

**ANKARA UNIVERSITY FACULTY OF
ENGINEERING
DEPARTMENT OF COMPUTER ENGINEERING**



BLM 4537 PROJE RAPORU

**Repair Service Flutter App
Onarım Servisi Flutter Uygulaması**

OMARI SAMI IBRAHİM ABUZİD

20290020

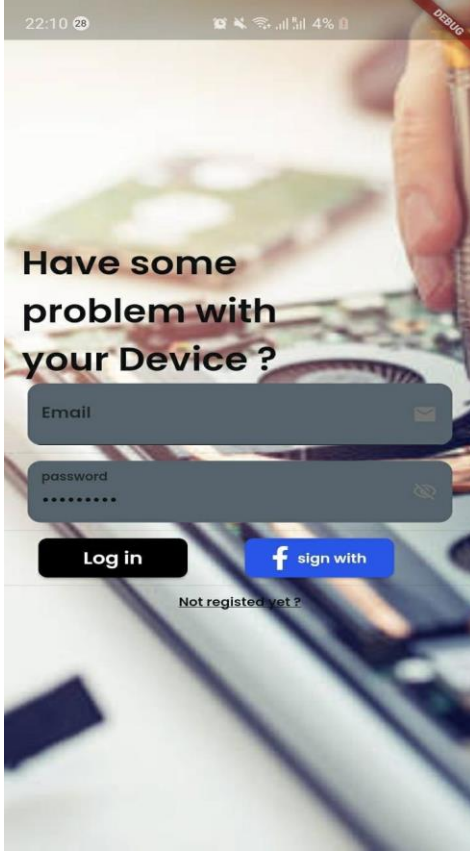
Dr.Enver bağcı

Özet

Uygulamanın önemli özelliklerinden biri, müşterinin cihazını bir tamirciye atamaktır. Bu, müşterinin cihazının tamir işlemi gerektiren özel bir uzman tarafından tamir edilmesini sağlar.Uygulama ayrıca, mobil telefonlar, PC'ler, kulaklıklar ve saatler gibi çeşitli cihazlar için sipariş verme seçeneği sunar. Bu, müşterilerin ne tür bir cihazları olduğu ne olursa olsun, uygulamayı tüm tamir ihtiyaçları için kullanabilecekleri anlamına gelir.Kullanıcı dostu arayüzü, müşterilerin sipariş vermelerini ve tamir durumlarını izlemelerini kolaylaştırmaktadır. Ayrıca, müşteriler doğrudan tamirci ile iletişim kurabilir ve tamir sürecine dair güncellemeler alabilir,Müşterilere cihazlarını tamir etmelerini ve işletmelerini kolaylaştırmak için uygulama, tamircilerin de siparişlerini yönetmelerine, programlarını izlemelerine ve müşterileri ile iletişim kurmalarına olanak tanır.

Sayfalar

1. Login Page :



Uygulamadaki giriş sayfası, kullanıcıların e-posta ve şifrelerini girerek uygulamaya erişebilmelerini sağlar. Sayfa, kullanıcının e-posta ve şifre bilgilerini girmesini sağlayan bir form içerir. E-posta TextFormField, e-posta denetleyicisini alır ve şifre TextFormField, şifre denetleyicisini alır. Sayfada, kullanıcının şifreyi yazarken gizleyebildiği veya gösterebildiği bir açma/kapama düğmesi de bulunmaktadır. Sayfa, formKey adlı global anahtarı kullanarak formun geçerli olup olmadığını kontrol eder ve alanların boş olup olmadığını kontrol eder. Kullanıcı formu gönderdiğinde, uygulama girdiyi doğrular ve girdi geçerli ise bir sonraki adıma devam eder, aksi takdirde girdinin geçersiz olduğunu belirten bir mesaj görüntüler. Kullanıcı giriş yaptıktan sonra, uygulama anasayfaya yönlendirir.

Firestore kullanarak kullanıcıların giriş yapmasını, çıkış yapmasını, şifrelerini sıfırlamasını, e-posta doğrulamasını ve hatta hesaplarını silebilmelerini sağlar. Kod parçası, kullanıcının giriş yaptığını veya hesap oluşturduğunu kontrol etmek için FirebaseAuthException'ı kullanır. Eğer bir hata oluşursa, kod parçası ScaffoldMessenger aracılığıyla kullanıcıya bir hata mesajı gösterir. Ayrıca, kod parçası, kullanıcının JWT Token'ını dinlemek için bir Akış oluşturur ve kullanıcının e-posta doğrulama durumunu kontrol etmek için kullanıcıyı yeniden yükler. Ayrıca, kod parçası, telefon doğrulaması için gerekli olan kodları ve web modunda kullanılan telefon oturum açma işlemlerini de işler.

service ▾

Authentication

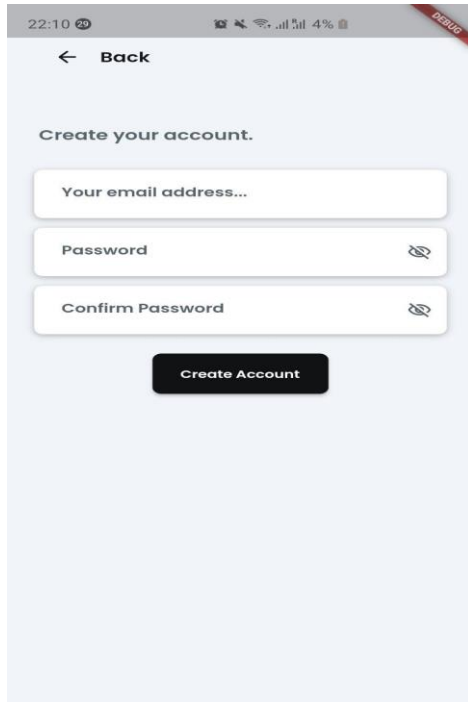
[Users](#)[Sign-in method](#)[Templates](#)[Usage](#)[Settings](#)

Add user ↻ ⋮

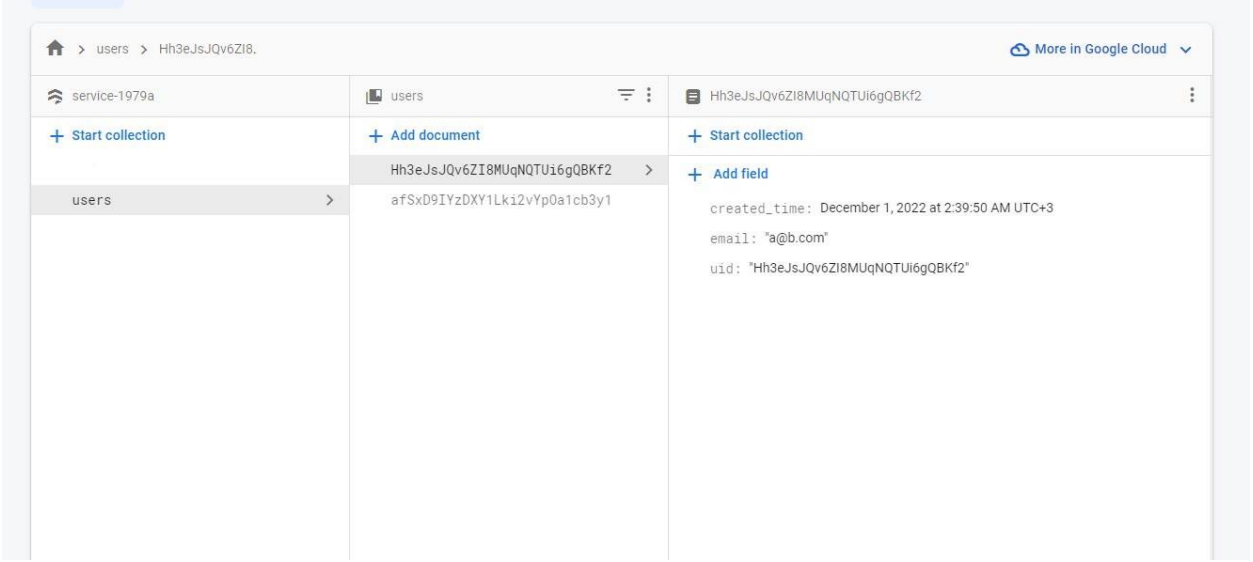
Identifier	Providers	Created ▾	Signed in	User UID
a@b.com	📧	Dec 1, 2022	Jan 12, 2023	Hh3eJsJQv6ZI8MUqNQTI6gQBK...
qassc: 📧 a@b.com	📧	Nov 30, 2022	Nov 30, 2022	afSxD9iYzDXy1Lki2vYp0a1cb3y1

Rows per page: 50 ▾ 1 – 2 of 2 < >

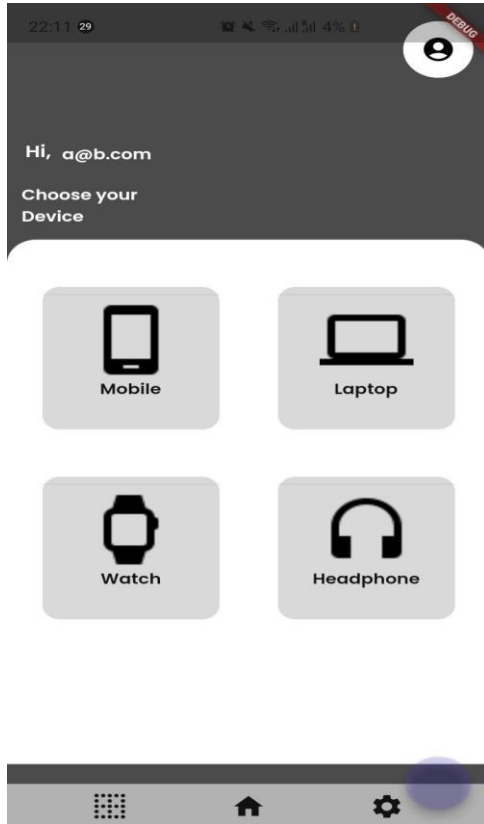
2.register page:



"Register" sayfası ve kullanıcı kaydını işlemek için Firebase Kimlik Doğrulaması'nı kullanır. Sayfa, kullanıcıların e-posta adreslerini ve doğru yazdıklarından emin olmak için onaylamaları gereken bir parola sağlayarak bir hesap oluşturmalarına olanak tanır. Bu sayfada, e-posta ve parola alanlarının girişi için kullanılan birkaç `TextEditingController` nesnesini başlatan durum bilgisi olan bir pencere ögesi vardır. Ayrıca, parola alanlarının görünürlüğünü işlemek için birkaç boolean'a sahiptir. Ek olarak, sayfanın düzenini tutan `Scaffold widget`'ini kontrol etmek için kullanılan global bir `scaffoldKey` anahtarına sahiptir. Kullanıcı e-postasını ve parolasını girip onayladığında, sağlanan e-posta ve parolayla yeni bir kullanıcı hesabı oluşturmak için Firebase Authentication API'yi kullanır. API, parolayı sağlama ve güvenli bir şekilde saklama ve sağlanan e-posta adresine bir doğrulama e-postası gönderme sürecini yönetecektir. API, geçersiz e-posta veya şifre gibi oluşabilecek kayıt hatalarını da ele alacaktır.



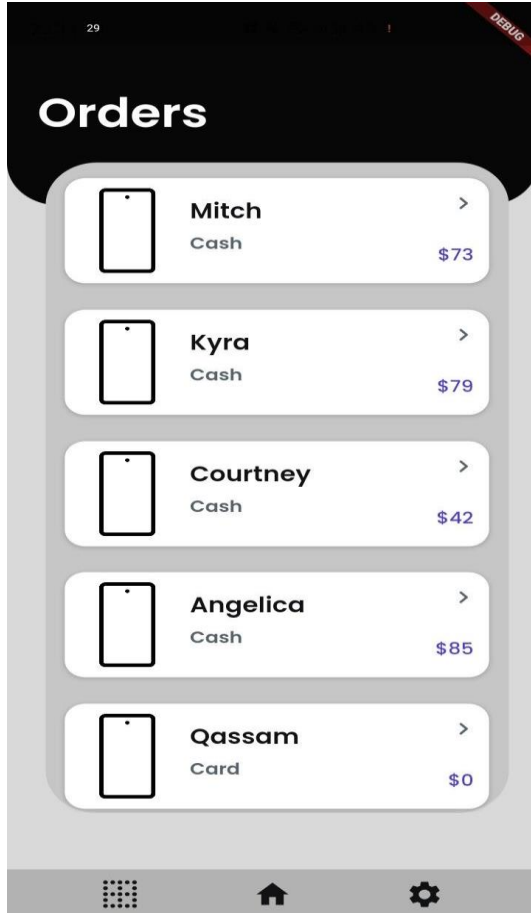
Mainpage :



müşterilerin gerektiği hizmeti seçebilecekleri sayfadır. Bu sayfa, kullanıcının erişebileceği farklı seçeneklere erişebilecekleri giriş noktası

olarak hizmet verir. Sayfa, kullanıcı dostu ve kolayca gezinebilir olacak şekilde tasarlanmıştır, böylece müşteriler hızlı ve kolay bir şekilde mobil, PC, kulaklık veya saatler için gerekli hizmeti seçebilirler. Bu sayfa, müşterilerin uygulama tarafından sunulan tamir hizmetlerine erişebilmeleri için esas olan bir parçasıdır.

Uygulamanın ana sayfasından müşteri istediği hizmeti seçtikten sonra, bilgi Firebase üzerinde bir veritabanına kaydedilir. Bu, iş için atanmış tamir teknisyeni, hizmet maliyeti, tamir edilen cihaz türü ve müşterinin seçtiği ödeme yöntemi dahil olmak üzere bilgileri içerir. Bu veri, müşterinin siparişini izlemek ve tamir işlemlerinin düzgün şekilde gerçekleşmesini sağlamak için kullanılır. Bu veritabanı, uygulamanın tüm siparişleri izleyebilmesini ve her müşterinin zamanında ve etkili bir şekilde hizmet.



Firebase Firestore kullanarak siparişleri kaydetmek için kullanılan bir sınıfı tanımlar. Sınıf, müşteri tarafından seçilen hizmetin adını, fiyatını, ödeme yöntemini ve cihaz türünü içerecek şekilde verileri kaydetmek için tasarlanmıştır. Sınıf, verileri kaydetmek için Firebase Firestore koleksiyonuna başvurur ve verileri almak için kullanabileceğiniz statik metotlar sağlar. Ayrıca, verileri kaydetmek için kullanabileceğiniz bir metod da içerir. Bu kod parçası, siparişleri işlemek için kullanılan uygulamanın arka ucunu tanımlar.

```
static void initializeBuilder(OrdersRecordBuilder builder) => builder
    ..name = ''
    ..salePrice = 0.0
    ..payment = ''
    ..type = '';

static CollectionReference get collection =>
    FirebaseFirestore.instance.collection('orders');

static Stream<OrdersRecord> getDocument(DocumentReference ref) => ref
    .snapshots()
    .map((s) => serializers.deserializeWith(serializer, serializedData(s)));

static Future<OrdersRecord> getDocumentOnce(DocumentReference ref) => ref
    .get()
    .then((s) => serializers.deserializeWith(serializer, serializedData(s)));

OrdersRecord._();
factory OrdersRecord([void Function(OrdersRecordBuilder) updates]) =
    _$OrdersRecord;

static OrdersRecord getDocumentFromData(
    Map<String, dynamic> data, DocumentReference reference) =>
    serializers.deserializeWith(serializer,
        {...mapFromFirestore(data), kDocumentReferenceField: reference});

Map<String, dynamic> createOrdersRecordData({
    String? name,
    double? salePrice,
    String? payment,
    String? type,
}) {
    final firestoreData = serializers.toFirestore(
        OrdersRecord.serializer,
        OrdersRecord(
            (o) => o
                ..name = name
                ..salePrice = salePrice
                ..payment = payment
                ..type = type,
        ),
    );
    return firestoreData;
```


<div>🏠 > orders > 1FLguNEEdMa9. </div> <div>More in Google Cloud</div>		
<div>🏠 service-1979a</div>	<div> orders </div>	<div> 1FLguNEEdMa9GWiyF9Hm </div>
<div>+ Start collection</div>	<div>+ Add document</div>	<div>+ Start collection</div>
<div>orders ></div>	<div>1FLguNEEdMa9GWiyF9Hm ></div>	<div>+ Add field</div>
<div>users</div>	<div>IZdcCsvnVdQSAv9SYFr6 Ltfoj5i1DFP1f0dAGDFM OMebL3IHZ5KpQ2E9f54g dThDb41WLXyiatDF7e0w</div>	<div>name: "Juany" payment: "Cash" sale_price: 132 type: "Mobile"</div>

```
Future maybeCreateUser(User user) async {
  final userRecord = UsersRecord.collection.doc(user.uid);
  final userExists = await userRecord.get().then((u) => u.exists);
  if (userExists) {
    currentUserDocument = await UsersRecord.getDocumentOnce(userRecord);
    return;
  }

  final userData = createUsersRecordData(
    email: user.email,
    displayName: user.displayName,
    photoUrl: user.photoURL,
    uid: user.uid,
    phoneNumber: user.phoneNumber,
    createTime: getCurrentTimestamp,
  );

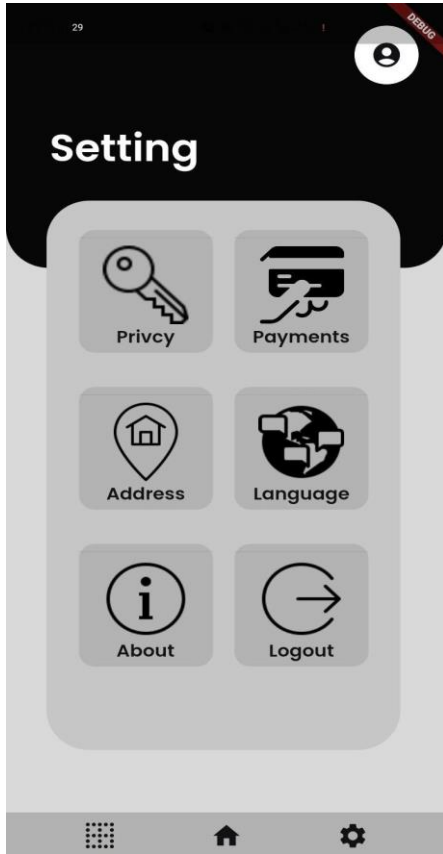
  await userRecord.set(userData);
  currentUserDocument =
    serializers.deserializeWith(UsersRecord.serializer, userData);
}
```

```
Future<FFFirestorePage<T>> queryCollectionPage<T>({
  Query collection,
  Serializer<T> serializer, {
  Query Function(Query)? queryBuilder,
  DocumentSnapshot? nextPageMarker,
  required int pageSize,
  required bool isStream,
}) async {
  final builder = queryBuilder ?? (q) => q;
  var query = builder(collection).limit(pageSize);
  if (nextPageMarker != null) {
    query = query.startAfterDocument(nextPageMarker);
  }
  Stream<QuerySnapshot>? docSnapshotStream;
  QuerySnapshot docSnapshot;
  if (isStream) {
    docSnapshotStream = query.snapshots();
    docSnapshot = await docSnapshotStream.first;
  } else {
    docSnapshot = await query.get();
  }
  final getDocs = (QuerySnapshot s) => s.docs
    .map(
      (d) => safeGet(
        () => serializers.deserializeWith(serializer, serializedData(d)),
        (e) => print('Error serializing doc ${d.reference.path}:\n$e'),
      ),
    )
    .where((d) => d != null)
    .map((d) => d!)
    .toList();
  final data = getDocs(docSnapshot);
  final dataStream = docSnapshotStream?.map(getDocs);
  final nextPageToken = docSnapshot.docs.isEmpty ? null : docSnapshot.docs.last;
  return FFFFirestorePage(data, dataStream, nextPageToken);
}
```

Firestore veritabanından kullanıcı ve sipariş kayıtlarını sorgulamak için fonksiyonlar içermektedir. Kullanıcı kayıtlarını sorgulamak için 'queryUsersRecord' ve 'queryUsersRecordOnce' gibi fonksiyonlar

kullanılabilir. Aynı şekilde sipariş kayıtları için de 'queryOrdersRecord' ve 'queryOrdersRecordOnce' gibi fonksiyonlar mevcuttur. Bu fonksiyonlar, sorguların sonucunu bir Stream veya Future olarak döndürür. Ayrıca, kullanıcıların sorgulamalarını filtrelemelerini sağlamak için 'queryBuilder' parametresi de mevcuttur.

3.Setting Page

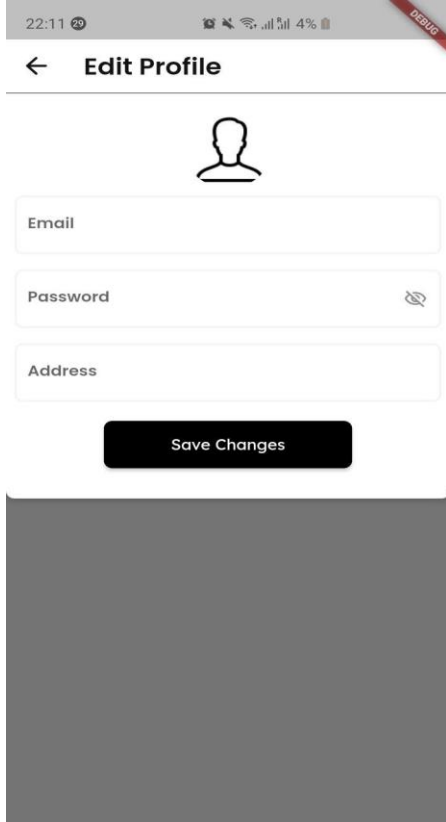


Bir ayar sayfası için bir açıklama yazın

parola, adres, ödeme yöntemi, çıkış, hakkında düğmeleri ile ve her düğmeyi açıklayın

Ayar sayfası, kullanıcının hesabını yönetebileceği ve kişisel bilgilerini güncelleyebileceği bir yerdir. Sayfa içinde şunlar bulunur:

Parola düğmesi: Kullanıcının hesabının parolasunu değiştirmesine olanak tanır.



22:11 29

← Edit Profile

Profile Icon

Email

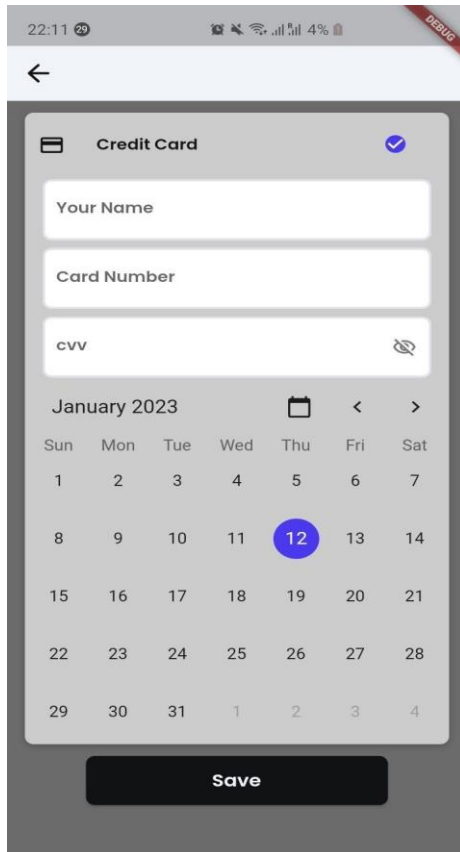
Password

Address

Save Changes

Adres düğmesi: Kullanıcının teslimat adresini güncelleyebileceği bir sayfayı açar.

Ödeme yöntemi düğmesi: Kullanıcının ödeme bilgilerini güncelleyebileceği bir sayfayı açar.



Çıkış düğmesi: Kullanıcının hesabından çıkmasını sağlar.

Hakkında düğmesi: Uygulama hakkında bilgi verir.

FireBase :

```
Future initFirebase() async {  
  if (kIsWeb) {  
    await Firebase.initializeApp(  
      options: FirebaseOptions(  
        apiKey: "AIzaSyC83rD6MLIO3n88WcwVUkHoduCstxNtehE",  
        authDomain: "service-1979a.firebaseio.com",  
        projectId: "service-1979a",  
        storageBucket: "service-1979a.appspot.com",  
        messagingSenderId: "80847212688",  
        appId: "1:80847212688:web:f384e02e8d72910caa8815"));  
  } else {  
    await Firebase.initializeApp();  
  }  
}
```