ANKARA ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

(BLM4538) IOS ile Mobil Uygulama Geliştirme II

Mertcan Özdemir
GitHub: mertcan-ozdemir
21290194

Github repo link:

https://github.com/mertcan-ozdemir/BLM4538-21290194

Video linki:

https://www.youtube.com/shorts/o8yMgc4lLDc

MOVIEAPP

Özet

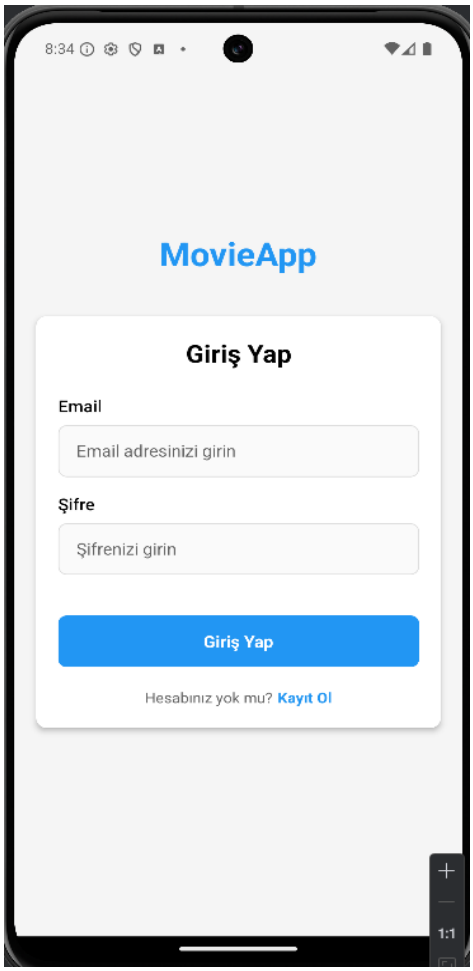
Bu çalışma, kullanıcıların filmleri inceleyip puanlayabileceği, sevdikleri filmleri kaydedip kişisel listeler oluşturabileceği bir film değerlendirme platformudur. Çalışma, film keşfetme, film bilgilerinin görüntülenmesi, yorum yapma ve liste oluşturma gibi işlevler sunmaktadır. Veriler, TMDB API aracılığıyla temin edilmekte ve kullanıcı aktiviteleri bir veri tabanı üzerinden yönetilmektedir. Projeye ait genel işlevler:

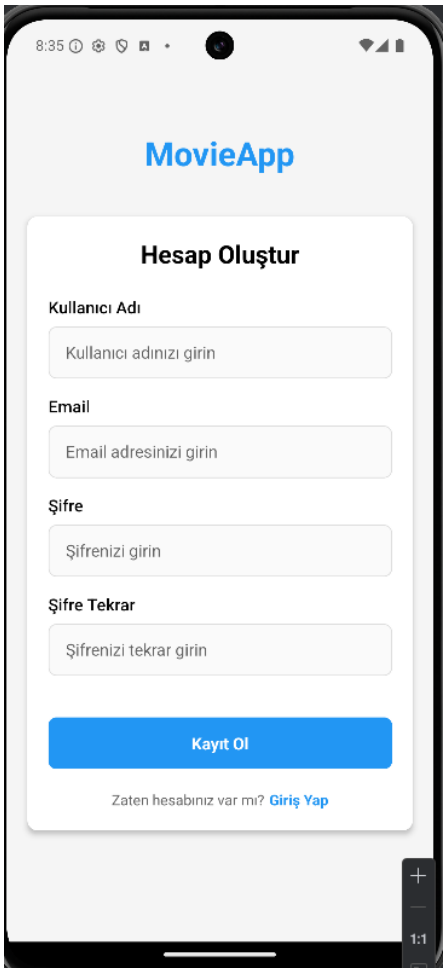
- Kaydolma, giriş yapma, beğenilen veya kaydedilen filmleri yönetme
- Filmler için değerlendirme, yorum yapma ve liste ekleme özellikleri.
- TMDB API entegrasyonu ile güncel film bilgileri.
- Popüler, trend veya en çok puan alan filmleri görüntüleyebilme

Uygulamanın Ara yüzleri

1. Giriş ve Kayıt Olma Sayfaları

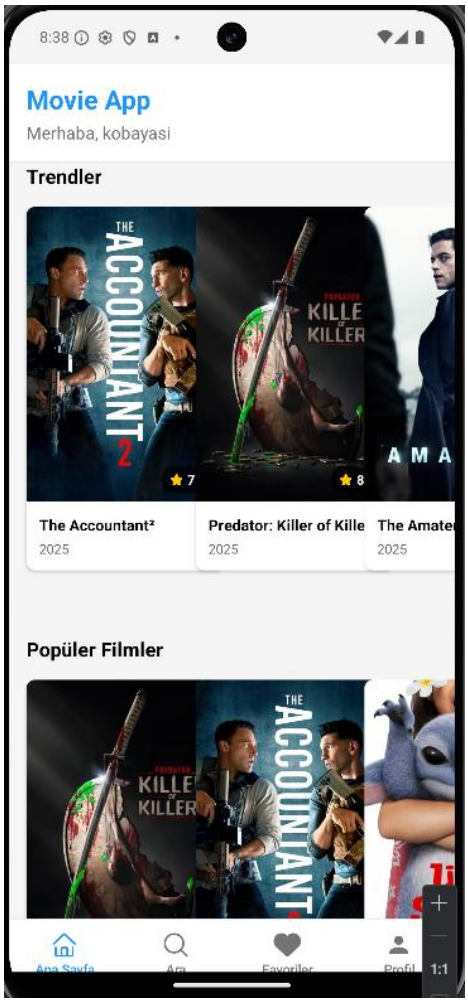
Uygulamayı açılınca kullanıcıların ilk karşılaştığı ekran giriş ekranıdır. Burada eğer sisteme kayıtlıysa giriş yapabilir eğer bir kaydı yoksa kayıt butonuyla sisteme kayıtlarını gerçekleştirebilirler.





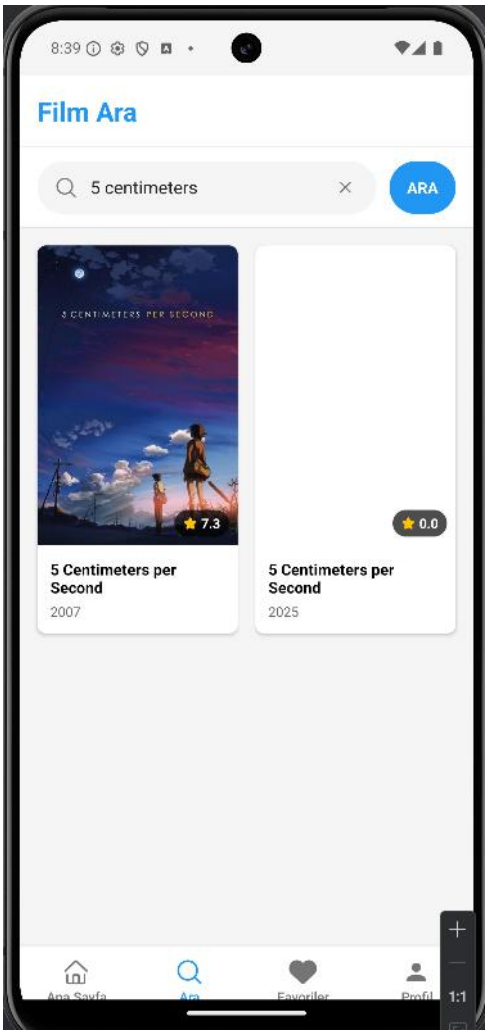
2. Ana Sayfa

Ana sayfa, kullanıcıların başarılı bir şekilde giriş yaptıktan sonra karşılaştıkları ana ekrandır. Bu sayfada kullanıcıların ilgisini çekebilecek popüler filmler, trend filmler ve genel olarak en yüksek puan alan filmler listelenir. Sayfadaki herhangi bir filme tıklanarak o filmin detaylarına erişilebilir. Alt kısımda bulunan navigasyon çubuğu kullanıcıların uygulama deneyimlerini kolaylaştırmaktadır.



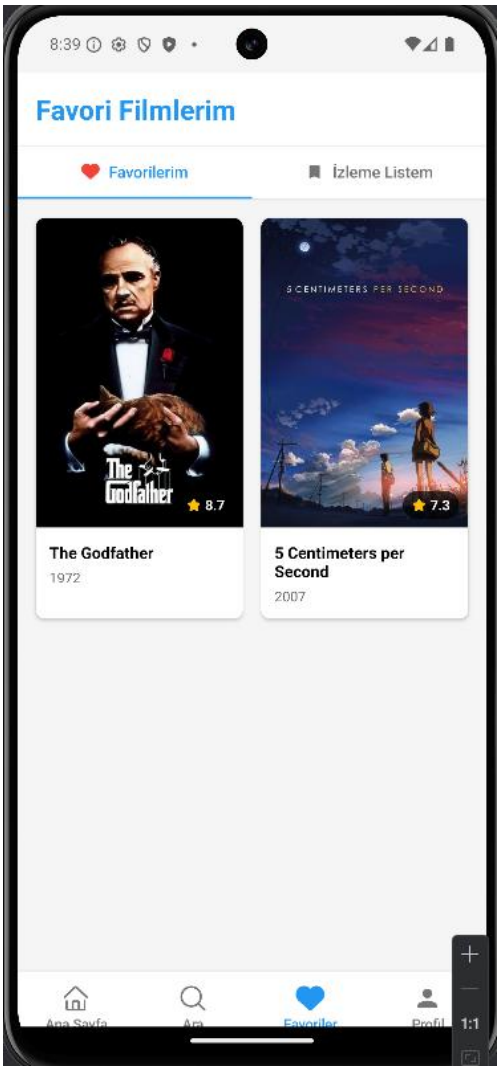
3. Arama Sayfası

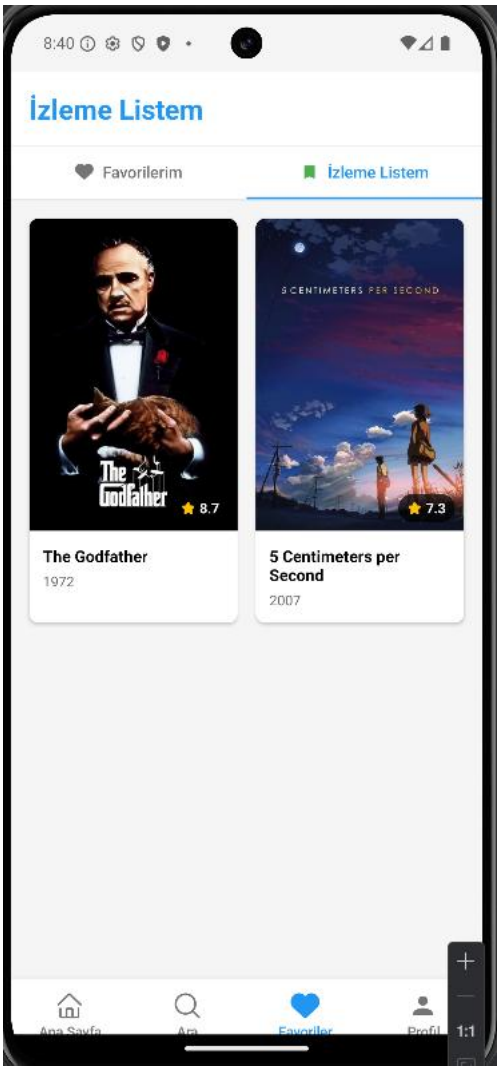
Bu sayfada ise kullanıcılar erişmek istediği filmin ismini yazarak aratabilirler. Bu sayfaya erişmek için aşağıdaki navigasyon çubuğundan “Ara” butonuna basarak erişebilirler



4. Favori Filmler ve İzlenecek Filmler Sayfası

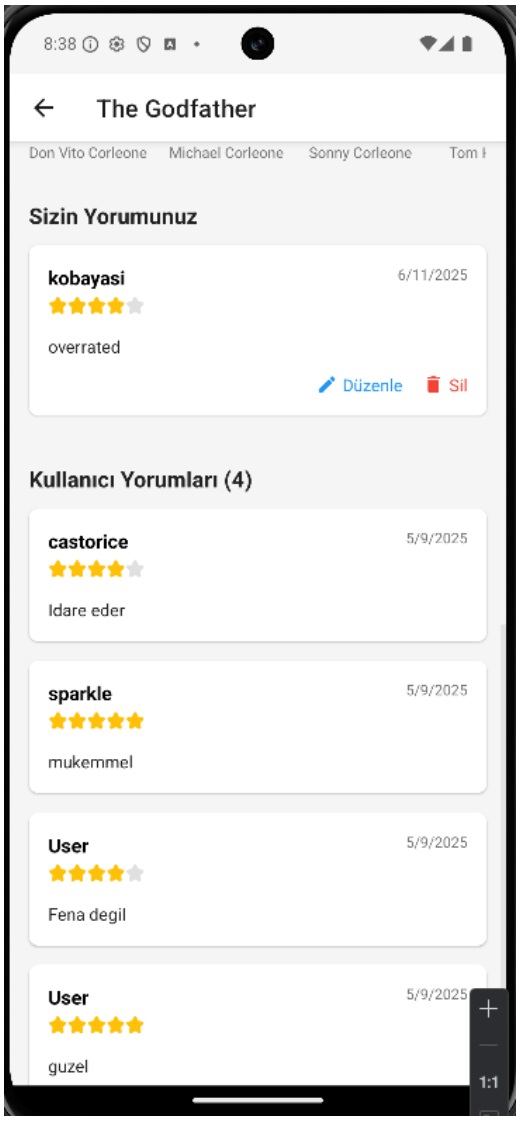
Bu sayfalarda ise kullanıcılar beğendikleri filmleri veya daha sonra izlemek için kaydettikleri filmleri görüntüleyebilmektedir.





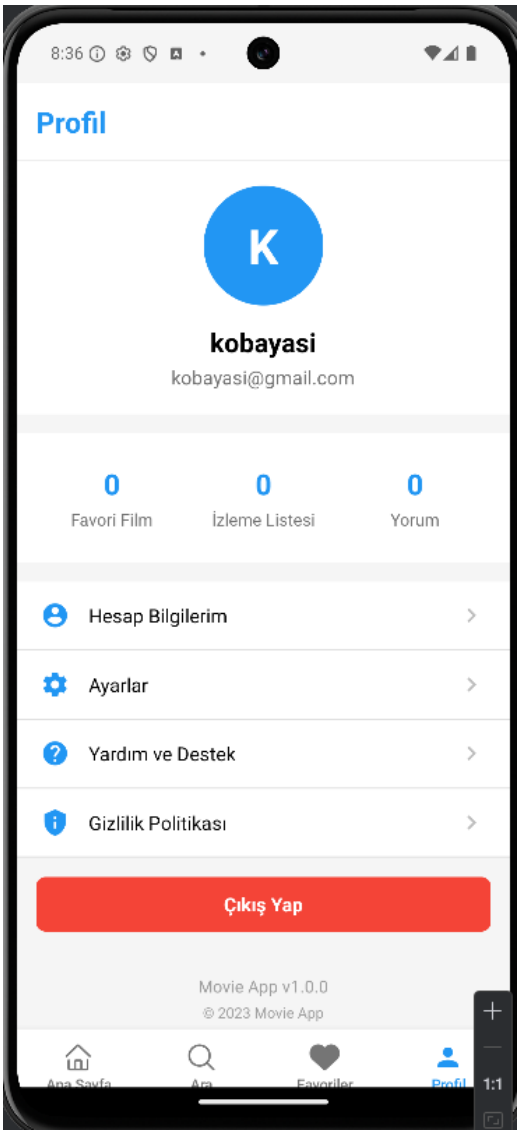
5. Film Detayları Sayfası

Bu sayfaya herhangi bir filmin kartına basıldıktan sonra erişilebilmektedir. Kullanıcılar bu ekran da film ile ilgili bilgilere erişebilmektedir. Bu bilgiler arasında filmin kartı, puanı, yayın tarihi, uzunluğu, türleri, özet ve filmde oynayan oyuncu bilgilerini görüntüleyebilmektedir. Favori butonuna basarak filmi favorilere ekleyebilir, izleme listesine basarak aynı şekilde izleme listesine ekleyebilmektedir. Değerlendir butonuna basarak filme puan verebilir ve inceleme yazabilmektedir. Sayfanın alt kısmında bulunan yorumlar bölümünde ise filme yapılan yorumları görüntüleyebilmektedir.



6. Profil Sayfası

Bu sayfada ise kullanıcılar favori, izleme listesindeki film ve yorum sayılarını görebilmektedir. Henüz diğer butonlar çalışmamakta fakat ilerideki güncellemelerde eklenecektir. Çıkış yap butonuna basarak ise oturumlarını kapatabilmektedirler.



Veri Tabanı Tarafı

Bu proje kapsamında, kullanıcıya özgü veri işlemlerinin ve film bilgisi temelli öneri sisteminin yönetilebilmesi için çeşitli veri kaynakları entegre edilmiştir. Uygulamada kullanılan temel veri kaynakları ve amaçları aşağıda özetlenmiştir:

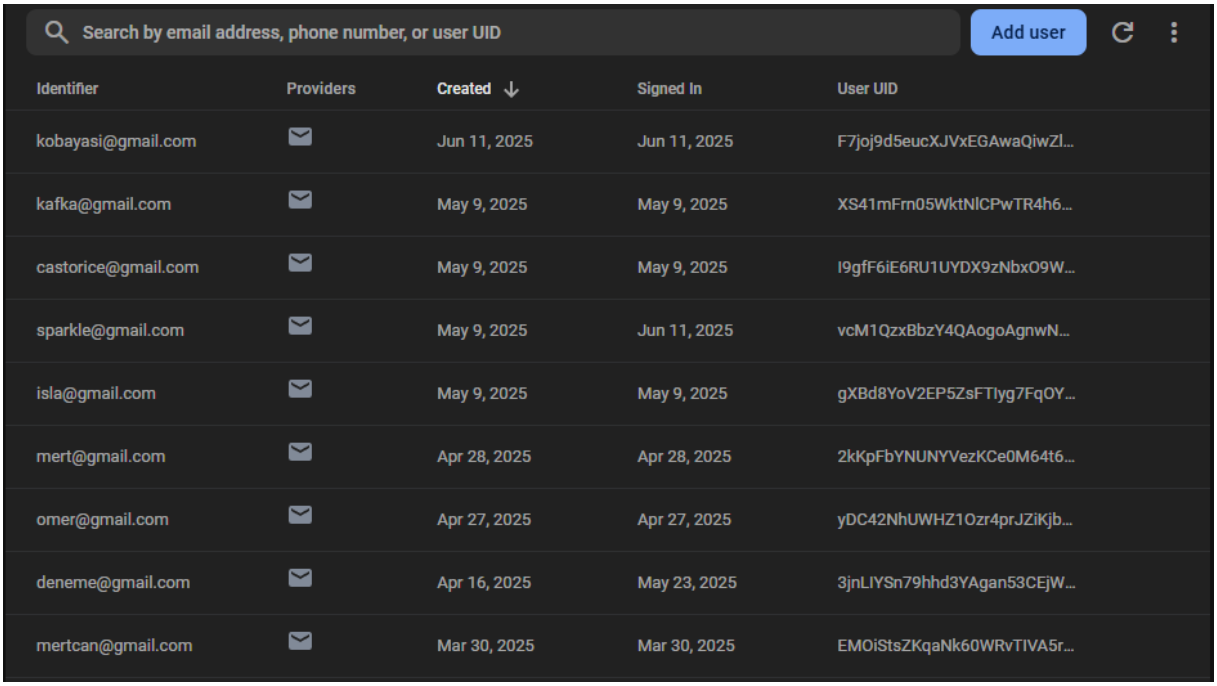
- **Firebase Realtime Database**:
  Kullanıcıya ait profil verileri, favori filmler listesi, kullanıcı kayıt ve giriş işlemleri için gerekli olan e-posta ve şifre bilgileri Firebase üzerinde güvenli şekilde saklanmaktadır. Firebase, gerçek zamanlı veri senkronizasyonu sayesinde kullanıcı etkileşimlerini anlık olarak yönetmeye olanak sağlar.
- **TMDB (The Movie Database) API**:
  Film içeriklerine dair detaylı bilgiye erişim sağlamak amacıyla TMDB API kullanılmaktadır. API anahtarı (API key) aracılığıyla, uygulamada

gösterilecek filmlerin açıklamaları, afişleri ve diğer metadata bilgileri gerçek zamanlı olarak TMDB üzerinden çekilmektedir.

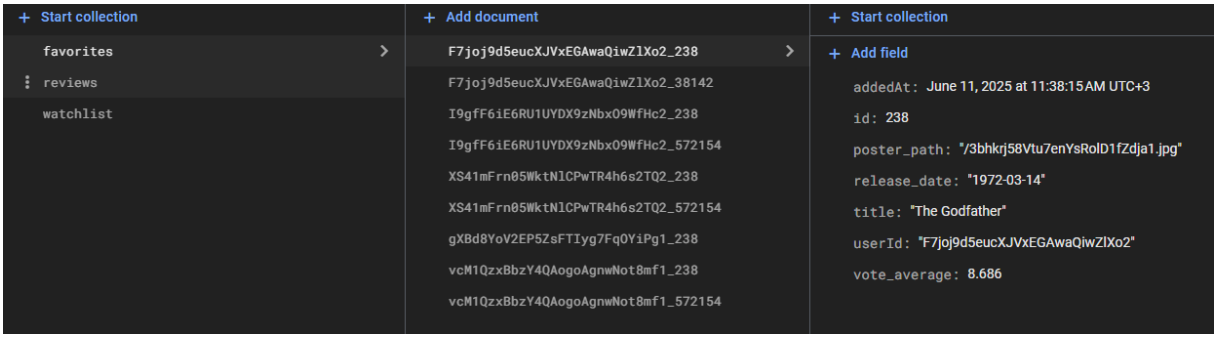
- **Firebase Authentication**:

Kullanıcıların güvenli bir şekilde kayıt olabilmesi ve sisteme giriş yapabilmesi için Firebase Authentication servisi kullanılmaktadır. Bu servis aracılığıyla e-posta ve şifre tabanlı kimlik doğrulama işlemleri gerçekleştirilir. Firebase Auth, parola yönetimi, oturum kontrolü ve doğrulama işlemlerini güvenli ve hızlı bir şekilde sunarak kullanıcı yönetimini kolaylaştırır.

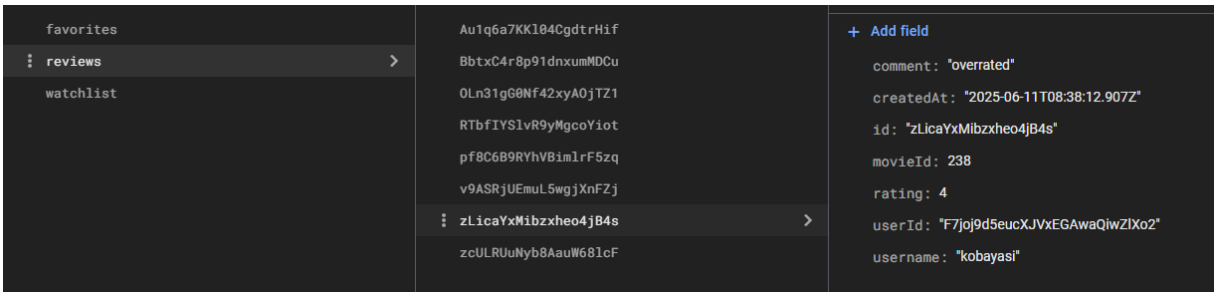
Bu yapı sayesinde, hem kullanıcıya özel içerik yönetimi sağlanmakta hem de dinamik ve güncel film verisi sunulmaktadır.



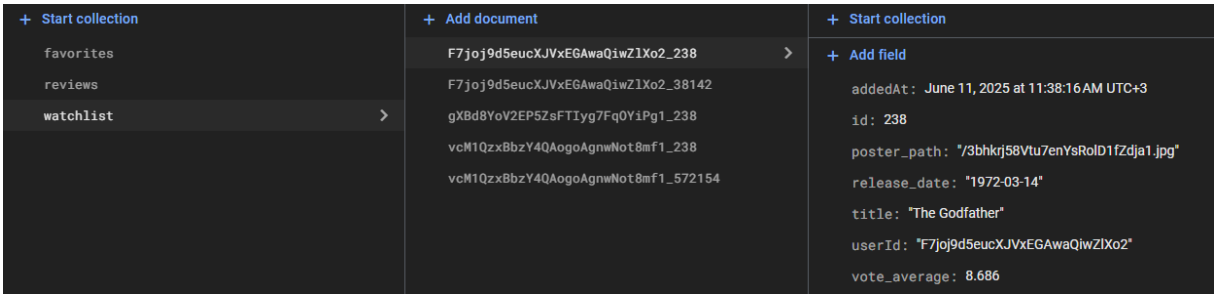
Firebase Authentication kullanıma örnek görsel



Kullanıcıların favori bilgilerinin saklandığı database yapısı



Kullanıcıların yorumlarının saklandığı database yapısı



Kullanıcıların izleme listelerinin saklandığı database yapısı

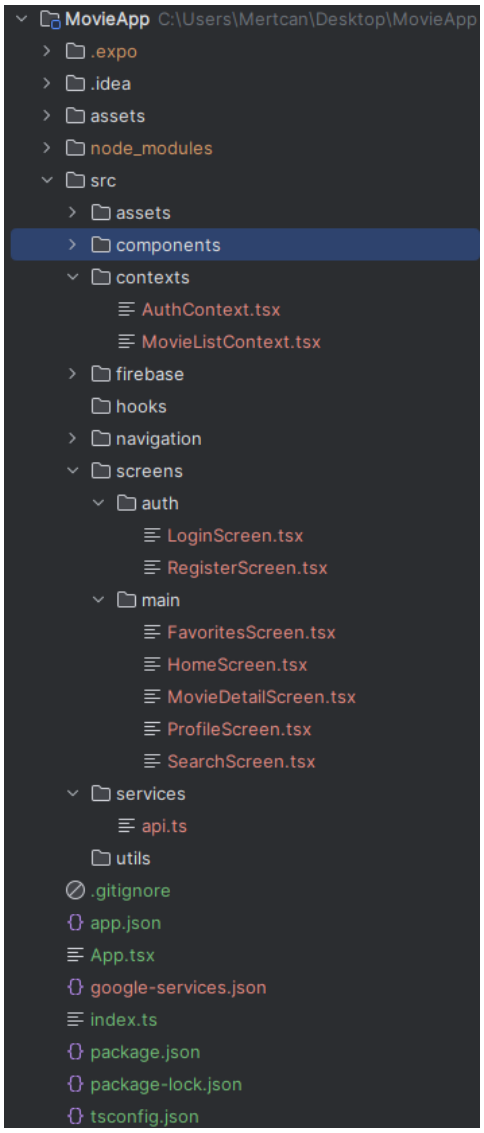
Projenin Kod Yapısı ve İşlevleri

Proje, okunabilirlik ve sürdürülebilirlik açısından modern bir React Native mimarisiyle yapılandırılmıştır. Ana dizin altında, uygulamanın temel yapı taşlarını barındıran src klasörü yer almaktadır.

- **components:** Uygulamada tekrar kullanılabilir arayüz bileşenleri burada tutulur.
- **contexts:** Uygulamanın global durum yönetimi (örneğin kullanıcı oturumu ve film listeleri) için context dosyaları burada bulunur.
- **firebase:** Firebase ile ilgili konfigürasyon ve bağlantı dosyaları bu klasörde yer alır.
- **hooks:** Özel React hook'ları burada saklanır.
- **navigation:** Uygulamanın sayfalar arası geçişlerini yöneten navigasyon dosyaları bu klasörde bulunur.
- **screens:** Uygulamanın ana ekranları (Login, Register, Home, Profile, MovieDetail, Favorites, Search) mantıksal olarak alt klasörlere ayrılmıştır.
- **services:** Dış API çağrıları ve veri çekme işlemleri için servis dosyaları burada yer alır.
- **utils:** Yardımcı fonksiyonlar ve yardımcı dosyalar için kullanılır.

Kök dizinde ise proje yapılandırma dosyaları (package.json, tsconfig.json, app.json), Firebase yapılandırma dosyası (google-services.json) ve uygulamanın giriş noktası olan App.tsx dosyası bulunmaktadır.

Bu yapı sayesinde kodun bakımı, geliştirilmesi ve yeni özelliklerin eklenmesi oldukça kolay ve düzenli bir şekilde gerçekleştirilebilmektedir.



1. Giriş Ekranı

```tsx
type LoginScreenProps = {
  navigation: NativeStackNavigationProp<any>;
};

const LoginScreen: React.FC<LoginScreenProps> = ({ navigation }) => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [isLoading, setIsLoading] = useState(false);
  const { signIn } = useAuth();

  const handleLogin = async () => {
    if (!email || !password) {
      Alert.alert('Hata', 'Lütfen email ve şifrenizi giriniz.');
      return;
    }

    setIsLoading(true);

    try {
      await signIn(email, password);
    } catch (error) {
      Alert.alert('Giriş Hatası', 'Email veya şifre hatalı.');
    } finally {
      setIsLoading(false);
    }
  };
```

Kullanıcıların e-posta ve şifre ile giriş yapmasını sağlar. Giriş işlemi sırasında Firebase Authentication kullanılır. Başarılı girişte kullanıcı ana ekrana yönlendirilir, hatalı girişte kullanıcıya uyarı gösterilir.

2. Kayıt Ekranı

```tsx
const RegisterScreen: React.FC<RegisterScreenProps> = ({ navigation }) => {
  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [confirmPassword, setConfirmPassword] = useState('');
  const [isLoading, setIsLoading] = useState(false);
  const { signUp } = useAuth();

  const handleRegister = async () => {
    // Validate inputs
    if (!username || !email || !password || !confirmPassword) {
      Alert.alert('Hata', 'Lütfen tüm alanları doldurunuz.');
      return;
    }

    if (password !== confirmPassword) {
      Alert.alert('Hata', 'Şifreler eşleşmiyor.');
      return;
    }

    // Simple email validation
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
      Alert.alert('Hata', 'Lütfen geçerli bir email adresi giriniz.');
      return;
    }

    setIsLoading(true);

    try {
      await signUp(username, email, password);
    } catch (error) {
      Alert.alert('Kayıt Hatası', 'Kayıt olurken bir hata oluştu. Lütfen tekrar deneyin.');
    } finally {
      setIsLoading(false);
    }
  };
```

Yeni kullanıcıların e-posta, şifre ve kullanıcı adı ile kayıt olmasını sağlar. Kayıt işlemi Firebase Authentication üzerinden yapılır ve kullanıcı adı, profil bilgisi olarak kaydedilir. Kayıt sonrası kullanıcı otomatik olarak giriş yapar.

3. Ana Ekran

```tsx
const HomeScreen: React.FC<HomeScreenProps> = ({ navigation }) => {
  const { user } = useAuth();
  const [trendingMovies, setTrendingMovies] = useState<Movie[]>([]);
  const [popularMovies, setPopularMovies] = useState<Movie[]>([]);
  const [topRatedMovies, setTopRatedMovies] = useState<Movie[]>([]);
  const [isLoading, setIsLoading] = useState(true);
  const [isRefreshing, setIsRefreshing] = useState(false);

  useEffect(() => {
    loadMovies();
  }, []);

  const loadMovies = async () => {
    setIsLoading(true);
    try {
      const [trending, popular, topRated] = await Promise.all([
        fetchTrendingMovies(),
        fetchPopularMovies(),
        fetchTopRatedMovies(),
      ]);

      setTrendingMovies(trending);
      setPopularMovies(popular);
      setTopRatedMovies(topRated);
    } catch (error) {
      console.error('Error loading movies:', error);
    } finally {
      setIsLoading(false);
      setIsRefreshing(false);
    }
  };

  const handleRefresh = () => {
    setIsRefreshing(true);
    loadMovies();
  };

  const handleMoviePress = (movieId: number, title: string) => {
    navigation.navigate('MovieDetail', { id: movieId, title });
  };
```

Kullanıcıya popüler, trend ve en iyi filmler gibi çeşitli film listelerini gösterir. Filmler harici bir API'den çekilir ve kullanıcılar buradan detay sayfasına geçiş yapabilir.

4. Film Detayları Ekranı

```tsx
const MovieDetailScreen: React.FC<MovieDetailProps> = ({ route, navigation }) => {
  const { id, title } = route.params;
  const { user } = useAuth();
  const {
    favorites,
    watchlist,
    toggleFavorite,
    toggleWatchlist,
    isFavorite,
    isInWatchlist,
    addReview,
    updateReview,
    deleteReview,
    getMovieReviews,
    getUserReview,
  } = useMovieList();

  const [movie, setMovie] = useState<any>(null);
  const [isLoading, setIsLoading] = useState(true);
  const [cast, setCast] = useState<CastMember[]>([]);
  const [reviews, setReviews] = useState<Review[]>([]);
  const [userReview, setUserReview] = useState<Review | undefined>(undefined);
  const [showReviewForm, setShowReviewForm] = useState(false);
  const [isEditing, setIsEditing] = useState(false);
  const [allReviews, setAllReviews] = useState<Review[]>([]);

  useEffect(() => {
    loadMovieDetails();
    loadAllReviews();
  }, [id]);

  useEffect(() => {
    if (user) {
      const movieReviews = getMovieReviews(id);
      setReviews(movieReviews);
      setUserReview(getUserReview(id));
    }
  }, [user, id, getMovieReviews, getUserReview]);
```

Film Detaylarına Ait kod bloğu 1.kısım

```
const loadAllReviews = async () => {
  try {
    const reviewsQuery = query(
      collection(firestore, 'reviews'),
      where('movieId', '==', id),
      orderBy('createdAt', 'desc')
    );
    const reviewsSnapshot = await getDocs(reviewsQuery);
    const reviewsData = reviewsSnapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    } as Review));
    setAllReviews(reviewsData);
  } catch (error) {
    console.error('Error loading all reviews:', error);
  }
};

const loadMovieDetails = async () => {
  setIsLoading(true);
  try {
    const data = await fetchMovieDetails(id);
    setMovie(data);

    // Extract cast from credits
    if (data.credits && data.credits.cast) {
      setCast(data.credits.cast.slice(0, 10)); // Limit to 10 cast members
    }

  } catch (error) {
    console.error('Error loading movie details:', error);
    Alert.alert(
      'Hata',
      'Film detayları yüklenirken bir hata oluştu. Lütfen daha sonra tekrar deneyin.'
    );
    navigation.goBack();
  } finally {
    setIsLoading(false);
  }
};
```

Film Detaylarına Ait kod bloğu 2.kısım

Seçilen filmin detaylarını, oyuncu kadrosunu, özetini ve kullanıcı yorumlarını gösterir. Kullanıcılar bu ekranda filme puan verip yorum yapabilir, favorilere veya izleme listesine ekleyebilir. Yorumlar ve puanlamalar Firestore'da saklanır ve tüm kullanıcılar tarafından görülebilir.

5. Favoriler ve İzleme Listesi Ekranı

```tsx
type TabType = 'favorites' | 'watchlist';

const FavoritesScreen: React.FC<FavoritesScreenProps> = ({ navigation }) => {
  const { favorites, watchlist } = useMovieList();
  const [activeTab, setActiveTab] = useState<TabType>('favorites');

  const handleMoviePress = (movieId: number) => {
    // Find the movie to get its title
    const movies = activeTab === 'favorites' ? favorites : watchlist;
    const movie = movies.find((m) => m.id === movieId);

    if (movie) {
      navigation.navigate('MovieDetail', { id: movieId, title: movie.title });
    }
  };

  const renderEmptyState = () => {
    return (
      <View style={styles.emptyStateContainer}>
        {activeTab === 'favorites' ? (
          <>
            <AntDesign name="heart" size={64} color="#E0E0E0" />
            <Text style={styles.emptyStateText}>Favori film bulunamadı</Text>
            <Text style={styles.emptyStateSubtext}>
              Filmleri favori listenize eklemek için film detay sayfasındaki kalp simgesine dokunun
            </Text>
          </>
        ) : (
          <>
            <MaterialIcons name="bookmark" size={64} color="#E0E0E0" />
            <Text style={styles.emptyStateText}>İzleme listenizde film yok</Text>
            <Text style={styles.emptyStateSubtext}>
              Filmleri izleme listenize eklemek için film detay sayfasındaki yer işareti simgesine dokunun
            </Text>
          </>
        )}
      </View>
    );
  };
```

Kullanıcının favori filmleri ve izleme listesine eklediği filmleri gösterir. Kullanıcılar bu ekrandan filmleri detay sayfasına geçiş yapabilir veya listeden çıkarabilir.

6. Profil Ekranı

```
const ProfileScreen: React.FC = () => {
  const { user, signOut } = useAuth();
  const { favorites, watchlist, reviews } = useMovieList();

  const handleSignOut = () => {
    Alert.alert(
      'Çıkış Yap',
      'Hesabınızdan çıkış yapmak istediğinize emin misiniz?',
      [
        { text: 'İptal', style: 'cancel' },
        {
          text: 'Çıkış Yap',
          onPress: async () => {
            try {
              await signOut();
            } catch (error) {
              console.error('Error signing out:', error);
              Alert.alert('Hata', 'Çıkış yapılırken bir hata oluştu.');
            }
          },
        },
      ]
    );
  };
```

Kullanıcıya ait temel bilgileri (kullanıcı adı, e-posta) ve kullanıcının yaptığı yorumları, favori ve izleme listesindeki filmleri özetler. Kullanıcı bu ekrandan çıkış yapabilir.

7. Arama Ekranı

```tsx
const SearchScreen: React.FC<SearchScreenProps> = ({ navigation }) => {
  const [query, setQuery] = useState('');
  const [results, setResults] = useState<Movie[]>([]);
  const [isLoading, setIsLoading] = useState(false);
  const [hasSearched, setHasSearched] = useState(false);

  const handleSearch = async () => {
    if (!query.trim()) return;

    setIsLoading(true);
    setHasSearched(true);
    Keyboard.dismiss();

    try {
      const searchResults = await searchMovies(query);
      setResults(searchResults);
    } catch (error) {
      console.error('Error searching movies:', error);
    } finally {
      setIsLoading(false);
    }
  };

  const handleMoviePress = (movieId: number) => {
    // Find the movie to get its title
    const movie = results.find((m) => m.id === movieId);
    if (movie) {
      navigation.navigate('MovieDetail', { id: movieId, title: movie.title });
    }
  };

  const renderEmptyState = () => {
    if (!hasSearched) {
      return (
        <View style={styles.emptyStateContainer}>
          <AntDesign name="search1" size={64} color="#BDBDBD" />
          <Text style={styles.emptyStateText}>
            Film adı veya anahtar kelime girerek arama yapın
          </Text>
        </View>
      );
    }
```

Kullanıcıların film adı veya anahtar kelime ile film araması yapmasını sağlar. Sonuçlar harici bir film API'sinden çekilir ve kullanıcılar buradan film detayına geçiş yapabilir.

Gerektiği durumlarda kullanılması için
TMDB API Key: 8683f807867bbdbb9a7b6c872498705