



KOCAELİ ÜNİVERSİTESİ

Mühendislik Fakültesi  
Yazılım Mühendisliği Bölümü

## Programlama Laboratuvarı III

### PROJE RAPORU

Uçak Bileti Rezervasyon Sistemi

**Ad - Soyad:** Sena Nur Dülger      **No:** 220229053

**Ad - Soyad:** Serhat Can Bakır      **No:** 220229036

**Tarih:** 2 Aralık 2025

# 1. Proje Tanıtımı ve Amacı

## 1.1. Proje Tanıtımı

Bu projenin konusu, Uçak Bileti Rezervasyon Sistemi'dir. Proje, tamamlandığında kullanıcıların uçak yolculukları için ihtiyaç duyduğu temel rezervasyon işlemlerini gerçekleştirebileceği, tam işlevsel bir yazılım platformu olmalıdır.

Uçuş ve bilet rezervasyonu süreci, gerçek hayatı sıkılıkla karşılaşılan, dinamik veri akışlarının ve kısıtların bulunduğu, birçok nesnenin birbiriyle etkileşim içinde çalıştığı karmaşık bir yapıyı temsil etmektedir. Bu bağlamda geliştirilecek sistem, hem son kullanıcıya (müsteriye) yönelik temel işlevleri (uçuş arama, koltuk seçimi, rezervasyon) hem de sistem yöneticisine (admin) yönelik yönetimsel işlemleri (uçuş ekleme, güncelleme) kapsamalıdır.

Bu kompleks sistemin geliştirilmesi sürecinde, projenin teknik gereksinimi olarak C# programlama dili kullanılacaktır. C#'ın sağladığı kurumsal düzeydeki Nesne Yönelimli Programlama (OOP) olanakları, projenin mimarisini oluşturmada temel araç olacaktır.

## 1.2. Projenin Amacı

Projenin temel amacı, Nesne Yönelimli Programlama (OOP) prensiplerinin gerçek bir senaryo olan uçak bilet rezervasyon sistemi üzerinden uygulanması olarak öğrenilmesi ve pekiştirilmesidir. Bu amaç doğrultusunda, projenin teknik gereksinimlerine uygun olarak, güçlü ve modern bir dil olan C# programlama dili kullanılmıştır.

Bu temel amacın yanı sıra, projenin hedefleri şunlardır:

- **OOP Prensiplerini Uygulamak:** Abstraction (Soyutlama), Encapsulation (Kapsülleme), Inheritance (Kalıtım) ve Polymorphism (Çok Biçimlilik) gibi temel OOP ilkelerinin projede aktif olarak kullanılması ve bu sayede modüler, okunabilir bir kod yapısı oluşturulması.
- **Tam İşlevsel Sistem Geliştirmek:** Kullanıcının uçuş arama, koltuk seçimi, rezervasyon oluşturma ve iptal etme yeteneklerine sahip olduğu, uçtan uca çalışır bir platform oluşturmak.
- **Kullanıcı Etkileşimi Sağlamak:** Bir GUI (Grafiksel Kullanıcı Arayüzü) üzerinden kullanıcı etkileşimi sağlamak ve kullanıcı (müsteri/admin) giriş ekranları ile sistem güvenliğini sağlamak.
- **Gerçekçi Kısıtları Modellemek:** Doluluk, uçuşa kalan gün, koltuk tipi ve promosyon gibi faktörleri hesaba katan Dinamik Fiyatlandırma mekanizmasını hayata geçirmek.
- **Veri Yönetimi Becerilerini Geliştirmek:** Tüm uçuş, yolcu ve rezervasyon bilgilerini kalıcı olarak dosyada veya veritabanında saklama yeteneği eklemek.

## **2. Sistem Analizi ve Gereksinimler**

Bu bölümde, Uçak Bileti Rezervasyon Sistemi'nin analiz süreci, temel nesne modeli, kullanıcı yetkilendirme yapısı ile birlikte işlevsel ve işlevsel olmayan gereksinimler detaylı bir şekilde ele alınmaktadır. Analiz aşaması; gerçek hayatı karşılaşan uçuş-rezervasyon sürecinin yazılımsal bir sisteme dönüştürülmesi, gereksinimlerin belirlenmesi ve mimari çerçeveyin hazırlanması amacıyla gerçekleştirilmiştir.

### **2.1. Temel Nesne Modeli ve Sınıf Yapısı**

Sistem, Nesne Yönelimli Programlama (OOP) prensiplerine göre modellenmiştir. Tüm kullanıcı rolleri ve sistem bileşenleri, uygun sorumluluklara sahip sınıflar halinde yapılandırılmıştır.

#### **2.1.1. Kullanıcı (Soyut Sınıf)**

Sisteme erişen tüm aktörlerin ortak davranışlarını tanımlayan soyut bir yapıdır. Abstraction prensibi doğrultusunda tasarlanmış olup aşağıdaki temel metodları içerir:

- Giriş Yap()
- Çıkış Yap()

#### **2.1.2. Müşteri (Concrete Sınıf)**

Kullanıcı sınıfından türetilmiş olup yolcu kimliğini temsil eder. Rezervasyon yapma, koltuk seçme ve rezervasyon iptal etme gibi operasyonları içerir.

#### **2.1.3. Admin (Concrete Sınıf)**

Kullanıcı sınıfından türetilmiş yönetici rolüdür. Uçuş, uçak ve havaalanı işlemlerini gerçekleştirebilir. Kalıtım ve çok biçimlilik ilkeleri etkin şekilde uygulanmıştır.

#### **2.1.4. Uçak (Sınıf)**

Uçağın fiziksel kapasitesi, mevcut boş koltuk sayısı ve koltuk konfigürasyonu gibi bilgiler bu sınıfta tutulur.

#### **2.1.5. Uçuş (Sınıf)**

Gerçekleştirilecek yolculuğa ait bilgiler (uçuş numarası, kalkış-varış yeri, tarih, saat) ile ilgili uçak nesnesi bu sınıfta yer alır.

#### **2.1.6. Rezervasyon (Sınıf)**

Bir müşterinin belirli bir uçuş için yaptığı satın alma işlemine ilişkin bilgileri içerir:

- Müşteri Kimliği
- Koltuk Numarası
- Fiyat Bilgisi
- Rezervasyon Durumu

## **2.2. Kullanıcı Rollerleri ve Yetki Kapsamı**

Sistem, rol tabanlı erişim yönetimi ile tasarlanmıştır. Kullanıcı türüne göre erişilebilin ekranlar ve işlem yetkileri aşağıdaki tabloda gösterilmektedir:

Rol	Sorumluluk	Admin Yetkileri / Müşteri Yetkilileri
Admin	Sistem envanter yönetimi	Uçuş, Uçak ve Havaalanı ekleme / silme / düzenleme; rezervasyon iptali
Müşteri	Uçuş arama ve rezervasyon işlemleri	Kayıt olma, uçuş arama, koltuk seçimi, rezervasyon yapma/iptal, check-in

## **2.3. İşlevsel Gereksinimler (Functional Requirements)**

Sistemin kullanıcıya sunduğu tüm işlevleri tanımlayan gereksinimler aşağıda belirtilmiştir.

### **2.3.1. Ortak Hesap Yönetimi**

- **1-** Kullanıcıların kimlik doğrulama ile sisteme giriş yapabilmesi.
- **2-** Güvenli oturum yönetimi uygulanması.
- **3-** Profil bilgilerinin görüntülenebilmesi.

### **2.3.2. Yönetim (Admin)**

- **4-** Uçak ekleme, düzenleme ve silme.
- **5-** Uçuş bilgilerini yönetme.
- **6-** Havaalanı bilgilerini yönetme.
- **7-** Uçaklara koltuk konfigürasyonu ekleme.
- **8-** Her rezervasyon üzerinde iptal yetkisi.

### **2.3.3. Müşteri İşlevleri**

- **9-** Yeni kullanıcı kaydı oluşturma.
- **10-** Rota ve tarihe göre uçuş arama.
- **11-** Müsait koltuk seçimi.
- **12-** Rezervasyon yapma.
- **13-** Rezervasyon görüntüleme.
- **14-** Rezervasyon iptali.
- **15-** Check-in işlemi.
- **16-** Doluluk, sezon, koltuk tipi ve zaman gibi faktörlere göre dinamik fiyatlandırma.

## 2.4. İşlevsel Olmayan Gereksinimler (Non-functional Requirements)

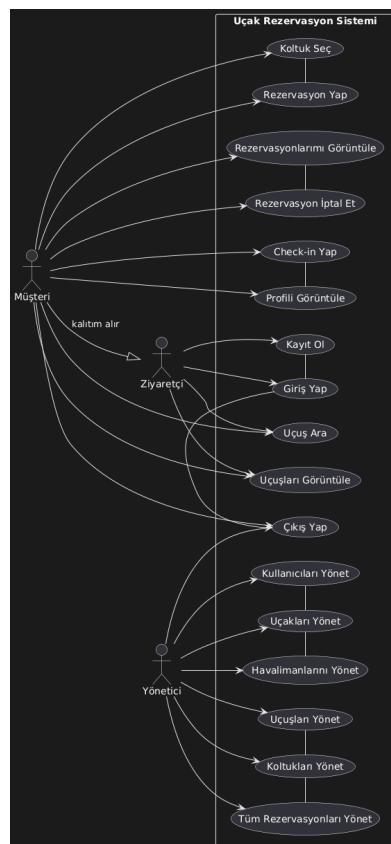
- 1- Sistem C# dili ile geliştirilmeli, OOP ilkeleri (Encapsulation, Inheritance, Polymorphism, Abstraction) uygulanmalıdır.
- 2- Uçuş, kullanıcı ve rezervasyon verileri kalıcı olarak dosya veya veritabanında saklanmalıdır.
- 3- Kullanıcı dostu bir GUI arayüzü bulunmalıdır.
- 4- Hatalı işlemler için uygun hata mesajları sağlanmalı, sistem kararlı çalışmalıdır.

## 3. UML Diyagramları

UML Diyagramları:

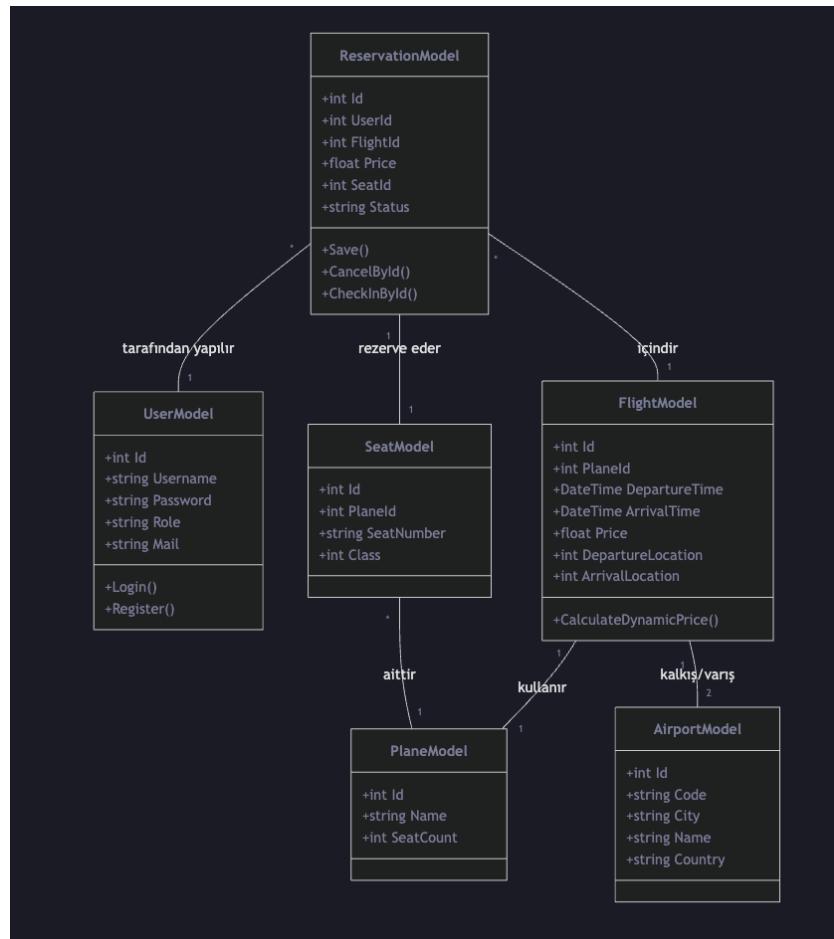
- Kullanım Senaryosu (Use Case Diagram)
- Sınıf Diyagramı (Class Diagram)
- Akış Diyagramları (Activity / Sequence Diagram)

### 3.1. Kullanım Senaryosu (Use Case Diagram)



Şekil 1: Uçak Bileti Rezervasyon Sistemi Use Case Diyagramı

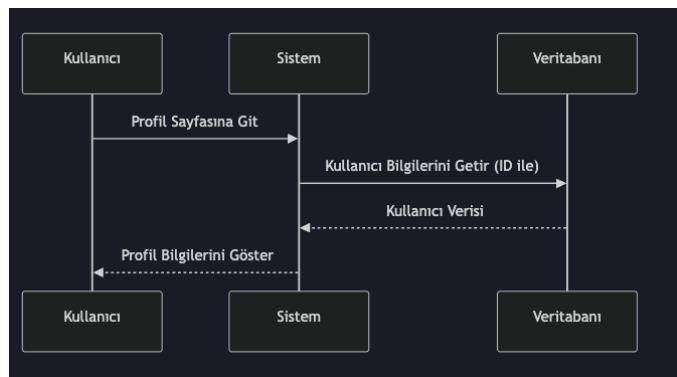
### 3.2. Sınıf Diyagramı (Class Diagram)



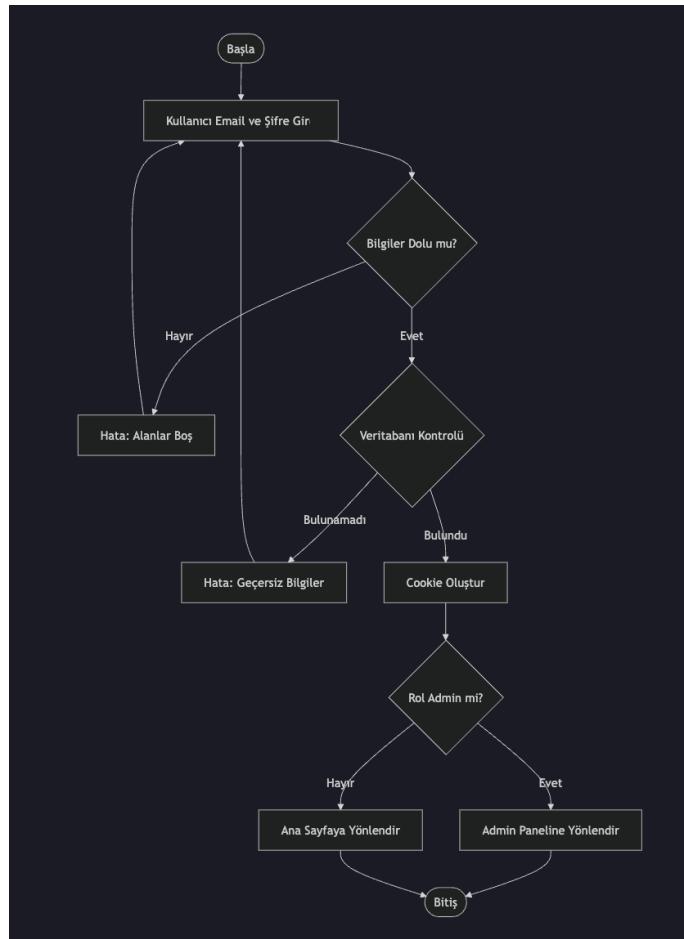
Şekil 2: Uçak Bileti Rezervasyon Sistemi Sınıf Diyagramı

### 3.3. Akış Diyagramı (Activity / Sequence Diagram)

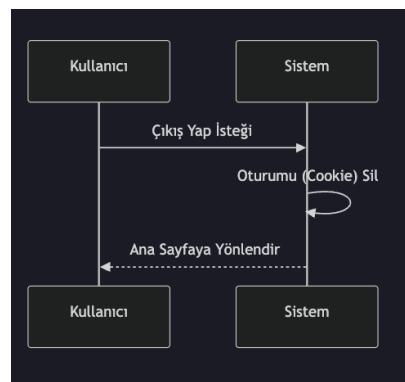
#### 3.3.1. Ortak Akış Diyagramları



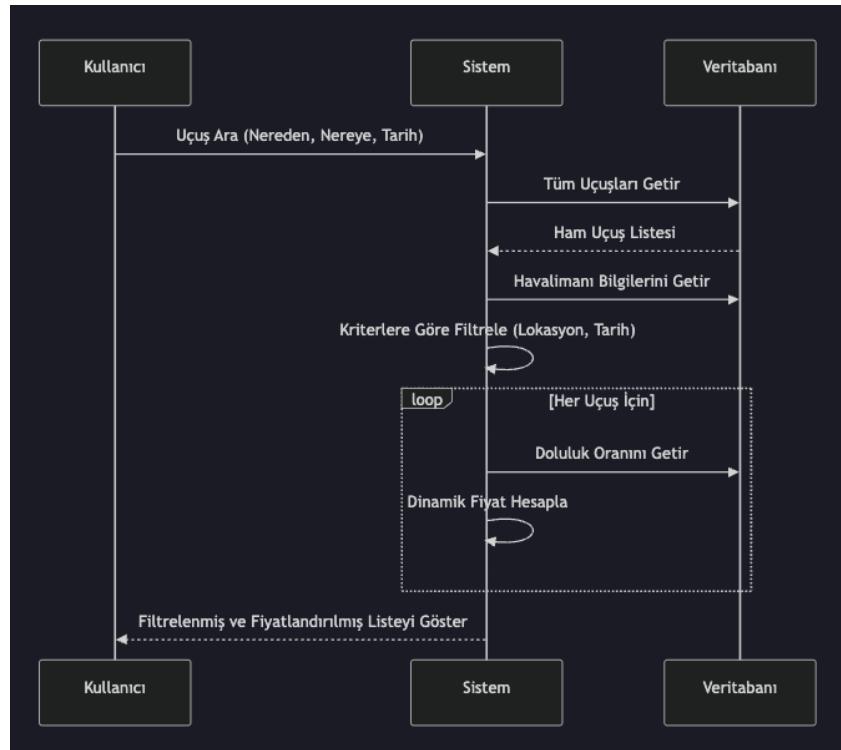
Şekil 3: Ortak Akış Diyagramı – Profil Görüntüleme



Şekil 4: Ortak Akış Diyagramı – Giriş Yap

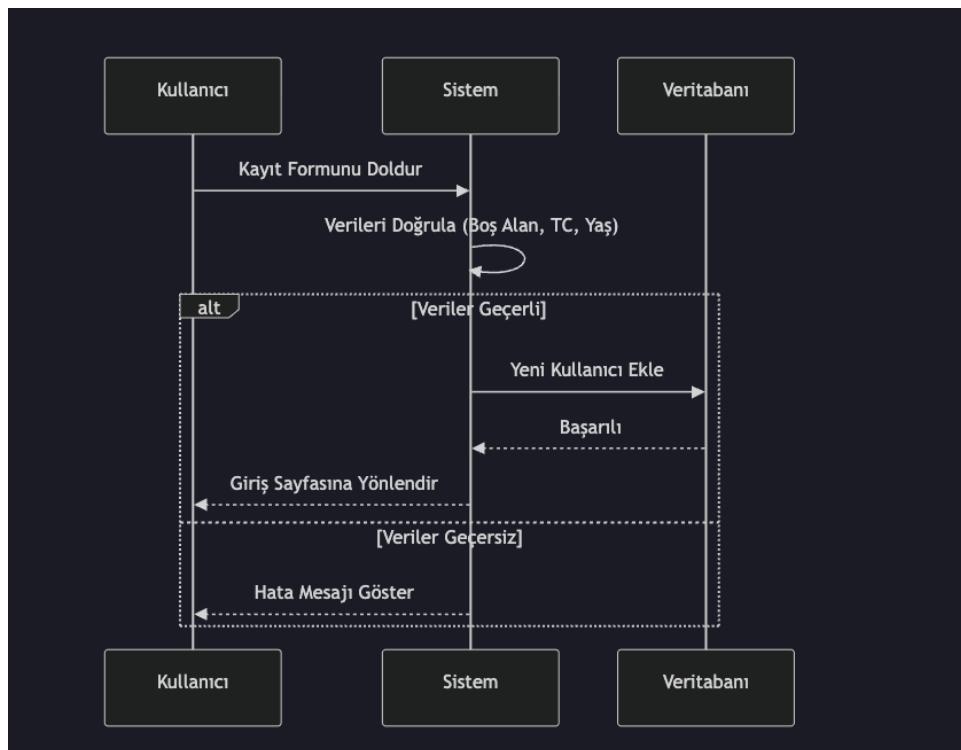


Şekil 5: Ortak Akış Diyagramı – Çıkış Yap

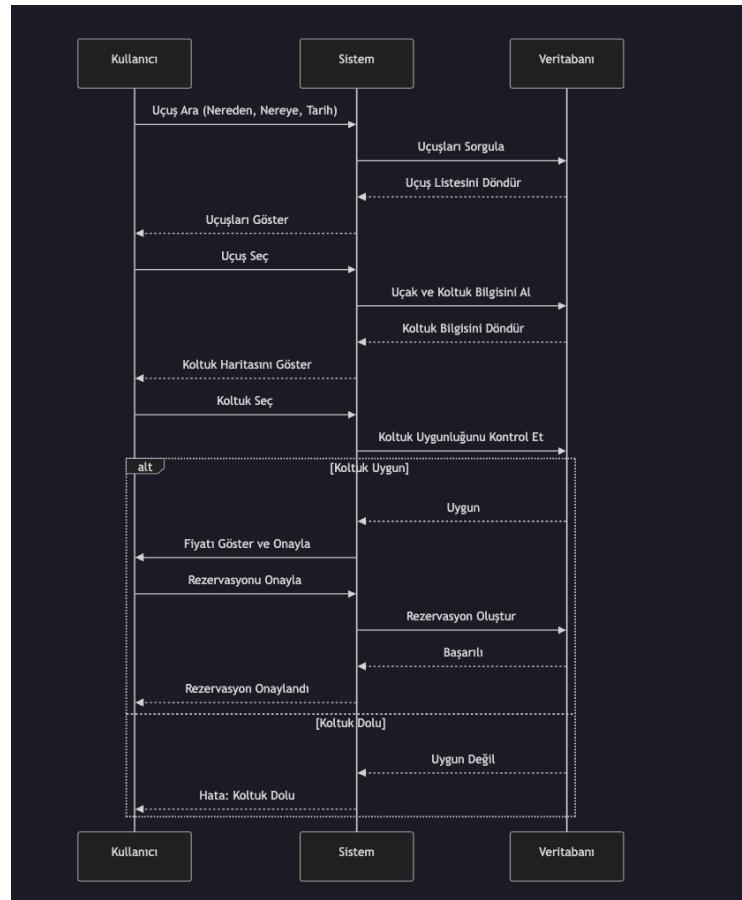


Şekil 6: Ortak Akış Diyagramı – Uçuş Arama

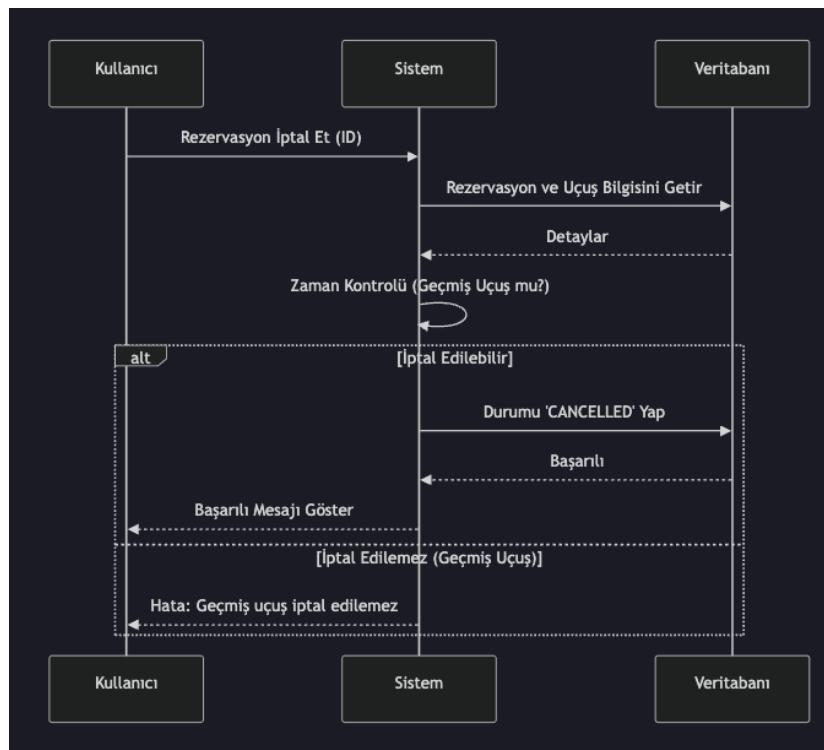
### 3.3.2. Müşteri Akış Diyagramları



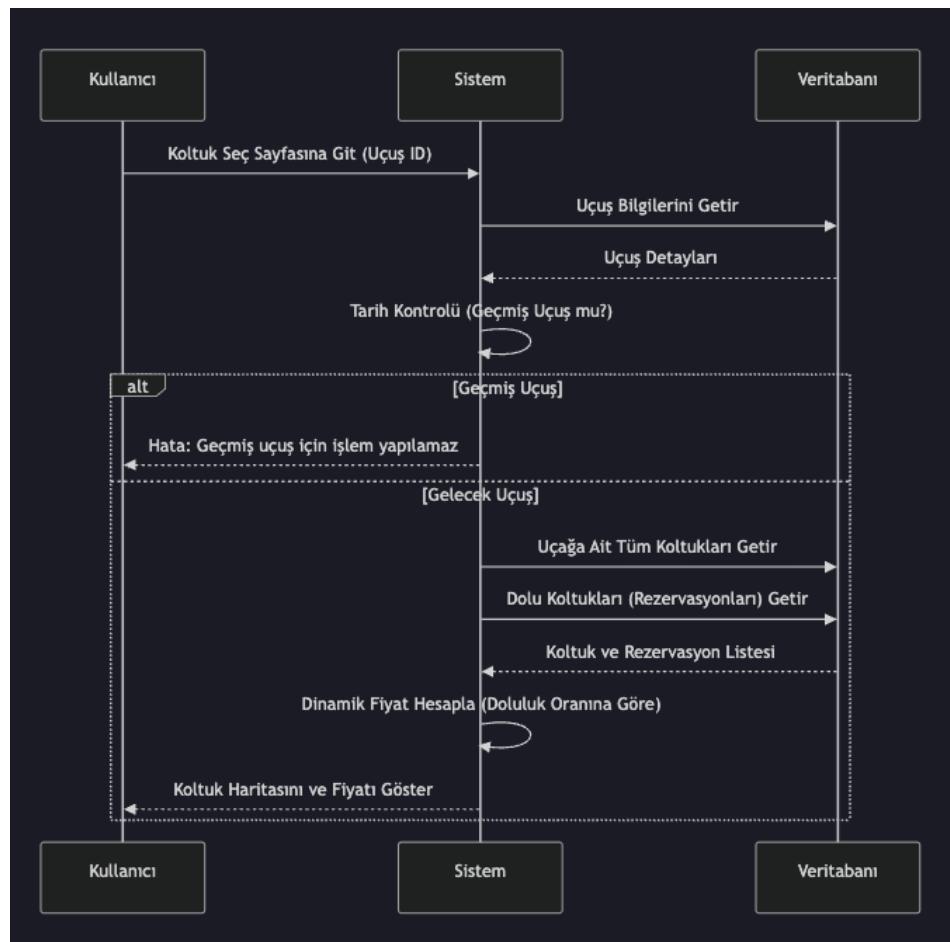
Şekil 7: Müşteri Akış Diyagramı – Kayıt Ol



Şekil 8: Müşteri Akış Diyagramı – Rezervasyon Yapma



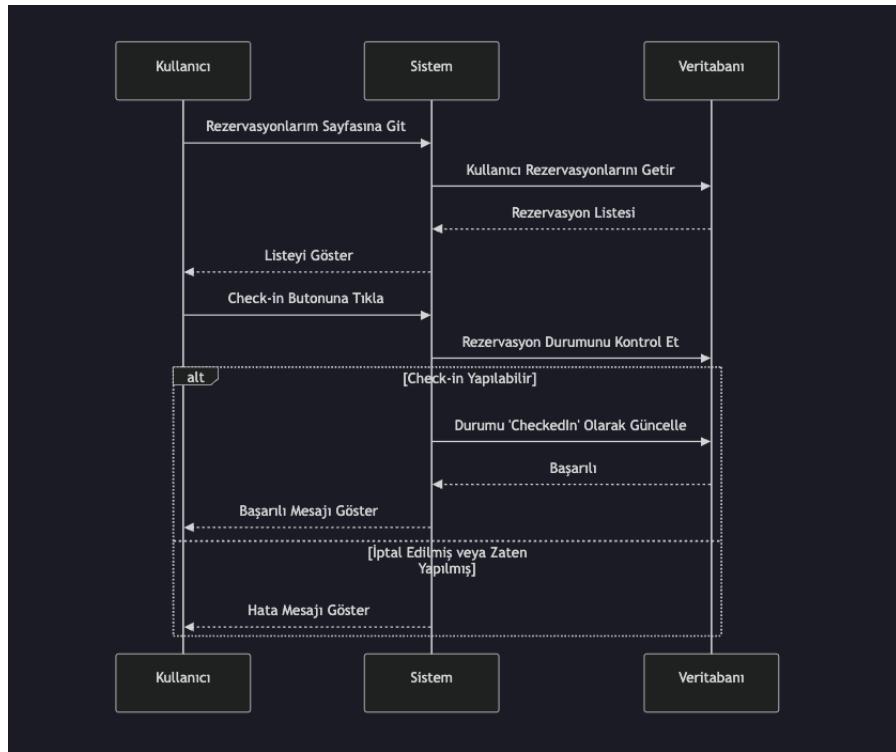
Şekil 9: Müşteri Akış Diyagramı – Rezervasyon İptal Etme



Şekil 10: Müşteri Akış Diyagramı – Koltuk Seçimi

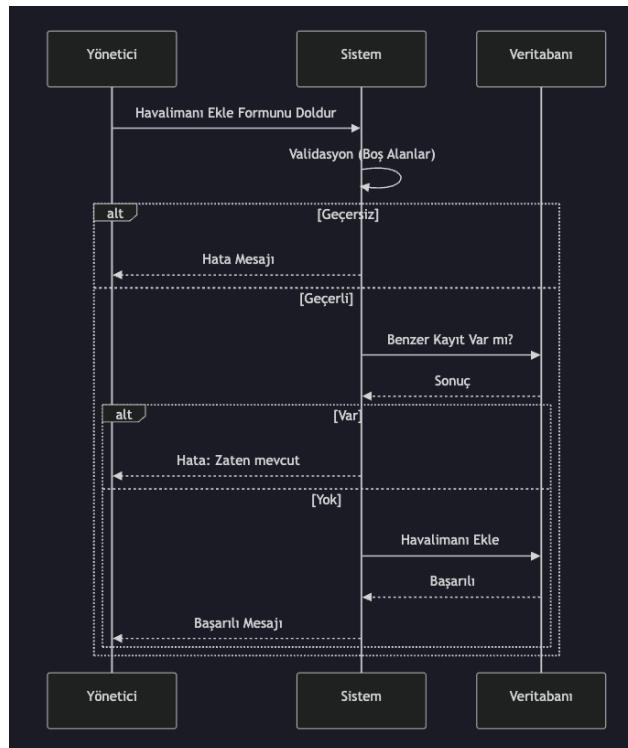


Şekil 11: Müşteri Akış Diyagramı – Rezervasyonlarının Görüntülenmesi

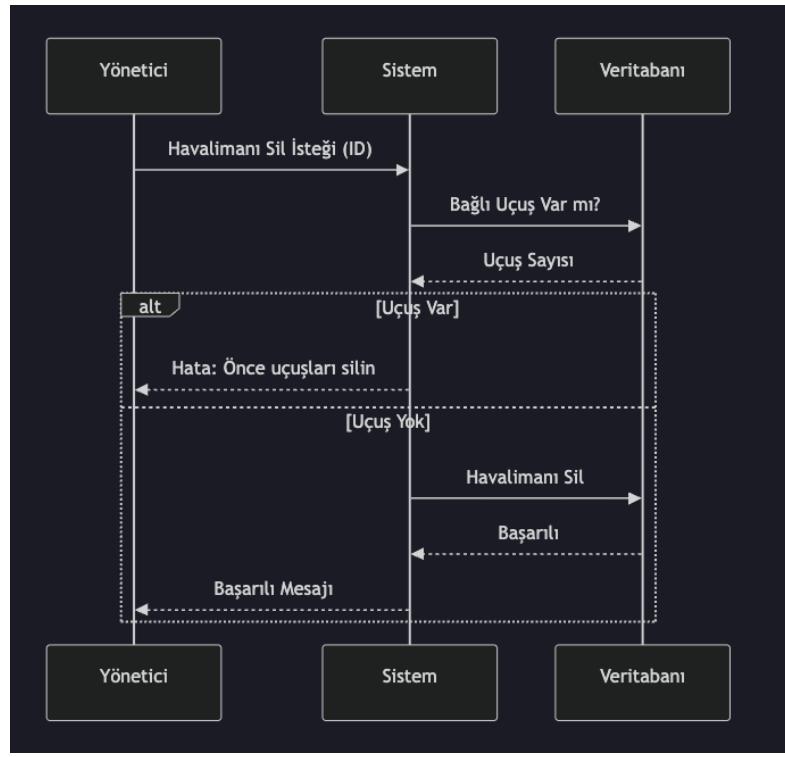


Şekil 12: Müşteri Akış Diyagramı – Check-in İşlemi

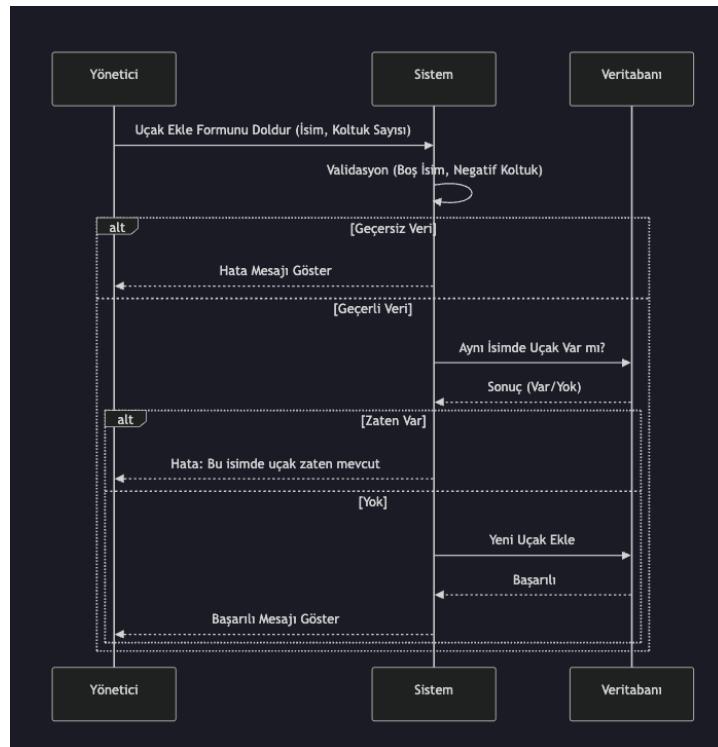
### 3.3.3. Admin Akış Diyagramları



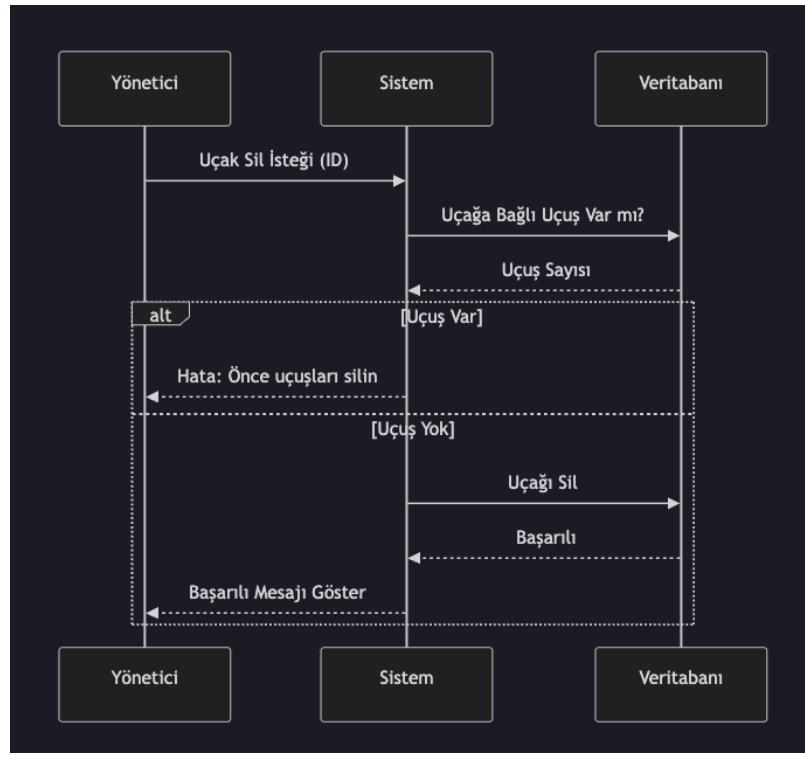
Şekil 13: Admin Akış Diyagramı – Havaalanı Ekleme



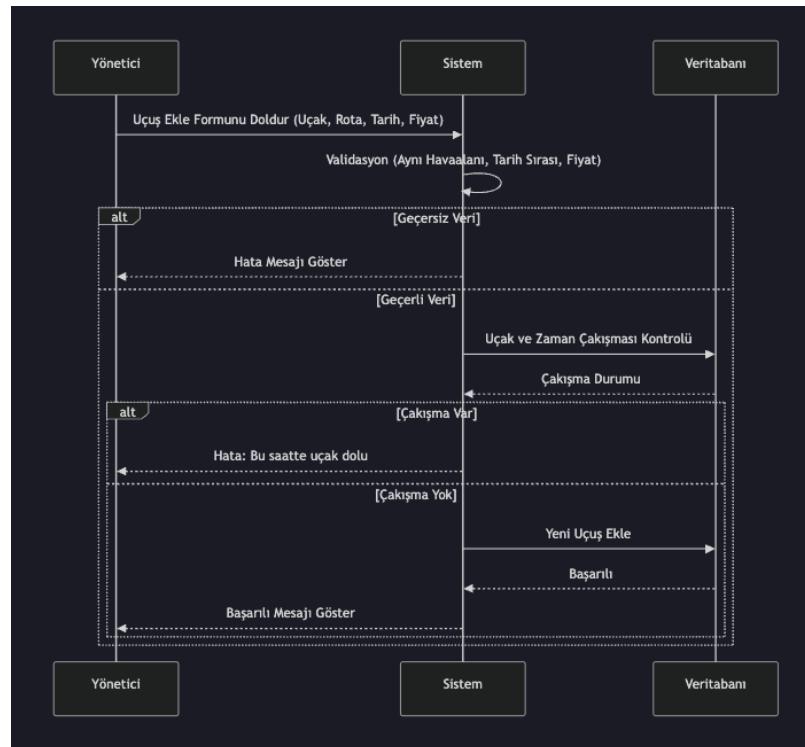
Şekil 14: Admin Akış Diyagramı – Havaalanı Silme



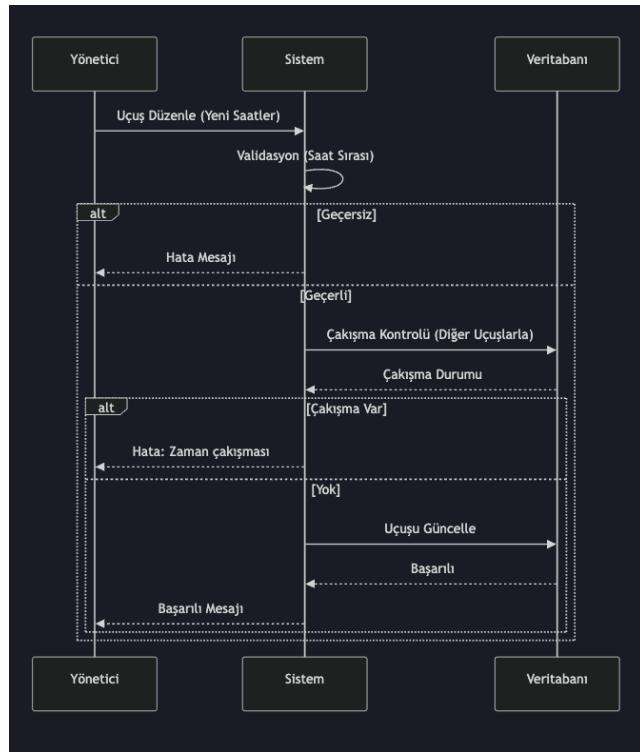
Şekil 15: Admin Akış Diyagramı – Uçak Ekleme



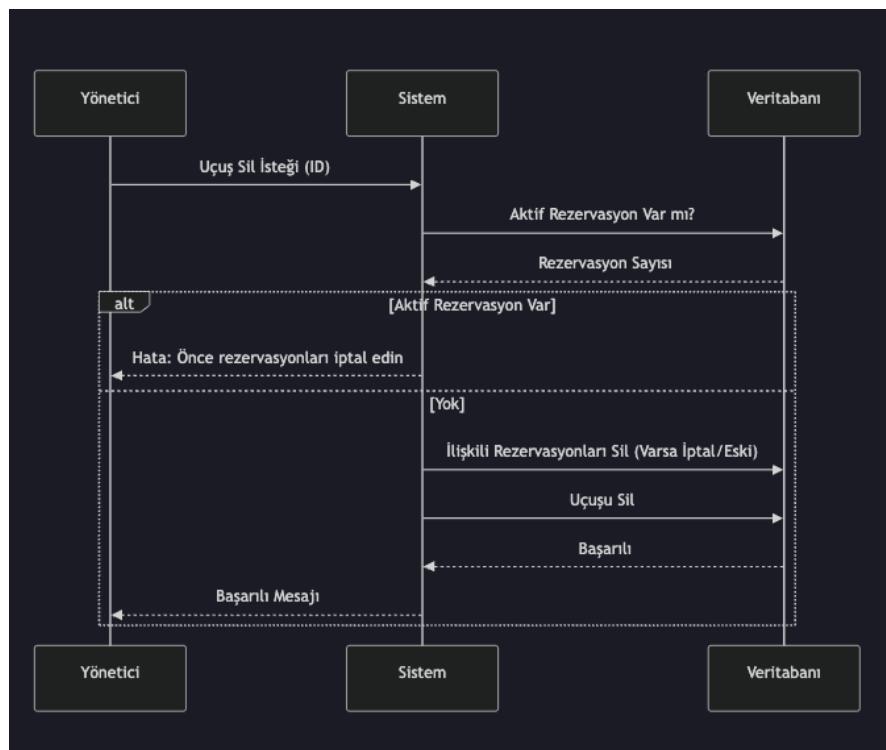
Şekil 16: Admin Akış Diyagramı – Uçak Silme



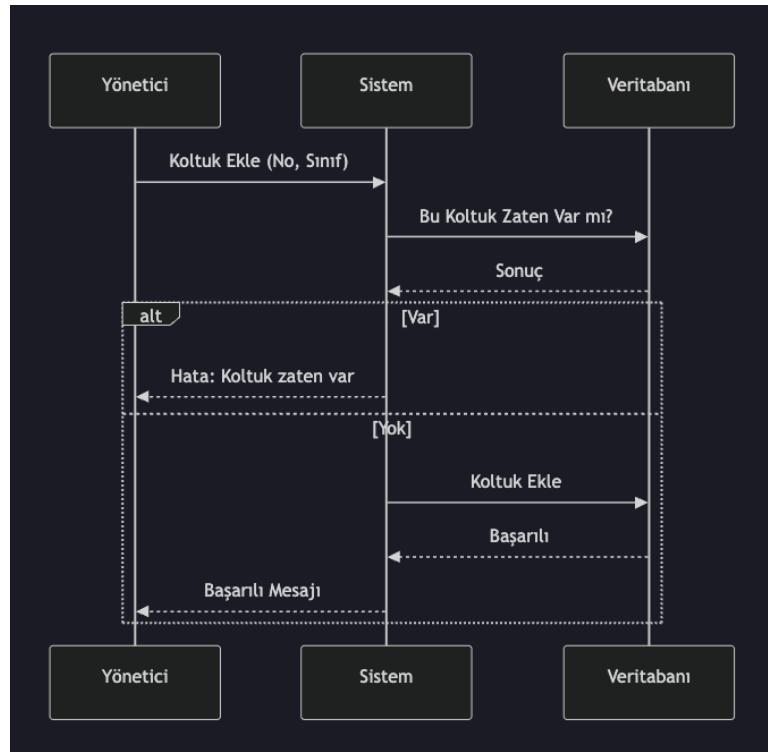
Şekil 17: Admin Akış Diyagramı – Uçuş Ekleme



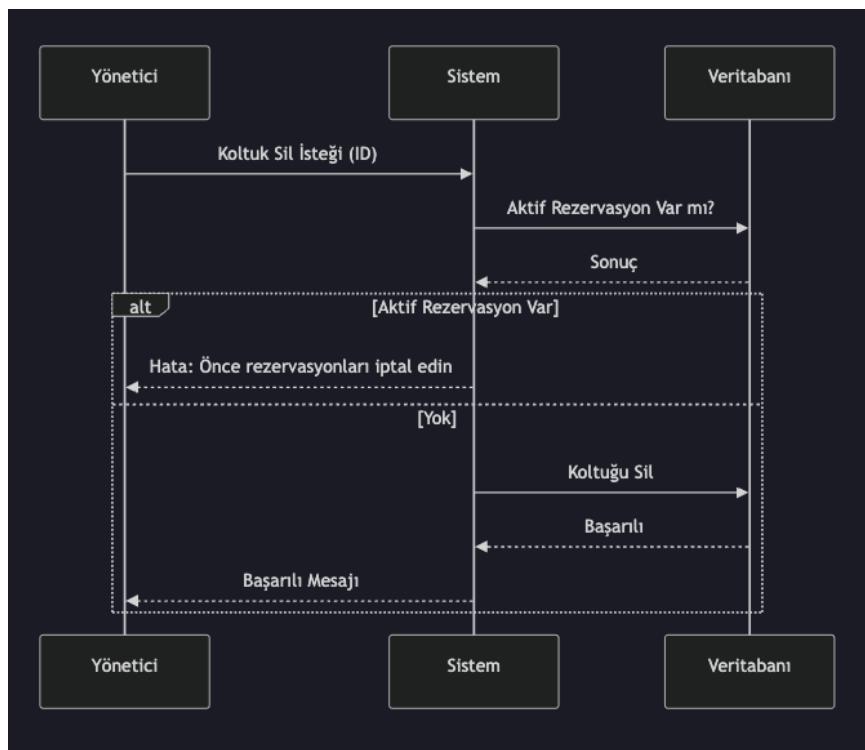
Şekil 18: Admin Akış Diyagramı – Uçuş Düzenleme



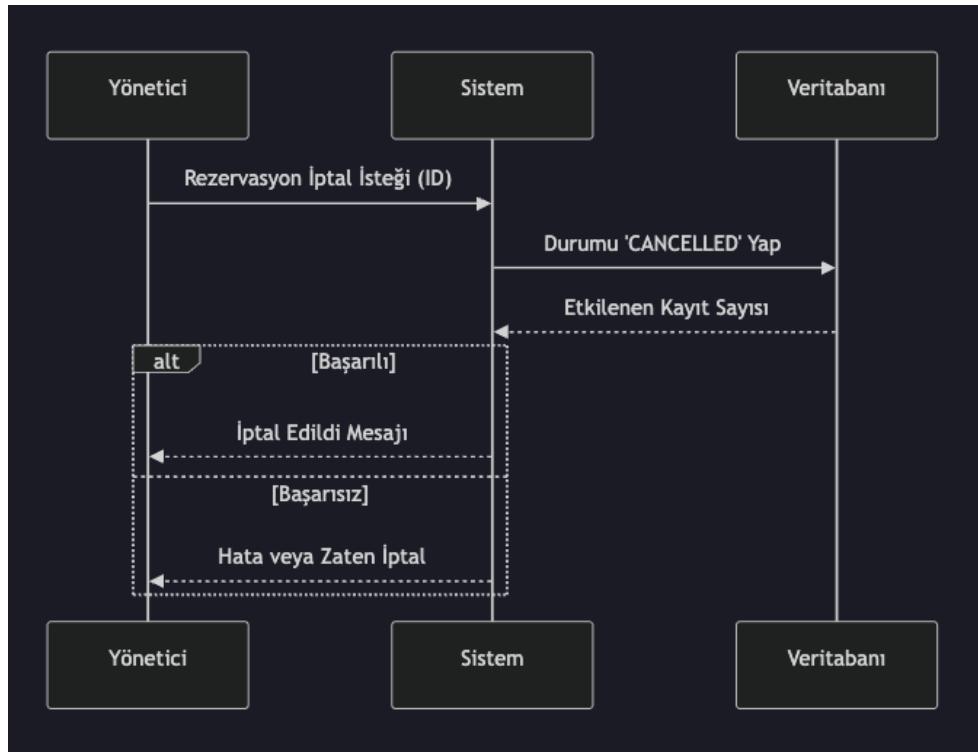
Şekil 19: Admin Akış Diyagramı – Uçuş Silme



Şekil 20: Admin Akış Diyagramı – Koltuk Ekleme



Şekil 21: Admin Akış Diyagramı – Koltuk Silme



Şekil 22: Admin Akış Diyagramı – Rezervasyon İptali

## 4. Sınıf Tanımları ve Kod Açıklamaları

Bu kısım, Uçak Rezervasyon Sistemi projesinde kullanılan temel sınıfların, veri modellerinin ve kontrolcülerin görevlerini ve işleyişlerini açıklamaktadır.

### 4.1. Veri Modelleri (Models)

Modeller, veritabanı tablolarının nesne tabanlı karşılıklarıdır ve iş mantığını kapsarlar.

#### 4.1.1. UserModel (Kullanıcı Modeli)

Bu sınıf, sistemdeki tüm kullanıcıların (Yönetici ve Müşteri) ortak özelliklerini ve metodlarını barındıran temel (*abstract*) sınıfıdır.

##### 1. Sınıf Tanımı ve Özellikler

Kullanıcıların kimlik, isim, e-posta, rol gibi temel bilgileri burada tanımlanır. `BaseModel` sınıfından türetilmiştir.

```

using System;

namespace prgmlab3.Models
{
    public abstract class UserModel : BaseModel
    {
        public const int RoleCustomer = 0;
        public const int RoleAdmin = 1;

        protected int _id;
        protected string _username = "";
        protected string _mail = "";
        protected string _password = "";
        protected int _role;
        protected string _tcNo = "";
        protected string _firstName = "";
        protected string _lastName = "";
        protected DateTime _birthDate = DateTime.MinValue;
        protected string _phone = "";

        // ----- Public Properties -----
        public int Id => _id;
        public string Username => _username;
        public string Mail => _mail;
        public int Role => _role;
        public string TcNo => _tcNo;
        public string FirstName => _firstName;
        public string LastName => _lastName;
        public DateTime BirthDate => _birthDate;
        public string Phone => _phone;
    }
}

```

Şekil 23: UserModel.cs sınıfına ait kod görünümü

## 2. Giriş Yapma (Login)

Bu metot, kullanıcının e-posta ve şifresini kontrol ederek sisteme giriş yapmasını sağlar. Önce alanların boş olup olmadığı denetlenir, ardından veritabanında sorgu çalıştırılır; geçerli kayıt bulunursa MapRowToUser ile kullanıcı nesnesi döndürülür, aksi halde null dönülür.

```

// ----- LOGIN -----
public static UserModel? Login(string mail, string password)
{
    if (string.IsNullOrWhiteSpace(mail) ||
    string.IsNullOrWhiteSpace(password))
        return null;

    mail = mail.Trim();

    var res = Query(
        "SELECT * FROM users WHERE mail=@mail AND
password=@pass",
        cmd =>
    {
        cmd.Parameters.AddWithValue("@mail", mail);
        cmd.Parameters.AddWithValue("@pass", password);
    });
    if (res.Count == 0)
        return null;

    var row = res[0];
    return MapRowToUser(row);
}

```

Şekil 24: Kullanıcı girişini gerçekleştiren Login metodu

## 3. Kayıt Olma (Register)

Bu metot, sisteme yeni müşteri eklemek için kullanılır. Form verileri doğrulanır, aynı

e-posta veya TC numarasına sahip kullanıcı olup olmadığı kontrol edilir; tüm kontroller başarılıysa yeni kayıt oluşturulur, aksi durumda uygun hata mesajı üretilir.

```

public static bool Register(
    string tcNo,
    string firstName,
    string lastName,
    DateTime birthDate,
    string mail,
    string phone,
    string password)
{
    // Temel boşluk kontrolleri
    if (string.IsNullOrWhiteSpace(tcNo) ||
        string.IsNullOrWhiteSpace(firstName) ||
        string.IsNullOrWhiteSpace(lastName) ||
        string.IsNullOrWhiteSpace(mail) ||
        string.IsNullOrWhiteSpace(phone) ||
        string.IsNullOrWhiteSpace(password))
    {
        return false;
    }

    tcNo = tcNo.Trim();
    firstName = firstName.Trim();
    lastName = lastName.Trim();
    mail = mail.Trim();
    phone = phone.Trim();

    if (tcNo.Length != 11)
        return false;

    if (password.Length < 4)
        return false;

    // 18 yaş kontrolü
    var today = DateTime.Today;
    var age = today.Year - birthDate.Year;
    if (birthDate.Date > today.AddYears(-age)) age--;

    if (age < 18)
        return false;

    // Aynı mail veya TCKN kayıtlı mı?
    var existing = Query(
        "SELECT id FROM users WHERE mail=@mail OR tc_no=@tc",
        cmd =>
    {
        cmd.Parameters.AddWithValue("@mail", mail);
        cmd.Parameters.AddWithValue("@tc", tcNo);
    });

    if (existing.Count > 0)
        return false;

    // username için varsayılan: Ad + Soyad
    var username = $"{firstName} {lastName}".Trim();

    Execute(
        @"INSERT INTO users
            (tc_no, first_name, last_name, birth_date, phone,
            username, mail, password, role)
            VALUES (@tc, @fn, @ln, @bd, @ph, @un, @mail, @pass,
            @role)",
        cmd =>
    {
        cmd.Parameters.AddWithValue("@tc", tcNo);
        cmd.Parameters.AddWithValue("@fn", firstName);
        cmd.Parameters.AddWithValue("@ln", lastName);
        cmd.Parameters.AddWithValue("@bd", birthDate.ToString("yyyy-MM-dd"));
        cmd.Parameters.AddWithValue("@ph", phone);
        cmd.Parameters.AddWithValue("@un", username);
        cmd.Parameters.AddWithValue("@mail", mail);
        cmd.Parameters.AddWithValue("@pass", password);
        cmd.Parameters.AddWithValue("@role", RoleCustomer); // 0 = müşteri
    });
    return true;
}
}

```

Şekil 25: Yeni kullanıcı oluşturma işlemini yapan Register metodu

#### 4.1.2. CustomerModel (Müşteri Modeli)

Müşteriye özgü işlemleri barındıran sınıfı `UserModel` sınıfından miras alır. **1.**

##### **Yapıcı Metot (Constructor)**

Müşteri nesnesi oluşturulurken temel bilgileri (id, kullanıcı adı, e-posta) set eder ve rol bilgisini 0 (Müşteri) olarak tanımlar.

```

namespace prgmlab3.Models
{
    public class CustomerModel : UserModel
    {
        public CustomerModel(int id, string username, string mail)
        {
            _id = id;
            _username = username;
            _mail = mail;
            _role = 0;
        }
    }
}

```

Şekil 26: CustomerModel sınıfında yapıcı metot (constructor)

## 2. Rezervasyon Oluşturma

Müşterinin seçtiği uçuş ve koltuk için rezervasyon kaydı oluşturur. Önce uçuş bilgisi `FlightModel.GetById` ile alınır; uçuş geçerliyse `ReservationModel` nesnesi yaratılır, veritabanına kaydedilir ve geriye döndürülür.

```

public ReservationModel? CreateReservation(int flightId, int seatId)
{
    var flight = FlightModel.GetById(flightId);
    if (flight == null) return null;

    ReservationModel reservation = new ReservationModel(_id,
    flightId, (float)flight.Price, seatId);
    reservation.Save();
    return reservation;
}
}

```

Şekil 27: CustomerModel içinde rezervasyon oluşturma metodu

### 4.1.3. AdminModel (Yönetici Modeli)

Yöneticiye özgü işlemleri (uçak ve uçuş ekleme/güncelleme/silme, rezervasyon iptali vb.) barındıran sınıfıtır ve `UserModel` sınıfından miras alır. **1. Yapıcı Metot (Constructor)**

Admin nesnesi oluşturulurken temel kullanıcı bilgilerini set eder ve rol alanını 1 (Admin) olarak işaretler.

```

using System;

namespace prgmlab3.Models
{
    // Sistemde yönetimsel işlemleri yapan kullanıcı modelidir.
    // Kullanıcı hiyerarşisinde UserModel'den kalıtım alır.

    public class AdminModel : UserModel
    {
        public AdminModel(int id, string username, string mail)
        {
            _id = id;
            _username = username;
            _mail = mail;
            _role = 1;
        }
    }
}

```

Şekil 28: AdminModel sınıfında yapıcı metot

## 2. Uçak Yönetimi (Ekleme / Güncelleme / Silme)

Yönetici; yeni uçak ekleme, mevcut uçağın adını ve koltuk sayısını güncelleme veya

sistemi temiz tutmak için uçağı silme işlemlerini bu metodlar üzerinden gerçekleştirir. Girdi doğrulamaları (boş ad, geçersiz koltuk sayısı, hatalı ID vb.) yapılır.

```
#region Uçak İşlemleri
    // Sisteme yeni uçak ekler.
    public void AddPlane(string name, int seatCount)
    {
        if (string.IsNullOrWhiteSpace(name))
            throw new ArgumentException("Uçak adı boş olamaz.", nameof(name));

        if (seatCount <= 0)
            throw new ArgumentOutOfRangeException(nameof(seatCount), "Koltuk sayısı 0'dan büyük olmalıdır.");

        Execute("INSERT INTO planes (name, seat_count) VALUES (@n, @s)", cmd =>
        {
            cmd.Parameters.AddWithValue("@n", name.Trim());
            cmd.Parameters.AddWithValue("@s", seatCount);
        });
    }

    // Var olan bir uçağın bilgilerini günceller.
    public void UpdatePlane(int planeId, string name, int seatCount)
    {
        if (planeId <= 0)
            throw new ArgumentOutOfRangeException(nameof(planeId), "Geçersiz uçak ID.");

        if (string.IsNullOrWhiteSpace(name))
            throw new ArgumentException("Uçak adı boş olamaz.", nameof(name));

        if (seatCount <= 0)
            throw new ArgumentOutOfRangeException(nameof(seatCount), "Koltuk sayısı 0'dan büyük olmalıdır.");

        Execute("UPDATE planes SET name=@n, seat_count=@s WHERE id=@id", cmd =>
        {
            cmd.Parameters.AddWithValue("@id", planeId);
            cmd.Parameters.AddWithValue("@n", name.Trim());
            cmd.Parameters.AddWithValue("@s", seatCount);
        });
    }

    // Sistemdeki bir uçağı siler.
    public void RemovePlane(int planeId)
    {
        if (planeId <= 0)
            throw new ArgumentOutOfRangeException(nameof(planeId), "Geçersiz uçak ID.");

        Execute("DELETE FROM planes WHERE id=@id", cmd =>
        {
            cmd.Parameters.AddWithValue("@id", planeId);
        });
    }
#endregion
```

Şekil 29: AdminModel içinde uçak yönetimi metodları

### 3. Uçuş Yönetimi

Yönetici, belirli bir uçağa bağlı kalkış/varış noktaları ve saatleriyle birlikte yeni bir uçuş ekleyebilir, var olan bir uçuşun bilgilerini güncelleyebilir veya uçuşu silebilir. Zaman, fiyat ve lokasyon alanları için geçerlilik kontrolleri yapılır.

```

#region Uçuş İşlemleri
// Yeni uçuş ekler.
public void AddFlight(
    int planeId,
    int dep,
    int arr,
    DateTime depTime,
    DateTime arrTime,
    decimal basePrice)
{
    if (planeId <= 0)
        throw new
ArgumentOutOfRangeException(nameof(planeId), "Geçersiz uçak ID.");
    if (dep <= 0 || arr <= 0)
        throw new ArgumentOutOfRangeException(nameof(dep),
"Kalkış/varış lokasyonu geçersiz.");
    if (depTime >= arrTime)
        throw new ArgumentException("Kalkış saatı varış
saatinden sonra olamaz.");
    if (basePrice <= 0)
        throw new
ArgumentOutOfRangeException(nameof(basePrice), "Fiyat 0'dan büyük
olmalıdır.");
    Execute(@"
        INSERT INTO flights
        (plane_id, departure_location, arrival_location,
        departure_time, arrival_time, price)
        VALUES (@pid, @dep, @arr, @dtim, @atim, @price)",
cmd =>
{
    cmd.Parameters.AddWithValue("@pid", planeId);
    cmd.Parameters.AddWithValue("@dep", dep);
    cmd.Parameters.AddWithValue("@arr", arr);
    cmd.Parameters.AddWithValue("@dtim", depTime);
    cmd.Parameters.AddWithValue("@atim", arrTime);
    cmd.Parameters.AddWithValue("@price", basePrice);
});
}

// Var olan bir uçuşu günceller.
public void UpdateFlight(
    int flightId,
    int planeId,
    int dep,
    int arr,
    DateTime depTime,
    DateTime arrTime,
    decimal basePrice)
{
    if (flightId <= 0)
        throw new
ArgumentOutOfRangeException(nameof(flightId), "Geçersiz uçuş ID.");
    if (planeId <= 0)
        throw new
ArgumentOutOfRangeException(nameof(planeId), "Geçersiz uçak ID.");
    if (dep <= 0 || arr <= 0)
        throw new ArgumentOutOfRangeException(nameof(dep),
"Kalkış/varış lokasyonu geçersiz.");
    if (depTime >= arrTime)
        throw new ArgumentException("Kalkış saatı varış
saatinden sonra olamaz.");
    if (basePrice <= 0)
        throw new
ArgumentOutOfRangeException(nameof(basePrice), "Fiyat 0'dan büyük
olmalıdır.");
    Execute(@"
        UPDATE flights SET
        plane_id = @pid,
        departure_location = @dep,
        arrival_location = @arr,
        departure_time = @dtim,
        arrival_time = @atim,
        price = @price
        WHERE id = @fid", cmd =>
{
    cmd.Parameters.AddWithValue("@fid", flightId);
    cmd.Parameters.AddWithValue("@id", planeId);
    cmd.Parameters.AddWithValue("@dep", dep);
    cmd.Parameters.AddWithValue("@arr", arr);
    cmd.Parameters.AddWithValue("@dtim", depTime);
    cmd.Parameters.AddWithValue("@atim", arrTime);
    cmd.Parameters.AddWithValue("@price", basePrice);
});
}

// Uçuşu siler. Üzerinde rezervasyon varsa önce
rezervasyonları kontrol etmen önerilir.
public void RemoveFlight(int flightId)
{
    if (flightId <= 0)
        throw new
ArgumentOutOfRangeException(nameof(flightId), "Geçersiz uçuş ID.");
    Execute("DELETE FROM flights WHERE id=@id", cmd =>
{
    cmd.Parameters.AddWithValue("@id", flightId);
});
}

#endregion

```

Şekil 30: AdminModel içinde uçuş ekleme/güncelleme/silme metodları

## 4. Rezervasyon İptali (Admin Yetkisi)

Yönetici, sistemdeki herhangi bir rezervasyonu CANCELLED durumuna çekerek iptal edebilir. Geçersiz rezervasyon ID'si için koruma amaçlı kontrol yapılır.

```
#region Rezervasyon İşlemleri (Admin Tarafı)

    // Admin tarafından bir rezervasyonu iptal eder.
    public void CancelReservation(int reservationId)
    {
        if (reservationId <= 0)
            throw new
        ArgumentException(nameof(reservationId), "Geçersiz
        rezervasyon ID.");

        Execute("UPDATE reservations SET status = 'CANCELLED'
WHERE id=@id", cmd =>
{
    cmd.Parameters.AddWithValue("@id", reservationId);
});
    }

    #endregion
}
```

Şekil 31: AdminModel tarafından rezervasyon iptal etme metodu

### 4.1.4. FlightModel (Uçuş Modeli)

Uçuş bilgilerini tutan ve dinamik fiyatlandırma mantığını içeren model sınıfıdır.

#### 1. Özellikler (Properties)

Bir uçuşun temel bileşenlerini (uçak, kalkış/varış havaalanı, kalkış/varış zamanı ve baz fiyat) tanımlar. Gerekli alanlar için doğrulama (validation) öznitelikleri kullanılır.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using prgmlab3.data;

namespace prgmlab3.Models
{
    // Uçuş bilgilerini temsil eden model.
    public class FlightModel : BaseModel
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Uçak seçimi zorunludur.")]
        [Display(Name = "Uçak")]
        public int PlaneId { get; set; }

        [Required(ErrorMessage = "Kalkış havaalanı zorunludur.")]
        [Display(Name = "Kalkış Havaalanı")]
        public int DepartureLocation { get; set; }

        [Required(ErrorMessage = "Varış havaalanı zorunludur.")]
        [Display(Name = "Varış Havaalanı")]
        public int ArrivalLocation { get; set; }

        [Required(ErrorMessage = "Kalkış zamanı zorunludur.")]
        [Display(Name = "Kalkış Zamanı")]
        public DateTime DepartureTime { get; set; }

        [Required(ErrorMessage = "Varış zamanı zorunludur.")]
        [Display(Name = "Varış Zamanı")]
        public DateTime ArrivalTime { get; set; }

        // Baz fiyat
        [Required(ErrorMessage = "Baz fiyat zorunludur.")]
        [Range(0.01, double.MaxValue, ErrorMessage = "Baz fiyat 0'dan
        büyük olmalıdır.")]
        [Display(Name = "Baz Fiyat")]
        public double Price { get; set; }
    }
}
```

Şekil 32: FlightModel sınıfının temel özellikleri

## 2. Dinamik Fiyatlandırma Algoritması

Bilet fiyatını; uçuşa kalan gün, doluluk oranı, koltuk sınıfı ve sezon bilgisini birlikte değerlendirek hesaplar ve sonuç fiyatı iki basamaklı olarak döndürür.

```
#region Dinamik Fiyatlandırma - tek fonksiyon
    // Uçuşa kalan gün + doluluk + koltuk tipi + sezon
    // seatClass : 0 = Economy, 1 = Business
    // occupancyRatio : 0.0-1.0 arası doluluk oranı
    public double CalculateDynamicPrice(int seatClass, double
occupancyRatio, DateTime? nowOverride = null)
    {
        var now = nowOverride ?? DateTime.Now;

        // === 1) Zaman faktörü ===
        var timeToDeparture = DepartureTime - now;
        var days = timeToDeparture.TotalDays;

        double timeFactor;
        if (days < 0)
        {
            // Geçmiş uçuş, zaman faktörü 1 ve direkt baz fiyat
            timeFactor = 1.0;
        }
        else if (days >= 30)
        {
            // 30+ gün varsa %40 indirim
            timeFactor = 0.60;
        }
        else
        {
            // 0-30 gün arasında lineer olarak 0.6 ~ 1.0
            // days=30 => 0.6, days=0 => 1.0
            timeFactor = 1.0 - (days / 30.0) * 0.4;
        }

        // === 2) Doluluk faktörü ===
        // 0-30% : 0.9 (indirim)
        // 30-70% : 1.0 (normal)
        // 70-90% : 1.2 (artış)
        // 90-100% : 1.4 (yüksek artış)
        double occ = Math.Clamp(occupancyRatio, 0.0, 1.0);
        double occupancyFactor;
        if (occ < 0.30)
            occupancyFactor = 0.90;
        else if (occ < 0.70)
            occupancyFactor = 1.00;
        else if (occ < 0.90)
            occupancyFactor = 1.20;
        else
            occupancyFactor = 1.40;

        // === 3) Koltuk tipi faktörü ===
        // Economy : 1.0
        // Business: 1.5
        double seatFactor = seatClass == 1 ? 1.50 : 1.00;

        // === 4) Sezon faktörü (ay bazlı) ===
        // Yüksek sezon: Haziran-Eylül (6-9)      => 1.20
        // Orta sezon : Mayıs, Ekim, Aralık (5,10,12)=> 1.05
        // Düşük sezon : diğer aylar                => 0.95
        int m = now.Month;
        double seasonFactor;
        if (m >= 6 && m <= 9)
        {
            seasonFactor = 1.20;
        }
        else if (m == 5 || m == 10 || m == 12)
        {
            seasonFactor = 1.05;
        }
        else
        {
            seasonFactor = 0.95;
        }

        double finalFactor = timeFactor * occupancyFactor *
seatFactor * seasonFactor;
        double result = Price * finalFactor;

        return Math.Round(result, 2,
MidpointRounding.AwayFromZero);
    }

#endregion
```

Şekil 33: FlightModel içinde dinamik fiyat hesaplayan CalculateDynamicPrice metodu

#### 4. Arama Fonksiyonu

Kalkış ve varış havaalanı adlarına göre uçuşları filtreleyerek ilgili kayıtları döndürür. Havaalanı isimleri üzerinden JOIN ile sorgu yapar.

```

#region Arama
    public static List<Dictionary<string, object>>
SearchFlights(string departureLocationName, string
arrivalLocationName)
{
    {
        return Query<Flight>(
            "SELECT
                f.id,
                a1.name AS departure_name,
                a2.name AS arrival_name,
                f.plane_id,
                f.departure_time,
                f.arrival_time,
                f.price
            FROM flights f
            JOIN airports a1 ON f.departure_location = a1.id
            JOIN airports a2 ON f.arrival_location = a2.id
            WHERE a1.name = @departure_name
                AND a2.name = @arrival_name",
            cmd =>
            {
                cmd.Parameters.AddWithValue("@departure_name",
                departureLocationName);
                cmd.Parameters.AddWithValue("@arrival_name",
                arrivalLocationName);
            });
    }
    #endregion
}

```

Şekil 34: Kalkış ve varış yerine uçuş arayan SearchFlights metodu

#### 4.1.5. ReservationModel (Rezervasyon Modeli)

Rezervasyon kaydını temsil eden model sınıfıdır.

##### 1. Özellikler ve Durumlar

Rezervasyonun hangi kullanıcıya ve hangi uçuşa ait olduğu, koltuk numarası, ödenen fiyat ve durum bilgisi (Beklemede, İptal, Check-in Yapıldı) bu sınıfta tutulur. Varsayılan durum Pending olarak belirlenmiştir.

```

namespace prgmlab3.Models
{
    public class ReservationModel : BaseModel
    {
        public const string StatusPending = "Pending";
        public const string StatusCancelled = "Cancelled";
        public const string StatusCheckedIn = "CheckedIn";
        public int Id { get; set; }
        public int UserId { get; set; }
        public int FlightId { get; set; }
        public float Price { get; set; }
        public int SeatId { get; set; }
        public string Status { get; set; } = StatusPending;

        public ReservationModel(int userId, int flightId, float
price, int seatId)
        {
            UserId = userId;
            FlightId = flightId;
            Price = price;
            SeatId = seatId;
        }
    }
}

```

Şekil 35: ReservationModel sınıfının temel özellikleri ve kurucu metodu

##### 2. İşlemler

Rezervasyonu veritabanına kaydetme, belirli bir rezervasyonu iptal etme veya check-in yapma işlemleri bu metodlarla gerçekleştirilir. Ayrıca, bir uçuşa ait dolu koltukları ve belirli bir kullanıcıya ait tüm rezervasyonları listeleyen yardımcı fonksiyonlar sunar.

```

public void Save()
{
    Execute("INSERT INTO reservations (user_id, flight_id,
price, seat_id, status) VALUES (@u, @f, @p, @s, @st)", cmd =>
    {
        cmd.Parameters.AddWithValue("@u", UserId);
        cmd.Parameters.AddWithValue("@f", FlightId);
        cmd.Parameters.AddWithValue("@p", Price);
        cmd.Parameters.AddWithValue("@s", SeatId);
        cmd.Parameters.AddWithValue("@st", Status);
    });
}

public static void CancelById(int id)
{
    Execute(@"
        UPDATE reservations
        SET status = @st
        WHERE id = @id
        AND (status IS NULL OR status <> @st);
    ", cmd =>
    {
        cmd.Parameters.AddWithValue("@id", id);
        cmd.Parameters.AddWithValue("@st", StatusCancelled);
    });
}

public static void CheckInById(int id)
{
    Execute(@"
        UPDATE reservations
        SET status = @st
        WHERE id = @id;
    ", cmd =>
    {
        cmd.Parameters.AddWithValue("@id", id);
        cmd.Parameters.AddWithValue("@st", StatusCheckedIn);
    });
}

public void Cancel()
{
    if (Id <= 0)
        throw new InvalidOperationException("Cancel() için
geçerli bir Id gerekli.");
    CancelById(Id);
    Status = StatusCancelled; // local durumunu da güncelle
}

public static List<int> GetReservedSeatIds(int flightId)
{
    var list = new List<int>();
    var rows = Query("SELECT seat_id FROM reservations WHERE
flight_id=@fid AND (status IS NULL OR status <> @st)", cmd =>
    {
        cmd.Parameters.AddWithValue("@fid", flightId);
        cmd.Parameters.AddWithValue("@st", StatusCancelled);
    });
    foreach (var r in rows)
    {
        list.Add(Convert.ToInt32(r["seat_id"]));
    }
    return list;
}

public static List<ReservationModel> GetByUserId(int userId)
{
    var list = new List<ReservationModel>();
    var rows = Query("SELECT * FROM reservations WHERE
user_id=@uid", cmd => cmd.Parameters.AddWithValue("@uid", userId));
    foreach (var r in rows)
    {
        list.Add(new ReservationModel(
            Convert.ToInt32(r["user_id"]),
            Convert.ToInt32(r["flight_id"]),
            Convert.ToString(r["price"]),
            Convert.ToInt32(r["seat_id"])
        ));
        if (r["status"] != null)
            Status = (r["status"] as string) ?? StatusPending;
    }
    return list;
}
}

```

Şekil 36: ReservationModel üzerinde kayıt, iptal, check-in ve listeleme işlemleri

#### 4.1.6. PlaneModel (Uçak Modeli)

Uçak bilgilerini ve uçağa bağlı koltuk oluşturma mantığını içeren model sınıfıdır.

##### 1. Özellikler ve Listeleme

Uçağın adı ve koltuk sayısı gibi temel bilgileri tutar. Doğrulama öznitelikleri ile boş ad veya geçersiz koltuk sayısı gibi hatalı girişler engellenir. `GetAll` metodu, veritabanındaki tüm uçak kayıtlarını listeleyerek `PlaneModel` nesnelerine dönüştürür.

```

namespace prgmlab3.Models
{
    // Uçak bilgisini temsil eden model.
    // Admin tarafından uçak ekleme, listeleme ve koltuk üretimi için
    // kullanılır.
    public class PlaneModel : BaseModel
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Uçak adı zorunludur.")]
        [Display(Name = "Uçak Adı")]
        [StringLength(100, ErrorMessage = "Uçak adı en fazla 100
        karakter olabilir.")]
        public string Name { get; set; } = "";

        [Required(ErrorMessage = "Koltuk sayısı zorunludur.")]
        [Range(1, int.MaxValue, ErrorMessage = "Koltuk sayısı en az 1
        olmalıdır.")]
        [Display(Name = "Koltuk Sayısı")]
        public int SeatCount { get; set; }

        /// Tüm uçakları döndürür,
        public static List<PlaneModel> GetAll()
        {
            var res = Query("SELECT * FROM planes", cmd => { });
            List<PlaneModel> planes = new();

            foreach (var row in res)
            {
                planes.Add(new PlaneModel
                {
                    Id = Convert.ToInt32(row["id"]),
                    Name = Convert.ToString(row["name"]) ?? "",
                    SeatCount = Convert.ToInt32(row["seat_count"])
                });
            }
            return planes;
        }
    }
}

```

Şekil 37: PlaneModel sınıfının temel özellikleri ve GetAll metodu

## 2. Kaydetme ve Otomatik Koltuk Oluşturma

Save metodu, yeni bir uçak eklerken veya mevcut uçağı güncellerken kullanılır. Yeni uçak eklendiğinde, belirtilen koltuk sayısı kadar kayıt otomatik olarak `seats` tablosuna eklenir ve her koltuk `Ekonomi` sınıfı (`class= 0`) olarak işaretlenir. Delete metodu ise önce uçağa bağlı tüm koltukları, ardından uçağın kendisini siler.

```

// Yeni uçak ekler veya mevcut uçuşu günceller.
// Yeni uçak için otomatik koltuk kayıtları oluşturulur.
public void Save()
{
    Name = (Name ?? "").Trim();
    if (string.IsNullOrWhiteSpace(Name))
        throw new ArgumentException("Uçak adı boş olamaz.", nameof(Name));

    if (SeatCount < 1)
        throw new ArgumentOutOfRangeException(nameof(SeatCount), "Koltuk sayısı en az 1 olmalıdır.");
    if (Id == 0)
    {
        // Yeni uçak ekle
        SqLiteDbHelper.ExecuteNonQuery(
            "INSERT INTO planes (name, seat_count) VALUES (@name, @seat)",
            cmd =>
            {
                cmd.Parameters.AddWithValue("@name", Name);
                cmd.Parameters.AddWithValue("@seat", SeatCount);
            });
    }

    // Son eklenen ID
    var id = SqLiteDbHelper.ExecuteScalar<long>("SELECT last_insert_rowid()", null);
    Id = (int)id;

    // Bu uçak için otomatik koltuk oluştur
    for (int i = 1; i <= SeatCount; i++)
    {
        var seatNumber = i.ToString();
        SqLiteDbHelper.ExecuteNonQuery(
            "INSERT INTO seats (plane_id, seat_number, class) VALUES (@pid, @sn, @class)",
            cmd =>
            {
                cmd.Parameters.AddWithValue("@pid", Id);
                cmd.Parameters.AddWithValue("@sn", seatNumber);
                cmd.Parameters.AddWithValue("@class", 0);
            });
        // 0 = Ekonomi
    }
}
else
{
    // Mevcut uçuşu güncelle
    SqLiteDbHelper.ExecuteNonQuery(
        "UPDATE planes SET name=@name, seat_count=@seat WHERE id=@id",
        cmd =>
        {
            cmd.Parameters.AddWithValue("@id", Id);
            cmd.Parameters.AddWithValue("@name", Name);
            cmd.Parameters.AddWithValue("@seat", SeatCount);
        });
}

// NOT: Gerekirse burada koltuk ekleme/silme mantığı eklenebilir (TODO).
}

/// Önce uçuşa bağlı koltukları, ardından uçuşa siler.
public static void Delete(int id)
{
    // Önce koltukları sil
    SqLiteDbHelper.ExecuteNonQuery(
        "DELETE FROM seats WHERE plane_id=@id",
        cmd => cmd.Parameters.AddWithValue("@id", id));
}

// Sonra uçuşu sil
SqLiteDbHelper.ExecuteNonQuery(
    "DELETE FROM planes WHERE id=@id",
    cmd => cmd.Parameters.AddWithValue("@id", id));
}
}
}

```

Şekil 38: PlaneModel üzerinde kaydetme, otomatik koltuk üretimi ve silme işlemleri

#### 4.1.7. SeatModel (Koltuk Modeli)

Uçak koltuğunu temsil eden model sınıfıdır.

##### 1. Özellikler ve Getirme

Koltuk numarası, sınıfı (0: Economy, 1: Business) ve bağlı olduğu uçuşın ID bilgilerini tutar. `GetByPlane` metodu belirli bir uçuşa ait tüm koltukları,  `GetById` ise tek bir koltuk kaydını döndürür.

```

namespace prgmlab3.Models
{
    // Uçak koltuğu modelidir.
    // 0: Ekonomi, 1: Business
    public class SeatModel : BaseModel
    {
        public int Id { get; set; }

        [Required]
        [DisplayName = "Uçak"]
        public int PlaneId { get; set; }

        [Required(ErrorMessage = "Koltuk numarası zorunludur.")]
        [DisplayName = "Koltuk No"]
        [StringLength(10, ErrorMessage = "Koltuk numarası en fazla 10 karakter olabilir.")]
        public string SeatNumber { get; set; } = "";

        [Range(0, 1, ErrorMessage = "Koltuk sınıfı 0 (Economy) veya 1 (Business) olmalıdır.")]
        [DisplayName = "Sınıf"]
        public int Class { get; set; } // 0: Economy, 1: Business

        // Belirli bir uçağa ait tüm koltukları getirir.
        public static List<SeatModel> GetByPlane(int planeId)
        {
            var res = Query(
                "SELECT * FROM seats WHERE plane_id=@id",
                c => c.Parameters.AddWithValue("@id", planeId));
            List<SeatModel> list = new();
            foreach (var row in res)
            {
                list.Add(new SeatModel
                {
                    Id      = Convert.ToInt32(row["id"]),
                    PlaneId = Convert.ToInt32(row["plane_id"]),
                    SeatNumber = Convert.ToString(row["seat_number"]) ??
                    "",

                    Class   = Convert.ToInt32(row["class"])
                });
            }
            return list;
        }

        // Id'ye göre tek koltuk getirir.
        public static SeatModel? GetById(int id)
        {
            var res = Query(
                "SELECT * FROM seats WHERE id=@id",
                c => c.Parameters.AddWithValue("@id", id));
            if (res.Count == 0)
                return null;
            var row = res[0];
            return new SeatModel
            {
                Id      = Convert.ToInt32(row["id"]),
                PlaneId = Convert.ToInt32(row["plane_id"]),
                SeatNumber = Convert.ToString(row["seat_number"]) ??
                "",

                Class   = Convert.ToInt32(row["class"])
            };
        }
    }
}

```

Şekil 39: SeatModel özellikleri ve GetByPlane / GetById metodları

## 2. Kaydetme ve Silme

`Save` metodu, yeni koltuk ekleme veya mevcut koltuğu güncelleme işlemlerini yapar; koltuk numarası, uçak ID'si ve sınıf değeri için geçerlilik kontrolleri içerir. `Delete` metodu ise ilgili koltuğu veritabanından siler.

```

// Yeni koltuk ekler veya mevcut koltuğu günceller.
public void Save()
{
    SeatNumber = (SeatNumber ?? "").Trim();
    if (string.IsNullOrWhiteSpace(SeatNumber))
        throw new ArgumentException("Koltuk numarası boş olamaz.", nameof(SeatNumber));
    if (PlaneId <= 0)
        throw new ArgumentOutOfRangeException(nameof(PlaneId), "Geçersiz uçak ID.");
    if (Class is < 0 or > 1)
        throw new ArgumentOutOfRangeException(nameof(Class), "Koltuk sınıfı 0 veya 1 olmalıdır.");
    if (Id == 0)
    {
        // INSERT
        SqliteDbHelper.ExecuteNonQuery(
            "INSERT INTO seats (plane_id, seat_number, class)
VALUES (@pid, @sn, @c)",
            cmd =>
        {
            cmd.Parameters.AddWithValue("@pid", PlaneId);
            cmd.Parameters.AddWithValue("@sn", SeatNumber);
            cmd.Parameters.AddWithValue("@c", Class);
        });
        // Son eklenen id
        var newId = SqliteDbHelper.ExecuteScalar<long>(
            "SELECT last_insert_rowid();", null);
        Id = (int)newId;
    }
    else
    {
        // UPDATE
        SqliteDbHelper.ExecuteNonQuery(
            "UPDATE seats SET plane_id=@pid, seat_number=@sn,
class=@c WHERE id=@id",
            cmd =>
        {
            cmd.Parameters.AddWithValue("@id", Id);
            cmd.Parameters.AddWithValue("@pid", PlaneId);
            cmd.Parameters.AddWithValue("@sn", SeatNumber);
            cmd.Parameters.AddWithValue("@c", Class);
        });
    }
}

// Koltuğu siler
public static void Delete(int id)
{
    SqliteDbHelper.ExecuteNonQuery(
        "DELETE FROM seats WHERE id=@id",
        cmd => cmd.Parameters.AddWithValue("@id", id));
}
}

```

Sekil 40: SeatModel üzerinde kaydetme (Save) ve silme (Delete) işlemleri

#### 4.1.8. AirportModel (Havalimanı Modeli)

Havaalanı bilgisini temsil eden model sınıfıdır.

##### 1. Özellikler ve Getirme

Havaalanının kodu, şehri, adı ve ülkesi gibi temel alanları tutar. GetById metodu, verilen ID'ye göre tek bir havaalanı kaydını döndürür; kayıt yoksa null verir.

```

namespace prgmlab3.Models
{
    // Havaalanı bilgisini temsil eden model.
    public class AirportModel : BaseModel
    {
        public int Id { get; set; }

        [Display(Name = "Kod")]
        [StringLength(10, ErrorMessage = "Kod en fazla 10 karakter
olabilir.")]
        public string Code { get; set; } = "";

        [Required(ErrorMessage = "Şehir bilgisi zorunludur.")]
        [Display(Name = "Şehir")]
        [StringLength(100, ErrorMessage = "Şehir adı en fazla 100
karakter olabilir.")]
        public string City { get; set; } = "";

        [Required(ErrorMessage = "Havaalanı adı zorunludur.")]
        [Display(Name = "Havaalanı Adı")]
        [StringLength(150, ErrorMessage = "Havaalanı adı en fazla 150
karakter olabilir.")]
        public string Name { get; set; } = "";

        [Display(Name = "Ülke")]
        [StringLength(100, ErrorMessage = "Ülke adı en fazla 100
karakter olabilir.")]
        public string Country { get; set; } = "";

        // Id'ye göre tek bir havaalanı kaydı döndürür.
        public static AirportModel? GetById(int id)
        {
            var res = Query(
                "SELECT * FROM airports WHERE id=@id",
                c => c.Parameters.AddWithValue("@id", id)
            );
            if (res.Count == 0)
                return null;
            var r = res[0];
            return new AirportModel
            {
                Id      = Convert.ToInt32(r["id"]),
                Code   = Convert.ToString(r["code"]) ?? "",
                City   = Convert.ToString(r["city"]) ?? "",
                Name   = Convert.ToString(r["name"]) ?? "",
                Country = Convert.ToString(r["country"]) ?? ""
            };
        }
    }
}

```

Sekil 41: AirportModel özellikleri ve GetById metodu

## 2. Kaydetme ve Silme

`Save` metodu, `Id` değeri 0 ise yeni havaalanı kaydı ekler, aksi durumda mevcut kaydı günceller. `Kod` ve metin alanları trim edilip büyük harfe çevrilerek veri tutarlılığı sağlanır. `Delete` metodu ise belirtilen `Id`'ye ait havaalanı kaydını siler.

```

// Yeni kayıt ekler veya mevcut kaydı günceller.
// Id = 0 ise insert, değilse update.
public void Save()
{
    // Null güvenliği + trim
    Code   = (Code   ?? "").Trim();
    City   = (City   ?? "").Trim();
    Name   = (Name   ?? "").Trim();
    Country = (Country ?? "").Trim();

    if (!string.IsNullOrEmpty(Code))
        Code = Code.ToUpperInvariant();

    if (Id == 0)
    {
        // Yeni kayıt
        SqliteDbHelper.ExecuteNonQuery(
            "INSERT INTO airports (code, city, name, country)"
            VALUES (@code, @city, @name, @country)",
            cmd =>
            {
                cmd.Parameters.AddWithValue("@code", Code);
                cmd.Parameters.AddWithValue("@city", City);
                cmd.Parameters.AddWithValue("@name", Name);
                cmd.Parameters.AddWithValue("@country",
                    Country);
            });
        var id = SqliteDbHelper.ExecuteScalar<long>("SELECT
last_insert_rowid();", null);
        Id = (int)id;
    }
    else
    {
        // Güncelleme
        SqliteDbHelper.ExecuteNonQuery(
            "UPDATE airports SET code=@code, city=@city,
name=@name, country=@country WHERE id=@id",
            cmd =>
            {
                cmd.Parameters.AddWithValue("@id", Id);
                cmd.Parameters.AddWithValue("@code", Code);
                cmd.Parameters.AddWithValue("@city", City);
                cmd.Parameters.AddWithValue("@name", Name);
                cmd.Parameters.AddWithValue("@country",
                    Country);
            });
    }
}

// Id'ye göre havaalanı kaydını siler.
public static void Delete(int id)
{
    SqliteDbHelper.ExecuteNonQuery(
        "DELETE FROM airports WHERE id=@id",
        cmd => cmd.Parameters.AddWithValue("@id", id)
    );
}

```

Şekil 42: AirportModel üzerinde kayıt (Save) ve silme (Delete) işlemleri

## 5. Kullanılan Teknolojiler ve Araçlar

Kategori	Teknoloji / Araç	Detaylı Açıklama
Programlama Dili	C#	Nesne Yönelimli Programlama (OOP) ilkelerinin uygulandığı temel dildir.
Geliştirme Çatısı	ASP.NET Core 8.0 MVC	Modüler, yüksek performanslı ve platformlar arası uyumlu bir web uygulama çatısıdır. Model-View-Controller (MVC) mimari deseni kullanılmıştır.
Veritabanı	SQLite	Hafif, sunucusuz ve dosya tabanlı yapısı nedeniyle tercih edilen yerel veritabanı çözümüdür.
Veri Erişim Kültüphanesi	Microsoft.Data.Sqlite	Veritabanı bağlantısı ve ham SQL sorgularının yürütülmesi için kullanılan kültüphanedir.
Ön Yüz (Frontend)	HTML5, CSS3, JavaScript	Arayüz tasarımı, stil ve istemci taraflı etkileşimler için kullanılan temel web standartlarıdır.
Dinamik İçerik	Razor View Engine	Sunucu tarafında dinamik HTML içeriği ve View katmanını oluşturmak için kullanılmıştır.
Geliştirme Ortamı (IDE)	Visual Studio Code	Kodlama, hata ayıklama ve projenin derlenebilir halde teslimini sağlamak için kullanılan entegre geliştirme ortamıdır.

Tablo 1: Projede kullanılan temel teknolojiler

## 6. Test Senaryoları ve Ekran Çıktıları

### 6.1. Örnek Test Senaryoları

Aşağıda sistemin temel işlevlerini kapsayan iki örnek test senaryosu verilmiştir.

- **Test Senaryosu 1 – Rezervasyon Oluşturma**

**Ön Koşul:** Kullanıcı sisteme giriş yapmıştır ve sistemde en az bir aktif uçuş bulunmaktadır.

**Adımlar:**

1. Kullanıcı ana sayfadan kalkış ve varış havalimanı ile tarih bilgilerini girerek “Uçuş Ara” butonuna tıklar.
2. Listelenen uçuşlardan birini seçerek “Koltuk Seç” ekranına geçer.
3. Boş bir koltuk seçer.
4. “Rezervasyonu Onayla” butonuna tıklar.

**Beklenen Sonuç:** Seçilen koltuk ilgili uçuş için kullanıcıya atanır, rezervasyon veritabanına kaydedilir ve ekranda rezervasyonun başarıyla oluşturulduğuna dair bilgilendirme mesajı gösterilir.

- **Test Senaryosu 2 – Adminin Koltuk Silmesi**

**Ön Koşul:** Admin kullanıcı sisteme giriş yapmıştır ve ilgili uçak için tanımlı en az bir koltuk bulunmaktadır.

**Adımlar:**

1. Admin, admin panelinden “Koltuk Yönetimi” ekranına gider.
2. Koltuk listesinden silmek istediği koltuğu seçer.
3. İlgili koltuğun karşısındaki “Sil” butonuna tıklar.
4. Sistem, seçilen koltuğa ait aktif bir rezervasyon olup olmadığını kontrol eder.
5. Varsa onay penceresinde silme işlemi ile ilgili bilgilendirme yapılır.

**Beklenen Sonuç:**

- Eğer koltuğa ait aktif bir rezervasyon **varsa**, sistem koltuğun silinmesine izin vermez ve admin kullanıcıya ilgili koltuk için rezervasyon bulunduğuna dair uyarı mesajı gösterir.
- Eğer koltuğa ait aktif bir rezervasyon **yoksa**, seçilen koltuk veritabanından silinir, koltuk listesi güncellenir ve admin kullanıcıya koltuğun başarıyla silindiğine dair bilgilendirme mesajı gösterilir.

## 6.2. Ekran Çıktıları

### 6.2.1. Ortak Arayüzler

Şekil 43: Ana sayfa, kayıt ol, giriş yap ve profil bilgilerim ekranları

## 6.2.2. Müşteri Arayüzleri

ID	Kalkış	Tarih	Koltuk	Fiyat	Durum	İşlemler
1	İstanbul -> Ankara	15.12.2020 13:09	12	\$12.08 k	Checkedin	<span style="color: green;">Check-in</span> <span style="color: red;">Sil</span>
2	İstanbul -> Ankara	15.12.2020 13:09	3	\$41.59 k	Pending	<span style="color: green;">Check-in</span> <span style="color: red;">Sil</span>

Şekil 44: Müşterinin uçuş arama ve rezervasyonları ekranları

## 6.2.3. Admin Arayüzleri

ID	TC No	Ad - Soyad	E-posta	Doğum Tarihi	Telefon	Rol
1	1111111111	Admin Admin	admin@admin.com	05/11/2015	5555555555	<span style="color: green;">Ekle</span> <span style="color: red;">Sil</span>
2	12345678901	Sena Nur Döğer	deneemedenedene.com	19.07.2003	0511111111	<span style="color: blue;">Ekle</span> <span style="color: red;">Sil</span>

ID	Uçak	Koltuk Sayısı	İşlemler
1	Oncel_Uçak	10	<span style="color: green;">Koltuk Ekle</span> <span style="color: red;">Sil</span>

ID	Koltuk No	Sıra	Uçak	İşlemler
1	1	1	Oncel_Uçak	<span style="color: red;">Sil</span>
2	2	2	Oncel_Uçak	<span style="color: red;">Sil</span>
3	3	3	Oncel_Uçak	<span style="color: red;">Sil</span>
4	4	4	Oncel_Uçak	<span style="color: red;">Sil</span>
5	5	5	Oncel_Uçak	<span style="color: red;">Sil</span>
6	6	6	Oncel_Uçak	<span style="color: red;">Sil</span>
7	7	7	Oncel_Uçak	<span style="color: red;">Sil</span>
8	8	8	Oncel_Uçak	<span style="color: red;">Sil</span>
9	9	9	Oncel_Uçak	<span style="color: red;">Sil</span>
10	10	10	Oncel_Uçak	<span style="color: red;">Sil</span>
11	11	11	Oncel_Uçak	<span style="color: red;">Sil</span>
12	12	12	Oncel_Uçak	<span style="color: red;">Sil</span>

ID	Kalkış	Kalkış Tarihi	Vergi	Kalkış Zamanı	Vergi Zamanı	Bilek Fiyatı	İşlemler
1	Oncel_Uçak	İstanbul	Ankara	15.12.2020 13:09	15.12.2020 14:09	600	<span style="color: green;">Bilek Ekle</span> <span style="color: red;">Sil</span>

ID	Koltuk No	Sıra	Uçak	İşlemler
1	1	1	Oncel_Uçak	<span style="color: red;">Sil</span>
2	2	2	Oncel_Uçak	<span style="color: red;">Sil</span>
3	3	3	Oncel_Uçak	<span style="color: red;">Sil</span>
4	4	4	Oncel_Uçak	<span style="color: red;">Sil</span>
5	5	5	Oncel_Uçak	<span style="color: red;">Sil</span>
6	6	6	Oncel_Uçak	<span style="color: red;">Sil</span>
7	7	7	Oncel_Uçak	<span style="color: red;">Sil</span>
8	8	8	Oncel_Uçak	<span style="color: red;">Sil</span>
9	9	9	Oncel_Uçak	<span style="color: red;">Sil</span>
10	10	10	Oncel_Uçak	<span style="color: red;">Sil</span>
11	11	11	Oncel_Uçak	<span style="color: red;">Sil</span>
12	12	12	Oncel_Uçak	<span style="color: red;">Sil</span>

ID	Kalkış	Kalkış Tarihi	Vergi	Kalkış Zamanı	Vergi Zamanı	Bilek Fiyatı	İşlemler
1	Sena Nur Döğer	1	12	\$12.08 k	<span style="color: green;">Check-in</span>	<span style="color: red;">Sil</span>	
2	Sena Nur Döğer	1	3	\$41.59 k	<span style="color: green;">Check-in</span>	<span style="color: red;">Sil</span>	

Şekil 45: Admin paneli ve yönetim ekranları

## **7. Sonuç ve Değerlendirme**

Bu proje kapsamında, işlevsel ve kullanıcı dostu bir Uçak Rezervasyon Sistemi geliştirilmiştir. Proje, hem son kullanıcıların (müşterilerin) kolayca uçuş arayıp rezervasyon yapabildiği, hem de yöneticilerin sistemdeki tüm operasyonları (uçak, uçuş, havaalanı yönetimi) kontrol edebildiği kapsamlı bir yapı sunmaktadır.

C# ve ASP.NET MVC mimarisi ile SQLite veritabanının birlikte kullanılması, hem katmanlı yapı hem de veri yönetimi konusunda deneyim kazandırmıştır.

Bundan sonraki adımlarda kullanıcı arayüzlerinin iyileştirilmesi ve projede eksik görülen kısımların tamamlanması planlanmaktadır.