Praktikum 2 – Grundgerüst einer Applikation

Es soll eine einfache Android-Anwendung *DriverHelp* zur Ermittlung des Benzinverbrauchs entwickelt und im Emulator getestet werden. In der App können Tachometer-Stände (letztes Tanken = Start, jetziges Tanken = Stop) eingegeben, sowie Treibstoffmenge und Kosten angegeben werden. Die App rechnet dann den Verbrauch sowie optional die Kosten pro 100 km Fahrtstrecke aus.

Schritt 1: Projekt anlegen

1. Über den Android Studio-Wizard können Sie ein neues Projekt anlegen. Sie müssen folgende Einstellungen im **Wizard** vornehmen:

    a. Wir implementieren eine Applikation für ‚*Phone and Tablet*'.

    b. Es soll eine ‚*Empty Activity*' angelegt werden.

    c. Name der Applikation: **`DriverHelp`**

    d. Package Name: **`de.hsos.driverhelp`**

    e. Als Sprache soll **Kotlin** verwendet werden.

    f. Das Minimum SDK muss festgelegt werden: z.B. *Android 5.0*. Gleichbedeutend damit ist der API-Level: für das obige Beispiel wird der somit der *API-Level 21* eingestellt. Je niedriger das Minimum SDK gewählt wird, umso mehr Geräte können erreicht werden. Die Geräte-Abdeckung (z.B. 98%) wird im Wizard indiziert.

2. Sobald Sie ‚*Finish*' auswählen, wird das Projekt angelegt. Sehen Sie sich die Struktur und insbesondere die **Manifest-Datei** (unter **`app/manifests/AndroidManifest.xml`**) genau an. Sie sollte folgendes Aussehen haben:

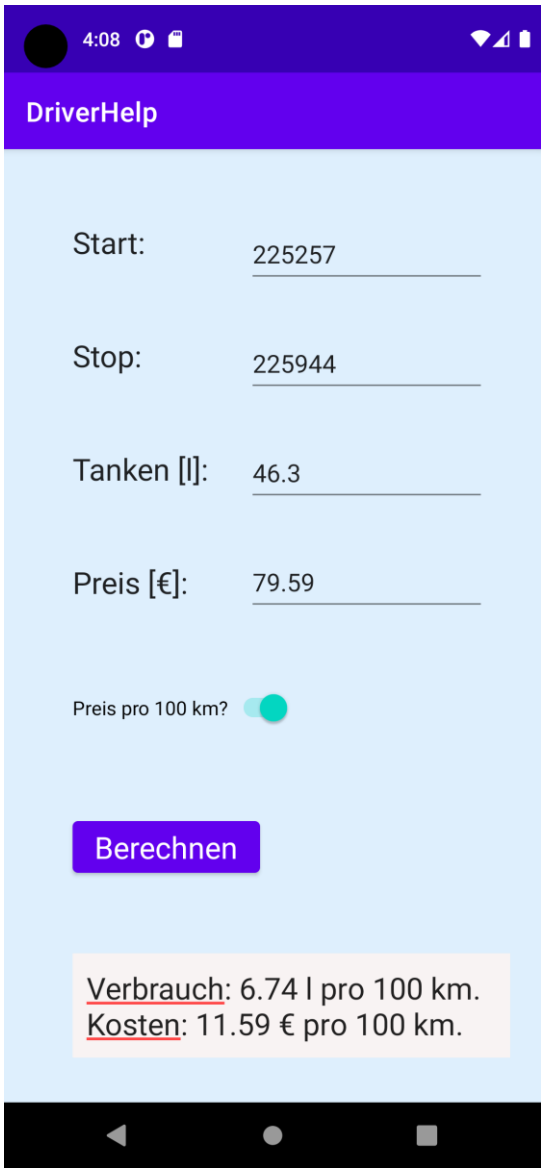
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="de.hsos.driverhelp">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.DriverHelp"
        tools:targetApi="31">
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

```
    </application>
</manifest>
```

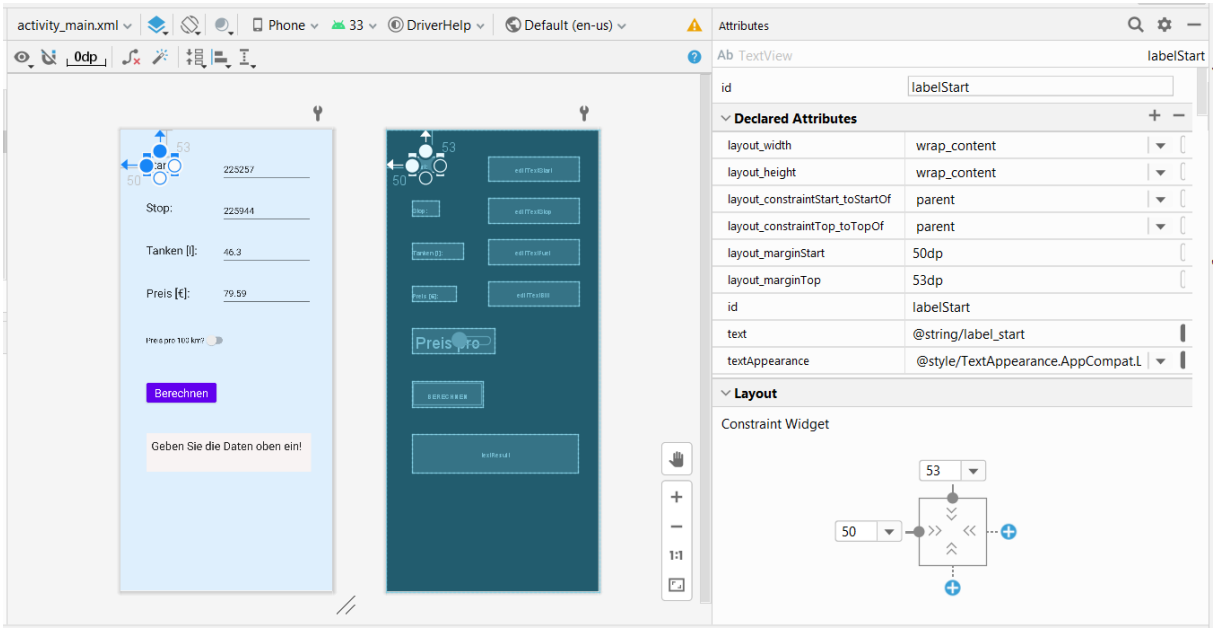
3. Das Projekt könnte nun sofort aufgerufen werden, was wir aber zunächst nicht tun.

Schritt 2: Basis-Layout der `MainActivity` definieren

1. Das Layout der Anwendung des Hauptbildschirms finden Sie unten. Es verwendet als View-Elemente: Labels (**`<TextView>`**) zur Kennzeichnung der Eingabefelder, die Eingabe-Felder (**`<EditText>`**) und ein Feld für die Ausgabe der Berechnung, einen Schalter (**`<Switch>`**) sowie einen Button (**`<Button>`**) zum Start der Berechnung.



2. Das Layout kann mit dem eingebauten UI-Builder erzeugt werden oder direkt im XML-Code editiert werden. Speichern Sie es in der Datei **`res/layout/activity_main.xml`**, denn dieser Name wird in der Methode **`setContentView`** der Aktivität **`MainActivity`** verwendet.



3. Als Basis-Layout stellt der Wizard ein **`ContraintLayout`** ein. Für das vorliedenge UI wäre ein **`LinearLayout`** vielleicht geeigneter. Man wird dann aber einen Hinweis bekommen, doch ein **`ConstraintLayout`** verwenden. Informationen dazu finden Sie unter: https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout
   Bei diesem Layout müssen die **Constraints** der View-Elemente angegeben werden, ansonsten wird eine Fehlermeldung im IDE angezeigt und beim Start der App die Elemente auf die Position (0,0) gelegt.
   Vorgeschlagene Vorgehensweise: legen Sie die benötigten View-Elemente zunächst mit dem UI-Builder aus und erzeugen dann die Constraints. Die Label / Beschriftungen an der linken Seite sollten immer Constraints nach oben (für das Label ‚*Start*' ist das ‚*parent*') und nach links haben (zum ‚*parent*'). Gemäß dieser Logik bezieht sich bei ‚*Stop*' der Constraint nach oben übrigens auf das Label ‚*Start*'.

   Die Text-Felder auf der rechten Seite haben Constraints nach rechts und nach oben.

4. Das Layout kann auch durch direkte Editierung der Datei activity_main.xml angepasst werden. Sehen Sie hier einen Ausschnitt:

```
<TextView
    android:id="@+id/labelStart"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="50dp"
    android:layout_marginTop="53dp"
    android:text="@string/label_start"
    android:textAppearance="@style/TextAppearance.AppCompat.Large"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/editTextStart"
    android:layout_width="175dp"
```

```
    android:layout_height="48dp"
    android:layout_marginTop="53dp"
    android:layout_marginEnd="40dp"
    android:ems="10"
    android:importantForAutofill="no"
    android:inputType="number"
    android:text="225257"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:ignore="SpeakableTextPresentCheck" />
```

Beim Anlegen der UI-Elemente sollten Sie folgendes beachten:

- Die View-Elemente müssen immer mit einer eindeutigen **`android:id`** versehen werden. Hierfür haben sich Konventionen etabliert, z.B. wird hier in der ID der Typ (z.B. *label*) und ein Merkmal des Elementes (*Start*) kodiert (ergo *labelStart*).
- Bei den Eingabefeldern kann ein im UI hinterlegter initialer Wert eingestellt werden (z.B. der Tachostand 225527, siehe **`android:text`**). Dann müssen nach dem Neustarten der App nicht umständlich Werte eingegeben werden. Gleichwohl sollte die Behandlung von fehlenden Eingaben erfolgen! In der Anwendung werden numerische Werte erfasst. Deshalb wird bei der Definition der Felder das Attribut **`android:inputType="number"`** angegeben. Für ‚*Nachkommastellen*' muss dabei der . (Punkt) verwendet werden (und nicht ein Komma).
- Falls Sie per Editor in der Layout-Datei arbeiten: mit der Tastenkombination *<STRG>-<SHIFT>-L* kann die Datei formatiert werden.

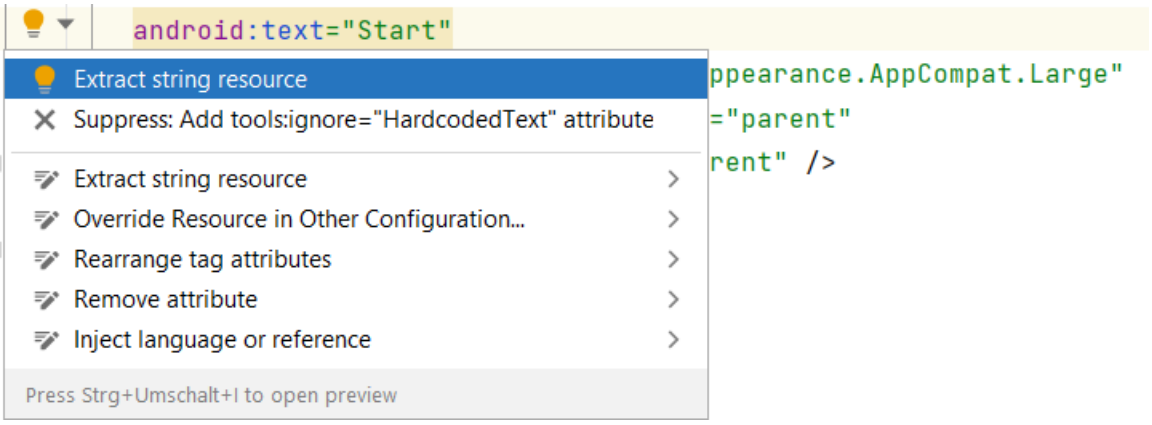
5. Die Texte erscheinen initial recht klein. Sie können mit dem folgenden Attribut vergrößert werden: **`android:textAppearance="@style/TextAppearance.AppCompat.Large"`**

6. Die Ergebnisse der Berechnungen sollen in einem Text-Feld ausgegeben werden. Dieses sollte speziell formatiert sein. Hier ist zunächst einmal die Einstellung eines Abstands des Anzeigetextes zu den Container-Grenzen via **`android:padding`** vorzunehmen. Sinnvoll ist die Einstellung einer Hintergrundfarbe (zur Hervorhebung sollte auch ein Hintergrund auf Layout-Container eingestellt werden; das ist über Interface-Builder recht einfach möglich) via **`android:background`**,…. Sie können sich an folgenden Einstellungen orientieren:

```
<EditText
    android:id="@+id/textResult"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="50dp"
    android:layout_marginTop="53dp"
    android:width="320dp"
    android:height="300dp"
    android:background="#F8F3F3"
    android:ellipsize="none"
    android:ems="10"
    android:gravity="start|top"
    android:inputType="textMultiLine"
    android:minLines="2"
    android:padding="10dp"
    android:text="@string/result_view"
    android:textAppearance="@style/TextAppearance.AppCompat.Large"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/buttonCalc" />
```

Schritt 3: Ressourcen anlegen

1. Für die Beschriftungen muss das Attribut (z.B. für das Label ‚Start') **`android:text="Start"`** gesetzt werden. Es wird Ihnen dann durch das IDE angeboten, eine String-Ressource zu extrahieren.



   Im Hinblick auf eine mögliche Internationalisierung des UI ist dies sinnvoll. Als Name für die Ressource sollten Sie auch hier gewissen Konventionen folgen, z.B. **`android:text="@string/label_start"`**

2. Die generierten String-Ressourcen werden im Verzeichnis **`res/values`** in der Datei **`strings.xml`** abgelegt:

```
<resources>
    <string name="app_name">DriverHelp</string>
    <string name="label_start">Start:</string>
    <string name="label_stop">Stop:</string>
    <string name="label_fuel">Tanken [l]:</string>
    <string name="label_price">Preis [€]:</string>
    <string name="result_view">Geben Sie die Daten oben ein!</string>
    <string name="button">Berechnen</string>
    <string name="switchText">Preis pro 100 km?</string>
</resources>
```

Schritt 4: Aktivität `MainActivity` implementieren

1. Die Funktionalität der App wird in der Datei **`MainActivity.kt`** umgesetzt.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        // Berechnung durchführen
        …
        }
    }
}
```

2. Vor der Durchführung der eigentlichen Berechnung, müssen die relevanten View-Elemente für die Eingabe referenziert werden. Hierzu kann die Methode **`findViewById()`** eingesetzt

werden[1]. Hierzu ist zu bemerken, dass jegliche Ressourcen intern in einer Klasse **R** gespeichert werden. Für den Zugriff gibt es verschiedene Alternativen:

```
// Verschiedene Möglichkeiten des Aufrufs von findViewById
val buttonCalc = findViewById(R.id.buttonCalc) as Button
val editTextStart = findViewById (R.id.editTextStart) as EditText
val editTextStop = findViewById<EditText> (R.id.editTextStop)
…
```

(Hinweis: mit der Tastenkombination *<STRG>-<ALT>-O* werden die verwendeten Klassen importiert. Die vorgeschlagenen Imports müssen jeweils mit *<ALT>-<Eingabe> bestätigt werden.*)

3. Auf dem Button wird dann ein *Listener* registriert. Dieser ermittelt und verwaltet zunächst die zur Berechnung notwendigen Werte. Beim Auslesen der Text-Felder sind dabei fehlende Eingaben zu monieren. Dafür gibt es verschiedene Möglichkeiten:

```
buttonCalc.setOnClickListener {
    var distance = -1.0
    var fuel = -1.0
    var bill = -1.0
    try {
        val start = editTextStart.text.toString().toDouble()
        val stop = editTextStop.text.toString().toDouble()
        distance = stop - start
    }
    catch (e: Exception) {
        // Option 1 im Fehlerfall: Nutzung des Ausgabefeldes für Warnung.
        textResult.setText("Geben Sie bitte Start und Stop an!")
        // Option 2 im Fehlerfall: Nutzung eines Toasts.
        Toast.makeText(this@MainActivity, "Geben Sie bitte Start und Stop
an!", LENGTH_SHORT).show()
        return@setOnClickListener
    }
    …
}
```

4. Schließlich findet im Listener die eigentliche Berechnung statt:

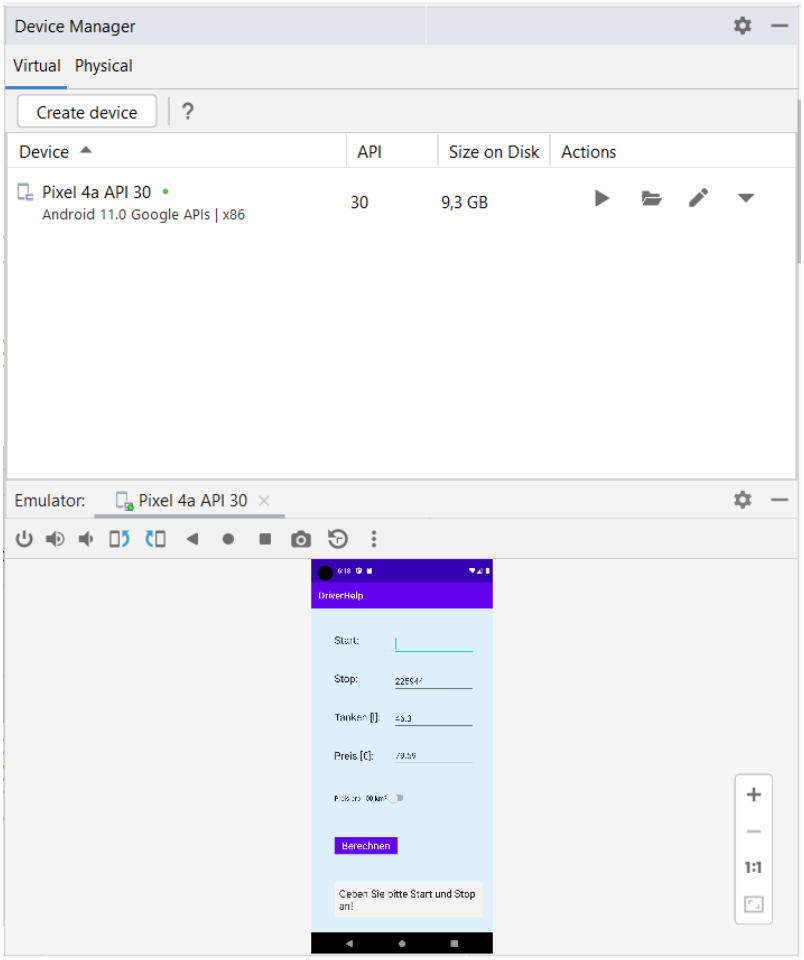
```
if (distance > 0.0 && fuel > 0.0 && bill > 0.0) {
    val consumption = fuel / distance * 100
    val consumption_formatted = "%.2f".format(consumption)
    val cost_per_100_formatted = "%.2f".format(bill / fuel * consumption)
    var msg: String = "Verbrauch: $consumption_formatted l pro 100 km.\n"
    val switch = findViewById (R.id.switchCostPer100) as Switch
    if (switch.isChecked())
        msg += "Kosten: $cost_per_100_formatted € pro 100 km."
    textResult.setText(msg)
}
```

[1] In manchen Büchern und Unterlagen zu Android Studio wird im Kotlin-Code über eine Variable, deren Name identisch mit der ID des View-Elementes ist, direkt auf das Element zugegriffen und bspw. ein Listener registriert. Das funktioniert zum einen nur durch Vewendung von Kotlin JetPack-Extensions (wobei die entsprechenden Bibliotheken sind mittlerweile *deprecated*); zum anderen fällt die Variable quasi vom Himmel. Deshalb wird hier zunächst der traditionelle Weg gewählt.

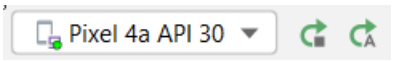
Schritt 6: Anwendung auf virtuellem Gerät testen

Sie können die Anwendung nun auf einem emulierten Gerät testen. Dazu müssen Sie zuvor ein entsprechendes *Virtual Device* mit dem *Device Manager* angelegt haben. Über den Device Manager kann das virtuelle Gerät auch gestartet werden, wobei der Start einige Zeit beanspruchen kann, da das Gerät im Emulator gebootet werden muss.

Es empfiehlt sich, das virtuelle Gerät während der Entwicklung nicht zu schließen. Ggf. sind nach Abschluss des Bootvorgangs noch Initialisierungen vorzunehmen. Wenn dann die Applikation nicht angezeigt wird, sollten Sie diese im Studio noch einmal aufrufen.

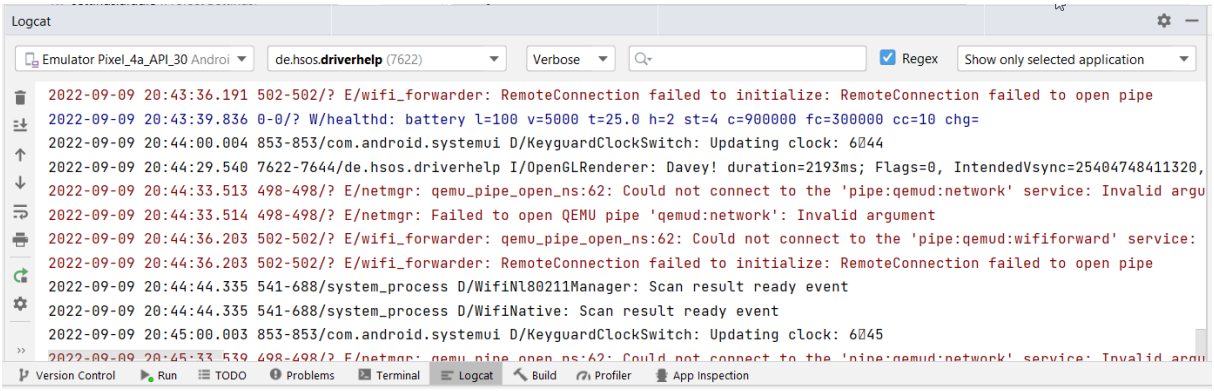


Ändern Sie den Code der Anwendung (auch das Layout), so reicht es, erneut ‚*Run..*' aufzurufen. Die App wird dann neu erzeugt, auf dem Gerät installiert und gestartet. Die relevanten Controls sind unten dargestellt.



Speziell initial benötigt der Erzeugungsprozess einige Zeit. Der Status der Gradle Build Prozesses kann durch Anklicken der Statusanzeige unten rechts sichtbar gemacht werden.

Bei Ausführung der Anwendung werden verschiedene Logs angezeigt. Nützlich sind häufig die Ausgaben im sog. *Logcat* . Die Ausgaben bieten sich zur Ursachenforschung bei Systemabstürzen an.



Optionale Aufgaben

Natürlich kann die App Optimierungen erfahren:

- Das Layout der Anwendung kann weiter verbessert werden. Z.B. können die Constraints so eingestellt werden, dass Größe der Eingabefelder je nach Display angepasst wird.
- Das **`Switch`**-Element könnte durch **`SwitchCompat`** aus der Material Design Bibliothek ersetzt werden.
- Es könnte die Eingabe von Zahlenwerten in den Textfeldern mit Kommata statt Punkt ermöglicht werden.
- …

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 Seite https://developer.android.com/index.html (Android Developers Resources) verwenden.