



## **BLM4537 IOS ile Mobil Uygulama Geliştirme**

**Final Raporu**

**Ad Soyad: Eda Nur ARSLAN**

**Öğrenci No: 22290210**

**Öğretim Görevlisi: Enver BAĞCI**

**GitHub Link: <https://github.com/edanurarslan/Office-Reservation-System-Mobile>**

**Video Link: <https://www.youtube.com/watch?v=rLIBbfLjvCs>**

## **İÇİNDEKİLER**

- **BÖLÜM 1: Proje Özeti**
- **BÖLÜM 2: Kullanılan Teknolojiler**
- **BÖLÜM 3: Sistem Mimarisi**
- **BÖLÜM 4: Uygulama Özellikleri**
- **BÖLÜM 5: Arayüz ve Tasarım**
- **BÖLÜM 6: Yapılan Değişiklikler ve Geliştirmeler**
- **BÖLÜM 7: Testler ve Sonuçlar**
- **BÖLÜM 8: Proje Gelişim Analizi**
- **BÖLÜM 9: Sonuç ve Değerlendirme**

# 1. PROJE ÖZETİ

**Ofis Yönetim Sistemi**, modern iş dünyasında ofis kaynaklarını optimize etmek amacıyla geliştirilmiş, kurumsal düzeyde bir yönetim platformudur. Sistem; çalışma alanlarının, toplantı odalarının ve çalışan etkileşimlerinin dijital bir ekosistem üzerinden uçtan uca yönetilmesini sağlar.

## 1.1. Projenin Temel Amacı ve Vizyonu

Projenin ana odağı, şirketlerin sahip olduğu fiziksel ofis kaynaklarını verimli bir şekilde yöneterek, çalışanların bu kaynaklara erişimini kolaylaştırmak ve operasyonel maliyetleri minimize etmektir. Sistem, manuel ilerleyen rezervasyon süreçlerini tamamen dijitalleştirerek insan hatasını azaltmayı ve veri odaklı bir yönetim anlayışını benimsemeyi hedefler.

## 1.2. Rol Bazlı Erişim ve Kişiselleştirme

Uygulama, karmaşık kurumsal hiyerarşileri destekleyen **Rol Tabanlı Erişim Kontrolü** mekanizması üzerine inşa edilmiştir:

- Çalışanlar (Employee):** Kendi rezervasyonlarını kolayca yapabilir, QR kod teknolojisi ile hızlı check-in işlemlerini gerçekleştirebilir ve kişisel profil bilgilerini güncelleyebilir.
- Yöneticiler (Manager):** Sorumlu oldukları ekiplerin rezervasyon süreçlerini denetleyebilir, onay iş akışlarını yönetebilir ve birim bazlı kullanım raporlarına erişebilir.
- Sistem Yöneticileri (Admin):** Tüm ofis lokasyonlarını tanımlayabilir, sistem genelindeki iş kurallarını (maksimum rezervasyon süresi, çalışma saatleri vb.) belirleyebilir ve kapsamlı denetim günlüklerini inceleyebilir.

## 2. KULLANILAN TEKNOLOJİLER

Projenin teknoloji yığını, hem mobil istemci tarafında hem de sunucu katmanında yüksek performans ve ölçeklenebilirlik sunacak şekilde yapılandırılmıştır:

### **2.1. Frontend (Mobil Arayüz)**

- **Flutter & Dart:** Uygulamanın arayüzü için tercih edilmiştir.
- **Riverpod:** Uygulama içi durum yönetimi süreçleri için kullanılmıştır.
- **GoRouter:** Uygulama içi gelişmiş sayfa yönlendirmeleri için entegre edilmiştir.
- **Google Fonts:** Modern ve okunabilir tipografi desteği için tercih edilmiştir.

### **2.2. Backend (Sunucu Katmanı)**

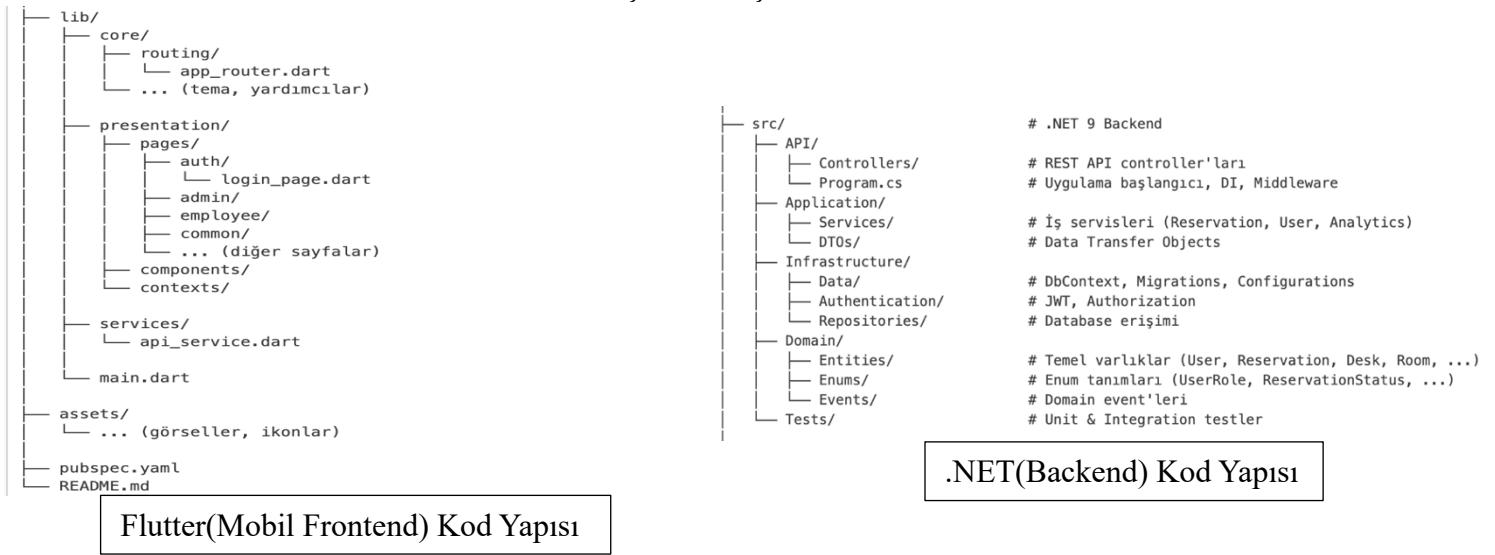
- **ASP.NET Core (C#):** RESTful API mimarisiyle, yüksek performanslı ve güvenli backend servislerinin geliştirilmesinde kullanılmıştır.
- **Entity Framework Core:** Veritabanı işlemleri için modern bir Object-Relational Mapping aracı olarak tercih edilmiştir.
- **JWT (JSON Web Token):** Kimlik doğrulama ve yetkilendirme süreçlerini yöneten temel güvenlik mekanizmasıdır.

### **2.3. Veri Yönetimi ve Operasyon**

- **PostgreSQL:** Üretim ortamında verilerin güvenli ve tutarlı şekilde saklanması için kullanılan ilişkisel veritabanı sistemidir.
- **Audit Logging:** Sistemdeki her kritik işlem denetim günlüğü tablolarında kayıt altına alınarak tam şeffaflık sağlanır.

### 3. SİSTEM MİMARİSİ

Proje, yüksek performans ve güvenlik standartlarını karşılamak amacıyla modern bir **istemci-sunucu** mimarisi üzerine inşa edilmiştir.



Flutter(Mobil Frontend) Kod Yapısı

.NET(Backend) Kod Yapısı

#### Örnek: Rezervasyon Oluşturma Akışı

1. Kullanıcı "Yeni Rezervasyon" butonuna tıklar  
↓
2. Rezervasyon formu ekranda açılır (UI render edilir)  
↓
3. Kullanıcı oda, tarih, saat seçer (etkileşim)  
↓
4. "Oluştur" butonuna tıklar  
↓
5. Form verisi işlenir ve server'a gönderilir (backend ile iletişim)  
↓
6. Başarı/hata mesajı gösterilir (UI güncellenir)

#### 3.1.2. Sunucu ile İletişimini RESTful API Mimarisi Üzerinden Gerçekleştirmesi

Flutter uygulaması, backend sunucu ile HTTP protokolü kullanarak RESTful API aracılığıyla iletişim kurar. Bu, standart web servisi modelidir.

#### HTTP İsteği Örneği:

Bir rezervasyon oluşturduğumuzda ne olur?

POST <http://localhost:5088/api/v1/reservations>

Headers:

- Content-Type: application/json
- Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Body (JSON):

```
{  
    "roomId": "room-123",  
    "userId": "user-456",  
    "startTime": "2026-01-15T14:00:00Z",  
    "endTime": "2026-01-15T15:00:00Z",  
    "purpose": "Ekip Toplantısı"  
}
```

#### HTTP Yanımı:

Status: 201 Created

Body (JSON):

```
{  
    "id": "reservation-789",  
    "roomId": "room-123",  
    "userId": "user-456",  
    "startTime": "2026-01-15T14:00:00Z",  
    "endTime": "2026-01-15T15:00:00Z",  
    "purpose": "Ekip Toplantısı",  
    "status": "confirmed",  
    "createdAt": "2026-01-13T10:30:00Z"  
}
```

### 3.1.3. Durum Yönetimi: Riverpod Kullanımı

Riverpod, uygulamanın tüm durumunu (state) merkezi olarak yönetir. Bunu, bir kontrol merkezi gibi düşünebilirsiniz.

#### State Nedir?

- Uygulamanın "belleği" - o an verilerin durumu
- Örn: Giriş yapan kullanıcı bilgisi, seçilen rezervasyon, form içeriği

#### Riverpod Provider'ları:

```
// 1. Token state'i - Kullanıcının giriş tokeni
final authTokenProvider = StateProvider<String?>((ref) => null);

// 2. Kullanıcı bilgisi state'i
final userProvider = StateProvider<User?>((ref) => null);

// 3. Rezervasyonlar listesi (API'den çekilir)
final reservationsProvider = FutureProvider<List<Reservation>>((ref) async {
    final token = ref.watch(authTokenProvider);
    return ApiService().getReservations(token: token);
});

// 4. Hesaplanan state - giriş yapıp yapmadığı
final isLoggedInProvider = Provider<bool>((ref) {
    return ref.watch(authTokenProvider) != null;
});
```

---

#### Riverpod'un Faydaları:

- **Reactive:** Bir state değişirse, onu kullanan tüm widget'lar otomatik güncellenir
- **Testable:** Kolayca mock ve test edilebilir
- **Type-safe:** Tür güvenliği sağlanır
- **Scope-controlled:** Her state'in yaşam süresi kontrol edilir

#### 4. Sayfa Navigasyonu: GoRouter Entegrasyonu

GoRouter, uygulamanın farklı sayfaları arasında geçiş ve yönlendirmeyi yönetir.

**Route (Rota) Nedir?** Bir route, uygulamada bir sayfaya giden bir "yol" veya "adresidir. Tıpkı web sitelerindeki URL'ler gibi:

/login	→ Giriş sayfası
/home	→ Ana sayfa
/reservations	→ Rezervasyonlar sayfası
/admin/users	→ Kullanıcı yönetimi (admin paneli)
/admin/backup	→ Yedekleme (admin paneli)

#### Navigasyon Kullanımı:

```
// Sayfaya git  
context.go('/home');  
  
// Parametreli navigasyon  
context.go('/reservations/123'); // ID ile rezervasyon detayı  
  
// Geri git  
context.pop();  
  
// Sayfa açıp sonucunu bekle  
final result = await context.push('/confirmDialog');
```

## 5. JWT Tabanlı Kimlik Doğrulama ile Güvenli Erişim

JWT (JSON Web Token), kullanıcının kimliğini güvenli şekilde doğrulamak için kullanılır.

### JWT Nasıl Çalışır?

1. Kullanıcı giriş yapar (email + şifre)  
↓
2. Backend, kimlik doğrular ve JWT token oluşturur  
↓
3. Flutter uygulaması tokeni alır ve cihazda saklar  
↓
4. Her API isteğinde token'ı header'a ekler  
↓
5. Backend, tokeni doğrular ve erişim izni verir/vermez

## Flutter'da Giriş Akışı:

```
Future<void> login(String email, String password, WidgetRef ref) async {
  try {
    // 1. API'ye giriş isteği gönder
    final response = await http.post(
      Uri.parse('http://localhost:5088/api/v1/auth/login'),
      headers: {'Content-Type': 'application/json'},
      body: jsonEncode({
        'email': email,
        'password': password,
      }),
    );

    if (response.statusCode == 200) {
      final data = jsonDecode(response.body);
      final token = data['token'];
      final user = User.fromJson(data['user']);

      // 2. Token'i güvenli şekilde sakla
      await FlutterSecureStorage().write(
        key: 'auth_token',
        value: token,
      );

      // 3. Global state'i güncelle
      ref.read(authTokenProvider.notifier).state = token;
      ref.read(userProvider.notifier).state = user;

      // 4. Ana sayfaya yönlendir
      context.go('/home');
    } else {
      throw Exception('Giriş başarısız');
    }
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Hata: $e')),
    );
  }
}
```

---

#### Token'ı Her İsteğe Ekleme:

```
class ApiService {
  Future<List<Reservation>> getReservations({String? token}) async {
    final response = await http.get(
      Uri.parse('http://localhost:5088/api/v1/reservations'),
      headers: {
        'Content-Type': 'application/json',
        'Authorization': 'Bearer $token', // ← Token burada eklenir
      },
    );

    if (response.statusCode == 200) {
      final List<dynamic> data = jsonDecode(response.body);
      return data.map((json) => Reservation.fromJson(json)).toList();
    } else if (response.statusCode == 401) {
      // Token süresi dolmuş veya geçersiz
      throw Exception('Oturum süresi doldu. Lütfen tekrar giriş yapın.');
    } else {
      throw Exception('Veriler getirilemedi');
    }
  }
}
```

#### Çıkış (Logout):

```
Future<void> logout(WidgetRef ref, BuildContext context) async {
  // 1. Token'i sil
  await FlutterSecureStorage().delete(key: 'auth_token');

  // 2. State'i sıfırla
  ref.read(authTokenProvider.notifier).state = null;
  ref.read(userProvider.notifier).state = null;

  // 3. Giriş sayfasına yönlendir
  context.go('/login');
}
```

## 3.2. Backend API (.NET Core)

### *3.2.1. Sistemin İş Kurallarının ve Veri İşleme Süreçlerinin Yürüttüğü Merkezi Katman*

Backend, uygulamanın tüm mantığını içerir. Gelen istekleri alır, kontrol eder, veritabanına kaydeder ve sonuç döndürür.

#### Örnek iş akışları:

- Rezervasyon yapılırken: Oda boş mu? Kullanıcının yetkisi var mı? Kota aşındı mı? Kontrolleri yapılır
- Rapor oluşturulurken: Tarih aralığındaki veriler işlenir, istatistikler hesaplanır
- Kullanıcı silinirken: Rezervasyonları nereye gidecek? Loglanacak mı? gibi soruları cevaplar

### *3.2.2. Kullanıcı Yönetimi, Rezervasyon, Oda/Kaynak, Raporlama, Loglar*

Backend şu modüllerle çalışır:

- **Kullanıcı Yönetimi:** Giriş/kayıt, profil güncelleme, rol atama
- **Rezervasyonlar:** Oda reservation sistemi, onay/iptal işlemleri
- **Oda/Kaynak:** Ofis odaları ve ekipmanları tanımlama
- **Raporlar:** Kullanım istatistikleri, analizler
- **Denetim Logları:** Kim ne yaptığı kaydeden

*Ana API Modülleri:*

1. **Authentication Module (/api/auth)**
  - o POST /api/auth/login - Kullanıcı giriş işlemi
  - o POST /api/auth/register - Yeni kullanıcı kaydı
  - o POST /api/auth/refresh-token - Token yenileme
  - o GET /api/auth/me - Oturum açmış kullanıcı bilgisi
  - o POST /api/auth/logout - Oturum kapatma
2. **Reservations Module (/api/reservations)**
  - o GET /api/reservations - Tüm rezervasyonları listele
  - o POST /api/reservations - Yeni rezervasyon oluştur
  - o GET /api/reservations/{id} - Belirli bir rezervasyonun detaylarını görüntüle
  - o PUT /api/reservations/{id} - Rezervasyonu güncelle
  - o DELETE /api/reservations/{id} - Rezervasyonu iptal et
  - o GET /api/reservations/availability/check - Masa/oda uygunluğu kontrolü
  - o POST /api/reservations/{id}/approve - Yönetici onayı (Manager/Admin)
  - o POST /api/reservations/{id}/reject - Yönetici reddi
3. **Locations Module (/api/locations)**
  - o GET /api/locations - Tüm ofis konumlarını listele
  - o POST /api/locations - Yeni konum oluştur (Admin)
  - o GET /api/locations/{id} - Konumun detaylarını görüntüle
  - o PUT /api/locations/{id} - Konumu güncelle (Admin)

- o DELETE /api/locations/{id} - Konumu sil (Admin)

#### 4. Desks Module (/api/desks)

- o GET /api/desks - Tüm masaları listele
- o GET /api/desks/{id} - Masa detaylarını görüntüle
- o POST /api/desks - Yeni masa oluştur (Admin)
- o PUT /api/desks/{id} - Masa bilgisini güncelle (Admin)
- o DELETE /api/desks/{id} - Masayı sil (Admin)

#### 5. Rooms Module (/api/rooms)

- o GET /api/rooms - Tüm toplantı odalarını listele
- o GET /api/rooms/{id} - Oda detaylarını görüntüle
- o POST /api/rooms - Yeni oda oluştur (Admin)
- o PUT /api/rooms/{id} - Oda bilgisini güncelle (Admin)
- o DELETE /api/rooms/{id} - Odayı sil (Admin)

#### 6. Users Module (/api/users)

- o GET /api/users - Kullanıcı listesi (Manager/Admin)
- o GET /api/users/{id} - Kullanıcı detayları
- o POST /api/users - Yeni kullanıcı oluştur (Admin)
- o PUT /api/users/{id} - Kullanıcı bilgisini güncelle
- o DELETE /api/users/{id} - Kullanıcıyı sil (Admin)
- o POST /api/users/{id}/change-role - Kullanıcı rolünü değiştir (Admin)

#### 7. Check-in/Check-out Module (/api/checkins)

- o POST /api/checkins - QR kodu ile check-in
- o POST /api/checkins/{id}/checkout - Check-out işlemi
- o GET /api/checkins - Check-in geçmişi

#### 8. Analytics Module (/api/analytics)

- o GET /api/analytics/dashboard - Ana dashboard istatistikleri
- o GET /api/analytics/reports - Detaylı raporlar

- o GET /api/analytics/usage - Kullanım analizi (günlük/haftalık/aylık)
- o GET /api/analytics/peak-hours - En yoğun saatler analizi

#### 9. Notifications Module (/api/notifications)

- o GET /api/notifications - Kullanıcının bildirimlerini listele
- o POST /api/notifications/{id}/read - Bildirimi oku olarak işaretle
- o DELETE /api/notifications/{id} - Bildirimi sil

#### 10. Audit Logs Module (/api/logs)

- o GET /api/logs - Sistem audit loglarını listele (Admin)
- o GET /api/logs?userId={userId} - Belirli kullanıcının işlemlerini görüntüle

#### *3.2.3. Entity Framework Core (EF Core) ile Veritabanı İletişimi*

EF Core, C# ve veritabanı arasında köprüdür. SQL yazmak yerine C# nesneleri ile çalışırsınız.

```
// Tüm kullanıcıları getir (SQL yazmanız gerekmekz)
var users = await _db.Users.ToListAsync();

// Belirli e-maili bul
var user = await _db.Users
    .FirstOrDefaultAsync(u => u.Email == "user@example.com");

// Bugünün rezervasyonları
var today = await _db.Reservations
    .Where(r => r.CreatedAt.Date == DateTime.Now.Date)
    .ToListAsync();
```

#### *3.2.4. RESTful API: Endpoint'ler ve Standartlaştırılmış Modüller*

REST, API tasarımı için standart yaklaşımıdır:

HTTP Metod	İşlem
GET	Veri oku
POST	Yeni veri oluştur
PUT	Veriyi güncelle
DELETE	Veriyi sil

Örnek:

GET /api/v1/users	→ Tüm kullanıcıları getir
POST /api/v1/users	→ Yeni kullanıcı ekle
PUT /api/v1/users/123	→ Kullanıcı 123'ü güncelle
DELETE /api/v1/users/123	→ Kullanıcı 123'ü sil

### 3.2.5. Güvenlik ve Yetkilendirme: JWT ve Rol Kontrol

#### JWT Kimlik Doğrulama:

- Kullanıcı giriş yaparsa sunucu JWT token oluşturur
- Flutter app, her istekte bu tokeni gönderir
- Backend, tokeni doğrular ve işleme izin verir

#### Rol Bazlı Erişim (RBAC):

- Admin: Tüm işlemleri yapabilir
- Manager: Takımını yönetebilir, raporları görebilir
- Employee: Sadece kendi rezervasyonlarını yapabilir

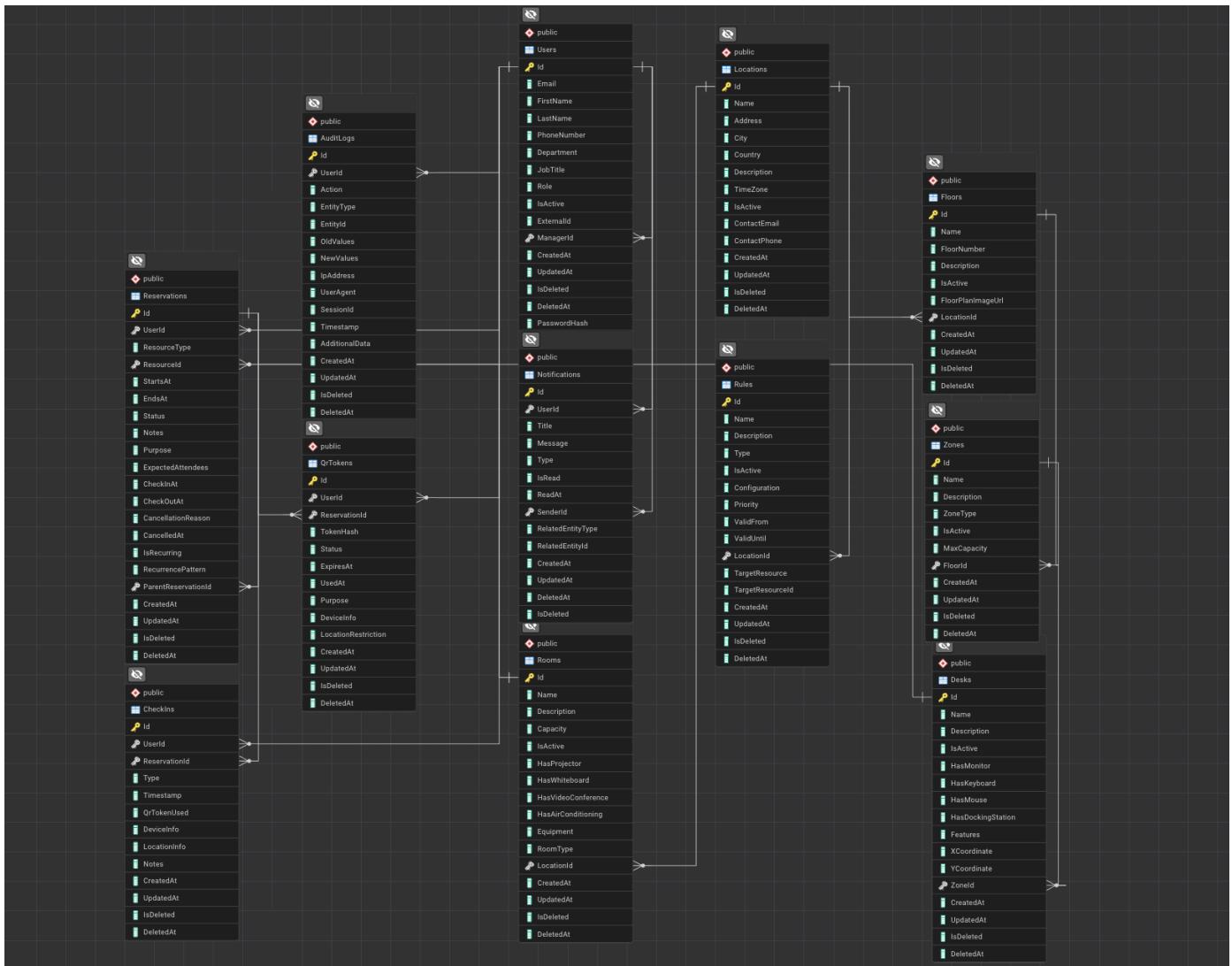
```
[Authorize(Roles = "Admin")]
public async Task<IActionResult> DeleteUser(string id)
{
    // Sadece Admin bu işlemi yapabilir
}

[Authorize(Roles = "Admin,Manager")]
public async Task<IActionResult> GetReports()
{
    // Admin ve Manager görebilir
}
```

---

### 3.3. PostgreSQL Veritabanı

- Kullanıcı bilgileri, roller, rezervasyon kayıtları, kaynak detayları ve sistem kuralları gibi tüm verilerin tutarlı ve kalıcı bir şekilde saklandığı ilişkisel veritabanı katmanıdır.



## 4. UYGULAMA ÖZELLİKLERİ

Ofis Yönetim Sistemi, hem çalışanların hem de yöneticilerin ihtiyaçlarını karşılamak üzere tasarlanmış, geniş kapsamlı ve entegre bir özellik seti sunmaktadır. Sistemde öne çıkan temel fonksiyonel modüller şunlardır:

### 4.1. Kullanıcı Erişimi ve Rol Yönetimi

- **Hiyerarşik Yetkilendirme:** Admin, Yönetici ve Çalışan rolleri için tanımlanmış farklı erişim seviyeleri ve yetki sınırları mevcuttur.

### 4.2. Gelişmiş Oda ve Kaynak Rezervasyonu

- **Anlık Rezervasyon:** Toplantı odaları, çalışma masaları ve diğer ofis ekipmanları sistem üzerinden anlık olarak rezerve edilebilir.

### 4.3. QR Kod Tabanlı Giriş ve Doğrulama

- **Dinamik QR Kodlar:** Her kullanıcıya özel, belirli süreli ve otomatik olarak yenilenen QR kodlar aracılığıyla ofis kaynaklarına hızlı erişim sağlanır.

### 4.4. Kurallar, Log Yönetimi ve Bildirimler

- **İş Kuralları Yönetimi:** Ofis içi kullanım kuralları sistem üzerinden görüntülenebilir, yeni kural eklenenebilir veya mevcut kurallar güncellenebilir.
- **Denetim Günlükleri (Logging):** Tüm kullanıcı işlemleri ve sistem aktiviteleri, güvenlik ve denetim amacıyla şeffaf bir şekilde loglanır.
- **Hatırlatıcılar ve Uyarılar:** Kullanıcılara rezervasyon saatleri için hatırlatıcılar ve kritik sistem güncellemeleri için anlık bildirimler iletılır.

### 4.5. Analistik Raporlama ve Çoklu Platform Desteği

- **Veri Analizi:** Rezervasyon oranları, oda doluluk yüzdeleri ve kullanıcı bazlı kullanım verileri; grafik ve tablolarla desteklenen analiz ekranlarında sunulur.
- **Esnek Erişim:** Uygulama, hibrit çalışma modellerini desteklemek amacıyla mobil (Android/iOS) ve web platformları üzerinden tam uyumlu erişim imkanı sağlar.

## 5. ARAYÜZ VE TASARIM

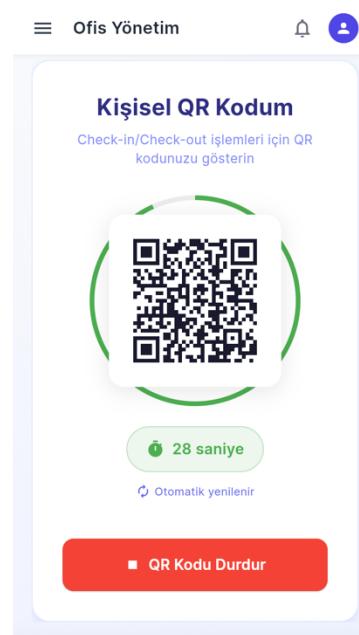
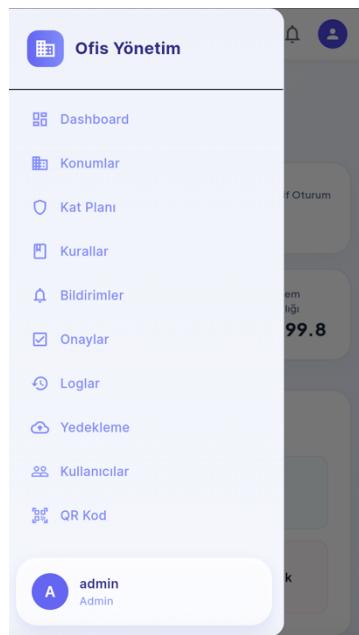
**Ofis Yönetim Sistemi**, estetik görünümün yanı sıra kullanıcı dostu, sade ve akıcı bir deneyim sunacak şekilde tasarlanmıştır.

### 5.1. Görsel Kimlik ve Estetik

- Tasarım Detayları:** Uygulamanın görsel dili, modern tasarım trendlerini kurumsal bir yaklaşımla birleştirir. Ana renk paletinde morun modern tonları tercih edilerek profesyonel bir atmosfer oluşturulmuştur.
- Görünüm:** Arayüzde kullanılan glassmorphism efektli, yumuşak köşeli kart yapıları ferah ve çağdaş bir görünüm sağlar. Modern yazı tipleri ve açıklayıcı ikonlar sayesinde okunabilirlik ve görsel hiyerarşi güçlendirilmiştir.

### 5.2. Kullanıcı Deneyimi (UX)

- Dinamik Tasarım:** Sistem, sezgisel ve akıcı bir kullanım deneyimi sunar. Sayfa geçişleri ve buton etkileşimlerinde kullanılan yumuşak animasyonlar, arayüzü daha dinamik hale getirir.
- Responsive Tasarım:** Duyarlı tasarım yapısı sayesinde uygulama farklı ekran boyutlarında sorunsuz çalışır. Sade navigation yapısı ve rol bazlı arayüz adaptasyonu ile kullanıcılar yalnızca yetkili oldukları içeriklere hızlı ve kolay şekilde erişebilir.



## **6. YAPILAN GELİŞTİRMELER VE DEĞİŞİKLİKLER**

Proje geliştirme sürecinde, hem işlevsel derinliği artırmak hem de kullanıcı deneyimini mükemmelleştirmek adına kritik iyileştirmeler yapılmıştır:

### **6.1. Backend Entegrasyonu ve Güvenlik**

- Sayfaların veritabanı bağlantıları tamamlanıp veri akışı sağlanırken, tüm temel veri işlemleri (CRUD) API üzerinden standartlaştırılmıştır.
- JWT tabanlı kimlik doğrulama ile Rol Bazlı Erişim Kontrolü (RBAC) mekanizması sisteme tam entegre edilmiştir.

### **6.2. Kullanıcı Arayüzü (UI) İyileştirmeleri**

- Giriş (Login) sayfası, modern mor tema ve glassmorphism detaylarıyla estetik olarak yenilenmiştir.
- QR kod ekranı, otomatik ve süreli yenilenen dinamik bir yapıya kavuşturulmuştur.
- Tüm sayfaların ve bileşenlerin birbiriyle uyumlu olması ve modern bir yapı oluşturması önceliklendirilmiştir.

### **6.3. Performans ve Kod Kalitesi**

- Riverpod ile merkezi ve sürdürülebilir bir durum yönetimi mimarisi kurulmuştur.
- Hata yönetimi süreçleri iyileştirilerek kullanıcıya yol gösteren bilgilendirici uyarı sistemleri eklenmiştir.

## **7. TESTLER VE SONUÇLAR**

Uygulamanın kararlılığını ve doğruluğunu teyit etmek amacıyla manuel ve otomatik test yöntemleri uygulanmıştır:

### **7.1. Manuel Fonksiyonel Testler**

- Tüm kullanıcı rolleri (Admin, Yönetici, Çalışan) ile sisteme giriş ve yetki sınırları başarıyla test edilmiştir.
- Oda ve kaynak rezervasyonu işlemleri, çakışma kontrol mekanizması ve takvim entegrasyonu gerçek senaryolarla doğrulanmıştır.

### **7.2. Kullanıcı Deneyimi ve Uyumluluk**

- Uygulamanın farklı cihazlarda (masaüstü, tablet, mobil) duyarlı (responsive) çalışma performansı denetlenmiştir.
- Sayfa geçiş hızları, form alanlarının doğruluğu ve renk kontrast oranlarının okunabilirliği onaylanmıştır.

### **7.3. Güvenlik ve Performans**

- API uç noktalarında yetkilendirme ve hata yönetimi süreçleri kontrol edilerek veri güvenliği doğrulanmıştır.
- Vite ile optimize edilen derleme süreleri ve sayfa yükleme hızlarının hedeflenen standartlarda olduğu gözlemlenmiştir.

## 8. PROJE GELİŞİM ANALİZİ

Teknik şartnamedeki hedeflerin, final aşamasında ne ölçüde karşılandığı ve projenin gelişim süreci özetlenecek olursa:

### **8.1. Başlangıç Vizyonu ve Teknik Beklentiler**

Proje başlangıcında, esnek çalışma modelini destekleyen ve web/API katmanlarının doğal bir uzantısı olan "native-like" bir Flutter uygulaması hedeflenmiştir:

- **Mimari Hedef:** Flutter 3.x ve Dart 3 kullanarak iOS ve Android platformlarında yüksek performanslı, Riverpod tabanlı bir state yönetimi kurmak.
- **Kritik Akışlar:** Gün/hafta görünümünde masa/oda müsaitliği izleme, dinamik QR kod üretimi ve check-in/out süreçlerinin mobil üzerinden tamamlanması.
- **Güvenlik:** OIDC tabanlı kimlik doğrulama, AES şifrelemeli yerel depolama (`flutter_secure_storage`) ve rol temelli yetkilendirme (RBAC).

### **8.2. Uygulama Süreci ve Nihai Çıktılar**

Geliştirme süreci sonunda, şartnamede yer alan teknik gereksinimlerin ötesine geçilerek kullanıcı deneyimi odaklı şu sonuçlar elde edilmiştir:

- **Tam Entegrasyon:** Başlangıçta hedeflenen Riverpod mimarisi ve GoRouter navigasyonu başarıyla uygulanmış; API katmanıyla asenkron ve güvenli bir veri yolu kurulmuştur.
- **Fonksiyonel Başarı:** Masa ve oda rezervasyonları, dinamik QR kod üretimi ve yönetici onay mekanizmaları, tasarım şartnamesindeki akış şemalarına sadık kalınarak hayata geçirilmiştir.
- **Görsel Modernizasyon:** Başlangıç planındaki "fonksiyonel" arayüz hedefi, uygulama aşamasında **Glassmorphism**, kurumsal mor degrade temalar ve **Google Fonts** tipografisi ile birleşerek premium bir kullanıcı deneyimine (UX) dönüştürülmüştür.

## 9. SONUÇ VE DEĞERLENDİRME

**Ofis Yönetim Sistemi mobil uygulaması**, modern yazılım teknolojileri ve kullanıcı odaklı tasarım prensipleri rehberliğinde başarıyla hayatı geçirilmiştir. Proje, başlangıçta hedeflenen teknik kriterleri karşılamadan ötesine geçerek kurumsal ihtiyaçlara yönelik bütüncül bir dijital deneyim sunmaktadır.

### 9.1. Proje Kazanımları ve Çıktılar

Uygulama geliştirme süreci sonunda elde edilen temel başarılar şu şekilde özetlenebilir:

- Tam Fonksiyonellik ve Dijitalleşme:** Ofis içi kaynakların yönetiminden rezervasyon süreçlerine, QR kod tabanlı güvenli giriş sisteminden gerçek zamanlı bildirimlere kadar tüm operasyonel akışlar dijitalleştirilmiştir.
- Güvenli ve Esnek Altyapı:** OIDC tabanlı kimlik doğrulama, rol bazlı yetkilendirme ve güvenli veri saklama katmanları ile farklı kullanıcı rolleri için yüksek güvenlik standartlarında, esnek bir erişim altyapısı oluşturulmuştur.
- Üst Düzey Performans ve Kalite:** Kullanıcı deneyimi, performans kriterleri ve görsel kalite standartları ön planda tutularak; soğuk açılış süresi ve çökme oranları gibi kalite metriklerinde hedeflenen seviyelere ulaşılmıştır.

### 9.2. Gelecek Vizyonu ve Ölçeklenebilirlik

Yapılan kapsamlı testler sonucunda uygulamanın kararlı, hızlı ve hatasız bir şekilde çalıştığı doğrulanmıştır. Kullanıcı geri bildirimleri doğrultusunda gerçekleştirilen sürekli iyileştirmelerle sistem, daha verimli ve kullanıcı dostu bir yapıya kavuşturulmuştur.

Sonuç olarak bu proje; ofis yönetimi, kaynak planlaması ve kurumsal dijital dönüşüm süreçlerinde kurumlara önemli bir katma değer sunmaktadır. Mevcut modüler yapısı sayesinde toplantı odası yönetimi, ziyaretçi akışları ve AI destekli koordinasyon önerileri gibi yeni özelliklerle zenginleştirilmeye ve ölçeklenmeye tam uyumludur.