Desenvolvimento de Aplicações Móveis
Mobile Application Development
DAM

Tutorial 3 - Pokedex - Part 1

Ana Duarte Correia
ana.correia@isel.pt

Pedro Fazenda
pedro.fazenda@isel.pt

Abstract

Jetpack is a suite of libraries to help developers follow best practices, reduce boilerplate code, and write code that works consistently across Android versions and devices so that developers can focus on the code they care about.

Deadline: -

Android Jetpack

11 April 2023

Contents

List of Figures

1 Introduction

In the next two tutorials, we will develop a more complex application following patterns used in professional Android app development, and we will also cover Jetpack, which is a collection of libraries developed by Google that assist in application development and contain some of the patterns we will address.

Follows a set of links that you should consult during the development of this work:

- Android Jetpack: `https://developer.android.com/jetpack`
- Android JetPack - Navigation: `https://developer.android.com/guide/navigation`
- Android JetPack - Paging: `https://developer.android.com/topic/libraries/architecture/paging/v3-overview`
- Android JetPack - WorkManager: `https://developer.android.com/topic/libraries/architecture/workmanager`
- Android JetPack - Data Binding : `https://developer.android.com/topic/libraries/data-binding/`
- Android JetPack - Lifecycle: `https://developer.android.com/topic/libraries/architecture/lifecycle`
- Android JetPack - LiveData: `https://developer.android.com/topic/libraries/architecture/livedata`
- Android JetPack - Room: `https://developer.android.com/training/data-storage/room`
- Android JetPack - ViewModel: `https://developer.android.com/topic/libraries/architecture/viewmodel`
- Clean Architecture - `https://developer.android.com/topic/architecture`

2 Pokedex

The application we will implement is the Pokedex, which will have the following functionalities:

1. List the regions of the Pokemon franchise.
2. List all existing Pokemon in the Pokemon franchise.
3. List the existing Pokemon in a selected region.
4. Display details of the selected Pokemon from the Pokemon list.
5. Filter Pokemon listings by type and sort in descending/ascending order by ID or name.
6. Search for Pokemon in the listing by name or ID.

7. List the Pokemon teams created by the user.
8. Remove Pokemon teams.
9. Display information of the selected team.
10. Add/Remove Pokemon to/from the selected team.
11. Edit basic team data: name, image.
12. Authentication with username and password via Firebase.
13. Authentication with Google account.
14. Register users via username/password or Google account.

// TODO Pictures

3 Clean Architecture

Clean Architecture is a software architectural pattern for developing Android applications that focuses on separation of concerns, testability, and maintainability. It provides a structured approach to organizing code into different layers, each with a specific responsibility. These layers interact with each other in a specific way to create a modular and scalable architecture.

The main layers in Clean Architecture for Android are:

1. **Presentation Layer**: This layer is responsible for rendering the user interface and handling user interactions. It includes components such as activities, fragments, views, or view models (depending on the chosen architecture pattern, such as *MVP* or *MVVM*). The presentation layer is responsible for displaying data to the user and capturing user inputs, but it should not contain business logic or data manipulation. It communicates with the domain layer to request data and trigger actions based on user interactions.

2. **Domain Layer**: Also known as the business logic layer, this layer encapsulates the core logic of the application. It includes use cases (or interactors) that represent the individual actions or operations that can be performed by the application. Use cases are independent of any framework or external dependencies and define the business rules and logic of the application. The domain layer communicates with the data layer to fetch or store data, and it defines the contracts or interfaces that are implemented by the data layer.

3. **Data Layer**: This layer is responsible for fetching and storing data from external sources, such as databases, APIs, or repositories. It includes components such as data sources, repositories, and mappers that handle data conversion between different layers. The data layer implements the contracts or interfaces defined in the domain layer and provides the data in a format suitable for the domain layer. The data layer may also handle caching, offline data access, or other data-related operations.

The iterations in Clean Architecture refer to the flow of data and control between the layers. The typical flow of data and control in Clean Architecture is as follows:

1. The Presentation Layer communicates with the Domain Layer to request data or trigger actions based on user interactions.

2. The Domain Layer contains the business logic and rules of the application and communicates with the Data Layer to fetch or store data.

3. The Data Layer is responsible for fetching and storing data from external sources and implementing the contracts or interfaces defined in the Domain Layer.

4. The Data Layer provides the requested data back to the Domain Layer, which then processes it and returns the result to the Presentation Layer.

5. The Presentation Layer renders the user interface or takes appropriate actions based on the data received from the Domain Layer.

It's important to note that Clean Architecture promotes a one-way flow of data and control, with dependencies pointing inward, from the outer layers to the inner layers. This ensures that the inner layers, such as the Domain Layer, are independent of any framework or external dependencies, making them highly testable and maintainable.

4 JetPack

Most Android apps now use the support libraries to help users add all kinds of updated widgets and to address compatibility issues across Android devices and OS versions. You'd be hard-pressed to find an Android app that doesn't use something from them, and they're included as a dependency in template projects made in Android Studio. Widgets as fundamental as RecyclerView are included in them.

The support libraries are great to use, but they have evolved quite a bit over the years and the names have become somewhat confusing. There are com.android.support:support-v4 and com.android.support:support-v13, for example. Looking at the names alone, do you know what classes are in them? The names are supposed to designate what version of Android they support, but, as they have evolved, the minimum version has increased to API Level 14.

Google has realized that this variety of ever-changing libraries is very confusing for new (and old) developers and has decided to start over. Consequently, it has created Android Jetpack that is of components, tools and guidelines, that aims to facilitate the creation of applications. As shown in Figure 1, the Android Jetpack components bring together the support library which is a group of components that facilitate the use of the new Android features and the Architecture Components that was designed to facilitate data management during system changes application lifecycle. The components are organized into four categories:

1. Architecture;

2. UI;

3. Behavior;

4. Foundation.

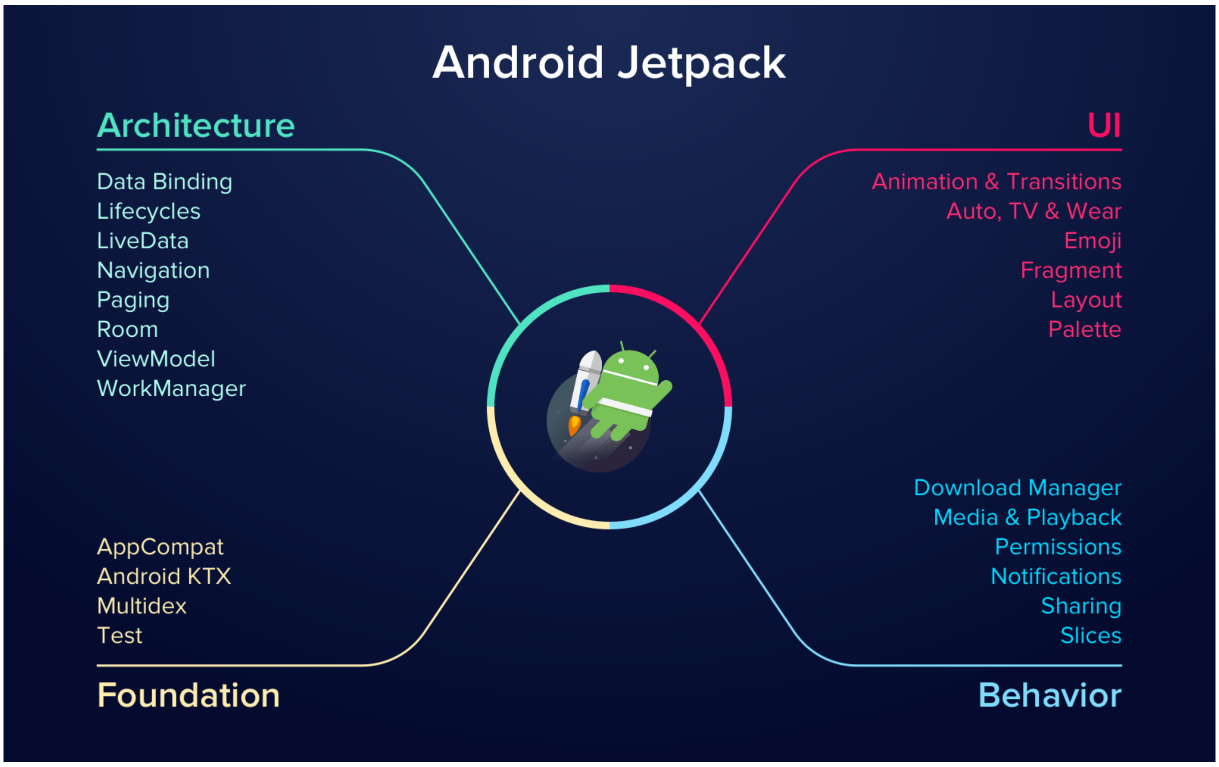

Figure 1: Android Jetpack Categories.

4.1 Architecture Components

It is a collection of libraries that help to develop applications robust, testable and easy to maintain. This library helps to manage the life cycle of UI components and better control the persistence of data even with changes in the application life cycle. In the next topics we present the most importants libraries that make part of Architecture Components.

4.1.1 Room

If you have ever struggled working with the SQLite database in an Android app, you will appreciate what the Room library does for you. You create several simple classes that define your data and how to access them, and the Room library will do most of the rest. The only SQL code you have to write is for queries, which are usually pretty straightforward. And you gain compile-time checks of your SQL code when using Room.

There are three important classes you need to use with Room: Database (this contains your main entry point and holds a reference to the database object for the app), Entity (you create one for each table in the database), and DAO (this contains the methods for retrieving and managing the data).

4.1.2 Lifecycle

The Lifecycle library helps you listen for lifecycle events in other components besides an activities and fragments. This allows you to have lifecycle-aware logic in places other than just an Activity or Fragment. The library works by using annotations on methods so you get notified for the events that you are interested in.

You implement LifecycleObserver, annotate methods and add this observer to a lifecycle owner, which is usually an Activity or Fragment. The LifecycleObserver class can get the current lifecycle state by calling lifecycle.getCurrentState() and can use this information to avoid calling code when not in the correct state.

A LifecycleOwner is an object that has Android lifecycle events. The support library Activity and Fragment classes already implement the LifecycleOwner methods. A LifecycleOwner has a Lifecycle object that it can return to let the observer know what the current state is.

4.1.3 ViewModel

While the Room library persists your data in permanent storage, the ViewModel class allows you to hold onto data in device memory in a lifecycle-aware manner. One of the nice features of a ViewModel is that it can survive the re-construction of an Activity or Fragment over a configuration change such as a device rotation. The system will hold onto that ViewModel re-associate it with the Activity or Fragment. ViewModels are also where you can load data in the background and use LiveData to notify listeners that the data is ready.

4.1.4 LiveData

The LiveData library uses the Observer pattern for data but handles it in a lifecycle-aware manner. You get the benefits of automatic UI updates when data changes without calling UI elements when the UI is not in the correct state.

LiveData is the class that implements the observer pattern, holds that data, and notifies listeners when that data has changed.

4.1.5 Paging

Have you ever had to deal with large amounts of related data? Maybe too much for you to download at once? The Paging library will help by providing ways to handle the paging of data in a RecyclerView.

The Paging library uses several key classes: PagedList, PagedListAdapter, and DataSource. PagedList is a list that loads data lazily from a DataSource, allowing the app to load data in chunks or pages. PagedListAdapter is a custom RecyclerView.Adapter that handles pages with a DiffUtil callback.

For the DataSource, you will use one of three different subclasses: PageKeyedDataSource, ItemKeyedDataSource, or PositionalDataSource.

4.1.6 WorkManager

Over the years, there have been several systems built into Android for handling background jobs or alarms. They differ on different versions of Android and you have to write a lot of code to handle the different versions of the OS.

WorkManager solves this problem and gives you one library for creating deferrable, asynchronous tasks and defining when they should run. You can define one-time jobs or repeating jobs.

5 MVVM in android

MVVM (Model-View-ViewModel) is a widely used architectural pattern in Android app development that provides a clear separation of concerns and promotes a modular, maintainable, and testable codebase. When used in conjunction with Clean Architecture, MVVM can help create scalable and maintainable Android applications. Here's how MVVM can be implemented in Clean Architecture in an Android app using Jetpack components:

1. Model: The Model in MVVM represents the data and business logic of the application. In Clean Architecture, it typically includes the following components:

2. Entity: It represents the domain model, which is a pure Kotlin/Java representation of the data entities in the application. It contains only data and no business logic.

3. Repository: It acts as an interface between the data sources (e.g., local database, remote server) and the domain layer. It defines the contract for accessing and managing data, and it can implement different data sources based on the application's requirements.

4. Use Case/Interactor: It encapsulates the business logic of the application, defining the different use cases or interactions that can be performed by the application. It communicates with the repositories to fetch or modify data and defines the input and output models for each use case.

5. View: The View represents the UI components of the application, including activities, fragments, and views. In MVVM, the View is passive and only responsible for displaying data and capturing user input. It does not contain any business logic. The View observes the ViewModel and updates the UI based on the data received from the ViewModel.

6. ViewModel: The ViewModel acts as an intermediary between the View and the Model. It holds the UI-related state and business logic of the View. It exposes data to the View through observable properties, and it receives input from the View through commands or events. The ViewModel communicates with the Model to fetch or modify data and maps the data into a format that can be easily consumed by the View. The ViewModel is also responsible for handling UI-related events, such as user input validation and navigation.

7. Data Binding: MVVM in Android often utilizes data binding, which is a feature provided by Android Jetpack. Data binding allows for declarative binding of UI components to data models, eliminating the need for manual UI updates. This means that the View automatically updates when the data in the ViewModel changes, and vice versa.

8. Observables: Observables are used to enable communication between the View and the ViewModel. The ViewModel exposes observables (e.g., LiveData, ObservableField) that the View observes to get notified of data changes. When data changes in the Model or ViewModel, the observables emit notifications, which trigger updates in the View.

9. Jetpack Components: Android Jetpack provides several components that can be used in MVVM, such as LiveData and ViewModel. LiveData is a lifecycle-aware observable data holder that can be used to expose data from the ViewModel to the View, and it automatically handles UI updates based on the lifecycle of the View. ViewModel is a lifecycle-aware component that is designed to store and manage UI-related data, and it survives configuration changes, such as screen rotations.

6 Let's Start

n this tutorial, the focus will be mainly on the Presentation layer, although for testability, the base model of the application will be implemented, and fictitious (mock) data and some rules will be used in the Domain layer. In Tutorial 4, the focus will be on the Model and Domain layers.

First, create a new project using the "Navigation Drawer Activity" template, which will create an application with a hamburger menu.

Before analyzing the project, you can delete some parts that will not be necessary for this tutorial:

1. MainActivity: should only contain this code:

```
class MainActivity : AppCompatActivity() {

private lateinit var appBarConfiguration: AppBarConfiguration
private lateinit var binding: ActivityMainBinding

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    setSupportActionBar(binding.appBarMain.toolbar)

    val drawerLayout: DrawerLayout = binding.drawerLayout
    val navView: NavigationView = binding.navView
    val navController = findNavController(R.id.nav_host_fragment_content_main)
    // Passing each menu ID as a set of Ids because each
    // menu should be considered as top level destinations.
    appBarConfiguration = AppBarConfiguration(
        setOf(
```

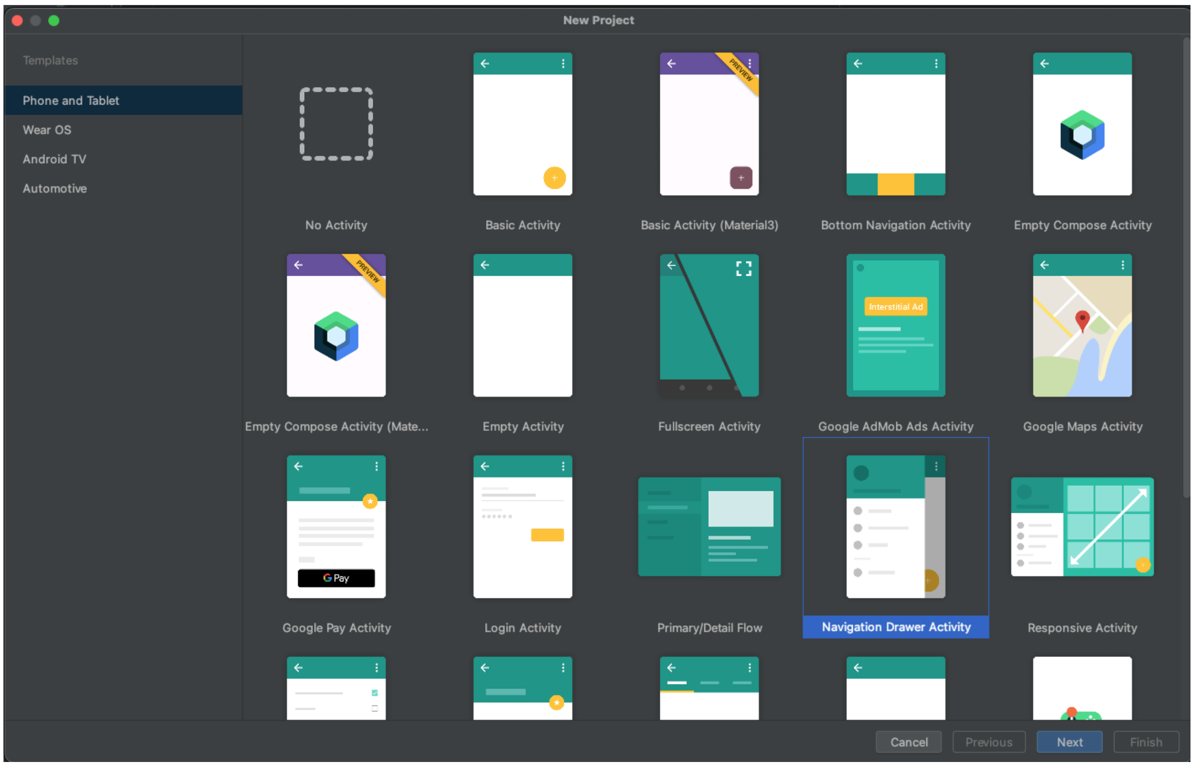

Figure 2: Project Template

```
               R.id.nav_regions, R.id.nav_gallery, R.id.nav_slideshow, R.id.
                   nav_home
           ), drawerLayout
       )
        setupActionBarWithNavController(navController, appBarConfiguration)
         navView.setupWithNavController(navController)
    }

    override fun onSupportNavigateUp(): Boolean {
        val navController = findNavController(R.id.nav_host_fragment_content_main)
        return navController.navigateUp(appBarConfiguration) || super.
            onSupportNavigateUp()
    }
}
```

Listing 1: Activity Main code.

2. Delete the `/menu/main.xml` file.

3. In the `mobile_navigation/app_bar_main.xml` file, only the following code should remain.

In the Moodle, you will find a zip file with the resources that will be used in the project. These resources are divided into the following folders:

1. drawable

2. strings

3. colors

You should copy or import these files into the new project.

Add to /build.gradle the fowolling lines in correct places:

```
plugins {
    ...
    id 'org.jetbrains.kotlin.kapt'
    id 'kotlin-parcelize'
}

android {
    ...
    buildFeatures {
        viewBinding true
        dataBinding true
    }
}

dependencies {

...

/* Glide Libraries for download images using urls*/
implementation 'com.github.bumptech.glide:glide:4.15.0'
annotationProcessor 'com.github.bumptech.glide:compiler:4.15.0'

/* Palette Library for obtain colors from images */
implementation 'androidx.palette:palette:1.0.0'

/* Progress View library */
implementation "com.github.skydoves:progressview:1.1.3"
}
```

Listing 2: /build.gradle.

So let's start to implement the Pokedex APP

And create a the following packages(Figure 5):

1. ui - Contains all the classes of UI (Fragments and ViewModels).
2. domain - Contains the classes that handle uses cases operations.
3. data - Contains the application's data model classes.

6.1 Basic Data Layer and Basic Domain

We will create the necessary basic entities of our data model to be able to test the application.

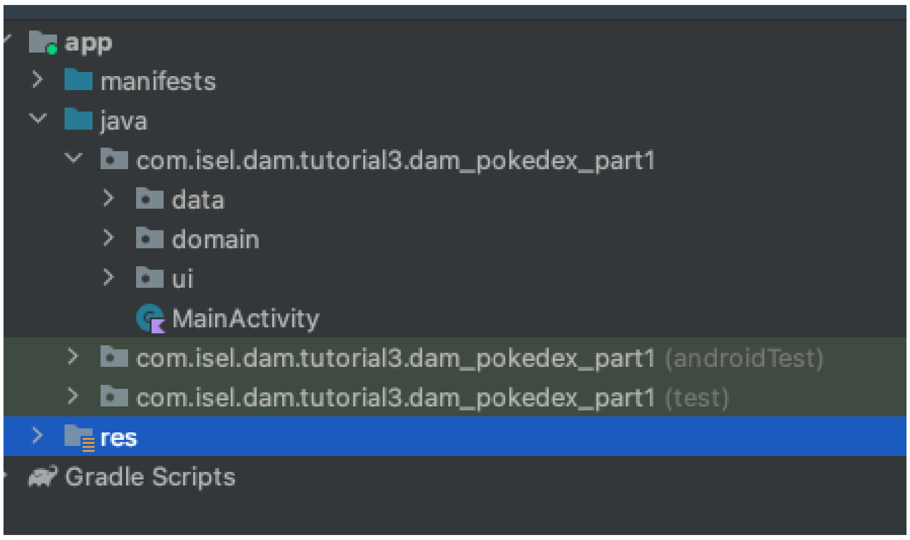
app
manifests
java
com.isel.dam.tutorial3.dam_pokedex_part1
data
domain
ui
MainActivity
com.isel.dam.tutorial3.dam_pokedex_part1 (androidTest)
com.isel.dam.tutorial3.dam_pokedex_part1 (test)
res
Gradle Scripts

Figure 3: Project packages

1. **PokemonRegion**: which will contain the base information of the region - *id, name, backgroundImage, startersImage* - 30
2. **Pokemon**: which will contain the base information of the pokemon - *id, name, imageUrl, regions, types* - 28
3. **PokemonDetail**: which will contain the detailed information of the pokemon - *id, pokemon, description, weight, height, stats, evolutions* - 29
4. **PokemonType**: which will contain the information of the pokemon type - *id, name, iconImage, color* - 31
5. **PokemonEvolution**: which will contain the information of the pokemon Evolution chain - *id, pokemon* - 33
6. **PokemonStats**: which will contain the information of the pokemon "powers" - *id, hp, attack, defense, specialDefence, speed, experience* - 32

As one of the requirements of the application is to consume data using the POKEAPI, but we will only implement it in the next tutorial, it is necessary to use mock data to test the application. For this purpose, we will create the static object *PokemonMockData*, which is located in the package *data/mocks* 34.

We will also create the necessary basic functions to obtain data for testing the application.

```
class PokemonDomain
{
    fun getAllRegions() : LiveData<List<PokemonRegion>>
    {
        return MutableLiveData<List<PokemonRegion>>(PokemonMockData.regions)
    }

    fun getAllPokemons(): LiveData<List<Pokemon>>
    {
        return MutableLiveData<List<Pokemon>>(PokemonMockData.pokemons)
    }
```

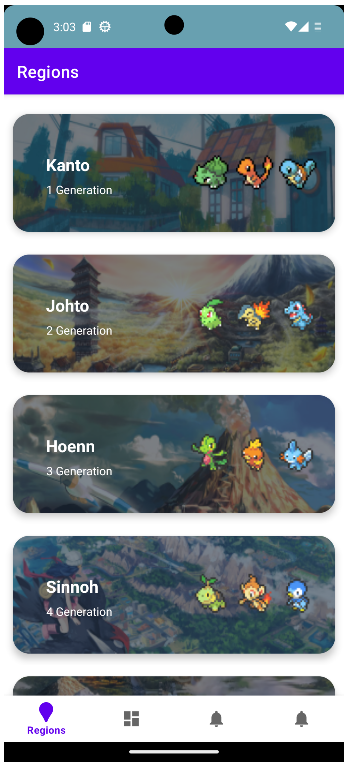

Figure 4: Region Screen.

```
        fun getPokemonsByRegion(region: PokemonRegion): LiveData<List<Pokemon>>
        {
            return MutableLiveData<List<Pokemon>>(PokemonMockData.pokemons.filter { it.
                region == region })
        }

        fun getPokemonTypes(): List<PokemonType>
        {
            return ArrayList<PokemonType>(PokemonMockData.pokemonTypeMock)
        }

        fun getPokemonDetail(pokemon:Pokemon): LiveData<PokemonDetail>
        {
            return MutableLiveData(
                PokemonMockData.pokemonDetail.find { it.pokemon == pokemon })
        }
    }
```

Listing 3: Region Fragment Layout.

7 First Feature - Region List

Create a fragment where shows the list of regions of the pokemon, like the Figure 4

7.1 RecyclerView

RecyclerView is an advanced and flexible version of ListView, which is a commonly used widget in Android to display a scrollable list of data. RecyclerView is part of the Android Jetpack library and offers several advantages over ListView, such as better performance, extensibility, and a more flexible layout manager system. RecyclerView recycles the views that are no longer visible on the screen, reducing the memory usage of the app and improving its performance. The RecyclerView consists of three main components:

1. **Layout Manager**: It is responsible for measuring and positioning the items in the RecyclerView. It can be either a built-in layout manager like LinearLayoutManager, GridLayoutManager, or StaggeredGridLayoutManager, or a custom layout manager.

2. **Adapter**: It is responsible for providing the views that represent the data items in the RecyclerView. It creates a view holder for each item and binds the data to the views in the ViewHolder.

3. **ViewHolder**: It holds the views that represent the data items in the RecyclerView. It provides a reference to these views so that the adapter can bind the data to them.

First let's create the Fragment with View Model with the name *RegionFragment* and in layout let's create a a RecyclerView with the LinearLayoutManager:

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".ui.region.RegionFragment">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/regionsRecyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layoutManager="LinearLayoutManager"/>

</FrameLayout>
```

Listing 4: Region Fragment Layout.

Second, let's create the region item layout, for that you have to add a new Layout Resource File and add the fowolling xml code:

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.appcompat.widget.LinearLayoutCompat android:id="@+id/
   relativeLayoutBackground"
    android:layout_width="match_parent"
    android:layout_height="170dp"
    android:clickable="true"
    android:clipToPadding="true"
    android:focusable="true"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">
```

```
<com.google.android.material.card.MaterialCardView
    android:layout_width="match_parent"
    android:layout_height="170dp"
    android:layout_margin="0dp"
    app:cardCornerRadius="20dp"
    app:cardElevation="5dp"
    app:cardPreventCornerOverlap="true"
    app:cardUseCompatPadding="true"
    android:padding="0dp">

    <androidx.appcompat.widget.AppCompatImageView
        android:id="@+id/regionBgImage"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY"
        android:foreground="#96222f3e"
        android:foregroundTintMode="src_in"
        tools:src="@drawable/bg_hoenn"/>


    <androidx.appcompat.widget.AppCompatImageView
        android:id="@+id/regionStartersImageView"
        android:layout_width="197dp"
        android:layout_height="120dp"
        android:layout_gravity="center|right"
        android:layout_margin="0dp"
        android:contentDescription="@string/app_name"
        android:scaleType="center"
        tools:src="@drawable/pk_hoenn"/>

    <androidx.appcompat.widget.AppCompatTextView
        android:id="@+id/regionNameTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="left|center_vertical"
        android:layout_marginStart="40dp"
        android:layout_marginBottom="10dp"
        android:gravity="center|left"
        android:textColor="@color/white"
        android:textSize="20sp"
        android:textStyle="bold"
        tools:text="Hoenn"/>

    <androidx.appcompat.widget.AppCompatTextView
```

```
                    android:id="@+id/regionIdTextView"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="left|center_vertical"
                    android:layout_marginLeft="40dp"
                    android:layout_marginTop="20dp"
                    android:gravity="center"
                    android:textColor="@color/white"
                    android:textSize="14sp"
                    android:textStyle="bold"
                    app:fontFamily="sans-serif-thin"
                    tools:text="3 Generation"/>
            </com.google.android.material.card.MaterialCardView>

        </androidx.appcompat.widget.LinearLayoutCompat>
```

Listing 5: Region Item Layout.

Third, let's create the *Region Adapter* with the ViewHolder.

```
    class RegionAdapter(
    private val pkRegionList: List<PokemonRegion>,
    private val context: Context
) : RecyclerView.Adapter<RegionAdapter.ViewHolder>() {

    class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        private val viewBinding = ItemRegionBinding.bind(itemView)
        fun bindView(regionItem: PokemonRegion)
        {
            viewBinding.regionNameTextView.text = regionItem.name
            viewBinding.regionIdTextView.text = "${regionItem.id} ${R.string.pk_generations}
                ".uppercase()
            viewBinding.regionBgImage.setImageDrawable(ContextCompat.getDrawable(itemView.
                context,regionItem.bg))
            viewBinding.regionStartersImageView.setImageDrawable(ContextCompat.getDrawable(
                itemView.context,regionItem.starters))
            itemView.setOnClickListener{
                Toast.makeText(itemView.context, String.format("Click in %s Region",
                    regionItem.name), Toast.LENGTH_LONG).show()
            };
        }
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val inflater = LayoutInflater.from(context)
        val view = inflater.inflate(R.layout.item_region, parent, false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val item = pkRegionList[position]
        holder.bindView(item)
    }

    override fun getItemCount(): Int {
        return pkRegionList.size
    }
}
```

Listing 6: Region Adapter Class.

The Adapter is a crucial component of the RecyclerView as it provides a bridge between the data source and the views that represent that data in the RecyclerView. The Adapter is responsible for creating and managing the ViewHolder objects, which are responsible for holding references to the views that represent each item in the RecyclerView. The Adapter also binds the data to the views in the ViewHolder.

The Adapter class is an abstract class, and you need to extend it and implement its three methods:

1. **onCreateViewHolder()** : This method creates a new ViewHolder object for each item in the RecyclerView. It inflates the layout for the item view and returns a new instance of the ViewHolder.

2. **onBindViewHolder()**: This method binds the data to the views in the ViewHolder. It takes the position of the item in the data source and updates the views in the ViewHolder with the corresponding data.

3. **getItemCount()**: This method returns the total number of items in the data source.

In the constructor of the RegionAdapter class, the list of PokemonRegion objects and a context are passed as parameters. The context is used to create a LayoutInflater instance, which is used to inflate the item layout.

The ViewHolder class is defined as a nested class within the adapter, and it extends the RecyclerView.ViewHolder class. In the ViewHolder class, the view binding is done using the ItemRegionBinding class, which is generated by the Android Data Binding Library. The bindView() method is used to bind the view with the data for a single item in the list.

In the onCreateViewHolder() method, the item layout is inflated and a ViewHolder instance is created and returned.

In the onBindViewHolder() method, the data for a single item is retrieved from the list, and the ViewHolder's bindView() method is called to bind the view with the data.

Finally, the getItemCount() method is used to return the number of items in the list.

Third create the *RegionViewModel* aims to obtain the list of regions that are exist in pokemon franchise.

```
class RegionViewModel : ViewModel()
{
    private val _pokemonDomain = PokemonDomain()

    fun getRegions(): LiveData<List<PokemonRegion>>
    {
        return _pokemonDomain.getAllRegions()
    }
}
```

Listing 7: Region View Model Class.

ViewModels offer a number of benefits:

1. ViewModel‘s are lifecycle-aware, which means they know when the attached Activity/Fragment is destroyed and can immediately release data observers and other resources.
2. They survive configuration changes, so if your data is observed or fetched through a ViewModel, it’s still available after your Activity or Fragment is re-created. This means you can re-use the data without fetching it again.
3. ViewModel takes the responsibility of holding and managing data. It acts as a bridge between your Repository and the View. Freeing up your Activity or Fragment from managing data allows you to write more concise and unit-testable code.

To enable the usage of view models in your Android application, add the following snippet to the app/build.gradle file.

```
android {
  ....
  buildFeatures {
    viewBinding true
  }
}
```

Listing 8: Region Fragment.

For the last, lets connect all this components in Fragment:

```
class RegionFragment : Fragment() {

  private var _regionViewModel: RegionViewModel? = null

  private var _binding: FragmentRegionBinding? = null

  private val binding get() = _binding!!

  private val regionViewModel get() = _regionViewModel!!

  override fun onCreateView(
      inflater: LayoutInflater,
      container: ViewGroup?,
      savedInstanceState: Bundle?
  ): View {
      _regionViewModel =
          ViewModelProvider(this)[RegionViewModel::class.java]

      _binding = FragmentRegionBinding.inflate(inflater, container, false)

      return binding.root
  }

  override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
      super.onViewCreated(view, savedInstanceState)

      regionViewModel.getRegions().observe(viewLifecycleOwner) {
          val regions: List<PokemonRegion> = it
```

```
            binding?.regionsRecyclerView?.adapter = RegionAdapter(regions, view.context)
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

Listing 9: View Models Gradle.

In the *onCreateView()* method, the fragment inflates its layout using the FragmentRegionBinding class. This class is generated by the Android Data Binding Library and provides a type-safe way to access views in the layout. The *RegionViewModel* is also initialized using the ViewModelProvider class.

In the *onViewCreated()* method, the RecyclerView adapter is set up using the RegionAdapter class, which takes a list of PokemonRegion objects and a context. The adapter is set on the RecyclerView using the binding object, which is the instance of FragmentRegionBinding that was created in the *onCreateView()* method. The adapter is updated whenever the data in the RegionViewModel changes.

So lets testing this feature, for that we have to do little changes in app navigation.

Jetpack Navigation is a library provided by Google to simplify the navigation process between different screens (Fragments, Activities) in an Android application. With Jetpack Navigation, developers can easily create a navigation graph that maps out the different screens in their app and the possible paths between them.

The Jetpack Navigation library provides several benefits for developers:

1. Simplified Navigation: Jetpack Navigation provides a simple and consistent way to navigate between different screens in an Android application. It eliminates the need for developers to write boilerplate code to handle navigation, which can be tedious and error-prone.

2. Type-safe navigation: Jetpack Navigation uses the Safe Args plugin to generate type-safe classes for passing arguments between different screens. This ensures that developers can catch errors early in the development process and avoid runtime crashes due to passing the wrong data type between screens.

3. Back Stack Management: Jetpack Navigation provides automatic back stack management for fragments. This means that when the user presses the back button, the application automatically navigates back to the previous screen, instead of exiting the app.

Open the file *mobilenavigation.xml* in the folder *res/navigation.*

```
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/mobile_navigation"
    app:startDestination="@+id/nav_home">
    <fragment
        android:id="@+id/nav_home"
        android:name="com.isel.dam.tutorial3.dam_pokedex_part1.ui.home.
            HomeFragment"
        android:label="@string/menu_home"
        tools:layout="@layout/fragment_home" >

    </fragment>

    <fragment
        android:id="@+id/nav_gallery"
        android:name="com.isel.dam.tutorial3.dam_pokedex_part1.ui.gallery.
            GalleryFragment"
        android:label="@string/menu_gallery"
        tools:layout="@layout/fragment_gallery" />

    <fragment
        android:id="@+id/nav_slideshow"
        android:name="com.isel.dam.tutorial3.dam_pokedex_part1.ui.slideshow.
            SlideshowFragment"
        android:label="@string/menu_slideshow"
        tools:layout="@layout/fragment_slideshow" />

</navigation>
```

Listing 10: Navigation Code.

This is an XML code that defines the navigation graph for an Android application using Jetpack Navigation. A navigation graph is a collection of destinations, which represent different screens in the app, and the actions that connect them.

The navigation graph is defined using the *¡navigation¿* tag with the following attributes:

android:id sets the ID for the navigation graph to `mobile_navigation`. **app:startDestination** sets the starting destination to `nav_home`. The navigation graph contains four fragments, each represented by the **¡fragment¿** tag:

1. **`nav_home`** is the home screen of the app and is defined by the HomeFragment class.
2. **`nav_gallery`** is a gallery screen and is defined by the GalleryFragment class.
3. **`nav_slideshow`** is a slideshow screen and is defined by the SlideshowFragment class.

Each ¡fragment¿ tag specifies the following attributes:

1. **android:id** sets the ID for the fragment.
2. **android:name** sets the fully qualified name of the fragment's class.
3. **android:label** sets the label that is displayed for the fragment in the app's navigation drawer or toolbar.

4. **tools:layout** specifies the layout resource for the fragment's view.

The navigation graph represents the hierarchy of the app's screens and the transitions between them. It enables the app to manage and handle user navigation in a structured and organized manner.

Let's add the Region Fragment to our navigation graph:

```xml
 <navigation xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:id="@+id/mobile_navigation"
   app:startDestination="@+id/nav_home">

   <fragment
       android:id="@+id/nav_home"
       android:name="com.isel.dam.tutorial3.dam_pokedex_part1.ui.home.
           HomeFragment"
       android:label="@string/menu_home"
       tools:layout="@layout/fragment_home" >

   </fragment>

   <fragment
       android:id="@+id/nav_gallery"
       android:name="com.isel.dam.tutorial3.dam_pokedex_part1.ui.gallery.
           GalleryFragment"
       android:label="@string/menu_gallery"
       tools:layout="@layout/fragment_gallery" />

   <fragment
       android:id="@+id/nav_slideshow"
       android:name="com.isel.dam.tutorial3.dam_pokedex_part1.ui.slideshow.
           SlideshowFragment"
       android:label="@string/menu_slideshow"
       tools:layout="@layout/fragment_slideshow" />

   <fragment
       android:id="@+id/nav_regions"
       android:name="com.isel.dam.tutorial3.dam_pokedex_part1.ui.region.
           RegionFragment"
       android:label="@string/menu_regions"
       tools:layout="@layout/fragment_region" />

</navigation>
```

Listing 11: Navigation with Region Fragment.

And add the Regions option to Drawer Menu in `res/menu/activity_main_drawer`:

```
    <item
    android:id="@+id/nav_regions"
    android:icon="@drawable/ic_region"
    android:title="@string/menu_regions" />
```

Listing 12: Navigation with Region Fragment.

7.2 Data Binding

Data Binding is a feature provided by Android Jetpack that allows you to bind UI components in your layout directly to data sources in your app's code. This means you can avoid writing a lot of boilerplate code to update your UI components with data from your app.

To use Data Binding in this app, you need to enable it by adding the following to your module's build.gradle file:

```
    android {
    ....
    buildFeatures {
      databinding true
    }
  }
```

Listing 13: Navigation with Region Fragment.

Once you've enabled Data Binding, you can start using it in your layout files by enclosing your layout in a ¡layout¿ tag. Within the ¡layout¿ tag, you can use the @ syntax to bind data to UI components. For example in Region item Layout, you can bind a string to a *regionIdTextView* and *regionNameTextView* like this:

```
    <layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>
        <variable
            name="region"
            type="com.isel.dam.tutorial3.dam_pokedex.data.model.PokemonRegion" />

    </data>
<androidx.appcompat.widget.LinearLayoutCompat
    android:id="@+id/relativeLayoutBackground"
    android:layout_width="match_parent"
    android:layout_height="170dp"
    android:clickable="true"
    android:clipToPadding="true"
    android:focusable="true">

    <com.google.android.material.card.MaterialCardView
        android:layout_width="match_parent"
        android:layout_height="170dp"
        android:layout_margin="0dp"
        app:cardCornerRadius="20dp"
        app:cardElevation="5dp"
        app:cardPreventCornerOverlap="true"
        app:cardUseCompatPadding="true"
        android:padding="0dp"

        >

        <androidx.appcompat.widget.AppCompatImageView
            android:layout_width="match_parent"
            android:id="@+id/regionBgImage"
            android:layout_height="match_parent"
            android:scaleType="fitXY"
            android:src="@{region.bg}"
            android:foreground="#96222f3e"
            android:foregroundTintMode="src_in"/>


        <androidx.appcompat.widget.AppCompatImageView
            android:id="@+id/regionStartersImageView"
            android:layout_width="197dp"
            android:layout_height="120dp"
            android:layout_gravity="center|right"
            android:layout_margin="0dp"
            android:contentDescription="@string/app_name"
            android:scaleType="center"
            android:src="@{region.starters}" />

        <androidx.appcompat.widget.AppCompatTextView
            android:id="@+id/regionTitleTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="left|center_vertical"
            android:layout_marginStart="40dp"
            android:layout_marginBottom="10dp"
            android:gravity="center|left"
            android:text="@{region.name}"
            android:textColor="@color/white"
            android:textSize="20sp"
            android:textStyle="bold" />

        <androidx.appcompat.widget.AppCompatTextView
            android:id="@+id/regionSubtitleTextView"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="left|center_vertical"
            android:layout_marginLeft="40dp"
            android:layout_marginTop="20dp"
            android:gravity="center"
            android:text="@{region.id.toString()}"
            android:textColor="@color/white"
            android:textSize="14sp"
            android:textStyle="bold"
            app:fontFamily="sans-serif-thin" />



    </com.google.android.material.card.MaterialCardView>


</androidx.appcompat.widget.LinearLayoutCompat>
    </layout>
```

Listing 14: Navigation with Region Fragment.

For the *regionBgImage* and *regionStartersImageView* we need to create a custom binding for setting the image resource (src) of an ImageView, because we need translate integer to Bitmap

```
    object ViewBinding {

    @JvmStatic
    @BindingAdapter("android:src")
    fun setImageResource(imageView: AppCompatImageView, resource: Int) {
        imageView.setImageDrawable(ContextCompat.getDrawable(imageView.context,resource))
    }

}
```

Listing 15: Navigation with Region Fragment.

1. *@JvmStatic*: This annotation is used to indicate that the function should be treated as a static method in Java. This allows the function to be called from Java code as a static method, rather than as an instance method.
2. *@BindingAdapter("android:src")*: This is a custom binding adapter annotation provided by the Data Binding library. It specifies that this binding adapter is intended to be used to bind the "android:src" attribute of an ImageView in a layout XML file.
3. *fun setImageResource(imageView: AppCompatImageView, resource: Int)*: This is the definition of the binding adapter function. It takes two parameters:
   (a) *imageView*: An instance of AppCompatImageView, which is the ImageView to which the binding adapter is being applied.
   (b) *resource*: An integer value, which represents the image resource ID that needs to be set as the source of the ImageView.

   imageView.setImageDrawable(ContextCompat.getDrawable(imageView.context, resource)): This line of code sets the image resource for the ImageView. It uses the ContextCompat.getDrawable() method to retrieve a Drawable object from the given resource ID using the context of the ImageView, and then sets it as the drawable source of the ImageView using the setImageDrawable() method.

We need to change the Adapter View Holder in *bindView* to bind the property region:

```
    fun bindView(regionItem: PokemonRegion)
        {
            /*viewBinding.regionNameTextView.text = regionItem.name
            viewBinding.regionIdTextView.text = "${regionItem.id} ${R.string.pk_generations
                }".uppercase()
            viewBinding.regionBgImage.setImageDrawable(ContextCompat.getDrawable(itemView.
                context,regionItem.bg))
            viewBinding.regionStartersImageView.setImageDrawable(ContextCompat.getDrawable(
                itemView.context,regionItem.starters))*/
            viewBinding.region = regionItem
            itemView.setOnClickListener{
                Toast.makeText(itemView.context, String.format("Click in %s Region",
                    regionItem.name), Toast.LENGTH_LONG).show()
            };
        }
}
```

Listing 16: Binding property Region.

Let's test the Region Fragment:

Go to `res/navigation/mobile_navigation.xml` and add the RegionFragments using the icon screen with green plus and change the id Fragment to "`nav_regions`" Go to `res/menu/activity_main_drawer.xml` and new item:

```
    <item
    android:id="@+id/nav_regions"
    android:icon="@drawable/ic_region"
    android:title="@string/menu_regions" />
```

Listing 17: Add new Menu item.

Go to `MainActivity` and change the AppBarConfiguration:

```
    appBarConfiguration = AppBarConfiguration(
        setOf(
            R.id.nav_regions, R.id.nav_gallery, R.id.nav_slideshow, R.id.nav_home
        ), drawerLayout
    )
```

Listing 18: Add new Menu item.

8 Pokemon List

Create a new layout `item_pokemon.xml`

```
    <?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>


        <variable
            name="pokemon"
            type="com.isel.dam.tutorial3.dam_pokedex_part1.data.model.Pokemon
                " />


    </data>


    <com.google.android.material.card.MaterialCardView
        android:id="@+id/cardView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="6dp"
        android:foreground="?attr/selectableItemBackground"
        app:cardBackgroundColor="@color/white"
        app:cardCornerRadius="14dp"
        app:cardElevation="4dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:cardBackgroundColor="@color/water">

        <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:foreground="?attr/selectableItemBackground"
            app:cardBackgroundColor="@color/white">

            <androidx.appcompat.widget.AppCompatImageView
                android:id="@+id/image"
                android:layout_width="90dp"
                android:layout_height="90dp"
                android:layout_margin="20dp"
                android:layout_marginStart="10dp"
                android:layout_marginTop="5dp"
                android:layout_marginEnd="10dp"
                android:layout_marginBottom="10dp"
                android:adjustViewBounds="true"
                android:scaleType="fitCenter"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toEndOf="@+id/pokemonType1"
                app:layout_constraintTop_toBottomOf="@+id/pokemonID"
                app:paletteCard="@{cardView}"
                app:paletteImage="@{pokemon.imageUrl}"

                tools:src="@drawable/charmander"/>

            <androidx.appcompat.widget.AppCompatTextView
                android:id="@+id/name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:elevation="4dp"
                android:gravity="center"
                android:maxLines="1"
                android:padding="12dp"
            android:text="@{pokemon.name}"
            android:textColor="@color/white"
            android:textSize="20sp"
            android:textStyle="bold"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            tools:text="Charmander"/>

        <androidx.appcompat.widget.AppCompatTextView
            android:id="@+id/pokemonID"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:layout_marginEnd="10dp"
            android:alpha="0.25"
            android:gravity="top"
            android:textColor="@color/black"
            android:textSize="18sp"
            android:textStyle="bold"
            android:text="@{@string/pk_id(pokemon.id)}"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintTop_toTopOf="parent"
            tools:text="#4"/>


        <androidx.cardview.widget.CardView
            android:id="@+id/pokemonType1"
            android:layout_width="70dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="10dp"
            android:layout_marginBottom="50dp"
            android:maxWidth="20dp"
            app:cardBackgroundColorType="@{pokemon.types[0].color}"
            app:cardCornerRadius="14dp"
            app:cardElevation="2dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            tools:cardBackgroundColor="@color/fire">


            <androidx.constraintlayout.widget.ConstraintLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:padding="2dp">
                <androidx.appcompat.widget.AppCompatImageView
                    android:id="@+id/imageView4"
                    android:layout_width="15dp"
                    android:layout_height="15dp"
                    android:layout_marginStart="5dp"
                    android:layout_marginTop="5dp"
                    android:layout_marginBottom="5dp"
                    android:background="@drawable/bg_image_pokemon_type"
                    android:scaleType="fitCenter"
                    android:src="@{pokemon.types[0].icon}"
                    app:layout_constraintBottom_toBottomOf="parent"
                    app:layout_constraintStart_toStartOf="parent"
                    app:layout_constraintTop_toTopOf="parent"
                    app:srcCompat="@drawable/bug"
                    tools:src="@drawable/fire"/>

                <androidx.appcompat.widget.AppCompatTextView
                    android:id="@+id/textView3"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_marginStart="5dp"
                    android:layout_marginTop="5dp"
                    android:layout_marginEnd="10dp"
                    android:layout_marginBottom="5dp"
                    android:gravity="center|center_vertical"
                    android:text="@{pokemon.types[0].name}"
                    android:textColor="@color/white"
                    android:textSize="14sp"
                    app:layout_constraintBottom_toBottomOf="parent"
                    app:layout_constraintEnd_toEndOf="parent"
                    app:layout_constraintStart_toEndOf="@+id/imageView4"
                    app:layout_constraintTop_toTopOf="parent"
                    tools:text="Fire"/>
            </androidx.constraintlayout.widget.ConstraintLayout>
        </androidx.cardview.widget.CardView>


        <androidx.cardview.widget.CardView
            android:id="@+id/pokemonType2"
            android:layout_width="70dp"
            android:layout_height="wrap_content"
            android:layout_marginStart="10dp"
            android:layout_marginTop="20dp"
            android:layout_marginBottom="20dp"
            android:maxWidth="20dp"
        app:cardBackgroundColorType="@{pokemon.types[1].color}"
        app:cardCornerRadius="14dp"
        app:cardElevation="2dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/pokemonType1"
        tools:cardBackgroundColor="@color/bug">


        <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:padding="2dp">

            <androidx.appcompat.widget.AppCompatImageView
                android:id="@+id/pokemonType2Icon"
                android:layout_width="18dp"
                android:layout_height="15dp"
                android:layout_marginStart="5dp"
                android:layout_marginTop="5dp"

                android:layout_marginBottom="5dp"
                android:background="@drawable/bg_image_pokemon_type"
                android:scaleType="fitCenter"
                android:src="@{pokemon.types[1].icon}"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toTopOf="parent"
                tools:src="@drawable/bug"/>

            <androidx.appcompat.widget.AppCompatTextView
                android:id="@+id/pokemonType2Label"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginStart="5dp"
                android:layout_marginTop="5dp"
                android:layout_marginEnd="10dp"
                android:layout_marginBottom="5dp"
                android:gravity="center|center_vertical"
                android:text="@{pokemon.types[1].name}"
                android:textColor="@color/white"
                android:textSize="14sp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toEndOf="@+id/
```

```
                              pokemonType2Icon"
                          app:layout_constraintTop_toTopOf="parent"
                          tools:text="Bug"/>
                    </androidx.constraintlayout.widget.ConstraintLayout>
                 </androidx.cardview.widget.CardView>


              </androidx.constraintlayout.widget.ConstraintLayout>

           </com.google.android.material.card.MaterialCardView>

</layout>
}
```

Listing 19: Binding property Region.

```
    /**/
    @JvmStatic
    @BindingAdapter("app:cardBackgroundColorType")
    fun setCardBackgroundColor(carview: CardView, resource: Int)
    {
        carview.setCardBackgroundColor(ContextCompat.getColor(carview.context,resource))
    }

    /**/
    @JvmStatic
    @BindingAdapter("paletteImage", "paletteCard")
    fun bindLoadImagePalette(view: AppCompatImageView, url: String, paletteCard:
        MaterialCardView) {
        Glide.with(view.context)
            .asBitmap()
            .load(url)
            .listener(object : RequestListener<Bitmap>
            {
                override fun onLoadFailed(
                    e: GlideException?,
                    model: Any?,
                    target: com.bumptech.glide.request.target.Target<Bitmap>?,
                    isFirstResource: Boolean
                ): Boolean {

                    Log.d("TAG", e?.message.toString())
                    return false
                }

                override fun onResourceReady(
                    resource: Bitmap?,
                    p1: Any?,
                    p2: com.bumptech.glide.request.target.Target<Bitmap>?,
                    p3: DataSource?,
                    p4: Boolean
                ): Boolean {
                    Log.d("TAG", "OnResourceReady")
                    if (resource != null) {
                        val p: Palette = Palette.from(resource).generate()

                        val rgb = p?.lightMutedSwatch?.rgb
                        if (rgb != null) {
                            paletteCard.setCardBackgroundColor(rgb)


                        }
                    }
                    return false
                }
            })
            .into(view)
    }
```

Listing 20: New Custom Bindings.

The function *bindLoadImagePalette* is responsible for binding an image loaded with Glide to an AppCompatImageView, and dynamically setting the background color of a MaterialCardView based on the palette of the loaded image.

The function has three parameters:

1. view: An instance of AppCompatImageView, which represents the image view where the image will be loaded.
2. url: A String parameter representing the URL of the image to be loaded.
3. paletteCard: An instance of MaterialCardView, which represents the card view whose background color will be set based on the palette of the loaded image.

Inside the function, Glide is used to load the image from the given URL (url) as a bitmap. The listener method is used to attach a custom implementation of RequestListener to handle events related to the image loading process.

The onLoadFailed method of the RequestListener is called if the image fails to load, and it logs the error message to the console.

The onResourceReady method of the RequestListener is called when the image is successfully loaded. Inside this method, the Palette library is used to generate a palette from the loaded image, and the lightMutedSwatch is used to get the RGB value of the dominant color in the light muted swatch. This RGB value is then used to set the background color of the MaterialCardView (paletteCard) using the setCardBackgroundColor method.

Finally, the into method is called to load the image into the AppCompatImageView (view) using Glide. The function returns false to indicate that the event is not consumed and should continue propagating to other listeners.

The function *setCardBackgroundColor* is responsible for setting the background color of a CardView based on a resource ID, which represents a color value.

The function has two parameters:

1. cardview: An instance of CardView, which represents the card view whose background color will be set.
2. resource: An Int parameter representing the resource ID of the color value to be used as the background color of the CardView.

Inside the function, the ContextCompat.getColor method is used to retrieve the color value associated with the given resource ID from the context of the CardView (carview.context). This color value is then passed to the setCardBackgroundColor method of the CardView (carview) to set the background color accordingly.

8.1 Pokemons Fragment and ViewModel

Create a new Fragment *PokemonsFragment* with ViewModel and go to `fragment_pokemons.xml` and create a RecyclerView.

```xml
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    tools:context=".ui.pokemon.PokemonsFragment">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/pokemonsRecyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="0dp"
        android:layout_marginBottom="0dp"
        android:clipToPadding="false"
        android:fadeScrollbars="false"
        android:padding="6dp"
        android:scrollbarStyle="outsideOverlay"
        android:scrollbars="vertical"
        app:layoutManager="GridLayoutManager"
        app:spanCount="2"
        tools:listitem="@layout/item_pokemon"
        tools:spanCount="2"
        />

</FrameLayout>
```

Listing 21: Pokemons Fragment Layout.

```kotlin
class PokemonsFragment : Fragment() {
    private var _binding: FragmentPokemonsBinding? = null
    private val viewModel: PokemonsViewModel by viewModels()

    private val binding get() = _binding!!

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _binding = FragmentPokemonsBinding.inflate(inflater, container, false)
        val root: View = binding.root
        return root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
```

```
            val region = checkNotNull(arguments?.getParcelable("region", PokemonRegion::
                class.java))
            viewModel.getListPokemonsByRegion(region).observe(viewLifecycleOwner, Observer {
                val pokemons: List<Pokemon> = it
                binding?.pokemonsRecyclerView?.adapter = PokemonsAdapter(pokemons,view.
                    context)
            })
        }

        override fun onDestroyView() {
            super.onDestroyView()
            _binding = null
        }

    }
```

Listing 22: Pokemons Fragment code.

```
    class PokemonsViewModel : ViewModel() {

    private val pokemonDomain = PokemonDomain()

    private lateinit var listPokemons : LiveData<List<Pokemon>>

    fun getListPokemonsByRegion(region: PokemonRegion): LiveData<List<Pokemon>>
    {
        listPokemons = pokemonDomain.getPokemonsByRegion(region)
        return listPokemons
    }
}
```

Listing 23: Pokemons ViewModel code.

With the Pokemon List feature already created, the desired behavior is to navigate to the Pokemon page and show only the Pokemon that belong to the selected region when clicking on a region.

To implement this use case, it is necessary to add the navigation behavior between these two fragments. For this, we will use the Jetpack Navigation Library, which is already included in the templates of this project.

8.2 Navigation Android

Jetpack Navigation is a set of libraries provided by Google as part of the Android Jetpack architecture components. It aims to simplify the implementation of navigation and UI flow in Android apps by providing a declarative and graphical way to define navigation paths between different screens or destinations within an app.

At the core of Jetpack Navigation is the Navigation Graph, which is an XML resource that defines the structure of the app's navigation flow. The Navigation Graph contains a collection of destinations, which represent the individual screens or fragments in the app, and the actions or connections between these destinations.

The key components of Jetpack Navigation are:

1. Navigation Graph: It is an XML resource that represents the navigation structure of the app. It defines the destinations (screens or fragments) in the app and the con-

nections between them. Destinations are defined with unique IDs and can have arguments, which are used to pass data between destinations. `res/navigation/mobile_navigation.xm`

2. NavHost: It is a container that hosts the fragments or screens defined in the Navigation Graph. It acts as a placeholder for the different destinations in the app and manages the navigation between them. The NavHost is typically defined in the app's XML layout file and can be either a FragmentContainerView or a NavHostFragment. `res/layouts/contain_main.xml`

3. NavController: It is an object that manages the navigation between destinations in the app. It is responsible for handling user interactions, such as button clicks or gestures, and navigating to the appropriate destinations based on the actions defined in the Navigation Graph. The NavController can be obtained using the Navigation.findNavController() method, passing in the NavHost as a parameter.

4. Navigation Actions: These are connections between destinations in the Navigation Graph that define the possible navigation paths in the app. Actions can be associated with user interactions, such as clicking a button, or triggered programmatically in code. Actions can also have arguments, which are used to pass data between destinations.

5. Navigation UI Components: Jetpack Navigation also provides UI components, such as NavigationView, BottomNavigationView, and Toolbar, that can be used to implement common navigation patterns, such as a navigation drawer, bottom navigation bar, or an app bar with back and up buttons. `res/layouts/activity_main.xml`

6. Safe Args: It is a code generation plugin provided by Jetpack Navigation that generates type-safe arguments classes for passing data between destinations. It eliminates the need for manually handling argument bundles or parsing data from Intent extras, making the code safer and less error-prone.

So, we need to modify the adapter so that when creating the cell for the region, when clicking on the cell, it navigates to the PokemonList page and passes the associated region object as an argument.

To avoid passing the NavController as an argument in the Adapter's constructor, a universal clickListener was created. Create a new package *ui/events* e add new kotlin File with following code:

```kotlin
typealias OnItemClickedListener = (arg: Any) -> Unit
```

Listing 24: Universal Click Listener.

In Kotlin, typealias is used to provide an alternative name for a function type or a complex type. It allows you to create a new name that can be used interchangeably with the original type. In this case, the typealias OnItemClickedListener is defined as a function type that takes an argument of type Any and returns Unit. The Any type is a generic type that can represent any type of object, while Unit is a special type in Kotlin that indicates a function that doesn't return any value, similar to void in Java. This typealias can be used to define a callback function that is invoked when an item is clicked in a UI component, such as a RecyclerView item or a button. The OnItemClickedListener can be used as a type for a function parameter or a property, making the code more concise and readable. When this callback is invoked, it will receive an argument of type Any, which can be cast or processed as needed within the function body.

In the `res/navigation/mobile_navigation` , we will add the Pokemons Fragment and Regions Fragment and add the navigation action between RegionsFragment and PokemonsFragment by simply dragging the mouse from the RegionsFragment white point to the PokemonFragment.

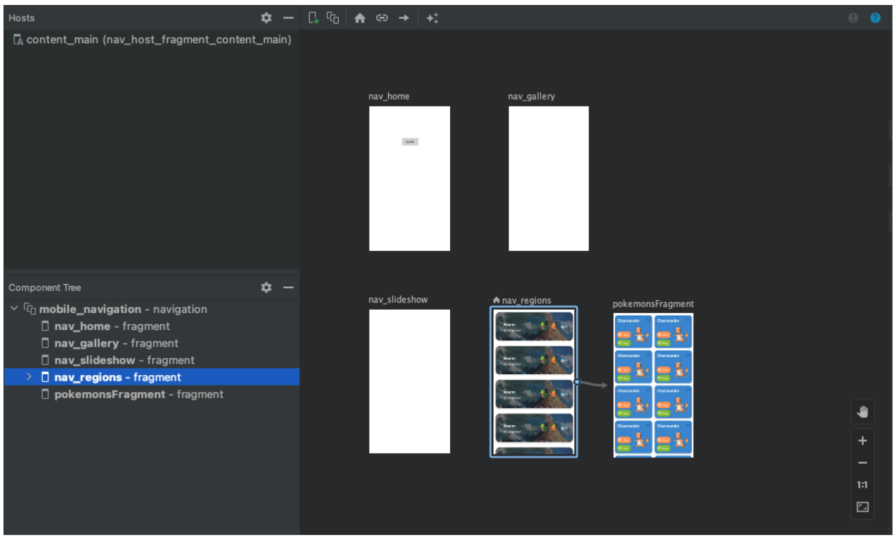

Figure 5: Navigation XML

8.3 Modify Region Fragment and Adapter to navigato to Pokemons List

```
    class RegionAdapter(
    private val pkRegionList: List<PokemonRegion>,
    private val itemClickedListener: OnItemClickedListener? = null,
    private val context: Context
) : RecyclerView.Adapter<RegionAdapter.ViewHolder>() {

    class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        private val viewBinding = ItemRegionBinding.bind(itemView)
        fun bindView(regionItem: PokemonRegion, itemClickedListener: OnItemClickedListener
            ?)
        {
            /*viewBinding.regionNameTextView.text = regionItem.name
 viewBinding.regionIdTextView.text = String.format(itemView.context.getString(R.string.
     pk_generations), regionItem.id)
            viewBinding.regionBgImage.setImageDrawable(ContextCompat.getDrawable(itemView.
                context,regionItem.bg))
            viewBinding.regionStartersImageView.setImageDrawable(ContextCompat.getDrawable(
                itemView.context,regionItem.starters))*/

            viewBinding.region = regionItem
            itemView.setOnClickListener{
                //Toast.makeText(itemView.context, String.format("Click in %s Region",
                    regionItem.name), Toast.LENGTH_LONG).show()
                itemClickedListener?.invoke(regionItem)
            };
        }
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        val inflater = LayoutInflater.from(context)
        val view = inflater.inflate(R.layout.item_region, parent, false)
        return ViewHolder(view)
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
        val item = pkRegionList[position]
        holder.bindView(item, itemClickedListener)
    }

    override fun getItemCount(): Int {
        return pkRegionList.size
    }

}
```

Listing 25: Region Adapter to suport navigation code.

```
    class RegionFragment : Fragment() {

    private var _regionViewModel: RegionViewModel? = null

    private var _binding: FragmentRegionBinding? = null

    private val binding get() = _binding!!

    private val regionViewModel get() = _regionViewModel!!

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        _regionViewModel =
            ViewModelProvider(this)[RegionViewModel::class.java]

        _binding = FragmentRegionBinding.inflate(inflater, container, false)

        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        regionViewModel.getRegions().observe(viewLifecycleOwner) {
            val regions: List<PokemonRegion> = it
            binding?.regionsRecyclerView?.adapter = RegionAdapter(regions,
                itemClickedListener = {region->

                    val bundle = bundleOf(
                        "region" to region
                    )

                    findNavController()
                        .navigate(
                            R.id.action_nav_regions_to_pokemonsFragment,
                        bundle,
                        null
                    )
            },view.context)
        }
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }

}
```

Listing 26: Region Fragment to suport navigation code.

The regionsRecyclerView is associated with an adapter called RegionAdapter, which is initialized with a list of regions as its data source (regions). The adapter also takes an itemClickedListener as a parameter, which is a lambda function that defines the behavior when an item in the RecyclerView is clicked.

Inside the itemClickedListener lambda function, the clicked region object is passed as an argument (region). Then, a Bundle is created with the clicked region object as its data, using the bundleOf() function.

Next, the findNavController() function is used to obtain the NavController associated with the current Fragment or Activity. The navigate() function is then called on the NavController to perform the navigation action.

The `R.id.action_nav_regions_to_pokemonsFragment` parameter specifies the navigation action to be performed, which navigates from the current fragment (`nav_regions`) to the PokemonsFragment.

The bundle is passed as an argument to the navigate() function, so that it can be accessed in the destination fragment (PokemonsFragment) using arguments.

To obtain the region that was passed as a parameter in PokemonsFragment, we added the following line of code in onViewCreated.

```
    val region = checkNotNull(arguments?.getParcelable("region", PokemonRegion::class.java)
        )
```

Listing 27: Pokemons Fragment to suport navigation code.

9 Pokemon Detail

10 Extras

10.1 Add Search in Pokemon List

10.2 Add Filters in Pokemon List

A Model Classes

```
    data class Pokemon(
        var id: Int,
        var name:String,
        var imageUrl: String,
        var region: PokemonRegion?,
        var types: List<PokemonType>
    )
```

Listing 28: Pokemon Data Class.

```
    data class PokemonDetail(
        var pokemon: Pokemon,
        var description: String,
        var types:List<PokemonType>?,
        var weight:Double?,
        var height: Double?,
        var stats: PokemonStats?,
        var evolution: List<PokemonEvolution>?
    )
```

Listing 29: Pokemon Detail Data Class.

```
    data class PokemonRegion(
        var id:Int,
        var name: String,
        @DrawableRes val bg: Int,
        @DrawableRes val starters: Int
    )
```

Listing 30: Pokemon Region Data Class.

```
    data class PokemonType(
        var id: Int,
        var name:String,
        @DrawableRes val icon: Int,
        @ColorRes val color: Int)
)
```

Listing 31: Pokemon Type Data Class.

```
    data class PokemonStats(
        var id: Int = 0,
        var hp:Int= Random.nextInt(1,300),
        var attack:Int= Random.nextInt(1,300),
        var defense:Int= Random.nextInt(1,300),
        var specialDefense: Int= Random.nextInt(1,300),
        var speed: Int= Random.nextInt(1,300),
        var exp:Int= Random.nextInt(1,300),
        var maxHp: Int= 300,
        var maxSpeed: Int= 300,
        var maxAttack: Int= 300,
        var maxDefense: Int= 300,
        var maxSpecialDefense: Int= 300,
        var maxExp: Int= 300)
    )
```

Listing 32: Pokemon Stats Data Class.

```
    data class PokemonEvolution(
        var id: Int,
        var pokemon: Pokemon,
        var isBaby: Boolean,
        var minLevel: Int?,
        var item: String?,
    var minHappiness: Int?,
    var time: String?
)
```

Listing 33: Pokemon Stats Data Class.

B Data Mock Object

```
object PokemonMockData {

private var pokemonDetailDescription: String = "Pokem ipsum dolor " +
        "sit amet Crustle Grotle" +
        " Dragonair Palkia Shellder Terrakion. " +
        "Hive Badge Pokeball Spinda Seedot James Vullaby " +
        "Helix Fossil. Water Gun Professor Oak Marowak Spearow " +
        "Dunsparce Chimchar Nidorino." +
        " Silver Azumarill Tyranitar Trubbish " +
        "Fighting sunt in culpa qui officia Mothim. " +
        "Celadon City Mantine Clefable Piplup Scizor " +
        "excepteur sint occaecat cupidatat non proident Terrakion."


var regions = listOf<PokemonRegion>(
    PokemonRegion(1, "Kanto", R.drawable.bg_kanto, R.drawable.pk_kanto),
    PokemonRegion(2, "Johto", R.drawable.bg_johto, R.drawable.pk_johto),
    PokemonRegion(3, "Hoenn", R.drawable.bg_hoenn, R.drawable.pk_hoenn),
    PokemonRegion(4, "Sinnoh", R.drawable.bg_sinnoh, R.drawable.pk_sinnoh),
    PokemonRegion(5, "Unova", R.drawable.bg_unova, R.drawable.pk_unova),
    PokemonRegion(6, "Kalos", R.drawable.bg_kalos, R.drawable.pk_kalos),
    PokemonRegion(7, "Alola", R.drawable.bg_alola, R.drawable.pk_alola),
    PokemonRegion(8, "Galar", R.drawable.bg_galar, R.drawable.pk_galar),
    )

var pokemonTypeMock= listOf<PokemonType>(
    PokemonType(1,"water", R.drawable.water, R.color.water),
    PokemonType(2,"fire", R.drawable.fire, R.color.fire),
    PokemonType(3,"bug", R.drawable.bug, R.color.bug),
    PokemonType(4,"ghost", R.drawable.ghost, R.color.ghost),
    PokemonType(5,"grass", R.drawable.grass, R.color.grass),
    PokemonType(6,"ground", R.drawable.ground, R.color.ground),
    PokemonType(7,"rock", R.drawable.rock, R.color.rock),
    PokemonType(8,"dark", R.drawable.dark, R.color.dark),
    PokemonType(9,"dragon", R.drawable.dragon, R.color.dragon),
    PokemonType(10,"electric", R.drawable.electric, R.color.electric),
    PokemonType(11,"fairy", R.drawable.fairy, R.color.fairy),
    PokemonType(12,"fighting", R.drawable.fighting, R.color.fighting),
    PokemonType(13,"ice", R.drawable.ice, R.color.ice),
    PokemonType(14,"normal", R.drawable.normal, R.color.normal),
    PokemonType(15,"psychic", R.drawable.psychic, R.color.psychic),
    PokemonType(16,"flying", R.drawable.flying, R.color.flying),
    PokemonType(17,"poison", R.drawable.poison, R.color.poison),
    PokemonType(18,"steel", R.drawable.steel, R.color.steel)
)

var pokemons = listOf(
    Pokemon(1,
        "bulbasaur",
        "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                "/sprites/pokemon/other/official-artwork/1.png",
            regions[0], listOf(pokemonTypeMock[1], pokemonTypeMock[10])

        ),
        Pokemon(4,
            "charmander",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/1.png",
            regions[0], pokemonTypeMock.take(2)
        ),
        Pokemon(6,
            "squirtle",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/7.png",
            regions[0],
            pokemonTypeMock.take(2)
        ),
        Pokemon(10,
            "caterpie",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/10.png",
            regions[0],
            pokemonTypeMock.take(2)),
        Pokemon(13,
            "weedle",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/13.png",
            regions[0],
            pokemonTypeMock.take(2)),
        Pokemon(16,
            "pidgey",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/16.png",
            regions[0],
            pokemonTypeMock.take(2)),
        Pokemon(19,
            "rattata",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/19.png",
            regions[0],
            pokemonTypeMock.take(2)),
        Pokemon(21,
            "spearow",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/21.png",
            regions[0],
            pokemonTypeMock.take(2)),
        Pokemon(23,
            "ekans",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/23.png",
            regions[0],
            pokemonTypeMock.take(2)),
        Pokemon(25,
            "pikachu",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/25.png",
            regions[0],
            pokemonTypeMock.take(2)),
        Pokemon(27,
            "sandshrew",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/27.png",
            regions[0],
            pokemonTypeMock.take(2)),
        Pokemon(29,
            "nidoran",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/29.png",
            regions[0],
            pokemonTypeMock.take(2)),
        Pokemon(35,
            "clefairy",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/35.png",
            regions[0],
            pokemonTypeMock.take(2)),
        Pokemon(37,
            "vulpix",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/37.png",
            regions[0],
            pokemonTypeMock.take(2)),
        Pokemon(39,
            "jigglypuff",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/39.png",
            regions[0],
            pokemonTypeMock.take(2)),
        Pokemon(41,
            "zubat",
            "https://raw.githubusercontent.com/PokeAPI/sprites/master" +
                    "/sprites/pokemon/other/official-artwork/41.png",
            regions[0],
            pokemonTypeMock.take(2)),

        )

    var pokemonDetail = pokemons.map {
        PokemonDetail(
            it,
            pokemonDetailDescription,
            pokemonTypeMock.asSequence().shuffled().take(1).toList(),
            ( Random.nextDouble(20.0,50.0) * 100.0).roundToInt() / 100.0,
            (Random.nextDouble(0.20, 2.0) * 100.0).roundToInt() / 100.0,
            PokemonStats(),
            generateSequence {
                PokemonEvolution(1, pokemons.random(), false,
                    0,"", 0, "")
            }.take(Random.nextInt(1,3)).toList()
        )
    }

}
```

Listing 34: PokemonMockData Object.