JWT – IDENTİTY DERS NOTLARI

✅ **Konu Tanımı**  
ASP.NET Core Identity, bir web uygulamasında **kullanıcı yönetimi** (login, register, logout, role, password reset vb.) işlemlerini kolayca yapmamızı sağlayan **hazır bir üyelik sistemi**dir.

🧠 **Mantığı**  
Her projede kullanıcı kaydı, şifre doğrulama, roller, oturum yönetimi gibi tekrar eden ihtiyaçlar vardır. Bunları sıfırdan yazmak hem güvenlik riski doğurur hem de zaman kaybettirir. Identity, Microsoft tarafından sağlanan **güvenli ve esnek bir çözüm** sunar.

📍 **Nerede ve Nasıl Kullanılır?**

* Kullanıcı girişi olan web uygulamalarında (e-ticaret, blog, yönetim panelleri).
* Rollerle yetkilendirme yapılması gerektiğinde (admin, user).
* Güvenli şifreleme, cookie-based authentication gibi standart çözümler gerektiğinde.

Alternatifler:

* **Custom Authentication** (elle yazılmış giriş sistemleri).
* **Third-party providers** (Google, Facebook login).
* **JWT Authentication** (özellikle API’lerde).

📌 **Günlük Hayattan Basit Bir Örnek**  
Bir **spor salonu** düşün: Her üyenin bir kartı (kullanıcı hesabı) var. Kartla giriş yapar, yetkisine göre bazı alanlara girer (ör. “VIP salon”). Identity, işte bu kartların verilmesini, doğrulanmasını ve yetkilendirilmesini organize eden sistemdir.

💻 **C# Kod Örneği**

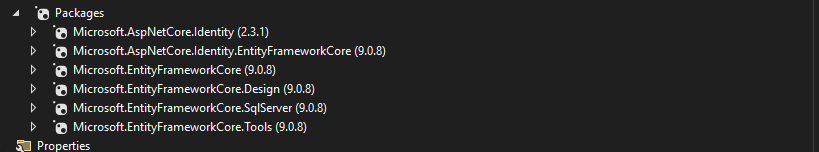


🔁 **Alternatifler ve Kıyaslama**

* **Identity**: Hazır, güvenli, .NET ekosistemine tam uyumlu.
* **Custom Authentication**: Daha esnek ama güvenlik açıklarına açık.
* **JWT**: API’lerde token bazlı kimlik doğrulama için daha uygun.
* **OAuth / OpenID**: Google, Facebook gibi üçüncü taraf servislerle giriş için kullanılır.

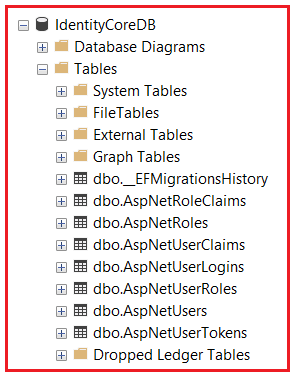
📚 **Terim Sözlüğü**

| **İngilizce Terim** | **Türkçe Açıklama** |
| --- | --- |
| Identity | Üyelik ve kimlik doğrulama sistemi |
| Authentication | Kimlik doğrulama |
| Authorization | Yetkilendirme |
| Role | Rol, yetki grubu |
| IdentityUser | Varsayılan kullanıcı sınıfı |
| IdentityRole | Rol sınıfı |
| Cookie-based auth | Çerez tabanlı kimlik doğrulama |
| JWT (JSON Web Token) | Token tabanlı kimlik doğrulama |
|  |  |



Örnek paketler

Tablolar Kolonlar ve işlevleri



Asp.net users :

✅ **Konu Tanımı**  
AspNetUsers tablosu, uygulamanızdaki tüm kullanıcıların temel bilgilerini saklar. Bu tablo; kullanıcıların benzersiz kimliklerini, giriş bilgilerini, güvenlik ayarlarını ve iletişim bilgilerini içerir. Identity sisteminde kullanıcıyla ilgili tüm verilerin **birincil kaynağı**dır.

🧠 **Mantığı**  
Kullanıcılarla ilgili bilgiler merkezi bir tabloda tutulursa; yönetim, güvenlik ve yetkilendirme işlemleri daha kolay ve güvenli hâle gelir. Tüm Identity tabloları bu ana tabloya bağlı olarak çalışır.

📍 **Nerede ve Nasıl Kullanılır?**

* Kullanıcı kayıt, giriş, şifre sıfırlama gibi işlemlerde.
* Rol ve yetki atamalarında (AspNetUserRoles).
* Kullanıcı hakları ve özel izinlerde (AspNetUserClaims).
* Harici giriş sağlayıcılarıyla ilişkilerde (AspNetUserLogins).
* Token yönetiminde (AspNetUserTokens).

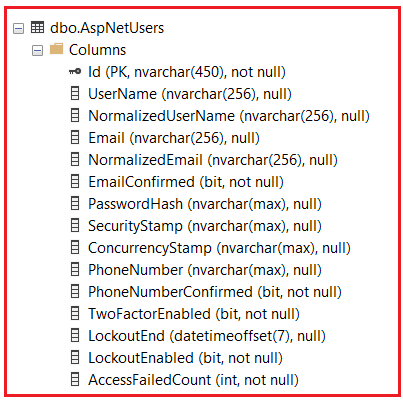
📌 **İlişkiler (Relationships)**

* Her kullanıcı birden fazla role sahip olabilir → AspNetUserRoles.
* Her kullanıcı birden fazla claim’e sahip olabilir → AspNetUserClaims.
* Her kullanıcı birden fazla harici login’e sahip olabilir → AspNetUserLogins.
* Her kullanıcı birden fazla token’a sahip olabilir → AspNetUserTokens.

Bu ilişkilerde AspNetUsers.Id sütunu, diğer tablolar için **foreign key** görevi görür.

📌 **Günlük Hayattan Basit Bir Örnek**  
Bir spor salonu düşün:

* AspNetUsers → Üyeler (Ali, Ayşe, Mehmet)
* AspNetUserRoles → Hangi üyenin VIP veya Standart üye olduğu
* AspNetUserClaims → Üyeye özel izinler, örn. Sauna kullanabilir
* AspNetUserLogins → Üyenin Google veya Facebook ile giriş bilgisi
* AspNetUserTokens → Üyenin mobil uygulama oturum anahtarları



| **Sütun Adı** | **Türkçe Açıklama** |
| --- | --- |
| **Id** | Her kullanıcı için benzersiz kimlik değeri (primary key). |
| **UserName** | Kullanıcının giriş (login) adı. |
| **NormalizedUserName** | Büyük harfe çevrilmiş UserName, büyük/küçük harf duyarsız aramalar için. |
| **Email** | Kullanıcının e-posta adresi. |
| **NormalizedEmail** | Büyük harfe çevrilmiş Email, büyük/küçük harf duyarsız aramalar için. |
| **EmailConfirmed** | Kullanıcının e-postasının doğrulanıp doğrulanmadığını gösterir. |
| **PasswordHash** | Kullanıcının şifresinin hashlenmiş hâli (asla düz metin olarak saklanmaz). |
| **SecurityStamp** | Kullanıcının güvenlikle ilgili bilgileri değiştiğinde rastgele değişen değer. Örn: Şifre değişince veya resetlenince değişir. Eski oturumların geçersiz kalmasını sağlar. |
| **ConcurrencyStamp** | Kullanıcı profili güncellendiğinde değişen benzersiz değer. Optimistic concurrency control için kullanılır. Aynı kaydı aynı anda iki farklı işlem güncellemek istediğinde veri tutarlılığını korur. |
| **PhoneNumber** | Kullanıcının telefon numarası. |
| **PhoneNumberConfirmed** | Telefon numarasının doğrulanıp doğrulanmadığını gösterir. |
| **TwoFactorEnabled** | Kullanıcıda iki aşamalı doğrulamanın (2FA) aktif olup olmadığını gösterir. |
| **LockoutEnd** | Başarısız giriş denemelerinden sonra hesabın kilitli kalacağı son tarih/saat. |
| **LockoutEnabled** | Kullanıcı için hesap kilitleme özelliğinin aktif olup olmadığını gösterir. |
| **AccessFailedCount** | Başarısız giriş denemelerinin sayısını tutar. Gerekirse lockout tetiklenir. |

**AspNetRoles:**

**✅ Konu Tanımı  
AspNetRoles tablosu, uygulamadaki tüm tanımlı rolleri saklar. Roller, kullanıcıya atanabilen farklı grup veya yetki seviyelerini temsil eder (ör. Admin, Moderator, User).**

**🧠 Mantığı  
Kullanıcılara roller atamak, yetkilendirme işlemlerini standart ve merkezi bir şekilde yönetmeyi sağlar. Her rolün belirli izinleri (claims) olabilir ve kullanıcılar bu rollere göre yetkilendirilir.**

**📍 Nerede ve Nasıl Kullanılır?**

* **Kullanıcıya rol atamak için (AspNetUserRoles).**
* **Role özel hak/izin atamak için (AspNetRoleClaims).**
* **Çoklu kullanıcı-rol ilişkilerinde tercih edilir.**

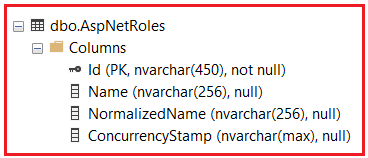
**📌 İlişkiler (Relationships)**

* **AspNetRoles ↔ AspNetUserRoles: Kullanıcılara rollerin atanması.**
* **AspNetRoles ↔ AspNetRoleClaims: Rollere hak/izinlerin atanması.**

**📌 Günlük Hayattan Basit Bir Örnek  
Bir spor salonu düşün:**

* **Roller: Admin, Eğitmen, Üye**
* **Admin: Tüm sistem ayarlarını yönetebilir**
* **Eğitmen: Sadece ders programlarını yönetebilir**
* **Üye: Sadece kendi profilini görebilir**

**Bu roller AspNetRoles tablosunda tutulur ve her kullanıcıya uygun rol AspNetUserRoles üzerinden atanır.**

****

| **Sütun Adı** | **Türkçe Açıklama** |
| --- | --- |
| **Id** | Her rol için benzersiz kimlik değeri (genellikle GUID). |
| **Name** | Rolün okunabilir adı (örnek: “Admin”). |
| **NormalizedName** | Rol adının büyük harfe çevrilmiş hâli, büyük/küçük harf duyarsız eşleştirme için. |
| **ConcurrencyStamp** | Rol güncellendiğinde değişen değer. Optimistic concurrency control için kullanılır; aynı rolü aynı anda birden fazla işlem güncellerse veri tutarlılığını sağlar. |

**AspNetUserRoles:**

**✅ Konu Tanımı  
AspNetUserRoles tablosu, kullanıcılar ile roller arasındaki çoktan-çoğa (many-to-many) ilişkiyi temsil eder. Bu tablo, kullanıcıları uygun rollere bağlayarak rol tabanlı yetkilendirme sağlar.**

**🧠 Mantığı  
Her kullanıcı birden fazla role sahip olabilir ve her rol de birden fazla kullanıcıya atanabilir. AspNetUserRoles tablosu, bu ilişkiyi yönetmek için kullanılır ve sistemin rol bazlı yetkilendirme yapabilmesini sağlar.**

**📍 Nerede ve Nasıl Kullanılır?**

* **Kullanıcılara roller atamak için.**
* **Bir kullanıcının sahip olduğu tüm rolleri sorgulamak için.**
* **Bir rolün tüm kullanıcılarını listelemek için.**

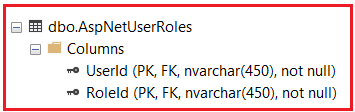
**📌 İlişkiler (Relationships)**

* **AspNetUsers ↔ AspNetUserRoles: Kullanıcı-rol bağlantısı.**
* **AspNetRoles ↔ AspNetUserRoles: Rol-kullanıcı bağlantısı.**

**📌 Günlük Hayattan Basit Bir Örnek  
Bir spor salonu düşün:**

* **Kullanıcı Ali: Roller = Admin, Eğitmen**
* **Kullanıcı Ayşe: Roller = Üye**
* **Kullanıcı Mehmet: Roller = Eğitmen**

**AspNetUserRoles tablosu bu bilgileri tutar, böylece sistem “Ali Admin yetkisine sahip mi?” veya “Eğitmen rolüne sahip tüm kullanıcılar kimler?” sorularını kolayca yanıtlayabilir.**

****

|  |  |
| --- | --- |
| **UserId** | AspNetUsers tablosundaki **Id** sütununa karşılık gelir. Composite primary key’in (birleşik birincil anahtar) bir parçasıdır. Kullanıcıyı temsil eder ve AspNetUsers tablosuna **foreign key** (yabancı anahtar) bağlantısı kurar. |
| **RoleId** | AspNetRoles tablosundaki **Id** sütununa karşılık gelir. Composite primary key’in diğer parçasıdır. Rolü temsil eder ve AspNetRoles tablosuna **foreign key** bağlantısı kurar. |

**AspNetUserLogins:**

**✅ Konu Tanımı  
AspNetUserLogins tablosu, kullanıcıların giriş yapmak için kullandığı harici kimlik doğrulama sağlayıcıları (Google, Facebook, Microsoft vb.) hakkında bilgi saklar. Bu tablo, harici girişleri yerel kullanıcı hesaplarıyla ilişkilendirir.**

**🧠 Mantığı  
Kullanıcıların sosyal hesaplarıyla giriş yapabilmesini sağlar. Her harici giriş, sistemdeki bir kullanıcıyla eşleştirilir ve böylece aynı kullanıcı farklı sağlayıcılarla da oturum açabilir.**

**📍 Nerede ve Nasıl Kullanılır?**

* **Sosyal girişlerin (Google, Facebook, Microsoft vb.) yönetiminde.**
* **Kullanıcının hangi harici sağlayıcı ile giriş yapacağını belirlemede.**
* **Kullanıcının tüm harici loginlerini listelemede.**

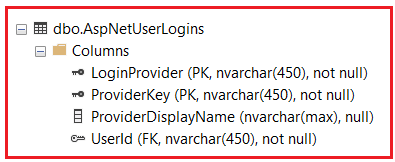
**📌 İlişkiler (Relationships)**

* **AspNetUsers ↔ AspNetUserLogins: Kullanıcıyı harici login bilgisiyle bağlar.**
* **Her kayıt, bir kullanıcı (UserId) ile bir harici kimlik doğrulama sağlayıcısını eşleştirir.**

**📌 Günlük Hayattan Basit Bir Örnek  
Bir spor salonu uygulaması düşün:**

* **Kullanıcı Ali: Google hesabıyla giriş yapıyor**
* **Kullanıcı Ayşe: Facebook hesabıyla giriş yapıyor**
* **Kullanıcı Mehmet: Hem Google hem Microsoft hesabıyla giriş yapıyor**

**AspNetUserLogins tablosu bu bilgileri tutar, böylece sistem “Ali Google hesabıyla oturum açabilir mi?” sorusunu kolayca yanıtlayabilir.**

****

|  |  |
| --- | --- |
| **LoginProvider** | Harici kimlik doğrulama sağlayıcısının adı (ör. Google, Facebook, Microsoft vb.). Tablo için composite primary key’in bir parçasıdır. |
| **ProviderKey** | Harici sağlayıcının kullanıcıya verdiği benzersiz kimlik değeri. Örneğin Google ile girişte Google’ın atadığı unique ID burada saklanır. Composite primary key’in diğer parçasıdır. |
| **ProviderDisplayName** | Harici sağlayıcının kullanıcıya gösterilecek adı (ör. “Google” → “google.com” yerine). Daha çok UI (arayüz) tarafında gösterim için kullanılır. |
| **UserId** | AspNetUsers tablosundaki **Id** sütununa karşılık gelir. Bu harici girişin hangi yerel kullanıcıya ait olduğunu belirtir. Foreign key ile AspNetUsers tablosuna bağlanır. |

**AspNetUserTokens:**

**✅ Konu Tanımı  
AspNetUserTokens tablosu, kullanıcılar için oluşturulan token’ları saklar. Bu token’lar; şifre sıfırlama, e-posta doğrulama veya iki faktörlü kimlik doğrulama gibi çeşitli amaçlarda kullanılır.**

**🧠 Mantığı  
Token’lar, güvenlik gerektiren işlemleri geçici ve güvenli bir şekilde yönetmek için kullanılır. Her token, bir kullanıcıyla ilişkilidir ve genellikle tek kullanımlık veya sınırlı süreli güvenlik işlemleri için oluşturulur.**

**📍 Nerede ve Nasıl Kullanılır?**

* **Şifre sıfırlama süreçlerinde.**
* **E-posta doğrulama linklerinde.**
* **İki faktörlü kimlik doğrulama (2FA) işlemlerinde.**
* **Diğer güvenlik veya workflow adımlarında tek seferlik erişim sağlamak için.**

**📌 İlişkiler (Relationships)**

* **Her token bir kullanıcıya bağlıdır (UserId → AspNetUsers.Id).**
* **Token kayıtları, güvenlik veya workflow işlemlerinde kullanılır.**

**📌 Günlük Hayattan Basit Bir Örnek  
Bir spor salonu uygulaması düşün:**

* **Ali, “Şifremi unuttum” diyerek şifre sıfırlama talebi oluşturuyor.**
* **Sistem, Ali’ye bir tek kullanımlık token gönderiyor (AspNetUserTokens).**
* **Ali token’ı kullanarak yeni şifresini oluşturuyor.**

**Benzer şekilde, email doğrulama veya 2FA işlemleri de bu tabloda saklanan token’lar üzerinden yürütülür.**

****

|  |  |
| --- | --- |
| **UserId** | **AspNetUsers tablosundaki kullanıcı ID’si. Token’ın hangi kullanıcıya ait olduğunu belirtir.** |
| **LoginProvider** | **Token’ı oluşturan sağlayıcının adı (ör. Email, Google, Facebook vb.).** |
| **Name** | **Token’ın türünü belirten ad. Örneğin: PasswordReset (şifre sıfırlama), EmailConfirmation (e-posta doğrulama), AccessToken (erişim token’ı) veya TwoFactorAuth gibi.** |
| **Value** | **Token’ın değeri. Örneğin şifre sıfırlama için oluşturulan rastgele string.** |

**AspNetUserClaims:**

**✅ Konu Tanımı  
AspNetUserTokens tablosu, kullanıcılar için oluşturulan token’ları saklar. Bu token’lar; şifre sıfırlama, e-posta doğrulama veya iki faktörlü kimlik doğrulama gibi çeşitli amaçlarda kullanılır.**

**🧠 Mantığı  
Token’lar, güvenlik gerektiren işlemleri geçici ve güvenli bir şekilde yönetmek için kullanılır. Her token, bir kullanıcıyla ilişkilidir ve genellikle tek kullanımlık veya sınırlı süreli güvenlik işlemleri için oluşturulur.**

**📍 Nerede ve Nasıl Kullanılır?**

* **Şifre sıfırlama süreçlerinde.**
* **E-posta doğrulama linklerinde.**
* **İki faktörlü kimlik doğrulama (2FA) işlemlerinde.**
* **Diğer güvenlik veya workflow adımlarında tek seferlik erişim sağlamak için.**

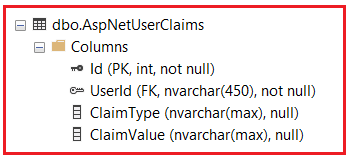
**📌 İlişkiler (Relationships)**

* **Her token bir kullanıcıya bağlıdır (UserId → AspNetUsers.Id).**
* **Token kayıtları, güvenlik veya workflow işlemlerinde kullanılır.**

**📌 Günlük Hayattan Basit Bir Örnek  
Bir spor salonu uygulaması düşün:**

* **Ali, “Şifremi unuttum” diyerek şifre sıfırlama talebi oluşturuyor.**
* **Sistem, Ali’ye bir tek kullanımlık token gönderiyor (AspNetUserTokens).**
* **Ali token’ı kullanarak yeni şifresini oluşturuyor.**

**Benzer şekilde, email doğrulama veya 2FA işlemleri de bu tabloda saklanan token’lar üzerinden yürütülür.**

****

|  |  |
| --- | --- |
| **Id** | **Kullanıcı claim’i için birincil anahtar (primary key). Tam sayı (int) tipindedir.** |
| **UserId** | **Claim’in bağlı olduğu kullanıcının ID’si. AspNetUsers tablosuna foreign key ile bağlanır.** |
| **ClaimType** | **Claim’in türü (ör. “birthdate” – doğum tarihi, “role” – rol vb.).** |
| **ClaimValue** | **Claim’in değeri (ör. “1980-01-01”, “Admin”).** |

**AspNetRoleClaims:**

**✅ Konu Tanımı  
AspNetRoleClaims tablosu, rollere ait claim’leri saklar. Bu claim’ler, ilgili role sahip tüm kullanıcılar için geçerlidir ve rol bazlı izin/claim yönetimi sağlar.**

**🧠 Mantığı  
Bir role ait claim’ler, rolün tüm kullanıcılarına otomatik olarak uygulanır. Böylece aynı izinleri birden fazla kullanıcıya tek tek atamak yerine, role tanımlamak yeterli olur ve yönetim kolaylaşır.**

**📍 Nerede ve Nasıl Kullanılır?**

* **Rol bazlı izin atamalarında.**
* **Bir rolün sahip olduğu tüm hakları tanımlamak için.**
* **Kullanıcılara otomatik olarak rol üzerinden izin vermek için.**

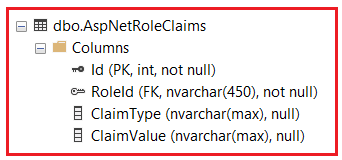
**📌 İlişkiler (Relationships)**

* **Her claim bir role bağlıdır (RoleId → AspNetRoles.Id).**
* **Role sahip tüm kullanıcılar bu claim’leri miras alır.**

**📌 Günlük Hayattan Basit Bir Örnek  
Bir spor salonu uygulaması düşün:**

* **Rol: Eğitmen**
* **ClaimType = CanEditSchedule, ClaimValue = True**
* **Rol: Üye**
* **ClaimType = CanAccessGym, ClaimValue = True**

**Bu sayede “Eğitmen” rolüne sahip tüm kullanıcılar ders programını düzenleyebilir, “Üye” rolüne sahip tüm kullanıcılar ise spor salonuna erişebilir.**

****

|  |  |
| --- | --- |
| **Id** | Rol claim’i için benzersiz birincil anahtar (integer). |
| **RoleId** | Claim’in bağlı olduğu rolün ID’si. AspNetRoles tablosuna foreign key ile bağlanır. |
| **ClaimType** | Claim’in türünü tutar. Örnekler: Permission (izin), AccessLevel (erişim seviyesi) veya uygulama bağlamına uygun diğer tipler. |
| **ClaimValue** | Claim’in değeri. Örneğin ClaimType = Permission ise ClaimValue = Edit\_User veya View\_Reports gibi. |

public class AppUser :IdentityUser

{

public string Name { get; set; }

public string Surname { get; set; }

public string? ImageUrl { get; set; }

public string? City { get; set; }

}

Ekle böylelikle

Id = table.Column<string>(type: "nvarchar(450)", nullable: false),

Name = table.Column<string>(type: "nvarchar(max)", nullable: false),

Surname = table.Column<string>(type: "nvarchar(max)", nullable: false),

ImageUrl = table.Column<string>(type: "nvarchar(max)", nullable: true),

City = table.Column<string>(type: "nvarchar(max)", nullable: true),

UserName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),

NormalizedUserName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),

Email = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),

NormalizedEmail = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),

EmailConfirmed = table.Column<bool>(type: "bit", nullable: false),

PasswordHash = table.Column<string>(type: "nvarchar(max)", nullable: true),

SecurityStamp = table.Column<string>(type: "nvarchar(max)", nullable: true),

ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true),

PhoneNumber = table.Column<string>(type: "nvarchar(max)", nullable: true),

PhoneNumberConfirmed = table.Column<bool>(type: "bit", nullable: false),

TwoFactorEnabled = table.Column<bool>(type: "bit", nullable: false),

LockoutEnd = table.Column<DateTimeOffset>(type: "datetimeoffset", nullable: true),

LockoutEnabled = table.Column<bool>(type: "bit", nullable: false),

AccessFailedCount = table.Column<int>(type: "int", nullable: false)

Tabloda istediğin verileri ıdentity tablomuza ekliyoruz.

FAYDALI BİLGİLER SERİSİ

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using NotikaIdentityEmail.Entities;

using NotikaIdentityEmail.Models;

using System.Threading.Tasks;

namespace NotikaIdentityEmail.Controllers

{

public class RegisterController : Controller

{

private readonly UserManager<AppUser> \_userManager;

//bunun sayesinde gizli sadece okunabilir usermanager ıdentityu kütüphanesi ile gelir içine app user tanımlı sınıfımız geldi ve buradan field yani erişim kullanacağımız değişken ürettik

// bu ise yapıcı metottur yani örnek veriyoruz ilk veriyi hazırlar mesela başladık neye başladık bir öğrenci verdik ad soyad numara burası ctor ile yani consturctor ad baran soyad daşdemir numara 111 numarası atadı

public RegisterController(UserManager<AppUser> userManager)

{

\_userManager = userManager;

}

//bu yapının tamamı dependenjy injection oldu kahve mantığı gibi düşün kahve makinesinin türk kahvesi öğütücüsü koyarsan türk kahvesi olur espresso öğütücü koyarsan espresso olur işte bu öğütücü dependecy injectiondur

//bunların hepsi asenkron çalışırsa olur senkron kodu yukarıdan aşağıya okur ama asenkron ile işlem yapılırken diğer işlemlerde yapılır örnek senkron garson sipariş alır başka işlem yapmaz asenkron ise garson sipariş alır başka müşteridende şipariş alır

[HttpGet]

public IActionResult CreateUser()

{

return View();

}

//bir metodun birden fazla aynı isimde farklı şekillerde çağırılmasına overloading denir burası overloadingtir

[HttpPost]

public async Task<IActionResult> CreateUser(RegisterUserViewModel model)

{

AppUser appUser = new AppUser()

{

Name = model.Name,

Surname = model.Surname,

Email=model.Email,

UserName = model.Username

};

var result = await \_userManager.CreateAsync(appUser,model.Pasword);// appuser türünde parametre istiyor // burada oluşturma işlemi yaptık

if (result.Succeeded)// bu işlemler başarılı olursa

{

return RedirectToAction("UserLogin", "Login"); //buraya gönder

}

else //başarısız

{

foreach (var item in result.Errors) // eğer başarırsız olursa hataları döndür errordan gelen hataları

{

ModelState.AddModelError("", item.Description);// key ve mesaj

}

}

return View();

}

//tasklar geri dönüş yapar await ile çalışır ve uygulamayı dondurmaz kodun başka işlemleri engellemeden yapmasını sağlar

// örnek vermek gerekirse bilgisayarda 2 işlem yaparsın 1ini arkaya atarsın burada task olur

//await işe uygulamanın donmamasını sağlar

}

}

builder.Services.AddDbContext<EmailContext>();

builder.Services.AddIdentity<AppUser, IdentityRole>().AddEntityFrameworkStores<EmailContext>();

//bağımlılıkları enjeksiyon etmemiz gerekir yoksa hata verir

namespace NotikaIdentityEmail.Models

{

public class RegisterUserViewModel //ihtiyacımız olan sınıfları çağırmak için bir nevi dto

{

public string Name { get; set; }

public string Surname { get; set; }

public string Username { get; set; }

public string Email { get; set; }

public string Password { get; set; }

public string ConfirmPassword { get; set; }

}

}

Peki sistemi türkçeleştirme hata mesajlarını nasıl yaparız

using Microsoft.AspNetCore.Identity;

namespace NotikaIdentityEmail.Models

{

public class CustomIdentityValidator : IdentityErrorDescriber

{

public override IdentityError PasswordTooShort(int length)

{

return new IdentityError()

{

Code = "PasswordTooShort",

Description = $"Şifreniz {length} karakter içermelidir!"

};

}

public override IdentityError PasswordRequiresLower()

{

return new IdentityError()

{

Code = "PasswordRequiresLower",

Description = "Şifreniz En az 1 Tane küçük Harf içermelidir."

};

}

public override IdentityError PasswordRequiresUpper()

{

return new IdentityError()

{

Code = "PasswordRequiresUpper",

Description = "Şifreniz En az 1 Tane Büyük Harf içermelidir."

};

}

public override IdentityError PasswordRequiresDigit()

{

return new IdentityError()

{

Code = "PasswordRequiresDigit",

Description = "Şifreniz En az 1 Tane Rakam içermelidir."

};

}

public override IdentityError PasswordRequiresNonAlphanumeric()

{

return new IdentityError

{

Code = "PasswordRequiresNonAlphanumeric",

Description = "Şifreniz En az 1 Tane sembol içermelidir."

};

}

public override IdentityError DuplicateUserName(string userName)

{

return new IdentityError()

{

Code = "DuplicateUserName",

Description = $"{userName} adlı kullanıcı adı sistemimizde mevcut"

};

}

}

}

builder.Services.AddIdentity<AppUser, IdentityRole>().AddEntityFrameworkStores<EmailContext>().AddErrorDescriber<CustomIdentityValidator>();

ve ekleme yapıyoruz

<**div** class="text-danger" **asp-validation-summary**="ModelOnly"></**div**>

VİEWDA BU ÖNEMLİ

Tüm bu işler register içindi peki login?

private readonly SignInManager<AppUser> \_signInManager;

public LoginController(SignInManager<AppUser> signInManager)

{

\_signInManager = signInManager;

}

[HttpGet]

public IActionResult UserLogin()

{

return View();

}

[HttpPost]

public async Task<IActionResult> UserLogin(UserLoginViewModel model)

{

var result = await \_signInManager.PasswordSignInAsync(model.Username, model.Password, true, true);

if (result.Succeeded)

{

return RedirectToAction("MyProfile", "Profile");

}

else

{

ModelState.AddModelError("", "Kullanıcı adı veya şifre hatalı");

return View();

}

//return View();

}

@model UserLoginViewModel

@{

Layout = null;

}

<!DOCTYPE html>

<html lang="en">

<head>

<title>Giriş Sayfası</title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<!--===============================================================================================-->

<**link** rel="icon" type="image/png" href="~/Login\_v4/images/icons/favicon.ico" />

<!--===============================================================================================-->

<**link** rel="stylesheet" type="text/css" href="~/Login\_v4/vendor/bootstrap/css/bootstrap.min.css">

<!--===============================================================================================-->

<**link** rel="stylesheet" type="text/css" href="~/Login\_v4/fonts/font-awesome-4.7.0/css/font-awesome.min.css">

<!--===============================================================================================-->

<**link** rel="stylesheet" type="text/css" href="~/Login\_v4/fonts/iconic/css/material-design-iconic-font.min.css">

<!--===============================================================================================-->

<**link** rel="stylesheet" type="text/css" href="~/Login\_v4/vendor/animate/animate.css">

<!--===============================================================================================-->

<**link** rel="stylesheet" type="text/css" href="~/Login\_v4/vendor/css-hamburgers/hamburgers.min.css">

<!--===============================================================================================-->

<**link** rel="stylesheet" type="text/css" href="~/Login\_v4/vendor/animsition/css/animsition.min.css">

<!--===============================================================================================-->

<**link** rel="stylesheet" type="text/css" href="~/Login\_v4/vendor/select2/select2.min.css">

<!--===============================================================================================-->

<**link** rel="stylesheet" type="text/css" href="~/Login\_v4/vendor/daterangepicker/daterangepicker.css">

<!--===============================================================================================-->

<**link** rel="stylesheet" type="text/css" href="~/Login\_v4/css/util.css">

<**link** rel="stylesheet" type="text/css" href="~/Login\_v4/css/main.css">

<!--===============================================================================================-->

</head>

<body>

<div class="limiter">

<div class="container-login100" style="background-image: url('/Login\_v4/images/bg-01.jpg');">

<div class="wrap-login100 p-l-55 p-r-55 p-t-65 p-b-54">

<**form** class="login100-form validate-form" method="post">

<span class="login100-form-title p-b-49">

Giriş Yap

</span>

<div class="wrap-input100 validate-input m-b-23" data-validate="Kullanıcı Adı zorunlu">

<span class="label-input100">Kullanıcı Adınız</span>

<**input** class="input100" **type**="text" **asp-for**="Username" placeholder="Kullanıcı adı">

<span class="focus-input100" data-symbol="&#xf206;"></span>

</div>

<div class="wrap-input100 validate-input" data-validate="Şifre zorunlu">

<span class="label-input100">Şifreniz</span>

<**input** class="input100" **type**="password" **asp-for**="Password" placeholder="Şifre">

<span class="focus-input100" data-symbol="&#xf190;"></span>

</div>

<div class="text-right p-t-8 p-b-31">

<a href="/UserPassword/ForgotMyPassword/">

Şifremi Unuttum?

</a>

</div>

<**div** class="text-danger" **asp-validation-summary**="ModelOnly"></**div**>

<div class="container-login100-form-btn">

<div class="wrap-login100-form-btn">

<div class="login100-form-bgbtn"></div>

<button class="login100-form-btn">

Giriş Yap

</button>

</div>

</div>

<div class="txt1 text-center p-t-54 p-b-20">

<span>

Veya Bunları dene!

</span>

</div>

<div class="flex-c-m">

<a href="#" class="login100-social-item bg1">

<i class="fa fa-facebook"></i>

</a>

<a href="#" class="login100-social-item bg2">

<i class="fa fa-twitter"></i>

</a>

<a href="#" class="login100-social-item bg3">

<i class="fa fa-google"></i>

</a>

</div>

<div class="flex-col-c p-t-155">

<span class="txt1 p-b-17">

Bir hesabın yokmu?

</span>

<a href="/Register/CreateUser/" class="txt2">

Kayıt ol

</a>

</div>

</**form**>

</div>

</div>

</div>

<div id="dropDownSelect1"></div>

<!--===============================================================================================-->

<**script** src="~/Login\_v4/vendor/jquery/jquery-3.2.1.min.js"></**script**>

<!--===============================================================================================-->

<**script** src="~/Login\_v4/vendor/animsition/js/animsition.min.js"></**script**>

<!--===============================================================================================-->

<**script** src="~/Login\_v4/vendor/bootstrap/js/popper.js"></**script**>

<**script** src="~/Login\_v4/vendor/bootstrap/js/bootstrap.min.js"></**script**>

<!--===============================================================================================-->

<**script** src="~/Login\_v4/vendor/select2/select2.min.js"></**script**>

<!--===============================================================================================-->

<**script** src="~/Login\_v4/vendor/daterangepicker/moment.min.js"></**script**>

<**script** src="~/Login\_v4/vendor/daterangepicker/daterangepicker.js"></**script**>

<!--===============================================================================================-->

<**script** src="~/Login\_v4/vendor/countdowntime/countdowntime.js"></**script**>

<!--===============================================================================================-->

<**script** src="~/Login\_v4/js/main.js"></**script**>

</body>

</html>

Mail işlemleri

//random sınıfından random bir nesne oluşturduk

//random sınıfı random sayılar üretir

int activationCode = random.Next(100000, 999999);

//random sınıfından random bir nesne oluşturduk ve 100000 ile 999999 arasında rastgele bir sayı ürettik

AppUser appUser = new AppUser()

{

Name = model.Name,

Surname = model.Surname,

Email = model.Email,

UserName = model.Username,

ActivationCode = activationCode,

};

var result = await \_userManager.CreateAsync(appUser, model.Password);// appuser türünde parametre istiyor // burada oluşturma işlemi yaptık

if (result.Succeeded)// bu işlemler başarılı olursa

{

//kullanıcıyı aktif etme işlemi için activation kodu gönderilecek kurzpwyuuaybvfzf

MimeMessage mimeMessage = new MimeMessage();

MailboxAddress mailboxAddressFrom = new MailboxAddress("NotikaAdminChat", "ebtuyou1@gmail.com");

mimeMessage.From.Add(mailboxAddressFrom); // kimden geldiğini belirtiyoruz yani kullanıcıya gözükecek mail

MailboxAddress mailboxAddressTo = new MailboxAddress("User", model.Email);//bir kullanıcı ve modeldeki email adresi

mimeMessage.To.Add(mailboxAddressTo); // kime gideceğini belirtiyoruz yani kullanıcıya gidecek mail

var bodybuilder = new BodyBuilder();

bodybuilder.TextBody = $"Merhaba {model.Name} {model.Surname},\n\n" +

$"Hesabınızı aktifleştirmek için lütfen aşağıdaki kodu kullanın:\n\n" +

$"{activationCode}\n\n" +

"NotikaAdminChat Ekibi" + $"Hello {model.Name} {model.Surname},\n\n" +

$"Please use the following code to activate your account:\n\n" +

$"{activationCode}\n\n" +

"NotikaAdminChat Team";

mimeMessage.Body = bodybuilder.ToMessageBody(); // mailin içeriğini belirtiyoruz

mimeMessage.Subject = "Notika Hesap Aktivasyon Kodu - Notika Account Activation Code "; // mailin konusunu belirtiyoruz

SmtpClient smtpClient = new SmtpClient();

smtpClient.Connect("smtp.gmail.com", 587,false); // gmail smtp sunucusuna bağlanıyoruz

smtpClient.Authenticate("ebtuyou1@gmail.com", "kurzpwyuuaybvfzf"); // gmail hesabımızın kullanıcı adı ve şifresi ile kimlik doğrulaması yapıyoruz

smtpClient.Send(mimeMessage); // maili gönderiyoruz

smtpClient.Disconnect(true); // bağlantıyı kesiyoruz

return RedirectToAction("UserActivation", "Activation"); //buraya gönder

}

var timeDiff = DateTime.Now - Model.SendDate;

string timeAgo;

if (timeDiff.TotalMinutes<1)

{

timeAgo = "Şimdi";

}

else if (timeDiff.TotalMinutes<60)

{

timeAgo = $"{(int)timeDiff.TotalMinutes} dakika önce";

}

else if (timeDiff.TotalHours<24)

{

timeAgo = $"{(int)timeDiff.TotalHours} saat önce";

}

else if (timeDiff.TotalDays<7)

{

timeAgo = $"{(int)timeDiff.TotalHours} gün önce";

}

else

{

timeAgo = Model.SendDate.ToString("dd MMMM yyyy");

}

Tarihi alma

JWT

✅ **Konu Tanımı**  
→ **JSON Web Token (JWT)**, iki taraf arasında güvenli bir şekilde bilgi taşımak için kullanılan, dijital olarak imzalanmış bir “token” yapısıdır.

🧠 **Mantığı**  
→ Kullanıcı giriş yaptıktan sonra her istekte tekrar tekrar şifre girmesini önlemek için **sunucu** kullanıcıya bir **JWT** verir. Bu token, kimliği doğrulamak için taşınır. Böylece oturum yönetimi kolaylaşır ve güvenli hale gelir.

📍 **Nerede ve Nasıl Kullanılır?**  
→ Özellikle **Web API**’lerde, mobil uygulamalarda ve mikro servisler arasında kimlik doğrulamada kullanılır.  
→ Alternatif olarak **Session** veya **Cookie-based Authentication** kullanılabilir ama **JWT** daha hafif ve dağıtık sistemlerde avantajlıdır.

📌 **Günlük Hayattan Basit Bir Örnek**  
→ Sinema bileti gibi düşün. Kasadan bilet aldığında (JWT), üzerinde senin ismin ve hangi filme gireceğin bilgisi yazılıdır.  
→ Kapıdaki görevli (API) her seferinde kim olduğunu sormaz, sadece bileti (JWT) kontrol eder.

💻 **C# Kod Örneği**

using System;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using Microsoft.IdentityModel.Tokens;

using System.Text;

class Program

{

static void Main()

{

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("supersecretkey12345"));

var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

var claims = new[]

{

new Claim(ClaimTypes.Name, "Ali"),

new Claim(ClaimTypes.Role, "Admin")

};

var token = new JwtSecurityToken(

issuer: "myApp",

audience: "myUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(30),

signingCredentials: credentials

);

string jwt = new JwtSecurityTokenHandler().WriteToken(token);

Console.WriteLine("JWT: " + jwt);

}

}

🔁 **Alternatifler ve Kıyaslama**

* **Session**: Sunucuda tutulur, yük dağıtımında zorluk çıkarabilir.
* **Cookie**: Tarayıcı tabanlıdır, küçük veriler için uygundur.
* **JWT**: Sunucusuz taşınabilir, dağıtık sistemlerde daha uygun.

✅ **Konu Tanımı**

* **Authentication** → Kullanıcının **kim olduğunu doğrulama** sürecidir.
* **Authorization** → Doğrulanan kullanıcının **hangi işlemleri yapabileceğini belirleme** sürecidir.

🧠 **Mantığı**

* **Authentication**: "Bu kişi gerçekten Ali mi?" sorusuna cevap arar.
* **Authorization**: "Ali, Admin sayfasına girebilir mi?" sorusunu cevaplar.

Yani önce **Authentication** yapılır, ardından **Authorization** kontrolü gelir.

📍 **Nerede ve Nasıl Kullanılır?**

* **Authentication** → Kullanıcı giriş (Login) ekranlarında. (Örn: email + şifre doğrulama)
* **Authorization** → Kullanıcı giriş yaptıktan sonra yetkilendirilmiş sayfalarda. (Örn: sadece Admin rolündeki kullanıcı ayarları değiştirebilir.)

Alternatif olarak farklı kimlik doğrulama yöntemleri vardır:

* **Basic Authentication**
* **JWT**
* **OAuth2**

📌 **Günlük Hayattan Basit Bir Örnek**  
Bir apartman düşün:

* **Authentication**: Kapıdaki görevliye kimliğini gösterip “Ben burada oturuyorum” demek.
* **Authorization**: Apartmanda sadece senin katına çıkmana izin verilmesi. (Başka katlara çıkamazsın.)

💻 **C# Kod Örneği**

// Authentication: Kullanıcı giriş kontrolü

if(username == "ali" && password == "1234")

{

Console.WriteLine("Authentication başarılı: Ali giriş yaptı.");

// Authorization: Yetki kontrolü

string role = "Admin";

if(role == "Admin")

Console.WriteLine("Authorization başarılı: Admin paneline erişim sağlandı.");

else

Console.WriteLine("Authorization başarısız: Erişim reddedildi.");

}

else

{

Console.WriteLine("Authentication başarısız: Giriş bilgileri hatalı.");

}

🔁 **Alternatifler ve Kıyaslama**

* **Authentication** olmadan **Authorization** yapılamaz.
* **Authentication** = Kimlik doğrulama.
* **Authorization** = Yetki kontrolü.
* Örneğin **JWT** ikisini bir arada taşıyabilir: JWT’nin içinde kullanıcı **Claim** bilgileri vardır (Authentication) ve role göre erişim kısıtlanabilir (Authorization).

✅ **Konu Tanımı**  
→ **JSON Web Token (JWT)** üç parçadan oluşur:

1. **Header** → Algoritma ve token tipi bilgisini taşır.
2. **Payload** → Kullanıcı bilgilerini (claims) taşır.
3. **Signature** → Token’ın bütünlüğünü ve doğruluğunu garanti eder.

🧠 **Mantığı**

* **Header** = “Bu token nasıl imzalandı?”
* **Payload** = “Bu token kimin bilgilerini içeriyor?”
* **Signature** = “Bu token gerçekten doğru mu, değiştirilmiş mi?”

Bu üçlü sayesinde JWT hem **bilgi taşıyıcı** hem de **güvenli** hale gelir.

📍 **Nerede ve Nasıl Kullanılır?**

* **Header** → Token doğrulama sürecinde hangi algoritmanın kullanılacağını belirtir.
* **Payload** → Kullanıcının kimliği, rolü, token geçerlilik süresi gibi bilgiler burada taşınır.
* **Signature** → Token’ın değiştirilmediğini garanti eder, güvenliği sağlar.

JWT yapısı:

xxxxx.yyyyy.zzzzz

Header.Payload.Signature

📌 **Günlük Hayattan Basit Bir Örnek**  
Bir uçak bileti düşün:

* **Header** → “Bu bilet THY tarafından basıldı, QR kod formatında” gibi teknik bilgiler.
* **Payload** → Yolcunun adı, uçuş tarihi, koltuk numarası.
* **Signature** → Biletin altındaki barkod/QR kod. Sahte olmadığını ispatlar.

💻 **C# Kod Örneği**

using System;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using Microsoft.IdentityModel.Tokens;

using System.Text;

class Program

{

static void Main()

{

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("supersecretkey12345"));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var claims = new[]

{

new Claim(ClaimTypes.Name, "Ali"),

new Claim(ClaimTypes.Role, "Admin")

};

var token = new JwtSecurityToken(

issuer: "myApp",

audience: "myUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(10),

signingCredentials: creds

);

var handler = new JwtSecurityTokenHandler();

string jwt = handler.WriteToken(token);

Console.WriteLine("JWT: " + jwt);

Console.WriteLine("\n--- Header ---");

foreach (var h in token.Header) Console.WriteLine($"{h.Key}: {h.Value}");

Console.WriteLine("\n--- Payload ---");

foreach (var c in token.Claims) Console.WriteLine($"{c.Type}: {c.Value}");

Console.WriteLine("\n--- Signature ---");

Console.WriteLine("Token’ın son kısmı (güvenlik imzası).");

}

}

Örnek JWT (3 parçalı):

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 → Header

eyJ1c2VyTmFtZSI6IkFsaSIsInJvbGUiOiJBZG1pbiJ9 → Payload

abc123xyzSignature456 → Signature

🔁 **Alternatifler ve Kıyaslama**

* **Header** = Sabit yapı, algoritma bilgisi.
* **Payload** = Dinamik yapı, kullanıcı verileri.
* **Signature** = Değiştirilemez güvenlik katmanı.  
  👉 **Payload şifrelenmez, sadece Base64 ile encode edilir.** Bu yüzden gizli bilgi koymak tehlikelidir.  
  👉 Gerçek güvenlik, **Signature** kısmındadır.

Uygulama

Apsetting jsona gir

"JwtSettings": {

"Key": "notikaidentityprojesicorenotikaidentityprojesicore", //bu ayarlanacak şifre

"Issuer": "localhost", //dinleyici

"Audience": "localhost", //Alıcı

"ExpireMinutes": 5 //Tokenın Geçerli olacağı süre

},

Modelview ve ya dto oluştur

public class JwtSettingsModel

{

public string Key { get; set; } //jsonda verilen değer

public string Issuer { get; set; } //jsonda verilen değer

public string Audience { get; set; } //jsonda verilen değer

public string ExpireMinutes { get; set; } //jsonda verilen değer

}

Program.cs

builder.Services.Configure<JwtSettingsModel>(builder.Configuration.GetSection("JwtSettings"));

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

}).AddJwtBearer(opt =>

{

var jwtSettings = builder.Configuration.GetSection("JwtSettings").Get<JwtSettingsModel>();

opt.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = jwtSettings.Issuer,

ValidAudience = jwtSettings.Audience,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings.Key))

};

});

private readonly JwtSettingsModel \_jwtSettingsModel;

public TokenController(IOptions< JwtSettingsModel> jwtSettingsModel)

{

\_jwtSettingsModel = jwtSettingsModel.Value;

}

[HttpGet]

public IActionResult Generate()

{

return View();

}

[HttpPost]

public IActionResult Generate(SimpleUserViewModel model)

{

var claim = new[]

{

new Claim("name",model.Name),

new Claim("Surname",model.Surname),

new Claim("Username",model.Username),

new Claim("City",model.City),

new Claim(JwtRegisteredClaimNames.Jti,Guid.NewGuid().ToString()), //token ürettik

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_jwtSettingsModel.Key));

var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

issuer: \_jwtSettingsModel.Issuer,

audience: \_jwtSettingsModel.Audience,

claims: claim,

expires: DateTime.UtcNow.AddMinutes(\_jwtSettingsModel.ExpireMinutes),

signingCredentials: creds);

model.Token = new JwtSecurityTokenHandler().WriteToken(token);

return View(model);

}

@if (!string.IsNullOrEmpty(Model?.Token))

{

<div class="alert-validate alert-success mt-3">

<strong>Token: </strong><br />

<textarea rows="5" class="form-control">@Model.Token</textarea>

</div>

}

Şifre yenileme

builder.Services.AddIdentity<AppUser, IdentityRole>().AddEntityFrameworkStores<EmailContext>().AddErrorDescriber<CustomIdentityValidator>().AddTokenProvider<DataProtectorTokenProvider<AppUser>>(TokenOptions.DefaultProvider);

var olan kodu buna çevirdik

private readonly UserManager<AppUser> \_userManager;

public PasswordChangeController(UserManager<AppUser> userManager)

{

\_userManager = userManager;

}

[HttpGet]

public IActionResult ForgetPassword()

{

return View();

}

[HttpPost]

public async Task<IActionResult> ForgetPassword(ForgetPasswordViewModel model)

{

var user = await \_userManager.FindByEmailAsync(model.Email);

string passwordResetToken = await \_userManager.GeneratePasswordResetTokenAsync(user);

var passwordResetTokenLink = Url.Action("ResetPassword", "PasswordChange", new

{

userId = user.Id,

token = passwordResetToken

}, HttpContext.Request.Scheme);

MimeMessage mimeMessage = new MimeMessage();

MailboxAddress mailboxAddressFrom = new MailboxAddress("Notika Admin", "ebtuyou1@gmail.com");

mimeMessage.From.Add(mailboxAddressFrom);

MailboxAddress mailboxAddressTo = new MailboxAddress("User", model.Email);

mimeMessage.To.Add(mailboxAddressTo);

var bodyBuilder = new BodyBuilder();

mimeMessage.Subject = "Şifre Sıfırlama Talebi";

bodyBuilder.TextBody = $"Merhaba,\n\n" +

$"Hesabınızın Şifresini değiştirmek için lütfen aşağıdaki alanı kullanın:\n\n" +

$"{passwordResetTokenLink}\n\n" +

"NotikaAdminChat Ekibi\n\n" + $"Hello \n\n" +

$"To change your account password, please use the field below.:\n\n" +

$"{passwordResetTokenLink}\n\n" +

"NotikaAdminChat Team";

mimeMessage.Body = bodyBuilder.ToMessageBody();

SmtpClient smtpClient = new SmtpClient();

smtpClient.Connect("smtp.gmail.com", 587, false); // gmail smtp sunucusuna bağlanıyoruz

smtpClient.Authenticate("ebtuyou1@gmail.com", "kurzpwyuuaybvfzf"); // gmail hesabımızın kullanıcı adı ve şifresi ile kimlik doğrulaması yapıyoruz

smtpClient.Send(mimeMessage); // maili gönderiyoruz

smtpClient.Disconnect(true); // bağlantıyı kesiyoruz

return View();

}

[HttpGet]

public IActionResult ResetPassword(string userId, string token)

{

TempData["userId"] = userId;

TempData["token"] = token;

return View();

}

[HttpPost]

public async Task<IActionResult> ResetPassword(ResetPasswordViewModel model)

{

var userId = TempData["userId"];

var token = TempData["token"];

if (userId== null || token==null)

{

ModelState.AddModelError("", "Hata oluştur");

}

var user = await \_userManager.FindByIdAsync(userId.ToString());

var result = await \_userManager.ResetPasswordAsync(user, token.ToString(), model.Password);

if (result.Succeeded)

{

return RedirectToAction("UserLogin", "Login");

}

else

{

foreach (var item in result.Errors)

{

ModelState.AddModelError("", item.Description);

}

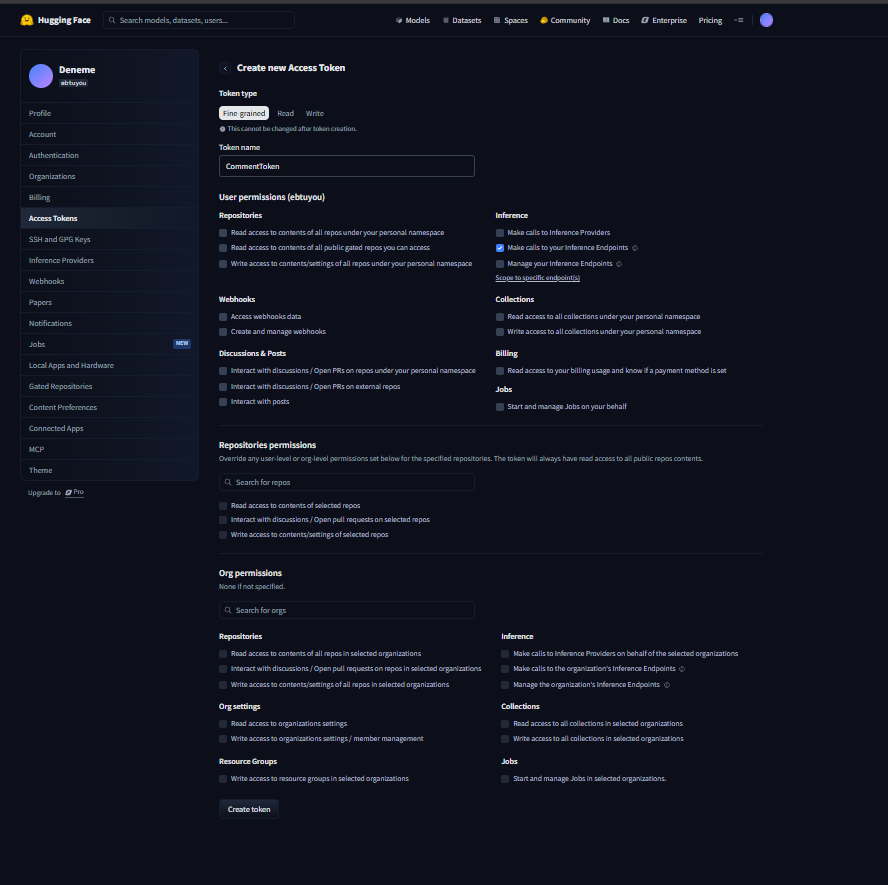
return View();

}

}

Hugging face entegre etme

Üye ol ardından billingslere bak ardından create Token de sayfa böyle olsun api çağrısı için



Tokenı al create token de

Sonra kodlar

[HttpPost]

public async Task<IActionResult> CreateComment(Comment comment)

{

var values = await \_userManager.FindByNameAsync(User.Identity.Name);

comment.AppUserId = values.Id;

comment.CommentDate = DateTime.Now;

//toxic analiz

using (var client = new HttpClient())

{

var apiKey = "apikeygelecek";

client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", apiKey);

try

{

var translateRequestBody = new

{

inputs = comment.CommentDetail

};

var translateJson = JsonSerializer.Serialize(translateRequestBody);

var translateContent = new StringContent(translateJson, Encoding.UTF8, "application/json");

var translateResponse = await client.PostAsync("https://api-inference.huggingface.co/models/Helsinki-NLP/opus-mt-tr-en", translateContent);

var translateResponseString = await translateResponse.Content.ReadAsStringAsync();

string englishText = comment.CommentDetail;

if (translateResponseString.TrimStart().StartsWith("["))

{

var translateDoc = JsonDocument.Parse(translateResponseString);

englishText = translateDoc.RootElement[0].GetProperty("translation\_text").GetString();

}

var toxicRequestBody = new

{

inputs = englishText

};

var toxicJson = JsonSerializer.Serialize(toxicRequestBody);

var toxicContent = new StringContent(toxicJson, Encoding.UTF8, "application/json");

var toxicResponse = await client.PostAsync("https://api-inference.huggingface.co/models/unitary/toxic-bert", toxicContent);

var toxicResponseString = await toxicResponse.Content.ReadAsStringAsync();

if (toxicResponseString.TrimStart().StartsWith("["))

{

var toxicDoc = JsonDocument.Parse(toxicResponseString);

foreach (var item in toxicDoc.RootElement[0].EnumerateArray())

{

string label = item.GetProperty("label").GetString();

double score = item.GetProperty("score").GetDouble();

if (score > 0.3)

{

comment.CommentStatus = "Toksik Yorum";

break;

}

}

}

if (string.IsNullOrEmpty(comment.CommentStatus))

{

comment.CommentStatus = "Onaylandı";

}

}

catch (Exception ex)

{

comment.CommentStatus = "Onay Bekliyor";

}

\_context.Comments.Add(comment);

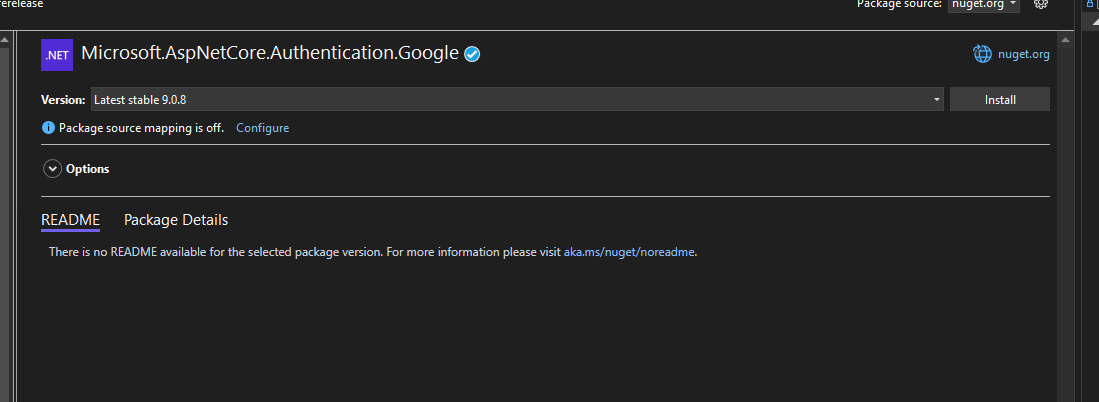
\_context.SaveChanges();

return RedirectToAction("UserCommentList");

}

}

GOOGLE KONFİGURASYONLARI



BUNU KURACAĞIZ

Program.cs ayarı

//GOOGLE AUTHENTİCATİON KONFİGURASYONU

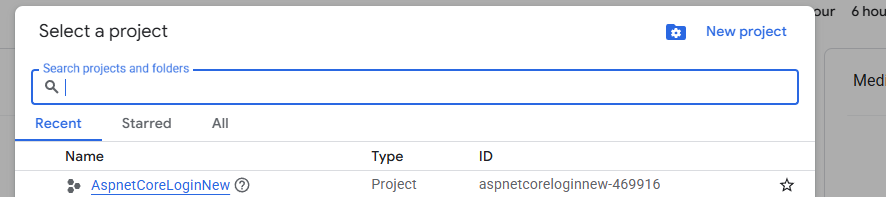
builder.Services.AddAuthentication().AddGoogle(opt =>

{

opt.ClientId = "google client ıd gelecek";

opt.ClientSecret = " google client secret gelecek";

});

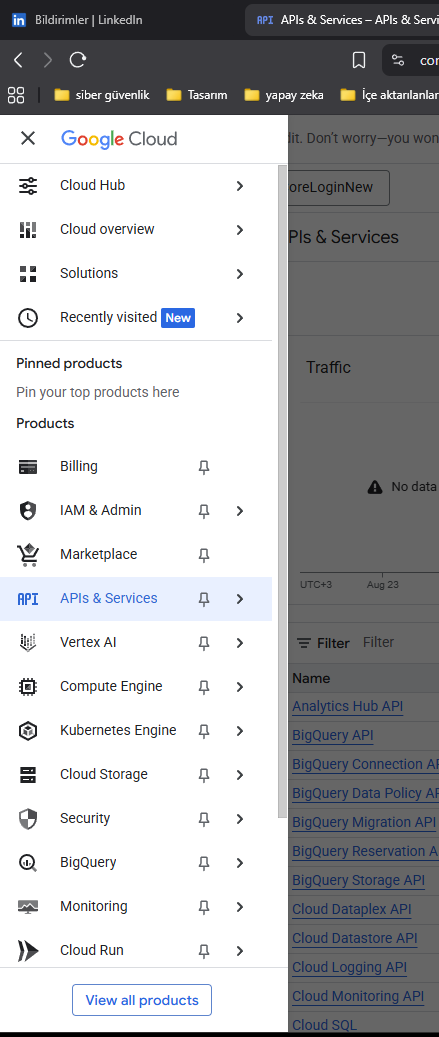


Developer consolea gir

<https://console.cloud.google.com/>

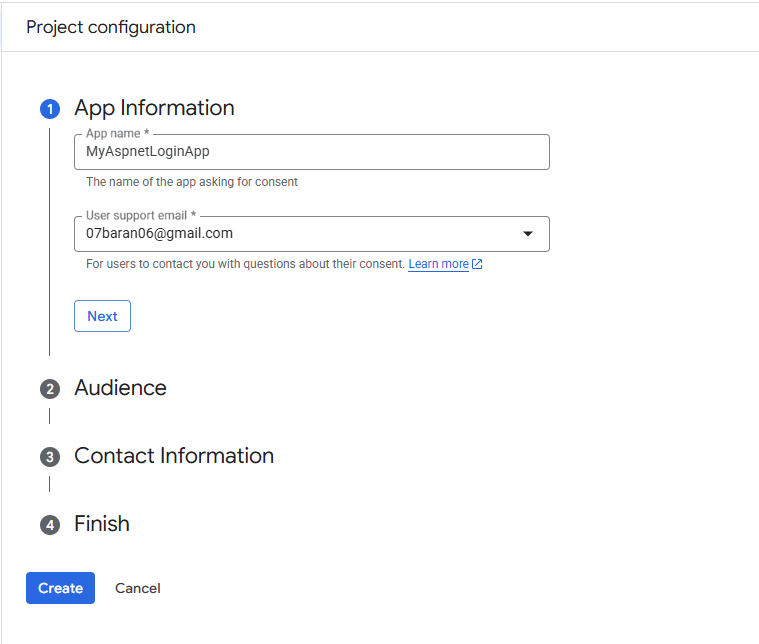
buradan proje oluştur adını ver

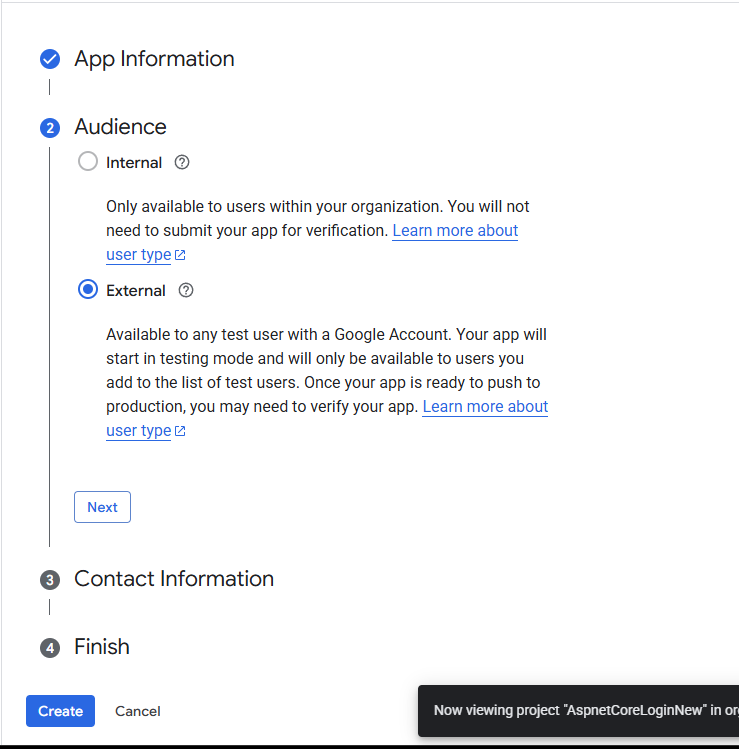
üstüne bas ve projeyi seç

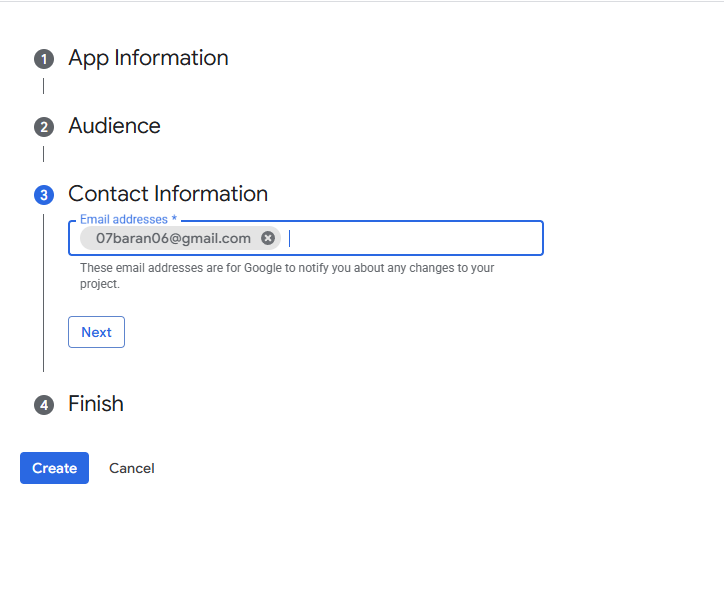


apiservice Bas ve oauth servise seç

Get started de isim ver

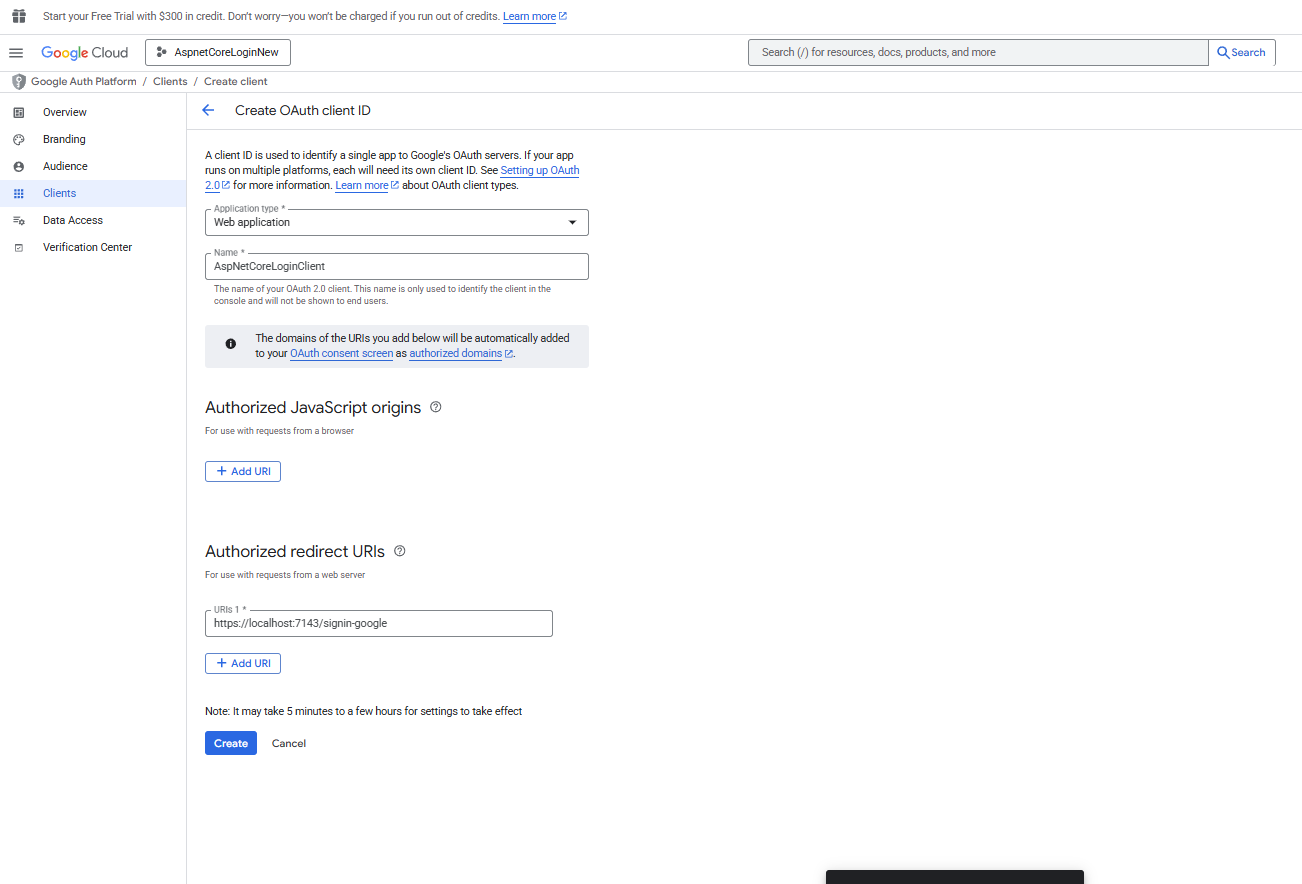






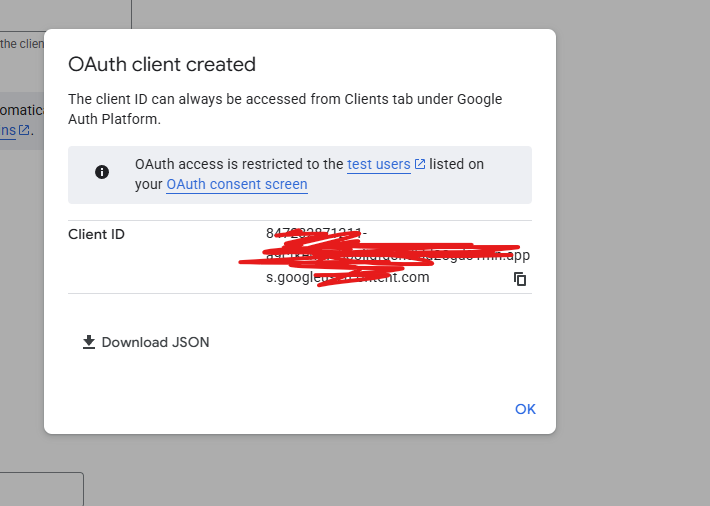
Ayarlar bu kadar.

Gelen sayfadan create oauth client de ve

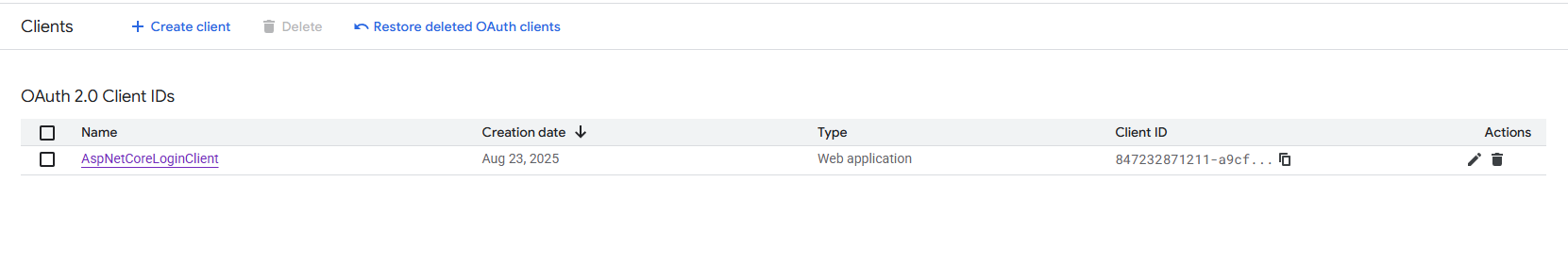


Özellikle Authorized redirect URIs

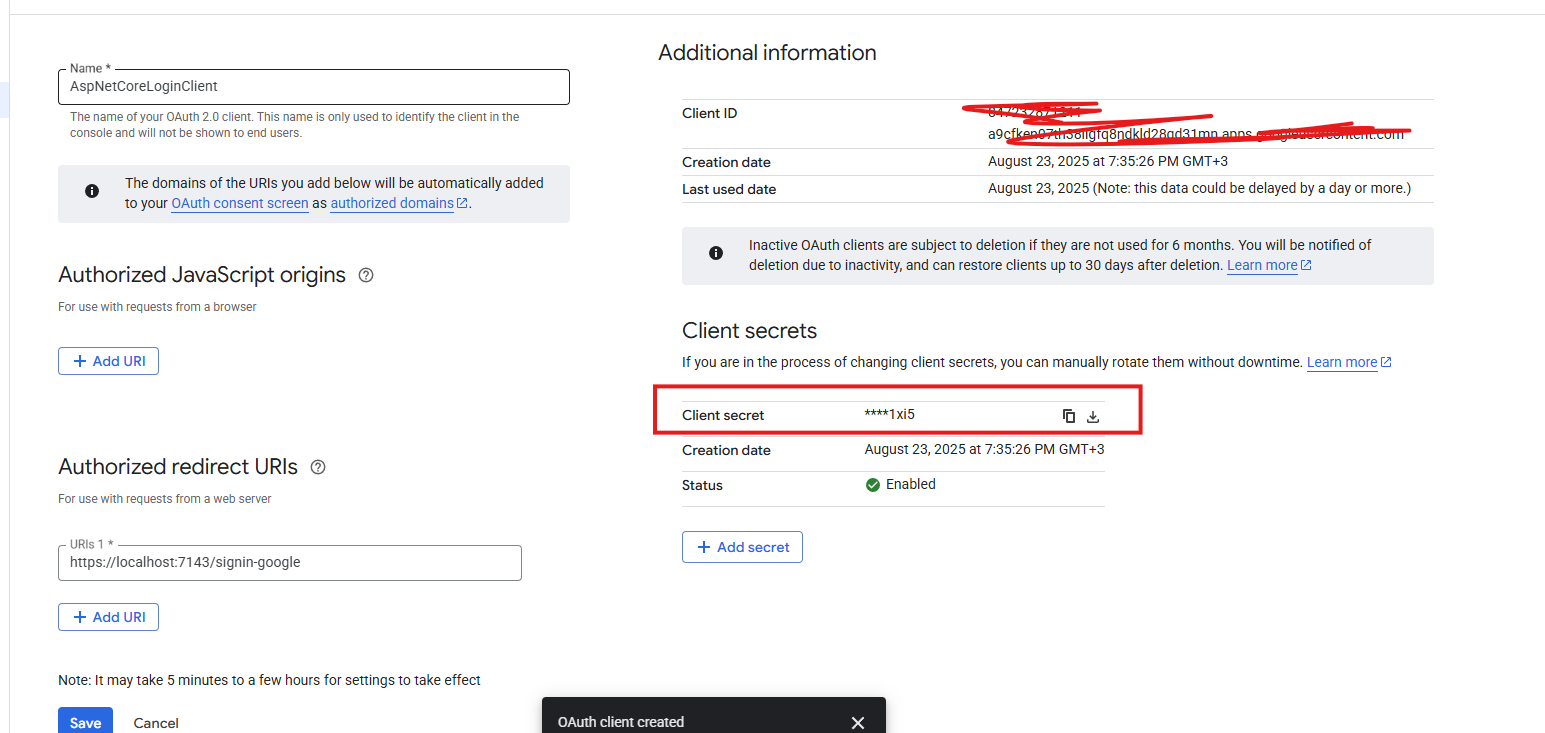
Önemli burası senin localhost ne gelirse o



İşte bu kadar ıd



Üstüne tıkla ve



Ve ıd karşında

using Microsoft.AspNetCore.Identity;

using Microsoft.AspNetCore.Mvc;

using NotikaIdentityEmail.Context;

using NotikaIdentityEmail.Entities;

using NotikaIdentityEmail.Models.IdentityModels;

using System.Security.Claims;

using System.Threading.Tasks;

namespace NotikaIdentityEmail.Controllers

{

public class LoginController : Controller

{

private readonly SignInManager<AppUser> \_signInManager;

private readonly EmailContext \_emailContext;

private readonly UserManager<AppUser> \_userManager;

public LoginController(SignInManager<AppUser> signInManager, EmailContext emailContext, UserManager<AppUser> userManager)

{

\_signInManager = signInManager;

\_emailContext = emailContext;

\_userManager = userManager;

}

[HttpGet]

public IActionResult LoginWithGoogle()

{

return View();

}

[HttpGet]

public IActionResult UserLogin()

{

return View();

}

[HttpPost]

public async Task<IActionResult> UserLogin(UserLoginViewModel model)

{

var value = \_emailContext.Users.Where(x => x.UserName == model.Username).FirstOrDefault();

if (value == null)

{

ModelState.AddModelError(string.Empty, "Kullanıcı bulunamadı.");

return View(model);

}

if (!value.EmailConfirmed)

{

ModelState.AddModelError(string.Empty, "E-Mail Adresinizi henüz onaylanmamış.");

return View(model);

}

var result = await \_signInManager.PasswordSignInAsync(model.Username, model.Password, true, true);

if (result.Succeeded)

{

return RedirectToAction("EditProfile", "Profile");

}

ModelState.AddModelError(string.Empty, "Kullanıcı adı veya şifre yanlış");

return View(model);

//return View();

}

[HttpPost]

public IActionResult ExternalLogin(string provider , string? returnUrl=null)

{

var redirectUrl = Url.Action("ExternalLoginCallBack", "Login", new { returnUrl });

var properties = \_signInManager.ConfigureExternalAuthenticationProperties(provider, redirectUrl);

return Challenge(properties, provider);

}

[HttpGet]

public async Task<IActionResult> ExternalLoginCallBack(string? returnUrl=null,string? remoteError = null)

{

returnUrl ??= Url.Content("~/");

if (remoteError != null)

{

ModelState.AddModelError("", $"External Provider Error: {remoteError}");

return RedirectToAction("Login");

}

var info = await \_signInManager.GetExternalLoginInfoAsync();

if (info==null)

{

return RedirectToAction("Login");

}

var result = await \_signInManager.ExternalLoginSignInAsync(info.LoginProvider, info.ProviderKey, false);

if (result.Succeeded)

{

return RedirectToAction("Inbox", "Message");

}

else

{

var email = info.Principal.FindFirstValue(ClaimTypes.Email);

var user = new AppUser()

{

UserName = email,

Email = email,

Name = info.Principal.FindFirstValue(ClaimTypes.GivenName)??"Google",

Surname = info.Principal.FindFirstValue(ClaimTypes.Surname) ?? "User"

};

var identityResult = await \_userManager.CreateAsync(user);

if (identityResult.Succeeded)

{

await \_userManager.AddLoginAsync(user, info);

await \_signInManager.SignInAsync(user, isPersistent: false);

return RedirectToAction("Inbox", "Message");

}

return RedirectToAction("UserLogin");

}

}

}

}

<div class="container d-flex justify-content-center align-items-center" style="min-height: 100vh;">

<div class="card shadow p-4" style="max-width: 400px; width: 100%;">

<h4 class="card-title text-center mb-4">Google İle Giriş Yap</h4>

<**form** method="post" **asp-action**="ExternalLogin" **asp-controller**="Login">

<input type="hidden" name="provider" value="Google" />

<button type="submit" class="btn btn-outline-danger w-100 d-flex align-items-center justify-content-center">

<i class="fa fa-google me-2"></i> Google ile Giriş Yap

</button>

</**form**>

</div>

</div>

// JWT Ayarlarý

builder.Services.Configure<JwtSettingsModel>(builder.Configuration.GetSection("JwtSettings"));

// Cookie + JWT birlikte authentication

builder.Services.AddAuthentication(options =>

{

options.DefaultScheme = CookieAuthenticationDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = CookieAuthenticationDefaults.AuthenticationScheme;

})

.AddCookie(CookieAuthenticationDefaults.AuthenticationScheme, options =>

{

options.LoginPath = "/Login/UserLogin";

options.AccessDeniedPath = "/Error/403";

})

.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, opt =>

{

var jwtSettings = builder.Configuration.GetSection("JwtSettings").Get<JwtSettingsModel>();

opt.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = jwtSettings.Issuer,

ValidAudience = jwtSettings.Audience,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings.Key))

};//GOOGLE AUTHENTİCATİON KONFİGURASYONU

}).AddGoogle(GoogleDefaults.AuthenticationScheme, opt =>

{

opt.ClientId = "key gelecek";

opt.ClientSecret = "key gelecek";

opt.CallbackPath = "/signin-google";

});

Böylelikle google ile giriş sağladık