Praktikum 3 – Layout erweitern und Tests

Die erste Version der Android-App soll nun um zusätzliche Ansichten erweitert werden. Dazu werden Fragmente eingefügt und ein Menü angelegt. Wir verwenden dazu die mit Jetpack zur Verfügung stehenden Mechanismen.

Schritt 1: Projekt sichern

Sichern Sie den Zustand des aktuellen Projektes. Das sollten Sie immer machen, wenn Sie Weiterentwicklungen vornehmen. Am einfachsten ist das folgende Vorgehen:

1. Am Anfang sollte das Projekt bereinigt werden. Rufen Sie dazu im Menu ‚*Build*' -> ‚*Clean Project*' auf.

2. Man könnte das Projekt nun nach GIT (GITLab) hochladen. Noch einfacher ist es, das Projekt als zip-Archiv zu sichern. Versehen Sie das Archiv mit einer Versionsnummer, ergo: **`DriverHelp_v1.zip`**.

Schritt 2: Fragmente anlegen

1. Wir legen im vorhandenen Projekt über den Wizard zunächst zwei Fragmente an. Das Vorgehen hatten wir in der Vorlesung kennengelernt.

    a. Wir nennen das *erste* Fragment: **`DriverHelp`**. Das Layout heißt: **`fragment_driver_help`**.

    b. Wir nennen das *zweite* Fragment: **`DriverHelp_Information`**. Das Layout heißt dementsprechend: **`fragment_driver_help_information`**.

2. Das Layout des ersten Fragments ersetzen Sie durch das bereits vorhandene *ConstraintLayout* in **`activity_main.xml`** der Main Activity.

3. Die Auswertungen und Anzeigen sollen nun im ersten Fragment vorgenommen werden. Deshalb migrieren wir den Code in der **`onCreate()`** Methode der *Main Activity* (**`MainActivity.kt`**) siehe in die **`onCreateView()`**-Methode des Fragmentes **`DriverHelp.kt`**.

    Wir merken uns in dieser Methode zusätzlich noch das erzeugte View-Objekt:

    ```
    val view = inflater.inflate(R.layout.fragment_driver_help,
            container, false)
    ```

    Auf diesem Objekt können dann die Bedienelemente im Layout gesucht werden, z.B.:

    ```
    val buttonCalc = view.findViewById(R.id.buttonCalc) as Button
    ```

    Die Zuweisungen für die anderen Elemente sind zu ergänzen.

    Schließlich geben wir am Ende von **`onCreateView()`** den erzeugten View per **`return view`** zurück.

Schritt 3: Navigation anlegen

1. Unsere App mit den zwei Fragmenten ist zwar wenig komplex, wir legen aber schon einmal eine Navigations-Ressource **`res/navigation/navigation.xml`** über den Wizard an.

2. Im Editor der Navigationsressource werden die beiden Fragmente hinzugefügt. Beim Hinzufügen der Ressource werden gleichzeitig die Projekt-Abhängigkeiten (*Dependencies*) im **`build.gradle`** angepasst. Hier wird auf eine Inkompatibilität hingewiesen, die wir ignorieren. Derartige Dinge passieren bei der Android-Entwicklung durchaus häufiger und es kann zu unschönen Effekten kommen, was hier aber nicht der Fall ist.

3. Es muss nun eine Navigation vom Fragment **`driverHelp`** zum Fragment **`driverHelp_Information`** angelegt werden. Diese Verbindung kann durch den Editor recht einfach erzeugt werden und wird unter dem Ausgangsfragment **`driverHelp`** unter dem Namen **`action_driverHelp_to_driverHelp_Information`** angezeigt.

4. Im Layout der **`activity_main.xml`** der Main Activity muss der Navigations-Controller instanziiert werden, der sich genau in das Layout einpassen soll. Dazu müssen entsprechende *Constraints* (Rand oben/unten und links/rechts jeweils 0dp) eingestellt werden.

```
<androidx.fragment.app.FragmentContainerView
    android:id="@+id/fragmentContainerView"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:defaultNavHost="true"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:navGraph="@navigation/navigation" />
```

Schritt 4: Erzeugen und Testen

Bevor wir die Entwicklung fortsetzen, generieren wir zunächst einmal die App und starten diese im Emulator. Das Ergebnis könnte ernüchternd sein: gegenüber der ursprünglichen Version hat sich (oberflächlich gesehen) nichts geändert. Allerdings wird das Formular nun als Fragment im Container der Main Activity angezeigt.

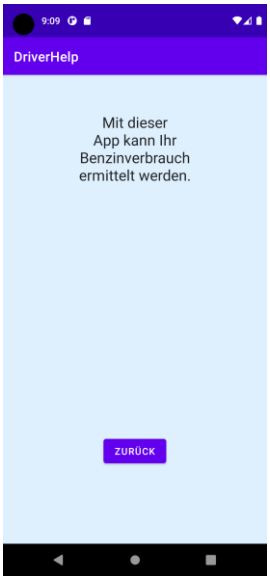
Schritt 5: Info Seite gestalten und aufrufen

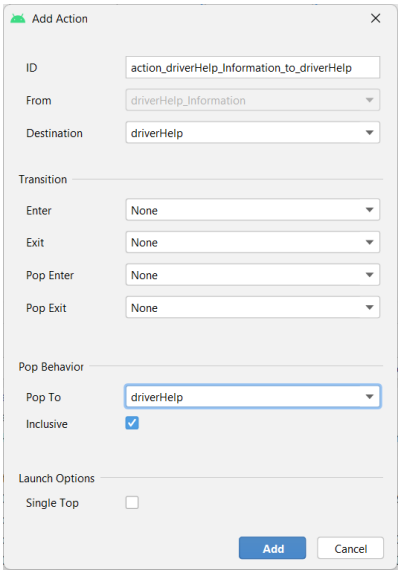
1. Um das Fragment **`DriverHelp_Information`** aufrufen zu können, bauen wir in das Layout des Fragmentes **`DriverHelp`** einen Button ein, dessen Id ‚*buttonInfo*' und Beschriftung ‚*Info*' lauten könnten.
2. Im Code des ersten Fragmentes ergänzen wir in **`onCreateView()`** den Click-Listener:

```
view.findViewById<Button>(R.id.buttonInfo)?.setOnClickListener {
    val navContr = findNavController()
    navContr.navigate (R.id.action_driverHelp_to_driverHelp_Information)
}
```

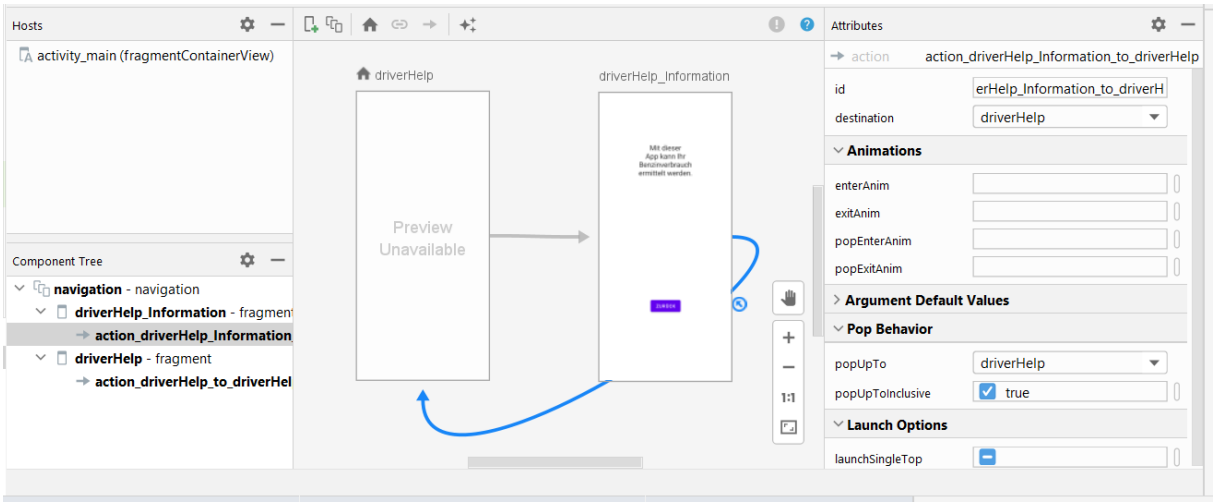
Dabei muss noch der Import **`androidx.navigation.fragment.findNavController`** ergänzt werden.

3. Das Layout des zweiten Fragmentes kann noch verbessert werden. In einem sollte ein kurzer Text gezeigt. Hierzu kann – wie bereits behandelt – ein **`<TextView>`** eingesetzt werden. Besser wäre die Nutzung eines **`<WebView>`**, in den eine lokale Datei (html) eingeladen werden kann. Die genaue Umsetzung soll hier nicht beschrieben werden und stellt eine freiwillige Zusatzaufgabe dar.
   Des Weiteren ist Button zur Rückwärtsnavigation vorzusehen.





5. In der Navigationsdatei **`navigation.xml`** wird eine Rückwärtsnavigation eingebaut. Es wird auf das erste Fragment zurückgesprungen und der ‚*Back Stack*' zurück gesetzt. Das wäre hier bei den zwei Fragmenten wohl nicht notwendig, aber wenn später weitere Fragmente hinzukommen, ist dies sinnvoll.



6. Im Fragment **`DriverHelp_Information.kt`** muss nun wie oben beim ersten Fragment beschrieben noch die Rückwärtsnavigation im Click-Listener des ‚*Zurück*' Buttons realisiert werden. Dazu die **`onCreateView()`** Methode entsprechend ändern.

Schritt 6: Navigation App Bar einrichten

1. Wie in der Vorlesung beschrieben, ist noch die Navigation App Bar in der Main Activity einzurichten.

2. Wir benötigen die Imports:

```
import androidx.navigation.fragment.findNavController
import androidx.navigation.ui.NavigationUI
```

3. Wir modifizieren die Main Activity darüber hinaus wie folgt:

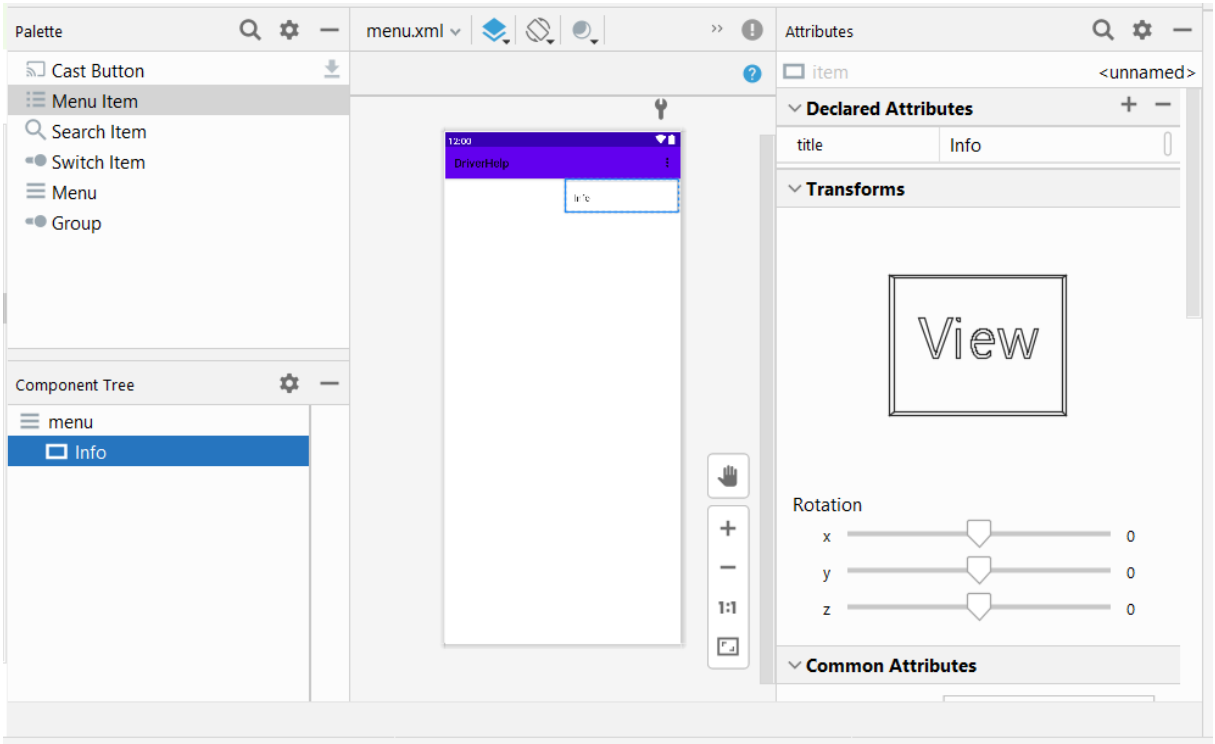
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        // Navigation App Bar einbauen
        findViewById<View>(R.id.fragmentContainerView).post {
            val navCtr = findNavController(R.id.fragmentContainerView)
            NavigationUI.setupActionBarWithNavController(this, navCtr)
        }
    }

    override fun onSupportNavigateUp(): Boolean {
        val navCtr = findNavController(R.id.fragmentContainerView)
        return navCtr.navigateUp()
    }
}
```

Schritt 7: Menu einrichten

In der Vorlesung wurde auch die Einrichtung von Menüs beschrieben.

Richten Sie eine solche Menü-Ressource ein. Ein Eintrag soll dabei zum Info-Fragment führen. Beachten Sie, dass die ID des Menu-Eintrages und die ID des Fragmentes im Navigation Controller übereinstimmen müssen!



Schritt 8: Testen der Anwendung

Sie können die Anwendung nun testen. Probieren Sie insbesondere die unterschiedlichen Arten der Navigation aus (per Button-Click, über das Menü oder per Swipe-Geste).

Optionale Aufgaben

Natürlich kann die App Optimierungen erfahren:

- Anzeige eines WebViews mit Informationen zur App im zweiten Fragment.
- Weitere Menü-Einträge vorsehen.

Ausblick

Alternativ kann das Testen der App auch auf einem realen Gerät durchgeführt werden. Zur Bedienung aus dem IDE heraus sind allerdings zusätzliche Einstellungen vorzunehmen. Das genaue Vorgehen wird zu einem späteren Zeitpunkt erläutert.

Informationen

Als Referenz für die Implementierung können Sie die Unterlagen zur Vorlesung verwenden.

Weitere Informationen auf der Seite https://developer.android.com/index.html (Android Developers Resources).