İçindekiler Tablosu

1. Giriş .....2
2. Kütüphane Yapısı.....2
3. Model Gereksinimleri ve Generic Yapı .....3
4. Kurulum ve Yapılandırma .....4
5. Kullanım Senaryoları.....5
6.Interface Metotları — Kısa Açıklamalar.....6
7.Sonuç.....7

# 1. Giriş

Projelerimde her defasında aynı şeyleri tekrar ettiğimi fark ettim.
JWT servisleri, interface tanımları, cookie ayarları, refresh token yönetimi…
Her yeni projede “kopyala-yapıştır” yaparak başlamak yerine, hepsini tek bir sade, düzenli ve genişletilebilir yapı altında toplamak istedim.

İşte Yunus.JwtKit bu ihtiyaçtan doğdu.
Standart JWT çözümleri genellikle sadece Access Token üretimiyle sınırlı.
Ancak gerçek bir kimlik doğrulama süreci yalnızca token üretmekten ibaret değil.
JwtKit, Refresh Token yönetimi, cookie bazlı oturum desteği ve ASP.NET Identity uyumu sayesinde,
hem *güvenli* hem de *esnek* bir kimlik doğrulama altyapısı sunar.

Kütüphane, Clean Code ve Plug & Play mantığıyla geliştirildi:
Her proje için sıfırdan servis veya interface yazmana gerek kalmaz.
İster büyük bir kurumsal API projesinde, ister küçük bir kişisel denemede kullan — Yunus.JwtKit tüm senaryolara uyum sağlar.

Amaç, geliştiricinin sıfırdan token sistemi kurmakla uğraşmadan, kolayca **program akışını** çalıştırabilmesidir.

**Desteklenen ortamlar:**

- Local (Swagger, Postman)
- Cross-Origin (React, Next.js, Angular)
- Production (HTTPS, HttpOnly, Secure cookie)

**Özellikler:**

- Access ve Refresh token üretimi
- Cookie veya Header temelli token taşıma
- XSS ve CSRF’e karşı cookie güvenliği
- ASP.NET Identity uyumu (ama zorunlu değil)
- Plug & Play kurulum: 5 dakikada aktif

# 2. Kütüphane Yapısı

Kütüphane 2 ana bileşenden oluşur:

**a) Interface – IJwtTokenService<TUser, TUserInfo>**

Token üretimi, yenileme, cookie işlemleri gibi tüm temel metotları tanımlar.
Bu interface, dışarıya “sözleşme” sunar. Proje bağımlılığını azaltır.

**b) Service – JwtService<TUser, TUserInfo>**

Interface'in implementasyonudur.
Token üretir, doğrular, refresh token'ı kaydeder, cookie işlemlerini yönetir.
AppSettings ve UserManager yapısı üzerinden tamamen dinamik çalışır.

**Generic tipler (TUser, TUserInfo)** paketin en güçlü tarafıdır.
İstersen IdentityUser'la çalışır, istersen kendi modelinle.

# 3. Model Gereksinimleri ve Generic Yapı

JwtService iki generic tip ister:

## 1- TUser

Kullanıcı modelidir.
Refresh token ve süresini tutmak için aşağıdaki interface'i uygulamalıdır:

```
public interface IHasRefreshToken
{
    string? RefreshToken { get; set; }
    DateTime? RefreshTokenExpiryTime { get; set; }
}
```

🔷 **Eğer ASP.NET Identity kullanıyorsan:**

```
public class AppUser : IdentityUser, IHasRefreshToken
{
    public string? RefreshToken { get; set; }
    public DateTime? RefreshTokenExpiryTime { get; set; }

    public string? FirstName { get; set; }
    public string? LastName { get; set; }
}
```

🔷 **Eğer Identity kullanmıyorsan:**

```
public class User : IHasRefreshToken
{
    public int Id { get; set; }
    public string Email { get; set; } = string.Empty;
    public string PasswordHash { get; set; } = string.Empty;

    public string? RefreshToken { get; set; }
    public DateTime? RefreshTokenExpiryTime { get; set; }
}
```

### 2-TUserInfo

Token response içinde dönecek kullanıcı bilgisi (DTO).
Yani API dışına hangi bilgileri göndermek istiyorsan sadece onları içerir.

```
public class JwtResponse
{
    public string? FirstName { get; set; }
    public string? LastName { get; set; }
    public string? Email { get; set; }
}
```

Token response şu şekilde görünür:

```
{
  "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiVGVzdCIsInN1YiI6IjEyMzQ1Njc4OTA
  "refreshToken": "U9xZp4mKk5vT3sF1p2rQ9lS4jM7bH6dP3tG8qV1xE5yN0sC2dJ5oF0aG8uW6rQ2l=",
  "tokenType": "Bearer",
  "expiresIn": 3600,
  "issuedAt": "2025-11-07T13:00:00Z",
  "user": {
    "firstName": "Test",
    "lastName": "User",
    "email": "test@example.com"
  }
}
```

# 4. Kurulum ve Yapılandırma

**NuGet Üzerinden Kurulum**

```
dotnet add package Yunus.JwtKit
```

**appsettings.json**

```
"JwtSettings": {
  "SecretKey": "your_super_secret_key",
  "Issuer": "your_app",
  "Audience": "your_users",
  "AccessTokenExpiryMinutes": 60,
  "RefreshTokenExpiryDays": 7
}
```

**Program.cs – Servis Kaydı**

```
ilder.Services.AddScoped<IJwtTokenService<AppUser, JwtResponse>, JwtService<AppUser, JwtResponse>>(s

  var service = new JwtService<AppUser, JwtResponse>(
      sp.GetRequiredService<UserManager<AppUser>>(),
      sp.GetRequiredService<ILogger<JwtService<AppUser, JwtResponse>>>(),
      sp.GetRequiredService<IConfiguration>()
  );

  // 🔷 Local (Swagger / Postman) ortamı için (⚙️ Default değerler)
  // CookieHttpOnly = false
  // CookieSecure = false
  // CookieSameSite = SameSiteMode.Lax

  // 🔷 Cross-Origin (React / Next.js) ortamı için (örnek konfigürasyon)
  // service.CookieHttpOnly = true;
  // service.CookieSecure = true;
  // service.CookieSameSite = SameSiteMode.None;

  return service;
;
```

# 5. Kullanım Senaryoları

**5.1 Local Geliştirme (Swagger / Postman) Cookie zorunlu değildir.**
Token body veya header üzerinden taşınabilir.

- HttpOnly = false
- Secure = false
- SameSite = Lax

```
{
  "success": true,
  "data": {
    "accessToken": "eyJhbGciOiJUzTESTaccesstoken123...",
    "refreshToken": "sd9f8sdfTESTrefreshtoken456...",
    "user": {
      "firstName": "Test",
      "lastName": "User",
      "email": "testuser@example.com"
    }
  }
}
```


**5.2 Cross-Origin (React / Next.js)**

React tarafında cookie bazlı oturum açılacaksa:

- HttpOnly = true
- Secure = true
- SameSite = None

Bu durumda JWT tarayıcıda gözükmez (XSS güvenliği).
İsteklerde otomatik olarak cookie gönderilir (withCredentials: true).

**5.3 Header ile Token Gönderimi**

Cookie kullanmak istemeyenler için alternatif.

# 6.Interface Metotları — Kısa Açıklamalar

| Metot | Açıklama |
|---|---|
| **Task<string> GenerateAccessTokenAsync(TUser user, Func<TUser, IEnumerable<Claim>>? extraClaims = null)** | Verilen kullanıcıya ait JWT Access Token üretir. Opsiyonel olarak ekstra claim eklenebilir. |
| **string GenerateRefreshToken()** | Rastgele 64 byte uzunluğunda bir Refresh Token oluşturur (Base64 formatında). |
| **ClaimsPrincipal GetPrincipalFromExpiredToken(string token)** | Süresi dolmuş bir token'dan kullanıcı bilgilerini (claim'leri) çıkarır. Refresh işlemleri için kullanılır. |
| **Task<bool> ValidateRefreshTokenAsync(TUser user, string refreshToken)** | Verilen refresh token'ın kullanıcıya ait olup olmadığını ve süresinin dolmadığını doğrular. |

| | |
|---|---|
| **Task SaveRefreshTokenAsync(TUser user, string refreshToken)** | Kullanıcının refresh token'ını ve son kullanma tarihini veritabanına kaydeder. |
| **Task RevokeRefreshTokenAsync(TUser user)** | Kullanıcının refresh token'ını sıfırlar (logout işlemi sırasında kullanılır). |
| **string GetUserIdFromToken(string token)** | Token içindeki NameIdentifier (veya sub) claim'inden kullanıcı ID'sini döndürür. |
| **bool IsTokenValid(string token)** | Token'ın geçerli olup olmadığını (imza, süre, issuer, audience gibi) doğrular. |
| **string? GetTokenFromRequest(HttpRequest request)** | Gelen istekteki token'ı şu sırayla arar: Authorization Header, Cookie, Query Param. |
| **void SetAccessTokenCookie(HttpResponse response, string token, int expiryMinutes)** | Access token'ı cookie'ye yazar. Expiry süresi dakikadır. XSS'e karşı HttpOnly kullanılabilir. |
| **void SetRefreshTokenCookie(HttpResponse response, string refreshToken, int expiryDays)** | Refresh token'ı cookie'ye yazar. Expiry süresi gündür. Tarayıcı oturumları için idealdir. |
| **string? GetTokenFromCookie(HttpRequest request, string cookieName)** | Cookie içinden Access veya Refresh token'ı okur. |
| **void ClearTokenCookies(HttpResponse response)** | Access ve Refresh token cookie'lerini geçersiz hale getirir (logout sonrası çağrılır). |
| **Task<JwtTokenResponse<TUserInfo>> CreateTokenResponseAsync(...)** | Access ve Refresh token'ları birleştirerek istemciye gönderilecek yanıt nesnesini (JwtTokenResponse) oluşturur. |

Bu methodlardan sonra projenize entegre ettiğiniz paketi dilerseniz servic yapınızda dilerseniz controllerda çağırarak  tüm Jwt işlemleri içeren sistemlerde kullanabilirsiniz.

## 7.Sonuç ve Kapanış

Bu kütüphaneyi, her projede aynı JWT yapısını baştan kurmaktan sıkıldığım için yazdım. Amacım "herkesin anlayabileceği kadar sade" ama "profesyonel bir yapıya sahip" bir sistem kurmaktı.

Yunus.JwtKit, bir token servisinden fazlası —
tekrar etmeyen, anlaşılır ve genişletilebilir bir altyapı felsefesidir.

Kullan, incele ve eksik  hatalı bir şey varsa her zaman kendimi geliştirmek için geri dönütlere açığım.

Projelerinizde kullanmanız dileğiyle.