T.C.
SAKARYA UNIVERSITY
FACULTY OF COMPUTER AND INFORMATION SCIENCES

E-COMMERCE ANDROID APPLICATION - VIRTUAL SALES OF FOOD "E-Restaurant"

GRADUATION PROJECT

Rıdvan Al HOURANI

Department of Computer Engineering

Thesis Advisor: Assoc. Prof. Dr . Cüneyt BAYILMIŞ

2019-2020 Summer Semester

T.C.
SAKARYA ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

E-TİCARET ANDROİD UYGULAMASI – SANAL OLARAK YEMEK SATIŞI "E-Restoran"

BSM 498 - BİTİRME ÇALIŞMASI

Rıdvan AL Hourani – G1512.10575

Fakülte Anabilim Dalı : BİLGİSAYAR MÜHENDİSLİĞİ

Bu tez .. / .. / … tarihinde aşağıdaki jüri tarafından oybirliği / oyçokluğu ile kabul edilmiştir.

…….	…….	…….
Jüri Başkanı	**Üye**	**Üye**

ÖNSÖZ

Günümüzde internet çağında, cep telefonları neredeyse her bir kişinin ayrılmaz bir parçasıdır. Ve küresel ekonomik eğilim, kişinin refahını iyileştirmeyi ve artırmayı ve mağaza sahiplerinin satış kapasitesini artırmayı amaçlayan trend ile cep telefonu ve cep telefonları için uygulamalar alanına odaklanıyor, bundan çıkarak, müşteriler ve restoranlar arasında bir satış hattı oluşturmak amaçlayan bir akıllı telefon uygulaması tasarlama fikri geldi, ayrıca bu fikre yardım edebilecek, uygulamaya ek özellikler eklenecek.

Tez bitirmeni hazırlamamda ilk önce Allah'a sonra da bana destek olan anneme, babama, kardeşlerime, sevgili aileme değerli Hocam sayın Doç. Dr. Cüneyt Bayılmış'a, yardımlarını esirgemeyen başta Erkam olmak üzere bütün arkadaşlarıma ve bana kaynak sağlayan kurumlar samimi teşekkürlerimi sunarım.

Abstract

In today's internet era, mobile phones are an integral part of almost every single person. And the global economic trend is focusing on the field of applications for smartphones.

From the fact that I want to improve myself and advance my career on cryptology and machine learning and apply this knowledge on a tangible thing as such smartphones engineering and with a step to understand this sector more I decided to make an Android app that have a tangible value and common application such as E-shops.

With the trend aiming to improve and increase the well-being of the person and increase the sales capacity of the store owners, and , out of these came the idea to design a smartphone application that aims to create a sales line between customers and restaurants, that have Additional features that make me understand the concepts of Android system more deeply and in an applying way as a first step in this major, I decided to build an E-Restaurant android application for ordering foods.

In my application I tried to learn more about Android concepts and how to handle different common used scenarios, the project consists of three combined applications, to explain briefly, we have a "Customer" who wants to buy a food and this lead us to the "Restaurant" who needs to make that food to this customer and the bridge between this two is the "Delivery guy". Everyone from this three needs to handle a certain work to make the final service available.

To convert this to a software solution, how it should be? Yes, three separated applications.

A custom application for the Restaurant,
A custom application for the customer,
And a custom application for the delivery guy,

Those three apps with its different features such as maps, payment methods, notification handling, databases and User Interface friendly using; will exchange information between each other and work together to provide a full-scale restaurant service operation.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

API	: Uygulama Programlama Arayüzü
SDK	: Yazılım Geliştirme Kiti
APK	: Android Package Kit
APP	: Uygulama
GUI	: Grafiksel Kullanıcı Arayüzü
GPS	: Küresel Konumlama Sistemi
JSON	: JavaScript Nesne Gösterimi
XML	: Genişletilebilir İşaretleme Dili
FCM	: Firebase Bulut Mesajlaşma
POJO	: Düz Eski Java Nesnesi
DAO	: Veri Erişim Nesnesi

ŞEKİLLER LİSTESİ

ÖZET

Anahtar kelimeler: Android, Mobil Uygulama, E-Ticaret

Kullanıcılar kolayca kendi akıllı telefonlarından bizim uygulamayı açarak istedikleri yemeği sanal bir şekilde sipariş verirler, sipariş alan restoran ise bu siparişi kabul edip ve onu hazırladıktan sonra nakliyeci adama verecek. Nakliyeci yemek siparişi ilgili müşteriye ulaştıracak.

Bu üçlü döngüyü (Müşteri – Restoran – Paketçi Kargocu) arasında bizim uygulamamızı kolaylaştırıcı bir iletişim aracı oluşturuyor.

Uygulamayı yapmak için Android Native Java dili kullanıyoruz, ayrıca gerekli 3cü parti kütüphaneler, API'ler (Uluslararası ödeme API'si, GBS API'si) ve cloud servisler (Firebase) vb. servisler kullanmamı oldu.

Sonuç olarak bu üçlü döngüyü yapmak için çıkan sonuç üç uygulama tasarlanmasına ihtiyaç duyuldu. Müşteriler için bir, Restoran sahiplere bir ve nakliyeciye toplamda üç uygulama.

BÖLÜM 1. GİRİŞ

Günümüzde internet çağında, cep telefonları neredeyse her bir kişinin ayrılmaz bir parçasıdır. Ve küresel ekonomik eğilim, kişinin refahını iyileştirmeyi ve artırmayı ve mağaza sahiplerinin satış kapasitesini artırmayı amaçlayan trend ile cep telefonu ve cep telefonları için uygulamalar alanına odaklanıyor, bundan çıkarak, müşteriler ve restoranlar arasında bir satış hattı oluşturmak amaçlayan bir akıllı telefon uygulaması tasarlama fikri geldi, ayrıca bu fikre yardım edebilecek, uygulamaya ek özellikler eklenecek.

1.1. Problemin Tanımı

Bu projede çözülmeye çalışıldığı problemi, Restoranlar ve Müşteriler arasında temassız bir şekilde iletişim bağı kurmaktır.

Çözmek gereken problemler ilk başta gelen

- Restoran kendi ürünleri sunarak müşteriler gezinip görüntülensin.
- Müşteri satın almak istediği ürünü Restorana para ödemesidir
- Ayrıca müşteri satın aldığı ürünü teslim etmek için; Restoran ve Müşteri arasında bir fiziksel bağı kurmak gerekiyor.

Dolaysıyla, bunu en iyi şekilde yapmak için üç uygulama yapmak iyi bir çözüm olacaktır.

İlk Uygulama, Müşteri uygulaması ve bu uygulama içinde sağlanacak çözümler, müşteriler restoranı sunmak istediği ürünleri gezinip satın alabilmiş olacaklardır hem de bu uygulamada bir sanal ödeme sistemi çözümü sağlayıp restoran hak ettiği paralar alabilsin.

İkinci uygulamada, özellikle Restoran için olacaktır, bu uygulamada restorana müşterilerden gelen siparişler ile ilgilenecek. Hem de Restoran kendi sunmak istediği ürünler bu uygulama ara yüzünden kolayca ekleme – silme ve ürün güncelleme çözümleri sağlayacaktır. Üstelikle restoran bu uygulamadan hangi kargocu müsait olanları görüp onlara kargo ürün teslimi görevi verme olanağı verecektir.

Üçüncü uygulamada paketçi kargocu için olacaktır. Kargocuya özel harita gösterimleri (Müşterinin adresi - kargocunu kendisi) üstelikle müşteriyi telefon ile arama özelliği, kargocu gitmek istediği başka bir yere yerler aratıp ve bir rota çizme çözümler sağlanacaktır.

1.2. Başarı Ölçütleri

Bu çalışma Android Mobil uygulaması ve E-ticaret dalını temsil ettiğinden bazı temel noktalar kapsamak gerekir:

1. Uygulamaların arayüzleri kullanıcı dostu bir deneyim vermeli,
2. Uygulamalar bir E-ticaret mağazasına temel özellikler içermeli,
3. İnşa edilecek uygulamalar birbirine iletişim sağlanmalı

Projede ilerlerken bu noktaları göz önünde bulundurarak geliştirmeye çalıştım. İlk olarak tasarımlar dinamik ve kolay anlaşılmasına önem gösterdim. Sonra da projede ilerleyerek uygulamanın yapımında en güncel teknolojiler kullanmaya çalıştım ve gerekli e-ticaret özellikleri ve özellikle ödeme sistemi harici API'den yardım alarak bir ödeme sistemi inşa ettim.

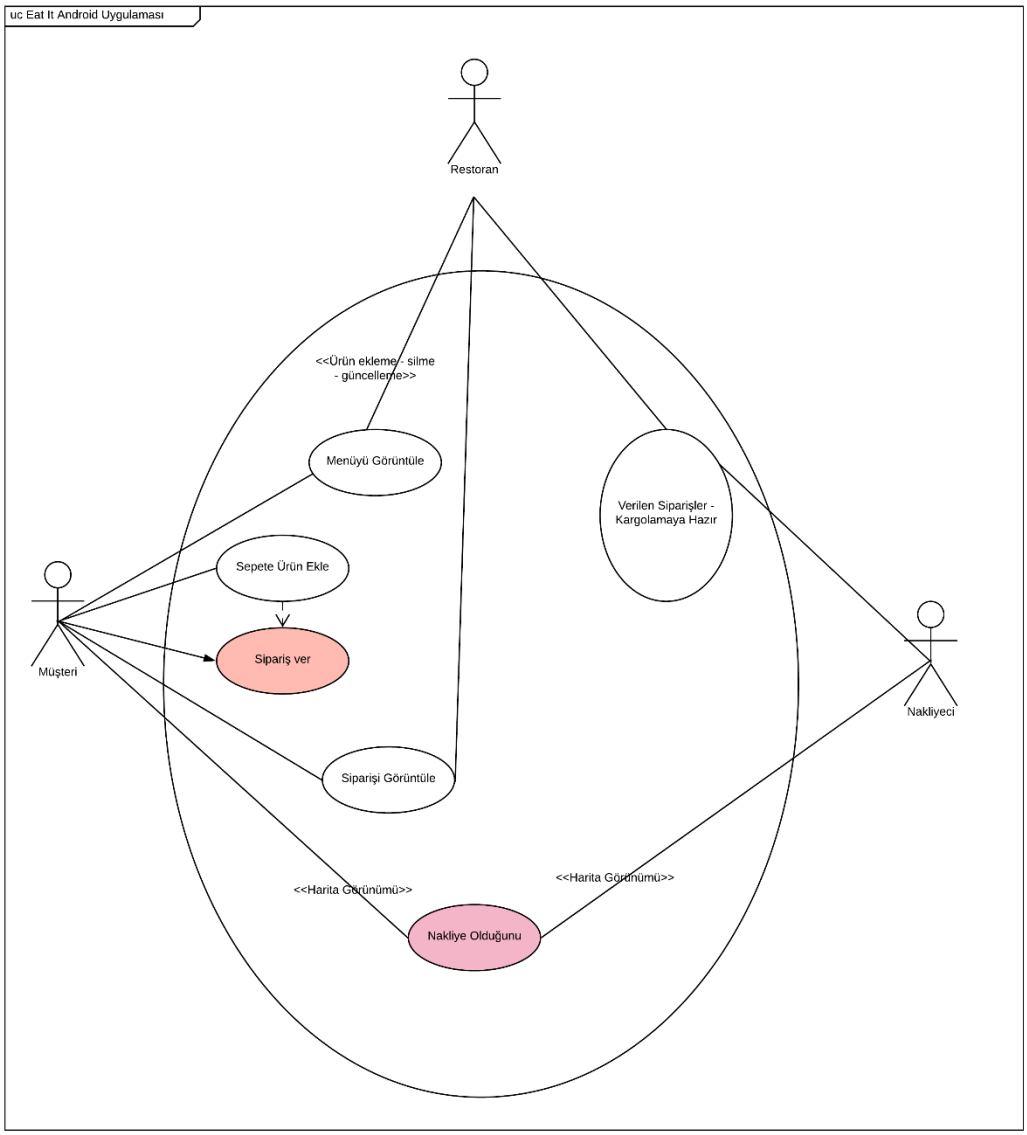
Proje yapımında ilerlerken uygulamalar birbirine iletişim sağlanacaktır ve aralarında bulut hizmetleri kullanarak bilgi dağıtım olacaktır.

Uygulamalar kullanarak:

- ✓ Uzaktan satış işlemi ve sanal ödeme yapabiliyoruz.
- ✓ Siparişler gerçek zamanlı olarak hangi aşamada olduğunu takip edebiliyoruz.
- ✓ Müşteri – Restoran ve Paketçi aralarında sürekli uygulamalar sağladığı arayüzleri aralarında bilgi dağıtımı ve iletişimi çok kolayca sağlıyoruz.

BÖLÜM 2. TASARIMLAR

2.1. Projenin Use Case Diyagramı



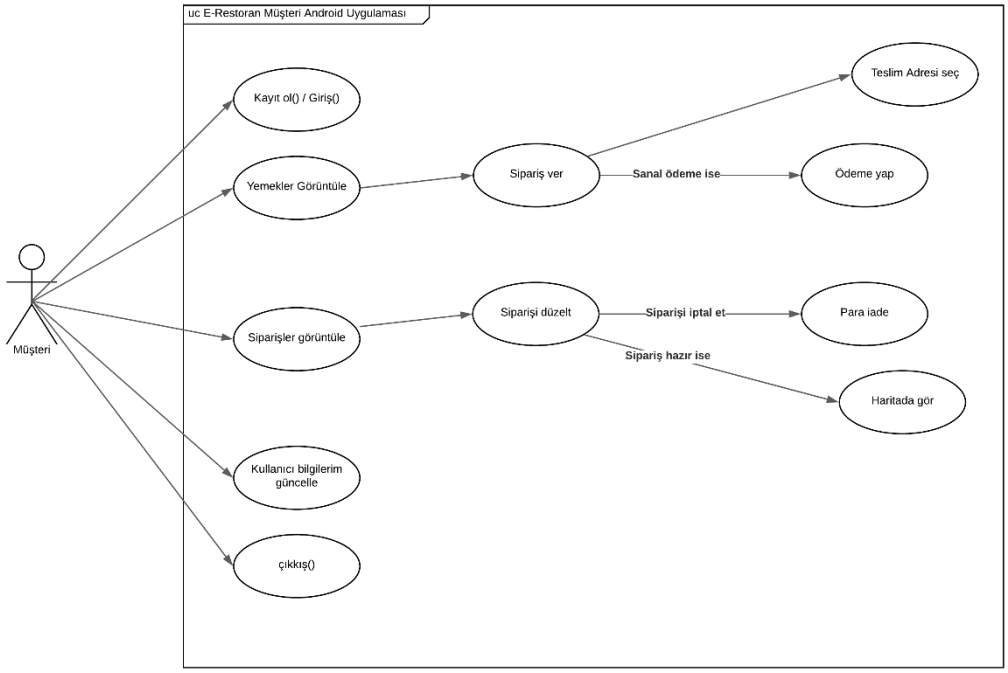
Şekil 2.1. E-Restoran Use Case diyagramı

2.2. Uygulamanın Arayüzü Tasarımları (Scratches)

Benim projemde toplamda üç adet uygulama olacak; birincisi müşteriler tarafından kullanılacak, ikincisi restoran sahiplerinden kullanılacak, üçüncüsü ise kargocu kullanacak.

2.2.1. Müşteri uygulaması için tasarım arayüzleri (Client)

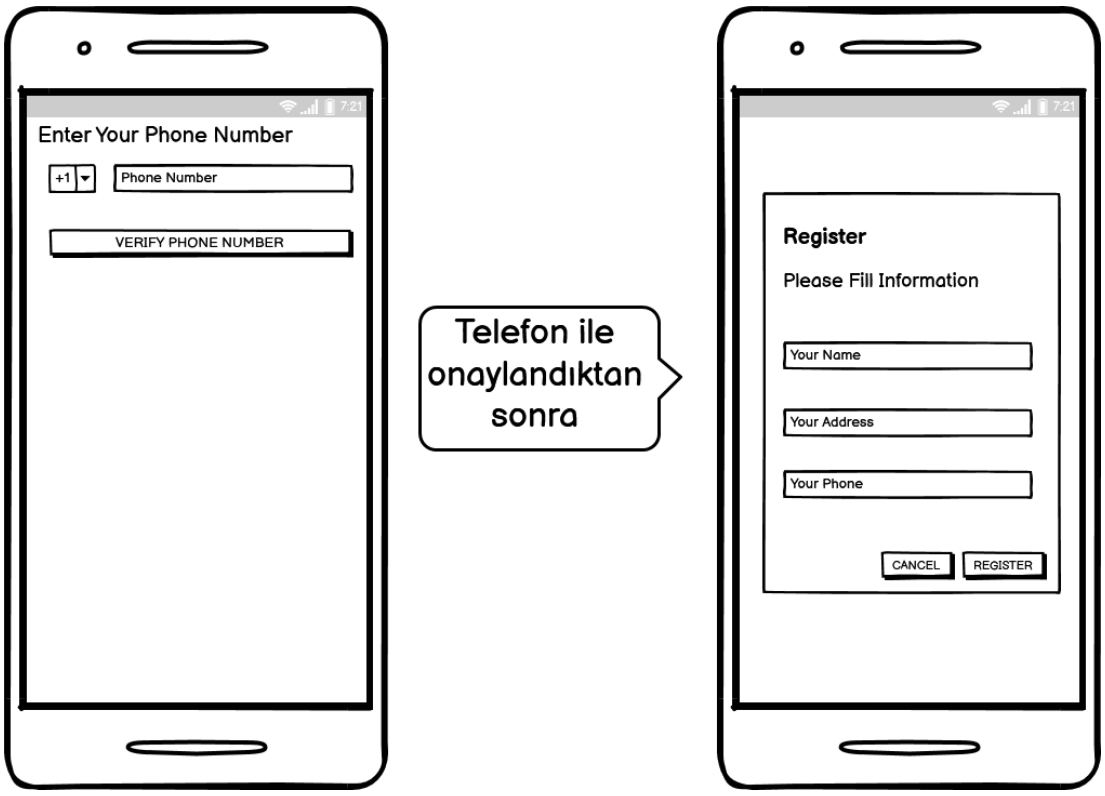
2.2.1.1. Müşteri Uygulaması Use Case UML Diyagramı



Şekil 2.2. E-Restoran – Müşteri uygulaması Use Case diyagramı

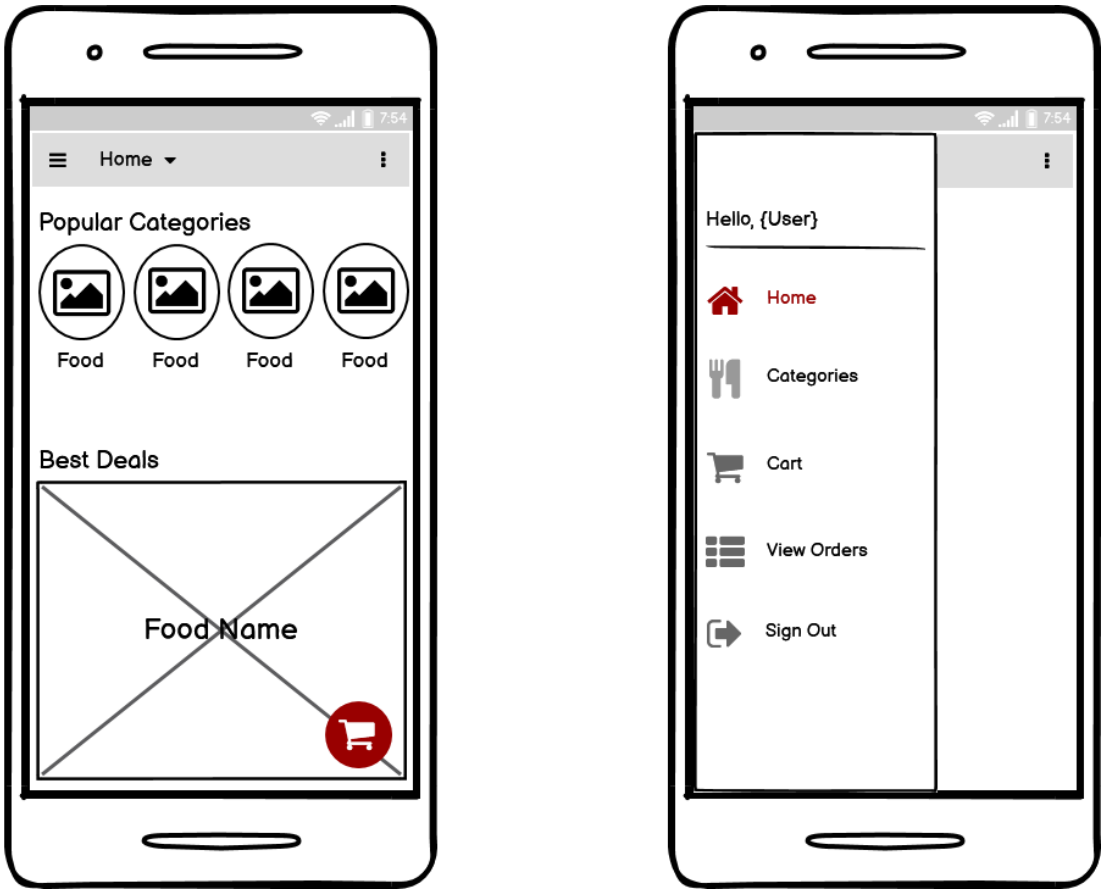
2.2.1.2. Müşteri Uygulaması ilk tasarımı

- Üye kayıt ekranı



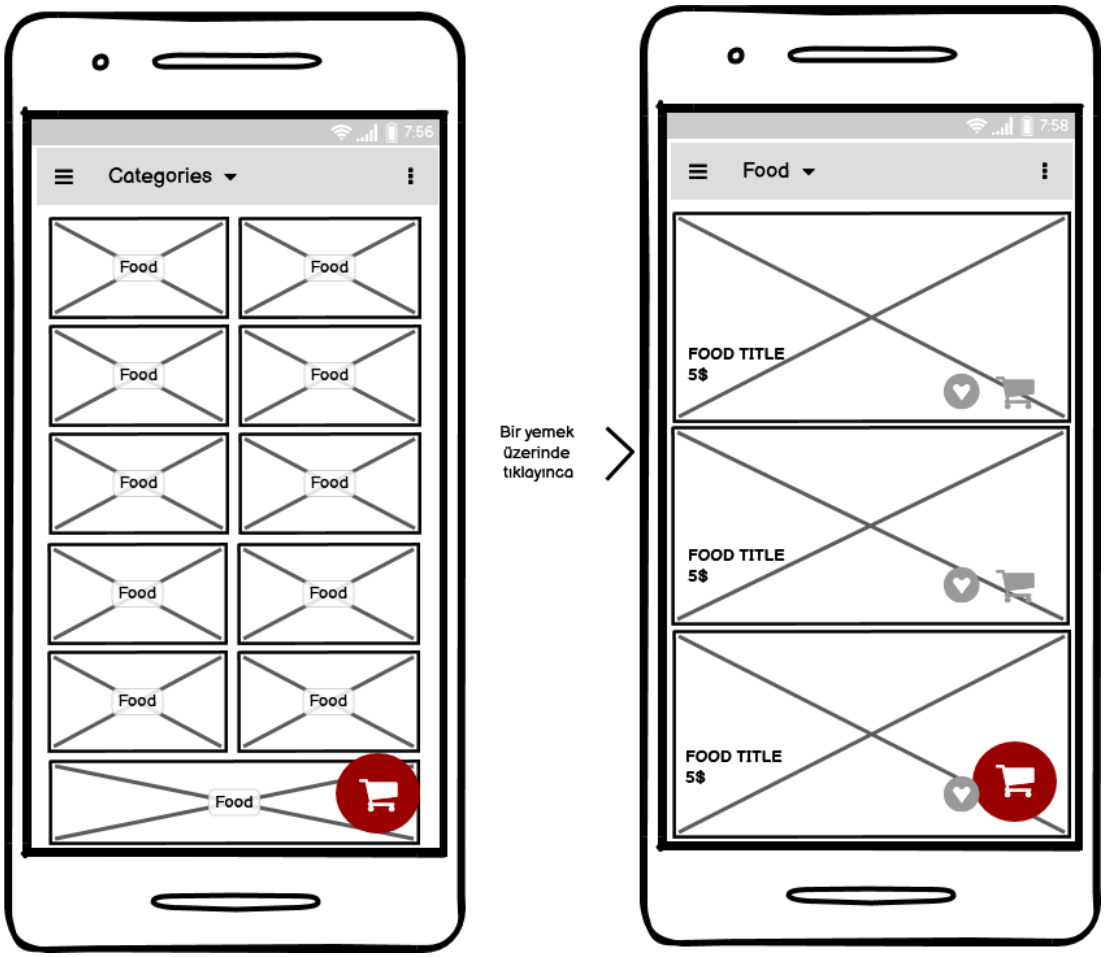
Şekil 2.3. Client uygulaması, Üye kayıt arayüzü

- Ana sayfa ekranı



Şekil 2.4. Client uygulaması, Ana sayfa arayüzü

- Kategoriler ve Yemekler ekranı

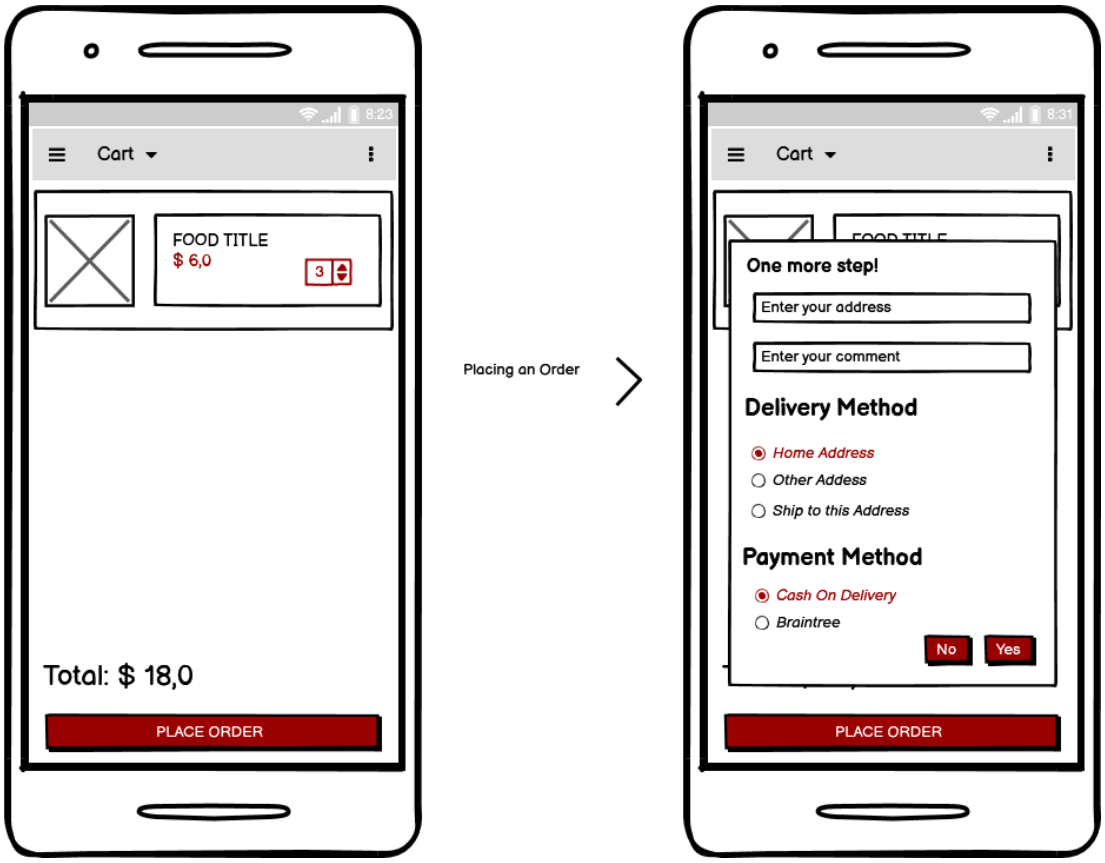


Şekil 2.5. Client uygulaması, Kategoriler ve yemek listesi arayüzleri

- Yemek detayları

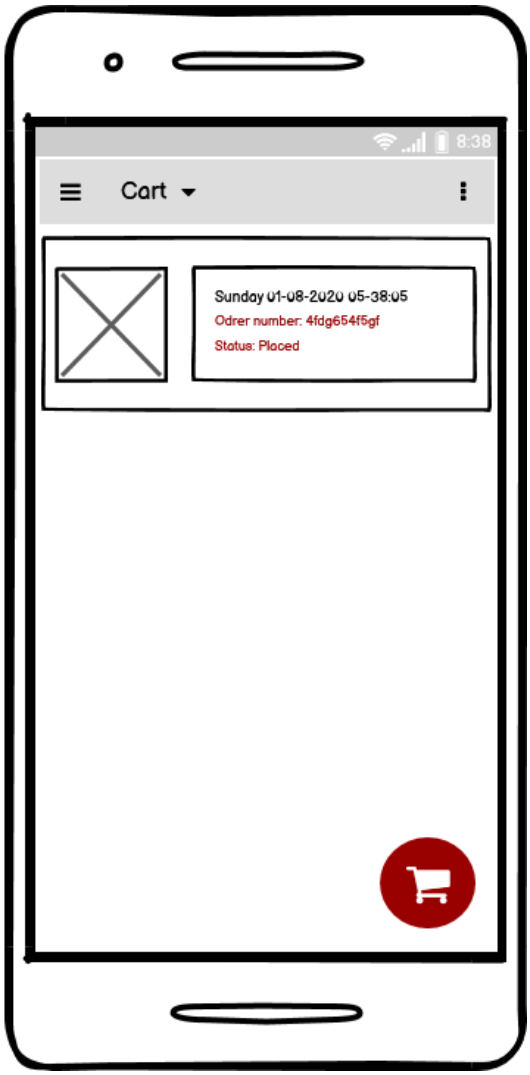
Şekil 2.6. Client uygulaması, Yemek detayları arayüzü

- Kart ekranı



Şekil 2.7. Client uygulaması, Sepete arayüzü

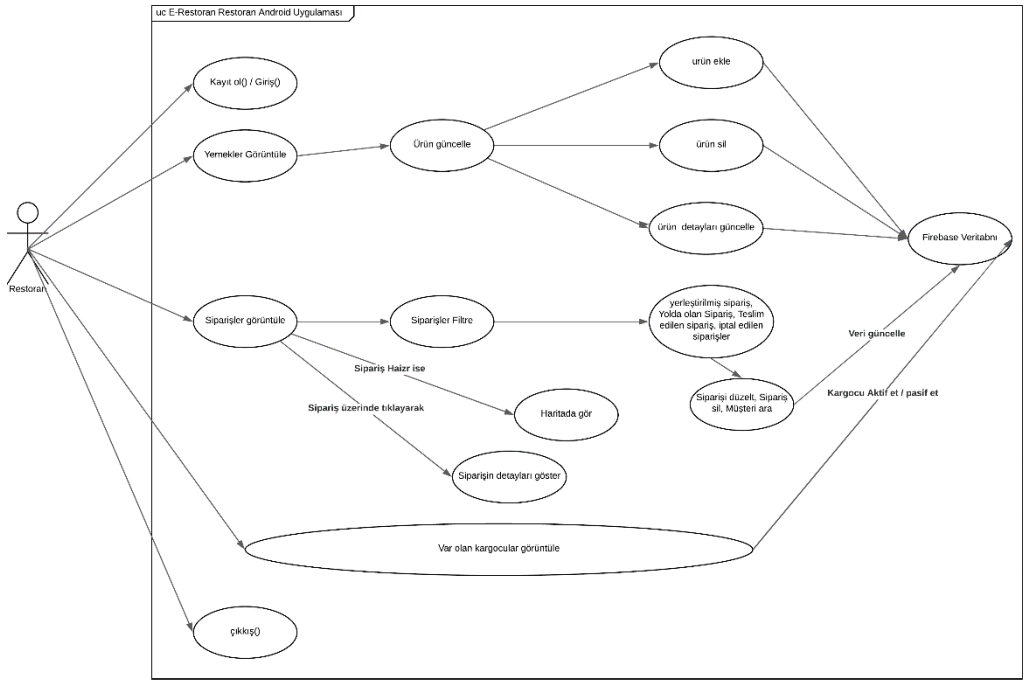
- Siparişleri görüntüle ekranı



Şekil 2.8. Client uygulaması, Siparişleri görüntüle arayüzü

2.2.2. Restoran uygulaması için tasarım Arayüzleri (Scratches)

2.2.2.1. Restoran Uygulaması Use Case UML Diyagramı



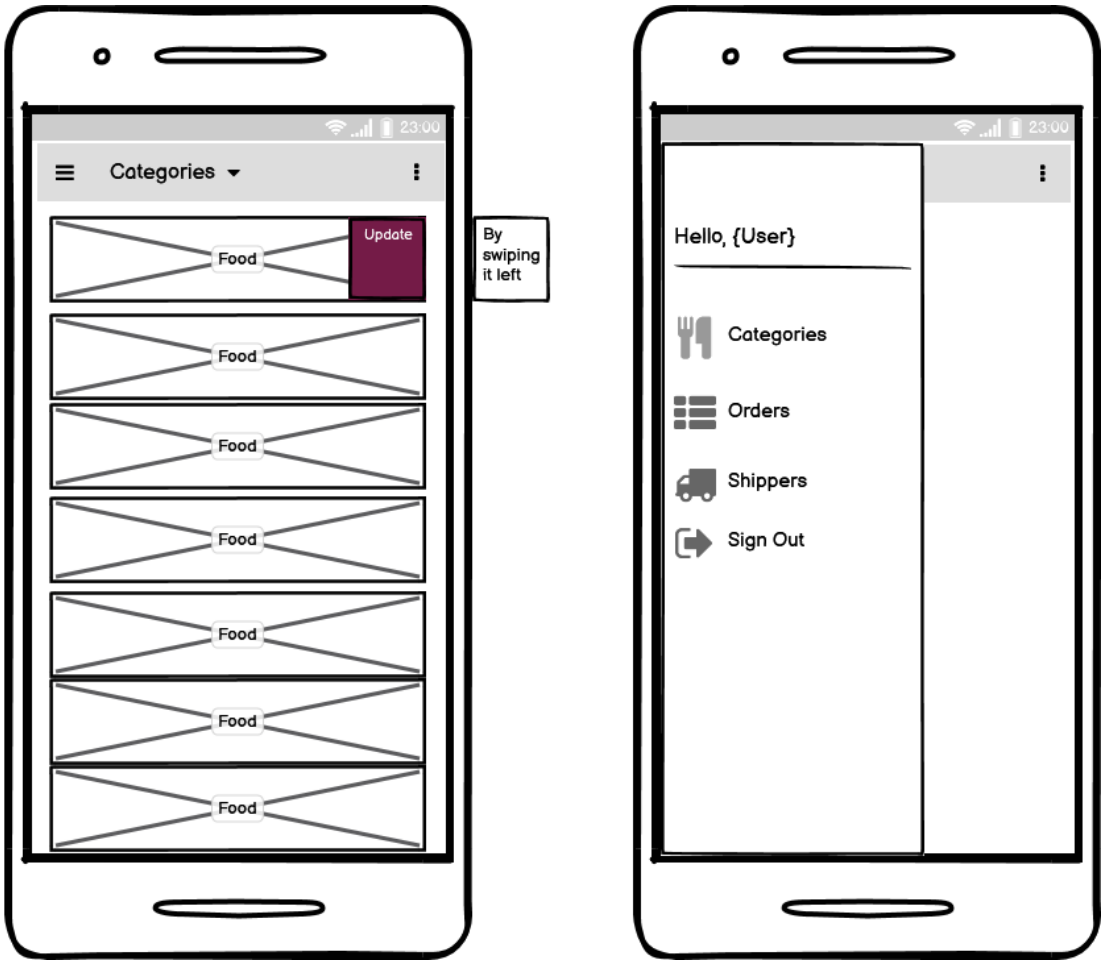
Şekil 2.9. E-Restoran – Restoran uygulaması Use Case diyagramı

2.2.2.2. Restoran Uygulaması arayüzü tasarımları

- Üye kayıt ekranı

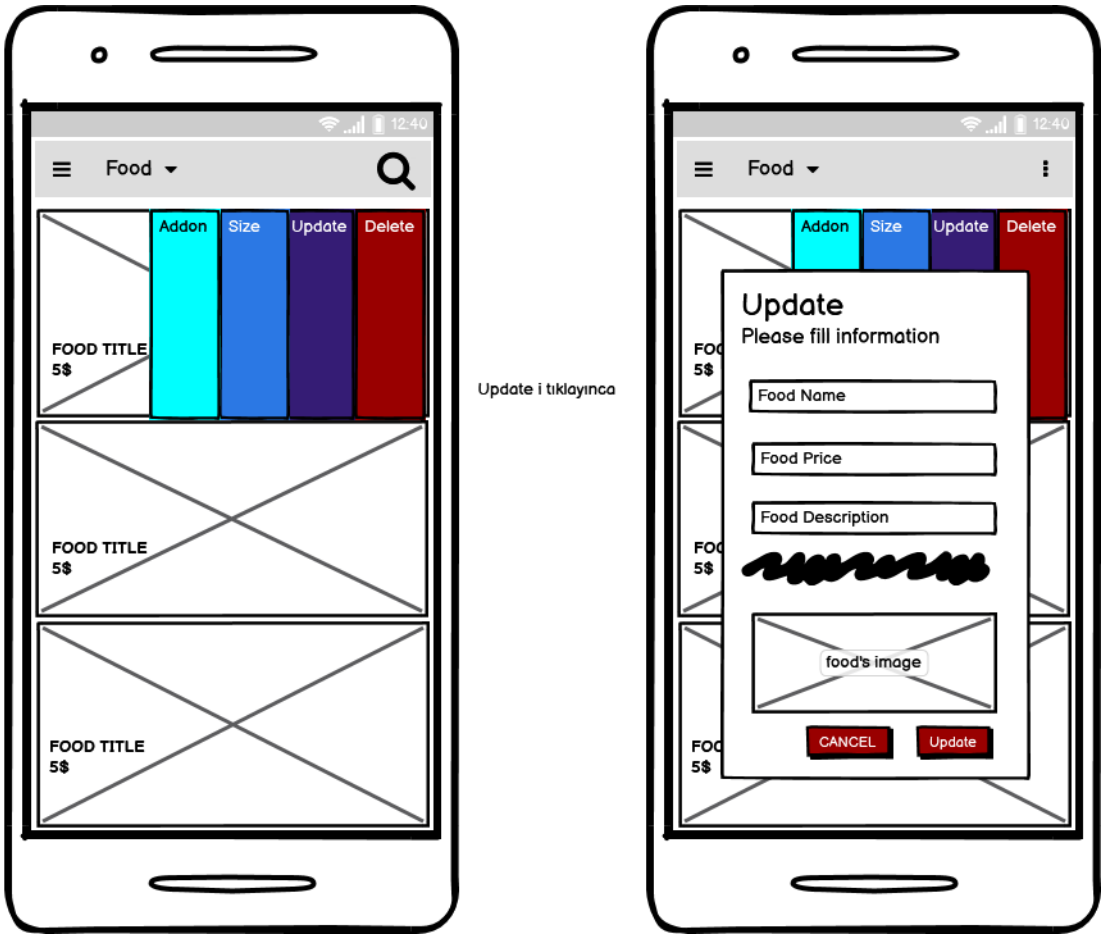
Şekil 2.10. Server uygulaması, üye ol arayüzü

- Kategoriler, ana sayfa olarak



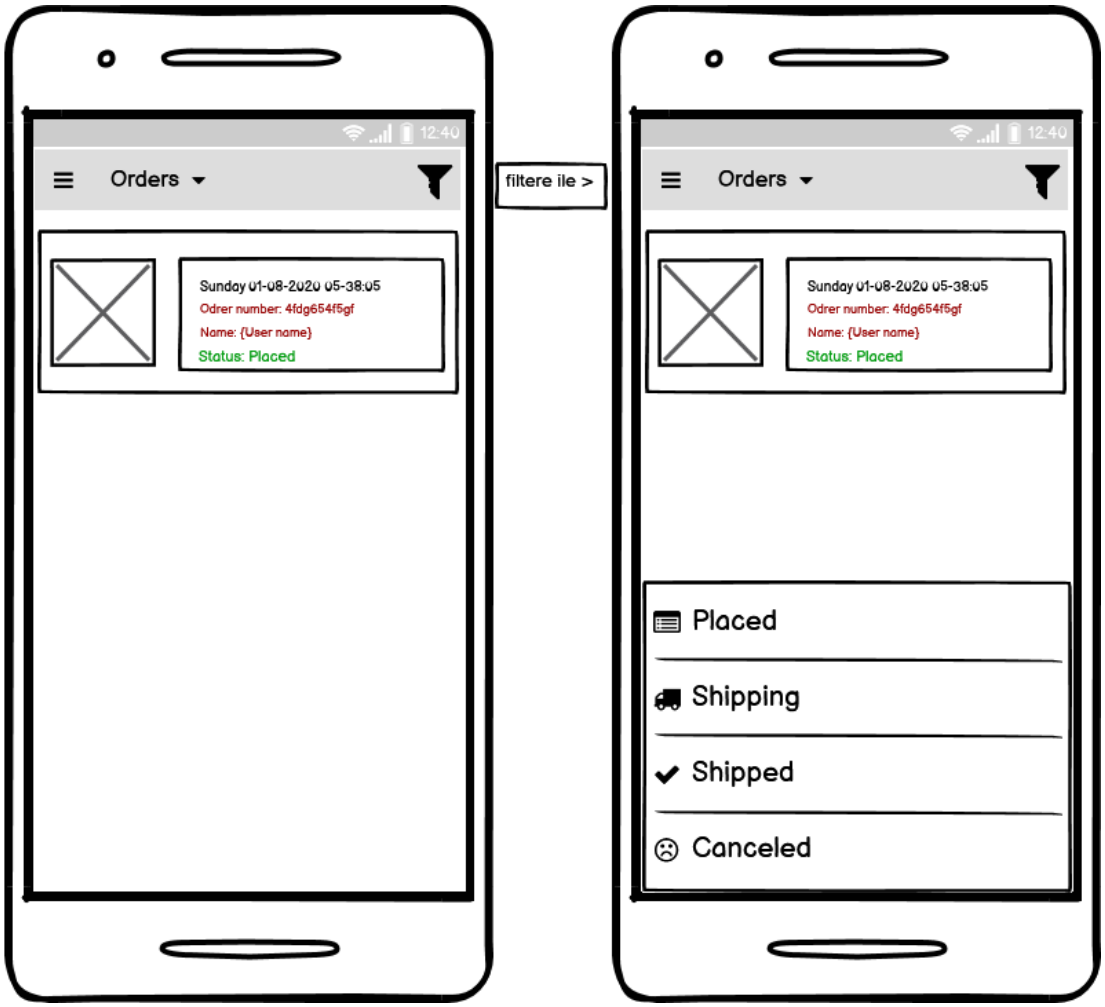
Şekil 2.11. Server uygulaması, Ana sayfa arayüzü

- Yemek listesi



Şekil 2.12. Server uygulaması, Yemek listesi arayüzü

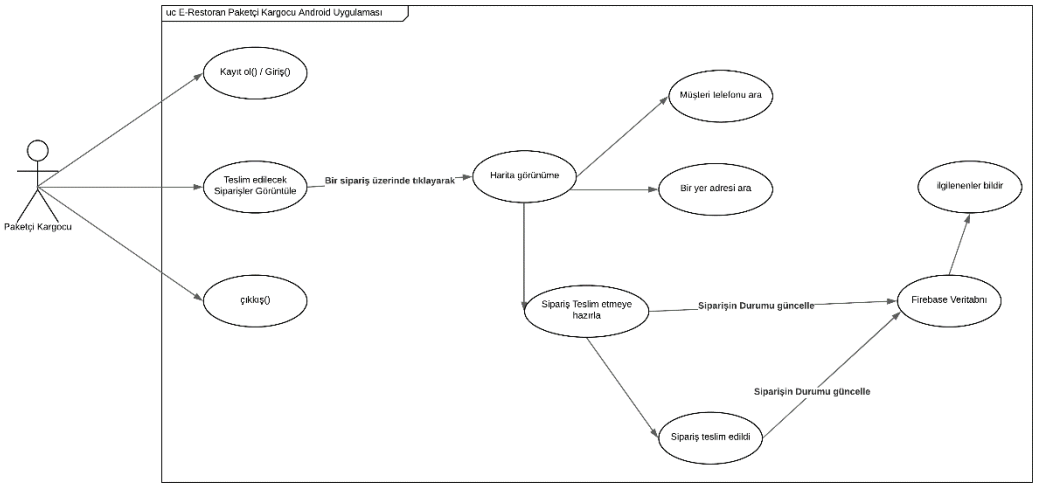
- Siparişler



Şekil 2.13. Server uygulaması, Siparişler arayüzü

2.2.3. Kargocu Uygulaması İçin Tasarım Arayüzleri

2.2.3.1. UML Diyagramı



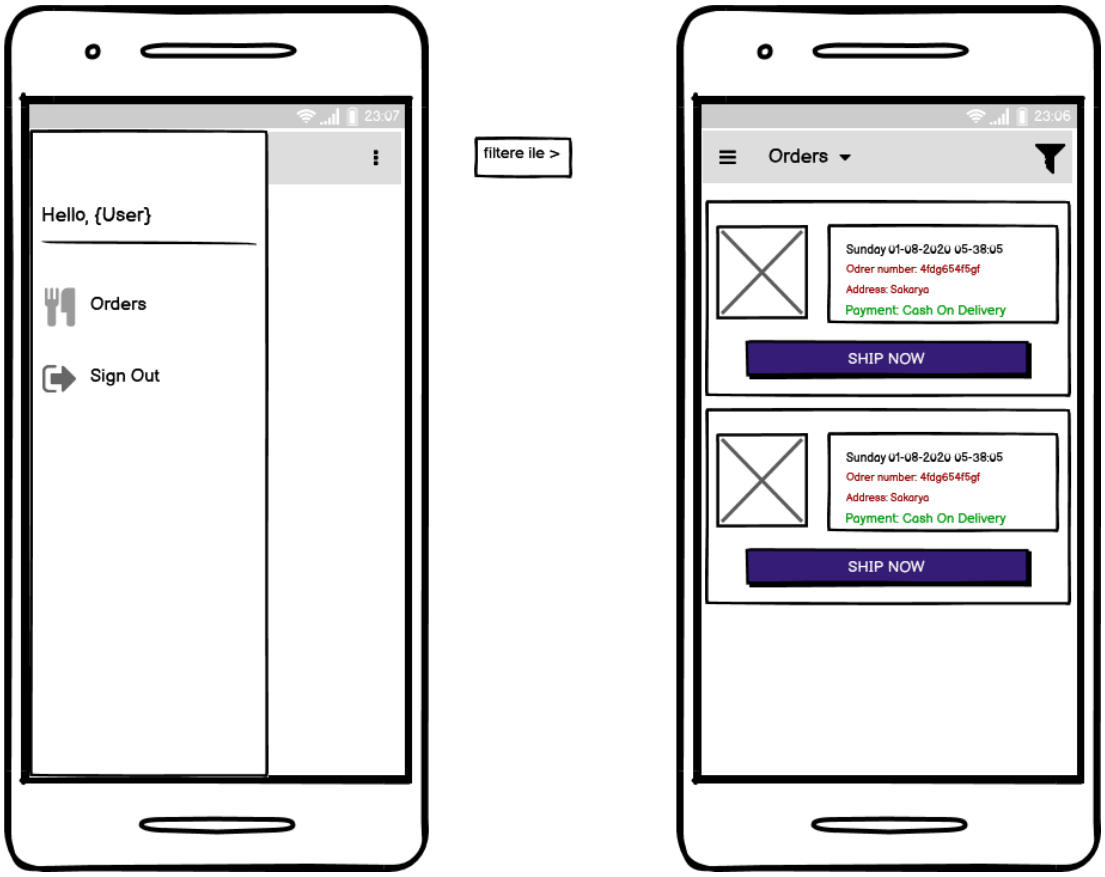
Şekil 2.14. E-Restoran – Kargocu uygulaması Use Case diyagramı

2.2.3.2. Kargocu Uygulaması Tasarım Arayüzleri

- Üye kayıt arayüzü

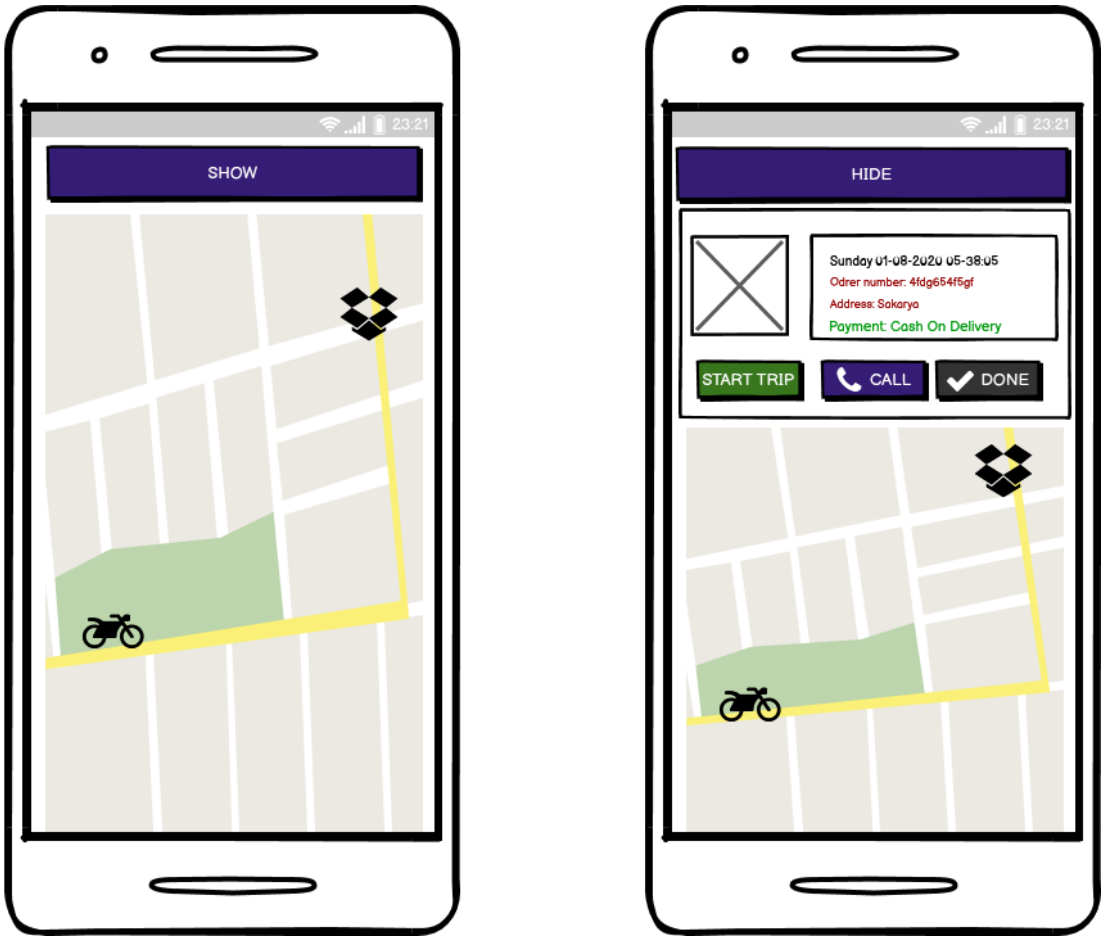
Şekil 2.15. Shipper uygulaması, üye ol arayüzü

- Ana ekran arayüzü



Şekil 2.16. Shipper uygulaması, ana ekran arayüzü

- Harita Arayüzü



Şekil 2.17. Shipper uygulaması, harita arayüzü

BÖLÜM 3. ANDROID GENEL KAVRAMLARI

Bu bölümde temel ve genel Android kavramalarından bahsedeceğim, bu kavramlar yaptığım projede temel yapı oluşturacaklardır

3.1. Android Nedir?

Android; Google ve Open Handset Alliance tarafından, mobil cihazlar için geliştirilmekte olan, Linux tabanlı özgür ve ücretsiz bir işletim sistemidir. Sistem açık kaynak kodlu olsa da, kodlarının ufak ama çok önemli bir kısmı Google tarafından kapalı tutulmaktadır. Android'in desteklenen uygulama uzantısı ".apk"dır.

Android, aygıtların fonksiyonelliğini genişleten uygulamalar yazan geniş bir geliştirici grubuna sahiptir. Android için hali hazırda 1 milyondan fazla uygulama bulunmaktadır. Google Play Store ise, Android işletim sistemi uygulamalarının çeşitli sitelerden indirilebilmesinin yanı sıra, Google tarafından işletilen kurumsal uygulama mağazasıdır. Geliştiriciler, ilk olarak aygıtı, Google'ın Java kütüphanesi aracılığıyla kontrol ederek Java dilinde yazmışlardır.

Android, Linux çekirdeği üzerine inşa edilmiş bir mobil işletim sistemidir. Bu sistem ara katman yazılımı, kütüphaneler ve API C diliyle yazılmıştır. Uygulama yazılımları ise, Apache harmony üzerine kurulu Java-uyumlu kütüphaneleri içine alan uygulama iskeleti üzerinden çalışmaktadır. Android, derlenmiş Java kodunu çalıştırmak için dinamik çevirmeli Android Runtime (ART) kullanır ve cihazların fonksiyonelliğini artıran uygulamaların geliştirilmesi için çalışan geniş bir programcı-geliştirici çevresine sahiptir.

Şekil 3.1. Android logosu

1. Çekirdek: Linux kernelidir. Güvenlik, hafıza yönetimi, süreç yönetimi, ağ yığınları ve sürücü modellerini içermektedir.

2. Android Runtime: Sanal makinedir. Dalvik Sanal Makinesini de içermektedir. 5.0 ile Dalvik kaldırılmış ve ART'ye geçilmiştir.
3. Kütüphaneler: Veritabanı kütüphaneleri, web tarayıcı kütüphaneleri, grafik ve arayüz kütüphanelerini içermektedir.
4. Uygulama Çatısı: Uygulama geliştiricilere geniş bir platform sunan kısımdır.
5. Uygulama Katmanı: Doğrudan Java (programlama dili) ile geliştirilmiş uygulamaları içermektedir. [1].

3.2. Activity

Activity, kullanıcıların etkileşimde bulunabileceği bir görünüm sağlayan bir uygulama bileşenidir. Bir uygulama birden fazla aktiviteden oluşur. “Main Activity”, uygulama ilk kez başlatıldığında kullanıcıya sunulur. Bizim durumumuzda, Eat It iki “main” activity arasında seçim yapar: Giriş Activity ve Main Activity. Birincisi
Eat It ilk açıldığında ve ikincisi bir kullanıcı kayıt olduktan sonra sunulur.

Her activity farklı eylemler gerçekleştirmek için başka bir activity başlatabilir. Her yeni activity başlatıldığında, önceki activity durdurulur, ancak sistem bir yığındaki activity (yani “arka yığın”) korur. Yeni bir activity başladığında, arka yığın üzerine itilir ve kullanıcının odağını alır. Kullanıcı geçerli activity ile işini bitirip Geri düğmesine bastığında, yığından çıkarılır ve yok edilir ve önceki activity’ı devam eder.

Activity’ler sisteme erişilebilmesi için bildirim dosyasında (AndroidManifest.xml dosyası) bildirilmesi gerekir. Bu bildirim dosyası, uygulama hakkında temel bilgileri Android sistemine sunar. Aşağıdaki gibi bilgiler: minimum SDK (Yazılım Geliştirme Kiti) sürümü, izinler, activity’ler, hizmetler…

Her bir activity oluştururken bir adet Java dosyası bir adet de XML dosyası oluşmaktadır. Bu XML dosyası tasarım kodlarını barındırır Java kodları da activitynin arkaplan kodlarını barındırmaktadır.

3.2.1. Yaşam Döngüsü

Her bir acitivitynin başlangıcından öldürülmesine kadar geçen bir döngü bulunmaktadır. Bu döngüde çeşitli fonksiyonlar çalışmaktadır. Biz bu fonksiyonlar içerisine yeni fonksiyonlar yazabilirsiniz. Bu fonksiyonlar şu şekildedir:

```
public class ExampleActivity extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // The activity is being created.
  }
  @Override
  protected void onResume() {
    super.onResume();
    // The activity has become visible (it is now "resumed").
  }
  @Override
  protected void onPause() {
    super.onPause();
    // Another activity is taking focus (this activity is about to be "paused").
  }
  @Override
  protected void onDestroy() {
    super.onDestroy();
    // The activity is about to be destroyed.
  }
}
```

onCreate : Activity başlatıldığında ilk çalışan fonksiyon

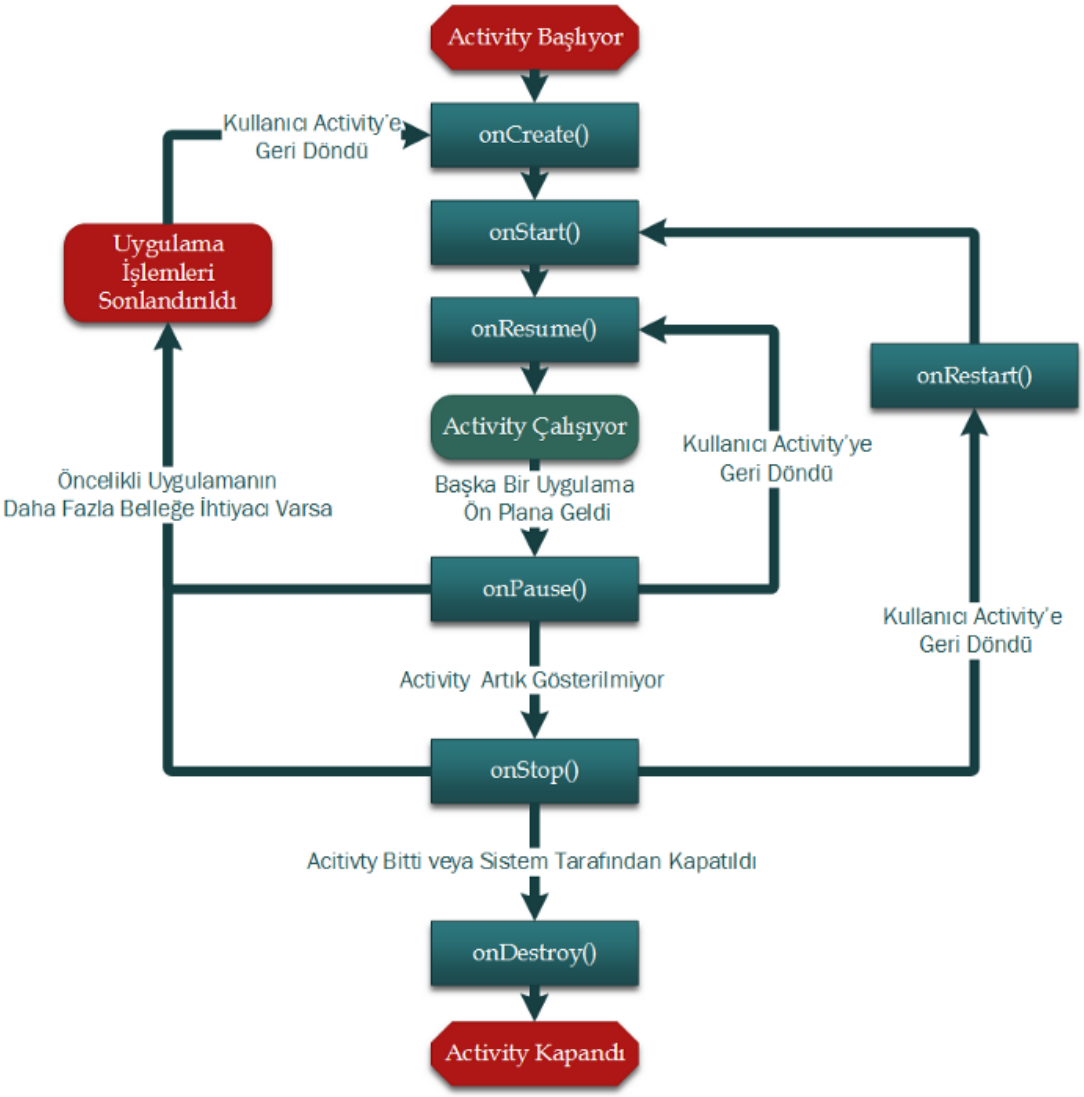
onStart : onCreate methodu işlemini tamamladıktan sonra çalışan fonksiyon

onResume : Activity çalıştıktan sonra bütün ön yüzdeki işlemler bittiğinde activity devamı için çalışan kısımdır. Activity arka plana geçip tekrar ön plana geldiğinde onResume fonksiyonu tetiklenir ve Activity çalışmaya devam eder

onPause : Activity arka plana alındığında bu fonksiyon tetiklenir

onStop : Aktivite arkaplana atıldığında çalışan bir diğer metoddur. Kullanıcı Activitye geri döndüğünde onRestart methodu tetiklenir ve onStart methodundan itibaren yaşam döngüsü devam eder. Eğer kullanıcı geri dönmezse onDestroy methodu tetiklenir.

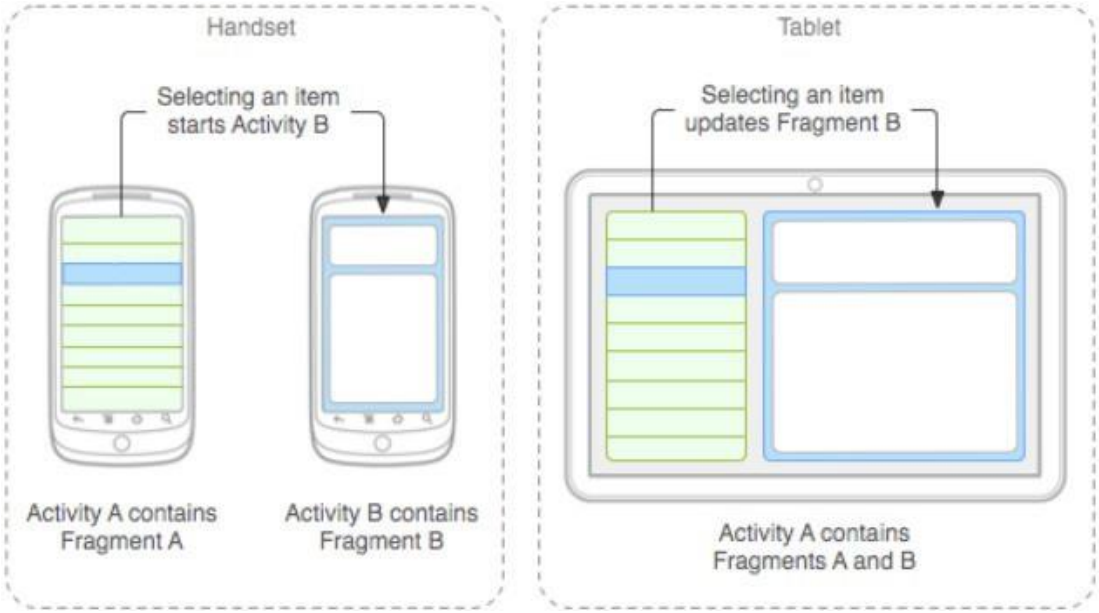
onDestroy : Yaşam döngüsünün en son kısmıdır. Activity sonlandığında tetiklenir. Bellekte uygulama için kullanılan tüm kaynaklarda sonlandırılmış olur



Şekil 3.2. Activity yaşam döngüsü

3.3. Fragment

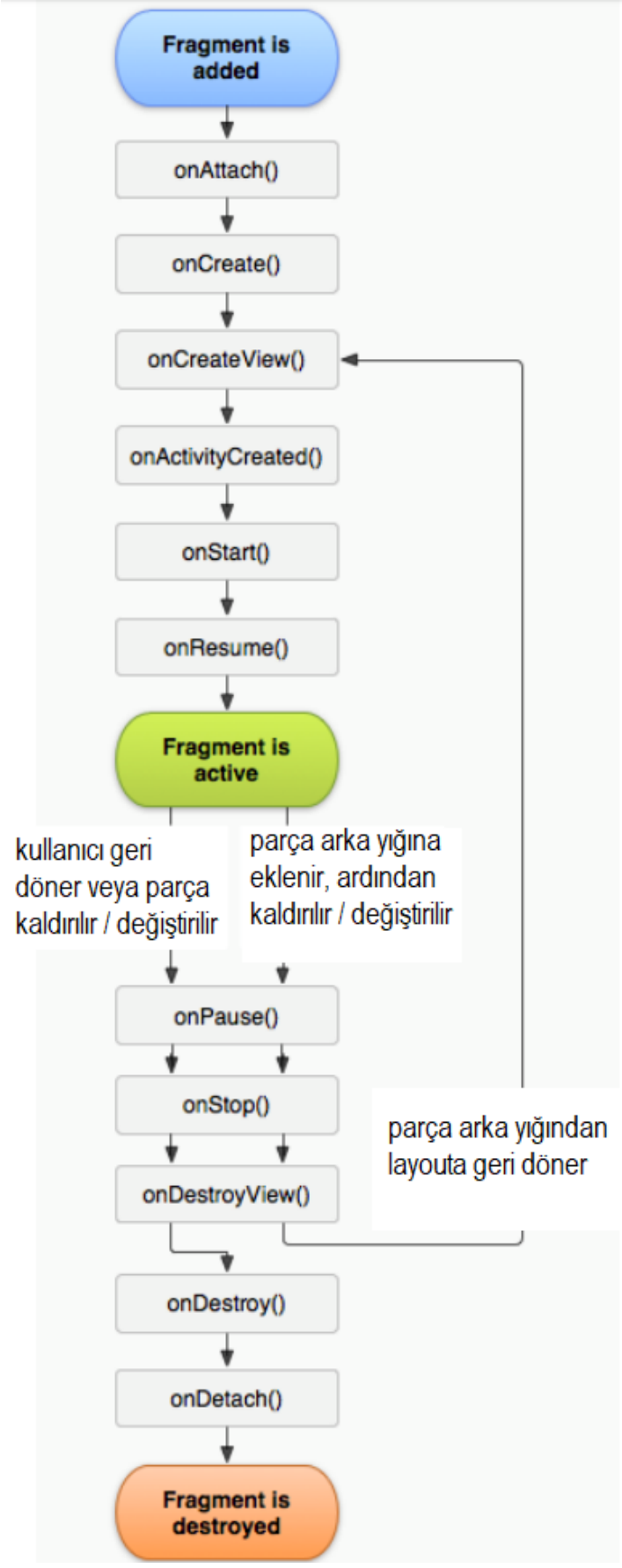
Bir Fragment, bir Activity'de activity veya kullanıcı arabiriminin bir bölümünü temsil eder. Çok bölmeli bir UI oluşturmak ve birden fazla etkinlikte bir parçayı yeniden kullanmak için birden çok parçayı tek bir activity'de birleştirebiliriz. Bir parçayı, kendi yaşam döngüsü olan, kendi girdi olaylarını alan ve etkinlik çalışırken ekleyebileceğimiz veya kaldırabileceğimiz bir modüler bölüm olarak düşünebiliriz.



Şekil 3.3. Fragment

3.3.1. Yaşam Döngüsü

Avtivity gibi aynı yöntemlere sahiptir, ayrıca ek olarak onCreateView() metodu, bu yöntem ilk başta çağrılır, kullanıcı arayüzü çizmek için bir defa çağrılır.



Şekil 3.4. Fragment yaşam döngüsü

BÖLÜM 4. KULLANILAN TEKNOLOJİLERİ

4.1. Butter Knife (Android için bir View kütüphanesi)

Butter Knife kütüphanesi Jake Wharton tarafından geliştirilen bir kütüphanedir, bu kütüphane bir View Injector kütüphanelerden sayılır.

Bilindiği üzere ButterKnife bir View Injector kütüphanesidir. Bir başka deyişle, önyüzde tanımladığımız view bileşenlerini, annotation'lar kullanarak projemize dâhil etmemize olanak sağlıyor. Sadece view'leri değil, aynı zamanda projemizde kullandığımız String, Drawable, Color ve Dimen gibi kaynakları da aynı yöntemle pratik bir şekilde projemize ekleyip bizi kod tekrarı yapmaktan kurtarıyor.

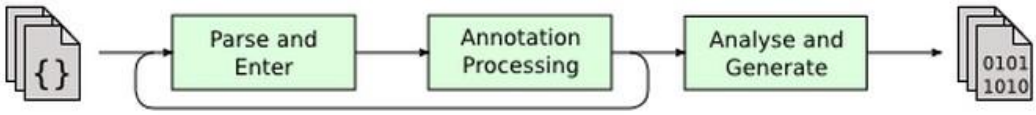
```
public class FoodDetailFragment extends Fragment implements TextWatcher {


    private Unbinder unbinder;

    //view needs inflate
    ChipGroup chip_group_addon;
    EditText edt_search;

    @BindView(R.id.img_food)
    ImageView img_food;
    @BindView(R.id.btnCart)
    CounterFab btnCart;
    @BindView(R.id.btn_rating)
    FloatingActionButton btn_rating;
    @BindView(R.id.food_name)

        // kodun devamı

                ...

public View onCreateView(@NonNull LayoutInflater inflater,
                         ViewGroup container, Bundle savedInstanceState) {

    unbinder = ButterKnife.bind(this,root);

}
```

Görüldüğü üzere findViewById() metodunu tekrar tekrar kullanmak zorunda kalmıyoruz.

Peki, ButterKnife'ın çalışma prensibi nedir?

ButterKnife'ın arkasında yatan güç, Annotation Processing'dir. Annotation Processing, derleme işleminin aşamalarından biridir.

```
implementation 'com.jakewharton:butterknife:10.2.1'
annotationProcessor 'com.jakewharton:butterknife-compiler:10.2.1'
```



Şekil 4.1. Butter Knife kütüphanesi görünümler xml'den java koda bağlama işelmi

Annotationlar derleme zamanında tespit edilir ve normal kod satırlarına çevrilirler. Bir başka deyişle, runtime dediğimiz gerçek çalışma zamanında annotation diye bir şey yoktur. Projemizi derlediğimizde, Annotation Processor devreye girer, projedeki tüm Java sınıflarını tarayıp, annotationları bulur ve normal kod satırlarına çevirir. Üretilen bu yeni kod, normal bir Java sınıfı gibi yeniden derlenir. Annotationlar sadece yazılım geliştirme sürecini kolaylaştırmak için görünürde vardır, derleme zamanının ötesine geçemezler. Unutulmamalıdır ki, Annotation Processorlar sadece annotationları normal satırlara çevirmeye yarar. Derleme anında, sınıflara yeni metodlar eklemek gibi değişiklikler yapamazlar.

Java'da kendi annotationlarımızı yazma olanağımız var. Bunun için aynı zamanda bu annotationları işleyecek bir Annotation Processor yazmamız gerekli. ButterKnife için yapılan da tam olarak bundan ibaret. Bir Annotation Processor yazmak için AbstractProcessor.java sınıfını extend eden bir sınıf yazmamız gereklidir. AbstractProcessor.java içinde tanımlı olan ve bizim kullanacağımız process() sınıfı, asıl işin yapıldığı yer.

4.1.1. ButterKnife Çalışma Yapsıs

Android projemizi ButterKnife kullanarak yazdımığız ve Build düğmesine basıp, projeyi derlediğimiz. Arka planda olanlar kabaca şu şekilde:

Derleyici, ButterKnife'in annotation processorunun process() metodunu çağırır.
Bu metot içinde, projemizdeki tüm Java sınıfları taranır ve ButterKnife annotationları nerelerde kullanılmış, tespit edilir.
Annotation kullanılmış bir sınıf bulunduğu zaman, yeni bir Java dosyası üretilir: <Sınıf_Adi>$$ViewInjector.java (Bildiğimiz bir Java sınıfı)
Bu yeni ViewInjector sınıfı içindeki annotationlar, bizim bildiğimiz eski stil kodlarla yer değiştirilir. Kısacası, findViewById() ve view.setOnClickListener() gibi satırlar tam bu noktada projemize eklenir.
Son olarak da ButterKnife.inject(this) satırı çağırıldığında, üretilen tüm VievInjector dosyalarının inject() metotları çağrılır ve bu dosyalar derlenmeye başlanır.
Yukarıdaki verdiğim kod derlenirken, ButterKnife tarafından üretilen kodu aşağıda görebiliyoruz: [4]

```
public class FoodDetailFragment_ViewBinding implements Unbinder {

      private FoodDetailFragment target;

      private View view7f09008a;

      private View view7f09008f;
```

```
View view;
target.img_food = Utils.findRequiredViewAsType(source, R.id.img_food,
"field 'img_food'", ImageView.class);
view = Utils.findRequiredView(source, R.id.btnCart, "field 'btnCart' and
method 'onCartItemAdd'");
target.btnCart = Utils.castView(view, R.id.btnCart, "field 'btnCart'",
CounterFab.class);
view7f09008a = view;
view.setOnClickListener(new DebouncingOnClickListener() {
  @Override
  public void doClick(View p0) {
    target.onCartItemAdd();
  }
});
view = Utils.findRequiredView(source, R.id.btn_rating, "field 'btn_rating'
and method 'onRatingButtonClick'");
target.btn_rating = Utils.castView(view, R.id.btn_rating, "field
'btn_rating'", FloatingActionButton.class);
view7f09008f = view;
view.setOnClickListener(new DebouncingOnClickListener() {
  @Override
  public void doClick(View p0) {
    target.onRatingButtonClick();
  }
});
```

4.2. Retrofit

Retrofit, Android Developerlar arasında sıklıkla kullanılan bir networking kütüphanesidir. Retrofit'in bu kadar popüler olmasında REST API'lara kolaylıkla erişebilme, test edilebilir ve kolay kullanımı etkendir.

REST(Representational State Transfer) adından da anlaşılacağı gibi uzak sistemdeki mantıksal kaynakları HTTP protokolü ile GET, POST, PUT, PATCH, DELETE methodlarını çağırarak kullanmaktır.

Retrofit kütüphanesi altında farklı kütüphaneler var, onlardan benim projemde kullandığım GSON, ve Scalar dönüştürücülerdir, bu kütüphaneler uzak API'lerle bilgi transferi kolaylar -Json'dan istediğim dile-

Kurulumundan bahsetmek gerekirse build.gradle dosyasında bu şekilde tanımlanır

```
//retrofit2
implementation 'com.squareup.retrofit2:adapter-rxjava2:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
implementation 'com.squareup.retrofit2:converter-scalars:2.9.0'
```

Kullanımı, ilk önce bir Java sınıfı oluşturup sonra kullanmak istediğimiz yerde oluşturduğumuz Retrofit sınıfından nesne oluşturup kullanmaktır

```
package com.ds.androideatitv2client.Remote;

import retrofit2.Retrofit;
import retrofit2.adapter.rxjava2.RxJava2CallAdapterFactory;
import retrofit2.converter.gson.GsonConverterFactory;

public class RetrofitCloudClient {
    private static Retrofit instance;
    public static Retrofit getInstance(){
        if(instance == null)
            instance = new Retrofit.Builder()
                    .baseUrl("https://us-central1-androideatitv2client-
3f83d.cloudfunctions.net/widgets/")
                    .addConverterFactory(GsonConverterFactory.create())

.addCallAdapterFactory(RxJava2CallAdapterFactory.create())
                    .build();
        return instance;
```

```
    }
}
```

Üstte sınıfın constructor metodunda retrofit nesnesini oluşturduk. Burada baseUrl() metoduna üzerinde çalışağımız API adresini yazdık. addConverterFactory() metodu içerisinde üstte convert için dahil ettiğimiz kütüphanenin GsonConverterFactory.create() metodunu ekledik. Burada farklı converter kütüphaneleri kullanılabilir. getInstance() metodunda ise oluşturduğumuz retrofit nesnesini döndüren bir yapı kurduk. Her seferinde yeni bir nesne oluşturmak yerine bir kere oluşturup sonrasında var olan nesneyi çağırılır.
API'dan okuyacağımız verileri POJO classlara aktarmamız gerekiyor. [5]

POJO (Plain Old Java Object) sınıflar:
Düz Eski Java Nesne bir Java nesnesi, Java Dil Belirtimi tarafından zorlananlar dışında herhangi bir kısıtlama ile bağlı olmayan sınıflardır.

Bunun gibi:

```
public class FoodModel {
    private String key;
    private String name,image,id,description;
    private Long price;


    private List<AddonModel> userSelectedAddon;
    private SizeModel userSelectedSize;

    public FoodModel() {}

    public String getName() {}

    public String getImage() {return image;}

    public String getId() {return id;}

}
```

Kurucu fonksiyonu yok, ve alanları private, public veya protected kısıtlama koymadan olan Java sınıflardır.

Sonrasında Retrofit i devreye sokmak için kullanmak istediğimiz API arayüzünde retrofit metodları kullanarak API Json lü şekilde çekildikten sonra yukarıda bahestiğim üzer POJO sınıflara aktararak istediğimiz yerde bilgi kullanımı sağlarız.

Aşağıdaki kod parçası Braintree Payment API'i retrofit metotları kullanarak çağrılma yöntemi göstermekte

```
import io.reactivex.Observable;
import retrofit2.http.Field;
import retrofit2.http.FormUrlEncoded;
import retrofit2.http.GET;
import retrofit2.http.HeaderMap;
import retrofit2.http.POST;

public interface ICloudFunctions {
    @GET("token")
    Observable <BraintreeToken> getToken (@HeaderMap Map<String, String>
headers);

    @POST("checkout")
    @FormUrlEncoded
    Observable <BraintreeTransaction> submitPayment (
            @HeaderMap Map<String, String> headers,
            @Field("amount") double amount,
            @Field("payment_method_nonce") String nonce);
}
```

4.3. Retrofit GSON

Retrofit GSON kütüphanesi yukarıda retrofit anlattığımı kütüphanelerinden bir tanesi. Amacını API'lerden gelen Json veri işlemektir.

Build.gradle dosyasında bunu ekledikten sonra
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

sonrasında herhangi bir API'nin sınıfında GsonConverterFactory.create()) çağırarak JSON veriyi POJO'ya dönüştürerek kullanabiliriz.

```
.addConverterFactory(GsonConverterFactory.create())
```

4.4. Retrofit Convertor Scalars

Bu kütüphane Retrofit GSON kütüphanesi gibi API'den gelen veriyi istediğimiz türe dönüştürme olanağı sağlar, Scalars kütüphanesi gelen veriyi String tipine veriyi dönüştürür.

Google Maps API'sinden gelen yanıtlar String tipinde ve bu veriler işlemek için Retrofit Scalar kullanmamı gerekiyordu.

```
public class RetrofitGoogleAPIClient {
    private static Retrofit instance;

    public static Retrofit getInstance(){
        return instance == null ? new Retrofit.Builder()
                    .baseUrl("https://maps.googleapis.com/")
                    .addConverterFactory(ScalarsConverterFactory.create())

.addCallAdapterFactory(RxJava2CallAdapterFactory.create())
                    .build() : instance;

    }
}
```

4.5. Looping View Pager

Gradle.build dosyasına:

```
implementation 'com.asksira.android:loopingviewpager:1.1.4'
```

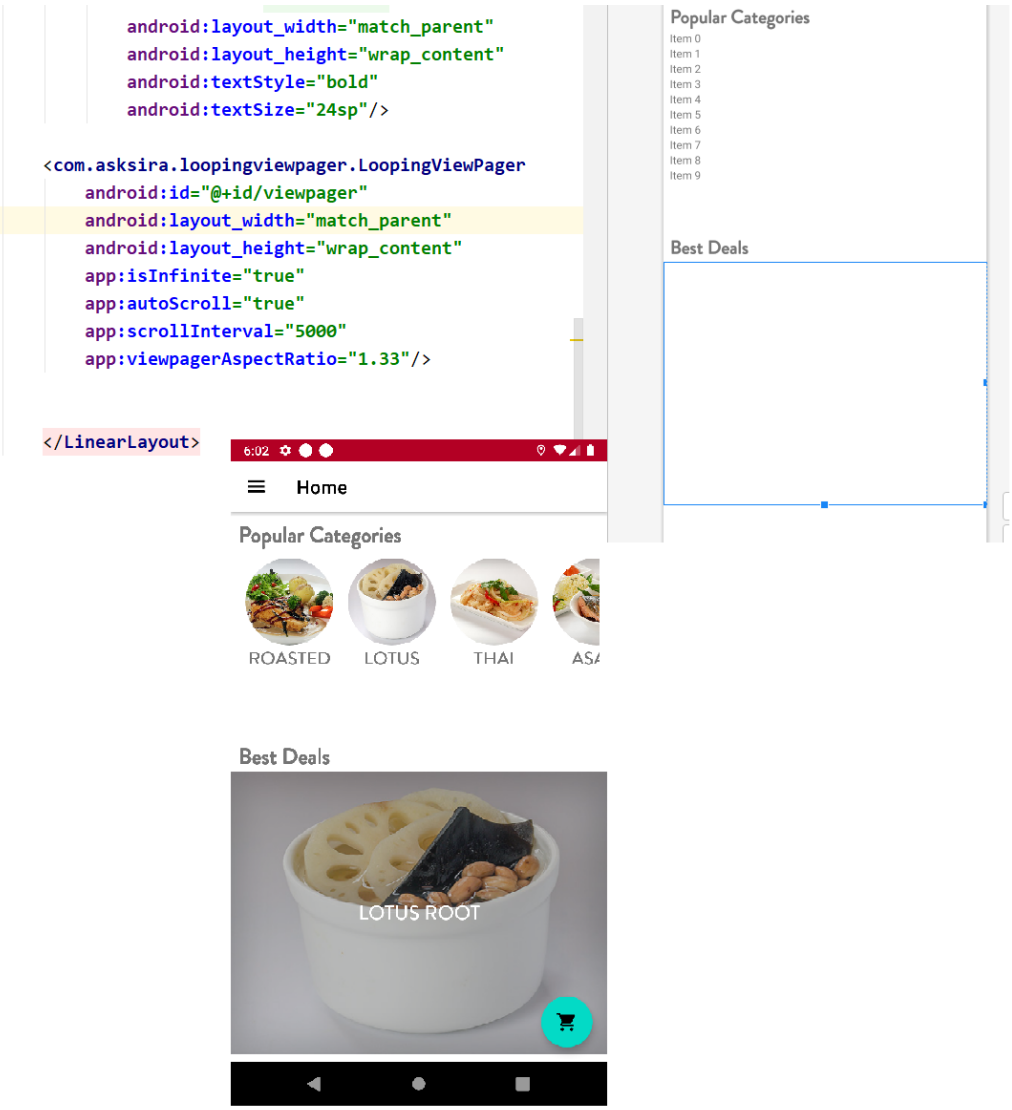
LoobingViewPager kütüphanesi isminden viewler döngülü bir şekilde (Slider) göstermesini sağlar, uygulamamda ana sayfada yemeklerin döngülü bir şekilde sürekli görüntülemelerini göstermek istediğim için Looping view pager kullanmayı seçtim.

```
<com.asksira.loopingviewpager.LoopingViewPager
    android:id="@+id/viewpager"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:isInfinite="true"
    app:autoScroll="true"
    app:scrollInterval="5000"
    app:viewpagerAspectRatio="1.33"/>
```

```
homeViewModel.getBestDealList().observe(this,bestDealModels -> {
    MyBestDealsAdapter adapter = new MyBestDealsAdapter(getContext(),
```

```
bestDealModels,isInfinite true);
    viewPager.setAdapter(adapter);
});
```

```
@Override
public void onResume(){
    super.onResume();
    viewPager.resumeAutoScroll();
}

@Override
public void onPause(){
    viewPager.pauseAutoScroll();
    super.onPause();
}
```



Şekil 4.2. Best Deals altındaki slide – looping view pager kütüphanesi ile yapılabilir

Yukarıdaki Best Deals kısmı sürekli sürekli kayma yapıyor, ilk önce istediğimiz sınıf içerisinde viewPager'i tanımladıktan sonra, kaydırma devam etme ve durdurma k için onResume() ve onPause() metotlarda ayarlar ekleriz.

4.6. Glide Kütüphasnesi

Gradle.build dosyasına:

```
implementation 'com.github.bumptech.glide:glide:4.11.0'
annotationProcessor 'com.github.bumptech.glide:compiler:4.11.0'
```

Glide Android işletim sistemi için Bump Technologies tarafından geliştirilmiş resim, gif ve yerel video'ları kolayca uygulamamıza dahil etmek, memory ve disk cache gibi olayları düşünmeden hızlıca uygulama geliştirmek için geliştirilmiş açık kaynak bir kütüphanedir.[6]

Projemde yemeklerin resimleri işlemek için glide kütüphanesi kullandım, böylece hafiza kullanımı açısından resimler iternetten indirilirken daha hızlı load işlemi yapar.

```
private void displayInfo(FoodModel foodModel) {

//buradaki glide ile resim firebase veri tabanından çekme işlemi
//gösteriyor

    Glide.with(getContext()).load(foodModel.getImage()).into(img_food);
    food_name.setText(new StringBuilder(foodModel.getName()));
    food_description.setText(new
StringBuilder(foodModel.getDescription()));
    food_price.setText(new
StringBuilder(foodModel.getPrice().toString()));
```

4.7. Circle Image View

Bu kütüphane resimlerin stylleri şekillendiren bir kütüphanedir,

Kullanımı, gradle.build dosyasına:

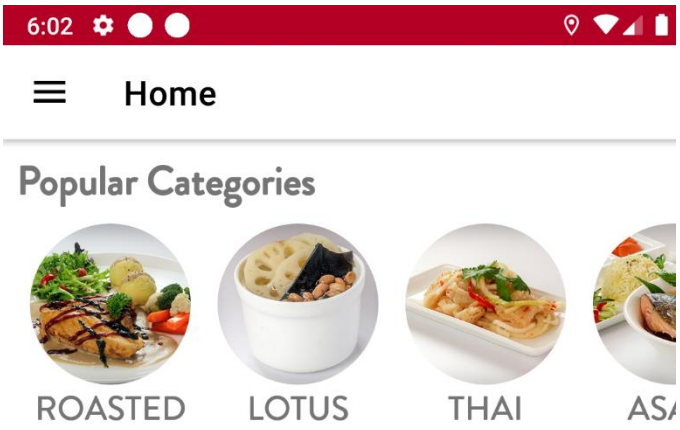
```
implementation 'de.hdodenhof:circleimageview:3.1.0'
```

XML'de

```
<de.hdodenhof.circleimageview.CircleImageView
    android:id="@+id/category_image"
    android:layout_width="96dp"
    android:layout_height="96dp"
    />
```

```
@Override
public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {

Glide.with(context).load(popularCategoryModelList.get(position).getImage()
).into(holder.category_image);
```

Sonuç:

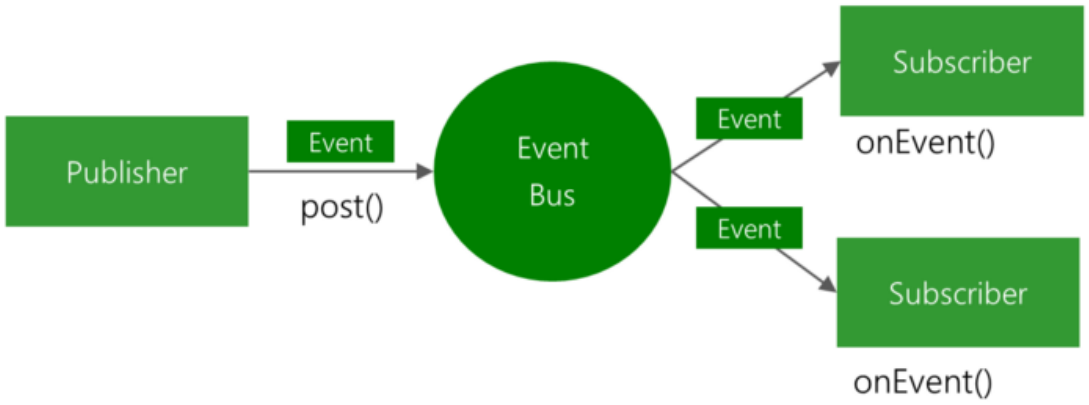


Şekil 4.3. Circle image view kütüphanesi

4.8. Event Bus

EventBus; Android için Activiteler, Fragmentler, Threadler, Servisler bileşenler arasındaki iletişimi daha az kod ile çok daha basit hale getirmek için optimize edilmiş bir publish/subscribe kütüphanesidir.[7]

Çalışma yapısı ise aşağıdaki diyagramda gösterebilirim



Şekil 4.4. Event Bus çalışma mantığı

Projemde en çok bu kütüphaneyi kullandım, çok yerlerde kodların yeniden kullanılabilirlikleri arttırmak için işime çok yaradı.

Kullanımından basitçe örnekle göstereceğim, diyelim bir müşterinin her bir ürün seçince sepete eklenmeli, eklendikçe sepete üzerindeki sayı arttırılması gerekir. Bunu gerçekleştirmek için CounterItemsInCart() metodu çağrılmalı.

CounterItemsInCart() HomeActivity sınıfında bulunuyor, ancak sepete ürün ekleme işlemi farklı sınıflardan yapılacak. Şu işlemi düzenli bir şekilde yapmak için EventBus kütüphanesi buna bakar, alttaki kod parçasında sepete bir ürün eklendiğinde işlenecek

İlk başta EventBus i projemize runtime zamanında bağlamalıyız

```
@Override
protected void onStart() {
```

```
    super.onStart();
    EventBus.getDefault().register(this);

}
```

Sonra da Eventi çağırmak istediğimiz yerde … burada bir ürün eklendiğinde eventiye firing yapacağız:

```
@Override
public void onSuccess(Integer integer) {
    Toast.makeText(getContext(), "Update Cart Success",
Toast.LENGTH_SHORT).show();
    EventBus.getDefault().postSticky(new CounterCartEvent(true));
}
```

CounterCartEvent normal bir POJO Java sınıfıdır, iki tane metot kapsıyor: isSuccess() ve setSuccess().

Uygulamanın ana sayfasında (Kod olarak HomeActivity yani), şu @Subscribe ekleriz

```
@Subscribe(sticky = true, threadMode = ThreadMode.MAIN)
public void onCartCounter(CounterCartEvent event)
{
    if (event.isSuccess())
    {
        countCartItem();
    }
}

…. // Kod devamı
```

Sonra da

countCartItem() { /*TODO*/ } metodu çağrılır. Böylece Event i gerçekleştirmiş oluruz.

En son uygulamanın çalışma yaşamında HomeActivity sınıfı yaşamı bittiğinde, yani bir Activity nin onStop() metodunda EventBus ine unbind işlemi yapacağız:

```
@Override
protected void onStop() {
    EventBus.getDefault().unregister(this);
    super.onStop();
}
```

4.9. Counter Fab

Counter Fab kütüphanesi bir buton üzerinde sayı göstermek ve arttırma yada azalma işlemi sağlayabilen bir kütüphanedir. Bilindiği üzere yaptığım uygulama bir E-Restoran yani E-ticaret mağazası ve sepete olmak şarttır. Sepet içerisinde kaç tane ürün göstermek için bu kütüphane kullandım.

Çalışma yapısı ise ilk önce gradle.build dosyasına

```
implementation 'com.github.andremion:counterfab:1.2.0'
```

Sonrasında kullanacağımız layout içerisinde bir XML tag'i ekleyelim

```
<com.andremion.counterfab.CounterFab
    android:id="@+id/btnCart"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_baseline_shopping_cart_24"
    app:backgroundTint= "@android:color/white"
    app:elevation="6dp"
    app:layout_anchor="@id/app_bar_layout"
    app:layout_anchorGravity="bottom|right|end"
    app:pressedTranslationZ="12dp"
```

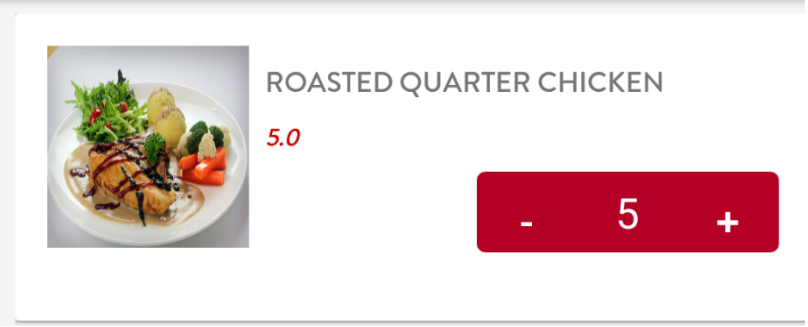
En son kodumuz içerisinde bağlama ve çağırma işlemi

```
@BindView(R.id.btnCart)
CounterFab btnCart;

// … diğer kodlar
// ..
// .
@Override
public void onSuccess(Integer integer) {
    fab.setCount(integer);
}
```

Şekil 4.5. Sepete simgesi üzerinde sayı gösterimi

4.10. Cepheuen Elegent Number Button

Bu kütüphanede XML ile kullanılıyor, bir buton sağında ve solunda eksi ve art işaretleri koymak için kullanılır, tabii ki şu eski ve artı işaretler sadece bir style değildir; onlar bu şekilde kullanır:



Şekil 4.6. – ve + botunlar view kütüphanesi

```
<com.cepheuen.elegantnumberbutton.view.ElegantNumberButton
    android:layout_gravity="right"
    android:id="@+id/number_button"
    android:layout_width="150dp"
    android:layout_height="40dp"
    android:layout_marginBottom="18dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    app:backGroundColor="@color/colorAccent"
    app:finalNumber="20"
    app:initialNumber="1"
    app:textColor="@android:color/white"
    app:textSize="8sp"/>
```

// … Bir java sınıfı içinde

```
@BindView(R.id.number_button)
ElegantNumberButton numberButton;
```

//gösterilecek sayı ayarlama

```
holder.numberButton.setNumber(String.valueOf(cartItemList.get(position).ge
tFoodQuantity()));
```

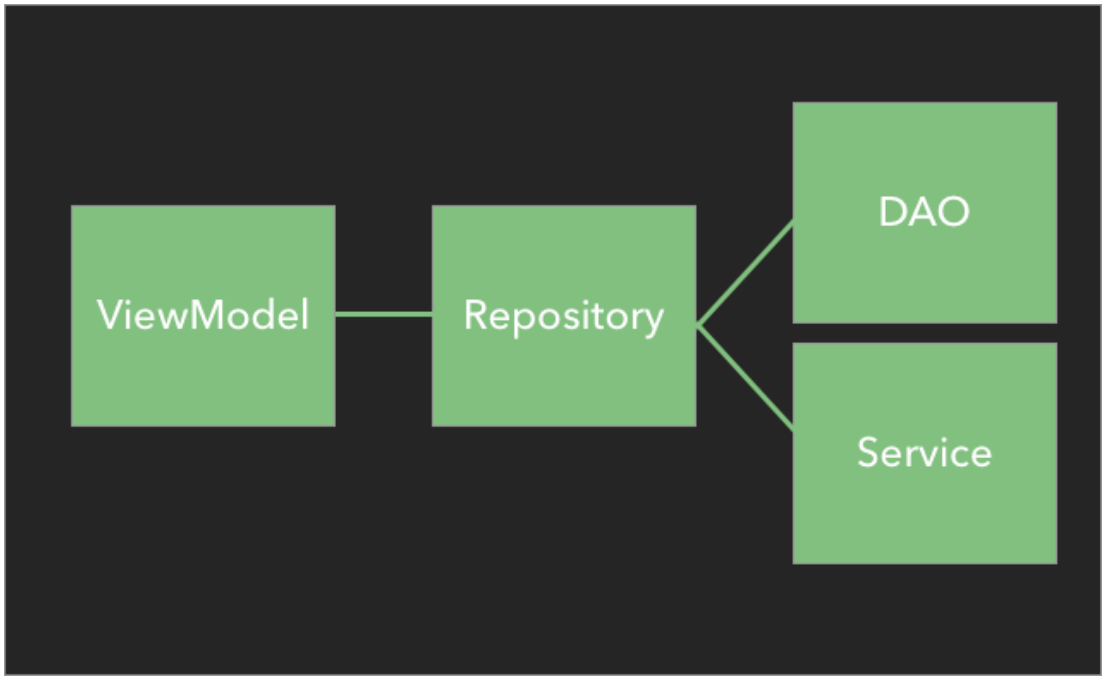
// Sepet içinde ürün sayısı güncelle

```
//Event
holder.numberButton.setOnValueChangeListener((view, oldValue, newValue) ->
{
    //when user click this button we will update our database
```

```
    cartItemList.get(position).setFoodQuantity(newValue);
    EventBus.getDefault().postSticky(new
UpdateItemInCart(cartItemList.get(position)));
});
```

4.11. RxJava Rooms

Uygulamayı çevrimdışı olarak çalıştırmak için dahili bir veri tabanı kullanmalıdır. Tam bu noktada RxJava Rooms kütüphanesi bu kolaylık bize sağlar.



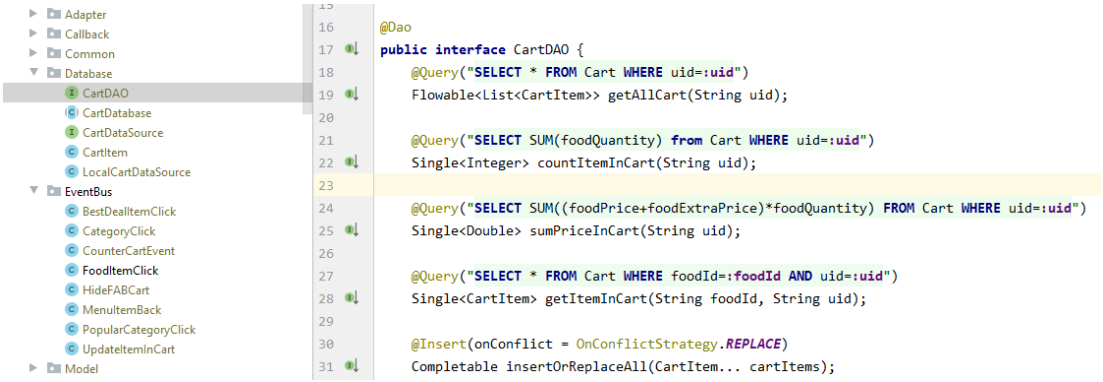
Şekil 4.7. RxJava Rooms dahili veritabanı çalışma yapısı

4.11.1. RxJava çalışma yapısı

- ViewModel, Repository'i çağırır ve çekilecek verilerin nasıl alınacağını umursamıyor,
- Repository, veri silme, veri ekleme ve veri güncelleme işlemlerine bakar,
- Repository verileri kaydetmek, güncellemek ve sorgulamak için bir DAO (Veri erişim nesnesi – Data Access Object) 'ne ihtiyacı var
- Repository -Havuz- canlı veri'yi almak için bir RemoteService'e (Retrofit, firebase etc.) ihtiyacı var.

Sonradan, yapısı oluşturduktan sonra çalışma döngüyü aşağıdaki şekilde devam eder

- Veri tabanından bilgi Repository'i sorgular ve hemen ViewModel'e gönderir (Yükleme durumu ile).
- Repository, uzak veri tabanından veri'yi alır ve dahili VT'ya yerleştirir
- Repository, Dahili veri tabanından sorgular ve ViewModel'e gönderir (Başarı durumuyla). [8]



Şekil 4.8. RxJava rooms nasıl çalışır parça kodu

4.12. Firebase Ui Auth

"Kullanıcılar, Firebase Authentication sayesinde zaten kullandıkları ve güvendikleri bir sistemi kullanarak uygulamanızda oturum açarlar. Uygulamanız daha sonra kullanıcının verilerini güvenli bir şekilde buluta kaydeder ve tüm cihazlarında aynı kişiselleştirilmiş deneyimi sunar." Developer.android.com

Uygulamamda doğrulama metodu olarak Telefon numarayla doğrulama yaptım, bu metot sayısından her kullanıcıya eşsiz bir ID anahtarı oluşturuyoruz. Firebase UI Auth kendi bir arayüzü tanımlar, kullanıcıya ilk başta o ekran gelir ve telefon numarası ekleyerek uygulamaya erişebilir.

Firebase hizmetleri ve özellikle Doğrulama işlemi gerçekleştirmek için Android studio içinden kolayca yapılabilir, Android studio'dan Tool menü > Firebase seçeneğe tıklayarak ve sonradan diğer talimat takip ederek yapılabilir.
En başta gradle.build dosyasında kullanacağımız kütüphaneler eklenmeli

```
implementation 'com.firebaseui:firebase-ui-auth:5.1.0'
```

sonrada bizim MainActivity yani uygulamada çalışacak ilk Activity'yi bir firebase auth nesnesi oluşturup kullanırız, aynı zamanda aşağıdaki kod parçasında Location erişim izni istenmesi gösteriyor, bunu kullanıcıya daha iyi bir kullanıcı deneyimi sağlamak için kullanıcını bulunduğu ülkede, ülke telefon kodu seçmek için yapılıyor ve böylece kullanıcıyı kendi ülkesine aramak yerine biz ona direk sunarız.

```
listener = firebaseAuth -> {

    Dexter.withActivity(this)
            .withPermission(Manifest.permission.ACCESS_FINE_LOCATION)
            .withListener(new PermissionListener() {
                @Override
                public void onPermissionGranted(PermissionGrantedResponse
response) {

                    FirebaseUser user = firebaseAuth.getCurrentUser();

                    if (user != null) {
                        //Account is already logged in
                        CheckUserFromFirebase(user);
                        // Toast.makeText(MainActivity.this, "Already
Logged In", Toast.LENGTH_SHORT).show();
                    } else {
                        phoneLogIn();
}
```

4.13. Karumi Dexter

Dexter, uygulama çalışma zamanında izin erişimleri talep etme sürecini basitleştiren bir Android kütüphanedir. [9]

Tarihte geriye bir az geri dönersek, izinler bir uygulama yükleme zamanında kullanıcılardan erişim izinleri hepsi aynı anda isteniyordu. Sonra da Android Marshmallow - Android 6.0 – versiyonunda izinleri kullanıcı uygulamayı kullanmak

esnasında, yeni bir özellik kullanmak istediğinde -Resim çekecek mesele, kameraya erişim izni istenirdir – ama bunu yapmak için epey kod yazmak gerekiyordu. Dexter kütüphanesi bunu daha basit ve kolay bir şekilde yapar.
Gradle.build dosyasına Dexter kütüphanesi ekleyelim

```
implementation 'com.karumi:dexter:6.2.1'
```

Sonrada yukarıda hatırlarsanız telefon numara ile doğrulama yapacağımız zaman dikkat ederseniz Dexter.withActivity {} diye bloğu var, onun içinde erişmek istediğimiz özelliği kullanmadan önce izinler isteyip ve kodumuz yazarız.

Bu işlem daha basit bir kodla gösterecek olursam github/dexter kütüphanenin sayfasında bu örnek ekliyorum

Kütüphaneyi kullanmak için geçerli bir Context ile çağrılması yeter:

```
public MyActivity extends Activity {
        @Override public void onCreate() {
                super.onCreate();
                Dexter.withContext(activity)
                        .withPermission(permission)
                        .withListener(listener)
                        .check();
        }
}
```

4.14. GMS Konum Servisi

Google Mobil Servisleri -GMS- ile bilinen Google tarafından sağlanan konumlar servisidir, uygulamama harita entegre etmek için bu kütüphaneyi kullandım, ilk önce gereken kütüphaneyi ekledikten sonra Android studioda File > New > Google > Google Maps Activity oluştur, bir harita activity uygulamamıza eklenir.

```
implementation 'com.google.android.gms:play-services-location:17.0.0'
```

Bu kütüphaneyi sadece bir harita konteyneri oluşturur, daha sonra bu activity içerisinde konum, yer, rotasyon API'leri kullanarak uygulamamıza gerçek zamanında kargo takibi ekleyeceğiz.

4.15. GMS Konum, Yer ve Rotasyon Google Maps API'si

Haritada konum arama, iki nokta arasında rotasyon hattı çizimi yapmak için Google Maps API'si kullandım. Bu API'yi kullanmak için Retofit kütüphanesi kullandım - Retrofit: REST APIlere kolayca erişim sağlayan bir networking kütüphanesidir – Ayrıca Harita APIsinden gelen verileri String şeklinde alınır ve bunan dolayı Retrofit Converter Scalar kullanması gerekir

```
implementation 'com.squareup.retrofit2:converter-scalars:2.9.0'
implementation 'com.google.android.gms:play-services-maps:17.0.0'
implementation 'com.google.android.gms:play-services-location:17.0.0'
```

Google Harita API hakkında önemli bilgileri vermek gerekirse, ilk önce ve önemli olan şey API eşsiz anahtarıdır,

```
<string name="google_maps_key" templateMergeStrategy="preserve"
translatable="false">AIzaSyCGgcKrK4MOnbVCb-*********</string>
```

Bu anahtar kullanarak APIden eşsiz kimliğimi doğrulamak için kullanılır, en başta API'yı initialize edelim, burada GMS Google Places servisi kullanıyoruz ve Kargocunu konumunu alıyoruz hem de aynı zamanda müşterinin sağladı teslim adresi de almak için kullanacaktır.

```
private void initPlaceClient() {
    Places.initialize(this, getString(R.string.google_maps_key));
    placesClient = Places.createClient(this);
}
```

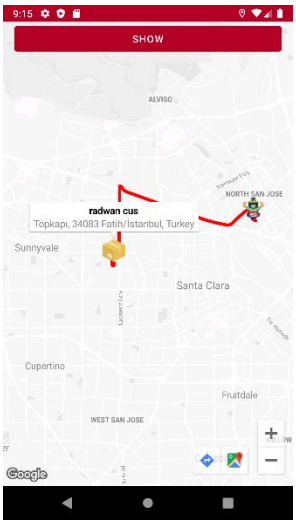
Sonrasında şu aldığım iki nokta arasında bir rotasyon çizmek için yine de Google Rotasyon Hartia APIsinden JSONObject şeklinde alınacak ve veri işledikten sonra APIden önerilen rotasyon yolu çizeceğim.

```
compositeDisposable.add(iGoogleAPI.getDirections("driving",
        "less_driving",
        from,to,
        getString(R.string.google_maps_key))
        .subscribeOn(Schedulers.io())
```

```
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(s -> {

            try {
                //Parse json
                JSONObject jsonObject = new JSONObject(s);
                JSONArray jsonArray = jsonObject.getJSONArray("routes");
                for (int i = 0; i < jsonArray.length(); i++) {
                    JSONObject route = jsonArray.getJSONObject(i);
                    JSONObject poly =
route.getJSONObject("overview_polyline");
                    String polyLine = poly.getString("points");
                    polyLineList = Common.decodePoly(polyLine);
                }
                polylineOptions = new PolylineOptions();
                polylineOptions.color(Color.RED);
                polylineOptions.width(12);
                polylineOptions.startCap(new SquareCap());
                polylineOptions.jointType(JointType.ROUND);
                polylineOptions.addAll(polyLineList);
                yellowPolyline = mMap.addPolyline(polylineOptions);
            }
```

burada Composite Disposable kütüphanesinden bir nesne oluşturdum, Composite Disposable görevei Android uygulamam içinde Activitlere yapılan bindler aynı yerden kontrol etmek amacıyla yazılan bir kütüphanedir.

Son sonuç böyle bir şey elde ederiz



Şekil 4.9. Harita API görünümü

4.16. Braintree Payments System API

Braintree Payments System API, sanal ödem sistemi sağlayan API'sidir, uygulamamda onu kullanamak için yukarıda diğer API çağrımı gibi oluyor, ilk önce Retrofite bağlayıp ve sonrasında gereken parametreler vererek API'ya bağlamış oluruz.

Braintree ödeme sistemi API'sı ile kolayca Visa kart, master kart ve benzer sanal kartlara erişim sağlayıp ödeme yaptırır.

Projeye entegre işlemi

İlk başta gradle.build dosyasına kütüpjaneyi eklenmeli

```
implementation 'com.google.android.gms:play-services-location:17.0.0'
```

Sonra Retrofit Servisi kullanarak API'ye bağlayacağız

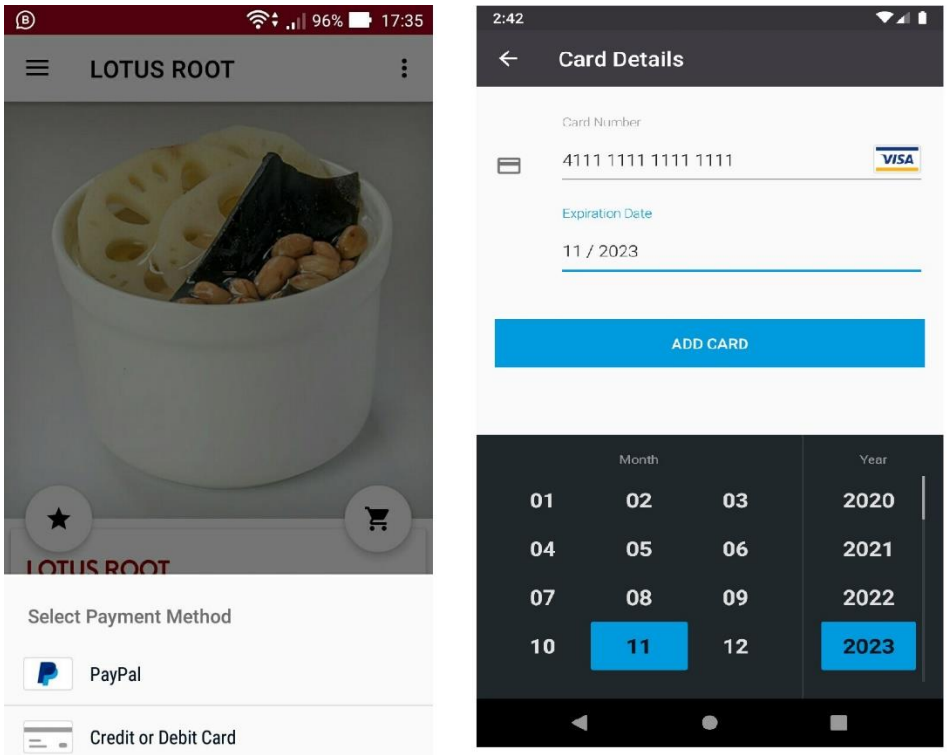
```
import retrofit2.http.Field;
import retrofit2.http.FormUrlEncoded;
import retrofit2.http.GET;
import retrofit2.http.HeaderMap;
import retrofit2.http.POST;

public interface ICloudFunctions {
    @GET("token")
    Observable <BraintreeToken> getToken (@HeaderMap Map<String, String>
headers);

    @POST("checkout")
    @FormUrlEncoded
    Observable <BraintreeTransaction> submitPayment (
            @HeaderMap Map<String, String> headers,
            @Field("amount") double amount,
            @Field("payment_method_nonce") String nonce);
}
```

Yukarıda getToken() metodu BraintreeToken diğer sınıfımızda bulunan metodu izliyor, getToken() metodu çağrıldığında ICloudFunctions arayüz içerisinde getToken() metdou işlenir ve bir kullanıcını token i getirir. Sonrasında da aynı şekilde BraintreeTransaction sınıfımızda bulunan submitPayment() metodu çağrılınca yine de

ICloudFunctions arayüzünde bulunan submitPayment çalışacak ve API'ye veri gönderecek.

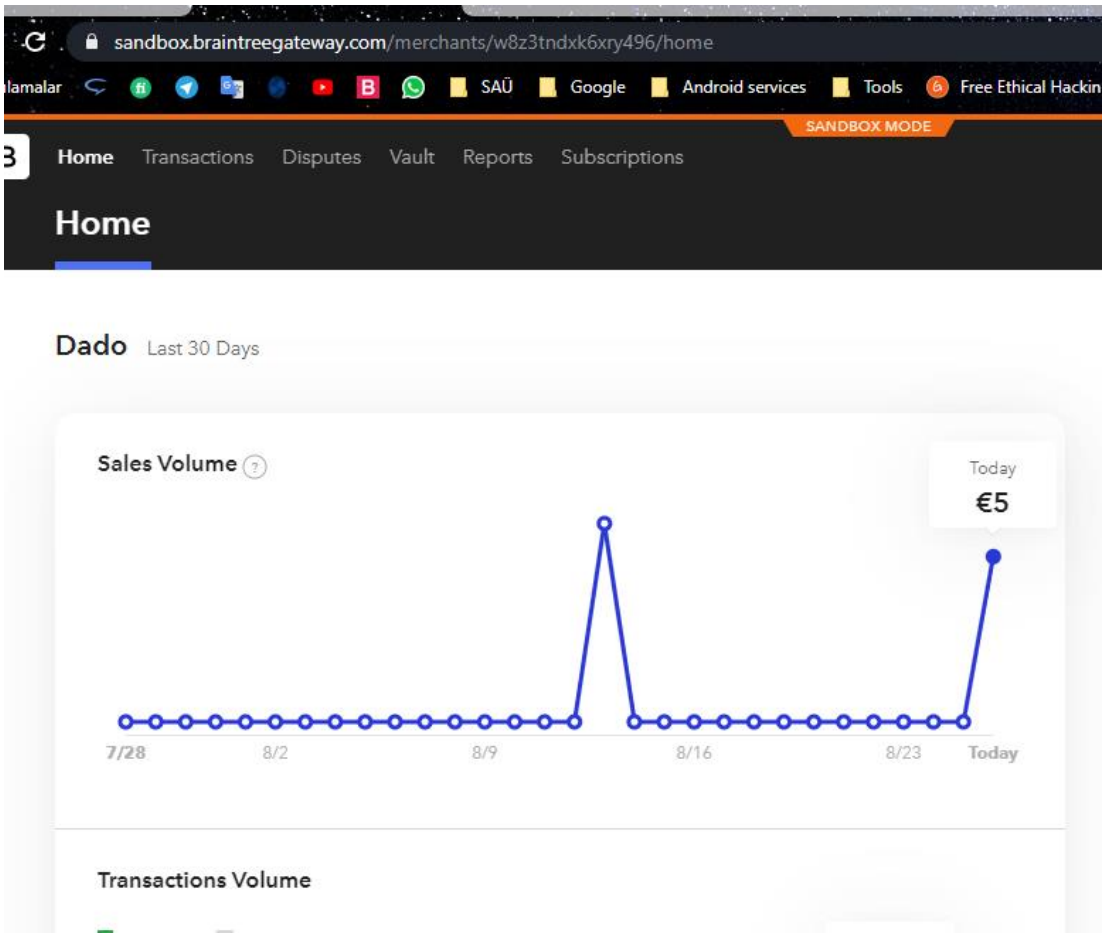
BraintreeTransaction ve BraintreeToken sınıflar bir POJO sınıflarıdır yani içerisinde sadece getter ve setter metotları içerir.

Sonuç olarak kart sınıfında gerekli kodlar yazarak ve arayüzü komponentlere bağladıktan sonra böyle bir sonuç elde ederiz



Şekil 4.10. Sanal ödeme

Bu ekran görüntüde Braintree sitesinde son 30 gün içerisinde uygulamamızda yapılan satışları gösterir



Şekil 4.11. Braintree ödeme sistemi APIsi - Dashboard

4.17. Firebase Messaging API

FCM (Firebase Cloud Messaging Yani Bulut Mesajlaşma Uygulaması) Google tarafından geliştirilen ve Kullanıcılara anlık Push-Notification göndermemize olanak sağlayan bir servistir. Firebase uygulamamızı tanıttıktan sonra Firebase yönetim konsolundan veya HTML veya XMPP protokolleri yardımıyla platform fark etmeksizin (Android, IOS vs.) Mobil cihazlara anlık bildirimler gönderebiliyoruz. [10]

Çalışma yapış

İlk önce gradle.build dosyasına FCM kütüphanesi projemize entegre edelim

Sonrasında bildirimler gönderme işlemi yapmak için bildirimler API'sine bağlanmalıyım ancak burada FCM konsolundan bize verilen doğrulama anahtarı vermemiz gerekiyordur

```
import io.reactivex.Observable;
import retrofit2.http.Body;
import retrofit2.http.Headers;
import retrofit2.http.POST;

public interface IFCMService {
    @Headers({
            "Content-Type:application/json",

"Authorization:key=AAAAKK2HWaQ:APA91bGDvCgZvaQFUB1kg17PWD5WCl7qlzBx2rh9-
y8EylcI7IOE44qfIScmrZxZosLi1a_DzmXal0uXmsQexLyiBwLtHOG5ffvhIyJg1110uZsT-
gdSrzO3PF71IEUNi7Ar****"
    })
    @POST("fcm/send")
    Observable<FCMResponse> sendNotification(@Body FCMSendData body);
}
```

Sonrasında FCM API'sinden veriyi çekmek için retrofit yardımıyla bağlanalım

```
public class RetrofitFCMClient {
    private static Retrofit instance;
    public static Retrofit getInstance(){

        if(instance == null)
            instance = new Retrofit.Builder()
                    .baseUrl("https://fcm.googleapis.com/")
                    .addConverterFactory(GsonConverterFactory.create())
```

```
.addCallAdapterFactory(RxJava2CallAdapterFactory.create())
                    .build();
        return instance;
    }
```

Gelen veriler JSON şeklinde alınır, sonrasından normal POJO (sadece getter ve setter içeren sınıflar)'a veriler aktararak istediğimiz yerde kullanabiliriz

Burada sipariş verme gösteren kod parçası, sipariş başarıla verilince, restoran/server uygulamaya bir bildirim göderiliyor.

```
cartDataSource.cleanCart(Common.currentUser.getUid())
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(new SingleObserver<Integer>() {
            @Override
            public void onSubscribe(Disposable d) {

            }

            @Override
            public void onSuccess(Integer integer) {

                //This is for the notification
                Map<String,String> notiData = new HashMap<>();
                notiData.put(Common.NOTI_TITLE, "New Order");
                notiData.put(Common.NOTI_CONTENT, "You have new order from
"+Common.currentUser.getPhone());

                FCMSendData sendData = new
FCMSendData(Common.creatTopicOrder(), notiData);


compositeDisposable.add(ifcmService.sendNotification(sendData)
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(fcmResponse -> {

                    //Order placed and notification sent
                    Toast.makeText(getContext(), "Order placed
Successfully!", Toast.LENGTH_SHORT).show();
                    EventBus.getDefault().postSticky(new
CounterCartEvent(true));

                }, throwable -> {
                    Toast.makeText(getContext(), "Order was sent but
failed to send notification!", Toast.LENGTH_SHORT).show();
                    EventBus.getDefault().postSticky(new
CounterCartEvent(true));
                }));
            }
```

4.18. Android Widgets Formatted Text

```
implementation 'com.androidwidgets:formateditext:0.2.0'
```

Yukarıdaki satir gradle.build dosyasına ekleyerek bu formatted text kütüphanesi projemize ekleyebiliriz. Bu kütüphaneyi bize bir input hangi şekilde ve biçim/formatta olacağını olanak verir. Bilindiği gibi uygulamamda "Para iade" özelliği eklemişim ve bunu ilk önce kullanıcıdan kullandığı sanal kart bilgileri almam gerektiğini duyar.

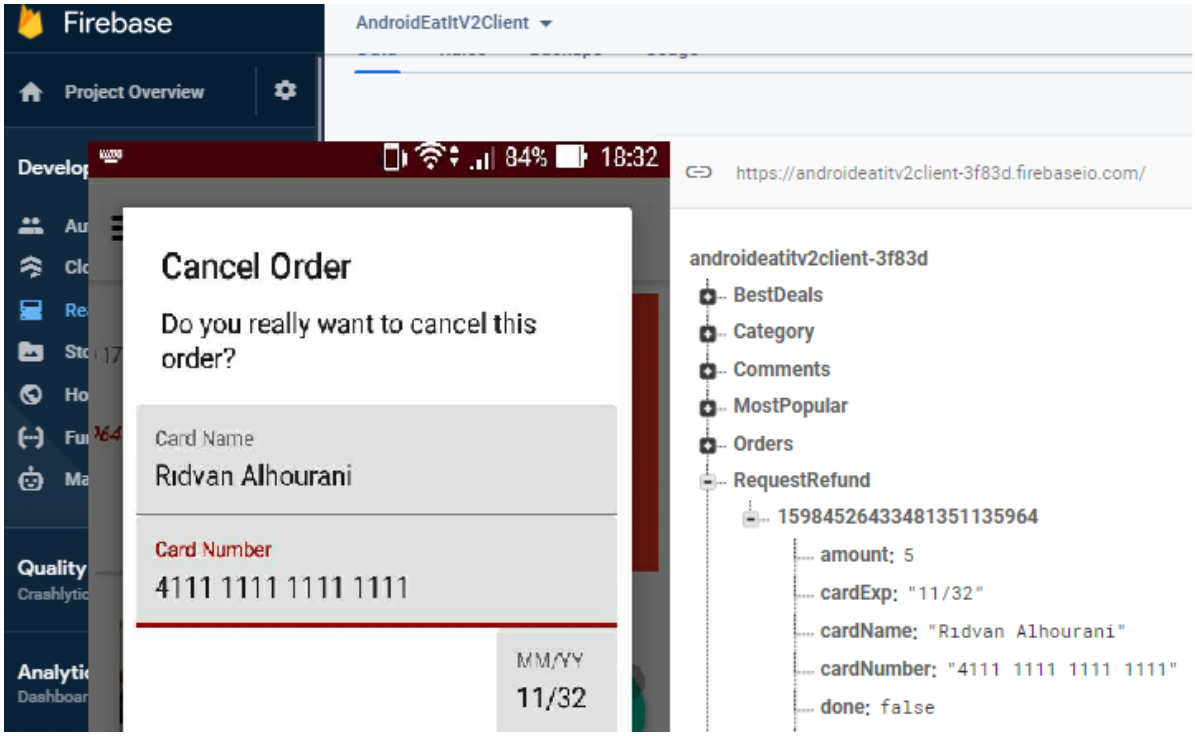
Para iade layoutunda kullanılır

```
<com.androidwidgets.formateditext.widgets.FormatEditText
    android:id="@+id/edt_card_number"
    android:hint="Card Number"
    android:inputType="number"
    android:maxLines="1"
    android:maxLength="20"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>

<com.androidwidgets.formateditext.widgets.FormatEditText
    android:id="@+id/edt_exp"
    android:hint="MM/YY"
    android:inputType="number"
    android:maxLines="1"
    android:maxLength="1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
```

Ve sonrasında Java sınıfımızda bu şekilde yukarıdaki taglara bağlanıp kullanırız

```
EditText edt_name = (EditText)
layout_refund_request.findViewById(R.id.edt_card_name);
FormatEditText edt_card_number = (FormatEditText)
layout_refund_request.findViewById(R.id.edt_card_number);
FormatEditText edt_card_exp = (FormatEditText)
layout_refund_request.findViewById(R.id.edt_exp);

//Format credit card
edt_card_number.setFormat("---- ---- ---- ----"); // 16 haneli
edt_card_exp.setFormat("--/--"); // ikişer haneli
```

Bunu en son böyle bir sonuç elde edeceğiz ayrıca format dışında herhangi bir format kabul etmeyeceği için hatasız işlem almamızı garanti ediyoruz



Şekil 4.12. Sadece 16 haneli rakam formatında kabul ettirilir

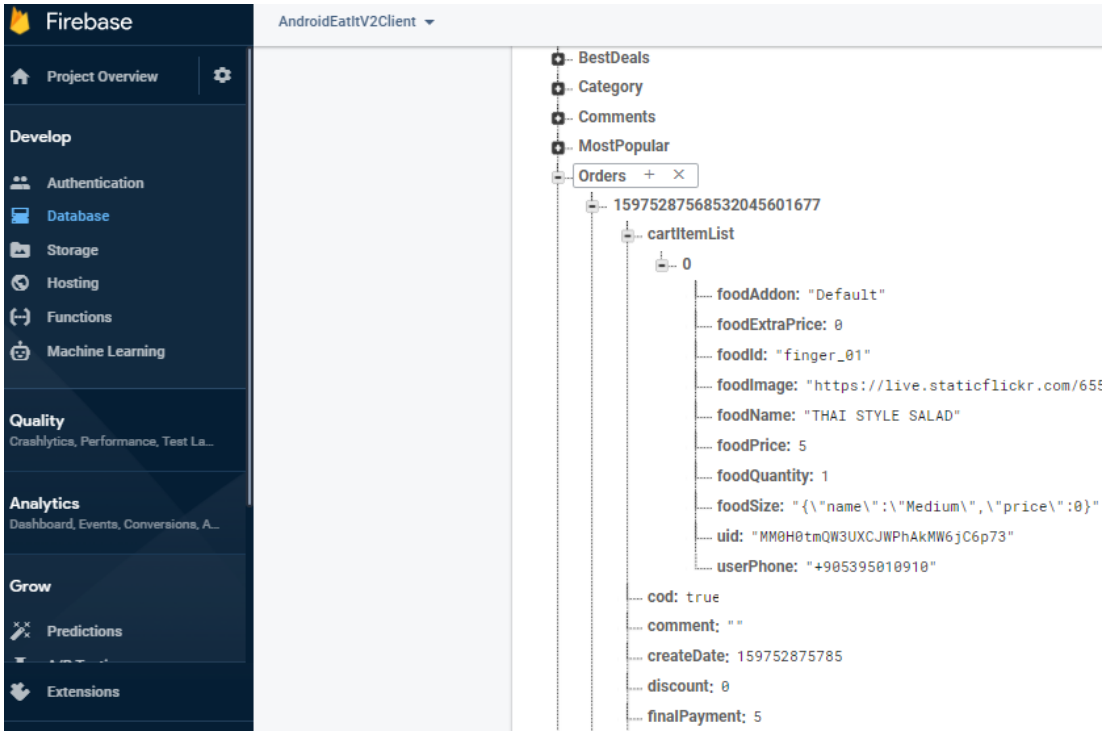
BÖLÜM 5. VERİ TABANI

Veri tabanı işlemede iki tür veri tabanına ihtiyaç vardı, birincisi bütün kullanıcılara (Firebase Cloud Database) diğeri ise herhangi bir kullanıcının kendi telefonu içinde bilgileri saklamak içindir (RXJava Rooms).

5.1. Firebase Veri Tabanı (Remote)

Google Firebase; web ve mobil uygulamalarının server tarafıyla geliştiricinin uğraşmasına gerek kalmadan kullanıcı giriş yetkilendirmeli ve verilerini gerçek zamanlı ve senkron bir şekilde tutulmasını sağlayan bir platformdur. Günümüzde ki projeler tüm markete hitap etmesi açısından iOS , Android ve web platformlarında geliştirilir fakat her platformun kendine ait yazılım dili ve bağlantı şekilleri vardır. Server-Side dediğimiz arkaplanda ki verilerin tutulması ve gerektiği zaman kullanıcıya kullanılması her platformun ortak sorunudur ve Google Firebase bu konuda geliştirilmiş ortak bir çözümdür. [2]

Firebase veritabani NoSQL olarak biliniyor, Firebase veri yapısı nasıl? Sorsak: Bir JSON ağacıdır, Cevap alınacak, her bir veri eklendiğinde bir node olacak, şu nodelere eşsiz bir anahtara sahip olacak, ve dongülü bir şekilde istediğim Ana düğümden çocuk düğümlere ulaşılır.



Şekil 5.1. Firebase Veri tabanı yapısı

5.2. Dahili Veri Tabanı (RXJava - Rooms)

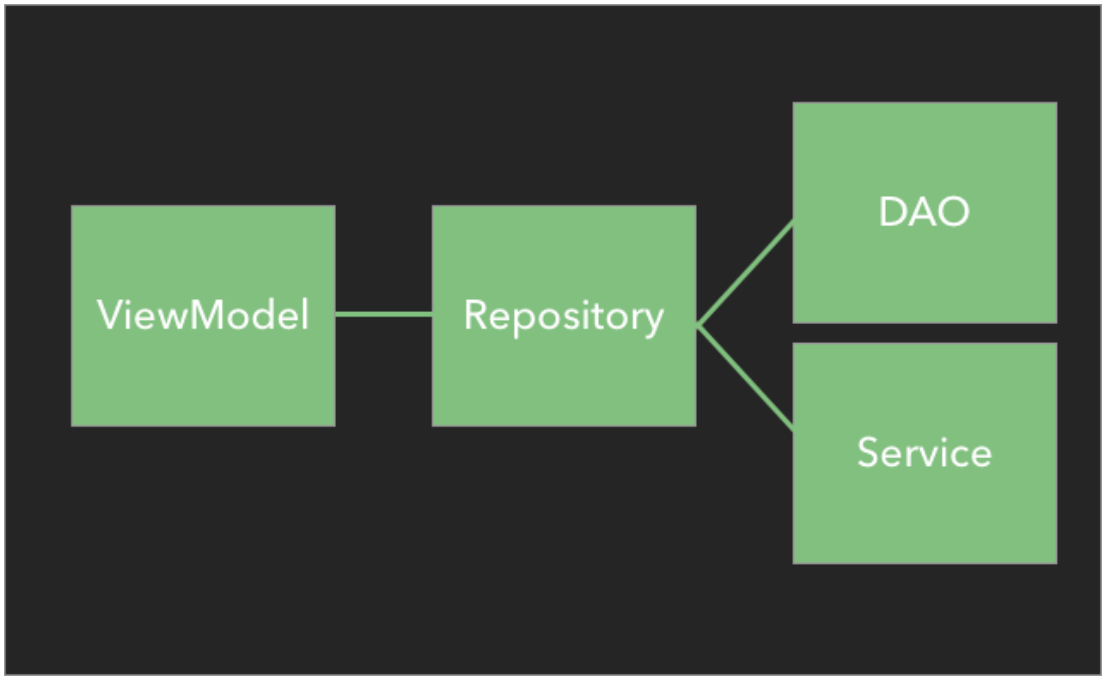
RxJava da ReactiveX'den türetilmiş java için kullanabileceğiniz bir kütüphanedir. ReactiveX yapısında iki temel unsur vardır. Bunlar Observer ve Observable.

Observable : Veriyi dışa aktaran yapıdır.

Observer : Observable tarafından dışa aktarılan veriyi dinleyen yapıdır.

Observer ,Observable'ı dinleyerek Observable tarafından aktarılan veriyi alır. Dinleme işlemi farklı threadlerde gerçekleştirebilir. [3]

Uygulamayı çevrimdışı olarak çalıştırmak için dahili bir veri tabanı kullanmalıdır. Tam bu noktada RxJava Rooms kütüphanesi bu kolaylık bize sağlar.



Şekil 5.2. RxJava Rooms çalışma yapısı

5.2.1. RxJava çalışma yapısı

- ViewModel, Repository'i çağırır ve çekilecek verilerin nasıl alınacağını umursamıyor,
- Repository, veri silme, veri ekleme ve veri güncelleme işlemlerine bakar,
- Repository verileri kaydetmek, güncellemek ve sorgulamak için bir DAO (Veri erişim nesnesi – Data Access Object) 'ne ihtiyacı var
- Repository -Havuz- canlı veri'yi almak için bir RemoteService'e (Retrofit, firebase etc.) ihtiyacı var.

Sonradan, yapısı oluşturduktan sonra çalışma döngüyü aşağıdaki şekilde devam eder

- Veri tabanından bilgi Repository'i sorgular ve hemen ViewModel'e gönderir (Yükleme durumu ile).
- Repository, uzak veri tabanından veri'yi alır ve dahili VT'ya yerleştirir

- Repository, Dahili veri tabanından sorgular ve ViewModel'e gönderir (Başarı durumuyla). [8]

Bir Room veri tabanı oluşturalım

```
@Database(version = 1,entities = CartItem.class,exportSchema = false)
public abstract class CartDatabase extends RoomDatabase {
    public abstract CartDAO cartDAO();
    private static CartDatabase instance;

    public static CartDatabase getInstance(Context context) {
        if(instance == null)
            instance =
Room.databaseBuilder(context,CartDatabase.class,"EatItV2DB2").build();
        return instance;
    }
}
//..
```

Aşağıdaki kod parçasında gösterilen CartDAO sınıfı, onu kullanarak bir Data Access nesnesi oluşturup ve uygulamanın içinde sakalarız. Kullanıcı tarafından bir değişiklik yapmadığı sürece var olan DAO'ları kullanılır, değişiklik olduğunda yukarıda anlattığım yaşam döngüye devan eder.

```
import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.OnConflictStrategy;
import androidx.room.Query;
import androidx.room.Update;

import java.util.List;

import io.reactivex.Completable;
import io.reactivex.Flowable;
import io.reactivex.Single;

@Dao
public interface CartDAO {
    @Query("SELECT * FROM Cart WHERE uid=:uid")
    Flowable<List<CartItem>> getAllCart(String uid);

    @Query("SELECT SUM(foodQuantity) from Cart WHERE uid=:uid")
    Single<Integer> countItemInCart(String uid);

    @Query("SELECT SUM((foodPrice+foodExtraPrice)*foodQuantity) FROM Cart
WHERE uid=:uid")
    Single<Double> sumPriceInCart(String uid);

    @Query("SELECT * FROM Cart WHERE foodId=:foodId AND uid=:uid")
```

```
    Single<CartItem> getItemInCart(String foodId, String uid);

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    Completable insertOrReplaceAll(CartItem... cartItems);

    @Update(onConflict = OnConflictStrategy.REPLACE)
    Single<Integer> updateCartItems(CartItem cartItem);

    @Delete
    Single<Integer> deleteCartItem(CartItem cartItem);

    @Query("DELETE FROM Cart WHERE uid=:uid")
    Single<Integer> cleanCart(String uid);

    @Query("SELECT * FROM Cart WHERE foodId=:foodId AND uid=:uid AND
foodSize=:foodSize AND foodAddon=:foodAddon")
    Single<CartItem> getItemWithAllOptionsInCart(String foodId, String
uid, String foodSize, String foodAddon);


}
```

Sonrasında istediğim yerlerde ve özellikle kullanıcı tarafından bir değişiklik yaptığında yukarıdaki metotlar çağırıp kullanacağız, HomeActivity sınıfında kart ürünler ekleme gösteren kod parçası

// Room veri taban nesnesi oluşturalım

```
cartDataSource = new
LocalCartDataSource(CartDatabase.getInstance(this).cartDAO());
```

/// Sepet bilgisi güncelleme

```
private void countCartItem() {
    cartDataSource.countItemInCart(Common.currentUser.getUid())
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(new SingleObserver<Integer>() {
                @Override
                public void onSuccess(Integer integer) {
                    fab.setCount(integer);
                }

                @Override
                public void onError(Throwable e) {
                    if(!e.getMessage().contains("Query returned empty"))
                    {
                        Toast.makeText(HomeActivity.this, "[COUNT
CART]"+e.getMessage(), Toast.LENGTH_SHORT).show();

                 }
                    else {fab.setCount(0); }
```

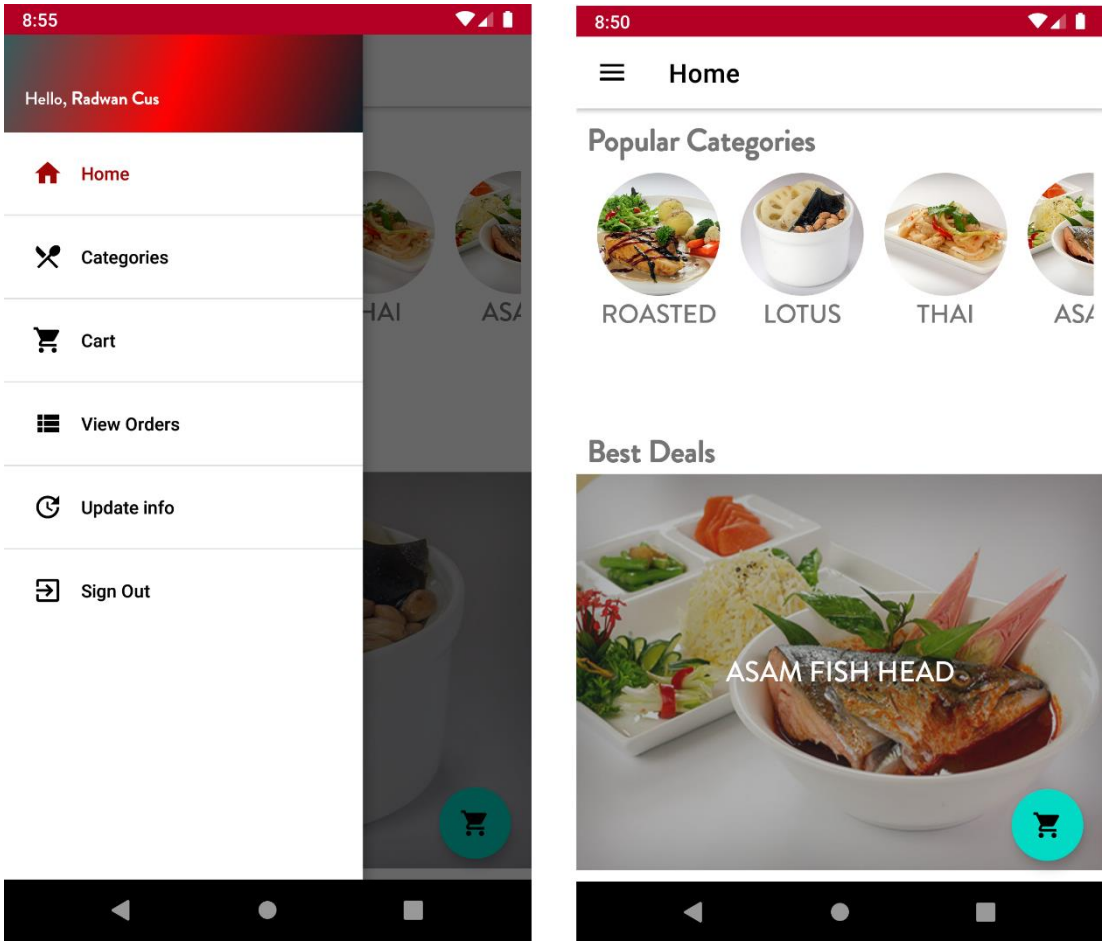
BÖLÜM 6. UYGULAMANIN ÖZELLİKLERİ

Bu kısımda uygulamanın özelliklerinden ve uygulamanın performansı denetiminde yaptığım testler ve ölçütlerinden bahsedeceğim.

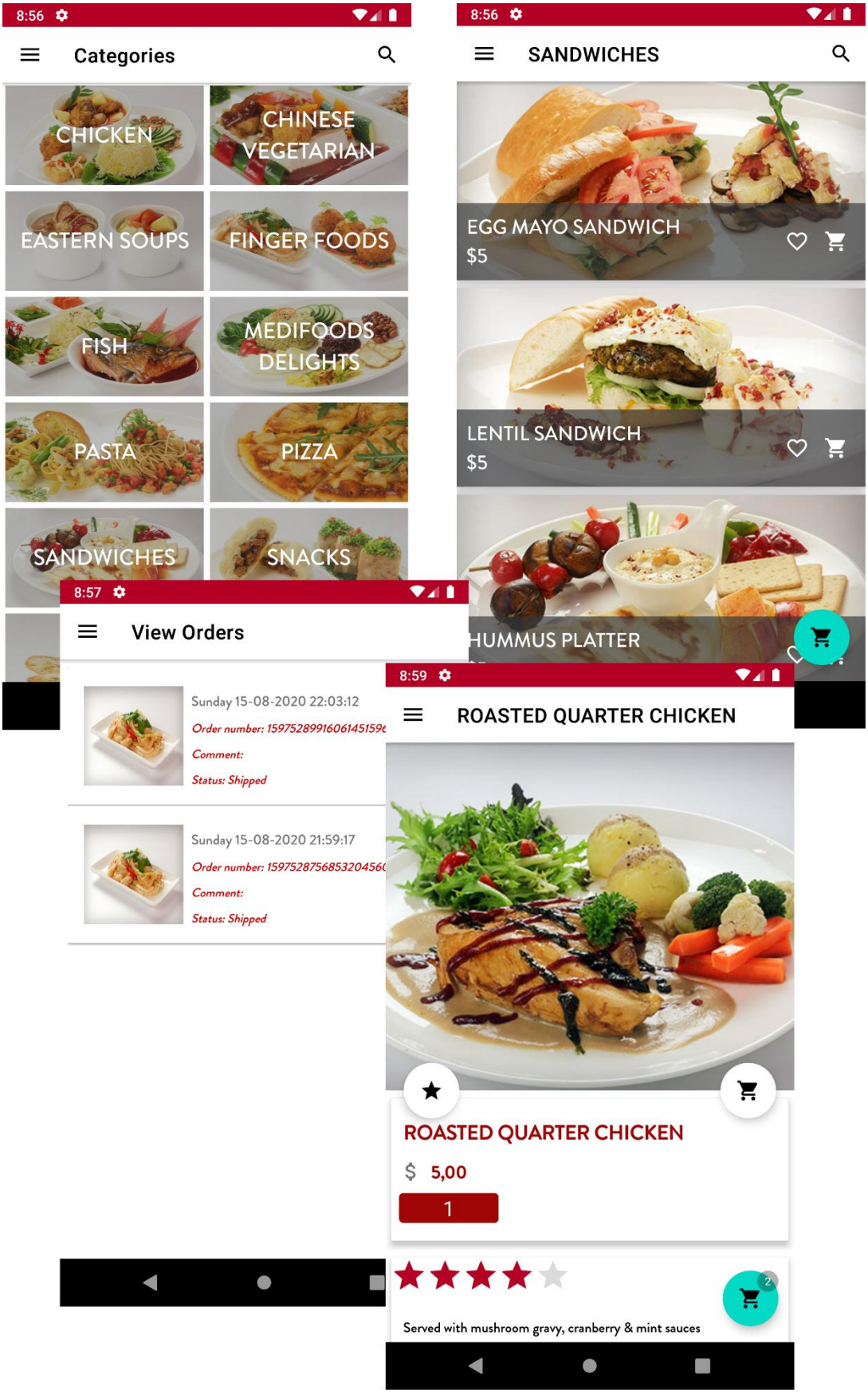
6.1. E-Restoran Client Uygulaması

6.1.1. Kullanıcı Dostu Tasarımı

Güzel ve hoş bir tasarıma sahip, basit ve anlaşılırdır.



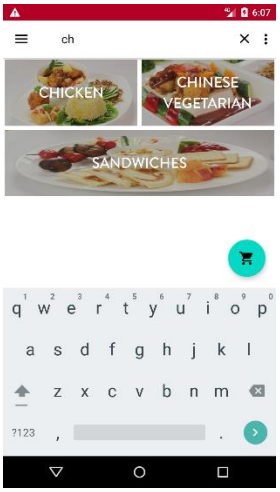
Şekil 6.1. Client Ana ekran görüntüsü



Şekil 6.2. Client UI

6.1.2. Yemek Arama

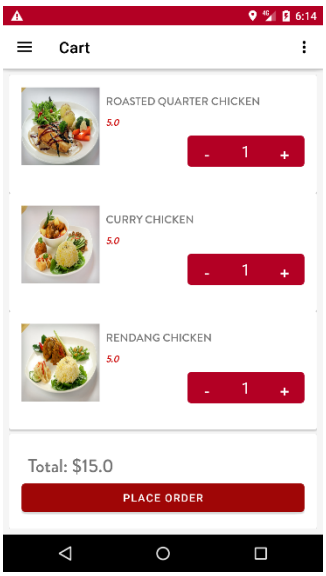
Yemek arama özelliği ile kullanıcı kolayca istediği yemeği hızlıca bulabilir ve sipariş edebilir.



Şekil 6.3. Client - Yemek arama ekran görüntüsü

6.1.3. Sepete Özelliği

Kullanıcı istediği ürünler sepete ekleyebilir, sepete eklenen ürünler detaylarını da kolayca görüntülenebilir



Şekil 6.4. Client – Sepete ekran görüntüsü

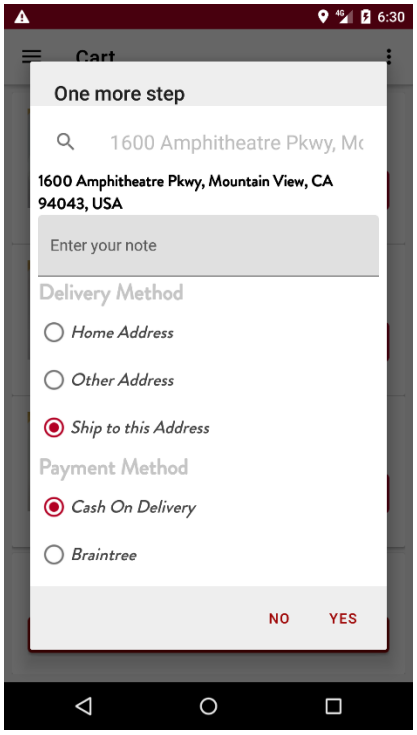
6.1.4. Sipariş Verme

Müşteri istediği ürünler sepeteye ekledikten sonra, sipariş vermek için sepeteden görüntülenebilir, istediği ürünü çoğaltabilir yada silebilir, ve en son "Sipariş ver" butonuna tıklandığında adres bilgileri istenecek, ve burada bizim uygulamamız üç seçenek verecek:

1. Ev adresi,
2. Diğer adres,
3. Ve şimdiki olan müşterinin adresi, bunu GBS kullanarak alınır.

Sonra ödeme seçenekleri de var:

- 1.Seçenek kapıda ödeme, nakit
- 2.Seçenek kredi kart kullanarak ödeme

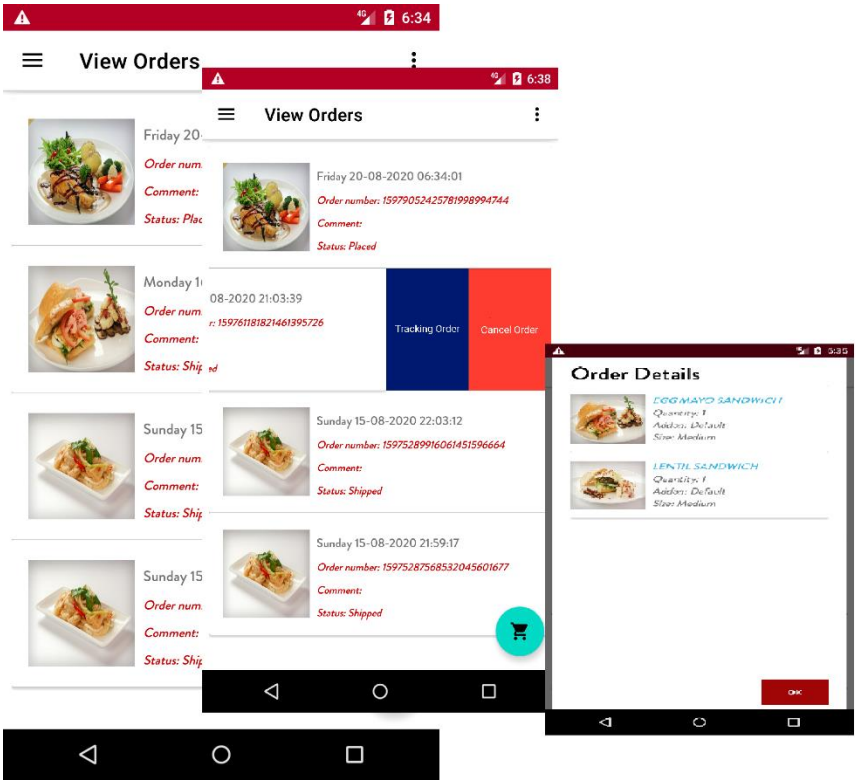


Şekil 6.5. Client – Sepete verme ekran görüntüsü

6.1.5. Sipariş Gerçek Zamanlı Takibi

Siparişler hangi aşamada takip etmek için bir arayüzü tasarladım, müşteri siparişler sayfasına giderek ettikleri siparişler görüntüleyebilir, herhangi bir sipariş sola kaydırarak iki seçenek gelecek:

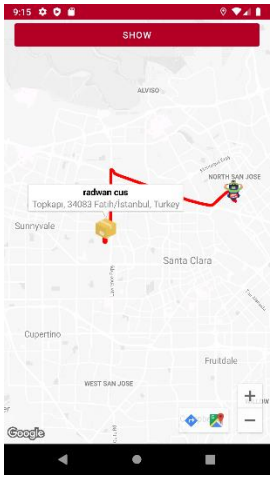
1. Sipariş iptali
    - Bu durumda müşteri siparişi iptal etmek istiyorsa, iptali gerçekleştirmeden önce para iadesi yapmak için müşteriden Visa kartı bilgisi alınacak.
2. Haritada görüntüle (Sipariş kargo aşamasında)
    - Sipariş kargo aşamasında ise müşteriyi yeni bir sayfaya yönlendirilecek ve harita üzerinde kargo nerede olduğunu görebilecek.
    - Kargo aşamasında değilse, bir uyarı mesajı görüntüleyecek, "Sizin siparişiniz kargo aşamasında değil, ve hangi aşamada olduğunu söylenecek"



Şekil 6.6. Client – Siparişlerin ekran görüntüsü

6.1.6. Gerçek Zamanlı Harita

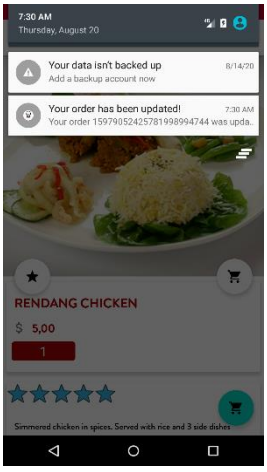
Google Locations API, Google Places API ve Google Directions API leri kullanarak, projeme gerçek zamanlı bir şekilde müşteriler kargocu nerede olduğunu görüntüleyebilirler.



Şekil 6.7. Client –Harita ekran görüntüsü

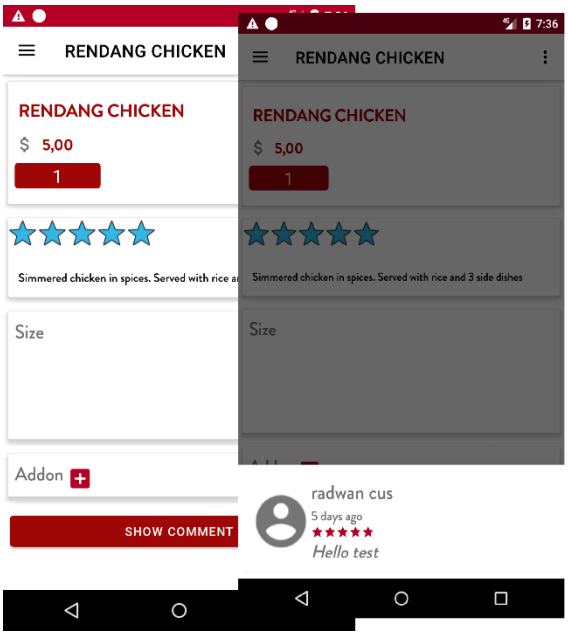
6.1.7. Bildirim Özelliği

Önemli bir özellik, FCM Firebase Cloud Mesajlaşma, Google ve Firebase tarafından sağlanan APİ'si kullanarak projeme entegre yaptım, bu bildirimler siparişlerin durumu değişime uğradığında müşteriye bir bildirim gönderilir.



Şekil 6.8. Client –Bildirimler ekran görüntüsü

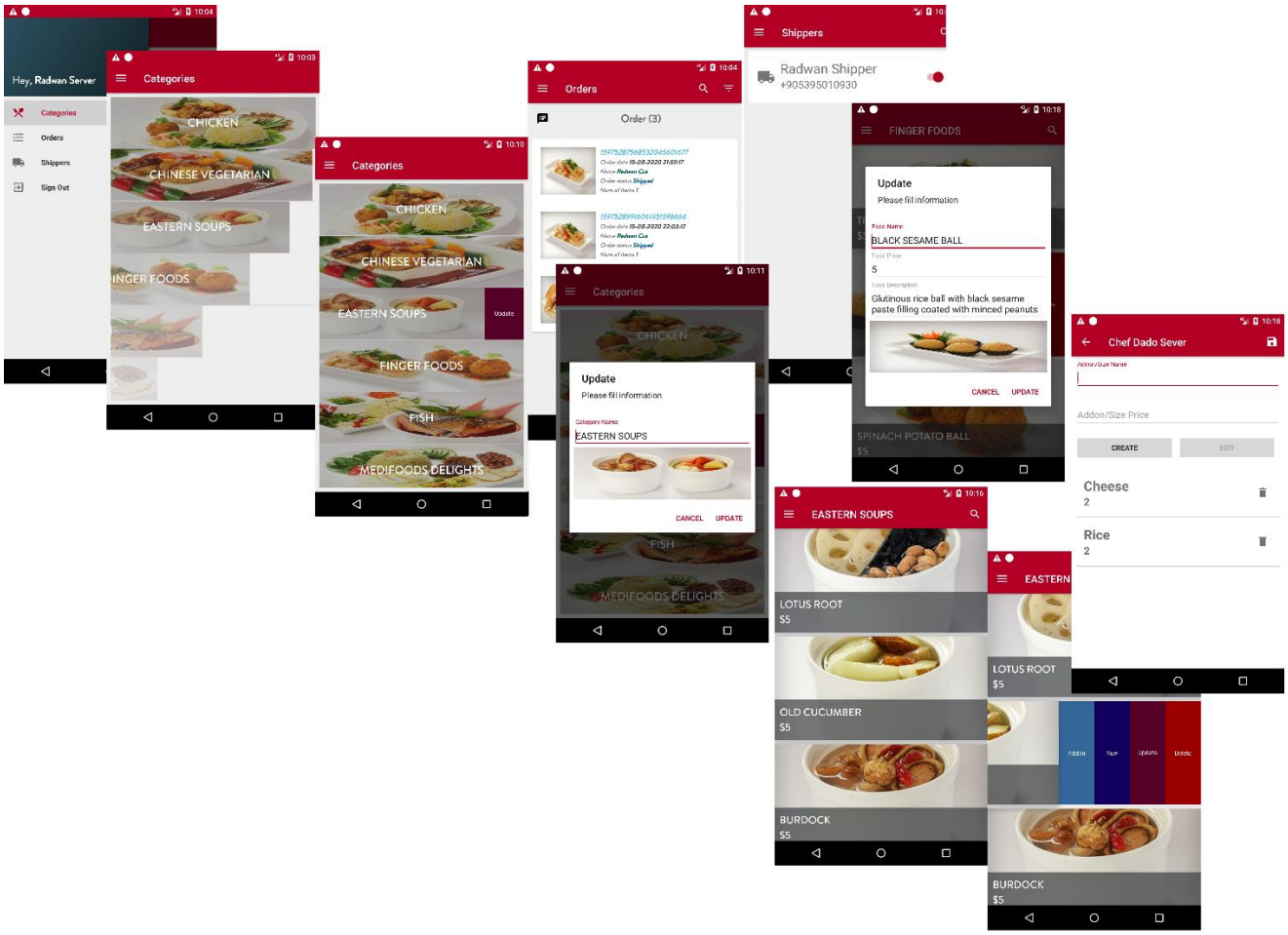
6.1.8. Yorumlar ve Değerlendirmeler



Şekil 6.9. Client –Yorumlar ve değerlendirme ekran görüntüsü

6.2. E-Restoran Server Uygulaması

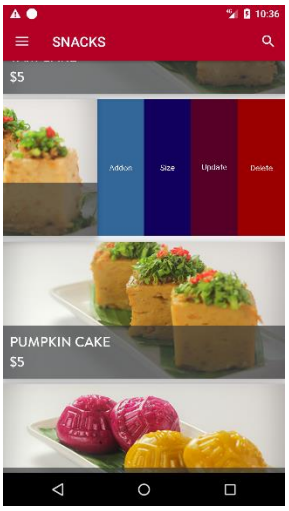
6.2.1. Hoş tasarım



Şekil 6.10. Server UI

6.2.2. E-ticaret mağaza temel özellikleri

Restoranlara yemeklerin güncelleme seçeneği veriliyor, yemek ekle, yemek sil, yemek fiyatı, resmi, ve benzeri düzeltmeler gibi E-ticaret mağazaya temel gereken özellikler. Ayrıca yemeklerle gelen eklentiler için ayrı bir arayüzü tasarımı yaptık ve veri tabanından var olan yemek eklentiler güncelleme ekleme özelliği sağladım.



Şekil 6.11. Server – Yemekler güncelle ekran görüntüsü

6.2.3. Gelen siparişler görüntüleme

Müşterilerden gelen siparişleri filtre yaparak siparişleri görüntülenebilir, ve herhangi bir sipariş üzerinde tıklanarak siparişin detayları görüntülenebilir.

Siparişi sola kaydırarak kullanıcıya 4 seçenek gelir

1. Harita üzerinde takip

Böylece restoran kargocu harita üzerinde gerçek zamanlı bir şekilde nerede olduğunu görebilir.

2. Telefon Arama

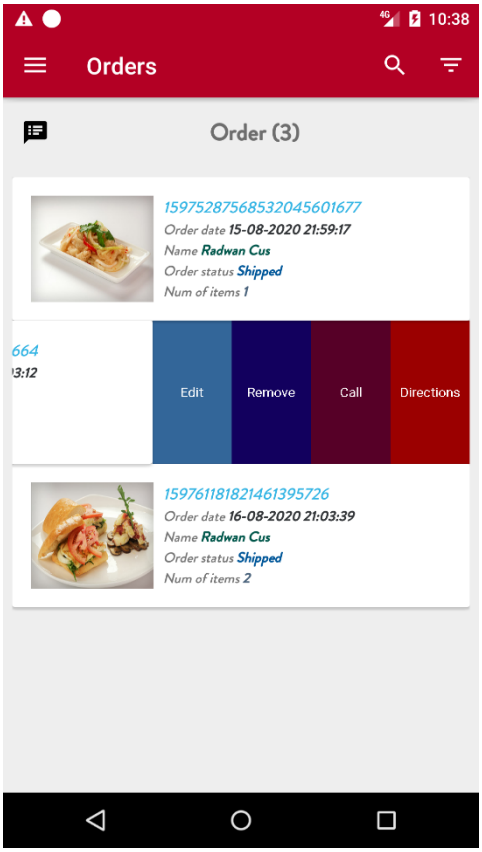
Müşterilere normal arama yaptırır

3. Düzelt

Burada restoran, siparişin durumu manuel olarak düzeltebilir, "Kargolamaya Hazır" mesele

4. Sil

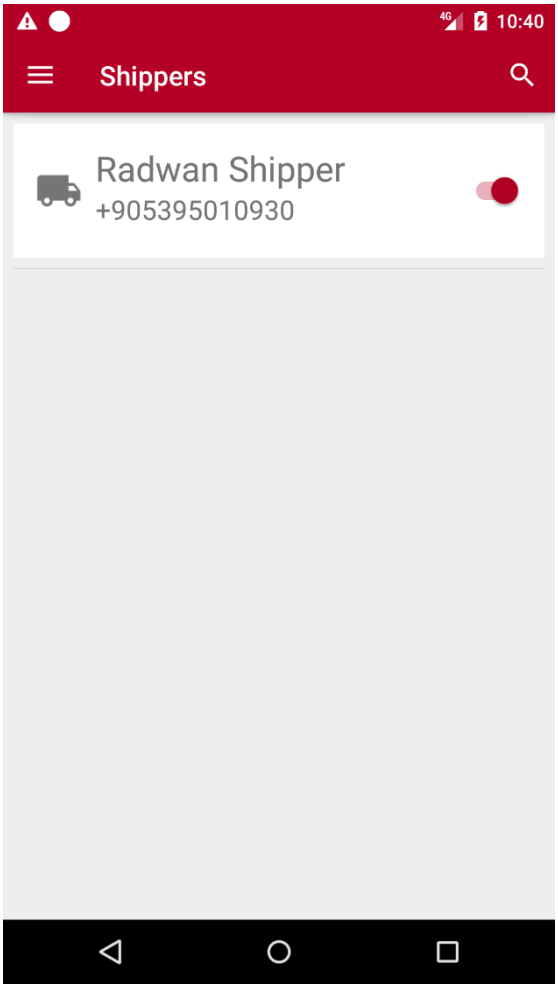
Siparişi silinebilir.



Şekil 6.12. Server – Siparişler güncelle ekran görüntüsü

6.2.4. Kargocular

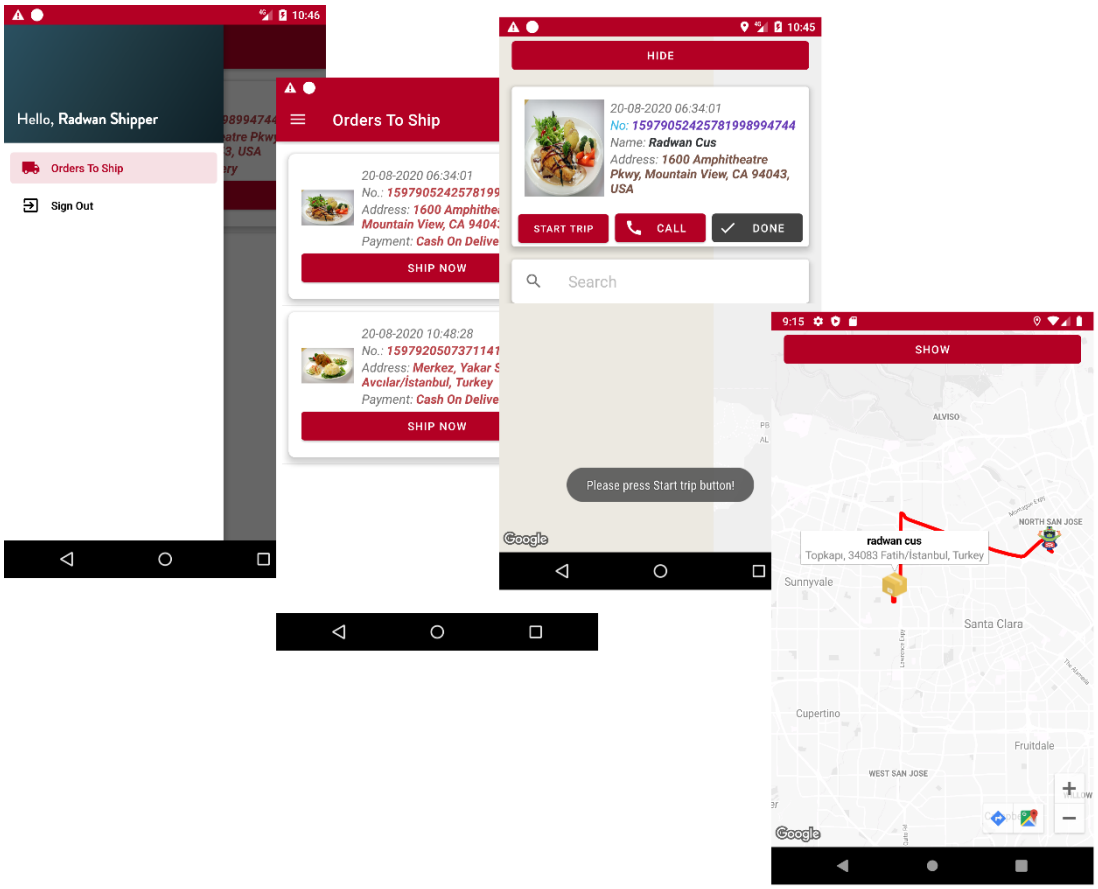
Bu ara yüzde restoranla kargocu çalışanları görüntülenebilir, restoran bu ara yüzden de kargocular aktive veya pasife alınabilir.



Şekil 6.13. Server – Kargocular listesi ekran görüntüsü

6.3. Kargocu Uygulaması Özellikleri

6.3.1. Hoş, basit ve kullanıcı dostu arayüzü



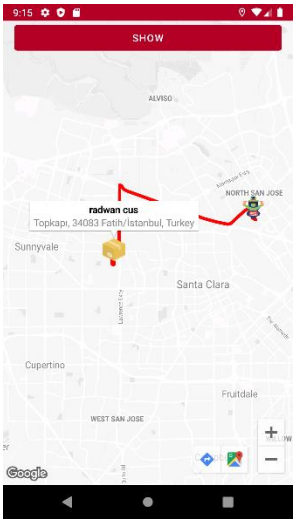
Şekil 6.14. Shipper UI

6.3.2. Harita

Kargocu müşteri bulmak için farklı yöntemler sağladım

1. Müşteri verdiği adres, direk olarak haritada APİ Directions kullanarak bir route çizer ve kargocu-müşteri arasında yol haritası oluşturur.
2. Kargocu istediği adresi aratarak Places APİ'si kullanıp Directions API'siyle özel bir rota çizdirir.
3. Kargocu müşteriye normal arama yaptırarak müşteri ile iletişime geçebilir.

4. Son özellik ise, yemeği ulaştırdığında "Verildi" butona basarak, siparişi müşteriye verildiğini, hem müşteriye hem restorana bildirim gönderme sağlanır.

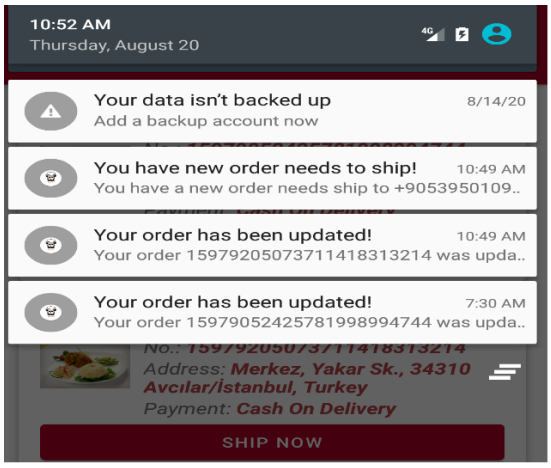


Şekil 6.15. Shipper - Harita

6.3.3. Bildirimler

Bildirimler sistemi tıpkı müşteri ve server uygulamalarda uygulandığı gibi kargocu için uygulandı, bir siparişin kargolamaya hazır flag'i ile işaretlendiğinde kargocuya "telefon numarası olan müşteriden kargo teslim edeceğiniz bir siparişiniz var" diye bildirim geliyor.

Teknik olarak FCM hazır Firebase Cloud Mesajlaşma API'si kullandım.



Şekil 6.16. Shipper - Bildirimler

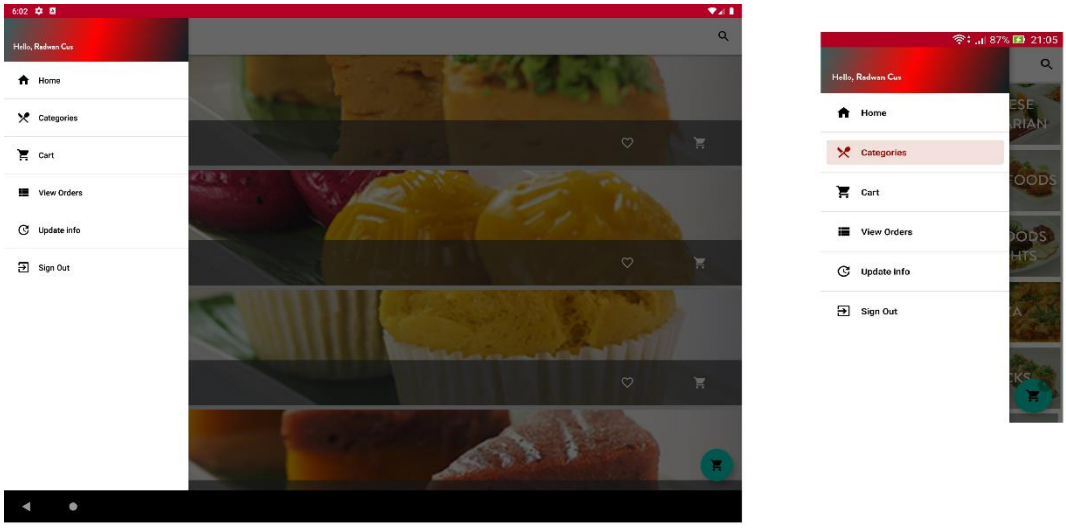
BÖLÜM 7. PERFOTMANS ÖLÇÜMÜ VE TESTLER

Uygulamanın performans değerlendirme açısından birden fazla yol izledim. İlk önce tasarım aşamasında kullanıcı dostu bir tasarım yapmaya özen gösterdim. İkinci yol ise uygulamayı bittikten sonra farklı Android sistem ve cihaz üzerinde denettim. Ve en son farklı normal kullanıcılar tarafından beta test denettim.

Tasarım hakkında yukarıda epey anlattığımdan dolayı burada daha fazla bahsetmeyeceğim ancak diğer iki testi hakkında, izlediğim yol ve nasıl bir hata mekanizma uyguladığımdan bahsedeceğim.

7.1. Farklı Android Cihazı üzerinde Hafıza ve Ekran Çözünürlüğü denetimi

- **Ekran çözünürlüğü** Tablet, telefon, ve farklı akıllı telefon ekran boyutu üzerinde denedim, renkler, yazı boyutları ve ekran yataya çevirdiğimizde görüntüler nasıl olacağını ve gerekli bir düzeltme olduğunda uyguladım.



Şekil 7.1. Farklı ekran çözünürlüğü denetimi

- **Hafıza kullanımı** açısından ise, uygulamayı export yaptıktan sonra ve gerçek fiziksel cihazlar üzerinde çalıştırdığımda boyutunu ve hafıza kullanımı bir az büyük olduğunu fark ettim. Böylece bir araştırma yaptıktan sonra benim uygulamam içinde var olan Log kod satırları ve var olan ve kullanılmayan

kaynak dosyaları sildim. Ve en son da Android tarafından optimize edilmiş "Bundle APK" export seçeneği uyguladım.

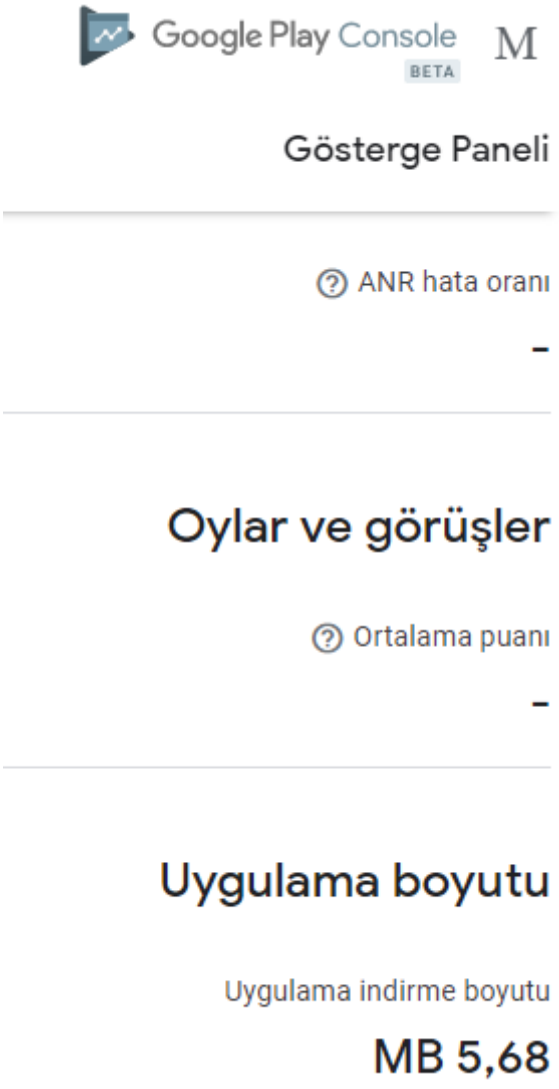
Bundle APK, kullanıcı kullandığı Android cihazına özel tasarlanmış bir export'tur. Bunu daha fazla anlatmak için bir siteden alıntı yapıyorum:

"Android uygulamalarımızı build edip, cihazımıza yüklemek istediğimizde .apk uzantılı bir dosya oluşturulur.
Oluşturulan bu dosya içerisinde uygulamada kullanılan stringler, ekran çözünürlüğüne göre resimler, sesler, CPU mimarileri ve diğer tüm kaynakları barındırır.
Apk'nın yüklendiği cihaz, bu kaynakların hepsine birden ihtiyaç duymadığından dolayı, kullanılmayan kaynaklar gereksiz yer kaplanmasına sebep olacak ve kullanıcı cihaz hafızası konusunda sıkıntılar yaşayabilecektir.

Uygulamanın cihazda kapladığı alana göre kullanıcılar uygulamayı kaldırabilir ve bu durum da ciddi bir kullanıcı kaybına neden olabilir.
İşte bu gibi sıkıntılardan dolayı, uygulama boyutunun büyük olmaması için App Bundle 'ı projenizde kullanarak optimize yapmanız gerekir." [11]

Bunu yaptıktan sonra uygulamanın indirme boyutu sadece 5.68 MB oldu. Sonra da uygulamanın bir cihaz üzerine yükleyip çalışıldığı halde 20 ~ 40 MB arasında cihazın türü ve hangi Android sistemin üzerinde ilişkin bir boyut almaktadır.

Aşağıdaki resim Google Play mağazasına yüklemek çalıştığım Müşteri uygulaması indirme boyutu gösteriyor.



Şekil 7.2. Google Play mağazasında uygulamayı indirme boyutu

7.2. Testler ve Hata Denetimi

Uygulamayı geliştirme aşamasında hep Emulator cihazı üzerinde denetimler yapıyordum ve genelde fazla hata karşılaşmadım ancak uygulamayı son aşamada gerçek bir cihaza yükleyip çalıştırmayı çalıştığımda normalden fazla uygulamayı çökmesine neden olan hatalar aldım. Böylece hatalar nerede olduklarını tespit ettikten sonra bug'ler çözmeye başladım.

Bazı hatalar mantıksal hatalardı, bazıları ise 3. Parti API lardan oluşan hatalardır.

Mantıksal hatalar programlı olarak çözdükten sonra, diğer hatalar yani API'den gelen bilgiler bazen boş bilgi verdiği için bir NULL pointer Expetion hatası alıyordum, bunu engellemek için Try-Catch blokları kullandım, ve sonra da kullanıcıya bir hata masajı göstermeyi çalıştım ve nasıl bir hareket etmesini -kullanıcı- bilgi vermeye çalıştım.

Uygulamayı en son durumunda kullanıcı uygulamayı kullandığı sürece güzel bir kullanıcı deneyim sağladım. Ve en azından uygulamada herhangi bir sebepten dolayı bir özellik çalışmadığı taktirde, uygulamayı çökmez durumda oldu.

Aşağıdaki kod parçasında Google Harita API'sinden bilgi alınıyor ama örnek atıyorum bizim paketçi kargocu denizler geçemediği için, müşterinin adresi başka bir ülkede olduğunu varsayarak, motorcu, müşteri bulunduğu adrese ulaşmak için; Google API motorcu için bir rota çizmeyi deneyecek ama bir deniz / engel var olduğundan herhangi bir bilgi geri veremeyecek ve bunu NULL hatası oluşturup uygulamayı çökmesine neden olacak. Bunu Try-Catch kullanarak hata ne olduğunu kullanıcıya belirttim.

```
compositeDisposable.add(iGoogleAPI.getDirections("driving",
        "less_driving",
        from,to,
        getString(R.string.google_maps_key))
        .subscribeOn(Schedulers.io())
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe(s -> {

//JsonObject boş olacak ve böylece bir NULL hatası fırlatacak

            try {
                //Parse json
                JSONObject jsonObject = new JSONObject(s);
                JSONArray jsonArray = jsonObject.getJSONArray("routes");
```

```
                for (int i = 0; i < jsonArray.length(); i++) {
                    JSONObject route = jsonArray.getJSONObject(i);
                    JSONObject poly =
route.getJSONObject("overview_polyline");// api parametresi
                    String polyLine = poly.getString("points");
                    polyLineList = Common.decodePoly(polyLine); }
                polylineOptions = new PolylineOptions();
                polylineOptions.color(Color.RED);
                polylineOptions.width(12);
                polylineOptions.startCap(new SquareCap());
                polylineOptions.jointType(JointType.ROUND);
                polylineOptions.addAll(polyLineList);
                yellowPolyline = mMap.addPolyline(polylineOptions);
            }
            catch (Exception e)
     { // burada hatayı yakalyıp ve bir text mesajı kullanıcıya gösterilir
                Toast.makeText(this, "Uygun bir yol
bulunmadı"+e.getMessage(), Toast.LENGTH_SHORT).show(); }
```

BÖLÜM 8. SONUÇLAR VE ÖNERİLER

Uygulamayı Google Play mağazasına yüklemeden önce uygulamayı birden fazla cihaz ve farklı Android sistem versiyonları üzerinde denemeler yaptım, o testlere dayalı sonuçlar gereken Error Handling teknikleri uyguladım.

Sonuç olarak uygulamalar min. SDK'si 21 [Android sistemi 5.0] ve mak. “targetSDKversionu” 29 olarak ayarladım. Bunu 120,000’den fazla Android cihazı üzerinde çalışmasını sağlanır, yani %92 Android cihazlardandır.

Uygulamalar birbiriyle bilgi paylaşımı ve gerçek zamanlı bilgi dağıtımı yapılıyor, bunu bu cağda çok önemli faktörlerden sayılıyor.

Ayrıca uygulamanın kullanıcı dostu tasarımları ile kullanıcıyı uygulama içinde daha fazla zaman ayarlamasına neden oldu

Böyle mobile yönelik E-Ticaret uygulamaları gittikçe önemi artıyor.

KAYNAKLAR

[1] https://tr.wikipedia.org/wiki/Android_(i%C5%9Fletim_sistemi).

[2] https://medium.com/furkanpacikgoz/google-firebase-nedir-ae013e495a74

[3] https://medium.com/@smhdk

[4] https://gelecegiyazanlar.turkcell.com.tr/blog/butterknife-nasil-calisir

[5] http://emre-kose.net/retrofit-nedir-nasil-kullanilir/

[6] https://medium.com/@ekrem.hatipoglu/kotlin-android-glide-kullan%C4%B1m%C4%B1-part-1-83c0b5a1c908

[7] https://medium.com/mobiletech/k%C4%B1sa-k%C4%B1sa-eventbus-nedir-339493d3ee6a

[8] https://medium.com/@iammert/offline-app-with-rxjava-2-and-room-ccd0b5c18101

[9] https://github.com/Karumi/Dexter

[10] https://mobilokulum.wordpress.com/2017/12/13/fcm-nedir/

[11] https://www.mobilhanem.com/android-app-bundle-kullanimi/

ÖZGEÇMİŞ

Rıdvan Alhourani, 01.01.1996 de Damascus'da doğdu. İlk, orta ve lise eğitimini Jober'de tamamladı. 2014 yılında Al-Saade özel Fen Lisesi'nden mezun oldu. 2015 yılında Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nü kazandı. 2017 yılında SAÜ Bilişim Fakültesinde yazılım stajını yapmıştır. 2015 yılında başladığı Sakarya Üniversitesi Bilgisayar Mühendisliği bölümünde lisans eğitimini sürdürmektedir.

BSM 498 BİTİRME ÇALIŞMASI
DEĞERLENDİRME VE SÖZLÜ SINAV TUTANAĞI

KONU : Mobil E-ticaret Android Uygulaması (E-Restoran)
ÖĞRENCİLER (G1512.10575/Rıdvan Alhourani):

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): DOÇ. DR. CÜNEYT BAYILMIŞ
DANIŞMAN İMZASI: