Praktikum 4 – Intents und Personalisierung der App

Apps können auch andere System-Komponenten integrieren, für deren Aufruf *Intents* verwendet werden. Als weiteres Thema wird in diesem Praktikum die Personalisierung von Apps behandelt.

Schritt 1: Projekt sichern

Sichern Sie zunächst den aktuellen Zustand des Projektes.

Schritt 1: Weitere Aktivität anlegen

1. Wir legen im Projekt über den Wizard eine weitere Aktivität an.
   Der Name der Aktivität ist **`StationFindingActivity`**. Das Layout heißt: **`activity_station_finding`**. Der vorgeschlagene Paketname wird übernommen.

2. Das Layout in **`activity_station_finding.xml`** ist recht einfach und basiert auf einem **`LinearLayout`**:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="20dp"
    android:orientation="vertical">
    <TextView
        android:id="@+id/locationLabel"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="100dp"
        android:text="Tankstelle suchen in:"
        android:textAppearance="@style/TextAppearance.AppCompat.Large" />
    <AutoCompleteTextView
        android:id="@+id/locationText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Ort" />
    <Space android:layout_width="match_parent" android:layout_height="50dp"
/>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:orientation="vertical">
        <Button
            android:id="@+id/btnSubmit"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Karte anzeigen"
            app:icon="@android:drawable/ic_dialog_map" />
    </LinearLayout>
</LinearLayout>
```

   Im Prinzip wird hier ein *Suchformular* realisiert. Die Eingabe erfolgt über ein View Element vom Typ **`AutoCompleteTextView`**. Über das Feld werden Suchvorschläge für Orte

gemacht, die konfiguriert werden können. Nach einer Eingabe kann ein Button (dieser wird mit einem Material-Design Icon verziert) angeklickt werden, durch den Google-Maps mit einem Angebot an lokalen Tankstellen aufgerufen wird.

3. Damit die Aktivität aufgerufen werden kann, muss diese im *Manifest* eingetragen sein. Das erledigt der Wizard für uns automatisch; sie sehen dort den Eintrag:

```
<activity
    android:name=".StationFindingActivity"
    android:exported="false" />
```

4. Wir wollen die Aktivität über einen Menu-Eintrag aufrufen. [Alternativ können Sie in der App auch einen Button erstellen, der dann die Aktivität aufruft.]

   a. Wir erstellen also den Menu-Eintrag:

   ```
   <item android:id="@+id/driverHelp_StationFinder"
       android:title="Tankstellen" />
   ```

   b. Das Menu hatten wir im Hauptfragment **`DriverHelp.kt`** der Applikation angelegt. Dort muss in **`onOptionsItemSelected()`** die Reaktion auf die Anwahl des Menü-Eintrages ergänzt werden. Diese besteht darin, die neue Aktivität über einen Intent aufzurufen.

   ```
   if (item.itemId == R.id.driverHelp_StationFinder) {
       val intent = Intent(this.getContext(),
                           StationFindingActivity::class.java)
       startActivity(intent)
       return true
   }
   ```

   Die neue Aktivität kann nun bereits über das Menu aufgerufen werden.

Schritt 2: Aktivität zur Auswahl ausimplementieren

1. Innerhalb von **`StationFindingActivity`** müssen wir einige Definitionen vornehmen:

```
// Standard-Orte, damit in der App schon etwas zur Verfügung steht.
val defaultLocations = arrayListOf<String>(
    "Osnabrück",
    "Hannover",
    "Oldenburg",
    "Melle",
    "Wallenhorst",
    "Rheine",
    "Georgsmarienhütte"
)

// Gespeicherte Liste mit Orten
private var locations = arrayListOf<String>()
// Aktueller Eintrag
private var location : String = ""

// Der Adapter zeigt die Liste mit Orten an.
var adapter: ArrayAdapter<String>? = null
```

Über **`defaultLocations`** konfigurieren wir die Vorschläge für das Auswahl-Element. Die Liste der Vorschläge wird über einen sog. *ArrayAdapter* verwaltet, den wir später noch initialisieren müssen. Es gibt verschiedene Adapter, die für die Verwaltung von Listen eingesetzt werden können. Beim ArrayAdapter kann nicht direkt auf die verwaltete Liste zugegriffen werden. Aus diesem Grund merken wir uns die Einträge in einer Variablen **`locations`** und den aktuellen Suchparameter in einer Variablen **`location`**.

2. Die Konfiguration des Adapters bzgl. der Vorschlagsliste erfolgt in der **`onCreate()`** Methode. Für das Auswahl-Element wird ein **`doOnTextChanged`** Listener hinterlegt: sobald Eingaben gemacht werden, ändert sich die Variable **`location`**. Im Click-Listener des Buttons wird aus der Variablen und einer Basis-URL eine Anfrage an Google Maps generiert.

   Die Konfiguration des Intents (siehe Vorlesung) müssen Sie noch ergänzen.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_station_finding)

    adapter = ArrayAdapter<String>(
        this,
        android.R.layout.select_dialog_item,
        defaultLocations
    )
    // val locationText = findViewById<View>(R.id.locationText) as
    // AutoCompleteTextView
    locationText.threshold = 2
    locationText.setAdapter(adapter)
    locationText.doOnTextChanged { text, start, before, count ->
        location = text.toString()
    }
    // Eingabefeld initialisieren mit erstem Eintrag.
    locationText.setText (locations.first())

    // val btnSubmit = findViewById<View>(R.id.btnSubmit) as Button
    btnSubmit.setOnClickListener {
        val uri =
Uri.parse("https://www.google.de/maps/search/Tankstellen+" +
            location)
        /* zu implementieren */ …
    }
}
```

   Fehlende Imports können durch *<ALT>-<Eingabe>* ergänzt werden. Um auf **`findViewById()`** Aufrufe wie hier im Code dargestellt zu verzichten, kann im Import Folgendes ausgewiesen werden:

```
import kotlinx.android.synthetic.main.activity_station_finding.*
```

   Damit dieser vereinfachende Kotlin-Mechanismus genutzt werden kann, müssen in der Gradle-Datei des Moduls bei den Plugins die Extensions für Kotlin hinzugefügt werden:

```
id 'kotlin-android'
id 'kotlin-android-extensions'
```

Schritt 3: Personalisierung von Anfragen

Gerade bei mobilen Anwendungen mit den eingeschränkten Modalitäten der Eingabe sind Personalisierungs- und Auto-Konfigurationsmöglichkeiten wichtig um eine komfortable Bedienung zu ermöglichen.

In der Funktion zum Finden von Tankstellen haben wir eine fest definierte Liste von Auswahlmöglichkeiten vorgesehen. Der Nutzer könnte aber auch einen anderen Ort eingeben. Die Eingabe wird dabei nicht gespeichert, der Ort muss also immer jeweils neu eingegeben werden. Diesen Missstand kann man durch persistente Speicherung getätigten Eingaben beseitigen. Wir verwenden dazu die sog. *SharedPreferences*. Die Daten dazu werden in einer xml-Datei auf dem Gerät gespeichert, sie können gelesen und verändert werden.

1. Die folgende Methode zeigt das Laden von Einstellungen:

```
fun loadDataFromSharedPreferences (): ArrayList<String> {
    val sharedPreferences = getSharedPreferences("shared preferences",
MODE_PRIVATE)
    // Die Einträge werden im JSON-Format in Dateien gespeichert.
    val gson = Gson()
    val json = sharedPreferences.getString("locations", null)

    // Konvertierung in eine Liste mit Strings.
    val type: Type = object : TypeToken<ArrayList<String?>?>() {}.type
    val saved_locations = gson.fromJson<Any>(json, type)

    // Gibt es schon gespeicherte Einträge?
    if (saved_locations == null) {
        locations = defaultLocations.clone() as ArrayList<String>
    } else {
        locations = saved_locations as ArrayList<String>
    }
    return locations
}
```

Die Listeneinträge werden im *JSON-Format* abgelegt. Wir benötigen einen Parser (GSON) zum Einlesen der Daten. Speziell bei der Client-/Server-Kommunikation wird dieses Format sehr häufig eingesetzt. Wir benötigen die Imports:

```
import com.google.gson.Gson
import com.google.gson.reflect.TypeToken
```

In der Gradle-Datei des Moduls muss eine *Dependency* eingefügt werden und ein ‚sync' erfolgen.

```
implementation 'com.google.code.gson:gson:2.8.7'
```

Es wird dann versucht die Shared Prefs zu lesen. Ggf. ist das Ergebnis nach dem Versuch einer Konvertierung in eine **`ArrayList`** *null*. In dem Fall wurde die App zum erste Mal

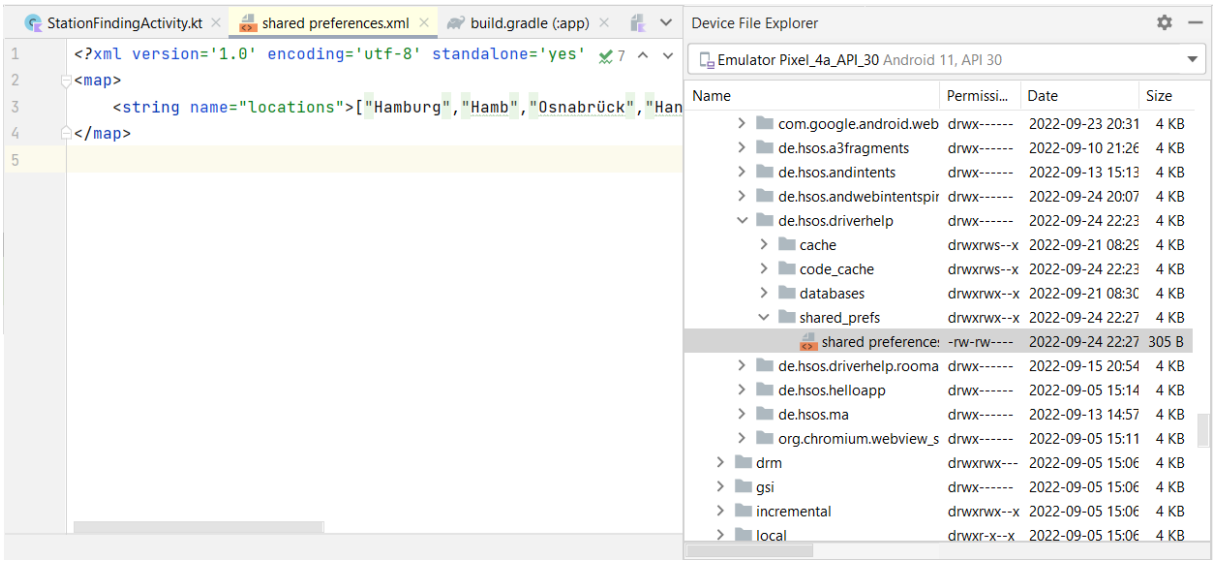
gestartet und es wird eine Auswahlliste mit den Default-Einträgen **defaultLocations** erzeugt.

2. Das Speichern erfolgt ganz analog zum Laden:

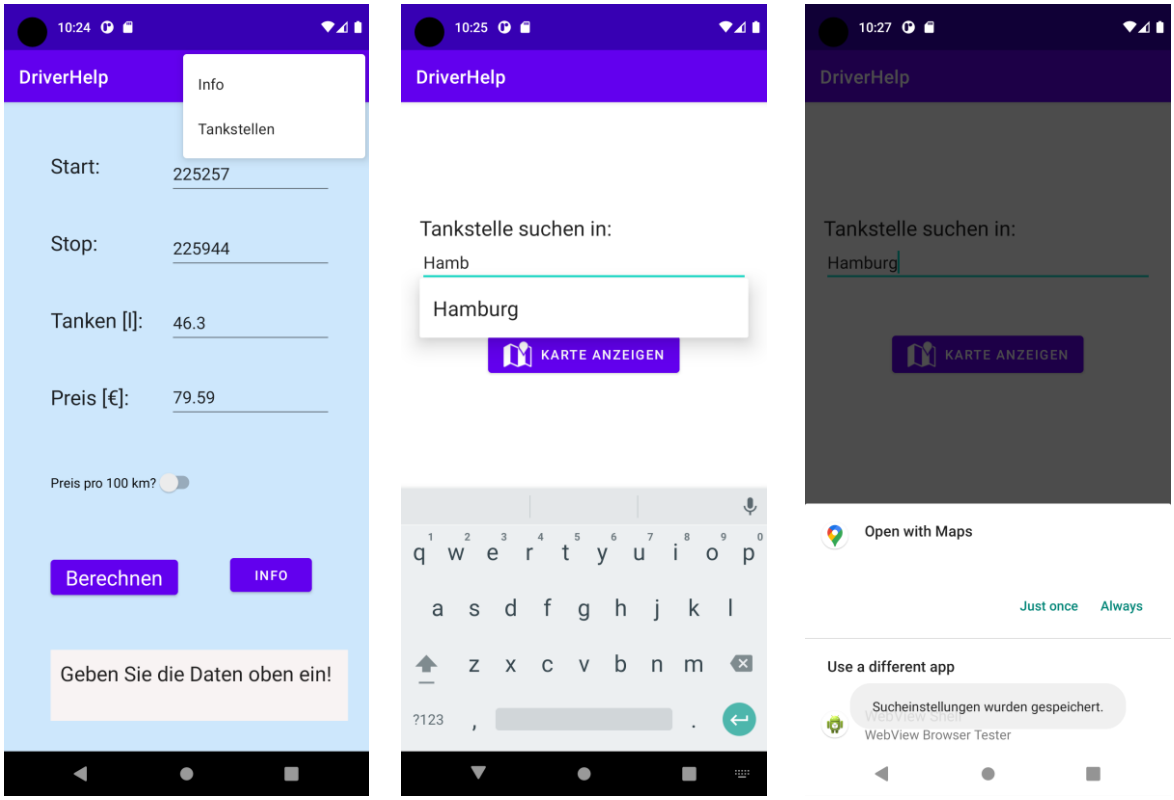
```
fun saveDataToSharedPreferences () {
    val sharedPreferences = getSharedPreferences("shared preferences",
MODE_PRIVATE)

    // Editor erzeugen.
    val editor = sharedPreferences.edit()

    // Erzeugen einer Liste.
    val gson = Gson()
    val json = gson.toJson(locations)

    // Daten speichern in den Shared Prefs.
    editor.putString("locations", json)
    editor.apply()

    // Der Toast mit Info erscheint immer unten im UI.
    Toast.makeText(
        applicationContext,
        "Sucheinstellungen wurden gespeichert.",
        Toast.LENGTH_SHORT
    ).apply { setGravity(Gravity.CENTER,0,0) }.show()
}
```

3. Sie müssen die Initialisierung des Adapters nun so ändern, dass dieser mit dem Ergebnis von **loadDataFromSharedPreferences()** initialisiert wird.

4. Es stellt sich nun folgende Frage: *Wann wird die Auswahlliste neu gespeichert?*
Das sollte dann der Fall sein, wenn mit einem eingegebenen Parameter Google Maps aufgerufen wurde. Es empfiehlt sich, dazu eine überschaubare Funktion **handleHistory()** einführen, ausimplementieren und an der richtigen Stelle aufrufen.
Ergänzen Sie den Code entsprechend!



5. Sie werden sich vielleicht Fragen, wo sich die Einstellungen genau auf dem Dateisystem der Gerätes befinden. Dazu können Sie mit dem ‚*Device File Explorer*' durch das Dateisystem navigieren. Gehen Sie dazu unter `/data/data/de.hsos.driverhelp/shared_prefs`. Sie sehen dann die JSON-formatierte Liste, die als String in der xml-Datei gespeichert ist.



Optionale Aufgaben

- Testen der Applikation auf einem realen Gerät.
- Es gibt in jetzigen Version keine Möglichkeit, die Präferenzen in der App auch wieder zurückzusetzen.
- Die Präferenzen können auch für andere App-Daten angewendet werden. Z.B. könnte der letzte Stop-Wert als Start-Wert für die nächste Eingabe als Präferenz gespeichert werden. Über die Präferenzen würde der Eintrag auch einen Neustart der App überstehen.
- Anstatt Google Maps über einen Intent aufzurufen, könnte auch ein WebView hinterlegt werden, in das die entsprechende Map über die generierte URL eingeladen wird.

Informationen

- Array-Adapter: https://guides.codepath.com/android/Using-an-ArrayAdapter-with-ListView
- Shared Preferences: https://developer.android.com/reference/android/content/SharedPreferences