**KARADENİZ TEKNİK ÜNİVERSİTESİ**
**MÜHENDİSLİK FAKÜLTESİ**
**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

# ANDROİD MOBİL MARKET UYGULAMASI

**BİTİRME PROJESİ**

**Kürşat ERCAN**
**Burak ALTUNTAŞ**

**2020-2021 BAHAR DÖNEMİ**

**KARADENİZ TEKNİK ÜNİVERSİTESİ**
**MÜHENDİSLİK FAKÜLTESİ**
**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

# ANDROİD MOBİL MARKET UYGULAMASI

**BİTİRME PROJESİ**

**Kürşat ERCAN**
**Burak ALTUNTAŞ**

**2020-2021 BAHAR DÖNEMİ**

# IEEE Etik Kuralları
# IEEE Code of Ethics

Mesleğime karşı şahsi sorumluluğumu kabul ederek, hizmet ettiğim toplumlara ve üyelerine en yüksek etik ve mesleki davranışta bulunmaya söz verdiğimi ve aşağıdaki etik kurallarını kabul ettiğimi ifade ederim:

1. Kamu güvenliği, sağlığı ve refahı ile uyumlu kararlar vermenin sorumluluğunu kabul etmek ve kamu veya çevreyi tehdit edebilecek faktörleri derhal açıklamak;
2. Mümkün olabilecek çıkar çatışması, ister gerçekten var olması isterse sadece algı olması, durumlarından kaçınmak. Çıkar çatışması olması durumunda, etkilenen taraflara durumu bildirmek;
3. Mevcut verilere dayalı tahminlerde ve fikir beyan etmelerde gerçekçi ve dürüst olmak;
4. Her türlü rüşveti reddetmek;
5. Mütenasip uygulamalarını ve muhtemel sonuçlarını gözeterek teknoloji anlayışını geliştirmek;
6. Teknik yeterliliklerimizi sürdürmek ve geliştirmek, yeterli eğitim veya tecrübe olması veya işin zorluk sınırları ifade edilmesi durumunda ancak başkaları için teknolojik sorumlulukları üstlenmek;
7. Teknik bir çalışma hakkında yansız bir eleştiri için uğraşmak, eleştiriyi kabul etmek ve eleştiriyi yapmak; hatları kabul etmek ve düzeltmek; diğer katkı sunanların emeklerini ifade etmek;
8. Bütün kişilere adilane davranmak; ırk, din, cinsiyet, yaş, milliyet, cinsi tercih, cinsiyet kimliği, veya cinsiyet ifadesi üzerinden ayırımcılık yapma durumuna girişmemek;
9. Yanlış veya kötü amaçlı eylemler sonucu kimsenin yaralanması, mülklerinin zarar görmesi, itibarlarının veya istihdamlarının zedelenmesi durumlarının oluşmasından kaçınmak;
10. Meslektaşlara ve yardımcı personele mesleki gelişimlerinde yardımcı olmak ve onları desteklemek.

IEEE Yönetim Kurulu tarafından Ağustos 1990'da onaylanmıştır.

# ÖNSÖZ

"Android Mobil Market Uygulaması" isimli bu çalışma Karadeniz Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü'nde bitirme projesi olarak hazırlanmıştır.

Mobil uygulama geliştirme alanında hem öğrendiklerimizi pekiştirme hem de proje süreci yönetimi açısından değerli tecrübeler kazandığımıza inanıyoruz. Bu alanda ilk defa kapsamlı bir proje yaparak dışarıdan baktığımızda ürkütücü görünen proje arka planlarının aslında ne kadar da üstesinden gelebileceğimiz konular olduğunun farkına vardık. Böyle bir deneyimin bizleri daha ileri ufuklara taşıyacağına inanıyoruz.

En başta danışman hocamız Doç. Dr. Bekir DİZDAROĞLU'na ve desteğini hiçbir zaman esirgemeyen kıymetli ailelerimize teşekkürlerimizi sunarız.

Kürşat ERCAN
Burak ALTUNTAŞ
Trabzon 2021

# İÇİNDEKİLER

# ÖZET

Projenin amacı insanların yetenek ve uzmanlıklarını sergiledikleri, ilanlar açtıkları aynı zamanda girişimcilerin de ihtiyaçları doğrultusunda destek arama amacıyla ilanlar açabildiği ve bu ilanlar vasıtasıyla kullanıcıların iletişim kurabilecekleri bir platform oluşturmaktır.

Platformun geliştirilme motivasyonu insanların yeteneklerini sergileyebildikleri, hobilerine maddi karşılıklar bulabildikleri, istedikleri ve sevdikleri işi, ne zaman, nerede ve kiminle yapacaklarına karar verme özgürlüğüne sahip oldukları bir pazar yeri düşüncesinden beslenmektedir.

# 1. GENEL BİLGİLER

## 1.1. Giriş

Firmaların çalışma saatlerindeki esnek olmayan katı kuralları insanların kişisel gelişmelerinde engel, sosyalleşmelerinde olumsuz etken ve üretkenliklerinde performans kaybına neden olabilmektedir. Ayrıca insanların nereden ve nasıl iş bulacakları konusunda yaşadıkları endişeler yüzünden mutsuz oldukları, çalıştıkları yerlerde hak ettikleri değeri göremedikleri, işverenlerin bütçelerini zorlayan giderleri ve doğru çalışanı bulamadıkları bir zaman diliminde yaşamaktayız.

İnsanlar hem sevdikleri işi yapmak hem de mutlu olmayı isterler. Geliştirmekte olduğumuz bu proje ile yetenekli insanlara yeteneklerini değerlendirebilecekleri, öğrencilere ve iş arayanlara gelir elde edebilecekleri ve firmaların ihtiyaçlarına uygun fiyat ile hızlı olarak çözümler elde edebilecekleri bir ortam sunmayı hedeflemekteyiz. Geliştirilen bu platform ile kullanıcılara hobileri ile gelir elde etme imkânı, istedikleri ve sevdikleri işi, ne zaman, nerede ve kiminle yapacaklarına karar verme özgürlükleri sağlanmaktadır.

2019 yılında 57 milyon serbest çalışan varken 2020 yılı için yapılan araştırmalarda bu sayı 1,1 milyara ulaşmıştır. Aynı dönemde toplam küresel iş gücü 3.5 milyar civarındadır. Bu da serbest çalışanların, küresel işgücünün yaklaşık %35'ini içerdiği anlamına gelmektedir. Yine aynı araştırma sonucunda serbest çalışanların %68'i çalışma programlarının istedikleri esnekliğe sahip olmamasından dolayı bu çalışma şeklini benimsedikleri ortaya çıkmıştır.[1] Bu istatistikler sonucunda proje vizyonunun değeri çok net olarak görülebilmektedir.

## 2. YAPILAN ÇALIŞMALAR

### 2.1. FİREBASE ENTEGRASYONU

Firebase konsolu üzeninden yeni bir proje oluşturulup geliştirilmekte olan uygulama projeye eklenmiştir. Ekleme yapıldıktan sonra konsol üzerinden indirilen Firebase Android yapılandırma dosyası (google-services.json) uygulama içerisine eklenip App level Gradle içerisinde eklenti uygulanmıştır.

```
dependencies {
    classpath 'com.google.gms:google-services:4.3.4'
}
```

Uygulamada kullanılacak olan Firebase'in Authentication, Cloud Firestore ve Storage hizmetlerine ait SDK'ları uygulama düzeyinde Gradle içerisine eklenmesi gerekmektedir. Bu konuda Firebase'in Malzeme Listesi olarak adlandırdığı BoM hizmeti ile SDK'ların sürümlerini otomatik olarak güncellemek mümkündür.

```
implementation platform('com.google.firebase:firebase-bom:26.2.0')
implementation 'com.google.firebase:firebase-auth:20.0.2'
implementation 'com.google.firebase:firebase-firestore:22.0.2'
implementation 'com.google.firebase:firebase-storage:19.2.1'
```

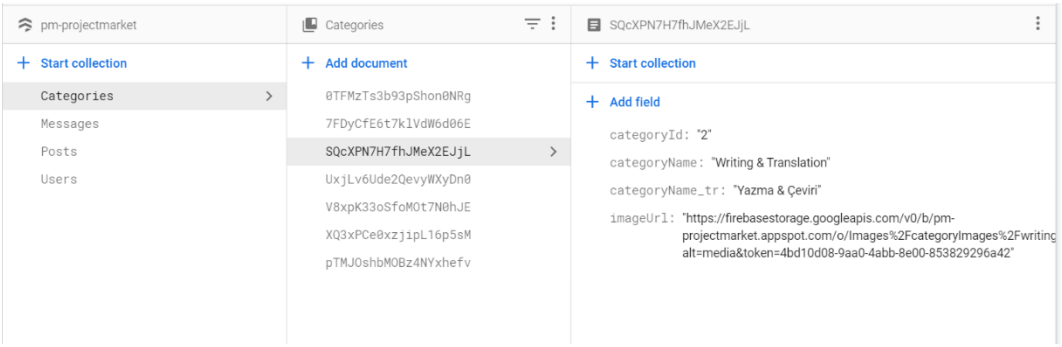
### 2.2.VERİTABANI İLİŞKİLERİNİN OLUŞTURULMASI

Veri tabanı olarak Cloud Firestore kullanılmaktadır. Firestore büyük verileri saklamak ve hızlıca sorgulamak amacıyla geliştirilmiş doküman tabanlı Nosql bir veritabanıdır.

Firestore'un esnek yapısı uygulama için veritabanı seçilmesinde büyük öneme sahiptir. Veriler dokuman olarak saklanır. Aynı sınıfa ait dokümanların oluşturduğu topluluğa koleksiyon adı verilir. Ayrıca dokümanlarında kendi içlerinde koleksiyonlara sahip olabilmesi Firestore'un avantajlarından birisidir. Uygulama için veritabanı olarak seçilmesindeki diğer önemli özellikleri ise veritabanında değişiklik olduğunda otomatik senkronizasyon yaparak uygulama içerisinde verilerin kendiliğinden güncellenmesini sağlaması ve internet olmadığında cihazın lokalinden verileri okuyup yazabilmesidir [2].

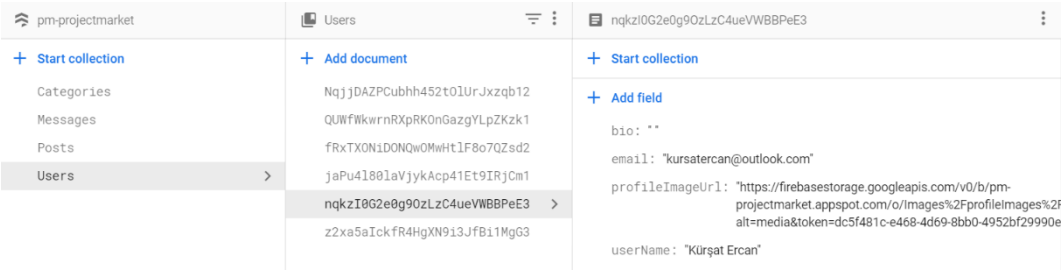
Proje kapsamında dört tane koleksiyon iki tane de alt koleksiyon kullanılmıştır. Ana koleksiyonlar kategoriler, mesajlar , postlar ve kullanıcılardan oluşmaktadır. Alt koleksiyonlar ise mesajlar içerisindeki dokümanlara ait mesaj detayları koleksiyonu ve postlar koleksiyonundaki dokümanlara ait yorumlar koleksiyonlarıdır.

Kategoriler koleksiyonu dokümanları dört fielddan oluşmaktadır. Bunlar id, Türkçe ve İngilizce isimleri ve kategori görselinin url'sidir. Koleksiyon şeması Görsel-1 de verilmiştir.



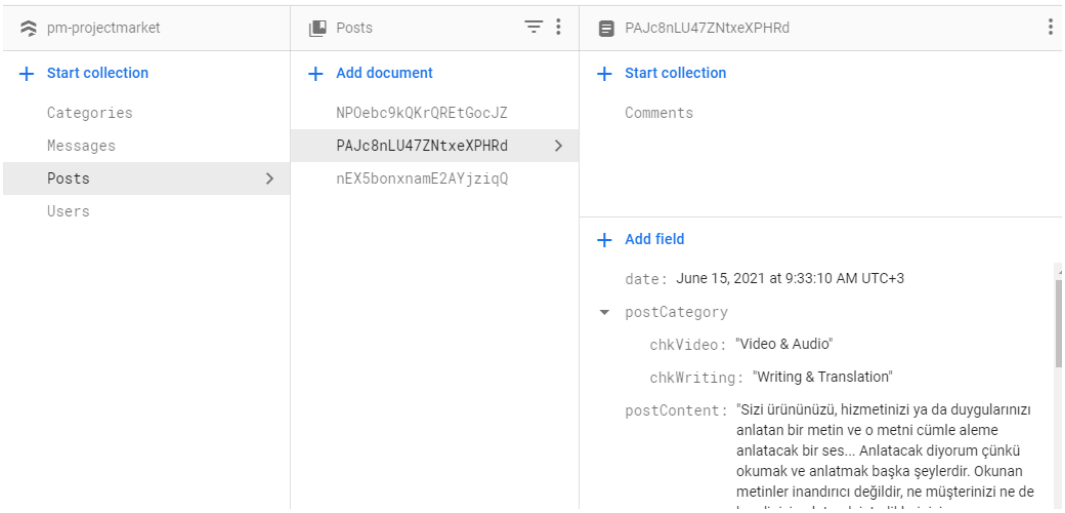

Görsel-1

Kullanıcılar koleksiyonu dokümanları 4 fielddan oluşmaktadır. Bunlar kullanıcı adı, e-posta adresi, biyografisi ve kullanıcı profili görselinin url'sidir. Biyografi bilgisinin uygulama içerisinde kullanılmadığı ancak ileride kullanılması öngörüldüğü için eklendiğini belirtmekte fayda vardır. Koleksiyon şeması Görsel-2'de verilmiştir.



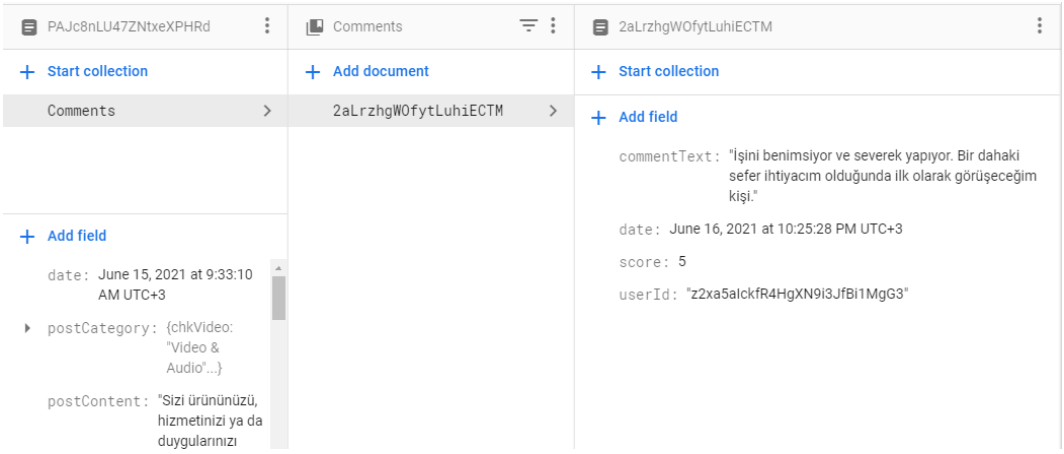

Görsel-2

Postlar koleksiyonu dokümanları tarih, kategori bilgisi, içerik, başlık, görsel urlsi, fiyat, puan ve paylaşan idsi olmak üzere sekiz field ve bir tanede yorumlar koleksiyonu içermektedir. Koleksiyona ait yapının bir kısmını içeren görüntü Görsel-3'de verilmiştir.



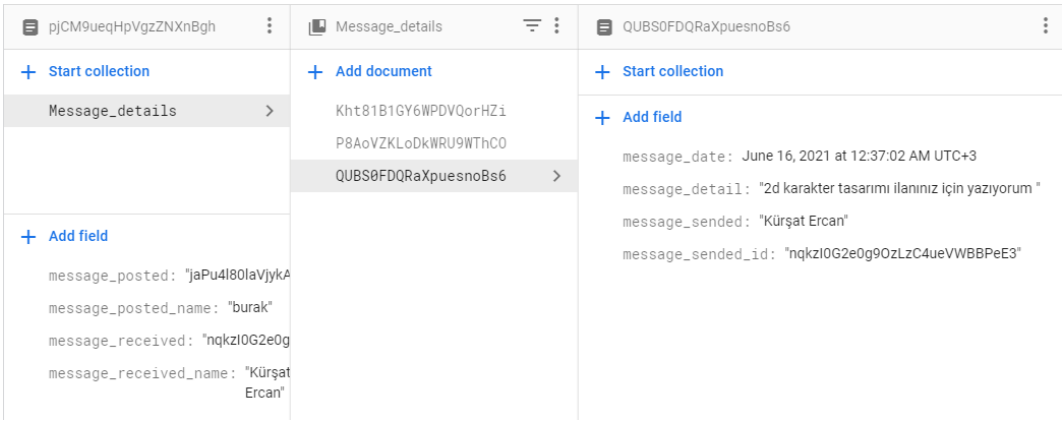

Görsel-3

Yorumlar koleksiyonu ise yorum içeriği, puan bilgisi, yorum yapan kişinin idsi ve tarih bilgileri içeren dokümanlardan oluşmaktadır. Koleksiyona ait görüntü Görsel-4'de verilmiştir.




Görsel-4

Mesajlar koleksiyonu dokümanları ise gönderen ve alıcının idleri, kullanıcı adları ve detaylar alt koleksiyonundan oluşmaktadır. Detaylar alt koleksiyonu ise tarih, içerik, gönderen adı ve gönderenin idsi ni içeren dokümanlardan oluşmaktadır. Koleksiyonların yapısını içeren görsel Görsel-5'de verilmiştir.




Görsel-5

## 2.3. SPLASH EKRANI TASARIMI VE KODLANMASI

Splash ekranı Intro Activity içerisinde oluşturulmuştur. Burada kısa süreli animasyon olacak şekilde uygulamanın logosu ve ismi bulunmaktadır. Splash ekranı Görsel-6'da verilmiştir.


Görsel-6

Animasyon, *animate* fonksiyonu kullanılıp öncelikle ilgili komponentlerin opaklığını 0 olarak ayarlayıp ardından belirlenen sürede opaklığı 1 yaparak sağlanmıştır. İlgili kod parçası aşağıda mevcuttur.

```
pm_logo.animate().alpha(0f).setDuration(0);
projectMarket.animate().alpha(0f).setDuration(0);

pm_logo.animate().alpha(1f).setDuration(1000)
       .setListener(new AnimatorListenerAdapter() {
    @Override
    public void onAnimationEnd(Animator animation) {
        projectMarket.animate().alpha(1f).setDuration(800);
    }
});
```

## 2.4.İNTERNET KONTROLÜ

İnternet kontrolü de Intro Activity içerisinde sağlanmıştır. Intro Activity uygulama açılışında ilk başlatılan activity olduğu için burada internet kontrolü yapılıp eğer internet yoksa kullanıcıya uyarı mesajı verilmektedir. İnternet bağlanmadan mesaj geçilirse ve eğer uygulama kapatılmadan önce kullanıcı oturumu kapatmamışsa önceden önbelleğe alınan eski postlar ve yorumumlar incelenebilir. İnternet bağlandığı taktirde ise eğer daha önceden oturum açılmışsa kullanıcı Main Activity'ye açılmamışsa Login Activity'ye yönlendirilmektedir.

İnternet kontrolü şu şekilde sağlanmıştır.

```
public boolean isOnline() {
    ConnectivityManager conMgr = (ConnectivityManager) getApplicationContext()
            .getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo netInfo = conMgr.getActiveNetworkInfo();

    if(netInfo == null || !netInfo.isConnected() || !netInfo.isAvailable()){
        return false;
    }
    return true;}
```

### 2.5. DİL İÇİN KONFİGÜRASYON AYARI

İlerleyen sayfalarda dil seçeneğinden bahsedilecektir. Ayarlar sayfasından dil değiştirebilme imkânı mevcuttur. Uygulamada ön tanımlı dil İngilizce olarak ayarlanmıştır. Ancak dil tercihi yapıldıktan sonra uygulama her açıldığında dilin ayarlanması için dil tercihi *Shared Preferences* ile *"lang"* keyi hafızaya kaydedilmiştir.

Hafızaya yazılan dil tercihi aşağıda verilen kod parçası ile okunup konfigürasyon tercihe göre ayarlanmaktadır.

```
SharedPreferences sharedPreferences =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
String lang = sharedPreferences.getString("lang","en");

Locale myLocale = new Locale(lang);
Resources res = getResources();
DisplayMetrics dm = res.getDisplayMetrics();
Configuration conf = res.getConfiguration();
conf.locale = myLocale;
res.updateConfiguration(conf, dm);
```

### 2.6. KAYIT VE GİRİŞ EKRANI

Kayıt sırasında kullanıcıdan e mail, kullanıcı adı ve parola bilgilerinin girmesi beklenmektedir. Ayrıca parolanın en az 6 karakterden oluşması koşulu vardır. Bu koşulların sağlanmaması durumunda Toast mesajı ile kullanıcıya uyarı verilmektedir. Tüm bu gereksinimler giriş butonuna tıklandıktan sonra *signUpClicked* fonksiyonu içerisinde kontrol edilmektedir. Koşullar sağlandıktan sonra ise Firebase Authentication üzerinde kullanıcı oluşturulmak üzere *registerUser* fonksiyonu çağrılmaktadır. Fonksiyon içeriği aşağıda verilmiştir.

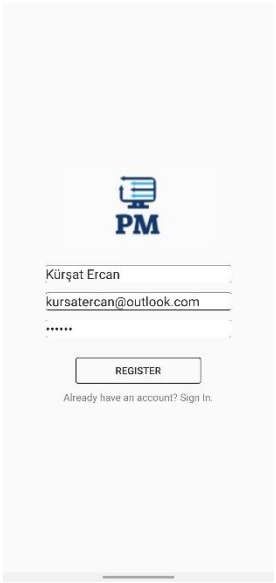
```
private void registerUser(String userName, String email, String password){
    FirebaseAuth firebaseAuth = FirebaseAuth.getIntance();
    firebaseAuth.createUserWithEmailAndPassword(email, password)
.addOnSuccessListener(authResult -> {
        createUser(userName, email);
    }).addOnFailureListener(e -> {
        Toast.makeText(RegisterActivity.this, e.getLocalizedMessage(),
Toast.LENGTH_LONG).show();
        progressDialog.dismiss();
    });}
```

Kullanıcı oluşturma başarılı olursa success listener içerisinden *createUser* fonksiyonu çağrılmaktadır. Bu fonksiyonla ise Firestore Database içerisinde oluşturulan User koleksiyonunda kullanıcıya ait bir doküman oluşturulur ve kullanıcının tüm bilgileri burada depolanır. İlgili kod parçası aşağıda verilmiştir.

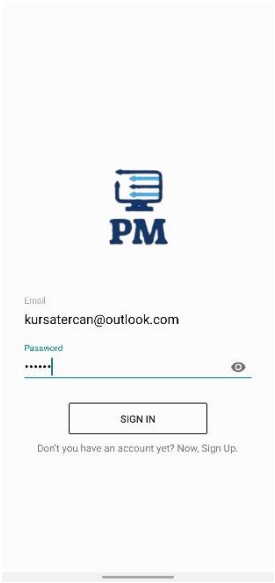
```
String userId = (String) firebaseAuth.getCurrentUser().getUid();
Map<String, Object> user = new HashMap<>();
    user.put("userId",userId);
    user.put("userName", userName);
    user.put("email", email);
    user.put("bio", "");
    user.put("profileImageUrl",placeHolder);
    db.collection("Users").document(userId).set(user)
            .addOnSuccessListener(aVoid -> {
                Intent intent = new Intent(RegisterActivity.this,
                          LoginActivity.class);
             intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK);
             startActivity(intent);
             finish();
             }).addOnFailureListener(e -> {Toast.makeText(RegisterActivity.this,
                    e.getLocalizedMessage(),Toast.LENGTH_LONG).show();
             });
}
```

Success listener içerisinde de görülebileceği gibi kullanıcı bilgileri database'e kaydedildikten sonra kullanıcı giriş ekranına yönlendirilmektedir. Giriş ekranında ise kullanıcıdan e mail ve parolası ile giriş yapması beklenmektedir.

Kayıt ekranı Görsel-7'de giriş ekranı ise Görsel-8'de verilmiştir.




Görsel-7




Görsel-8

## 2.7. BOTTOM NAVIGATION VIEW İLE MENÜ YAPISI

Kullanıcının uygulamaya giriş yapmasıyla uygulamanın temel activity'si olan Main Acticity başlatılmaktadır. Ana menümüz ve profil, kategoriler, ilanlar ve mesaj ekranları bu aktivity'ye bağlı fragmentler olarak tasarlanmıştır.

Menü aracılığı ile fragmentler arasında geçiş sağlanmaktadır. Menü üzerinde home, post, mesaj ve profil buttonları olmak üzere 4 adet button vardır. Postbutonu activity'i diğer butonlar ise fragmentleri başlatmaktadır.

Uygulamaya girildikten sonra kullanıcıyı kategoriler fragment'i karşılamaktadır. Bunun için *onCreate* içerisinde main fragment ile categories fragment'i replace edilmiştir.

```
getSupportFragmentManager().beginTransaction()
        .replace(R.id.mainFragment,new CategoriesFragment()).commit();
```

Menü aracılığı ile fragmentler arasında gezinme ise *navigation item select listener* içerisinde kurulan aşağıdaki yapı ile sağlanmaktadır.

```
@SuppressLint("NonConstantResourceId")
private final BottomNavigationView.OnNavigationItemSelectedListener
navigationItemSelectListener =
        item -> {
            id=item.getItemId();
            switch(id){
                case R.id.nav_home:
                    selectedFragment = new CategoriesFragment();
                    break;
                case R.id.nav_post:
                    selectedFragment = null;
                    Intent intent = new Intent(MainActivity.this,PostActivity.class);
                    startActivity(intent);
                    break;
                case R.id.nav_message:
                    selectedFragment = new MessageFragment();
                    break;
                case R.id.nav_profile:
                    selectedFragment = new ProfileFragment();
                    break;
                default:
                    break;
            }
           if(selectedFragment != null){
              getSupportFragmentManager().beginTransaction()
                .replace(R.id.mainFragment,selectedFragment).commit();
    }
    return true;
};
```

### 2.8. KATEGORİLER MENÜSÜ

Kategori seçimi için menü yapısı bölüm 2.7'de bahsedilen Categories Fragment içerisindedir. Bu ekranda kullanıcıdan ilgilendiği kategoriyi seçmesi beklenmektedir. Seçimine göre bir sonraki ekranda ilanlar filtrelenmiş olarak sıralanacaktır.

Kategori itemları, kategori görseli ve kategori ismi olmak üzere iki kompanent içerir. İtemları gösterebilmek için *recycler view* kullanılmıştır. Veri tabanından çekilen isim ve görsel url'si adapter kullanılarak recycler view'a bağlanmıştır. Adapter kurulumu aşağıdaki kod parçasıyla sağlanmıştır.

```
RecyclerView recyclerView = view.findViewById(R.id.rv_categories);
recyclerView.setHasFixedSize(true);
recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));

CategoryRecyclerAdapter adapter;
adapter = new CategoryRecyclerAdapter(categoryList_db,
      categoryImageUrlList_db,categoryId_db);
recyclerView.setAdapter(adapter);
```

Adapter kurulduktan sonra veritabanından categori bilgileri *getDataFromDB* fonksiyonu ile aşağıdaki gibi çekilmektedir.

```
public void getDataFromDB() {
    SharedPreferences sharedPreferences = PreferenceManager.
      getDefaultSharedPreferences(getContext());
    String lang = sharedPreferences.getString("lang","en");

    db.collection("Categories").get().addOnCompleteListener(task -> {
        if (task.isSuccessful()) {
            for (QueryDocumentSnapshot doc : task.getResult()) {
                CategoryCard category = doc.toObject(CategoryCard.class);
                if(lang == "en") categoryList_db.add(category.getCategoryName());
                else categoryList_db.add(category.getCategoryName_tr());
                categoryImageUrlList_db.add(category.getImageUrl());
                categoryId_db.add(category.getCategoryId());
            }
            adapter.notifyDataSetChanged();
        }
    });
}
```

Veritabanında kategori isimleri Türkçe ve İngilizce olarak iki ayrı field olarak tutulur. Kullanıcının dil tercihini shared preferences ile bellekte tuttuğumuzdan bahsetmiştik. Kategori isimlerini de yine tercihe göre ayrıt edebilmek için adaptere gönderilen kategori ismi koşul bloklarıyla dile göre seçilmiştir.

Kategoriler veritabanındaki id numaralarına göre yerleştirilmişlerdir. Bu nedenle seçilen kategori tıklandığında holder içerisinde adapter pozisyonunu kullanılarak hangi kategorinin seçildiğine karar verilir. Elde edilen id bilgisi ilanları listelemek için Feed Fragment başlatılırken filtre numarası olarak fragmente verilir.

```
cHolder.category_item.setOnClickListener(v -> {
    Bundle filter = new Bundle();

filter.putString("filterNum",categoryIdList.get(cHolder.getAdapterPosition()));

    FeedFragment feedFragment = new FeedFragment();
    feedFragment.setArguments(filter);

    ((FragmentActivity) v.getContext()).getSupportFragmentManager()
            .beginTransaction()
            .replace(R.id.mainFragment, feedFragment).commit();
});
```

Kategori ekranı Görsel-9'da verilmiştir.


Görsel-9

### 2.9. İLANLAR SAYFASI

İlanlar sayfası kullanıcının kategori seçimine göre filtrelenmiş ilanları listeleyen arayüzdür. *Feed Frangment* içerisindedir. Kullanıcı kategori seçimi yaptığında tıkladığı kategoriye ait id numarasını filtre olarak Feed Fragmente ilettiğini söylemiştik. Burada öncelikle filtre numarası bundle içeisinden alınmaktadır. İlgili kod parçası aşağıda verilmiştir.

```
Bundle bundle = this.getArguments();
if(bundle != null){ filterNum = bundle.getString("filterNum","0"); }
```

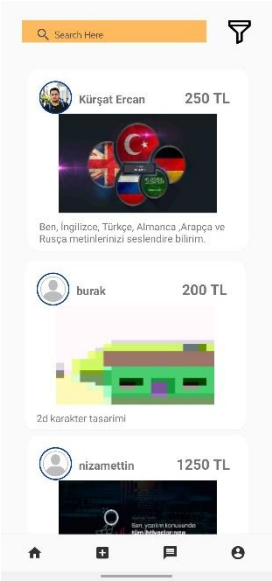
Filtre numarası alındıktan sonra *getDataFromDB* fonksiyonu içerisinde whereEqualTo sorgusuyla veritabanından postlar filtrelenerek çekilmektedir. Çekilen postlar Adapter kullanılarak recycler view içerisinde gösterilmektedir. Postların veritabanından çekilmesi için kullanılan kodlar aşağıda verilmiştir.

```
db.collection("Posts").whereEqualTo("categoryId",filterNum).
        orderBy("date",Query.Direction.DESCENDING)
        .addSnapshotListener((value, error) -> {
    if(value != null){
        for(DocumentSnapshot doc : value.getDocuments()){
            Post post = doc.toObject(Post.class);
            assert post != null;
            DocumentReference docRef = db.collection("Users")
                    .document(post.getUserId());

            docRef.get()
                    .addOnSuccessListener(new OnSuccessListener<DocumentSnapshot>() {
                @Override
                public void onSuccess(DocumentSnapshot documentSnapshot) {
                    User user = documentSnapshot.toObject(User.class);
                    ppost.add(new Post(post.getUserId(),
                            post.getUserName(),
                            post.getPrice(),
                            post.getTitle(),
                            post.getPostContent(),
                            post.getPostImageUrl(),
                            post.getScore(),
                            user.getProfileImageUrl(),
                            doc.getId()));
                    postRecyclerAdapter.notifyDataSetChanged();
                }
            });
        }
    }
});
```

Postlar sorgulanıp listelendikten sonra kullanıcı posta tıkladığında doküman idsi token olarak detaylar ekranına gönderilir. İlgili kodlar aşağıda mevcuttur.

```
public void onMessageClick(int position) {

    Bundle args = new Bundle();
    args.putString("token", ppost.get(position).getToken());
    PostDetailsFragment postDetailsFragment = new PostDetailsFragment();
    postDetailsFragment.setArguments(args);
    assert getFragmentManager() != null;
    postDetailsFragment.show(getFragmentManager(),"My Dialog");
}
```

İlanlar sayfasına ait görüntü Görsel-10’da verilmiştir.




Görsel-10

## 2.10. İLAN DETAYLARI VE İLAN YORUMLARI

İlan detayları *Post Details Fragment* üzerinde gösterilmektedir. Bu fragment ise iki tane child fragmente sahiptir. Bunlar detayların gösterildiği *Details Fragment* ve ilgili posta gelen yorumların gösterildiği *Comment Fragmenttir*. Fragment üzerinde bulunan tablayout ve view pager sayesinde iki fragment arasında kaydırılarak geçiş sağlanabilmektedir.

Tablayout ve view pager kurulumu ve adapter bağlantısı aşağıda verilmiştir.

```
TabLayout tabs = view.findViewById(R.id.tabLayout);
ViewPager viewPager = view.findViewById(R.id.view_pager);

viewPager.setAdapter(new
DetailsFragmentPageAdapter(getChildFragmentManager(),getContext(),args));
tabs.setupWithViewPager(viewPager);
```

Adapter içerisinde fragmentler arası geçiş ise *getItem* fonksiyonu ile sağlanmaktadır.

```
@NonNull
@Override
public Fragment getItem(int position) {
    Fragment fragment = new DetailsFragment();
    if(position == 0) {
        fragment = new DetailsFragment();
    }
    else if(position == 1) {
        fragment = new CommentFragment();
    }
    fragment.setArguments(data);
    return fragment;
}
```

Fragmentler arasında geçiş tamamlandıktan sonra sırada posta ait detayların gösterildiği *Details Fragment* var. Token'ın ilanlar sayfasında posta tıklanıldığında post details fragmente gönderildiğinden bahsedilmişti. Details fragmentte ilana ait bilgilerin çekile bilmesi için öncelikle bu token alınmalıdır. Tokenın nasıl alındığı ve Firestore bağlantıları aşağıdaki kodda verildiği gibi sağlanmıştır.

```
Bundle args = this.getArguments();
token=args.getString("token");
FirebaseFirestore db = FirebaseFirestore.getInstance();
DocumentReference docRef = db.collection("Posts").document(token);
```

İlgili posta ait bilgiler docRef.get() komutuyla veritabanından ilgili doküman getilir ve ait oldukları komponentlere set edilerek kullanıcıya bilgiler gösterilmektedir. Bu bölümün devamında post başlığı, paylaşanın adı ve fiyat gibi bilgilerden ziyade diğer kullanıcıların verdikleri puan değerleri ile postun toplam puanını hesaplamayı ve ilan görselini ilgili kompanente bağlama gösterilecektir.

Posta ait bütün yorumlarda puan değerleri olduğundan bahsedilmişti. Postun genel puanı ise tüm yorumlardaki puanların toplamı bölü toplam yorum sayısı olarak hesaplanmaktadır. Hesaplama ve rating barda gösterimi aşağıda verilen kodlar ile gerçekleştirilmiştir.

```
Post post = document.toObject(Post.class);
//get PostScore
db.collection("Posts/"+document.getId()+"/Comments")
      .addSnapshotListener((value, error) -> {
        if(value != null){
          for(DocumentSnapshot doc : value.getDocuments()){
            score+=doc.getLong("score");
            counter++;
          }
          totalScore=0.0f;
          if(counter==0){
            totalScore = 0;
          }
          else{
            totalScore=score/counter;
          }
          ratingBar_post.setRating(totalScore);
        }
      });
```

Posta ait görselin url'si ise Picasso kütüphanesi kullanılarak aşağıdaki şekilde image view'a bağlanmaktadır.

```
Picasso.get().load(post.getPostImageUrl())
      .resize(postImage.getWidth(),postImage.getHeight())
      .into(postImage);
```

Kullanıcının kendi postuna yorum yapamama şartı bulunmaktadır. Bunu sağlamak için post sahibi ile anlık kullanıcının uid'leri karşılaştırılır. Eğer aynı ise yorum ve puan verememesi için *ratingBar, commentText* ve buton gizlenmektedir.

```
if(post.getUserId().equals(user.getUid())){
    commentText.setVisibility(View.GONE);
    ratingBar_comment.setVisibility(View.GONE);
    addCommentButton.setVisibility(View.GONE);
}
```

Eğer postu inceleyen ve postu yayınlayan aynı kişi değil ise bu komponentler gizlenmez ve kullanıcıdan istediği taktirde yorumunu yapıp yorum yap butonuna tıklaması beklenir. Bu sayede yorum ve oy veritabanına kaydedilir. İlgili kod parçası aşağıda mevcuttur.

```
addCommentButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        float score = ratingBar_comment.getRating();
        if(score!=0 && commentText.getText()!=null){
            HashMap<String, Object> commentData = new HashMap<>();
            commentData.put("userId", FirebaseAuth
                    .getInstance().getCurrentUser().getUid());
            commentData.put("date",new Timestamp(new Date()));
            commentData.put("commentText",commentText.getText().toString());
            commentData.put("score",score);
            db.collection("Posts/"+token+"/Comments")
                    .add(commentData);
            Toast toast = Toast.makeText(getContext(),
                    R.string.your_comment_has_been_sent, Toast.LENGTH_SHORT);
            toast.show();
        }
    }
});
```

Profili ziyaret et butonuna tıklanıldığında ise profile fragment başlatılarak ilan sahibinin profili gösterilmektedir.

```
profileClick.setOnClickListener(v -> {
    Bundle args1 = new Bundle();
    args1.putString("userId", post.getUserId());
    args1.putString("userName", post.getUserName());
    ProfileFragment profileFragment = new ProfileFragment();
    profileFragment.setArguments(args1);
    FragmentManager fragmentManager = getActivity().getSupportFragmentManager();
    FragmentTransaction ft = fragmentManager.beginTransaction();
    ft.replace(R.id.mainFragment, profileFragment, "tag");
    ft.addToBackStack(null).commit();
});
```

İlan detayları sayfasıyla ilgili önemli özellikler anlatıldıktan sonra sırada yorumlar sayfası var. Yorumların veritabanından çekilebilmesi için de token bilgisinin alınması lazım. Token'ın nasıl alındığı bölümün başlarında gösterilmişti. Comments Fragment de child fragment olduğu için yie aynı yöntemle bundle içerisinde token bilgisi alınabilmektedir.

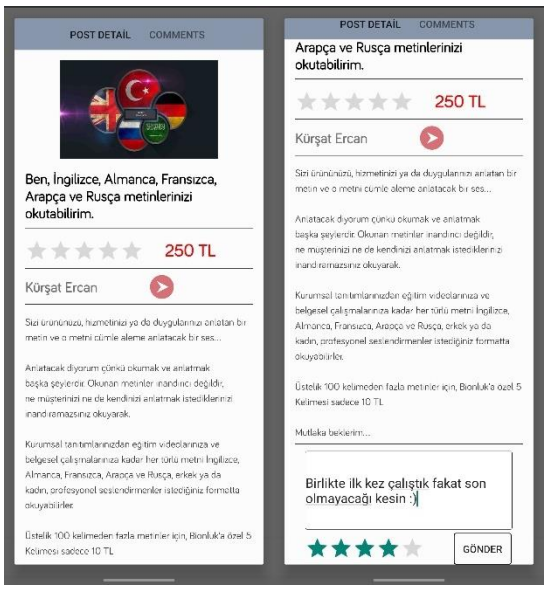
Yorumlar veritabanından aşağıda verilen getPostComments fonksiyonu ile çekilmektedir.

```
public void getPostComments(String token){
    db.collection("Posts/"+token+"/Comments")
            .addSnapshotListener(new EventListener<QuerySnapshot>() {
                @Override
                public void onEvent(@Nullable QuerySnapshot value,
                                    @Nullable FirebaseFirestoreException e) {
                    if (e != null) {
                        Log.w(TAG, "Listen failed.", e);
                        return;
                    }

                    for (QueryDocumentSnapshot document : value) {
                        String date=getDate((Timestamp)document.get("date"));
                        comment.add(new Comment(
                                document.get("commentText").toString(),
                                date,document.getLong("score"),document.getId()));
                    }
                    commentRecyclerAdapter.notifyDataSetChanged();
                }
            });
```

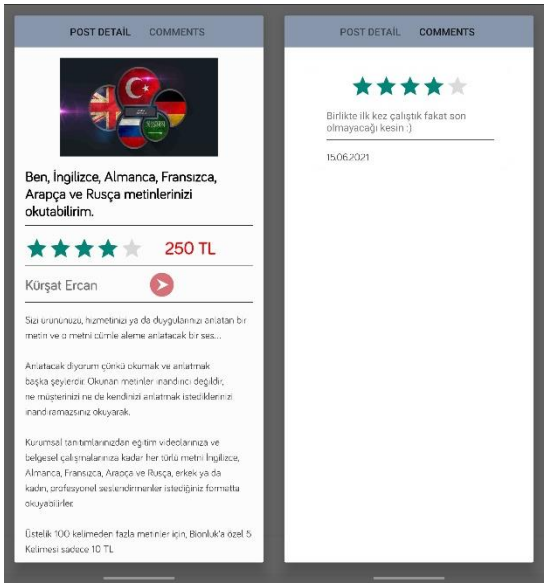
Veritabanından çekilen veriler adapter yardımıyla ilgili kompanentlere bağlanmıştır. Koddanda anlaşılacağı üzere veritabanından alınan tarih bilgisi dd.mm.yyyy formatında gösterilmek için getDate fonksiyonunda işlenmiştir. Fonksiyon içeriği aşağıda verilmiştir.

```
private String getDate(Timestamp milliSeconds) {
    // Create a DateFormatter object for displaying date in specified
    // format.
    SimpleDateFormat formatter = new SimpleDateFormat("dd.MM.yyyy");
    // Create a calendar object that will convert the date and time value in
    // milliseconds to date.
    Calendar calendar = Calendar.getInstance();
    calendar.setTimeInMillis( milliSeconds.getSeconds()*1000);
    return formatter.format(calendar.getTime());
}
```

Post detayları ekranı Görsel-11'de yorum yapıldıktan sonra güncellenmiş hali ile detaylar ve yorumlar ekranları ise Görsel-12'de verilmiştir.




Görsel-11




Görsel-12

## 2.11. İLAN PAYLAŞIMI

İlan paylaşımı ekranında kullanıcıdan ilanı daha dikkat çekici hale getirecek bir görsel, ilan başlığı, ilan içeriğinin detaylı açıklaması, ilanın ait olduğu kategoriler ve fiyat bilgisini girmesi beklenmektedir.

Uygulama üzerinden galeriye erişebilmek ve okuma yapabilmek için depolamaya erişim izninin verilmesi gerekmektedir. Bunun için *Manifest* dosyasında aşağıda verilen komut kullanılır.

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Kullanıcı resim seçmek için görsel yükle butonuna tıkladığında galerinin açılıp resim tipinde dosyalar yükleyebilmesi için intent oluşturulur.

```
static final int SELECTED_IMAGE=1;

public void uploadImage(View v){
    Intent intent =new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_GET_CONTENT);
    startActivityForResult(intent,SELECTED_IMAGE);
}
```

Seçilen resmin uri'sini alabilmek için *onActivityResult* methodu kullanılır. Uri bilgisi metot parametrelesi olan intent içerisinde *getData* fonksiyonu ile alınabilmektedir. Uri bilgisi alınan görsel image view içerisinde açılmıştır.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data)
{
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode == SELECTED_IMAGE && resultCode == RESULT_OK) {
        assert data != null;
        if (data.getData() != null) {
            imageData = data.getData();

            Picasso.get().load(imageData)
                    .resize(postImage.getWidth(),postImage.getHeight())
                    .into(postImage);
        }
    }
}
```

Kullanıcı görseli seçtikten sonra ilan başlığı, içeriği, fiyatı ve kategorileri seçtikten sonra paylaş butonuna tıkladığında öncelikle zorunlu bilgilerin girilip girilmediği kontrol edilir ardından tüm bilgiler veritabanına kaydedilir.

Bir post birden fazla kategoriye ait olabileceğinden kategori bilgileri dizi içerisine kaydedilmiş ve veri tabanında da dizi olarak tutulmuştur. Ayrıca seçildikten sonra image viewda gösterilen görsel bitmap ile byte şeklinde alınmıştır. Storage'a yüklenmeden önce JPEG formatında kalite değeri 25 olacak şekilde sıkıştırılmıştır.

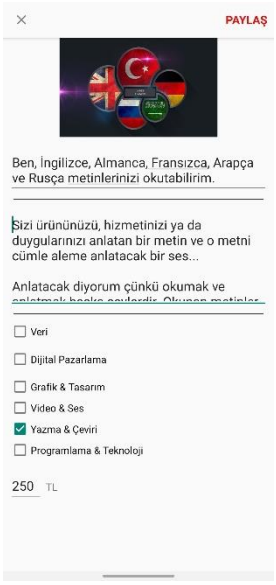
Storage referansı alma ve görselin sıkıştırıp storage'a yüklenmesi aşağıda verilen kod parçasıyla sağlanmıştır.

```
StorageReference fileReference= storageReference
        .child("postImages/"+System.currentTimeMillis());
Bitmap bmp = MediaStore.Images
        .Media.getBitmap(getContentResolver(), imageData);
ByteArrayOutputStream baos = new ByteArrayOutputStream();
bmp.compress(Bitmap.CompressFormat.JPEG, 25, baos);
byte[] data = baos.toByteArray();
UploadTask uploadTask2 = fileReference.putBytes(data);
```

Yükleme sonrası upload task geri dönmektedir. Upload task için success listener çağırılarak success durumunda posta ait tüm bilgiler veritabanında *Posts* koleksiyonu altında oluşturulan yeni doküman içerisine kaydedilir. İlgili kod parçası aşağıda verilmiştir. Ancak kol kalabalığı yapmamak adına rapor içerisinde koşul durumları verilmemiştir.

```
uploadTask2.addOnSuccessListener(taskSnapshot -> {
    fileReference.getDownloadUrl().addOnSuccessListener(uri -> {
        String downloadUrl = uri.toString();
        HashMap<String,Object> postData = new HashMap<>();
        postData.put("date",new Timestamp(new Date()));
        postData.put("postImageUrl", downloadUrl);
        postData.put("price",priceText.getText().toString());
        postData.put("title",title.getText().toString());
        postData.put("postContent",postContent.getText().toString());
        postData.put("userId", user.getUid());
        postData.put("userName",userName);
        postData.put("postCategory",array);
        postData.put("score","0");

        cfr = db.collection("Posts");
        cfr.add(postData);
        startActivity(new Intent(PostActivity.this, MainActivity.class));
        finish();
    });
});
```

İlan paylaşım sayfası Görsel-13'de verilmiştir.




Görsel-13

## 2.12. MESAJ KUTUSU VE MESAJLAŞMA

Mesaj kutusu *Message Fragment*'i olarak ayarlanmıştır. Burada kullanıcının daha önceden mesajlaştığı kullanıcılarla tekrar mesajlaşabilmek için mesajlaşma arayüzüne yani *Messages Activity*'e yönlendirir.

Kullanıcıya ait daha önceden bir mesajlaşma aktivitesi oluşup oluşmadığının kontrolü, Messages koleksiyonundan *DocumentSnapshot* ile çekilen dokümanların for döngüsü ile taranarak alıcı veya gönderen tarafta kullanıcının olduğu bir dokumanın varlığını kontrol ederek sağlanır. İlgili koşul bloğu aşağıda verilmiştir.

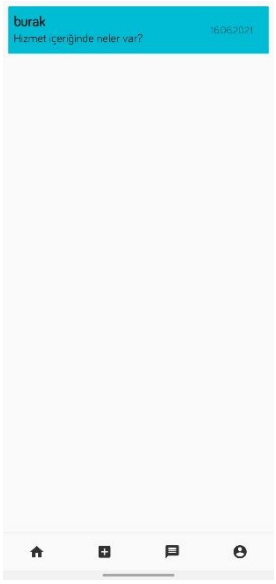
```
if(doc1.get("message_received").equals(user.getUid()) ||
        doc1.get("message_posted").equals(user.getUid()))
```

Koşulun sağlanması ile alınan karşı taraftaki kullanıcının adı, son mesaj içeriği ve tarih bilgisi adapter ile kullanıcıya gösterilir. Bu işlemler rapor içerisinde çokça gösterildi için veri tabanından verileri alma işlemlerinin gösterilmesine gerek duyulmamıştır.

Kullanıcı açmak istediği mesaj kutusuna tıkladığında mesajların gösterilmesi için mesaj click interface'i kullanılmıştır.

```
public void onMessageClick(int position) {
    Intent intent=new Intent(getActivity(), MessagesActivity.class);
    intent.putExtra("token",MessageBox.get(position).getToken());
    intent.putExtra("userN",messageUser);
    startActivity(intent);
}
```

Mesaj Kutusu ekranı Görsel-14'de verilmiştir.




Görsel-14

Kullanıcı herhangi bir mesajı açtığında ise Message Activity başlatılmaktadır. Burada daha önceden olan konuşmalar veritabanından çekilip gösterilmektedir. Ancak veritabanından çekilen verilerin gösterilmesinde mesajı gönderen kişiye göre mesajın ekranın sağına veya soluna dayalı olması önemlidir. Bunu sağlamak sağ ve sol olmak üzere iki farklı view oluşturulmuş göndericiye göre bu viewlar adapter içerisinde layout inflater ile holdera bağlanmıştır.

Mesajların veritabanından çekilmesi işlemleri aşağıda verilen kod satırlarıyla gerçekleştirilmiştir.

```
if(user!=null && token != null) {
    db = FirebaseFirestore.getInstance();
    if(token!=null){
        db.collection("Messages/"+token+"/Message_details")
                .orderBy("message_date", Query.Direction.ASCENDING)
                .addSnapshotListener(new EventListener<QuerySnapshot>() {
                    @Override
                    public void onEvent(@Nullable QuerySnapshot value,
                                        @Nullable FirebaseFirestoreException e){
                        if (e != null) {
                            Log.w(TAG, "Listen failed.", e);
                            return;
                        }
                        MessageSend.clear();
                        for (QueryDocumentSnapshot doc1 : value) {
                            MessageSend.add(
                                    new MessageSend(doc1.get("message_detail")
                                       .toString(),
                              doc1.get("message_sended_id").toString()));
                              }
                   Adapter.notifyDataSetChanged();
                    }
                });
    }
}
```

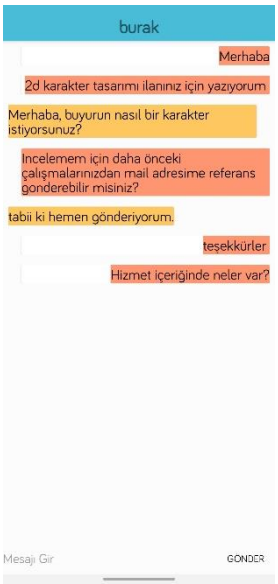
Sağ ve sol viewların bağlanması ise şu kod satırlarıyla gerçekleştirilmiştir.

```
@NonNull
@Override
public MessageSendViewHolder onCreateViewHolder(@NonNull  ViewGroup parent,
                                                int viewType) {
    if(viewType==msg_type_right) {
        View v = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.chat_right, parent, false);
        MessageSendViewHolder evh = new MessageSendViewHolder(v);
        return evh;    }
    else {
        View v = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.chat_left, parent, false);
        MessageSendViewHolder evh = new MessageSendViewHolder(v);
        return evh;
    }}
```

Kullanıcı mesajını yazıp gönder butonuna bastığında ise mesajı veritabanına kaydedilmektedir.

```
btnSend.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Map<String, Object> message = new HashMap<>();
         CollectionReference cfr;
        cfr=db.collection("Messages/"+token+"/Message_details");
        message.put("message_date",new Timestamp(new Date()));
        message.put("message_detail",msgDetail.getText().toString());
        message.put("message_sended",userName);
        message.put("message_sended_id",user.getUid().toString());
        if(msgDetail.getText().toString().isEmpty())
            Toast.makeText(MessagesActivity.this,
                    R.string.you_can_write_your_message,
                    Toast.LENGTH_LONG).show();
        else{
            cfr.add(message);
            msgDetail.setText("");
        }
    }
});
```

Mesajlaşma ekranı Görsel-15'de verilmiştir.




Görsel-15

## 2.13. PROFİL SAYFASI

Profil sayfası fragment sınıfındadır ve adı *Profile Fragment'dir*. Profil fragmenti iki farklı yöntemle kullanılır. İlk olarak bottom navigation menü üzerinden kullanıcının kendi profil sayfasına gidebilmesi için, ikinci olarak da post detayları penceresinden post sahibinin profilini ziyaret et butonuna tıklanılarak profil bilgilerini görebilmek için kullanılır.

Bölüm 2.9'da profili ziyaret et butonu için click listener içeriği verilmişti. Kod tekrar incelenirse profil fragmenti başlatılırken içerisine bundle tipinde argüman atandığı görülecektir. Bundle içerisinde ise kullanıcı id'si bulunmaktadır. Profil fragmenti başlatmak için iki yol olduğunu söylemiştik. İlk yöntemle fragment başlatıldığında bundle null olarak alınmakta ikinci yöntemle fragment başlatıldığında ise bundle içerisinde kullanıcı id'si iletilmektedir. İki yöntem arasındaki bu fark fragmentin nasıl başlatıldığını tespit edebilmek için kullanılmıştır.

```
profileId=FirebaseAuth.getInstance().getCurrentUser().getUid();

Bundle bundle= this.getArguments();
if(bundle==null){
    flbtn.hide();
}
if(bundle!=null) {
    settings.setVisibility(View.GONE);
    if (profileId.equals(bundle.getString("userId"))) {
        flbtn.hide();
        settings.setVisibility(View.VISIBLE);
    }
    profileId = bundle.getString("userId");
    profileName = bundle.getString("userName");
}
```

Koddan anlaşılabileceği üzere kullanıcının kendi profiline bakması durumunda ayarlar için settings butonu görünür olmuş ve mesaj sayfasını açan flbtn (Floating action button) gizlenmiş yani kullanıcının kendisiyle mesajlaşması engellenmiştir.

Profil fragmenti başlatılıp esas kullanıcı kendi profiline mi bakıyor yoksa başka bir kullanıcının profilini mi inceliyor kararı verildikten sonra kullanıcının puanını hesaplamak için *getUserScore* fonksiyonu çağrılır. Puan hesaplaması kullanıcının paylaştığı tüm ilanlara gelen puanlar toplanıp toplam yorum sayısına bölünerek elde edilir. İlgili fonksiyon içeriği aşağıda verilmiştir.

```
private void getUserScore(){
        db.collection("Posts").whereEqualTo("userId",profileId)
       .addSnapshotListener((value, error) -> {
            if(value != null){
                for(DocumentSnapshot doc : value.getDocuments()){
                    db.collection("Posts/"+doc.getId()+"/Comments")
                            .addSnapshotListener(new EventListener<QuerySnapshot>() {
                                @Override
                                public void onEvent(@Nullable QuerySnapshot value,
                                                    @Nullable FirebaseFirestoreException e) {
                                    if (e != null) {
                                        Log.w(TAG, "Listen failed.", e);
                                        return; }
                                    for (QueryDocumentSnapshot document : value) {
                                        score+=document.getLong("score");
                                        counter++; }

                                    totalScore=0.0f;
                                    if(counter==0){ totalScore = 0; }
                                    else{ totalScore=score/counter; }
                                    ratingBar.setRating(totalScore);
                                }
                            });
                } } });
       }
```

Kullanıcı bilgilerini veritabanından çekmek için userInfo fonksiyonu çağrılır. Fonksiyon içeriği aşağıdaki gibidir.

```
private void userInfo(){
    DocumentReference reference = db.collection("Users").document(profileId);
    reference.addSnapshotListener((value, error) -> {
        if(error == null){
            if(getContext() == null){ return; }
            if(value != null){
                User user = value.toObject(User.class);

                try {
                    assert user != null;
                    Picasso.get()
                            .load(user.getProfileImageUrl())
                            .into(imageView_profileImage);
                    textView_UserName.setText(user.getUserName());
                }catch (Exception e){
                    System.out.println("Exception : " + e);
                }
            }
        }
    });
}
```

Profile fragment, post details fragmente benzer yapıda kurulmuştur. Profile fragmenti iki tane child fragmente sahiptir. Bu child fragmentler *Profile Post Fragment* ve *Comment Fragmentleridir*. İki fragment arasındaki geçişler yine aynı şekilde tablayout ve view pager kullanılarak sağlanmıştır. Comment fragment ilan detayları sayfasında yorumları göstermek için kullanılan fragment olduğundan bu fragment bölüm 2.9 da anlatılmıştır. Bu nedenle bu bölümde tekrar anlatılmayacaktır. Profil sayfasında postların gösterildiği fragment ile feed fragment arasındaki tek fark ise eğer kullanıcı kendi profilinde ise istediği posta uzun tıklayarak postu silebilir. Bunun için adapter içerisinde *onLongClick* metodu tanımlanmıştır. Metod içeriği aşağıda verildiği gibidir.

```
public void onLongClick(int position) {
    if(!ppost.get(position).getUserId().equals(user.getUid()))
        return;
    new AlertDialog.Builder(getContext())
            .setMessage(getText(R.string.are_you_sure_you_want_to_delete))
            .setCancelable(false)
            .setPositiveButton(getText(R.string.yes),(dialog, which) ->{
                db.collection("Posts")
                        .document(ppost.get(position).getToken())
                        .delete();
                Toast toast = Toast.makeText(getContext(),
                        R.string.the_post_is_successfully_deleted, Toast.LENGTH_SHORT);
                toast.show();
            })
            .setNegativeButton(getText(R.string.no), null)
            .show();
}
```

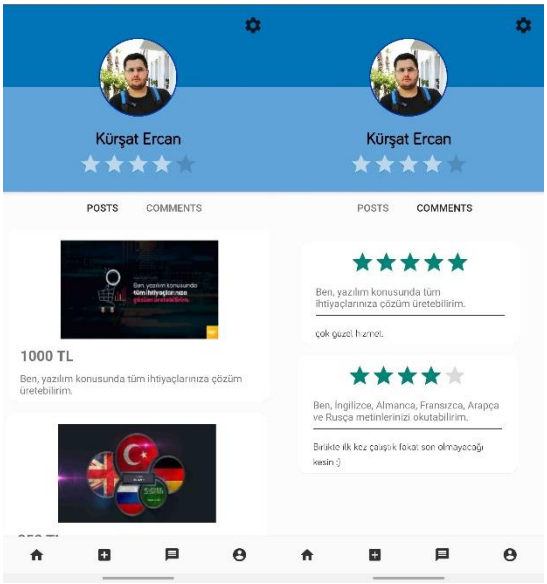
Önceki satırlarda profiline bakılan kullanıcıyla mesajlaşa bilmek için mesajlaşma ara yüzünün açıldığından bahsedilmişti. Veritabanından Message koleksiyonundaki tüm dökümanlar gezilerek iki kullanıcının arasında daha önceden bir mesajlaşma olmuş mu diye sorgu yapılır. Eğer aralarında daha önceden bir mesajlaşma yoksa veritabanında *Messages* koleksiyonunda bir doküman oluşturulur ve *Message Activiti* başlatılır. Eğer önceden bir mesajlaşma olmuş ise gerekli bilgiler intent içerisine koyularak Message Activitiy başlatılır.

```
db.collection("Messages").add(messageSend)
       .addOnSuccessListener(new OnSuccessListener<DocumentReference>() {
       @Override
       public void onSuccess(DocumentReference documentReference) {
       token=documentReference.getId();
       Intent intent=new Intent(getActivity(), MessagesActivity.class);
       intent.putExtra("userId",profileId);
       intent.putExtra("userName",profileName);
       intent.putExtra("token",token);
       startActivity(intent);
       return; }
});
```

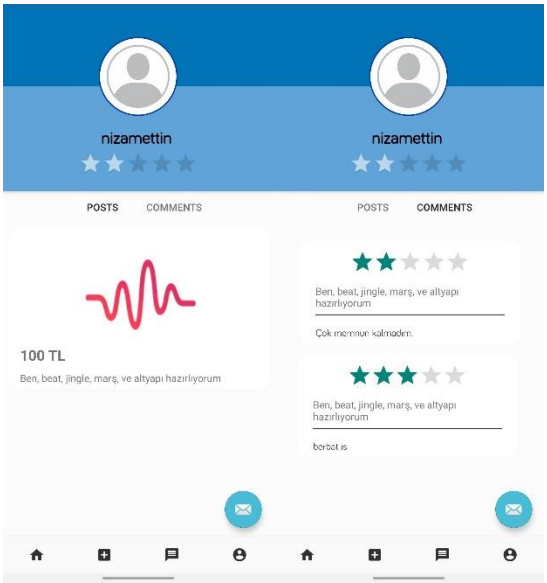
Eğer kullanıcı kendi profiline bakıyorsa ayarlar butonunun erişilebilir olduğundan bahsedilmişti. Kullanıcı ayarlar butonuna tıklar ise ayarlar menüsüne yönlendirilmektedir.

```
settings.setOnClickListener(v -> {
    Intent intent = new Intent(getContext(), SettingsActivity.class);
    startActivity(intent);
});
```

Kullanıcının kendi profiline baktığı anda profil ekranı görüntüsü Görsel-X-16'de, başka kullanıcının profiline baktığı görüntü ise Görsel-Y-17'de verilmiştir.




Görsel-16




Görsel-17

## 2.14. AYARLAR MENÜSÜ

Bu menü aktivite olarak hazırlanmıştır. Menü üzerinde karanlık modu ve bildirimleri açıp kapatabilmek için iki tane switch, profil düzenleme sayfasına gitmek, parola değiştirmek, dil seçebilmek ve uygulamadan çıkış yapabilmek için 4 tane buton bulunmaktadır.

Profili düzenle butonuna basıldığında kullanıcı *Edit Profile Activity'e* yönlendirilir.

```
editProfile.setOnClickListener(v -> {
    Intent intent = new Intent(
            SettingsActivity.this, EditProfileActivity.class);
    startActivity(intent);
});
```

Kullanıcı Change Password butonuna tıklarsa ekranda dialog penceresi açılır ve kullanıcıdan eski parolasını, yeni parolasını ve parolayı doğrulamak için confirm parolasını girmesi istenir.

Firebase Authentication üzerinde kullanıcı parolası değiştirmek için kullanıcı kimliği referansı oluşturulması gerekmektedir. Referans ise e-mail yetki sağlayıcısıyla oluşturulmaktadır. Referans sağlandıktan sonra parola reauthenticate metodu ile değiştirilebilir. İlgili kod bölümü aşağıda verilmiştir.

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
String email = user.getEmail();
AuthCredential credential= EmailAuthProvider.getCredential(email,oldPassword);

user.reauthenticate(credential).addOnCompleteListener(new OnCompleteListener<Void>() {
    @Override
    public void onComplete(@NonNull Task<Void> task) {
        if(task.isSuccessful()){
            if(!newPassword.equals(confirmPassword)) {
                Toast.makeText(SettingsActivity.this,
                        R.string.passwords_do_not_match,
                        Toast.LENGTH_LONG).show();
            }else{
                user.updatePassword(newPassword)
                        .addOnCompleteListener(new OnCompleteListener<Void>() {
                    @Override
                    public void onComplete(@NonNull Task<Void> task) {
                        if(!task.isSuccessful()){
                            Toast.makeText(SettingsActivity.this,
                                    R.string.something_went_wrong,
                                    Toast.LENGTH_LONG).show();
                            dialog.cancel();
                        }else {
                                Toast.makeText(SettingsActivity.this,
                                      R.string.password_successfully_modified,
                                        Toast.LENGTH_LONG).show();
                                dialog.cancel();}
                    } });
            }
        }else {Toast.makeText(SettingsActivity.this,
                    R.string.make_sure_you_enter_your_password_correctly,
                    Toast.LENGTH_LONG).show(); }
    } });
```

Parola değiştirme arayüzü Görsel-18’ de verilmiştir.


Görsel-18

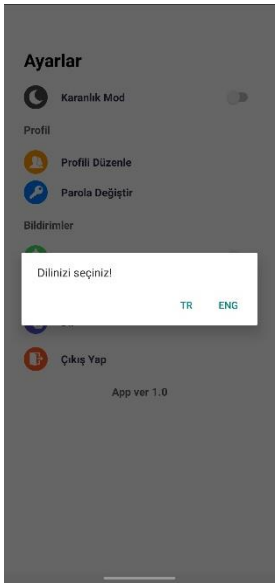
Kullanıcı eğer dil değiştirme butonuna tıklarsa kullanıcıya Türkçe ve İngilizce arasında tercih yapması için diyalog penceresi gösterilir. Kullanıcının seçimine göre lokal konfigürasyon ayarları tekrar yapılandırılır.

Bölüm 2.3 içerisinde kullanıcının internet tercihinin shared preferences ile belleğe kaydedildiğinden bahsedilmişti. Burada da kullanıcı dil tercihini girdiğinde refresh yapılması için öncelikle kullanıcının eski tercihi okunur ardından yeni tercihiyle kıyaslanır ve değişiklik varsa refresh yapılır.

Kullanıcının eski tercihinin okunması ve yeni tercihi olarak İngilizce seçeneğini seçtiği durum için kontrol blokları aşağıda verilmiştir. Türkçe seçeneği için de if bloğu aynı şekildedir.

```
SharedPreferences sharedPreferences = PreferenceManager
      .getDefaultSharedPreferences(getApplicationContext());

if(!lang.equals("en")){
    SharedPreferences saved_values = PreferenceManager
       .getDefaultSharedPreferences(getApplicationContext());
    SharedPreferences.Editor editor=saved_values.edit();
    editor.putString("lang","en");
    editor.commit();

    Locale myLocale = new Locale("en");
    Resources res = getResources();
    DisplayMetrics dm = res.getDisplayMetrics();
    Configuration conf = res.getConfiguration();
    conf.setLocale(myLocale);
    //conf.locale = myLocale;
    res.updateConfiguration(conf, dm);
    Intent refresh = new Intent(SettingsActivity.this, IntroActivity.class);
    finish();
    startActivity(refresh);
}
```

Dil seçim arayüzü Görsel-19'de verilmiştir.




Görsel-19

Kullanıcı çıkış yap butonuna bastığında ise onayı istenir. Onaylaması durumda ise oturumu sonlandırılır ve intro activity'e yönlendirilir. İlgili kod parçası aşağıda verilmiştir.

```
logout.setOnClickListener((v) -> new AlertDialog.Builder(this)
        .setMessage("Are you sure you want to Logout?")
        .setCancelable(false)
        .setPositiveButton("Yes",(dialog, which) ->{
            FirebaseAuth.getInstance().signOut();
            Intent intent = new Intent(
                    SettingsActivity.this,
                    IntroActivity.class);
            intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK
                    | Intent.FLAG_ACTIVITY_CLEAR_TASK);
            startActivity(intent);
        })
        .setNegativeButton("No", null)
        .show());
```

## 2.15. PROFİLİ DÜZENLEME

Kullanıcı profili düzenle butonuna bastığında *Edit Profile Activity* başlatılır. Burada kullanıcıya profil fotoğrafını, kullanıcı adını, e-posta adresini değiştirmeyi ve hesabını silme imkânı tanınır.

Activity başlatıldığında ilk olarak userInfo fonksiyonu çağrılarak kullanıcının bilgileri veri tabanından çekilerek ilgili komponentlere yerleştirilir. Kodun ilgi kısımları aşağıda verilmiştir.

```
private void userInfo(){
    DocumentReference reference = db.collection("Users")
            .document(firebaseAuth.getCurrentUser().getUid());
    reference.addSnapshotListener((value, error) -> {
        if(error == null){
            if(getContext() == null){ return; }
            if(value != null){
                User userModel = value.toObject(User.class);
                assert user != null;
                Picasso.get().load(userModel.getProfileImageUrl())
                        .into(imageView_profilePhoto);
                editText_name.setText(userModel.getUserName());
                editText_mailAddress.setText(userModel.getEmail());
                bioTw.setText(userModel.getBio());
            }
        }
    });
}
```

Kullanıcı bilgilerinin ekranda gösterilmesinden sonra kullanıcı değiştirmek istediği bilginin üzerine tıklayarak değişiklikleri onaylayarak profilini güncellemiş olur. Bölüm 2.10'da cihaz depolamasından fotoğrafın nasıl alındığı ve izinlerin nasıl sağlandığı anlatıldığından dolayı bu bölümde cihaz hafızasından fotoğrafın nasıl alındığı anlatılmayacaktır.

Kullanıcının tüm bilgilerini değiştirdikten sonra profili güncelle butonuna bastığında Gerekli alanların boş bırakılmadığı kontrol edilip eğer her şey tamamsa veri tabanına yükleme işlemi gerçekleşir. Burada profil fotoğrafının Firebase Storage'a yüklenmesi sırasında dosya ismini kullanıcının userId'si olarak ayarlandığını ve bu nedenle yükleme sırasında eski profil fotoğrafının üzerine yazılacağını ve tekrar eski fotoğrafı silmek için ekstra bir efor harcamaya gerek kalmadığını belirtmekte fayda var.

```
StorageReference fileReference= storageReference
      .child("profileImages/"+firebaseAuth.getCurrentUser().getUid());
```

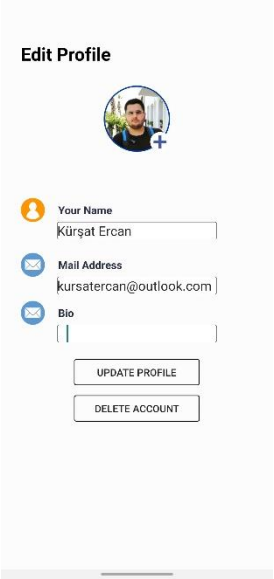
Refereans oluştururken fotoğrafın kaydedileceği konum profileImage klasörü içerisinde kullanıcının uniq id'si olacak şekilde ayarlanmıştır. Ardından fotoğraf uri'si byte şeklinde alınmıştır. Storage'e yüklenmeden önce dosyakalite seviyesi 25 seçilip jpeg formatında sıkıştırılarak veri tabanına yüklenmiştir. Kodun ilgili kısımları aşağıda verilmiştir.

```
Bitmap bmp = MediaStore.Images
        .Media.getBitmap(getContentResolver(), imageData);
ByteArrayOutputStream baos = new ByteArrayOutputStream();
bmp.compress(Bitmap.CompressFormat.JPEG, 25, baos);
byte[] data = baos.toByteArray();
UploadTask uploadTask2 = fileReference.putBytes(data);
```

Koddan anlaşılacağı üzere fotoğrafın referansa yüklenmesiyle geriye *Upload Task* döndürmektedir. Task için *addOnSuccessListener* tanımlayarak success durumunda veritabanından kullanıcının adı, e-posta adresi ve biyografisi güncellenmektedir. Ayrıca Authentication kısmından da kullanıcının email'i güncellemektedir. Authentication tarafındaki güncellemeyle ilgili kısım aşağıda verilmiştir.

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
user.updateEmail(email).addOnCompleteListener(new OnCompleteListener<Void>() {
    @Override
    public void onComplete(@NonNull Task<Void> task) {
        if (task.isSuccessful()) {
            Toast.makeText(EditProfileActivity.this,
                   R.string.opps,Toast.LENGTH_SHORT).show();
                    }
      }
});
```

Profil düzenleme sayfası Görsel-20'de verilmiştir. Burada bio'nun şu anda uygulama sürecinde kullanılmadığını ilerleyen zamanlarda kullanılacağını ve bu yüzen boş bırakılabildiğini belirtmek gerekir.




Görsel-20

## 2.16. HESAP SİLME

Kullanıcı hesap sil butonuna bastığında öncelikle emin olduğunun doğrulamak için bir dialog penceresi açılır. Eğer kullanıcı pencerede hesabı gerçekten silmek istediğini onaylarsa öncelikle kullanıcıya ait postlar silinir.

```
db.collection("Posts").whereEqualTo("userId",user.getUid())
        .get().addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
    @Override
    public void onComplete(@NonNull Task<QuerySnapshot> task) {
        if (task.isSuccessful()) {
            for (QueryDocumentSnapshot document : task.getResult()) {
                db.collection("Posts").document(document.getId()).delete();
            }
        }
      }
      });
```

Ardından storage'den kullanıcının profil fotoğrafı silinir.

```
StorageReference storageRef = FirebaseStorage.getInstance().getReference();
StorageReference desertRef = storageRef
.child("profileImages/"+firebaseAuth.getCurrentUser().getUid());
desertRef.delete();
```

Kullanıcılar koleksiyonundan kullanıcıya ait doküman silinir.

```
db.collection("Users").document(user.getUid()).delete();
```

Ve son olarak kullanıcı Authentication'dan silinerek uygulama intro aktivitesine döndürülür.

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
user.delete();
```

## 3. SONUÇLAR

Projenin özellikle arayüz ve kullanıcı deneyimleri açısından geliştirilmeye ihtiyacı var. Kısıtlı zaman ve kısıtlı imkanlar dahilinde olabildiğince hızlı sonuçlar almaya ve daha fazla fonksiyonellik katabilmek için çabalandığından şu anda ne kadar amacını karşılıyor, gereksinim raporunda tanımlanan gereksinimleri ve proje hedeflerinin büyük çoğunluğunu sağlıyor olsa da uygulama yayınlanmaya hazır değil. Ancak ürünü ve ürün geliştirme süreci bütün olarak ele aldığında bizlere çok büyük deneyim kazandırdığını, ufkumuzu geliştirdiğini rahatlıkla söyleyebiliriz.

# 4. ÖNERİLER

Projede veritabanı olarak Cloud Firestore kullanılmıştır. Firestore veri okumak için çok ideal bir ortam aynı zamanda offline kullanım imkanı sağlıyor, koleksiyondaki değişiklikleri sürekli dinleyerek güncellemeye imkan veriyor. Ayrıca servisler oluşturmaya gerek kalmıyor. Ancak asenkron veri çekme konusunda bazen sorunlar çıkabiliyor, veri yazma konusunda da zaman zaman yavaşlıklar olabiliyor. Bu nedenle veri depolamak için başka çözümler üretmek daha verimli olacaktır.

Kullanıcı deneyimi açısından tüm gereklilikler çok iyi analiz edilip proje gereksinimleri de bu bağlamda hazırlanmalıdır. Aksi taktirde geliştirme süresinde takım içinde “Bu fonksiyon gerçekten gerekli mi?” tartışması zaman kaybına neden olabilmektedir.

Arayüzlerin tasarımları için tasarımcı bulunması, yazılımcının üzerinden büyük bir yük alarak onu rahatlatacağı gibi kod yazma aşamasında da daha verimli çalışmaya sevk edecektir.

## 5. KAYNAKLAR

1. https://ddiy.co/freelance-statistics/ , Freelance Statistics, 11 Haziran 2021.

2. https://yunuskaygun.com/firebase/google-cloud-firestore-nedir/ , Google Cloud Firestore Nedir, 16 HAZİRAN 2021

## 6. EKLER

Elektrik-Elektronik Mühendisliği bölümüyle ortak yapılan çalıştaya proje takımının iki üyesi de farklı gruplarla katılmıştır.

Kürşat ERCAN çalıştaya Arş. Gör. Şeyma AYMAZ'ın sorumluluğunda olan G3 grubuyla dahil olmuştur. Grup e-posta adreslerinin değiştirilmesine karar verildiğinde aynı e-posta adresiyle kayıtlı olan tüm hesaplar için tek tek e-posta bilgisinin değiştirilmesi problemini ele alarak gerekli çalışmaları sağlamıştır.

Burak ALTUNTAŞ çalıştaya Arş. Gör.Özge AYDOĞDU'nun sorumluluğunda olan G2 grubuyla dahil olmuş ve güneş panellerinin verimliliğini düşüren kirlenme problemine otomatik şekilde temizlik yapan bir sistem ile çözüm üretmişdir..

İletişim için gerekmesi halinde kullanılabilecek mail adresleri;

Kürşat ERCAN: 348338@ogr.ktu.edu.tr , kursat_ercan98@gmail.com
Burak ALTUNTAŞ: 340828@ogr.ktu.edu.tr , burak.altuntas@yandex.com

**STANDARTLAR ve KISITLAR FORMU**

Projenin hazırlanmasında uyulan standart ve kısıtlarla ilgili olarak, aşağıdaki soruları cevaplayınız.

1. Projenizin tasarım boyutu nedir? (Yeni bir proje midir? Var olan bir projenin tekrarı mıdır? Bir projenin parçası mıdır? Sizin tasarımınız proje toplamının yüzde olarak ne kadarını oluşturmaktadır?)

| |
|---|
| Projenin benzerleri mevcuttur. Aynı amaçla yapılmış farklı projelerde bulunmaktadır. Ancak geliştirilmesi ve uygulanması açısından proje özgün bir projedir. Tamamen araştırarak ve deneyerek uygulanmış, hiçbir kısmı bir bütün halinde kopyalayarak geliştirilmemiştir. |

2. Projenizde bir mühendislik problemini kendiniz formüle edip, çözdünüz mü? Açıklayınız.

| |
|---|
| Tasarım aşamasında çözülmesi gereken bir mühendislik problemiyle karşılaşmadık. |

3. Önceki derslerde edindiğiniz hangi bilgi ve becerileri kullandınız?

| |
|---|
| Nesne Yönelimli Programlama dersinde öğrendiğimiz nesne kavramını, Veritabanı Yönetimi dersinde öğrendiğimiz tablo ilişkilerini ve daha bir çok bilgiyi proje içerisinde kullanmaktayız. |

4. Kullandığınız veya dikkate aldığınız mühendislik standartları nelerdir? (Proje konunuzla ilgili olarak kullandığınız ve kullanılması gereken standartları burada kod ve isimleri ile sıralayınız).

| |
|---|
| ISO/IEC 12207 Yazılım Yaşam Döngüsü Süreçleri |

5. Kullandığınız veya dikkate aldığınız gerçekçi kısıtlar nelerdir? Lütfen boşlukları uygun yanıtlarla doldurunuz.

a) Ekonomi

| |
|---|
| Firebase hizmetleri için belirli bir kullanım kapasitesinden sonra aylık ücret istemektedir. |

b) Çevre sorunları:

| |
|---|
| Kısıt yoktur. |

c) Sürdürülebilirlik:

| |
|---|
| Uygulamanın tüm Android cihazlarda çalışabilmesi için API ve SDK sürümleri kontrol edilerek işlemler yapılıyor. |

d) Üretilebilirlik:

| |
|---|
| Yeni bir pazar uygulaması oluşturuldu. |

e) Etik:

| |
|---|
| Kısıt yoktur. |

f) Sağlık:

| |
|---|
| Kısıt yoktur. |

g) Güvenlik:

| |
|---|
| Güvenlik problemi oluşturmamaktadır. |

h) Sosyal ve politik sorunlar:

| |
|---|
| Kısıt yoktur. |