

T.C.
ONDOKUZ MAYIS ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ



2024-2025 GÜZ DÖNEMİ
MOBİL PROGRAMLAMA DERSİ FİNAL ÖDEVİ RAPORU
SANAYİ TEDARİK UYGULAMASI

ALİ EREN ÖZGEN - 21060609
EYÜPHAN BİNİCİ - 22060653
MUHAMMED MURAT KAYA - 22060390

GİRİŞ

Projenin Amacı

Bu projenin amacı, sanayi sektörüne yönelik olarak tedarik ve tedarikçi ihtiyaçlarının dijital bir platform üzerinden etkin bir şekilde yönetilmesini sağlamaktır. Geliştirilen mobil uygulama, kullanıcıların hem tedarik ihtiyaçlarını paylaşımlarına hem de mevcut ihtiyaçlara başvuruda bulunmalarına olanak tanımaktadır. Bu sayede, firmalar arasında daha hızlı ve verimli bir tedarik zinciri iletişimi kurulması hedeflenmektedir.

Uygulama, kullanıcıların sektöre özgü tedarik bilgilerini kolayca paylaşabilecekleri ve ihtiyaçlarına uygun tedarikçileri hızlıca bulabilecekleri bir ortam sunmayı amaçlamaktadır. Bildirim sistemi ve kullanıcı dostu bir arayüz ile, firmaların zaman tasarrufu sağlaması ve süreçlerini dijitalleştirmesi hedeflenmiştir. Proje aynı zamanda Firebase altyapısı kullanılarak güvenilir bir veri tabanı ve bildirim sistemi üzerine inşa edilmiştir.

KODLAR

application_page.dart

1.Imports

```
1  import 'package:flutter/material.dart';
2  import 'package:cloud_firestore/cloud_firestore.dart';
3  import 'package:firebase_auth/firebase_auth.dart';
4  import 'public_profile_page.dart';
5
6  import '../widgets/tedarik_card.dart';
7
```

- flutter/material.dart: Flutter'da widget'lar ve temel yapılar için kullanılır.
- cloud_firestore/cloud_firestore.dart: Firebase Firestore'daki verileri yönetmek için kullanılır.
- firebase_auth/firebase_auth.dart: Firebase kimlik doğrulama işlemleri için kullanılır.
- public_profile_page.dart: Kullanıcıların genel profil sayfasına yönlendirme için gereken bir sayfa.
- tedarik_card.dart: Tedarik bilgilerini göstermek için kullanılan özel bir widget (detaylar buraya özelleştirilmiş).

2. Giriş Kontrolü

```
16      if (currentUser == null) {
17          return Scaffold(
18              body: Center(child: Text("Lütfen giriş yapınız.")),
19          );
20      }
21
```

- Kullanıcı kimlik doğrulamasını kontrol ediyor.
- Eğer kullanıcı giriş yapmamışsa basit bir mesaj içeren bir ekran gösteriliyor.

3. Firestore'dan Başvuruların Çekilmesi

```
25     stream: FirebaseFirestore.instance
26         .collectionGroup('applications')
27         .where('userId', isEqualTo: currentUser.uid)
28         .snapshots(),
```

- **collectionGroup:** Tüm koleksiyonlar içinde **applications** alt koleksiyonlarını tarar.
- **where:** Şart belirterek yalnızca **userId** alanı oturum açmış kullanıcının **UID**'sine eşit olan belgeleri çeker.
- **snapshots():** Akışlı bir yapı sağlar ve veriler güncellendikçe otomatik yenileme sunar.

4. StreamBuilder ile Veri İşleme

```
29     builder: (context, appSnap) {
30         if (appSnap.connectionState == ConnectionState.waiting) {
31             return const Center(child: CircularProgressIndicator());
32         }
33         if (appSnap.hasError) {
34             return Center(child: Text('Bir hata oluştu: ${appSnap.error}'));
35         }
36         if (!appSnap.hasData || appSnap.data!.docs.isEmpty) {
37             return const Center(child: Text("Henüz bir başvurunuz yok."));
38         }
39     }
```

- **Connection State:** Veriler yükleniyorsa bir yükleme göstergesi (**CircularProgressIndicator**) gösterilir.
- **Error Handling:** Veri çekme sırasında hata olursa mesaj gösterilir.
- **Empty State:** Eğer başvurular yoksa kullanıcıya mesaj iletilir.

5. Listeleme ve Bağlantılı Verilerin Çekilmesi

```
42     return ListView.builder(  
43         itemCount: applicationDocs.length,  
44         itemBuilder: (context, index) {  
45             final applicationDoc = applicationDocs[index];  
46             final supplyRef = applicationDoc.reference.parent.parent;  
47         }
```

- **applicationDocs:** Kullanıcının başvurularını temsil eden belgeler.
- **supplyRef:** İlgili tedarik (supply) belgesine referans verir. Bu, başvurunun ait olduğu tedarik bilgisini almak için kullanılır.

6. Supply (Tedarik) Belgesinin Yüklenmesi

```
54     return FutureBuilder<DocumentSnapshot>(  
55         future: supplyRef.get(),  
56         builder: (context, supplySnap) {  
57             if (supplySnap.connectionState == ConnectionState.waiting) {  
58                 return const Center(child: CircularProgressIndicator());  
59             }  
60             if (supplySnap.hasError) {  
61                 return Text(  
62                     "Tedarik yüklenirken hata: ${supplySnap.error}");  
63             }  
64             if (!supplySnap.hasData || !supplySnap.data!.exists) {  
65                 return const ListTile(  
66                     title: Text("Tedarik bulunamadı veya silinmiş."),  
67                 );  
68             }  
69         }
```

- **FutureBuilder:** Asenkron olarak tedarik belgesini yükler.
- **Hata ve Durum Kontrolleri:** Tedarik silinmiş veya hata varsa kullanıcı bilgilendirilir.

7. Kullanıcı Bilgilerinin Çekilmesi

```
81 return FutureBuilder<DocumentSnapshot>(
82   future: FirebaseFirestore.instance
83     .collection('users')
84     .doc(createdBy)
85     .get(),
86   builder: (context, userSnap) {
87     if (userSnap.connectionState == ConnectionState.waiting) {
88       return const Center(child: CircularProgressIndicator());
89     }
90     if (userSnap.hasError) {
91       return Text(
92         "Kullanıcı bilgileri yüklenirken hata: ${userSnap.error}");
93     }
94     if (!userSnap.hasData || !userSnap.data!.exists) {
95       return const ListTile(
96         title: Text("Kullanıcı bulunamadı."),
97       );
98     }
99   }
```

- Tedarik belgesindeki createdBy alanını kullanarak ilgili kullanıcı bilgileri çekilir.
- Kullanıcı adı gibi bilgileri göstermek için kullanılır.

8. Tedarik Bilgilerinin Gösterimi

```
105 return Padding(
106   padding: const EdgeInsets.symmetric(
107     vertical: 8.0, horizontal: 16.0),
108   child: SizedBox(
109     height: 350,
110     child: TedarikCard(
111       docId: docId,
112       userId: createdBy,
113       username:
114         username, // Kullanıcı adını buraya ekleyin
115       title: title,
116       price: price,
117       description: description,
118       sector: sector,
119       imageUrl: imageUrl,
120       createdBy: createdBy,
121       onUsernameTap: (tappedUserId) {
122         // Kullanıcı profil sayfasına yönlendirme
123         Navigator.push(
124           context,
125           MaterialPageRoute(
126             builder: (context) =>
127               PublicProfilePage(userId: tappedUserId),
128           ),
129         );
130       },
131     ),
132   );
```

- TedarikCard: Tedarik bilgilerini gösteren özel bir widget.
- Kullanıcı adına tıklandığında, PublicProfilePage sayfasına yönlendirilir.

home_page.dart

1. Firebase Auth Kullanımı

```
20 final FirebaseAuth _auth = FirebaseAuth.instance;
21 final currentUser = FirebaseAuth.instance.currentUser;
```

- **Amaç:** Firebase ile oturum açmış kullanıcıya erişmek için FirebaseAuth kullanılıyor.

Açıklama:

- FirebaseAuth.instance ile Firebase kimlik doğrulama servisine bağlanılıyor.
- currentUser, şu anda oturum açmış kullanıcıyı döner.

2. Veri Akışını (Stream) Hazırlama

```
33 Stream<List<Map<String, String>>> fetchTedarikItems() {
34   return FirebaseFirestore.instance
35     .collection('supplies')
36     .snapshots()
37     .asycnMap((snapshot) async {
38       List<Map<String, String>> tedarikList = [];
39       for (var doc in snapshot.docs) {
40         String createdBy = doc['created_by'];
41         String name = await _fetchUsernameFromUid(createdBy);
42
43         tedarikList.add({
44           'docId': doc.id,
45           'title': doc['title'],
46           'description': doc['description'],
47           'price' : doc['price'],
48           'sector': doc['sector'],
49           'name': name,
50           'file_url': doc['file_url'] ?? '',
51           'user_id': createdBy,
52           'created_by': doc['created_by'] ?? '',
53         });
54       }
55       return tedarikList;
56     });
57 }
58
```

Amaç: Firestore'dan gelen tedarik verilerini bir listeye dönüştürmek.

Detaylar:

1. FirestoreBağlantısı:

FirebaseFirestore.instance.collection('supplies') ile supplies koleksiyonuna bağlanılır.

2. Gerçek Zamanlı Güncellemeler:

snapshots() ile anlık veri değişiklikleri dinlenir.

3. Veri Dönüşümü:

asycnMap, gelen belgeleri işleyerek bir listeye dönüştürür. Kullanıcının ismi _fetchUsernameFromUid fonksiyonu ile çekilir.

3. Kullanıcı İsmi Alma

```
59 Future<String> _fetchUsernameFromUid(String created_by) async {
60   try {
61     DocumentSnapshot userDoc = await FirebaseFirestore.instance
62       .collection('users')
63       .doc(created_by)
64       .get();
65
66     if (userDoc.exists) {
67       return userDoc['name'] ?? 'Bilinmiyor';
68     } else {
69       return 'Bilinmiyor';
70     }
71   } catch (e) {
72     print("Error fetching name: $e");
73     return 'Bilinmiyor';
74   }
75 }
```

- **Amaç:** Belirtilen uid'ye sahip kullanıcı adını Firestore'daki users koleksiyonundan almak.
- **Açıklama:**
- `FirebaseFirestore.instance.collection('users').doc(created_by).get()` ile `created_by` değerine göre bir belge çekiliyor.
- Eğer belge mevcutsa name alanı döndürülüyor. Mevcut değilse "Bilinmiyor" döndürülüyor.

4. Veri Filtreleme

```
122 final filteredItems = snapshot.data!
123     .where((item) =>
124       item['title']!.toLowerCase().contains(searchQuery.toLowerCase()) ||
125       item['description']!.toLowerCase().contains(searchQuery.toLowerCase()) ||
126       item['sector']!.toLowerCase().contains(searchQuery.toLowerCase()) ||
127       item['name']!.toLowerCase().contains(searchQuery.toLowerCase())
128     )
129     .toList();
```

- **Amaç:** Kullanıcının arama sorgusuna (`searchQuery`) uygun tedarik öğelerini filtrelemek.
- **Nasıl Çalışır:**
- `title`, `description`, `sector` ve `name` alanlarında arama yapılır.
- Tüm değerler küçük harfe dönüştürülerek arama yapılır (büyük/küçük harf duyarlılığı önlenir).

5. Yeni Tedarik Ekleme

```
301 Future<void> _addSupply(String title, String price,String description, String sector, File? image) async {
302   try {
303     final user = FirebaseAuth.instance.currentUser;
304
305     String? imageUrl;
306     if (image != null) {
307       imageUrl = await uploadImageToImgbg(image);
308       if (imageUrl == null) {
309         print('Resim yükleme başarısız. ');
310         return;
311       }
312
313       await FirebaseFirestore.instance.collection('supplies').add({
314         'title': title,
315         'price': price,
316         'description': description,
317         'sector': sector,
318         'created_by': user?.uid,
319         'created_at': Timestamp.now(),
320         'file_url': imageUrl,
321       });
322
323       print('Yeni tedarik başarıyla eklendi. ');
324     }
325   } catch (e) {
326     print("Tedarik eklerken hata oluştu: $e");
327   }
328 }
329
```

- **Amaç:** Kullanıcının yeni bir tedarik eklemesini sağlamak.
- **Nasıl Çalışır:**
 4. Resim varsa, uploadImageToImgbg ile yüklenir ve URL alınır.
 5. Gerekli bilgiler Firestore'daki supplies koleksiyonuna kaydedilir.
 6. Kullanıcı kimliği (created_by) ve zaman damgası (created_at) eklenir.

login.page.dart

1. Firebase Auth Kullanımı

```
21 final FirebaseAuth _auth = FirebaseAuth.instance;
```

- Firebase Authentication kullanılarak oturum açma işlemleri gerçekleştirilir.
- FirebaseAuth.instance ile mevcut kimlik doğrulama servisine erişilir.

2. FCM Token Alma

```
26 Future<String?> getFCMToken() async {
27   return await FirebaseMessaging.instance.getToken();
28 }
```

- **Amaç:** Firebase Cloud Messaging (FCM) ile kullanıcının cihazına özel bir bildirim token'ı almak.
- **Kullanım:** Bu token, kullanıcının cihazına push bildirim göndermek için gereklidir.

3. Kullanıcı Girişi

```

30 Future<void> loginUser() async {
31   try {
32     final UserCredential userCredential = await _auth.signInWithEmailAndPassword(
33       email: _emailController.text.trim(),
34       password: _passwordController.text.trim(),
35     );
36
37     // Get FCM token
38     String? fcmToken = await getFCMToken();
39     if (fcmToken != null) {
40       // Save FCM token to Firestore or backend
41       await Firestore.instance
42         .collection('users')
43         .doc(userCredential.user!.uid)
44         .update({'fcmToken': fcmToken});
45     }
46     // Navigate to MainPage after successful login
47     Navigator.pushReplacement(
48       context,
49       MaterialPageRoute(builder: (context) => MainPage() ),
50     );
51   } on FirebaseAuthException catch (e) {
52     setState() {
53       errorMessage = e.message;
54     };
55   }
56 }

```

- **Firestore ile Giriş:**
- Kullanıcı, e-posta ve şifre ile giriş yapar.
- `signInWithEmailAndPassword` metodu kullanılır.
- **FCM Token Kaydetme:**
- `fcmToken` alınıp Firestore'daki ilgili kullanıcı belgesine kaydedilir.
- Bu işlem, bildirim göndermek için gereklidir.
- **Hata Yönetimi:** `FirebaseAuthException` yakalanarak hata mesajı ekranda gösterilir.

profile_page.dart

1. Firebase'den Kullanıcı Verisi Çekme

```
45 Future<void> _fetchUserData() async {
46   final currentUser = _auth.currentUser;
47   if (currentUser == null) return;
48
49   try {
50     final userDoc = await FirebaseFirestore.instance
51       .collection('users')
52       .doc(currentUser.uid)
53       .get();
54
55     if (userDoc.exists) {
56       final data = userDoc.data();
57       setState(() {
58         _userName = data?['name'] ?? '';
59         _profilePhotoUrl = data?['profile_photo'] ?? '';
60         _userEmail = data?['email'] ?? currentUser.email ?? '';
61       });
62       _nameController.text = _userName;
63     }
64   } catch (e) {
65     print('Kullanıcı verisi alınırken hata oluştu: $e');
66   }
67 }
```

- Bu fonksiyon, Firebase Firestore'dan kullanıcı verilerini çekiyor (name, profile_photo, email gibi).
- Veriler başarılı şekilde alındığında setState ile UI güncelleniyor.

2. Kullanıcıya Ait Tedarikleri Çekme

```
70 Future<List<Map<String, String>>> _fetchUserTedarikItems() async {
71   final currentUser = _auth.currentUser;
72   if (currentUser == null) return [];
73   try {
74     QuerySnapshot snapshot = await FirebaseFirestore.instance
75       .collection('supplies')
76       .where('created_by', isEqualTo: currentUser.uid)
77       .get();
78
79     List<Map<String, String>> tedarikList = [];
80     for (var doc in snapshot.docs) {
81       // Tedarik dokümanındaki alanları okuyoruz
82       tedarikList.add({
83         'docId': doc.id,
84         'title': doc['title'],
85         'description': doc['description'],
86         'price': doc['price'],
87         'sector': doc['sector'],
88         // Kullanıcı adı olarak, anlık _userName'i kullanmak istiyorsan:
89         'username': _userName.isNotEmpty ? _userName : 'Bilinmiyor',
90         // Tedarik resminin URL'si
91         'image_url': doc['file_url'] ?? '',
92         'user_id': currentUser.uid,
93         'created_by': doc['created_by'] ?? '',
94       });
95     }
96     return tedarikList;
97 }
```

- Kullanıcının Firestore'daki tedariklerini çekiyor.
- Kullanıcıya ait supplies koleksiyonundaki veriler döngüye alınarak listeleniyor.

3. Fotoğraf Seçme ve Yükleme

```
104 Future<void> _pickImage() async {
105   try {
106     final pickedFile = await _picker.pickImage(source: ImageSource.gallery);
107     if (pickedFile != null) {
108       setState() {
109         _imageFile = File(pickedFile.path);
110       });
111     }
```

Bu fonksiyon, kullanıcının galeriden bir fotoğraf seçmesini sağlıyor.

4. Profil Fotoğrafını Yükleme

```
117 Future<String?> _uploadImageToImgbb(File image) async {
118   try {
119     final url = Uri.parse(
120       'https://api.imgbb.com/1/upload?key=b8e8e63a0125d7bcb819f6833cc22e5b',
121     );
122
123     var request = http.MultipartRequest('POST', url)
124       ..files.add(await http.MultipartFile.fromPath('image', image.path));
125
126     var response = await request.send();
127     if (response.statusCode == 200) {
128       final responseData = await response.stream.bytesToString();
129       final jsonResponse = json.decode(responseData);
130       final imageUrl = jsonResponse['data']['url'];
131       return imageUrl;
132     } else {
133       print('Resim yükleme başarısız. Kod: ${response.statusCode}');
134       return null;
135     }
136   }
```

Bu fonksiyon, seçilen profil fotoğrafını İmgBB API'ye yükler ve URL döner.

5. Firestore'a Profil Güncelleme

```
143 Future<void> _updateUserProfile() async {
144   final currentUser = _auth.currentUser;
145   if (currentUser == null) return;
146   try {
147     String? newPhotoUrl = _profilePhotoUrl;
148     // Eğer yeni resim seçildiyse önce İmgBB'ye yükle
149     if (_imageFile != null) {
150       final uploadedUrl = await _uploadImageToImgbb(_imageFile!);
151       if (uploadedUrl != null) {
152         newPhotoUrl = uploadedUrl;
153       }
154     }
155     // Firestore'u güncelle
156     await Firestore.instance.collection('users').doc(currentUser.uid).update({
157       'name': _nameController.text.trim(),
158       'profile_photo': newPhotoUrl,
159     });
160
161     // State'i güncelleyerek ekranda yansıtmasını sağla
162     setState() {
163       _userName = _nameController.text.trim();
164       _profilePhotoUrl = newPhotoUrl ?? '';
165     });
166   }
```

- Kullanıcı adı ve fotoğrafını Firestore'da günceller.
- Yeni fotoğraf varsa, önce İmgBB'ye yükleyip URL'yi alır.

public_profile_page.dart

1. Kullanıcı Bilgilerini Çekme (_fetchUserData)

```
37 Future<void> _fetchUserData() async {
38   try {
39     DocumentSnapshot userDoc = await FirebaseFirestore.instance
40       .collection('users')
41       .doc(widget.userId)
42       .get();
43     if (userDoc.exists) {
44       final data = userDoc.data() as Map<String, dynamic>;
45       setState(() {
46         _userName = data['name'] ?? 'Bilinmiyor';
47         _profilePhotoUrl = data['profile_photo'] ?? '';
48         _userEmail = data['email'] ?? 'Bilinmiyor';
49       });
50     }
51   } catch (e) {
52     print("PublicProfilePage - Kullanıcı verisi alınırken hata: $e");
53   }
54 }
```

- Bu fonksiyon, widget.userId ile belirtilen kullanıcının profil verilerini çekiyor.
- name, profile_photo ve email verileri alınıp UI güncelleniyor.

2. Kullanıcıların Paylaşımlarını (Tedariklerini) Çekme (fetchOtherUserTedarikItems)

```
57 Future<List<Map<String, String>>> _fetchOtherUserTedarikItems(String userId) async {
58   try {
59     QuerySnapshot snapshot = await FirebaseFirestore.instance
60       .collection('supplies')
61       .where('created_by', isEqualTo: userId)
62       .get();
63
64     List<Map<String, String>> tedarikList = [];
65     for (var doc in snapshot.docs) {
66       // Dokumandaki verileri okuyoruz
67       final data = doc.data() as Map<String, dynamic>;
68
69       tedarikList.add({
70         'docId': doc.id,
71         'title': data['title'] ?? 'No Title',
72         'price': data['price'] ?? 'No Price',
73         'description': data['description'] ?? '',
74         'sector': data['sector'] ?? '',
75         'username': _userName.isNotEmpty ? _userName : 'Bilinmiyor',
76         'image_url': data['file_url'] ?? '',
77         'created_by': data['created_by'] ?? '',
78         'user_id': userId,
79       });
80     }
81     return tedarikList;
82   } catch (e) {
83     print("PublicProfilePage - Tedarikler alınırken hata: $e");
84     return [];
85   }
86 }
```

- Bu fonksiyon, userId'ye göre kullanıcının paylaştığı tedarikleri çekiyor.
- Tedariklerin verileri (başlık, fiyat, açıklama vb.) listeleniyor.

3. Profil ve Tedarik Kartlarını Görüntüleme

```
142 Expanded(
143   child: FutureBuilder<List<Map<String, String>>>{
144     future: _otherUserTedarikItems,
145     builder: (context, snapshot) {
146       if (snapshot.connectionState == ConnectionState.waiting) {
147         return Center(child: CircularProgressIndicator());
148       } else if (snapshot.hasError) {
149         return Center(
150           child: Text('Bir hata oluştu: ${snapshot.error}'),
151         );
152       } else if (!snapshot.hasData || snapshot.data!.isEmpty) {
153         return Center(
154           child: Text('Bu kullanıcı henüz paylaşım yok.'),
155         );
156       }
157
158       final tedarikList = snapshot.data!;
159
160       return GridView.builder(
161         padding: EdgeInsets.all(8),
162         gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
163           crossAxisCount: 2, // 2 sütun
164           crossAxisSpacing: 16,
165           mainAxisSpacing: 16,
166           childAspectRatio: 0.65,
167         ),
168         itemCount: tedarikList.length,
169         itemBuilder: (context, index) {
170           final item = tedarikList[index];
171           return TedarikCard(
172             userId: item['user_id']!,
173             createdBy: item['created_by']!, // onUsernameTap vb. için
174             username: item['username']!,
175             title: item['title']!,
176             price: item['price']!,
177             description: item['description']!,
178             sector: item['sector']!,
179             imageUrl: item['image_url']!,
180             docId: item['docId']!,
181
182             onApply: () {
183               _applyForTedarik(item['docId']!);
184               button color: Colors.green;
185             }
186           );
187         }
188       );
189     }
190   );
191 }
192
```

- FutureBuilder kullanılarak, kullanıcıya ait tedarikler bir GridView içinde gösteriliyor.
- Eğer kullanıcıya ait tedarik varsa, her bir tedarik bir TedarikCard widget'ı ile listeleniyor.

4. Başvuru Yapma (Başvuru Butonu)

```
202 Future<void> _applyForTedarik(String docId) async {
203   final FirebaseAuth _auth = FirebaseAuth.instance;
204
205   final currentUser = _auth.currentUser;
206
207   if (currentUser == null) return;
208   try {
209     final applicationsRef = FirebaseFirestore.instance
210       .collection('supplies')
211       .doc(docId)
212       .collection('applications');
213
214     // Basit bir şekilde doc id olarak currentUser.uid kullanabilirsin
215     await applicationsRef.doc(currentUser.uid).set({
216       'userId': currentUser.uid,
217       'createdAt': FieldValue.serverTimestamp(),
218     });
219     print('Başvuru başarılı!');
220   } catch (e) {
221     print('Başvuru sırasında hata: $e');
222   }
223 }
```

- Bu fonksiyon, kullanıcının başvurduğu tedarik için Firestore'daki applications alt koleksiyonuna başvuru bilgilerini ekler.
- Kullanıcı başvurduktan sonra başarılı olduğuna dair bir çıktı alır.

register_page.dart

1. Kullanıcı Kayıt Fonksiyonu (registerUser)

```
27 Future<void> registerUser() async {
28   setState(() {
29     _isLoading = true;
30   });
31
32   try {
33     final UserCredential userCredential =
34       await _auth.createUserWithEmailAndPassword(
35         email: _emailController.text.trim(),
36         password: _passwordController.text.trim(),
37       );
38
39     final User? user = userCredential.user;
40     if (user != null) {
41       await _firestore.collection('users').doc(user.uid).set({
42         'email': user.email,
43         'name': _nameController.text.trim(),
44         'uid': user.uid,
45         'created_at': FieldValue.serverTimestamp(),
46         'profile_picture': '',
47       });
48     }
49     Navigator.pushReplacement(
50       context,
51       MaterialPageRoute(builder: (context) => LoginPage()),
52     );
53   } on FirebaseAuthException catch (e) {
54     setState(() {
55       errorMessage = e.message;
56     });
57   } catch (e) {
58     setState(() {
59       errorMessage = "Bir hata oluştu. Lütfen tekrar deneyin.";
60     });
61   } finally {
62     setState(() {
63       _isLoading = false;
64     });
65   }
66 }
```

- Bu fonksiyon, kullanıcıdan alınan e-posta ve parola ile Firebase Authentication kullanarak yeni bir kullanıcı oluşturuyor.
- Kullanıcı oluşturulduktan sonra, Firestore veritabanına kullanıcıya ait bilgiler (email, name, uid) kaydediliyor.
- Kayıt başarılı olursa, kullanıcı login sayfasına yönlendiriliyor.
- Hata durumunda, hata mesajı ekranda gösteriliyor.

2. Kullanıcı Arayüzü (UI)

```
70 return Scaffold(
71   appBar: AppBar(
72     title: Text("Kopernik Pizza", style: TextStyle(color: Color(0xff0e7f3f))),
73     iconTheme: IconThemeData(color: Color(0xff0e7f3f)),
74   ),
75 ),
76 body: SingleChildScrollView(
77   padding: EdgeInsets.all(16.0),
78   child: Column(
79     children: [
80
81       Center(
82         child:
83           Image.asset(
84             'assets/images/logo.png',
85             height: 210,
86             width: 210,
87           ),
88       ),
89
90       ),
91       SizedBox(height: 40),
92       CustomTextField(
93         controller: _nameController,
94         labelText: 'Kullanıcı Adı',
95         keyboardType: TextInputType.name,
96         onChanged: (value) {},
97       ),
98     ],
99   ),
100 );
101
102 if (errorMessage != null)
103   Padding(
104     padding: const EdgeInsets.symmetric(vertical: 10),
105     child: Text(
106       errorMessage!,
107       style: TextStyle(color: Colors.red),
108     ),
109   ),
110   ),
111   ),
112   ),
113   ),
114   ),
115   ),
116   ),
117   ),
118   ),
119   ),
120   ),
121   ),
122   ),
123   ),
124   ),
125   ),
126   ),
127   ),
128   ),
129   ),
130   ),
131   ),
132   ),
133   ),
134   ),
135   ),
136   ),
137   ),
138   ),
139   ),
140   ),
141   ),
142   ),
143   ),
144   ),
145   ),
146   ),
147   ),
148   ),
149   ),
150   ),
151   ),
152   ),
153   ),
154   ),
155   ),
156   ),
157   ),
158   ),
159   ),
160   ),
161   ),
162   ),
163   ),
164   ),
165   ),
166   ),
167   ),
168   ),
169   ),
170   ),
171   ),
172   ),
173   ),
174   ),
175   ),
176   ),
177   ),
178   ),
179   ),
180   ),
181   ),
182   ),
183   ),
184   ),
185   ),
186   ),
187   ),
188   ),
189   ),
190   ),
191   ),
192   ),
193   ),
194   ),
195   ),
196   ),
197   ),
198   ),
199   ),
200   ),
201   ),
202   ),
203   ),
204   ),
205   ),
206   ),
207   ),
208   ),
209   ),
210   ),
211   ),
212   ),
213   ),
214   ),
215   ),
216   ),
217   ),
218   ),
219   ),
220   ),
221   ),
222   ),
223   ),
224   ),
225   ),
226   ),
227   ),
228   ),
229   ),
230   ),
231   ),
232   ),
233   ),
234   ),
235   ),
236   ),
237   ),
238   ),
239   ),
240   ),
241   ),
242   ),
243   ),
244   ),
245   ),
246   ),
247   ),
248   ),
249   ),
250   ),
251   ),
252   ),
253   ),
254   ),
255   ),
256   ),
257   ),
258   ),
259   ),
260   ),
261   ),
262   ),
263   ),
264   ),
265   ),
266   ),
267   ),
268   ),
269   ),
270   ),
271   ),
272   ),
273   ),
274   ),
275   ),
276   ),
277   ),
278   ),
279   ),
280   ),
281   ),
282   ),
283   ),
284   ),
285   ),
286   ),
287   ),
288   ),
289   ),
290   ),
291   ),
292   ),
293   ),
294   ),
295   ),
296   ),
297   ),
298   ),
299   ),
300   ),
301   ),
302   ),
303   ),
304   ),
305   ),
306   ),
307   ),
308   ),
309   ),
310   ),
311   ),
312   ),
313   ),
314   ),
315   ),
316   ),
317   ),
318   ),
319   ),
320   ),
321   ),
322   ),
323   ),
324   ),
325   ),
326   ),
327   ),
328   ),
329   ),
330   ),
331   ),
332   ),
333   ),
334   ),
335   ),
336   ),
337   ),
338   ),
339   ),
340   ),
341   ),
342   ),
343   ),
344   ),
345   ),
346   ),
347   ),
348   ),
349   ),
350   ),
351   ),
352   ),
353   ),
354   ),
355   ),
356   ),
357   ),
358   ),
359   ),
360   ),
361   ),
362   ),
363   ),
364   ),
365   ),
366   ),
367   ),
368   ),
369   ),
370   ),
371   ),
372   ),
373   ),
374   ),
375   ),
376   ),
377   ),
378   ),
379   ),
380   ),
381   ),
382   ),
383   ),
384   ),
385   ),
386   ),
387   ),
388   ),
389   ),
390   ),
391   ),
392   ),
393   ),
394   ),
395   ),
396   ),
397   ),
398   ),
399   ),
400   ),
401   ),
402   ),
403   ),
404   ),
405   ),
406   ),
407   ),
408   ),
409   ),
410   ),
411   ),
412   ),
413   ),
414   ),
415   ),
416   ),
417   ),
418   ),
419   ),
420   ),
421   ),
422   ),
423   ),
424   ),
425   ),
426   ),
427   ),
428   ),
429   ),
430   ),
431   ),
432   ),
433   ),
434   ),
435   ),
436   ),
437   ),
438   ),
439   ),
440   ),
441   ),
442   ),
443   ),
444   ),
445   ),
446   ),
447   ),
448   ),
449   ),
450   ),
451   ),
452   ),
453   ),
454   ),
455   ),
456   ),
457   ),
458   ),
459   ),
460   ),
461   ),
462   ),
463   ),
464   ),
465   ),
466   ),
467   ),
468   ),
469   ),
470   ),
471   ),
472   ),
473   ),
474   ),
475   ),
476   ),
477   ),
478   ),
479   ),
480   ),
481   ),
482   ),
483   ),
484   ),
485   ),
486   ),
487   ),
488   ),
489   ),
490   ),
491   ),
492   ),
493   ),
494   ),
495   ),
496   ),
497   ),
498   ),
499   ),
500   ),
501   ),
502   ),
503   ),
504   ),
505   ),
506   ),
507   ),
508   ),
509   ),
510   ),
511   ),
512   ),
513   ),
514   ),
515   ),
516   ),
517   ),
518   ),
519   ),
520   ),
521   ),
522   ),
523   ),
524   ),
525   ),
526   ),
527   ),
528   ),
529   ),
530   ),
531   ),
532   ),
533   ),
534   ),
535   ),
536   ),
537   ),
538   ),
539   ),
540   ),
541   ),
542   ),
543   ),
544   ),
545   ),
546   ),
547   ),
548   ),
549   ),
550   ),
551   ),
552   ),
553   ),
554   ),
555   ),
556   ),
557   ),
558   ),
559   ),
560   ),
561   ),
562   ),
563   ),
564   ),
565   ),
566   ),
567   ),
568   ),
569   ),
570   ),
571   ),
572   ),
573   ),
574   ),
575   ),
576   ),
577   ),
578   ),
579   ),
580   ),
581   ),
582   ),
583   ),
584   ),
585   ),
586   ),
587   ),
588   ),
589   ),
590   ),
591   ),
592   ),
593   ),
594   ),
595   ),
596   ),
597   ),
598   ),
599   ),
600   ),
601   ),
602   ),
603   ),
604   ),
605   ),
606   ),
607   ),
608   ),
609   ),
610   ),
611   ),
612   ),
613   ),
614   ),
615   ),
616   ),
617   ),
618   ),
619   ),
620   ),
621   ),
622   ),
623   ),
624   ),
625   ),
626   ),
627   ),
628   ),
629   ),
630   ),
631   ),
632   ),
633   ),
634   ),
635   ),
636   ),
637   ),
638   ),
639   ),
640   ),
641   ),
642   ),
643   ),
644   ),
645   ),
646   ),
647   ),
648   ),
649   ),
650   ),
651   ),
652   ),
653   ),
654   ),
655   ),
656   ),
657   ),
658   ),
659   ),
660   ),
661   ),
662   ),
663   ),
664   ),
665   ),
666   ),
667   ),
668   ),
669   ),
670   ),
671   ),
672   ),
673   ),
674   ),
675   ),
676   ),
677   ),
678   ),
679   ),
680   ),
681   ),
682   ),
683   ),
684   ),
685   ),
686   ),
687   ),
688   ),
689   ),
690   ),
691   ),
692   ),
693   ),
694   ),
695   ),
696   ),
697   ),
698   ),
699   ),
700   ),
701   ),
702   ),
703   ),
704   ),
705   ),
706   ),
707   ),
708   ),
709   ),
710   ),
711   ),
712   ),
713   ),
714   ),
715   ),
716   ),
717   ),
718   ),
719   ),
720   ),
721   ),
722   ),
723   ),
724   ),
725   ),
726   ),
727   ),
728   ),
729   ),
730   ),
731   ),
732   ),
733   ),
734   ),
735   ),
736   ),
737   ),
738   ),
739   ),
740   ),
741   ),
742   ),
743   ),
744   ),
745   ),
746   ),
747   ),
748   ),
749   ),
750   ),
751   ),
752   ),
753   ),
754   ),
755   ),
756   ),
757   ),
758   ),
759   ),
760   ),
761   ),
762   ),
763   ),
764   ),
765   ),
766   ),
767   ),
768   ),
769   ),
770   ),
771   ),
772   ),
773   ),
774   ),
775   ),
776   ),
777   ),
778   ),
779   ),
780   ),
781   ),
782   ),
783   ),
784   ),
785   ),
786   ),
787   ),
788   ),
789   ),
790   ),
791   ),
792   ),
793   ),
794   ),
795   ),
796   ),
797   ),
798   ),
799   ),
800   ),
801   ),
802   ),
803   ),
804   ),
805   ),
806   ),
807   ),
808   ),
809   ),
810   ),
811   ),
812   ),
813   ),
814   ),
815   ),
816   ),
817   ),
818   ),
819   ),
820   ),
821   ),
822   ),
823   ),
824   ),
825   ),
826   ),
827   ),
828   ),
829   ),
830   ),
831   ),
832   ),
833   ),
834   ),
835   ),
836   ),
837   ),
838   ),
839   ),
840   ),
841   ),
842   ),
843   ),
844   ),
845   ),
846   ),
847   ),
848   ),
849   ),
850   ),
851   ),
852   ),
853   ),
854   ),
855   ),
856   ),
857   ),
858   ),
859   ),
860   ),
861   ),
862   ),
863   ),
864   ),
865   ),
866   ),
867   ),
868   ),
869   ),
870   ),
871   ),
872   ),
873   ),
874   ),
875   ),
876   ),
877   ),
878   ),
879   ),
880   ),
881   ),
882   ),
883   ),
884   ),
885   ),
886   ),
887   ),
888   ),
889   ),
890   ),
891   ),
892   ),
893   ),
894   ),
895   ),
896   ),
897   ),
898   ),
899   ),
900   ),
901   ),
902   ),
903   ),
904   ),
905   ),
906   ),
907   ),
908   ),
909   ),
910   ),
911   ),
912   ),
913   ),
914   ),
915   ),
916   ),
917   ),
918   ),
919   ),
920   ),
921   ),
922   ),
923   ),
924   ),
925   ),
926   ),
927   ),
928   ),
929   ),
930   ),
931   ),
932   ),
933   ),
934   ),
935   ),
936   ),
937   ),
938   ),
939   ),
940   ),
941   ),
942   ),
943   ),
944   ),
945   ),
946   ),
947   ),
948   ),
949   ),
950   ),
951   ),
952   ),
953   ),
954   ),
955   ),
956   ),
957   ),
958   ),
959   ),
960   ),
961   ),
962   ),
963   ),
964   ),
965   ),
966   ),
967   ),
968   ),
969   ),
970   ),
971   ),
972   ),
973   ),
974   ),
975   ),
976   ),
977   ),
978   ),
979   ),
980   ),
981   ),
982   ),
983   ),
984   ),
985   ),
986   ),
987   ),
988   ),
989   ),
990   ),
991   ),
992   ),
993   ),
994   ),
995   ),
996   ),
997   ),
998   ),
999   ),
1000  )
```

- **AppBar:** Uygulama başlığı ve simgeleri ayarlanmış.
- **Image:** Uygulamanın logosu ekranda gösteriliyor.
- **CustomTextField:** Kullanıcı adı, e-posta ve parola için özel metin alanları kullanılmış. Her biri farklı controller ile bağlı.
- **Error Message:** Hata mesajı, eğer errorMessage değeri varsa, kırmızı renkte ekranda görüntüleniyor.
- **CustomButton:** Kayıt ol butonu ve giriş yap butonu özel bir buton widget'ı ile oluşturulmuş. Kayıt ol butonunda işlem yapılırken, yükleniyor simgesi (CircularProgressIndicator) gösteriliyor.

tedarik_detail_page.dart

Bu Flutter uygulamasının temel amacı, Firestore'daki "tedarik" (supply) bilgilerini görüntülemek ve bu bilgilere müdahale etmek. Uygulama bir tedarik detay sayfası (TedarikDetailPage) içeriyor ve kullanıcıya şunları sağlar:

Tedarik Bilgilerini Görüntüleme: Uygulama Firestore'dan tedarik bilgilerini çekiyor ve bu bilgileri ekranda gösteriyor (başlık, açıklama, fiyat, sektör, resim gibi).

Tedarik Sahibi Kontrolü: Eğer tedarik sahibi, kendisiyle eşleşiyorsa, düzenleme ve silme gibi işlemleri yapabilmesi için bir buton görüntüleniyor.

Resim Yükleme: Kullanıcı, yeni bir resim seçip yüklemek isteyebilir. Bu resim, imgbb API'sine yükleniyor.

Tedarik Düzenleme ve Silme: Eğer kullanıcı tedarik sahibi ise, "Düzenle / Sil" butonu görünür ve bir popup (dialog) aracılığıyla tedarik bilgilerini güncelleyebilir veya silebilir.

Başvuru Yapma: Kullanıcı tedarik sahibi değilse, "Başvur" butonu görünür ve bu buton tedarik sahibine başvuru yapmasını sağlar.

Başvuranları Görüntüleme: Tedarik sahibi, başvuruda bulunan kişileri bir liste olarak görebilir.

Profil Sayfası Linki: Tedarik sahibinin profil fotoğrafına tıklanarak, profil sayfasına yönlendirme yapılabilir.

Uygulama, kullanıcının profil fotoğrafını, başvuruları, tedarik düzenleme işlemlerini ve resim yükleme işlemlerini verimli bir şekilde yönetiyor. Bu sayede, Firestore veritabanı ile etkileşim kurarak dinamik içerik sunuyor.

index.js

1. Firestore'dan Verileri Çekmek

Başvuru yapıldığında, ilgili tedarik ve başvuran bilgileri Firestore'dan çekilir. İlk olarak, supplyId ve applicantId parametreleri kullanılarak, ilgili tedarik ve başvuran verileri sorgulanır.

```
31     const supplyDoc = await getFirestore().collection('supplies').doc(supplyId).get();
32     if (!supplyDoc.exists) {
33         console.error('Ürün bulunamadı:', supplyId);
34         return null;
35     }
```

Burada, supplies koleksiyonundaki tedarik belgesi sorgulanır. Eğer belge bulunamazsa, fonksiyon sonlandırılır.

2. Tedarik Sahibi Bilgilerini Çekmek

Tedarik belgesinden, tedarik sahibinin kimliği (created_by) alınır ve bu ID ile kullanıcı bilgileri sorgulanır.

```
50     const sellerDoc = await getFirestore().collection('users').doc(sellerId).get();
51     if (!sellerDoc.exists) {
52         console.error('Satıcı bulunamadı:', sellerId);
53         return null;
54     }
```

Tedarik sahibinin kimliği kullanılarak users koleksiyonundan satıcı bilgisi çekilir. Eğer satıcı bulunamazsa, fonksiyon sonlandırılır.

3. Başvuran Kullanıcı Bilgilerini Çekmek

Başvuran kullanıcı bilgileri de benzer şekilde sorgulanır.

```
41     const applicantDoc = await getFirestore().collection('users').doc(applicantId).get();
42     if (!applicantDoc.exists) {
43         console.error('Başvuran kullanıcı bulunamadı:', applicantId);
44         return null;
45     }
```

Başvuranın adı alınır, eğer kullanıcı adı bulunamazsa, bir yedek değer (Bir müşteri) kullanılır.

4. Bildirim Gönderme

Son olarak, `getMessaging().send()` fonksiyonu ile bildirim gönderilir. Eğer bildirim başarıyla gönderilirse, bir başarı mesajı yazdırılır. Hata durumunda hata mesajı konsola basılır.

```
71     console.log('Bildirim gönderiliyor:', message);
72
73     try {
74         await getMessaging().send(message);
75         console.log('Bildirim başarıyla gönderildi.');
```

Özet:

Bu fonksiyon, başvuru oluşturulduğunda, başvuruyu yapan kullanıcının bilgilerini ve tedarik sahibinin FCM token'ını alarak, tedarik sahibine başvuru bildirimi gönderir. Fonksiyon, her adımda hata kontrolü yapar ve bildirim gönderme işlemi sonunda başarılı ya da başarısız durumları konsola yazdırır.

theme.dart

1. Renk Paleti

```
5 class AppTheme {
6   // Renk Paleti
7   static const Color primaryColor = Color(0xFF5F5F5F); // Derin Lacivert
8   static const Color primaryContainerColor = Color(0xFF334155); // Daha koyu bir lacivert varyant
9   static const Color secondaryColor = Color(0xFFFF5722); // Canlı Turuncu
10  static const Color secondaryContainerColor = Color(0xFF42A5F5); // Daha koyu turuncu varyant
11  static const Color backgroundColor = Color(0xFFFFF5F5); // Çok Açık Gri
12  static const Color textColor = Color(0xFF212121); // Koyu Gri Yazı
13  static const Color textLightColor = Color(0xFF757575); // Açık Gri Yazı
14  static const Color cardBackground = Colors.white; // Kart Arka Planı
15  static const Color price = Color(0xFFFF4336);
```

Tema, uygulamanın genel görünümünü belirlemek için kullanılan renkleri içerir. Örneğin:

- `primaryColor`: Ana renk, açık gri.
- `primaryContainerColor`: Daha koyu bir lacivert varyant.
- `secondaryColor`: Canlı turuncu, düğme ve vurgulanan öğeler için kullanılır.
- `backgroundColor`: Arka plan rengi, çok açık gri.

2. Font Ailesi ve TextStyle

```
18     static const String fontFamily = 'Montserrat';
19
20     // TextStyle Tanımlamaları
21     static TextStyle textStyle({
22         double fontSize = 16,
23         FontWeight fontWeight = FontWeight.normal,
24         Color color = textColor,
25     }) {
26         return TextStyle(
27             fontFamily: fontFamily,
28             fontSize: fontSize,
29             fontWeight: fontWeight,
30             color: color,
31         );
32     }
```

- fontFamily: Yazı tipini Montserrat olarak belirler.
- textStyle fonksiyonu, metin stilini özelleştirir. Font boyutu, kalınlık ve renk gibi özellikler ayarlanabilir.

3. ThemeData

```
35     static ThemeData get theme {
36         return ThemeData(
37             primaryColor: primaryColor,
38             scaffoldBackgroundColor: backgroundColor,
39             cardColor: cardBackground,
40             colorScheme: ColorScheme(
41                 primary: primaryColor,
42                 primaryContainer: primaryContainerColor,
43                 secondary: secondaryColor,
44                 secondaryContainer: secondaryContainerColor,
45                 surface: Colors.white,
46                 background: backgroundColor,
47                 error: Colors.red,
48                 onPrimary: Colors.white,
49                 onSecondary: Colors.white,
50                 onSurface: textColor,
51                 onBackground: textColor,
52                 onError: Colors.white,
53                 brightness: Brightness.light,
54             ),
55         );
56     }
```

theme fonksiyonu, tüm uygulama için ThemeData objesini döndürür. Bu nesne, renkler, yazı tipleri, buton stilleri gibi tüm tema yapılandırmalarını içerir.

4. Text Theme

```
55     textTheme: TextTheme(
56         headlineLarge: textStyle(fontSize: 32, fontWeight: FontWeight.bold),
57         headlineMedium: textStyle(fontSize: 28, fontWeight: FontWeight.bold),
58         headlineSmall: textStyle(fontSize: 24, fontWeight: FontWeight.w600),
59         titleLarge: textStyle(fontSize: 20, fontWeight: FontWeight.w600),
60         titleMedium: textStyle(fontSize: 18, fontWeight: FontWeight.w500),
61         titleSmall: textStyle(fontSize: 16, fontWeight: FontWeight.w500),
62         bodyLarge: textStyle(fontSize: 16, fontWeight: FontWeight.normal),
63         bodyMedium: textStyle(fontSize: 14, fontWeight: FontWeight.normal),
64         bodySmall: textStyle(fontSize: 12, fontWeight: FontWeight.normal, color: textLightColor),
65         labelLarge: textStyle(fontSize: 16, fontWeight: FontWeight.w400),
66         labelMedium: textStyle(fontSize: 14, fontWeight: FontWeight.w400),
67         labelSmall: textStyle(fontSize: 12, fontWeight: FontWeight.w400),
68     ),
```

Uygulamanın farklı bölümlerindeki metinlerin stilini tanımlar. headlineLarge, titleLarge, bodyMedium gibi başlık ve metin stilleri burada ayarlanır.

5. Buton Temaları

```
76     buttonTheme: ButtonThemeData(  
77         buttonColor: secondaryColor,  
78         textTheme: ButtonTextTheme.primary,  
79         shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(8)),  
80     ),  
81     elevatedButtonTheme: ElevatedButtonThemeData(  
82         style: ElevatedButton.styleFrom(  
83             backgroundColor: secondaryColor, // 'primary' yerine 'backgroundColor'  
84             foregroundColor: Colors.white, // 'onPrimary' yerine 'foregroundColor'  
85             textStyle: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),  
86             shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(8)),  
87         ),  
88     ),
```

Butonların stillerini özelleştirmek için ButtonThemeData, ElevatedButtonThemeData, TextButtonThemeData ve OutlinedButtonThemeData kullanılır. Düğmelerin arka plan rengi, yazı tipi ve şekli belirlenmiştir.

6. Input Dekorasyonu

```
103     inputDecorationTheme: InputDecorationTheme(  
104         filled: true,  
105         fillColor: Colors.white,  
106         hintStyle: TextStyle(color: textLightColor),  
107         labelStyle: TextStyle(),  
108         border: OutlineInputBorder(  
109             borderRadius: BorderRadius.circular(8),  
110             borderSide: BorderSide(color: primaryColor),  
111         ),  
112         focusedBorder: OutlineInputBorder(  
113             borderRadius: BorderRadius.circular(8),  
114             borderSide: BorderSide(color: secondaryColor),  
115         ),  
116     ),
```

Metin alanlarının ve giriş bileşenlerinin stilini tanımlar. Burada, giriş alanlarının kenarlıkları, dolgu rengi ve odaklanma rengini belirledik.

7. AppBar Tema

```

70 appBarTheme: AppBarTheme(
71   color: primaryColor,
72   iconTheme: IconThemeData(color: Colors.white),
73   titleTextStyle: textStyle(fontSize: 20, fontWeight: FontWeight.bold, color: Colors.white),
74   elevation: 0,
75 ),

```

Uygulamanın üst kısmındaki AppBar için renkler ve stil ayarlanır. Burada, başlık rengi ve simgelerin rengi belirlenir.

8. Icon Tema

```

117 iconTheme: IconThemeData(color: primaryColor),
118 visualDensity: VisualDensity.adaptivePlatformDensity,
119 );

```

Uygulamanın ikonlarının rengini belirledik.

main.dart

1. Firebase Entegrasyonu ve FCM (Firebase Cloud Messaging)

```

157 Future<void> setupFirebaseMessaging() async {
158   FirebaseMessaging messaging = FirebaseMessaging.instance;
159
160   NotificationSettings settings = await messaging.requestPermission(
161     alert: true,
162     badge: true,
163     sound: true,
164   );
165
166   if (settings.authorizationStatus == AuthorizationStatus.authorized) {
167     print('Kullanıcı bildirimlere izin verdi.');
```

```

168   } else {
169     print('Kullanıcı bildirimlere izin vermedi.');
```

```

170   }
171
172   String? token = await messaging.getToken();
173   print('FCM Token: $token');
```

```

174
175   FirebaseMessaging.onMessage.listen((RemoteMessage message) {
176     print('Ön planda bildirim alındı: ${message.notification?.title}');
```

```

177
178     // Bildirimi manuel olarak göster
179     showNotification(
180       message.notification?.title ?? 'Bildirim',
181       message.notification?.body ?? 'Yeni bildirim',
182     );
183   });
184 }

```

Firebase'i uygulamaya entegre eder ve Firebase Cloud Messaging (FCM) ile bildirimler alır.

- **Firebase Auth** kullanılarak, kullanıcı girişi kontrolü sağlanır.
- **Firebase Messaging** kullanarak bildirim alınır ve kullanıcıya bildirimi gösterir.

2. Yerel Bildirimler

```
296 Future<void> showNotification(String title, String body) async {
297   const AndroidNotificationDetails androidPlatformChannelSpecifics =
298     AndroidNotificationDetails(
299       'high importance channel', // channelId
300       'High Importance Notifications', // channelName
301       channelDescription:
302         'This channel is used for important notifications.', // channelDescription
303       importance: Importance.max,
304       priority: Priority.high,
305       largeIcon: DrawableResourceAndroidBitmap("notification"),
306       icon: "@mipmap/ic_launcher",
307       playSound: true,
308       sound: RawResourceAndroidNotificationSound("notification_sound"), // Ses dosyası adı (uzantı olmadan)
309     );
310
311   const NotificationDetails platformChannelSpecifics =
312     NotificationDetails(android: androidPlatformChannelSpecifics);
313
314   await flutterLocalNotificationsPlugin.show(
315     0,
316     title,
317     body,
318     platformChannelSpecifics,
319   );
320 }
```

flutter_local_notifications paketi kullanılarak uygulama içindeki bildirimler yönetilir. Kullanıcıya bildirimlerin görsel ve sesli olarak iletilmesi sağlanır.

- Bildirimler, uygulama açıkken ve arka planda çalışırken gösterilir.
- Özel bir ses dosyası ve simge ile bildirimler özelleştirilir.

3. Splash Screen ve Ses

```
59 void playSound() async {
60   // Ses dosyasını çal
61   await audioPlayer.play(AssetSource('sounds/splash_sound.mp3'));
62
63   // Sesin bitmesini beklemek için bir Completer kullan
64   var completer = Completer<void>();
65
66   // Ses bittiğinde Completer'ı tamamla
67   audioPlayer.onPlayerComplete.listen((event) {
68     completer.complete();
69   });
70
71   // Ses bitene kadar bekle
72   await completer.future;
73 }
74
```

Splash ekranı, animasyonlu bir ekran ile açılır. Ekran süresi boyunca bir ses çalar ve ardından giriş kontrolüne geçilir.

- Lottie ile animasyonlu splash ekranı gösterilir.
- audioplayers paketi ile ses dosyası çalınır.

4. Ana Sayfa ve Navigasyon

```

102   Widget build(BuildContext context) {
103     return StreamBuilder<User?>(
104       stream: FirebaseAuth.instance.authStateChanges(),
105       builder: (context, snapshot) {
106         if (snapshot.connectionState == ConnectionState.waiting) {
107           return const Scaffold(
108             body: Center(child: CircularProgressIndicator()),
109           );
110         }
111
112         return snapshot.hasData ? MainPage() : LoginPage();
113       },
114     );
115   }
116 }
117

```

Ana sayfa, üç ana sekmeden oluşur: **Ana Sayfa**, **Başvurularım**, ve **Profilim**. Kullanıcı alt menüdeki seçeneklere tıklayarak sekmeler arasında geçiş yapabilir.

- BottomNavigationBar ile menü seçimleri yapılır.
- StreamBuilder kullanılarak kullanıcının oturum durumu kontrol edilir ve giriş yapmışsa ana sayfaya yönlendirilir.

5. Tema Kullanımı

```

36   return MaterialApp(
37     title: 'Kopernik Pizza',
38     theme: AppTheme.theme,
39     home: SplashScreen(),
40   );

```

Uygulamanın tüm görsel stilini tanımlamak için AppTheme sınıfı kullanılır. Ana renkler, yazı tipi, butonlar, bildirimler gibi tüm UI bileşenlerinin görünümü burada belirlenir.

6. Logout İşlemi

```

249   IconButton(
250     icon: const Icon(Icons.logout,
251       color: AppTheme.secondaryColor, size: 28),
252     onPressed: () async {
253       await FirebaseAuth.instance.signOut();
254     },
255   ),

```


Kullanıcı, profil ekranında "Çıkış Yap" butonuna tıkladığında Firebase oturumu sonlandırılır.

7. Ana Sayfa Başlığı

```
132 String getPageTitle(int index) {
133     switch (index) {
134         case 0:
135             return 'Ana Sayfa';
136         case 1:
137             return 'Başvurularım';
138         case 2:
139             return 'Profilim';
140         default:
141             return '';
142     }
143 }
144
```

Sayfanın başlığını dinamik olarak değiştiren bir yapı bulunuyor. MainPage'de, alt menüdeki seçime göre başlık güncelleniyor.

settings.gradle

1. pluginManagement Bloğu

pluginManagement, projenin eklenti yönetimini tanımlar. Bu bölümde eklentilerin nereden alınacağı ve hangi Flutter SDK'sının kullanılacağı belirtilir.

Flutter SDK Yolu

```
1 pluginManagement {
2     def flutterSdkPath = {
3         def properties = new Properties()
4         file("local.properties").withInputStream { properties.load(it) }
5         def flutterSdkPath = properties.getProperty("flutter.sdk")
6         assert flutterSdkPath != null, "flutter.sdk not set in local.properties"
7         return flutterSdkPath
8     }
9 }
```

- **local.properties** dosyası, Flutter SDK'nın kurulu olduğu yolu tanımlar.
- Eğer bu yol tanımlı değilse, bir hata mesajı gösterir.

Flutter'ın Build Araçları Dahil Ediliyor

```
10 includeBuild("$flutterSdkPath/packages/flutter_tools/gradle")
11
```

Flutter araçları, Gradle'a özel olarak dahil edilir.

Depo Tanımları

```
12     repositories {
13         google()
14         mavenCentral()
15         gradlePluginPortal()
16     }
```

- **google()**: Google'ın Maven deposu.
- **mavenCentral()**: Java ve Android projeleri için sıkça kullanılan genel bir Maven deposu.
- **gradlePluginPortal()**: Gradle eklentilerini indirir.

2. plugins Bloğu

```
19     plugins {
20         id "dev.flutter.flutter-plugin-loader" version "1.0.0"
21         id "com.android.application" version "8.3.2" apply false
22         id "org.jetbrains.kotlin.android" version "1.8.22" apply false
23     }
```

- **flutter-plugin-loader**: Flutter projelerini Gradle ile yönetmek için bir eklenti.
- **com.android.application**: Android uygulamalarını derlemek için gerekli.
- **org.jetbrains.kotlin.android**: Kotlin dilinde Android geliştirme desteği.

pubspec.yaml

```
33     dependencies:
34         flutter:
35             sdk: flutter
36         firebase_core: ^3.9.0
37         firebase_auth: ^5.3.4
38         cloud_firestore: ^5.6.0
39         image_picker: ^1.1.2
40         http: ^1.2.2
41         firebase_messaging: ^15.1.6
42         cupertino_icons: ^1.0.8
43         flutter_local_notifications: ^18.0.1
44         flutter_native_splash: ^2.4.4
45         audioplayers: ^6.1.0
46         lottie: ^3.3.0
```

Firestore Bağımlılıkları

Firestore servislerini projeye entegre etmek için kullanılan paketler:

- **firebase_core**: Firestore özelliklerini kullanmak için temel yapı taşıdır.
- **firebase_auth**: Kullanıcı kayıt, giriş, kimlik doğrulama işlemlerini yönetir.
- **cloud_firestore**: Firestore'ın gerçek zamanlı veritabanını kullanmanızı sağlar.

- **firebase_messaging**: Push bildirimleri gönderme ve alma işlevini destekler.

UI ve Medya İşlevleri

Kullanıcı arayüzünü zenginleştiren ve medya işlemlerini kolaylaştıran paketler:

- **image_picker**: Fotoğraf veya video seçmek için kullanılır.
- **audioplayers**: Ses dosyalarını çalmak için gerekli işlevleri sunar.
- **lottie**: Lottie animasyonları ile modern ve etkileyici UI tasarımları sağlar.
- **flutter_local_notifications**: Cihazda yerel bildirimlerin gösterimini destekler.

Destekleyici Araçlar

Uygulamanın genel işlevselliğini artıran yardımcı paketler:

- **http**: REST API'lerle veri alışverişini yönetmek için kullanılır.
- **flutter_native_splash**: Özelleştirilmiş splash ekranları oluşturmayı kolaylaştırır.
- **cupertino_icons**: iOS için Cupertino tarzı ikon seti sağlar.

WIDGETLAR

Custombutton kodları

```
class CustomButton extends StatelessWidget {
  final String text;
  final VoidCallback onPressed;
  final Color? backgroundColor; // Buton arka plan rengi
  final Color? textColor; // Buton metin rengi
  final TextStyle? textStyle; // Buton metin stili
  final BorderRadius? borderRadius; // Buton köşe yuvarlaklığı

  const CustomButton({
    Key? key,
    required this.text,
    required this.onPressed,
    this.backgroundColor, // Opsiyonel
    this.textColor, // Opsiyonel
    this.textStyle, // Opsiyonel
    this.borderRadius, // Opsiyonel
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return SizedBox(
      width: double.infinity, // Butunun tam genişlik almasını sağlar
      child: ElevatedButton(
        onPressed: onPressed,
        child: Text(
          text,
          style: textStyle ??
            TextStyle(
              color: textColor ?? Colors.white,
              fontSize: 16,
              fontWeight: FontWeight.bold,
            ),
        ),
      ),
      style: ElevatedButton.styleFrom(
        backgroundColor: backgroundColor ?? Theme.of(context).colorScheme.primary, // 'primary' yerine 'backgroundColor' kullanıldı
        foregroundColor: textColor ?? Colors.white, // Metin rengi
        shape: RoundedRectangleBorder(
          borderRadius: borderRadius ?? BorderRadius.circular(8),
        ),
        padding: EdgeInsets.symmetric(vertical: 16, horizontal: 24),
      ),
    );
  }
}
```

Sınıf Yapısı

1. Değişkenler

- text (Zorunlu):** Buton üzerinde görüntülenecek metni belirtir.
- onPressed (Zorunlu):** Butona tıklandığında çalışacak geri çağırma fonksiyonudur.
- backgroundColor (Opsiyonel):** Butonun arka plan rengini özelleştirmek için kullanılır.
- textColor (Opsiyonel):** Buton üzerindeki metnin rengini belirtir.
- textStyle (Opsiyonel):** Buton metninin stilini (font boyutu, kalınlık, vb.) özelleştirmek için kullanılır.
- borderRadius (Opsiyonel):** Butonun köşe yuvarlaklığını ayarlamak için kullanılır.

2. Yapıcı (Constructor)

- CustomButton:** Butonun özelliklerini tanımlayan yapıcıdır.
- required** ile işaretlenen **text** ve **onPressed**, bileşen oluşturulurken zorunlu olarak sağlanmalıdır.
- this.backgroundColor**, **this.textColor**, **this.textStyle**, ve **this.borderRadius** opsiyonel olarak özelleştirilebilir.

build Metodu

- **Amaç:** Bileşenin kullanıcı arayüzünde nasıl görüneceğini tanımlar.
- **İşleyiş:**
 - **SizedBox:**
 - **width: double.infinity:** Butonun genişliğini olabildiğince büyük yapar (bulunduğu alanı kaplar).
 - **ElevatedButton:**
 - Flutter'ın temel buton widget'ıdır. Özel stillendirme ve tıklama özelliklerini destekler.
 - **onPressed:**
 - Kullanıcının butona tıklaması durumunda çağrılan fonksiyondur.
 - **child:**
 - Buton üzerinde görüntülenecek Text widget'ını tanımlar.
 - **style: textStyle** değişkeni sağlanmışsa onu kullanır; aksi takdirde varsayılan bir stil uygular.
 - **style:**

- `backgroundColor`: Butonun arka plan rengi. Belirtilmemişse, varsayılan olarak `Theme.of(context).colorScheme.primary` kullanılır.
- `foregroundColor`: Buton üzerindeki metnin rengi. Belirtilmemişse, `Colors.white` varsayılır.
- `shape`:
 - Butonun şekli ve köşe yuvarlaklığı. `borderRadius` sağlanmamışsa varsayılan olarak `BorderRadius.circular(8)` atanır.
- `padding`:
 - Butonun iç boşluğunu dikeyde 16, yatayda 24 olacak şekilde ayarlar.

CustomCard kodları

```

import 'package:flutter/material.dart';
import '../theme/theme.dart'; // theme.dart dosyasını doğru şekilde import etmek için

class CustomCard extends StatelessWidget {
  final String title;
  final String description;
  final String sector;
  final String imageUrl; // Resim URL'si
  final VoidCallback onTap;

  CustomCard({
    required this.title,
    required this.description,
    required this.sector,
    required this.imageUrl, // Resim URL'si parametresi
    required this.onTap,
  });

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: onTap,
      child: Card(
        elevation: 8, // Daha derin gölge
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(16), // Daha yuvarlak köşeler
        ),
        color: Colors.white, // Kartın beyaz arka planı
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            // Fotoğraf kısmı
            ClipRect(
              borderRadius: BorderRadius.only(
                topLeft: Radius.circular(16),
                topRight: Radius.circular(16),
              ),
              child: Image.network(
                imageUrl,
                width: double.infinity, // Resim kartın genişliğine uyum sağlar
                height: 180, // Resim yüksekliği
                fit: BoxFit.cover, // Resmi kapsayacak şekilde yerleştirme
              ),
            ),
            Padding(
              padding: const EdgeInsets.all(16.0),
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  // Başlık kısmı
                  Text(
                    title,
                    style: TextStyle(
                      fontSize: 20,
                      fontWeight: FontWeight.bold,
                      color: AppTheme.primaryColor, // Vurgulu renk
                    ),
                  ),
                  SizedBox(height: 12),

```



```

    ),
    Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          // Başlık kısmı
          Text(
            title,
            style: TextStyle(
              fontSize: 20,
              fontWeight: FontWeight.bold,
              color: AppTheme.primaryColor, // Vurgulu renk
            ),
          ),
          SizedBox(height: 12),

          // Açıklama kısmı
          Text(
            description,
            style: TextStyle(fontSize: 14, color: Colors.grey[700]),
          ),
          SizedBox(height: 12),

          // Sektör bilgisi ve ikon
          Row(
            children: [
              Icon(
                Icons.business, // Sektör ikonu
                size: 16,
                color: AppTheme.primaryColor,
              ),
              SizedBox(width: 8),
              Text(
                "Sector: $sector",
                style: TextStyle(fontSize: 12, color: Colors.grey[500]),
              ),
            ],
          ),
        ],
      ),
    ),
  ],
),
);
}

```

Sınıf Yapısı

1. Değişkenler

- title (Zorunlu):** Kartın başlık kısmında görüntülenecek metin.
- description (Zorunlu):** Kartın açıklama kısmında gösterilecek metin.
- sector (Zorunlu):** Kartta sektör bilgisini belirtir.
- imageUrl (Zorunlu):** Kartın üst kısmında gösterilecek bir resim URL'si.
- onTap (Zorunlu):** Kartın tıklanması durumunda çalışacak geri çağırma fonksiyonu.

2. Yapıcı (Constructor)

- CustomCard:** Kart bileşeninin özelliklerini tanımlayan yapıcıdır.
- required** ile işaretlenen tüm değişkenler, bileşen oluşturulurken sağlanmalıdır.

build Metodu

- **Amaç:** Bileşenin kullanıcı arayüzünde nasıl görüneceğini tanımlar.
- **İşleyiş:**
 - **GestureDetector:**
 - Kartın üzerine tıklanabilir olmasını sağlar.
 - **onTap:** Kullanıcının tıklama hareketini yakalayan geri çağırma fonksiyonu.
 - **Card:**
 - Flutter'ın temel kart widget'ıdır. Gölge, yuvarlatılmış köşeler ve renk gibi özellikler içerir.
 - **elevation: 8:** Daha belirgin bir gölge sağlar.
 - **shape:** Yuvarlatılmış köşeler oluşturur (16 birim).
 - **Column:**
 - Kart içeriğini dikey olarak düzenler.
 - **ClipRRect:**
 - Kartın üst kısmındaki resmi yuvarlatılmış köşelere sahip hale getirir.
 - **Image.network:**
 - Verilen **imageUrl** ile internet üzerinden bir resim yükler.
 - **fit: BoxFit.cover:** Resmi, alanı tamamen kaplayacak şekilde yerleştirir.

CustomTextField kodları

```
import 'package:flutter/material.dart';

class CustomTextField extends StatelessWidget {
  final TextEditingController controller;
  final String labelText;
  final TextInputType keyboardType;
  final Function(String) onChanged;
  final InputDecoration? decoration;
  final bool obscureText; // Yeni parametre

  const CustomTextField({
    Key? key,
    required this.controller,
    required this.labelText,
    required this.keyboardType,
    required this.onChanged,
    this.decoration,
    this.obscureText = false, // Varsayılan olarak false
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return TextField(
      controller: controller,
      keyboardType: keyboardType,
      onChanged: onChanged,
      obscureText: obscureText, // Yeni parametre kullanıldı
      decoration: decoration ??
        InputDecoration(
          labelText: labelText,
          filled: true,
          fillColor: Colors.white,
          border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(8),
          ),
          focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(8),
            borderSide: BorderSide(
              color: Colors.blue, // İstediğiniz renk
            ),
          ),
        ),
    );
  }
}
```

Sınıf Yapısı

1. Değişkenler

- controller (Zorunlu):** Metin alanındaki değerleri kontrol etmek için kullanılan bir `TextEditingController`.
- labelText (Zorunlu):** Metin alanının etiketini belirler.
- keyboardType (Zorunlu):** Klavye türünü ayarlar (ör. metin, sayı, e-posta).
- onChanged (Zorunlu):** Kullanıcının giriş yaptığı her değişiklikte çalıştırılan geri çağırma fonksiyonu.
- decoration (Opsiyonel):** Metin alanının görünümünü özelleştirmek için kullanılan bir `InputDecoration`.
- obscureText (Opsiyonel):** Metin alanındaki girişlerin gizlenip gizlenmeyeceğini belirler. Genellikle şifre girişlerinde kullanılır (varsayılan: false).

2. Yapıcı (Constructor)

- CustomTextField:** Metin alanının işlevselliğini ve görünümünü tanımlayan yapıcıdır.
- required** ile işaretlenen **controller**, **labelText**, **keyboardType**, ve **onChanged**, bileşen oluşturulurken zorunlu olarak sağlanmalıdır.
- decoration** ve **obscureText** isteğe bağlı olarak özelleştirilebilir.

build Metodu

- **Amaç:** Bileşenin kullanıcı arayüzünde nasıl görüneceğini tanımlar.
- **İşleyiş:**
 - **TextField:**
 - Flutter'ın temel metin giriş widget'ıdır.
 - **controller:**
 - Metin alanındaki değerleri kontrol eder ve dışarıya aktarır.
 - **keyboardType:**
 - Klavye türünü belirtir (ör. `TextInputType.emailAddress`, `TextInputType.number`).
 - **onChanged:**
 - Kullanıcı metin alanına her yeni harf veya rakam girdiğinde çalışır.
 - **obscureText:**

- Girişlerin gizlenip gizlenmeyeceğini kontrol eder (şifre alanları için kullanılır).
- **decoration:**
 - Sağlanmamışsa, varsayılan bir InputDecoration kullanılır.
 - Varsayılan dekorasyon:
 - **labelText:** Etiket metni.
 - **filled** ve **fillColor:** Arka planı beyaz olarak doldurur.
 - **border:** Yuvarlatılmış bir kenar çizer.
 - **focusedBorder:** Odaklanıldığında mavi bir kenar çizer.

FullScreenImageView kodları

```
import 'package:flutter/material.dart';

class FullScreenImageView extends StatelessWidget {
  final String imageUrl;

  const FullScreenImageView({Key? key, required this.imageUrl}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.black,
      body: Stack(
        children: [
          Center(
            child: InteractiveViewer(
              panEnabled: true, // Resmi kaydırmayı etkinleştir
              minScale: 0.5, // Minimum yakınlaştırma seviyesi
              maxScale: 4.0, // Maksimum yakınlaştırma seviyesi
              child: Image.network(
                imageUrl,
                fit: BoxFit.contain,
              ),
            ),
          ),
          Positioned(
            top: 40,
            left: 16,
            child: IconButton(
              icon: const Icon(Icons.close, color: Colors.white),
              onPressed: () {
                Navigator.pop(context);
              },
            ),
          ),
        ],
      ),
    );
  }
}
```

Sınıf Yapısı

1. Değişkenler

- a. **imageUrl** (**Zorunlu**): Görüntülenecek resmin URL'si.

2. Yapıcı (Constructor)

- a. **FullScreenImageView**: Tam ekran görüntüleme bileşenini oluşturur.
- b. **required** ile işaretlenen **imageUrl**, bileşen oluşturulurken sağlanmalıdır.

build Metodu

- **Amaç**: Bileşenin kullanıcı arayüzünde nasıl görüneceğini tanımlar.
- **İşleyiş**:
 - **Scaffold**:
 - Uygulama sayfasını temsil eder.
 - **backgroundColor: Colors.black**:
 - Arka plan rengi siyah olarak ayarlanır.
 - **Stack**:
 - Katmanlı bir yapı oluşturur (resim ve kapatma düğmesi için).
 - **Center**:
 - Resmi sayfanın ortasında hizalar.
 - **InteractiveViewer**:
 - Kullanıcıların resmi yakınlaştırma ve kaydırma işlemlerini yapmasına olanak tanır.
 - **panEnabled: true**: Resim kaydırılabilir.
 - **minScale**: Resmin minimum yakınlaştırma seviyesi.
 - **maxScale**: Resmin maksimum yakınlaştırma seviyesi.
 - **child**: Görüntü olarak **Image.network** kullanılır.
 - **Positioned**:
 - Kapatma düğmesini yerleştirmek için kullanılır.
 - Sol üst köşeye hizalanır (**top: 40, left: 16**).
 - **IconButton**:
 - Bir kapatma ikonu içerir.
 - **onPressed**:
 - Tıklandığında, **Navigator.pop(context)** ile tam ekran modundan çıkılır.

SupplyCard Kodları

```
import 'package:flutter/material.dart';

class SupplyCard extends StatelessWidget {
  final String title;
  final String description;
  final String imageUrl;
  final String price;

  SupplyCard({
    required this.title,
    required this.description,
    required this.imageUrl,
    required this.price,
  });

  @override
  Widget build(BuildContext context) {
    return Card(
      elevation: 5,
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(15),
      ),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          ClipRRect(
            borderRadius: BorderRadius.only(
              topLeft: Radius.circular(15),
              topRight: Radius.circular(15),
            ),
            child: Image.network(
              imageUrl,
              width: double.infinity,
              height: 180,
              fit: BoxFit.cover,
            ),
          ),
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: Text(
              title,
              style: TextStyle(
                fontSize: 18,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 8.0),
            child: Text(
              description,
              maxLines: 2,
              overflow: TextOverflow.ellipsis,
              style: TextStyle(fontSize: 14, color: Colors.grey),
            ),
          ),
          Spacer(),
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: Text(
              'Price: $price',
              style: TextStyle(
                fontSize: 16,
                fontWeight: FontWeight.bold,
                color: Colors.green,
              ),
            ),
          ),
        ],
      ),
    );
  }
}
```


Sınıf Yapısı

1. Değişkenler

- title (Zorunlu):** Kartın başlık kısmında görüntülenecek metin.
- description (Zorunlu):** Kartın açıklama kısmında gösterilecek metin.
- imageUrl (Zorunlu):** Kartın üst kısmında gösterilecek bir resim URL'si.
- price (Zorunlu):** Kartın fiyat bilgisini belirtir.

2. Yapıcı (Constructor)

- SupplyCard:** Kart bileşeninin özelliklerini tanımlayan yapıcıdır.
- required** ile işaretlenen tüm değişkenler, bileşen oluşturulurken sağlanmalıdır.

build Metodu

- Amaç:** Bileşenin kullanıcı arayüzünde nasıl görüneceğini tanımlar.
- İşleyiş:**
 - Card:**
 - Flutter'ın temel kart widget'ıdır.
 - elevation: 5:** Daha belirgin bir gölge sağlar.
 - shape:** Kartın köşelerini 15 birimle yuvarlar.
 - Column:**
 - Kartın içeriğini dikey bir düzen içerisinde yerleştirir.
 - crossAxisAlignment: CrossAxisAlignment.start:** İçeriği sola hizalar.

Kart İçeriği

1. Resim (imageUrl)

- ClipRRect:**
 - Resmi yuvarlatılmış köşelere sahip hale getirir.
- Image.network:**
 - Verilen imageUrl ile internet üzerinden bir resim yükler.
 - width: double.infinity:** Resmi kartın genişliğine göre ayarlar.
 - height: 180:** Resmin sabit yüksekliğini belirler.
 - fit: BoxFit.cover:** Resmi kartın alanını tamamen kaplayacak şekilde yerleştirir.

2. Başlık (title)

a. **Text:**

i. Kartın başlık kısmını temsil eder.

ii. **Stil:**

1. Font boyutu: 18

2. Kalınlık: Bold

3. Açıklama (description)

a. **Text:**

i. Kartın açıklama kısmını temsil eder.

ii. **Stil:**

1. Font boyutu: 14

2. Renk: Colors.grey

iii. **maxLines: 2:**

1. En fazla 2 satır gösterir.

iv. **overflow: TextOverflow.ellipsis:**

1. Uzun metinlerde üç nokta kullanarak kesme yapar.

4. Fiyat (price)

a. **Spacer:**

i. Fiyat bilgisini kartın alt kısmına yerleştirmek için diğer içeriklerle boşluk bırakır.

b. **Text:**

i. Fiyat bilgisini temsil eder.

ii. **Stil:**

1. Font boyutu: 16

2. Kalınlık: Bold

3. Renk: Colors.green

TedarikCard Kodları

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:odev/theme/theme.dart';
import '../pages/tedarik_detail_page.dart';

class TedarikCard extends StatefulWidget {
  final String docId;
  final String username;
  final String createdBy;
  final String userId;
  final String title;
  final String description;
  final String price;
  final String sector;
  final String imageUrl;
  final VoidCallback? onApply;
  final Function(String userId)? onUsernameTap;

  const TedarikCard({
    Key? key,
    required this.docId,
    required this.username,
    required this.createdBy,
    required this.userId,
    required this.title,
    required this.description,
    required this.sector,
    required this.imageUrl,
    required this.price,
    this.onApply,
    this.onUsernameTap,
  }) : super(key: key);

  @override
  State<TedarikCard> createState() => _TedarikCardState();
}

class _TedarikCardState extends State<TedarikCard> {
  @override
  Widget build(BuildContext context) {
    final currentUser = FirebaseAuth.instance.currentUser;
    final bool isOwner =
      (currentUser != null && currentUser.uid == widget.createdBy);

    // Eğer kendi ilanımız ise buton göstermeyelim
    if (isOwner) {
      return _buildCard(context, null, hasApplied: false, isOwner: true);
    }

    // Eğer kullanıcı giriş yapmamışsa yine buton olmayabilir
    if (currentUser == null) {
      return _buildCard(context, null, hasApplied: false, isOwner: false);
    }

    // Burada "başvuru var mı?" kontrolü için StreamBuilder kullanıyoruz
    final docRef = FirebaseFirestore.instance
      .collection('supplies')
      .doc(widget.docId)
      .collection('applications')
      .doc(currentUser.uid);

    return StreamBuilder<DocumentSnapshot>({
      stream: docRef.snapshots(),
      builder: (context, snapshot) {
        // Henüz veri gelmedi
        if (snapshot.connectionState == ConnectionState.waiting) {
          return _buildCard(context, null, hasApplied: false, loading: true);
        }
      }
    });
  }
}
```

```

/// [applyCallback] null ise başvuru butonuna bastı, null değilse aktif.
/// [hasApplied] => true ise buton gri / "Başvuru Yapıldı"
Widget _buildCard(
  BuildContext context,
  VoidCallback? applyCallback, {
  required bool hasApplied,
  bool loading = false,
  bool isOwner = false,
}) {
  return GestureDetector(
    // Kartın tamamına tıklayınca TedarikDetailPage(docId: docId) aç
    onTap: () {
      Navigator.push(
        context,
        MaterialPageRoute(
          builder: (context) => TedarikDetailPage(docId: widget.docId),
        ),
      );
    },
    child: Card(
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(16),
      ),
      elevation: 4,
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          // Üstteki fotoğraf
          ClipRRect(
            borderRadius: const BorderRadius.only(
              topLeft: Radius.circular(16),
              topRight: Radius.circular(16),
            ),
            child: Image.network(
              widget.imageUrl,
              fit: BoxFit.cover,
              height: 100,
              width: double.infinity,
              errorBuilder: (context, error, stackTrace) {
                // Eğer imageUrl hatalıysa, bir yedek resim gösterebilirsin
                return Container(
                  height: 100,
                  color: Colors.grey[300],
                  alignment: Alignment.center,
                  child: const Icon(Icons.image_not_supported),
                );
              },
            ),
          ),
          // Kullanıcının adı satırı
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                // Kullanıcı adını tıklanabilir yap
                GestureDetector(
                  onTap: () => widget.onUsernameTap?.call(widget.userId),
                  child: Text(
                    widget.username,
                    style: const TextStyle(
                      fontSize: 14,
                      color: Colors.grey,
                      fontStyle: FontStyle.italic,
                      decoration: TextDecoration.underline,
                    ),
                  ),
                ),
                const Icon(Icons.person, size: 16, color: Colors.grey),
              ],
            ),
          ),
        ],
      ),
    ),
  );
}

```

```

class _TedarikCardState extends State<TedarikCard> {
  }) {
    return GestureDetector(
      child: Card(
        child: Column(
          // Sektör
          Text(
            widget.sector,
            maxLines: 1,
            overflow: TextOverflow.ellipsis,
            style: const TextStyle(
              fontWeight: FontWeight.bold,
              fontSize: 18,
              color: Colors.black54,
            ),
          ),
          Spacer(),
          // Fiyat
          Text(
            widget.price + " TL",
            maxLines: 1,
            overflow: TextOverflow.ellipsis,
            style: const TextStyle(
              fontSize: 20,
              color: AppTheme.price,
              fontWeight: FontWeight.w700,
            ),
          ),
        ),
      ),
    ),

    // Eğer Loading ise (snapshot bekleniyorsa), CircularProgressIndicator gösterebilirsiniz
    if (loading)
      Container(
        padding: const EdgeInsets.symmetric(vertical: 12),
        child: const Center(child: CircularProgressIndicator()),
      )
    else if (!isOwner)
      // Kullanıcı sahibi değil => Buton göster
      GestureDetector(
        onTap: applyCallback, // null ise pasif olur
        child: Container(
          width: double.infinity,
          padding: const EdgeInsets.all(16),
          decoration: BoxDecoration(
            color: hasApplied
              ? AppTheme.textLightColor
              : AppTheme.secondaryColor,
            borderRadius: const BorderRadius.only(
              bottomLeft: Radius.circular(16),
              bottomRight: Radius.circular(16),
            ),
          ),
        ),
        alignment: Alignment.center,
        child: Text(
          hasApplied ? 'Başvuru Yapıldı' : 'Başvur',
          style: const TextStyle(
            color: Colors.white,
            fontSize: 16,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
    ),
  ),
);
}

```

Sınıf Yapısı

1. Değişkenler

- docId (Zorunlu):** Kartın Firebase doküman kimliği.
- username (Zorunlu):** Tedarik kartını oluşturan kullanıcının adı.
- createdBy (Zorunlu):** Tedarik kartını oluşturan kullanıcının UID'si.
- userId (Zorunlu):** Tedarik kartına başvuru yapan kullanıcının UID'si.
- title (Zorunlu):** Kartın başlığı.
- description (Zorunlu):** Kartın açıklama kısmında gösterilecek metin.
- price (Zorunlu):** Kartın fiyat bilgisi.
- sector (Zorunlu):** Kartın sektör bilgisi.
- imageUrl (Zorunlu):** Kartın üst kısmında gösterilecek bir resim URL'si.
- onApply (Opsiyonel):** Kullanıcı başvuru butonuna tıkladığında çağrılacak geri çağırma fonksiyonu.
- onUsernameTap (Opsiyonel):** Kullanıcı adı üzerine tıklanıldığında çağrılacak bir fonksiyon.

2. Yapıcı (Constructor)

- TedarikCard:** Kart bileşeninin tüm özelliklerini tanımlayan yapıcıdır.
- required** ile işaretlenen değişkenler, bileşenin oluşturulması sırasında zorunlu olarak sağlanmalıdır.

build Metodu

- Amaç:** Bileşenin kullanıcı arayüzünde nasıl görüneceğini tanımlar.
- İşleyiş:**
 - Firestore kullanarak, kullanıcı oturumuna göre kartın dinamik olarak nasıl görüntüleneceğine karar verir.
 - Kullanıcı kartın sahibiyse veya oturum açmamışsa buton göstermez.
 - Kullanıcı kartın sahibi değilse başvuru durumu kontrol edilir ve uygun buton gösterilir.

Kart İskeleti: _buildCard

1. Üst Kısım: Resim

- ClipRect:**
 - Kartın üst kısmındaki resmi yuvarlatılmış köşelere sahip hale getirir.

b. **Image.network:**

i. Verilen `imageUr1` ile internet üzerinden bir resim yükler.

ii. **errorBuilder:**

1. Resim yüklenemediğinde yedek bir içerik (ikon) gösterir.

2. Kullanıcı Adı Satırı

a. **GestureDetector:**

i. Kullanıcı adını tıklanabilir hale getirir.

b. **Text:**

i. Kullanıcının adı, italik ve altı çizili bir yazı stiliyle görüntülenir.

3. Başlık

a. **Container:**

i. Kartın başlığını içeren bir alan sağlar.

b. **Stil:**

i. Maksimum 1 satır, kalın ve büyük font boyutuyla yazılmıştır.

4. Açıklama

a. **Text:**

i. Açıklama metni, en fazla 4 satır ve taşma durumunda üç nokta (...) ile kesilir.

5. Alt Kısım: Sektör ve Fiyat

a. **Row:**

i. Sektör ve fiyat bilgisi, yan yana bir düzen içinde gösterilir.

b. **Sektör:**

i. Bir ikon ve yazıyla temsil edilir.

c. **Fiyat:**

i. Fiyat bilgisi büyük ve kalın fontla, uygulama temasıyla uyumlu renkte gösterilir.

6. Alt Kısım: Buton

a. Kullanıcı kartın sahibi değilse:

i. **applyCallback:**

1. Kullanıcı başvuru yapmamışsa aktif hale gelir.

ii. **Stil:**

1. Başvuru yapılmışsa buton gri ve pasif; aksi takdirde uygulama temasına uygun renkli ve aktif.

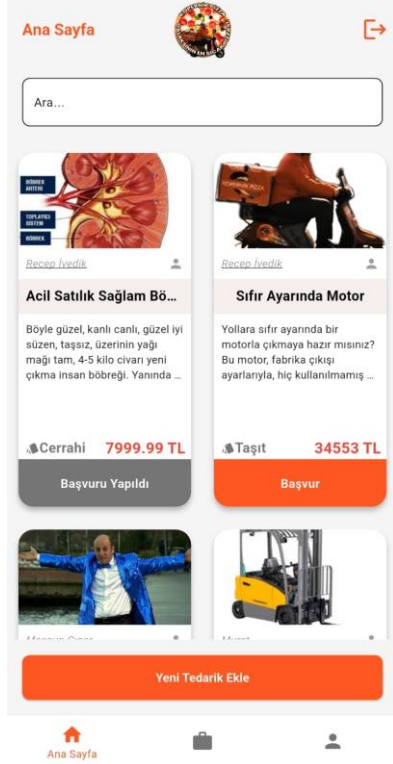
7. Yükleme Durumu

a. **CircularProgressIndicator:**

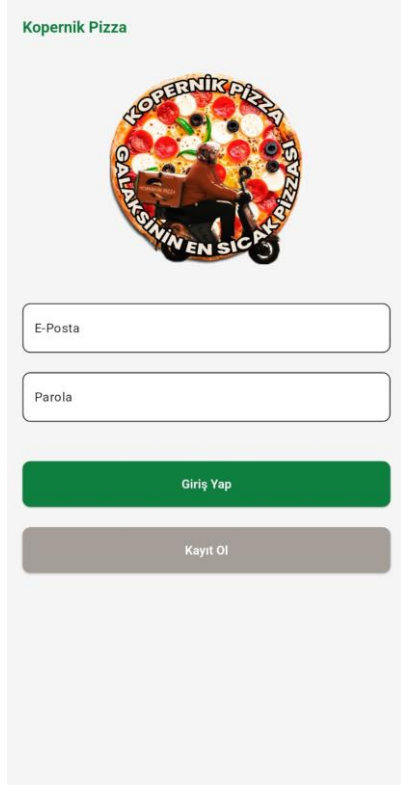
i. Kullanıcı verileri beklenirken yükleme göstergesi çıkar.

UYGULAMA GÖRÜNTÜLERİ

Ana Sayfa




Giriş Yapma Ekranı



Hesap Oluřturma Ekranı


[←](#) **Kopernik Pizza**




Kayıt Ol

Zaten hesabınız mı var? Giriř yapın

Başvurularım Sayfası

Başvurularım 




[Recep İvedik](#)

Acil Satılık Sağlam Böbrek

Böyle güzel, kanlı canlı, güzel iyi süzen, taşsız, üzerinin yağı mağı tam, 4-5 kilo civarı yeni çıkma insan böbreği. Yanında kan yapan dalak, kokoreçlik ince bağırsak promosyonu

Cerrahi 7999.99 TL

Başvuru Yapıldı






[Mecnun Çınar](#)

Ceket



Şıklığınızı tamamlayacak bu ceket, hem tarzınızı hem de konforunuzu ön planda tutuyor. Kaliteli kumaşı, modern tasarımı ve farklı renk seçenekleriyle her kıyafete uyum sağlar. İster resmi bir etkinlikte ister günlük kullanımda, bu ceketle her zaman öne çıkın. ...


Giyim 45 TL

  **Başvurular** 

Profil Sayfası

Profilim







Ad: Murat

E-posta: murat@gmail.com

Hesap Açılış Tarihi: 2024-12-24




Paylaştığınız Tedarikler




Murat

Çevre Dostu Elektrik...

Çevre dostu elektrikli forklift, depo ve lojistik sektörü için mükemmel bir çözüm. 2,5 ton taşıma kapasitesi, uzun...


 Taşıt **986555 TL**




Murat

Endüstriyel Su Arıtım...

Sanayi tesisleri ve fabrikalar için profesyonel su arıtma sistemi. Yüksek kapasiteli filtreleme, ters ozmoz tekn...


 Ekipman **186545 TL**





Murat

CNC Kesici Takım Seti

CNC makineleri için yüksek...








 Profilim

Profil Düzenleme Ekranı

Profilim






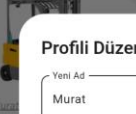
Ad: Murat

E-posta: murat@gmail.com

Hesap Açılış Tarihi: 2024-12-24




Paylaştığınız Tedarikler




Murat

Çevre Dostu Elektrik...

Çevre dostu elektrikli forklift, depo ve lojistik sektörü için mükemmel bir çözüm. 2,5 ton taşıma kapasitesi, uzun...


 Taşıt **986555 TL**




Murat

Endüstriyel Su Arıtım...

Sanayi tesisleri ve fabrikalar için profesyonel su arıtma sistemi. Yüksek kapasiteli filtreleme, ters ozmoz tekn...


 Ekipman **186545 TL**





Murat

CNC Kesici Takım Seti

CNC makineleri için yüksek...





 Profilim

Profil Düzenle

Yeni Ad

Murat

Fotoğraf Seç

İptal

Güncelle

Tedarik Sayfası



Çevre Dostu Elektrikli Forklift

986555 TL

Çevre dostu elektrikli forklift, depo ve lojistik sektörü için mükemmel bir çözüm. 2,5 ton taşıma kapasitesi, uzun pil ömrü ve sessiz çalışma özelliğiyle iç mekanlarda rahatlıkla kullanılabilir. Ergonomik tasarımı ve güvenlik özellikleriyle operatör konforunu ön planda tutar.

Taahhüt



Murat

Tedariki Düzenle / Sil

Başvuranlar



erennn

Tedarik Düzenleme Ekranı

Tedariki Düzenle

Başlık

Çevre Dostu Elektrikli Forklift

Açıklama

Çevre dostu elektrikli forklift, depo ve lojis

Fiyat

986555

Sektör

Taahhüt

Fotoğraf Seç

Vazgeç

Kaydet

Sil

Tedariki Düzenle / Sil

Başvuranlar

erennn

GİTHUB LİNKİ VE VİDEO

GitHub Linki:

<https://github.com/alierenzgenn/Kopernik>

Video:

<https://youtu.be/mYGxTR4yIDc?si=em2mj7OBgx7BnK-P>

