**ANKARA ÜNİVERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

****

**BLM 4538 IOS ile Mobil Uygulama Geliştirme II**

**FİNAL RAPORU**

**Yiğit ÇOLAK**

**21290484**

**Enver BAĞCI**

**https://github.com/COLAKYIGIT/BLM4538**

**https://youtube.com/shorts/HXvbVIc8L\_I?feature=share**

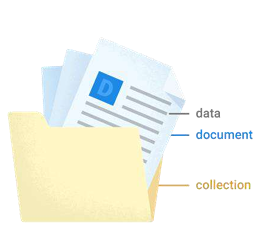
**Haziran 2025**

# AÇIKLAMA

Proje başlangıcında yapılacakların listelendiği ve konsept tasarımların bulunduğu bir rapor verilmişti. Verilen o rapordaki gif hazırlama ve tasarım kısımları eksiktir. Örneğin; sensör verilerinin görüntülendiği kutucukların arkaplan rengi, cihazın uyguladığı hedef parametre değerleri ile cihazda oluşan ortamın verisinin (örneğin sıcaklığın hedef değeri 27 °C ancak cihazda soğutma işlevi olmadığı için sıcaklık cihaz içerisinde 29 °C olması gibi) uzaklığına göre kırmızı ile yeşil arası renklendirilmesi planlanmıştı gibi tasarım detayları eksiktir. Proje kapsamında mobil uygulamanın işlevlerine yoğunlaşılmıştır.

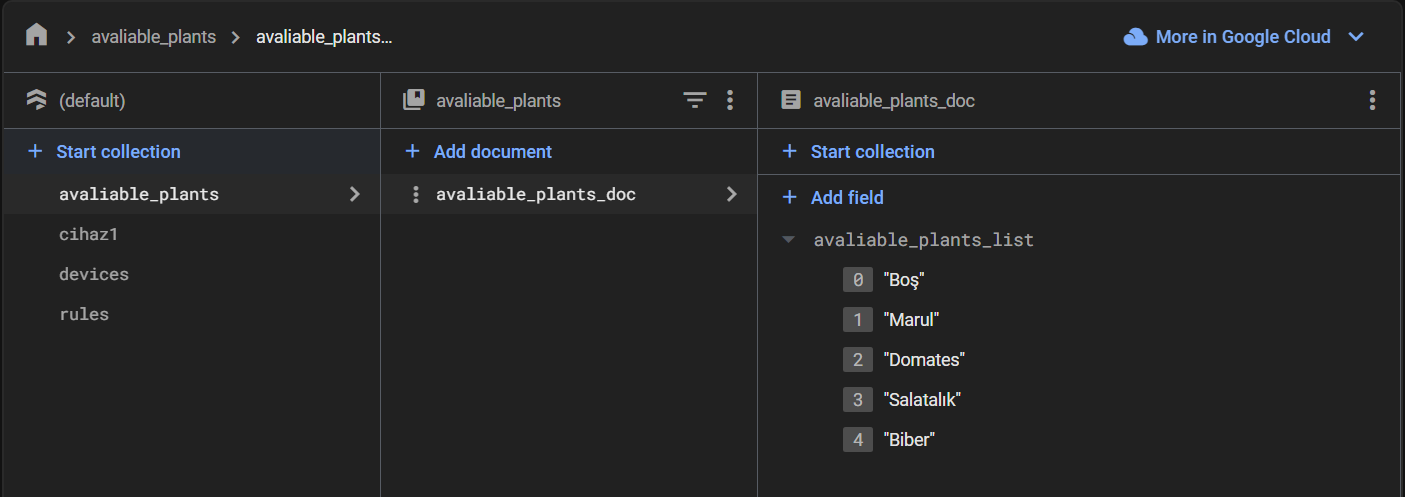
# VERİTABANI VE BİLDİRİM TETİKLEYİCİSİ

Projenin amacı içerisinde bitki yetiştirilmesine uygun ortam hazırlayan, sensörler ve kontrol elemanlarıyla donatılı bir topraksız tarım cihazı ile cihaz kullanıcısı arasında bir iletişim sağlamak için bir mobil uygulama geliştirmektir. Bu bağlamda, kullanıcı cihaz içerisinde yetiştirdiği bitki verisini (marul, domates, biber vb.) mobil uygulama ve veritabanı üzerinden cihaza yollayabilir. Cihaz aldığı bu yetiştirilen bitkiye en uygun ortamın parametrelerini yine veri tabanından alır. Cihaz ise bitki yetişirken oluşturulan ortamın verilerini (sıcaklık, nem, ışık şiddeti, sudaki toplam çözünmüş besin vb.) kullanıcının takip edebilmesi amacıyla mobil uygulamada görüntülenebilecek şekilde yollayabilir. Ek olarak cihaz bir sorun algıladığında (örn: “Sıcaklık düşürülemiyor, fan arızalanmış olabilir.”) bu sorun mesajını kullanıcıya uygulama bildirimi üzerinden iletir.

Bu amacın gerekliliklerine en uygun veritabanı olarak Firebase Firestore kullanılmıştır. Bu veritabanı NoSql bir veritabanıdır. Veritabanı koleksiyonlar, dokümanlar ve verilerin tutulduğu alanlardan oluşur. Bu hiyerarşi koleksiyon, doküman, koleksiyon, doküman, alan olarak da genişletilebilir.

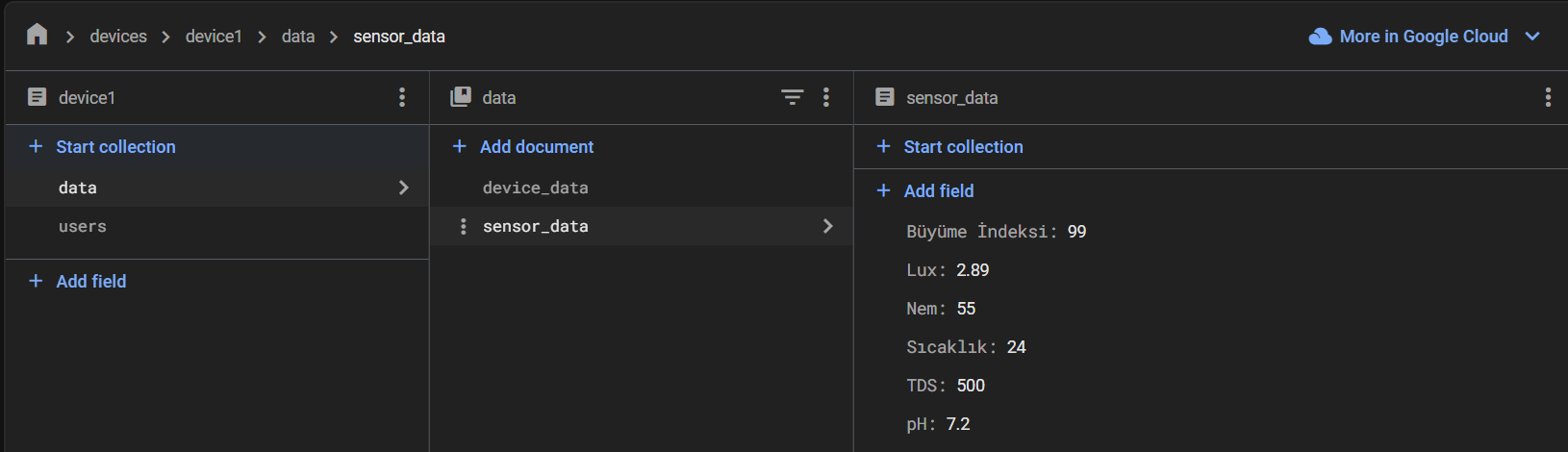
### Şekil 1.1. Firebase Firestore Veri Yapısı

Veritabanında kullanıcının seçebileceği bitkiler listesi ‘avaliable\_plants’ koleksiyonu altında ‘avaliable\_plants\_doc’ dökümanı altında ‘avaliable\_plants\_list’ alanında liste olarak tutulur.



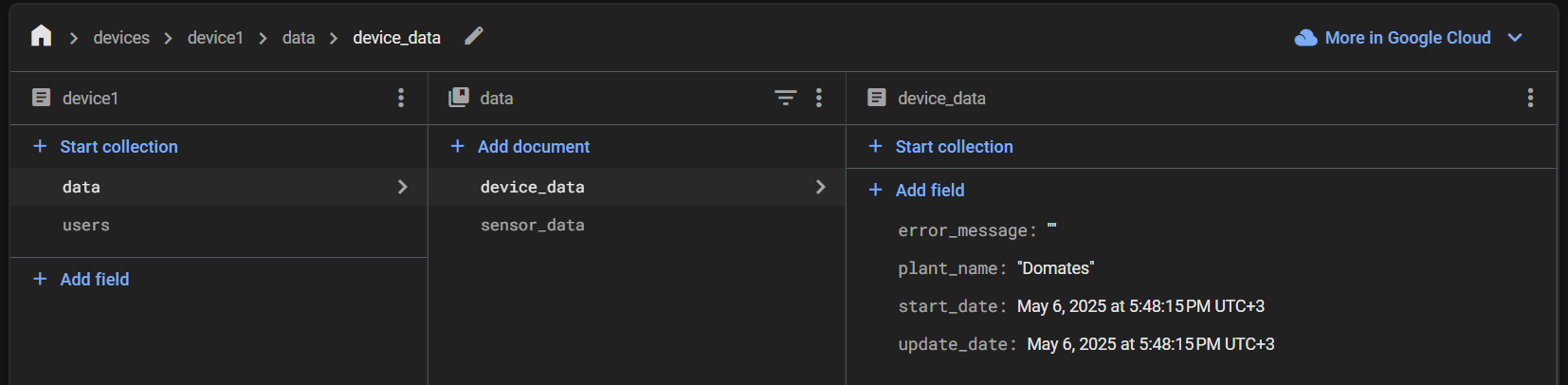
### Şekil 1.2. ‘avaliable\_plants’ Koleksiyonu

Cihazın bitki yetiştirme sırasında topladığı sensör verileri, ‘devices’ koleksiyonu altında cihazın kimlik numarası adındaki (örn: device1, device2) doküman altında data koleksiyonu altında sensor\_data dokümanı altındaki sensör alanlarında tutulur.



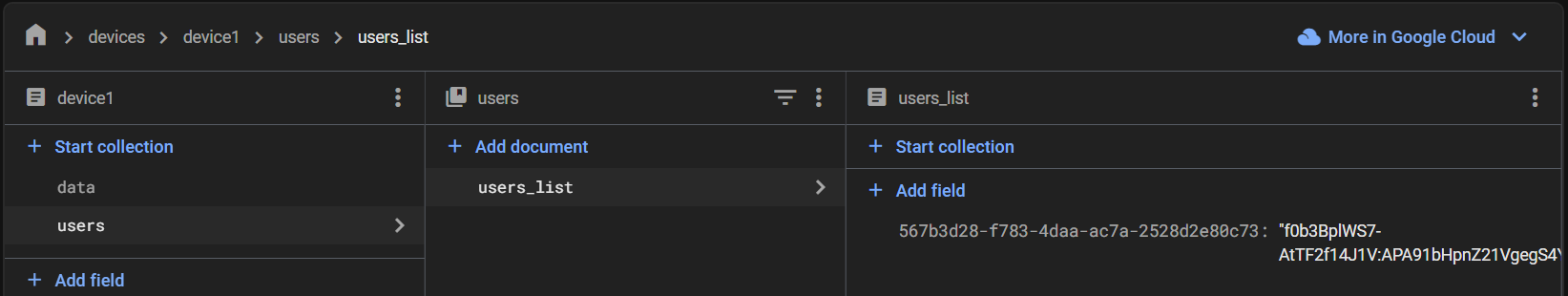
### Şekil 1.3. ‘sensor\_data’ Dokümanı

Cihazın bitki yetiştirmesi ile ilgili veriler, örneğin hata mesajları, bitkinin yetiştirilmeye başlandığı tarih, sensör verilerinin güncellendiği tarih, yetiştirilen bitki gibi, ‘device\_data’ dokümanındaki alanlarda tutulur.



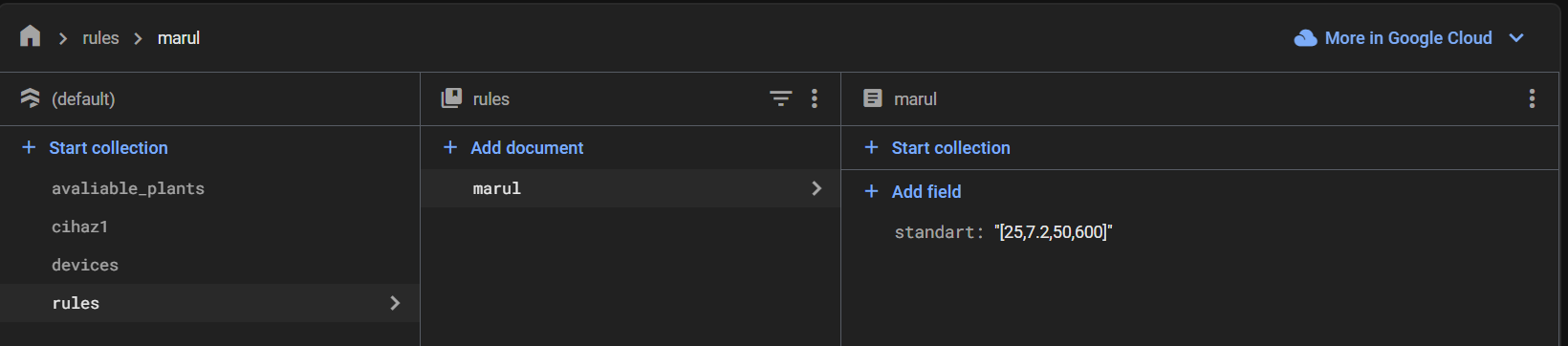
### Şekil 1.4. ‘device\_data’ Dokümanı

Cihazın yazdığı hata mesajlarının bildirim olarak gönderilmesi için gereken ‘FCM Token’ anahtarları ‘users’ koleksiyonu altında ‘users\_list’ dokümanı altında mobil uygulama kimlik numarası alanında tutulur.



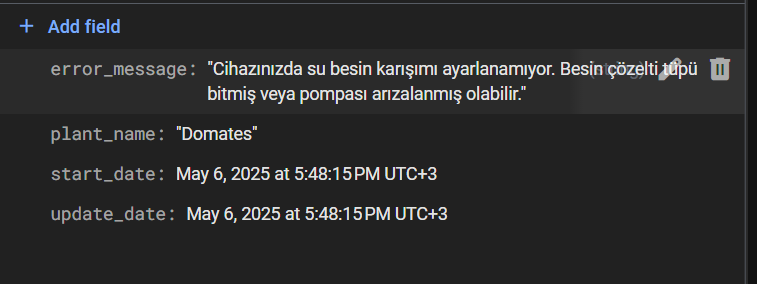
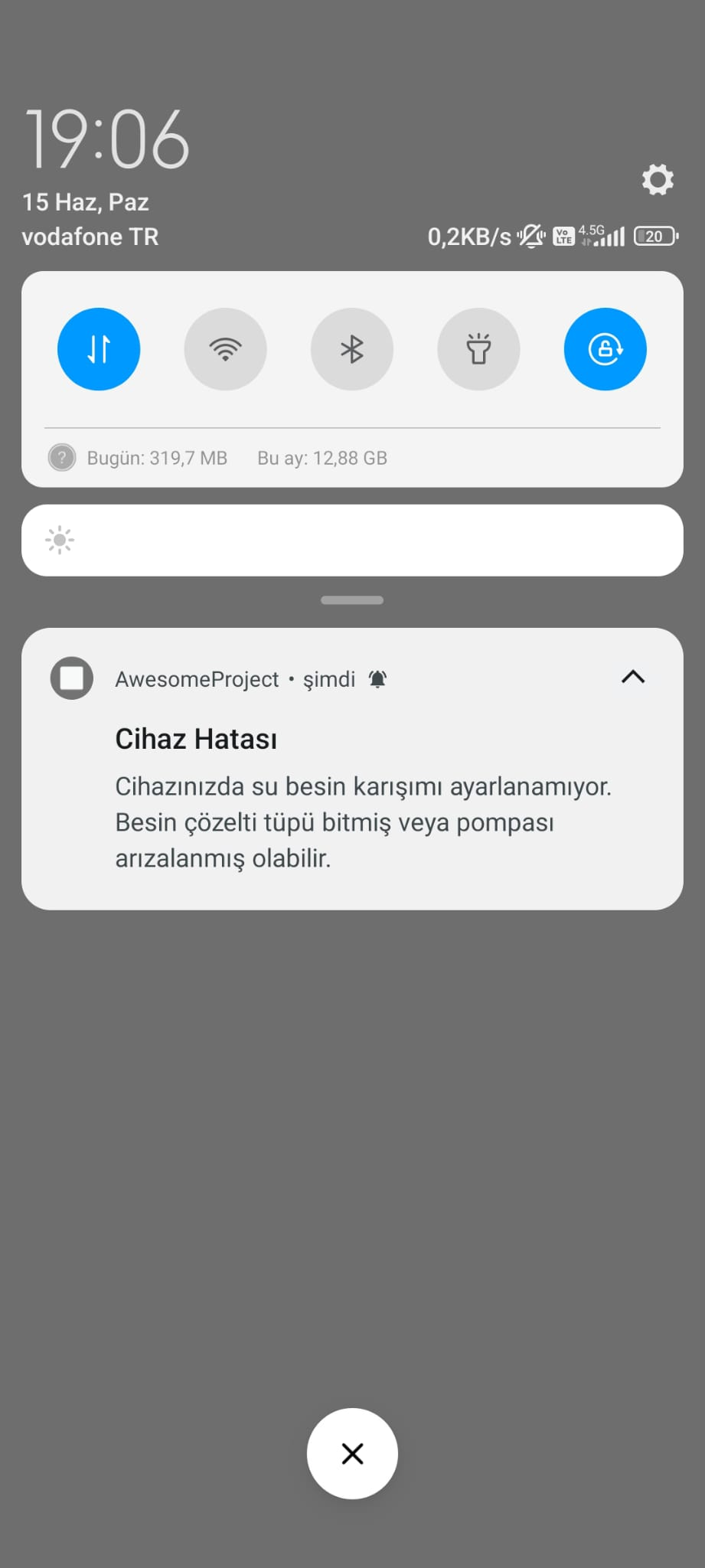
### Şekil 1.5. ‘users’ Koleksiyonu

Cihaz, içerisinde yetiştirilen bitki için daha önceden yapay zeka modeli tarafından belirlenmiş ve veritabanına yazılmış en uygun ortam parametrelerini ‘rules’ koleksiyonu altındaki bitkinin adi dokümanındaki alandan alır. Örneğin şekildeki durumda cihaz, sıcaklığı 25, pH’ı 7,2, nemi %50’ de tutmaya çalışacaktır.



### Şekil 1.6. ‘rules’ Koleksiyonu

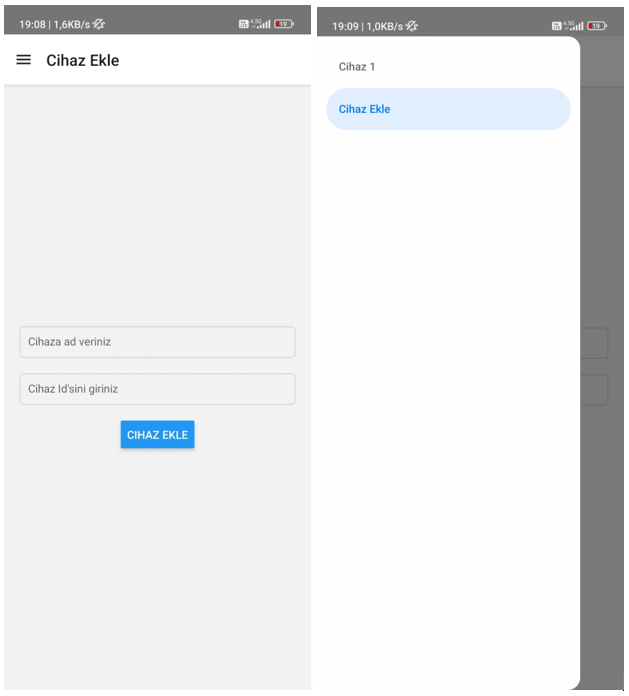
Veritabanı tetikleyicisi Google Cloud Functions’ta oluşturulmuştur. Tetikleyici hata mesajının tutulduğu ‘error\_message’ değiştiğinde tetiklenir ve hata mesajı gönderen cihazın ‘users\_list’ dokümanında tutulan kullanıcıların ‘FCM Token’ verilerini alarak bu veriler üzerinden ilgili kullanıcı uygulamalarına hata mesajını bildirim olarak yollar.

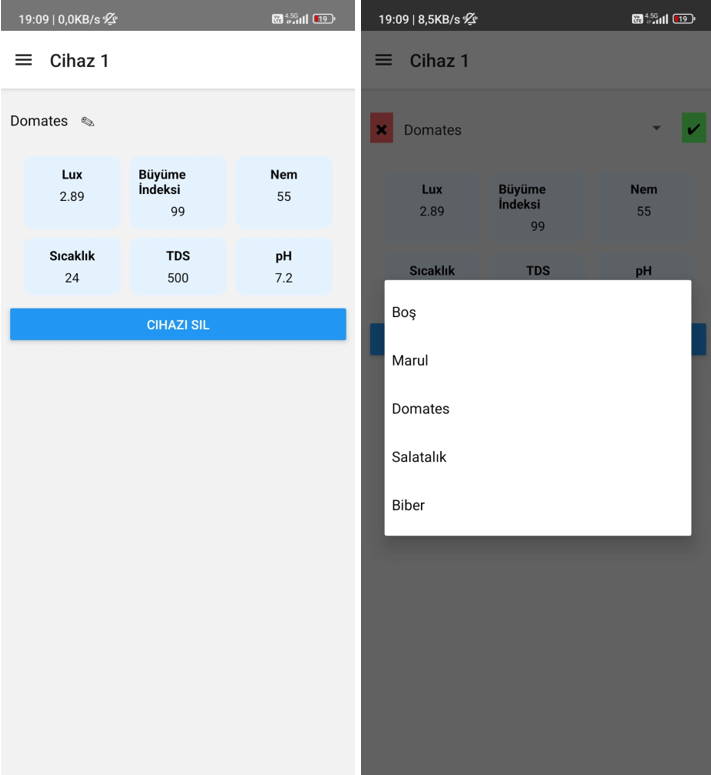
### Şekil 1.7. Bildirim

# MOBİL UYGULAMA

Mobil uygulama iki ekran ve bir menüden oluşur. İlk ekran cihaz ekleme ekranıdır. Bu ekranda eklenecek topraksız tarım cihazına isim verme ve cihazın kimlik numarası girme alanları ve ‘CIHAZ EKLE’ butonu vardır. Butona basıldığında alanlara girilen veriler mobil cihazın yerel depolamasına kaydedilir ve menüye eklenir.

Menü uygulama ilk açıldığında yerel depolamadan alınan verilerle oluşturulur. Menüde eklenmiş cihazlar ve cihaz ekleme sayfası listelenir sayfalar arası geçiş menü üzerinden yapılır.

### Şekil 2.1. Cihaz Ekleme Sayfası ve Menü

Eklenmiş cihazın sayfasında cihazda yetiştirilen bitki bilgisi ve değiştirme butonu, cihazın ortam verilerinin görüntülendiği kutucuklar ve cihazı silme butonu yer alır. Yetiştirilen bitkiyi değiştirme butonun basıldığında yetiştirilen bitki verisinin sağ tarafında onaylama butonu, sol tarafında iptal etme butonu yer alır. Bu durumda yetiştirilen bitkiye tıklandığında bir liste açılır ve buradan yeni yetiştirilecek bitki seçilir. Seçimden sonra onayla butonuna basıldığında bu veri veritabanına yollanır ve listelenen sensör bilgileri sıfırlanır. Ekran değiştirme butonuna basılmadan önceki durumuna döner. Onayla butonu yerine iptal et butonuna basıldığında eski haline döner. Cihazı sil butonuna basıldığında cihaz verileri yerel depolamadan silinir ve görüntülenen ekran menünün en üstündeki ekran olur.

### Şekil 2.2. Cihaz Ekranı ve Yetiştirilecek Bitki Seçimi

# MOBİL UYGULAMANIN KODU

Mobil uygulamanın kodu dört farklı bölüm ve dosyadan oluşur. Bu dosyalar ‘App.jsx’, ‘AddDeviceScreen.jsx’, ‘DeviceScreen.js’, ‘EditableItem.jsx’ olarak adlandırılmıştır. ‘AddDeviceScreen.jsx’ ve ‘DeviceScreen.js’, ‘App.jsx’ dosyası içerisinde kullanılmak üzere içe aktarılmıştır (import). ‘EditableItem.jsx’ ise ‘DeviceScreen.js’ içerisinde kullanılmıştır.

## EditableItem.jsx

‘EditableItem.jsx’ özet olarak cihaz sayfalarındaki yetiştirilen bitki kısmını ve bu verinin liste üzerinden seçilerek değiştirilmesi fonksiyonlarını oluşturur. Bu kodun içindeki ‘EditableItem’ fonksiyonu argüman olarak ‘value’, ‘options’ değişkenlerini ve ‘onConfirm’ fonksiyonunu alır. Kodun başında ‘isEditing’ durum değişkeni (state değişkeni) ve bu değişkeni değiştiren ‘SetIsEditing’ fonksiyonu oluşturulmuştur. Durum değişkeninin başlangıç değeri ‘false’ olarak ayarlanmıştır. Durum değişkeni ‘false’ olduğu zaman bu dosyaya argüman olarak gelen yetiştirilen bitki adını tutan ‘value’ değişkeni ekrana ‘text’ olarak basılır. Sağına da düzenleme işareti butonu eklenir. Bu butona basıldığında ‘handleEdit’ fonksiyonu çağırılır. Bu fonksiyon ‘setIsEditing’ fonksiyonu üzerinden ‘isEditing’ durum değişkenini ‘true’ yapar. ‘React Hook’ mantığından dolayı durum değişkeni değiştiğinde ilgili bileşen tekrar ‘render’ edilir. Bu bileşen ‘isEditing’ ‘true’ olduğundan dolayı iptal etme butonu, ‘options’ değişkeninden gelen verilerle bir liste (picker) ve onaylama butonu olarak ‘return’ edilir. Onaylama butonuna basıldığında ‘handleConfirm’ fonksiyonunu çağırılır. Bu fonksiyon liste seçimini argüman olarak gelen ‘onConfirm’ fonksiyonuna ileterek çalıştırır.

### Şekil 3.1. ‘EditableItem.jsx’

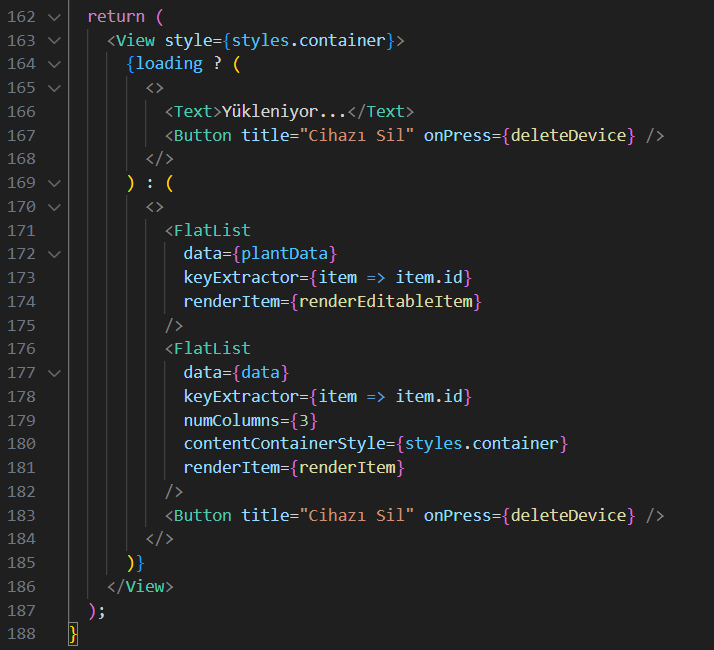
## DeviceScreen.js

‘DeviceScreen.js’, özet olarak cihaz ekranlarının oluşturur. Dosyada yazılı ‘DeviceScreen’ fonksiyonu argüman olarak ‘deviceId’, ‘options’, ‘persistentId’ değşkenlerini ve ‘refreshDevices’ fonksiyonunu alır.

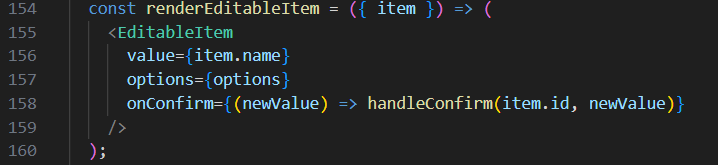
### Şekil 3.2. ‘DeviceScreen’ Fonksiyonu Argümanları

‘useEffect’ kısmında veritabanından sensör verilerini ve bitki adını alır ve dinleme (onSnapshot) başlatır.

### Şekil 3.3. ‘useEffect’ Bölümü

Fonksiyon ‘return’ olarak ‘renderEditableItem’, ‘renderItem’ ve cihazı sil butonu döndürür.

### Şekil 3.4. ‘DeviceScreen’ Fonksiyonunun ‘return’ Bölümü

‘renderEditableItem’ fonksiyonu diğer dosyada bulunan ‘EditableItem’ı gerekli argümanlarıyla çalıştırarak bileşeni oluşturtur. Burada ‘onConfirm’ adı altında argüman olarak geçirilen fonksiyon ‘handleConfirm’dür.

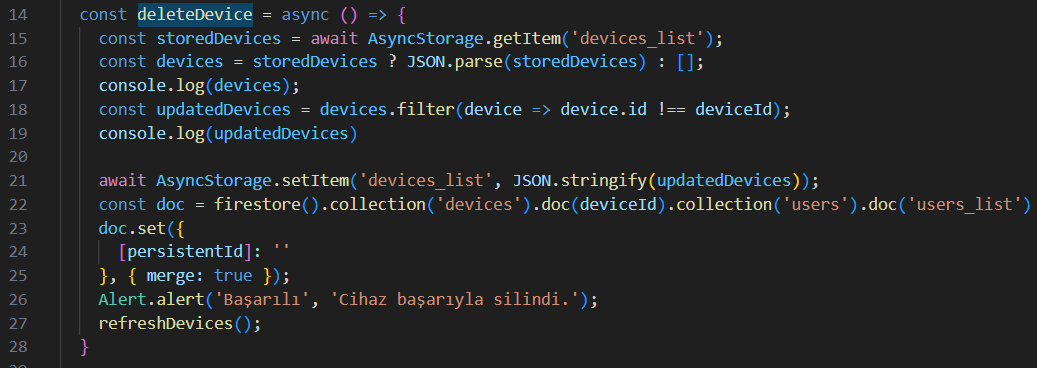
### Şekil 3.5. ‘renderEditableItem’ Fonksiyonu

Eğer bileşendeki ‘onConfirm’ ve dolayısıyla ‘handleConfirm’ çalışırsa seçilen yeni bitki ve başlangıç tarihi gibi veriler veritabanına yollanır. Aynı zamanda yeni bir bitki yetiştirilmeye başlandığından sensör verileri silinir.

### Şekil 3.6. ‘handleConfirm’ Fonksiyonu

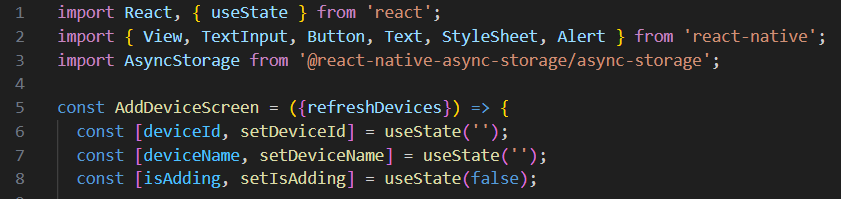
‘renderItem’ fonksiyonu veritabanından çekilmiş olan sensör verilerini kutu yapısında oluşturur.

### Şekil 3.7. ‘renderItem’ Fonksiyonu

Cihazı sil butonuna basıldığında ‘deleteDevice’ fonksiyonu çağırılır. Bu fonksiyon ‘AsyncStorage’ ile cihazın verilerini yerel depolamadan siler. Aynı zamanda veritabanında cihazların kullanıcılarının listelendiği ‘users\_list’ten de kendi verilerini siler.

### Şekil 3.8. ‘deleteDevice’ Fonksiyonu

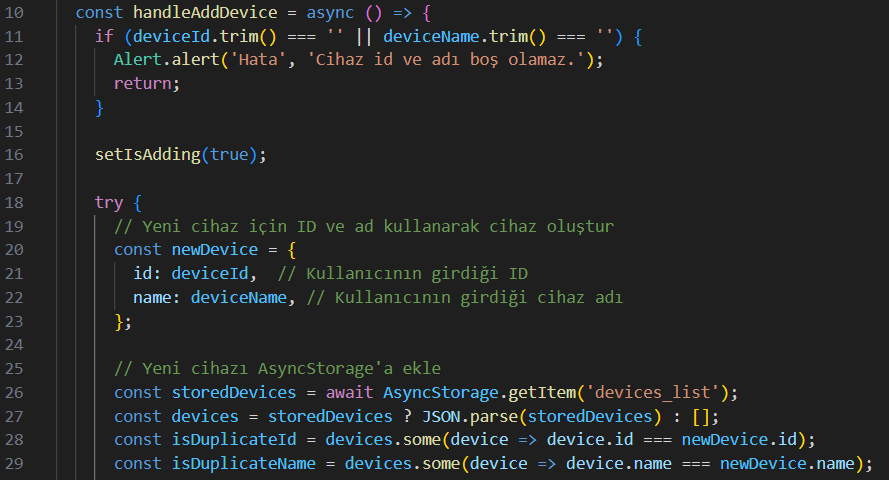
## AddDeviceScreen.jsx

AddDeviceScreen.jsx, cihaz ekleme sayfasını oluşturur. Koddaki ‘AddDeviceScreen’ fonksiyonu argüman olarak ‘refreshDevices’ fonksiyonunu alır.

### Şekil 3.9. ‘AddDeviceScreen’ Fonksiyonu Argümanı

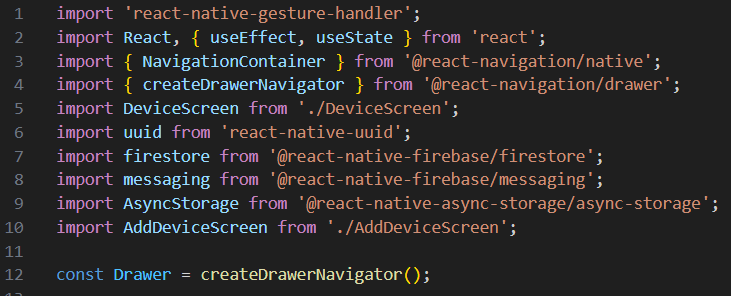
Fonksiyon ‘return’ olarak cihaza verilecek ismin girildiği ve cihazın kimlik numarasının girildiği iki ‘TextInput’ oluşturur. Ek olarak cihaz ekleme butonu oluşturur.

### Şekil 3.10. ‘AddDeviceScreen’ Fonksiyonu ‘return’ Kısmı

Alanlar doldurulduktan sonra butona basıldığında ‘handleAddDevice’ fonksiyonu çağırılır. Bu fonksiyon alana girilen bölgelerin boş olup olmamasını ve hali hazırda yerel depolamada bu isimde veya kimlik numarasına sahip veri olup olmadığının kontrolünü yapar. Sorun yoksa verileri yerel depolamaya kaydeder ve bileşenlerin güncellenmesi için argüman olarak aldığı ‘refreshDevices’ fonksiyonunu çağırır.

### Şekil 3.11. ‘handleAddDevice’ Fonksiyonu

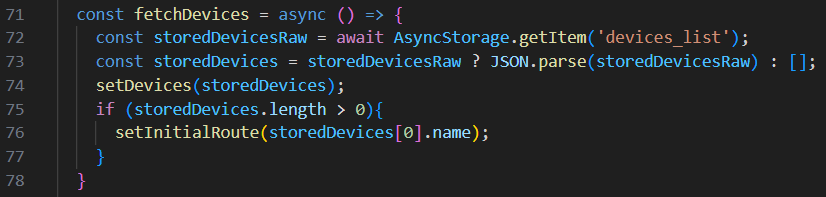
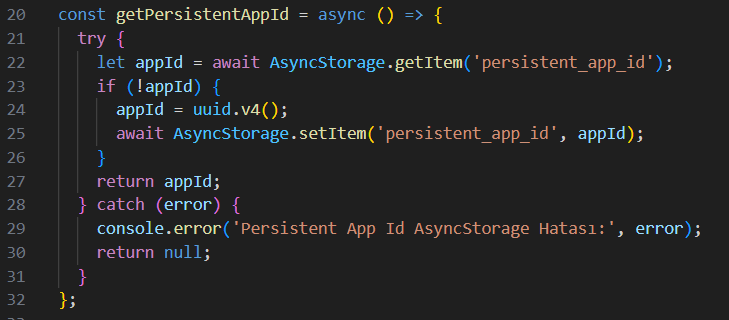
## App.jsx

‘App.jsx’ içinde ‘import’lar başlangıçta menüyü oluşturan ‘Drawer’ tanımlanır.

### Şekil 3.12. ‘import’lar ve ‘Drawer’ Tanımlanması

‘App’ fonksiyonu içerisinde çeşitli durum değişkenlerini oluşturur.

### Şekil 3.13. Tanımlanan Durum Değişkenleri

Yerel depolamadan verilerin alındığı ‘fetchDevices’ fonksiyonu ve veritabanına ‘FCM Token’ gönderiminde kullanılacak uygulama kimlik numarasını yerel depolamadan varsa alan yoksa yerel depolamada oluşturan ‘getPersistentAppId’ fonksiyonu tanımlanır.

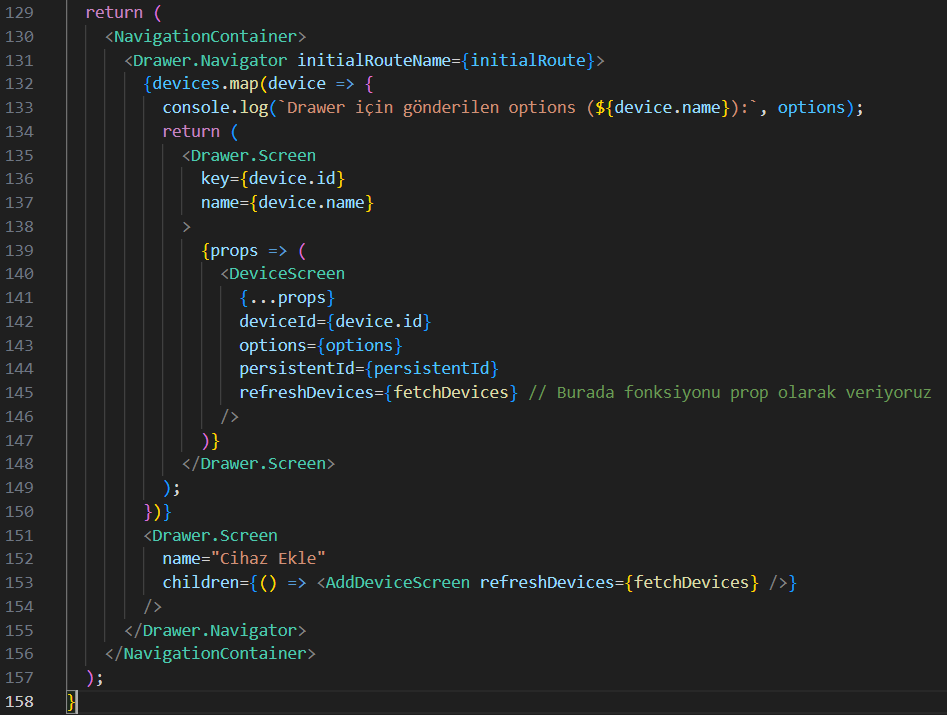
### Şekil 3.14. ‘fetchDevices’ ve ‘getPersistentAppId’ Fonksiyonu

Veritabanından yetiştirilebilir bitki listesi olan ‘avaliable\_plants’ alanından listeyi alan ‘getOptionsFromFirebase’ fonksiyonu tanımlanır.

### Şekil 3.15. ‘getOptionsFromFirestore’ Fonksiyonu

Tüm bu tanımlanan fonksiyonlar ‘useEffect’ içerisinde çağırılır. Ek olarak ‘useEffect’ içerisinde ‘FCM Token’ alınarak veritabanına ‘persistentAppId’ alan adıyla gönderilir. Bu useEffect’in tetikleyicisi olarak ‘devices’ atanır. Yani ‘devices’ değişkeni değiştiğinde ‘useEffect’ tekrar çağırılır.

### Şekil 3.16. ‘useEffect’ Bölümü

‘App’ fonksiyonu ‘return’ olarak ‘devices’ değişkeni ile menü elemanları olan ‘Drawer.Screen’ler döndürür. Cihazların ekranları olan ‘Drawer.Screen’ler içerisinde ‘DeviceScreen’ fonksiyonu, fonksiyona ‘prop’ olarak geçirilmiş argümanlarıyla birlikte, çağırılarak cihaz ekranları oluşturtulmuş olur. Cihaz ekleme sayfası için oluşturulmuş ‘Drawer.Screen’ içerisinde ise ‘AddDeviceScreen’ fonksiyonu çağırılarak cihaz ekleme sayfası oluşturtulur. Burada her iki fonksiyonun ‘refreshDevices’ argümanı olarak ‘fetchDevices’ fonksiyonu geçirilir.

### Şekil 3.17. ‘App’ Fonksiyonunun ‘return’ Bölümü

Bu sayede kod tamamlanmış olur.