ÇANAKKALE ONSEKİZ MART ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

ÇANAKKALE ÖĞRENCİ YAŞAMI

BEYZA ÇOBAN - BAŞAK KILIÇ

ALPER KÜRŞAT UYSAL

Haziran, 2021
ÇANAKKALE

ÇANAKKALE ÖĞRENCİ YAŞAMI

Çanakkale Onsekiz Mart Üniversitesi Mühendislik Fakültesi
Bitirme Ödevi
Bilgisayar Mühendisliği Bölümü

BAŞAK KILIÇ – BEYZA ÇOBAN

ALPER KÜRŞAT UYSAL

Haziran, 2021
ÇANAKKALE

Beyza ÇOBAN ve Başak KILIÇ, tarafından **Doç. Dr. Alper Kürşat UYSAL** yönetiminde hazırlanan **"ÇANAKKALE ÖĞRENCİ YAŞAMI"** başlıklı çalışma tarafımızdan okunmuş, kapsamı ve niteliği açısından bir Bitirme Ödevi olarak kabul edilmiştir.

.....

Danışman

Bölüm Başkanı
Bilgisayar Mühendisliği Bölümü

TEŞEKKÜR

Bitirme projemizin tüm aşamaları boyunca bilgilerini bizimle paylaşan ve desteğini hiçbir zaman esirgemeyen danışman hocamız ; Doç. Dr. Alper Kürşat UYSAL'a, bugüne kadar edindiğimiz bilgileri kazanmamızı sağlayan tüm hocalarımıza, dönem boyunca tüm zorlukları aşıp her aşamada elimizden gelenin en iyisini yapmaya çalışarak gerek proje yönetimini gerekse dostluğumuzu en güzel şekilde devam ettirip projemizi tamamladığımız için birbirimize ve hayatımızın her evresinde olduğu gibi bu zorlu süreçte de bize destek olup güç veren ailelerimize sonsuz teşekkürlerimizi sunuyoruz.

BAŞAK KILIÇ – BEYZA ÇOBAN

İÇİNDEKİLER

1. GİRİŞ:

Bitirme projesi konusu Çanakkale'de Öğrenci yaşamı olarak belirlenmiştir.

Bu projenin uygulamayı kullanacak üniversite öğrencilerine birçok alanda kolaylık sağlayacağı ve çeşitli işlemleri hızlandıracağı düşünülmektedir.

2. İLGİLİ ÇALIŞMALAR

2.1. Hacettepe Üniversitesi

Hacettepe Üniversitesi Resmi Mobil Uygulaması ile duyurular, transkript, danışman, yemek listesi, akademik takvim ve servis saatleri gibi çeşitli bilgilere ulaşılır.
Sisteme giriş üniversite tarafından belirlenen kullanıcı adı ve şifre ile sağlanır. Giriş yapılmadan yemek listesi, akademik takvim, duyurular ve servis saatleri menüleri görüntülenebilir. Öğrencilerin görüş, öneri, istek ve talepleri için geri bildirim menüsünü bulunur.

https://play.google.com/store/apps/details?id=mobiversite.hacettepemobil&gl=TR

2.2. Üniaktivite

Hem kampüste hem de kampüs dışında ilgi çekecek tüm aktiviteleri keşfetme imkanı sunar. Uygulama içerisinde farklı üniversitelerin de takip edilebilmesi mümkündür. Etkinliklere karekod (QR) ile giriş yapılıp detaylara harita ve konum bilgisi ile erişilebilir. Seçilen etkinliğe kimlerin geleceği görüntülenebilir. Etkinliklerin paylaşılmasına olanak sağlar. Özel indirim, kampanya ve fırsatlar sunulur.

https://play.google.com/store/apps/details?id=com.uniaktivite.android&gl=TR

2.3. Ortam

TOBB ETÜ tarafından üniversiteler için yapılan bu uygulama ile kişiselleştirilmiş topluluk sayfalarının takibi, arkadaş takibi ve üyelik başvurusu gibi özelliklerin yanı sıra eş zamanlı bilgilendirme ve özel mesajlaşma altyapısı da Ortam içinde bulunur.

https://www.horato.com/tr/case-studies/tobbetu-ortam

2.4. Sonuç

Yukarıdaki uygulamalar ve benzerleri incelendiğinde tek bir amaç üzerine tasarlandığı görülmüştür. Bunlardan bazılarında sadece öğrenci bilgi sistemi hizmeti sunulurken bazılarında üniversite topluluklarının etkinlikleri üzerine yoğunlaşılmıştır.

Araştırmalarımız sonucunda öğrencilerin üniversite hayatında ihtiyaç duyabileceği birçok alanı içinde barındıran bir uygulamanın olmadığını fark ettik. Aynı zamanda öğrencilik hayatımız süresince bu tarz bir uygulamanın eksikliğini biz de hissettik. Proje konumuzu belirlerken de bu sebepleri göz önünde bulundurduk.

3. PROJENİN TANIMI

Uygulamamız Çanakkale'de öğrenci yaşamını kolaylaştırmak adına yapılmıştır. Uygulamayı kullanmak isteyen öğrenciler üye olup giriş yapmalıdır. Projemiz iki ana başlık çerçevesinde geliştirilmişir. İlk bölümde Çanakkale Onsekiz Mart Üniversitesi'nde düzenlenen topluluk etkinlikleri üzerine yoğunlaşılmıştır. İkinci bölüm ise öğrencilerin hızlı ulaşabilmesi adına ev,eşya ilanlarının bir araya getirilmesinden oluşturulmuştur.

İlk bölümde yer alan topluluk etkinlikleri kısmında öğrenciler, düzenlenen tüm etkinlikleri görebilecek ve etkinlik hakkındaki bilgilere kolaylıkla ulaşabilecektir. Bu bilgiler içerisinde etkinliğin adı, yeri, zamanı, konusu ve konuşma yapacak katılımcıların isimleri yer almalıdır. Aynı zamanda kişilerin katılım durumlarını bildirebilecekleri bir anket içerir. Ayrıca topluluk yönetimi opsiyonel olarak bilgi ekleyebilir.

Topluluk yönetimindeki öğrenciler düzenledikleri etkinlik bilgilerini eksiksiz bir şekilde sisteme yükledikten sonra gerekli durumlarda bu bilgiler üzerinde istenilen değişikliği yapabilir.

Uygulamayı kullanmak isteyen öğrenciler kendi kriterleri doğrultusunda etkinlikleri filtreleyebilir. Seçilen etkinliğin bilgilerini görüntüledikten sonra katılmak istiyorsa bunu bildirir ve verilen izine göre ilgili tarih takvimine eklenir. Bu sayede etkinlik tarihi yaklaştığında hatırlatma bildirimi alabilir. Daha sonra etkinliğe katılacak kişileri listeleyebilir. Olumsuz bir durum oluşması halinde uygulama üzerinden katılamayacağını bildirmesi beklenir.

Bu kısımda topluluk etkinliklerine ek olarak öğrencilerin ders ilanı oluşturabileceği bir yapı sunulur. İkinci bölümde yer alan ilanlar kısmında öğrenciler Çanakkale'deki kiralık evleri ve diğer öğrenciler tarafından satılmak istenen eşyaları görüntüleyebilir. Aynı zamanda bu kısımda uygulamayı kullanacak öğrenciler ev arkadaşı da arayabilirler.
Kiraya verilecek evler için fiyat, konum, büyüklük, oda sayısı, bulunduğu kat , apartmandaki kat sayısı ve evin eşyalı olup olmadığı bilgilerine ek olarak fotoğraflar da eklenmelidir. Satılacak eşyalar için ise eşya türü, fiyatı ve fotoğrafları eklenmelidir.

Bunların yanı sıra opsiyonel olarak ve kiralık ev ve satılacak eşya bilgilerine eklemeler yapılabilir. Eklenen ilan bilgilerinde gerekli durumlarda güncelleme yapılması mümkündür.

Uygulamadan faydalanacak öğrenciler istekleri doğrultusunda filtreleyebilir. İlgilendikleri ilan sahibi ile kolay bir şekilde birebir iletişim kurabilirler.

4. SİSTEMİ KİMLER KULLANACAK

Topluluk etkinliklerinden haberdar olmak isteyen ve aynı zamanda ev, eşya, ders ilanı vermek isteyen öğrenciler. Kullanıcılar sistemi şu amaçlarla kullanır;

Öğrenci: Etkinlik, ev ve eşya ilanlarını hızlıca ulaştırmak. Diğer öğrencilerin de kolaylıkla erişimini sağlamak.

4.1. Actor - Goal Model

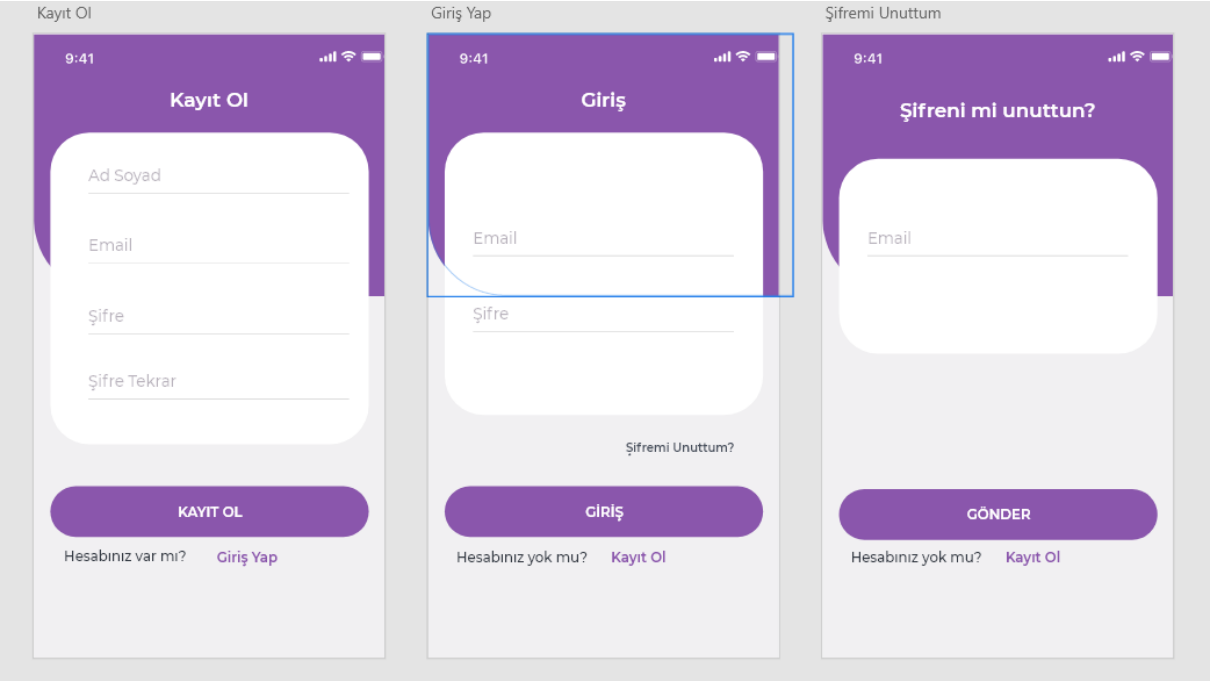
ACTOR	GOAL
Öğrenci	• Sisteme kayıt olur. • İlanların bilgi girişini gerçekleştirir. • İlanları görüntüler. • İlan bilgilerinde güncelleme yapabilir. • Kriterler doğrultusunda filtreleyebilir • Etkinliklere katılım durumlarını bildirir. • Etkinliklere katılacak kişileri listeyebilir. • Ev arkadaşı arayabilir. • İlan sahibiyle iletişim kurabilir.

5. SİSTEMİN HEDEFLERİ

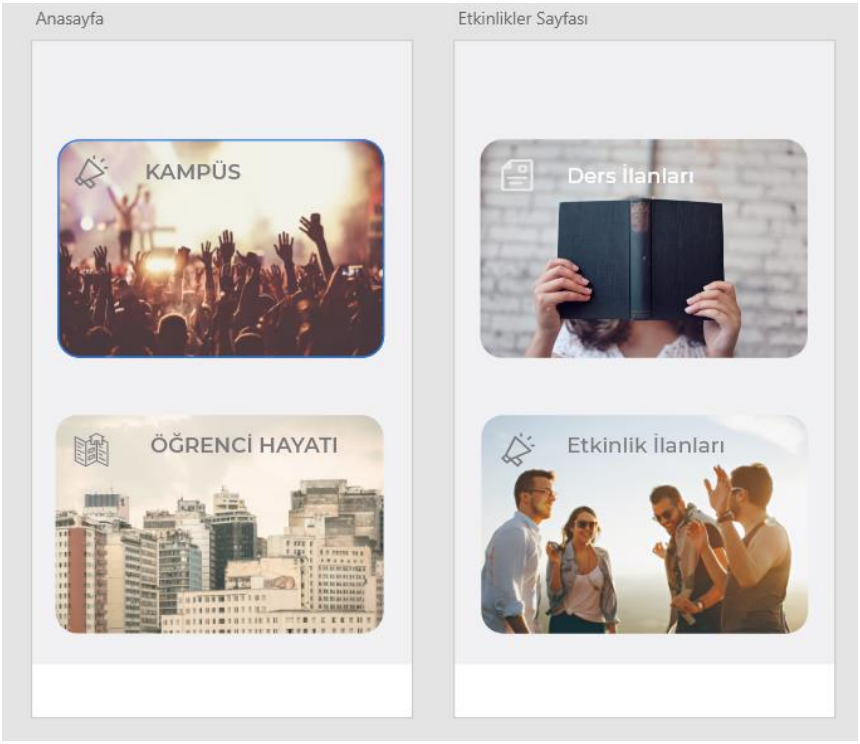
- Öğrencilerin düzenlenen topluluk etkinliklerinden haberdar olmasını sağlamak.
- Etkinliklerdeki etkinliğe katılıp katılmama durumuna göre kontenjan kontrolünü sağlamak.
- Öğrencilerin katılacağı etkinlikleri takvim üzerinden takibini ve bildirim almasını sağlamak.
- Öğrencilerin kolay ve hızlı şekilde ev ve eşya bulmasına imkan sunmak.
- Öğrencilerin kendi aralarında iletişim kurmalarına fırsat tanımak.
- Kurulan iletişimler sonucunda en ekonomik şartlarda eşya satın almak ve ev kiralayabilmek.
- Birebir kurulan iletişimler sayesinde istenilen kriterlere uygun ev arkadaşı bulmak.

6. UI TASARIMLARI

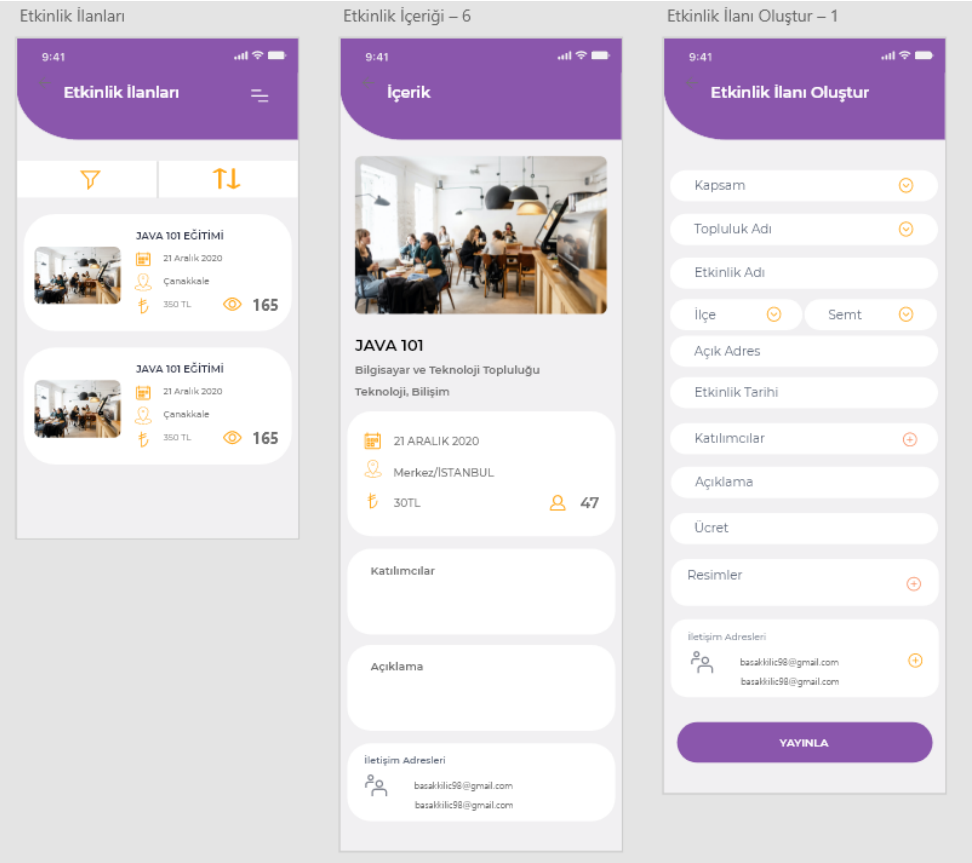
Uygulama geliştirmeye başlamadan önce kullanacağımız bileşenleri kolaylıkla belirleyebilmek ve uygulama gereksinimlerini anlayabilmek için taslak niteliğinde UI tasarımlarını oluşturduk.
Adobe XD üzerinden arayüzleri hazırladıktan sonra Prototype yardımıyla senaryo işleyişini takip ettik.



Şekil 1 Kullanıcı Giriş Ekranları



Şekil 2 Ana Sayfa, İlan Başlangıç Sayfaları



Şekil 3 İlan Bilgilendirme, Oluşturma ve Listeleme Sayfaları

7. ANALİZ - TASARIM SÜREÇLERİ

Yazılım geliştirirken sistematik ilerleyebilmek ve süreçleri daha profesyonel gerçekleştirebilmek adına ilk aşamada yazılım geliştirme metodu seçtik. Yazılımımızdaki değişiklikleri göz önünde bulundurduğumuzda iteratif ilerleyişi sağlayabilmek için RUP metodolojisine karar verdik. Bu metodolojide Deneysel Geliştirmeyi tercih ederek küçük bir sistem parçasıyla başlayıp büyük sisteme artışlı bir şekilde ulaşabilmeyi amaçladık.

Uygulamamızın Etkinlik İlanı bölümü için bu metodolojinin gerektirdiği şekilde aktiviteleri tamamladık. Aşağıda yaptığımız her bir aktivite için bir örnek bulunmaktadır.

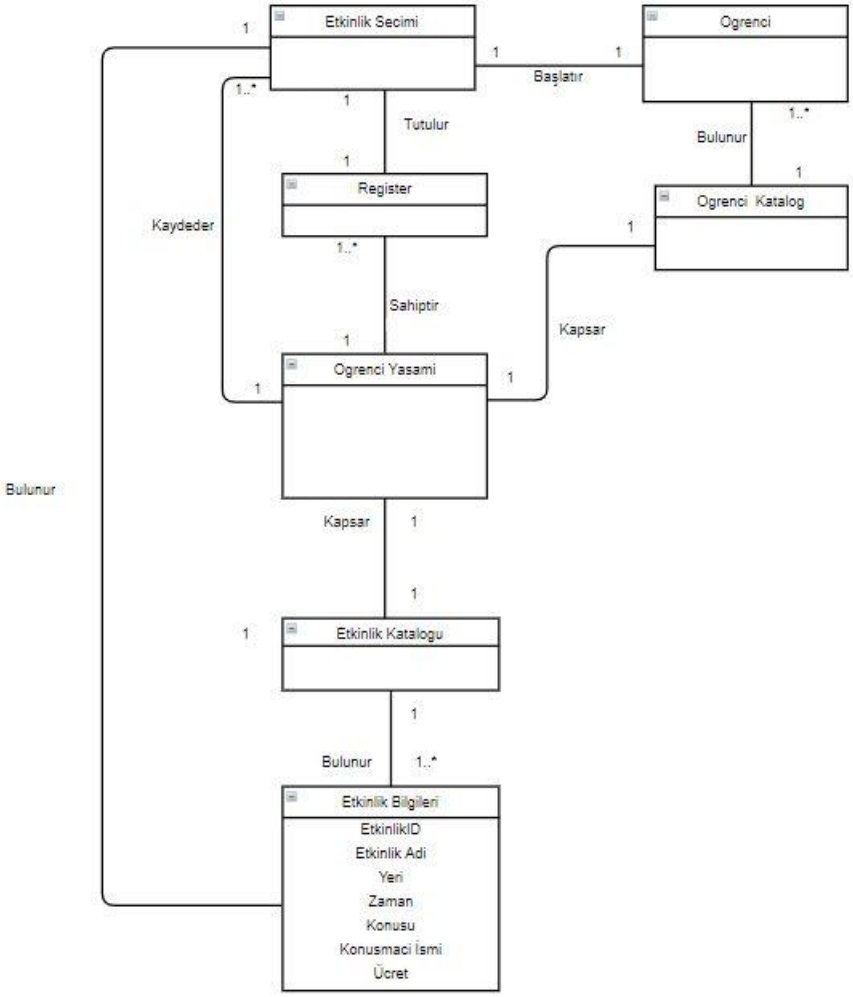
7.1. Etkinlik Seçimi - Full Dressed Form Use Case

Aktörler ve sistem arasındaki etkileşimleri tanımlayabilmek için sistemin her bir gereksinimine ait Use Caseleri oluşturduk. Bunu yaparken adımları daha net görmek adına Fully Dressed formda ilerledik.

Scope:	Çanakkale'de Öğrenci Yaşamı
Name:	Etkinlik Seçimi
Level:	User Goal
Primary Actor:	Öğrenci
Stakeholders and Interests:	Öğrenci : Kendisi için en uygun etkinliği seçmek ister.
Pre-Conditions:	Öğrenci sisteme üye olup giriş yapmalı.
Post Conditions:	Öğrencinin listelenen etkinliklerden birini seçmesi.
Main Success Scenario:	1. Öğrenci etkinlik seçimi işlemini başlatır. 2. Oluşturulmuş etkinlikler opsiyonel olarak filtrelenerek sistem tarafından listelenir. 3. Öğrenci listelenen etkinliklerden en uygun olanını seçer. 4. Öğrenci, etkinliğe katılma isteğini bildirir. 5. Öğrencinin yaptığı seçim başarıyla kaydedilir.
Extensions:	*a. Sistem herhangi bir anda çökerse; 1.Sistem yeniden başlatılır.

7.2. Etkinlik Seçimi - Domain Model

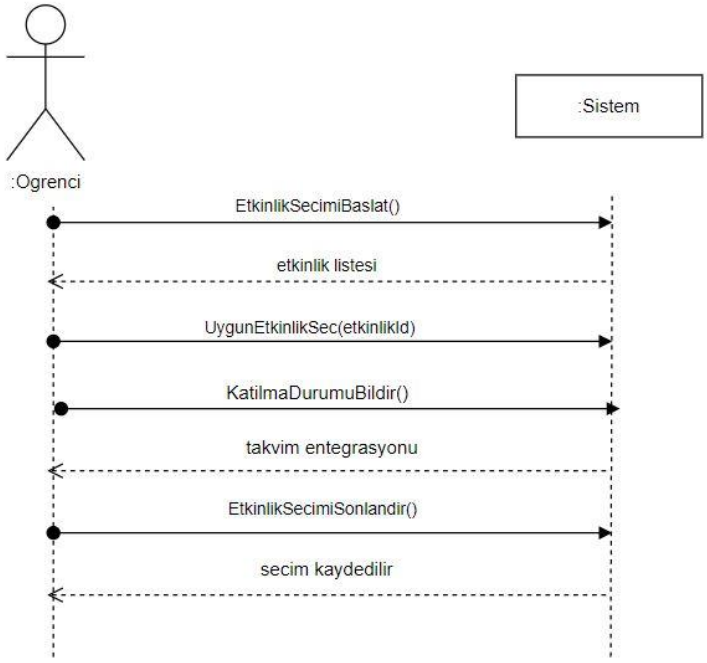
Her bir gereksinimin nasıl işlediğini kavrayabilmek için senaryolarımızı gerçek dünya nesneleriyle ifade edecek Domain Modeller oluşturduk.



Şekil 4 Etkinlik Seçimi - Domain Model

7.3. Etkinlik Seçimi - SSD

Use Caselerden yola çıkarak primary actor ve sistem arasındaki etkileşimi adım adım eventler ile ifade edebileceğimiz SSD diyagramları hazırladık.



Şekil 5 Etkinlik Seçimi - SSD

Sistem gereksinimlerini anlayabilmek ve eventler karşısında sistemin içereceği operasyonları belirleyebilmek için Operasyonel Taahhütnameler oluşturduk.

7.4. Etkinlik Seçimi - Operasyonel Taahhütnameler

Operation Contract - 1

OPERATION	EtkinlikSecimiBaslat()
CROSS REFERENCES	UseCases: Uygun Etkinliğin Seçimi
PRECONDITIONS	Yok
POST CONDITIONS	• Etkinlik Seçimi nesnesi es adıyla oluşturulmuş olmalı. • es nesnesi Register nesnesi ile ilişkilendirilmiş olmalı

Operation Contract - 2

OPERATION	UygunEtkinlikSec(etkinlikId)
CROSS REFERENCES	UseCases: Uygun Etkinliğin Seçimi
PRECONDITIONS	Devam eden etkinlik seçme talebi olmalı
POST CONDITIONS	• Etkinlik Secimi nesnesi ile parametre olarak gelen etkinlikId'nin belirlirttiği etkinlik ilişkilendirilmiş olmalı

Operation Contract - 3

OPERATION	KatilmaDurumuBildir()
CROSS REFERENCES	UseCases: Uygun Etkinliğin Seçimi
PRECONDITIONS	Devam eden etkinlik seçme talebi olmalı

POST CONDITIONS	• Etkinlik Secimi nesnesinin katil özelliği true olarak değiştirilmiş olmalı • Daha önceden oluşturulmuş olan Etkinlik Bilgileri nesnesinin counter özelliği bir arttırılmalı

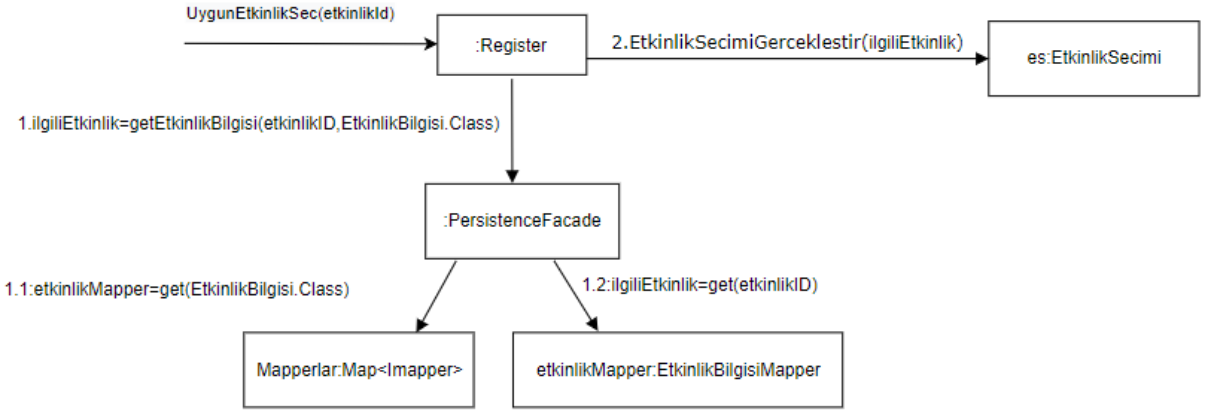
Operation Contract - 4

OPERATION	EtkinlikSecimiSonlandir()
CROSS REFERENCES	UseCases: Uygun Etkinliğin Seçimi
PRECONDITIONS	Devam eden etkinlik seçme talebi olmalı
POST CONDITIONS	• Etkinlik Secimi nesnesinin ebitti özelliği true olarak değiştirilmiş olmalı • Etkinlik seçimi nesnesi ile Öğrenci Yaşami ilişkilendirilmiş olmalı

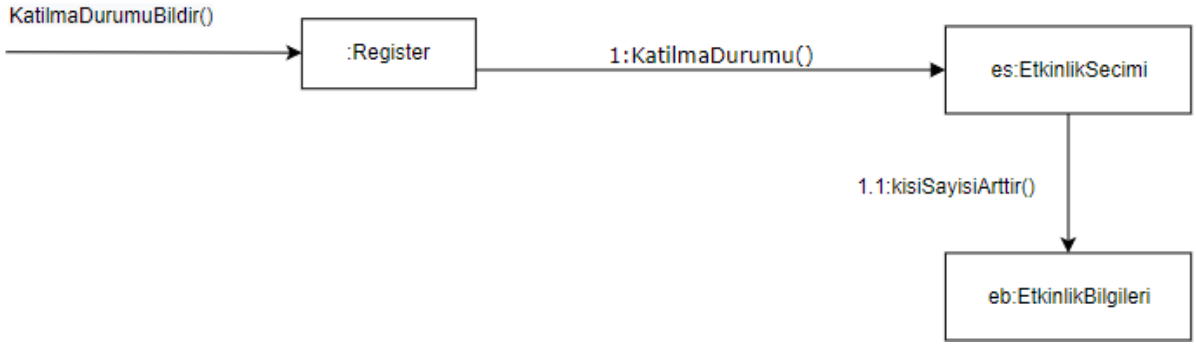
7.5. Etkinlik Seçimi – Interaction Diagram

Sistem gereksinimlerini belirledikten sonra her operasyon için nesnelere sorumluluk dağıtımını gerçekleştirdiğimiz Interaction Diagramları oluşturduk.

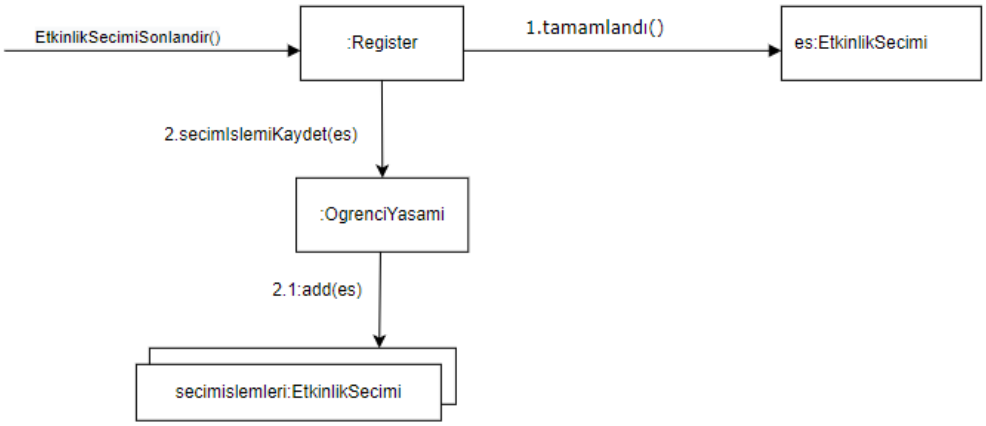
Şekil 6 Operation Contract – 1



Şekil 7 Operation Contract – 2



Şekil 8 Operation Contract - 3



Şekil 9 Operation Contract – 4

8. KULLANILAN TEKNOLOJİLER

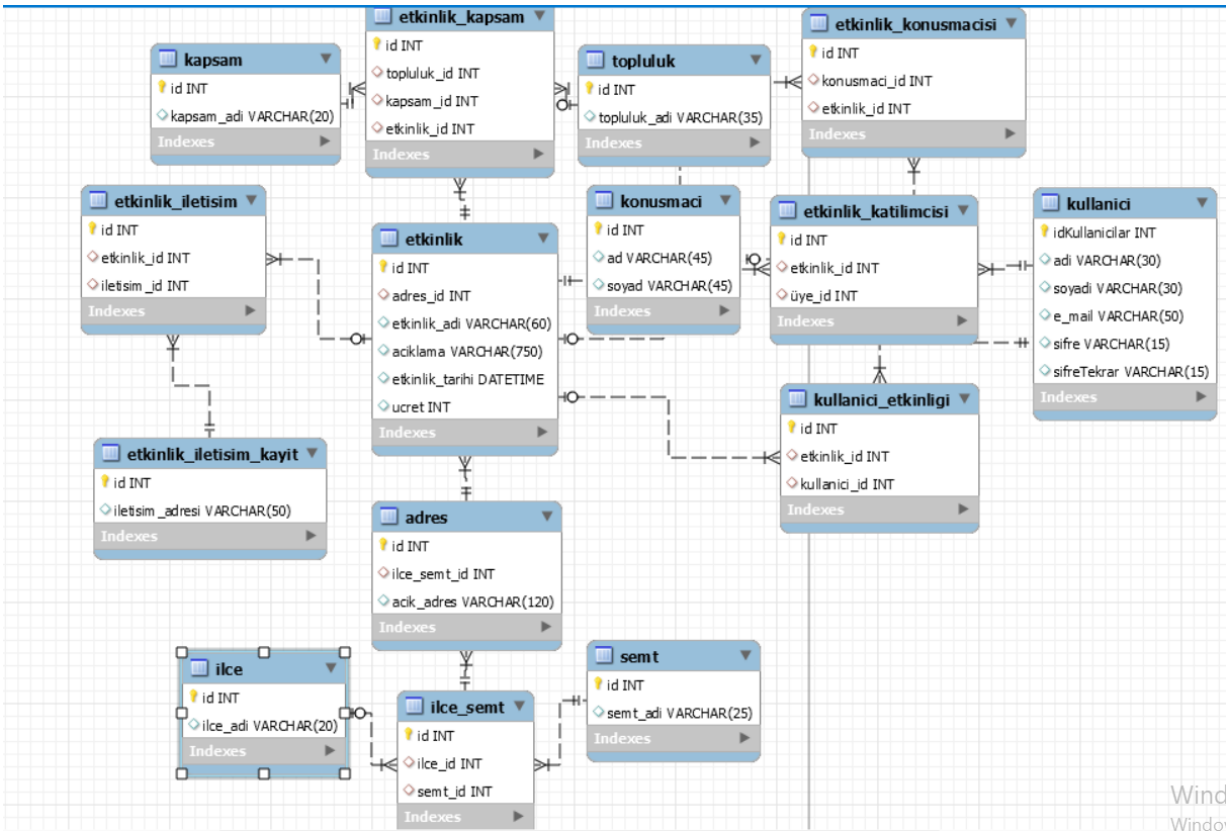
8.1 Mobil Uygulama Platformu

Hedef kullanıcı kitlesini ve uygulama kapsamını göz önünde bulundurduğumuzda mobil cihazlar üzerinden daha kolay ve verimli şekilde gereksinimlerimizin karşılanabileceğini düşündük. Bu yüzden uygulama ortamının mobil tarafta olmasının daha kullanışlı olacağına karar verdik. Geliştirme için Android Studio ortamını belirledik. Daha az kod satırıyla daha güvenli kod yazmaya olanak sağladığı için ve diğer avantajları da dikkate alındığında Kotlin programlama diliyle de uygulamamızı kodlamaya karar verdik.

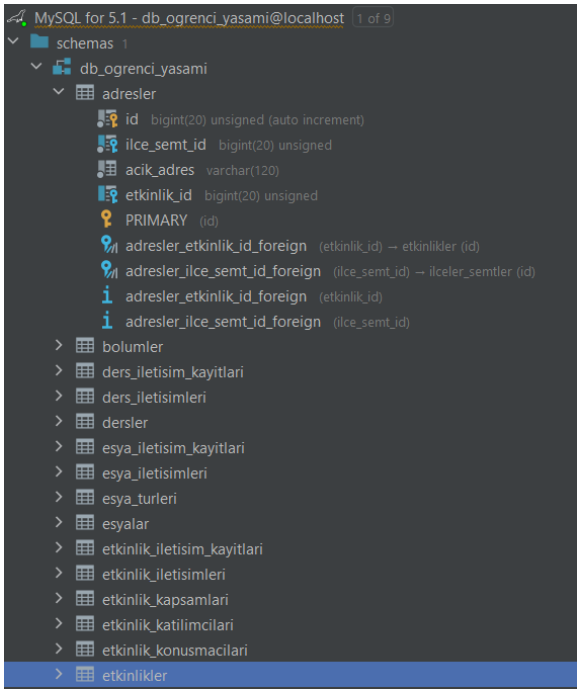
8.2 Veritabanı Tasarımı

Uygulama gereksinimlerini incelediğimizde veritabanında oluşturacağımız birçok tablonun birbiriyle ilişkili olması gerektiğini fark ettik. Bundan dolayı SQLite, MySQL, Microsoft SQL Server, Postgre SQL gibi ilişkisel veritabanları üzerine yoğunlaştık. Bu araştırmalar sonucunda MySQL veritabanına karar verdik.

Veritabanı tasarımımızı öncelikle MySQL Workbench üzerinde yaparak tablolar arasındaki ilişkileri daha net kavradık. Tüm modüllerin ER şemalarını hazırladıktan sonra PhpStorm üzerinden migration işlemleriyle veritabanımızı geliştirme ortamımıza aktarmış olduk.



Şekil 10 MySQL Veritabanı Diyagramı



Şekil 11 PHP Storm DB Tabloları

8.3 Rest API

Uygulamamızda verilerin uzak sunucuda tutulması ve internet üzerinden ulaşılması gereksinimi vardı. Ayrıca veritabanı işlemlerinin tek merkezde yapılıp platform ve dil bağımsızlığının sağlanması gerekliydi. Bu doğrultuda bir API hazırlamaya karar verdik.

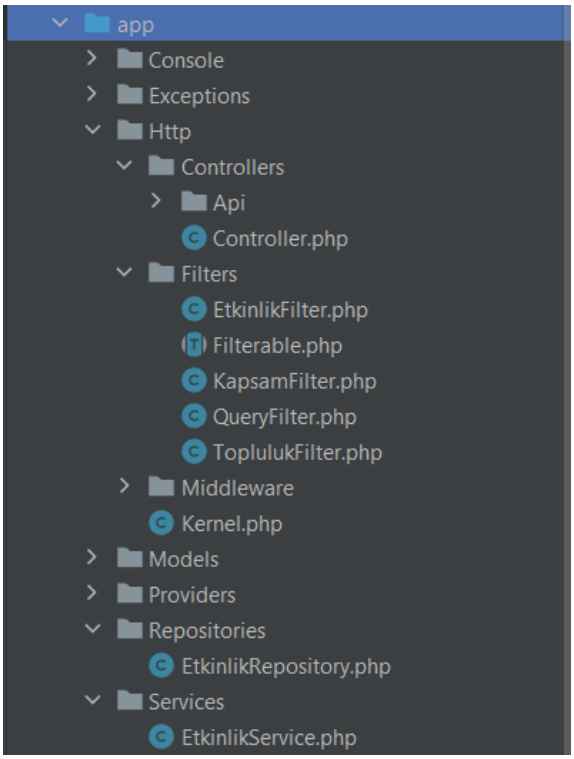
Restful standartı, daha eski olan Soap servislerine bir çözüm olarak geliştirildiğinden bizim de tercihimiz bu yönde oldu. Rest Api ve web servisi gibi konularda yaptığımız araştırmalar sonucunda kullanacağımız frameworkler ve yazılımları belirledik.

Hem hızlı uygulama geliştirmeyi mümkün kılan bir kod altyapısını kullanıcılarına sunarak, daha sade bir ortamda kullanıcılara kolaylık sağladığı için hem de MVC yapısında API uygulama geliştirmeye imkan sunduğu için Laravel framework'üne karar verdik. IDE olarak PHP Storm kullanmayı tercih ettik.

Veri tabanıyla etkili şekilde çalışmak için laravelin sunduğu elequent orm yapısını kullandık.

Controller ve modellerimizi hazırlarken kodlama aşamasında bazı gereksinimlerin eksik olduğunu fark ettik. Bu eksikleri gidererek API'nin bir demosunu tamamladık.

Postman ile isteklerimizi test ettik. Kodlarımızı daha temiz ve değişime açık şekilde yazmak için projemizi bir mimari üzerine inşa ettik. Ayrıca çeşitli tasarım desenlerinden yararlandık, Repository Pattern bunlardan biriydi.



Şekil 12 API Proje Yapısı

```php
public function store(Request $request)
{

    $data = $request->all();
   try {
        $result= [
            'status' => 200,
            'etkinlik' => $this->etkinlikService->storeEtkinlik($data)
        ];
    } catch (Exception $e) {
        $result=[
            'status'=>500,
            'error'=> $e->getMessage()
        ];
    }
    return $result;

}
```

```php
    public function store($data)
    {
        $etkinlik = $this->etkinlik->create($data);
        return $etkinlik->id;
    }
```

```php
public function storeEtkinlik($data)
{

   $validator=Validator::make ($data,[
        'etkinlik_adi' => 'required',
        'aciklama'=>'required',
        'etkinlik_tarihi'=>'required'
    ]);

    if($validator->fails())
    {
        throw new InvalidArgumentException($validator->errors()->first());
    }

    $result=$this->etkinlikRepository->store($data);
    return $result;
}
```

Şekil 13 Etkinlik Ekleme (Controller-Servis-Repository)

```php
class Etkinlik extends Model
{
    use Filterable;
    protected $table='etkinlikler';
    protected $guarded=[];

    public function kullanicilari(){...}
    public function topluluklari(){...}
    public function kapsamlari(){...}

    public function adresi(){...}
    public function konusmacilari(){...}

    public static function etkinlikListe(){...}
    public static function iletisimAdresi($id){...}
    public static function konusmaciAdi($id){...}
    public static function konusmaciSoyadi($id){...}
```

Şekil 14 Etkinlik Model

```php
class QueryFilter
{
    protected $request;
    protected $builder;
    public function __construct(Request $request)
    {
        $this->request = $request;
    }

    public function apply(Builder $builder)
    {
        $this->builder = $builder;

        foreach ($this->filters() as $name => $value) {
            if ( ! method_exists($this, $name)) {...}

            if(is_array($value)){...}
            elseif(is_string($value))
                $this->$name($value);
            else
                $this->$name();
        }
        return $this->builder;
    }

    public function filters()
    {
        return $this->request->all();
    }
}
```

```php
trait Filterable
{
    public function scopeFilter($query, QueryFilter $filters)
    {
        return $filters->apply($query);
    }

}
```

Şekil 15 Filtreleme Classları (QueryFilter - Filterable)

```php
class EtkinlikFilter extends QueryFilter
{
    protected $request;
    public function __construct(Request $request)
    {
        $this->request = $request;
        parent::__construct($request);
    }
    public function etkinlikTarihi($type) {...}
    public function ucret($type) {
        return $this->builder->join('adresler', 'adresler.etkinlik_id', '=', 'etkinlikler.id')
            ->join('ilceler_semtler','ilceler_semtler.id','=','adresler.ilce_semt_id')
            ->join('ilceler','ilceler.id','=','ilceler_semtler.ilce_id')
            ->join('semtler','semtler.id','=','ilceler_semtler.semt_id')->orderBy('ucret', ($type == 'asc') ? 'asc' : 'desc');
    }

}
```

Şekil 16 Etkinlik Filtreleme

8.4 Mobil Uygulama Geliştirme

8.4.1 Retrofit

İsteklerimizi işlemek için network kütüphanelerinden AsyncTask, Volley ve Retrofit üzerine yoğunlaştık. Araştırmalarımız sonucunda AsyncTask yapısının isteklerde yavaş çalışma probleminin olduğunu öğrendik. Volley kütüphanesinin ise Json/Xml parsing işleminde özel istekler oluşturulmasına ihtiyaç duyduğunu bundan dolayı ekstra kodlar yazılması gerektiğini gördük.

Veri çekme ve network işlemleri için API'lara kolaylıkla erişebilme, test edilebilir ve kolay kullanım özelliklerinden dolayı açık kaynak kodlu REST istemcisi olan retrofite karar verdik.

```kotlin
//etkinlik görüntüleme

@GET( value: "etkinlikler/detay")
fun showEtkinlik(@Query( value: "id") id:Int): Single<EtkinlikGoruntulemeResponse>

//etkinlik sil
@DELETE( value: "etkinlikler/{id}")
fun deleteEtkinlik(@Path( value: "id") id:Int ) : Single<Unit>


// etkinlik guncelle
@PATCH( value: "etkinlikler/{id}")
fun updateEtkinlik(@Path( value: "id") id:Int, @Body etkinlik: EtkinlikOlusturmaRequest): Single<EtkinlikOlusturmaResponse>

//etkinlik güncelle - KAPSAM
@PATCH( value: "etkinlikKapsamlari/{id}")
fun updateEtkinlikKapsami(@Path( value: "id") etkinlikId:Int, @Body etkinlik: EtkinlikGuncellemeRequest): Single<Unit>

@GET( value: "kapsamlar/id")
fun getKapsamId(@Query( value: "kapsam_adi") kapsamAdi:String): Single<Int>

@GET( value: "topluluklar/id")
fun getToplulukId(@Query( value: "topluluk_adi[]") toplulukAdi:List<String>): Single<List<Int>>
```

```
//etkinlik listeleme
@GET( value: "etkinlikler/etkinlikler")
fun getEtkinlikler(): Single<List<EtkinlikListelemeResponse>>

//etkinlik oluşturma
@POST( value: "etkinlikler")
fun createEtkinlik(@Body etkinlik: EtkinlikOlusturmaRequest): Single<EtkinlikOlusturmaResponse>

@FormUrlEncoded
@POST( value: "kullaniciEtkinlikleri")
fun createKullaniciEtkinlik(@Field( value: "etkinlik_id") etkinlikId:Int?,
                            @Field( value: "kullanici_id") kullaniciId:Int): Single<Unit>

@FormUrlEncoded
@POST( value: "etkinlikKapsamlari")
fun createEtkinlikKapsamlari(@Field( value: "etkinlik_id") etkinlikId:Int?,
                             @Field( value: "topluluk_id[]") toplulukId:List<Int>,
                             @Field( value: "kapsam_id") kapsamId:Int): Single<Unit>

@FormUrlEncoded
@POST( value: "adresler")
fun createEtkinlikAdresi(@Field( value: "etkinlik_id") etkinlikId:Int?,
                @Field( value: "ilce_semt_id") ilceSemtId:Int,
                @Field( value: "acik_adres") acikAdres:String): Single<Unit>
```

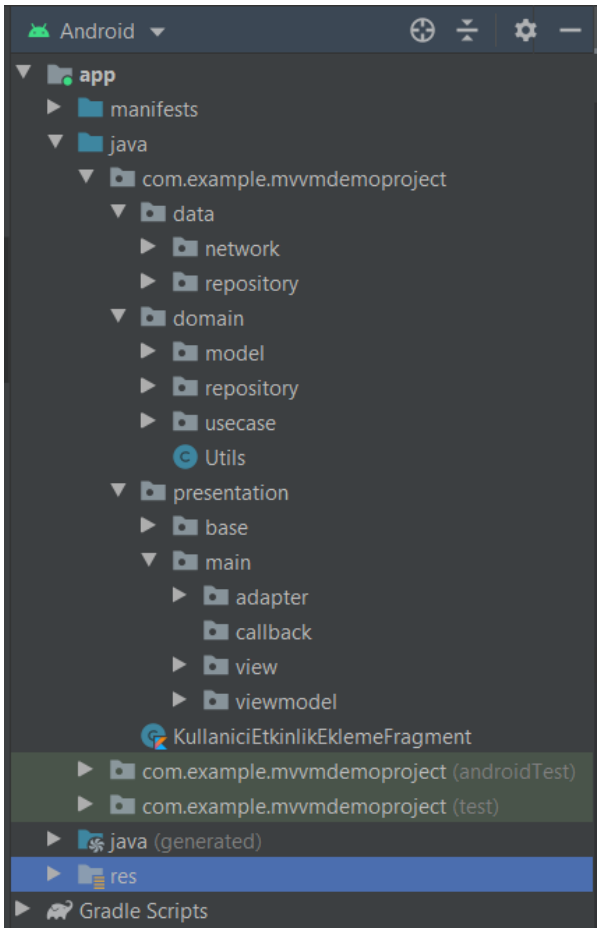
Şekil 17 Retrofit İstekler

8.4.2 MVVM

Web serviste hazırladığımız gibi projemizi mimari üzerine inşa etmek istedik. Uygulamamız hangi mimarinin daha uygun olduğuna karar vermek için MVC,MVP ve MVVM mimarileri üzerine araştırma yaptık.

Kullanıcı ara birimiyle ilişkili verileri yaşam döngüsü bilinçli şekilde tutmayı ve yönetmeyi sağladığı bu sayede de yapılandırma değişikliklerini koruduğu için MVVM üzerine karar verdik.

Kullanım durumları ile veri akışı mantığını uygulayarak bu akışın view modellerde yeniden kullanılabilir olmasını sağladık.



Şekil 18 Android Studio Proje Yapısı

```
class ApiClient {
    companion object {

        var retrofit: Retrofit? = null
        var BaseURL:String="http://192.168.1.104/api/"

        fun getClient(): Retrofit {
            if (retrofit == null) {
                retrofit = Retrofit.Builder()
                    .baseUrl(BaseURL)
                    .addConverterFactory(GsonConverterFactory.create())
                    .addCallAdapterFactory(RxJava3CallAdapterFactory.create())
                    .client(OkHttpClient())
                    .build()
                return retrofit as Retrofit
            }
            return retrofit as Retrofit
        }
    }
}
```

Şekil 19 Retrofit İstemci

```kotlin
data class EtkinlikListelemeResponse(
    @SerializedName( value: "etkinlik_adi")
    val etkinlikAdi: String,
    @SerializedName( value: "aciklama")
    val aciklama: String,
    @SerializedName( value: "etkinlik_tarihi")
    val etkinlikTarihi: String,
    @SerializedName( value: "ucret")
    val ucret: Number,
    @SerializedName( value: "acik_adres")
    val acikAdres:String,
    @SerializedName( value: "ilce_adi")
    val ilceAdi:String,
    @SerializedName( value: "semt_adi")
    val semtAdi:String
)
```

```kotlin
interface EtkinlikListelemeRepository {
    fun getEtkinlikler():Single<List<EtkinlikListelemeResponse>>
    fun getKullaniciEtkinlikleri(kullaniciId:Int):Single<List<EtkinlikListelemeResponse>>
    fun getId(ad:String,ucret:Number,tarih:String,aciklama:String): Single<Int>
    fun getTopluluklar(): Single<List<String>>
    fun getKapsamlar():Single<List<String>>
    fun filterKapsam(kapsam: String): Single<List<EtkinlikListelemeResponse>>
    fun filterTopluluk(topluluk: String): Single<List<EtkinlikListelemeResponse>>
    fun getEtkinlikSiralama(ucret:String?,tarih:String?): Single<List<EtkinlikListelemeResponse>>
}
```

```kotlin
class EtkinlikListelemeRepositoryImpl : EtkinlikListelemeRepository{

    var etkinlikAPI = ApiClient.getClient().create(EtkinlikService::class.java)

    override fun getEtkinlikler(): Single<List<EtkinlikListelemeResponse>> = etkinlikAPI.getEtkinlikler()
    override fun getKullaniciEtkinlikleri(kullaniciId: Int): Single<List<EtkinlikListelemeResponse>> = etkinlikAPI.getKullaniciEtkinlikleri(kullaniciId)

    override fun getId(ad:String,ucret:Number,tarih:String,aciklama:String): Single<Int> = etkinlikAPI.getId(ad,ucret,tarih,aciklama)
    override fun getTopluluklar(): Single<List<String>> = etkinlikAPI.getTopluluklar()
    override fun getKapsamlar(): Single<List<String>> = etkinlikAPI.getKapsamlar()
    override fun filterKapsam(kapsam: String): Single<List<EtkinlikListelemeResponse>> = etkinlikAPI.getKapsamaGore(kapsam)
    override fun filterTopluluk(topluluk: String): Single<List<EtkinlikListelemeResponse>> = etkinlikAPI.getToplulugaGore(topluluk)
    override fun getEtkinlikSiralama(ucret: String?, tarih: String?): Single<List<EtkinlikListelemeResponse>> = etkinlikAPI.getEtkinlikSiralama(ucret, tarih)
}
```

```kotlin
class EtkinlikListelemeUseCase(private val etkinlikListelemeRepository: EtkinlikListelemeRepository ) {
    operator fun invoke():Single<List<EtkinlikListelemeResponse>> = etkinlikListelemeRepository.getEtkinlikler()
    fun getKullaniciEtkinlikleri(kullaniciId: Int): Single<List<EtkinlikListelemeResponse>> = etkinlikListelemeRepository.getKullaniciEtkinlikleri(kullaniciId)

    fun getId(ad:String,ucret:Number,tarih:String,aciklama:String): Single<Int> = etkinlikListelemeRepository.getId(ad, ucret, tarih, aciklama)

    fun getTopluluklar(): Single<List<String>> = etkinlikListelemeRepository.getTopluluklar()
    fun getKapsamlar(): Single<List<String>> = etkinlikListelemeRepository.getKapsamlar()
    fun filterKapsam(kapsam: String): Single<List<EtkinlikListelemeResponse>> = etkinlikListelemeRepository.filterKapsam(kapsam)
    fun filterTopluluk(topluluk: String): Single<List<EtkinlikListelemeResponse>> = etkinlikListelemeRepository.filterTopluluk(topluluk)
    fun getEtkinlikSiralama(ucret:String?,tarih:String?): Single<List<EtkinlikListelemeResponse>> = etkinlikListelemeRepository.getEtkinlikSiralama(ucret, tarih)

}
```

Şekil 10 Etkinlik Listeleme (Model- Repository Interface -Reporsitory -UseCase)

```
class EtkinlikListelemeFragment : Fragment() {
    private lateinit var etkinlikListelemeViewModel: EtkinlikListelemeViewModel
    private lateinit var adapter: EtkinlikAdapter
    private lateinit var rcListener:EtkinlikAdapter.RecyclerviewClickListener
    lateinit var recyclerView: RecyclerView
    lateinit var rootView: View
    var siraliEtkinlik:List<EtkinlikListelemeResponse> = listOf()
    var repository= EtkinlikListelemeRepositoryImpl()
    var etkinlikListelemeUseCase= EtkinlikListelemeUseCase(repository)


    var kullaniciId=0

    val listOfEtkinlikType: Type = object : TypeToken<List<EtkinlikListelemeResponse?>?>() {}.type

    val mutablelistOfEtkinlikType: Type = object : TypeToken<MutableList<List<EtkinlikListelemeResponse?>?>>() {}.type

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        arguments?.let { it: Bundle
            if(it.containsKey(ARG_PARAM3)) {
                val liste3 :String?  = it.getString(ARG_PARAM3)
                siraliEtkinlik =
                    Utils.getGsonParser().fromJson(liste3,listOfEtkinlikType)
            }
            if(it.containsKey(ARG_PARAM4)) {
                val liste4 :String?  = it.getString(ARG_PARAM4)
                filtreliEtkinlik =
                    Utils.getGsonParser().fromJson(liste4,mutablelistOfEtkinlikType)
```

Şekil 11 Etkinlik Listeleme Fragment

```
class EtkinlikListelemeViewModel(private val etkinlikListelemeUseCase: EtkinlikListelemeUseCase) : ViewModel() {

    private val compositeDisposable = CompositeDisposable()
    private val etkinlikler=MutableLiveData<List<EtkinlikListelemeResponse>>()
    private val kullaniciEtkinlikleri=MutableLiveData<List<EtkinlikListelemeResponse>>()
    private val repoLoadError = MutableLiveData<Boolean>()
    private val loading = MutableLiveData<Boolean>()
    private val id = MutableLiveData<Int>()
    private val kapsamlar = MutableLiveData<List<String>>()
    private val topluluklar = MutableLiveData<List<String>>()
    private val siraliEtkinlikler = MutableLiveData<List<EtkinlikListelemeResponse>>()
    val mapped : MutableList<MutableLiveData<List<EtkinlikListelemeResponse>>> = mutableListOf()
    private var kapsamaGoreResponse =MutableLiveData<List<EtkinlikListelemeResponse>>()
    private var toplulugaGoreResponse = MutableLiveData<List<EtkinlikListelemeResponse>>()

    var kullaniciId = 0

    init {
        fetchEtkinlikler()
    }


    private fun fetchEtkinlikler() {
        compositeDisposable.add(
            etkinlikListelemeUseCase.invoke()
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe({etkinlikList->etkinlikler.postValue(etkinlikList)},{throwable ->etkinlikler.postValue(arrayListOf())})
```

```
class EtkinlikListelemeViewModelFactory(private val etkinlikListelemeUseCase: EtkinlikListelemeUseCase) : ViewModelProvider.Factory {

    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(EtkinlikListelemeViewModel::class.java)) {
            return EtkinlikListelemeViewModel((etkinlikListelemeUseCase)) as T
        }
        throw IllegalArgumentException("Unknown class name")
    }

}
```

Şekil 22 Etkinlik Listeleme View Model

8.4.3 RxJava

Veritabanındaki verilerin gecikmeli gelebileceğini göz önünde tutarak arayüz işlemleri ile istek sonuçlarının birbirini bloklamaması ,ana ve arka plan iş parçacığını birbirinden ayrılabilmesi için eş zamansız çağrılar yapmamız gerekliydi. Bundan dolayı isteklerimizi işleyebilmemizi sağlayacak çeşitli kütüphaneler araştırdık

Bu kütüphaneler arasından

- Operatörler ve veri akış manipülasyonlarının kolaylıkla yapılabilmesi
- Observable veri tipleri sayesinde istek yapısına uygun veriler işlenebilmesi
- Zincir yapısı sayesinde iç içe ağ aramasından kurtarması
- Kodun çalıştırılacağı zamanlayıcıyı değiştirme olanağı sağlaması
- Hata işlemeyi kolaylaştırması

gibi avantajlarının bulunmasından dolayı RxJava'ya karar verdik.

```
private fun  create(id:Int?) : Single<Boolean> {
    val x : Disposable! = Single.merge(
        etkinlikOlusturmaUseCase.getIlceSemt(etkinlikFormModel.ilceId,etkinlikFormModel.semtId).flatMap { it: Int!
            etkinlikOlusturmaUseCase.postEtkinlikAdresi(id,it,etkinlikFormModel.adres) }.subscribeOn(Schedulers.io()),
        etkinlikOlusturmaUseCase.postEtkinlikKapsamlari(id, etkinlikFormModel.toplulukList,etkinlikFormModel.kapsamList.get(0)).subscribeOn(Schedulers.io()),
        etkinlikOlusturmaUseCase.postKonusmaci(etkinlikFormModel.konusmaciAd, etkinlikFormModel.konusmaciSoyad).flatMap { it: Array<Int>!
            etkinlikOlusturmaUseCase.postEtkinlikKonusmacisi(id,it)}.subscribeOn(Schedulers.io()),
        etkinlikOlusturmaUseCase.postIletisimAdresleri(etkinlikFormModel.iletisimAdresi.toList()).flatMap { it: Array<Int>!
            etkinlikOlusturmaUseCase.postEtkinlikIletisimAdresleri(id,it)}.subscribeOn(Schedulers.io())
    )

        .observeOn(AndroidSchedulers.mainThread())
        .subscribe()

    val y : Disposable! = etkinlikOlusturmaUseCase.postKullaniciEtkinlik(id,kulaniciID.toInt())
                        .subscribeOn(Schedulers.io())
                        .observeOn(AndroidSchedulers.mainThread())
                        .subscribe()
    return Single.just( item: true)

}
```

```
fun mainIletisimAdresiUpdate(iletisimListesi:List<String>) : Single<Boolean> {
    compositeDisposable.add(
        etkinlikGuncellemeUseCase.getIletisimId(id)
            .flatMap { etkinlikGuncellemeUseCase.modifyIletisimAdresi(iletisimListesi,it) }
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())

            .subscribe({ Log.e( tag: "UPDATE", msg: "i")},{throwable -> })


    )
    return Single.just( item: true)

}
```

Şekil 23 RxJava Örnekleri

8.4.4 Data Binding

Arayüzümüz kullanıcı odaklı olduğu için daha hızlı ve düzenli kod yazmamızı sağlayacak veri bağlama kütüphanesine ihtiyacımız vardı.

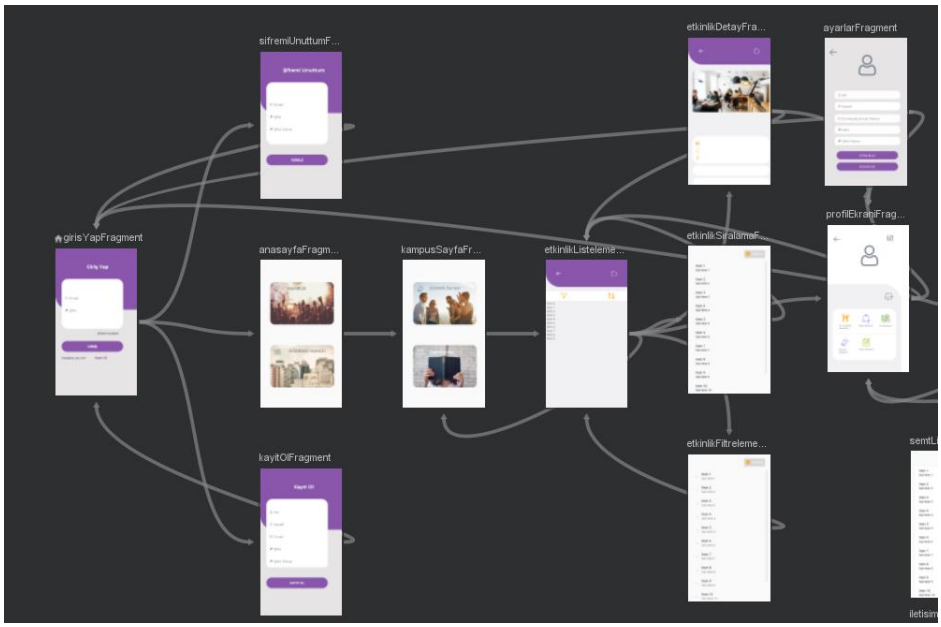
Jetpack mimari bileşenlerde sunulan data binding framework'ünü kullanmaya karar verdik. Böylece veri modelini XML dosyasına bağladık.

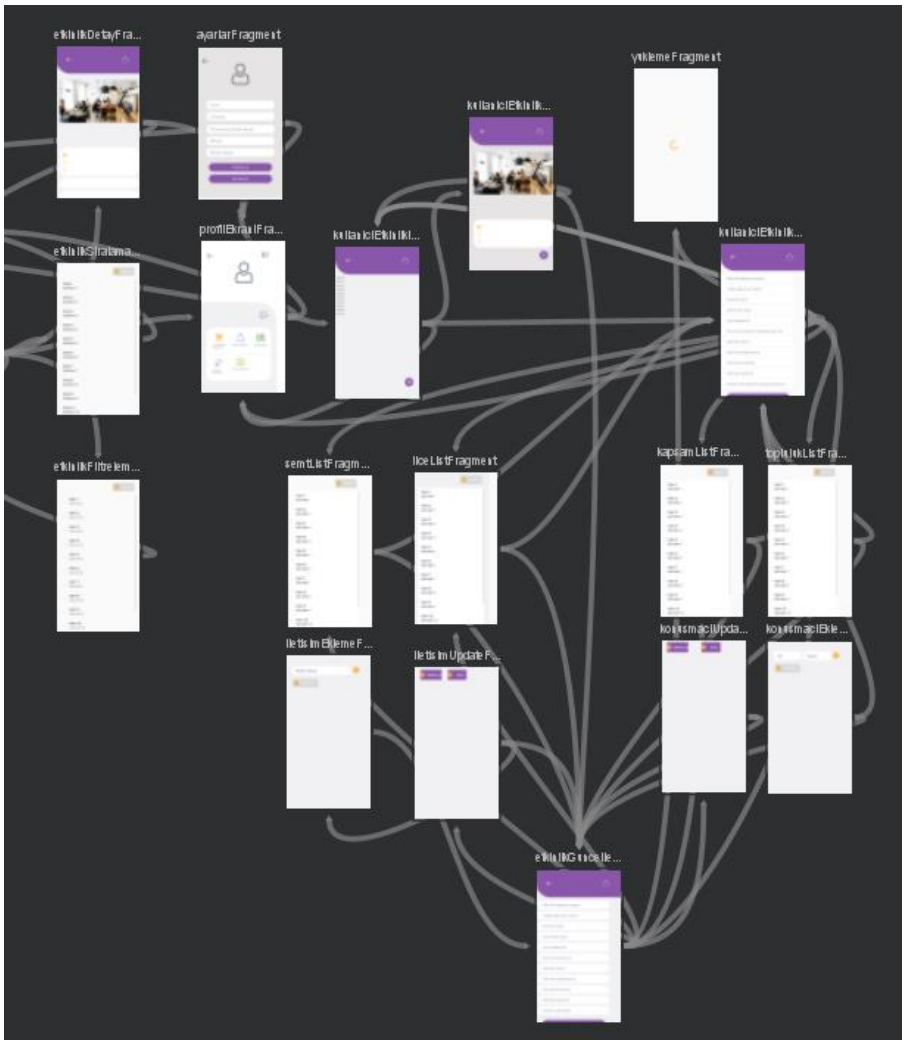
```
<?xml version="1.0" encoding="utf-8"?>
<layout>
    <data>
        <variable
            name="etkinlik"
            type="com.example.mvvmdemoproject.domain.model.EtkinlikListelemeResponse"/>
    </data>

    <androidx.cardview.widget.CardView
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:id="@+id/root"
        android:layout_width="match_parent"
        android:layout_height="150dp"
        app:cardCornerRadius="40dp"
        android:background="@drawable/form_item"
        android:layout_margin="15dp">

        <!--Etkinlik Raw Item-->
        <androidx.constraintlayout.widget.ConstraintLayout...>


    </androidx.cardview.widget.CardView>
</layout>
```

Şekil 24 Data binding XML Uygulaması

8.4.5 Navigation

Uygulamamızın baştan sona bir bütün halinde gezinme akışını sağlamak için Jetpack mimari bileşenlerde bulunan Navigation componenti kullanmaya karar verdik. Bu sayede fragmentler arası geçişi kolaylıkla ele alıp geri akışları etkili şekilde tamamladık.





Şekil 25 Tüm Uygulama Navigation

KAYNAKÇA

1. https://www.udemy.com/course/sifirdan-ileri-seviyeye-kotlin-ve-android-kursu/
2. https://www.udemy.com/course/android-mobil-uygulama-kursu-seviye-2/
3. https://www.udemy.com/course/laravel-ile-sifirdan-restful-api-uygulamalari-gelistirme/
4. https://dev.to/safventure/implement-crud-with-laravel-service-repository-pattern-1dkl
5. StackOverFlow
6. https://medium.com/@marslan.ali/laravel-best-practices-every-developer-should-know-and-follow-it-cebccfb1cc3e
7. https://dev.to/stefant123/create-filters-in-laravel-with-oop-best-practices-pm9
8. https://laravel.gen.tr/d/3886-checkbox-ile-filtreleme-yapimi
9. https://medium.com/@marslan.ali/laravel-best-practices-every-developer-should-know-and-follow-it-cebccfb1cc3e
10. https://www.mobilhanem.com/android-retrofit-kullanimi-post-islemi/
11. https://medium.com/@linkusman21/achieving-clean-architecture-with-mvvm-part-i-2b33a023422e
12. https://www.raywenderlich.com/7026-getting-started-with-mvp-model-view-presenter-on-android#toc-anchor-008
13. https://www.journaldev.com/14886/android-mvp
14. https://medium.com/@hasann.kucuk/dagger-2-nedir-f7be8c293a85
15. https://www.gencayyildiz.com/blog/dependency-injection-nedir-nasil-uygulanir/
16. https://tugrulbayrak.medium.com/dependency-injection-kavrami-nedir-ne-zaman-kullanilir-efb21db9db32
17. https://blog.mindorks.com/essential-guide-for-designing-your-android-app-architecture-mvp-part-1-74efaf1cda40
18. https://hsmnzaydn.medium.com/mvp-retrofit-dagger-2-ve-rxjava-kullanarak-android-uygulama-geli%C5%9Ftirme-1-2cbb7ae8e1f7
19. https://medium.com/@ismail.gungor.92/nedir-bu-mvp-model-view-presenter-tasar%C4%B1m-deseni-439be068ba6c
20. https://medium.com/@ismail.gungor.92/nedir-bu-mvp-model-view-presenter-tasar%C4%B1m-deseni-439be068ba6c
21. https://www.simform.com/mvc-mvp-mvvm-android-app-development/
22. http://www.digigene.com/category/android/android-architecture/
23. https://www.mobilhanem.com/mvp-retrofit-rxjava-recyclerview-kullanimi/
24. https://www.geeksforgeeks.org/difference-between-architectural-style-architectural-patterns-and-design-patterns/
25. https://wmaraci.com/forum/mobil-uygulama-gelistirme/android-uygulama-icin-veritabani-secimi-539583.html
26. https://medium.com/@betul.sandikci/android-aktivite-ya%C5%9Fam-d%C3%B6ng%C3%BCs%C3%BC-activity-life-cycle-bir-aktivite-olmak-i-e9b7c88ec67c
27. https://proandroiddev.com/how-rxjava-chain-actually-works-2800692f7e13
28. https://levelup.gitconnected.com/rxjava-connecting-network-calls-made-easy-d7b406c9bc66
29. https://medium.com/@DoorDash/synchronizing-network-calls-with-rxjava-d30f7db66fb9
30. https://mertkesgin.com/retrofit-kullanimikotlin/

31. https://mertkesgin.com/retrofit-kullanimi-3/
32. https://blog.jgrossi.com/2018/queryfilter-a-model-filtering-concept/
33. https://github.com/alexeymezenin/laravel-best-practices/blob/master/turkish.md
34. http://tucker-eric.github.io/EloquentFilter/
35. https://mykeels.medium.com/writing-clean-composable-eloquent-filters-edd242c82cc8
36. https://www.technopat.net/sosyal/konu/rest-api-ile-mobil-uygulama-yapma.678146/
37. https://www.dahiweb.com/laravel-tablolar-arasi-iliskiler-relations/
38. https://www.journaldev.com/9708/android-asynctask-example-tutorial
39. https://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html
40. https://www.yazilimbilisim.net/android/android-arka-plan-isleme-zaman-uyumsuz-asenkron-programlama-asynctask-kullanimi/
41. http://emre-kose.net/retrofit-nedir-nasil-kullanilir/
42. http://emre-kose.net/androidde-sik-kullanilan-8-kutuphane/
43. https://medium.com/@manuelvicnt/rxjava-android-mvvm-app-structure-with-retrofit-a5605fa32c00
44. https://medium.com/@ruwan.se.uok/android-apps-using-mvvm-with-clean-architecture-e50216f4a52b
45. https://www.raywenderlich.com/3595916-clean-architecture-tutorial-for-android-getting-started#toc-anchor-014
46. https://hsmnzaydn.medium.com/androidte-clean-architecture-kullan%C4%B1m%C4%B1-ca648d71b017
47. https://proandroiddev.com/mvvm-architecture-viewmodel-and-livedata-part-1-604f50cda1
48. https://blog.mindorks.com/mvvm-architecture-android-tutorial-for-beginners-step-by-step-guide
49. https://www.paulund.co.uk/eloquent-builder-vs-scopes
50. https://medium.com/@mwaysolutions/10-best-practices-for-better-restful-api-33a25a4e92c1