SAKARYA ÜNİVERSİTESİ

BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

NESNE YÖNELİMLİ ANALİZ VE TASARIM DERSİ YIL İÇİ PROJE ÇALIŞMASI

NESNELERİN İNTERNETİ SİSTEMLERİ İÇİN AKILLI CİHAZ (NESNE) TASARIMI

Hazırlayanlar:

G181210078

Gökay İŞERİ

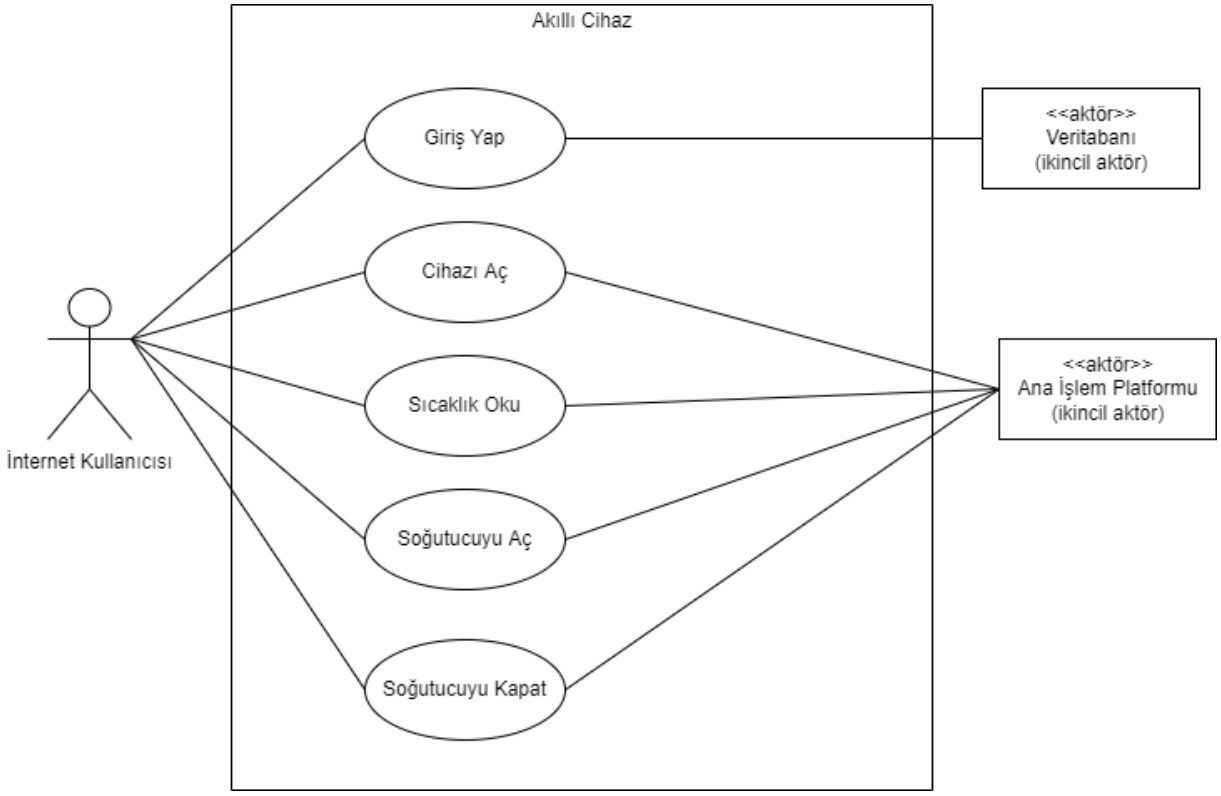
2.Öğretim A Şubesi

G201210065

Ceren YILDIRIM

2.Öğretim B Şubesi

1. “İnternet kullanıcısı” için kullanım durumu (Use Case) diyagramı:



2. Kullanım Durumları:

2.1. Kullanım Durumunun Tanımı:

Kullanım durumu adı: Sıcaklığı görüntüle

Hazırlayanlar: Gökay İşeri, Ceren Yıldırım

Sürüm: V1.0.1

Tarih: 12.04.2022

İlgili Aktörler: İnternet Kullanıcısı, Ana İşlem Platformu (ikincil), Veri tabanı (ikincil)

Giriş Koşulu: Müşteri internet sitesine girer. Kullanıcı doğrulanır.

Çıkış Koşulu: Müşteri sıcaklık değerini okur. Başka herhangi bir işlem yapmaz.

Ana Olay Akışı:

1. Kullanıcı sisteme giriş yapar.

2. Ara yüz merkezi sisteme doğrulama gönderir.

3. Merkezi sistem kullanıcıyı doğrular.

4. Ara yüz kullanıcıya işlem seçeneklerini sunar ve yapmak istediği işlemi ister.

5. Kullanıcı sıcaklık göster işlemini girer.

6. Ara yüz merkezi sisteme işlem doğrulaması gönderir.

7. Merkezi sistem işlemi doğrular ve sıcaklık algılayıcıya işlemi gönderir.

8. Sıcaklık algılayıcı sıcaklığı okur ve merkezi sisteme gönderir.

9. Merkezi sistem sıcaklığı alır, ara yüze iletir.

10. Ara yüz sıcaklığı gösterir. Kullanıcıya başka işlem yapıp yapılmayacağı sorulur.

Alternatif Olay Akışı:

A1. Kullanıcı doğrulanamaz. Üç kez yanlış girişte çıkılır.

A2. İşlem doğrulanamaz. Tekrar işlem alınır.

Özel Gereksinimler: UI gereksinimleri, İşlem gecikmesi en fazla 1 dk. olmalı, 24 saat çalışmalı.

2.2. Kullanım Durumunun Tanımı:

Kullanım durumu adı: Soğutucu Aç

Hazırlayanlar: Gökay İşeri, Ceren Yıldırım

Sürüm: V1.0.1

Tarih: 12.04.2022

İlgili Aktörler: İnternet Kullanıcısı, Ana İşlem Platformu (ikincil), Veri tabanı (ikincil)

Giriş Koşulu: Müşteri internet sitesine girer. Kullanıcı doğrulanır.

Çıkış Koşulu: Müşteri soğutucuyu açar. Başka herhangi bir işlem yapmaz.

Ana Olay Akışı:

1. Kullanıcı sisteme giriş yapar.

2. Ara yüz merkezi sisteme doğrulama gönderir.

3. Merkezi sistem kullanıcıyı doğrular.

4. Ara yüz kullanıcıya işlem seçeneklerini sunar ve yapmak istediği işlemi ister.

5. Kullanıcı soğutucu aç işlemini girer.

6. Ara yüz merkezi sisteme işlem doğrulaması gönderir.

7. Merkezi sistem işlemi doğrular ve Eyleyici birime işlemi gönderir.

8. Eyleyici soğutucu açar ve işlemin tamamlandığını merkezi sisteme gönderir.

9. Merkezi sistem soğutucu açıldı bildirisini ara yüze iletir.

10. Ara yüz soğutucu açıldı bildirisini gösterir. Kullanıcıya başka işlem yapıp yapılmayacağı sorulur.

Alternatif Olay Akışı:

A1. Kullanıcı doğrulanamaz. Üç kez yanlış girişte çıkılır.

A2. İşlem doğrulanamaz. Tekrar işlem alınır.

Özel Gereksinimler: UI gereksinimleri, İşlem gecikmesi en fazla 1 dk. olmalı, 24 saat çalışmalı

2.3. Kullanım Durumunun Tanımı:

Kullanım durumu adı: Soğutucu Aç

Hazırlayanlar: Gökay İşeri, Ceren Yıldırım

Sürüm: V1.0.1

Tarih: 12.04.2022

İlgili Aktörler: İnternet Kullanıcısı, Ana İşlem Platformu (ikincil), Veri tabanı (ikincil)

Giriş Koşulu: Müşteri internet sitesine girer. Kullanıcı doğrulanır.

Çıkış Koşulu: Müşteri soğutucuyu kapar. Başka herhangi bir işlem yapmaz.

Ana Olay Akışı:

1. Kullanıcı sisteme giriş yapar.

2. Ara yüz merkezi sisteme doğrulama gönderir.

3. Merkezi sistem kullanıcıyı doğrular.

4. Ara yüz kullanıcıya işlem seçeneklerini sunar ve yapmak istediği işlemi ister.

5. Kullanıcı soğutucu kapa işlemini girer.

6. Ara yüz merkezi sisteme işlem doğrulaması gönderir.

7. Merkezi sistem işlemi doğrular ve Eyleyici birime işlemi gönderir.

8. Eyleyici soğutucu kapar ve işlemin tamamlandığını merkezi sisteme gönderir.

9. Merkezi sistem soğutucu kapandı bildirisini ara yüze iletir.

10. Ara yüz soğutucu kapandı bildirisini gösterir. Kullanıcıya başka işlem yapıp yapılmayacağı sorulur.

Alternatif Olay Akışı:

A1. Kullanıcı doğrulanamaz. Üç kez yanlış girişte çıkılır.

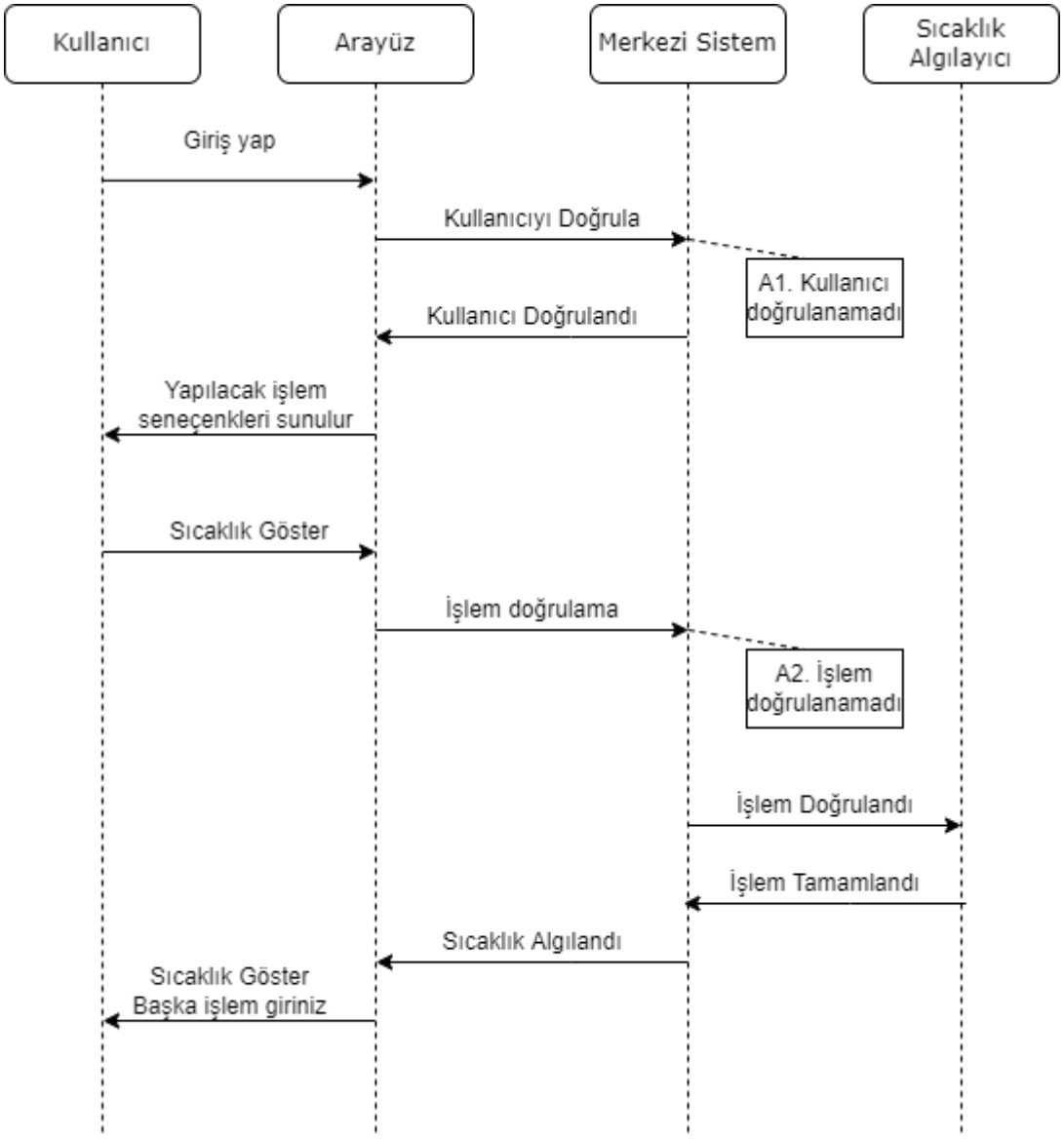
A2. İşlem doğrulanamaz. Tekrar işlem alınır.

Özel Gereksinimler: UI gereksinimleri, İşlem gecikmesi en fazla 1 dk. olmalı, 24 saat çalışmalı

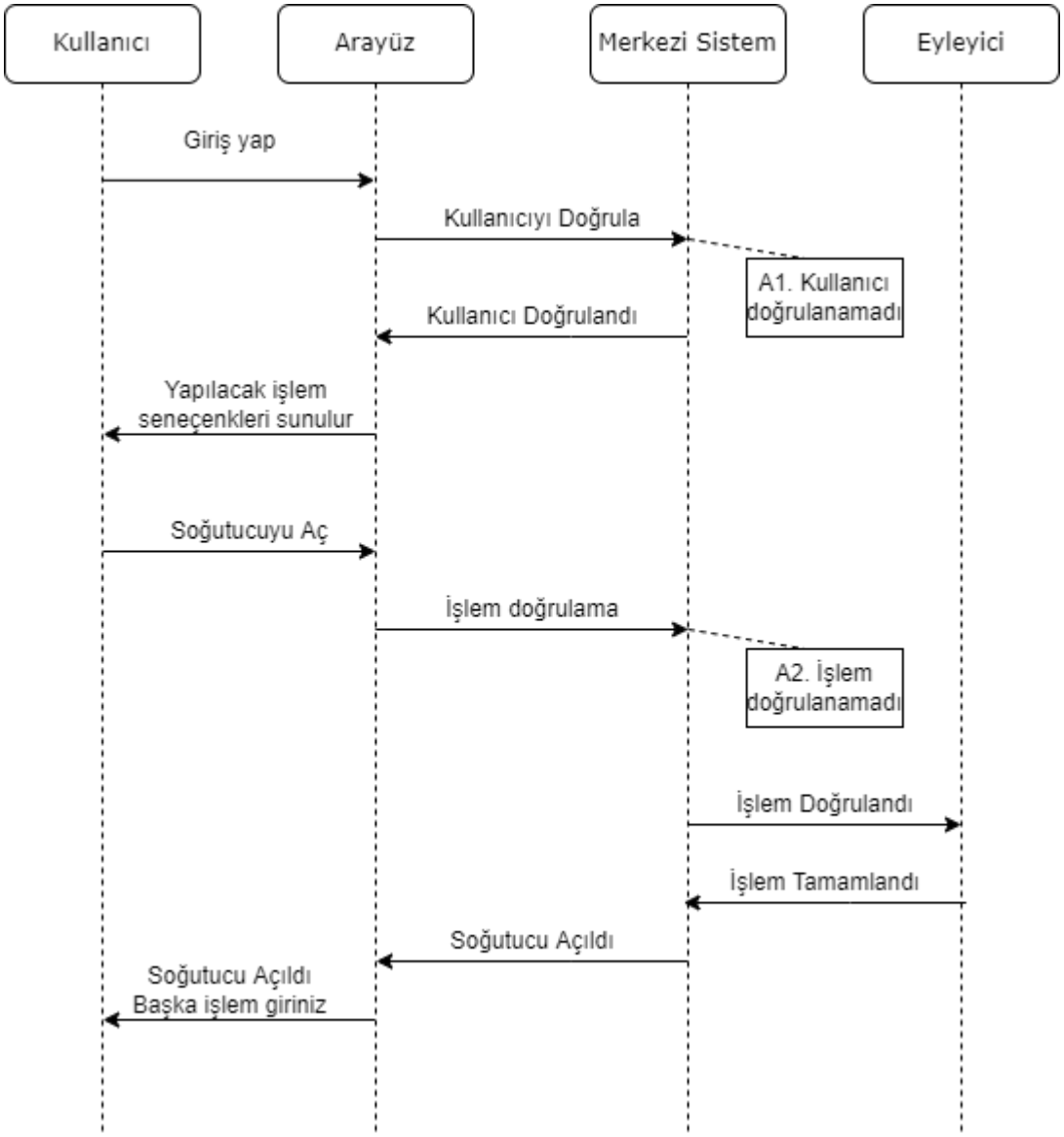
3. Sıralama ve Etkinlik Şemaları:

3.1. Sıralama Şeması (sequence diagram):

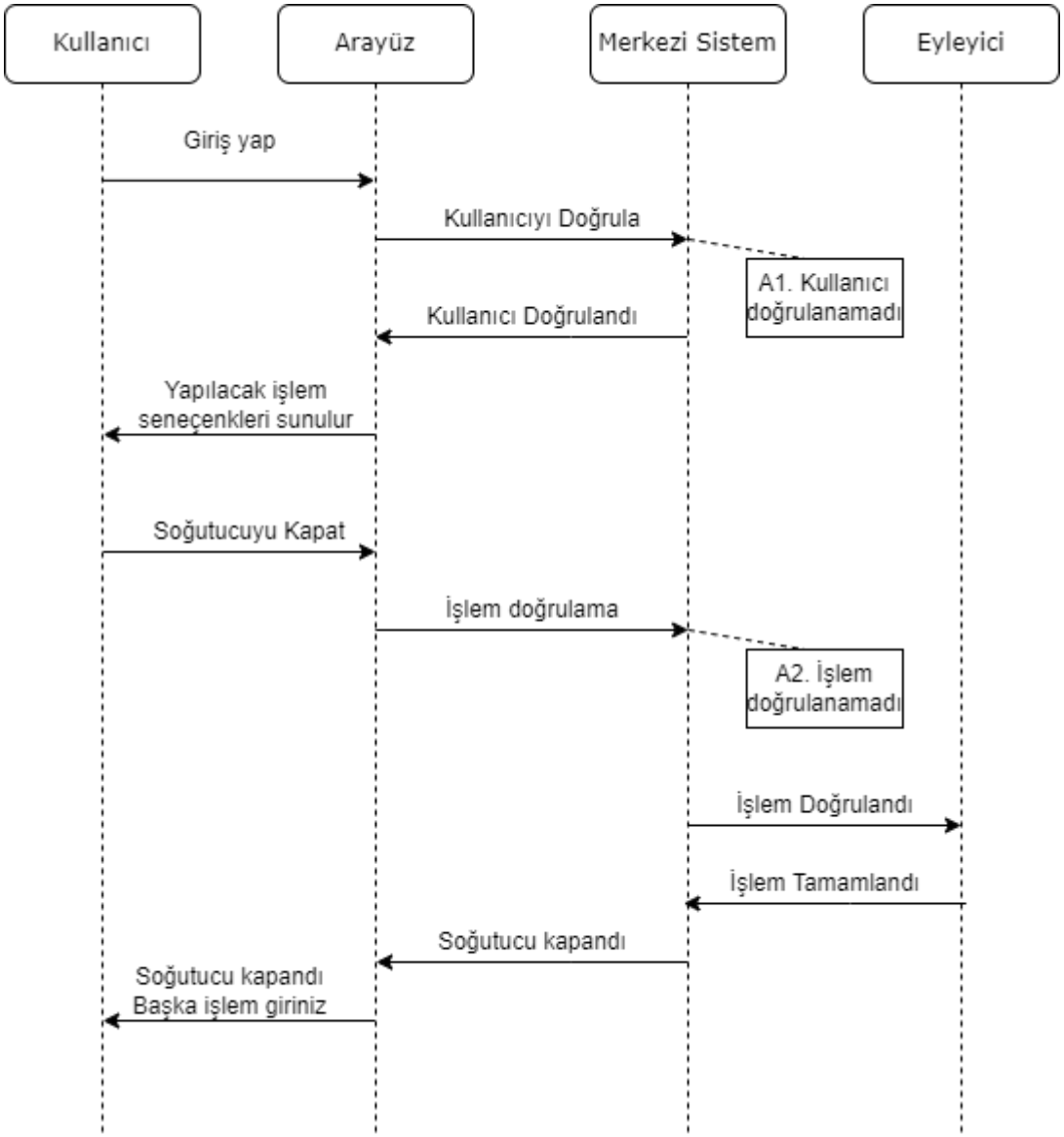
3.1.1. Sıcaklığın Görüntülenmesi:



3.1.2. Soğutucunun Açılması:

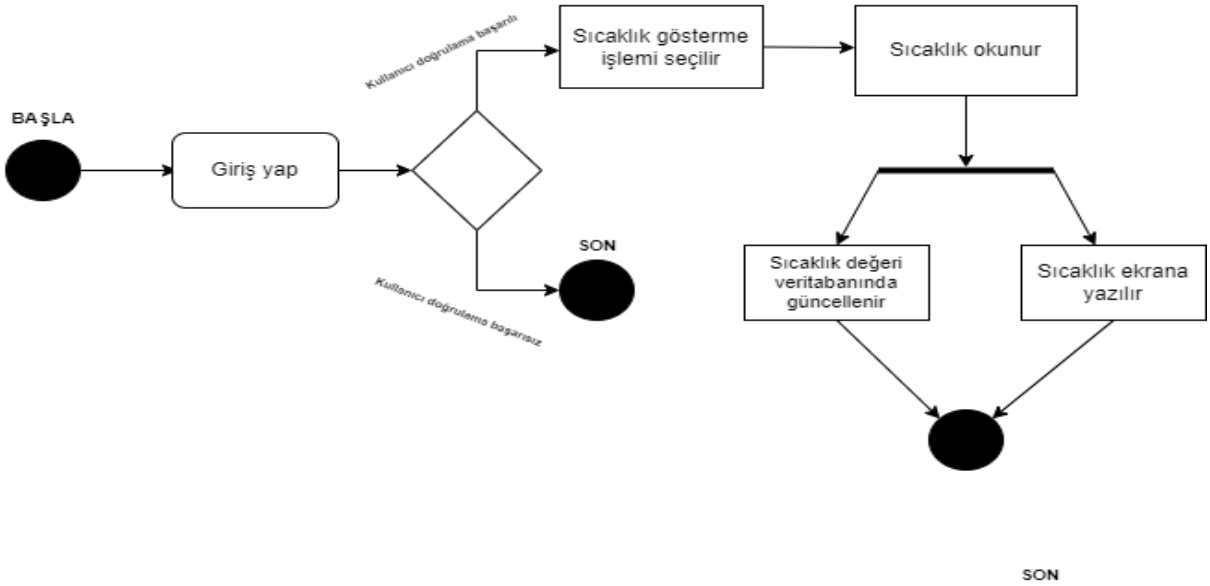


3.1.3. Soğutucunun Kapanması:

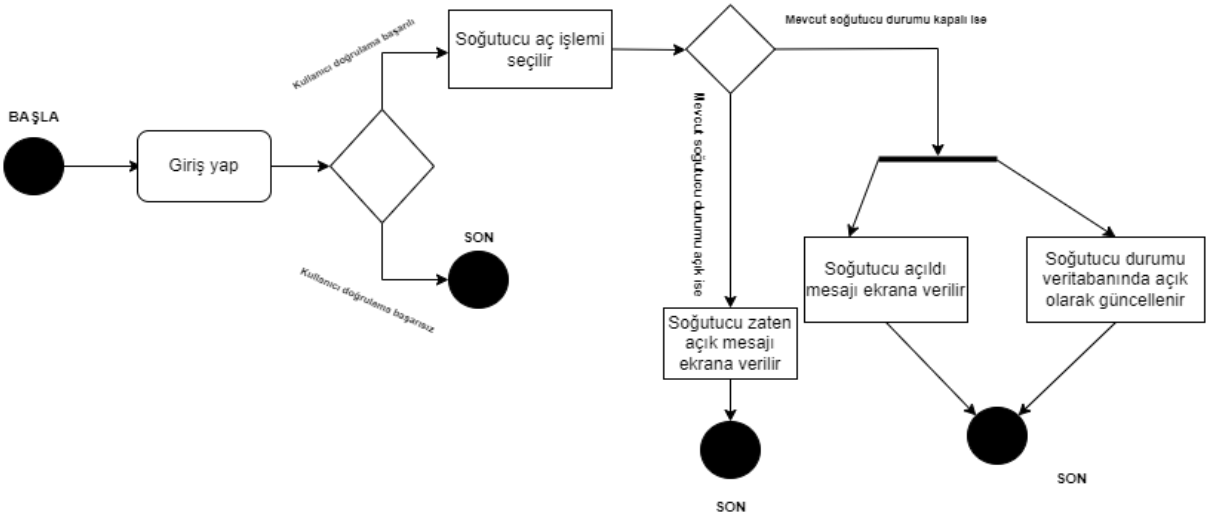


3.2. Etkinlik Şeması (activity diagram):

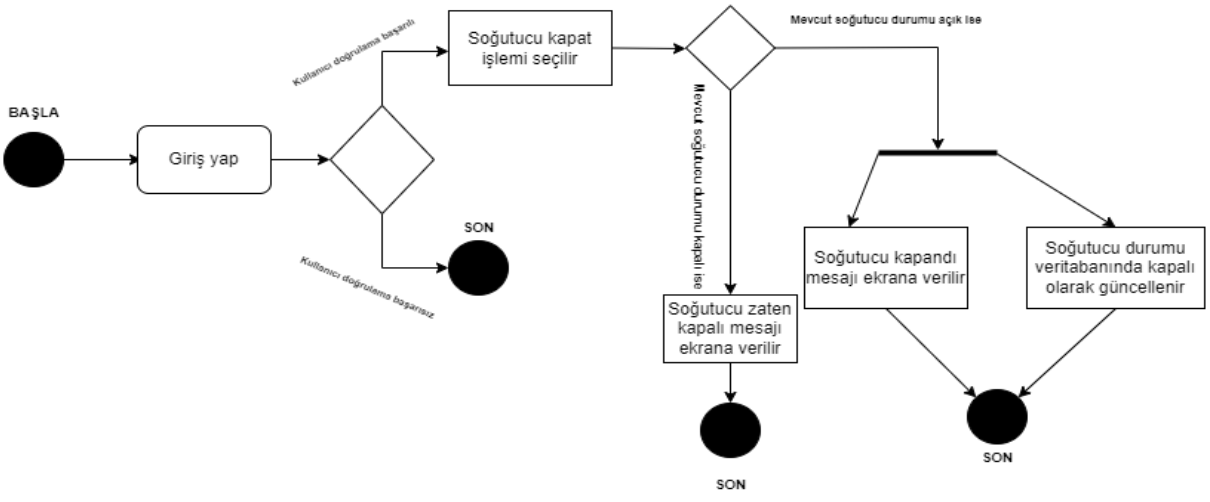
3.2.1. Sıcaklığın Görüntülenmesi:



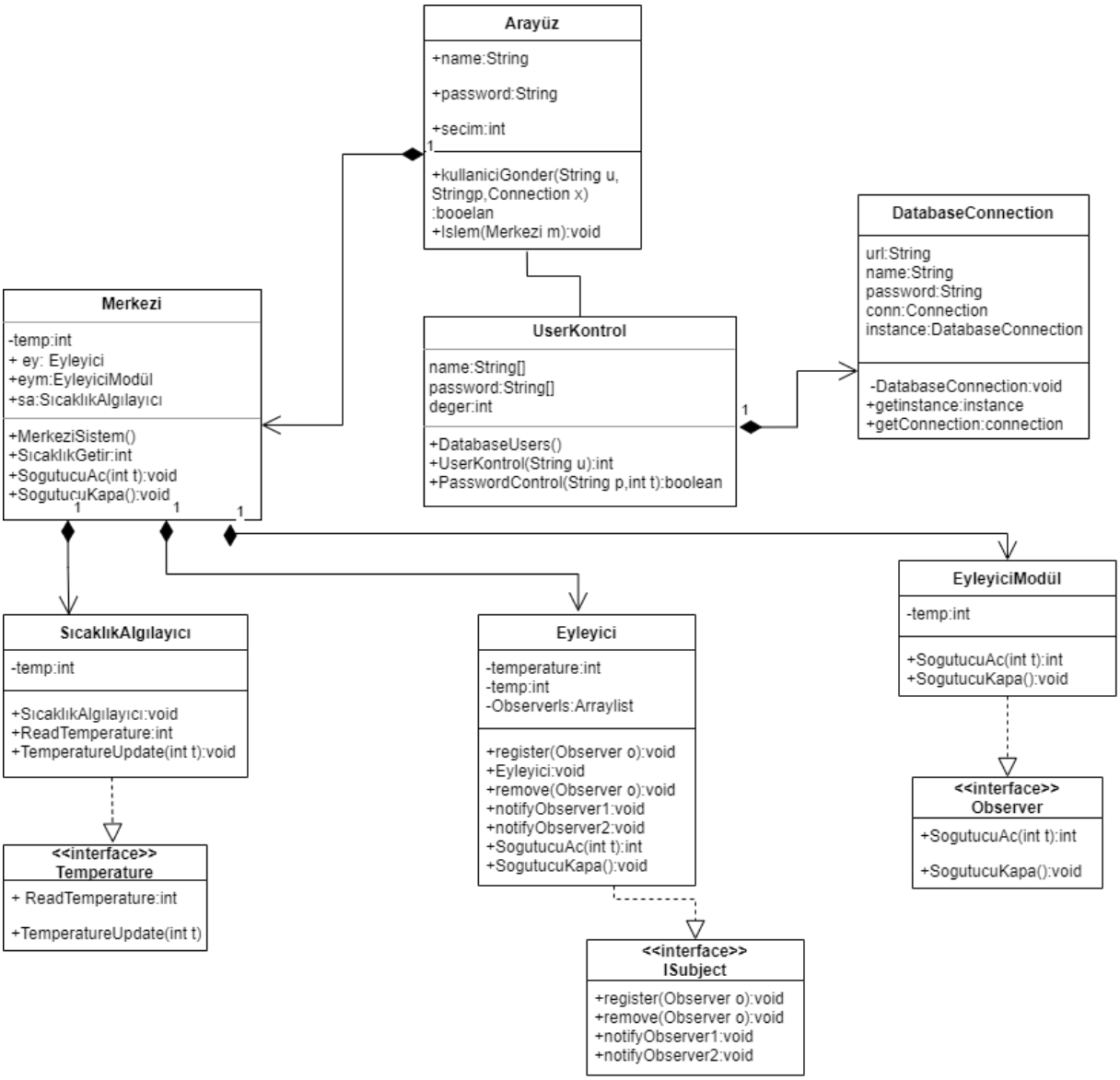
3.2.2. Soğutucunun Açılması:



3.2.3. Soğutucunun Kapanması:



4. Sınıf Şeması:

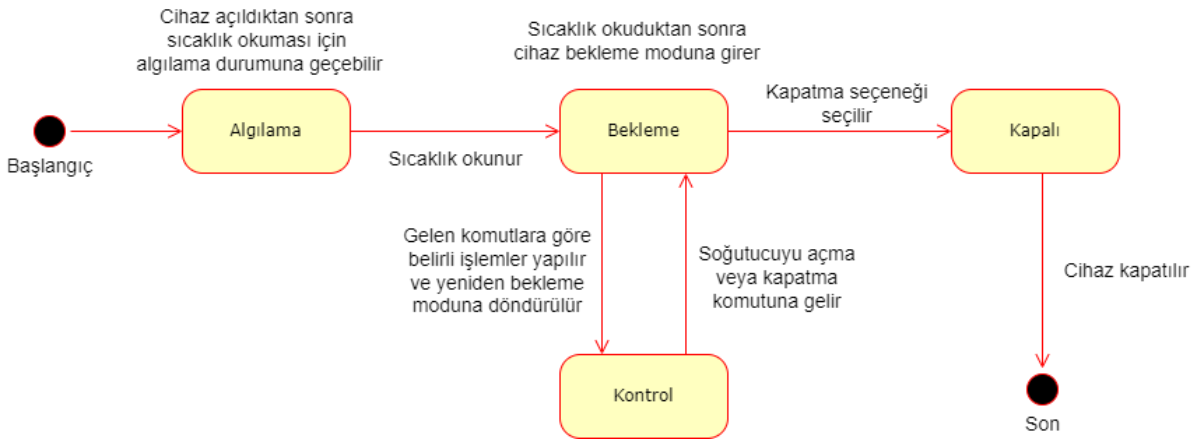


5. CRC Kartlar:

Merkezi (Temel İşlemleri yapar)	
Sorumluluk	İşbirliği Yapılan Sınıf
Sıcaklığı göster	Sıcaklığı Algılayıcı
Soğutucuyu Aç	Eyleyici, Eyleyici Modül
Soğutucuyu Kapat	Eyleyici, Eyleyici Modül

Arayüz (Kullanıcı girişi ve işlem seçimi)	
Sorumluluk	İşbirliği Yapılan Sınıf
Kullanıcı gönder	UserKontrol
İşlem seç	Merkezi

6. Durum Makinesi Diyagramı:



7. Kullanıcı Doğrulama Ekranı:

Username: user

Password: 123456

Service classında bulunan girisYap() fonksiyonu ile veri tabanına bağlanarak veritabanında kayıtlı olan username ve passwordu kontrol ederiz. Eğer username ve password doğru ise giriş yapar değil ise giriş yapılmaz.

```
Giris Yapiniz
username :
user
password :
123456
Giris Basarili..
1) Sicaklik Olcme
2) Sogutucu Ac
3) Sogutucu Kapat
4) Kapat
```

8. Sıcaklığın Görüntülenmesi, Soğutucunun açılıp kapatılması:

“1” secimi yapılırsa random bir sıcaklık oluşturulur. Ekrana yazdırılır. Service classındaki sicaklikGuncelle() fonksyonu ile ölçülen değer veritabanına gönderilir.

```
1
Sicaklik : 9.28

1) Sicaklik Olcme
2) Sogutucu Ac
3) Sogutucu Kapat
4) Kapat
```

“2” secimi yapılırsa Api package içerisinde bulunan Service classındaki sogutucuGuncelle() fonksyonu sayesinde veritabanından soğutucunun durumunu güncelleriz. Eyleyici classında sogutucuAc() fonksyonu içinde çağırılan service.sogutucuGuncelle(true) fonksyonu true döner ve sogutucu açılır. Eğer halihazırda açıksa “zaten açık” mesajı ekrana yazdırılır.

```
2
Cihaz acildi.
1) Sicaklik Olcme
2) Sogutucu Ac
3) Sogutucu Kapat
4) Kapat
```

Aynı şekilde “3” secimi yapılırsa Api package içerisinde bulunan Service classındaki sogutucuGuncelle() fonksyonu sayesinde veritabanından soğutucunun durumunu güncelleriz. Eyleyici classında sogutucuKapat() fonksyonu

içinde çağırılan service.sogutucuGuncelle(false) fonksyonu false döner ve sogutucu kapatılır. Eğer halihazırda kapalıysa “zaten kapalı” mesajı ekrana yazdırılır

```
3
Cihaz Kapandi.
1) Sicaklik Olcme
2) Sogutucu Ac
3) Sogutucu Kapat
4) Kapat
3
Cihaz Kapanmadi!
Cihaz zaten kapali
1) Sicaklik Olcme
2) Sogutucu Ac
3) Sogutucu Kapat
4) Kapat
```

9. Veri tabanı:

Soğutucu(Device) tablosu:

	id [PK] integer	open boolean
1	1	false

Sıcaklık(Temperature) tablosu:

	id [PK] integer	value double precision
1	1	9.282864703966318

Kullanıcı(User) tablosu:

	username character varying (128)	password character varying (128)
1	user	123456

10. Dependency Inversion:

Dependency inversion alt ve üst sınıfların birbirine bağımlılığının interface ile giderilmesine dayanan bir SOLID ilkesidir. Bu ilkenin amacı projelerdeki zaman maliyetini düşürmek ve koda sonradan eklenecek yapıların projeye girişini, okunabilirliği kolaylaştırmak.

Kodumuzda bu ilkeyi mesaj gönderimi yapmak için kullandık. Yazici classı oluşturdu ve araya soyut bir IObserver interface yazdık. IObserver interfaceini Subscriber sınıfına implement ettik. Böylelikle ikinci bir Subscribe sınıfına ihtiyaç duyulursa koda kolayca eklenebilecek. Ve yüksek seviyeli bir sınıfın alt seviyeli sınıflara bağlılığını azaltmış oluyoruz. Artık işlemler soyut bir sınıf üzerinden ilerleyecek.

IObserver.java

```
1  package ui;
2
3  public interface IObserver {
4      public void update(String msg);
5  }
6
```

IObserver.java | Subscriber.java

```
1  package ui;
2
3  /**
4   * subscriber Observerin alicisi
5   */
6
7  public class Subscriber implements IObserver{
8      @Override
9      public void update(String msg) {
10         System.out.print(msg);
11     }
12 }
```

IObserver.java | Subscriber.java | IYazici.java | Yazici.java

```java
package ui;

/**
 * Print sinifi bir nevi Publisher IObserver icin alan
 **/
public class Yazici implements IYazici {

    private IObserver observer;

    public Yazici(IObserver observer) {
        this.observer = observer;
    }


    @Override
    public void EkranaYaz(String text) {
        observer.update(String.format("%s\n", text));
    }

    @Override
    public void EkranaYaz(String format, Object... args) {
        observer.update(String.format(format, args));
    }
}
```

```java
IObserver sub1 = new Subscriber();
Yazici p = new Yazici(sub1); // dependency inversion
```

11.Factory Metod & Observer Design Pattern:

Factory metodu yazamadığımız için builder design pattern ile yazdık.

Builder Design Pattern: Bir nesne oluşturduğumuzda sınıfımızın içerisindeki değişkenlerden ürettiğimiz constructor metod çok fazla veya gereksiz parametreden oluşabilir. Nesne yaratılırken bu constructor metod içerisindeki kadar parametre alması gerekir. Farklı parametreler alan ya da istediğimiz kadar parametre alabilen metod oluşturumunda Builder Pattern kullanabiliriz.

```
Service service = new Service.ServiceBuilder("SmartDeviceDB")
        .username("postgres")
        .password("123698745goko")
        .build();
```

Kodumuzda builderı Service classı içinde kullandık. Builder sayesinde database bilgilerini ServiceBuildera parametre olarak verdik.

```
93      public static class ServiceBuilder {
94          private String name; // database adi
95          private String username; // database usernamei
96          private String password; // database sifresi
97
98          public ServiceBuilder(String name) {
99              this.name = name;
.00          }
.01
.02          public ServiceBuilder username(String username) {
.03              this.username = username;
.04              return this;
.05          }
.06
.07          public ServiceBuilder password(String pass) {
.08              this.password = pass;
.09              return this;
.10          }
.11
.12          public Service build() {
.13              return new Service(this);
.14          }
.15      }
.16  }
```

Observer Design Pattern: Observer pattern, bir sistem içerisindeki farklı bölümlerin birbirine engel olmadan çalışmasını sağlar. Çok sayıda nesneye gözlemledikleri olayı bildirmek için kullanırız. Kodumuzda ekrana mesaj gönderiminde observer kullandık.

```java
package ui;

public interface IObserver {
    public void update(String msg);
}
```

```java
package ui;

/**
 * subscriber Observerin alicisi
 */

public class Subscriber implements IObserver{
    @Override
    public void update(String msg) {
        System.out.print(msg);
    }
}
```