MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ
MOBİL PROGRAMLAMA DERSİ FİNAL PROJESİ

Proje Adı

Bilgi Yarışması Mobil Uygulaması

Hazırlayanlar

21120205054, Zeynep KICIKOĞLU

21120205025, Elif AKGÜL

Ders Sorumlusu

Dr. Öğr. Üyesi Muhammet Sinan BAŞARSLAN

Ocak, 2025

İstanbul Medeniyet Üniversitesi, İstanbul

İÇİNDEKİLER

No **Sayfa**

ÖZET

Bu proje, Firebase altyapısı ile desteklenen bir bilgi yarışması mobil uygulamasıdır. Uygulama, kullanıcılara dinamik bir şekilde veri yönetimi sağlayarak, Firestore veritabanından çekilen sorularla quiz yapma imkanı sunar. Kullanıcılar doğru ve yanlış cevaplarını anlık olarak görebilir ve test sonunda sonuçlarını öğrenebilir. Flutter framework'ü kullanılarak geliştirilen uygulama, kullanıcı dostu bir arayüzle eğlenceli ve etkileşimli bir deneyim sunar. Ayrıca, uygulama esnek ve ölçeklenebilir bir yapıya sahiptir, bu da yeni soruların kolayca eklenebilmesine olanak tanır.

Anahtar Kelimeler: Flutter, Firebase, Firestore, Mobil Uygulama Geliştirme, Dinamik Veri Yönetimi

SUMMARY

This project is a mobile quiz application powered by Firebase infrastructure. The application provides users with dynamic data management by enabling quizzes with questions fetched from the Firestore database. Users can see their correct and incorrect answers instantly and view their results at the end of the test. Developed with the Flutter framework, the application offers a user-friendly interface for an engaging and interactive experience. Additionally, the application features a flexible and scalable structure, allowing new questions to be added effortlessly.

Keywords: Flutter, Firebase, Firestore, Mobile App Development, Dynamic Data Management

1. GİRİŞ

Bu bölümde uygulamanın amacı, proje geliştirme sürecinde karşılaşılan kısıtlar ve literatür araştırmalarına yer verilmiştir.

1.1. Projenin Amacı

Projenin amacı, kullanıcıların bilgi birikimlerini artırabilecekleri ve eğlenceli bir şekilde öğrenme deneyimi yaşayabilecekleri bir **Quiz Uygulaması** geliştirmektir. Uygulama, Firebase ile entegre bir yapıya sahip olup, dinamik olarak soruların bir veritabanından çekilmesini sağlamaktadır. Temel hedef, kullanıcıların eğlenirken öğrenmelerini ve sonuçlarını anlık olarak görebilmelerini sağlamaktır.

Proje boyunca yapılan araştırmalarda, benzer uygulamalar arasında **Kahoot** ve **Quizlet** gibi örnekler incelenmiştir. Kullanıcı dostu arayüz, basit ve anlaşılır bir yapı öncelikli tasarım kriterleri arasında yer almıştır.

1.2. Proje Kısıtları

Kısıt 1: Firebase Kullanımı ve Yapılandırma Sorunları

Uygulama geliştirme sürecinin başlarında Firebase'in yapılandırılması sırasında çeşitli sorunlar yaşanmıştır. Özellikle Firestore'dan veri çekme işlemi sırasında yanlış yapılandırma hataları ile karşılaşılmıştır. Bu hataların çözümü için Firebase dökümantasyonu incelenmiş ve deneme-yanılma yöntemiyle sorun giderilmiştir. Bu süreç, projeye beklenenden daha fazla zaman harcanmasına neden olmuştur.

Çözüm: Firebase yapısının doğru şekilde entegre edilmesi sağlanmış ve fetchQuestionsFromFirestore() fonksiyonu ile Firestore'dan soruların başarılı bir şekilde çekilmesi mümkün hale getirilmiştir.

Kısıt 2: Flutter Widget'larının Dinamik Kullanımı

Proje geliştirme sürecinde kullanıcıların verdiği doğru ve yanlış cevapların dinamik olarak görselleştirilmesi için Wrap widget'ı kullanılmıştır. Ancak, doğru ve yanlış cevapların eş zamanlı olarak eklenmesi sırasında widget güncellemesiyle ilgili sorunlar yaşanmıştır.

Çözüm: Kullanıcı cevaplarına göre ikonların eklenmesi için setState() fonksiyonu doğru bir şekilde entegre edilmiş ve UI dinamik hale getirilmiştir.

Kısıt 3: Firestore Veritabanından Veri Çekme Süreci

Firestore veritabanındaki soruların doğru formatta uygulamaya çekilmesi sırasında veri modeliyle ilgili sorunlar yaşanmıştır. Özellikle, soruların eksik veya yanlış formatta gelmesi uygulamanın çalışmasını kesintiye uğratmıştır.

Çözüm: Veritabanı belgelerinin Soru sınıfı modeline uygun hale getirilmesi sağlanmış ve eksik veri kontrolü eklenmiştir. Bu sayede uygulama sorunsuz bir şekilde çalışmıştır.

```
List<Soru> sorular = snapshot.docs.map((doc) {
  return Soru(
    soruMetni: doc.data()['question'] ?? "Bilinmeyen Soru",
    soruYaniti: doc.data()['correctAnswer'] ?? false,
  );
}).toList();
```

Şekil.1 Soru Sınıfı Modeli Kodu

Kısıt 4: Kullanıcı Arayüzü Tasarımında Karşılaşılan Zorluklar

Uygulama tasarımında, özellikle küçük ekran cihazlarında metinlerin ve düğmelerin uygun şekilde yerleştirilmesi zorluk yaratmıştır. UI tasarımının kullanıcı dostu olabilmesi için birçok düzenleme yapılmıştır.

Çözüm: Flutter'ın Flexible ve Expanded widget'ları kullanılarak tasarım mobil cihazlara uyumlu hale getirilmiştir.

Kısıt 5: Doğru ve Yanlış Sayısı Hesaplamaları

Kullanıcıların verdiği doğru ve yanlış cevapların doğru şekilde hesaplanması sırasında hatalar meydana gelmiştir. Özellikle testin sonunda toplam doğru ve yanlış sayılarının gösterilmesi için ek bir işleme ihtiyaç duyulmuştur.

Çözüm: Doğru ve yanlış sayıları ayrı değişkenlerle (dogruSayisi ve yanlisSayisi) takip edilerek, AlertDialog içinde dinamik bir şekilde gösterilmesi sağlanmıştır.

```
AlertDialog alert = AlertDialog(
  title: const Text(
    "TEBRİKLER!",
    textAlign: TextAlign.center,
    style: TextStyle(
      fontSize: 30,
    ), // TextStyle
  ), // Text
  content: Text(
      "TESTİ BİTİRDİNİZ!\n\nDoğru Sayısı: $dogruSayisi\nYanlış Sayısı:
      $yanlisSayisi"), // Text
  actions: [
    okButton,
  ],
); // AlertDialog
```

Şekil.2 AlertDialog Kodu

Kısıt 6: Flutter ve Firebase Entegrasyonu İçin Bilgi Eksikliği

Proje geliştirilirken kullanılan Flutter ve Firebase araçları hakkında yeterli bilgiye sahip olunmaması öğrenme sürecini uzatmıştır. Özellikle `Firebase.initializeApp()` fonksiyonunun doğru şekilde çağrılmaması uygulamanın çalışmasını kesintiye uğratmıştır.

Çözüm: Firebase ve Flutter dökümantasyonları detaylı bir şekilde incelenmiş, ardından eksik bilgi tamamlanarak sorunlar giderilmiştir.

Tablo.1 Şekil ve Açıklamaları

Şekil Numarası	Açıklama
Şekil 1.1,	Firebase Firestore üzerinde oluşturulan "questions" koleksiyonu.
Şekil 1.2,	Uygulamanın kullanıcı arayüzü (Quiz ekranı).

2. LİTERATÜR TARAMASI

Uygulamanın olumlu yönleri:

- **Etkileşimli Kullanıcı Deneyimi:** Uygulama, kullanıcıların bilgi birikimini test edebilmesi için eğlenceli ve etkileşimli bir ortam sağlar. Kullanıcılar her soruya cevap verdikçe doğru veya yanlış sonuçları simgelerle anında görerek anlık geri bildirim alır.
- **Dinamik Soru Sistemi:** Firebase Firestore entegrasyonu sayesinde soruların dinamik olarak veritabanından çekilmesi sağlanmıştır. Bu özellik, soruların kolayca güncellenebilmesine ve çeşitlendirilebilmesine olanak tanır.
- **Başarı Takibi:** Uygulama sonunda doğru ve yanlış cevap sayılarının gösterilmesiyle kullanıcılar kendi başarı düzeylerini gözlemleyebilir. Bu, kullanıcıların eksiklerini belirlemelerine ve öğrenim süreçlerini iyileştirmelerine yardımcı olur.
- **Basit ve Anlaşılır Arayüz:** Kullanıcı dostu tasarımı sayesinde her yaştan kullanıcının kolayca erişebileceği bir yapı sunar. Arayüzde sade ve net bir tasarım dili benimsenmiştir.

Uygulamanın olumsuz yönleri:

- **Tekrarlanabilirlik Eksikliği:** Uygulamada aynı testin tekrar çözülmesi sırasında kullanıcıya farklı bir deneyim sunulmamaktadır. Soruların rastgele sıralanması gibi bir özellik eklenmesi bu sorunu çözebilir.
- **Çeşitli Oyunlaştırma Unsurlarının Eksikliği:** Kullanıcı motivasyonunu artıracak oyunlaştırma (gamification) unsurları bulunmamaktadır. Örneğin, başarı rozetleri veya liderlik tablosu gibi ek özelliklerle kullanıcılar daha uzun süre uygulamada kalabilir.
- **Kapsamlı Analiz Eksikliği:** Kullanıcıların hangi soru türlerinde zorlandığını veya hangi kategorilerde daha başarılı olduklarını analiz eden bir raporlama özelliği bulunmamaktadır. Bu özellik, uygulamanın öğrenme deneyimini geliştirmek için faydalı olabilir.

Tablo 2. Literatür Taraması

Uygulama Adı	Teknoloji	Özellikler
Kahoot	Bulut Tabanlı	Çoklu oyuncu desteği, skor takibi
Quizizz	Mobil Uygulama	Gerçek zamanlı testler, raporlama

3. MATERYAL VE YÖNTEM

- **Proje dili:** Flutter
  Flutter, Google tarafından geliştirilen açık kaynaklı bir UI yazılım geliştirme kitidir. Dart dilini kullanır ve hem Android hem de iOS platformları için uygulama geliştirme imkanı sunar. Bu projede, hızlı geliştirme süreci ve performansı sebebiyle Flutter tercih edilmiştir.

- **Proje geliştirme ortamı:** Visual Studio Code
  Visual Studio Code, hafif ve genişletilebilir bir kod düzenleme ortamıdır. Bu projede, çoklu platform desteği ve geniş eklenti desteği sayesinde geliştirme süreci bu ortamda gerçekleştirilmiştir.

- **Kullanılan veri tabanı:** Firebase Firestore
  Firebase Firestore, NoSQL bir veri tabanı sistemidir. Bu projede soruların depolanması ve kullanıcı verilerinin yönetimi için tercih edilmiştir.

- **Backend platformu:** Firebase
  Firebase Authentication ve Firestore Database modülleri projede kullanıcı yönetimi ve veri depolama için kullanılmıştır.

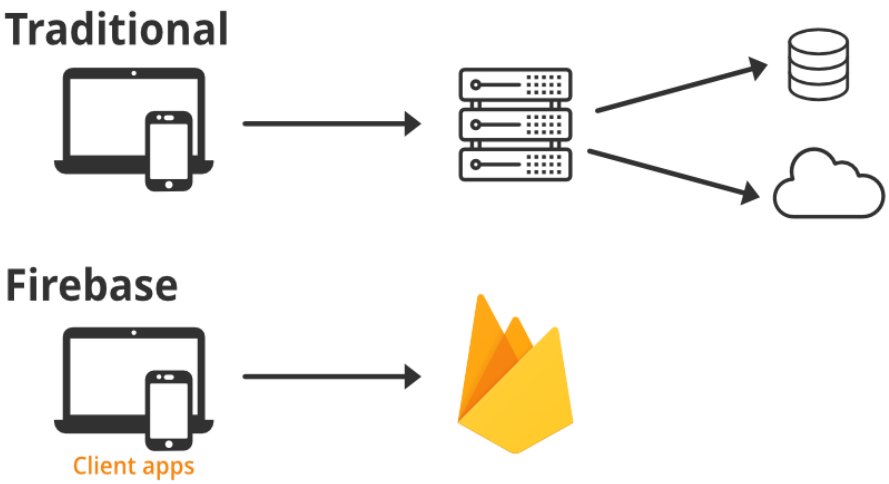
- **UI tasarımı:** Material Design
  Uygulama içerisinde kullanılan tüm UI bileşenleri Google'ın Material Design yönergelerine uygun olarak tasarlanmıştır. Bu, uygulamanın modern ve kullanıcı dostu bir görünüme sahip olmasını sağlamaktadır.

3.1 Flutter: Projede Kullanılan Geliştirme Kiti

- **Neden Flutter tercih edildi?**
  Flutter, tek bir kod tabanı üzerinden birden fazla platform için uygulama geliştirme imkanı sunar. Hızlı geri bildirim döngüsü, zengin widget desteği ve Dart dilinin sağladığı basitlik, bu projede Flutter'ın seçilmesinin temel sebepleridir.

3.2 Firebase: Backend Yönetiminde Kullanılan Platform

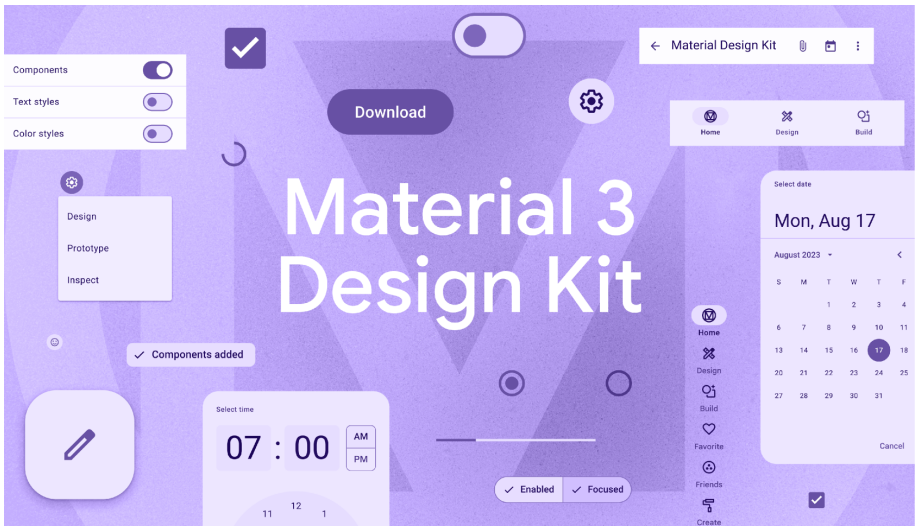
- **Firebase Authentication:**
  Firebase Authentication, kullanıcıların güvenli bir şekilde giriş yapmasını sağlar. Bu projede kullanıcı oturumlarını yönetmek için kullanılmıştır.
- **Firebase Firestore:**
  Firebase Firestore, dinamik ve gerçek zamanlı bir veri tabanı yönetimi sağlar. Sorular ve kullanıcıların yanıt verileri Firestore üzerinde depolanmıştır. NoSQL mimarisi, esnek veri depolama yapıları sunmaktadır.



Şekil.3 Firebase ve Geleneksel Sunucu Tabanlı Mimariler Arasındaki Karşılaştırma

3.3 Material Design: UI Tasarımı

- **Material Design neden tercih edildi?**
  Material Design, modern ve minimal bir kullanıcı arayüzü tasarımını standartlaştırır. Bu projede kullanılan butonlar, ikonlar ve diğer UI öğeleri Material Design kütüphanesinden alınarak uyarlanmıştır.



Şekil.4 Material Design ile UI Tasarımı

4. UYGULAMA

4.1. Firebase Kullanımı

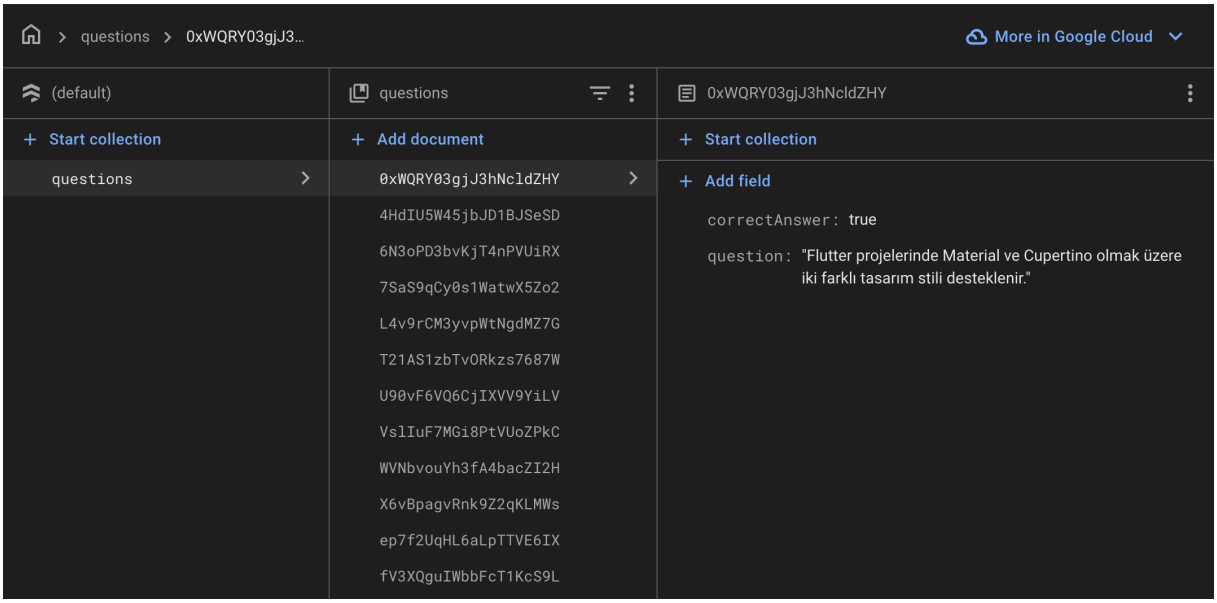
Uygulama, Firebase teknolojisini kullanarak veri çekme ve yükleme işlemlerini gerçekleştirir. Firebase Firestore, soruların saklanması ve düzenlenmesi için tercih edilmiştir. Kullanıcı, quiz sorularını Firebase Firestore'dan çekmekte ve doğru/yanlış cevaplara göre anlık olarak işlem yapmaktadır.

```dart
import 'soru.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

class TestVeri {
  int _soruIndex = 0;

  Future<List<Soru>> fetchQuestionsFromFirestore() async {
    print("Sorular Firestore'dan çekiliyor...");
    try {
      QuerySnapshot<Map<String, dynamic>> snapshot =
          await FirebaseFirestore.instance.collection('questions').get();

      List<Soru> sorular = snapshot.docs.map((doc) {
        return Soru(
          soruMetni: doc.data()['question'] ?? "Bilinmeyen Soru",
          soruYaniti: doc.data()['correctAnswer'] ?? false,
        );
      }).toList();

      print("Firestore'dan gelen sorular: $sorular");
      return sorular;
    } catch (e) {
      print("Hata: $e");
      return [];
    }
  }

  void testisifirla() {
    _soruIndex = 0;
  }
}

void uploadQuestionsToFirestore(List<Soru> soruBankasi) async {
  for (var soru in soruBankasi) {
    await FirebaseFirestore.instance.collection('questions').add({
      'question': soru.soruMetni,
      'correctAnswer': soru.soruYaniti,
    });
  }
  print('Sorular Firebase Firestore\'a yüklendi!');
}
```

Şekil.5 Firestore'dan Verileri Çekme Kodu

Kod Açıklaması:

Şekil.5'de görüldüğü üzere fetchQuestionsFromFirestore() metodu, Firestore'dan soruları çekmek ve listeye dönüştürmek için kullanılır. Sorular, Firestore'da questions koleksiyonunda saklanır. Her bir belge, question ve correctAnswer alanlarını içerir. Sorular başarılı bir şekilde çekildikten sonra kullanıcı arayüzünde gösterilir.



Şekil.6 Firebase'de Yer Alan Sorular

4.2. Uygulama Arayüzleri

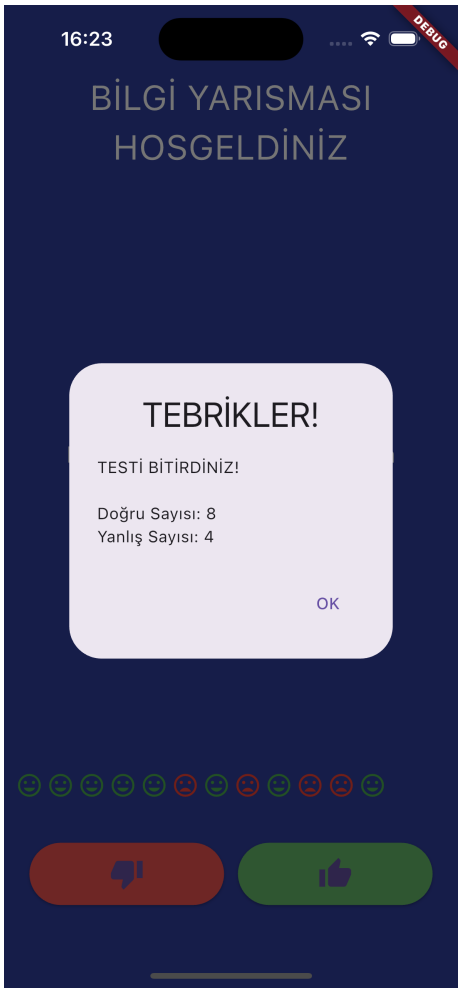
Uygulamanın arayüzleri, kullanıcıların kolaylıkla quiz sorularını cevaplayabilmesi için sade ve anlaşılır bir şekilde tasarlanmıştır.

Kullanıcıların bir soruyu okuyup doğru ya da yanlış seçeneklerinden birini seçebileceği ekran. Firebase'den gelen sorular burada dinamik olarak gösterilir. Soru metni `Text` widget'ı ile görüntülenir. Kullanıcı, iki butondan birine basarak cevabını belirtir.

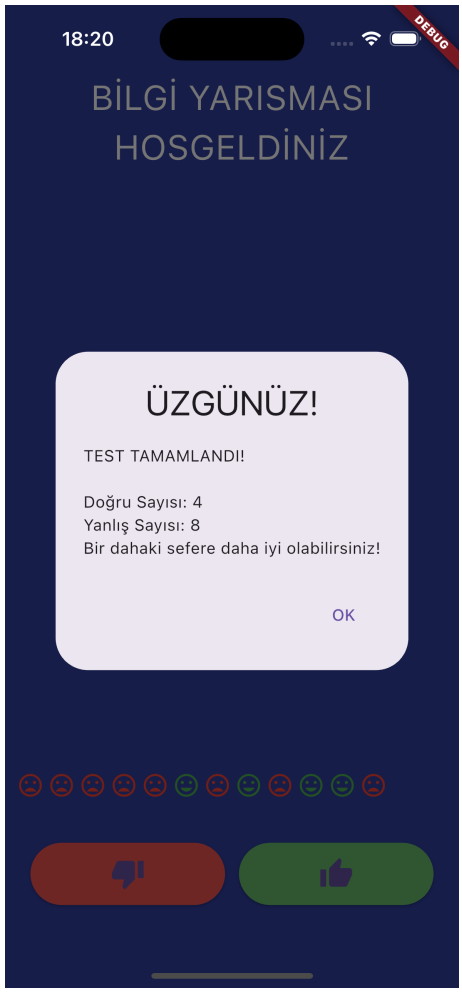
Şekil.7 Soru Gösterim Ekranı

Quiz tamamlandığında, kullanıcının doğru ve yanlış sayıları bir AlertDialog içinde gösterilir. Kullanıcı bu ekrandan geri dönerek tekrar oynayabilir. Doğru ve yanlış sayıları, bir AlertDialog widget'ında gösterilir. Kullanıcı "OK" butonuna tıklayarak teste tekrar başlayabilir.

Sonuç ekranları:



Şekil.8 Başarılı Sonuç Ekranı



Şekil.9 Başarısız Sonuç Ekranı

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

const Icon kDogruIconu = Icon(Icons.mood, color: Colors.green);
const Icon kYanlisIconu = Icon(Icons.mood_bad, color: Colors.red);
```

Şekil.10 Doğru ve Yanlış Cevaplar için constants.dart Tanımlamaları

Şekil.10 daki kod, uygulamada tekrar eden sabit değerleri bir araya toplamak için oluşturulmuş. Doğru ve yanlış cevaplar için görsel geri bildirimde kullanılan iki farklı ikon burada tanımlı.

İçerik Detayları:

- **kDogruIconu**: Bu ikon, doğru cevaplar için tasarlanmış. Yeşil renkte gülen bir yüz simgesi kullanılarak, kullanıcıya cevaplarının doğru olduğu belirtiliyor.
- **kYanlisIconu**: Yanlış cevaplar için ise kırmızı renkte üzgün bir yüz ikonu tanımlanmış. Bu ikon, kullanıcıya verdiği yanıtın yanlış olduğunu görsel olarak ifade ediyor.

Bu dosyanın amacı, tasarımda tutarlılığı sağlamak ve kod tekrarını önlemek.

```dart
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:yarisma/test_veri.dart';
import 'constants.dart';
import 'soru.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

Run | Debug | Profile
void main() async {
  WidgetsFlutterBinding.ensureInitialized();

  // Firebase'i başlatıyoruz
  await Firebase.initializeApp();

  runApp(const BilgiTesti());
}

class BilgiTesti extends StatelessWidget {
  const BilgiTesti({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            backgroundColor: Colors.indigo[700],
            body: SafeArea(
                child: Padding(
              padding: const EdgeInsets.symmetric(horizontal: 10.0),
              child: SoruSayfasi(),
            )))); // Padding // SafeArea // Scaffold // MaterialApp
  }
}

class SoruSayfasi extends StatefulWidget {
  const SoruSayfasi({super.key});

  @override
  _SoruSayfasiState createState() => _SoruSayfasiState();
}
```

Şekil.11 Main Dosyası

Şekil.11 de belirtilen kod, uygulamanın ana giriş noktasıdır ve uygulamanın temel yapı taşlarını barındırır. Kod, Flutter ile yazılmış bir quiz uygulamasının işlevselliğini sağlamak için Firebase ile entegre edilmiştir. Aşağıda dosyada yapılan her şey detaylı bir şekilde açıklanmıştır.

1. Firebase Entegrasyonu

Kodun başlangıcında Firebase ile entegrasyon sağlanmıştır:

```
// Firebase'i başlatıyoruz
await Firebase.initializeApp();
```

Şekil.12 Firebase Entegrasyonu

- Firebase'i kullanmadan önce Firebase.initializeApp() çağrısı yapılır.
- Bu işlem, uygulamanın Firebase projesine bağlanmasını sağlar.
- Firebase, quiz sorularını saklamak ve çekmek için Firestore adlı bulut tabanlı bir veritabanını kullanır.

```
class BilgiTesti extends StatelessWidget {
  const BilgiTesti({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        backgroundColor: Colors.indigo[700],
        body: SafeArea(
          child: Padding(
            padding: const EdgeInsets.symmetric(horizontal: 10.0),
            child: SoruSayfasi(),
          ))); // Padding // SafeArea // Scaffold // MaterialApp
  }
}
```

Şekil.13 BilgiTesti Sınıfının Kodu

Şekil.13 deki sınıf, uygulamanın ana widget'ıdır. Tüm uygulama bu sınıfın altında tanımlanmıştır:

MaterialApp:

- Flutter uygulamalarında temel yapı taşlarından biridir. Uygulamanın genel teması, yönlendirme yapısı ve başlangıç ekranı burada tanımlanır.
- home parametresi ile uygulamanın ana ekranı belirlenmiştir (SoruSayfasi).

Scaffold:

- Uygulamanın temel iskelet yapısını sağlar.
- Arka plan rengi **indigo[700]** olarak ayarlanmıştır.

SafeArea:

- Cihazın çentik ve kenar boşlukları gibi alanlarından kaçınmak için kullanılmıştır. Böylece içerik düzgün bir şekilde ekranda gösterilir.

Padding:

- İçeriğin ekrandaki kenar boşluklarını ayarlamak için kullanılır.

```
class SoruSayfasi extends StatefulWidget {
  const SoruSayfasi({super.key});

  @override
  _SoruSayfasiState createState() => _SoruSayfasiState();
}
```

Şekil.14 SoruSayfası Sınıfının Kodu

Şekil.14 deki sınıf, quiz uygulamasının asıl mantığını yönetir. Kullanıcıların soruları görmesi, yanıtlaması ve sonuçları öğrenmesi bu sınıfta gerçekleşir.

A. Durum Yönetimi (StatefulWidget)

StatefulWidget kullanılarak, kullanıcıdan gelen girişlere (örneğin doğru veya yanlış cevaba tıklanması) göre ekran güncellenir.

B. Veri Çekme (initState Metodu)

```dart
@override
void initState() {
  super.initState();
  print("Sorular Firestore'dan çekiliyor...");
  testVeri.fetchQuestionsFromFirestore().then((data) {
    setState(() {
      soruBankasi = data;
    });
    print("Firestore'dan gelen sorular: $soruBankasi");
  }).catchError((e) {
    print("Firestore'dan veri çekerken hata oluştu: $e");
  });
}
```

Şekil.15 initState Metodu Kodu

Bu metod, uygulama çalıştığında Firestore'dan soruların çekilmesini sağlar. Sorular çekildikten sonra soruBankasi listesine atanır ve ekranda gösterilmeye hazır hale gelir.

C. Quiz Mantığı

1. Cevaplama Fonksiyonu:

```dart
void butonFonksiyonu(bool secilenbuton) {
  if (_soruIndex + 1 >= soruBankasi.length) {
    Widget okButton = TextButton(
      child: const Text("OK"),
      onPressed: () {
        Navigator.of(context).pop();
        setState(() {
          _soruIndex = 0;
          secimler = [];
          dogruSayisi = 0;
          yanlisSayisi = 0;
        });
      },
    ); // TextButton

    String baslik = dogruSayisi >= yanlisSayisi ? "TEBRİKLER!" : "ÜZGÜNÜZ!";
    String mesaj = dogruSayisi >= yanlisSayisi
        ? "TESTİ BİTİRDİNİZ!\n\nDoğru Sayısı: $dogruSayisi\nYanlış Sayısı: $yanlisSayisi"
        : "TEST TAMAMLANDI!\n\nDoğru Sayısı: $dogruSayisi\nYanlış Sayısı: $yanlisSayisi\nBir dahaki sefere daha iyi
        olabilirsiniz!";
```

Şekil.16 Quiz Mantığında Cevaplama İşlemi

Kullanıcının doğru ya da yanlış butonuna basmasını işler. Eğer doğru cevap verilmişse, dogruSayisi bir artırılır. Yanlış cevapta ise yanlisSayisi bir artırılır. Kullanıcının verdiği cevaba uygun görsel bir ikon (kDogruIconu ya da kYanlisIconu) eklenir.

Tüm sorular cevaplandığında, bir AlertDialog ile doğru ve yanlış sayıları kullanıcıya gösterilir. Eğer doğru sayısı, yanlış sayısından fazla ya da eşitse, "TEBRİKLER!" başlığı gösterilir. Aksi halde, "ÜZGÜNÜZ!" mesajı ile kullanıcı bilgilendirilir.

```
@override
Widget build(BuildContext context) {
  return Column(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: <Widget>[
      const Text(
        'BİLGİ YARISMASI\n'
        'HOSGELDİNİZ',
        textAlign: TextAlign.center,
        style: TextStyle(
          fontSize: 30,
          color: Colors.white,
        ), // TextStyle
      ), // Text
      Expanded(
        flex: 4,
        child: Padding(
          padding: const EdgeInsets.all(10.0),
          child: Center(
            child: Text(
              soruBankasi.isEmpty
                  ? "Sorular Yükleniyor..."
                  : soruBankasi[_soruIndex].soruMetni,
              textAlign: TextAlign.center,
              style: const TextStyle(
                fontSize: 20.0,
                color: Colors.white,
              ), // TextStyle
            ), // Text
          ), // Center
        ), // Padding
      ), // Expanded
      Wrap(spacing: 3, runSpacing: 3, children: secimler),
      Expanded(
        flex: 1,
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 6.0),
          child: Row(children: <Widget>[
            Expanded(
              child: Padding(
                padding: const EdgeInsets.symmetric(horizontal: 6),
                child: ElevatedButton(
                  style: ElevatedButton.styleFrom(
                    backgroundColor: Colors.red[400],
                    foregroundColor: Colors.white,
                  ),
                  onPressed: soruBankasi.isEmpty
                      ? null
                      : () {
                          butonFonksiyonu(false);
                        },
                  child: const Padding(
                    padding: EdgeInsets.all(12.0),
                    child: Icon(Icons.thumb_down, size: 30.0),
                  ), // Padding
                ))), // ElevatedButton // Padding // Expanded
```

Şekil.17 Widget Yapısı ile Quiz Ekranı Tasarımı

Şekil.17 deki kod, Flutter kullanılarak bir quiz arayüzü oluşturmak için tasarlanmış bir build metodudur. Ekranın düzeni, tüm bileşenleri kapsayan bir Column widget'ı ile oluşturulmuştur. Bu widget, içerikleri dikey eksende düzenler ve bileşenler arasına boşluk eklerken, yatay eksende uzatır. Ekranın üst kısmında, kullanıcıya "BİLGİ YARIŞMASI\nHOŞGELDİNİZ" metni büyük yazı stili ve beyaz renk kullanılarak gösterilir. Orta kısımda, soru metni bir Padding içinde hizalanarak merkezde gösterilir. Eğer soru bankası boşsa, kullanıcıya "Sorular Yükleniyor..." mesajı sunulur; aksi takdirde, mevcut soru metni ekrana yazdırılır.

Cevap seçenekleri, esnek bir yapı sunan Wrap widget'ı ile düzenlenir. Bu yapı, seçeneklerin ekrana sığmasını kolaylaştırır ve düzenli bir görünüm sağlar. Ekranın alt kısmında ise bir Row widget'ı içinde kullanıcıya cevap verme işlevi sunan butonlar yer alır. Örneğin, kırmızı bir buton yanlış cevaplar için tasarlanmıştır ve butonun tıklanmasıyla buttonFonksiyonu(false) çağrılır. Butonlar, etraflarına eklenen Padding ile daha düzenli bir görünüm kazanır ve ikonlar ile zenginleştirilir.

Bu kod, dinamik yapısıyla soru bankasının dolu ya da boş olmasına göre farklı içerikler sunar ve ekranın her boyuta uyum sağlamasını hedefler. Kullanıcıya görsel olarak düzenli, işlevsel ve kolay anlaşılır bir deneyim sunan bu yapı, Flutter ile bir quiz ekranı tasarlamanın temel örneklerinden biridir.

```dart
class Soru {
  String soruMetni;
  bool soruYaniti;

  Soru({required this.soruMetni, required this.soruYaniti});
}
```

Şekil.18 Soru Verisinin Modelleme Yapısı (soru.dart)

Şekil.18 deki dosya, uygulamada bir sorunun ne şekilde temsil edileceğini tanımlıyor. Soruların veri modellemesi burada yapılıyor.

İçerik Detayları:

- **Soru Sınıfı**:
  - Sorunun metni ve doğru/yanlış yanıt bilgisi olmak üzere iki özelliği var:
    - soruMetni: Sorunun açıklaması (örneğin, "Flutter'ın temel yapı taşı nedir?").
    - soruYaniti: Sorunun doğru cevabı (örneğin, true ya da false).
  - Bu sınıf sayesinde uygulama, her soruyu tek bir nesne olarak ele alabiliyor.

```dart
import 'soru.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

class TestVeri {
  int _soruIndex = 0;

  Future<List<Soru>> fetchQuestionsFromFirestore() async {
    print("Sorular Firestore'dan çekiliyor...");
    try {
      QuerySnapshot<Map<String, dynamic>> snapshot =
          await FirebaseFirestore.instance.collection('questions').get();

      List<Soru> sorular = snapshot.docs.map((doc) {
        return Soru(
          soruMetni: doc.data()['question'] ?? "Bilinmeyen Soru",
          soruYaniti: doc.data()['correctAnswer'] ?? false,
        );
      }).toList();

      print("Firestore'dan gelen sorular: $sorular");
      return sorular;
    } catch (e) {
      print("Hata: $e");
      return [];
    }
  }

  void testisifirla() {
    _soruIndex = 0;
  }
}

void uploadQuestionsToFirestore(List<Soru> soruBankasi) async {
  for (var soru in soruBankasi) {
    await FirebaseFirestore.instance.collection('questions').add({
      'question': soru.soruMetni,
      'correctAnswer': soru.soruYaniti,
    });
  }
  print('Sorular Firebase Firestore\'a yüklendi!');
}
```

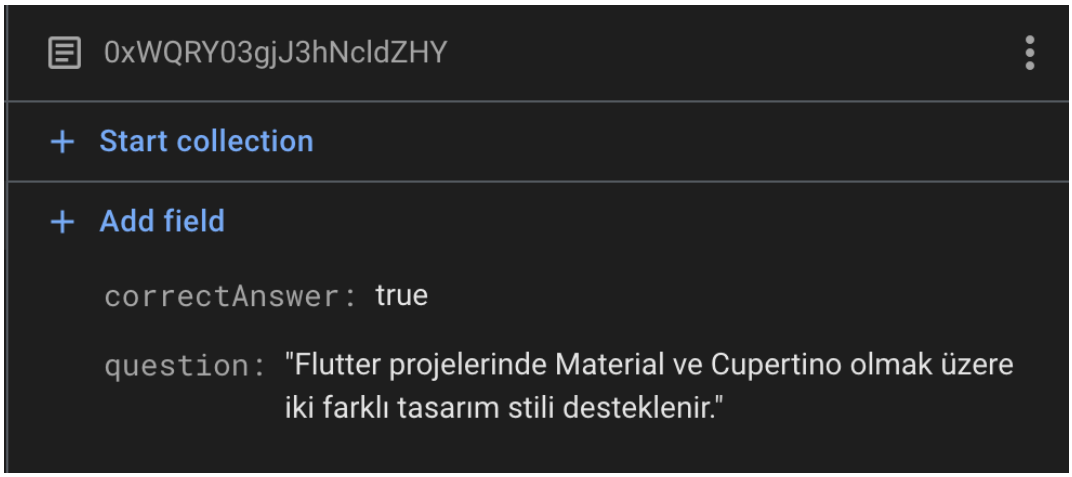
Şekil.19 Firebase'den Veri Çekme ve Soru Ekleme Kodları

Şekil.19 daki dosya uygulamanın Firebase'den veri çekme ve veritabanına soru ekleme işlevlerini yerine getiriyor. Firebase Firestore kullanılarak soruların saklandığı bir koleksiyon üzerinde çalışılıyor.

İçerik Detayları:

- **fetchQuestionsFromFirestore**: Bu metot, Firestore'daki questions koleksiyonundan soruları çekiyor. Çekilen her bir belge, Soru sınıfından bir nesneye dönüştürülüyor. Yani, Firestore'daki sorular, uygulama tarafından kullanılabilir hale getiriliyor.

- **uploadQuestionsToFirestore**: Eğer Firebase'e yeni sorular yüklemek istiyorsanız, bu metot devreye giriyor. Mevcut soruları questions koleksiyonuna ekleyerek veritabanını güncelliyor.



Şekil.20 Firebase Firestore'da Quiz Sorularının Saklanması

Uygulamada Firebase Firestore, quiz sorularını dinamik bir şekilde saklamak ve çekmek için kullanılmıştır. Firestore'da "questions" adlı bir koleksiyon oluşturulmuş ve her soru, benzersiz bir belge kimliği ile saklanmaktadır. Belgeler, sorunun metni (question) ve doğru yanıtını (correctAnswer) içermektedir. Sorular Firestore'dan çekilerek uygulamada dinamik olarak görüntülenmekte ve kullanıcı arayüzünde kullanılmaktadır. Veri çekme işlemi fetchQuestionsFromFirestore fonksiyonu ile gerçekleştirilmiş, çekilen veriler Soru nesnelerine dönüştürülmüştür. Ayrıca, yeni soruların eklenmesi uploadQuestionsToFirestore fonksiyonu ile sağlanmıştır. Firestore kullanımı sayesinde sorular kolayca güncellenebilir ve bu değişiklikler uygulamaya anında yansımaktadır. Bu yapı, uygulamanın esnekliğini ve dinamizmini artırmaktadır.

5. SONUÇ VE TARTIŞMA

Bu çalışmada, bir bilgi yarışması uygulaması geliştirilerek Firebase Firestore ile dinamik veri yönetimi sağlandı. Uygulamanın temel amacı, kullanıcıların eğlenerek öğrenmesini sağlamak ve geliştiricilere Firebase ile uygulama geliştirme süreçlerine dair bir örnek sunmaktı.

Sonuçlar:

1. **Dinamik Veri Yönetimi:** Uygulama, Firebase Firestore kullanılarak dinamik bir veri yapısı ile oluşturuldu. Sorular, Firestore'dan çekildiği için kolayca güncellenebilir ve uygulama yeniden dağıtılmadan yeni içerikler eklenebilir hale geldi.

2. **Kullanıcı Deneyimi:** Sade ve işlevsel bir arayüz tasarımı ile kullanıcıların uygulamayı kolayca kullanabilmesi sağlandı. Kullanıcılar doğru ve yanlış cevaplarını anlık olarak görebildi, bu da kullanıcı deneyimini olumlu yönde etkiledi.

3. **Geri Bildirim Mekanizması:** Doğru ve yanlış cevaplar için görsel ikonlar kullanılarak kullanıcıların öğrenme sürecine katkı sağlandı. Quiz tamamlandığında kullanıcıya doğru ve yanlış sayılarının gösterilmesi, kullanıcıya somut bir değerlendirme sundu.

Tartışma:

- **Başarılar:** Uygulamanın Firebase ile entegrasyonu başarılı bir şekilde gerçekleştirildi. Bu, gerçek zamanlı veri yönetimi ve ölçeklenebilir bir yapı sundu. Ayrıca, Flutter ile kullanıcı dostu bir arayüz tasarımı yapıldı.

- **Zorluklar:** Firestore'dan veri çekme sürecinde internet bağlantısına bağımlılık gibi bir sorun gözlemlendi. Bağlantı olmadığında veya yavaş olduğunda uygulama soruların yüklenmesini beklemek zorunda kalıyor. Bu durum, önbellekleme mekanizmalarıyla iyileştirilebilir.

- **Gelecek Çalışmalar:** Uygulamanın geliştirilebilir birçok yönü bulunuyor. Örneğin:

    - **Yeni Özellikler:** Daha fazla soru türü (çoktan seçmeli, açık uçlu vb.) ve kullanıcıların başarı seviyelerini takip edebilecek bir profil sistemi eklenebilir.
    - **Analitik:** Firebase Analytics kullanılarak kullanıcı davranışları analiz edilebilir ve uygulama daha kullanıcı odaklı bir şekilde optimize edilebilir.

Bu proje, Firebase ile Flutter'ın entegrasyonunun ne kadar güçlü olduğunu göstermektedir. Dinamik verilerle çalışan bir bilgi yarışması uygulaması başarıyla geliştirilmiştir. Gelecekte, çok oyunculu mod veya kullanıcı bazlı skor takibi gibi özellikler eklenebilir.

6. KAYNAKÇA

1. **"Firebase Documentation,"** [Online]. Available: https://firebase.google.com/.
   Firebase'in entegrasyonu ve Firestore'un kullanımı hakkında bilgi alınmıştır.

2. **"Flutter Documentation,"** [Online]. Available: https://flutter.dev/docs.
   Flutter ile uygulama geliştirme sürecinde temel bilgiler ve widget kullanımı öğrenilmiştir.

3. **"Dart Programming Language,"** [Online]. Available: https://dart.dev.
   Dart programlama dili ile ilgili bilgiler bu kaynak üzerinden edinilmiştir.

4. **"BTK Akademi Flutter Kursu,"** [Online]. Available: https://btkakademi.gov.tr/.
   Flutter hakkında temel ve ileri düzey bilgiler bu kurs aracılığıyla öğrenilmiştir.

5. **"Material Design,"** [Online]. Available: https://m3.material.io/
   Flutter'daki tasarım prensipleri ve Material Design'ın entegrasyonu için rehber.

6. **"Google Fonts,"** [Online]. Available: https://fonts.google.com/
   Uygulamada kullanılan yazı tipleriyle ilgili bilgi için önemli bir kaynak.

7. **"Color Hunt,"** [Online]. Available: https://colorhunt.co/
   Arayüz tasarımında kullanılan renk paletleri için bir kaynak.