LAPORAN PRAKTIKUM
PEMROGRAMAN BERBASIS MOBILE
PERTEMUAN KE-13

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-13
(MENGAMBIL DATA DAN MENAMPILKAN GAMBAR DARI INTERNET)

TUJUAN
Mahasiswa dapat membuat aplikasi memuat dan menampilkan gambar dari internet menggunakan Retrofit, Moshi dan Glide.

DASAR TEORI

Apa yang akan Anda Pelajari

- Cara menggunakan pustaka Glide untuk memuat dan menampilkan gambar dari URL web.
- Cara menggunakan RecyclerView dan adaptor grid untuk menampilkan gambar secara grid.
- Bagaimana menangani potensi kesalahan saat gambar diunduh dan ditampilkan.

Apa yang akan Anda lakukan

- Ubah aplikasi MarsRealEstate untuk mendapatkan URL gambar dari data properti Mars, dan menggunakan Glide untuk memuat dan menampilkan gambar itu.
- Tambahkan animasi pemuatan dan ikon kesalahan ke aplikasi.
- Gunakan RecyclerView untuk menampilkan gambar secara grid properti Mars.
- Tambahkan status dan penanganan error ke RecyclerView.

2. Ikhtisar aplikasi

Dalam modul ini (dan modul sebelumnya), 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.

Versi aplikasi yang Anda buat di codelab ini mengisi halaman ringkasan, yang menampilkan kisi gambar. Gambar adalah bagian dari data properti yang diperoleh aplikasi Anda dari layanan web real estate Mars. Aplikasi Anda akan menggunakan pustaka Glide untuk memuat dan menampilkan gambar, dan RecyclerView untuk membuat tata letak grid untuk gambar. Aplikasi Anda juga akan menangani kesalahan jaringan dengan baik.

3. Tugas: Menampilkan image internet

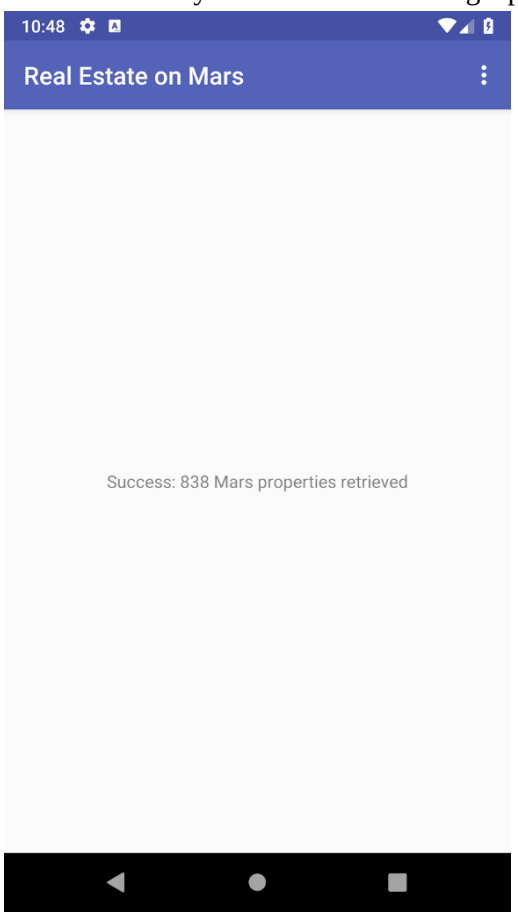
Menampilkan foto dari URL web mungkin terdengar mudah, tetapi ada sedikit rekayasa untuk membuatnya berfungsi dengan baik. Gambar harus diunduh, di-buffer, dan diterjemahkan dari format terkompresi menjadi gambar yang dapat digunakan Android. Gambar harus di-cache ke cache dalam memori, cache berbasis penyimpanan, atau keduanya. Semua ini harus terjadi di thread latar belakang dengan prioritas rendah sehingga UI tetap responsif. Selain itu, untuk performa jaringan dan CPU terbaik, Anda mungkin ingin mengambil dan mendekode lebih dari satu gambar sekaligus. Mempelajari cara memuat gambar secara efektif dari jaringan bisa menjadi codelab itu sendiri.

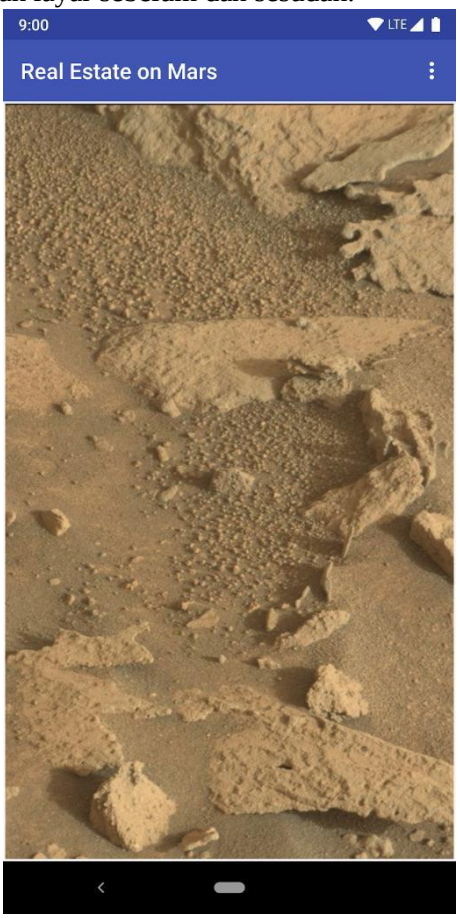
Untungnya, Anda dapat menggunakan pustaka yang dikembangkan komunitas yang disebut Glide untuk mengunduh, menyangga, mendekode, dan menyimpan gambar Anda ke dalam cache. Glide meninggalkan Anda dengan pekerjaan yang jauh lebih sedikit daripada jika Anda harus melakukan semua ini dari awal.

Glide pada dasarnya membutuhkan dua hal:

- URL gambar yang ingin Anda muat dan tampilkan.
- Objek ImageView untuk menampilkan gambar itu.

Dalam tugas ini, Anda mempelajari cara menggunakan Glide untuk menampilkan satu gambar dari layanan web real estate. Anda menampilkan gambar yang mewakili properti Mars pertama dalam daftar properti yang dikembalikan layanan web. Berikut tangkapan layar sebelum dan sesudah:

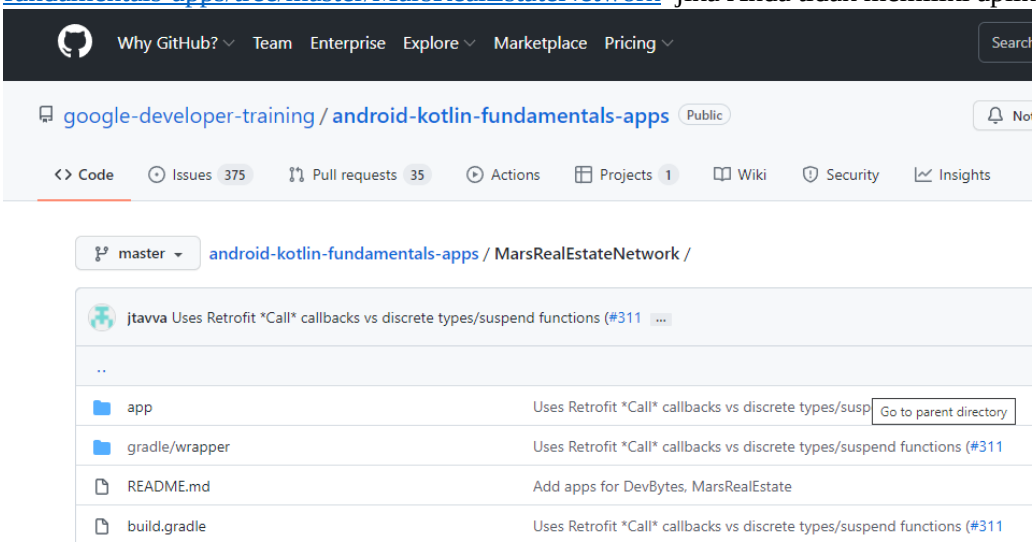




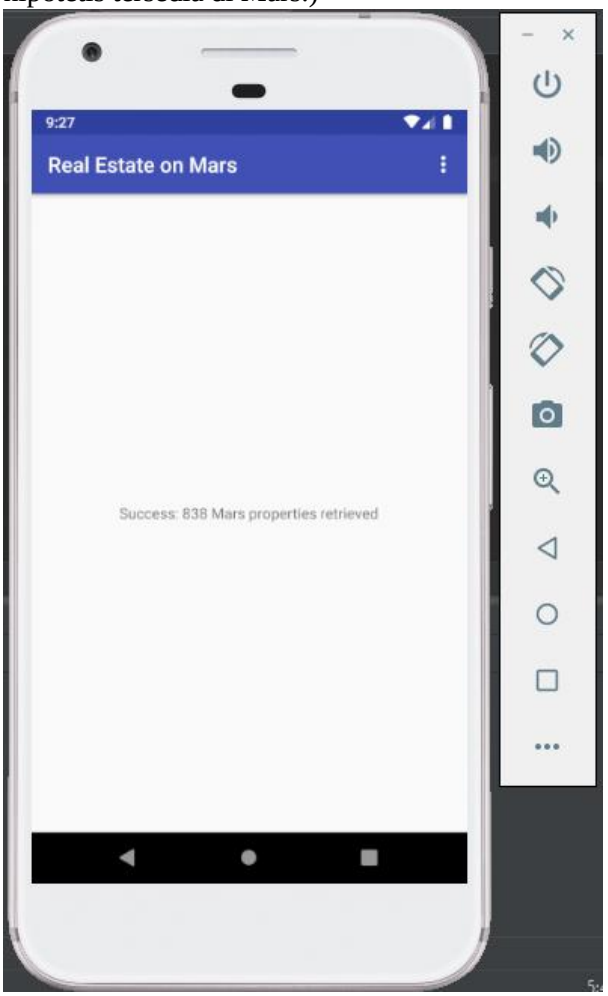
PRAKTIK

Langkah 1: Tambahkan dependensi Glide

1. Buka aplikasi MarsRealEstate dari codelab terakhir. (Anda dapat mengunduh MarsRealEstateNetwork di https://github.com/google-developer-training/android-kotlin-fundamentals-apps/tree/master/MarsRealEstateNetwork jika Anda tidak memiliki aplikasinya.)



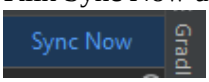
2. Jalankan aplikasi untuk melihat fungsinya. (Ini menampilkan detail teks dari properti yang secara hipotetis tersedia di Mars.)



3. Buka build.gradle (Module: app).

4. Di bagian dependensi, tambahkan baris ini untuk pustaka Glide:
   implementation "com.github.bumptech.glide:glide:$version_glide"

5. Klik Sync Now untuk membangun kembali proyek dengan dependensi baru.



Langkah 2: Perbarui model tampilan

Selanjutnya Anda memperbarui kelas OverviewViewModel untuk menyertakan data langsung untuk satu properti Mars.

1. Buka overview/OverviewViewModel.kt. Tepat di bawah LiveData untuk _response, tambahkan live data internal (mutable/dapat diubah) dan eksternal (immutable/tidak dapat diubah) untuk satu objek MarsProperty. Import kelas MarsProperty (com.example.android.marsrealestate.network.MarsProperty) saat dibutuhkan.

```
private val _property = MutableLiveData<MarsProperty>()

val property: LiveData<MarsProperty>
    get() = _property
```

2. Dalam metode getMarsRealEstateProperties(), temukan baris di dalam blok try/catch {} yang menyetel _response.value ke jumlah properti. Tambahkan tes yang ditunjukkan di bawah ini setelah try/catch. Jika objek MarsProperty tersedia, pengujian ini menetapkan nilai _property LiveData ke properti pertama di listResult.

```
try {
    val listResult = MarsApi.retrofitService.getProperties()
    _response.value = "Success: ${listResult.size} Mars properties retrieved"
    if (listResult.size > 0) {
        _property.value = listResult[0]
    }
} catch (e: Exception) {
    _response.value = "Failure: ${e.message}"
}
```

3. Buka file res/layout/fragment_overview.xml. Di elemen <TextView>, ubah android:text untuk mengikat ke komponen imgSrcUrl dari properti LiveData:

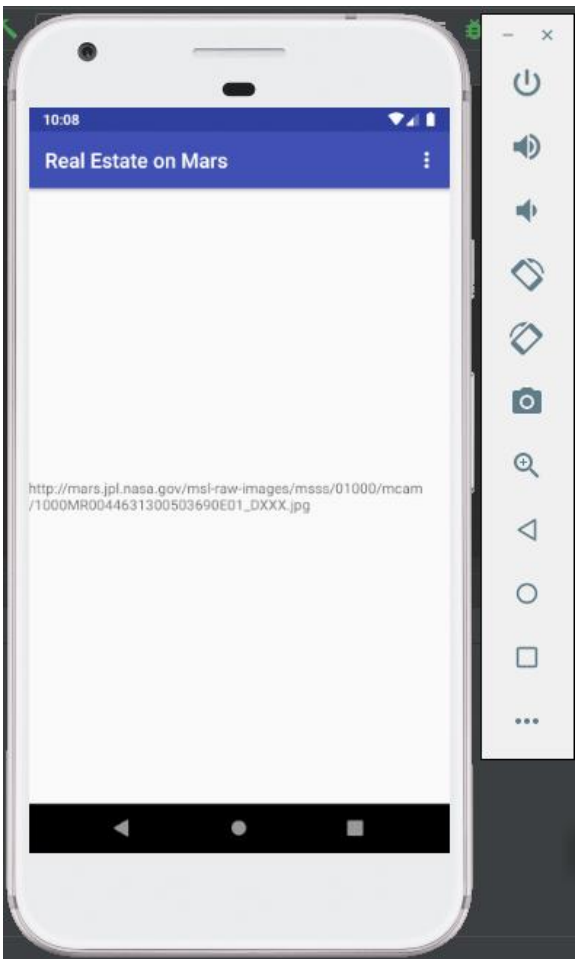
```
android:text="@{viewModel.property.imgSrcUrl}"
```

4. Jalankan aplikasi. TextView hanya menampilkan URL gambar di properti Mars yang pertama. Yang telah Anda lakukan sejauh ini adalah menyiapkan model tampilan dan data langsung untuk URL tersebut.

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)



Langkah 3: Buat adaptor pengikat dan panggil Glide

Sekarang Anda memiliki URL gambar untuk ditampilkan, dan inilah waktunya untuk mulai bekerja dengan Glide untuk memuat gambar itu. Dalam langkah ini, Anda menggunakan binding adater untuk mengambil URL dari atribut XML yang terkait dengan ImageView, dan Anda menggunakan Glide untuk memuat gambar. Adaptor binding adalah metode ekstensi yang berada di antara tampilan dan data terikat untuk memberikan perilaku kustom saat data berubah. Dalam hal ini, perilaku kustomnya adalah memanggil Glide untuk memuat gambar dari URL ke ImageView.

1. Buka BindingAdapters.kt. File ini akan menampung adaptor pengikat yang Anda gunakan di seluruh aplikasi.

2. Buat fungsi bindImage() yang menggunakan ImageView dan String sebagai parameter. Beri anotasi fungsi dengan @BindingAdapter. Anotasi @BindingAdapter memberi tahu data binding bahwa Anda ingin adaptor binding ini dijalankan ketika item XML memiliki atribut imageUrl.

   Impor androidx.databinding.BindingAdapter dan android.widget.ImageView jika diminta.

```
@BindingAdapter( ...value: "imageUrl")
fun bindImage(imgView: ImageView, imgUrl: String?) {

}
```

3. Di dalam fungsi bindImage(), tambahkan blok let {} untuk argumen imgUrl:

```
imgUrl?.let { it: String
}
```

4. Di dalam blok let {}, tambahkan baris yang ditunjukkan di bawah ini untuk mengubah string URL (dari XML) menjadi objek Uri. Impor androidx.core.net.toUri jika diminta. Anda ingin objek Uri final menggunakan skema HTTPS, karena server tempat Anda menarik gambar memerlukan skema itu. Untuk menggunakan skema HTTPS, tambahkan buildUpon.scheme ("https") ke toUri builder. Metode toUri() adalah fungsi ekstensi Kotlin dari pustaka inti Android KTX, jadi itu hanyalah seperti bagian dari kelas String.

```
val imgUri = imgUrl.toUri().buildUpon().scheme( scheme: "https").build()
```

5. Masih di dalam let {}, panggil Glide.with () untuk memuat gambar dari objek Uri ke ImageView. Impor com.bumptech.glide.Glide jika diminta.

```
Glide.with(imgView.context)
        .load(imgUri)
        .into(imgView) ^let
```

Anda mungkin harus menambahkan opsi berikut ke file modul build.gradle Anda:

```
kotlinOptions {
    jvmTarget = JavaVersion.VERSION_1_8.toString()
}
```

Langkah 4: Perbarui tata letak dan fragmen

Meskipun Glide telah memuat gambar, belum ada yang bisa dilihat. Langkah selanjutnya adalah memperbarui tata letak dan fragmen dengan ImageView untuk menampilkan gambar.

1. Buka res/layout/grid_view_item.xml. Ini adalah file sumber daya tata letak yang akan Anda gunakan untuk setiap item di RecyclerView nanti. Anda menggunakannya sementara di sini hanya untuk menampilkan satu gambar.

2. Di atas elemen <ImageView>, tambahkan elemen <data> untuk data binding, dan ikat ke kelas OverviewViewModel:

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

3. Tambahkan atribut app:imageUrl ke elemen ImageView untuk menggunakan adaptor pengikat pemuatan gambar baru:

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

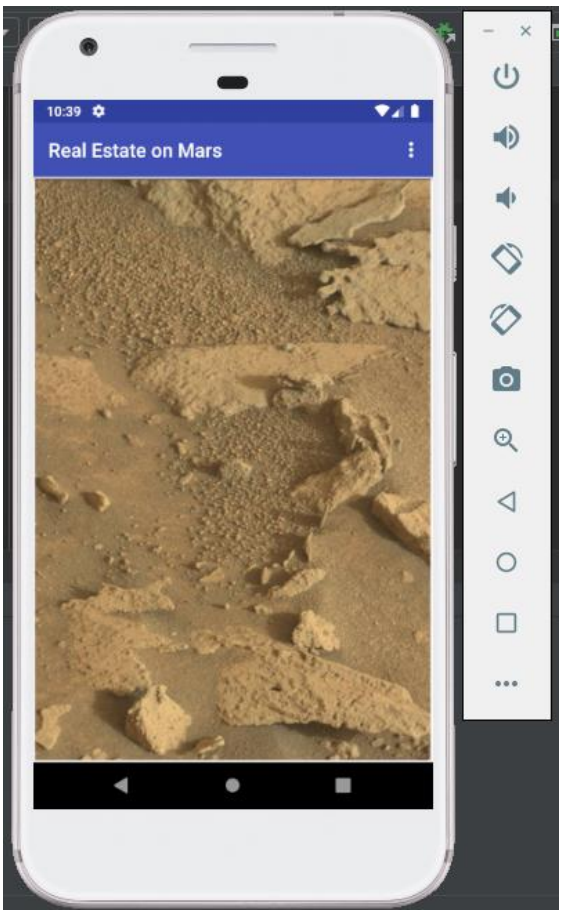
4. Buka overview/OverviewFragment.kt. Dalam metode onCreateView(), beri komentar pada baris yang memekarkan (inflate) kelas FragmentOverviewBinding dan tetapkan ke variabel binding. Ini hanya sementara; Anda akan kembali lagi nanti.

```
//val binding = FragmentOverviewBinding.inflate(inflater)
```

5. Tambahkan baris untuk memekarkan kelas GridViewItemBinding. Impor com.example.android.marsrealestate.databinding.GridViewItemBinding saat diminta.

```
val binding = GridViewItemBinding.inflate(inflater)
```

6. Jalankan aplikasinya. Sekarang Anda akan melihat foto gambar dari MarsProperty pertama dalam daftar hasil.

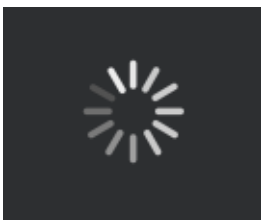


Langkah 5: Tambahkan gambar pemuatan dan kesalahan sederhana

Glide dapat meningkatkan pengalaman pengguna dengan menampilkan gambar placeholder saat memuat gambar dan gambar kesalahan jika pemuatan gagal, misalnya jika gambar hilang atau rusak. Pada langkah ini, Anda menambahkan fungsionalitas tersebut ke adaptor binding dan tata letak.

1. Buka res/drawable/ic_broken_image.xml, dan klik tab Preview di sebelah kanan. Untuk gambar kesalahan, Anda menggunakan ikon gambar rusak yang tersedia di pustaka ikon bawaan. Drawable vektor ini menggunakan atribut android:tint untuk mewarnai ikon dengan warna abu-abu.

2. Buka res/drawable/loading_animation.xml. Drawable ini adalah animasi yang ditentukan dengan tag <animate-rotate>. Animasi memutar gambar drawable, loading_img.xml, di sekitar titik tengah. (Anda tidak melihat animasi di pratinjau.)

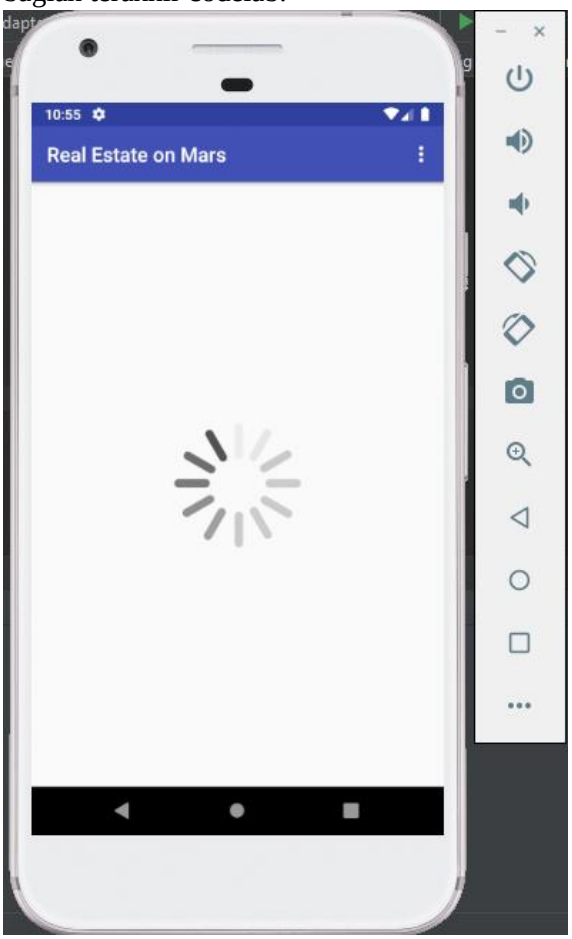


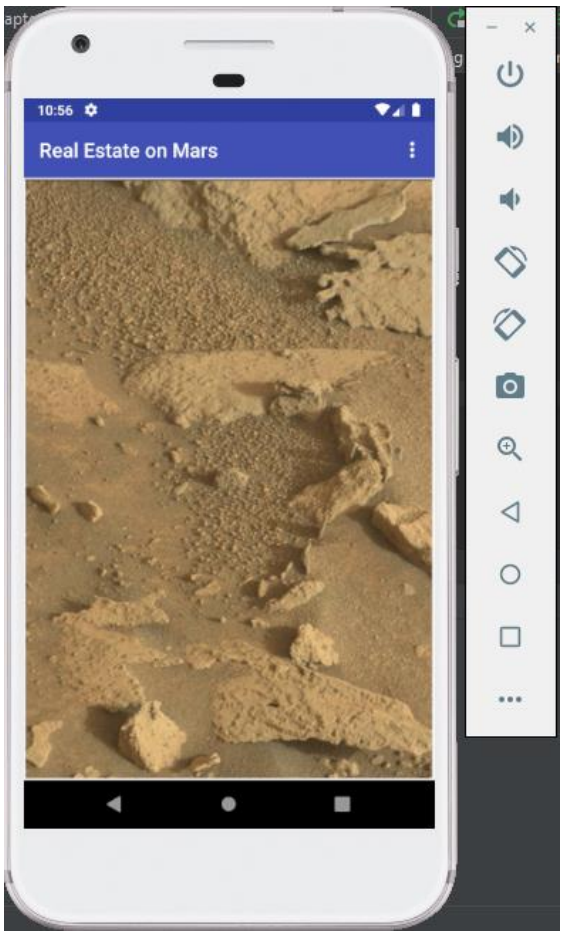
3. Kembali ke file BindingAdapters.kt. Dalam metode bindImage(), perbarui panggilan ke Glide.with() untuk memanggil fungsi apply() antara load() dan into(). Impor com.bumptech.glide.request.RequestOptions jika diminta.

   Kode ini menyetel gambar pemuatan placeholder untuk digunakan saat memuat (drawable loading_animation). Kode juga menyetel gambar untuk digunakan jika pemuatan gambar gagal (drawable broken_image). Metode bindImage() lengkap sekarang terlihat seperti ini:

```
@BindingAdapter( ...value: "imageUrl")
fun bindImage(imgView: ImageView, imgUrl: String?) {
    imgUrl?.let { it: String
        val imgUri =
                imgUrl.toUri().buildUpon().scheme( scheme: "https").build()
        Glide.with(imgView.context)
                .load(imgUri)
                .apply(RequestOptions()
                        .placeholder(R.drawable.loading_animation)
                        .error(R.drawable.ic_broken_image))
                .into(imgView) ^let
    }
}
```

4. Jalankan aplikasinya. Bergantung pada kecepatan koneksi jaringan Anda, Anda mungkin melihat sekilas gambar pemuatan saat Glide mendownload dan menampilkan gambar properti. Tetapi Anda belum melihat ikon gambar rusak, meskipun Anda mematikan jaringan — Anda memperbaikinya di bagian terakhir codelab.





---

4. Tugas: Menampilkan kisi gambar dengan RecyclerView

Aplikasi Anda sekarang memuat informasi properti dari internet. Dengan menggunakan data dari item daftar MarsProperty pertama, Anda telah membuat properti LiveData dalam view model, dan Anda telah menggunakan URL gambar dari data properti tersebut untuk mengisi ImageView. Tapi tujuannya adalah agar aplikasi Anda menampilkan gambar secara grid/kisi, jadi Anda ingin menggunakan RecyclerView dengan GridLayoutManager.

Langkah 1: Perbarui view model

Saat ini model tampilan memiliki _property LiveData yang menyimpan satu objek MarsProperty—yang pertama dalam daftar respons dari layanan web. Pada langkah ini, Anda mengubah LiveData untuk menampung seluruh daftar objek MarsProperty.

1. Buka overview/OverviewViewModel.kt.
2. Ubah variabel privat _property menjadi _properties. Ubah jenisnya menjadi daftar objek MarsProperty.
```
private val _properties = MutableLiveData<List<MarsProperty>>()
```

3. Ganti property eksternal live data dengan properties. Tambahkan daftar ke tipe LiveData di sini juga:
```
val properties: LiveData<List<MarsProperty>>
    get() = _properties
```

4. Gulir ke bawah ke metode getMarsRealEstateProperties() . Di dalam blok try {}, ganti seluruh pengujian yang Anda tambahkan di tugas sebelumnya dengan baris yang ditampilkan di bawah ini. Karena MarsApi.retrofitService.getProperties() mengembalikan daftar objek MarsProperty, Anda bisa menetapkannya ke _properties.value sebagai ganti menguji respons yang berhasil.
```
_properties.value = MarsApi.retrofitService.getProperties()
```

   Seluruh blok try/catch sekarang terlihat seperti ini:
```
try {
    _properties.value = MarsApi.retrofitService.getProperties()
    _response.value = "Success: Mars properties retrieved"
} catch (e: Exception) {
    _response.value = "Failure: ${e.message}"
}
```

Langkah 2: Perbarui tata letak dan fragmen

Langkah selanjutnya adalah mengubah tata letak dan fragmen aplikasi untuk menggunakan tampilan recycler view dan tata letak kisi, bukan tampilan gambar tunggal.

1. Buka res/layout/grid_view_item.xml. Ubah data binding dari OverviewViewModel menjadi MarsProperty, dan ganti nama variabel menjadi "property".
```
<variable
    name="property"
    type="com.example.android.marsrealestate.network.MarsProperty" />
```

2. Di <ImageView>, ubah atribut app:imageUrl untuk merujuk ke URL gambar di objek MarsProperty:
```
app:imageUrl="@{property.imgSrcUrl}"/>
```

3. Buka overview/OverviewFragment.kt. Di onCreateview(), hapus komentar pada baris yang

memekarkan (inflate) FragmentOverviewBinding. Hapus atau komentari baris yang memekarkan GridViewItemBinding. Perubahan ini membatalkan perubahan sementara yang Anda buat di tugas terakhir.

```
val binding = FragmentOverviewBinding.inflate(inflater)
//val binding = GridViewItemBinding.inflate(inflater)
```

4. Buka res/layout/fragment_overview.xml. Hapus seluruh elemen <TextView>.

5. Tambahkan elemen <RecyclerView> ini sebagai gantinya, yang menggunakan GridLayoutManager dan layout grid_view_item untuk satu item:

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/photos_grid"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:padding="6dp"
    android:clipToPadding="false"
    app:layoutManager="androidx.recyclerview.widget.GridLayoutManager"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:spanCount="2"
    tools:itemCount="16"
    tools:listitem="@layout/grid_view_item" />
```

Langkah 3: Tambahkan adaptor kisi foto

Sekarang layout fragment_overview memiliki RecyclerView sedangkan layout grid_view_item memiliki satu ImageView. Pada langkah ini, Anda mengikat data ke RecyclerView melalui adaptor RecyclerView.

1. Buka overview/PhotoGridAdapter.kt.

2. Buat kelas PhotoGridAdapter, dengan parameter konstruktor yang ditunjukkan di bawah ini. Kelas PhotoGridAdapter memperluas ListAdapter, yang konstruktornya memerlukan jenis item daftar, pemegang tampilan, dan implementasi DiffUtil.ItemCallback.

   Impor kelas androidx.recyclerview.widget.ListAdapter dan com.example.android.marsrealestate.network.MarsProperty jika diminta. Dalam langkah- langkah berikut, Anda menerapkan bagian lain yang hilang dari konstruktor ini yang menghasilkan kesalahan.

```
class PhotoGridAdapter : ListAdapter<MarsProperty,
        PhotoGridAdapter.MarsPropertyViewHolder>(DiffCallback) {
```

3. Klik di mana saja di kelas PhotoGridAdapter dan tekan Control + i untuk mengimplementasikan metode ListAdapter, yaitu onCreateViewHolder() dan onBindViewHolder().

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): PhotoGridAdapter.MarsPropertyViewHolder {
    TODO( reason: "Not yet implemented")
}
override fun onBindViewHolder(holder: PhotoGridAdapter.MarsPropertyViewHolder, position: Int) {
    TODO( reason: "Not yet implemented")
}
```

4. Di akhir definisi kelas PhotoGridAdapter, setelah metode yang baru saja Anda tambahkan, tambahkan definisi objek pendamping untuk DiffCallback, seperti yang ditunjukkan di bawah ini.

Impor androidx.recyclerview.widget.DiffUtil jika diminta. Objek DiffCallback memperluas DiffUtil.ItemCallback dengan jenis objek yang ingin Anda bandingkan — MarsProperty.

```
companion object DiffCallback : DiffUtil.ItemCallback<MarsProperty>() {
```

5. Tekan Control + i untuk mengimplementasikan metode komparator untuk objek ini, yaitu areItemsTheSame() dan areContentsTheSame().

```
companion object DiffCallback : DiffUtil.ItemCallback<MarsProperty>() {
    override fun areItemsTheSame(oldItem: MarsProperty, newItem: MarsProperty): Boolean {
        TODO( reason: "Not yet implemented")
    }
    override fun areContentsTheSame(oldItem: MarsProperty, newItem: MarsProperty): Boolean {
        TODO( reason: "Not yet implemented")
    }
}
```

6. Untuk metode areItemsTheSame(), hapus TODO. Gunakan operator persamaan referensial Kotlin (===), yang mengembalikan nilai true jika referensi objek untuk oldItem dan newItem sama.

```
override fun areItemsTheSame(oldItem: MarsProperty, newItem: MarsProperty): Boolean {
    return oldItem === newItem
}
```

7. Untuk areContentsTheSame(), gunakan operator persamaan standar hanya pada ID dari oldItem dan newItem.

```
override fun areContentsTheSame(oldItem: MarsProperty, newItem: MarsProperty): Boolean {
    return oldItem.id == newItem.id
}
```

8. Masih di dalam kelas PhotoGridAdapter, di bawah objek pendamping (companion), tambahkan definisi kelas dalam (inner class) untuk MarsPropertyViewHolder, yang memperluas RecyclerView.ViewHolder.

   Impor androidx.recyclerview.widget.RecyclerView dan com.example.android.marsrealestate.databinding.GridViewItemBinding jika diminta. Anda memerlukan variabel GridViewItemBinding untuk mengikat MarsProperty ke layout, jadi teruskan variabel tersebut ke MarsPropertyViewHolder. Karena kelas ViewHolder dasar memerlukan tampilan dalam konstruktornya, Anda meneruskannya tampilan root binding.

```
class MarsPropertyViewHolder(private var binding: GridViewItemBinding):
        RecyclerView.ViewHolder(binding.root) {
}
```

9. Di MarsPropertyViewHolder, buat metode bind() yang menggunakan objek MarsProperty sebagai argumen dan setel binding.property ke objek itu. Panggil executePendingBindings() setelah menyetel properti, yang menyebabkan pembaruan segera dijalankan.

```
fun bind(marsProperty: MarsProperty) {
    binding.property = marsProperty
    binding.executePendingBindings()
}
```

10. Di onCreateViewHolder(), hapus TODO dan tambahkan baris yang ditunjukkan di bawah ini. Impor android.view.LayoutInflater jika diminta.

```
return MarsPropertyViewHolder(GridViewItemBinding.inflate(
        LayoutInflater.from(parent.context)))
```

11. Dalam metode onBindViewHolder(), hapus TODO dan tambahkan baris yang ditunjukkan di bawah ini. Di sini Anda memanggil getItem() untuk mendapatkan objek MarsProperty yang terkait dengan posisi RecyclerView saat ini, lalu meneruskan properti tersebut ke metode bind() di MarsPropertyViewHolder.

```
val marsProperty = getItem(position)
holder.bind(marsProperty)
```

Langkah 4: Tambahkan adaptor pengikat dan sambungkan bagian-bagiannya

Terakhir, gunakan BindingAdapter untuk menginisialisasi PhotoGridAdapter dengan daftar objek MarsProperty. Menggunakan BindingAdapter untuk menyetel data RecyclerView menyebabkan data binding mengamati LiveData secara otomatis untuk daftar objek MarsProperty. Kemudian adaptor pengikat dipanggil secara otomatis saat daftar MarsProperty berubah.

1. Buka BindingAdapters.kt.

2. Di akhir file, tambahkan metode bindRecyclerView() yang menggunakan RecyclerView dan daftar objek MarsProperty sebagai argumen. Beri anotasi metode tersebut dengan @BindingAdapter.

   Impor androidx.recyclerview.widget.RecyclerView dan com.example.android.marsrealestate.network.MarsProperty jika diminta.

```
@BindingAdapter( ...value: "listData")
fun bindRecyclerView(recyclerView: RecyclerView,
                     data: List<MarsProperty>?) {
}
```

3. Di dalam fungsi bindRecyclerView(), transmisikan recyclerView.adapter ke PhotoGridAdapter, dan panggil adapter.submitList() dengan datanya. Ini memberi tahu RecyclerView saat daftar baru tersedia.

   Impor com.example.android.marsrealestate.overview.PhotoGridAdapter jika diminta.

```
val adapter = recyclerView.adapter as PhotoGridAdapter
adapter.submitList(data)
```

4. Buka res/layout/fragment_overview.xml. Tambahkan atribut app:listData ke elemen RecyclerView dan setel ke viewmodel.properties menggunakan data binding.

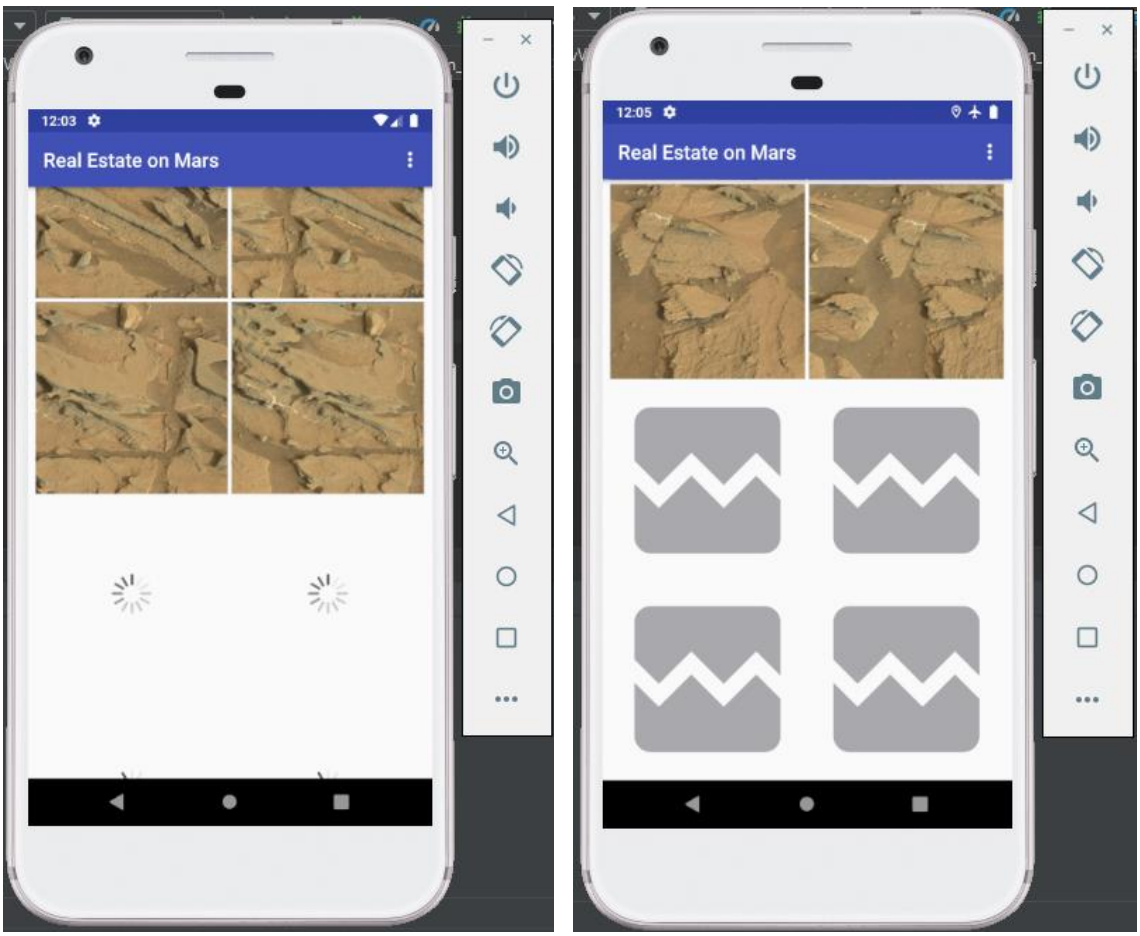
```
app:listData="@{viewModel.properties}"/>
```

5. Buka overview/OverviewFragment.kt. Di onCreateView(), tepat sebelum panggilan ke setHasOptionsMenu(), inisialisasi adaptor RecyclerView di binding.photosGrid ke objek PhotoGridAdapter baru.

```
binding.photosGrid.adapter = PhotoGridAdapter()
```

6. Jalankan aplikasinya. Anda akan melihat kisi gambar MarsProperty. Saat Anda menggulir untuk melihat gambar baru, aplikasi menampilkan ikon proses pemuatan sebelum menampilkan gambar itu sendiri. Jika Anda mengaktifkan mode pesawat, gambar yang belum dimuat muncul sebagai ikon gambar rusak.

(screenshot aplikasi di halaman selanjutnya)



---

5. Tugas: Menambahkan penanganan error di RecyclerView

Aplikasi MarsRealEstate menampilkan ikon gambar rusak ketika gambar tidak dapat diambil. Tetapi ketika tidak ada jaringan, aplikasi menampilkan layar kosong. Ini bukan pengalaman pengguna yang luar biasa. Dalam tugas ini, Anda menambahkan penanganan kesalahan dasar, untuk memberi pengguna gambaran yang lebih baik tentang apa yang terjadi. Jika internet tidak tersedia, aplikasi akan menampilkan ikon kesalahan koneksi. Saat aplikasi mengambil daftar MarsProperty, aplikasi akan menampilkan animasi pemuatan.

Langkah 1: Tambahkan status ke view model

Untuk memulai, Anda membuat LiveData dalam view model untuk mewakili status permintaan web. Ada tiga kondisi untuk dipertimbangkan — memuat, sukses, dan gagal. Status pemuatan terjadi saat Anda menunggu data dalam panggilan ke await().

1. Buka overview/OverviewViewModel.kt. Di bagian atas file (setelah impor, sebelum definisi kelas), tambahkan enum untuk mewakili semua status yang tersedia:

```
enum class MarsApiStatus { LOADING, ERROR, DONE }
```

2. Ganti nama definisi live data _response internal dan eksternal di seluruh kelas OverviewViewModel menjadi _status. Karena sebelumnya Anda telah menambahkan dukungan untuk _properties LiveData di codelab ini, respons layanan web lengkap tidak digunakan. Anda memerlukan LiveData di sini untuk melacak status saat ini, jadi Anda bisa mengganti nama variabel yang ada.

Juga, ubah tipe dari String ke MarsApiStatus.

```
private val _status = MutableLiveData<MarsApiStatus>()

val status: LiveData<MarsApiStatus>
    get() = _status
```

3. Gulir ke bawah ke metode getMarsRealEstateProperties() dan perbarui _response ke _status di sini juga. Ubah string "Success" ke status MarsApiStatus.DONE, dan string "Failure" menjadi MarsApiStatus.ERROR.

4. Setel status ke MarsApiStatus.LOADING sebelum blok try {}. Ini adalah status awal saat coroutine berjalan dan Anda menunggu data. Blok try / catch {} lengkap sekarang terlihat seperti ini:

```
_status.value = MarsApiStatus.LOADING
try {
    _properties.value = MarsApi.retrofitService.getProperties()
    _status.value = MarsApiStatus.DONE
} catch (e: Exception) {
    _status.value = MarsApiStatus.ERROR
}
```

5. Setelah status error di blok catch {}, setel _properties LiveData ke daftar kosong. Ini menghapus RecyclerView.

```
} catch (e: Exception) {
    _status.value = MarsApiStatus.ERROR
    _properties.value = ArrayList()
}
```

Langkah 2: Tambahkan adaptor pengikat untuk status ImageView

Sekarang Anda memiliki status dalam model tampilan, tetapi itu hanya sekumpulan status. Bagaimana Anda membuatnya muncul di aplikasi itu sendiri? Pada langkah ini, Anda menggunakan ImageView, terhubung ke data binding, untuk menampilkan ikon untuk status pemuatan dan kesalahan. Saat aplikasi dalam status pemuatan atau status kesalahan, ImageView harus terlihat. Saat aplikasi selesai memuat, ImageView seharusnya tidak terlihat.

1. Buka BindingAdapters.kt. Tambahkan adaptor binding baru yang disebut bindStatus() yang menggunakan nilai ImageView dan MarsApiStatus sebagai argumen. Impor com.example.android.marsrealestate.overview.MarsApiStatus jika diminta.

```
@BindingAdapter( ...value: "marsApiStatus")
fun bindStatus(statusImageView: ImageView,
               status: MarsApiStatus?) {
}
```

2. Tambahkan when {} di dalam metode bindStatus() untuk beralih di antara status yang berbeda.

```
when (status) {

}
```

3. Di dalam when {}, tambahkan kasus untuk status pemuatan (MarsApiStatus.LOADING). Untuk status ini, setel ImageView menjadi terlihat, dan tetapkan animasi pemuatan. Ini adalah drawable animasi yang sama yang Anda gunakan untuk Glide di tugas sebelumnya. Impor android.view.View saat diminta.

```
MarsApiStatus.LOADING -> {
    statusImageView.visibility = View.VISIBLE
    statusImageView.setImageResource(R.drawable.loading_animation)
}
```

4. Tambahkan kasus untuk status kesalahan, yaitu MarsApiStatus.ERROR. Serupa dengan apa yang Anda lakukan untuk status LOADING, setel status ImageView menjadi terlihat dan gunakan kembali drawable kesalahan koneksi.

```
MarsApiStatus.ERROR -> {
    statusImageView.visibility = View.VISIBLE
    statusImageView.setImageResource(R.drawable.ic_connection_error)
}
```

5. Tambahkan kasus untuk status selesai, yaitu MarsApiStatus.DONE. Di sini Anda mendapatkan respons yang berhasil, jadi matikan visibilitas status ImageView untuk menyembunyikannya.

```
MarsApiStatus.DONE -> {
    statusImageView.visibility = View.GONE
}
```

Langkah 3: Tambahkan status ImageView ke layout

1. Buka res/layout/fragment_overview.xml. Di bawah elemen RecyclerView, di dalam ConstraintLayout, tambahkan ImageView yang ditunjukkan di bawah ini.

   ImageView ini memiliki batasan yang sama dengan RecyclerView. Namun, lebar dan tinggi menggunakan wrap_content untuk memusatkan gambar daripada meregangkan gambar untuk mengisi tampilan. Perhatikan juga atribut app:marsApiStatus, yang memiliki panggilan tampilan BindingAdapter Anda saat properti status dalam model tampilan berubah.

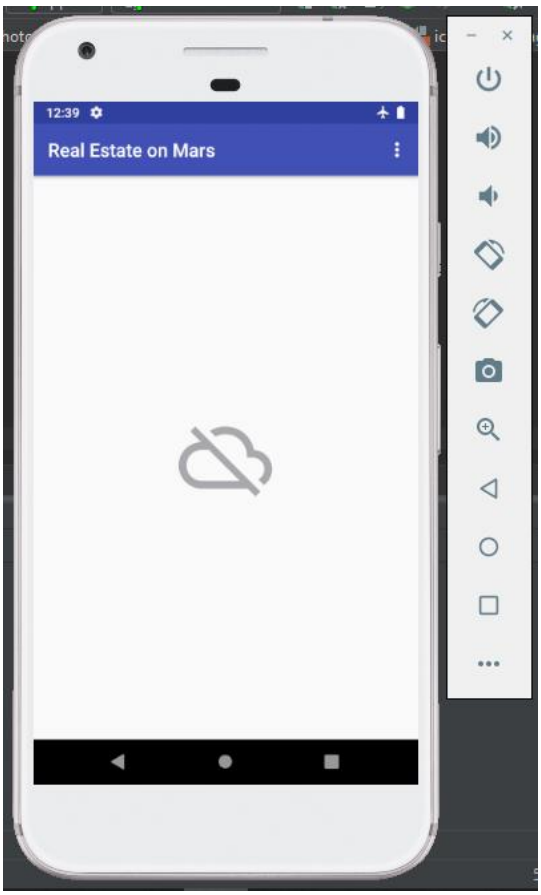
```
<ImageView
    android:id="@+id/status_image"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:marsApiStatus="@{viewModel.status}" />
```

2. Aktifkan mode pesawat di emulator atau perangkat Anda untuk menyimulasikan koneksi jaringan yang hilang. Kompilasi dan jalankan aplikasi, dan perhatikan bahwa gambar kesalahan muncul:

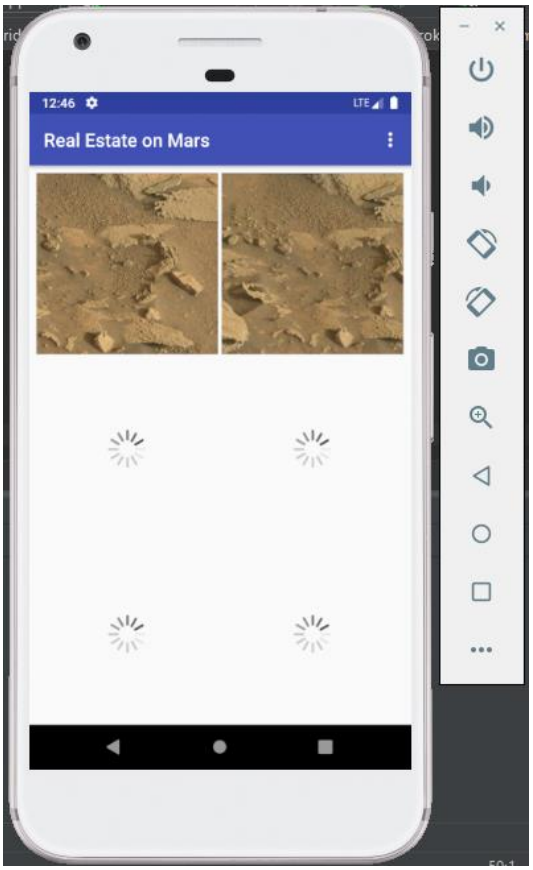
(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)

(screenshot aplikasi di halaman selanjutnya)



3. Ketuk tombol Kembali untuk menutup aplikasi, dan matikan mode pesawat. Gunakan layar terbaru untuk mengembalikan aplikasi. Bergantung pada kecepatan koneksi jaringan Anda, Anda mungkin melihat putaran pemuatan yang sangat singkat saat aplikasi menanyakan layanan web sebelum gambar mulai dimuat.



TUGAS

Buat aplikasi baru dengan mengembangkan project di atas

Di bagian tugas ini, saya mengembangkan project di atas dengan menambahkan gambar “for sale” pada real estate Mars yang dijual

Step 1: Update MarsProperty untuk menunjukkan type (tipe)

Pada langkah ini, kita menambahkan beberapa logika ke kelas MarsProperty untuk menunjukkan apakah sebuah properti “rent”(disewakan) atau “buy” (tidak disewakan (dijual))

1. Buka network/MarsProperty.kt. Tambahkan body ke definisi class MarsProperty, dan tambahkan method getter untuk isRental yang mengembalikan nilai true jika objek bertipe "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"
}
```

Step 2: Update grid item layout

1. Buka res/layout/grid_view_item.xml. file ini adalah file layout untuk setiap cell dalam grid layout untuk RecyclerView. Saat ini file hanya berisi elemen <ImageView> untuk gambar properti.

2. Di dalam elemen <data>, tambahkan elemen <import> untuk kelas View. Kita menggunakan impor saat kita ingin menggunakan komponen kelas di dalam ekspresi data binding dalam file layout. Kita akan menggunakan konstanta View.GONE dan View.VISIBLE sehingga kita memerlukan akses ke kelas View.

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

3. Kelilingi seluruh ImageView dengan FrameLayout, untuk membuat drawable logo dollar ditumpuk di atas property image.

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="170dp">
<ImageView
    android:id="@+id/mars_image"
    android:layout_width="match_parent"
    android:layout_height="170dp"
    android:scaleType="centerCrop"
    android:adjustViewBounds="true"
    android:padding="2dp"
    tools:src="@tools:sample/backgrounds/scenic"
    app:imageUrl="@{property.imgSrcUrl}"/>
</FrameLayout>
```

4. Untuk ImageView, ubah atribut android:layout_height menjadi match_parent, untuk mengisi FrameLayout.

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

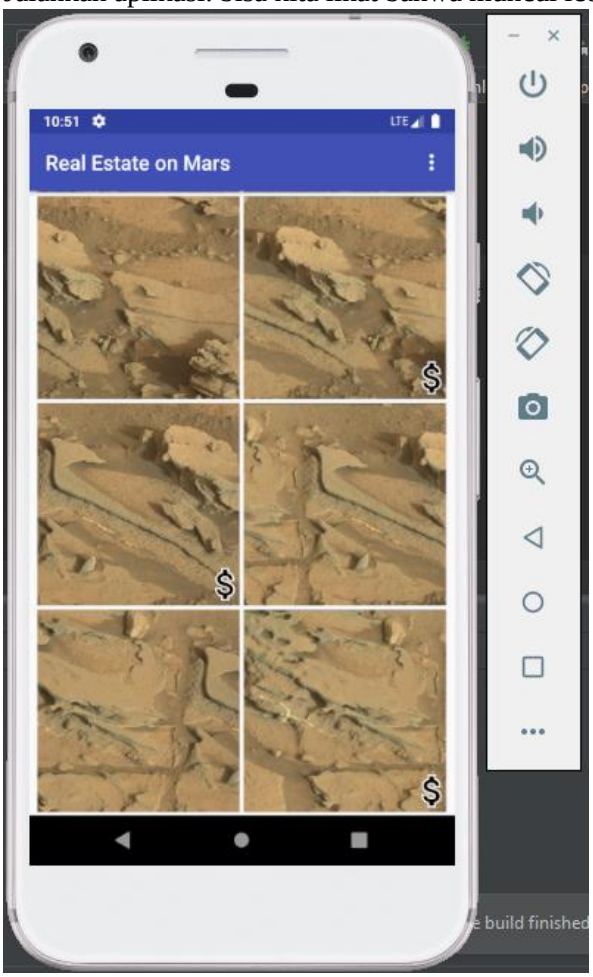
5. Tambahkan elemen <ImageView> kedua tepat di bawah yang pertama, di dalam FrameLayout. ImageView ini muncul di sudut kanan bawah grid item. Elemen ini menggunakan drawable yang didefinisikan dalam res/drawable/ic_for_sale_outline.xml untuk icon logo dollar.

```
    <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"/>
</FrameLayout>
```

6. Tambahkan atribut android:visibility ke ImageView 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}"/>
```

7. Jalankan aplikasi. bisa kita lihat bahwa muncul icon dollar pada properti yang bukan rental.



KESIMPULAN

Pada laporan ini, saya berhasil memenuhi tujuan dari praktikum ke-13 ini yaitu membuat aplikasi memuat dan menampilkan gambar dari internet menggunakan Retrofit, Moshi dan Glide. Saya mengerjakan banyak hal di praktikum ini seperti mengubah aplikasi MarsRealEstate untuk mendapatkan URL gambar dari data properti Mars, menggunakan RecyclerView untuk menampilkan gambar secara grid properti Mars, dan menambahkan status dan penanganan error ke RecyclerView. Di bagian tugas saya juga mengembangkan project mars tersebut dengan memberikan tanda dolar pada setiap item real estate.

Terima Kasih