**ANKARA ÜNİVERSİTESİ BİLGİSAYAR MÜHENDİSLİĞİ**
**İOS İLE MOBİL UYGULAMA GELİŞTİRME DERSİ**

**PROJE RAPORU**


# KUAFÖR RANDEVU YÖNETİM SİSTEMİ

**Ad: Samet**
**Soyad: YILDIRIMÇAKAR**
**Öğrenci Numarası: 21290144**
**Github: https://github.com/sametyildirim314**
**Video: https://www.youtube.com/watch?v=dY-kmmytbO4**

## ÖZET

Bu doküman, müşteri–işletme–admin rollerini aynı platformda birleştiren bir “Kuaför Randevu Sistemi”nin uçtan uca tasarımını ve gerçekleştirimini akademik bir çerçevede incelemektedir. Sistem; Flutter tabanlı istemci uygulaması (frontend), Node.js/Express tabanlı REST servis katmanı (backend) ve MySQL tabanlı ilişkisel veritabanı (data layer) bileşenlerinden oluşmaktadır. Çalışmada; mimari katmanların ayrıştırılması, veri modelinin normalizasyon ilkeleri doğrultusunda kurgulanması, kimlik doğrulama için JWT kullanımı, parola güvenliği için bcrypt hashleme, SQL injection riskini azaltmak için hazırlanmış sorgular (prepared statements) ve API sözleşmesinin OpenAPI ile belgelendirilmesi ayrıntılı biçimde ele alınmıştır. Ayrıca, iş kurallarının kullanıcı arayüzüne doğru biçimde yansıtılması bağlamında “beklemede olup tarihi geçmiş randevuların ‘Süre Doldu’ olarak gösterimi” ile “çalışanların dolu ve boş saatlerinin (kırmızı/yeşil) hem müşteri hem işletme ekranlarında görünür kılınması” gereksinimleri örneklendirilmiştir. Doküman, sistemin sınırlılıklarını ve üretim ortamı için önerilen geliştirmeleri (RBAC, refresh token, indeksleme, test otomasyonu, gözlemlenebilirlik) de kapsamaktadır.



**Anahtar Kelimeler: Flutter, Node.js, Express.js, MySQL, REST API, JWT, bcrypt, OpenAPI, Randevu Yönetimi, Çok Rollü Sistem, Glassmorphism UI**

**İÇİNDEKİLER**


1. **Giriş**
2. **Problem Tanımı ve Hedefler**
3. **Sistem Gereksinimleri**
4. **Teknoloji Yığını ve Gerekçeli Seçimler**
5. **Uçtan Uca Mimari Tasarım**
6. **Frontend (Flutter) Mimarisi ve Modüler Yapı**
7. **Backend (Node.js/Express) Mimarisi, Katmanlar ve Akışlar**
8. **API Tasarımı, Sözleşme Yönetimi ve OpenAPI**
9. **Veritabanı Tasarımı (MySQL): Şema, İlişkiler ve Bütünlük**
10. **İş Kuralları ve Kritik Özelliklerin Gerçekleştirilmesi**
11. **Güvenlik Yaklaşımı ve Risk Analizi**
12. **Performans, Ölçeklenebilirlik ve İyileştirme Önerileri**
13. **Test Stratejisi ve Doğrulama Yaklaşımı**
14. **Kurulum, Çalıştırma ve Operasyonel Notlar**
15. **Sonuç**

## 1. GİRİŞ

Hizmet sektöründe randevu süreçleri; müşteri memnuniyetini doğrudan etkileyen, planlama ve kapasite yönetimi açısından kritik bir işlevdir. Geleneksel yöntemler (telefonla randevu, defter takibi, mesajlaşma uygulamaları) veri bütünlüğü, çakışma önleme, geçmiş kayıtların izlenebilirliği ve personel iş yükünün dengelenmesi gibi konularda önemli sınırlılı klar taşımaktadır. Bu proje, söz konusu operasyonel sorunlara yanıt vermek amacıyla, randevu süreçlerini merkezi bir bilgi sistemi üzerinden yönetebilen, rol bazlı erişimi destekleyen ve kullanıcı deneyimini modern arayüz tasarım ilkeleriyle güçlendiren bir uygulama olarak tasarlanmıştır.

## 2. PROBLEM TANIMI VE HEDEFLER

### 2.1 Problem Tanımı

Temel problem; işletme tarafında çalışan ve hizmet yönetimiyle birlikte randevu planlamasının sağlıklı biçimde yürütülememesi, müşteri tarafında ise doğru çalışan/zaman dilimi seçiminin “gerçek doluluk bilgisi” ile desteklenmemesi ve randevu durumlarının anlaşılır şekilde gösterilememesidir.2.2 Proje HedefleriAşağıdaki hedefler sistemin kapsamını belirlemiştir:

- Çok rollü yapı: Müşteri, işletme ve admin kullanıcı tiplerinin ayrıştırılması.
- Randevu yaşam döngüsü: Beklemede (pending), onaylandı (confirmed), tamamlandı (completed), iptal (cancelled).
- Çakışma kontrolü: Aynı işletme–çalışan–tarih–saat için çifte randevunun engellenmesi.
- Görsel geri bildirim: Çalışan saatlerinin dolu/boş durumunun renklerle (kırmızı/yeşil) gösterimi.
- Durum türetimi: Beklemede olup tarihi geçmiş randevuların “Süre Doldu” şeklinde sunulması.
- Kurumsal arayüz: Premium ve modern UI (glassmorphism, gradient, blur) ile yüksek kullanılabilirlik.

## 3. SİSTEM GEREKSİNİMLERİ

### 3.1 Fonksiyonel Gereksinimler

- Kullanıcı kayıt ve giriş işlemleri (müşteri/işletme)
- Admin girişi
- İşletme bazlı çalışan ve hizmet yönetimi (ekleme, güncelleme, silme)
- Randevu oluşturma, güncelleme, listeleme
- Müşteri tarafında randevu oluştururken seçilen çalışan için dolu saatlerin görünmesi
- İşletme tarafında çalışanların gün içi dolu saatlerinin görünmesi
- Beklemede olup tarihi geçmiş randevular için “Süre Doldu” gösterimi

### 3.2 Fonksiyonel Olmayan Gereksinimler

- Güvenlik: Parola hashleme, JWT, SQL injection koruması
- Kullanılabilirlik: Responsive tasarım, açık geri bildirim (SnackBar/Badge)
- Sürdürülebilirlik: Modüler kod organizasyonu, dokümantasyon (OpenAPI)
- Performans: Bağlantı havuzu, uygun indeksleme önerileri

- Taşınabilirlik: Flutter ile web/mobil çalıştırma

## 4. TEKNOLOJİ YIĞINI VE GEREKÇELİ SEÇİMLER

### 4.1 Frontend: Flutter

Flutter; tek kod tabanıyla web ve mobil hedefleri desteklemesi,
widget tabanlı UI yaklaşımı ve performans avantajları nedeniyle seçilmiştir.
Projede ayrıca HTTP iletişimi, local saklama (SharedPreferences) ve tarih/saat işlemleri (intl) gibi paketler kullanılmıştır.4.2 Backend: Node.js/ExpressNode.js, asenkron I/O modeliyle API servisleri için uygun; Express ise minimal ve modüler bir routing/middleware yapısı sunan bir web framework'tür. Projede JWT (jsonwebtoken), parola hashleme (bcryptjs) ve MySQL sürücüsü (mysql2/promise) kullanılmıştır.4.3 Veritabanı: MySQLMySQL; ilişkisel veri modelinin ihtiyaç duyulduğu (müşteri–işletme–çalışan–hizmet–randevu) senaryolarda olgun, performanslı ve yaygın olarak kullanılan bir çözümdür. Foreign key ve bütünlük kısıtlarıyla veri tutarlılığı desteklenir.

## 5. UÇTAN UCA MİMARİ TASARIM

Sistem üç ana katmanda ele alınabilir:

- Sunum Katmanı (Flutter UI): Ekranlar ve kullanıcı etkileşimleri
- Uygulama/Servis Katmanı (Express API): İş kuralları, doğrulamalar, yetkilendirme
- Veri Katmanı (MySQL): Kalıcı veri saklama ve ilişkiler

Bu ayrım; bakım kolaylığı, bileşenlerin bağımsız geliştirilebilmesi ve ileride ölçekleme/yeniden kullanım açısından tercih edilmiştir.

## 6. FRONTEND (FLUTTER) MİMARİSİ VE MODÜLER YAPI

### 6.1 Proje Yapısı (Özet)

- lib/main.dart: Uygulama başlangıcı, tema, AuthWrapper yönlendirme
- lib/config/api_config.dart: Base URL ve endpoint sabitleri
- lib/models/: Model sınıfları (Appointment, Customer, Business, Employee, Service, Admin)
- lib/services/: API ve iş mantığı servisleri (ApiService, AuthService, AppointmentService, BusinessService, AdminService)
- lib/screens/: Ekranlar (welcome/login/register/home/create appointment)
- lib/widgets/responsive_wrapper.dart: Responsive yardımcıları

### 6.2 Oturum Yönetimi ve Yönlendirme

AuthWrapper,
SharedPreferences'ta bulunan token ve kullanıcı tipini okuyarak (customer/business/admin) ilgili home ekranına yönlendirir. Bu yaklaşım,
uygulama kapatılıp açıldığında kullanıcı deneyimini kesintisiz kılar.
Mevcut uygulamada istemci tarafında ek bir "oturum süresi" mantığı (ör. 5 dakika) bulunmaktadır; bu, güvenlik açısından faydalı olsa da
üretim senaryosunda refresh token ile daha dengeli bir model önerilir.6.3
Servis Katmanı ve API İletişimiApiService; singleton olarak tasarlanmıştır. Bu servis:

- HTTP GET/POST/PUT/DELETE isteklerini standartlaştırır

- Token varsa Authorization header'ını otomatik ekler
- Response sözleşmesini normalize eder (success/message/data)

Bu tasarım; ekranlarda tekrar eden HTTP kodlarını azaltır ve hata yönetimini tutarlı kılar.6.4 UI/UX Tasarım DiliUygulamada, premium algıyı yükselten modern bir tasarım dili uygulanmıştır:

- Gradient arka planlar
- Glassmorphism paneller (blur + yarı saydam katmanlar)
- Modern CTA butonları ve durum etiketleri
- Responsive yapı: büyük ekranlarda içerik genişliği kısıtlanır (ResponsiveWrapper

## 7. BACKEND (NODE.JS/EXPRESS) MİMARİSİ, KATMANLAR VE AKIŞLAR

### 7.1 Proje Yapısı (Özet)

- server.js: Express app, CORS, JSON parsing, routes mount, health check, error handling
- config/database.js: MySQL connection pool
- middleware/auth.js: JWT doğrulama + token üretimi
- routes/: authRoutes, appointmentRoutes, businessRoutes, dataRoutes
- controllers/: authController, appointmentController, businessController, dataController
- openapi.yaml: API sözleşmesi
- seed_data.js: örnek veri üretimi

### 7.2 Middleware Tasarımı

authenticateToken middleware'i,
Authorization header'ındaki Bearer token'ı doğrular ve decoded payload'u req.user içine taşır.
Böylece korunan endpoint'lerde kullanıcı kimliği taşınır.7.3 Controller KatmanıController'lar;
request doğrulaması, iş kuralı kontrolleri (ör. çakışma kontrolü),
SQL sorguları ve response formatının üretiminden sorumludur. Routes katmanı yalnızca endpoint haritalamasını yapar.7.4 Hata Yönetimiserver.js üzerinde 404
handler ve global error handler bulunur. Bu yaklaşım; istemciye tutarlı hata mesajları dönmeyi ve backend loglamasını merkezi yapmayı kolaylaştırır.

## 8. API TASARIMI, SÖZLEŞME YÖNETİMİ VE OPENAPI

### 8.1 REST Yaklaşımı

API; kaynak odaklı ve HTTP metotlarına uygun olacak şekilde tasarlanmıştır.
Endpoint'ler fonksiyonel gruplara ayrılmıştır:

- /api/auth/*
- /api/appointments/* (korumalı)
- /api/business/*
- /api/* (genel veri çekme)
- /api/health

### 8.2 Sözleşme (Contract) ve Yanıt Formatı

Genel yanıt biçimi şu şekildedir:

- Başarılı: success = true, data alanı dolu
- Hatalı: success = false, message alanı dolu

Bu standardizasyon, istemci tarafında hata gösterimini basitleştirir.8.3 OpenAPI (openapi.yaml)OpenAPI dokümantasyonu:

- Endpoint'lerin şemasını, request/response modellerini, güvenlik şemasını tanımlar
- API test süreçlerini hızlandırır (Scalar vb.)
- Bakım ve ekip içi iletişimde sözleşme uyumluluğunu güçlendirir

## 9. VERİTABANI TASARIMI (MYSQL): ŞEMA, İLİŞKİLER VE BÜTÜNLÜK

### 9.1 Temel Tablolar

- admins: yönetici kullanıcılar
- businesses: işletmeler
- customers: müşteriler
- employees: işletmeye bağlı çalışanlar
- services: işletmeye bağlı hizmetler
- appointments: randevular (merkez tablo)
- employee_schedules: çalışan haftalık çalışma aralıkları

### 9.2 İlişkiler

appointments tablosu; customer_id, business_id, employee_id,
service_id alanlarıyla diğer tablolara bağlanır. Bu ilişkiler, randevunun kim tarafından, hangi işletmede, hangi çalışan ve hangi hizmet için oluşturulduğunu temsil eder.9.3
Bütünlük ve TutarlılıkForeign key kısıtlarıyla referans bütünlüğü sağlanır.
Ayrıca uygulama katmanında çakışma kontrolü ile aynı zaman diliminde double-booking riski azaltılır. Üretim senaryosunda,
veritabanı düzeyinde de uygun indeksleme ve gerekirse unique constraint yaklaşımları önerilir.

## 10. İŞ KURALLARI VE KRİTİK ÖZELLİKLERİN GERÇEKLEŞTİRİLMESİ

### 10.1 "Süre Doldu" (Beklemede + Tarihi Geçmiş)

"Süre Doldu", veritabanına yeni bir status eklemek yerine türetilmiş bir durum olarak ele alınmıştır:

- Eğer status = pending ve randevu tarihi bugünden önce ise, UI'da "Süre Doldu" gösterilir.

Bu yaklaşım şemayı sade tutar; ancak tüm istemcilerde tutarlı uygulanması gerekir.
Alternatif yaklaşım olarak backend, computed alan döndürerek bu türetimi merkezileştirebilir.10.2
Çalışan Dolu/Boş Saatlerinin GösterimiSistem, "dolu saat" kavramını seçili tarihte ilgili çalışana ait confirmed/completed randevu saatleri üzerinden türetir:

- Dolu saatler: kırmızı ve "DOLU" etiketi
- Boş saatler: yeşil/neutral

Bu özellik hem müşteri randevu oluşturma ekranında hem işletme
dashboard'unda aynı mantıkla görünür kılınmıştır. Böylece müşteri yanlış saat seçmez, işletme ise personel kapasitesini anlık görebilir.Proje notu (rapora dahil edilmesi istenen ifade):"2. gösterimde eksik olan çalışanların dolu saatlerinin gösterilmemesi sorunu giderildi.
Artık müşteri ve işletme sayfalarında dolu ve boş saatler görünecek."10.3

Çakışma KontrolüRandevu oluşturulurken backend, aynı işletme–çalışan–tarih–
saat için (cancelled hariç) mevcut randevu olup olmadığını kontrol eder. Bu kontrol;
kullanıcı hatalarını azaltır ve takvim tutarlılığı sağlar. İleri seviye
senaryolarda transaction/locking ve veritabanı kısıtlarıyla daha güçlü garanti sağlanabilir.

## 11. GÜVENLİK YAKLAŞIMI VE RİSK ANALİZİ

### 11.1 Mevcut Önlemler

- Parola güvenliği: bcrypt ile hash karşılaştırması
- Kimlik doğrulama: JWT Bearer token
- SQL injection önleme: prepared statements
- Hata yönetimi: global error handler ve kontrollü mesajlar

### 11.2 Riskler ve Eksikler (Üretim Perspektifi)

- Yetkilendirme: Token doğrulama var; ancak rol bazlı erişim (RBAC) ve kaynak sahipliği kontrolleri daha sistematik hale getirilmelidir.
- CORS: Development ortamında tüm origin'lere izin verme yaklaşımı üretimde kısıtlanmalıdır.
- Token yaşam döngüsü: Refresh token, revoke mekanizması ve güvenli saklama önerilir.
- Rate limiting: brute-force saldırılarına karşı önemlidir.

## 12. PERFORMANS, ÖLÇEKLENEBİLİRLİK VE İYİLEŞTİRME ÖNERİLERİ

- MySQL connection pool kullanımı temel performans kazanımı sağlar.
- Randevu listeleme ve saat doluluk sorguları için indeksleme önerilir (ör. employee_id + appointment_date + appointment_time).
- Büyük veri hacminde pagination şarttır.
- Sık değişmeyen referans veriler (services vb.) cache edilebilir.
- Gözlemlenebilirlik için structured logging, request-id, metrik toplama önerilir.

## 13. TEST STRATEJİSİ VE DOĞRULAMA YAKLAŞIMI

- Backend: unit/integration test (örn. supertest), test DB veya container yaklaşımı
- Frontend: widget test, golden test, form validasyon testleri
- E2E: temel kullanıcı akışları (login → listele → oluştur → güncelle) otomasyonlaştırılabilir
- Veri senaryoları: seed_data.js ile doğrulama verileri üretimi

## 14. KURULUM, ÇALIŞTIRMA VE OPERASYONEL NOTLAR

### 14.1 Backend Çalıştırma

- npm install
- npm run dev (nodemon) veya npm start
- Health check: /api/health

### 14.2 Frontend Çalıştırma

- flutter pub get
- flutter run -d chrome (web) veya mobil cihaz/emülatör

### 14.3 API Testi

- openapi.yaml üzerinden Scalar ile dokümantasyon ve test yapılabilir.

## 15. SONUÇ

Bu doküman, Flutter–Node.js/Express–MySQL tabanlı çok rollü bir randevu yönetim sistemini akademik bir yaklaşımla uçtan uca ele almıştır. Sistem; modern UI/UX prensipleri, modüler kod organizasyonu, RESTful API sözleşmesi, güvenlik temelleri (JWT, bcrypt, prepared statements) ve işlevsel olarak kritik iki özellik (Süre Doldu ve çalışan dolu/boş saatleri) ile operasyonel değer üretmektedir. Üretim seviyesinde RBAC, refresh token, rate limiting, kapsamlı test otomasyonu ve performans/izlenebilirlik iyileştirmeleriyle sistemin olgunluk seviyesi artırılabilir.