

# **LAPORAN PRAKTIKUM**

## **PEMROGRAMAN BERBASIS MOBILE**

### **PERTEMUAN KE-14**



**Disusun Oleh :**

**NAMA** : Raden Isnawan Argi Aryasatya  
**NIM** : 195410257  
**JURUSAN** : Informatika  
**JENJANG** : S1  
**KELAS** : 5

**Laboratorium Terpadu**  
**Sekolah Tinggi Manajemen Informatika Komputer**  
**AKAKOM**  
**YOGYAKARTA**

**2021**

## **PERTEMUAN KE-14**

### **(FILTERING AND DETAIL VIEWS WITH INTERNET DATA)**

#### **TUJUAN**

Mahasiswa mampu membuat aplikasi yang dapat memfilter dan menampilkan gambar dari internet menggunakan Retrofit, Moshi dan Glide.

#### **DASAR TEORI**

### **1. Selamat datang**

Dalam codelab sebelumnya untuk pelajaran ini, Anda telah mempelajari cara mendapatkan data tentang real estate di Mars dari layanan web, dan cara membuat RecyclerView dengan tata letak kisi untuk memuat dan menampilkan gambar dari data tersebut. Dalam codelab ini, Anda menyelesaikan aplikasi MarsRealEstate dengan menerapkan kemampuan untuk memfilter properti Mars berdasarkan apakah properti tersebut tersedia untuk disewa atau dibeli. Anda juga membuat tampilan detail sehingga jika pengguna mengetuk foto properti di ikhtisar, mereka melihat tampilan detail dengan detail tentang properti itu.

#### **Apa yang akan Anda Pelajari**

- Cara menggunakan ekspresi binding kompleks dalam file layout Anda.
- Cara membuat permintaan Retrofit ke layanan web dengan opsi kueri.

#### **Apa yang akan Anda lakukan**

- Ubah aplikasi MarsRealEstate untuk menandai properti Mars yang akan dijual (versus yang disewakan) dengan ikon tanda dolar.
- Gunakan menu opsi di halaman ringkasan untuk membuat permintaan layanan web yang memfilter properti Mars berdasarkan jenis.
- Buat fragmen detail untuk properti Mars, hubungkan fragmen itu ke kisi ringkasan dengan navigasi, dan teruskan data properti ke dalam fragmen itu.

### **2. Ikhtisar aplikasi**

Dalam codelab ini (dan codelab terkait), Anda bekerja dengan aplikasi bernama MarsRealEstate, yang menampilkan properti untuk dijual di Mars. Aplikasi ini terhubung ke server internet untuk mengambil dan menampilkan data properti, termasuk detail seperti harga dan apakah properti tersedia untuk dijual atau disewakan. Gambar yang mewakili setiap properti adalah foto kehidupan nyata dari Mars yang diambil dari penjelajah Mars NASA.

-----

### **3. Tugas: Menambahkan gambar "for sale" ke overview**

#### **Langkah 1: Perbarui MarsProperty untuk memasukkan jenisnya**

Kelas MarsProperty menentukan struktur data untuk setiap properti yang disediakan oleh layanan web. Di codelab sebelumnya, Anda menggunakan pustaka Moshi untuk mengurai respons JSON mentah dari layanan web Mars menjadi objek data MarsProperty individual. Pada langkah ini, Anda menambahkan beberapa logika ke kelas MarsProperty untuk menunjukkan apakah sebuah properti disewakan atau tidak (yaitu, apakah jenisnya adalah string "rent" atau "buy"). Anda akan menggunakan logika ini di lebih dari satu tempat, jadi lebih baik menyimpannya di sini, di kelas data daripada menirunya.

1. Buka aplikasi MarsRealEstate dari codelab terakhir. (Anda dapat mengunduh MarsRealEstateGrid jika Anda tidak memiliki aplikasinya, juga disediakan di ELA ini)

2. Buka network/MarsProperty.kt. Tambahkan body ke definisi kelas MarsProperty, dan tambahkan getter kustom untuk isRental yang mengembalikan nilai true jika objek berjenis "rent".

```
data class MarsProperty(  
    val id: String,  
    // used to map img_src from the JSON to imgSrcUrl in our class  
    @Json(name = "img_src") val imgSrcUrl: String,  
    val type: String,  
    val price: Double): {  
    val isRental  
    get() = type == "rent"
```

## Langkah 2: Perbarui tata letak item kisi

Sekarang Anda memperbarui tata letak item untuk kisi dari gambar-gambar agar menampilkan drawable bertanda dolar hanya pada gambar properti yang akan dijual. Dengan ekspresi data binding, Anda dapat melakukan pengujian ini sepenuhnya dalam tata letak XML untuk item kisi.

1. Buka res/layout/grid\_view\_item.xml. Ini adalah file tata letak untuk setiap sel dalam tata letak kisi untuk RecyclerView. Saat ini file tersebut hanya berisi elemen <ImageView> untuk gambar properti.
2. Di dalam elemen <data>, tambahkan elemen <import> untuk kelas View. Anda menggunakan impor saat Anda ingin menggunakan komponen kelas di dalam ekspresi data binding dalam file tata letak. Dalam kasus ini, Anda akan menggunakan konstanta View.GONE dan View.VISIBLE, jadi Anda memerlukan akses ke kelas View.

```
<import type="android.view.View"/>
```

3. Kelilingi seluruh tampilan gambar dengan FrameLayout, untuk memungkinkan drawable bertanda dolar ditumpuk di atas gambar properti.

```
<FrameLayout  
    android:layout_width="match_parent"  
    android:layout_height="170dp">  
    <ImageView  
        android:id="@+id/mars_image"
```

4. Untuk ImageView, ubah atribut android:layout\_height menjadi match\_parent, untuk mengisi FrameLayout induk baru.

```
android:layout_height="match_parent"
```

5. Tambahkan elemen <ImageView> kedua tepat di bawah yang pertama, di dalam FrameLayout. Gunakan definisi yang ditunjukkan di bawah ini. Gambar ini muncul di sudut kanan bawah dari item petak, di atas gambar Mars, dan menggunakan drawable yang ditentukan di res/drawable/ic\_for\_sale\_outline.xml untuk ikon tanda dolar.

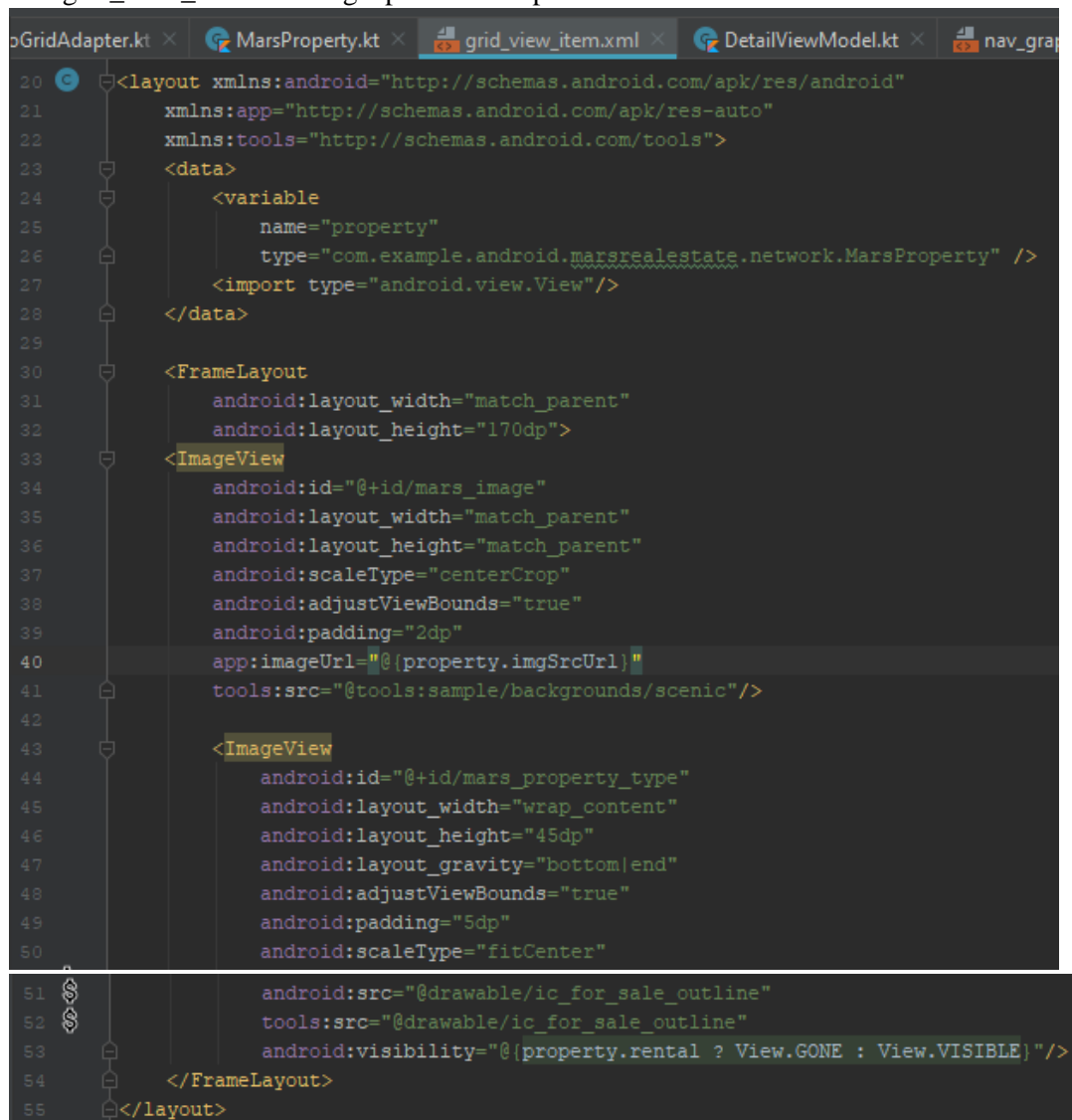
```
<ImageView  
    android:id="@+id/mars_property_type"  
    android:layout_width="wrap_content"  
    android:layout_height="45dp"  
    android:layout_gravity="bottom|end"  
    android:adjustViewBounds="true"  
    android:padding="5dp"  
    android:scaleType="fitCenter"  
    android:src="@drawable/ic_for_sale_outline"  
    tools:src="@drawable/ic_for_sale_outline"
```

6. Tambahkan atribut `android:visibility` ke tampilan gambar `mars_property_type`. Gunakan ekspresi binding untuk menguji jenis properti, dan tetapkan visibilitas ke `View.GONE` (untuk rental) atau `View.VISIBLE` (untuk pembelian).

```
android:visibility="@{property.rental ? View.GONE : View.VISIBLE}"/>
```

Sampai saat ini Anda hanya melihat ekspresi binding di tata letak yang menggunakan variabel individual yang ditentukan dalam elemen `<data>`. Ekspresi binding sangat berguna dan memungkinkan Anda melakukan operasi seperti pengujian dan penghitungan matematika sepenuhnya dalam tata letak XML Anda. Dalam kasus ini, Anda menggunakan operator ternary (`?:`) untuk melakukan pengujian (apakah objek ini rental?). Anda memberikan satu hasil untuk benar (sembunyikan ikon tanda dolar dengan `View.GONE`) dan satu lagi untuk salah (tunjukkan ikon itu dengan `View.VISIBLE`).

File `grid_view_item.xml` lengkap baru ditampilkan di bawah ini:

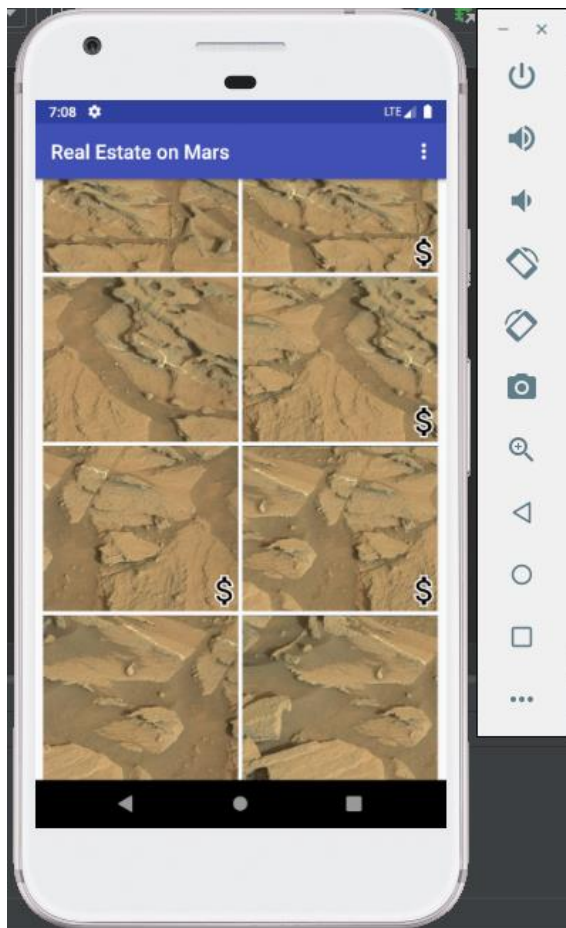


```
20 <layout xmlns:android="http://schemas.android.com/apk/res/android"
21     xmlns:app="http://schemas.android.com/apk/res-auto"
22     xmlns:tools="http://schemas.android.com/tools">
23     <data>
24         <variable
25             name="property"
26             type="com.example.android.marsrealestate.network.MarsProperty" />
27         <import type="android.view.View"/>
28     </data>
29
30     <FrameLayout
31         android:layout_width="match_parent"
32         android:layout_height="170dp">
33         <ImageView
34             android:id="@+id/mars_image"
35             android:layout_width="match_parent"
36             android:layout_height="match_parent"
37             android:scaleType="centerCrop"
38             android:adjustViewBounds="true"
39             android:padding="2dp"
40             app:imageUrl="@{property.imgSrcUrl}"
41             tools:src="@tools:sample/backgrounds/scenic"/>
42
43         <ImageView
44             android:id="@+id/mars_property_type"
45             android:layout_width="wrap_content"
46             android:layout_height="45dp"
47             android:layout_gravity="bottom|end"
48             android:adjustViewBounds="true"
49             android:padding="5dp"
50             android:scaleType="fitCenter"
51             android:src="@drawable/ic_for_sale_outline"
52             tools:src="@drawable/ic_for_sale_outline"
53             android:visibility="@{property.rental ? View.GONE : View.VISIBLE}"/>
54     </FrameLayout>
55 </layout>
```

7. Kompilasi dan jalankan aplikasi, dan perhatikan bahwa properti yang bukan rental memiliki ikon tanda dolar.

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)



---

## 4. Tugas: Memfilter hasil

Saat ini aplikasi Anda menampilkan semua properti Mars di kisi overview. Jika pengguna sedang berbelanja properti sewaan di Mars, memiliki ikon untuk menunjukkan properti mana yang tersedia untuk dijual akan berguna, tetapi masih ada banyak properti untuk di-scroll didalam halaman. Dalam tugas ini, Anda menambahkan menu opsi ke fragmen overview yang memungkinkan pengguna untuk hanya menampilkan persewaan, hanya properti untuk dijual, atau menampilkan semua.

Salah satu cara Anda dapat menyelesaikan tugas ini adalah dengan menguji jenis untuk setiap MarsProperty di kisi overview dan hanya menampilkan properti yang cocok. Namun, layanan web Mars yang sebenarnya memiliki parameter atau opsi kueri (disebut filter) yang memungkinkan Anda hanya mendapatkan properti dari jenis sewa atau beli. Anda dapat menggunakan kueri filter ini dengan URL web service realestate di browser seperti ini:

<https://android-kotlin-fun-mars-server.appspot.com/realestate?filter=buy>

Dalam tugas ini, Anda memodifikasi kelas MarsApiService untuk menambahkan opsi kueri ke permintaan layanan web dengan Retrofit. Kemudian Anda menghubungkan menu opsi untuk mengunduh ulang semua data properti Mars menggunakan opsi kueri itu. Karena respons yang Anda dapatkan dari layanan web hanya berisi properti yang Anda minati, Anda tidak perlu mengubah logika tampilan tampilan untuk kisi overview sama sekali.

### Langkah 1: Perbarui layanan Mars API

Untuk mengubah permintaan, Anda perlu mengunjungi kembali kelas MarsApiService yang Anda terapkan di codelab pertama dalam seri ini. Anda mengubah kelas untuk menyediakan API pemfilteran.

1. Buka network/MarsApiService.kt. Tepat di bawah impor, buat enum yang disebut MarsApiFilter untuk menentukan konstanta yang cocok dengan nilai kueri yang diharapkan layanan web.

```
enum class MarsApiFilter(val value: String) {  
    SHOW_RENT( value: "rent"),  
    SHOW_BUY( value: "buy"),  
    SHOW_ALL( value: "all") }  
}
```

2. Ubah metode getProperties() untuk mengambil input string untuk kueri filter, dan beri anotasi input itu dengan @Query("filter"), seperti yang ditunjukkan di bawah ini.

Impor retrofit2.http.Query saat diminta.

Anotasi @Query memberi tahu metode getProperties() (dan dengan demikian Retrofit) untuk membuat permintaan layanan web dengan opsi filter. Setiap kali getProperties() dipanggil, URL permintaan menyertakan bagian ?filter=type, yang mengarahkan layanan web untuk merespons dengan hasil yang cocok dengan kueri tersebut.

```
suspend fun getProperties(@Query( value: "filter") type: String): List<MarsProperty>
```

## Langkah 2: Perbarui model tampilan ikhtisar

Anda meminta data dari MarsApiService dalam metode getMarsRealEstateProperties() di OverviewViewModel. Sekarang Anda perlu memperbarui permintaan itu untuk mengambil argumen filter.

1. Buka overview/OverviewViewModel.kt. Anda akan melihat kesalahan di Android Studio karena perubahan yang Anda buat di langkah sebelumnya. Tambahkan MarsApiFilter (enum nilai filter yang mungkin) sebagai parameter ke panggilan getMarsRealEstateProperties().

Impor com.example.android.marsrealestate.network.MarsApiFilter jika diminta.

```
private fun getMarsRealEstateProperties(filter: MarsApiFilter) {
```

2. Ubah panggilan ke getProperties() di layanan Retrofit untuk meneruskan kueri filter tersebut sebagai string.

```
_properties.value = MarsApi.retrofitService.getProperties(filter.value)
```

3. Dalam blok init {}, teruskan MarsApiFilter.SHOW\_ALL sebagai argumen ke getMarsRealEstateProperties(), untuk menampilkan semua properti saat aplikasi pertama kali dimuat.

```
init {  
    getMarsRealEstateProperties(MarsApiFilter.SHOW_ALL)  
}
```

4. Di akhir kelas, tambahkan metode updateFilter() yang mengambil argumen MarsApiFilter dan memanggil getMarsRealEstateProperties() dengan argumen itu.

```
fun updateFilter(filter: MarsApiFilter) {  
    getMarsRealEstateProperties(filter)  
}
```

## Langkah 3: Hubungkan fragmen ke menu opsi

Langkah terakhir adalah menghubungkan menu tambahan ke fragmen untuk memanggil updateFilter() pada model tampilan saat pengguna memilih opsi menu.

1. Buka res/menu/overflow\_menu.xml. Aplikasi MarsRealEstate memiliki menu tambahan yang ada

yang menyediakan tiga opsi yang tersedia: menampilkan semua properti, hanya menampilkan persewaan, dan menampilkan properti yang hanya untuk dijual.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/show_all_menu"
        android:title="Show all" />
    <item
        android:id="@+id/show_rent_menu"
        android:title="Rent" />
    <item
        android:id="@+id/show_buy_menu"
        android:title="Buy" />
</menu>
```

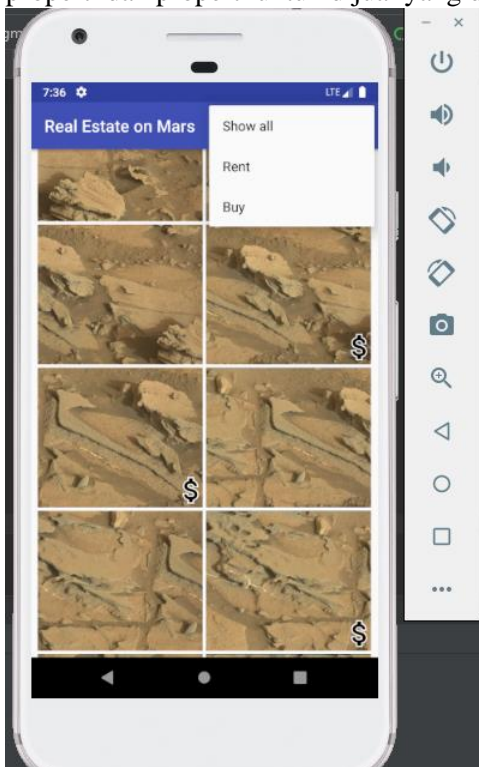
2. Buka `overview/OverviewFragment.kt`. Di akhir kelas, implementasikan metode `onOptionsItemSelected()` untuk menangani pemilihan item menu.

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
```

3. Di `onOptionsItemSelected()`, panggil metode `updateFilter()` pada model tampilan dengan filter yang sesuai. Gunakan Kotlin blok `when {}` untuk beralih di antara opsi. Gunakan `MarsApiFilter.SHOW_ALL` untuk nilai filter default. Kembalikan `true`, karena Anda telah menangani item menu. Impor `MarsApiFilter` (`com.example.android.marsrealestate.network.MarsApiFilter`) jika diminta. Metode lengkap `onOptionsItemSelected()` ditampilkan di bawah ini.

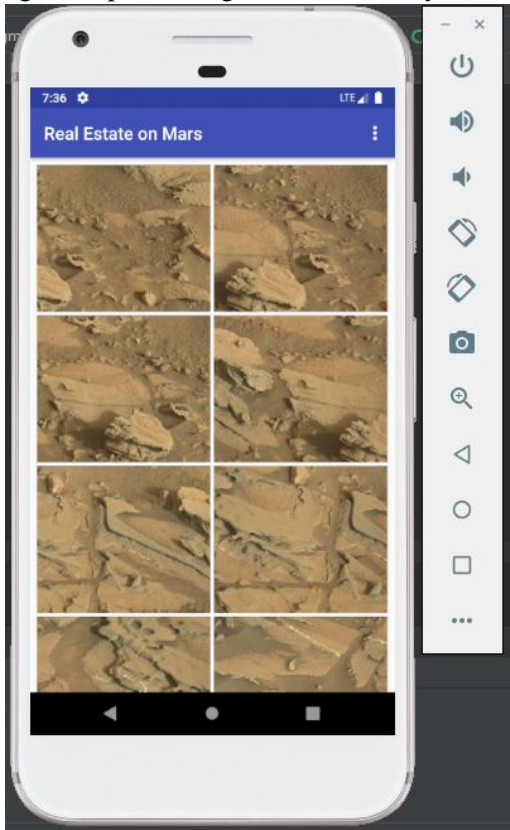
```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    viewModel.updateFilter(
        when (item.itemId) {
            R.id.show_rent_menu -> MarsApiFilter.SHOW_RENT
            R.id.show_buy_menu -> MarsApiFilter.SHOW_BUY
            else -> MarsApiFilter.SHOW_ALL
        }
    )
    return true
}
```

4. Kompilasi dan jalankan aplikasi. Aplikasi meluncurkan kisi overview pertama dengan semua jenis properti dan properti untuk dijual yang ditandai dengan ikon dolar.

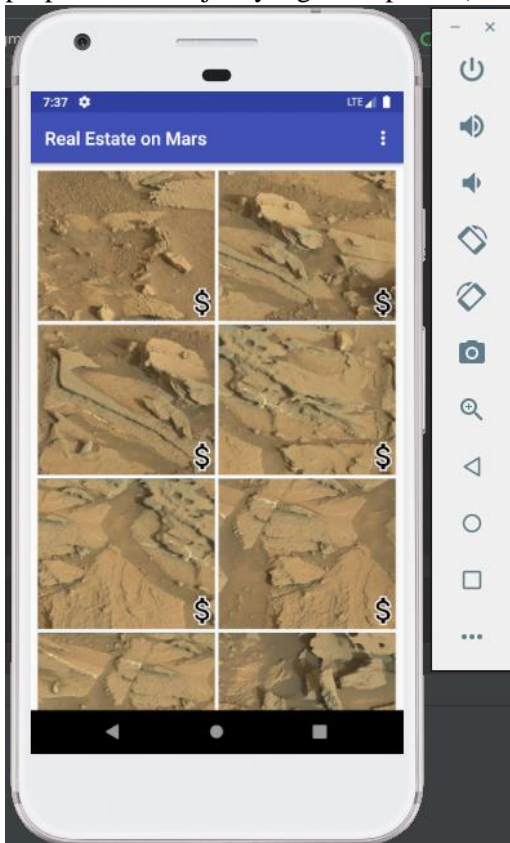




5. Pilih Rent dari menu opsi. Properti dimuat ulang dan tidak ada satupun yang muncul dengan ikon dolar. (Hanya properti persewaan yang ditampilkan.) Anda mungkin harus menunggu beberapa saat agar tampilan disegarkan untuk hanya menampilkan properti yang difilter.



6. Pilih Buy dari menu opsi. Properti dimuat ulang, dan semuanya muncul dengan ikon dolar. (Hanya properti untuk dijual yang ditampilkan.)





---

## 5. Tugas: Membuat halaman detail dan mengatur navigasi

Sekarang Anda memiliki kisi ikon yang bergulir untuk properti Mars, tetapi inilah saatnya untuk mendapatkan detail lebih lanjut. Dalam tugas ini, Anda menambahkan fragmen detail untuk menampilkan detail properti tertentu. Fragmen detail akan menampilkan gambar yang lebih besar, harga, dan jenis propertinya — apakah itu sewa atau untuk dijual.

Fragmen ini diluncurkan saat pengguna mengetuk gambar di kisi ringkasan. Untuk melakukannya, Anda perlu menambahkan listener onClick ke item kisi RecyclerView, lalu menuju ke fragmen baru. Anda menavigasi dengan memicu perubahan LiveData di ViewModel, seperti yang telah Anda lakukan sepanjang pelajaran ini. Anda juga menggunakan plugin Safe Args dari komponen Navigation untuk meneruskan informasi MarsProperty yang dipilih dari fragmen overview ke fragmen detail.

### Langkah 1: Buat view model detail dan perbarui layout detail

Mirip dengan proses yang Anda gunakan untuk overview view model dan fragmen overview, Anda sekarang perlu mengimplementasikan view model dan file layout untuk detail fragmen.

1. Buka detail/DetailViewModel.kt. Sama seperti file Kotlin terkait jaringan yang terdapat dalam folder network dan file overview dalam overview, folder detail berisi file yang terkait dengan tampilan detail. Perhatikan bahwa kelas DetailViewModel (kosong sekarang) menggunakan marsProperty sebagai parameter di konstruktor.

```
class DetailViewModel(marsProperty: MarsProperty,
                     app: Application) : AndroidViewModel(app) {
```

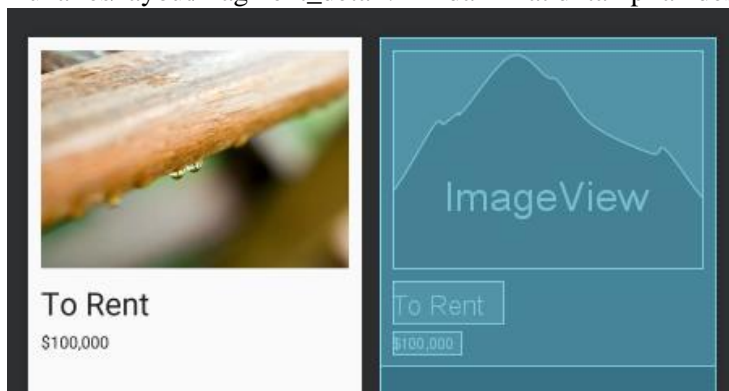
2. Di dalam definisi kelas, tambahkan LiveData untuk properti Mars yang dipilih, untuk memaparkan informasi tersebut ke tampilan detail. Ikuti pola biasa dalam membuat MutableLiveData untuk menampung MarsProperty itu sendiri, lalu buka properti LiveData publik yang tidak dapat diubah. Impor androidx.lifecycle.LiveData dan impor androidx.lifecycle.MutableLiveData jika diminta.

```
private val _selectedProperty = MutableLiveData<MarsProperty>()
val selectedProperty: LiveData<MarsProperty>
    get() = _selectedProperty
```

3. Buat blok init {} dan setel nilai properti Mars yang dipilih dengan objek MarsProperty dari konstruktor.

```
init {
    _selectedProperty.value = marsProperty
}
```

4. Buka res/layout/fragment\_detail.xml dan lihat di tampilan desain.



Ini adalah file layout untuk detail fragmen. Ini berisi ImageView untuk foto besar, TextView untuk properti type (sewa/rent atau jual/buy) dan TextView untuk harga/price. Perhatikan bahwa layout pembatas dibungkus dengan ScrollView sehingga secara otomatis akan di-scroll jika tampilan menjadi terlalu besar untuk tampilan, misalnya saat pengguna melihatnya dalam mode lanskap.

5. Buka tab Text untuk layout. Di bagian atas layout, tepat sebelum elemen <ScrollView>, tambahkan elemen <data> untuk mengaitkan model tampilan detail dengan tata letak.

```
<data>
  <variable
    name="viewModel"
    type="com.example.android.marsrealestate.detail.DetailViewModel" />
</data>
```

6. Tambahkan atribut app:imageUrl ke elemen ImageView. Setel ke imgUrl dari properti model tampilan yang dipilih. Adaptor binding yang memuat gambar menggunakan Glide secara otomatis juga akan digunakan di sini, karena adaptor tersebut mengawasi semua atribut app:imageUrl.

```
app:imageUrl="@{viewModel.selectedProperty.imgSrcUrl}"/>
```

## Langkah 2: Tentukan navigasi dalam overview view model

Saat pengguna menge-tap foto dalam model overview, ini akan memicu navigasi ke fragmen yang menampilkan detail tentang item yang diklik.

1. Buka overview/OverviewViewModel.kt. Tambahkan properti \_navigateToSelectedProperty MutableLiveData dan tampilkan dengan LiveData yang tidak dapat diubah.

Saat LiveData ini berubah menjadi non-null, navigasi akan dipicu. (Anda akan segera menambahkan kode untuk mengamati variabel ini dan memicu navigasi.)

2. Di akhir kelas, tambahkan metode displayPropertyDetails() yang menyetel \_navigateToSelectedProperty ke properti Mars yang dipilih.

```
fun displayPropertyDetails(marsProperty: MarsProperty) {
    _navigateToSelectedProperty.value = marsProperty
}
```

3. Tambahkan metode displayPropertyDetailsComplete() yang membatalkan nilai \_navigateToSelectedProperty. Anda memerlukan ini untuk menandai status navigasi selesai, dan untuk menghindari navigasi dipicu lagi saat pengguna kembali dari tampilan detail.

```
fun displayPropertyDetailsComplete() {
    _navigateToSelectedProperty.value = null
}
```

## Langkah 3: Siapkan pendengar klik di adaptor dan fragmen kisi

1. Buka overview/PhotoGridAdapter.kt. Di akhir kelas, buat kelas OnClickListener khusus yang mengambil lambda dengan parameter marsProperty. Di dalam kelas, tentukan fungsi onClick() yang disetel ke parameter lambda.

```
class OnClickListener(val clickListener: (marsProperty:MarsProperty) -> Unit) {
    fun onClick(marsProperty:MarsProperty) = clickListener(marsProperty)
}
```

2. Gulir ke atas ke definisi kelas untuk PhotoGridAdapter, dan tambahkan properti OnClickListener private ke konstruktor.

```
class PhotoGridAdapter( private val onClickListener: OnClickListener ) :
    ListAdapter<MarsProperty,
        PhotoGridAdapter.MarsPropertyViewHolder>(DiffCallback) {
```

3. Jadikan foto dapat diklik dengan menambahkan onClickListener ke item kisi di metode onBindViewHolder(). Tentukan pemroses klik di antara panggilan ke getItem() dan bind().

```
override fun onBindViewHolder(holder: MarsPropertyViewHolder, position: Int) {
    val marsProperty = getItem(position)
    holder.itemView.setOnClickListener { it: View!
        onClickListener.onClick(marsProperty)
    }
    holder.bind(marsProperty)
}
```

4. Buka overview/OverviewFragment.kt. Dalam metode onCreateView(), ganti baris yang menginisialisasi properti binding.photosGrid.adapter dengan baris yang ditunjukkan di bawah ini.

Kode ini menambahkan objek PhotoGridAdapter.onClickListener ke konstruktor PhotoGridAdapter, dan memanggil viewModel.displayPropertyDetails() dengan objek MarsProperty yang diteruskan. Ini memicu LiveData dalam model tampilan untuk navigasi.

```
// Sets the adapter of the photosGrid RecyclerView
binding.photosGrid.adapter = PhotoGridAdapter(PhotoGridAdapter.OnClickListener { it: MarsProperty
    viewModel.displayPropertyDetails(it)
})
```

#### Langkah 4: Ubah grafik navigasi dan buat MarsProperty menjadi parcelable

Saat pengguna mengetuk foto di kisi overview, aplikasi harus menavigasi ke fragmen detail dan melewati detail properti Mars yang dipilih sehingga tampilan detail dapat menampilkan informasi itu. Saat ini Anda memiliki listener klik dari PhotoGridAdapter untuk menangani ketukan, dan cara untuk memicu navigasi dari model tampilan. Tetapi Anda belum memiliki objek MarsProperty yang diteruskan ke fragmen detail. Untuk itu Anda menggunakan Safe Args dari komponen navigasi.

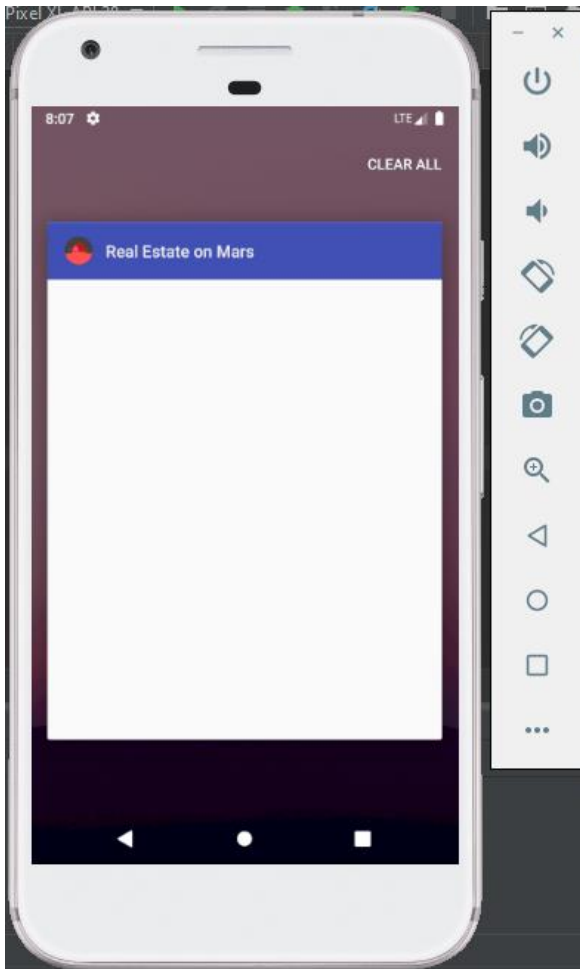
1. Buka res/navigation/nav\_graph.xml. Klik tab Text untuk melihat kode XML untuk grafik navigasi.
2. Di dalam elemen <fragment> untuk detail fragment, tambahkan elemen <argument> yang ditunjukkan di bawah ini. Argumen ini, disebut selectedProperty, memiliki tipe MarsProperty.

```
<argument
    android:name="selectedProperty"
    app:argType="com.example.android.marsrealestate.network.MarsProperty"
/>
```

3. Kompilasi aplikasi. Navigasi memberi Anda kesalahan karena MarsProperty tidak dapat parcelable. Antarmuka Parcelable memungkinkan objek untuk diserialkan, sehingga data objek dapat diteruskan di antara fragmen atau aktivitas. Dalam kasus ini, agar data di dalam objek MarsProperty diteruskan ke fragmen detail melalui Safe Args, MarsProperty harus mengimplementasikan antarmuka Parcelable. Kabar baiknya adalah Kotlin menyediakan jalan pintas yang mudah untuk mengimplementasikan antarmuka itu.

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)



-> aplikasi tidak bisa dibuka karena navigasi error

4. Buka `network/MarsProperty.kt`. Tambahkan anotasi `@Parcelize` ke definisi kelas. Impor `kotlinx.android.parcel.Parcelize` jika diminta. Anotasi `@Parcelize` menggunakan ekstensi Kotlin Android untuk secara otomatis mengimplementasikan metode dalam antarmuka `Parcelable` untuk kelas ini. Anda tidak perlu melakukan apapun!

```
@Parcelize
data class MarsProperty(
```

5. Ubah definisi kelas `MarsProperty` untuk memperluas `Parcelable`. Impor `android.os.Parcelable` saat diminta. Definisi kelas `MarsProperty` sekarang terlihat seperti ini:

```
@Parcelize
data class MarsProperty(
    val id: String,
    // used to map img_src from the JSON to imgSrcUrl in our class
    @Json(name = "img_src") val imgSrcUrl: String,
    val type: String,
    val price: Double): Parcelable {
```

## Langkah 5: Hubungkan fragmen

Anda masih belum menavigasi — navigasi sebenarnya terjadi di fragmen. Pada langkah ini, Anda menambahkan hal terakhir untuk mengimplementasikan navigasi antara overview dan detail fragmen.

1. Buka `overview/OverviewFragment.kt`. Di `onCreateView()`, di bawah baris yang menginisialisasi adaptor kisi foto (`PhotoGridAdapter`), tambahkan baris yang ditunjukkan di bawah ini untuk mengamati `navigatedToSelectedProperty` dari model tampilan overview.

Impor `androidx.lifecycle.Observer` dan impor `androidx.navigation.fragment.findNavController` jika diminta.

Pengamat menguji apakah `MarsProperty` — itu di lambda — bukan null, dan jika demikian, ia mendapatkan pengontrol navigasi dari fragmen dengan `findNavController()`. Panggil `displayPropertyDetailsComplete()` untuk memberi tahu model tampilan agar menyetel ulang `LiveData` ke status null, jadi Anda tidak akan memicu navigasi lagi secara tidak sengaja saat aplikasi kembali ke `OverviewFragment`.

```
viewModel.navigateToSelectedProperty.observe( owner: this, Observer { it: MarsProperty?
    if ( null != it ) {
        this.findNavController().navigate(
            OverviewFragmentDirections.actionShowDetail(it))
        viewModel.displayPropertyDetailsComplete()
    }
})
```

2. Buka `detail/DetailFragment.kt`. Tambahkan baris ini tepat di bawah pengaturan properti `binding.lifecycleOwner` dalam metode `onCreateView()`. Baris ini mendapatkan objek `MarsProperty` yang dipilih dari `Safe Args`.

Perhatikan penggunaan operator assertion not-null Kotlin (!!). Jika `selectedProperty` tidak ada di sana, sesuatu yang buruk telah terjadi dan Anda benar-benar ingin kode tersebut membuang pointer nol. (Dalam kode produksi, Anda harus menangani kesalahan itu dengan cara tertentu.)

```
val marsProperty = DetailFragmentArgs.fromBundle(arguments!!).selectedProperty
```

3. Tambahkan baris ini berikutnya, untuk mendapatkan `DetailViewModelFactory` baru. Anda akan menggunakan `DetailViewModelFactory` untuk mendapatkan instance `DetailViewModel`. Aplikasi starter menyertakan implementasi `DetailViewModelFactory`, jadi yang harus Anda lakukan di sini adalah menginisialisasinya.

```
val viewModelFactory = DetailViewModelFactory(marsProperty, application)
```

4. Terakhir, tambahkan baris ini untuk mendapatkan `DetailViewModel` dari factory dan untuk menghubungkan semua bagian.

```
binding.viewModel = ViewModelProvider(
    owner: this, viewModelFactory).get(DetailViewModel::class.java)
```

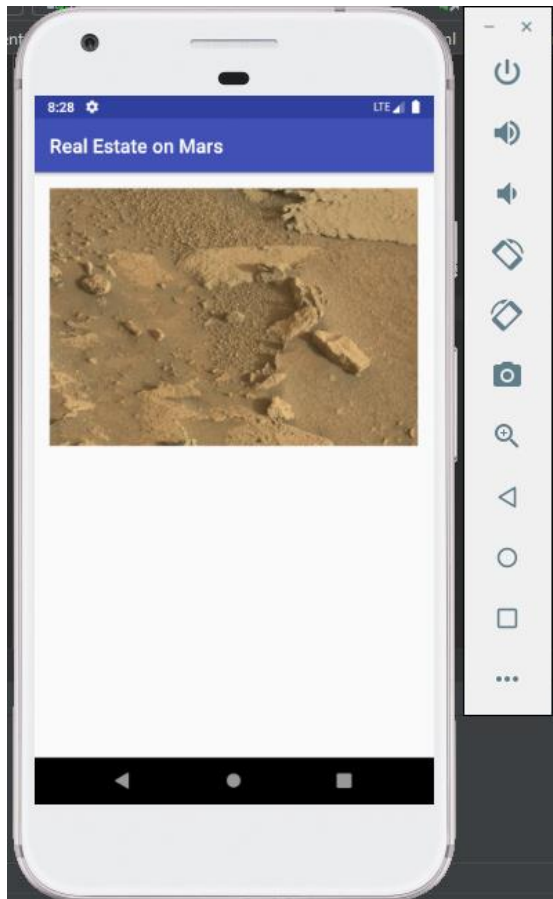
5. Kompilasi dan jalankan aplikasi, dan ketuk foto properti Mars mana saja. Fragmen detail muncul untuk detail properti itu. Ketuk tombol Kembali untuk kembali ke halaman ikhtisar, dan perhatikan bahwa layar detail masih agak jarang. Anda selesai menambahkan data properti ke halaman detail tersebut di tugas berikutnya.

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)



---

## 6. Tugas: Membuat halaman detail yang lebih berguna

Saat ini halaman detail hanya menampilkan foto Mars yang sama yang biasa Anda lihat di halaman overview. Kelas `MarsProperty` juga memiliki properti `type` (sewa atau beli) dan properti `price`. Layar detail harus menyertakan kedua nilai ini, dan akan membantu jika properti persewaan menunjukkan bahwa `price` adalah nilai per bulan. Anda menggunakan transformasi `LiveData` dalam view model untuk mengimplementasikan kedua hal tersebut.

1. Buka `res/values/strings.xml`. Kode awal menyertakan sumber daya string, yang ditunjukkan di bawah ini, untuk membantu Anda membuat string untuk tampilan detail. Untuk harga, Anda akan menggunakan resource `display_price_monthly_rental` atau resource `display_price`, bergantung pada jenis propertinya.

```
<string name="type_rent">Rent</string>
<string name="type_sale">Sale</string>
<string name="display_type">For %s</string>
<string name="display_price_monthly_rental">$$,.0f/month</string>
<string name="display_price">$$,.0f</string>
```

2. Buka `detail/DetailViewModel.kt`. Di bagian bawah kelas, tambahkan kode yang ditunjukkan di bawah ini. Impor `androidx.lifecycle.Transformations` jika diminta.

Transformasi ini menguji apakah properti yang dipilih adalah rental, menggunakan pengujian yang sama dari tugas pertama. Jika properti adalah persewaan, transformasi memilih string yang sesuai dari sumber daya dengan Kotlin `switch when {}`. Kedua string ini membutuhkan angka di bagian akhir, jadi Anda menggabungkan `property.price` sesudahnya.



```

val displayPropertyPrice = Transformations.map(selectedProperty) { it: MarsProperty!
    app.applicationContext.getString(
        when (it.isRental) {
            true -> "$%,.0f/month"
            false -> "$%,.0f"
        }, it.price)

```

3. Impor kelas R yang dihasilkan untuk mendapatkan akses ke sumber daya string dalam proyek.

```

import com.example.android.marsrealestate.R

```

4. Setelah transformasi displayPropertyPrice, tambahkan kode di bawah ini. Transformasi ini menggabungkan beberapa sumber daya string, berdasarkan apakah tipe properti tersebut adalah rental.

```

val displayPropertyType = Transformations.map(selectedProperty) { it: MarsProperty!
    "For {app.applicationContext.getString(switch (it.isRent..."
}

```

5. Buka res/layout/fragment\_detail.xml. Hanya ada satu hal lagi yang harus dilakukan, dan itu adalah mengikat string baru (yang Anda buat dengan transformasi LiveData) ke tampilan detail. Untuk melakukannya, Anda menyetel nilai field teks untuk teks properti type ke viewModel.displayPropertyType, dan field teks untuk teks nilai price ke viewModel.displayPropertyPrice.

```

<TextView
    android:id="@+id/property_type_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:textColor="#de000000"
    android:textSize="39sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/main_photo_image"
    android:text="@{viewModel.displayPropertyType}"
    tools:text="To Rent" />

<TextView
    android:id="@+id/price_value_text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:textColor="#de000000"
    android:textSize="20sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/property_type_text"
    android:text="@{viewModel.displayPropertyPrice}"
    tools:text="$100,000" />

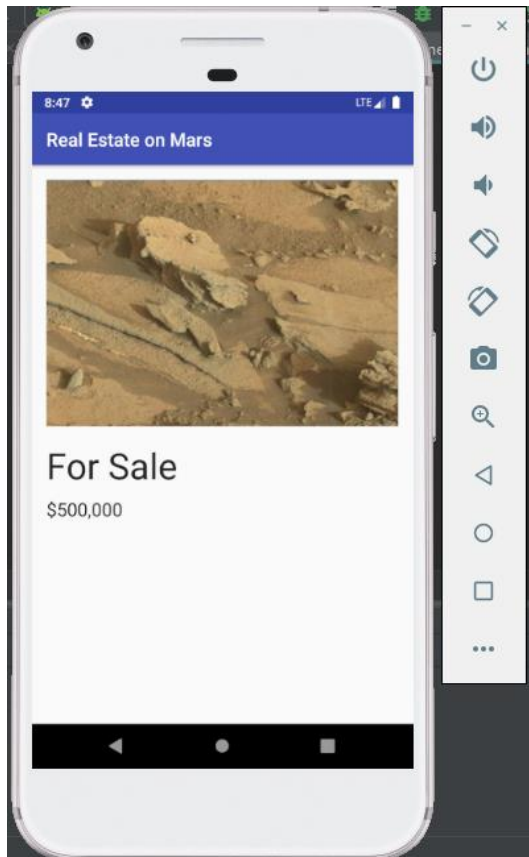
```

6. Kompilasi dan jalankan aplikasi. Sekarang semua data properti muncul di halaman detail, dengan format yang bagus.

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)



## TUGAS

### 1. Gantilah ikon tanda dolar dengan teks: "Dijual".

1. Langsung saja menuju ke `grid_view_item.xml`, lalu perhatikan `ImageView` berikut:

```
<ImageView
    android:id="@+id/mars_property_type"
    android:layout_width="wrap_content"
    android:layout_height="45dp"
    android:layout_gravity="bottom|end"
    android:adjustViewBounds="true"
    android:padding="5dp"
    android:scaleType="fitCenter"
    android:src="@drawable/ic_for_sale_outline"
    tools:src="@drawable/ic_for_sale_outline"/>
```

Baris-baris kode tersebut adalah baris kode untuk menampilkan icon dollar dengan memanggil widget `drawable` bernama `ic_for_sale_outline`. Karena kita ingin mengganti icon dollar menjadi teks, sehingga ada beberapa hal yang perlu diubah dari kode-kode tersebut yang akan kita bahas di nomor 2.

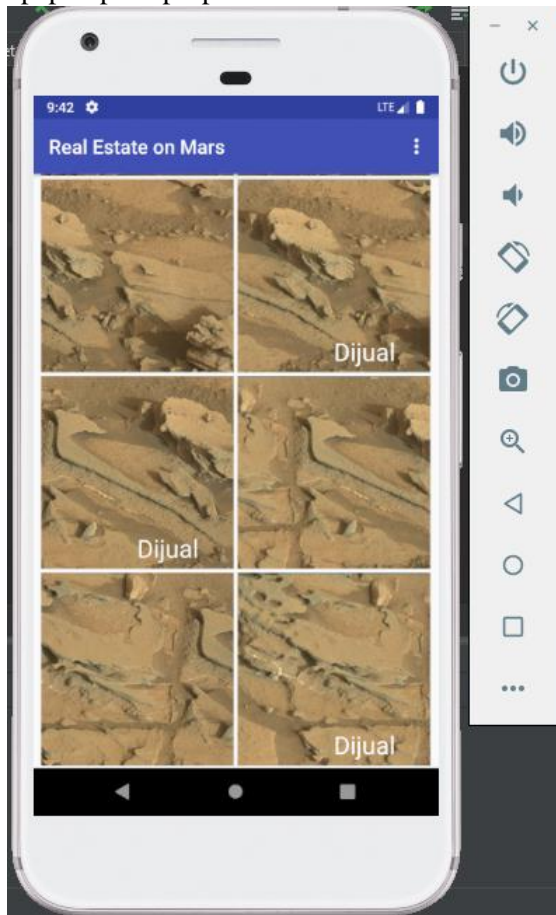
2. Ubah beberapa kode yaitu dengan mengganti `ImageView` menjadi `TextView` (karena ingin menampilkan text), kemudian set beberapa hal yang berkaitan dengan text yang akan ditampilkan yaitu tulisan pada `textview` yaitu **`android:text="Dijual"`**, ukuran text yaitu **`android:textSize="25dp"`**, warna putih text yaitu **`android:textColor="#FFFFFF"`**. Kemudian terakhir tulis kode **`android:visibility="@{property.rental ? View.GONE : View.VISIBLE}"`** yang berarti jika properti adalah rent maka tulisan "Dijual" tadi tidak akan muncul, kebalikannya jika properti adalah buy maka akan muncul `textview` "Dijual".

```

<TextView
    android:id="@+id/mars_property_type"
    android:layout_width="105dp"
    android:layout_height="45dp"
    android:layout_gravity="bottom|end"
    android:adjustViewBounds="true"
    android:padding="5dp"
    android:scaleType="fitCenter"
    android:text="Dijual"
    android:textSize="25dp"
    android:textColor="#FFFFFF"
    android:visibility="@{property.rental ? View.GONE : View.VISIBLE}" />

```

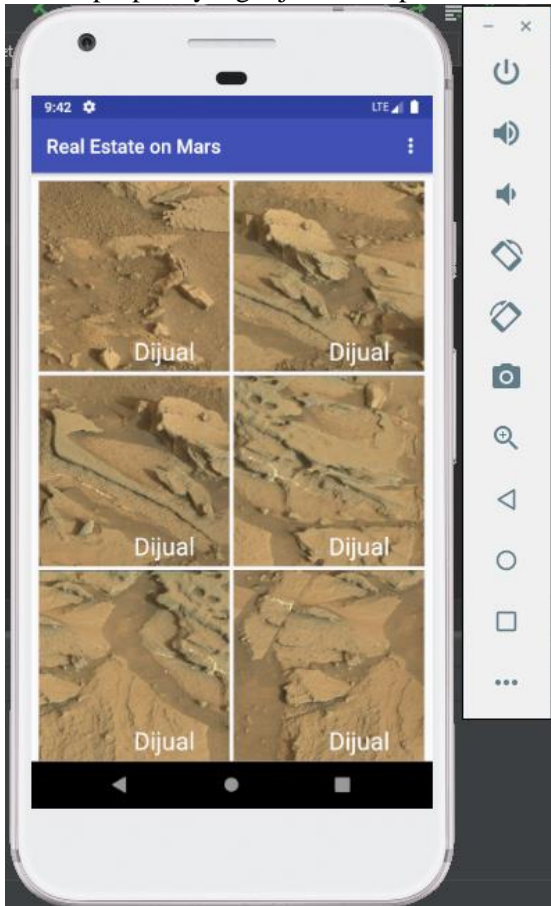
3. Jalankan aplikasi. Perhatikan bahwa muncul tulisan “Dijual” pada properti buy dan tidak ada tulisan apapun pada properti rent.



Klik “buy” pada opsi untuk menampilkan semua properti yang dijual



Semua properti yang dijual ditampilkan



## 2. Kemudian untuk yang properti type = “rent”, tambahkan ikon teks: “Disewa”

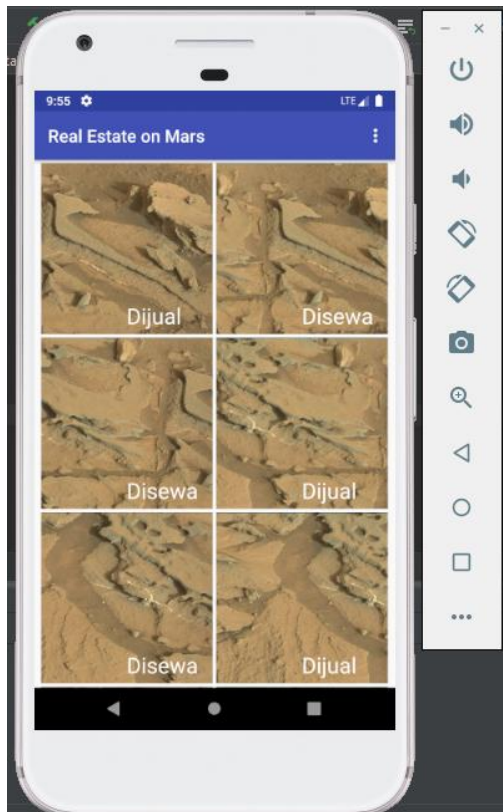
1. Masih di grid\_view\_item.xml. Kita tambahkan TextView yang menempati posisi yang sama di layout dengan TextView sebelumnya, yaitu:

```
<TextView
    android:id="@+id/mars_property_type2"
    android:layout_width="105dp"
    android:layout_height="45dp"
    android:layout_gravity="bottom|end"
    android:adjustViewBounds="true"
    android:padding="5dp"
    android:scaleType="fitCenter"
    android:text="Disewa"
    android:textSize="25dp"
    android:textColor="#FFFFFF"
    android:visibility="@{property.rental ? View.VISIBLE : View.GONE}" />
```

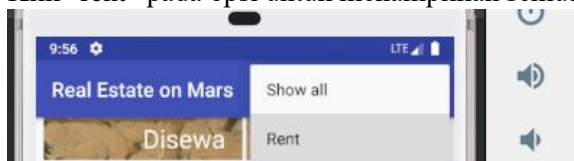
Bisa kita perhatikan bahwa ada beberapa perbedaan yaitu text yang diganti dari “Dijual” menjadi “Disewa”, dan bagian yang paling penting adalah mengganti android:visibility menjadi **android:visibility="@{property.rental ? View.VISIBLE : View.GONE}"**. arti dari baris kode tersebut adalah jika properti adalah rent maka textview “Disewa” akan ditampilkan (VISIBLE) dan jika bukan rent maka textview “Disewa” tidak akan ditampilkan (GONE).

2. Jalankan aplikasi. Sekarang di halaman utama sudah ditampilkan tulisan “Disewa” beserta tulisan “Dijual”.

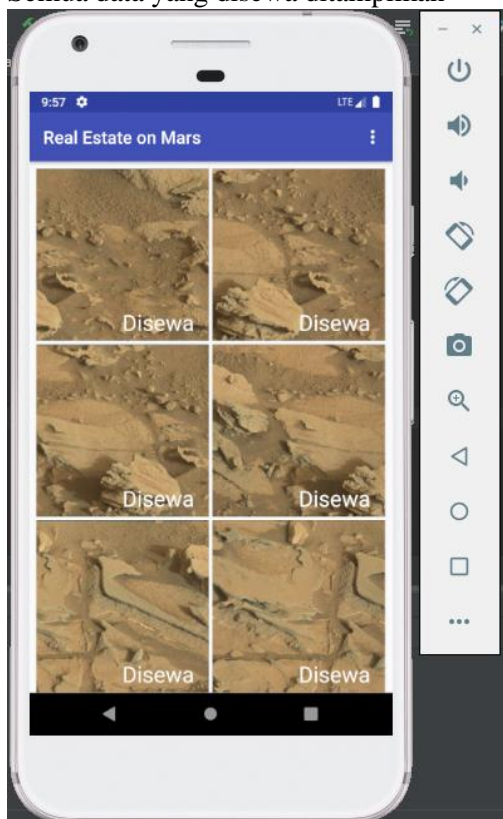
(screenshot aplikasi di halaman selanjutnya)



Klik “rent” pada opsi untuk menampilkan semua data yang disewa



Semua data yang disewa ditampilkan



---

## **KESIMPULAN**

Di pertemuan terakhir ini, saya berhasil memenuhi tujuan dibuatnya modul ke-14 ini yaitu mampu membuat aplikasi yang dapat memfilter dan menampilkan gambar dari internet menggunakan Retrofit, Moshi dan Glide. Pada bagian praktik, kita melakukan banyak hal yang berkaitan dengan filtering yaitu mengubah aplikasi MarsRealEstate untuk menandai properti Mars yang akan dijual (versus yang disewakan) dengan ikon tanda dolar, menggunakan menu opsi di halaman layout untuk membuat permintaan layanan web yang memfilter properti Mars berdasarkan jenis, dan membuat fragmen detail untuk properti Mars serta menghubungkan fragmen itu ke grid layout dengan navigasi, dan meneruskan data properti ke dalam fragmen itu. Di bagian terakhir, kita membuat tugas untuk mengganti ikon dolar (imageview) menjadi text “Dijual”(textview) dan menambahkan textview “Disewa” untuk properti yang disewa.

**Terima Kasih**