Fullstack Developer Teknik Mülakat Görevi: Öğrenci ve Ders Yönetimi Uygulaması

Amaç: Bu görev, bir Fullstack Developer adayının hem backend hem de frontend geliştirme becerilerini ölçmeyi amaçlayan kapsamlı bir web uygulaması projesidir. Adaydan, aşağıda belirtilen teknoloji seçenekleriyle, öğrenci ve ders yönetimini konu alan bir web uygulaması geliştirmesi beklenir. Proje boyunca güvenlik, yetkilendirme ve veri yönetimi konularında en iyi uygulamaların kullanılması önemlidir.

Teknoloji Seçenekleri

- **Backend:** Python (Django veya FastAPI), .NET Core, veya Node.js (Express.js veya Nest.js)
- **Frontend:** React.js
- **Veritabanı:** MSSQL, PostgreSQL veya MongoDB
- **Diğer:** Projenin Docker ile konteynerize edilmesi zorunludur (Docker Compose kullanılması önerilir).

Proje Konusu

Aday, *öğrenci ve ders yönetimi* ile ilgili bir web uygulaması geliştirecektir. Uygulamada kimlik doğrulama ve yetkilendirme mekanizmaları olmalı, farklı kullanıcı rolleri tanımlanmalı ve bu rollere göre farklı işlemlere izin verilmelidir. Veri girişleri ve işlemleri sırasında geçerli verilerin kullanılması, uygun doğrulamaların yapılması ve genel olarak temiz bir kod yapısı ile iyi veri yönetimi uygulamalarının uygulanması beklenmektedir.

Kullanıcı Hikayeleri (User Stories)

Aşağıdaki kullanıcı hikayeleri, uygulamanın temel işlevselliğini tanımlamaktadır:

1. **Öğrenci Yönetimi:**
    - Admin kullanıcı, yeni öğrenci ekleyebilir, mevcut bir öğrencinin bilgilerini güncelleyebilir ve öğrenci kaydını silebilir.
    - Öğrenci eklerken veya güncellerken, öğrencinin adı, soyadı, doğum tarihi gibi alanlar uygun şekilde doğrulanmalıdır. Örneğin, bu alanlar boş bırakılamaz ve doğum tarihi gelecekte bir tarih olamaz.
2. **Ders Yönetimi:**
    - Admin, sisteme yeni dersler ekleyebilir, ders bilgilerini güncelleyebilir ve dersleri silebilir.
    - Her dersin adı benzersiz olmalıdır; aynı isimde birden fazla ders eklenememelidir.
3. **Öğrenci-Ders Eşleştirmesi (Kayıt Olma):**
    - Öğrenci rolündeki kullanıcılar, kendi hesaplarıyla giriş yaptıktan sonra mevcut derslere kaydolabilir.
    - Aynı öğrencinin, aynı derse birden fazla kez kayıt olmaması gerekir. (Bir öğrenci bir dersi sadece bir kez alabilir.)

- Öğrenci, kayıt olduğu bir dersten çekilebilmeli (kaydını silebilmelidir).

4. **Listeleme ve Detay Sayfaları:**
    - Öğrenci listesi ve ders listesi sayfaları, sayfa başına belli sayıda kayıt gösterecek şekilde sayfalandırılmış (pagination) olmalıdır. Bu sayede liste çok uzun olduğunda kullanıcının verileri sayfa sayfa görmesi sağlanmalıdır.
    - Öğrenci listesinde bir öğrenci adına tıklandığında, o öğrenciye ait detay bilgiler (örneğin ad, soyad, doğum tarihi, kayıtlı olduğu dersler) bir modal/popup pencerede gösterilmelidir.
    - Benzer şekilde, ders listesinde de dersin detayları (örneğin dersin adı, dersi alan öğrenciler) görüntülenebilir.
    - Ayrıca, öğrenci-ders eşleşmeleri (hangi öğrenci hangi derslere kayıtlı) ayrı bir liste olarak gösterilmelidir.
5. **Kullanıcı Rolleri ve Yetkilendirme:**
    - Uygulamada en az iki rol bulunmaktadır: Admin ve Öğrenci (kullanıcı).
    - Admin rolündeki kullanıcılar, tüm öğrenci ekleme/güncelleme/silme, ders ekleme/güncelleme/silme ve öğrenci-ders eşleştirme (öğrenciye ders atama veya kaydını silme) işlemlerini yapabilir.
    - Öğrenci rolündeki kullanıcılar ise yalnızca kendi hesaplarıyla ilgili verileri görebilir. Kendi bilgilerini görüntüleyebilir/güncelleyebilir (örn. profil bilgileri), kayıt oldukları dersleri görebilir ve yeni derslere kaydolup mevcut ders kayıtlarını iptal edebilirler. Başka öğrencilerin veya genel sistemin verilerine erişimleri olmamalıdır.
    - Yetkilendirme kontrolleri kesinlikle backend tarafında uygulanmalıdır. (Frontend tarafında da görünüm kısıtlamaları olacak ancak bir kullanıcının aslında izin verilmemiş bir işlemi API üzerinden çağırmaya çalışması engellenmelidir.)
6. **Kimlik Doğrulama (Authentication):**
    - Uygulamada kimlik doğrulama JWT (JSON Web Token) veya benzeri bir mekanizma ile gerçekleştirilmelidir. Kullanıcılar (Admin ya da Öğrenci) sisteme kullanıcı adı/e-posta ve şifre ile giriş yapacaklardır.
    - Başarılı giriş yapan kullanıcıya, yetkilerini de içeren bir JWT veya oturum bilgisi verilmeli, bu sayede sonraki isteklerde bu token kullanılarak kullanıcının yetkisi doğrulanmalıdır.
    - Uygulamada güvenli çıkış (logout) işlemi olmalıdır. Kullanıcı logout olduğunda, token geçersiz kılınmalı veya client tarafında silinmelidir.

Ekstra Teknik Gereksinimler

- Proje, Docker kullanılarak konteynerleştirilmelidir. Tüm servislerin çalışması için tercihen bir Docker Compose yapılandırması sağlanmalıdır. (Adayın Docker kullanımı ve konteyner kavramlarına hakimiyeti görülecektir.)

- Frontend tarafında, React uygulaması için uygun bir state management çözümü kullanılmalıdır. (Örneğin Context API, Redux veya benzeri bir teknoloji ile uygulamanın durumu yönetilmelidir.)
- Frontend ile backend arasındaki API çağrıları güvenli bir şekilde ele alınmalıdır. Kimliği doğrulanmamış istekler backend tarafından reddedilmeli ve uygun yanıt kodları dönülmelidir. Tarayıcı tarafında da gerekli durumlarda kullanıcıyı giriş sayfasına yönlendirme gibi önlemler alınmalıdır.

Teslimat ve Değerlendirme Kriterleri

- Proje kodu, GitHub üzerinde bir repository olarak paylaşılabilecek formatta hazırlanmalıdır.. (README dosyası ile çalıştırma talimatları mutlaka sağlanmalıdır.)
- **Kod kalitesi önemlidir:** Gereksiz kod tekrarlarından kaçınılmalı, mümkün olduğunca yeniden kullanılabilir ve modüler bir yapı kurulmalıdır.
- **Kodlar okunaklı ve anlaşılır olmalıdır.** İyi isimlendirmeler kullanılmalı ve gerekliyse yorum satırları ile açıklamalar eklenmelidir.
- Backend tarafında, en azından temel fonksiyonları kapsayan birkaç adet test yazılması beklenmektedir (özellikle kritik API istekleri için örnek testler). Bu, adayın test yazma becerisini gösterecektir.
- Proje boyunca yapılan tercihler (seçilen teknoloji, mimari, kullanılan kütüphaneler vb.) ve projenin nasıl çalıştırılacağı/derleneceği ile ilgili bilgiler bir README dosyasında açıklanmalıdır.