GHENT UNIVERSITY

Lab 5: Item list with RecyclerView

Design and Development of Mobile Applications
E. De Coninck, S. Leroux, P. Simoens
2019-2020

In this session we will implement the list of notifications on the traffic in and around Ghent. The list will display each notification's type and date, as shown in Figure 1. The details of a traffic notification are shown when the user touches the corresponding list item. We will fetch remote data from the Open Data Tank of the Ghent city council.

You will learn the following concepts:

- Using API and Training documentation on `https://developer.android.com`
- Building lists using RecyclerView with custom Adapter and ViewHolders
- Building layouts in XML: RecyclerView
- Handling list item click events
- Asynchronous requests using Volley library
- Using Repository to create fetching strategy

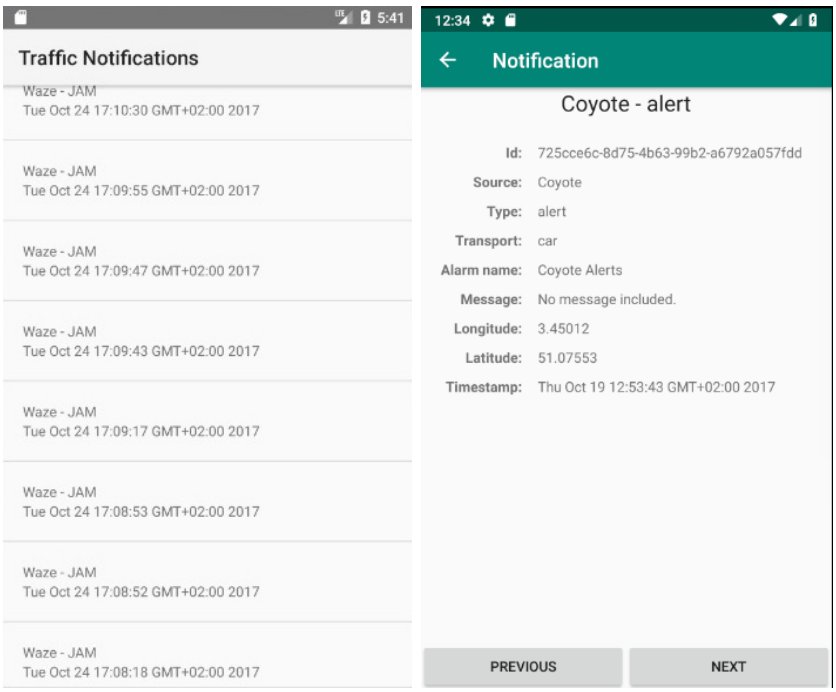

Figure 1: Example layout of the Traffic Notifications Application with a list of items using the RecyclerView.

1 Project set-up

- You continue from your solution of the previous session.
- Enable the ability to fetch remote data with the correct permissions. (`https://developer.android.com/training/basics/network-ops/connecting`)

2 Displaying lists using RecyclerView

In this session, we will focus on the MainFragment and the repository of our appication. The main Fragment shows a list of notification items using a `RecyclerView`, an `Adapter` and one or more `ViewHolder(s)`.

2.1 RecyclerView, Adapter and ViewHolder

- Add a RecyclerView to the MainFragment layout file and obtain a handle to this object in the Fragment's class. A RecyclerView requires a layout manager to determine the placement of the items. Add a LinearLayoutManager to the RecyclerView in code or in xml. The training documentation on creating list cards can help you through this part (see `https://developer.android.com/training/material/lists-cards.html`).
- Each recyclerview needs an adapter that reads from an underlying data source (a file, a database) and knows how to lay out individual list items. Create a new class which extends RecyclerView.Adapter<T> where T is an inner class of the adapter which extends RecyclerView.ViewHolder. The adapter provides access to the items in your data set, creates views for items, and replaces the content of some of the views with new data items when the original item is no longer visible. Implement the new adapter and override all required methods.

The ViewHolder does one thing: it holds on to a View. A typical implementation of a ViewHolder receives an inflated View or a binding in the constructor. The ViewHolder typically has a public method to bind with a data item. In our case, the item is of type TrafficNotification.

- Create a new XML layout file **traffic_notification.xml** in the layout folder and add 2 text fields, one for the title (`toString()` of a notification) and one for the date, and bind these fields to fields of a specific notification (variable). This file will provide the layout of a single list item.
- Implement your ViewHolder so it accepts a `TrafficNotificationBinding` in the constructor and pass the root view of the binding to the constructor of the super class. Also provide a method that accepts a `TrafficNotification` and that uses databinding to set the text of the textviews.
- Implement your adapter to make use of your ViewHolder class. Provide a constructor that requires a List of TrafficNotifcations and implement the `onCreateViewHolder(...)` and `onBindViewHolder(...)` methods. The `onCreateViewHolder(...)` method is only called when new views have to be created while `onBindViewHolder(...)` is called every time a new item scrolls into position.
- Finalize the list view by attaching the newly implemented adapter to your RecyclerView in the main Fragment. You will see that items are not clearly divided in the current list view. Clearly dividing list items can be accomplished with DividerItemDecoration.

2.2 Responding to presses

The `RecyclerView` does not handle click events and is not responsible to inform the user of list item clicks. Handling touch events is mostly up to you. There are a lot of methods to handle the item clicks. Since

each `ViewHolder`'s View is inflated using data binding with a specific notification we can use a handler class to handle the click events of the items.

- To forward click events to an interested listener we can create an interface `OnNotificationClickListener` with a single method `onNotificationClick(TrafficNotification n)`. Accept this listener as a parameter in your Adapter's constructor. This listener can be used as a handler for the ViewHolder's view clicks. Set this handler in the ViewHolder's consturctor.
- Implement `OnNotificationClickListener` in your main Fragment to be able to respond to notification clicks. When a notification is clicked this item is set as the selected item in the ViewModel and the detail Fragments is launched using the Navigation Architecture Component of the last session.

3 Fetching remote data

In the last part of this exercise, we will populate the notification list with remote data using an asynchronous request using Volley. Volley performs the network request asynchronously so there is no need to use an AsyncTask.

3.1 Asynchronous Volley request

- To use the Volley in your project, you will need to add it as a dependency of your app's module:
- Create a new Volley request queue in your repository's constructor.
- Launch a `JsonObjectRequest` to `http://users.ugent.be/~ejodconi/verkeersmeldingen.json` when the list is requested (only when at least one minute has passed since the last refresh) but immediately return the notification list from the DAO. Make sure you remove the code that fills the database with the contents of the dummy file.
- Update the database with the new results when the response arrives.
  Parse the result with `JsonUtils.parseJSON(response)`
- When you run the application, the list will not be visible because the network request might take a while to complete and the view is not notified when this happens. Store the notifications as LiveData in your ViewModel and change the DAO to return `LiveData<List<TrafficNotification>>` instead of just the list. Observe the LiveData in your MainFragment and update the adapter when new data becomes available. Use the `notifyDataSetChanged` method of the adapter to trigger a refresh of the view.