T.C.
SAKARYA ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BSM 498 BİTİRME ÇALIŞMASI

MOBİL DÖVİZ İŞLEMLERİ BANKACILIK UYGULAMASI

B191210067- ZELAL İNANÇ

Fakülte Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ
Tez Danışmanı : Prof. Dr. NEJAT YUMUŞAK

2022-2023 Yaz Dönemi

ÖNSÖZ

React Native, günümüzün modern mobil uygulama geliştirme alanında önemli bir yere sahip olan bir teknolojidir. Hem hızlı geliştirme süreci hem de platform bağımsız yapısı sayesinde, farklı mobil platformlarda (iOS ve Android) aynı kod tabanını kullanarak uygulama geliştirmek büyük bir avantaj sağlar. React Native, geliştirme sürecini kolaylaştıran birçok araç ve kütüphane sunarak, aynı kod tabanını farklı platformlarda kullanma olanağı sağlar.

İÇİNDEKİLER

BÖLÜM 4.
SONUÇLAR VE ÖNERİLER

ŞEKİLLER LİSTESİ

ÖZET

Anahtar kelimeler: Mobil Uygulama Geliştirme, Döviz İşlemleri, Firebase

Bu mobil uygulama, React Native teknolojisiyle tasarlanmış ve döviz işlemleri yapan bir bankacılık platformunu kullanıcılara sunmayı amaçlamaktadır. Uygulama, kullanıcıların kolayca döviz işlemleri yapabilmesine olanak tanırken, Firebase veritabanı entegrasyonu ile güvenli ve gerçek zamanlı bir deneyim sunmayı hedefler. Firebase'in sunduğu veri güvenliği ve anlık veri senkronizasyonu kullanıcı deneyimini zenginleştirir.Uygulama, modern teknoloji ve bankacılık ihtiyaçlarını birleştirerek kullanıcıların finansal işlemleri kolayca ve güvenli bir şekilde gerçekleştirmelerini sağlar.

BÖLÜM 1. GİRİŞ

React Native, Facebook tarafından geliştirilen bir mobil uygulama çerçevesidir. Bu çerçeve, kullanıcıların iOS ve Android platformlarında benzersiz deneyimler sunan doğal mobil uygulamalar geliştirmesine olanak tanırken, aynı kod tabanını kullanarak verimliliği artırır. React Native, JavaScript kullanarak, bileşen tabanlı bir yaklaşımla mobil uygulama geliştirmeyi sağlar.

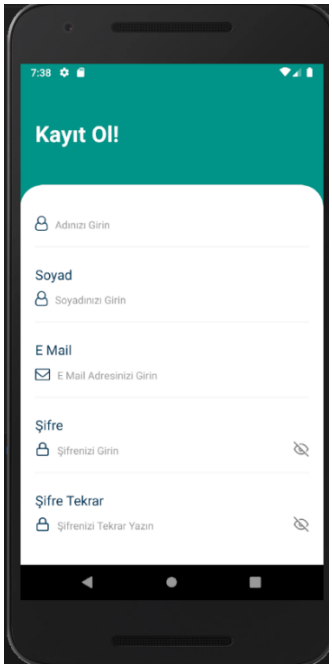
1.1. Proje Gereksinimleri Ve Proje Mimarisi

Dijital dönüşümün hızla ilerlediği bir dönemde, finansal işlemleri daha erişilebilir, hızlı ve kullanıcı dostu bir şekilde gerçekleştirmek giderek daha önemli hale geliyor. Bu bağlamda, mobil döviz işlemleri yapan bankacılık uygulamasını React Native teknolojisiyle hayata geçirmeye karar verdim. Bu projenin amacı döviz alım satım işlemlerinin yapılabildiği bir mobil banka hesabı oluşturulmasıdır.

1.1.1. Kayıt Olma

Kullanıcılar mobil uygulama üzerinden isim , soy isim, email adresi ve şifre girerek kayıt olabilmelidir. Kayıt başarısız olduğunda veya herhangi bir hata durumunda kullanıcıya uyarı mesajı gösterilmelidir. Kayıt olma işleminin başarılı olması durumunda login sayfasına yönlendirme yapılır.

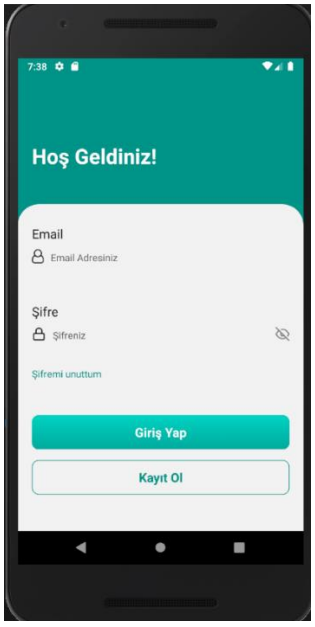
(1.1)



1.1.2. Giriş Yapma

Kullanıcılar Email adresleri ve şifre ile uygulamaya giriş yapabilmelidir.

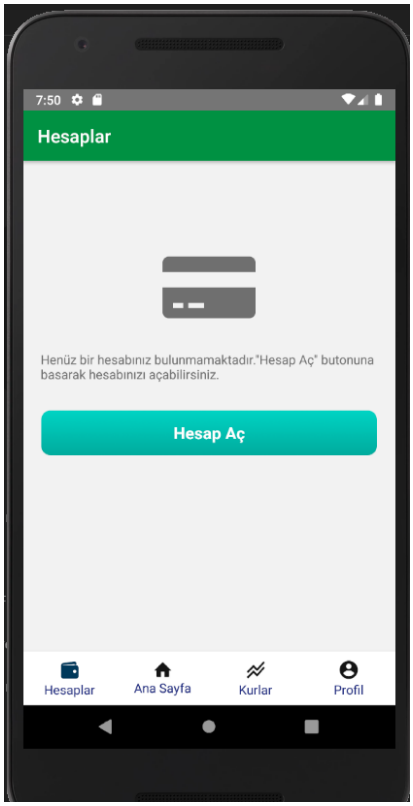
(1.2)



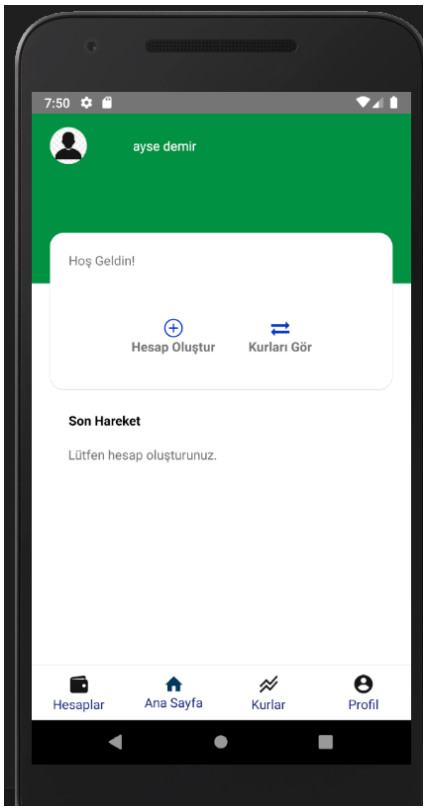
1.1.3. Hesap Oluşturma

Kullanıcıların döviz işlemlerini gerçekleştirebilmeleri için hesap oluşturmaları gerekmektedir. Bu hesaplar üzerinde bulunan bakiyeler transfer edilerek işlem gerçekleşir. Oluşturulan hesaplar döviz alış-satış işlemlerinde kullanılacaktır.

(1.3)



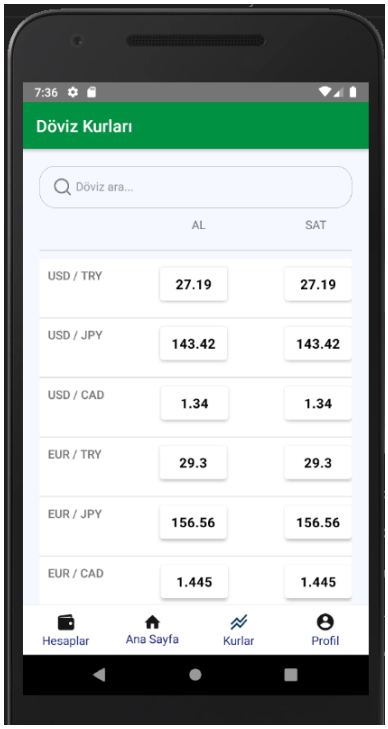
(1.4)



1.1.4. Döviz Kur Takip Sayfası

Kullanıcılar izledikleri bir döviz çiftine tıklayarak alım satım yapabilir veya istedikleri döviz kurunu arama kısmından bakabilirler. Burada alım yapacağı hesabı ve işlemin aktarılacağı hesabı seçmesi gerekmektedir. Bu ekran üzerinde kullanıcının işlem yapacağı tutarı girebileceği bir alan olmalıdır. Buraya girilen değer ile kur bilgisinden işlem yapılır. İşlem sonucunda tutar hesaplar arasında aktarılmış olur.

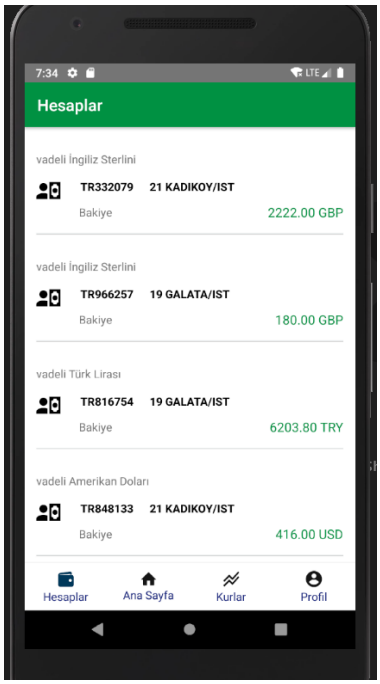
(1.5)



1.1.5. Kullanıya Ait Hesaplar

Kullanıcıların oluşturdukları tüm banka hesaplarının listelendiği bir ekrandır.

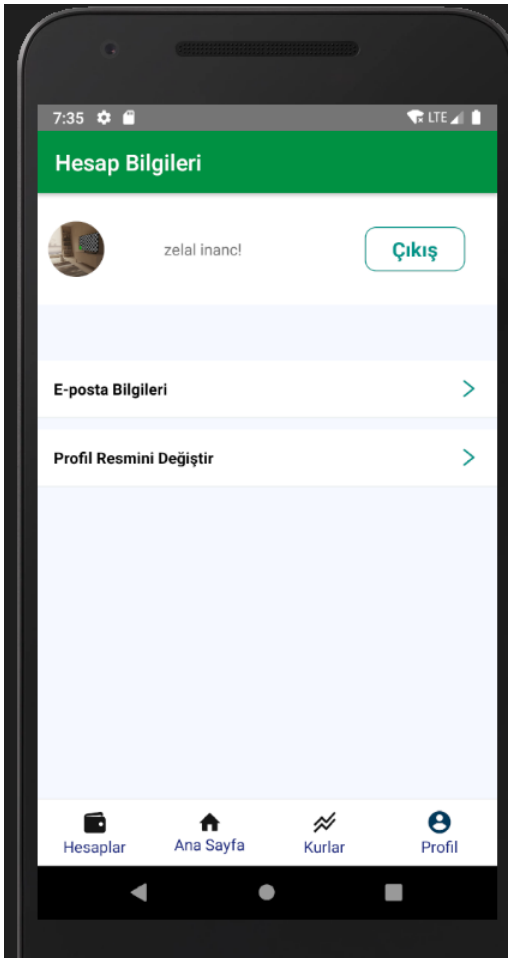
(1.6)

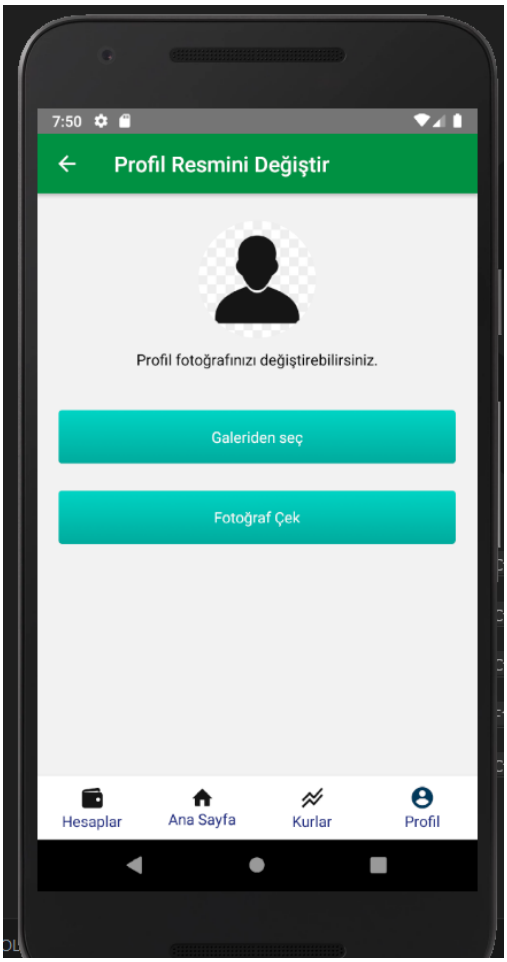


1.1.6. Kullanıcı Profil Sayfası

Kullanıcıların email adreslerini güncelleyebildiği ve profil fotoğrafını ekleyip, değiştirebildiği sayfadır.

(1.7) **(1.8)**





1.2 React Hook Kullanımı

Hooks, sınıf bileşenleri yerine fonksiyonel bileşenlerde daha kolay ve okunabilir kod yazmanıza olanak tanır.

Hooks, temel olarak bileşen içinde state (durum) yönetimi, yaşam döngüsü metotlarının yerine geçen işlemler ve diğer React özelliklerini daha temiz ve modüler

bir şekilde kullanmanıza yardımcı olur. Bununla birlikte, Hooks kullanımı bileşenler arasında mantıksal kodun paylaşımını da kolaylaştırır.

1.2.1 UseState Hook İle Durum Yönetimi

useState Hook'u, React fonksiyonel bileşenlerinde state (durum) yönetimi sağlayan bir mekanizmadır. Bu Hook'u kullanarak karmaşık sınıf bileşenlerinin yerini alarak, durum değerlerini tanımlamak ve güncellemek oldukça basit hale gelir.

useState fonksiyonu, state değerini ve bu değeri güncellemek için kullanılacak fonksiyonu döner. İlk parametre olarak başlangıç değeri verilir. Bu başlangıç değeri, bileşenin başlangıç durumunu belirtir. İkinci değer olarak, state değerini güncellemek için kullanılacak olan fonksiyon alınır. Bu fonksiyon, state değerini güncellediğinizde bileşenin yeniden render edilmesini tetikler.Aşağıda kod örneği kullanıcının kayıt olduğu sayfadan bir örnektir.Başta isim,soyisim,email ve şifre değerlerine boş değerler set edilir.Daha sonra kullanıcının girdiği inputlar ile bu değerler set edilmiş olur.

(1.9)

```
26
27      const [data, setData] = React.useState({
28          username: '',
29          userLastName: '',
30          userMail: '',
31          password: '',
32          confirm_password: '',
33          check_textInputChange: false,
34          check_textLastNameChange: false,
35          check_textEmailChange: false,
36          secureTextEntry: true,
37          isValidUser: true,
38          isLastName:true,
39          isEmail:true,
40          isValidPassword:true,
41          isValidConfirmPassword:true,
42          confirm_secureTextEntry: true,
43      });
44      const textLastNameChange = (val) => {
45          if( val.length >= 4 ) {
46              setData({
47                  ...data,
48                  userLastName: val,
49                  isLastName:true,
50                  check_textLastNameChange: true
51              });
52          } else {
53              setData({
54                  ...data,
55                  userLastName: val,
56                  isLastName: false,
57                  check_textLastNameChange: false
58              });
59          }
60      }
```

1.2.2 UseEffect Hook Yaşam Döngüsü ve Etkileşim Yönetimi

React Native'in fonksiyonel bileşenlerinde, bileşenin yaşam döngüsü metotlarının yerini alan useEffect Hook'u, bileşenin farklı aşamalarında işlemleri gerçekleştirmek için kullanılır. Bu Hook, bileşenin monte edilme, güncellenme ve çıkarılma aşamalarında tetiklenir ve ayrıca state veya prop değişikliklerine tepki verebilir.

Sınıf bileşenlerindeki yaşam döngüsü metotlarına alternatif bir yaklaşım sunar. Bu sayede, API çağrıları, veri alışverişi, DOM manipülasyonları, aboneliklerin yönetimi gibi işlemleri daha etkili bir şekilde gerçekleştirmemizi sağlar. Aşağıda kod örneği kullanıcının profil sayfasından bir örnektir.Bu sayfa her render edildiğinde getUserDetail fonksiyonu render edilmiş olur.Böylece bilgiler alınır.

(1.10)

```
const ProfileScreen = ({ navigation}) => {

const {logout,getUserDetail,userName,userLastName,userImage,userIdentity} = useContext(AuthContext);


 useEffect(() => {
   getUserDetail();


 }, []);
```

1.2.3 UseContext Hook Bileşenler Arası Veri Paylaşımını Kolaylaştırmak

React uygulamalarında, farklı bileşenler arasında veri paylaşımı önemli bir rol oynar. Ancak, veriyi props ile iletmek bazen karmaşık ve gereksiz tekrarlarla dolu bir süreç olabilir. Burada useContext Hook'u devreye girer ve veri paylaşımını daha temiz hale getirir.

(1.11)

```
const {getUserDetail,userName,userLastName,userImage,userAccounts} = useContext(AuthContext);
```

1.3. React Navigation Kullanımı

React Navigation, React Native uygulamalarında gezinme (navigation) işlemlerini yönetmek için kullanılan bir kütüphanedir. Uygulamanızın farklı ekranları arasında geçiş yapmak, parametreleri iletmek, gezinme çubuğunu (navigation bar) yönetmek gibi işlevleri sağlar. React Navigation, uygulamanızın karmaşık gezinme yapısını daha yönetilebilir hale getirir.

React Navigation, farklı türde gezinme stillerini (stack, tab, drawer vb.) destekler. "Stack Navigation", bu türden bir gezinme stiline örnektir. Stack Navigation, uygulamada sayfaları bir yığıt (stack) şeklinde yönetir. Bu, bir sayfa açıldığında önceki sayfanın üstüne eklenmesi ve geri tuşuna basıldığında en üstteki sayfanın kapatılması anlamına gelir.

Tab Navigation, React Navigation kütüphanesinin sağladığı bir gezinme (navigation) stili ve bileşenidir. Tab Navigation, uygulamanın farklı bölümlerini sekme (tab) şeklinde düzenler ve kullanıcıların bu sekmeler arasında hızlıca geçiş yapmalarını sağlar. Her sekmeye tıkladığınızda ilgili içeriğin görüntülendiği bir ekran açılır.

CreateBottomTabNavigator, React Navigation kütüphanesinin sunduğu bir komponenttir ve Tab Navigation'un bir türüdür. Bu yöntem, alt tab çubuğuna sahip bir Tab Navigation oluşturmanızı sağlar. Kullanıcılar alt tab çubuğundaki sekmelere tıklayarak farklı ekranlar arasında gezinebilirler. Bottom Tab Navigator, kullanıcıların alt tab çubuğu üzerinden farklı ekranlara hızlıca geçiş yapmasını sağlayan etkili bir gezinme stili sunar. Bu yöntem, uygulamanızdaki içerikleri kategorilere veya bölümlere ayırmamızı sağlar.

(1.12)

BÖLÜM 2. REACT NATİVE STATE YÖNETİMİ

React Native'de state yönetimi, uygulamanın dinamik yapısını ve kullanıcı etkileşimlerini yönetmek için kritik bir rol oynar. Bu, uygulama durumunu (state) saklama, güncelleme ve bileşenler arasında iletim süreçlerini içerir. Farklı state yönetim yöntemleri mevcutken, her biri belirli senaryolara ve projelerin ihtiyaçlarına yönelik avantajlar sunar.

2.1. State Yönetiminin Önemi

React Native uygulamalarında state yönetimi, uygulamanın durumunu (state) tutma ve güncelleme süreçlerini kontrol etmek açısından önemlidir. State, kullanıcının etkileşimleri, veri akışı ve bileşenlerin durumu gibi dinamik değişiklikleri yönetmek için gereklidir. Mobil uygulamalar genellikle kullanıcı etkileşimleri, API çağrıları ve veri güncellemeleri ile dinamik içeriği yönetir. State yönetimi, bu tür değişen durumları koordine ederek kullanıcı deneyimini düzenler. State, uygulama genelindeki veri akışını ve senkronizasyonu yönetir. Kullanıcı etkileşimleri ve veri güncellemeleri, doğru sıra ve zamanlamayla ele alınarak tutarlı bir kullanıcı

deneyimi sağlanır. İyi bir state yönetimi, kodun daha temiz, daha düzenli ve daha sürdürülebilir olmasını sağlar. Durumun merkezi bir şekilde yönetilmesi, kod tekrarını azaltır ve hataları en aza indirir.

2.2. Context API ile State Paylaşımı

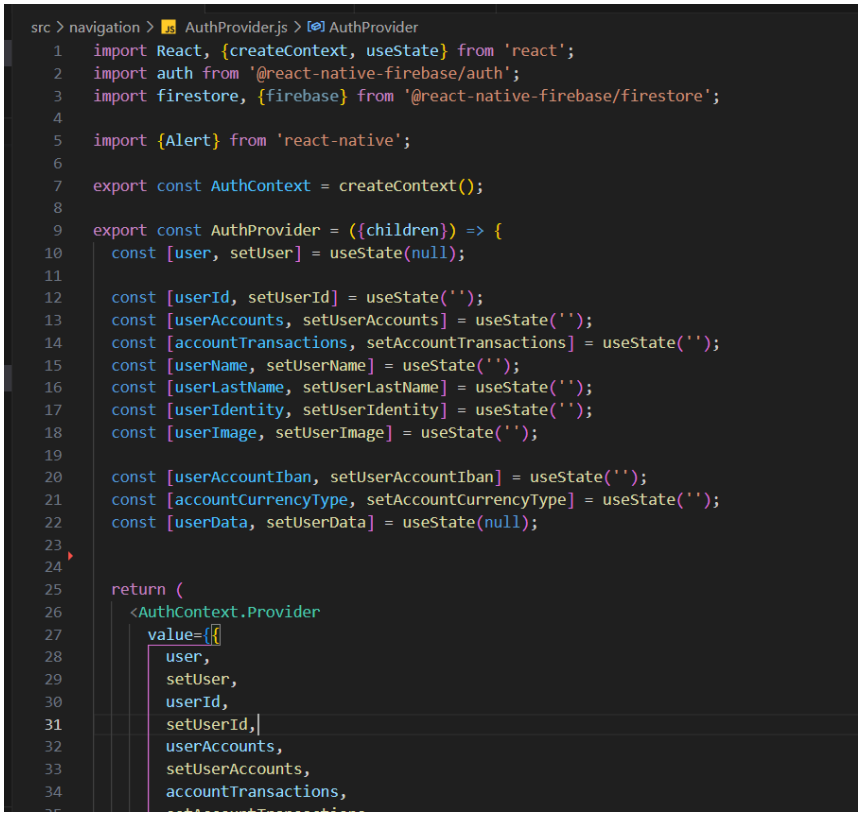
Context API, React uygulamalarında bileşenler arasında veri paylaşımını kolaylaştıran bir mekanizmadır. Bu API sayesinde, prop drilling sorununu çözmek (prop drilling, bir state'in, component ağacımızın daha yukarısında bulunan bir parent component'ten, component ağacımızın altlarında bulunan bir child component'e props yoluyla aktarılarak state'in ulaştırılmasına denir) ve veriyi daha derin seviyelerdeki bileşenlere iletmek daha basit hale gelir. Context API, genellikle orta ölçekli uygulamalarda veya belirli bileşenler arasında veri paylaşımını kolaylaştırmak için tercih edilir.

2.2.1 AuthContext Kullanımı

AuthContext provider, kullanıcı kimlik doğrulaması (authentication) ve yetkilendirme (authorization) verilerini bileşenler arasında paylaşmak ve yönetmek için kullanılan bir Context API sağlayıcısıdır. Bu tür bir sağlayıcı, kullanıcı oturum durumu, yetkilendirme bilgileri ve kimlik doğrulama işlemleri gibi kritik verileri uygulamanın farklı bileşenleri arasında iletmeyi ve güncellemeyi kolaylaştırır.

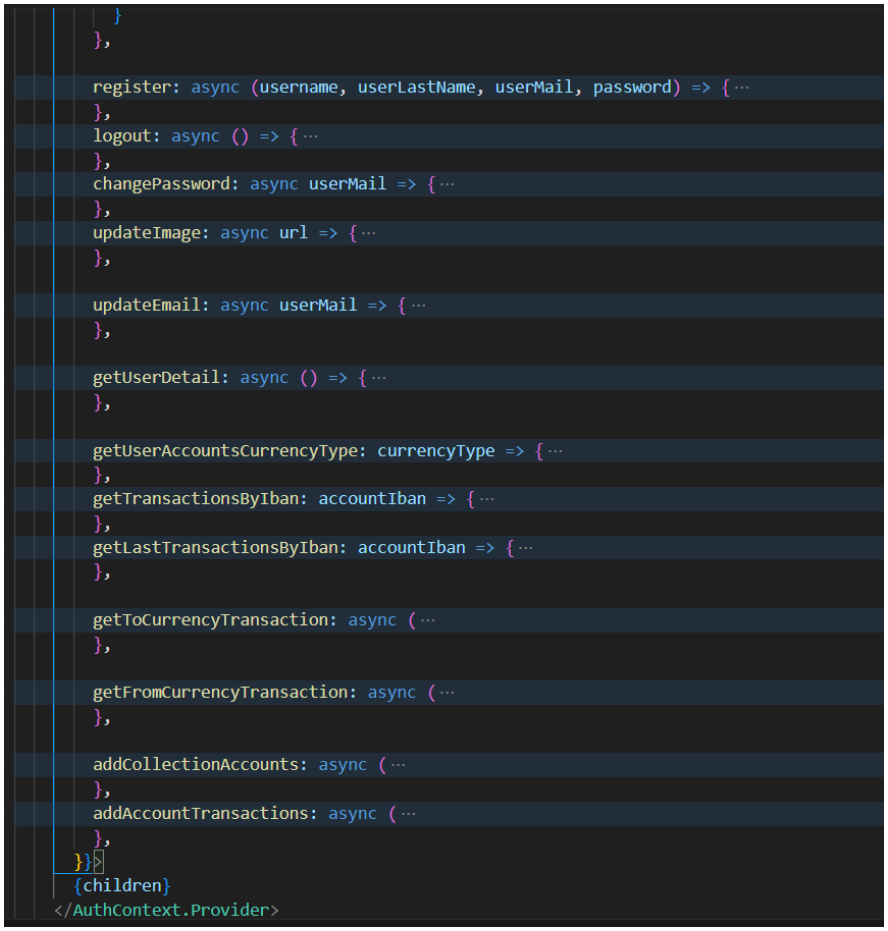
Bir AuthContext provider oluşturarak, uygulamanızın farklı bölgelerindeki bileşenlerin bu kimlik doğrulama ve yetkilendirme verilerine erişimini ve güncellemesini sağlayabilirsiniz. Bu tür bir yapı, kullanıcının giriş yapması, çıkış yapması, yetkilerini kontrol etmesi gibi işlemleri düzenlemeyi ve merkezi bir yerden yönetmeyi kolaylaştırır.

(2.1)



AuthContext içinde sadece kimlik doğrulama ve yetkilendirme ile ilgili fonksiyonlar değil, aynı zamanda kullanıcı ile ilgili diğer işlemleri de barındıracak fonksiyonlar da yazabiliriz.Benim projemde oluşturduklarım şifre değiştirme, resim güncelleme, mail adresi güncelleme, kullanıcı bilgilerini getirme, kullanıcıya hesap ekleme ve benzeri fonksiyonlardır.

(2.2)



BÖLÜM 3. REACT NATİVE VERİTABANI ENTEGRASYONU

React Native uygulamalarında veri entegrasyonu, uygulamanızın dışarıdan veri kaynaklarına erişmesi ve bu verileri kullanması anlamına gelir. Veri entegrasyonu, uygulamanızın dinamik ve gerçek zamanlı verileri almasını, göstermesini ve güncellemesini sağlar. Genellikle web servisleri veya API'lar aracılığıyla bu veri kaynaklarına erişilir.

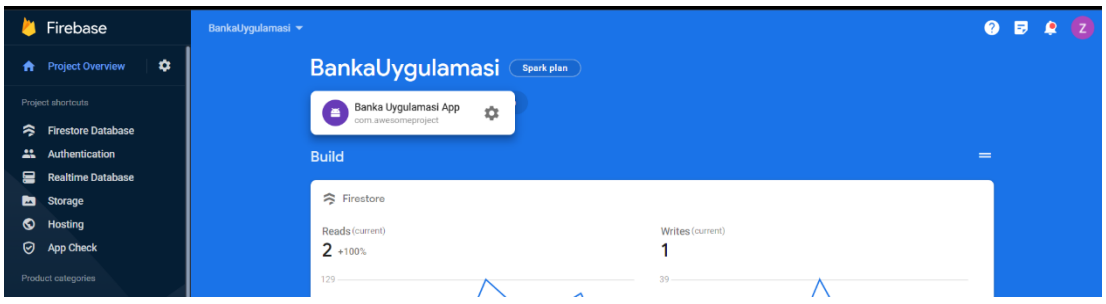
3.1. Firebase Nedir Ve Neden Kullanılır

React Native'de Firebase kullanarak veri entegrasyonu gerçekleştirmek oldukça yaygın bir yöntemdir. Firebase, Google tarafından sunulan bir mobil ve web uygulama geliştirme platformudur. Geliştiricilere kullanıcı kimlik doğrulaması, veritabanı, depolama, analiz, hata izleme, mesajlaşma ve daha pek çok aracı sunarak uygulama geliştirme süreçlerini hızlandırmayı ve kolaylaştırmayı amaçlar. Firebase, gerçek zamanlı veritabanı ve dosya depolama hizmetleri sunar. Bu hizmetler, uygulamanızın anlık güncellemelerle veri paylaşmasını veya kullanıcıların medya dosyalarını depolamasını sağlar. Firebase, özellikle küçük ve orta ölçekli projeler için hızlı, kullanışlı ve güvenilir bir geliştirme platformu sunar. Geliştiricilerin karmaşık altyapı işlerine odaklanmaktan ziyade uygulama mantığını geliştirmelerine imkan sağlar.

3.2. Firestore Veritabanı Verileri Saklama ve Yönetme

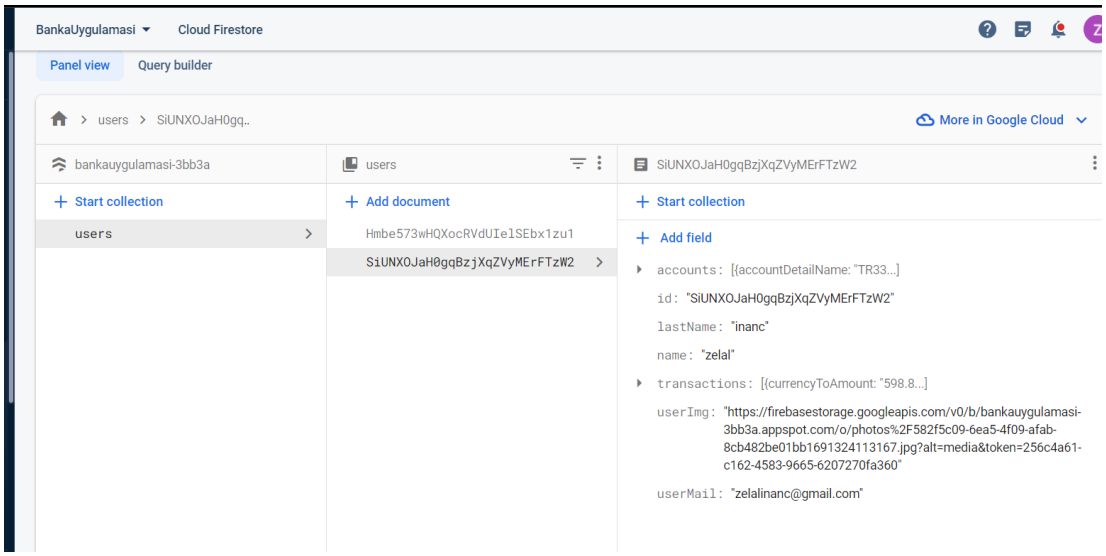
Firebase'in kendi sayfasından uygun adımları izleyerek gerekli konfigürasyon işlemleriyle projemize firebase bağlantısını kurabiliriz.

(3.1)



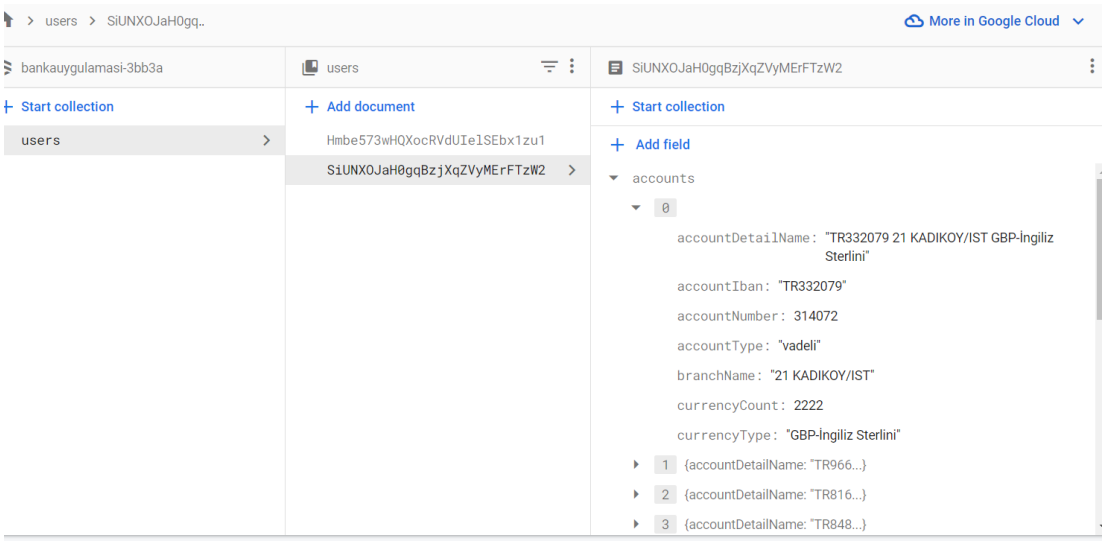
Firestore, Firebase tarafından sunulan bulut tabanlı bir gerçek zamanlı veritabanıdır. Firestore'u kullanarak veri saklamak ve yönetmek oldukça kolaydır. Veri modelinizi tasarlarken koleksiyonlar ve belgeler kullanılır. Koleksiyonlar, belgelerin gruplandığı yerlerdir ve belgeler ise verilerin depolandığı birimlerdir.Bu projede "users" koleksiyonu içinde her bir kullanıcıya ait belgeler bulunmaktadır.Belgeler kullanıcıların id lerine göre isimlendirilmiştir.

(3.2)

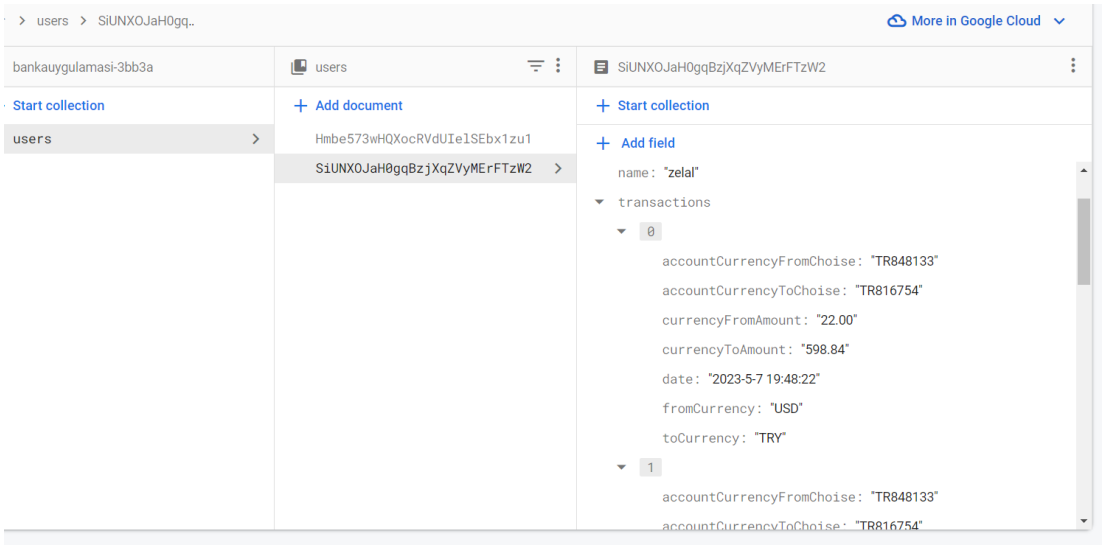


Kullanıcının name,lastName,userImg ve userMail özellikleri string olarak tutulurken kullanıcıya ait birden fazla accounts ve bu accountslara ait transaction işlemleri olacağından dolayı accounts ve transactions özelliği array olarak oluşturulmuştur.

(3.3)



(3.4)



3.2.1. Veri Ekleme Ve Güncelleme

Firestore'da veri eklemek veya güncellemek için, ilgili koleksiyon ve belgeye erişim sağlamanız gerekmektedir. Koleksiyon ve belge referansları kullanarak set() veya update() gibi metodları kullanarak veri ekleyebilir veya güncelleyebiliriz.

Projemde birçok yerde ekleme ve güncelleme metodlarından yararlandım.Örneğin kullanıcın profil fotoğrafını güncellemek için yazdığım kod aşağıdaki gibidir.

(3.5)

```
                {text: 'Tamam'},
              ]);
            });
        },
        updateImage: async url => {
          firestore()
            .collection('users')
            .doc(user.uid)
            .update({
              userImg: url,
            })
            .catch(error => {});
        },
```

Kullanıcının banka hesabı ekleme kodu da şu şekildedir.

(3.6)

```
addCollectionAccounts: async (
  accountType,
  currencyType,
  branchName,
  accountNumber,
  accountIban,
  currencyCount,
) => {
  let accounts = [];

  let tempUserAccounts;

  userAccounts && userAccounts.length
    ? (tempUserAccounts = userAccounts)
    : (tempUserAccounts = accounts);

  tempUserAccounts.push({
    accountType: accountType,
    currencyType: currencyType,
    branchName: branchName,
    accountNumber: accountNumber,
    accountIban: accountIban,
    currencyCount: parseFloat(currencyCount),
    accountDetailName:
      accountIban + ' ' + branchName + ' ' + currencyType,
  });

  firestore()
    .collection('users')
    .doc(user.uid)
    .update({
      accounts: tempUserAccounts,
    })
    .catch(error => {});
},
```

3.2.2. Veri Okuma Ve Listeleme

Firestore'dan veri okuma işlemi get() veya onSnapshot() gibi metodlarla yapılır. get() anlık bir sorgu sonucu dönerken, onSnapshot() gerçek zamanlı olarak verileri izlemenizi sağlar.

Projede kullanıcı bilgilerini getiren getUserDetail isimli bir fonksiyon da yazdım.

(3.7)

3.3. Firebase Authentication Kullanıcı Kimlik Doğrulama

Firebase Authentication, Firebase'in sunmuş olduğu kullanıcı kimlik doğrulama hizmetidir. Bu hizmet sayesinde uygulamamızdaki kullanıcıları güvenli bir şekilde yönetebilir, kullanıcıların giriş yapmasını, kaydolmasını ve çıkış yapmasını kolayca sağlayabiliriz.

(3.8)

```
register: async (username, userLastName, userMail, password) => {
  try {
    const response = await auth().createUserWithEmailAndPassword(
      userMail,
      password,
    );

    const useData = {
      id: response.user.uid,
      name: username,
      lastName: userLastName,
      userMail: userMail,
      userImg: '',
    };

    await firestore()
      .collection('users')
      .doc(auth().currentUser.uid)
      .set(useData);
  } catch (e) {
    console.log(e);
  }
},
logout: async () => {
  try {
    await auth().signOut();
  } catch (e) {
    console.log(e);
  }
},
```

(3.9)

```
login: async (userMail, password) => {
  try {
    await auth().signInWithEmailAndPassword(userMail, password);
  } catch (e) {
    const error = 'E Mail  ve şifre uyuşmuyor';

    Alert.alert(error);
  }
},
```

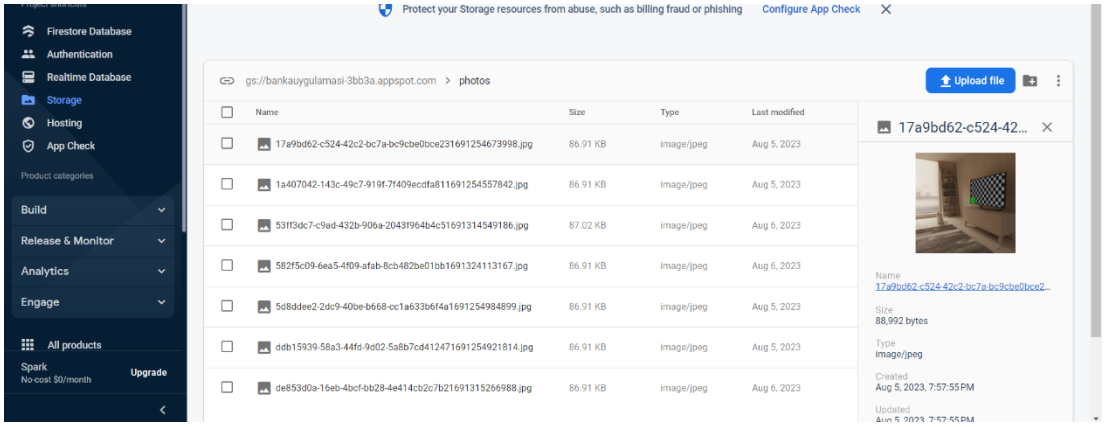
3.4. Storage API Medya Dosyalarını Yönetme

Firebase Storage, medya dosyalarını (resimler, videolar, ses dosyaları vb.) güvenli ve ölçeklenebilir bir şekilde depolamak ve yönetmek için kullanılan bir hizmettir. Firebase Storage API kullanarak medya dosyalarını yükleme, indirme, silme ve yönetme işlemleri yapabiliriz.Projemde kullanıcı profil fotoğrafını ekleme veya güncelleme işlemini gerçekleştirdikten sonra bu resimler storageda depolanır.

(3.10)

```
const uploadImage = async () => {
  const uploadUri = profile;

  let filename = uploadUri.substring(uploadUri.lastIndexOf('/') + 1);

  console.log(filename);

  const extension = filename.split('.').pop();
  const names = filename.split('.').slice(0, -1).join('.');
  filename = names + Date.now() + '.' + extension;

  setTransferred(0);

  const storageRef = storage().ref(`photos/${filename}`);
  const task = storageRef.putFile(uploadUri);


  task.on('state_changed', taskSnapshot => {
    console.log(
      `${taskSnapshot.bytesTransferred} transferred out of ${taskSnapshot.totalBytes}`,
    );

    setTransferred(
      Math.round(taskSnapshot.bytesTransferred / taskSnapshot.totalBytes) *
        100,
    );
  });

  try {
    await task;

    const url = await storageRef.getDownloadURL();
```

(3.11)



BÖLÜM 4. SONUÇLAR VE ÖNERİLER

Sonuç olarak projemde kullanıcı banka hesapları açabilecek, güncel döviz kur oranlarını görebilecek ve istediği döviz çiftleri arasında işlem yapabilecektir. Yaptığı işlemlerinin özetini de görebilecektir.

KAYNAKLAR

[1] https://rnfirebase.io/.

[2] https://reactnative.dev/.

[3] https://oblador.github.io/react-native-vector-icons/.

[4] https://medium.com/swlh/learn-firebase-authentication-using-react-native-0-60-and-react-native-firebase-2319c4d60ebe.

[5] https://medium.com/cleverprogrammer/the-react-context-api-364da590aa73.

ÖZGEÇMİŞ

Zelal İnanç, İstanbul'da 1997 yılında doğdu. Lise eğitimini İstanbul Büyükçekmece Atatürk Anadolu Lisesi'nde tamamladı. 2019 senesinde Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nde lisans eğitimine başladı.

BSM 498 BİTİRME ÇALIŞMASI
DEĞERLENDİRME VE SÖZLÜ SINAV TUTANAĞI

KONU : MOBİL DÖVİZ İŞLEMLERİ BANKACILIK UYGULAMASI
ZELAL İNANÇ B191210067

Değerlendirme Konusu	İstenenler	Not Aralığı	Not
Yazılı Çalışma			
Çalışma klavuza uygun olarak hazırlanmış mı?	x	0-5	
Teknik Yönden			
Problemin tanımı yapılmış mı?	x	0-5	
Geliştirilecek yazılımın/donanımın mimarisini içeren blok şeması (yazılımlar için veri akış şeması (dfd) da olabilir) çizilerek açıklanmış mı?			
Blok şemadaki birimler arasındaki bilgi akışına ait model/gösterim var mı?			
Yazılımın gereksinim listesi oluşturulmuş mu?			
Kullanılan/kullanılması düşünülen araçlar/teknolojiler anlatılmış mı?			
Donanımların programlanması/konfigürasyonu için yazılım gereksinimleri belirtilmiş mi?			
UML ile modelleme yapılmış mı?			
Veritabanları kullanılmış ise kavramsal model çıkarılmış mı? (Varlık ilişki modeli, noSQL kavramsal modelleri v.b.)			
Projeye yönelik iş-zaman çizelgesi çıkarılarak maliyet analizi yapılmış mı?			
Donanım bileşenlerinin maliyet analizi (prototip-adetli seri üretim vb.) çıkarılmış mı?			
Donanım için gerekli enerji analizi (minimum-uyku-aktif-maksimum) yapılmış mı?			
Grup çalışmalarında grup üyelerinin görev tanımları verilmiş mi (iş-zaman çizelgesinde belirtilebilir)?			
Sürüm denetim sistemi (Version Control System; Git, Subversion v.s.) kullanılmış mı?			
Sistemin genel testi için uygulanan metotlar ve iyileştirme süreçlerinin dökümü verilmiş mi?			
Yazılımın sızma testi yapılmış mı?			
Performans testi yapılmış mı?			
Tasarımın uygulamasında ortaya çıkan uyumsuzluklar ve aksaklıklar belirtilerek çözüm yöntemleri tartışılmış mı?			
Yapılan işlerin zorluk derecesi?	x	0-25	
Sözlü Sınav			
Yapılan sunum başarılı mı?	x	0-5	
Soruları yanıtlama yetkinliği?	x	0-20	
Devam Durumu			
Öğrenci dönem içerisindeki raporlarını düzenli olarak hazırladı mı?	x	0-5	
Diğer Maddeler			
Toplam			

DANIŞMAN (JÜRİ ADINA): NEJAT YUMUŞAK
DANIŞMAN İMZASI: