

T.C.
SAKARYA ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BSM 498 BİTİRME ÇALIŞMASI

“HOMEY” APART VE YURT UYGULAMASI

G171210303 – Furkan KOÇ

Fakülte Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ
Tez Danışmanı : Prof. Dr. Ümit KOCABIÇAK

2020-2021 Bahar Dönemi

T.C.
SAKARYA ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

“HOMEY” APART VE YURT SİSTEMLERİ

BSM 498 - BİTİRME ÇALIŞMASI

Furkan KOÇ

Fakülte Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ

Bu tez .. / .. / ... tarihinde aşağıdaki jüri tarafından oybirliği / oyçokluğu ile kabul edilmiştir.

.....
Jüri Başkanı

.....
Üye

.....
Üye

ÖNSÖZ

Bu projeye başlamadan önce zorlanacağımı biliyordum ama bu kadar zorlu olacağını tahmin edememiştim. Yeni teknolojiler öğrenmek ne kadar zevkli bir şey olsa da bazı zamanlar oldukça fazla can sıkıcı olabiliyor. Araştırmalar yapmak, makaleler okumak, eğitim videoları izlemek ve bunların sonunda öğrendiğimiz şeyleri bir araya getirip ortaya bir proje çıkartmak gerçekten zorlu ve sabır isteyen bir iş. Yeri geliyor 5 dakika da öğrenebildiğimiz şeyler oluyor yeri geliyor saatlerce ufacık bir yerde takılıp kalıyoruz. Ama ortaya çıkan projenin sonucuna baktığım zaman tam olarak istediğim yere gelemesem bile güzel bir iş çıkardığımı düşünmenin mutluluğunu yaşıyorum. Yeni teknolojiler öğrenmenin verdiği mutlulukta cabası. Anlıyorum ki öğrenecek daha çok şey var ve ben daha bu yolun çok başındayım. Bu projeden kazandığım deneyimlerle birlikte istediğim zaman çalışarak, çabalayarak ortaya güzel işler çıkartabileceğimi görmüş oldum. Artık daha kararlı ve istekli şekilde yoluma devam edeceğim.

İÇİNDEKİLER

ÖNSÖZ.....	iii
İÇİNDEKİLER.....	iv
ŞEKİLLER LİSTESİ.....	v
ÖZET.....	vii

BÖLÜM 1.

GİRİŞ.....	8
------------	---

BÖLÜM 2.

KULLANILAN TEKNOLOJİLER.....	8
2.1. Java : Spring Boot.....	8
2.1.1. Controller.....	9
2.1.2. Exception.....	11
2.1.3. Jwt (Json Web Token).....	12
2.1.4. Model.....	13
2.1.4.1 Entity.....	14
2.1.4.2. Dto.....	15
2.1.4.3. Mapper.....	16
2.1.4.4. Type.....	17
2.1.5. Repository.....	17
2.1.6. Service.....	18
2.2. Maven.....	19
2.3. Hibernate.....	20
2.4. Lombok.....	20
2.5. Spring Data JPA.....	21
2.6. RestFUL.....	22
2.7. Spring Mail.....	22
2.8. JavaScript.....	23

2.9. React-Native.....	23
2.10. PostgreSQL.....	24
BÖLÜM 3.	
UYGULAMA.....	24
BÖLÜM 4.	
SONUÇLAR VE ÖNERİLER.....	42
KAYNAKLAR.....	43
ÖZGEÇMİŞ.....	44
BSM 498 BİTİRME ÇALIŞMASI DEĞERLENDİRME VE SÖZLÜ SINAV TUTANAĞI.....	45

ŞEKİLLER LİSTESİ

Şekil 2.1.	Controller.....	10
Şekil 2.2.	Base Controller.....	11
Şekil 2.3.	Exception.....	12
Şekil 2.4.	Authentication Exception.....	12
Şekil 2.5.	Token Service.....	13
Şekil 2.6.	Entity.....	14
Şekil 2.7.	Dto.....	15
Şekil 2.8.	Mapper.....	16
Şekil 2.9.	Type.....	17
Şekil 2.10.	Repository.....	17
Şekil 2.11.	Service.....	19
Şekil 2.12.	Mail Service.....	22

Şekil 3.1.	Giriş Ekranı.....	25
Şekil 3.2.	Giriş Ekran Kodu.....	26
Şekil 3.3.	Giriş Ekranı onLogin Fonksiyonu.....	27
Şekil 3.4.	Auth Services Login Fonksiyonu.....	27
Şekil 3.5.	Paths.js Dosyası.....	28
Şekil 3.6.	Kayıt Ol Ekranı.....	28
Şekil 3.7.	Kayıt Ol Ekranı Kodu.....	29
Şekil 3.8.	Kayıt Ol Ekranı createUser Fonksiyonu.....	30
Şekil 3.9.	Yönetici Ana Sayfa Ekranı.....	31
Şekil 3.10.	Yönetici Ana Sayfa Ekranı Token Kaydetme Fonksiyonu...	32
Şekil 3.11.	Yönetici Apart Ekleme Ekranı.....	33
Şekil 3.12.	Yönetici Apart Listeleme Ekranı.....	33
Şekil 3.13.	Yönetici Kullanıcıya Kod Gönderme Ekranı.....	34
Şekil 3.14.	Mail.....	34
Şekil 3.15.	Yönetici Eşya Ekleme Ekranı.....	35
Şekil 3.16.	Yönetici Şikayetleri Görme Ekranı.....	36
Şekil 3.17.	Yönetici Şikayet Detay Görme Ekranı.....	36
Şekil 3.18.	Yönetici Çözülmüş Şikayetleri Görme Ekranı.....	37
Şekil 3.19.	Kullanıcı Profil Ekranı.....	37
Şekil 3.20.	Kullanıcı Aparta Kayıt Olma Ekranı.....	38
Şekil 3.21.	Kullanıcı Şikayet Oluşturma Ekranı.....	39
Şekil 3.22.	Bildirim.....	39
Şekil 3.23.	Kullanıcı Oluşturduğu Şikayetleri Görme Ekranı.....	40
Şekil 3.24.	Kullanıcı Eşya Listeleme Ekranı.....	40
Şekil 3.25.	Kullanıcı Tarih Seçme Ekranı.....	41
Şekil 3.26.	Kullanıcı Saat Aralığı Seçme Ekranı.....	41
Şekil 3.27.	Kullanıcı Aparta Kayıt Olma Ekranı.....	38
Şekil 3.28.	Kullanıcı Şikayet Oluşturma Ekranı.....	39
Şekil 3.29.	Bildirim	39

ÖZET

Anahtar kelimeler: React-Native, Java:Spring Boot, Mobil Uygulama, Apart, Yurt

Günümüzde birçok insan, genellikle öğrenci olanlar apartlarda ve yurtlarda kalmaktadır. Bu yerlerde kalırken bir şikayetleri olduğu zaman ya da oradaki araç gereçlerden birisini kullanmak istedikleri zaman sıkıntı yaşayabiliyorlardı. Geliştirdiğim uygulama ile birlikte apart ve yurtlarda kalan insanlar rahat bir şekilde şikayetlerini belirtip doğrudan yöneticiye dertlerini ulaştırbilecekler ve aynı zamanda sınırlı sayıda olan araç gereçleri örneğin bir çamaşır makinesini tarih ve saat belirleyerek rezervasyon oluşturup kullanabilecekler.

Geliştirdiğim mobil uygulamanın veritabanı olarak PostgreSQL kullandım. Mobil uygulama kısmında ise React-Native teknolojisini kullandım. Uygulamanın backendini de Java : Spring Boot ile geliştirdim. Uygulamayı geliştirirken güvenliğe önem verdiği için Spring Boot Security teknolojisini de kullandım. Bu teknoloji sayesinde uygulamayı kullanan kişiler giriş yaparken bir token alıyor ve bu token ile uygulama içerisindeki diğer işlemlerini gerçekleştiriyor. Token olmayan istekler backendde gittiğinde hiçbir şekilde servislere ulaşamıyor.

1. GİRİŞ

“Homey” yurt ve apartlarda rahatlıkla şikayetlerimizi belirtebileceğimiz ve oradaki sınırlı sayıdaki araç gereçler için rezervasyon oluşturarak kendi belirlediğimiz bir saatte kullanmamızı sağlayan bir mobil uygulamadır.

Projemi Code First metoduyla yaptım, bu mobil uygulamayı yaparken database tarafında PostgreSQL kullandım. Backend kısmında Java:Spring Boot, Maven, Jpa, Hibernate, Lombok, Spring Security, Spring Mail, JWT(Json Web Token) ve RESTful teknolojilerini kullandım. Frontend’ı yaparken ise Javascript ve React-Native teknolojilerini kullandım.

Bu projeyi yapmada belirttiğim teknolojileri seçmemin sebebi bu teknolojilerin birbirleriyle çok verimli ve uyumlu bir şekilde kullanıcı dostu olarak çalışabilmesidir. İçerideki teknolojiler sayesinde yazmak için çok fazla uğraşmamız gereken fonksiyonları metodlar halinde çağrıabiliyor ve kullanabiliyoruz.

Bundan sonraki bölümlerde kullanılan teknolojiler detaylı bir şekilde anlatılacak ve projenin bir uygulaması gösterilecektir.

2. KULLANILAN TEKNOLOJİLER

2.1. Java: Spring Boot

Resmi proje sayfasındaki tanıma göre Spring Boot bağımsız(stand-alone) Spring tabanlı uygulamalar geliştirmenizi kolaylaştırıyor yarar. Spring Boot’u bu kadar popüler yapan şey ise otomatik konfigürasyon özelliği. Çok az Spring konfigürasyonu ile çoğu Spring Boot projesini hayatı geçirebilirsiniz.

Spring Boot’un bazı temel özellikleri

- Bağımsız(stand-alone) Spring uygulamaları oluşturabilmek.

- Gömülü bir web sunucusu(Tomcat, Jetty, Undertow) ile gelmesi.
- Build konfigürasyonunu kolaylaştmak için sağladığı starter'lar.
- Otomatik konfigürasyon.
- Kod üretimi(code generation) ve XML konfigürasyona ihtiyaç duymaması.

Yukarıda kısaca Spring Boot'dan bahsettim. Şimdi şunu söylemeliyim ki Spring Boot da bir proje oluşturmak çok kolay. <https://start.spring.io/> sitesinden istediğimiz konfigürasyonlara göre bir spring projesi oluşturabiliriz. Burada bir Maven projesi yapmak istediğimi Java dilini kullanabileceğimi ve istediğimiz bağımlılıkları(dependency) seçebiliyoruz. Bağımlılıklar kısmında kullanacağımız teknolojileri yani PostgreSQL, Lombok, JPA, Hibernate ve RESTful'u seçerek bir projeyi hemen hayata geçirebiliriz. [4]

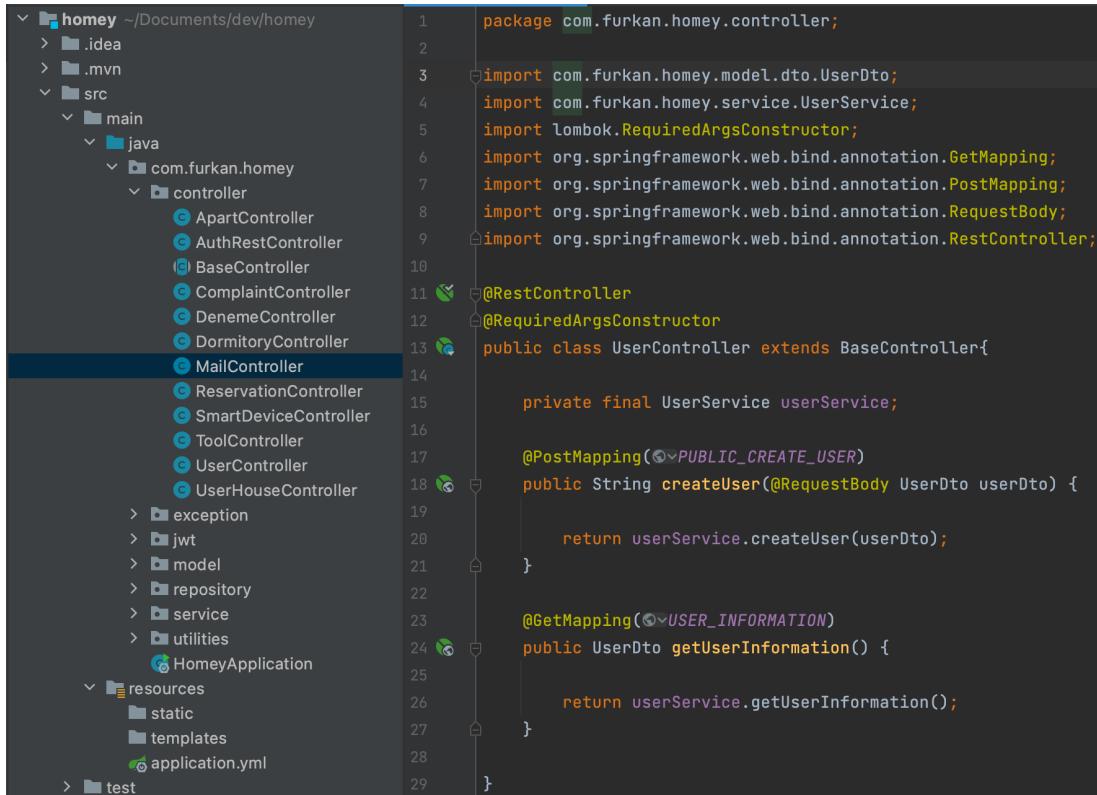
Spring Boot içerisinde kullandığım katmanlar ise şöyle:

- Controller
- Exception
- Jwt (Json Web Token)
- Model
 - o Entity
 - o Dto
 - o Mapper
 - o Type
- Repository
- Service

Şimdi gelelim bu katmanların detaylı anlatımına.

2.1.1. Controller

Frontend'den gelen ilk istek controllerimizda karşılanır ve buradan service katmanına yönlendirilir.



The screenshot shows a Java project structure on the left and the corresponding code for a controller on the right.

Project Structure:

- homey (~/Documents/dev/homey)
 - .idea
 - .mvn
 - src
 - main
 - java
 - com.furkan.homey
 - controller
 - ApartController
 - AuthRestController
 - BaseController
 - ComplaintController
 - DenemeController
 - DormitoryController
 - MailController
 - ReservationController
 - SmartDeviceController
 - ToolController
 - UserController
 - UserHouseController
 - exception
 - jwt
 - model
 - repository
 - service
 - utilities
 - HomeyApplication
 - resources
 - static
 - templates
 - application.yml
 - test

Code (UserController.java):

```

1 package com.furkan.homey.controller;
2
3 import com.furkan.homey.model.dto.UserDto;
4 import com.furkan.homey.service.UserService;
5 import lombok.RequiredArgsConstructor;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.PostMapping;
8 import org.springframework.web.bind.annotation.RequestBody;
9 import org.springframework.web.bind.annotation.RestController;
10
11 @RestController
12 @RequiredArgsConstructor
13 public class UserController extends BaseController{
14
15     private final UserService userService;
16
17     @PostMapping("PUBLIC_CREATE_USER")
18     public String createUser(@RequestBody UserDto userDto) {
19
20         return userService.createUser(userDto);
21     }
22
23     @GetMapping("USER_INFORMATION")
24     public UserDto getUserInformation() {
25
26         return userService.getUserInformation();
27     }
28
29 }

```

Şekil 2.1. Controller

Fotoğrafta da görüldüğü gibi bazı annotationlarımız bulunmaktadır. `@RestController` bu class'ın bir controller olduğunu belirtiyor. `@RequiredArgsConstructor` class'ımızın içerisindeki constructorları otomatik olarak oluşturuyor. Göründüğü gibi `@PostMapping` ile belirlediğimiz path üzerinden isteklerimiz ilk olarak controller'a düşüyor. Burada önemli olan şey ise gelen istek ile bizim belirlediğimiz istek aynı değişkenlere sahip olmalıdır buna `UserDto` örneğini verebiliriz ve `UserDto`'yu yazının ilerleyen bölümlerinde anlatacağım. Controller içerisinde Service'imizi tanımlayıp sonra bu Service'e istek atarak bize gelen bilgileri Service katmanına iletiyoruz. Burada `BaseController`dan kalıtım alıyoruz ve `BaseController` içerisinde path adreslerimiz yer alıyor.

```

package com.furkan.homey.controller;

public abstract class BaseController {

    protected static final String LOGIN = "/Login";
    protected static final String USER_INFORMATION = "/user";
    protected static final String PUBLIC_CREATE_USER = "/public/create/user";
    protected static final String CREATE_APART = "/apart";
    protected static final String GET_ALL_APART = "get/all/apart";
    protected static final String CREATE_DORMITORY = "/dormitory";
    protected static final String REGISTER_USER_HOUSE = "/userhouse";
    protected static final String CREATE_COMPLAINT = "/complaint";
    protected static final String GET_ALL_COMPLAINT = "/get/all/complaint";
    protected static final String GET_ALL_COMPLAINT_MANAGER = "/get/all/manager/complaint";
    protected static final String SEND_MAIL = "/sendmail";
    protected static final String SEND_UUID_FOR_APART = "/send/uuid/for/apart";
    protected static final String CREATE_TOOL = "/tool";
    protected static final String CREATE_RESERVATION = "/reservation";
    protected static final String GET_RESERVATIONS = "/get/reservations";
    protected static final String GET_TOOLS = "/get/tools";
    protected static final String SMART_DEVICE = "/smart/device";
    protected static final String COMPLAINT_TYPE_WORKING_ON = "/complaint/working/on";
    protected static final String COMPLAINT_TYPE_SOLVED = "/complaint/solved";
}

```

Şekil 2.2. Base Controller

2.1.2. Exception

Exception katmanımızda uygulamamıza özel hatalar yazıp eğer bir hata olursa kendi belirlediğimiz hata kodu ve mesajı ile istek atılan yere dönüş yapılmasını sağlıyoruz.

Aşağıdaki fotoğrafta da gördüğümüz üzere RunTimeExceptiondan kalıtım alarak uygulamamıza özel bir Exception sınıfı oluşturuyoruz. Burada exceptionlarımızın içinde neler olabileceğini belirtebiliyoruz. Daha sonrasında uygulamıza özel Exception sınıfından kalıtım alarak Exceptionlarımızı yazıyoruz.

```

public class HomeyException extends RuntimeException{

    private int code;
    private String message;
    private String description;
    private String[] fields;
    private String[] placeHolders;
    private HttpStatus httpStatus;

    protected static final DateTimeFormatter DATE_TIME_FORMATTER = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm");
    protected static final DateTimeFormatter DATE_FORMATTER = DateTimeFormatter.ofPattern("dd-MM-yyyy");

    public HomeyException(int code, String message, String description, String[] fields, String[] placeHolders, HttpStatus httpStatus) {
        super(message);
        this.code = code;
        this.message = message;
        this.description = description;
        this.fields = fields;
        this.placeHolders = placeHolders;
        this.httpStatus = httpStatus;
    }

    public HomeyException(int code, String[] fields, String[] placeHolders) {
        this(code, null, null, fields, placeHolders, HttpStatus.UNPROCESSABLE_ENTITY);
    }

    public HomeyException(int code, String[] fields) {
        this(code, null, null, fields, null, HttpStatus.UNPROCESSABLE_ENTITY);
    }

    public HomeyException(int code) { this(code, null, null, null, null, HttpStatus.UNPROCESSABLE_ENTITY); }

    public HomeyException(int code, HttpStatus httpStatus) { this(code, null, null, null, null, httpStatus); }
}

```

Şekil 2.3. Exception

Aşağıdaki fotoğrafta gördüğümüz gibi bir exception oluşturabiliyoruz ve bu Exception ile ilgili bir sorun oluştuğunda ilgili yerde “throw new ExceptionName” şeklinde hata fırlatma yapabiliyoruz.

```

package com.furkan.homey.exception;

import com.furkan.homey.utilities.exception.HomeyException;

public class AuthenticationRequiredException extends HomeyException implements IErrorCode {

    public AuthenticationRequiredException() {super(AUTH_REQUIRED, null, null);}
}

```

Şekil 2.4. Authentication Exception

2.1.3. Jwt (Json Web Token)

Jwt’nin projemizde çok önemli bir görevi var bu katmanda güvenliği sağlıyoruz. Bu katmanda kullanıcıların kullanıcı adı ve şifre ile birlikte backende attıkları istekler

icin bir token oluşturuluyor ve kullanıcıya geri dönderiliyor. Burada oluşturulan tokenların ne kadar süre aktif kalabileceğini ve token olmadan istek atılabilecek servisler (örneğin kullanıcı oluşturma servisi) gibi şeyleri de belirleyebiliyoruz.

```

private String SECRET_KEY = "furkan_koc";

// verilen token ait kullanıcı adını döndürür.
public String extractUsername(String token) { return extractClaim(token, Claims::getSubject); }

// verilen token ait token bitiş süresini verir.
public Date extractExpiration(String token) { return extractClaim(token, Claims::getExpiration); }

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
    final Claims claims = extractAllClaims(token);
    return claimsResolver.apply(claims);
}

// verilen token ait claims bilgisini alır.
private Claims extractAllClaims(String token) {
    return Jwts.parser().setSigningKey(SECRET_KEY).parseClaimsJws(token).getBody();
}

// token in geçerlilik süre doludu mu?
private Boolean isTokenExpired(String token) { return extractExpiration(token).before(new Date()); }

// userDetails objesini alır. createToken metoduna gönderir.
public String generateToken(UserDetails userDetails) {
    Map<String, Object> claims = new HashMap<>();
    return createToken(claims, userDetails.getUsername());
}

private String createToken(Map<String, Object> claims, String subject) {
    return Jwts.builder().setClaims(claims)
        .setSubject(subject) // ilgili kullanıcı
        .setIssuedAt(new Date(System.currentTimeMillis())) // baslangic
        .setExpiration(new Date(System.currentTimeMillis() + 5 * 60 * 60 * 1000)) // bitis
        .signWith(SignatureAlgorithm.HS256, SECRET_KEY) // kullanılan algoritma ve bu algoritma çalışırken kullanılacak
        .compact();
}

```

Şekil 2.5. Token Service

2.1.4. Model

Model katmanının entity kısmında projemizde kullanacağımız veritabanını gerçekleştiriyoruz ve dto kısmında ise frontendden gelecek istekte bulunacak olan verilerin neler olacağını belirtiyoruz. Daha sonrasında Mapperlar ile entity ve dto arasında dönüşüm işlemi yaptırabiliyoruz.

2.1.4.1. Entity

Bahsettiğim gibi veritabanı modelimizi bu katmanda gerçekleştiriyoruz.

Fotoğrafta görüleceği üzere yine bazı annotationlarımız var. `@Entity`'yi bu class'ın database de tablosunun olmasını sağlamak için kullanıyoruz. `@Getter` ve `@Setter` ise veritabanında olan kayıtlarımızı getlemek ve setlemek için kullanılır. `@Id` o değişkenin tablonun ID'si olduğunu gösterir. `@GeneretadValue` ise ID'nin nasıl üretilicegi işlemeye yarar burada birer birer artırmamasını istiyoruz. `@Size` veritabanında ayrılacak karakter uzunluğunu ayarlamamızı sağlar. `@OneToOne`, `@OneToMany`, `@ManyToOne` ve `@ManyToMany` ise tablolar arasındaki bağlantıların nasıl olduğunu ayarlamamıza yarar.

```

@Entity
@Getter
@Setter
@Table(name = "uuser")
public class User extends BaseEntity {

    private final static String SEQUENCE_NAME = "user_id";
    public final static String JOIN_COLUMN = "user_id";

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = SEQUENCE_NAME + "_gen")
    @SequenceGenerator(name = SEQUENCE_NAME + "_gen", sequenceName = SEQUENCE_NAME, allocationSize = 1)
    private Long id;

    private String name;

    private String surname;

    private String username;

    private String password;

    private String about;

    private String email;

    private String phone;

    @Enumerated(EnumType.STRING)
    private Role role;

    @OneToOne(fetch = FetchType.LAZY, mappedBy = "user")
    private Contact contact;

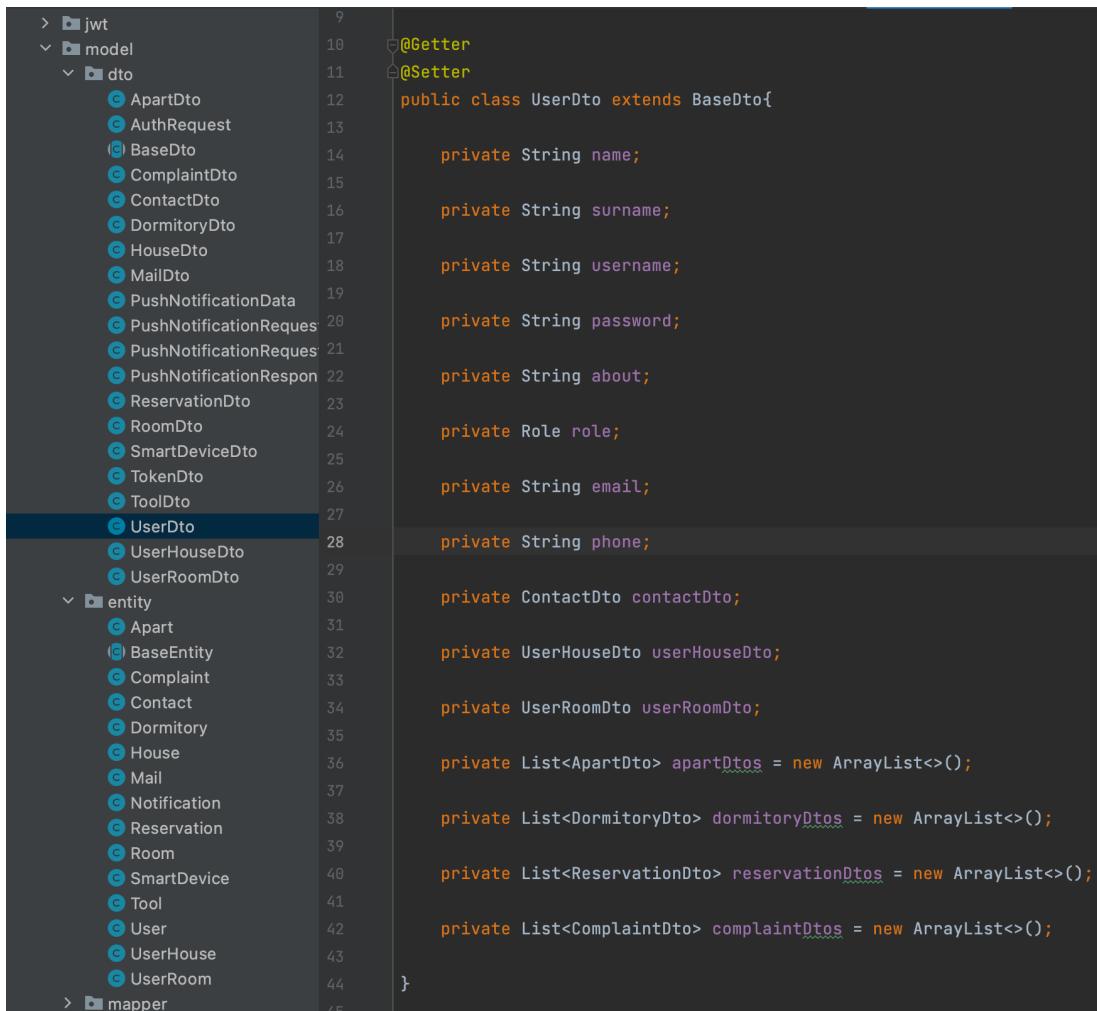
    @OneToOne(fetch = FetchType.LAZY, mappedBy = "user")
    private UserHouse userHouse;
}

```

Şekil 2.6. Entity

2.1.4.2. Dto

Dtos katmanında frontendden gelecek olan bilgilerin tutulacağı değişkenleri belirtiyoruz. Aynı zamanda geri dönüş değerini de dto olarak belirtirsek bu değerleri isteğin geldiği yere geri döndürebiliyoruz.



```

> jwt
  > model
    > dto
      C ApartDto
      C AuthRequest
      I BaseDto
      C ComplaintDto
      C ContactDto
      C DormitoryDto
      C HouseDto
      C MailDto
      C PushNotificationData
      C PushNotificationReques
      C PushNotificationReques
      C PushNotificationRespon
      C ReservationDto
      C RoomDto
      C SmartDeviceDto
      C TokenDto
      C ToolDto
      C UserDto
      C UserHouseDto
      C UserRoomDto
    > entity
      C Apart
      I BaseEntity
      C Complaint
      C Contact
      C Dormitory
      C House
      C Mail
      C Notification
      C Reservation
      C Room
      C SmartDevice
      C Tool
      C User
      C UserHouse
      C UserRoom
  > mapper
  
```

```

9
10  @Getter
11  @Setter
12  public class UserDto extends BaseDto{
13
14      private String name;
15
16      private String surname;
17
18      private String username;
19
20      private String password;
21
22      private String about;
23
24      private Role role;
25
26      private String email;
27
28      private String phone;
29
30      private ContactDto contactDto;
31
32      private UserHouseDto userHouseDto;
33
34      private UserRoomDto userRoomDto;
35
36      private List<ApartDto> apartDtos = new ArrayList<>();
37
38      private List<DormitoryDto> dormitoryDtos = new ArrayList<>();
39
40      private List<ReservationDto> reservationDtos = new ArrayList<>();
41
42      private List<ComplaintDto> complaintDtos = new ArrayList<>();
43
44  }
45
  
```

Şekil 2.7. Dto

Fotoğrafta görüldüğü üzere User için frontendden gelen bilgiler bu şekilde olmalı. `@Getter` ve `@Setter` ise service katmanında bunlara setleme ve getleme yapmaya yarar.

Dtos gelen veri için kullanılabileceği gibi istek atılan yere verilerin geri dönderileceği veri taşıma objeleri olarak da kullanılabilir o yüzden burada görülen bütün bilgilerin doldurulması zorunlu değildir, doldurulmayan değişkenlerin yerine otomatik olarak null ya da kendi belirleyebileceğiniz bir değer atanabilir.

2.1.4.3. Mapper

Mapperlarımız ise dto ve entity katmanları arasında dönüşüm yapmamızı sağlar. Entityden bir bilgi getirip dto ya dönüştürüp istek atan yere bu verimizi döndürebiliriz veya dto ile gelen bilgileri entity'ye çevirip veritabanımıza kaydedebiliriz.

```
public class UserMapper extends BaseMapper{

    public static UserDto mapTo(User entity) {
        if (entity == null) {
            return null;
        }
        UserDto dto = new UserDto();
        BaseMapper.mapToDto(dto, entity);

        dto.setAbout(entity.getAbout());
        dto.setName(entity.getName());
        dto.setRole(entity.getRole());
        dto.setSurname(entity.getSurname());
        dto.setPassword(entity.getPassword());
        dto.setUsername(entity.getUsername());
        dto.setEmail(entity.getEmail());
        dto.setPhone(entity.getPhone());

        return dto;
    }

    public static User mapTo(UserDto from, User to) {
        BaseMapper.mapToEntity(from, to);
        to.setAbout(from.getAbout());
        to.setName(from.getName());
        to.setRole(from.getRole());
        to.setSurname(from.getSurname());
        to.setPassword(from.getPassword());
        to.setUsername(from.getUsername());
        to.setStatus(from.getStatus());
        to.setEmail(from.getEmail());
        to.setPhone(from.getPhone());

        return to;
    }
}
```

Şekil 2.8. Mapper

2.1.4.4. Type

Bu katman sadece özel olarak oluşturduğumuz Enum ya da Interface'leri tutmamızı sağlar. Bu değerleri backendin istediğimiz bir yerinde istediğimiz şekilde kullanabiliriz ve işlemlerimizde kolaylıklar sağlayabiliriz.

```
public enum ReservationTime {

    TIME1( display: "09:00 - 12:00"), TIME2( display: "12:00-15:00"), TIME3( display: "15:00-18:00"),
    TIME4( display: "18:00-21:00"), TIME5( display: "21:00-24:00");

    private String display;

    private ReservationTime(String display) {this.display = display;}

    public String getDisplay() {return display;}
}
```

Şekil 2.9. Type

2.1.5. Repository

Repository katmanında database'imize ilgili işlemleri yapıyoruz.

Kalıtım aldığımız JPA Repository sayesinde kullanabildiğimiz hazır metotlar ile fazladan kod yazmaya gerek kalmadan CRUD (Create, Read, Update ve Delete) işlemlerini yaptırabiliyoruz. Eğer daha özel sorgulara ihtiyacımız var ise tabi ki bu sorguları da yazıp işlemlerimizi yaptırabiliriz.

```
public interface ApartRepository extends JpaRepository<Apart, Long> {

    List<Apart> findByUserId(Long userId);

    Apart findApartByUuid(String uuid);

    @Query("select a from Apart a where a.user.id =:userId")
    Apart findApartByUserId(Long userId);

}
```

Şekil 2.10. Repository

Fotoğrafta da göründüğü gibi kendimize özel sorguları @Query annotation'ını kullanarak yazabiliyoruz. Dönüş tipi(Fotoğrafta User ve Integer olarak belirtmişim) olarak istedigimize göre ayarlayabiliriz.

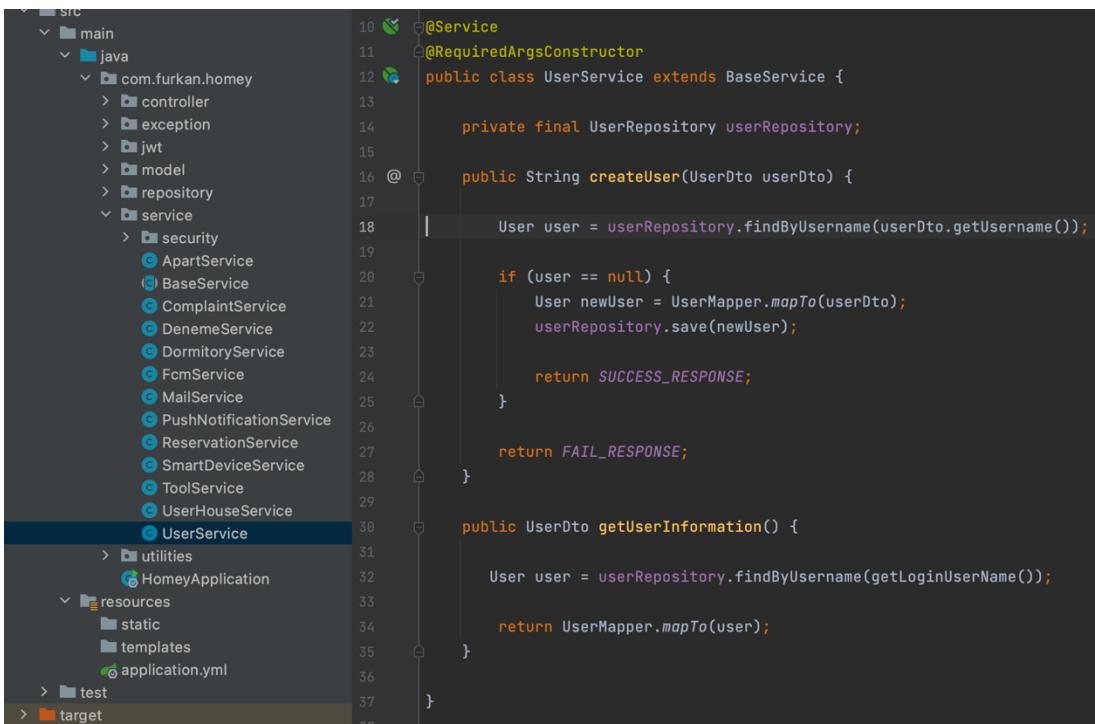
@Query annotation'ı olmadan da Jpa bizim isteklerimizi getirebilir yeter ki istediğimiz şeyleri onun istediği gibi anlamlı bir şekilde ifade edebilelim, fotoğrafta örnekleri mevcuttur.

2.1.6. Service

Service katmanı aslında en çok işi yaptığımız kısım diyebilirim. Bütün katmanları harmanlayarak kullandığımız yer Service katmanıdır. Controllerden gelen istege service katmanında istediğimiz işlemleri yapıp sonrasında controller'a geri gönderiyoruz.

@Service annotation'ı bu class'ın bir service olduğunu belirtiyor.

Fotoğrafta gördüğümüz gibi createUser metoduna gelen UserDto önce daha önce öyle bir kayıt olup olmadığı kontrol edilip eğer veritabanında aynı username ile kayıt yok ise kaydının yapılması sağlanıyor. Eğer aynı username ile bir kayıt varsa veritabanına kayıt yapılmayıp geriye FAIL_RESPONSE değeri döndürülüyor. Eğer user başarılı bir şekilde oluşturuldu ise SUCCESS_RESPONSE değeri geri döndürülüyor.



The screenshot shows a Java project structure in a code editor. The project is organized under 'src' with 'main' and 'java' folders. Inside 'java', there's a package 'com.furkan.homey' containing several service classes like ApartService, BaseService, ComplaintService, DenemeService, DormitoryService, FcmService, MailService, PushNotificationService, ReservationService, SmartDeviceService, ToolService, UserHouseService, and UserService. The 'UserService' class is highlighted. The code for 'UserService' includes methods for creating a user and getting user information, both utilizing a UserRepository.

```

10  @Service
11  @RequiredArgsConstructor
12  public class UserService extends BaseService {
13
14      private final UserRepository userRepository;
15
16      @Override
17      public String createUser(UserDto userDto) {
18
19          User user = userRepository.findByUsername(userDto.getUsername());
20
21          if (user == null) {
22              User newUser = UserMapper.mapTo(userDto);
23              userRepository.save(newUser);
24
25              return SUCCESS_RESPONSE;
26          }
27
28          return FAIL_RESPONSE;
29      }
30
31      public UserDto getUserInformation() {
32
33          User user = userRepository.findByUsername(getLoginUserName());
34
35          return UserMapper.mapTo(user);
36      }
37

```

Şekil 2.11. Service

Spring Boot'un katmanlarını bu şekilde özetleyebiliriz. Şimdi ise gelelim Springte kullandığımız teknolojileri kısaca anlatmaya.

2.2. Maven

Maven, POM(Project Object Model)'a dayanan güçlü bir proje yönetim aracıdır. Proje oluşturma, bağımlılık ve dokümantasyon için kullanılır. ANT gibi derleme işlemini basitleştirir. Ancak ANT'den çok daha ileri bir teknolojidir.

Kısaltası, Maven için herhangi bir Java tabanlı projeyi oluşturmak ve yönetmek için kullanılabilecek bir araçtır demek çokta yanlış olmayacağındır. Maven, Java geliştiricilerinin günlük işlerini kolaylaştırır ve genellikle herhangi bir Java tabanlı projenin anlaşılmasına yardımcı olur. Peki Maven'in yardımcı olduğu temel noktalar nelerdi?

- Projeleri kolayca build edebiliriz
- Hiç zorlanmadan JAR, WAR gibi istediğimiz paket formatında projeyi derleyebiliriz

- Projenin bağımlılıklarını kolayca yönetebiliriz
- Proje hakkında meta bilgilerini tutabılırız
- Projeleri Kaynak Kodu Yönetim Sistemlerine(Git, SVN) kolayca entegre edebiliriz [3]

2.3. Hibernate

Hibernate bir ORM Firework'udur.

Nesne tabanlı programlarda veritabanımızda kullanacağımız bir yapıdır Hibernate. Hibernate veritabanımıza iletişime geçen programımızdan her türlü işlemi yapmamızı sağlayan bir aracıdır. Neden Hibernate kullanmalıyız sorusuna ise yazım kolaylığı cevabını verebiliriz. Örneğin;

JDBC ile veritabanına kayıt eklerken kullandığımız yapı;

```
stmt.executeUpdate( "INSERT INTO musteri VALUES ('burak', 'kut bay')");
```

Hibernate ile kayıt eklerken ile şöyle yazmamız yeterli;

```
session.save(musteri);
```

Yazılım karmaşıklığından kurtarmakla birlikte düzgün bir yazılım yapısı kullanmamıza olanak sağlamaktadır. [5]

2.4.Lombok

Lombok, Java projesi geliştirirken IDE'ye entegre edilebilen bir anotasyon ile kod üretme (code generation) kütüphanesidir. Lombok ile daha temiz ve daha az kod yazmış oluruz.

Java'da proje geliştirirken yaygın olarak yapmamız gereken bazı işlemler bulunmaktadır. Bunlar projemizin iş tarafına gerçek bir değer getirmez iken kodumuzun çok fazla ayrıntı barındırmamasını zorunlu hale getiriyor.

Örnek olarak Bir POJOda (Plain Old Java Object) Sürekli Get-Set, Constructor, toString vb. methodları yazmamız gerekiyor. Birçok sınıfta olması gereken bu methodlar o kadar uzuyor ki sınıfların uzunluğu ve kod karmaşıklığı artıyor, clean code'dan uzaklaşıyoruz. Lombok sayesinde küçük anotasyonlar ile kodumuzu oldukça kısaltmış oluyoruz. [6]

2.5. Spring Data JPA

Spring Data, Spring kütüphanesinin bir parçasıdır. Spring Data JPA bir JPA sağlayıcısı değildir. JPA sağlayıcımızın (Hibernate gibi) üstüne ekstra bir soyutlama katmanı ekleyen bir kütüphane / çerçevedir.

Hibernate bir JPA uygulamasıdır, Spring Data JPA ise bir JPA veri erişimi soyutlamasıdır. Spring Data, GenericDAO uygulamalarına bir çözüm sunar. Spring Data JPA bir uygulama veya JPA sağlayıcısı değildir, veri erişim katmanlarını uygulamak için gerekli olan kaynak kodu miktarını önemli ölçüde azaltmak için kullanılan bir soyutlamadır. Hibernate, loose coupling avantajlarıyla ORM aracı olarak mükemmel bir seçimdir. Unutmayın, Spring Data JPA her zaman Hibernate veya Eclipse Link gibi JPA sağlayıcılarını gerektirir.

Spring Data JPA ile CRUD operasyonları için herhangi bir implementasyon yazmanız gereklidir. Örneğin;

```
repo.findAll();
```

Bu kod bloğu ile tablodaki tüm kayıtlar listeleneciktir. Eğer Hibernate kullanıyorsak, daha sonra repo.findAll () metodunun için, aşağıdaki gibi bir implementasyon yazmaları gereklidir. Örneğin;

```
Session session = this.sessionFactory.getCurrentSession();
List listUsers = session.createQuery("from
```

```
Customer").list();
return listUsers; [7]
```

2.6. RESTful

REST, client-server arasındaki haberleşmeyi sağlayan HTTP protokolü üzerinden çalışan bir mimaridir. REST ,servis yönelimli mimari üzerine oluşturulan yazılımlarda kullanılan bir transfer yöntemidir.İstemci ve sunucu arasında XML ve JSON verilerini taşıyarak uygulamanın haberleşmesini sağlar.REST mimarisini kullanan servislere ise RESTful servis denir. Bizim projemizde frontend ile backend arasındaki haberleşmeyi sağlar. [8]

2.7. Spring Mail

Spring framework email göndermek için birçok kütüphane sağlar. Spring, JavaMailSender adında kullanımı kolay mail interface i sağlar.

```
import com.furkan.homey.model.dto.MailDto;
import com.furkan.homey.model.entity.Apart;
import com.furkan.homey.model.entity.User;
import com.furkan.homey.repository.ApartRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import com.furkan.homey.repository.UserRepository;

@Service
@RequiredArgsConstructor
public class MailService extends BaseService{

    private final JavaMailSender javaMailSender;
    private final UserRepository userRepository;
    private final ApartRepository apartRepository;

    public String sendUuidForApart(MailDto mailDto) {

        User user = userRepository.findByUsername(getLoginUserName());

        SimpleMailMessage message = new SimpleMailMessage();
        Apart apart = apartRepository.getOne(mailDto.getApartId());

        message.setSubject("Apart Kayıt");
        message.setText("Apart'a kayıt olmak için bu kodu kullanınız : " + apart.getUuid());
        message.setTo(mailDto.getTo());
        //message.setFrom(mailDto.getFrom());

        javaMailSender.send(message);

        return SUCCESS_RESPONSE;
    }
}
```

Şekil 2.12. Mail Service

2.8. JavaScript

JavaScript'in rakipleriyle karşılaştırıldığında çeşitli avantajları bulunuyor, özellikle de belirli alanlarda. JavaScript avantajlarından bazıları şu şekildedir:

- Bir compiler (derleyici) kullanmanız gerekmek因为你 web tarayıcıları HTML ile yorumlar;
- Diğer yazılım dillerine nazaran öğrenmesi daha kolaydır;
- Hataları bulmak dolayısıyla çözmek daha kolaydır;
- Belirli web sayfası öğelerine veya özel durumlara ayarlanabilir, örneğin fare tıklaması veya imleci üzerine getirme gibi;
- JS birden fazla platformda, tarayıcıda çalışabilir;
- JavaScript kullanarak input değerlendirebilir ve manuel very kontrollerini azaltabilirsiniz;
- Web siteleri daha interaktif yapar ve ziyaretçilerin dikkatini çeker;
- Diğer yazılım dillerinden daha hızlı ve hafiftir.

2.9. React-Native

React Native için kısaca Facebook tarafından üretilen, cross-platform mobil uygulama geliştirme olanağı sağlayan bir framework diyebiliriz.

Peki “Cross-platform mobil uygulama geliştirme framework’ü” ne demek? “Tek bir dil ile mobil uygulama geliştirip hem Android hem de IOS da derlenip çalışmasını sağlayan bir frameworkdür.

Kendine özgü bir mobil uygulama uzantısı yok, uygulamayı geliştiriyoruz ve Android için .apk, iOS için .ipa uzantılarını alıyoruz. Android ortamına uygulama geliştirirken karşılaşacağımız hatalar Java hataları, iOS ortamına uygulama geliştirirken karşılaşacağımız hatalar Objective-C hataları. Ama kodlarken kullanacağınız dil React.

Kendine has bir dil formatı (JSX) olan React Native bizlere tek bir dil üzerinden kodlama yapabilme ve geliştirilen uygulamanın bir çok platformda çalışma olanağını sunuyor. Geliştiricilere bu desteği sağlayan React Native, cihaz ile arayüz arasında bir köprü görevi görerek geliştirilen mobil uygulamaların sorunsuzca çalışmasını sağlıyor. Siz geliştirme ortamında bir bileşen tanımlıyorsunuz (örneğin; <Text>) ve React Native ilgili ortamda o bileşenin karşılığı ne ise (Android'de TextView, iOS'da UIView) sizin yerinize oluşturuyor. Bunun için de yazılan React (yani Javascript) kodlarını native dile (Java, Objective-C) çevirirken “babel” adı verilen bir yapı devreye giriyor. [1]

2.10. PostgreSQL

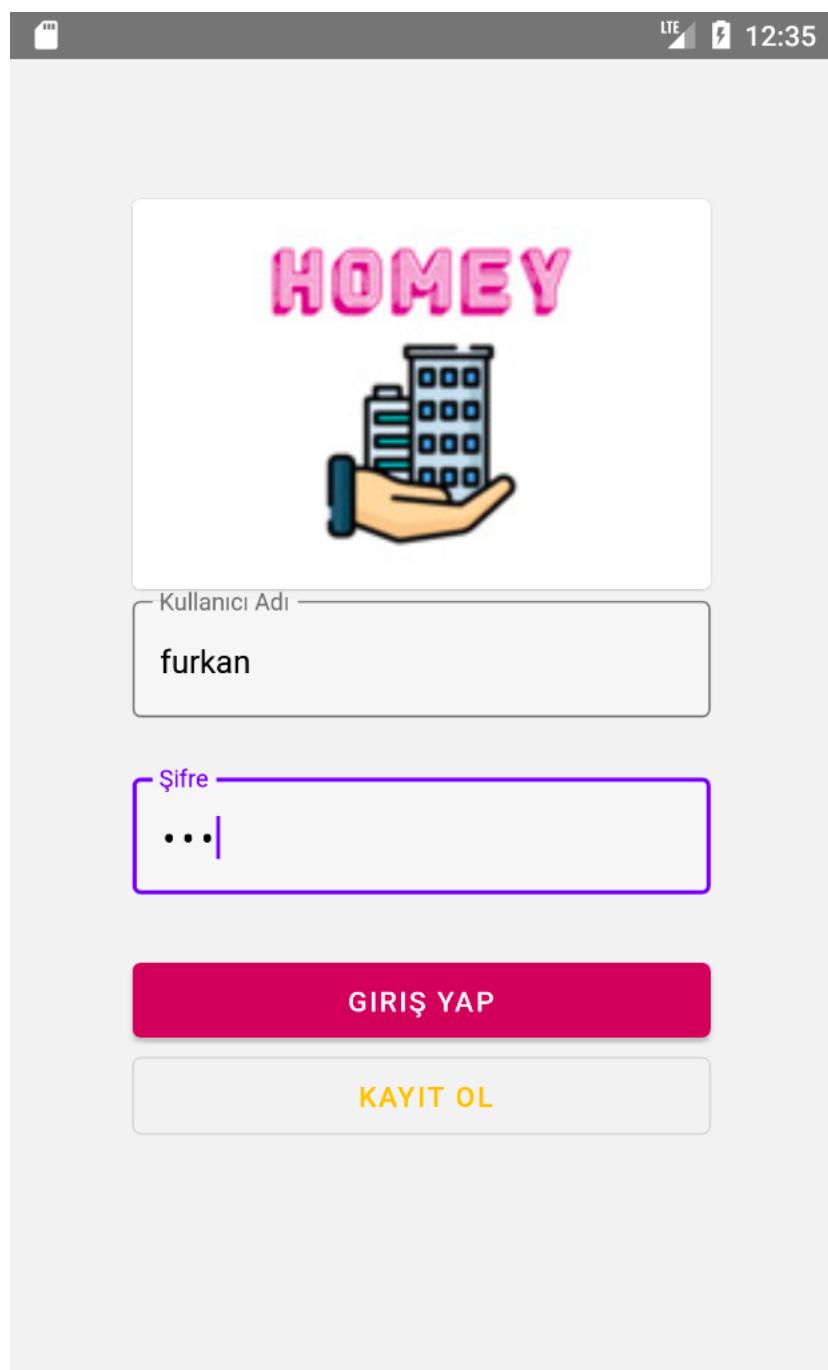
California Üniversitesi’nde 1986 yılında temelleri atılan ve 30 yıldan fazla aktif bir gelişime sahip olan PostgreSQL, yani kısaca Postgres; SQL dilini geliştirerek kullanan, verileri hızlı ve güvenli bir şekilde tutan, açık kaynaklı, obje tabanlı, ilişkisel bir veritabanı yönetim sistemidir. Güvenilirliği dünya çapında kanıtlanmıştır. Her bir sürümü defalarca kontrollerden geçirilmiş ve her bir beta sürümü en az bir aylık testlere tabi tutulmuştur. PostgreSQL veritabanı yönetim sisteminin kaynak koduna dünyanın her yerinden erişilebilir.

Object-Relational olması Postgres’i esnek ve güçlü kılarken, standart ilişkisel veritabanlarının desteklemediği bir takım önemli özellikleri destekleyerek, rakipleri olan MySQL, MariaDB, Firebird vb. veritabanlarına karşı avantaj sağlar. [2]

3. UYGULAMA

Artık mobil uygulamamızı anlatmaya başlayabiliriz. Mobil uygulamayı anlatırken uygulama içerisinde ekran görüntülerini göstereceğim ilk ekranın kodlarını da anlatıp sonrasında kodları es geçeceğim.

Mobil uygulama ilk açıldığında karşımıza kullanıcı giriş yapma ekranı geliyor. Bu ekrandan hem apart-yurt yöneticileri hem de normal kullanıcılar giriş yapabiliyor ama tabi ki giriş yaptıktan sonra yöneticilerin ve kullanıcıların görebildikleri ekranlar farklı oluyor.



Şekil 3.1. Giriş Ekranı

Aşağıdaki fotoğrafta kullanıcının gördüğü giriş yapma ekranının kod ile oluşturulduğu kısımları görüyoruz. Burada önemli ve dikkat etmemiz gereken şey ise “Giriş Yap” butonuna tıklanınca “onLogin” fonksiyonunu çağırması olacaktır. Bir başka ayrıntı ise “Kayıt Ol” butonuna tıklandığında “CreateUser” sayfasına yönlendirme yapıldığıdır.

```
return (
    <View style={styles.screen}>
        <View style={styles.authContainer}>
            <Card>
                <Card.Cover source={{ uri: DEFAULT_IMAGE }} />
            </Card>
            <TextInput mode="outlined" label="Kullanıcı Adı" value={username} onChangeText={(text) => setUsername(text)} />
            <HelperText type="error" visible={username ? false : true}>
                Kullanıcı Adı Zorunludur!
            </HelperText>
            <TextInput mode="outlined" secureTextEntry={true} label="Şifre" value={password} onChangeText={(text) => setPassword(text)} />
            <HelperText type="error" visible={password ? false : true}>
                Şifre Zorunludur!
            </HelperText>
            <View style={styles.buttonContainer}>
                <Button title="Login" mode="contained" color={Colors.primary} onPress={() => {onLogin()}>-Giriş Yap</Button>
            </View>
            <View style={styles.buttonContainer}>
                <Button
                    title="Switch to Sign Up"
                    mode="outlined"
                    color={Colors.accent}
                    onPress={() => navigation.navigate('CreateUser')}
                > Kayıt Ol</Button>
            </View>
        </View>
    </View>
);
```

Şekil 3.2. Giriş Ekranı Kodu

Aşağıdaki fotoğrafta ise “onLogin” fonksiyonunun içini görüyoruz. Burada kullanıcıdan alınan username ve password bir objenin içine atılarak “authServices” dosyasından “login” fonksiyonuna gönderiliyor. Eğer başarılı bir şekilde giriş yapıldıysa backendden bir token dönüyor ve bu tokena uygulamanın her yerinden erişebilmemiz için AsyncStorage ile kaydediyoruz. Bu işlemlerden sonra ise “getUserInformation” fonksiyonunu çağırarak giriş yapan userin bilgilerini alıyoruz ve en son olarak uygulama içerisindeki sayfalara yönlendirme yapıyoruz.

```

const onLogin = async () => {
  clearAsyncStorage();

  let login = {
    username: username,
    password: password,
  };

  const authServices = new AuthServices();
  authServices
    .login(login)
    .then(res => {
      AsyncStorage.setItem('token', res.token);
      getUserInformation(res.token);
    })
    .catch(error => {
      console.log('login err : ', error);
    });
}

```

Şekil 3.3. Giriş Ekranı onLogin Fonksiyonu

Aşağıdaki fotoğrafta ise AuthService.js dosyasındaki “login” fonksiyonunu görüyoruz. Burada login isteği attıktan sonra dönen tokeni “login” fonksiyonun çağırıldığı yere döndürüyoruz. Backendde attığımız her istekte “axios” kütüphanesini kullanıyoruz. Burada istek atacağımız url’i ise “Paths” dosyasından alıyoruz ki ileride baseUrl değişirse tek bir yerden bütün sorgularımızın baseUrl’ini değiştirebilelim.

```

import axios from 'axios';

import * as Paths from '../../services/api/paths';

export default class AuthServices {

  login(login) {

    return new Promise((resolve, reject) => {
      axios.post(Paths.login, login, {
        headers: { 'Accept-Language': 'tr' },
      })
        .then(res => {
          resolve(res.data);
        })
        .catch(err => {
          reject(err);
        });
    });
  }
}

```

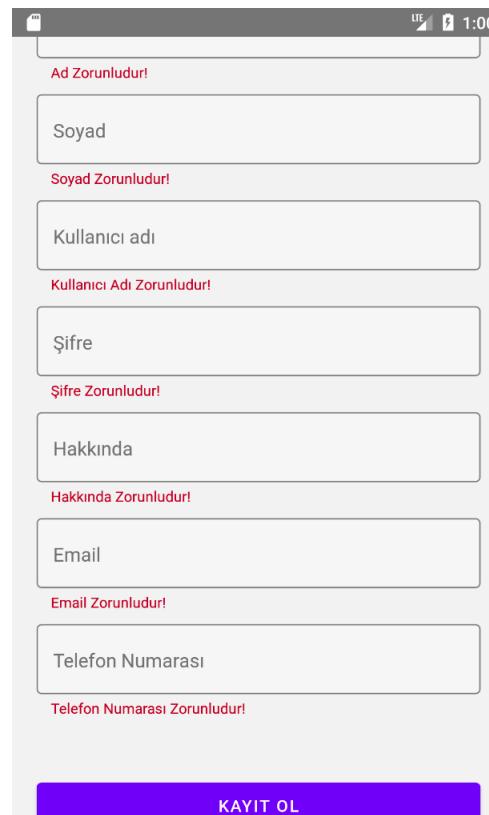
Şekil 3.4. Auth Services Login Fonksiyonu

Paths.js dosyası ise şu şekilde.

```
src > services > api > [js] paths.js > [o] user
1  const baseUrl = 'http://172.16.5.78:8082';
2
3  export const login = baseUrl + '/login';
4  export const apart = baseUrl + '/apart';
5  export const user = baseUrl + '/user';
6  export const getAllApart = baseUrl + '/get/all/apart';
7  export const createUser = baseUrl + '/public/create/user';
8  export const registerUserHouse = baseUrl + '/userhouse';
9  export const createComplaint = baseUrl + '/complaint';
10 export const getAllComplaint = baseUrl + '/get/all/complaint';
11 export const getAllComplaintManager = baseUrl + '/get/all/manager/complaint';
12 export const sendUuidForApart = baseUrl + '/send/uuid/for/apart';
13 export const createTool = baseUrl + '/tool';
14 export const getReservations = baseUrl + '/get/reservations';
15 export const createReservation = baseUrl + '/reservation';
16 export const getTools = baseUrl + '/get/tools';
17 export const smartDevice = baseUrl + '/smart/device';
18 export const complaintTypeWorkingOn = baseUrl + '/complaint/working/on';
19 export const complaintTypeSolved = baseUrl + '/complaint/solved';
```

Şekil 3.5. Paths.js Dosyası

“Kayıt Ol” butonuna tıkladığımızda ise kayıt olma sayfasına gidiyoruz. Burada gerekli bilgileri girdikten sonra kayıt olma işlemi yapabiliriz.



Şekil 3.6. Kayıt Ol Ekranı

Kullanıcının gördüğü kayıt olma ekranının kod ile oluşturulma kısmı aşağıdaki görseldeki gibidir. Burada “<ScrollView> sayfanın aşağı doğru kaydırılabilmesini ve <KeyboardAvoidingView> ise sayfada bir <TextInput>’a tıklandığında klavyenin TextInput altında açılmasını sağlar.

```

return (
  <View style={styles.main}>
    <ScrollView>
      <KeyboardAvoidingView
        behavior={Platform.OS === 'ios' ? 'padding' : 'height'}
        style={styles.container}
        keyboardVerticalOffset={-250}>
        <Title>KAYIT OL</Title>

        <TextInput
          mode="outlined"
          label="Ad"
          value={name}
          onChangeText={text => setName(text)}>
        </>
        <HelperText type="error" visible={name ? false : true}>
          Ad Zorunludur!
        </HelperText>
        <TextInput
          mode="outlined"
          label="Soyad"
          value={surname}
          onChangeText={text => setSurname(text)}>
        </>
        <HelperText type="error" visible={surname ? false : true}>
          Soyad Zorunludur!
        </HelperText>
        <TextInput
          mode="outlined"
          label="Kullanıcı adı"
          value={username}
          onChangeText={text => setUsername(text)}>
        </>
        <HelperText type="error" visible={username ? false : true}>
          Kullanıcı Adı Zorunludur!
        </HelperText>
        <TextInput
          mode="outlined"
          label="Şifre"
          secureTextEntry={true}
          value={password}
          onChangeText={text => setPassword(text)}>
        </>
        <HelperText type="error" visible={username ? false : true}>
  
```

Şekil 3.7. Kayıt Ol Ekranı Kodu

Kullanıcı oluşturma fonksiyonu ise aşağıdaki görseldeki gibidir. Burada bir user objesi oluşturuyoruz ve kullanıcından aldığımız bilgileri bu user objesinin içine atıyoruz. Aynı zamanda kullanıcının rolünü de burada “USER” olarak belirtiyoruz ki bu kullanıcının kimliği belli olsun. Kullanıcı oluşturuldu ise bir “SnackBar” çıkartıp

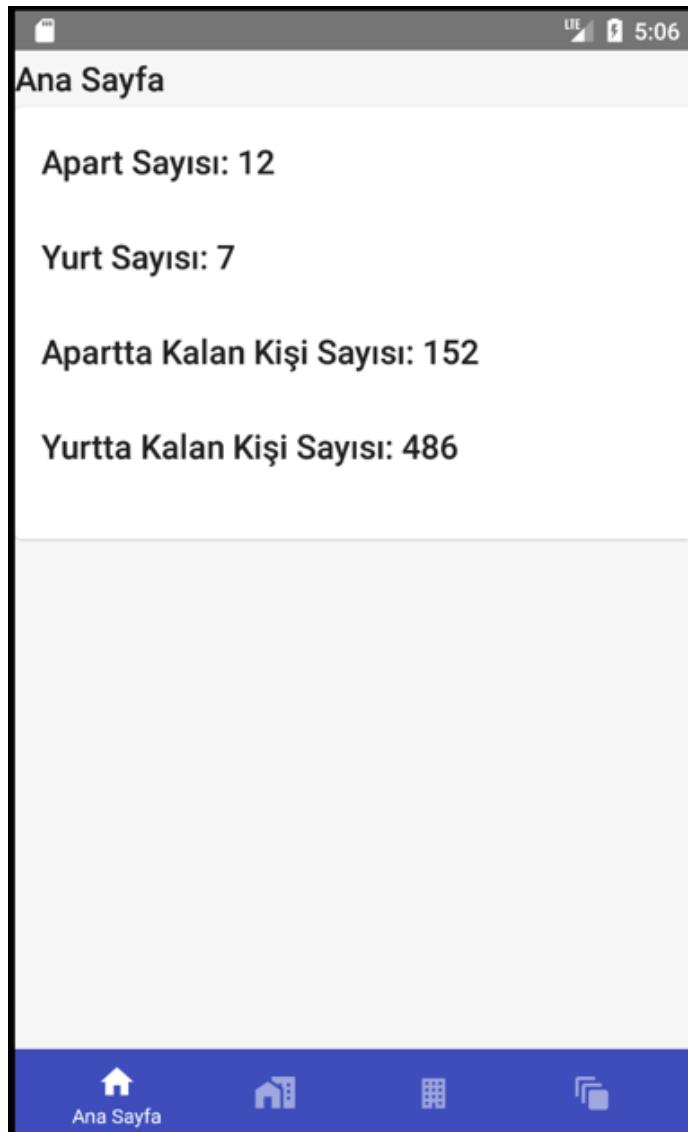
kullanıcıya başarılı bir şekilde kayıt olduğunu bildirip tekrar giriş yapma ekranına yönlendiriyoruz.

```
const createUser = async () => {
  let user = {
    name: name,
    surname: surname,
    username: username,
    password: password,
    about: about,
    email: email,
    phone: phone,
    role: 'USER',
    status: 1,
  };

  const authServices = new AuthServices();
  authServices
    .createUser(user)
    .then(res => {
      console.log('kayıt dönüş değeri = ' + res);
      setSnackbarVisible(true);
      navigation.navigate('SingIn');
    })
    .catch(error => {
      console.log('login err : ', error);
    });
};
```

Şekil 3.8. Kayıt Ol Ekranı createUser Fonksiyonu

Aşağıdaki görüntüde gördüğümüz yöneticinin giriş yaptıktan sonra gördüğü ekrandır. Bu ekranda yöneticinin yönettiği apart ve yurt sayısını ve aynı zamanda bu yurt ve apartlarda kalan kişi sayılarını özet bir şekilde görebiliyoruz. Bu sayfanın aslında görünmeyen ve önemli olan bir rolü ise giriş yapıldıktan sonra bu ekrana gelindiğinde arkaplanda backende giriş yapılan akıllı cihazın firebaseden aldığı tokeni göndermeye yarıyor. Backendde alınan bu tokeni kaydedip sonrasında ihtiyaç olduğunda bu token üzerinden telefona bildirim gönderebiliyoruz.



Şekil 3.9. Yönetici Ana Sayfa Ekranı

Aşağıdaki kodlarda ana sayfada token alma servisini ve daha sonrasında backend'e gönderdiğimiz servisi görüyoruz.

```

useEffect (()=>{
    requestUserPermission();
    messaging()
        .getToken()
        .then(token => [
            //Service token kayıt edilecek.
            createSmartDevice(token);
        ]);
    getAllApart();
}, []);
}

async function requestUserPermission(fcmToken) {
    const authStatus = await messaging().requestPermission();
    const enabled =
        authStatus === messaging.AuthorizationStatus.AUTHORIZED ||
        authStatus === messaging.AuthorizationStatus.PROVISIONAL;

    if (enabled) {
        console.log('Authorization status:', authStatus);
    }
}

const createSmartDevice = async (fcmToken) => {
    const token = await AsyncStorage.getItem('token');

    let smartDevice = {
        fcmToken:fcmToken
    };

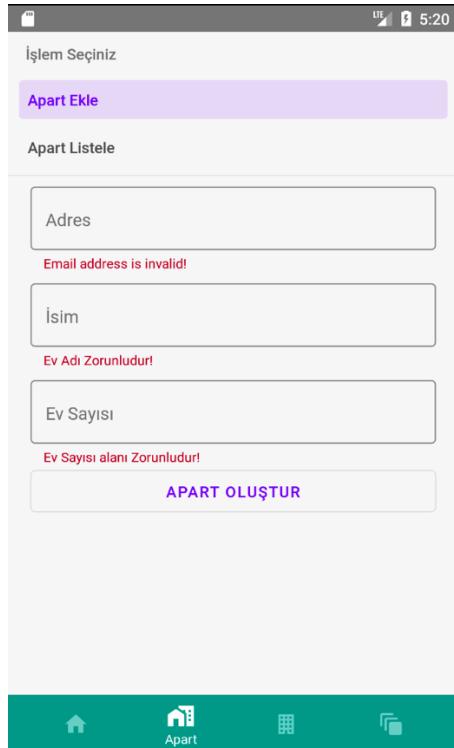
    const homeServices = new HomeServices();
    homeServices.createSmartDevice(smartDevice,token).then(res => {
        console.log("Smart Device Saved");

    }).catch(error => {
        console.log("get apart err : ", error)
    });
}

```

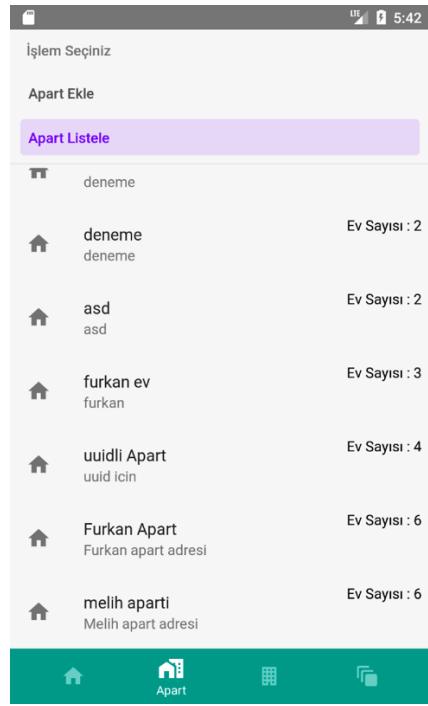
Şekil 3.10. Yönetici Ana Sayfa Ekranı Token Kaydetme Fonksiyonu

Aşağıdaki ekranда apart ekleme ekranını görüyoruz. Bu ekranda yönetici apartın gerekli bilgilerini doldurup bir apart oluşturabilir. Oluşturan kişi o apartin yöneticisi olarak otomatik olarak tanımlanır. Aynı zamanda bu ekranada oluşturulan apartları listeleyebiliriz.



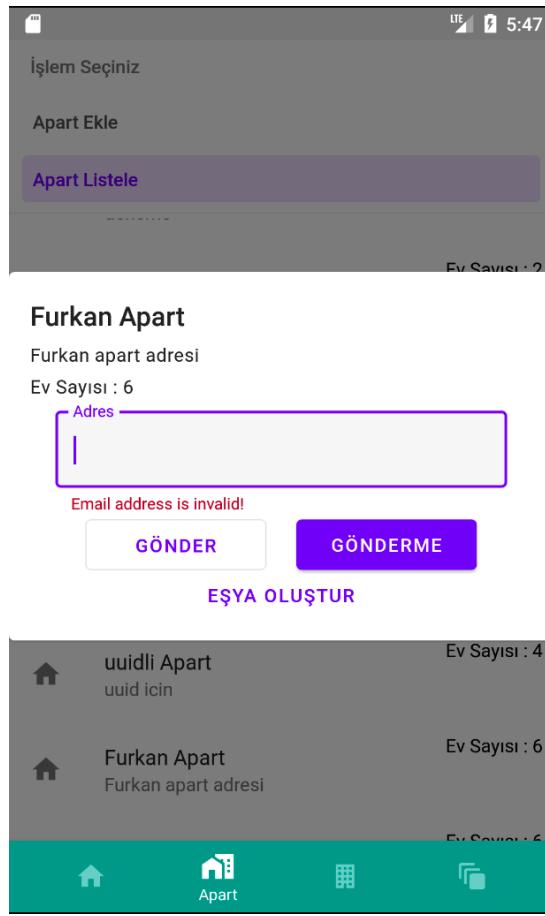
Şekil 3.11. Yönetici Apart Ekleme Ekranı

Aşağıdaki ekran apart listeleme ekranıdır. Bu ekranda istediğimiz apartın üstüne tıklayıp yeni kayıt olacak kullanıcılarla aparta özel kodu mail olarak gönderebiliriz ya da apart için yeni bir araç gereç ekleneceği zaman buradan ekleyebiliriz.



Şekil 3.12. Yönetici Apart Listeleme Ekranı

Aşağıdaki görüntüde apart için kod gönderme kısmını görüyoruz.



Şekil 3.13. Yönetici Kullanıcıya Kod Gönderme Ekranı

Aşağıdaki görüntüde aparta kayıt için gönderilen kodu görüyoruz.



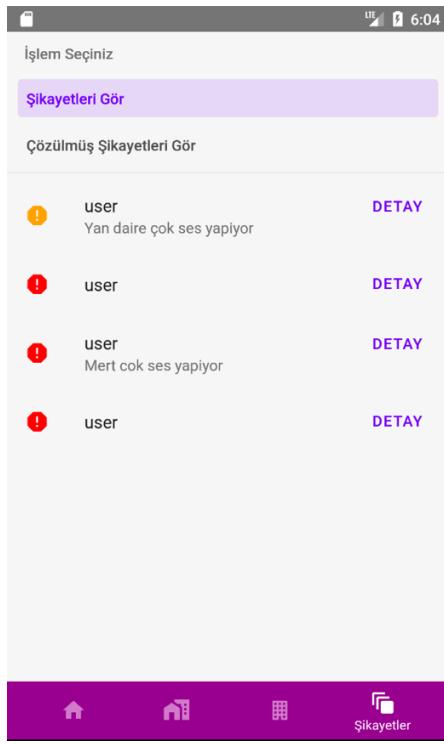
Şekil 3.14. Mail

Aşağıdaki görüntüde ise aparta yeni bir eşya ekleme kısmını görüyoruz.



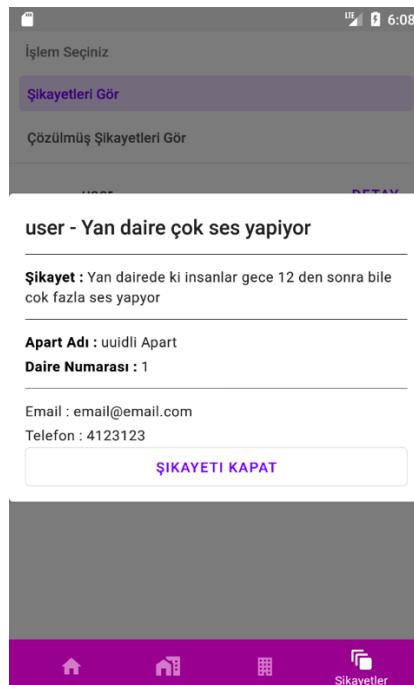
Şekil 3.15. Yönetici Eşya Ekleme Ekranı

Aşağıdaki ekranда yönetici kullanıcıların oluşturduğu şikayetleri görmektedir. Bu şikayetler 3 aşamada tutulmaktadır. Birincisi şikayet oluşturulma durumu ikincisi şikayetin işleme alındığı durum üçüncüsü ise şikayetin çözüldüğü durum. Burada yönetici ilk önce açılan şikayetin işleme alıp sonrasında bu şikayet çözüldüğü zaman şikayetini kapatabilir.



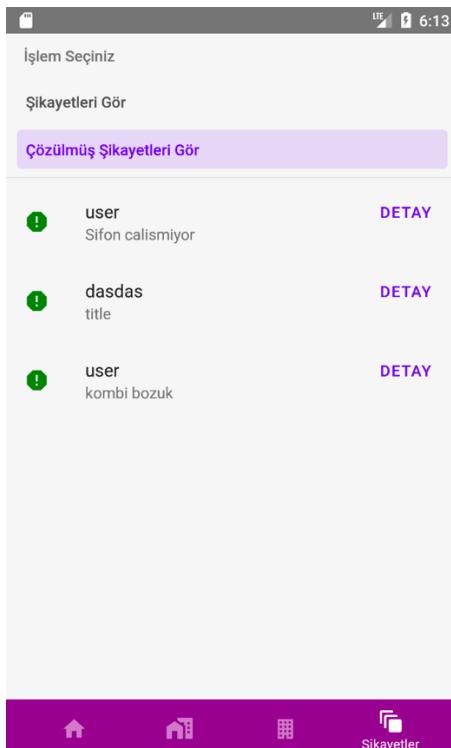
Şekil 3.16. Yönetici Şikayetleri Görme Ekranı

Aşağıdaki görüntüde oluşturulan şikayetleri detaylı olarak görebiliyoruz. Burada apart ya da yurt bilgilerini ve kullanıcının bilgilerini görebiliyoruz. Burada user yazan kısmında kullanıcının ismidir.



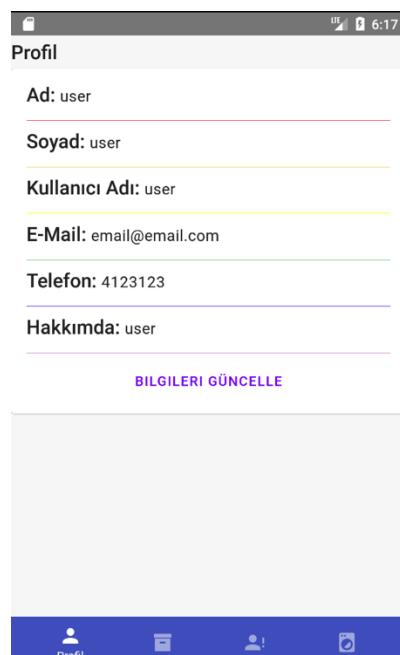
Şekil 3.17. Yönetici Şikayet Detayı Görme Ekranı

Aşağıdaki görüntüde ise çözülmüş şikayetleri listeliyoruz.



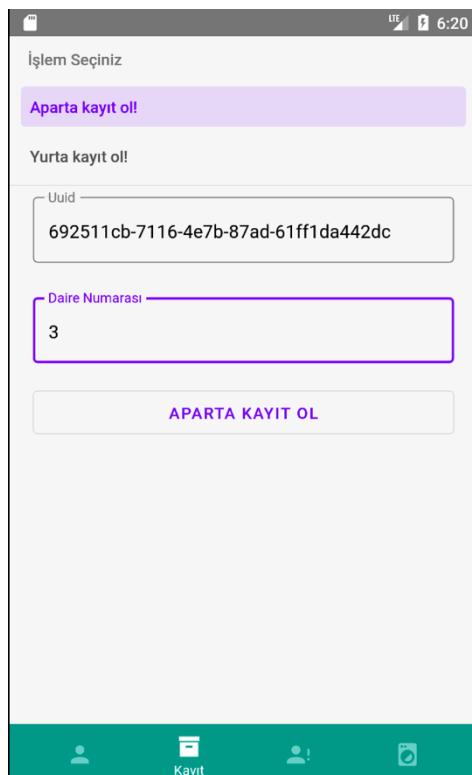
Şekil 3.18. Yönetici Çözülmüş Şikayetleri Görme Ekranı

Aşağıdaki görüntüde kullanıcının giriş yaptıktan sonra ilk gördüğü ekranı görüyoruz. Burada kullanıcının kendi bilgilerini görüyoruz.



Şekil 3.19. Kullanıcı Profil Ekranı

Aşağıdaki görselde kullanıcının bir aparta ya da yurta kayıt olmasi için kullanılan ekranı görüyoruz. Bu ekranda mail ile gelen uuid'yi ve daire numarasını girerek bir aparta kayıt olmayı gerçekleştirebiliyoruz.



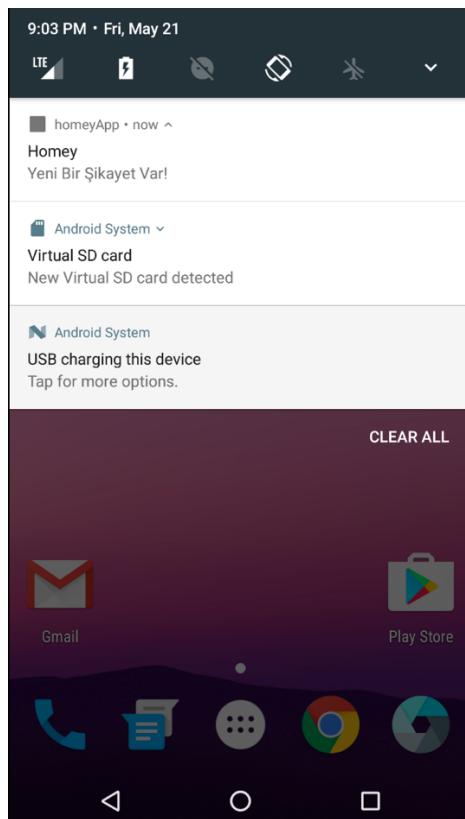
Şekil 3.20. Kullanıcı Aparta Kayıt Olma Ekranı

Aşağıdaki ekranda ise kullanıcıların şikayet oluşturabildiğini görüyoruz. Burada tek yapmak gereken şikayet başlığı ve şikayetü girmek ve şikayet oluştur demektir. Daha sonrasında kullanıcının kendi oluşturduğu şikayetleri görebildiği bir sekme de bulunuyor.



Şekil 3.21. Kullanıcı Şikayet Oluşturma Ekranı

Kullanıcı bir şikayet oluşturduğu zaman yöneticiye bir bildirim gidiyor ve yeni bir şikayet olduğunu uyarısı bu şekilde verilebiliyor.



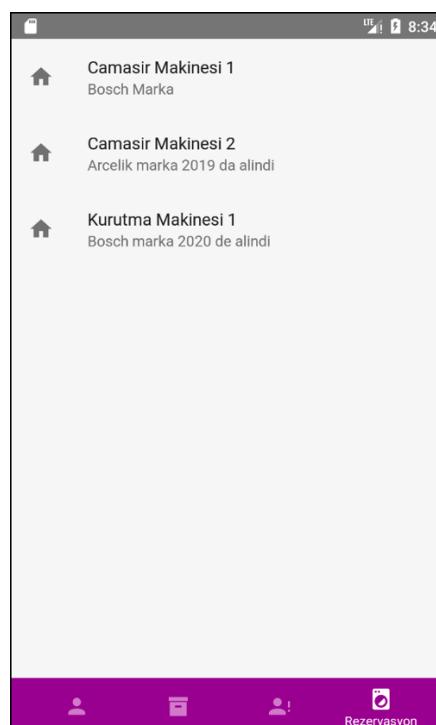
Şekil 3.22. Bildirim

Aşağıdaki ekranda oluşturulan şikayetleri görebiliriz.



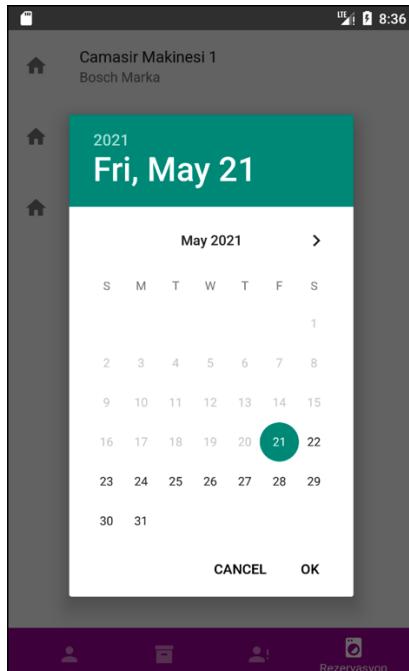
Şekil 3.23. Kullanıcı Oluşturduğu Şikayetleri Görme Ekranı

Aşağıdaki görselde ise kullanıcının kayıt olduğu apartin kullanılabilir eşyaları için rezervasyon oluşturabildiği ekranı görüyoruz



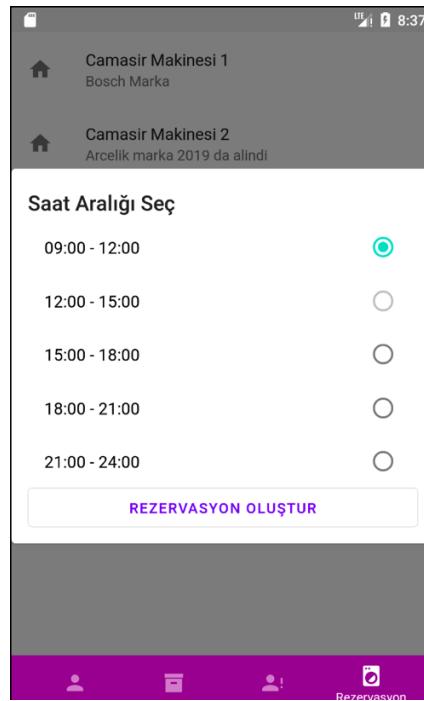
Şekil 3.24. Kullanıcı Eşya Listeleme Ekranı

Herhangi bir eşyanın üstüne tıklandığında tarih seçmemiz gerekiyor.



Şekil 3.25. Kullanıcı Tarih Seçme Ekranı

Tarih seçtikten sonra aşağıdaki görseldeki gibi saat aralığı seçmemiz gerekiyor. Eğer daha öncesinde o tarih ve saat aralığında birisi rezervasyon yaptıysa o seçenek pasif oluyor ve seçilemiyor.



Şekil 3.26. Kullanıcı Saat Aralığı Seçme Ekranı

4. SONUÇLAR VE ÖNERİLER

Sonuç olarak Homey mobil uygulaması sayesinde hem yurt ve apart yöneticileri hem de yurt ve apartlarda kalan kişiler çok kolay bir şekilde birbirlerine dertlerini anlatabilir ve buralarda daha rahat bir şekilde zaman planlaması yaparak araç gereçleri kullanabilir.

Kullanmış olduğum teknolojiler Java:Spring Boot, Maven, Hibernate, JPA, RESTful, Lombok, Spring Security, Spring Mail ve Jwt birbirleriyle çok uyumlu çalışan bir backend yapısı oluşturuyor ve backendi kolayca yazmamızı sağlıyor. Mobil uygulama kısmında kullanmış olduğum React-Native ise dinamik bir mobil uygulamayı hem ios hem de android tarafına kolay bir şekilde geliştirebileceğimizi gösteren frameworklerinden birisi olduğunu kanıtlıyor.

Ben projemde kullanmış olduğum teknolojileri uzun bir araştırma sürecinden sonra tam anlamıyla öğrenebildim. Bu raporda anlattığım şekilde basit bir yapı kurup siz de bu teknolojiler ile bir proje ortaya çıkartabilirsiniz.

Hedeflediğim yere tam olarak ulaşamasam da ana hatlarıyla bu projeyi bitirebildiğimi düşünüyorum. Eksik olan kısımları ise ilerleyen süreçlerde tamamlanabilir ve uygulama tam sürümü ile birlikte mobil marketlerde yerini alabilir.

KAYNAKLAR

- [1] <https://medium.com/kodcular/react-native-nedir-7b333d319597>
- [2] https://akademi.bilgeadam.com/courses/postgresql/?gclid=Cj0KCQjwkZiFBhD9ARIsAGxFX8AmZlxJPMCn7PZJUNdh_vVm4QvEpJo_qmURa3fSaQOW7vTyGYKVx5kaAkoFEALw_wcB
- [3] <https://medium.com/yazilim-vip/bu-yaz%C4%B1n%C4%B1n-amac%C4%B1-maven-ile-siz-okurlar%C4%B1-tan%C4%B1%C5%9Ft%C4%B1rmakt%C4%B1r-aab0f6ff91f4>
- [4] <https://medium.com/@furkanbegen/spring-boot-nedir-4cc3f41eb7de>
- [5] <https://blog.burakkutbay.com/hibernate-nedir-hibernate-dersleri.html/>
- [6] [https://medium.com/kodgemisi/project-lombok-6d2490df8adf#:~:text=Lombok%2C%20Java%20projesi%20geli%C5%9Ftirirken%20IDE,%C3%BCCretme%20\(code%20generation\)%20k%C3%BCCt%C3%BCphanesidir.&text=%C3%96rnek%20olarak%20Bir%20POJOda%20\(Plain,toString%20vb.%20methodlar%C4%B1%20yazmam%C4%B1z%20gerekliyor](https://medium.com/kodgemisi/project-lombok-6d2490df8adf#:~:text=Lombok%2C%20Java%20projesi%20geli%C5%9Ftirirken%20IDE,%C3%BCCretme%20(code%20generation)%20k%C3%BCCt%C3%BCphanesidir.&text=%C3%96rnek%20olarak%20Bir%20POJOda%20(Plain,toString%20vb.%20methodlar%C4%B1%20yazmam%C4%B1z%20gerekliyor)
- [7] <https://sangaibisi.medium.com/hibernate-ve-spring-data-jpa aras%C4%B1ndaki-fark-nedir-aab8274c41e4>
- [8] <https://medium.com/@bsrutmnr/rest-ve-restful-web-servi%CC%87s-nedi%CC%87r-7258b7db7f66>

ÖZGEÇMİŞ

Furkan KOÇ 01.03.1997 yılında İstanbul'da doğdu. Lise hayatından beri bilgisayar üzerine okumakta ve çalışmaktadır. Lisede Handan Hayrettin Yelkikanat Anadolu Teknik Lisesinde bilgisayar bölümünde okumuştur. Daha sonrasında Marmara Üniversitesi Bilgisayar Programcılığı bölümünde okumuş ve mezun olmuştur. Mezun olduktan sonra bir süre Bilgi İşlem Sorumlusu olarak çalışan Furkan bu süreçte DGS sınavına hazırlandı ve sınav sonucunda Sakarya Üniversitesi Bilgisayar Mühendisliği bölümünü kazandı. Burada okurken Bileşim A.Ş. firmasında staj yapmış ve şuan 4. Sınıfta öğrenciliği devam etmektedir. 4. Sınıfın başında Sakarya Teknokentte faaliyetlerine devam eden CMV Teknoloji firmasında çalışmaya başlamıştır ve hala çalışmaya devam etmektedir..

BSM 498 BİTİRME ÇALIŞMASI
DEĞERLENDİRME VE SÖZLÜ SINAV TUTANAĞI

KONU : Homey Apart ve Yurt Sistemi

ÖĞRENCİLER (Öğrenci No/AD/SOYAD): G171210303 / Furkan / KOÇ

Değerlendirme Konusu	İstenenler	Not Aralığı	Not
Yazılı Çalışma			
Çalışma klavuza uygun olarak hazırlanmış mı?	x	0-5	
Teknik Yönden			
Problemin tanımı yapılmış mı?	x	0-5	
Geliştirilecek yazılımın/donanımın mimarisini içeren blok şeması (yazılımlar için veri akış şeması (dfd) da olabilir) çizilerek açıklanmış mı?			
Blok şemadaki birimler arasındaki bilgi akışına ait model/gösterim var mı?			
Yazılımın gereksinim listesi oluşturulmuş mu?			
Kullanılan/kullanılması düşünülen araçlar/teknolojiler anlatılmış mı?			
Donanımların programlanması/konfigürasyonu için yazılım gereksinimleri belirtilmiş mi?			
UML ile modelleme yapılmış mı?			
Veritabanları kullanılmış ise kavramsal model çıkarılmış mı? (Varlık ilişki modeli, noSQL kavramsal modelleri v.b.)			
Projeye yönelik iş-zaman çizelgesi çıkarılarak maliyet analizi yapılmış mı?			
Donanım bileşenlerinin maliyet analizi (prototip-adetli seri üretim vb.) çıkarılmış mı?			
Donanım için gerekli enerji analizi (minimum-uyku-aktif-maksimum) yapılmış mı?			
Grup çalışmalarında grup üyelerinin görev tanımları verilmiş mi (iş-zaman çizelgesinde belirtilebilir)?			
Sürüm denetim sistemi (Version Control System; Git, Subversion v.s.) kullanılmış mı?			
Sistemin genel testi için uygulanan metodlar ve iyileştirme süreçlerinin dökümü verilmiş mi?			
Yazılımın sizme testi yapılmış mı?			
Performans testi yapılmış mı?			
Tasarımın uygulamasında ortaya çıkan uyumsuzluklar ve aksaklılıklar belirtilerek çözüm yöntemleri tartışılmış mı?			
Yapılan işlerin zorluk derecesi?	x	0-25	
Sözlü Sınav			
Yapılan sunum başarılı mı?	x	0-5	
Soruları yanıtlama yetkinliği?	x	0-20	
Devam Durumu			
Öğrenci dönem içerisindeki raporlarını düzenli olarak hazırladı mı?	x	0-5	
Diğer Maddeler			
Toplam			

DANIŞMAN (JÜRİ ADINA):

DANIŞMAN İMZASI: