

**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/HXvbVlc8L_I?feature=share

Haziran 2025

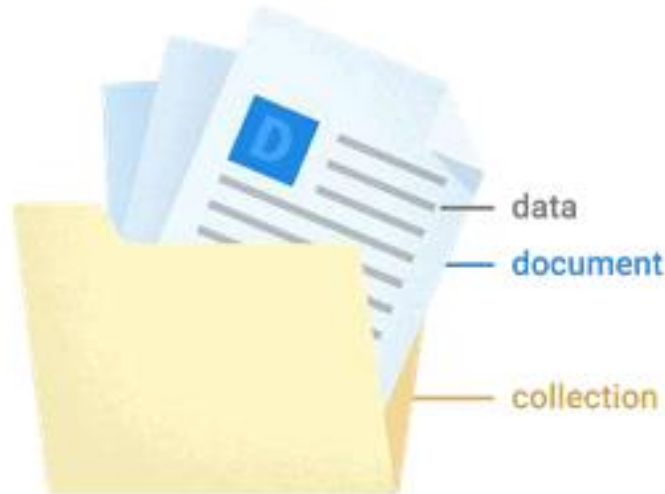
1. AÇIKLAMA

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 yoğunlaşmıştır.

2. VERİTABANI VE BİLDİRİM TETİKLEYİCİSİ

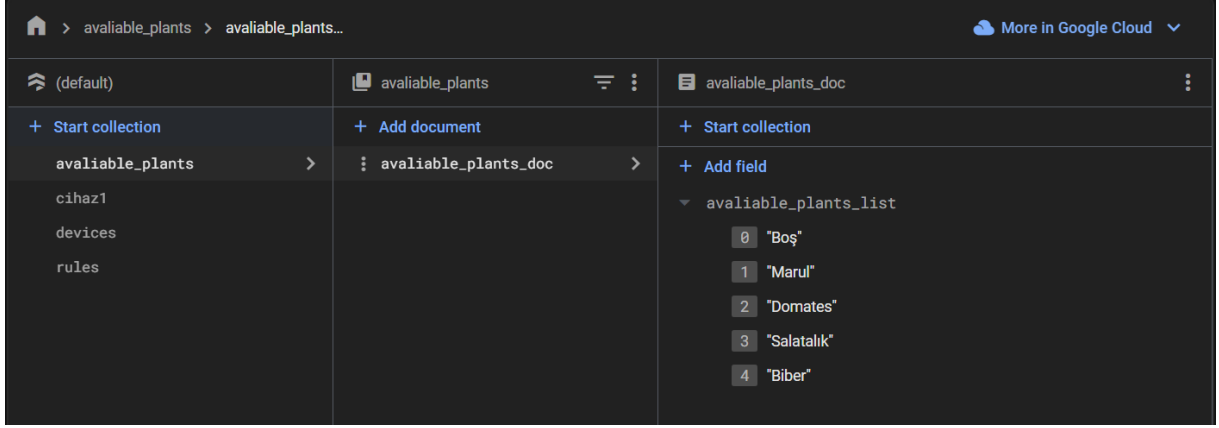
Projenin amacı içerisinde bitki yetiştirilmesine uygun ortam hazırlayan, sensörler ve kontrol elemanlarıyla donatılı bir topraksız tarım cihazı ile cihaz kullanıcısı arasında bir iletişim sağlamak için bir mobil uygulama geliştirmektir. Bu bağlamda, kullanıcı cihaz içerisinde yetiştirdiği bitki verisini (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.



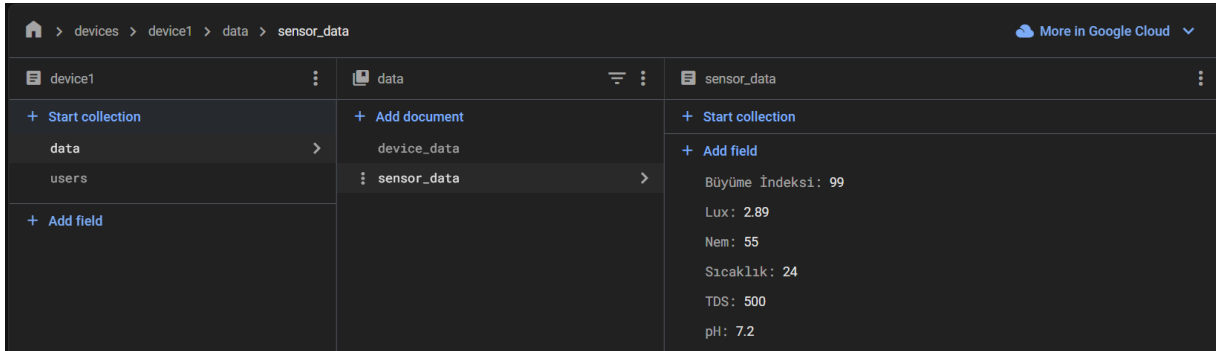
Şekil 1.1. Firebase Firestore Veri Yapısı

Veritabanında kullanıcının seçebileceği bitkiler listesi 'available_plants' koleksiyonu altında 'available_plants_doc' dokümanı altında 'available_plants_list' alanında liste olarak tutulur.



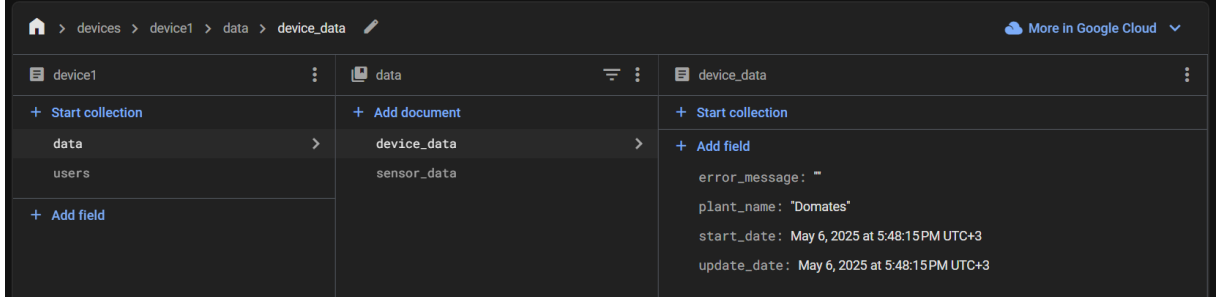
Şekil 1.2. 'available_plants' Koleksiyonu

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.



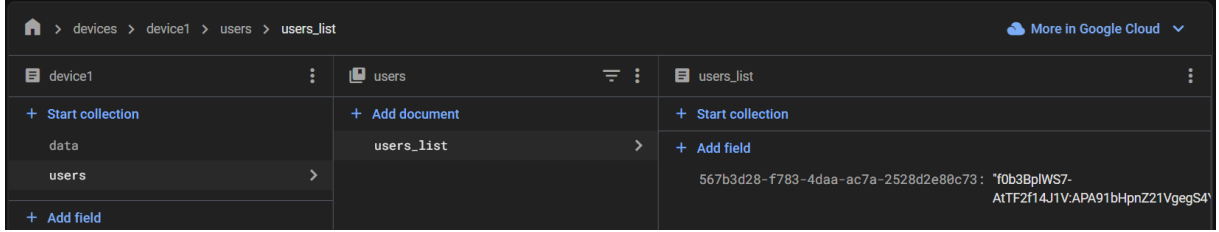
Şekil 1.3. 'sensor_data' Dokümanı

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.



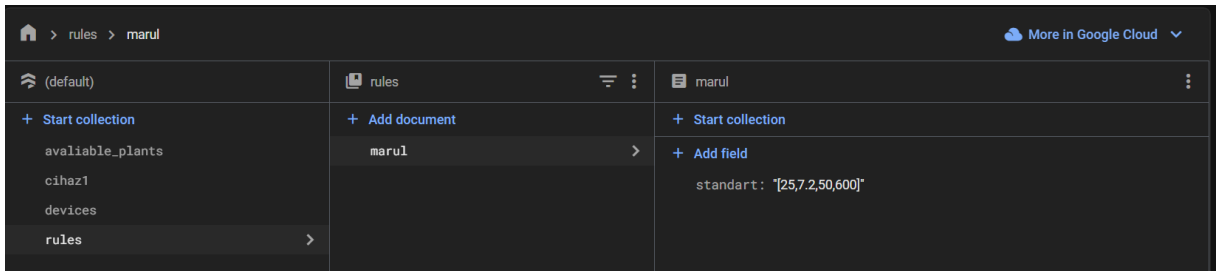
Şekil 1.4. 'device_data' Dokümanı

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

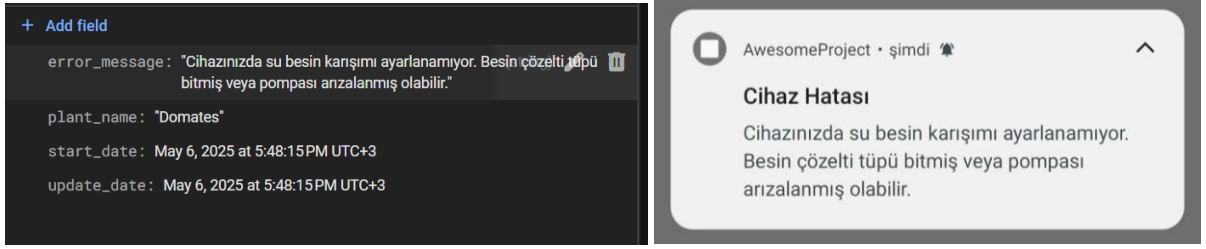
Cihaz, içerisinde yetiştirilen bitki için daha önceden yapay zeka modeli tarafından belirlenmiş ve veritabanına yazılmış en uygun ortam parametrelerini 'rules' koleksiyonu altındaki bitkinin adı dokümanındaki alandan alır. Örneğin şekildeki durumda cihaz, sıcaklığı 25, pH'ı 7,2, nemi %50' de tutmaya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

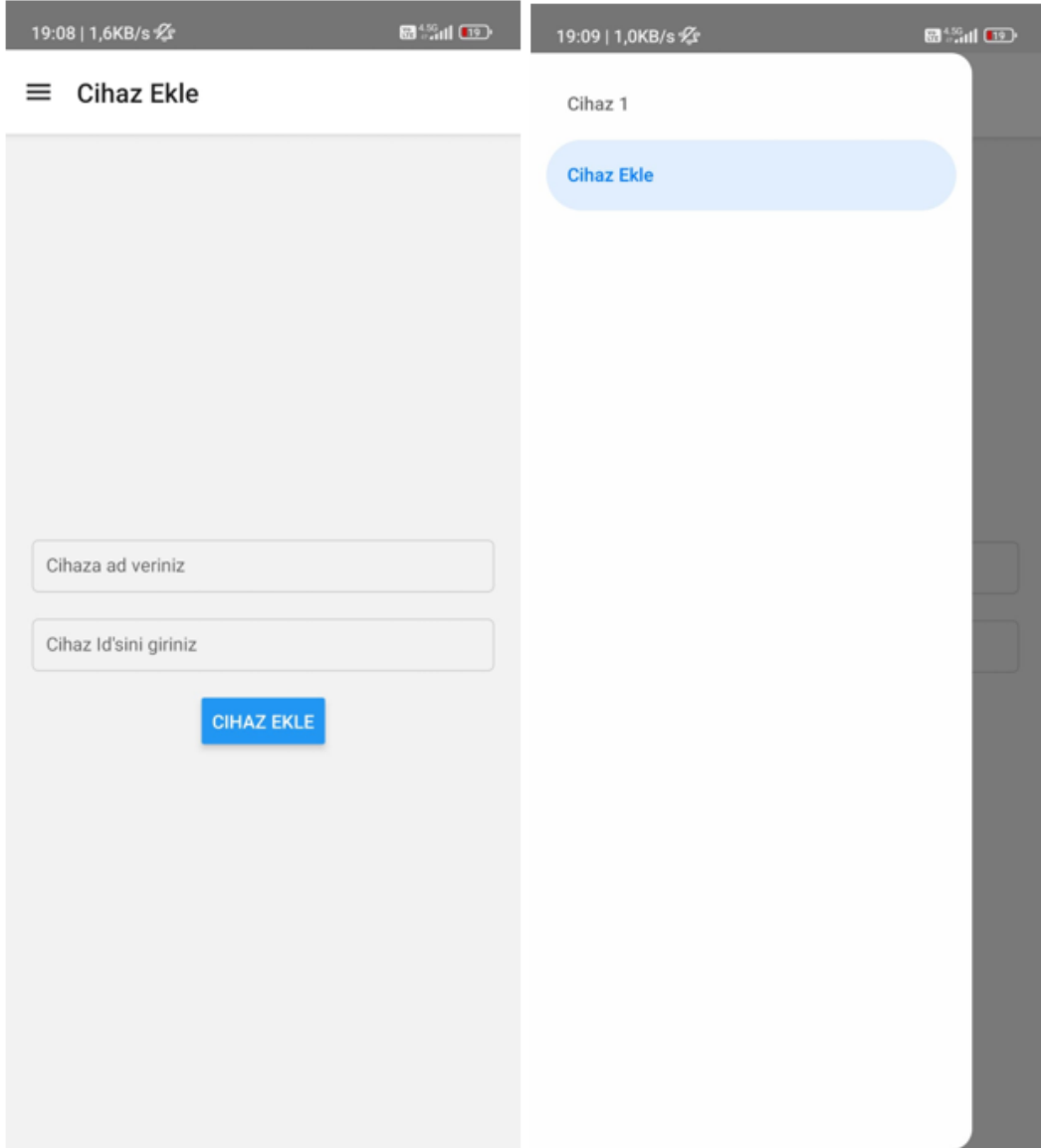


Şekil 1.7. Bildirim

3. MOBİL UYGULAMA

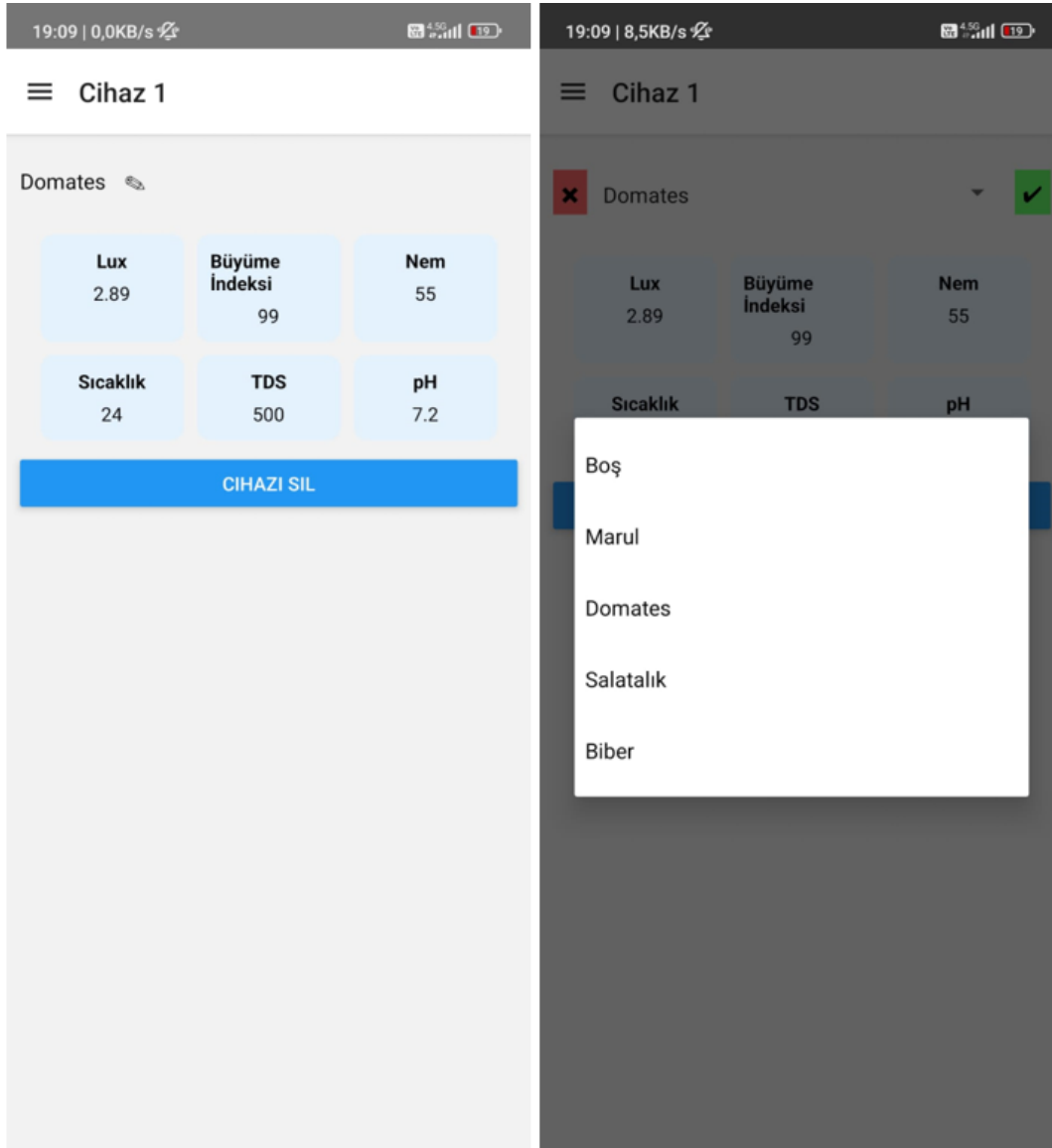
Mobil uygulama iki ekran ve bir menüden oluşur. İlk ekran cihaz ekleme ekranıdır. Bu ekranda eklenecek topraksız tarım cihazına isim verme ve cihazın kimlik numarası girme alanları ve 'CIHAZ EKLE' butonu vardır. Butona basıldığında alanlara girilen veriler mobil cihazın yerel depolamasına kaydedilir 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 görüntülediği kutucuklar ve cihazı silme butonu yer alır. Yetiştirilen bitkiyi değiştirme butonuna 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 görüntülen ekran menünün en üstündeki ekran olur.



Şekil 2.2. Cihaz Ekranı ve Yetiştirilecek Bitki Seçimi

4. 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.

4.1. 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.

```

1  // EditableItem.js
2
3  import React, { useState } from 'react';
4  import { View, Text, TouchableOpacity, StyleSheet } from 'react-native';
5  import { Picker } from '@react-native-picker/picker';
6
7  const EditableItem = ({ value, options, onConfirm }) => {
8    const [isEditing, setIsEditing] = useState(false);
9    const [tempValue, setTempValue] = useState(value);
10   console.log("editable'da tempValue");
11   console.log(tempValue);
12
13   const handleEdit = () => setIsEditing(true);
14   const handleCancel = () => {
15     setTempValue(value);
16     setIsEditing(false);
17   };
18   const handleConfirm = () => {
19     console.log("editable handle confirm çalıştı");
20     onConfirm(tempValue);
21     setIsEditing(false);
22   };
23
24   return (
25     <View style={styles.container}>
26       {isEditing ? (
27         <>
28           <TouchableOpacity onPress={handleCancel} style={styles.cancelButton}>
29             <Text style={styles.buttonText}><X</Text>
30           </TouchableOpacity>
31
32           <Picker
33             selectedValue={tempValue}
34             style={styles.picker}
35             onChange={itemValue => setTempValue(itemValue)}
36           >
37             {options.map(option => (
38               <Picker.Item key={option} label={option} value={option} />
39             ))}
40           </Picker>
41
42           <TouchableOpacity onPress={handleConfirm} style={styles.confirmButton}>
43             <Text style={styles.buttonText}><✓</Text>
44           </TouchableOpacity>
45         </>
46       ) : (
47         <>
48           <Text style={styles.label}><{tempValue}</Text>
49           <TouchableOpacity onPress={handleEdit} style={styles.editButton}>
50             <Text style={styles.buttonText}><✎</Text>
51           </TouchableOpacity>
52         </>
53       )}
54     </View>
55   );
56 };

```

Şekil 3.1. 'EditableItem.jsx'

4.2. DeviceScreen.js

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

```

7 export default function DeviceScreen({ deviceId, options, persistentId, refreshDevices }) {
8   // const {deviceId, options} = route.params;
9   const [data, setData] = useState([]);
10  const [plantData, setplantData] = useState([ { id: '1', name: 'None' } ]);
11  const [loading, setLoading] = useState(true); // Yükleniyor durumu
12  // const options = ['Boş', 'Marul', 'Domates', 'Salatalık', 'Biber'];

```

Şekil 3.2. 'DeviceScreen' Fonksiyonu Argümanları

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

```

41  useEffect(() => {
42    console.log("useEffect çalıştırıldı.");
43
44    // Firestore'dan gerçek zamanlı veriyi dinle
45    const getSensorData = firestore()
46      .collection("devices")
47      .doc(deviceId)
48      .collection("data")
49      .doc("sensor_data")
50      .onSnapshot(snapshot => {
51        if (snapshot.exists) {
52          const rawData = snapshot.data();
53          const formattedData = Object.keys(rawData).map(key => {
54            const value = rawData[key];
55            const formattedValue = key === 'tarih' && value
56              ? new Date(value._seconds * 1000).toLocaleString()
57              : value;
58
59            return {
60              id: key,
61              name: key,
62              value: formattedValue,
63            };
64          });
65          console.log(formattedData);
66          setData(formattedData);
67          console.log(data);
68        }
69      });
70
71    const getPlantName = firestore()
72      .collection("devices")
73      .doc(deviceId)
74      .collection("data")
75      .doc("device_data")
76      .onSnapshot(snapshot => {
77        if (snapshot.exists) {
78          const data = snapshot.data();
79          const plantName = data?.plant_name || 'None';
80          setplantData(prevData =>
81            prevData.map(item =>
82              item.id === '1' ? { ...item, name: plantName } : item
83            )
84          );
85          setLoading(false); // Veriler yüklendi
86        }
87      });
88
89    // Component unmount edildiğinde dinleyicileri temizle
90    return () => {
91      getSensorData();
92      getPlantName();
93    };

```

Şekil 3.3. 'useEffect' Bölümü

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

```
162  return (
163    <View style={styles.container}>
164      {loading ? (
165        <>
166          <Text>Yükleniyor...</Text>
167          <Button title="Cihazı Sil" onPress={deleteDevice} />
168        </>
169      ) : (
170        <>
171          <FlatList
172            data={plantData}
173            keyExtractor={item => item.id}
174            renderItem={renderEditableItem}
175          />
176          <FlatList
177            data={data}
178            keyExtractor={item => item.id}
179            numColumns={3}
180            contentContainerStyle={styles.container}
181            renderItem={renderItem}
182          />
183          <Button title="Cihazı Sil" onPress={deleteDevice} />
184        </>
185      )}
186    </View>
187  );
188 }
```

Şekil 3.4. 'DeviceScreen' Fonksiyonunun 'return' Bölümü

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

```
154  const renderEditableItem = ({ item }) => (
155    <EditableItem
156      value={item.name}
157      options={options}
158      onConfirm={newValue => handleConfirm(item.id, newValue)}
159    />
160  );
```

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.

```

104 const handleConfirm = async (id, newValue) => {
105   console.log("handle confirm çalıştı");
106
107   // UI verisini güncelle
108   setData(prev =>
109     prev.map(item =>
110       item.id === id ? { ...item, name: newValue } : item
111     )
112   );
113
114   try {
115     // Firestore'a plant_name güncelle
116     await firestore()
117       .collection("devices")
118       .doc(deviceId)
119       .collection("data")
120       .doc("device_data")
121       .set({
122         plant_name: newValue,
123         start_date: firestore.FieldValue.serverTimestamp(),
124         update_date: firestore.FieldValue.serverTimestamp()
125       }, { merge: true });
126
127     console.log('bitkiadi başarıyla güncellendi veya oluşturuldu!');
128
129     // sensor_data alanlarını temizle
130     const sensorDataRef = firestore()
131       .collection("devices")
132       .doc(deviceId)
133       .collection("data")
134       .doc("sensor_data");
135
136     const sensorDataDoc = await sensorDataRef.get();
137
138     if (sensorDataDoc.exists) {
139       const data = sensorDataDoc.data();
140       const clearedFields = {};
141
142       for (const key in data) {
143         clearedFields[key] = null;
144       }
145
146       await sensorDataRef.update(clearedFields);
147       console.log("Sensor verileri temizlendi.");
148     }
149   } catch (error) {
150     console.error('Hata:', error);
151   }
152 };

```

Şekil 3.6. 'handleConfirm' Fonksiyonu

'renderItem' fonksiyonu veritabanından çekilmiş olan sensör verilerini kutu yapısında oluşturur.

```

97   const renderItem = ({ item }) => (
98     <View style={styles.box}>
99       <Text style={styles.title}>{item.name}</Text>
100      <Text>{item.value}</Text>
101    </View>
102  );

```

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.

```

14   const deleteDevice = async () => {
15     const storedDevices = await AsyncStorage.getItem('devices_list');
16     const devices = storedDevices ? JSON.parse(storedDevices) : [];
17     console.log(devices);
18     const updatedDevices = devices.filter(device => device.id !== deviceId);
19     console.log(updatedDevices)
20
21     await AsyncStorage.setItem('devices_list', JSON.stringify(updatedDevices));
22     const doc = firestore().collection('devices').doc(deviceId).collection('users').doc('users_list')
23     doc.set({
24       [persistentId]: ''
25     }, { merge: true });
26     Alert.alert('Başarılı', 'Cihaz başarıyla silindi. ');
27     refreshDevices();
28   }

```

Şekil 3.8. 'deleteDevice' Fonksiyonu

4.3. AddDeviceScreen.jsx

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

```

1   import React, { useState } from 'react';
2   import { View, TextInput, Button, Text, StyleSheet, Alert } from 'react-native';
3   import AsyncStorage from '@react-native-async-storage/async-storage';
4
5   const AddDeviceScreen = ({refreshDevices}) => {
6     const [deviceId, setDeviceId] = useState('');
7     const [deviceName, setDeviceName] = useState('');
8     const [isAdding, setIsAdding] = useState(false);

```

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.

```
65     return (  
66       <View style={styles.container}>  
67         <TextInput  
68           style={styles.input}  
69           placeholder="Cihaza ad veriniz"  
70           value={deviceName}  
71           onChangeText={text => setDeviceName(text)}  
72         />  
73         <TextInput  
74           style={styles.input}  
75           placeholder="Cihaz Id'sini giriniz"  
76           value={deviceId}  
77           onChangeText={text => setDeviceId(text)}  
78         />  
79         <Button  
80           title={isAdding ? 'Ekleniyor...' : 'Cihaz Ekle'}  
81           onPress={handleAddDevice}  
82           disabled={isAdding} // Cihaz eklenirken butonu devre dışı bırak  
83         />  
84         {isAdding && <Text>Ekleniyor...</Text>}  
85       </View>  
86     );  
87   };  
88 };
```

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.

```
10 const handleAddDevice = async () => {  
11   if (deviceId.trim() === '' || deviceName.trim() === '') {  
12     Alert.alert('Hata', 'Cihaz id ve adı boş olamaz.');13     return;  
14   }  
15   setIsAdding(true);  
16   try {  
17     // Yeni cihaz için ID ve ad kullanarak cihaz oluştur  
18     const newDevice = {  
19       id: deviceId, // Kullanıcının girdiği ID  
20       name: deviceName, // Kullanıcının girdiği cihaz adı  
21     };  
22     // Yeni cihazı AsyncStorage'a ekle  
23     const storedDevices = await AsyncStorage.getItem('devices_list');  
24     const devices = storedDevices ? JSON.parse(storedDevices) : [];  
25     const isDuplicateId = devices.some(device => device.id === newDevice.id);  
26     const isDuplicateName = devices.some(device => device.name === newDevice.name);  
27   }  
28 };
```



```

30     if (isDuplicateId) {
31         Alert.alert('Hata', 'Bu id ile zaten bir cihaz eklenmiş.');
```

```

32         setDeviceId('');
33         setDeviceName('');
34         return;
35     }
36     else if (isDuplicateName) {
37         Alert.alert('Hata', 'Bu ad ile zaten bir cihaz eklenmiş.');
```

```

38         setDeviceId('');
39         setDeviceName('');
40         return;
41     }
42
43     devices.push(newDevice);
44
45     // Yeni cihazları AsyncStorage'a kaydet
46     await AsyncStorage.setItem('devices_list', JSON.stringify(devices));
47
48     // Başarılı ekleme mesajı
49     Alert.alert('Başarılı', 'Cihaz başarıyla eklendi.');
```

```

50
51     refreshDevices();
52
53     setDeviceId('');
54     setDeviceName('');
55
56 } catch (error) {
57     console.error('Cihaz eklerken hata oluştu:', error);
58     Alert.alert('Hata', 'Cihaz eklenirken bir hata oluştu.');
```

```

59 } finally {
60     setIsAdding(false);
61 }
62 };

```

Şekil 3.11. 'handleAddDevice' Fonksiyonu

4.4. App.jsx

'App.jsx' içinde 'import'lar başlangıçta menüyü oluşturan 'Drawer' tanımlanır.

```

1  import 'react-native-gesture-handler';
2  import React, { useEffect, useState } from 'react';
3  import { NavigationContainer } from '@react-navigation/native';
4  import { createDrawerNavigator } from '@react-navigation/drawer';
5  import DeviceScreen from './DeviceScreen';
6  import uuid from 'react-native-uuid';
7  import firestore from '@react-native-firebase/firestore';
8  import messaging from '@react-native-firebase/messaging';
9  import AsyncStorage from '@react-native-async-storage/async-storage';
10 import AddDeviceScreen from './AddDeviceScreen';
11
12 const Drawer = createDrawerNavigator();

```

Şekil 3.12. 'import'lar ve 'Drawer' Tanımlanması

'App' fonksiyonu içerisinde çeşitli durum değişkenlerini oluşturur.

```

64 export default function App() {
65   const [options, setOptions] = useState([]);
66   const [isReady, setIsReady] = useState(false); // hem token işleri hem options hazır mı
67   const [devices, setDevices] = useState([]);
68   const [initialRoute, setInitialRoute] = useState(null);
69   const [persistentId, setPersistentId] = useState(null);

```

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.

```

71 const fetchDevices = async () => {
72   const storedDevicesRaw = await AsyncStorage.getItem('devices_list');
73   const storedDevices = storedDevicesRaw ? JSON.parse(storedDevicesRaw) : [];
74   setDevices(storedDevices);
75   if (storedDevices.length > 0){
76     setInitialRoute(storedDevices[0].name);
77   }
78 }

20 const getPersistentAppId = async () => {
21   try {
22     let appId = await AsyncStorage.getItem('persistent_app_id');
23     if (!appId) {
24       appId = uuid.v4();
25       await AsyncStorage.setItem('persistent_app_id', appId);
26     }
27     return appId;
28   } catch (error) {
29     console.error('Persistent App Id AsyncStorage Hatası:', error);
30     return null;
31   }
32 };

```

Şekil 3.14. 'fetchDevices' ve 'getPersistentAppId' Fonksiyonu

Veritabanından yetiştirilebilir bitki listesi olan 'available_plants' alanından listeyi alan 'getOptionsFromFirebase' fonksiyonu tanımlanır.

```

44 const getOptionsFromFirestore = async () => {
45   try {
46     const doc = await firestore()
47       .collection("avaliable_plants")
48       .doc("avaliable_plants_doc")
49       .get();
50
51     if (doc.exists) {
52       const data = doc.data();
53       return data.avaliable_plants_list;
54     } else {
55       console.warn("Belge bulunamadı.");
56       return [];
57     }
58   } catch (error) {
59     console.error('Firestore Options Çekme Hatası:', error);
60     return [];
61   }
62 };

```

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.

```

79 useEffect(() => {
80   fetchDevices();
81 }, []);
82
83 useEffect(() => {
84   const init = async () => {
85     const id = await getPersistentAppId();
86     setPersistentId(id);
87     // const oldToken = await getOldToken();
88
89     const plants = await getOptionsFromFirestore();
90     console.log("Firestore'dan gelen plants:", plants);
91     setOptions(plants); // State asenkron olarak güncellenir
92
93     try {
94       const newToken = await messaging().getToken();
95       console.log('FCM Token:', newToken);
96
97       const batch = firestore().batch();
98
99       for (const device of devices) {
100         const deviceRef = firestore().collection('devices').doc(device.id);
101         await deviceRef.set({}, { merge: true });
102         const userRef = deviceRef.collection('users').doc('users_list');
103         batch.set(userRef, { [id]: newToken }, { merge: true });
104       }

```

```

105
106     await batch.commit();
107     // await AsyncStorage.setItem('old_token', newToken);
108     console.log('FCM token Firestore\'a başarıyla gönderildi.');
```

```

109
110   } catch (error) {
111     if (error instanceof Error) {
112       console.log('FCM token alırken hata oluştu:', error.message);
113       console.log(error.stack);
114     } else {
115       console.log('FCM token alırken hata oluştu (non-Error türü):', JSON.stringify(error));
116     }
117   }
118
119   setIsReady(true); // Hem options hem diğer işler hazır
120 };
121
122   init();
123 }, [devices]);

```

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 oluşturulmuş olur. Cihaz ekleme sayfası için oluşturulmuş 'Drawer.Screen' içerisinde ise 'AddDeviceScreen' fonksiyonu çağırılarak cihaz ekleme sayfası oluşturulur. Burada her iki fonksiyonun 'refreshDevices' argümanı olarak 'fetchDevices' fonksiyonu geçirilir.

```

129   return (
130     <NavigationContainer>
131       <Drawer.Navigator initialRouteName={initialRoute}>
132         {devices.map(device => {
133           console.log('Drawer için gönderilen options (${device.name}):', options);
134           return (
135             <Drawer.Screen
136               key={device.id}
137               name={device.name}
138             >
139               {props => (
140                 <DeviceScreen
141                   {...props}
142                   deviceId={device.id}
143                   options={options}
144                   persistentId={persistentId}
145                   refreshDevices={fetchDevices} // Burada fonksiyonu prop olarak veriyoruz
146                 />
147               )}
148             </Drawer.Screen>
149           );
150         })}
151       <Drawer.Screen
152         name="Cihaz Ekle"
153         children={() => <AddDeviceScreen refreshDevices={fetchDevices} />}
154       />
155     </Drawer.Navigator>
156   </NavigationContainer>
157 );
158

```

Şekil 3.17. 'App' Fonksiyonunun 'return' Bölümü

Bu sayede kod tamamlanmış olur.