

Mobil Programlama Dersi

Proje Ödevi

Hadi Bulsana - Apart Bulma Uygulaması

2112721044 Hüseyin KARABULUT

Uygulama Hakkında Bilgilendirme

Uygulama Adı: Hadi Bulsana

Uygulama Konusu: Apart Bulma Uygulaması

Uygulama Kullanıcıları: Apart arayan kiracılar ve apartını kiraya vermek isteyen ev sahipleri

Özet

Bu proje raporu, apart bulma uygulaması "Hadi Bulsana"nın tasarım ve geliştirme sürecini açıklamaktadır. Uygulama, apart arayan kiracılar ve apartlarını kiraya vermek isteyen ev sahipleri arasında bir platform sağlamaktadır. Kullanıcılar, ilanları filtreleyebilir, ev kiralayabilir ve kişisel bilgilerini yönetebilirler. Teknik olarak, uygulama Flutter kullanılarak dart programlama diliyle geliştirilmiş ve uygun bir veritabanı çözümüyle entegre edilmiştir.

Proje Amaçları

- Apart arayan kiracılar için uygun apartları bulmayı kolaylaştırmak.
- Ev sahiplerinin apartlarını kolayca ilan verebilmelerini sağlamak.
- Kullanıcıların ilanları filtreleyebilmeleri ve favorilerine ekleyebilmeleri.
- Kullanıcıların kişisel bilgilerini güncelleyebilmeleri ve hesap ayarlarını yönetebilmeleri.

Kullanıcı İşlevleri - Kiracı İşlevleri

1. Kayıt Olma/Giriş Yapma/Şifre Değiştirme

- Kullanıcılar, uygulamaya kaydolarak bir hesap oluşturabilirler.
- Var olan kullanıcılar mevcut hesaplarıyla giriş yapabilirler.
- Şifre değiştirme seçeneğiyle kullanıcılar şifrelerini güncelleyebilirler.

2. Şehir Bilgisi Girme ve İlan Görüntüleme

- Kiracılar, ilanları şehirlerine göre filtreleyebilirler.
- Yalnızca kendi şehirlerindeki ilanları görüntüleyebilirler.

3. Kişisel Bilgileri Görüntüleme/Değiştirme

- Kiracılar, hesaplarına ait kişisel bilgileri görüntüleyebilir ve düzenleyebilirler.
- Bu sayede iletişim bilgilerini güncelleyebilir veya profil fotoğrafını değiştirebilirler.

4. Yardım ve Diğer Sayfaları Görüntüleme

- Kiracılar, uygulama hakkında bilgilere ulaşabilecekleri "Hakkında" sayfasını görüntüleyebilirler.
- Yardım sayfası gibi diğer sayfalara da erişim sağlanabilir.

Kullanıcı İşlevleri - Ev Sahibi İşlevleri

1. Kayıt Olma/Giriş Yapma/Şifre Değiştirme

- Ev sahipleri, uygulamaya kaydolarak bir hesap oluşturabilirler.
- Var olan kullanıcılar mevcut hesaplarıyla giriş yapabilirler.
- Şifre değiştirme seçeneğiyle kullanıcılar şifrelerini güncelleyebilirler.

2. İlan Ekleme/Silme/Güncelleme/Listeleme

- Ev sahipleri, kiralık apart ilanlarını uygulamaya ekleyebilirler.
- Var olan ilanları güncelleyebilir veya silebilirler.
- Ev sahipleri, kendi ilanlarını listeleyebilir ve yönetebilirler.

3. Kişisel Bilgileri Görüntüleme/Değiştirme

- Ev sahipleri, hesaplarına ait kişisel bilgileri görüntüleyebilir ve düzenleyebilirler.
- Bu sayede iletişim bilgilerini güncelleyebilir veya profil fotoğrafını değiştirebilirler.

4. Yardım ve Diğer Sayfaları Görüntüleme

- Ev sahipleri, uygulama hakkında bilgilere ulaşabilecekleri "Hakkında" sayfasını görüntüleyebilirler.
- Yardım sayfası gibi diğer sayfalara da erişim sağlanabilir.

Utility Packages

intl: ^0.19.0
flutter_native_splash: ^2.3.7
flutter_slidable: ^3.0.1
shimmer: ^3.0.0

Icons

cupertino_icons: ^1.0.2
iconsax: ^0.0.8
font_awesome_flutter: ^10.2.1

State Management

provider: ^6.1.1

Images and Image Processing

images_picker: ^1.2.11
image_picker: ^1.0.5
cached_network_image: ^3.3.0
flutter_image_compress: ^2.1.0
photo_view: ^0.14.0
carousel_slider: ^4.2.1

UI

persistent_bottom_nav_bar: ^5.0.2
flutertoast: ^8.2.4

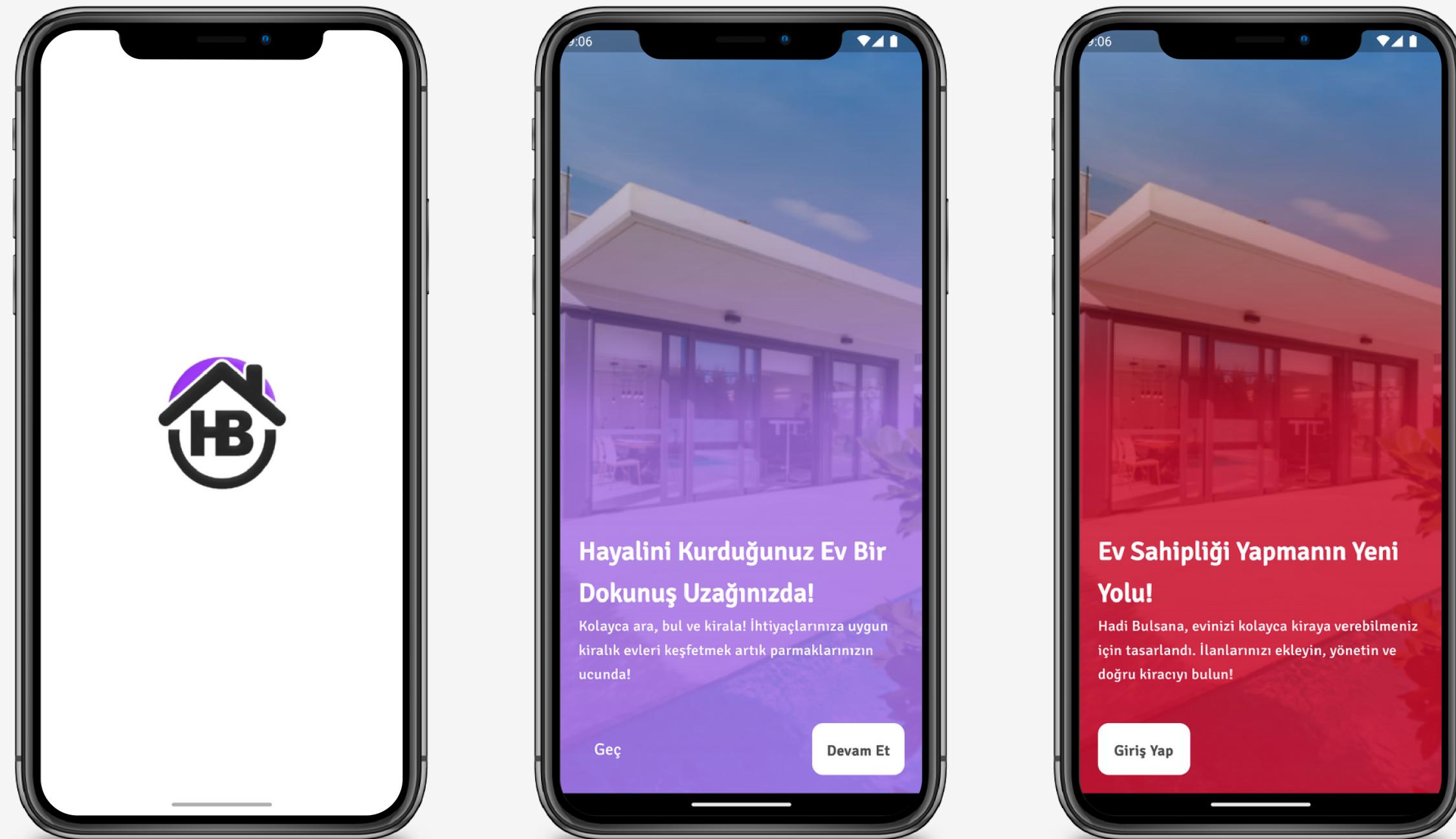
Database and Storage

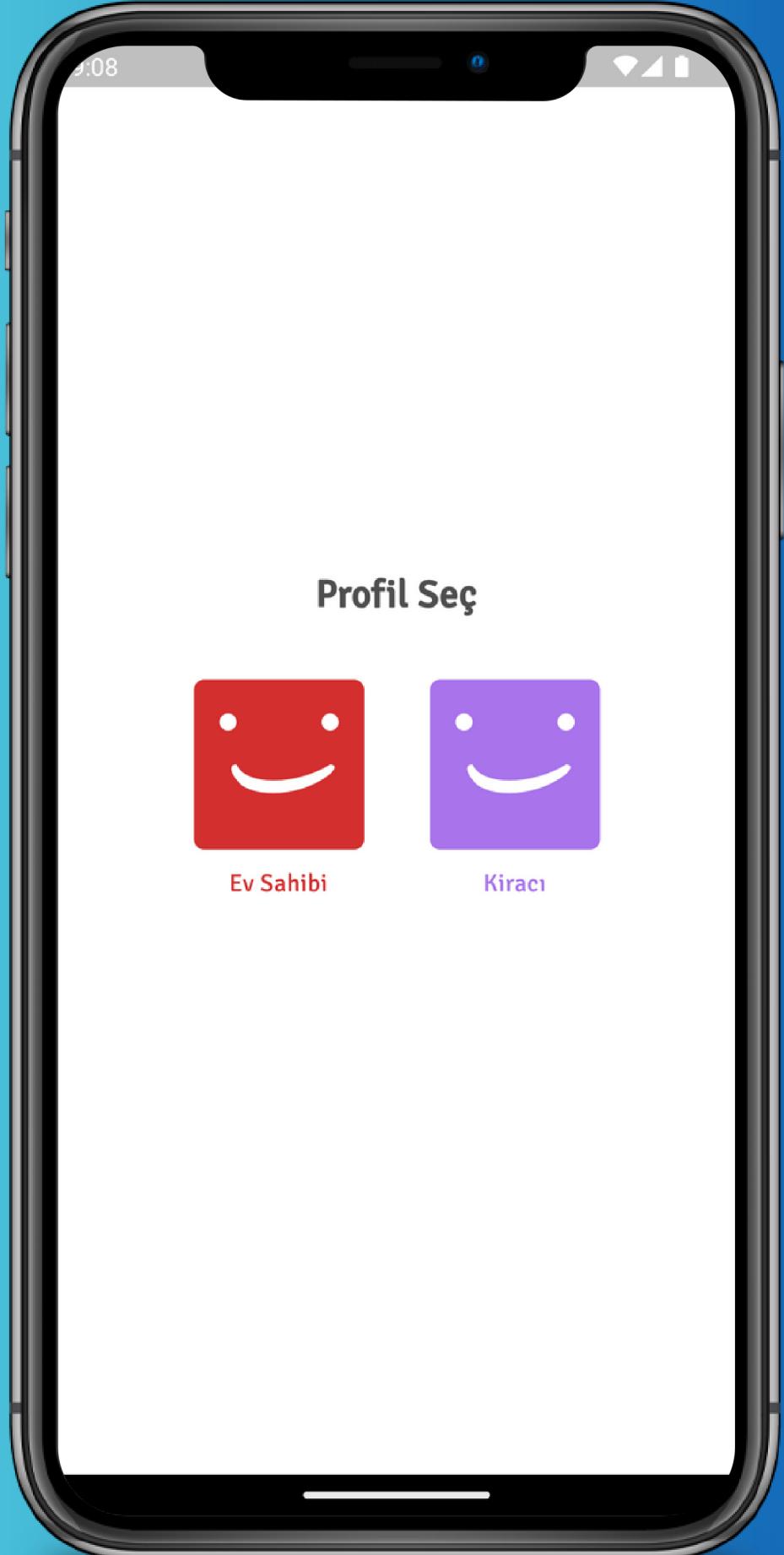
firebase_storage: ^11.5.6
firebase_auth: ^4.15.3
cloud_firestore: ^4.13.6
firebase_core: ^2.24.2

Kullandığım Kütüphaneler/Paketler

Başlıyoruz!

Uygulama ilk açıldığında kullanıcıyı önce Splash Screen ekranı karşılıyor. Sonrasında Onboard ekranında uygulama kullanıcıları hakkında kısa bilgi veriliyor kullanıcıya.



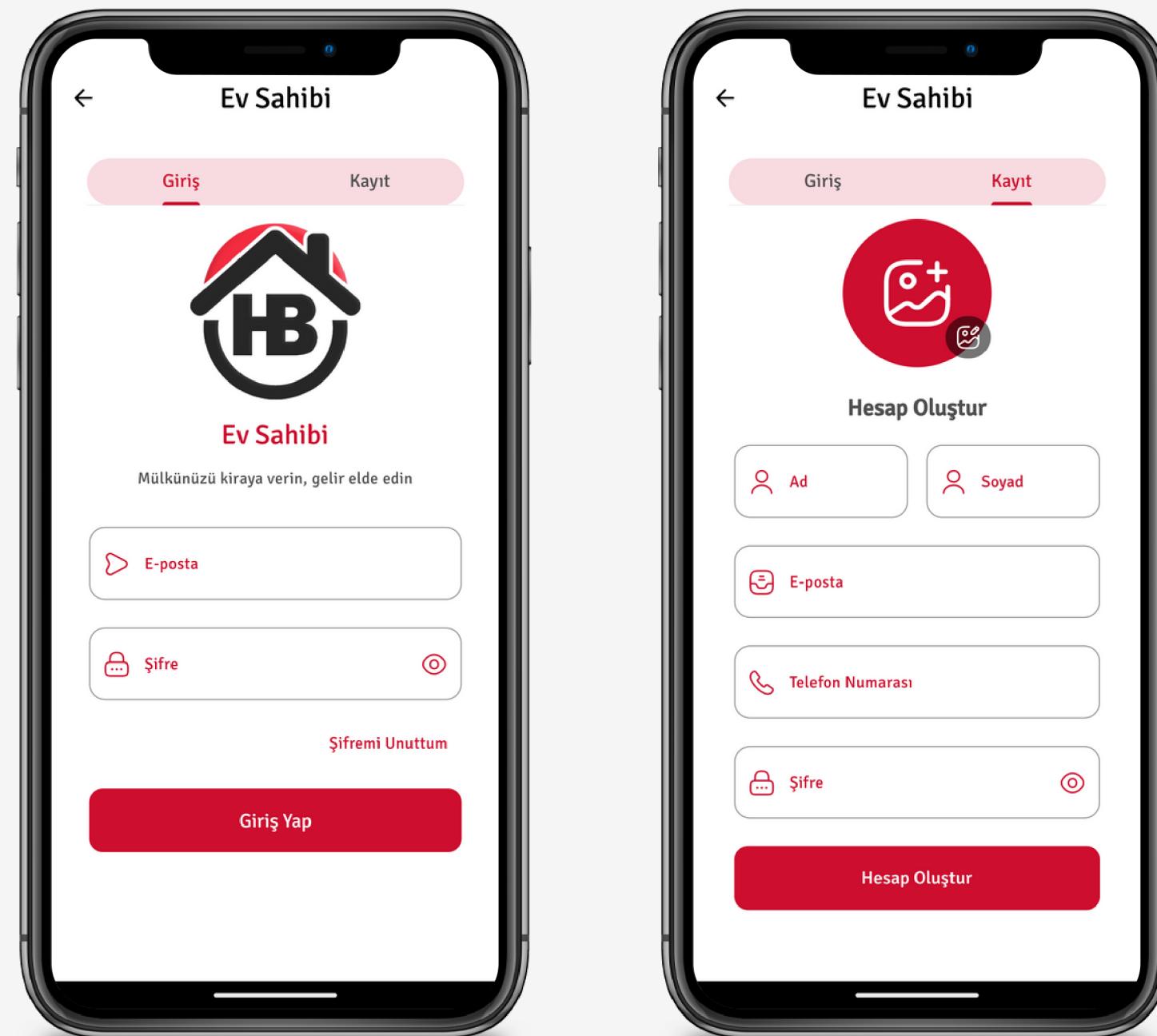


Profil Seçim Tercihi Ekranı

Burada ise kullanıcıya Profil seçmesi için seçim ekranı gösteriliyor. Kullanıcı kendisine göre Ev Sahibi veya Kiracı profillerinden birini seçiyor.

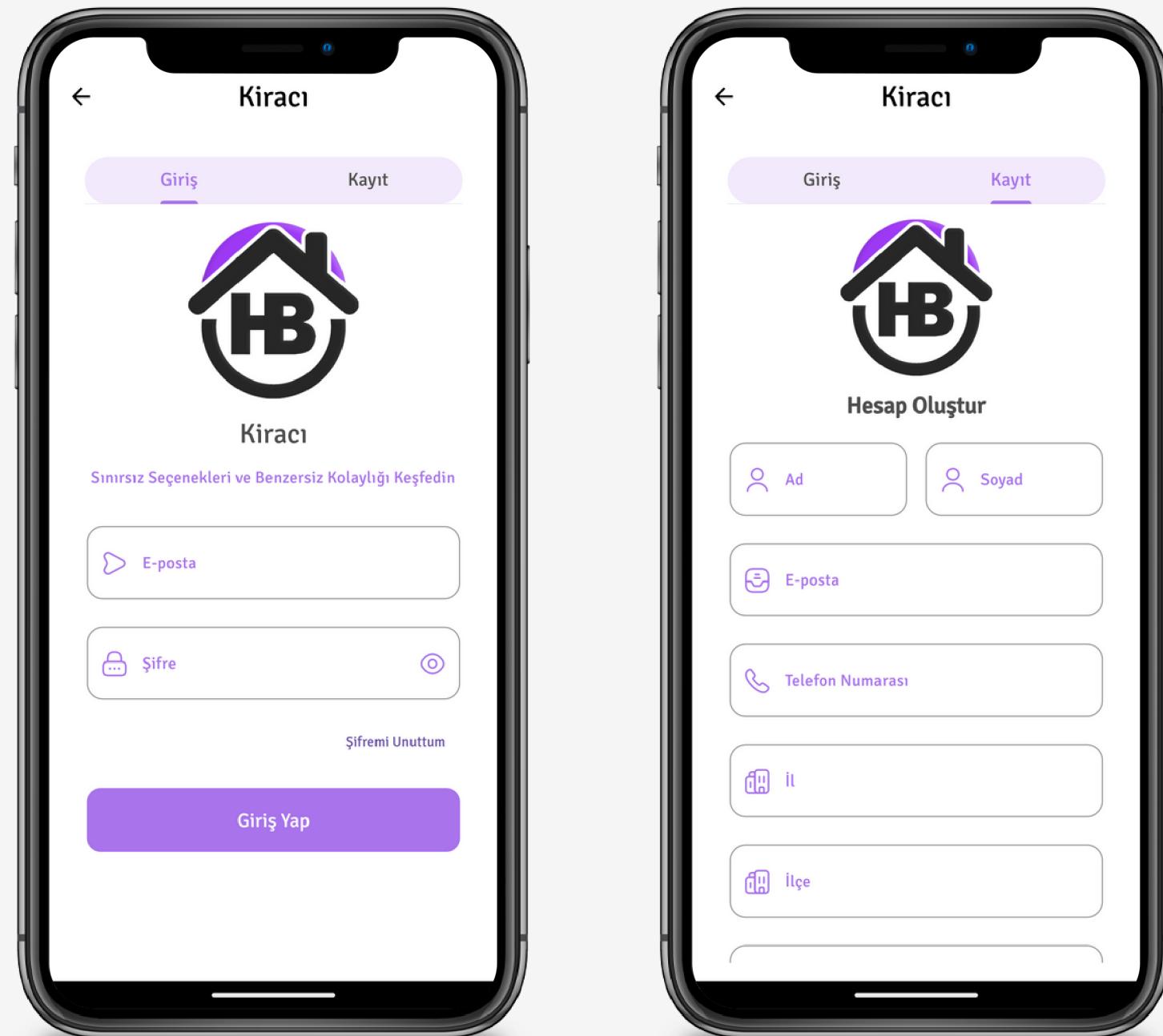
Ev Sahibi Giriş Yap ve Kayıt Ol Ekranları

Burada kullanıcı Ev Sahibi ise kendisiyle ilgili olan bu giriş sayfasında bilgilerini girip Giriş Yap butonuna tıkladığı zaman uygulamaya giriş yapabiliyor. Eğer kaydı yoksa sağ taraftaki resimde yer alan kayıt olma form ekranından bilgilerini girip Hesap Oluştur butonuna tıkladığı zaman uygulamaya başarılı bir şekilde kayıt oluyor.



Kiracı Giriş Yap ve Kayıt Ol Ekranları

Burada kullanıcı Kiracı ise kendisiyle ilgili olan bu giriş sayfasında bilgilerini girip Giriş Yap butonuna tıkladığı zaman uygulamaya giriş yapabiliyor. Eğer kaydı yoksa sağ taraftaki resimde yer alan kayıt olma form ekranından bilgilerini girip Hesap Oluştur butonuna tıkladığı zaman uygulamaya başarılı bir şekilde kayıt oluyor.

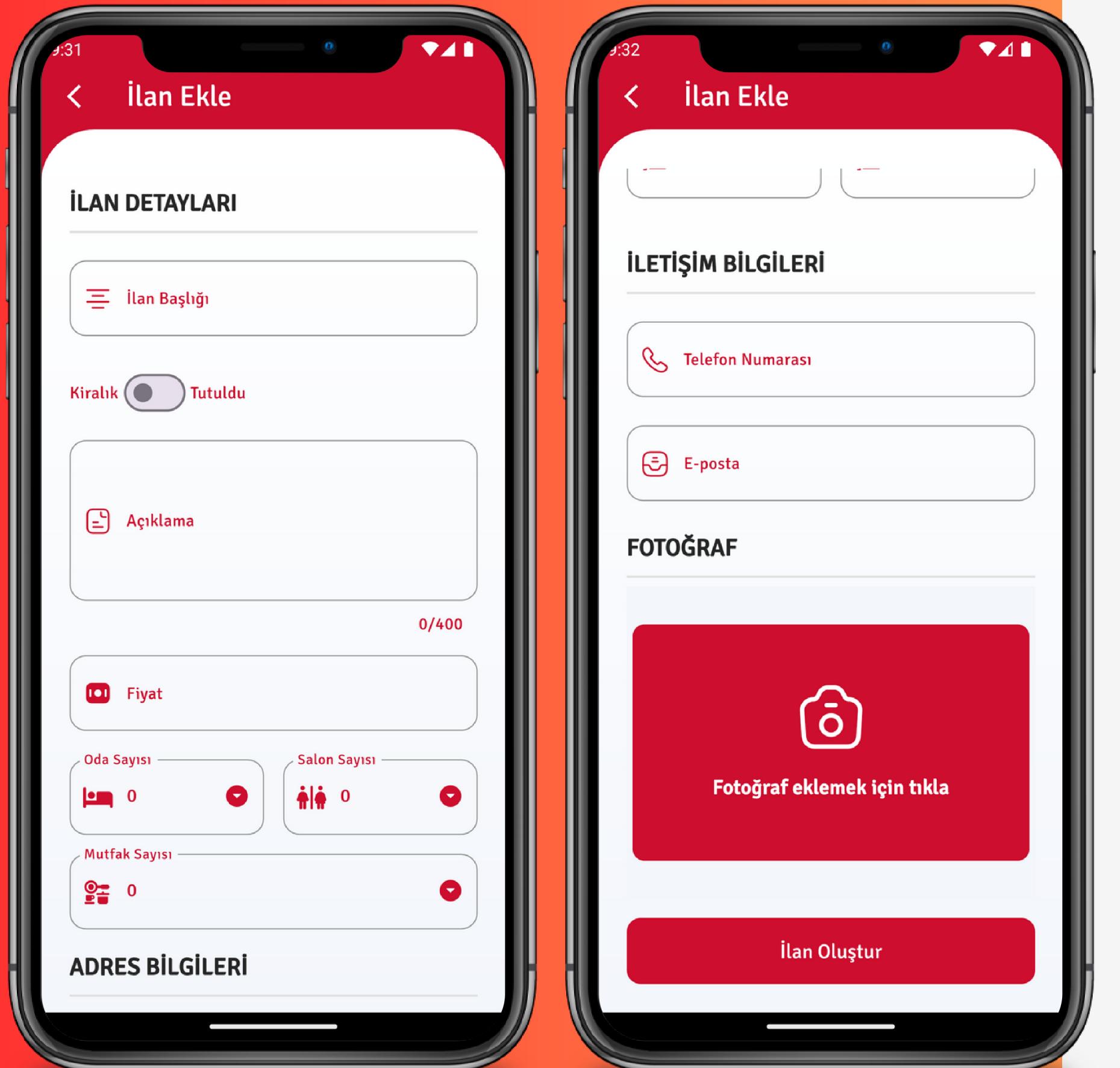




Ev Sahibi Anasayfa Ekranı

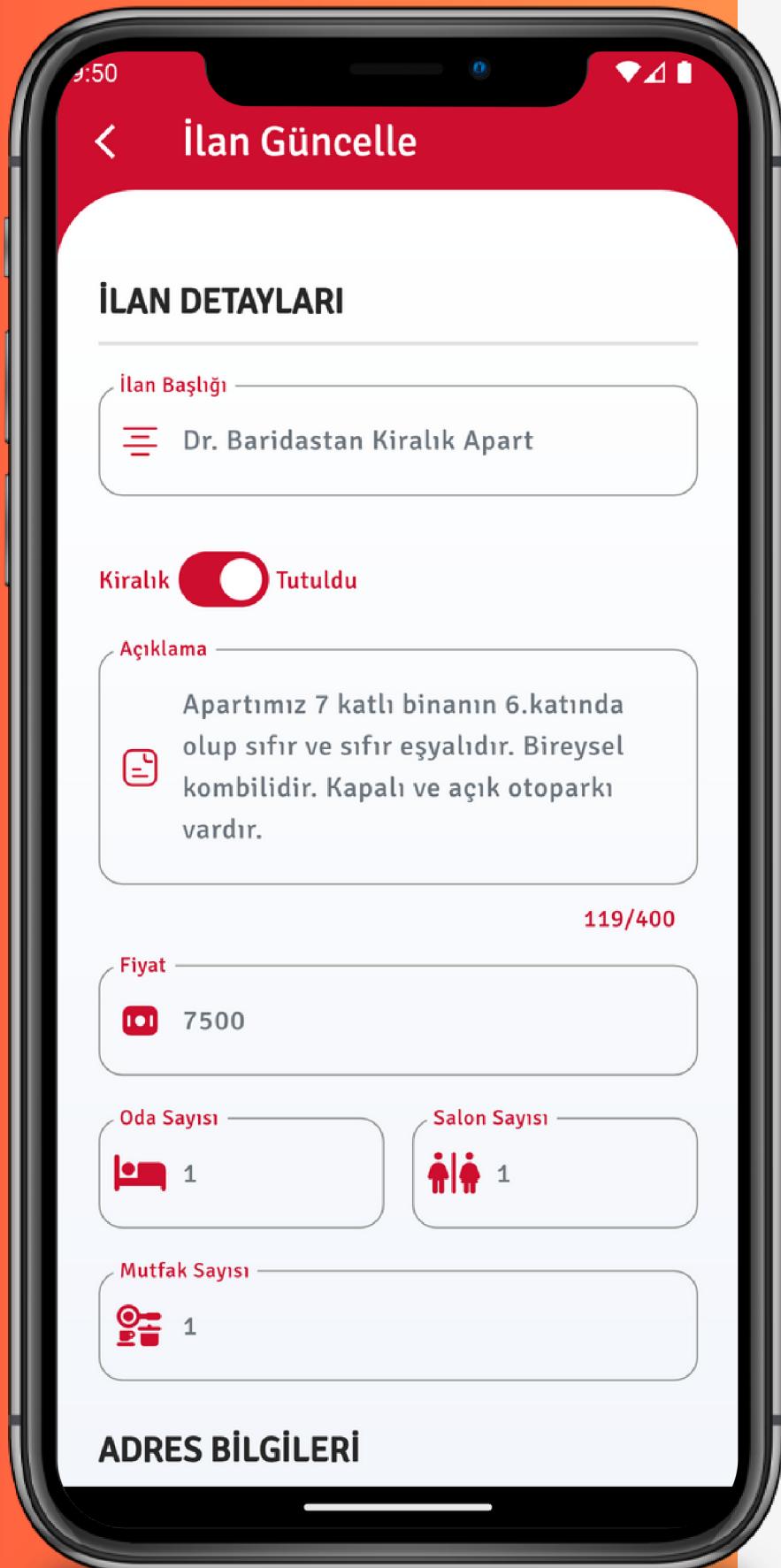
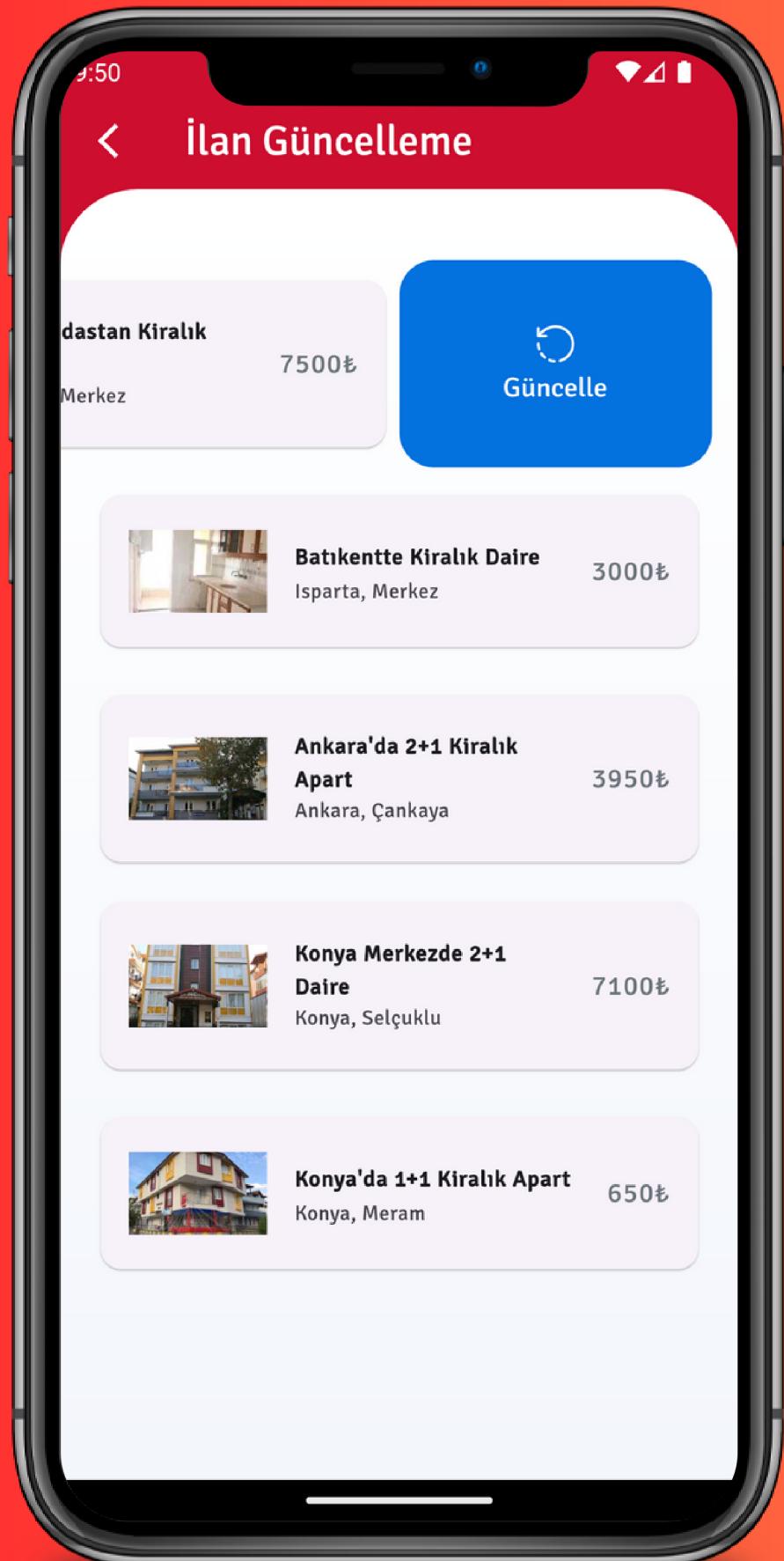
Bu sayfada yapabileceğini tüm işlemleri görebiliyor Ev Sahibi. Kısacası Ev Sahibinin (yani Adminin) yönetim paneli sayfası diyebiliriz bu sayfaya. Ev sahibi olan kullanıcılar burada:

- İlan Ekle modülü ile ilan ekleyebiliyor.
- İlan Güncelle modülü ile mevcut ilanlarından güncellemek istediği herhangi bir ilanı güncelleyebiliyor.
- İlanlarını Listele modülü ile sadece kendi açmış olduğu tüm ilanları görüntüleyebiliyor.
- İlan Sil modülü ile sadece kendi açmış olduğu tüm ilanlar arasından istediği herhangi bir ilanı silebiliyor.
- Hesabım modülü ile kendi profiline ulaşma, uygulama hakkında detaylı bilgi alma, iletişim vb. kanallara erişme imkanına sahip oluyor.
- Uygulama Hakkında modülü ile uygulamamız hakkındaki tüm bilgilere ulaşabiliyor.
- Kiralanmış İlanlar modülü ile kendi açmış olduğu ilanlar arasından kiracıların kiraladıkları ilanları görüntüleyebiliyor ve bunları ilandan kaldırabiliyorlar.
- Çıkış Yap modülü ile uygulamadan tamamen çıkış yapabilmeleri de mümkün.



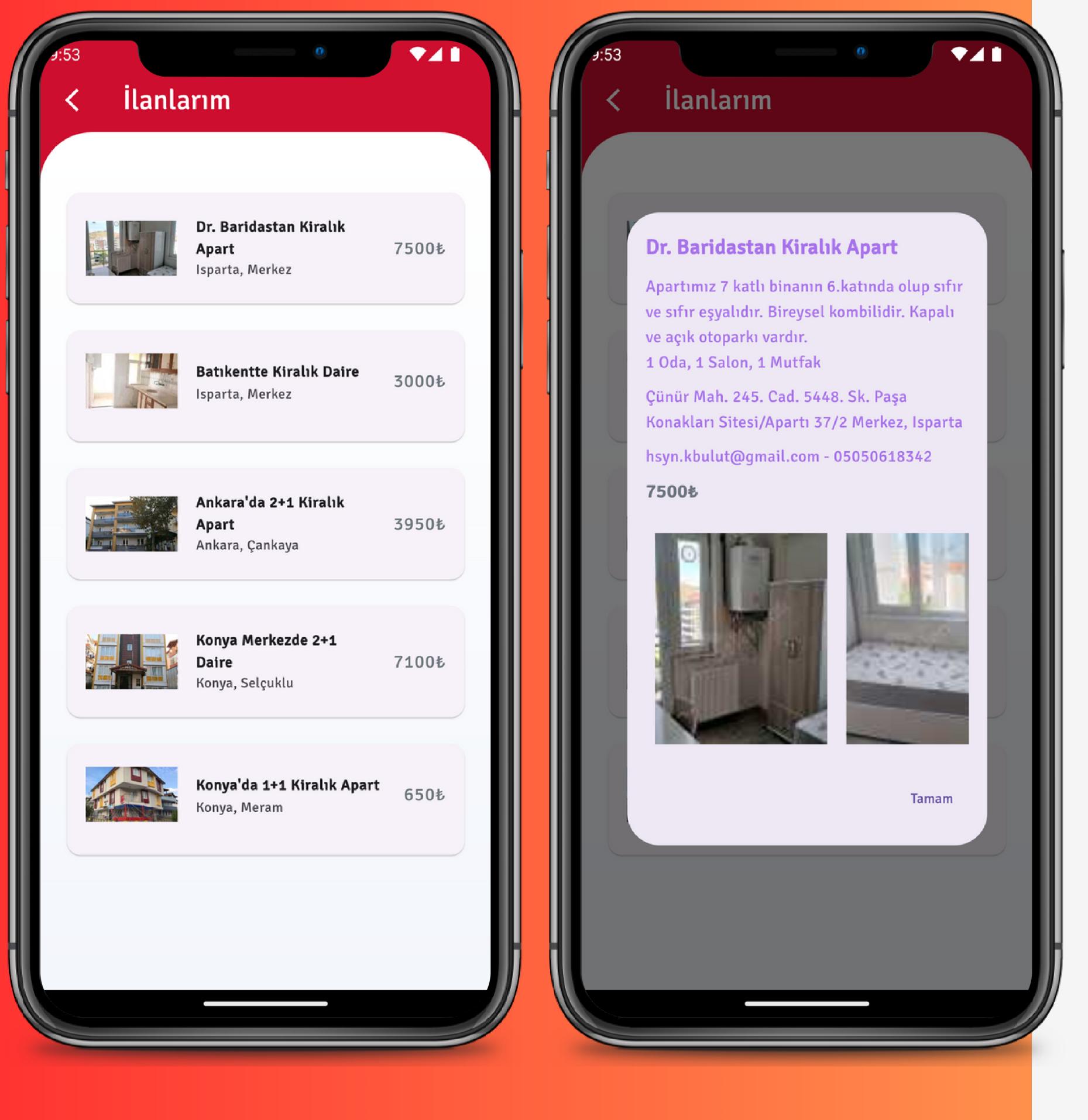
İlan Ekle Ekranı

Bu sayfada yer alan İlan Ekle modülü ile Ev Sahibi yeni ev ilan kaydı (ev ile ilgili bilgilerini girerek) ekleyebiliyor.



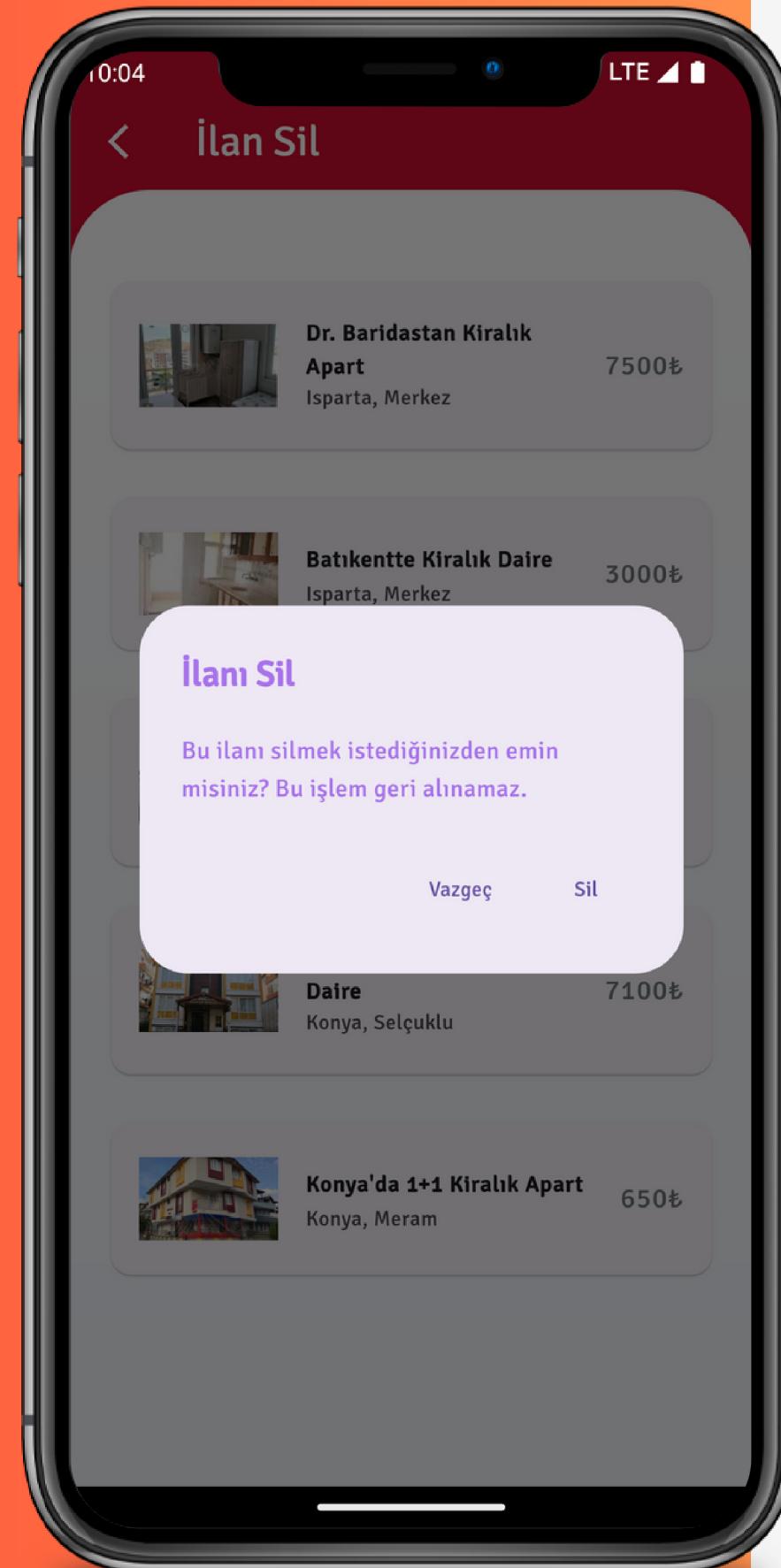
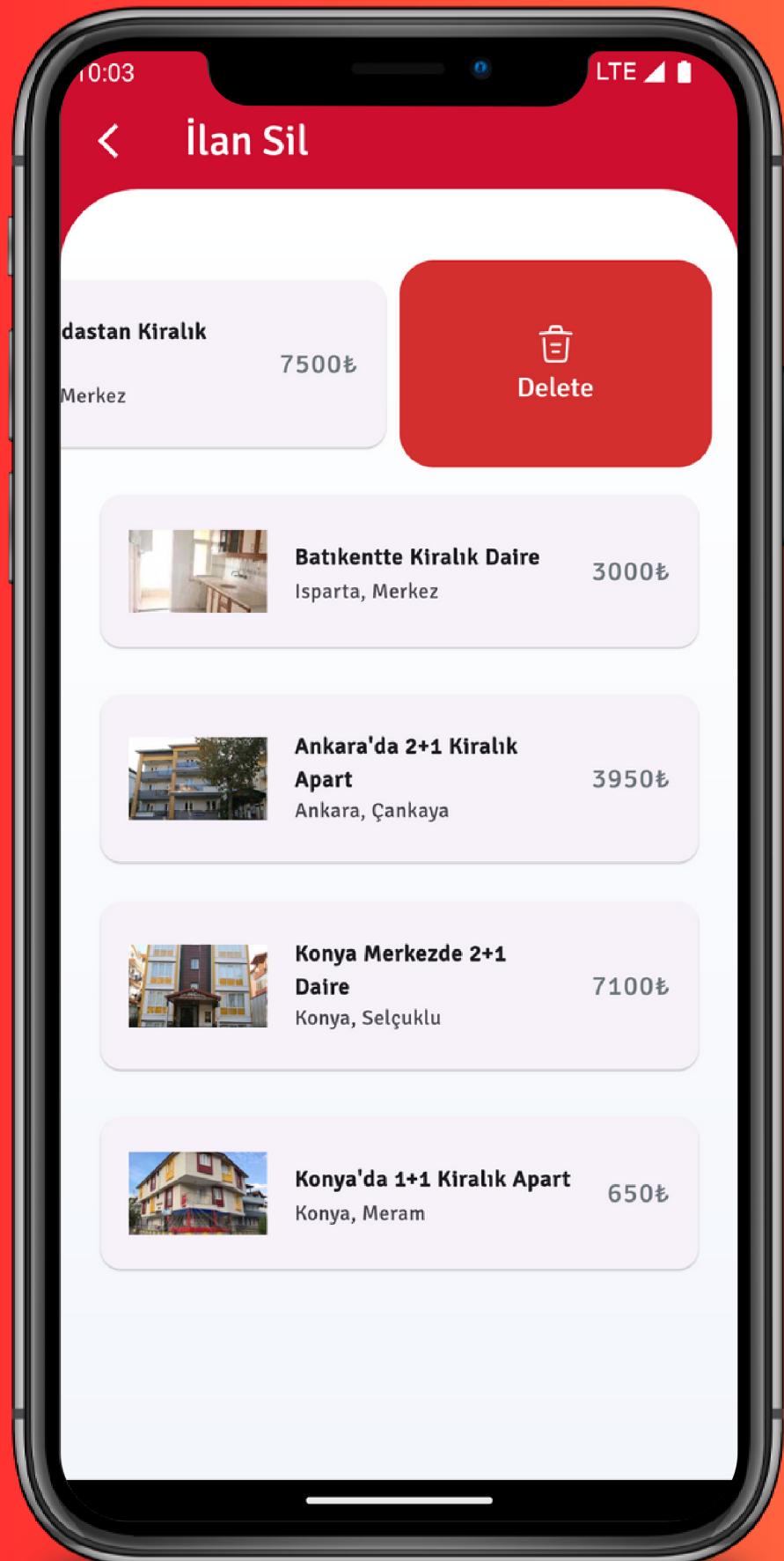
İlan Güncelle Ekranı

Bu sayfada Ev Sahibi, kendi açmış olduğu tüm ilanları listeleyip bunlar arasından herhangi bir mevcut ev ilanını seçip bilgilerini güncelleyebiliyor.



İlanlarını Liste Ekrani

Bu sayfada Ev Sahibi, kendi açmış olduğu tüm ilanları listeleyip bunlar arasından herhangi birine tıklaması durumunda ilanın detay bilgisine ulaşabiliyor.

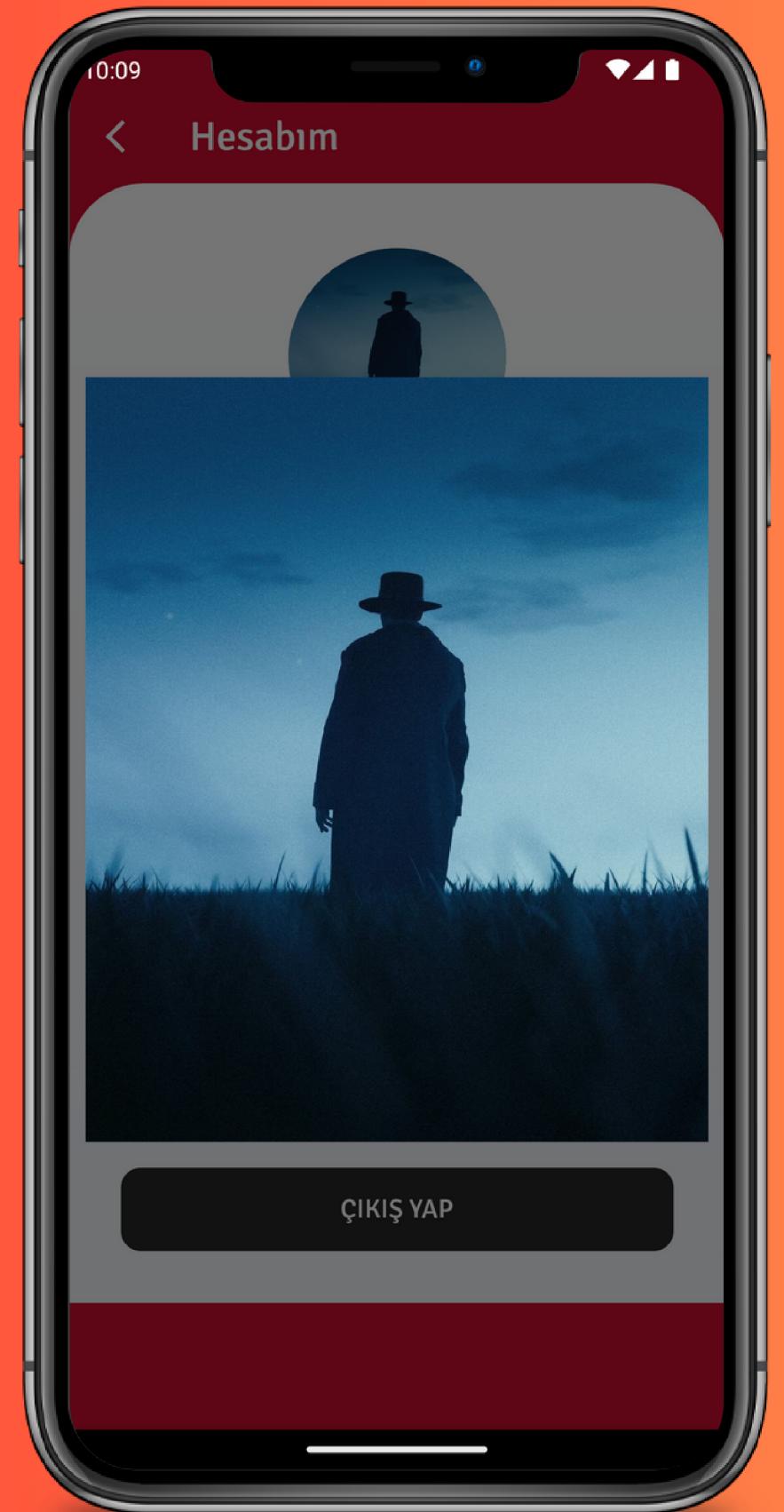
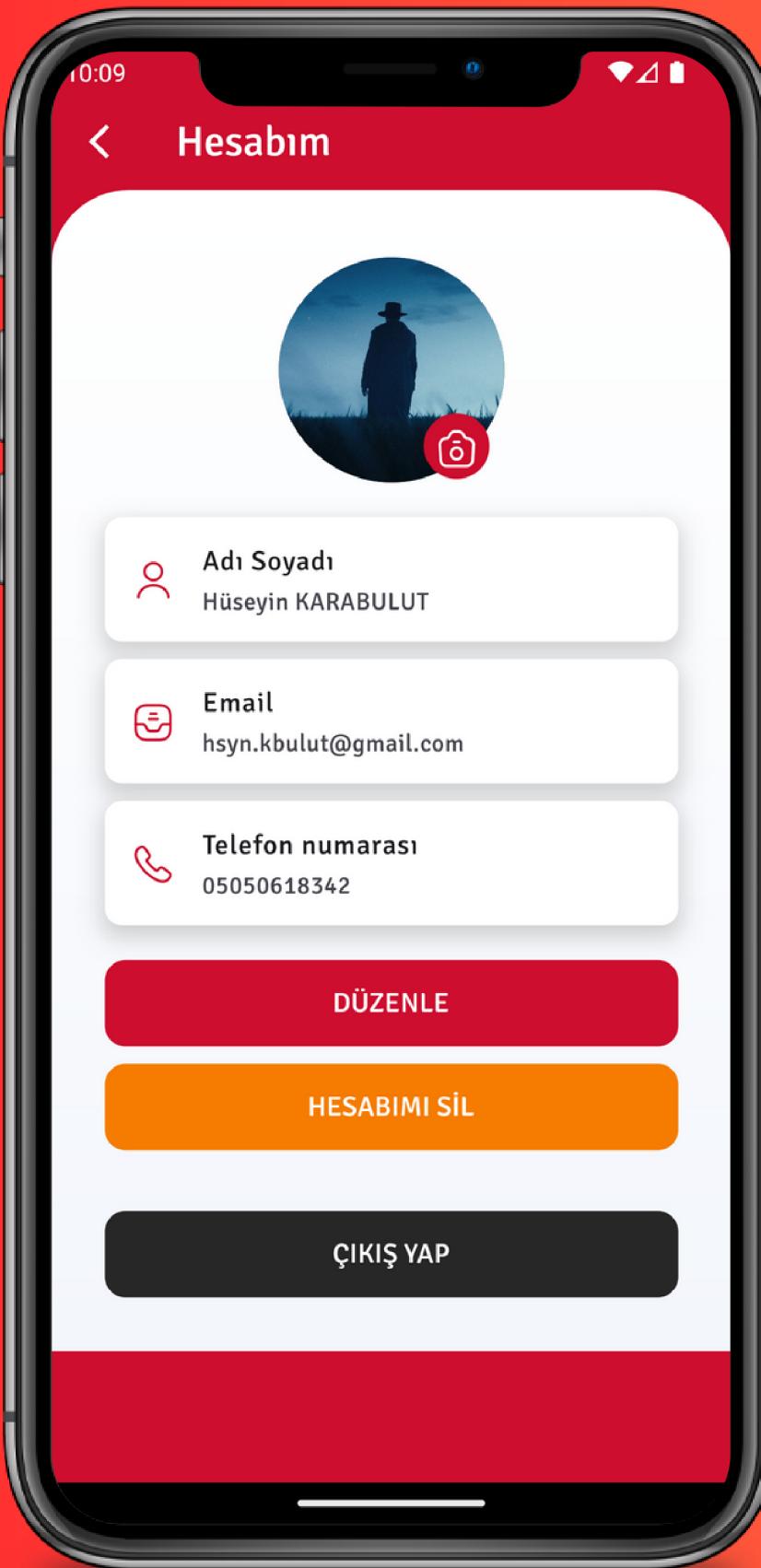


İlan Sil Ekranı

Bu sayfada Ev Sahibi, kendi açmış olduğu tüm ilanları listeleyip bunlar arasından istediği herhangi bir ilanı silebiliyor.

Hesabım Ekranı

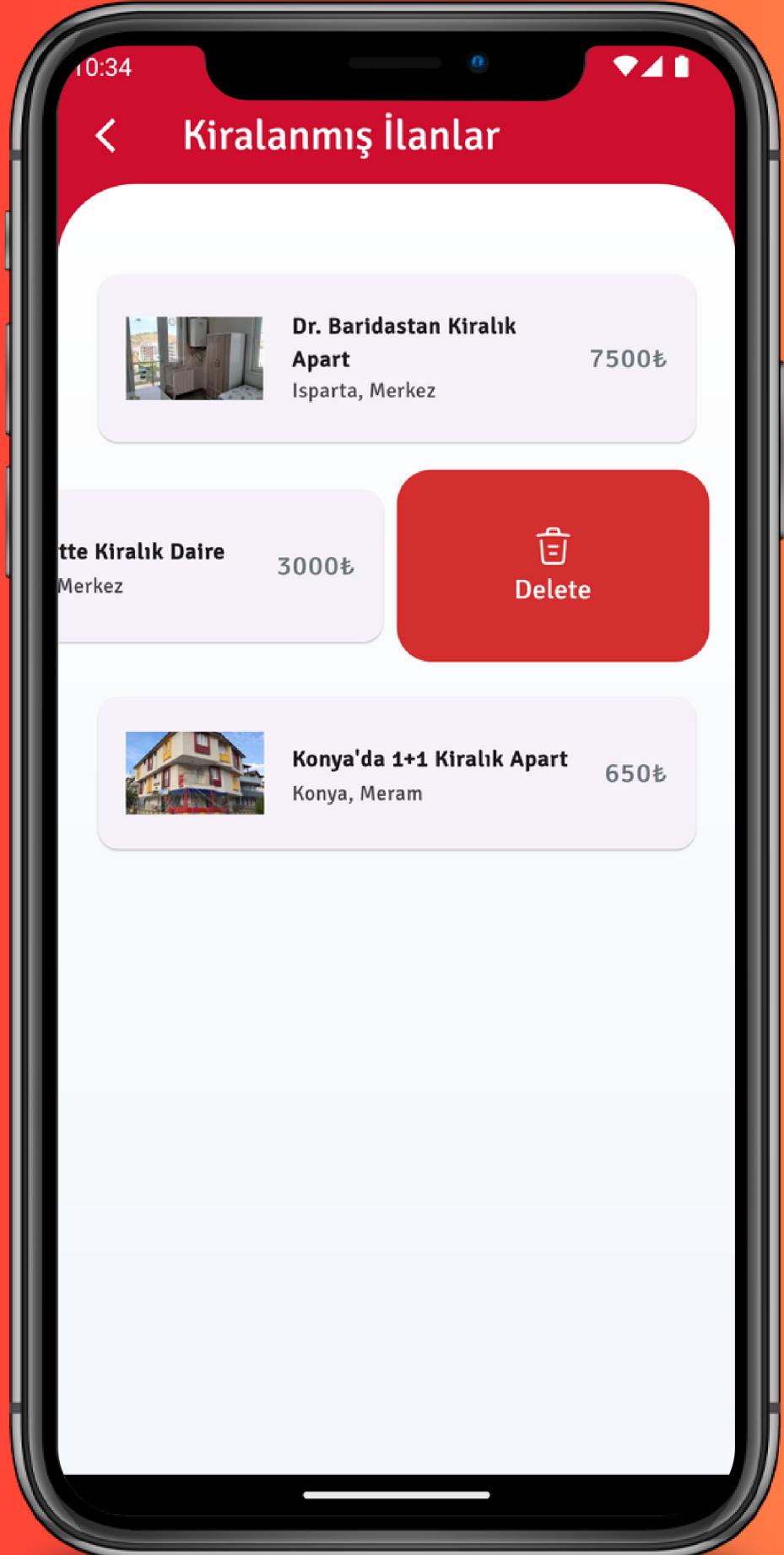
Bu sayfada Ev Sahibi, kendi kişisel bilgilerini görüntüleyebiliyor ve profil resmini zoomlayarak görebildiği gibi bu ekran üzerinden profil resmini değiştirebiliyor. Ayrıca Hesabımı Sil adlı butona tıklaması durumunda hesabını tamamen silebiliyor. Düzenle butonuna tıkladığı zaman Profil Düzenleme sayfası açılıyor ve buradan da istediği bilgilerini güncelleyebiliyor.





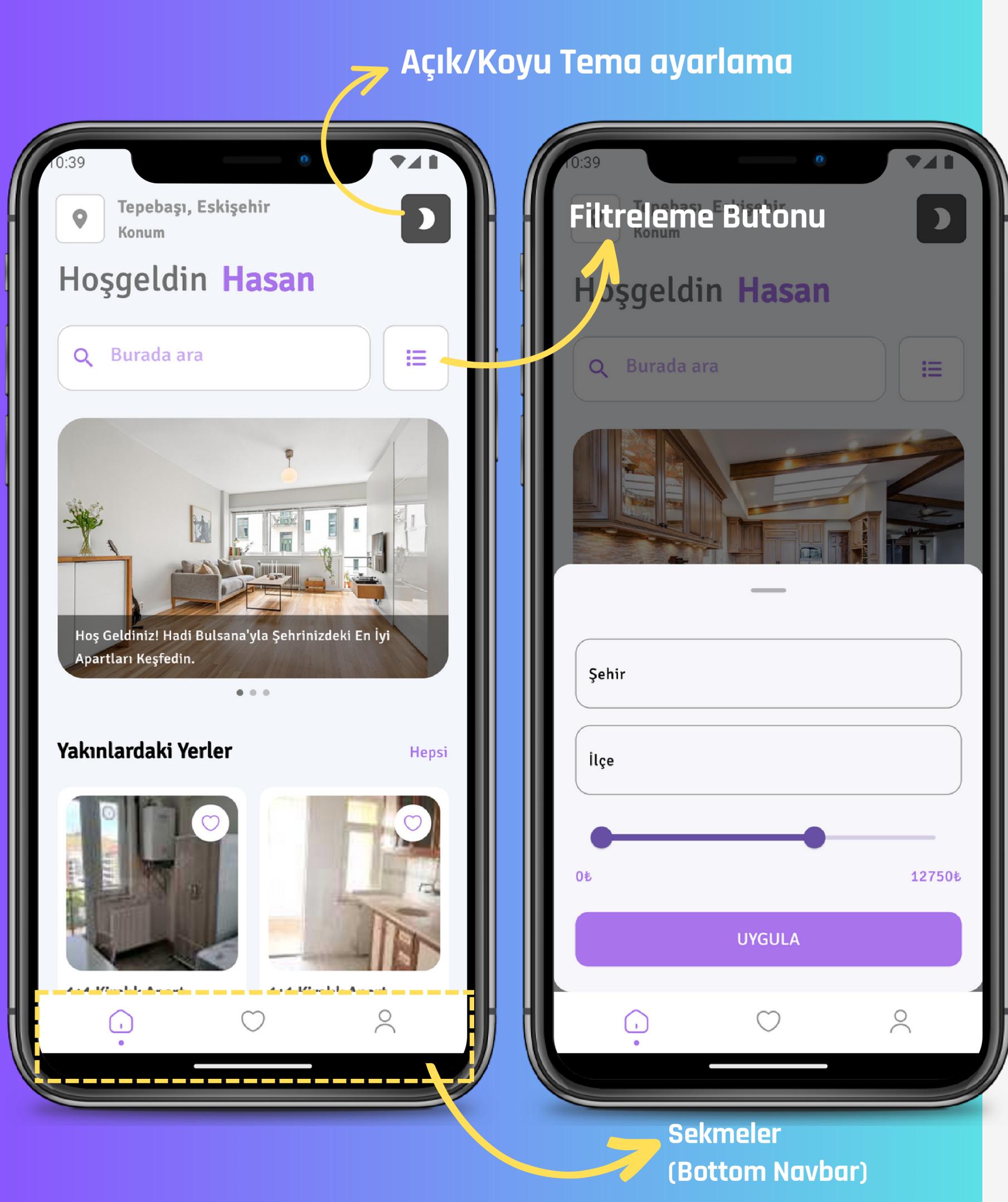
Hakkımızda Ekrani

Bu sayfada Ev Sahibi, uygulamamız hakkında detaylı bilgiye ulaşabilir ve hakkında bilgi edinebilir.



Kiralanmış İlanlar Ekranı

Bu sayfada Ev Sahibi, kendisine ait ev ilanlarından birisinin herhangi bir kiracı tarafından kiralanması durumunda Kiralanmış olan ilanlarını bu sayfada görüntüleyebilir ve ilgili ilan kiralanmış olduğu için ilanı listesinden silebilir.



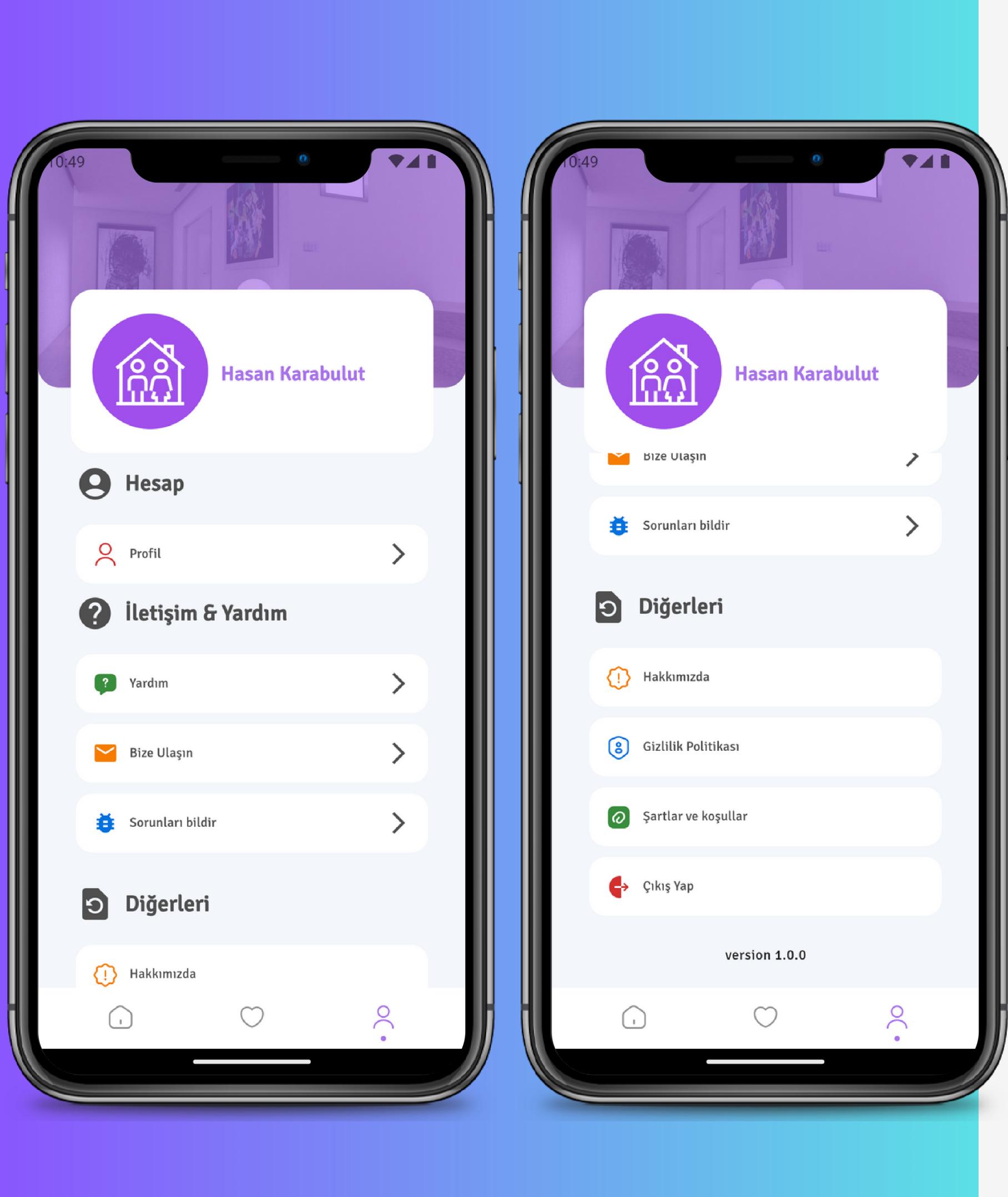
Kiracı Anasayfa Ekranı

Bu sayfada Kiracı, tüm ev ilanlarını görüntüleyebileceğ gibi filtrelemeyi kullanarak ilanları şehir, ilçe ve fiyat aralığına göre filtreleyip de görüntüleyebilir. Ayrıca ilanlar arasından ilgisini çeken bir ilana tıklaması durumunda ilgili ilanın detay sayfasına ulaşabilir.



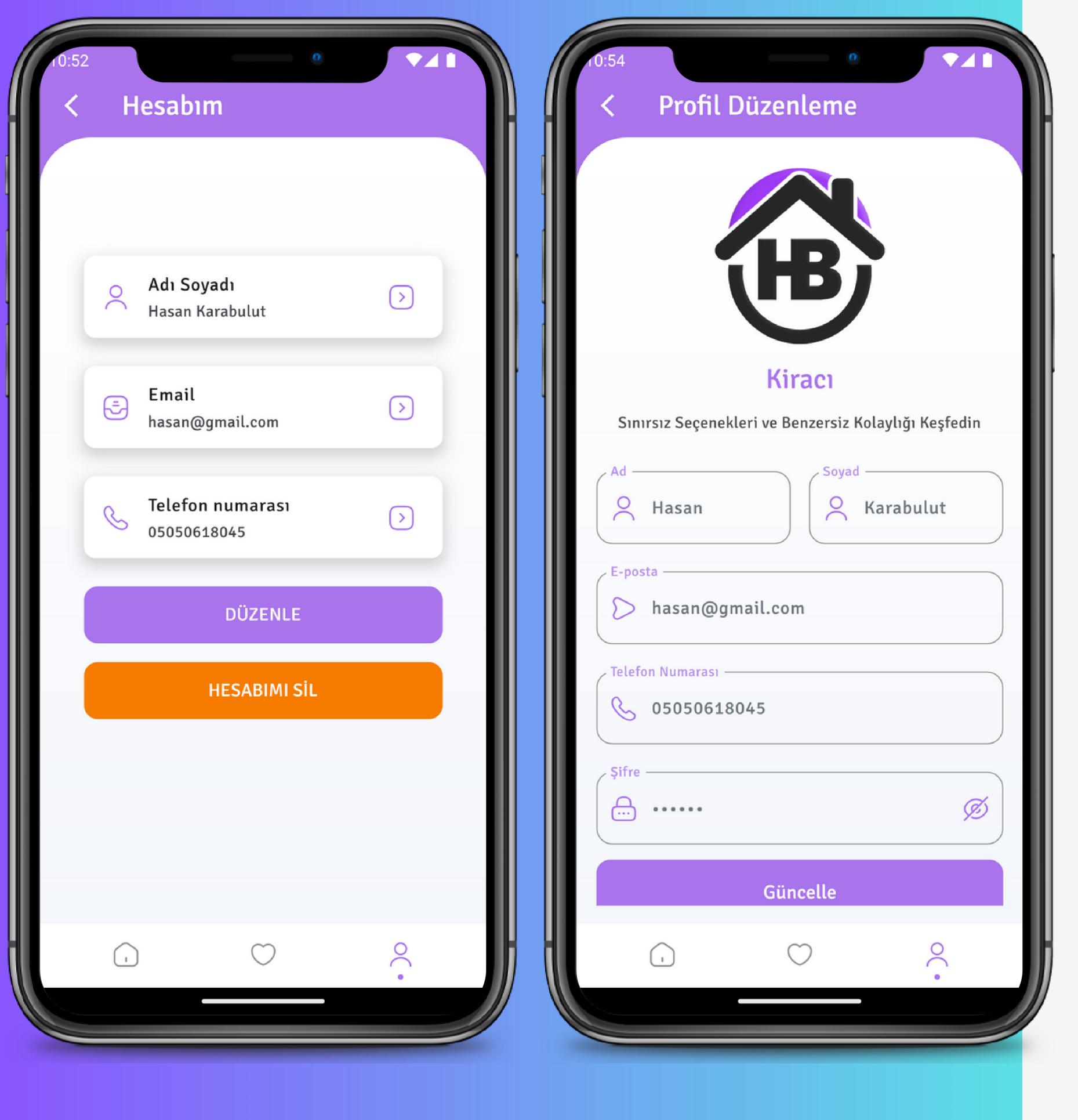
Kiracı İlan Detayı Ekranı

Bu sayfada Kiracı, ilanlar arasından ilgisini çeken bir ilana tıklaması durumunda ilgili ilanın detay sayfasına ulaşabilmekte ve buradan ilan sahibinin iletişim bilgilerini görüntüleyebilmektedir. Ayrıca Kirala adlı butona tıklaması durumunda ilgili ilanın butonunda Kirala yerine Kiralandı yazmakta ve bu kiralanan ilan, ev sahibinin Kiralanmış İlanlar listesine düşmektedir.



Kiracı Hesabım Ekranı

Bu sayfada Kiracı, kendi hesabını görüntüleyebiliyor ve çeşitli modüllerini kullanabiliyor. Ad soyad bilgisini vs. görebiliyor. Ayrıca burada diğer modüllerin hepsinin raporda tek tek gösterilmesine gerek duyulmadı.



Kiracı Profil Ekranı

Bu sayfada Kiracı, kendi kişisel bilgilerini görüntüleyebilir ve Hesabımı Sil adlı butona tıklayarak hesabını tamamen silebilir. Ayrıca Düzenle adlı butona tıklayarak Profil Düzenleme sayfasına ulaşabilir ve burada istediği bilgilerini değiştirmek için güncelleyebilir.



Hakkımızda Ekrani

Bu sayfada Kiracı, uygulamamız hakkında detaylı bilgiye ulaşabilir ve hakkımızda bilgi edinebilir.

Model Sınıflarının Yazılması

Projemde Ev Sahibi ve Kiracı adlı model sınıflarım için öncelikle IUsers adında bir interface oluşturuyorum ve bu arayüzü uygulayan Users adında ana bir model sınıf yazıyorum ve her iki kullanıcının ortak özelliklerini burada belirtiyorum.

```
// IUsers Interface
abstract class IUsers {
    String? get userID;
    String get name;
    String get surname;
    String get email;
    String get phoneNumber;
    String get password;
    Map<String, dynamic> toMap();
}

// Users Class
class Users implements IUsers {
    @override
    String? userID;
    @override
    String name;
    @override
    String surname;
    @override
    String email;
    @override
    String phoneNumber;
    @override
    String password;

    Users({
        this.userID,
        required this.name,
        required this.surname,
        required this.email,
        required this.phoneNumber,
        required this.password,
    });

    @override
    Map<String, dynamic> toMap() {
        return {
            'userID': userID,
            'name': name,
            'surname': surname,
            'email': email,
            'phoneNumber': phoneNumber,
            'password': password,
        };
    }
}
```

HomeOwner (Ev Sahibi) Model Sınıfının Yazılması

Users adındaki ana model sınıfını miras alan HomeOwner adında ev sahibi için bir model sınıfı oluşturuyorum. Ek olarak burada profilePhoto adında bir attribute ekliyorum model sınıfıma.

```
class HomeOwner extends Users {  
    String? profilePhoto;  
  
    HomeOwner({  
        required String userID,  
        required String name,  
        required String surname,  
        required String email,  
        required String phoneNumber,  
        required String password,  
        this.profilePhoto,  
    }) : super(  
        userID: userID,  
        name: name,  
        surname: surname,  
        email: email,  
        phoneNumber: phoneNumber,  
        password: password,  
    );  
  
    // HomeOwner sınıfını Map'e dönüştürme  
    @override  
    Map<String, dynamic> toMap() {  
        Map<String, dynamic> map = super.toMap();  
        //map['homeOwnerID'] = homeOwnerID;  
        map['profilePhoto'] = profilePhoto;  
        return map;  
    }  
  
    // Map'i HomeOwner sınıfına dönüştürme  
    HomeOwner.fromMap(Map<String, dynamic> map)  
        : //homeOwnerID = map['homeOwnerID'],  
        profilePhoto = map['profilePhoto'],  
        super(  
            userID: map['userID'],  
            name: map['name'],  
            surname: map['surname'],  
            email: map['email'],  
            phoneNumber: map['phoneNumber'],  
            password: map['password'],  
        );  
}
```

Tenant (Kiracı) Model Sınıfının Yazılması

Users adındaki ana model sınıfını miras alan Tenant adında kiracı için bir model sınıfı oluşturuyorum. Ek olarak burada city ve district adında birkaç attribute ekliyorum model sınıfıma.

```
class Tenant extends Users {
    String city;
    String district;

    Tenant({
        String? userID,
        required String name,
        required String surname,
        required String email,
        required String phoneNumber,
        required String password,
        required this.city,
        required this.district,
    }) : super(
        userID: userID,
        name: name,
        surname: surname,
        email: email,
        phoneNumber: phoneNumber,
        password: password,
    );

    @override
    Map<String, dynamic> toMap() {
        Map<String, dynamic> map = super.toMap();
        map['city'] = city;
        map['district'] = district;
        return map;
    }

    // Map'i HomeOwner sınıfına dönüştürme
    Tenant.fromMap(Map<String, dynamic> map)
        : city = map['city'],
        district = map['district'],
        super(
            userID: map['userID'],
            name: map['name'],
            surname: map['surname'],
            email: map['email'],
            phoneNumber: map['phoneNumber'],
            password: map['password'],
        );
}
```

HouseListing (Ev İlanı) Model Sınıfının Yazılması

HouseListing adında ev ilanları için bir model sınıfı oluşturuyorum.

```
class HouseListing {  
    String? houseListingID;  
    int price, buildingNo, flatNo;  
    String title, description, room, livingRoom, kitchen, city, district, neighborhood,  
    street, alley, apartmentName, phoneNumber, email;  
    List<String>? photos;  
    bool isRented;  
  
    HouseListing({  
        required this.houseListingID,  
        required this.title,  
        required this.description,  
        required this.price,  
        required this.room,  
        required this.livingRoom,  
        required this.kitchen,  
        required this.city,  
        required this.district,  
        required this.neighborhood,  
        required this.street,  
        required this.alley,  
        required this.apartmentName,  
        required this.buildingNo,  
        required this.flatNo,  
        required this.phoneNumber,  
        required this.email,  
        this.photos,  
        required this.isRented,  
    });  
  
    Map<String, dynamic> toMap() {  
        return {  
            'houseListingID': houseListingID,  
            'title': title,  
            'description': description,  
            'price': price,  
            'room': room,  
            'livingRoom': livingRoom,  
            'kitchen': kitchen,  
            'city': city,  
            'district': district,  
            'neighborhood': neighborhood,  
            'street': street,  
            'alley': alley,  
            'apartmentName': apartmentName,  
            'buildingNo': buildingNo,  
            'flatNo': flatNo,  
            'phoneNumber': phoneNumber,  
            'email': email,  
            'photos': photos,  
            'isRented': isRented,  
        };  
    }  
  
    HouseListing.fromMap(Map<String, dynamic> map) :  
        houseListingID = map['houseListingID'],  
        title = map['title'],  
        description = map['description'],  
        price = map['price'],  
        room = map['room'],  
        livingRoom = map['livingRoom'],  
        kitchen = map['kitchen'],  
        city = map['city'],  
        district = map['district'],  
        neighborhood = map['neighborhood'],  
        street = map['street'],  
        alley = map['alley'],  
        apartmentName = map['apartmentName'],  
        buildingNo = map['buildingNo'],  
        flatNo = map['flatNo'],  
        phoneNumber = map['phoneNumber'],  
        email = map['email'],  
        photos = List<String>.from(map['photos']),  
        isRented = map['isRented'];  
}
```

Authentication Service

Sınıflarının Yazılması

Projemde Ev Sahibi ve Kiracı adlı model sınıflarım için öncelikle IAuthenticationService adında bir interface oluşturuyorum ve bu arayüzü uygulayan AuthenticationService adında ana bir authentication sınıf yazıyorum ve her iki kullanıcının ortak fonksiyonlarını burada belirtiyorum.

```
abstract class IAuthenticationService {  
    Future<String?> signUpUser({required String name, required String surname, required String email, required String phoneNumber, required String password, String? city, String? district, File? profilePhoto});  
    Future<String?> signInUser({required String email, required String password});  
    Future<void> signOutUser();  
    Future<void> resetPasswordUser(BuildContext context, String email);  
    Future<void> deleteUser();  
  
    Future<void> updateUserInformation({  
        required String newName,  
        required String newSurname,  
        required String newEmail,  
        required String newPhoneNumber,  
        required String newPassword,  
        String? newCity,  
        String? newDistrict,  
        File? profilePhoto,  
    });  
}  
  
class AuthenticationService implements IAuthenticationService {  
    @override  
    Future<String?> signUpUser({required String name, required String surname, required String email, required String phoneNumber, required String password, String? city, String? district, File? profilePhoto}) async {  
        return null;  
    }  
  
    @override  
    Future<String?> signInUser({required String email, required String password}) async {  
        return null;  
    }  
  
    @override  
    Future<void> signOutUser() async {}  
    @override  
    Future<void> resetPasswordUser(BuildContext context, String email) async {}  
    @override  
    Future<void> deleteUser() async {}  
    @override  
    Future<void> updateUserInformation({  
        required String newName,  
        required String newSurname,  
        required String newEmail,  
        required String newPhoneNumber,  
        required String newPassword,  
        String? newCity,  
        String? newDistrict,  
        File? profilePhoto,  
    }) async {}  
}
```

AuthServiceHomeOwner adlı Authentication Sınıfının Yazılması

AuthServiceHomeOwner adında AuthenticationService sınıfından miras alan bir Authentication Service sınıfı oluşturuyorum. Bu sınıf kullanıcının uygulamaya kayıt olma, giriş yapma, çıkış yapma, şifre-email değiştirme gibi Firebase'in Authentication hizmetini kullanan bir service sınıfıdır.

```
class AuthServiceHomeOwner extends AuthenticationService {
    final FirebaseAuth _auth = FirebaseAuth.instance;
    final HomeOwnerService _homeOwnerService = HomeOwnerService();
    final HouseListingService _houseListingService = HouseListingService();

    @override
    Future<String?> signUpUser({required String name, required String surname, required String email, required String phoneNumber, required String password, File? profilePhoto, String? city, String? district}) async {
        try {
            UserCredential userCredential =
                await _auth.createUserWithEmailAndPassword(email: email, password: password);
            HomeOwner homeOwner = HomeOwner(
                userID: userCredential.user!.uid,
                name: name,
                surname: surname,
                email: email,
                phoneNumber: phoneNumber,
                password: password,
            );
            await _homeOwnerService.createUser(homeOwner, profilePhoto);
            Fluttertoast.showToast(msg: "Kayıt işlemi başarılı");

            return userCredential.user!.uid;
        } on FirebaseAuthException catch (e) {
            Fluttertoast.showToast(msg: "Uyarı: $e", toastLength: Toast.LENGTH_LONG);
            print("Uyarı mesajı: $e");
        }
        return null;
    }

    @override
    Future<String?> signInUser({required String email, required String password}) async {
        try {
            UserCredential userCredential = await _auth.signInWithEmailAndPassword(
                email: email,
                password: password,
            );
            Fluttertoast.showToast(msg: "Giriş işlemi başarılı", toastLength: Toast.LENGTH_SHORT);
            return userCredential.user!.uid;
        } on FirebaseAuthException catch (e) {
            String errorMessage = CustomErrorMessage.getLoginErrorMessage(e.code);
            Fluttertoast.showToast(msg: "Uyarı: $errorMessage", toastLength: Toast.LENGTH_LONG);
        }
        return null;
    }

    @override
    Future<void> signOutUser() async {
        try {
            await _auth.signOut();
        } catch (e) {
            print(e.toString());
        }
    }

    @override
    Future<void> resetPasswordUser(BuildContext context, String email) async {
        final navigator = Navigator.of(context);
        try {
            await _auth.sendPasswordResetEmail(email: email);
            Fluttertoast.showToast(
                msg: "E-posta adresinize bir şifre sıfırlama isteği gönderildi",
                toastLength: Toast.LENGTH_LONG,
            );
            navigator.pop();
        } on FirebaseAuthException catch (e) {
            String errorMessage =
                CustomErrorMessage.getForgotPasswordErrorMessage(e.code);
            Fluttertoast.showToast(msg: "Uyarı: $errorMessage", toastLength: Toast.LENGTH_LONG);
        }
    }
}
```

```
@override
Future<void> deleteUser() async {
    try {
        String uid = _auth.currentUser!.uid;
        await _houseListingService.deleteHouseListingsOfUser(uid);
        await _auth.currentUser!.delete();
        await _homeOwnerService.deleteUser(uid);

        Fluttertoast.showToast(
            msg: "Kullanıcı hesabı başarıyla silindi.",
            toastLength: Toast.LENGTH_LONG,
        );
        await signOutUser();
    } catch (e) {
        print("HATA: ${e.toString()}");
        Fluttertoast.showToast(
            msg: "Kullanıcı hesabını silearken bir hata oluştu.",
            toastLength: Toast.LENGTH_LONG,
        );
    }
    return;
}

@Override
Future<void> updateUserInformation(
    required String newName,
    required String newSurname,
    required String newEmail,
    required String newPhoneNumber,
    required String newPassword,
    String? newCity,
    String? newDistrict,
    File? profilePhoto) async {
    try {
        User? currentUser = _auth.currentUser;
        if (currentUser != null) {
            await currentUser.updateEmail(newEmail);
            await currentUser.updatePassword(newPassword);
            HomeOwner? homeOwner = await _homeOwnerService.getUser();

            if (homeOwner != null) {
                homeOwner.name = newName;
                homeOwner.surname = newSurname;
                homeOwner.email = newEmail;
                homeOwner.phoneNumber = newPhoneNumber;
                homeOwner.password = newPassword;
                await _homeOwnerService.updateUser(homeOwner, profilePhoto);
            }
        }
    } on FirebaseAuthException catch (e) {
        String errorMessage =
            CustomErrorMessage.getUpdateUserErrorMessage(e.code);
        Fluttertoast.showToast(msg: "Uyarı: $errorMessage", toastLength: Toast.LENGTH_LONG);
    }
}
```

NOT: Aynı işlemler AuthServiceTenant için de geçerli olduğundan onu ayrıca göstermeye gerek duymadım.

Service Sınıflarının Yazılması

Projemde Ev Sahibi ve Kiracı adlı model sınımlarım için şimdi de IUserService adında bir interface oluşturuyorum ve bu arayüzü uygulayan UserService adında ana bir service sınıfı yazıyorum ve her iki kullanıcının ortak fonksiyonlarını burada belirtiyorum.

```
abstract class IUserService<T extends Users> {
    Future createUser(T user, [File? profilePhoto]);
    Future updateUser(T user, [File? newProfilePhoto]);
    Future<void> deleteUser(String userID);
    Future<T?> getUser();
}

class UserService<T extends Users> implements IUserService<T> {
    @override
    Future createUser(T user, [File? profilePhoto]) async {
        throw UnimplementedError();
    }

    @override
    Future<void> deleteUser(String userID) {
        throw UnimplementedError();
    }

    @override
    Future<T?> getUser() {
        throw UnimplementedError();
    }

    @override
    Future updateUser(T user, [File? newProfilePhoto]) {
        throw UnimplementedError();
    }
}
```

HomeOwnerService adlı Service Sınıfının Yazılması

HomeOwnerService adında UserService sınıfımdan miras alan bir Service Service sınıfı oluşturmuyorum. Bu sınıf ev sahibi olan kullanıcının bilgilerini Firebase'in sunmuş olduğu Firestore Database adlı veritabanına kaydetmek, veritabanından kaydını silmek, güncellemek, bütün ev sahiplerini getirmek, belirli bir ev sahibini getirmek gibi temel CRUD işlemlerini gerçekleştiren Service sınıfıdır.

```
class HomeOwnerService extends UserService<HomeOwner> {
    final FirebaseFirestore _firestore = FirebaseFirestore.instance;
    final FirebaseAuth _auth = FirebaseAuth.instance;
    final FirebaseStorage _storage = FirebaseStorage.instance;

    @override
    Future createUser(HomeOwner homeOwner, [File? profilePhoto]) async {
        try {
            String uid = _auth.currentUser!.uid;
            DocumentReference userRef = _firestore.collection('homeowners').doc(uid);
            await userRef.set(homeOwner.toMap());
            if (profilePhoto != null) {
                String photoURL = await uploadProfilePhoto(uid, profilePhoto);
                await userRef.update({'profilePhoto': photoURL});
            }
        } catch (e) {
            print(e.toString());
        }
    }

    @override
    Future updateUser(HomeOwner user, [File? newProfilePhoto]) async {
        try {
            String uid = _auth.currentUser!.uid;
            await _firestore.collection('homeowners').doc(uid).update(user.toMap());
            if (newProfilePhoto != null) {
                String photoURL = await uploadProfilePhoto(uid, newProfilePhoto);
                await _firestore.collection('users').doc(uid).update({'profilePhoto': photoURL});
            }
        } catch (e) {
            print(e.toString());
        }
    }
}
```

```
@override
Future<void> deleteUser(String userID) async {
    try {
        await _firestore.collection('homeowners').doc(userID).delete();
        await deleteProfilePhoto(userID);
    } catch (e) {
        print(e.toString());
    }
}

@Override
Future<HomeOwner?> getUser() async {
    try {
        String uid = _auth.currentUser!.uid;
        DocumentSnapshot doc = await _firestore.collection('homeowners').doc(uid).get();
        if (doc.exists) {
            HomeOwner homeOwner = HomeOwner.fromMap(doc.data() as Map<String, dynamic>);
            return homeOwner;
        } else {
            return null;
        }
    } catch (e) {
        print(e.toString());
        return null;
    }
}

Future<String> uploadProfilePhoto(String userID, File photo) async {
    try {
        Reference ref = _storage.ref().child('homeowners').child(userID).child('profilePhoto.jpg');
        UploadTask task = ref.putFile(photo);
        await task.whenComplete(() {});
        String photoURL = await ref.getDownloadURL();
        return photoURL;
    } catch (e) {
        print(e.toString());
        throw Exception('Failed to upload profile photo.');
    }
}

Future<void> deleteProfilePhoto(String userID) async {
    try {
        Reference ref = _storage.ref().child('homeowners').child(userID).child('profilePhoto.jpg');
        await ref.delete();
    } catch (e) {
        print(e.toString());
    }
}
```

NOT: Aynı işlemler TenantService için de geçerli olduğundan onu ayrıca göstermeye gerek duymadım.

İlan Ekleme Sayfasının Tasarım Kodu

Burada ev sahibinin ilan açmak için İlan Ekleme sayfasını kullanması gerekmektedir. İlan ekleme sayfasının tasarım kodu çok uzun olduğu için bir kısmı yanda verilmiştir.

```
class AddHouseListingForm extends StatefulWidget {
  const AddHouseListingForm({super.key});
  @override
  State<AddHouseListingForm> createState() => _AddHouseListingFormState();
}

class _AddHouseListingFormState extends State<AddHouseListingForm> {
  final AddHouseListViewModel viewModel = AddHouseListViewModel();
  bool isAddListing = false;

  void _createHouseListing(BuildContext context) async {
    setState(() { isAddListing = true; });
    await viewModel.createHouseListing(context);
    setState(() { isAddListing = false; });
  }

  @override
  Widget build(BuildContext context) {
    return Form(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          HostTextFormField(
            keyboardType: TextInputType.text,
            labelText: TTexts.listingTitle,
            prefixIcon: Iconsax.textalign_center,
            maxLength: 30,
            controller: viewModel.titleController,
            maxLines: 1,
          ),
          const SizedBox(height: TSizes.spaceBtwInputFields),
          HostTextFormField(
            keyboardType: TextInputType.text,
            labelText: TTexts.listingDescription,
            prefixIcon: Iconsax.document_text,
            maxLength: 400,
            minLines: 4,
            maxLines: 4,
            showCounter: true,
            controller: viewModel.descriptionController,
          ),
          const SizedBox(height: TSizes.spaceBtwInputFields),
          HostTextFormField(
            keyboardType: TextInputType.number,
            labelText: TTexts.listingPrice,
            prefixIcon: Iconsax.money5,
            maxLength: 10,
            controller: viewModel.priceController,
            maxLines: 1,
          ),
          const SizedBox(height: TSizes.spaceBtwInputFields),
          HostTextFormField(
            keyboardType: TextInputType.phone,
            labelText: TTexts.phoneNo,
            prefixIcon: Iconsax.call,
            maxLength: 11,
            controller: viewModel.phoneNumberController,
            maxLines: 1,
          ),
          const SizedBox(height: TSizes.spaceBtwInputFields),
          SizedBox(
            width: double.infinity,
            child: ElevatedButton(
              onPressed: () {
                _createHouseListing(context);
              },
              style: Theme.of(context).elevatedButtonTheme.style!.copyWith(
                backgroundColor: MaterialStateColor.resolveWith(
                  (states) => AppColors.primaryColor2),
              ),
              child: Container(
                width: double.infinity,
                height: 20.0,
                alignment: Alignment.center,
                child: Text(TTexts.createListingTitle,
                  textAlign: TextAlign.center,
                  style: Theme.of(context).textTheme.titleLarge),
              ),
            ),
          );
        ],
      );
    }
}
```

İlan Güncelleme Sayfasının Tasarım Kodu

Burada ev sahibinin daha önce açmış olduğu ilan bilgisini değiştirip güncellemek için İlan Güncelleme sayfasını kullanması gerekmektedir. İlan güncelleme sayfasının tasarım kodu çok uzun olduğu için bir kısmı yanda verilmiştir.

```
class UpdateHouseListingForm extends StatefulWidget {
  final HouseListing houseListing;
  final UpdateHouseListViewModel viewModel;

  const UpdateHouseListingForm({
    super.key,
    required this.houseListing,
    required this.viewModel,
  });
  @override
  State<UpdateHouseListingForm> createState() => _UpdateHouseListingFormState();
}

class _UpdateHouseListingFormState extends State<UpdateHouseListingForm> {
  void initState() {
    widget.viewModel.titleController.text = widget.houseListing.title;
    widget.viewModel.priceController.text = widget.houseListing.price.toString();
    widget.viewModel.phoneNumberController.text = widget.houseListing.phoneNumber;
  }

  @override
  Widget build(BuildContext context) {
    return Form(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          HostTextFormField(
            keyboardType: TextInputType.text,
            labelText: TTexts.listingTitle,
            prefixIcon: Iconsax.textalign_center,
            maxLength: 30,
            controller: widget.viewModel.titleController,
            maxLines: 1,
          ),
          HostTextFormField(
            keyboardType: TextInputType.number,
            labelText: TTexts.listingPrice,
            prefixIcon: Iconsax.money5,
            maxLength: 10,
            controller: widget.viewModel.priceController,
            maxLines: 1,
          ),
          HostTextFormField(
            keyboardType: TextInputType.phone,
            labelText: TTexts.phoneNo,
            prefixIcon: Iconsax.call,
            maxLength: 11,
            controller: widget.viewModel.phoneNumberController,
            maxLines: 1,
          ),
          Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text(
                'FOTOGRAF',
                style: Theme.of(context).textTheme.headlineSmall!.copyWith(
                  color: dark ? AppColors.whiteColor : AppColors.dark,
                  fontFamily: 'SignikaNegative',
                  fontWeight: FontWeight.bold,
                ),
              ),
              Divider(
                thickness: 2,
                color: dark ? AppColors.whiteColor : AppColors.grey,
              ),
              SizedBox(
                height: height * 0.32,
                width: width * 0.9,
                child: PickImagesUpdateScreen(viewModel: widget.viewModel),
              ),
            ],
          ),
          SizedBox(
            width: double.infinity,
            child: ElevatedButton(
              onPressed: () {
                widget.viewModel.updateHouseListing(context, widget.houseListing);
              },
              child: Container(
                width: double.infinity,
                height: 28.0,
                alignment: Alignment.center,
                child: Text(TTexts.createListingTitle,
                  textAlign: TextAlign.center,
                  style: Theme.of(context).textTheme.titleLarge),
              ),
            ),
          );
        ],
      ),
    );
  }
}
```

İlan Silme Sayfasının Tasarım Kodu

Burada ev sahibinin daha önce açmış olduğu kendisine ait olan tüm ilanlar listelenir ve ev sahibi bunlardan istediği herhangi bir ilanı siler. İşte bu kısmın kullanıcı gözünden nasıl gözüktüğünün tasarım kodu yanda gösterilmiştir. Lakin tasarım kodu çok uzun olduğu için bir kısmı verilmiştir.

```
class DeleteHouseListingScreen extends StatefulWidget {
  const DeleteHouseListingScreen({Key? key}) : super(key: key);
  @override
  _DeleteHouseListingScreenState createState() => _DeleteHouseListingScreenState();
}

class _DeleteHouseListingScreenState extends State<DeleteHouseListingScreen> {
  List<HouseListing> _houseListings = [];
  final HouseListingService _houseListingService = HouseListingService();

  @override
  void initState() {
    super.initState();
    getHouseListings();
  }

  Future<void> getHouseListings() async {
    try {
      List<HouseListing> houseListings = await _houseListingService.getUserHouseListings();
      setState(() { _houseListings = houseListings; });
    } catch (e) {
      print(e.toString());
    }
  }

  Future<void> deleteHouseListing(int index) async {
    try {
      HouseListing deletedHouseListing = _houseListings[index];
      await _houseListingService.deleteHouseListing(deletedHouseListing.houseListingID!);
      await _houseListingService.deletePhotos(deletedHouseListing.houseListingID!);
      setState(() { _houseListings.removeAt(index); });
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('İlan silindi.')),
      );
    } catch (e) {
      print(e.toString());
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Container(
        padding: const EdgeInsets.only(top: 33.0),
        decoration: BackgroundGradient().buildGradient2(),
        child: Padding(
          padding: EdgeInsets.symmetric(horizontal: width * 0.04, vertical: height * 0.01),
          child: ListView.builder(
            itemCount: _houseListings.length,
            itemExtent: 120,
            itemBuilder: (context, index) {
              return Slidable(
                startActionPane: ActionPane(motion: const StretchMotion(), children: [
                  SlidableAction(
                    borderRadius: BorderRadius.all(Radius.circular(20)),
                    backgroundColor: AppColors.success, icon: Iconsax.information, label: 'Detay',
                    onPressed: (context) => {},
                  ),
                  endActionPane: ActionPane(motion: const BehindMotion(), children: [
                    SlidableAction(borderRadius: BorderRadius.all(Radius.circular(20)),
                      backgroundColor: AppColors.error, icon: Iconsax.trash, label: 'Delete',
                      onPressed: (context) => {
                        showDialog(context: context, builder: (BuildContext context) {
                          return AlertDialog(title: const Text('İlanı Sil'), content: const Text(
                            'Bu ilanı silmek istediyinizden emin misiniz? Bu işlem geri alınamaz.'),
                          actions: [
                            TextButton(onPressed: () { Navigator.pop(context); }, child: const Text('Vazgeç'),),
                            TextButton(onPressed: () { deleteHouseListing(index); Navigator.pop(context); },
                              child: const Text('Sil')),
                          ],
                        );
                      });
                  ]),
                ],
              ),
            ),
            child: Card(
              margin: EdgeInsets.symmetric(horizontal: width * 0.02, vertical: height * 0.014),
              child: ListTile(dense: true,
                contentPadding: EdgeInsets.symmetric(vertical: 12, horizontal: 16),
                leading: Image.network(_houseListings[index].photos![0], width: 80, height: 80, fit: BoxFit.cover),
                title: Text(_houseListings[index].title, style: TextStyle(fontWeight: FontWeight.bold)),
                subtitle: Text("${_houseListings[index].city}, ${_houseListings[index].district}"),
                trailing: Text('${_houseListings[index].price}€', style: Theme.of(context).textTheme.bodyLarge),
              ),
            ),
          ),
        ),
      ),
    );
  }
}
```

Bütün İlanların Anasayfada Liste şeklinde gösterilmesinin Tasarım Kodu

Burada bütün ev sahiplerinin daha önce açmış olduğu bütün ilanları liste şeklinde Kiracılara gösteren sayfaının tasarım kodu vardır. Lakin tasarım kodu çok uzun olduğu için bir kısmı verilmiştir.

```
class ExploreCard extends StatefulWidget {
  final List<HouseListing> houseListings;
  const ExploreCard({Key? key, required this.houseListings}) : super(key: key);

  @override
  State<ExploreCard> createState() => _ExploreCardState();
}

class _ExploreCardState extends State<ExploreCard> {
  List<HouseListing> _houseListings = [];

  @override
  void didUpdateWidget(covariant ExploreCard oldWidget) {
    super.didUpdateWidget(oldWidget);
    if (oldWidget.houseListings != widget.houseListings) {
      setState(() {
        _houseListings = List.from(widget.houseListings);
      });
    }
  }

  Future<void> getHouseListings() async {
    try {
      setState(() {
        _houseListings = widget.houseListings;
      });
    } catch (e) {
      print(e.toString());
    }
  }

  @override
  Widget build(BuildContext context) {
    return GridView.builder(
      physics: const NeverScrollableScrollPhysics(), shrinkWrap: true,
      gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
        crossAxisCount: 2, crossAxisSpacing: 12, mainAxisSpacing: 12, mainAxisExtent: 270),
      itemCount: _houseListings.length,
      itemBuilder: (_, index) {
        HouseListing populars = _houseListings[index];
        return GestureDetector(
          onTap: () => Navigator.push(context, MaterialPageRoute(builder: (_)> ProductDetailsScreen(houseListing: populars))),
          child: Container(
            padding: const EdgeInsets.all(8),
            decoration: BoxDecoration(borderRadius: BorderRadius.circular(12.0), color: dark ? AppColors.darkerGrey : AppColors.whiteColor),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.start, crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Stack(alignment: Alignment.topRight,
                  children: [
                    ClipRRect(
                      borderRadius: BorderRadius.circular(15),
                      child: populars.photos != null && populars.photos!.isNotEmpty
                        ? Image(fit: BoxFit.cover, height: 100, image: NetworkImage(populars.photos![0]))
                        : const Padding(padding: EdgeInsets.all(8.0),
                          child: Placeholder(color: AppColors.primaryColor, fallbackHeight: 120, fallbackWidth: 30),
                        ),
                    ),
                    Padding(
                      padding: const EdgeInsets.all(8),
                      child: Align(
                        alignment: Alignment.bottomRight,
                        child: Container(
                          padding: const EdgeInsets.all(8),
                          decoration: BoxDecoration(
                            color: AppColors.whiteColor,
                            borderRadius: BorderRadius.circular(20),
                          ),
                          child: const Icon(Icons.heart, size: 18, color: AppColors.primaryColor),
                        ),
                      ),
                    ),
                  ],
                ),
                Text(
                  '${populars.room}+${populars.livingRoom} Kiralik Apart',
                  style: Theme.of(context).textTheme.headlineSmall!.copyWith(fontWeight: FontWeight.bold, fontSize: 15, color: dark ? AppColors.whiteColor : AppColors.darkerGrey),
                ),
                Row(
                  children: [
                    Icon(Icons.location_pin, color: dark ? AppColors.whiteColor : AppColors.darkGrey),
                    Text(
                      '${populars.city}, ${populars.district}',
                      style: TextStyle(color: dark ? AppColors.whiteColor : AppColors.darkGrey),
                    ),
                  ],
                ),
                Text(
                  '${populars.price} TL/aylık',
                  style: TextStyle(color: dark ? AppColors.ratingColor : AppColors.primaryColor, fontWeight: FontWeight.w700),
                ),
              ],
            ),
          );
        );
      },
    );
  }
}
```

Teşekkürler

Rapor sunumumu incelediğiniz için teşekkür ederim.
Daha fazla detay kodlarda mevcuttur.

Saygılarımla,
2112721044 Hüseyin KARABULUT