PROJE SONU RAPORU:

THY Biletleme ve Rezervasyon Sistemi

Grup Üyeleri:

Kaan Baykal 221101028
Beril Aydın 231101002
Anıl Özişler 211101081

Proje GitHub Linki:
https://github.com/Berilay6/THY_Ticketing_App/tree/main

1. Gerçek Dünya Probleminin Tanımı

Havacılık sektöründe binlerce uçuşun, milyonlarca koltuğun ve anlık değişen operasyonel süreçlerin manuel veya izole sistemlerle yönetilmesi imkansızdır. Yolcuların saniyeler içinde biletleme yapma beklentisi ve havayolu şirketlerinin uçak/bakım envanterini hatasız yönetme zorunluluğu, yüksek performanslı ve bütünleşik bir veritabanı çözümünü zorunlu kılmaktadır.

Bu proje, gerçek dünyadaki bu karmaşık yapıyı simüle ederek şu problemleri çözmeyi hedefler:

- **Çifte Rezervasyon (Double Booking) Riski:** Aynı koltuğun aynı anda iki farklı yolcuya satılmasının engellenmesi.
- **Operasyonel Süreklilik:** Uçakların bakıma alınması veya sefer iptalleri durumunda bilet iade gibi süreçlerinin yönetimi.
- **Veri Bütünlüğü:** Ödeme alındığı anda biletin oluşması, aksi durumda işlemin tamamen geri alınması (Atomicity).

Projenin akademik bağlamda seçilme nedeni ise; yukarıda belirtilen bu gerçek hayat problemlerinin, Veritabanı Yönetim Sistemleri dersinin temel kazanımları olan **Eşzamanlılık Kontrolü (Concurrency)**, **Transaction Yönetimi** ve **İlişkisel Normalizasyon** prensiplerini uygulamak için uygun senaryoyu sunmasıdır.

2. Gereksinim Analizi

Projenin geliştirilme sürecinde hedeflenen sistem gereksinimleri; kullanıcı etkileşimleri, operasyonel yönetim ve sistemsel kısıtlar olmak üzere üç ana başlıkta analiz edilmiştir.

2.1. Fonksiyonel Gereksinimler

Sistem, "User" ve "Admin" olmak üzere iki farklı aktör için aşağıdaki fonksiyonları sağlamalıdır:

User Modülü:

- **Dinamik Uçuş Arama:** Kullanıcılar; kalkış noktası, varış noktası ve tarih kriterlerine göre veritabanındaki aktif uçuşları filtreleyebilmelidir.
- **Rezervasyon ve Biletleme (E-Ticket):** Kullanıcı, seçtiği uçuş için ödeme işlemini tamamlamalıdır. İşlem sonucunda sistemde ticket oluşturulmalıdır.

- **Koltuk Seçimi:** Sistem, biletleme sürecinde uygun koltukları kullanıcıya sunmalıdır.
- **Ödeme Sistemi:** Kullanıcılar kredi kartı bilgileri ile ödeme yapabilmelidir. Bir ödeme işlemi, birden fazla biletin ödemesini tek seferde kapsayabilmelidir.
- **Geçmiş Uçuşlar:** Kullanıcılar, kendi hesapları üzerinden geçmişte yaptıkları seyahatleri ve gelecek rezervasyonlarını "My flights" ekranında listeleyebilmelidir.
- **Profil Yönetimi:** Kullanıcılar kişisel bilgilerini güncelleyebilmeli ve sık kullandıkları kredi kartlarını sistemde saklayabilmelidir.

Admin Modülü:

- **Envanter Yönetimi:** Yönetici; sisteme yeni Havalimanı, Uçak ve Sefer (Flight) ekleyebilmelidir.
- **Otomasyon:** Sistem, bir uçak eklendiğinde uçağın fiziksel koltuklarını ve uçuş eklendiğinde uçuşa ait satılacak koltukları otomatik olarak veri tabanına eklemelidir.
- **Uçuş Durum Yönetimi:** Operasyonel gereklilikler durumunda uçuşlar "Cancelled" statüsüne çekilebilmelidir.
- **İade Süreci:** İptal edilen uçuşlarda, sistem o uçuşa ait satılmış tüm biletleri tespit etmeli ve bilet statülerini otomatik olarak "İade Edildi" konumuna getirerek para iadesini gerçekleştirmelidir.
- **Filo Yönetimi:** Arızalanan uçaklar "Maintenance" moduna alınabilmeli ve bu uçaklara atanmış gelecekteki seferler boşa çıkarılarak (Unassigned) yönetici uyarılmalıdır.

2.2. Veri ve İşlem Gereksinimleri (Database Constraints)

Veritabanı tasarımı, gerçek hayat senaryolarındaki tutarlılığı sağlamak adına aşağıdaki kurallara uymalıdır:

- **Atomik İşlem:** Bilet satın alma süreci (Ödeme Alma > Bilet Oluşturma > Koltuk Kapasitesini Düşürme) tek bir **Transaction** olarak işlenmelidir. Herhangi bir adımda hata olursa (örneğin yetersiz bakiye), tüm işlem geri alınmalı (Rollback) ve veritabanında tutarsız kayıt oluşmamalıdır.
- **Veri Bütünlüğü:** Silme işlemleri (Delete), veri kaybını önlemek amacıyla kısıtlanmalıdır. Bunun yerine "Soft Delete" veya "Durum Değişikliği (Status Change)" yöntemi uygulanarak geçmiş verilerin raporlanabilir kalması sağlanmalıdır.

2.3. Fonksiyonel Olmayan Gereksinimler

- **Güvenlik:** Kullanıcı şifreleri veritabanında açık metin olarak değil, BCrypt algoritması ile hashlenerek saklanmalıdır. API erişimleri JWT(JSON Web Token) ile yetkilendirilmelidir.
- **Kullanıcı Dostu Arayüz:** Sistem, kullanıcıların kolayca etkileşim kurabileceği, sezgisel ve temiz bir arayüze sahip olmalıdır. Yapılan tasarım değişiklikleri (giriş, güncelleme, silme) kullanıcı deneyimini hızlıca yansıtmalıdır.
- **Performans:** Uçuş arama gibi yoğun sorgular, veritabanı indeksleri (Indexing) ile optimize edilmeli ve sorgu süreleri minimize edilmelidir.

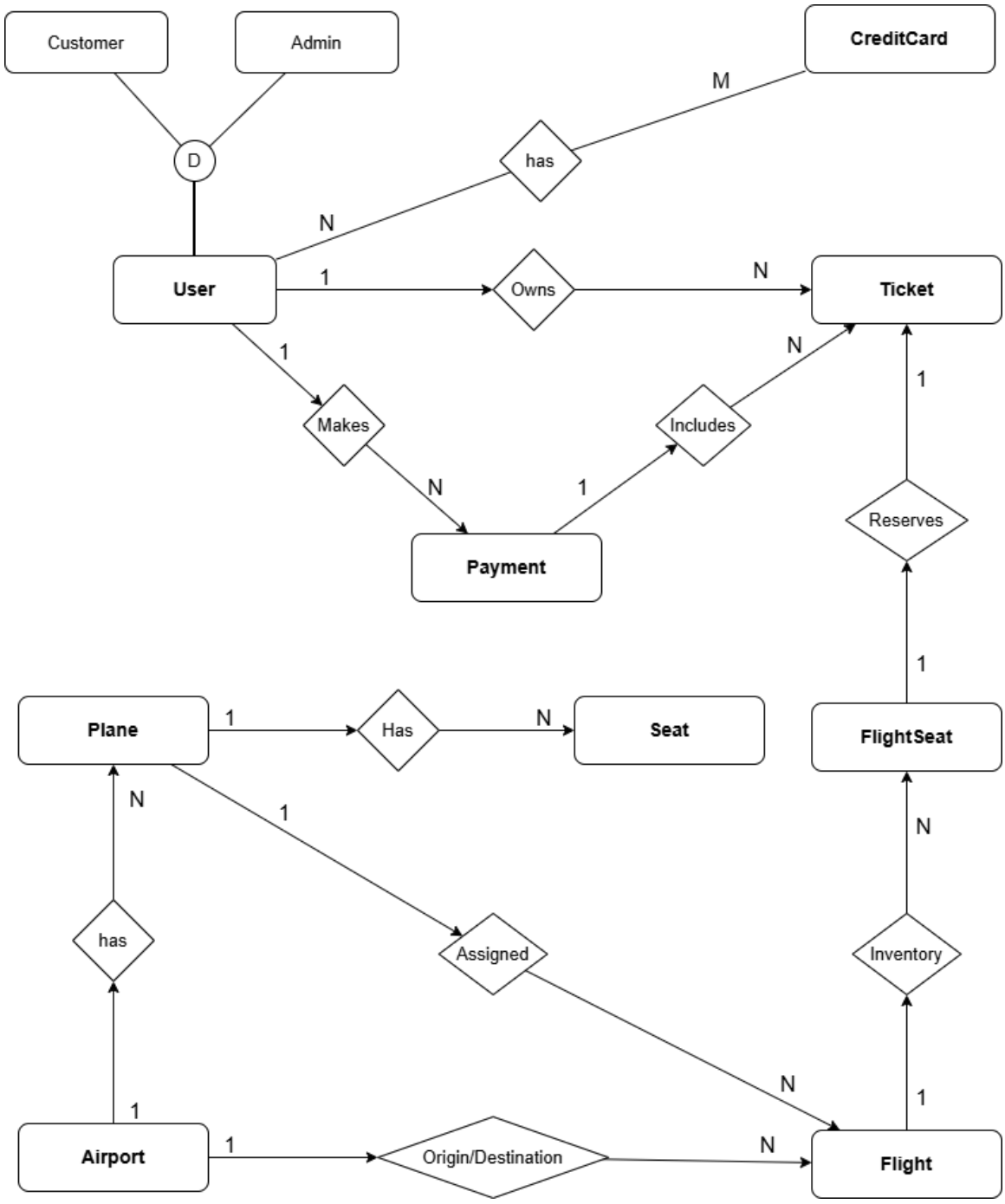
3. Kavramsal Tasarım (EER Diyagramı)

Bu bölümde, sistemin veritabanı yapısını oluşturan varlıklar (Entities), bu varlıkların öznitelikleri (Attributes) ve aralarındaki ilişkilerin (Relationships) kardinalite analizleri detaylandırılmıştır. Tasarım sürecinde BCNF normalizasyon kuralları gözetilmiş ve veri tutarlılığını garanti altına alacak ilişki tipleri seçilmiştir

3.1. Varlık İlişkileri

- **Airport - Plane (1:N):** Bir havalimanında birden fazla uçak bulunabilir.
- **Airport - Flight (1:N):** Bir havalimanından birçok uçuş kalkabilir (Origin) veya inebilir (Destination).
- **Plane - Flight (1:N):** Bir uçak farklı zamanlarda birçok uçuş gerçekleştirebilir.
- **Plane - Seat (1:N):** Bir uçağın fiziksel olarak birçok koltuğu vardır.
- **Flight - FlightSeat (1:N):** Bir uçuşta satılacak birçok koltuk vardır.
- **User - Payment (1:N):** Bir kullanıcı birden fazla ödeme yapabilir.
- **User - Ticket (1:N):** Bir kullanıcının birden fazla bileti olabilir.
- **Payment - Ticket (1:N):** Bir ödeme işleminde birden fazla bilet satın alınabilir.
- **FlightSeat - Ticket (1:1):** Belirli bir uçuşun belirli bir koltuğu sadece tek bir aktif bilete ait olabilir.
- **User - CreditCard (M:N):** Bir kullanıcının çok kartı olabilir, bir kart birden çok kullanıcıda ekli olabilir.

3.2 EER Diyagramı



4. Mantıksal Tasarım ve Şema Diyagramları

Bu bölümde, kavramsal tasarımın fiziksel veritabanı modeline dönüştürülmüş hali sunulmaktadır.

4.1. İlişkisel Şema (Relational Schema)

Tablolarımızın mantıksal yapısı, birincil anahtarlar (PK), birleşik anahtarlar (Composite PK) ve yabancı anahtarlar (FK) ile aşağıda tanımlanmıştır:

- **User:** (user_id [PK], email [Unique], phone_num [Unique], password, type, version)
- **Airport:** (airport_id [PK], iata_code [Unique], city, country, timezone)
- **Plane:** (plane_id [PK], model_type, status, airport_id [FK])
- **Seat:** (plane_id [PK, FK], seat_number [PK], type, status)
- **Flight:** (flight_id [PK], origin_id [FK], destination_id [FK], plane_id [FK], departure_time)
- **FlightSeat:** (flight_id [PK, FK], seat_number [PK], availability, price, version)
- **Payment:** (payment_id [PK], user_id [FK], total_amount, status)
- **Ticket:** (ticket_id [PK], flight_id [FK], seat_number [FK], payment_id [FK], user_id [FK])
- **CreditCard:** (card_id [PK])
- **User_CreditCards:** (user_id[PK], card_id[PK])

4.2. Normalizasyon (BCNF)

Veritabanı tasarımımızda veri tutarlılığını sağlamak, güncelleme anomalilerini önlemek ve disk alanını verimli kullanmak amacıyla tablolarımız BCNF seviyesinde normalize edilmiştir.

Normalizasyon sürecinde uyguladığımız spesifik adımlar ve gerekçeleri şöyledir:

1. Çoklu Değerlerin Ayrıştırılması (1NF Uygulaması):

- **Sorun:** Bir kullanıcının birden fazla kredi kartı olabilir. Eğer User tablosunda credit_cards sütunu açıp kart numaralarını virgülle ayırarak tutsaydık, atomiklik kuralı (1NF) ihlal edilirdi.

- **Çözüm:** Kredi kartı bilgileri CreditCard isimli ayrı bir tabloya taşınmış ve User_Cards ara tablosu ile kullanıcıya bağlanmıştır. Böylece her hücrede tek bir veri tutulması sağlanmıştır.

2. Kısmi Bağımlılıkların Giderilmesi (2NF Uygulaması):

- **Strateji:** Tasarımımızda Doğal Anahtarlar (Örn: TC Kimlik No, Email) yerine, sistem tarafından yönetilen Yapay Anahtarlar Birincil Anahtar olarak tercih edilmiştir.
- **Sonuç:** Tüm tablolarda, anahtar olmayan sütunlar , birleşik bir anahtarın parçasına değil, doğrudan ve tamamen id sütununa bağımlıdır. Bu sayede 2NF kuralı doğal olarak sağlanmıştır.

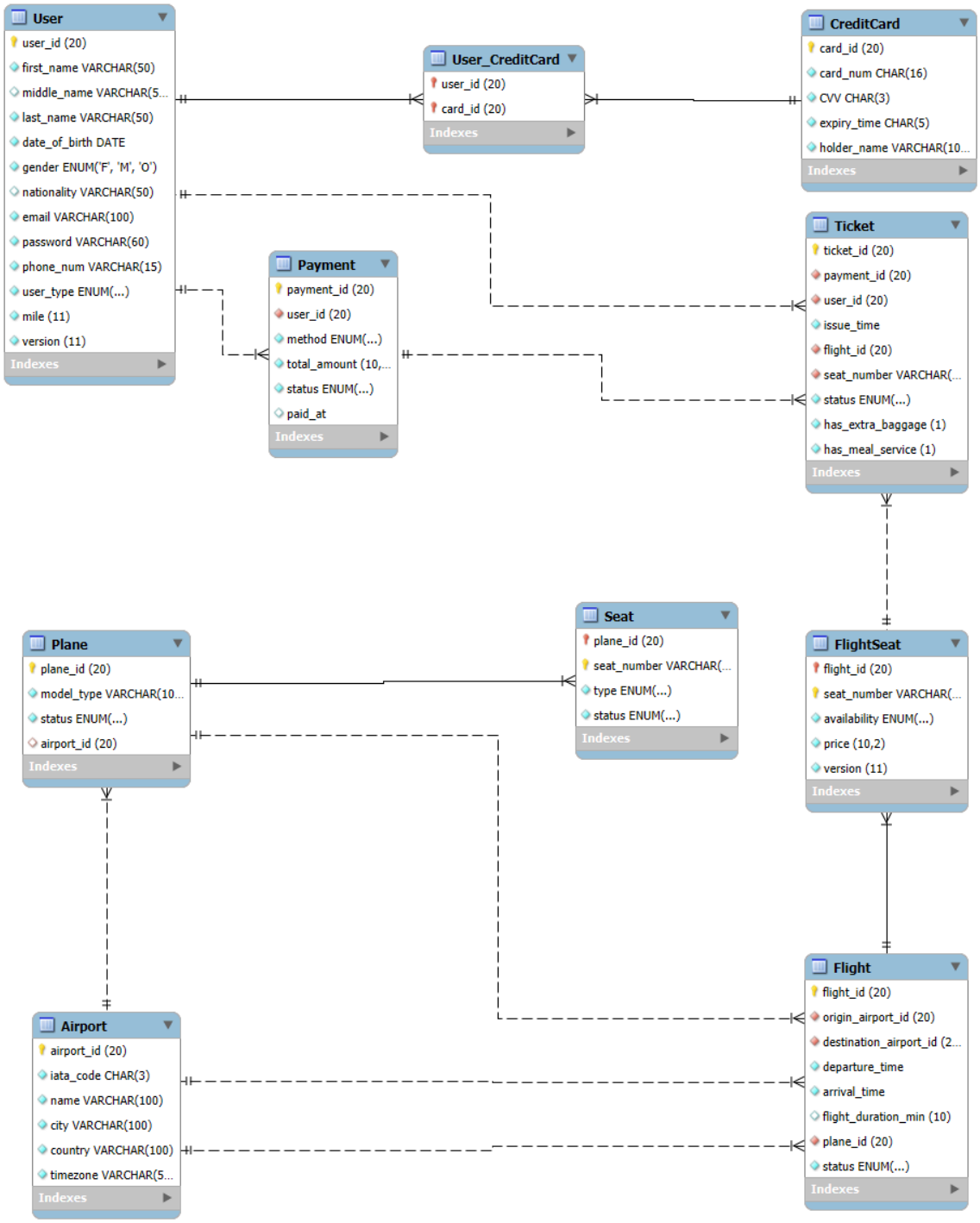
3. Geçişli (Transitive) Bağımlılıkların Giderilmesi (3NF Uygulaması):
Normalizasyonun en kritik adımı burada uygulanmıştır. Veri tekrarını önlemek için şu ayrıştırmalar yapılmıştır:

- **Havalimanı Detayları:** Flight tablosunda havalimanının city, country, timezone bilgileri tutulmamıştır. Bunun yerine sadece airport_id referans olarak tutulmuş, detaylar Airport tablosuna taşınmıştır. Böylece "İstanbul" ismini değiştirmek istediğimizde binlerce uçuş satırını güncellemek yerine sadece Airport tablosundaki tek bir satırı güncellememiz yeterli olur.
- **Uçak Modeli:** Ticket tablosunda yolcunun bindiği uçağın modeli (Boeing 737 vb.) tutulmaz. Bu bilgiye Ticket -> Flight -> Plane ilişkisi üzerinden erişilir.

4. BCNF (Boyce-Codd Normal Form) Analizi: Tasarımımızda 3NF'in ötesine geçilerek, tüm determinantların aday anahtar olması sağlanmıştır.

- **Benzersizlik Kısıtları:** User tablosunda email ve phone_num alanları, Airport tablosunda iata_code alanı UNIQUE olarak işaretlenmiştir. Bu alanlar fonksiyonel olarak diğer sütunları belirlese bile, kendileri de birer Candidate Key olduğu için BCNF kuralı ihlal edilmemiştir.

4.3. Şema Diyagramı



5. Tasarımın Uyarlanması (Implementation)

5.1 Yazılım ve Donanım Ortamı

- Backend: Java 21, Spring Boot 3.3.3
- Frontend: React 18 (Vite), Material-UI (MUI)
- Veritabanı: MySQL 8.0 (TIDB Cloud kullanılarak ortak erişim sağlanmıştır.)
- ORM (Object-Relational Mapping): Hibernate (JPA)
- JDBC: Spring Data JPA
- Build Tool: Maven

5.2 Tabloların Oluşturulması

Veritabanı tabloları önce Tables.sql dosyası ile manuel olarak oluşturulmuştur. Spring Boot application.properties dosyasında hibernate.ddl-auto=update ayarı ile entity sınıfları ile veritabanı şeması senkronize edilmiştir.

Temel Tablolar ve Alanları:

User Tablosu: Kullanıcıların tutulduğu tablo

- user_id (BIGINT, PK, AUTO_INCREMENT): Kullanıcı benzersiz kimliği
- first_name, middle_name, last_name (VARCHAR): Kullanıcı ad bilgileri
- date_of_birth (DATE): Doğum tarihi
- gender (ENUM: F, M, O): Cinsiyet bilgisi
- email (VARCHAR, UNIQUE): Benzersiz email adresi
- password (VARCHAR 60): Şifrelenmiş parola (BCrypt)
- phone_num (VARCHAR 15, UNIQUE): Benzersiz telefon numarası
- user_type (ENUM: customer, admin): Kullanıcı rolü
- mile (INT, DEFAULT 0): Müşteri mil puanı
- version (INT): Optimistic locking için versiyon numarası

Flight Tablosu: Uçuş bilgilerinin tutulduğu tablo

- flight_id (BIGINT, PK, AUTO_INCREMENT): Uçuş benzersiz kimliği
- origin_airport_id (BIGINT, FK → Airport): Kalkış havalimanı
- destination_airport_id (BIGINT, FK → Airport): Varış havalimanı
- departure_time (TIMESTAMP): Kalkış zamanı

- arrival_time (TIMESTAMP): Varış zamanı
- plane_id (BIGINT, FK → Plane): Uçak referansı
- status (ENUM: SCHEDULED, ACTIVE, COMPLETED, CANCELLED): Uçuş durumu

FlightSeat Tablosu (Composite Key): Aktif uçuşları olan uçakların koltuk bilgileri

- flight_id (BIGINT, PK, FK → Flight): Uçuş referansı
- seat_number (VARCHAR 3, PK): Koltuk numarası (örn: 12A)
- availability (ENUM: available, reserved, sold): Koltuk durumu
- price (DECIMAL 10,2): Koltuk fiyatı
- version (INT): Race condition önleme için versiyon numarası

Ticket Tablosu: Bilet bilgilerinin tutulduğu tablo

- ticket_id (BIGINT, PK, AUTO_INCREMENT): Bilet benzersiz kimliği
- payment_id (BIGINT, FK → Payment): Ödeme referansı
- user_id (BIGINT, FK → User): Kullanıcı referansı
- flight_id, seat_number (FK → FlightSeat): Uçuş ve koltuk bilgisi
- issue_time (TIMESTAMP): Bilet kesim zamanı
- status (ENUM: booked, cancelled, completed, pending): Bilet durumu
- has_extra_baggage (BOOLEAN): Ekstra bagaj seçeneği
- has_meal_service (BOOLEAN): Yemek servisi seçeneği

Payment Tablosu: Kullanıcının yaptığı harcamalar ve iade işlemleri (hesap hareketleri)

- payment_id (BIGINT, PK, AUTO_INCREMENT): Ödeme benzersiz kimliği
- user_id (BIGINT, FK → User): Ödeme yapan kullanıcı
- method (ENUM: card, mile, cash): Ödeme yöntemi
- total_amount (DECIMAL 10,2): Toplam tutar
- status (ENUM: pending, paid, refunded, failed): Ödeme durumu
- paid_at (TIMESTAMP): Ödeme zamanı

Airport Tablosu: Havaalanlarının bilgilerinin tutulduğu tablo

- airport_id (BIGINT, PK, AUTO_INCREMENT): Havalimanı benzersiz kimliği
- iata_code (CHAR 3, UNIQUE): IATA kodu (örn: IST, JFK)

- name, city, country (VARCHAR): Havalimanı bilgileri
- timezone (VARCHAR): Zaman dilimi (UTC formatında)

Plane Tablosu: Uçaklarının bilgilerinin tutulduğu tablo

- plane_id (BIGINT, PK, AUTO_INCREMENT): Uçak benzersiz kimliği
- model_type (VARCHAR): Uçak modeli
- status (ENUM: active, maintenance, retired): Uçak durumu
- airport_id (BIGINT, FK → Airport): Bulunduğu havalimanı

Seat Tablosu (Composite Key): Tüm **u**çakların koltuklarının bilgilerinin tutulduğu tablo

- plane_id (BIGINT, PK, FK → Plane): Uçak referansı
- seat_number (VARCHAR 3, PK): Koltuk numarası
- type (ENUM: economy, premium_economy, business, first): Koltuk sınıfı
- status (ENUM: active, unavailable): Koltuk durumu

CreditCard Tablosu: Kredi kartlarının bilgilerinin tutulduğu tablo

- card_id (BIGINT, PK, AUTO_INCREMENT): Kart benzersiz kimliği
- card_num (CHAR 16, UNIQUE): Kart numarası
- CVV (CHAR 3): Güvenlik kodu
- expiry_time (CHAR 5): Son kullanma tarihi (MM/YY)
- holder_name (VARCHAR 100): Kart sahibi adı

User_CreditCard Tablosu (Many-to-Many Join Table):

- user_id (BIGINT, PK, FK → User)
- card_id (BIGINT, PK, FK → CreditCard)

5.3 Görünümler (Views)

Projede view kullanılmamıştır. Bunun yerine JPA repository'lerinde JOIN işlemleri runtime'da gerçekleştirilmiştir.

View Kullanılmama Nedenleri:

1. Dinamik Veri Yapısı:

Projede flight ve ticket verileri sürekli değişmektedir (koltuk durumları, uçuş statüleri, ödeme durumları). View'lar statik sorgu sonuçlarını saklar ve her sorgulamada yeniden hesaplanır, ancak materialized view olmadığı için performans avantajı sınırlıdır. ORM (JPA) ile yapılan sorgular zaten optimize edilmiştir ve gerektiğinde cache mekanizması kullanılabilir.

2. ORM (JPA) Avantajları:

Spring Data JPA, JOIN işlemlerini JPQL veya method naming ile otomatik olarak yönetir. Entity ilişkileri (@ManyToOne, @OneToMany) sayesinde code-first yaklaşım ile veritabanı yapısı ve sorguları senkronize tutulur. View kullanımı entity mapping'lerini karmaşıklaştırır ve Hibernate'in lazy loading, caching gibi özelliklerinden faydalanmayı zorlaştırır.

3. Flexibility (Esneklik):

Farklı endpoint'lerde farklı JOIN kombinasyonları gerekebilir. Örneğin:

- Uçuş arama: Flight + Origin Airport + Destination Airport
- Bilet listeleme: Ticket + Flight + Airport + Payment + User
- Ödeme geçmişi: Payment + User (Flight bilgisi gerekmez)

Her kombinasyon için ayrı view oluşturmak ve sürekli yenilemek yerine, repository method'larında @Query ile ihtiyaca göre JOIN yapılması daha esnektir.

5.4 İndisler (Indexes)

Performans optimizasyonu için sık kullanılan sorgulara yönelik indexler oluşturulmuştur:

User Tablosu:

- email: Email ile login işlemi (authentication)
- phone_num: Telefon numarası ile kullanıcı arama

İki attribute de UNIQUE constraint ile otomatik indexlidir, bu nedenle ekstra index oluşturulmamıştır.

Airport Tablosu:

- idx_airport_iata: IATA kodu ile hızlı havalimanı bulma
- idx_airport_city: Şehre göre havalimanı listeleme
- idx_airport_country: Ülkeye göre havalimanı filtreleme

Flight Tablosu :

- idx_flight_origin_dest: (origin_airport_id, destination_airport_id) composite index - Uçuş arama sayfasında en sık kullanılan sorgu
- idx_flight_departure: departure_time ile tarih bazlı filtreleme
- idx_flight_plane: plane_id ile belirli uçağın uçuşlarını listeleme
- idx_flight_status: Status'e göre aktif/iptal edilmiş uçuşları filtreleme

FlightSeat Tablosu:

- idx_flightseat_flight: flight_id ile uçuşun tüm koltuklarını getirme
- idx_flightseat_availability: (flight_id, availability) composite index - Müsait koltukları hızlı bulma

Ticket Tablosu:

- idx_ticket_user: user_id ile kullanıcının tüm biletlerini listeleme (My Flights sayfası)
- idx_ticket_payment: payment_id ile ödemeye ait biletleri bulma
- idx_ticket_flight: flight_id ile uçuşa ait biletleri listeleme

Payment Tablosu:

- idx_payment_user: user_id ile kullanıcının ödeme geçmişini getirme (Payment History sayfası)
- idx_payment_status: Ödeme durumuna göre filtreleme (pending, paid, failed)

Plane Tablosu:

- idx_plane_airport: airport_id ile havalimanındaki uçakları listeleme

- idx_plane_status: Status'e göre aktif/bakımdaki uçakları filtreleme

5.5 Sorgu Tasarımları

JOIN Stratejileri

- INNER JOIN: Zorunlu ilişkilerde kullanılmıştır (Flight-Airport ilişkisi, her uçuşun mutlaka origin ve destination airport'u olmalı)
- LEFT JOIN FETCH: N+1 problemini önlemek için eager loading yapılmıştır (Örnek: TicketRepository'de kullanıcının biletleri çekilirken flight, airport, payment bilgileri tek sorguda getirilir)

Composite Key Kullanımı:

- FlightSeat: (flight_id, seat_number) - Belirli bir uçuştaki belirli koltuğu unique olarak tanımlar
- Seat: (plane_id, seat_number) - Belirli bir uçaktaki belirli koltuğu unique olarak tanımlar
- User_CreditCard: (user_id, card_id) - Many-to-Many ilişki için join table

Özel Sorgular (Custom Queries):

- TicketRepository: @Query ile kullanıcının biletlerini flight ve airport detayları ile birlikte getiren optimize edilmiş JPQL sorgusu
- FlightRepository: Origin ve destination airport'a göre, belirli tarih aralığında uçuş arama sorgusu
- PaymentRepository: Kullanıcının ödeme geçmişini paid_at tarihine göre sıralı getirme

Repository Method Naming Convention:

SELECT sorgularıdır, JPA Repository kolaylık sağlar. Örneğin:

- findByEmail, findByPhoneNum (UserRepository)
- findByOriginAirportAirportIdAndDestinationAirportAirportId (FlightRepository)
- findByFlightIdAndAvailability (FlightSeatRepository)
- findByUserUserId (TicketRepository)

5.6 Transaction Yönetimi

@Transactional Kullanımı:

- Bilet satın alma işlemi: Payment oluşturma, Ticket oluşturma ve FlightSeat güncelleme işlemleri tek transaction içinde yapılır. Herhangi bir hata durumunda tüm işlem geri alınır (rollback)
- Kredi kartı ekleme işlemi: User ve CreditCard many-to-many ilişkisi transaction içinde güncellenir
- Uçuş oluşturma: Flight oluşturulurken ilgili FlightSeat kayıtları da otomatik oluşturulur

Optimistic Locking (@Version):

- FlightSeat tablosunda version kolonu: Aynı koltuğu aynı anda birden fazla kullanıcının satın almasını önler (race condition)
- User tablosunda version kolonu: Kullanıcı bilgileri güncellenirken veri tutarlılığını sağlar
- Çakışma durumunda OptimisticLockException fırlatılır ve kullanıcıya bilgi verilir

Cascade İşlemleri:

- User-CreditCard: CascadeType.PERSIST, MERGE (Kullanıcı silindiğinde kartlar silinir - DELETE CASCADE)
- Flight-FlightSeat: FK constraint ile ON DELETE RESTRICT (Uçuşa ait bilet varsa uçuş silinemez)
- Payment-Ticket: ON DELETE RESTRICT (Ödeme kaydı korunmalı)

5.7 REST API Tasarımı ve Endpoint'ler:

Tüm API endpoint’leri RESTful mimari prensiplerine göre tasarlanmıştır.

Authentication Controller (/api/auth):

- POST /api/auth/signup: Kullanıcı kaydı
- POST /api/auth/login: Kullanıcı girişi, JWT token döner
    - Request: { email, password }
    - Response: { token, userId, email, firstName, lastName, userType }

Flight Controller (/api/flights):

- GET /api/flights: Tüm uçuşları listele
- GET /api/flights/{id}: ID'ye göre uçuş detayı
- POST /api/flights/search: Uçuş arama
    - Request: { originAirportId, destinationAirportId, departureDate }
    - Response: Flight listesi (origin, destination airport bilgileriyle)
- POST /api/flights (Admin): Yeni uçuş oluşturma
- GET /api/flights/{id}/seats: Uçuşun koltuk durumları

Ticket Controller (/api/tickets):

- GET /api/tickets/user/{userId}: Kullanıcının biletleri
- POST /api/tickets: Bilet satın alma (Payment ile birlikte transaction)

Payment Controller (/api/payments):

- POST /api/payments: Ödeme işlemi
- GET /api/payments/user/{userId}: Kullanıcının ödeme geçmişi
- POST /api/tickets/{ticketId}/cancel: Bilet iptali

User Controller (/api/users):

- GET /api/users/{id}: Kullanıcı bilgileri
- PUT /api/users/{id}: Kullanıcı güncelleme
- PUT /api/users/{id}/change-password: Şifre değiştirme
- GET /api/users/{emailOrPhoneNum}: Email veya telefon ile kullanıcı bilgisi
- DELETE /api/users/{userId}: Kullanıcı silme
- GET /api/users/{userId}/credit-cards: Kullanıcının kayıtlı kartları
- POST /api/users/{userId}/credit-cards: Kredi kartı ekleme
- DELETE /api/users/{userId}/credit-cards/{cardId}: Kredi kartı silme
- PUT /api/users/{userId}/change-password: Şifre değiştirme

Admin Controller (/api/admin):

- POST /api/admin/planes: Uçak ekleme
- GET /api/admin/planes: Uçak listesi

- GET /api/admin/planes: Tüm uçakları listele
- POST /api/admin/planes: Yeni uçak ekleme
- DELETE /api/admin/planes/{planeId}: Uçak silme
- GET /api/admin/planes/{planeId}: Uçak detayı
- GET /api/admin/planes/{planeId}/seats: Uçağın koltuk yapısı
- GET /api/admin/airports: Tüm havalimanlarını listele
- POST /api/admin/airports: Havalimanı ekleme
- DELETE /api/admin/airports/{airportId}: Havalimanı silme
- GET /api/admin/airports/{airportId}: Havalimanı detayı
- GET /api/admin/airports/{airportId}/planes: Havalimanındaki uçaklar
- GET /api/admin/flights: Tüm uçuşları listele
- POST /api/admin/flights: Yeni uçuş oluşturma
- POST /api/admin/flights/{flightId}/cancel: Uçuş iptal etme
- POST /api/admin/planes/{planeId}/malfunction: Koltuk arızası bildirme
- PUT /api/admin/planes/{planeId}/status: Uçak durumu güncelleme
- PUT /api/admin/planes/{planeId}/airport/{airportId}: Uçağın havalimanını değiştirme
- PUT /api/admin/planes/{planeId}/seats/{seatNumber}/status: Koltuk durumu güncelleme

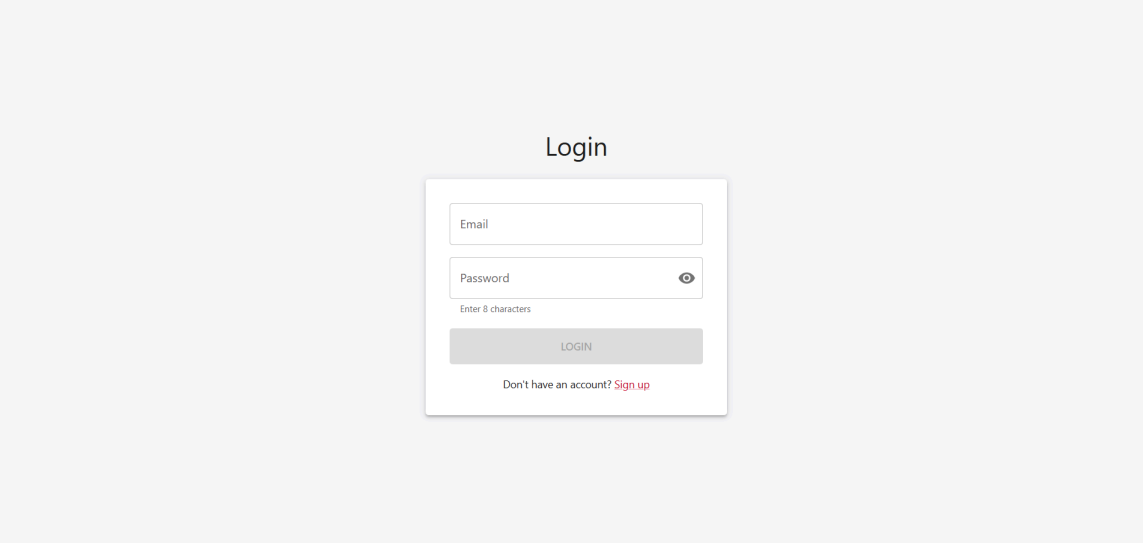
6. Uygulama Program Tanıtımı ve Örnek Kullanım

Geliştirilen sistemin arayüzleri, son kullanıcı ve admin deneyimini optimize edecek şekilde React (Material UI) ile tasarlanmıştır. Aşağıda temel kullanım senaryoları ve ekran görüntüleri yer almaktadır.

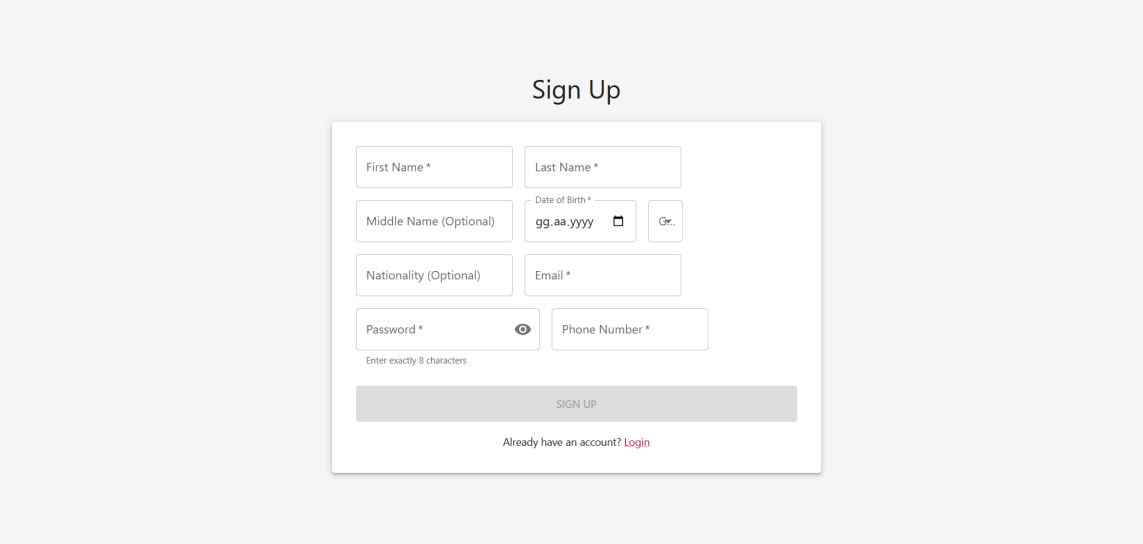
6.1. Giriş ve Kayıt Ekranı (Authentication)

Kullanıcıların güvenli bir şekilde sisteme giriş yapmasını sağlayan modüldür.

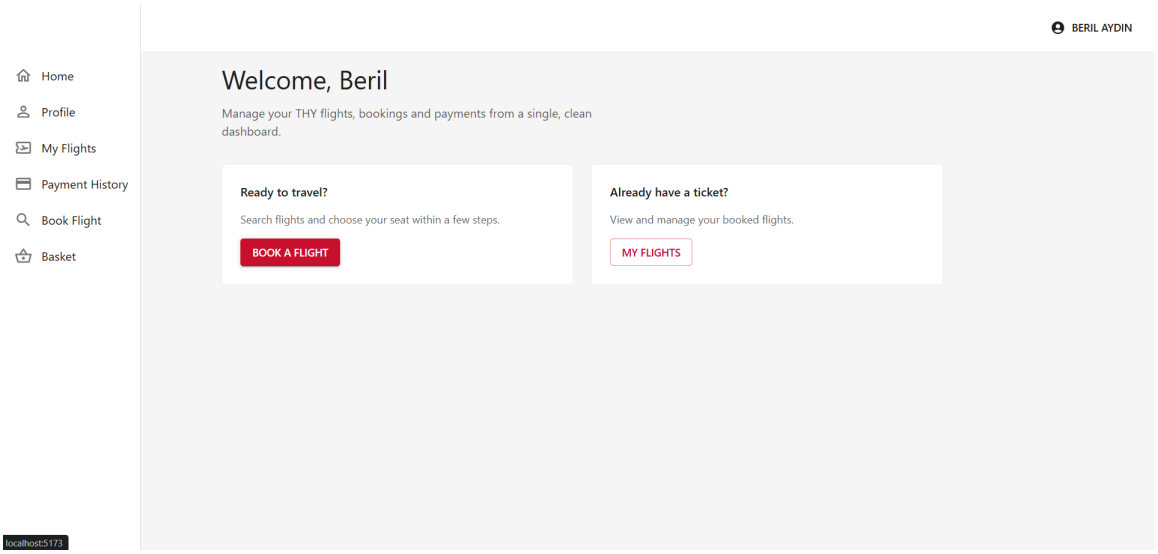
- **Teknik Detay:** Giriş yapıldığında Backend'den dönen JWT Token, Frontend tarafında localStorage'da saklanır ve sonraki tüm API isteklerinin Header kısmına Authorization: Bearer olarak eklenir.



Görsel 1: Login (Giriş) Ekranı



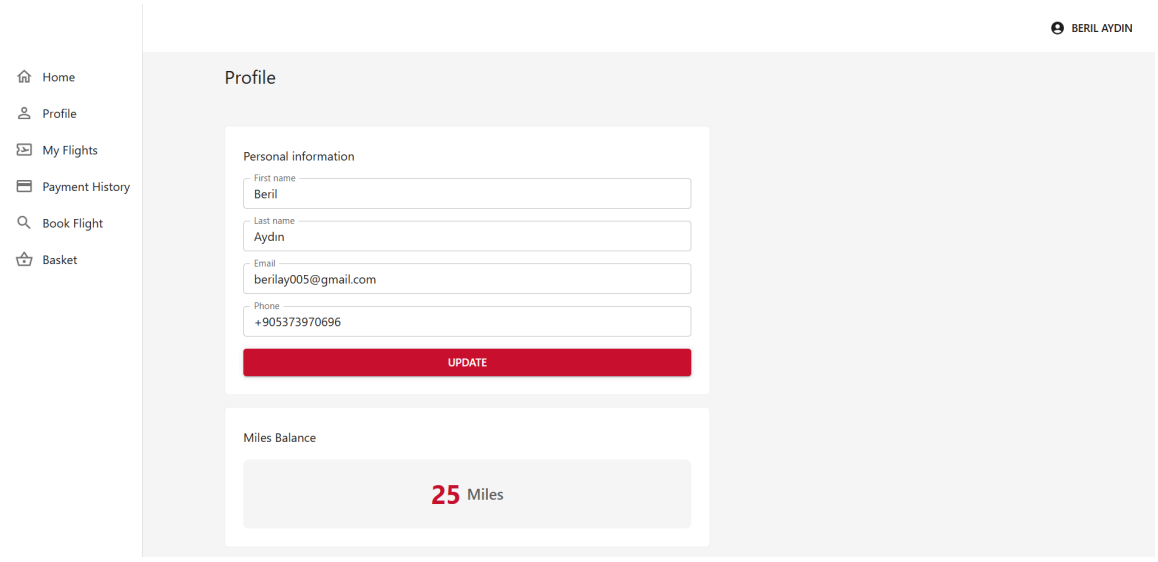
Görsel 2: Signup (Kayıt) Ekranı

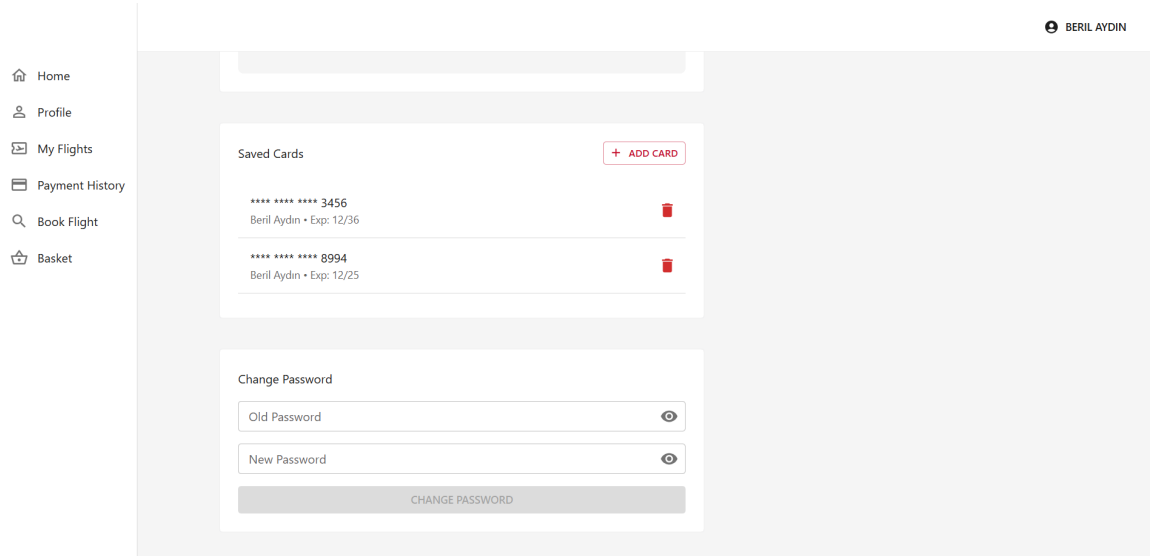


Görsel 3: Ana Ekran

6.2. Profil Ekranı (Profile)

Kullanıcının bilgilerini görebildiği ve güncelleme yapabildiği ekrandır.



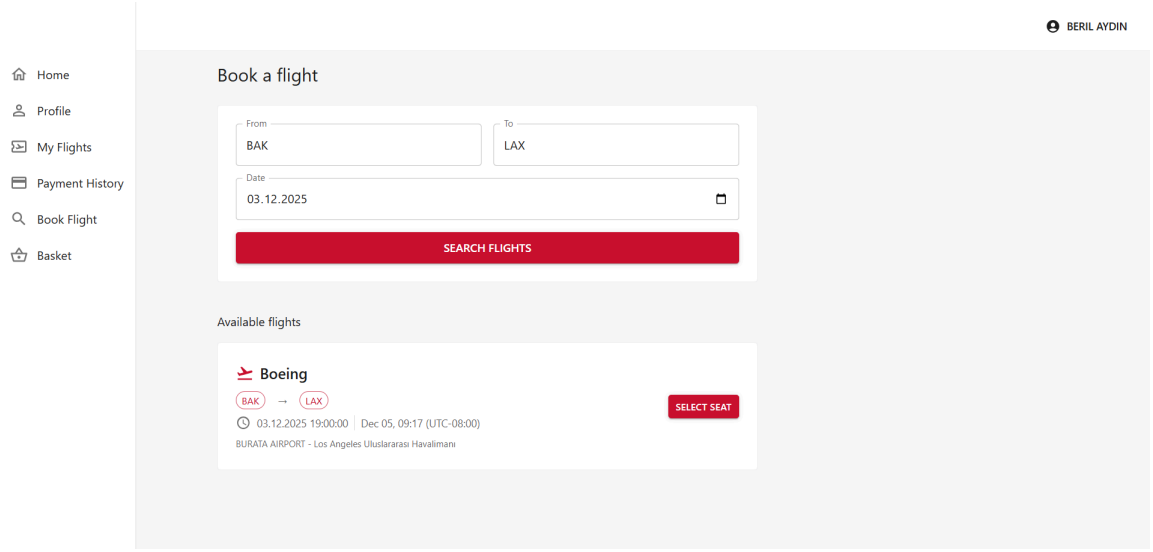


Görsel 4-5: Profil Ekranı

6.3. Uçuş Arama ve Listeleme (Flight Search)

Kullanıcının kalkış/varış noktası ve tarih seçerek uygun uçuşları listelediği ekrandır.

- **Teknik Detay:** Bu ekrandaki sorgu, veritabanında idx_flight_origin_dest indeksini kullanarak çalışır. Flight tablosu Airport tablosu ile INNER JOIN FETCH yapılarak, havalimanı isimleri tek bir sorguda getirilir.

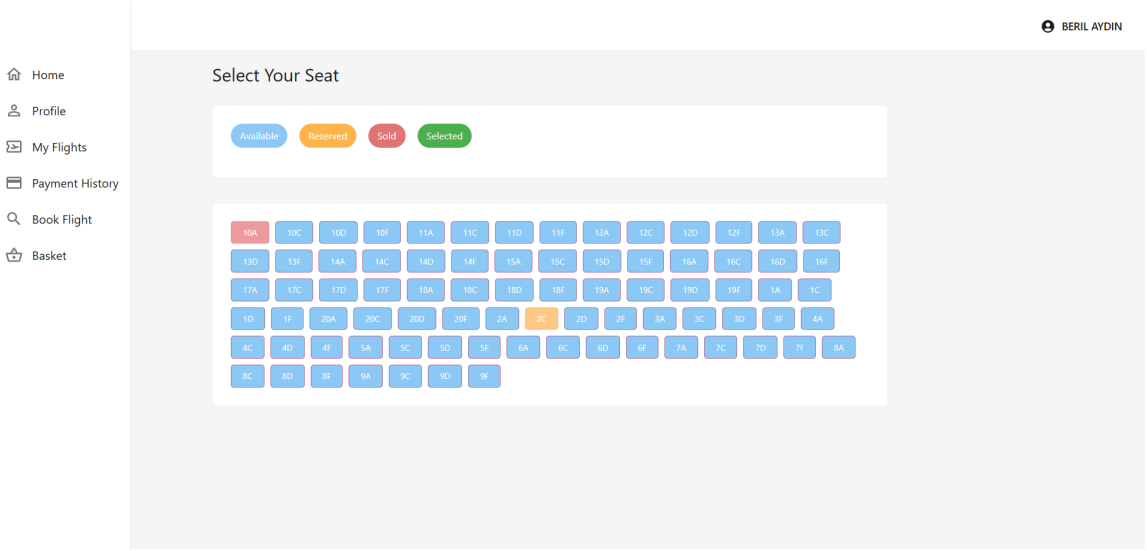


Görsel 6: Uçuş Arama Ekranı ve Örnek Sonuç

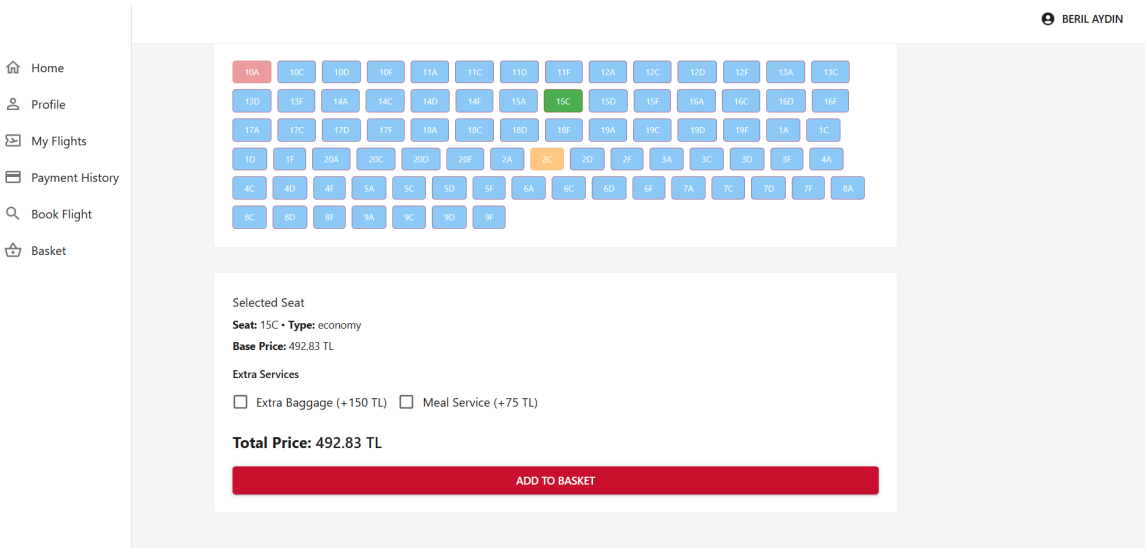
6.4. Koltuk Seçimi (Seat Selection)

Kullanıcının uçağın güncel doluluk durumuna göre koltuk seçtiği interaktif ekrandır.

- **Teknik Detay:** Bu ekran açıldığında GET /api/flights/{id}/seats endpoint'i çağrılır. FlightSeat tablosundan o uçuşa ait tüm koltukların availability durumu çekilir. "Sold" olanlar kırmızı, “Reserved” olanlar sarı,"Available" olanlar mavi gösterilir.



Görsel 7: Koltuk Seçim Ekranı

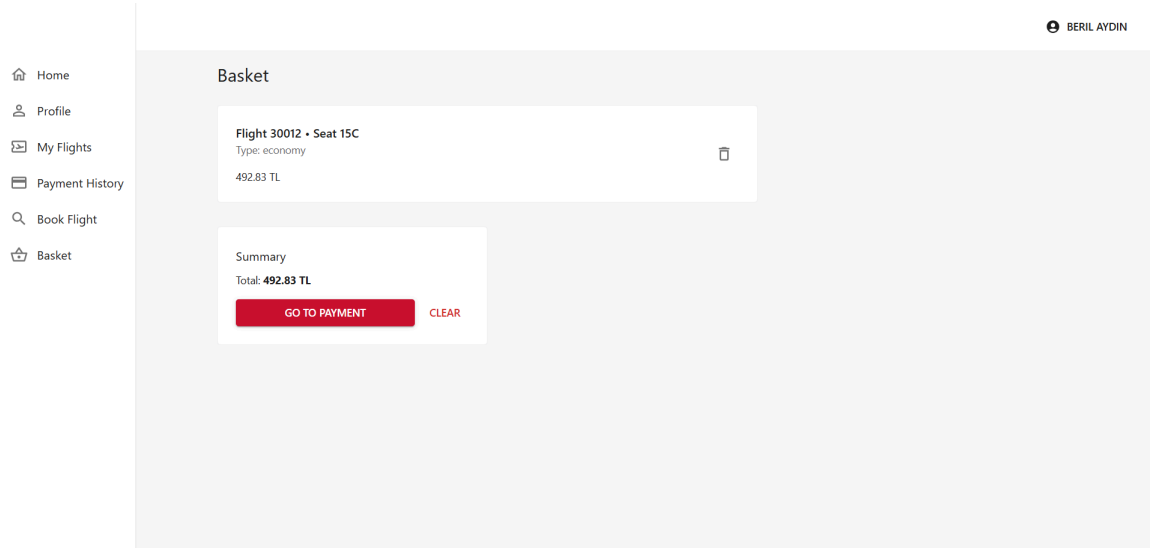


Görsel 8: Seçilen Koltuk Bilgileri ve Ekstra Servis Seçeneği

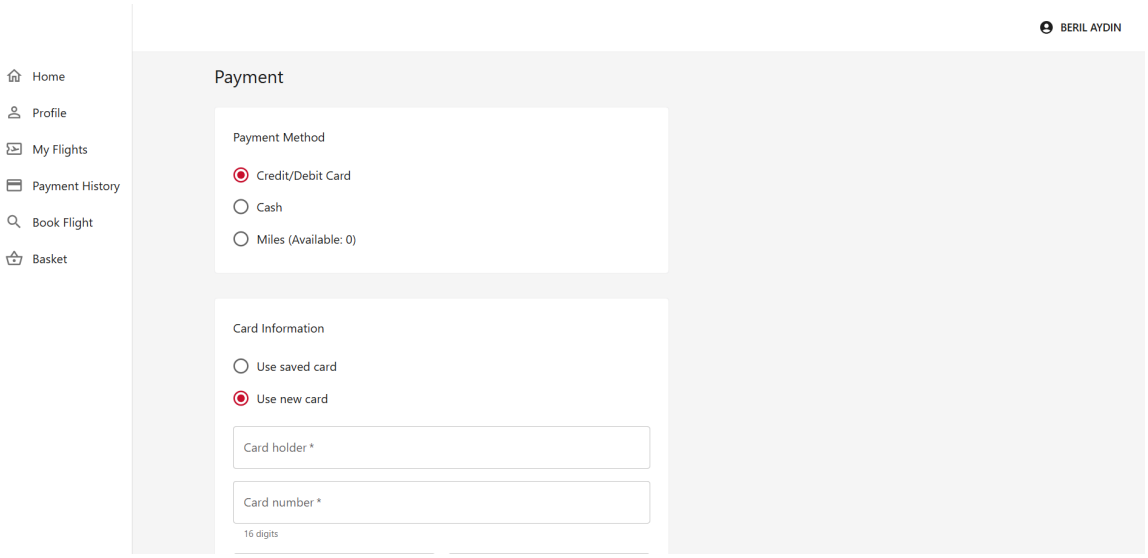
6.5. Ödeme ve Biletleme (Payment & Transaction)

Sepet ekranından ödeme ekranına gidilir. Burada ödeme yöntemi seçilir, kredi kartı seçildiyse kullanıcının kredi kartı bilgilerini girer veya kayıtlı kart seçilir ve işlem tamamlanır. Alınan biletler My Flight ekranında, harcamalar ve iadeler Payments ekranında gösterilir.

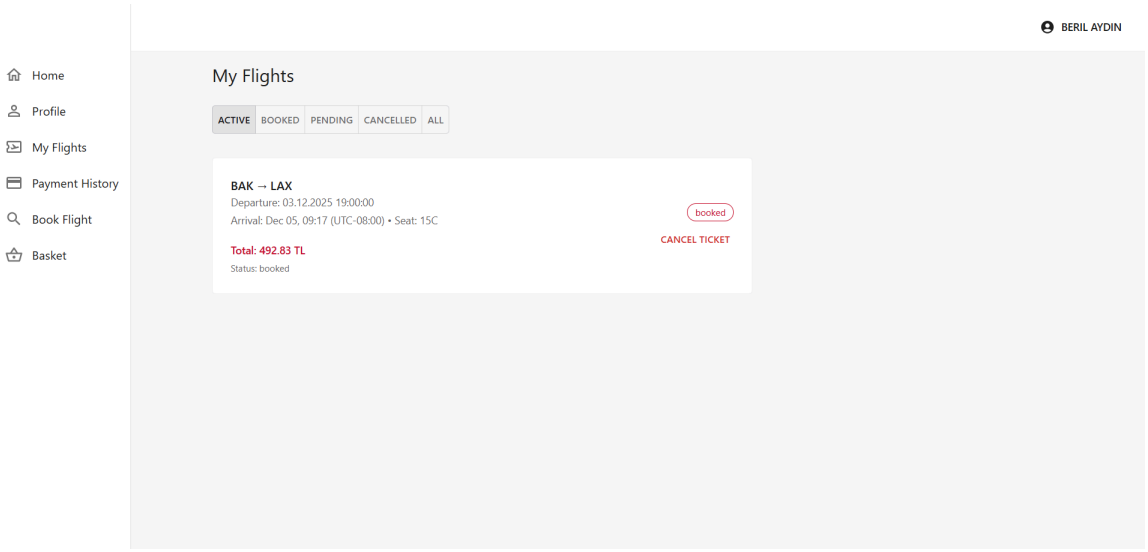
- **Teknik Detay (ACID):** "Ödeme Yap" butonuna basıldığında Backend'de @Transactional blok başlar.
    1. Ödeme (Payment) kaydı oluşturulur.
    2. Seçilen koltuğun FlightSeat tablosundaki durumu SOLD olarak güncellenir.
    3. Bilet (Ticket) oluşturulur.
    4. Tüm işlemler başarılıysa Commit edilir.



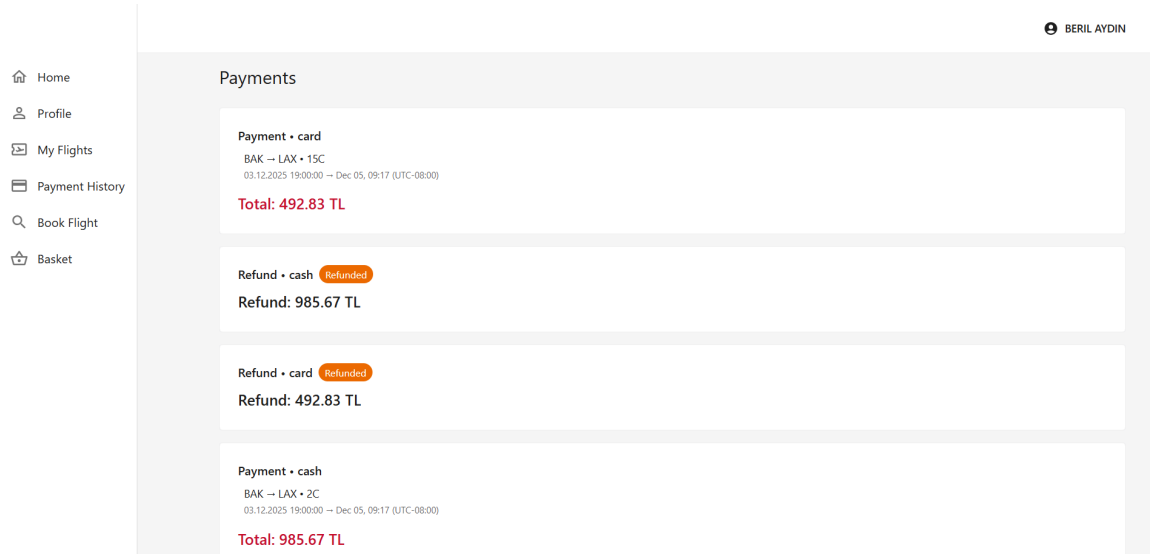
Görsel 9: Sepet ekranı



Görsel 10: Ödeme Ekranı



Görsel 11: Bilet Ekranı (My Flights)

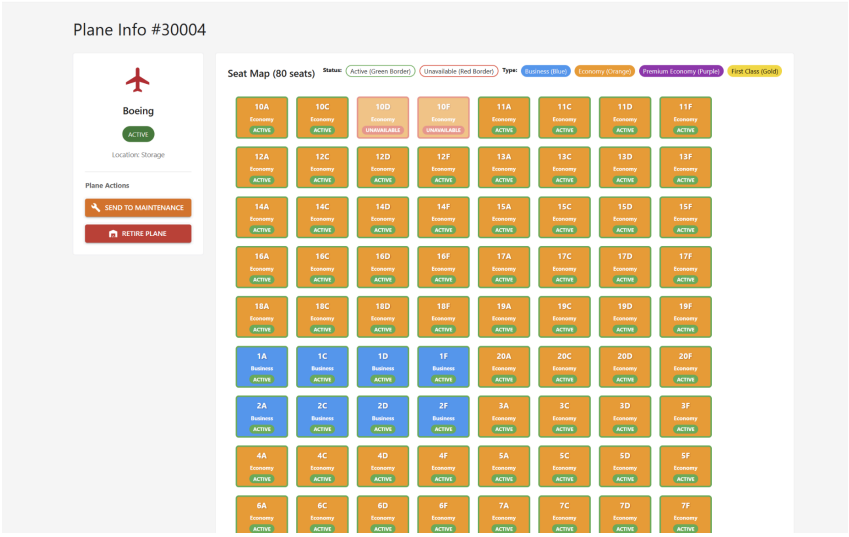


Görsel 12: Hesap Hareketleri Ekranı (Payments)

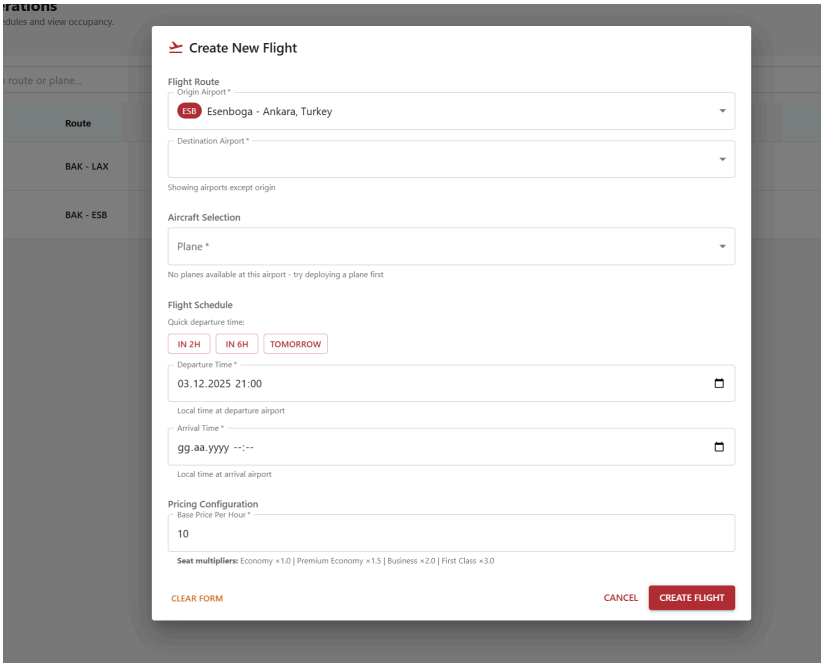
6.6. Admin Paneli (Operasyonel Yönetim)

Yöneticinin yeni uçak/uçuş eklediği veya uçuşları iptal ettiği yönetim panelidir.

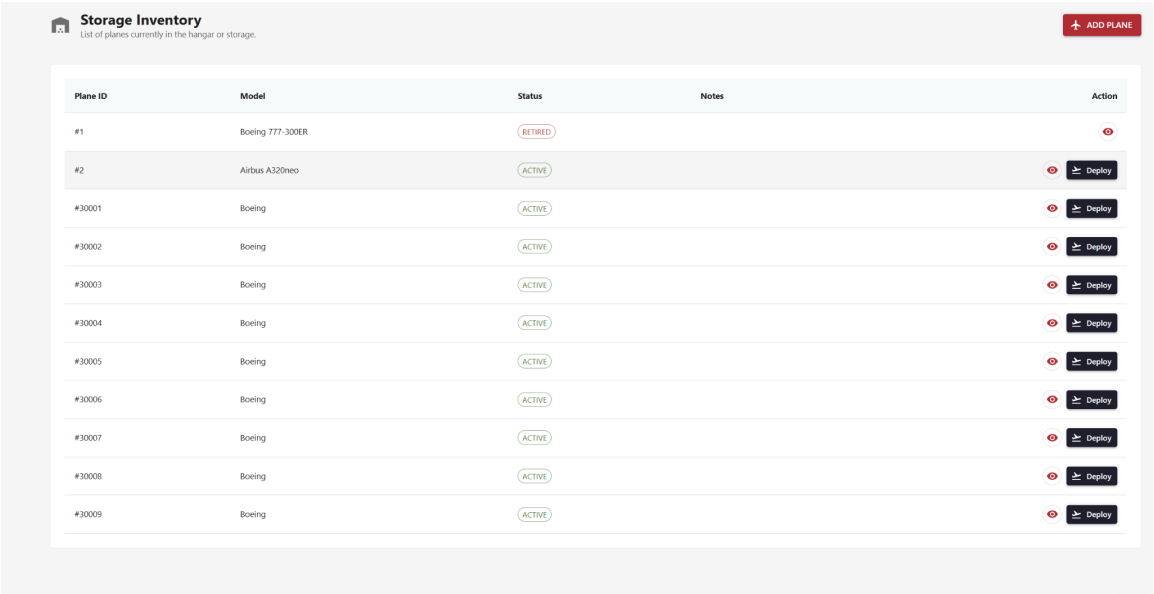
- **Teknik Detay:** Admin yeni bir uçuş eklediğinde, sistem arka planda o uçağın Seat tablosundaki fiziksel koltuk şablonunu kopyalar ve FlightSeat tablosuna o uçuş için binlerce satır (Envanter) oluşturur.



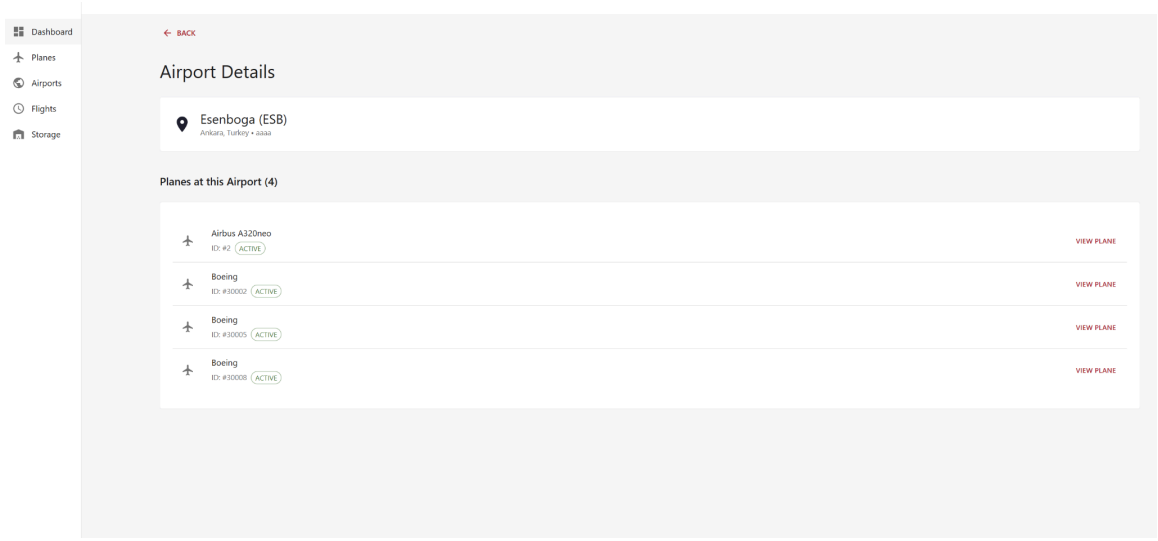
Görsel 13: Uçak Detayları Ekranı (Plane Info)



Görsel 14: Uçuş Oluşturma Ekranı



Görsel 15: Uçak Depolama Ekranı (Storage Page)



Görsel 15: Havaalanı Detay Ekranı (Airport Details Page)

7. Sonuç

Bu proje kapsamında, havacılık sektörünün yüksek eşzamanlılık (High Concurrency) ve veri tutarlılığı gerektiren karmaşık yapısı, ilişkisel veritabanı prensiplerine uygun olarak modellenmiştir.

Projenin en önemli kazanımları şunlardır:

1. **Veri Tutarlılığı:** Seat ve FlightSeat tablolarının ayrılması ve Composite Key kullanımı sayesinde, veritabanı seviyesinde veri tekrarı önlenmiş ve BCNF normalizasyon formu sağlanmıştır.
2. **Concurrency Yönetimi:** FlightSeat tablosuna eklenen version sütunu ve **Optimistic Locking** stratejisi sayesinde, "Çifte Rezervasyon (Double Booking)" problemi sistemsel olarak engellenmiştir.
3. **Performans:** Sık kullanılan sorgular (Uçuş arama, Bilet listeleme) için stratejik **İndeksler (B-Tree)** oluşturulmuş ve JOIN FETCH stratejisi ile N+1 sorgu problemi çözülmüştür.
4. **Güvenlik ve Mimari:** Spring Security ile JWT tabanlı güvenli bir mimari kurulmuş, RESTful prensiplerine sadık kalınmıştır.

Sonuç olarak; teorik veritabanı dersi kazanımları (Transaction, Normalizasyon, İndeksleme), gerçek hayat senaryosuna başarıyla uygulanmış ve ölçeklenebilir bir "Havayolu Rezervasyon Sistemi" ortaya çıkarılmıştır.

8. Referanslar

1. **Spring Boot Documentation (v3.3):** *Data Access & Transaction Management*. [https://docs.spring.io/spring-boot/docs/current/reference/html/data.html]
2. **MySQL 8.0 Reference Manual:** *InnoDB Locking and Transaction Model*. [https://dev.mysql.com/doc/refman/8.0/en/innodb-locking-transaction-model.html]
3. **Elmasri, R., & Navathe, S. B. (2016).** *Fundamentals of Database Systems (7th Edition)*. Pearson.
4. **Hibernate ORM Documentation:** *Optimistic Locking & Versioning*. [https://hibernate.org/orm/documentation/5.6/]