FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

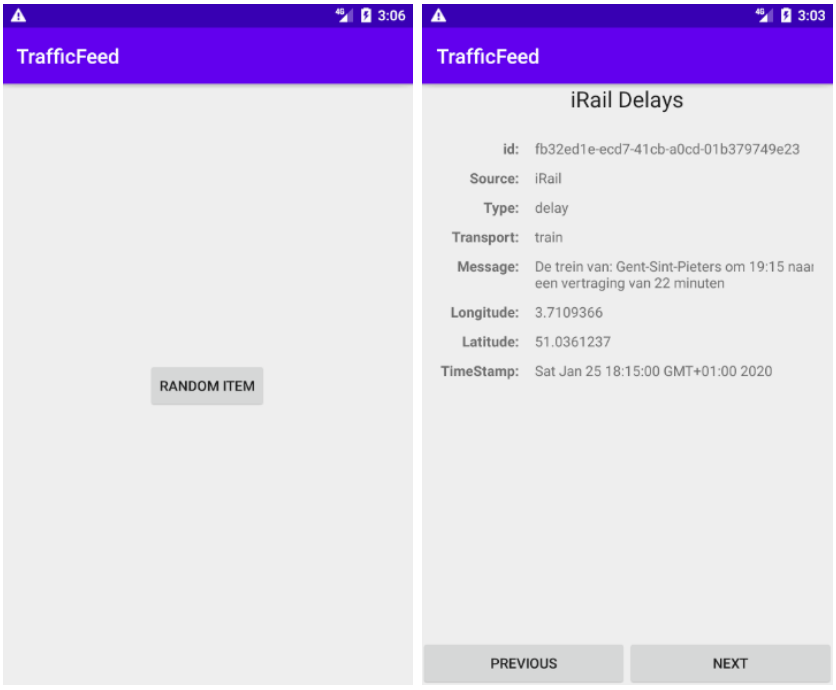
Labo Gebruikersinterfaces
Reeks 9: Navigation with fragments

17 mei 2022

In the previous session we made an application that shows a detail page for a single traffic notification from the traffic around Ghent. In this sesssion we will focus on managing the navigation from a simple fragment to this detail page. First we will need to refactor to fragments. The layout for this session is shown in Figure 1.

You will learn the following concepts:

- Using the Navigation Architecture Component https://developer.android.com/topic/libraries/architecture/navigation
- Create single Activity applications with fragments
- Provide multiple layout files for landscape and portrait
- Databinding with fragments



Figuur 1: Example layout of the Traffic Notifications Application with fragments.

In this session, we will setup our application with Navigation Architecture Component to manage the navigation from one screen to the other, selecting a random item from the notification list.

1 Project set-up

1. The start code project can be downloaded from Ufora. This start code is the solution of the last session.
2. Add the following dependencies for Navigation to your gradle file:

```
def nav_version = "2.3.5"

implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"
implementation "androidx.navigation:navigation-ui-ktx:$nav_version"
```

2 Provide a Navigation Graph

We will update the application according to the master detail flow within the MVVM structure. Therefor we need to refactor our code to use `Fragments` and provide a navigation graph resource file. The navigation graph requires a `NavHostFragment` inside the `Activity` and multiple `Fragments` to navigate between.

1. Create a new resource file with the **Navigation** resource type: Right click on the **res** directory and select for **new > Android Resource File**. Give your file the name **nav_graph** and select **Navigation** as **Resource Type**.
2. Open the resource file with the Navigation Editor. If it states that this is not available try syncing your project (File –> Sync with Gradle files).

   You'll find some documentation about the Navigation Editor here:
   developer.android.com/guide/navigation/navigation-getting-started.

3 Master Detail Flow

1. You can add fragments by clicking on the **add destination** button and then choose for **Create new destination**.
2. Create two blank `Fragments` from the Navigation editor: **MainFragment** and **DetailFragment**.
3. When a blank `Fragment` is created, Android Studio generates a factory method inside the Fragment's class. Since we will not pass any parameters to our fragments, you can remove this. Afterwards, the MainFragment class should look like this:

```
package com.example.trafficfeed

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
```

```
class MainFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_main, container, false)
    }
}
```

4. For each fragment you can edit the layout in the corresponding layout files. Add a constraint layout with a single button to the **MainFragment**, which will later be used to select a random item. Data binding will be used to handle click events. This `Fragment` will be responsible to handle the navigation with the Navigation Controller. (In item 11 this will explained in more detail.)

5. The **DetailFragment** should have the same layout as in **activity_main.xml** layout file.

6. Update `Fragments` (files `MainFragment.kt` and `DetailFragment.kt`) to support data-binding. The code below shows how you can use data binding with fragments.

```
override fun onCreateView(
    inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View? {
    val binding: FragmentMainBinding =
        DataBindingUtil.inflate(inflater, R.layout.fragment_main,
                                container, false)
    // TODO: set binding variables

    return binding.getRoot()
}
```

7. Set the correct variables. The repository is set when the `ViewModel` is used for the first time (`MainFragment`).

8. Use the `by activityViewModels()` Kotlin property delegate to get access to the viewmodel from the fragments. You can find more information here: https://developer.android.com/topic/libraries/architecture/viewmodel.

9. Connect the `Fragments` in the Navigation editor.

10. Update `MainActivity` resource file to provide a `NavHostFragment` (can be found in the design editor) inside a `FrameLayout`. The Navigation Controller will use this navigation host to manage navigation.

11. Bind the click event from the `MainFragment`'s button to show and navigate to a random traffic notification.
    - In the `ViewModel` provide a method to set the current notification to a random item. Make sure that the next() and previous() methods still function as designed.
    - Select a random notification when the button in the **MainFragment** is pressed.
    - Use `Navigation.findNavController(...)` to navigate using the defined connection.

12. When you get this error:

    ```
    Binary XML file line #15 in com.example.trafficfeed:layout/activity_main:
    Error inflating class fragment
    ```

    make sure you removed `setContentView(R.layout.activity_main)` in your MainActivity.

4 Master Detail in landscape

1. Create a new layout resource file with the same name as the **activity_main.xml** file.
   - Add an orientation qualifier so the layout file will only be used in landscape.
   - Add the Main and Detail Fragment to this layout in a horizontal `LinearLayout`. The left fragment should only be 1/3 of the screen.
2. Execute the navigation action only when the phone is in portrait orientation. You can determine the orientation by checking if the `FragmentContainerView` in the activity_main (portrait) is not null.

Warning: when testing the app, do not turn the emulator upside down (in portrait), a real-life device does not like that either. You won't see the right constellation - even if your code is ok.

5 Up navigation

1. Setup `MainActivity` to handle up navigation corrrectly. In most cases up navigation's behaviour is the same as the back button. Look at `NavigationUI` to setup the action bar with the Navigation controller (https://developer.android.com/guide/navigation/navigation-ui#action_bar).