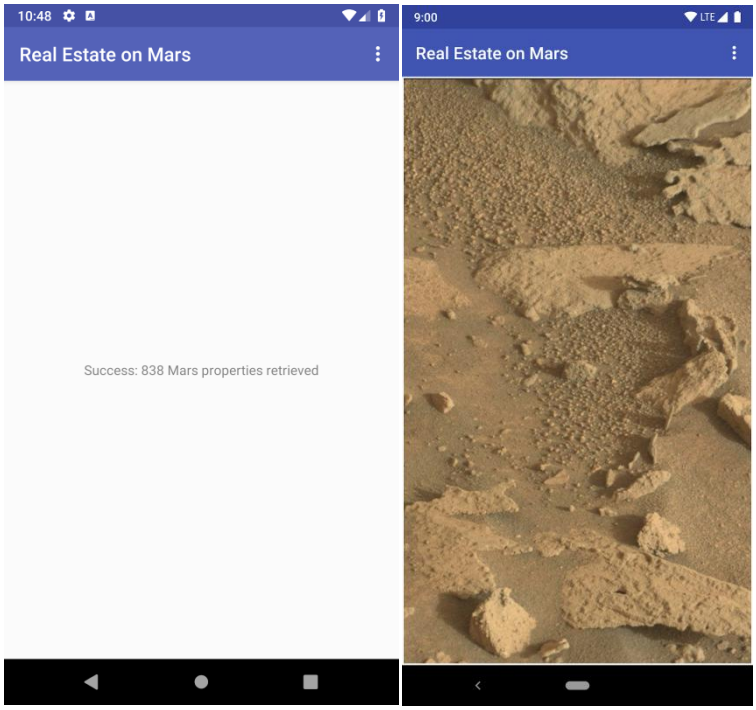
B. Zadanie: wyświetl obraz internetowy

Wyświetlanie zdjęcia z internetowego adresu URL może wydawać się proste, ale jest sporo inżynierii, aby działało dobrze. Obraz musi zostać pobrany, zbuforowany i zdekodowany ze skompresowanego formatu na obraz, którego może używać Android. Obraz powinien być buforowany w pamięci w pamięci podręcznej (in-memory cache), w pamięci masowej (storage-based cache), lub w obu tych miejscach. Wszystko to musi się zdarzyć w wątkach tła o niskim priorytecie, aby interfejs użytkownika pozostawał responsywny. Ponadto, aby uzyskać najlepszą wydajność sieci i procesora, możesz pobrać i zdekodować więcej niż jeden obraz na raz. Nauczenie się, jak skutecznie ładować obrazy z sieci, może być osobnym ćwiczeniem.

Na szczęście możesz używać biblioteki Glide do pobierania, buforowania, dekodowania i buforowania zdjęć. Glide pozostawia o wiele mniej pracy niż gdybyś musiał to wszystko zrobić od zera.
Glide potrzebuje w zasadzie dwóch rzeczy:

- Adresu URL obrazu, który chcesz załadować i pokazać.
- Obiektu `ImageView` do wyświetlenia tego obrazu.

W tym zadaniu nauczysz się, jak używać Glide, aby wyświetlić pojedynczy obraz z serwisu internetowego. Oto zrzuty ekranu przed i po:



Step 1: Dodaj zależność Glide

1. Otwórz **build.gradle (Module: app)**.
2. W sekcji `dependencies` dodaj ten wiersz do biblioteki Glide:

```
implementation "com.github.bumptech.glide:glide:$version_glide"
```

Zauważ, że numer wersji jest już zdefiniowany osobno w pliku Gradle projektu.

5. Kliknij opcję Synchronizuj teraz, aby odbudować projekt z nową zależnością.

Step 2: Zaktualizuj view model

Następnie zaktualizuj klasę `OverviewViewModel` , aby uwzględnić dane live data dla pojedynczej właściwości Marsa.

1. Otwórz `overview/OverviewViewModel.kt`. Tuż poniżej `LiveData` dla `_response`, dodaj zarówno dane wewnętrzne (mutable) jak i zewnętrzne (immutable) live data dla pojedynczego obiektu `MarsProperty`.

   Na żądanie zaimportuj klasę `MarsProperty` (`com.example.android.marsrealestate.network.MarsProperty`)

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

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

2. W metodzie `getMarsRealEstateProperties()` znajdź wiersz w bloku `try/catch {}` , który ustawia `_response.value` na liczbę właściwości. Dodaj test pokazany poniżej. Jeśli obiekty MarsProperty są dostępne, ten test ustawia wartość `_property` `LiveData` na pierwszą właściwość na `listResult`.

```
if (listResult.size > 0) {
    _property.value = listResult[0]
}
```

Cały blok `try/catch {}` wygląda teraz tak:

```
try {
   var listResult = getPropertiesDeferred.await()
   _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. Otwórz plik `res/layout/fragment_overview.xml` W elemencie `<TextView>` zmień `android:text` , aby powiązać ze składnikiem `imgSrcUrl` właściwości `property` `LiveData`:

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

4. Uruchom aplikację. `TextView` wyświetla tylko adres URL obrazu w pierwszej właściwości Marsa. Do tej pory skonfigurowałeś model widoku i dane na żywo dla tego adresu URL.

Step 3: Utwórz adapter wiązania i wywołaj Glide

Teraz masz adres URL obrazu do wyświetlenia i czas rozpocząć pracę z Glide, aby załadować ten obraz. W tym kroku używasz adaptera wiązania, aby pobrać adres URL z atrybutu XML powiązanego z ImageView, i użyć Glide, aby załadować obraz. Adaptery wiązania to metody rozszerzeń, które znajdują się między widokiem a powiązanymi danymi, aby zapewnić niestandardowe zachowanie w przypadku zmiany danych. W takim przypadku niestandardowe zachowanie polega na wywołaniu Glide w celu załadowania obrazu z adresu URL do `ImageView`.

1. Otwórz `BindingAdapters.kt`. Ten plik będzie zawierał łączniki, których używasz w aplikacji.
2. Utwórz funkcję `bindImage()`, która przyjmuje `ImageView` i `String` jako parametry. Opisz funkcję za pomocą `@BindingAdapter`. Adnotacja `@BindingAdapter` mówi powiązaniu danych, że ten adapter wiązania ma być wykonywany, gdy element XML ma atrybut `imageUrl`.

   Na żądanie zaimportuj `androidx.databinding.BindingAdapter` and `android.widget.ImageView`.

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

}
```

3. Wewnątrz funkcji `bindImage()` dodaj blok `let {}` dla argumentu`imgUrl`:

```
imgUrl?.let {
}
```

4. Wewnątrz bloku `let {}` dodaj wiersz pokazany poniżej, aby przekonwertować ciąg adresu URL (z XML) na obiekt Uri. Na żądanie zaimportuj `androidx.core.net.toUri`.

   Chcesz, aby końcowy obiekt Uri korzystał ze schematu HTTPS, ponieważ serwer, z którego pobierasz obrazy, wymaga tego schematu. Aby użyć schematu HTTPS, dołącz `buildUpon.scheme("https")` do konstruktora `toUri` Metoda `toUri()` jest funkcją rozszerzenia Kotlin z biblioteki podstawowej Android KTX core library, więc wygląda na to, że jest częścią klasy `String`.

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

5. Nadal w środku `let {}`, wywołaj `Glide.with()`, aby załadować obraz z obiektu `Uri` do `ImageView`. Na żądanie zaimportuj `com.bumptech.glide.Glide`.

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

Step 4: Zaktualizuj układ i fragmenty

Mimo że Glide załadował obraz, nie ma jeszcze nic do zobaczenia. Następnym krokiem jest aktualizacja układu i fragmentów za pomocą ImageView, aby wyświetlić obraz..

1. Otwórz plik `res/layout/gridview_item.xml`. Jest to plik zasobów układu, którego będziesz używać dla każdego elementu w `RecyclerView` później. Używasz go tymczasowo tutaj, aby wyświetlić tylko jeden obraz.
2. Nad elementem`<ImageView>` dodaj element`<data>` dla powiązania danych i połącz go z klasą `OverviewViewModel`:

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

3. Dodaj atrybut `app:imageUrl` do elementu `ImageView` , aby użyć nowego adaptera wiązania ładowania obrazu:

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

4. Otwórz `overview/OverviewFragment.kt`. W metodzie `onCreateView()` skomentuj wiersz, który napełnia klasę `FragmentOverviewBinding` i przypisuje ją do zmiennej powiązania. To jest tylko tymczasowe; wrócisz do tego później.

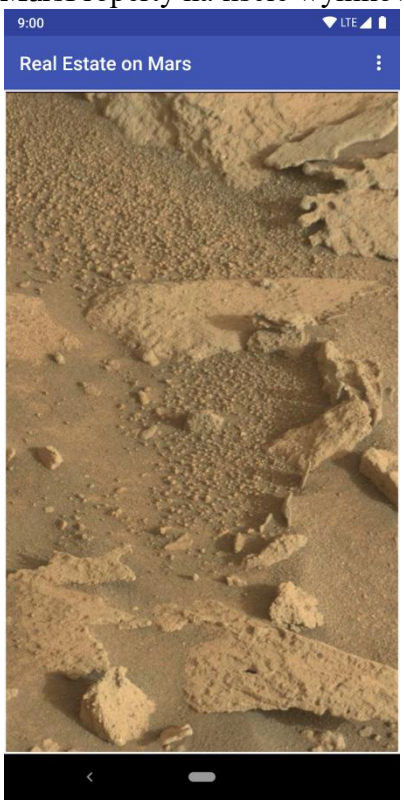
```
//val binding = FragmentOverviewBinding.inflate(inflater)
```

5. Zamiast tego dodaj linię, aby napompować klasę `GridViewItemBinding` Zaimportuj. `com.example.android.marsrealestate.databinding.GridViewItemBinding` na żądanie.

Note: ta zmiana może powodować błędy wiązania danych w Android Studio. Aby rozwiązać te błędy, może być konieczne wyczyszczenie i przebudowanie aplikacji.

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

6. Uruchom aplikację. Teraz powinieneś zobaczyć zdjęcie obrazu z pierwszego MarsProperty na liście wyników.

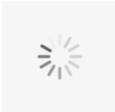


Step 5: Dodaj proste obrazy ładowania i błędów

może poprawić wrażenia użytkownika, wyświetlając obraz zastępczy podczas ładowania obrazu i obraz błędu, jeśli ładowanie się nie powiedzie, na przykład jeśli obraz jest brakujący lub uszkodzony. W tym kroku dodajesz tę funkcjonalność do adaptera wiązania i do układu.

1. Otwórz plik `res/drawable/ic_broken_image.xml`, i kliknij kartę Podgląd po prawej stronie. W przypadku obrazu błędu używasz ikony uszkodzonego obrazu, która jest dostępna we wbudowanej bibliotece ikon. Ten vector drawable używa atrybutu `android:tint` do pokolorowania ikony na szaro.

2. Otwórz plik `res/drawable/loading_animation.xml`. drawable animation to animacja jest zdefiniowana za pomocą znacznika`<animate-rotate>` Animacja obraca obraz, `loading_img.xml`, wokół punktu środkowego. (W podglądzie nie widać animacji).



3. Wróć do pliku `BindingAdapters.kt` W metodzie `bindImage()` zaktualizuj wywołanie do `Glide.with()` aby wywołać funkcję `apply()` między `load()` a `into()`.Na żądanie importuj `com.bumptech.glide.request.RequestOptions`

   Ten kod ustawia zastępczy obraz ładujący, który będzie używany podczas ładowania (`loading_animation` drawable). Kod ustawia również obraz, który będzie używany, jeśli ładowanie obrazu nie powiedzie się (`broken_image` drawable). Pełna metoda`bindImage()` wygląda teraz następująco:

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

4. Uruchom aplikację. W zależności od szybkości połączenia sieciowego możesz na krótko zobaczyć ładowany obraz, gdy Glide pobiera i wyświetla obraz właściwości. Ale nie zobaczysz jeszcze ikony zepsutego obrazu, nawet jeśli wyłączysz sieć - naprawisz to w ostatniej części kodu.

4. Zadanie: Wyświetl siatkę obrazów za pomocą RecyclerView

Twoja aplikacja ładuje teraz informacje o nieruchomościach z Internetu. Używając danych z pierwszego elementu listy MarsProperty, utworzyłeś właściwość LiveData w modelu widoku i użyłeś adresu URL obrazu z tych danych właściwości do wypełnienia ImageView. Ale celem jest, aby aplikacja wyświetlała siatkę obrazów, więc chcesz użyć `RecyclerView` z `GridLayoutManager`.

Step 1: Zaktualizuj model widoku view model

W tej chwili model widoku ma właściwość `_property` `LiveData` , która zawiera jeden obiekt `MarsProperty` pierwszy na liście odpowiedzi z usługi internetowej. W tym kroku zmienisz tę `LiveData` aby przechowywać całą listę obiektów `MarsProperty`.

1. Otwórz `overview/OverviewViewModel.kt`.
2. Zmień prywatną zmienną `_property` na `_properties`. Zmień typ na listę obiektów `MarsProperty`.

```
private val _properties = MutableLiveData<List<MarsProperty>>()
```

3. Zastąp `property` live data przez `properties`. Dodaj tutaj także listę do typu `LiveData`:

```
 val properties: LiveData<List<MarsProperty>>
        get() = _properties
```

4. Przewiń w dół do metody `getMarsRealEstateProperties()` W bloku `try {}` zamień cały test dodany w poprzednim zadaniu na wiersz pokazany poniżej. Ponieważ zmienna `listResult` przechowuje listę obiektów `MarsProperty` możesz po prostu przypisać ją do `_properties.value` zamiast testować pod kątem pomyślnej odpowiedzi.

```
_properties.value = listResult
```

Cały blok `try/catch` wygląda teraz tak:

```
try {
   var listResult = getPropertiesDeferred.await()
   _response.value = "Success: ${listResult.size} Mars properties
retrieved"
   _properties.value = listResult
} catch (e: Exception) {
   _response.value = "Failure: ${e.message}"
}
```

Step 2: Zaktualizuj układy i fragmenty

Następnym krokiem jest zmiana układu aplikacji i jej fragmentów, aby używać widoku recycler view i grid layout, zamiast widoku pojedynczego obrazu.

1. Otwórz plik `res/layout/gridview_item.xml`. Zmień powiązanie danych z `OverviewViewModel` na `MarsProperty`, i zmień nazwę zmiennej na `"property"`.

```
<variable
   name="property"
   type="com.example.android.marsrealestate.network.MarsProperty" />
```

2. W `<ImageView>`, zmień atrybut `app:imageUrl` aby odwoływał się do adresu URL obrazu w obiekcie `MarsProperty`:

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

3. Otwórz `overview/OverviewFragment.kt`. W `onCreateview()`,odkomentuj linię, która napełnia `FragmentOverviewBinding`. Usuń lub skomentuj linię, która napełnia `GridViewBinding`. Te zmiany cofają tymczasowe zmiany wprowadzone w ostatnim zadaniu.

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

4. Otwórz `res/layout/fragment_overview.xml`. Usuń cały element`<TextView>`.
5. Zamiast tego dodaj ten element `<RecyclerView>` który używa `GridLayoutManager` i układu `grid_view_item` dla pojedynczego elementu:

```
<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" />
```

Step 3: Dodaj adapter photo grid adapter

Teraz układ `fragment_overview` ma `RecyclerView` odczas gdy układ `grid_view_item` ma jeden `ImageView`. W tym kroku dane są powiązane z `RecyclerView` przez adapter `RecyclerView`.

1. Otwórz `overview/PhotoGridAdapter.kt`.
2. Utwórz klasę `PhotoGridAdapter` z parametrami konstruktora pokazanymi poniżej. Klasa `PhotoGridAdapter` rozszerza `ListAdapter`, , którego konstruktor potrzebuje typu elementu listy, uchwytu widoku i implementacji `DiffUtil.ItemCallback`.

   Na żądanie zaimportuj klasy `androidx.recyclerview.widget.ListAdapter` i `com.example.android.marsrealestate.network.MarsProperty` W poniższych krokach zaimplementujesz pozostałe brakujące części tego konstruktora, które powodują błędy.

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

}
```

3. Kliknij w dowolnym miejscu w klasie `PhotoGridAdapter` i naciśnij `Control+i` , aby zaimplementować metody `ListAdapter` , którymi są `onCreateViewHolder()` and `onBindViewHolder()`.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
PhotoGridAdapter.MarsPropertyViewHolder {
   TODO("not implemented")
}
```

```
override fun onBindViewHolder(holder:
PhotoGridAdapter.MarsPropertyViewHolder, position: Int) {
   TODO("not implemented")
}
```

4. Na końcu definicji klasy `PhotoGridAdapter` , po właśnie dodanych metodach, dodaj definicję obiektu companion dla `DiffCallback`, jak pokazano poniżej.

   Zaimportuj `androidx.recyclerview.widget.DiffUtil`

   Obiekt `DiffCallback` rozszerza `DiffUtil.ItemCallback` o typ obiektu, który chcesz porównać—`MarsProperty`.

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

5. Naciśnij `Control+i` , aby zaimplementować metody porównawcze dla tego obiektu, którymi są `areItemsTheSame()` i `areContentsTheSame()`.

```
override fun areItemsTheSame(oldItem: MarsProperty, newItem: MarsProperty):
Boolean {
   TODO("not implemented")
}

override fun areContentsTheSame(oldItem: MarsProperty, newItem:
MarsProperty): Boolean {
   TODO("not implemented") }
```

6. W przypadku metody `areItemsTheSame()` usuń TODO. Użyj referencyjnego operatora równości Kotlina (===), który zwraca true, jeśli odwołania do obiektów dla `oldItem` i `newItem` są takie same.

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

7. W przypadku `areContentsTheSame()`,użyj standardowego operatora równości tylko dla identyfikatora `oldItem` i `newItem`.

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

8. Nadal wewnątrz klasy `PhotoGridAdapter` , poniżej obiektu towarzyszącego, dodaj definicję klasy wewnętrznej dla `MarsPropertyViewHolder`, , która rozszerza`RecyclerView.ViewHolder`.

   Na żądanie importuj `androidx.recyclerview.widget.RecyclerView` i `com.example.android.marsrealestate.databinding.GridViewItemBinding`.

   Potrzebujesz zmiennej `GridViewItemBinding` do powiązania `MarsProperty` z układem, więc przekaż zmienną do `MarsPropertyViewHolder`. Ponieważ

podstawowa klasa `ViewHolder` wymaga widoku w swoim konstruktorze, przekazujesz mu wiążący widok główny.

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

}
```

9. W `MarsPropertyViewHolder`, utwórz metodę `bind()` która pobiera obiekt `MarsProperty` jako argument i ustawia `binding.property` na ten obiekt. Wywołaj `executePendingBindings()` po ustawieniu właściwości, co spowoduje natychmiastowe wykonanie aktualizacji.

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

Note: ta zmiana może powodować błędy wiązania danych w Android Studio. Aby rozwiązać te błędy, może być konieczne wyczyszczenie i przebudowanie aplikacji.

10. W `onCreateViewHolder()`,usuń TODO i dodaj linię pokazaną poniżej. Na żądanie zaimportuj `android.view.LayoutInflater`.

    Metoda `onCreateViewHolder()` musi zwrócić nowy `MarsPropertyViewHolder`, , utworzony przez nadmuchanie `GridViewItemBinding` i użycie `LayoutInflater` z nadrzędnego kontekstu `ViewGroup`.

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

11. W metodzie `onBindViewHolder()` usuń TODO i dodaj linie pokazane poniżej. W tym przypadku wywołujemy metodę `getItem()` aby uzyskać obiekt `MarsProperty` object powiązany z bieżącą pozycją `RecyclerView` , a następnie przekazać tę właściwość do metody `bind()` w `MarsPropertyViewHolder`.

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

Step 4: Dodaj binding adapter i połącz części

Na koniec użyj `BindingAdapter` , aby zainicjować `PhotoGridAdapter` z listą obiektów `MarsProperty` Użycie `BindingAdapter` do ustawienia danych `RecyclerView` powoduje, że powiązanie danych automatycznie obserwuje `LiveData` dla listy obiektów `MarsProperty` Następnie adapter wiązania jest wywoływany automatycznie, gdy zmienia się lista `MarsProperty`.

1. Otwórz `BindingAdapters.kt`.
2. Na końcu pliku dodaj metodę `bindRecyclerView()` która przyjmuje jako argument `RecyclerView` i listę obiektów `MarsProperty` Opisz tę metodę za pomocą

`@BindingAdapter`.

Na żądanie zaimportuj `androidx.recyclerview.widget.RecyclerView` i `com.example.android.marsrealestate.network.MarsProperty`.

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

3. Wewnątrz funkcji `bindRecyclerView()` rzutuj (cast) `recyclerView.adapter` n `PhotoGridAdapter`, wywołaj `adapter.submitList()` z danymi. To informuje `RecyclerView` , kiedy dostępna jest nowa lista.

Na żądanie zaimportuj
`com.example.android.marsrealestate.overview.PhotoGridAdapter`.

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

4. Otwórz `res/layout/fragment_overview.xml`. Dodaj atrybut `app:listData` do elementu `RecyclerView` i ustaw go na `viewmodel.properties` za pomocą powiązania danych.

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

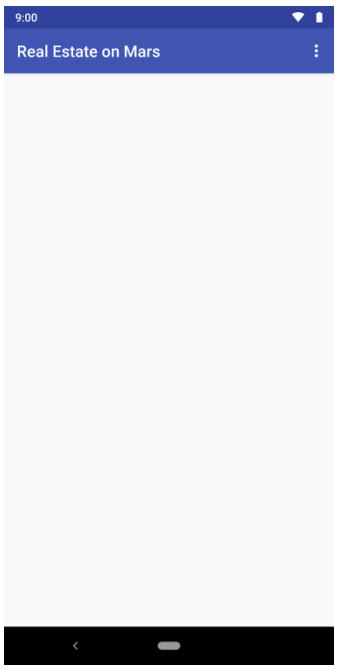
5. Otwórz `overview/OverviewFragment.kt`. W funkcji `onCreateView()`,tuż przed wywołaniem `setHasOptionsMenu()`,zainicjuj adapter `RecyclerView` w `binding.photosGrid` do nowego obiektu `PhotoGridAdapter`.

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

6. Uruchom aplikację. Powinieneś zobaczyć siatkę obrazów MarsProperty. Podczas przewijania, aby zobaczyć nowe obrazy, aplikacja wyświetla ikonę postępu ładowania przed wyświetleniem samego obrazu. Jeśli włączysz tryb samolotowy, obrazy, które nie zostały jeszcze załadowane, pojawią się jako ikony zepsutych obrazów.

5. Zadanie: dodaj obsługę błędów w RecyclerView

Aplikacja MarsRealEstate wyświetla ikonę uszkodzonego obrazu, gdy nie można pobrać obrazu. Ale gdy nie ma sieci, aplikacja wyświetla pusty ekran.



To nie jest wspaniałe doświadczenie użytkownika. W tym zadaniu dodajesz podstawową obsługę błędów, aby dać użytkownikowi lepszy obraz tego, co się dzieje. Jeśli internet nie jest dostępny, aplikacja wyświetli ikonę błędu połączenia. Podczas pobierania listy MarsProperty aplikacja wyświetli animację ładowania.

Step 1: Dodaj status do view model

Aby rozpocząć, należy utworzyć `LiveData` w modelu widoku reprezentującym status żądania internetowego. Należy wziąć pod uwagę trzy stany - ładowanie, sukces i niepowodzenie. Stan ładowania ma miejsce podczas oczekiwania na dane w wywołaniu `await()`.

1. Otwórz `overview/OverviewViewModel.kt`. W górnej części pliku (po zaimportowaniu, przed definicją klasy) dodaj wyliczenie `enum` reprezentujące wszystkie dostępne statusy:

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

2. Zmień nazwę zarówno wewnętrznych, jak i zewnętrznych definicji danych `_response` live data w klasie `OverviewViewModel` na`_status`. Ponieważ dodawano obsługę `_properties` `LiveData` wcześniej pełna odpowiedź usługi sieci Web nie była używana. Potrzebujesz tutaj a `LiveData` aby śledzić bieżący stan, więc możesz po prostu zmienić nazwę istniejących zmiennych.

   Zmień także typy z `String` na `MarsApiStatus`.

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

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

3. Przewiń w dół do metody `getMarsRealEstateProperties()` i tutaj zaktualizuj `_response` na `_status` Zmień `"Success"` string na stan `MarsApiStatus.DONE` a `"Failure"` string na `MarsApiStatus.ERROR`.
4. Dodaj status `MarsApiStatus.LOADING` na górze bloku `try {}` przed wywołaniem funkcji `await()`.Jest to stan początkowy, gdy coroutine działa i czekasz na dane. Cały blok `try/catch {}` wygląda teraz tak:

```
try {
    _status.value = MarsApiStatus.LOADING
   var listResult = getPropertiesDeferred.await()
   _status.value = MarsApiStatus.DONE
   _properties.value = listResult
} catch (e: Exception) {
   _status.value = MarsApiStatus.ERROR
}
```

5. Po stanie błędu w bloku `catch {}` ustaw `_properties` `LiveData` na pustą listę. Spowoduje to wyczyszczenie `RecyclerView`.

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

Step 2: Dodaj adapter wiązania dla statusu ImageView

Teraz masz status w modelu widoku, ale to tylko zestaw stanów. Jak sprawić, by pojawił się w samej aplikacji? W tym kroku używasz ImageView, podłączonego do powiązania danych, aby wyświetlić ikony stanów ładowania i błędów. Gdy aplikacja jest w stanie ładowania lub błędu, ImageView powinien być widoczny. Po zakończeniu ładowania aplikacji ImageView powinien być niewidoczny.

1. Otwórz `BindingAdapters.kt`. Dodaj nowy adapter wiązania o nazwie `bindStatus()` który przyjmuje argumenty `ImageView` i `MarsApiStatus` Na żądanie zaimportuj`com.example.android.marsrealestate.overview.MarsApiStatus`.

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

2. Dodaj `when {}` w metodzie `bindStatus()` aby przełączać się między różnymi statusami.

```
when (status) {

}
```

3. Wewnątrz `when {}`, dodaj case dla stanu ładowania (`MarsApiStatus.LOADING`). W tym stanie ustaw `ImageView` na widoczne i przypisz animację ładowania. Jest to ta sama animacja, której użyłeś w poprzednim zadaniu. Zaimportuj `android.view.View`.

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

4. Dodaj przypadek stanu błędu, którym jest `MarsApiStatus.ERROR`. Podobnie do tego, co zrobiłeś dla stanu ŁADOWANIE, ustaw status `ImageView` na widoczny i użyj ponownie rysowalnego błędu połączenia.

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

5. Dodaj case dla stanu ukończenia, którym jest `MarsApiStatus.DONE`. Tutaj masz udaną odpowiedź, więc wyłącz widoczność statusu ImageView, aby go ukryć..

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

Step 3: Dodaj status ImageView do układu

1. Otwórz `res/layout/fragment_overview.xml`. Pod elementem `RecyclerView` wewnątrz `ConstraintLayout`, dodaj `ImageView` pokazany poniżej.

   `ImageView` ma takie same ograniczenia jak `RecyclerView`. Jednak jako szerokość i wysokość używają `wrap_content` do wyśrodkowania obrazu zamiast rozciągania obrazu w celu wypełnienia widoku. Zwróć także uwagę na atrybut `app:marsApiStatus` który ma widok wywoływania `BindingAdapter` , gdy zmienia się właściwość status w modelu widoku.

```
<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. Włącz tryb samolotowy w emulatorze lub urządzeniu, aby zasymulować brakujące połączenie sieciowe. Skompiluj i uruchom aplikację i zauważ, że pojawia się obraz błędu:

3. Stuknij przycisk Wstecz, aby zamknąć aplikację i wyłączyć tryb samolotowy.
4. Uruchom aplikacje. W zależności od szybkości połączenia sieciowego może pojawić się wyjątkowo krótki mechanizm ładowania, gdy aplikacja wysyła zapytanie do usługi internetowej, zanim obrazy zaczną się ładować