AMD-07: MVVM-RecyclerView-Room-SharedPreferences

Applications for mobile devices & Course 2019-2020

Jordi Mateo Fornés jordi.mateo@udl.cat

- Dr. Jordi Mateo Fornés
- **Office**:
  - Office A.12 (Campus Igualada)
  - Office 3.08 (EPS Lleida)
- **Email**: jordi.mateo@udl.cat
- Doubts
  - During class
  - After class
  - Email
    - Topic: [AMD]: XXXXXXXXX

Recycler View

Recycler View Overview

Information extracted from: Android Developers - Recycler View.

- The **recycler view** is a widget that manages together several components to display our data.
- It is more advanced than *List View* or *Grid View*.
- Each element inside can be defined using a different layout.

Recycler View Holders

- The views in the list are represented by *view holder* objects (extends **RecyclerView.ViewHolder**).
- The main function of a holder is to display a single item in the view.
- **NOTE**: Recycler View only create the view holders needed to display the **on-screen** portion of the dynamic content. When the user scrolls it takes off-screen views and rebinds them with new data.

Recycler View Adapter

- The view holder is managed by *adapters* objects (extends *RecyclerView.Adapter*).
- Adapter binds data to view holder and creates view holders on demand. (*onBindViewHolder()*).
- When the item displayed changes, you can notify the adapter calling (*RecyclerView.Adapter.notify()*).

How it works

As the user scrolls the list, the RecyclerView creates new view holders on demand. It also saves the view holders which have scrolled off-screen, so they can be **reused**.

If the user switches the direction they were scrolling, the view holders which were scrolled off the screen can be brought right back.

On the other hand, if the user keeps scrolling in the same direction, the view holders which have been off-screen the longest can be re-bound to new data.

The view holder does not need to be created or have its view inflated; instead, the app just updates the view's contents to match the new item it was bound to.

Advantages

- Recycling views.
- Configurable distribution of the views.
- Automatic animations.
- Element separator. (Item decorator).
- Work together with CoordinationLayout.

Item Decorator

- A Decorator pattern can be used to attach additional responsibilities to an object either statically or dynamically. A Decorator provides an enhanced interface to the original object.
- Item Decorations allow us to draw on all 4 sides of the item in an Adapter based data-set.
- Dividers, spaces or special effects between items are delegated to Item Decorator.

Hands-On

Hands-On: Event App

- We are going to extend the functions of our Event App with a *List* of events.

All the code is available in the Github — Code

Hands-On: Room

- Add dependencies
- Entity: Room generates all the code required to create an SQLite table for this object and all the fields.
- PrimaryKey(autoGenerate = true): Make the member variable (id) into an auto-incrementing primary key.
- ColumnInfo: Annotation to define the name of the column.
- Ignore: Annotation to tell Room to ignore the variable.

More information in Developers.

Hands-On: Entity

```
@Entity(tableName = "event_table")
...
@PrimaryKey(autoGenerate = true)
    @SerializedName("id")
    private int id;
    @SerializedName("userId")
    private int userId;
    @SerializedName("start")
    private String start;
...
```

Hands-On: DAO

- We are going to code an interface to define all the DB operations for our entity.
- We are going to use *DAO* annotation.
- We use annotations for CRUD operations: *Insert*, *Query*,*Update*,*Delete*.
- We are going to return an instance of LiveData to notify of changes in real-time.

More information in Developers.

Hands-On: Database

- DB: Database holder and serves as the relational data. We use *Database*.
- We are going to extend *RoomDatabase*.
- We are going to create the EventDatabase as a static **singleton** with the *database builder*.
- Callback to the database builder to populate our database in the onCreate method. We init some events.

More information in Developers.

Hands-On: Repository

- Abstract the data layer from the rest of the app and mediates with different data sources such as a local or remote database. Moreover, we want to provide a clean API to the viewModel.
- Room queries can not be executed on the main thread, we need to us AsyncTasks.
- LiveData is fetched automatically in the worker thread.

Hands-On: ViewModel

- Is the gateway between the UI controller and the repository.
- We extend *AndroidViewModel*, to handle the application context, to store current configuration *onChanges()*.
- LiveData filtering using Transformation.switchMap.

Hands-On: Filter by user

```
private final MutableLiveData<String> userId =
new MutableLiveData<>();
events  = Transformations.switchMap(userId, trigger)
public void setUserId(String id) {
        userId.setValue(id);
}
```

- It creates **events** that reacts on changes of trigger *LiveData*, applies the given function to a new value of trigger **String id**.

Hands-On: Trigger function

```
trigger = new Function<String, LiveData<List<Event>>>() {
      @Override
        public LiveData<List<Event>> apply(String id) {
          if (id == null || id.equals("")) {
              events = repository.getEvents();
          }else{
              events = repository.getEvents(Integer.parseInt(id));
          }
        return events;
        }
      });
```

Hands-On: Recycler View

- Prepare the layout for the items. Check *items_events_list.xml*. We introduce the widget CardView and RatingBar.
- Optional: Here, we can use Android **Databindings**.
- Prepare the activity view layout with the RecyclerView *activity_event_list.xml*.

```
<androidx.recyclerview.widget.RecyclerView
        android:id="@+id/activityMainRcyMain"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:scrollbars="vertical"
        app:layoutManager="androidx.recyclerview.widget.LinearLayo
        tools:listitem="@layout/item_events_list" />
```

Hands-On: DiffCallback

- Utility class to calculate the difference between two lists and output a list of update operations that converts the first list into the second one.
- We need to overwrite equals function in Event class.

Hands-On: Adapter

- We extend ListAdapter to calculate async the differences between the old data set and the new one (background thread).
- We add a ViewHolder as an inner class and implement onCreateViewHolder, onBindViewHolder to inflate it and populate it with data.
- Data binding is done in Holder class.
- We add a click listener to manage the edition of the item in another activity.

Hands-On: Activity

- Observe changes on LiveData.
- Define the recycler view and sets the adapter.
- onSwiped method to delete an event.
- insert button to create a new event.
- uses shared preferences to obtain the current_user id.

Hands-On: Shared Preferences

- We create PreferenceProvider.
- We init the SharedPreference. Create App.class and extends *Application*.
- Update AndroidManifest: **android:name=".App"**.
- **.put{X}(*key*,*value*)** where {X} can be String, int,... to store a value in the preferences.
- **.get{X}(*key*,*default*)** where {X} can be String, int,... to retrieve a value from the preferences.

Activity (AC-03)

Activity (AC-03)

- The main goal is using this methodology into your GitHub events current project.

OR:

- Make a pull request to my repository correcting one of these items:
  - Add a view to introduce the current_user id, store the id in the shared preferences and go to the list view.
  - Add admin and user role, if current_user is admin can modify all events. If not only the events created by him/her.
  - Improve events using Date instead of strings and check form errors in creating an event.
  - Add event categories and make a category filter.
  - Filter events by date, incoming, done (past events)...
  - Improve the code.

That is all

www — jordimateofornes.com

github — github.com/JordiMateo

twitter — @MatForJordi

gdc — Distributed computation group