozuksa template yeniden yazılabilir. SignalR client auto reconnect kullanır.

## 9. Geliştirme Önerileri

Kalıcı DB, admin UI, audit log, config versiyonlama, erişim kontrolü (JWT).

## 10. Ek: Örnek İstek ve SignalR Event

Örnek POST (curl):  
curl -X POST http://localhost:5000/api/configs -H "Content-Type: application/json" -d '{  
 "Name":"IsFeatureEnabled",  
 "Type":"bool",  
 "Value":"true",  
 "IsActive":true,  
 "ApplicationName":"SERVICE-A"  
}'  
  
SignalR client örneği (browser):  
const connection = new signalR.HubConnectionBuilder().withUrl('/confighub').withAutomaticReconnect().build();  
connection.on('configChanged', rec => console.log('Config changed', rec));  
connection.start();

---  
Hazırlayan: İshak Güzel  
Not: Bu belge geliştirici referansı amaçlıdır.