BAB X
VIEW MODEL

A. THEORITICAL BASIS

10.1. ViewModel

ViewModel is part of the Android Architecture Component. ViewModel also has a function as a substitute for Loader. ViewModel class designed to save and manage UI. ViewModel class bring through to keep the data when configuration changes are like screen rotation.

You surely already know how the Android framework manages the lifecycle of the UI controller like Activity or Fragment. The UI controller functions to display data to the UI, provide action against user actions or handle information system communication such as permission requests or permissions. In addition, the UI controller also has the ability to retain data. As in the previous module, when you get a data from a certain action such as volume data, then to maintain it you need to do something. What is that? You need to save the data by calling onSaveInstancestate () and displaying the data in onCreate with the help of a bundle.

The above can be done because the data that you maintain is small. But what if the data you are keep are large, like a list of users or bitmap images? Of course the UI controller management will be problematic.

10.2. ViewModel Implementation

Architecture Components provide ViewModel class to help UI Controller prepare data that will display to UI. ViewModel object will always keep during configuration change. So that, the data it has will be immediately available for the next Activity or Fragment. For example if you need to display user data in your application, make

sure that you have saved user data to the ViewModel class. The following is an example of the code:

```
public class MyViewModel extends ViewModel {
    private MutableLiveData<List<User>> users;
    public LiveData<List<User>> getUsers() {
        if (users == null) {
            users = new MutableLiveData<List<User>>();
            loadUsers();
        }
        return users;
    }

    private void loadUsers() {
        // Melakukan proses asynchronous untuk mendapatkan data pengguna.
    }
}
```

Then you can access the user list from Activity as follows:

```
public class MyActivity extends AppCompatActivity {
    public void onCreate(Bundle savedInstanceState) {
        // Membuat ViewModel saat Activity berada di metode onCreate().
        MyViewModel model = new ViewModelProvider(this, new ViewModelProvider.NewInstanceFactory()).
    get(MyViewModel.class);
        model.getUsers().observe(this, users -> {
            // Ketika ada data users, maka UI secara otomatis terupdate.
        });
    }
}
```

In the code above when an Activity is reloaded, the Activity will use the same ViewModel instance when the Activity was first formed. When Activity is in process, the UI controller will call the onCleared method where the ViewModel object can be cleared.

AndroidViewModel is a subclass of ViewModel that has a similar function. If you need **context** in the ViewModel pattern, you can use AndroidViewModel, because we can call **getApplicationContext** in it.

B. SPECIAL INSTRUCTION PURPOSE

At this meeting, the objectives that must be achieved include

1. Students understand the implementation of ViewHolder to keep data during configuration changes

2. Students understand the use of ViewHolder to load weather data

C. TOOLS AND MATERIAL

Android application development requires equipment both hardware and software. The hardwares that used to build android applications are as follows.

1. Laptop / Personal Komputer with minimum spesification.
   - √ Processor with minimum Fifth Generation Intel Core i3, or AMD A8 processor.
   - √ 3 GB of minimum RAM, recommended 8 GB of RAM,
   - √ Minimum available disk space is 2 GB,
   - √ 1280 x 800 minimum screen resolution
2. *Smartphone* android that have minimum version Cream Sandwich (Android 4.0)
3. Cable data

While the software that must be installed on a Laptop / PC, are

1. Windows operating system (recommended to use windows 10), Linux (recommended to use Linux Mint), Macintosh (recommended to use version 10, yosemite)
2. Java Development Kit (JDK) 8 atau newest version
3. Android Studio

D. WORK STEP

10.3. Create a project that applies ViewHolder in Android Studio

1. Create New Project in Android Studio

Project Name	**MyViewHolder**
Target & SDK Target Minimum	**Phone and Table, API level 21**
Activity Type	**Empty Activity**
Activity Name	**MainActivity**

Use AndroidX Artifacts	**True**

2. After the project is formed, add some string names as in the following code.

```
<resources>
    <string name="app_name">MyViewModel</string>
    <string name="description">Deskripsi</string>
    <string name="temperature">Temperatur</string>
    <string name="name">Nama Kota</string>
    <string name="insert_city">Masukkan id kota misal Semarang = 1627896</string>
    <string name="search">Cari</string>
    <string name="id_pwt_wnb_smg">1630328,1621395,1627896</string>
</resources>
```

3. Make values resource file with name dimens, and add some code

```
<resources>
    <dimen name="vertical_margin">16dp</dimen>
    <dimen name="horizontal_margin">16dp</dimen>
</resources>
```

4. Next on the ***activity_main*** add the following code.

```
<RelativeLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/horizontal_margin"
    tools:context=".MainActivity">

    <ProgressBar
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        style="?android:attr/progressBarStyle"
        android:layout_centerInParent="true"
        android:visibility="gone"
        android:id="@+id/progressBar"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="horizontal"
        android:weightSum="1"
        android:id="@+id/inputText">

        <EditText
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="0.8"
            android:hint="@string/insert_city"
            android:text="@string/id_pwt_wnb_smg"
            android:id="@+id/editCity"/>

```

```
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="0.2"
            android:text="@string/search"
            android:id="@+id/btnCity"/>

    </LinearLayout>

    <androidx.recyclerview.widget.RecyclerView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/inputText"
        android:layout_marginTop="1dp"
        android:id="@+id/recyclerView"/>

</RelativeLayout>
```

5. In the gradle level module (build.gradle:module) add CircleImageView Library with adding this following line code

```
implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0-alpha04'
annotationProcessor 'androidx.lifecycle:lifecycle-compiler:2.2.0-alpha04'
implementation 'com.loopj.android:android-async-http:1.4.9'
implementation 'androidx.recyclerview:recyclerview:1.0.0'
```

6. Next, create a new layout that will be used to display information for each item. Right-click on the **layout directory** → **New** → **Layout resource file** and name it **weather_items**. After that, change the code in the class to be like this:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="@dimen/horizontal_margin"
        android:paddingTop="@dimen/vertical_margin"
        android:paddingRight="@dimen/horizontal_margin"
        android:text="@string/name"
        android:id="@+id/textKota"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingLeft="@dimen/horizontal_margin"
        android:paddingRight="@dimen/horizontal_margin"
        android:text="@string/temperature"
        android:id="@+id/textTemp"/>

```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingRight="@dimen/horizontal_margin"
        android:paddingLeft="@dimen/horizontal_margin"
        android:paddingBottom="@dimen/vertical_margin"
        android:text="@string/description"
        android:id="@+id/textDesc"/>

</LinearLayout>
```

7. Next, create a model class to load an informatio at each item. **Right click** on the main package → **new** → **Java Class**, then give the name **WeatherItems**.

```
public class WeatherItems {

    private int id;
    private String name;
    private String currentWeather;
    private String description;
    private String temperature;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getCurrentWeather() {
        return currentWeather;
    }

    public void setCurrentWeather(String currentWeather) {
        this.currentWeather = currentWeather;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
```

```
    }

    public String getTemperature() {
        return temperature;
    }

    public void setTemperature(String temperature) {
        this.temperature = temperature;
    }
}
```

8. Now, create new class as adapter from **RecyclerView**, give the name **WeatherAdapter**. Add inheritance from class **RecyclerView.Adapter<>**. If any error, click one the red line or **Alt + Enter.**

```
public class WeatherAdapter extends RecyclerView.Adapter<WeatherAdapter.WeatherViewHolder> {
}
```

after that, complete the code to be as follows.

```
public class WeatherAdapter extends RecyclerView.Adapter<WeatherAdapter.WeatherViewHolder> {
    private ArrayList<WeatherItems> mData = new ArrayList<>();

    public void setData(ArrayList<WeatherItems> items){
        mData.clear();
        mData.addAll(items);
        notifyDataSetChanged();
    }

    @NonNull
    @Override
    public WeatherViewHolder onCreateViewHolder(@NonNull ViewGroup viewGroup, int position) {
        View mView = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.weather_items, viewGroup,
false);
        return new WeatherViewHolder(mView);
    }

    @Override
    public void onBindViewHolder(@NonNull WeatherAdapter.WeatherViewHolder holder, int position) {
        holder.bind(mData.get(position));
    }

    @Override
    public int getItemCount() {
        return mData.size();
    }

    public class WeatherViewHolder extends RecyclerView.ViewHolder {
        TextView textViewNamaKota;
        TextView textViewTemperature;
        TextView textViewDescription;
```

```
31.
32.         public WeatherViewHolder(@NonNull View itemView){
33.             super(itemView);
34.             textViewNamaKota = itemView.findViewById(R.id.textKota);
35.             textViewTemperature = itemView.findViewById(R.id.textTemp);
36.             textViewDescription = itemView.findViewById(R.id.textDesc);
37.         }
38.
39.         public void bind(WeatherItems weatherItems) {
40.             textViewNamaKota.setText(weatherItems.getName());
41.             textViewTemperature.setText(weatherItems.getTemperature());
42.             textViewDescription.setText(weatherItems.getDescription());
43.         }
44.     }
45. }
```

9. After setting up adapters and other components, create a new class as a ViewModel class, and name it **MainViewModel**. After the class is formed, add a class instance of **ViewModel**:

```
1.  public class MainViewModel extends ViewModel {
2.      private static final String API_KEY = "ISI SESUAI API_KEY ANDA";
3.      private MutableLiveData<ArrayList<WeatherItems>> listWeathers = new MutableLiveData<>();
4.  }
```

Fill **API_KEY** with your own API. You can get **API_KEY** with register in https://openweathermap.org/api. After that, add some codes in **setWeather()** method to request API.

```
1.  public class MainViewModel extends ViewModel {
2.      private static final String API_KEY = "6393040e75c38e2deaf018f4ead33680";
3.      private MutableLiveData<ArrayList<WeatherItems>> listWeathers = new MutableLiveData<>();
4.
5.      void setWeather(final String cities) {
6.          AsyncHttpClient client = new AsyncHttpClient();
7.          final ArrayList<WeatherItems> listItems = new ArrayList<>();
8.          String url = "https://api.openweathermap.org/data/2.5/group?id=" + cities + "&units=metric&appid=" +
    API_KEY;
9.
10.         client.get(url, new AsyncHttpResponseHandler() {
11.             @Override
12.             public void onSuccess(int statusCode, Header[] headers, byte[] responseBody) {
13.                 try {
14.                     String result = new String(responseBody);
15.                     JSONObject responseObject = new JSONObject(result);
16.                     JSONArray list = responseObject.getJSONArray("list");
17.                     for (int i = 0; i < list.length(); i++) {
18.                         JSONObject weather = list.getJSONObject(i);
19.                         WeatherItems weatherItems = new WeatherItems();
20.                         weatherItems.setId(weather.getInt("id"));
21.                         weatherItems.setName(weather.getString("name"));
```

```
                    weatherItems.setCurrentWeather(weather.getJSONArray("weather").getJSONObject(0).getString("main"));
                    weatherItems.setDescription(weather.getJSONArray("weather").getJSONObject(0).getString("description"));
                    double tempInCelcius = weather.getJSONObject("main").getDouble("temp");
                    weatherItems.setTemperature(new DecimalFormat("##.##").format(tempInCelcius));
                    listItems.add(weatherItems);
                }
                listWeathers.postValue(listItems);
            } catch (Exception e) {
                Log.d("Exception", e.getMessage());
            }
        }

        @Override
        public void onFailure(int statusCode, Header[] headers, byte[] responseBody, Throwable error) {
            Log.d("onFailure", error.getMessage());
        }
    });
}

LiveData<ArrayList<WeatherItems>> getWeathers() {
    return listWeathers;
}
}
```

10. Next on **MainActivity** complete the existing code so that it becomes like the following example:

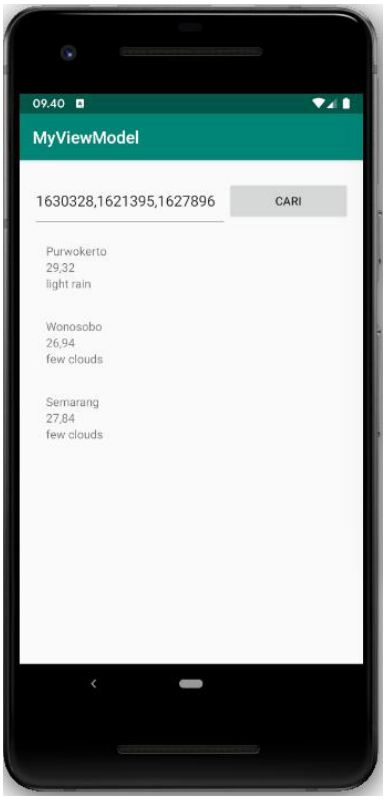
```
public class MainActivity extends AppCompatActivity {
    private WeatherAdapter adapter;
    private EditText edtCity;
    private ProgressBar progressBar;
    private Button btnCity;
    private MainViewModel mainViewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        edtCity = findViewById(R.id.editCity);
        progressBar = findViewById(R.id.progressBar);
        btnCity = findViewById(R.id.btnCity);

        RecyclerView recyclerView = findViewById(R.id.recyclerView);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        adapter = new WeatherAdapter();
        adapter.notifyDataSetChanged();
        recyclerView.setAdapter(adapter);

        mainViewModel = new ViewModelProvider(this, new ViewModelProvider.NewInstanceFactory()).get(MainViewModel.class);

        btnCity.setOnClickListener(new View.OnClickListener() {
```

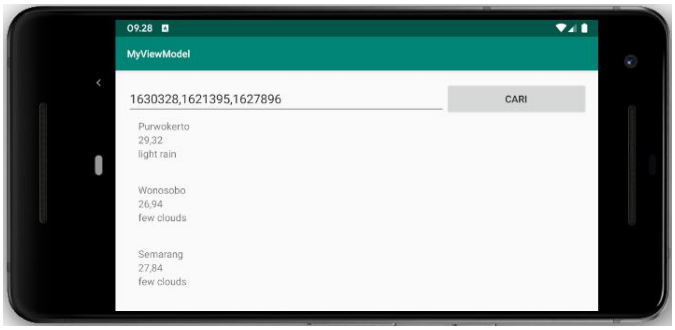
```
        @Override
        public void onClick(View view) {
            String city = edtCity.getText().toString();

            if(TextUtils.isEmpty(city)) return;

            mainViewModel.setWeather(city);
            showLoading(true);
        }
    });

    mainViewModel.getWeathers().observe(this, new Observer<ArrayList<WeatherItems>>() {
        @Override
        public void onChanged(ArrayList<WeatherItems> weatherItems) {
            if(weatherItems != null){
                adapter.setData(weatherItems);
                showLoading(false);
            }
        }
    });
}

private void showLoading(Boolean state){
    if(state){
        progressBar.setVisibility(View.VISIBLE);
    }else {
        progressBar.setVisibility(View.GONE);
    }
}
}
```

11. Because we are going to connect internet on an Android device, we need to add permissions to the application. Please add one line of permission to the **AndroidManifest.xml** file as follows:

```
1.  <uses-permission android:name="android.permission.INTERNET"/>
```

12. It's done! Run the application and see your application will load the weather data contained on the Android device. Weather data that displayed are city, temperature, and weather condition. Activity not destroyed, although the orientation changes


09.40
MyViewModel
1630328,1621395,1627896
CARI
Purwokerto
29,32
light rain
Wonosobo
26,94
few clouds
Semarang
27,84
few clouds


09.28
MyViewModel
1630328,1621395,1627896
CARI
Purwokerto
29,32
light rain
Wonosobo
26,94
few clouds
Semarang
27,84
few clouds