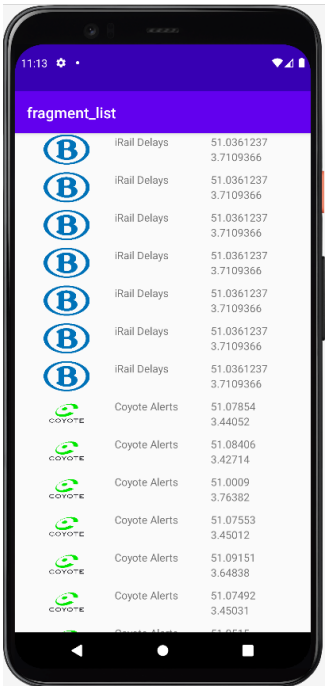
FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Labo Gebruikersinterfaces
Reeks 10: Recycler view en REST service

20 mei 2022

In deze reeks bouwen we verder op de applicatie van het vorige labo. Aan de hand van `RecyclerView` zullen we een lijst van alle verkeersnotificaties weergeven. In het tweede deel van de opdracht zullen we ervoor zorgen dat de notificaties via een REST-service worden opgehaald in plaats van uit een lokaal JSON-bestand.



Figuur 1: Voorbeeld

1 Recycler view

1.1 Adapter

1. Als eerst plaats je dit: `apply plugin: 'kotlin-android-extensions'` bovenaan in de gradle file.

2. Definieer een nieuw fragment, `ListFragment`, en voeg er een `RecyclerView` aan toe. Dit fragment zal de lijst van notificaties weergeven.

3. Voeg een nieuwe Layout Resource File, `notification_view.xml`, toe. In dit bestand definiëren we de layout van één individueel element in de lijst. Gebruik hiervoor een `LinearLayout` met voorlopig één `TextView`.

4. Definieer een adapter klasse, `TrafficNotificationAdapter` die overerft van de klasse `RecyclerView.Adapter` en een lijst van `TrafficNotifications` als input verwacht. Aangezien `RecyclerView.Adapter` een abstracte klasse is moeten volgende overerfde methodes geïmplementeerd worden: `onCreateViewHolder`, `onBindViewHolder` en `getItemCount` (zie verder).

5. De adapter-klasse zal in staan voor het aanmaken van `ViewHolder`-objecten en het binden van data met deze `ViewHolder`-objecten. Definieer in de adapter-klasse, een `ViewHolder` -klasse die overerft van `RecyclerView.ViewHolder`.

6. Een `ViewHolder` houdt een referentie bij van een `View`. Definieer in de `ViewHolder`-klasse een variabele `textView` en ken hieraan de `TextView` toe die je gedefinieerd hebt in `notification_view.xml`.

7. Implementeer `onCreateViewHolder`, in deze methode maken we een nieuwe instantie aan van de eerder gedefinieerde `ViewHolder`-klasse. Hiervoor hebben we een `View`-object nodig, dit kunnen we verkrijgen door `notification_view.xml` te "inflaten". Deze methode zal automatisch worden opgeroepen wanneer de `RecyclerView` een nieuwe `ViewHolder` nodig heeft om een bepaald element weer te geven.

8. De methode `onBindViewHolder` zal door `RecycleView` opgeroepen worden om data op een bepaalde positie weer te geven. Haal de notificatie op de gevraagde positie op en zet het `text`-attribuut van de TextView met de naam van de notificatie en het `tag`-attribuut met de positie in de lijst (dit zal later nog van pas komen).

9. In de methode `getItemCount`, geef je de lengte van de lijst van notificaties terug.

10. "Override" vervolgens de methode `onViewCreated` in `ListFragment`. Stel de `LayoutManager` en adapter als volgt in:

```
notification_list.apply {
    layoutManager = LinearLayoutManager(activity)
    adapter = viewModel.notifications.value?.let {
        TrafficNotificationAdapter(it)
    }
}
```

    Hierbij is `notification_list` de id van de `RecyclerView`. Merk op dat je eerst een referentie naar het `ViewModel` zal moeten verkrijgen. Zorg er eveneens voor dat de eigenschap `notifications` public is.

11. Voeg vervolgens een knop toe aan `MainFragment` die ervoor zorgt dat `ListFragment` getoond wordt. Je kan de app nu uittesten, alle notificaties zouden zichtbaar moeten zijn.

1.2 ListAdapter

1. Als we `ListFragment` instellen als de start destination in `nav_graph.xml` dan kan het zijn dat de notificaties niet zichtbaar zijn. Dit komt doordat op het moment dat de data met de adapter worden meegegeven deze nog niet geladen zijn door de repository. Om dit op te lossen zullen we gebruik maken van een `ListAdapter`, deze kan overweg met veranderende data. `MainFragment` hebben we nu eigenlijk niet meer nodig
2. Pas de klasse `TrafficNotificationAdapter` aan zodat die overerft van `ListAdapter`. Zorg ervoor dat je `import androidx.recyclerview.widget.ListAdapter` toevoegd en niet `import android.widget.ListAdapter`.
3. Om de verschillen in de lijst te bepalen heeft `ListAdapter` een object nodig die elementen in de lijst kan vergelijken. Definieer hiervoor de klasse, `TrafficNotificationDiffCallback` die overerft van `DiffUtil.ItemCallback`. Implementeer de methodes: `areItemsTheSame` en `areContentsTheSame`.
4. Verwijder de functie `getItemCount`, deze is reeds geïmplementeerd in de parent-klasse.
5. Aangezien de lijst van notificaties voortdurend kan veranderen houden we deze niet bij in onze klasse, om een notificatie op een bepaalde positie op te vragen kunnen we de methdode `getItem` gebruiken die gedefinieerd is in `ListAdapter`.
6. Tot slot moeten we enkel `onViewCreated` in `ListFragment.kt` nog aanpassen:
   (a) Verkrijg een referentie naar het `ViewModel`.
   (b) Instantieer een `TrafficNotificationAdapter` en ken dit object toe aan `adapter` binnenin de "`apply`".
   (c) Voeg een observer toe voor de lijst van notificaties die worden bijgehouden door `ViewModel`. Wanneer de lijst verandert, moet dit meegedeeld worden aan de adapter a.d.h.v. de methode `submitList`.

2 connectie met REST-server

In dit deel van de opdracht vervangen we het JSON bestand door een REST service. Op Ufora kan je een IntelliJ project `TrafficService` vinden. Download en run dit project (in IntelliJ, niet in Android Studio) om de REST server te starten. Als alles goed is zou je de API moet kunnen bekijken via http://localhost:8081/notifications.
Als er foutmeldingen komen omdat het *springframework* niet gevonden wordt, klik je in het linkervenster (het projectoverzicht) op de map `TrafficService` met de rechtermuisknop. Daar kies je voor *Add Framework Support...*, en dan vink je *Spring MVC* aan. Herstarten van het project is dan wel nodig.

1. Voeg `android:usesCleartextTraffic="true"` lijn toe aan de `apllication`-tag in je manifest bestand om HTTP verkeer tussen een server en je applicatie toe te laten.
2. Om toegang tot het internet toe te laten dien je in ditzelfde bestand volgende lijn toe te voegen buiten de `application`-tags:

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

3. Voeg een dependency toe voor de `Volley` bibliotheek, deze zullen we gebruiken om HTTP berichten te versturen en te ontvangen:

```
implementation "com.android.volley:volley:1.2.0"
```

4. Om onze lokale database op te vullen met data afkomstig van de REST-service gaan we in twee stappen te werk. Eerst halen we de data op en dan voegen we deze toe aan de Room database. Het opvullen van de database moet asynchroon gebeuren zodat de main thread niet geblokkeerd wordt. Het versturen van een HTTP-request moet ook asynchroon verlopen maar dit wordt afgehandeld door de `Volley` bibliotheek.

5. Voer volgende stappen uit om een HTTP-request te versturen:

   (a) In reeks 8 hebben we een callback toegevoegd voor de `onOpen`-functie in onze `TrafficNotificationDatabase` klasse. In deze callback starten we een asynchrone taak om de database op te vullen. Nu moeten we echter eerst een HTTP-request versturen, het opvullen van de database kunnen we pas doen eens we de data van de REST-service hebben ontvangen.

   (b) Om een HTTP-request te versturen, maak je een `JsonArrayRequest` object aan. Welke HTTP-methode moeten we hier gebruiken? In de `Response.Listener` zullen we de ontvangen data uiteindelijk moeten toevoegen aan onze lokale database. Voorlopig kan je hier de data loggen, zo kan je testen of je alle data correct hebt ontvangen.

   (c) Als URL kunnen we niet 127.0.0.1 of `localhost` gebruiken aangezien dit zal wijzen op het loopback IP adres van het Androidtoestel. Gebruik in plaats hiervan het IP-adres van je PC. Zoek dit op aan de hand van het commando `ipconfig` (Windows) of `/sbin/ifconfig` (Mac). De URL wordt dan bijvoorbeeld: http://192.168.0.254:8081/notifications.

   (d) Instantieer een `RequestQueue` en voeg de `JsonArrayRequest` er aan toe. De `Volley` bibliotheek zal deze request versturen en de meegegeven `Response.Listener` oproepen wanneer een antwoord van de server is ontvangen.

6. Als je gevalideerd hebt dat je applicatie alle data correct ontvangen heeft, kan je deze nu gebruiken om de Room database op te vullen. Voer hiervoor volgende stappen uit:

   (a) In de `Response.Listener` zullen we nu, analoog als voordien, de `WorkManager` en de `InitializeDatabase` klasse gebruiken om de database van data te voorzien. Je kan aan de hand van een companion object de `InitializeDatabase` klasse voorzien van de ontvangen `response` (de JSON data).

   (b) Pas nu de functie `doWork` in `InitializeDatabase` aan zodat de data afkomstig van de REST-service worden gebruikt i.p.v. het JSON-bestand. Voor ieder JSON-object in de ontvangen JSON array zal je een `Notification`-object moeten aanmaken en moeten toevoegen aan de database. Eens je een `Notification`-object hebt is het toevoegen aan de database analoog als wanneer de data afkomstig zijn vanuit een JSON-bestand.

7. Om zeker te zijn dat de data opgehaald worden via de REST-service, wis je best alle data op het Androidtoestel. Hiervoor klik je in de AVD manager op "Wipe data"voor het gebruikte toestel.

3 Verdere afwerking van de app

1. Pas de adapter en het layout-bestand aan zodat gebruik gemaakt wordt van databinding.
2. Zorg ervoor dat nu ook de coordinaten getoond worden voor ieder item.
3. Zorg ervoor dat de detail pagina getoond wordt wanneer op een element geklikt wordt. Voeg hiervoor een functie toe aan `MainActivity.kt`. Wanneer op een item geklikt is, kan je het `tag`-attribuut gebruiken om de positie te verkrijgen (in deel 1 hebben we in de adapter, de postitie toegekend aan het `tag`-attribuut.
4. Plaats een van de drie iconen bij iedere notificatie. Je kan de afbeeldingen terugvinden onder resources/drawables.