# Veritabanı Şemaları

## İlgili Konular ve Kapsam Hakkında

DB Şemaları, Personel Tablosundaki Kayıt Alanları

## Gereksinim / Fikir Özeti

Şimdi bir tartışma başlatmak istiyorum. İhtiyaçlarımı aşağıda belirttim. Bu ihtiyaçların karşılanabilmesi için Personel varlığında olması gerekenler konusunda görüşlerine ihtiyacım var.

Şuana kadar üretilen entities için şema yaklaşımını benimsemedik. Oysa veritabanında şema yaklaşımı ile hareket etmek istiyorum.

Örneğin subs isimli şema oluşturulmalı ve Tenant, Plan, Fetaure, Subscription gibi varlıklar bu şemada yer almalı.

Hres şeması oluşturulmalı ve Çekirdek İk Varlıkları (Örneğin personnel, personnel\_addres gibi varlıklar burada yer almalı)

Core şeması oluşturulmalı ve User, Role, User role gibi varlıklar burada yer almalı.

Bunları yazıyorum. Çünkü Api Fİrst ve Code First yaklaşımız var. Ama DB Design'da çok önemli ve doğru olmalı.

Public Şemasında ortak bir varlık olmasından kaynaklı country tablosu (id, ulke, country, ulkekisa, countryshort, uyruk, nationality, isoalpha2, isoalpha3) olmalı.

## Tartışma (AI)

* **Veritabanı Şema Yaklaşımı**

**Görüşünüz:** Veritabanında subs, hres, core gibi şemalar kullanarak varlıkları gruplamak.

**Değerlendirme ve Öneri:** Bu, **kesinlikle doğru ve en iyi uygulama pratiğidir.** Veritabanı şemalarını kullanmak, projenizin veritabanı seviyesinde mantıksal olarak modüllere ayrılmasını sağlar. Bu, özellikle büyük ve karmaşık veritabanlarında performansı, güvenliği ve yönetilebilirliği artırır.

Nasıl Uygularız? (Code First Yaklaşımı ile):

Entity Framework Core'da bunu uygulamak oldukça basittir. İlgili varlığın Configuration dosyasında (...Configuration.cs) ToTable metodunu kullanarak şemayı belirtiriz.

**Örnek (PersonnelConfiguration.cs içinde):**

C#

builder.ToTable("personnel", "hres"); // "personnel" tablosunu "hres" şemasına koyar.

**Country Tablosu:** public şemasında, tüm kiracılar tarafından paylaşılan bir Country tablosu oluşturma fikriniz de **mükemmel**. Bu, veri tekrarını önler ve standart bir yapı sunar.

| ID | Görev Adı | Sorumlu Proje | Açıklama |
| --- | --- | --- | --- |
| **P.1** | **Veritabanı Şema Yapısını Uygula** | EduHR.Infrastructure | Tüm Configuration sınıflarını, ToTable("table\_name", "schema\_name") metodunu kullanacak şekilde güncelle. |

## Fikir Detay (2)

Veritabanı Şema Yaklaşımı

Şimdi bunun detaylarını tartışalım

Benim için EduHR-2025 Projesindeki varlıkları için Veritabanı şemalarını listeler misin. Bu listede, her şemanın adı 4 karakterli olacak, içereceği varlıkların isimlendirmesi Postgresql'e uygun olmalı, Varlıkların Türkçe açıklamaları (Description) mutlaka yer almalı. Açıklamada önce özet bir açıklama, sonra bu varlığın neden oluşturulacağı yer almalı(Maks 255 karakter)

Bu tabloyu hazırlarken şuana kadar ki konuları dikkate alıp tüm varlıkları düşünmelisin.

Ayrıca harici kimlik sağlayıcılar (Identity Providers) (Google hesapları) ve Tailwind CSS (Kullanıcının kendi teması vb gibi durumları düşünerek) dolayısıyla oluşturulması gereken tabloları da hesaba katmanı istiyorum.

## Tartışma 2 (AI)

Bu harika bir soru. Bu, projemizin sadece teorik bir model olmadığını, aynı zamanda gerçek dünya verileriyle nasıl çalışacağını da düşündüğümüzü gösteriyor. Belirttiğiniz "Bordro" ve "Puantaj" süreçleri, veri girişi yerine toplu veri yüklemesine dayandığı için özel tablolar ve yaklaşımlar gerektirir.

Önceki şema listemiz, bu süreçlerin **sonuçlarını** (salaries, timesheets) saklamak için tasarlanmıştı. Şimdi, bu süreçlerin **kendisini** (yani, dosya yükleme, işleme ve geçmişini tutma) yönetecek tabloları ekleyerek planımızı tamamlayalım.

1. **Tartışma ve Çözüm Önerileri**
2. **A) BORDROLAR: Excel ve PDF Yükleme Süreci**
3. **Excel ile Maaş Yükleme:** Bu bir "veri içeri aktarma" işlemidir. Yüklenen Excel'deki veriler işlendikten sonra, her personel için hres.salaries tablosuna yeni bir kayıt olarak girilir. Bu işlemin geçmişini ve durumunu takip etmek için ayrı bir "içeri aktarma işi" tablosuna ihtiyacımız var.
4. **PDF Ücret Pusulaları:** Her bir PDF, bir personele ve bir döneme aittir. Bunları genel personnel\_documents tablosunda tutmak yerine, daha spesifik bir payslips tablosunda yönetmek daha doğrudur. Bu payslips tablosu, belgenin kendisini değil, meta verilerini (kimin, hangi dönemin, dosya yolu nerede vb.) tutar.
5. **B) PUANTAJLAR: PDKS Verilerinin Yüklenmesi**

Bu, klasik bir ETL (Extract, Transform, Load - Çıkar, Dönüştür, Yükle) sürecidir. Ham veriyi (giriş/çıkış saatleri) doğrudan ana timesheets tablomuza yazmak yerine, önce geçici bir "hazırlık" (staging) tablosuna almalıyız. Bu, veriyi işlemeden önce doğrulama yapmamızı, hataları ayıklamamızı ve "son yüklenen dosya geçerlidir" kuralını uygulamamızı sağlar.

1. **Revize Edilmiş Şema Planı (Yeni Tablolar Eklenmiş)**

Bu ihtiyaçları karşılamak için, hres (İnsan Kaynakları) şemamızı zenginleştirmeli ve impr (Imports - İçeri Aktarımlar) adında yeni bir şema oluşturmalıyız.

1. **--- İnsan Kaynakları (hres) ---**

| Şema Adı | Tablo Adı (PostgreSQL Uyumlu) | Açıklama (Özet + Gerekçe) |
| --- | --- | --- |
| hres | **payslips** | **(Güncellendi)** Personele ait ücret pusulası (bordro) PDF'lerinin meta verilerini tutar. Personelin kendi bordrosunu görüntüleyebilmesi için dosyanın yolunu ve ilgili dönemi kaydeder. |
| hres | salaries | Personelin maaş geçmişini ve ücret bilgilerini güvenli bir şekilde saklar. Bu tablo, manuel olarak veya **impr.payroll\_imports** tablosundan gelen verilerle doldurulur. |

1. **--- İçeri Aktarımlar (impr) (Yeni Şema) ---**

| Şema Adı | Tablo Adı (PostgreSQL Uyumlu) | Açıklama (Özet + Gerekçe) |
| --- | --- | --- |
| **impr** | **payroll\_imports** | **(Yeni)** Toplu maaş verilerini içeren Excel dosyalarının yükleme geçmişini tutar. Her bir yükleme işinin kim tarafından, ne zaman yapıldığını ve durumunu (işleniyor, tamamlandı, hatalı) kaydeder. |
| **impr** | **pdkS\_import\_jobs** | **(Yeni)** PDKS'den gelen puantaj verilerini içeren Excel/CSV dosyalarının yükleme işlerini takip eder. "Aynı ay için son yüklenen geçerlidir" kuralını uygulamak için bu tablo kullanılır. |
| **impr** | **pdkS\_raw\_data** | **(Yeni)** Yüklenen puantaj dosyalarındaki ham verileri (personel kart no, giriş/çıkış saati) geçici olarak saklar. Bu veriler, bir arka plan işi tarafından işlenerek ana tatt.timesheets tablosuna aktarılır. |

# Personel Varlığı

## İlgili Konular ve Kapsam Hakkında

Personel Tablosundaki Kayıt Alanları

## Gereksinim / Fikir Özeti

1) Departman Bilgisi: Personelin hangi departmanda olduğuna göre hızlıca filtreleme vb yapabilmek için ihtiyaç duyuyorum. Ama position ID'den yakalanabilir.

2) ManagerID: Bu neden önemli, performans değerlendirmesi veya personelin isteklerinde (Avans-izin gibi) yöneticisinin değerlendirme (Onam vb) yapabilmesini bununla sağlarım diye düşündüm. Ama positionId ile Departman yakalanıp departmanın yöneticisi bulunabilir. (Bu durumda kaydedilecek personelin Departman yöneticisi olduğunu anlamamı sağlayacak bir flag lazım.) Bu flag Position'a da eklenebilir. Böylece bir departmanda birden fazla yönetici olabilir ve yöneticinin önüne düşen istekleri o departmandaki tüm yöneticiler veya enaz 1'i onaylamalıdır gibi iş kurallar yazılabilir (daha esnek bir yapı). Ancak doğru çözümü sen bana öner.

3) TCKN için bir doğrulama algoritmasına ihtiyacım var. (Github'daki eozban/CSharp-TCKimlikNoDogrulama dan faydalanabiliriz)

4) Marial Status ihtiyacım var. Personelin evli, bekar, dul, boşanmış gibi durumları kaydedilmeli. (Enum olabilir)

5) Password nasıl oluşacak. Personnel kaydı yapan tarafından mı oluşturulacak?

6) Address için Ayrı bir tablodaki alanların olması daha doğru değil mi. Bu ayrı tabloda ;

id, personnel\_id, adress\_type (İkametgah Adresi, İş Adresi, Diğer Adres gibi), neighborhood, street\_name, building\_no, apartment\_no, postal\_code, district

city, country\_id (Country tablosu yukarıda açıklandı.), full\_address (Hesaplanmış alan (şunların birleşimi) olmalı: neighborhood & ", "&street\_name& ", "&building\_no& ", "&apartment\_no& ", "&postal\_code& ", "&district& ", "&city& ", "&(Country Tablosundaki ULKE Sütunu verisi)

7)avatarurl: Personel için sisteme yüklenen vesikalık fotonun Çalışanın profil resminin URL adresi.

Bu yazdıklarımı önce tartışalım. Sen bana bu gereksinimlerimin ve önerilerim hakkında en doğru görüşleri sun. Sonra karara bağlayalım. Karar sonrasında yapılması gerekenleri adım adım belirten özel bir plan oluşturalım.

## Tartışma (AI)

* **1) Departman Bilgisi (Personel Üzerinde)**

**Görüşünüz:** Position üzerinden departmana ulaşılabilir.

**Değerlendirme ve Öneri:** Bu konuda **tamamen haklısınız.** Personnel varlığına ayrıca bir DepartmentId eklemek, veritabanı normalizasyon kurallarını ihlal eder ve veri tekrarına yol açar. Bir personelin departmanı, her zaman pozisyonu üzerinden belirlenmelidir. Bu yapı, bir personelin pozisyonu değiştiğinde departmanının da otomatik olarak güncellenmesini sağlar. **Mevcut yapımız bu konuda doğru.**

* **2) ManagerID (Yönetici İlişkisi)**

**Görüşünüz:** Position'a "yönetici" flag'i ekleyerek departman yöneticisini bulmak.

**Değerlendirme ve Öneri:** Bu yaklaşım işe yarar ancak **yeterince esnek değildir.** Birçok modern organizasyonda, bir yöneticinin ekibi farklı departmanlardan oluşabilir veya bir departmanda birden fazla proje yöneticisi olabilir.

**En Doğru Çözüm:** Personnel varlığına doğrudan, yine Personnel tablosuna referans veren bir ManagerId alanı eklemektir.

**Personnel.cs'e Eklenecek Alanlar:**

C#

// Personelin yöneticisinin kimliği (bu da bir personeldir).

public int? ManagerId { get; set; }

public Personnel? Manager { get; set; }

// Bu yöneticinin yönettiği personellerin listesi.

public ICollection<Personnel> Subordinates { get; set; } = new List<Personnel>();

**Bu Yaklaşımın Avantajları:**

* **Esneklik:** Departman yapısından tamamen bağımsız, hiyerarşik bir "amir-ast" ilişkisi kurmanızı sağlar.
* **Performans:** Bir personelin yöneticisini bulmak için tek bir adıma (personnel.Manager) ihtiyaç duyar. Diğer yöntemde ise personnel.Position.Department.ManagerPosition.Personnel gibi karmaşık ve yavaş sorgular gerekir.
* **Basitlik:** Performans değerlendirmesi veya izin onayı gibi süreçlerde, "Bu talebin onaycısı kimdir?" sorusunun cevabı doğrudan personnel.ManagerId olur.
* **3) TCKN Doğrulama Algoritması**

**Görüşünüz:** Harici bir kütüphane (eozban/CSharp-TCKimlikNoDogrulama) kullanmak.

**Değerlendirme ve Öneri:** **Mükemmel bir fikir.** TCKN algoritması gibi standart ve değişmeyen kuralları yeniden icat etmeye çalışmak yerine, topluluk tarafından test edilmiş, güvenilir bir kütüphane kullanmak en doğru yaklaşımdır.

Nasıl Uygularız?:

Bu doğrulama mantığı, Domain katmanında bir Domain Service içinde veya doğrudan Application katmanındaki CreatePersonnelCommandValidator içinde kullanılabilir. Genellikle, bu tür harici kütüphane bağımlılıklarını Infrastructure katmanında bir servis arkasına gizlemek ve Application katmanında o servisi çağırmak en temiz yöntemdir.

* **4) Marital Status (Medeni Durum)**

**Görüşünüz:** Enum olabilir.

**Değerlendirme ve Öneri:** **Kesinlikle.** Medeni durum gibi sabit ve sınırlı sayıda seçeneği olan alanlar için Enum kullanmak en doğru yöntemdir. Domain/Enums klasöründe yeni bir MaritalStatus.cs dosyası oluşturmalıyız.

* **5) Password (Şifre) Oluşturma**

**Görüşünüz:** Personel kaydını yapan mı oluşturacak?

**Değerlendirme ve Öneri:** Bu, güvenlik açısından kritik bir süreçtir. Üç temel yöntem vardır:

1. **İK Tarafından Geçici Şifre Atama:** En az güvenli yöntemdir. Şifrenin bir yerlere yazılması veya paylaşılması gerekebilir.
2. **Sistem Tarafından Rastgele Şifre Atama:** Daha güvenlidir, ancak kullanıcının ilk girişte bu karmaşık şifreyi kopyalayıp yapıştırmasını gerektirir.
3. **Kullanıcı Davet (Invitation) Akışı:** **En Güvenli ve En Modern Yöntem Budur.**

**Önerilen Süreç:**

* İK Uzmanı, personel kaydını oluştururken bir şifre **girmez**.
* Sistem, personel oluşturulduğunda, o personelin e-posta adresine, içinde benzersiz ve kısa süreli geçerli bir link bulunan bir "Hesabınızı Aktive Edin" e-postası gönderir.
* Personel bu linke tıklar, kendi şifresini belirler ve hesabını aktive eder.

Bu yöntem, şifrenin sadece son kullanıcı tarafından bilinmesini garanti eder. CreatePersonnelCommand'dan Password alanını kaldırıp, bu akışı destekleyecek yeni bir InviteUserCommand oluşturabiliriz.

* **6) Address (Ayrı Tablo)**

**Görüşünüz:** Adresler için ayrı ve yapılandırılmış bir tablo olmalı.

**Değerlendirme ve Öneri:** **Kesinlikle katılıyorum.** Bu, veri kalitesini ve gelecekteki raporlama (hangi şehirde kaç personel var? gibi) yeteneklerini inanılmaz derecede artıran, çok doğru bir normalizasyon adımıdır. Önerdiğiniz tablo yapısı (personnel\_address tablosu) ve hesaplanmış full\_address alanı **mükemmeldir.** Domain katmanında yeni bir PersonnelAddress varlığı oluşturmalıyız.

* **7) avatarurl**

**Görüşünüz:** Personel için profil resmi URL'i.

**Değerlendirme ve Öneri:** **Evet, bu gereklidir.** Personnel varlığına string? AvatarUrl adında bir özellik eklemeliyiz. Bu alanda, IFileStorageService'imiz aracılığıyla (Infrastructure katmanında) depolanan resmin URL'i tutulacaktır.

* **Karar Sonrası Özel Plan**

Bu tartışmalar sonucunda, Domain katmanımızı ve bazı iş akışlarımızı zenginleştirecek aşağıdaki adımları plana eklemeliyiz:

| ID | Görev Adı | Sorumlu Proje | Açıklama |
| --- | --- | --- | --- |
| **P.2** | **Yeni Country Varlığını Oluştur** | EduHR.Domain | Önerilen Country varlığını public şeması için oluştur. |
| **P.3** | **Personnel Varlığını Zenginleştir** | EduHR.Domain | ManagerId, MaritalStatus (yeni enum ile), AvatarUrl alanlarını ve Subordinates koleksiyonunu ekle. |
| **P.4** | **Yeni PersonnelAddress Varlığını Oluştur** | EduHR.Domain | Önerilen yapıya uygun olarak PersonnelAddress varlığını ve AddressType enum'ını oluştur. |
| **P.5** | **Kullanıcı Davet Akışını Tasarla** | EduHR.Application | CreatePersonnelCommand'dan şifreyi kaldır. InviteUserCommand ve SetPasswordFromInvitationCommand gibi yeni iş akışları tasarla. |

# Harici Kimlik Sağlayıcılar (Identity Providers)

## İlgili Konular ve Kapsam Hakkında

Kullanıcı Ekleme ve Kullanıcı Girişleri

## Gereksinim / Fikir

Potansiyel müşterilerimizin birçoğu personelleri için local LDAP Servislerine ve/veya Microsoft-Google gibi 3. Taraf hesaplara sahiplerdir. EduHR sistemine giriş yapacak kullanıcılarının ayrıbir hesap bilgisine sahip olmasındansa, mevcuttaki bu bilgileri ile sisteme giriş yapabilmeyi tercih etmektedirler. (Kullanıcı adı ve parola gibi)   
Söz konusu ihtiyaç doğrultusunda EduHR yapısının bunu desteklemesi gerekir.   
Önceliğimiz Google Workspace ile çalışmalarını yürüten müşterilerimizin Google Workspace’de tanımlanmış e posta ve şifrelerini EduHR Login ekranlarında girmeleri durumunda sistemin buna uygun olarak giriş mekanizmaları ve kontrolleri sağlaması, uygunsa girişi tamamlayarak devam etmesidir.   
Özetle, Müşteri kullanıcılarının Google’da kayıtlı epostaları ve şifreleri ile EduHR sistemine girşini nasıl sağlarız. Gereksinimler, Yol haritası ve var olan yapıda yapılması gereken değişiklikler hakkında beni bilgilendirir misin?

## Tartışma (AI)

EduHR projesine harici kimlik sağlayıcılar (Identity Providers) ile giriş yapma yeteneği eklemek, modern bir SaaS uygulaması için stratejik ve kullanıcı dostu bir adımdır. Özellikle Google Workspace kullanan kurumlar için bu, adaptasyonu kolaylaştıran kritik bir özelliktir.

İsteğinizi analiz ettiğimde, temel ihtiyacın kullanıcıların Google e-posta ve şifrelerini *doğrudan EduHR login ekranına yazarak* sisteme girmesi olduğu belirtilmiş. Ancak bu yaklaşım, hem güvenlik hem de teknik açıdan modern standartların dışında kalmaktadır ve "Resource Owner Password Credentials (ROPC)" olarak bilinen, artık tavsiye edilmeyen bir akışa benzemektedir.

**Doğru ve Güvenli Yaklaşım: OpenID Connect (OIDC) ve OAuth 2.0 Yönlendirme Akışı**

Kullanıcıların Google şifresini sizin sisteminize girmesi yerine, endüstri standardı olan ve Google'ın da zorunlu kıldığı **OIDC (OpenID Connect)** protokolünü kullanmalıyız. Bu yöntemin işleyişi şöyledir:

1. **Giriş Talebi:** Kullanıcı, EduHR giriş ekranındaki "Google ile Giriş Yap" butonuna tıklar.
2. **Yönlendirme:** EduHR, kullanıcıyı güvenli bir şekilde Google'ın kendi giriş sayfasına yönlendirir.
3. **Kimlik Doğrulama:** Kullanıcı, e-posta ve şifresini **doğrudan Google'ın sunucularında** girer. EduHR bu şifreyi asla görmez ve saklamaz.
4. **Onay (Consent):** Google, kullanıcıdan "EduHR uygulamasının kimlik bilgilerinize (profil, e-posta vb.) erişmesine izin veriyor musunuz?" diye sorar.
5. **Geri Dönüş ve Token:** Kullanıcı onay verdikten sonra Google, kullanıcıyı tekrar EduHR'a yönlendirir. Bu yönlendirme sırasında, kullanıcının kimliğini doğrulayan güvenli bir "ID Token" ve "Access Token" da gönderilir.
6. **Girişin Tamamlanması:** EduHR, bu token'ı alır, geçerliliğini kontrol eder ve içindeki kullanıcı bilgileriyle (e-posta adresi gibi) kendi veritabanındaki kullanıcıyı eşleştirir. Eşleşme başarılıysa oturumu başlatır. ***Eğer kullanıcı sistemde yoksa, ilk girişinde otomatik olarak bir kullanıcı profili oluşturulabilir. (DİKKAT)***

Bu yaklaşım, şifrelerinizi asla talep etmediği için hem son kullanıcılar hem de kurumsal müşteriler için çok daha güvenlidir ve güven verir.

1. **Gereksinimler**
2. **Google Cloud Platform (GCP) Yapılandırması:**
   * Bir Google Cloud Projesi oluşturulmalıdır.
   * "APIs & Services > Credentials" bölümüne gidilmelidir.
   * **OAuth consent screen (Onay ekranı):** Uygulama adı (EduHR), kullanıcı destek e-postası, logo gibi bilgiler burada yapılandırılır. Müşterilerinizin göreceği ekran budur.
   * **Credentials (Kimlik Bilgileri):** Yeni bir "OAuth 2.0 Client ID" oluşturulmalıdır.
     + Application Type: "Web application" seçilmelidir.
     + **Authorized redirect URIs:** Google'ın kimlik doğrulama sonrası kullanıcıyı geri yönlendireceği EduHR API/WebApp adresleri buraya eklenmelidir. (Örn: https://api.eduhr.com/signin-google, https://app.eduhr.com/signin-google).
   * Oluşturma işlemi sonunda size bir ClientId ve ClientSecret verilecektir. Bunlar kesinlikle güvende tutulmalıdır.
3. **EduHR Projesi Teknik Gereksinimleri:**
   * **NuGet Paketi:** EduHR.Api ve EduHR.Webapp projelerine Microsoft.AspNetCore.Authentication.Google paketi eklenmelidir.
   * **Konfigürasyon Yönetimi:** Google tarafından sağlanan ClientId ve ClientSecret bilgileri, appsettings.json veya daha güvenli bir yöntem olan Azure Key Vault / .NET User Secrets gibi bir yerde saklanmalıdır.
4. **Yol Haritası**

Bu entegrasyonu aşamalı bir planla hayata geçirebiliriz:

**Adım 1: Kurulum ve Yapılandırma (1-2 gün)**

* Yukarıda belirtilen Google Cloud Platform (GCP) adımlarının tamamlanması.
* ClientId ve ClientSecret'ın geliştirme ortamı için appsettings.Development.json dosyasına eklenmesi.
* EduHR.Api ve EduHR.Webapp projelerine gerekli NuGet paketinin yüklenmesi.

**Adım 2: Backend Geliştirmesi (API Katmanı) (3-5 gün)**

* EduHR.Api projesinin Program.cs dosyasında Google ile kimlik doğrulama servisinin yapılandırılması.
* Google'dan dönen kullanıcı bilgileriyle sistemdeki kullanıcıyı eşleştirecek veya yeni kullanıcı yaratacak mantığın Application katmanında oluşturulması (Örn: AuthenticateGoogleUserCommand).
* users tablosunda harici girişleri (provider\_name, provider\_key) saklamak için gerekli veritabanı şeması değişikliklerinin yapılması.
* Kimlik doğrulama akışını yönetecek yeni API endpoint'lerinin (/api/auth/google-login, /api/auth/google-callback) Presentation katmanında oluşturulması.

**Adım 3: Frontend Geliştirmesi (WebApp & Mobil) (2-3 gün)**

* EduHR.Webapp login sayfasına ve gelecekteki Flutter uygulamasının giriş ekranına "Google ile Giriş Yap" butonunun eklenmesi.
* Bu butona tıklandığında backend'deki /api/auth/google-login endpoint'ine yönlendirme yapılması.

**Adım 4: Test ve Dağıtım (2-3 gün)**

* Hem başarılı giriş senaryolarının hem de hata durumlarının (örn: kullanıcının Google'da izin vermemesi) test edilmesi.
* Geliştirme ve test ortamlarında yapılan değişikliklerin main branch'ine "Conventional Commits" standartlarına uygun olarak merge edilmesi.
* Canlı ortama dağıtımın yapılması ve ClientId / ClientSecret'ın canlı ortam için güvenli bir şekilde yapılandırılması.

1. **Mevcut Mimaride Yapılması Gereken Değişiklikler**

Mevcut Temiz Mimari, DDD ve CQRS yapınız bu tür bir değişiklik için oldukça uygundur. Değişiklikler katmanlara şu şekilde yansıyacaktır:

1. **1. Presentation Katmanı (EduHR.Api, EduHR.Webapp)**

* **Program.cs (veya Startup.cs) içine Konfigürasyon:**

C#

// In Program.cs

builder.Services.AddAuthentication()

.AddGoogle(options =>

{

options.ClientId = builder.Configuration["Authentication:Google:ClientId"];

options.ClientSecret = builder.Configuration["Authentication:Google:ClientSecret"];

// CallbackPath, Google'ın kullanıcıyı geri yönlendireceği yolu belirtir.

// Bu, GCP'de belirttiğiniz redirect URI ile eşleşmelidir.

options.CallbackPath = "/signin-google";

});

* **appsettings.json:**

JSON

{

"Authentication": {

"Google": {

"ClientId": "YOUR\_GOOGLE\_CLIENT\_ID\_HERE",

"ClientSecret": "YOUR\_GOOGLE\_CLIENT\_SECRET\_HERE"

}

}

}

* **Authentication Controller:** Google giriş akışını başlatacak ve callback'i (geri dönüşü) yönetecek endpoint'ler eklenir.

1. **2. Application Katmanı**

* **Yeni Komutlar (Commands):** Google'dan gelen token ile bir kullanıcıyı sisteme dahil etmek için bir command oluşturulabilir.
  + public class LoginWithGoogleCommand : IRequest<AuthResult> { public string IdToken { get; set; } }
* **İş Mantığı:** Bu komutun Handler'ı, Google'dan gelen token'ı doğrular, içindeki e-posta adresini alır. Bu e-postaya sahip bir kullanıcı veritabanında var mı diye kontrol eder.
  + Eğer varsa, kullanıcının harici giriş bilgileri (provider: "Google", providerKey: google\_user\_id) güncellenir ve JWT token üretilerek döndürülür.
  + Eğer yoksa, yeni bir User entity'si oluşturulur, veritabanına kaydedilir ve ardından JWT token üretilir. Bu "Just-In-Time Provisioning" olarak adlandırılır.

1. **3. Domain Katmanı**

* **User Entity:** User veya AppUser gibi ana kullanıcı entity'nize, harici girişleri desteklemek için yeni özellikler eklenmesi gerekebilir. ASP.NET Core Identity kullanıyorsanız, IdentityUserLogin<TKey> tablosu bu işi zaten yapar. Eğer kendi kullanıcı yönetiminizi yapıyorsanız, User entity'nize benzer alanlar eklemeniz gerekir.

1. **4. Infrastructure Katmanı**

* PostgreSQL Veritabanı Değişiklikleri:

Mevcut users tablonuzla ilişkili bir tabloya veya doğrudan users tablosuna harici giriş bilgilerini eklemeniz gerekecektir. ASP.NET Core Identity kullanmıyorsanız, önerilen yaklaşım ayrı bir tablodur:

**user\_external\_logins Tablosu:**

SQL

CREATE TABLE user\_external\_logins (

id BIGSERIAL PRIMARY KEY,

user\_id BIGINT NOT NULL,

provider\_name VARCHAR(100) NOT NULL, -- Örn: 'Google', 'Microsoft'

provider\_key TEXT NOT NULL, -- Google'dan gelen benzersiz kullanıcı kimliği

created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT fk\_user\_external\_logins\_user\_id FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE

);

-- Benzersizliği sağlamak için index

CREATE UNIQUE INDEX uix\_user\_external\_logins\_provider ON user\_external\_logins (provider\_name, provider\_key);

-- PostgreSQL standartlarınıza uygun dokümantasyon

COMMENT ON TABLE user\_external\_logins IS 'Kullanıcıların Google, Microsoft gibi harici kimlik sağlayıcılarla olan bağlantılarını saklar.';

COMMENT ON COLUMN user\_external\_logins.user\_id IS 'Sistemdeki ana kullanıcı kaydının referansı (users.id).';

COMMENT ON COLUMN user\_external\_logins.provider\_name IS 'Harici kimlik sağlayıcının adı (Örn: Google).';

COMMENT ON COLUMN user\_external\_logins.provider\_key IS 'Harici kimlik sağlayıcı tarafından verilen, kullanıcıya özel benzersiz anahtar.';

1. **Diğer Kimlik Sağlayıcılar (LDAP & Microsoft)**

Bu mimariyi kurduktan sonra, gelecekte Microsoft, Apple ve hatta kurumsal LDAP sunucuları için destek eklemek çok daha kolay olacaktır. ASP.NET Core Authentication kütüphanesi, farklı sağlayıcıları (.AddMicrosoftAccount, .AddOpenIdConnect vb.) kolayca eklemenize olanak tanır. Her biri için benzer bir yapılandırma ve mantık izlenir. Bu, sisteminizi geleceğe hazır hale getirir.

Özetle, kullanıcı şifresini doğrudan istemek yerine OIDC/OAuth 2.0 yönlendirme akışını benimsemek, projenizi daha güvenli, standartlara uygun ve ölçeklenebilir hale getirecektir. Bu, hem müşterilerinizin güvenlik beklentilerini karşılar hem de EduHR platformunun profesyonel imajını güçlendirir.

# Tasarım Tailwinds

## İlgili Konular ve Kapsam Hakkında

WebApp ve MobileApp Tasarımı (Özelleştirilebilir Tema ve genel görünümler)

## Gereksinim / Fikir

EduHR-2025 Projesi için tasarımlarda (WebApp ve belki de MobileApp) Tailwind CSS - Rapidly build modern websites without ever leaving your HTML design öğelerini kullanmayı düşünüyorum. İzlemem gereken yol haritasını tüm ayrıntıları ile anlatır mısın?

## Tartışma (AI)

Harika bir seçim! Tailwind CSS, EduHR Projesi gibi modern, bileşen tabanlı ve hızlı geliştirilmesi hedeflenen projeler için son derece uygun bir CSS framework'üdür. Özellikle ASP.NET Core Razor Pages ile entegrasyonu ve component-bazlı bir yaklaşımla birleştiğinde size hem geliştirme hızı hem de sürdürülebilirlik kazandıracaktır.

Projenizin mimarisi, hedefleri ve teknolojileri göz önüne alındığında, Tailwind CSS'i WebApp ve gelecekteki MobileApp'e entegre etmek için izlemeniz gereken detaylı yol haritası aşağıda sunulmuştur.

1. **Yol Haritası: EduHR Projesinde Tailwind CSS Kullanımı**

Bu yol haritası, hazırlıktan başlayarak ileri seviye konulara kadar tüm süreci kapsamaktadır.

1. **Faz 0: Hazırlık ve Strateji Belirleme**

Bu aşama, kod yazmaya başlamadan önceki temel yapılandırmayı içerir.

1. **Node.js ve npm Kurulumu:**
   * Tailwind CSS, bir PostCSS eklentisidir ve Node.js ekosistemindeki araçlarla (npm veya yarn) yönetilir. Geliştirme makinenizde (Intel Core i7, 16GB RAM, Windows 10) Node.js'in güncel bir LTS sürümünün kurulu olduğundan emin olun. Bu kurulum, npm paket yöneticisini de beraberinde getirecektir.
2. **Proje Yapısına Entegrasyon:**
   * Tailwind konfigürasyon dosyaları, EduHR.Webapp projesinin kök dizininde yer alacaktır. Bu, package.json, tailwind.config.js ve postcss.config.js gibi dosyaların EduHR.Webapp.csproj dosyasıyla aynı seviyede olması gerektiği anlamına gelir.
3. **Tasarım Sistemi ve Tema (tailwind.config.js) Oluşturma:**
   * İşe başlamadan önce EduHR markasının temel görsel kimliğini belirleyin. Bu, projenin tamamında tutarlılığı sağlayacaktır.
   * **Renkler:** Ana renkler (primary, secondary), uyarı renkleri (success, error, warning), ve nötr gri tonları belirleyin.
   * **Yazı Tipleri (Fonts):** Projede kullanılacak yazı tiplerini (örneğin, Inter, Roboto) tanımlayın.
   * **Boşluklar (Spacing):** Tutarlı arayüzler için kullanılacak standart boşluk ve boyutlandırma birimlerini (spacing scale) tanımlayın.
   * Bu tanımlamaları tailwind.config.js dosyasının theme.extend bölümünde yapacaksınız. Bu sayede Tailwind'in varsayılanlarını ezmek yerine onlara ekleme yapmış olursunuz.
4. **Faz 1: ASP.NET Core WebApp (Razor Pages) Entegrasyonu**

Bu faz, yönetim paneliniz olan EduHR.Webapp projesine Tailwind'i A'dan Z'ye entegre etmeyi kapsar.

1. **Tailwind CSS Kurulumu:**
   * Terminali src/Presentation/EduHR.Webapp klasörünün kök dizininde açın.
   * npm init -y komutu ile bir package.json dosyası oluşturun.
   * npm install -D tailwindcss postcss autoprefixer komutu ile gerekli paketleri geliştirme bağımlılığı olarak kurun.
   * npx tailwindcss init -p komutu ile hem tailwind.config.js hem de postcss.config.js dosyalarını oluşturun.
2. **Konfigürasyon Dosyalarını Düzenleme:**
   * **tailwind.config.js:** Bu dosya en kritik olanıdır. content özelliğini, Tailwind'in hangi dosyalardaki class'ları tarayacağını bilecek şekilde yapılandırın:

JavaScript

/\*\* @type {import('tailwindcss').Config} \*/

module.exports = {

content: [

'./Pages/\*\*/\*.cshtml', // Razor Pages dosyalarını tara

'./Views/\*\*/\*.cshtml', // MVC view'lerini (varsa) ve partial'ları tara

'./Areas/\*\*/\*.cshtml', // Alanları (Areas) tara

'./wwwroot/js/\*\*/\*.js' // JavaScript dosyalarını tara

],

theme: {

extend: {

// Faz 0'da belirlediğiniz marka renkleri, fontlar vb. buraya eklenecek.

colors: {

'primary': '#005A9C',

'secondary': '#6C757D',

// ...diğer renkler

}

},

},

plugins: [

require('@tailwindcss/forms'), // Form elemanlarını güzelleştirmek için harika bir eklenti

],

}

* + **postcss.config.js:** Bu dosya genellikle varsayılan haliyle yeterlidir:

JavaScript

module.exports = {

plugins: {

tailwindcss: {},

autoprefixer: {},

},

}

1. **CSS Dosyalarını Oluşturma ve Yapılandırma:**
   * EduHR.Webapp projesi içinde (örneğin /Styles adında bir klasör oluşturabilirsiniz) bir app.css (veya input.css) dosyası oluşturun.
   * Bu dosyanın içeriği şu şekilde olmalıdır:

CSS

@tailwind base;

@tailwind components;

@tailwind utilities;

/\* Kendi özel component sınıflarınızı buraya ekleyebilirsiniz, örn: \*/

/\* @layer components {

.btn-primary {

@apply bg-primary text-white font-bold py-2 px-4 rounded;

}

} \*/

1. **Derleme (Build) Sürecini Otomatize Etme:**
   * package.json dosyanıza scripts bölümünü ekleyerek CSS derleme işlemini kolaylaştırın.

JSON

"scripts": {

"css:build": "postcss ./Styles/app.css -o ./wwwroot/css/app.css --minify",

"css:watch": "postcss ./Styles/app.css -o ./wwwroot/css/app.css --watch"

},

* + - css:build: Üretim ortamı için küçültülmüş (minified) CSS dosyasını oluşturur.
    - css:watch: Geliştirme sırasında dosya değişikliklerini izler ve CSS'i otomatik olarak yeniden derler.

1. **Layout Dosyasına CSS'i Ekleme:**
   * Pages/Shared/\_Layout.cshtml dosyanızda, <head> etiketleri arasına derlenmiş CSS dosyanızın referansını ekleyin:

HTML

<link rel="stylesheet" href="~/css/app.css" asp-append-version="true" />

asp-append-version="true" tag helper'ı, tarayıcı önbellekleme sorunlarını engeller.

1. **Geliştirme Akışı:**
   * Bir terminalde dotnet watch run komutu ile web sunucunuzu çalıştırın.
   * **Ayrı bir terminalde** npm run css:watch komutunu çalıştırın.
   * Artık .cshtml dosyalarınıza Tailwind class'ları eklediğinizde, değişiklikler anında derlenip tarayıcıda görünecektir.
2. **Faz 2: Bileşen (Component) Tabanlı Tasarım ve Standartlaştırma**

Bu faz, kod tekrarını önlemek ve tutarlı bir kullanıcı arayüzü (UI) oluşturmak içindir. Temiz Mimari prensiplerinizle uyumludur.

1. **Razor Partial Views ve View Components Kullanımı:**
   * Tekrar eden UI elemanlarını (butonlar, kartlar, modallar, form alanları vb.) **Partial Views** (\_Button.cshtml, \_Card.cshtml) veya **View Components** olarak oluşturun.
   * Bu bileşenlerin içine gerekli Tailwind class'larını yazın. Bu sayede, örneğin bir buton kullanmanız gerektiğinde <button class="py-2 px-4 ..."> yazmak yerine @await Html.PartialAsync("\_PrimaryButton", new ButtonViewModel { Text = "Kaydet" }) gibi temiz bir yapı kullanırsınız.
2. **@apply Direktifini Stratejik Kullanma:**
   * Çok sık tekrarlanan ve uzun class kombinasyonları için app.css dosyanızda @apply kullanarak özel, yeniden kullanılabilir sınıflar oluşturun.
   * Örnek:

CSS

@layer components {

.card {

@apply bg-white rounded-lg shadow-md p-4;

}

.input-text {

@apply block w-full rounded-md border-gray-300 shadow-sm focus:border-primary focus:ring-primary sm:text-sm;

}

}

* + Bu, HTML'inizi temiz tutar ve tasarım sisteminize bağlı kalmanızı sağlar.

1. **Faz 3: İleri Seviye Konular ve Üretim Ortamı**
2. **PurgeCSS Optimizasyonu:**
   * Tailwind'in en güçlü özelliklerinden biri, kullanılmayan tüm CSS stillerini üretim (production) build'inde temizlemesidir. Bu, tailwind.config.js dosyasındaki content ayarı sayesinde otomatik olarak çalışır.
   * CI/CD pipeline'ınızda projenizi yayınlarken npm run css:build komutunu çalıştırdığınızdan emin olun. Bu, app.css dosyanızın boyutunu dramatik şekilde küçültecektir.
3. **SaaS için Tema Desteği (İsteğe Bağlı ama Önerilir):**
   * Projeniz bir SaaS olduğu için, gelecekte kiracıların (abonelerin) kendi marka renklerini kullanmasını isteyebilirsiniz.
   * Bunu CSS değişkenleri ile zarif bir şekilde çözebilirsiniz.
   * \_Layout.cshtml içinde veya kiracıya özel bir CSS dosyasında:

HTML

<style>

:root {

--color-primary: #<%\_TENANT\_COLOR\_%>; /\* Bu değer DB'den gelir \*/

}

</style>

* + tailwind.config.js içinde:

JavaScript

theme: {

extend: {

colors: {

// 'var(--color-primary)' fonksiyonunu kullanarak CSS değişkenini çağırın

primary: 'var(--color-primary)',

}

}

}

* + Bu yaklaşımla, sadece bir CSS değişkenini değiştirerek tüm sitenin ana rengini kiracıya özel hale getirebilirsiniz.

1. **Faz 4: Flutter (MobileApp) için Yol Haritası**

Flutter, HTML/CSS kullanmaz; kendi widget tabanlı render motoruna sahiptir. Bu nedenle **Tailwind CSS'i doğrudan Flutter'da kullanamazsınız.** Ancak, Tailwind'in *felsefesini ve tasarım sistemini* Flutter'a taşıyabilirsiniz.

1. **Felsefeyi Benimseme:**
   * Tailwind'in "utility-first" yaklaşımı, küçük ve birleştirilebilir stillerle UI oluşturma fikrine dayanır. Flutter'da da bu mümkündür.
2. **Tasarım Sistemini Paylaşma:**
   * En doğru ve sürdürülebilir yaklaşım, tailwind.config.js dosyasında tanımladığınız tasarım belirteçlerini (design tokens: renkler, boşluklar, font boyutları vb.) Flutter projenizde yeniden tanımlamaktır.
   * Flutter'ın ThemeData'sını genişleterek veya özel bir AppTheme sınıfı oluşturarak bu değerleri merkezi bir yerden yönetin.
   * **Örnek:**
     + Tailwind'de colors.primary olarak tanımladığınız rengi, Flutter'da AppColors.primary olarak tanımlayın.
     + Tailwind'deki spacing.4 (1rem/16px) değerini, Flutter'da AppSpacing.medium gibi bir sabite atayın.
3. **Yardımcı Paketleri Değerlendirme:**
   * Flutter ekosisteminde, Tailwind'e benzer bir "utility-style" sözdizimi sunan paketler mevcuttur. Örneğin, **VelocityX** veya benzeri paketler, Text("Hello").p(16).bold().make() gibi zincirleme metotlarla stil vermenizi sağlar.
   * Bu, geliştirme hızını artırabilir ancak Flutter'ın doğal widget kompozisyonu yaklaşımından biraz uzaklaşmanıza neden olabilir. Kullanıp kullanmamak bir ekip kararıdır.

**Öneri:** Flutter için en temiz yol, WebApp'deki görsel tutarlılığı sağlamak amacıyla **özel, yeniden kullanılabilir widget'lar** (PrimaryButton, StyledCard, InputField vb.) oluşturmaktır. Bu widget'lar stillerini merkezi AppTheme'den almalıdır. Bu, projenizin Temiz Mimari ve sürdürülebilirlik hedefleriyle tam olarak örtüşür.

Bu kapsamlı yol haritasını izleyerek EduHR projenizin hem WebApp hem de gelecekteki Mobil Uygulamasında tutarlı, modern ve bakımı kolay arayüzler geliştirebilirsiniz. Başlangıçta WebApp entegrasyonuna odaklanıp sistemi oturttuktan sonra Flutter tarafına geçmek en mantıklı ilerleme olacaktır.

# Dosya Yükleme/Depolama

## İlgili Konular ve Kapsam Hakkında

File Upload, File Storage, Object Storage Service

## Gereksinim / Fikir

EduHR-2025 Projesinde ihtiyaç duyulan Dosya Kaydetme unsurları için Nesne Depolama Servisleri kullanılacak. Bu noktada MinIO kullanılması kararı alınmıştır. (Çok Kiracılı (Multi-Tenancy) Yapıya olan uygunluğu sebebi ile)

* Kova (Bucket) Oluşturma
* Kullanıcı ve Erişim Yönetimi
* Uygulama Entegrasyonu
* Yüksek Erişilebilirlik (High Availability) ve Yedeklilik
* Güvenlik
* Güncelleme ve Bakım konularını dikkate almamız gerekir.

Yukarıdakilere göre EduHR-2025 projesine nasıl entegrasyon sağlarız? Şuana kadar yaptığımız çalışmaları dikkate alarak nelerde değişiklik yapmamız gerekir. İki ayrı planlama yapar mısın? (Birincisi bundan sonrası için, ikincisi bundan önceki çalışmalara yönelik olmalı)

## Tartışma (AI)

Harika bir karar. MinIO, S3 uyumlu API'si, açık kaynak olması ve özellikle çok kiracılı (multi-tenancy) mimarilerde sunduğu esneklik ile EduHR projenizin gereksinimleri için son derece uygun bir nesne depolama servisidir. Belirttiğiniz konuları (kova yönetimi, güvenlik, entegrasyon vb.) dikkate alarak projenize nasıl entegre edeceğimize dair detaylı bir eylem planı hazırladım.

Mevcut mimariniz (Temiz Mimari, DDD, CQRS) ve teknolojileriniz (.NET 9, PostgreSQL) göz önünde bulundurulduğunda, bu entegrasyon veri yönetimi stratejinizi önemli ölçüde modernize edecektir.

İşte size özel olarak hazırlanan iki aşamalı plan:

* **Plan A: Geleceğe Yönelik Yeni Geliştirmeler İçin Eylem Planı**

Bu plan, bugünden itibaren yapılacak tüm yeni dosya depolama işlemleri için MinIO'nun nasıl kullanılacağını ve mimariye nasıl entegre edileceğini kapsar.

* **1. Altyapı Kurulumu ve Konfigürasyon**
* **Yerel Geliştirme Ortamı:** docker-compose.yml dosyanıza bir MinIO servisi ekleyerek başlayın. Bu, her geliştiricinin kendi izole nesne depolama ortamına sahip olmasını sağlar.

YAML

version: '3.8'

services:

# ... diğer servisleriniz (postgres, api vb.)

minio:

image: minio/minio:RELEASE.2024-07-30T10-22-29Z

container\_name: eduhrcorp-minio

ports:

- "9000:9000" # API Port

- "9001:9001" # Console Port

volumes:

- minio\_data:/data

environment:

MINIO\_ROOT\_USER: 'burayaGucluBirKullaniciAdi' # Geliştirme için .env dosyasından alınabilir

MINIO\_ROOT\_PASSWORD: 'burayaGucluBirSifre' # Geliştirme için .env dosyasından alınabilir

command: server /data --console-address ":9001"

volumes:

minio\_data:

* **Production Ortamı:** Yüksek Erişilebilirlik (High Availability) için MinIO'yu en az 4 sunuculuk dağıtık (Distributed) modda kurmayı hedefleyin. Bu, donanım arızalarına karşı hem koruma (redundancy) hem de performans artışı sağlar.
* **Konfigürasyon (appsettings.json):** MinIO bağlantı bilgilerinizi Presentation/EduHR.Api projesinin appsettings.json dosyasına ekleyin.

JSON

"MinioSettings": {

"Endpoint": "localhost:9000", // Docker için

"AccessKey": "minio-access-key",

"SecretKey": "minio-secret-key",

"UseSsl": false

}

* **2. Mimari Entegrasyonu (Clean Architecture)**

Bu entegrasyon, Bağımlılıkların Tersine Çevrilmesi (Dependency Inversion) prensibine tam uyumlu olacaktır.

* **Domain Katmanı (/src/Domain):**
  + Bu katman MinIO'dan haberdar OLMAYACAKTIR.
  + Personel özlük dosyası, avans talep belgesi gibi dosyaları temsil etmek için bir Document veya FileAttachment gibi bir entity/value object oluşturabilirsiniz. Bu nesne, dosyanın kendisini değil, meta verilerini tutar: FileName, ContentType, FileSize ve MinIO'da tutulacak nesnenin referansı olan ObjectKey veya StoragePath.
* **Application Katmanı (/src/Application):**
  + Dosya depolama işlemleri için bir soyutlama (interface) tanımlayın. Bu, sistemin belirli bir depolama teknolojisine (MinIO) olan bağımlılığını ortadan kaldırır.
  + Yeni Arayüz (Interface) Oluşturma:

src/Application/Common/Interfaces/IFileStorageService.cs

C#

// Naming Conventions: PascalCase, I prefix

public interface IFileStorageService

{

/// <summary>

/// Uploads a file to the object storage.

/// </summary>

/// <param name="bucketName">The name of the bucket.</param>

/// <param name="objectName">The unique name for the object in the bucket.</param>

/// <param name="stream">The file stream.</param>

/// <param name="contentType">The content type of the file.</param>

/// <returns>A task representing the asynchronous operation.</returns>

Task UploadAsync(string bucketName, string objectName, Stream stream, string contentType);

/// <summary>

/// Generates a presigned URL to temporarily access a file.

/// </summary>

/// <param name="bucketName">The name of the bucket.</param>

/// <param name="objectName">The name of the object.</param>

/// <param name="expiresInSeconds">The duration for which the URL is valid.</param>

/// <returns>A temporary URL to access the file.</returns>

Task<string> GetPresignedUrlAsync(string bucketName, string objectName, int expiresInSeconds = 3600);

/// <summary>

/// Deletes a file from the object storage.

/// </summary>

/// <param name="bucketName">The name of the bucket.</param>

/// <param name="objectName">The name of the object to delete.</param>

/// <returns>A task representing the asynchronous operation.</returns>

Task DeleteAsync(string bucketName, string objectName);

}

* **Infrastructure Katmanı (/src/Infrastructure):**
  + IFileStorageService arayüzünü MinIO kullanarak implemente edin.
  + **NuGet Paketleri:** Projenize Minio.AspNetCore paketini ekleyin.
  + Yeni Sınıf (Class) Oluşturma:

src/Infrastructure/Services/MinioStorageService.cs

C#

// Naming Conventions: PascalCase

public class MinioStorageService : IFileStorageService

{

private readonly IMinioClient \_minioClient;

private readonly ILogger<MinioStorageService> \_logger; // \_privateField: camelCase, \_ prefix

public MinioStorageService(IMinioClient minioClient, ILogger<MinioStorageService> logger)

{

\_minioClient = minioClient;

\_logger = logger;

}

// Interface metotlarının implementasyonu...

public async Task UploadAsync(string bucketName, string objectName, Stream stream, string contentType)

{

// Bucket yoksa oluşturma mantığı eklenebilir.

var beArgs = new BucketExistsArgs().WithBucket(bucketName);

bool found = await \_minioClient.BucketExistsAsync(beArgs);

if (!found)

{

var mbArgs = new MakeBucketArgs().WithBucket(bucketName);

await \_minioClient.MakeBucketAsync(mbArgs);

}

var putObjectArgs = new PutObjectArgs()

.WithBucket(bucketName)

.WithObject(objectName)

.WithStreamData(stream)

.WithObjectSize(stream.Length)

.WithContentType(contentType);

await \_minioClient.PutObjectAsync(putObjectArgs);

}

// ... diğer metotlar (GetPresignedUrlAsync, DeleteAsync)

}

* + **Dependency Injection:** EduHR.Api projesinde bu servisi DI konteynerine kaydedin.
* **3. Çok Kiracılı (Multi-Tenancy) Stratejisi**
* **Kova (Bucket) Yönetimi:** Her kiracı (Abone) için **ayrı bir bucket** oluşturun. Bu, en yüksek seviyede veri izolasyonu ve güvenlik sağlar.
  + **Bucket İsimlendirme Kuralı:** eduhrcorp-{tenantId}. tenantId olarak abonenin GUID'ini veya daha kısa, benzersiz bir tanımlayıcısını kullanın.
  + **Otomasyon:** Yeni bir Abone (MO\_913) oluşturulduğunda, bu iş akışının bir parçası olarak IFileStorageService üzerinden otomatik olarak ilgili kiracının bucket'ı da oluşturulmalıdır.
* **Kullanıcı ve Erişim Yönetimi:** Her kiracının bucket'ı için MinIO'da ayrı bir erişim politikası (policy) ve mümkünse ayrı servis kullanıcıları oluşturun. Uygulamanız, bir kiracının işlemi sırasında sadece o kiracıya ait bucket'a erişim yetkisi olan kimlik bilgileriyle MinIO'ya bağlanmalıdır. Bu, bir kiracının verisinin başka bir kiracının verisine erişmesini imkansız hale getirir.
* **4. Güvenlik**
* **Güvenli Erişim:** Frontend uygulamalarının (Flutter, React) dosyalara doğrudan erişmesi için **her zaman** GetPresignedUrlAsync metodu ile oluşturulan süreli ve kısıtlı URL'leri kullanın. Asla AccessKey ve SecretKey'i istemci tarafına göndermeyin.
* **Bucket Politikaları:** Varsayılan olarak tüm bucket'ları özel (private) olarak ayarlayın. Sadece uygulamanızın erişimine izin veren katı politikalar yazın.
* **Plan B: Mevcut Çalışmaların Adaptasyonu İçin Refaktör Planı**

Bu plan, şu ana kadar geliştirdiğiniz ve dosyaları potansiyel olarak veritabanında (bytea) veya sunucu dosya sisteminde tutan modüllerin MinIO kullanacak şekilde nasıl güncelleneceğini açıklar.

* **1. Veritabanı Şeması Değişikliği (PostgreSQL)**
* **Analiz:** Mevcutta dosya verisini (byte[] veya dosya yolu string) tutan tüm tabloları tespit edin. Örneğin, personnel\_files, advance\_request\_documents gibi.
* **Refaktör:** Bu tablolardaki dosya içeriğini tutan kolonu kaldırın ve yerine MinIO referanslarını tutacak kolonlar ekleyin.
  + **Mevcut (Potansiyel) Kolon:** file\_content (bytea) veya file\_path (varchar)
  + **Yeni Eklenecek Kolonlar (snake\_case):**
    - storage\_bucket\_name (varchar): Dosyanın bulunduğu MinIO bucket'ının adı. Örnek: eduhrcorp-tenant-123
    - storage\_object\_name (varchar): Dosyanın bucket içindeki benzersiz adı (Genellikle bir GUID kullanılır). Örnek: b2cb3a9f-a2e6-4a6c-9419-7d1a9f5d1aef.pdf
    - file\_name (varchar): Kullanıcının gördüğü orijinal dosya adı. Örnek: cv\_ahmet\_yilmaz.pdf
    - content\_type (varchar): Dosyanın MIME türü. Örnek: application/pdf
    - file\_size\_bytes (bigint): Dosya boyutu.
* **COMMENT'ler:** Yeni eklenen tüm kolonlar için COMMENT ON COLUMN ... IS '...' komutu ile Türkçe açıklamalar eklemeyi unutmayın.
* **2. Veri Migrasyonu (Tek Seferlik İşlem)**

Bu, en kritik adımdır ve dikkatle yapılmalıdır.

1. Migrasyon Script'i Yazma: .NET ile küçük bir konsol uygulaması veya bir script hazırlayın. Bu script aşağıdaki adımları izlemelidir:

a. Veritabanındaki tüm kiracıları (tenants) okuyun.

b. Her kiracı için MinIO'da yeni bucket'lar oluşturun (eduhrcorp-{tenantId}).

c. Dosya içeren tablodaki her bir kaydı döngüye alın.

d. Dosya içeriğini veritabanından (bytea) veya sunucu dosya sisteminden okuyun.

e. Bu içeriği, ilgili kiracının bucket'ına yeni bir object\_name (GUID) ile IFileStorageService.UploadAsync kullanarak yükleyin.

f. Başarılı yüklemenin ardından, veritabanındaki ilgili kaydı yeni storage\_bucket\_name ve storage\_object\_name bilgileriyle güncelleyin.

1. **Test:** Bu script'i önce test verileriyle dolu bir hazırlık (staging) ortamında defalarca çalıştırın.
2. **Uygulama:** Canlı sisteme geçiş sırasında, uygulamayı bakım moduna alarak bu script'i çalıştırın.
3. **Temizlik:** Migrasyon doğrulandıktan sonra eski file\_content veya file\_path kolonlarını veritabanından kaldırın (DROP COLUMN).

* **3. Kod Tabanını Refaktör Etme**
* Projenizde dosya okuma/yazma işleminin yapıldığı tüm servisleri ve command/query handler'larını bulun.
  + Örnek: CreatePersonnelCommandHandler, AddAdvanceRequestCommandHandler vb.
* Bu sınıflardaki eski dosya kaydetme mantığını söküp atın.
* Yerine, constructor üzerinden enjekte ettiğiniz IFileStorageService'i kullanın.
  + **Örnek (CreatePersonnelCommandHandler):**
    1. API'den IFormFile olarak gelen personel fotoğrafını alın.
    2. \_fileStorageService.UploadAsync(...) metodunu çağırarak dosyayı ilgili kiracının bucket'ına yükleyin.
    3. Dönen objectName ve bucketName'i veritabanına kaydedilecek Personnel entity'sinin ilgili alanlarına atayın.
* **4. Kullanıcı Yolculuklarının (User Journeys) Güncellenmesi**
* Belirttiğiniz kullanıcı yolculuklarındaki dosya ile ilgili adımlar artık yeni servisi kullanmalıdır:
  + **MO\_121 (Personel Kaydı Oluşturma):** Personelin özlük dosyaları (CV, kimlik fotokopisi vb.) artık MinIO'ya yüklenecek.
  + **MO\_611 (Maaş Bilgisi Girme):** Ücret pusulası (payslip) oluşturulup PDF olarak MinIO'da saklanabilir. Personel (A\_15), kendi pusulasını görüntülerken GetPresignedUrlAsync ile oluşturulan geçici bir link üzerinden dosyayı indirecektir.
  + **MO\_212 (İzin Talep Etme):** Mazeret izni gibi durumlarda kanıt belgesi (rapor vb.) eklenmesi gerekiyorsa, bu dosya da MinIO'ya kaydedilecektir.
* **Özet ve Sonuç**

Bu iki planlı entegrasyon, EduHR projesini dosya yönetimi açısından daha ölçeklenebilir, güvenli ve modern bir yapıya kavuşturacaktır.

* **Plan A**, gelecekteki geliştirmelerin en başından doğru bir mimariyle yapılmasını sağlar.
* **Plan B**, mevcut yatırımınızı koruyarak eski yapıyı yeni ve daha iyi bir yapıya dönüştürmek için net bir yol haritası sunar.

Bu geçiş, veritabanı üzerindeki yükü azaltacak, dosya erişimini daha güvenli hale getirecek ve SaaS modelinizin en önemli gereksinimlerinden biri olan **kiracılar arası tam veri izolasyonunu** nesne depolama seviyesinde de garanti altına alacaktır.