Fernuniversität Hagen

Seminar 1908 Moderne Programmiertechniken und -Methoden

# Entwicklung von Android Apps - Android SDK

*Allan Kaufmann*

*9502998*

supervised by
Dr. Daniela Keller

28. Juni 2018

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Quellcodeverzeichnis

# Teil I

# Android SDK Überblick

## 1.1 Thema dieser Arbeit

### 1.1.1 Android

Android ist eine Software-Plattform für mobile Endgeräte. Viele Hersteller, Konsumenten und Entwickler setzen auf dieses offene Betriebssystem, das mit einer leistungsfähigen Entwicklungsplattform daherkommt. Wenn Sie Android-Apps entwickeln, bedienen Sie einen großen und heterogenen Markt mit Geräten verschiedener Typen und Hersteller, auf denen nicht wenige Android-Versionen zugleich im Einsatz sind. [2, s. S. 1] Unter einer Plattform verstehen wir eine Menge von betriebssystem- und hardwarenahen Diensten und Verarbeitungselementen. Die Plattform wird durch die Prozessorarchitektur sowie durch Schnittstellen des auf der Hardware ausgeführten Betriebssystems bestimmt.[4, s. S. 10]

Android als Plattform, sowie das Android Software Development Kit (SDK), wurden am 12. November 2007 von der Firma Google, im Android Developer Blog, angekündigt [3]. Bis heute hat Android für mobile Geräte einen weltweiten Marktanteil von über 80% erreicht [1]. Zu diesen Geräten gehören Smartphones, sowie andere Geräte wie Tablets, Smartwatches oder Fernseher.

Anroid Runtime (ART)

Für die Entwicklung und Ausführung von Anwendungen kommt, seit Android 5, standardgemäß die Laufzeitumgebung Android Runtime (ART) zum Einsatz [6]. Sie bildet den Rahmen für die Ausführung aller Programme, die der Benutzer auf einem Android-System startet [2, s. S. 27]. Der Quelltext dieser Anwendungen wird in der Java-Syntax geschrieben. Der kompilierte Code unterscheidet sich jedoch vom Java-Bytecode und ist als Dex Bytecode spezifiziert [5].

### 1.1.2 Überblick

Android-Komponenten

In dieser Arbeit zeige ich Ihnen die wesentlichen Konzepte des Android Software Development Kits. Hierzu betrachten wir zunächst die Android Plattformarchitektur. Um eine mobile Anwendung entwickeln zu können, müssen wir uns mit den benötigten Komponenten des Android SDK befassen. Dazu gehören Aktivitäten, Fragmente, Konfigurationsdateien, sowie Views wie Eingabeelemente und Schaltflächen. Die wichtigsten Komponenten werden im ersten Teil dieser Arbeit vorgestellt.

App-Entwicklung

Im zweiten Teil dieser Arbeit zeige ich Ihnen den Entwicklungsprozess einer Android App, mithilfe der Entwicklungsumgebung Android Studio. Hierzu werde ich die wesentlichen Funktionen von Android Studio kurz vorstellen, eine Beispielanwendung spezifizieren und Ihnen die notwendigen Schritte erläutern, um die spezifizierte Beispielanwendung mit Android Studio umzusetzen.

## 1.2 Android Plattformarchitektur

Android ist eine Open Source Plattform, welche auf dem Linux Kernel basiert und für eine Vielzahl von Gerätetypen entwickelt wurde [7]. Die Systemarchitektur besteht aus mehreren Schichten, welche in Abbildung 1.1 auf Seite 7 dargestellt werden und in den nachfolgenden Abschnitten beschrieben werden.

### 1.2.1 Linux Kernel

Linux Kernel

Die Grundlage der Android Plattform ist der Linux Kernel. Er kümmert sich um die Prozesssteuerung, die Speicherverwaltung, die Netzwerkkommunikation sowie um das Thema Sicherheit [2, s. S. 28]. So baut beispielsweise die Android Laufzeitumgebung auf dem Linux Kernel auf und

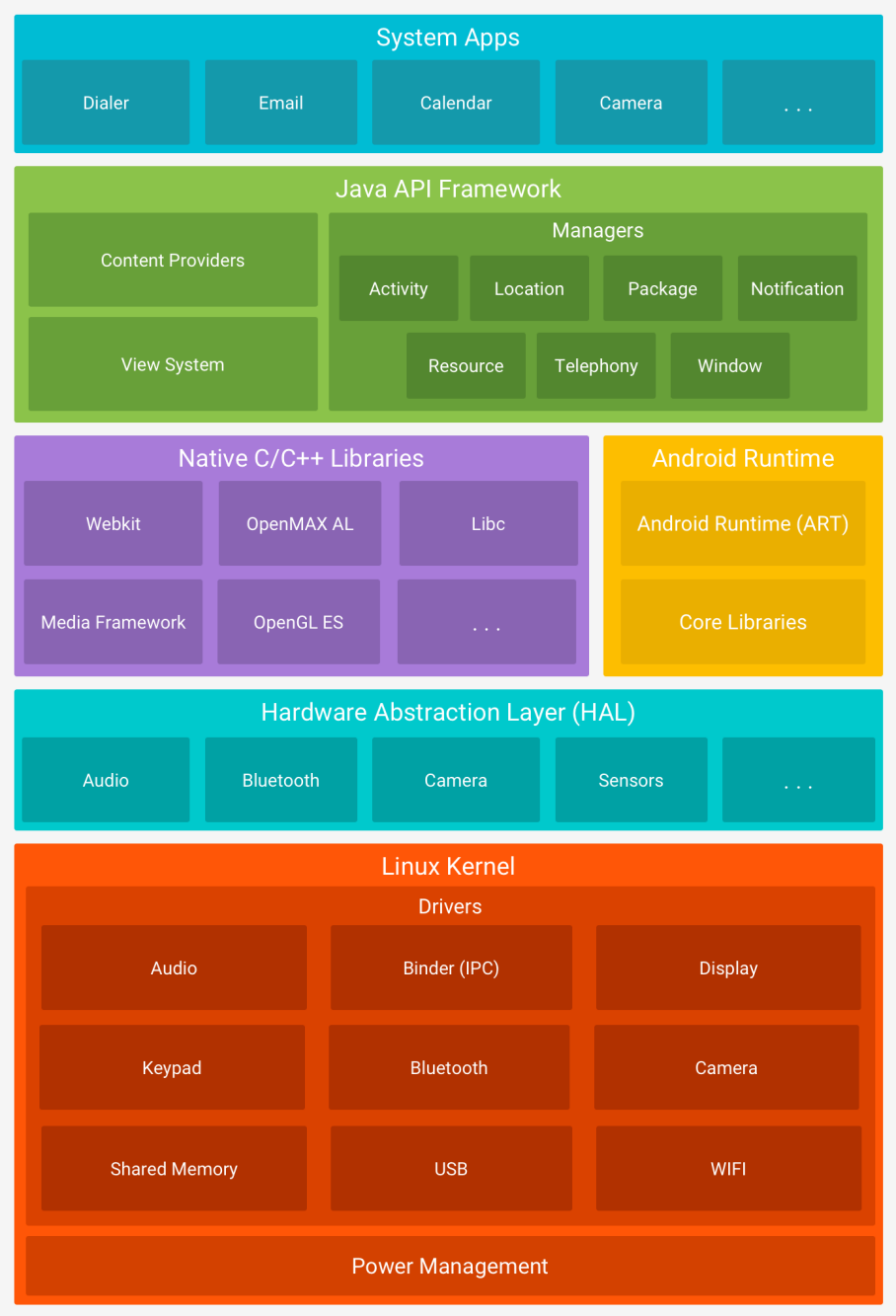


Abbildung 1.1: Android Systemarchitektur [7]

verwendet grundlegende Funktionen des Linux Kernels bspw. die Thread-Verwaltung. Ein weiterer Vorteil des Linux Kernel ist, dass es für die Hardwarehersteller einfacher ist, Gerätetreiber zu entwickeln, da der Linux Kernel hinreichend bekannt ist [7]. Entsprechend werden die benötigten Geräte (Speicher, Audio, Bluetooth) über Kerneltreiber im Linux Kernel angebunden.

### 1.2.2 Hardware Abstraction Layer

HAL

Die Hardwareabstraktionsschicht stellt, für die darüberliegenden Schichten, Schnittstellen für die Nutzung der Hareware-Ressourcen zur Verfügung. Die Abstraktionssicht besteht aus Modulen, welche die Implementierungsdetails zur Verwendung dieser Ressourcen enthalten [7].

### 1.2.3 Android Laufzeitumgebung (Anroid Runtime (ART))

ART

Seit Android Version 5 werden alle Android Anwendungen in einem eigenen Prozess und mit einer eigenen Instanz der Android Laufzeitumgebung ausgeführt. Die Laufzeitumgebung wurde so konzipiert, dass mehrere virtuelle Maschinen auf Geräten mit wenig Speicher den DEX Bytecode ausführen können [7].

### 1.2.4 Native C/C++ Libraries

Ein größerer Teil der Android Komponenten und Services wurden aus nativem Code gebaut und benötigen daher Bibliotheken, welche in C und C++ geschrieben wurden. Diese nativen Bibliotheken können ebenfalls über das Java API Framework verwendet werden [7] .

### 1.2.5 Java API Framework

Java API Framework

Für die Erstellung von Android Anwendungen werden die Komponenten des Java API Frameworks verwendet. Die Bestandteile dieses Frameworks sind in der Java-Syntax geschrieben und dienen als Schnittstelle zu den Komponenten und Services des Systems.

Zu dem Java API Framework gehören [7]:

Komponenten des Java API Framework

- das View-System, welches die Basis für die Erstellung der Benutzeroberflächen ist,
- der Resource Manager, welcher den Zugriff auf Texte, Grafiken und Layout-Dateien verwaltet,
- der Notification Manager, für die Benachrichtungen an den Anwender,
- der Activity Manager, um den Lebenszyklus der Anwendungen zu verwalten,
- der Content Provider - um Zugriff auf andere Anwendungen, beispielsweise Kontakte, zu ermöglichen.

### 1.2.6 System Apps

System-Anwendungen

Die Android Plattform enthält einige System-Anwendungen, beispielsweise für eMail und SMS-Nachrichten, für Kalenderfunktionen oder Kontakte. Die Systemanwendungen werden einerseits von den Anwendern verwendet, liefern andererseits für weitere Anwendungen wiederverwendbare Funktionen, wie beispielsweise den Nachrichtenversand [7].

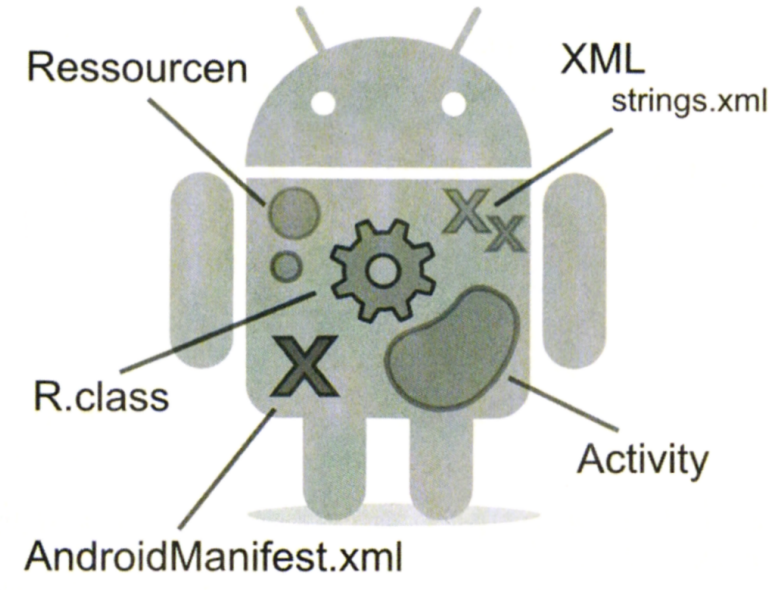


Abbildung 1.2: Bestandteile einer Android App [8, s. S. 35]

## 1.3 Bestandteile einer Android App

Aufbau einer App

Die Abbildung 1.2 zeigt den schematischen Aufbau einer typischen Android App. Demnach besteht jede App, in der Regel, aus folgenden Elementen:

- App Manifest, enthält die notwendigen Informationen, um die Anwendung auf Android auszuführen [9],
- App Bausteine, in Form von Aktivitäten, Fragmenten und Services,
- App Ressourcen, das sind unter anderen Bilder und Texte[10], welche über die sog. R-Klasse geladen werden,
- Benutzeroberfläche (View-System).

### 1.3.1 App Manifest

AndroidManifest.xml

Jede Android App enthält im Hauptverzeichnis die XML-Datei AndroidManifest.xml, welche alle notwendigen Informationen der App enthält [9]. Die Manifestdatei enthält u. A.:

- den Namen des Javapaketes, welches die Klassen der Anwendung enthält,
- den Namen und das Icon des App,
- die Beschreibung der App-Bausteine; dies sind Angaben zu den verwendeten Aktivitäten, Services oder Content-Providern,
- notwendige Berechtigungen auf das System oder auf andere Apps,

- die Android-Versionsnummer, die mindestens benötigt wird, um die App zu starten

Nachfolgend ein Beispiel für eine Manifestdatei:

Beispiel Manifestdatei

Listing 1.1: Beispiel für Manifestdatei

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.allan.myapplication">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

### 1.3.2 App Bausteine

Der API Guide unterscheidet folgende Bausteine [11]:

- Intents: sind Nachrichten zur Kommunikation zwischen App Komponenten,
- Aktivitäten: sind die vom Anwender wahrgenommenen Bausteine einer App [2, s. S. 67],
- Fragmente: sind Komponenten mit eigener Benutzeroberfläche [2, s. S. 143],
- Loaders: sind zum Laden von Daten, von einem Content Provider oder anderen Datenquellen, notwendig,
- Services: kommen zum Einsatz, wenn langläufige Operationen im Hintergrund ausgeführt werden sollen,
- Content Provider: ermöglichen den Austausch von Daten mit anderen Anwendungen,
- Broadcasts: dienen zum Versenden und Empfangen von systemweiten Nachrichten,
- App Widgets: sind kleine Anwendungen, welche auf dem Home screen oder in anderen Anwendungen eingefügt werden können,
- Processes und Threads: gehören zum Linux Betriebssystem und führen die Apps aus.

In den nachfolgenden Kapiteln stelle ich Ihnen die wichtigsten Bausteine einer App etwas näher vor.

#### 1.3.2.1 Aktivitäten

Activity als Architekturmuster

Eine Android-Activity ist eine Einheit der Benutzerinteraktion[...] - als auch eine Einheit der Ausführung. Wenn Sie ein interaktives Android-Programm erstellen, beginnen Sie damit, dass Sie die Klasse Activity erweitern. Activities dienen als wiederverwendbare, austauschbare Teile des Flusses der UI-Komponenten durch Android-Anwendungen [12, s.S. 89]. Unter Android ist das Zerlegen einer App in aufgabenoriente Teile bzw. Funktionsblöcke ein grundlegendes Architekturmuster[...] Eine Anwendung besteht aus mindestens einer solchen Activity, je nach Funktionsumfang können es aber auch viel mehr sein.[...] Activities bilden demnach die vom Anwender wahrgenommenen Bausteine einer App [2, s.S. 67].

Activity für Benutzerinteraktion

Mobile Anwendungen unterscheiden sich von vielen Desktopanwendungen u.A. dadurch, dass Benutzerinteraktionen nicht unbedingt über das gleiche Oberflächenelement gestartet werden. So wird bspw. eine eMail App einerseits über ein Icon auf dem Home Bildschirm geöffnet, andererseits kann das Schreiben einer eMail über eine andere App gestartet werden. Hierzu startet die aufrufende App eine Aktivität in der aufzurufenden Anwendung. Eine Activity erzeugt das Fenster, in dem die App die Oberflächenkomponenten einzeichnet. Das Fenster füllt oft den Bildschirm komplett aus. Eine Activity implementiert daher, für gewöhnlich, einen Bildschirm der Anwendung [13].

In der Manifestdatei (Listing 1.1) wurde die Activity `MainActivity` angegeben. Eine Android Aktivität wird von der Klasse Activity abgeleitet. Die Aktivität besteht aus verschiedenen Methoden, welche als Phasen des Lebenszyklus einer Aktivität zu verstehen sind. So werden in Listing 1.2 die benötigten Objekte der Benutzeroberfläche während der Erstellung der Activity-Klasse bereitgestellt, in dem die Objekte beim Aufruf der `onCreate`-Methode instanziert werden.

Listing 1.2: Beispiel für Activity

```
public class MainActivity extends Activity {

    private TextView    nachricht;
    private Button      weiterFertig;
    private EditText    eingabe;
    private boolean     ersterKlick;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        ...
    }
```

#### 1.3.2.2 Fragmente

Bausteine innerhalb von Activities

Fragmente sind Komponenten mit eigener Benutzeroberfläche und eigenem Lebenszyklus. Sie sind Bausteine innerhalb von Activities. Sie werden wie Views und ViewGroups entweder in Layoutdateien definiert oder per Code erzeugt. Fragmente werden stets im Kontext einer Activity ausgeführt und können nicht losgelöst von ihr verwendet werden [2, s.S. 143].

Mehrere Fragmente in einer Aktivität

Für eine Aktivität können mehrere Fragmente miteinander kombiniert werden, um innerhalb des Fensters mehrere Bereiche anzuzeigen. Hierbei kann jeder dieser Bereiche zu einem Fragment gehören, aber die Fragmente gehören zusammen zu einer Aktivität. Die Fragmente können aber in mehreren Aktivitäten verwendet werden [14]. Der Zusammenhang zwischen Aktivitäten und Fragmenten ist in der Abbildung 1.3 veranschaulicht. Demnach besteht für eine Anwendung, mit größerem Bildschirm, die Aktivität A aus den Fragmenten A und B, während die gleiche Anwendung, bei einem kleineren Bildschirm, aus zwei Aktivitäten A und B besteht, welche jeweils eines der Fragmente enthalten.

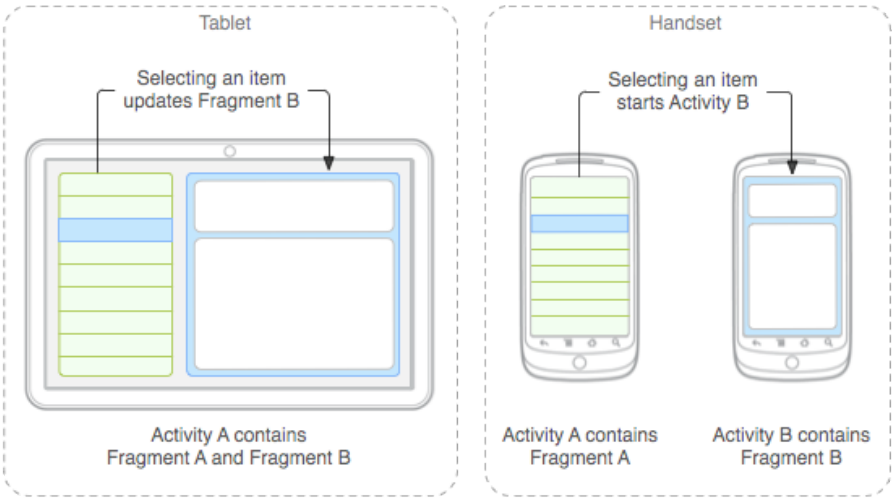


Abbildung 1.3: Aktivitäten und Fragmente [14]

#### 1.3.2.3 Services

Broadcast Receiver

Broadcast Receiver sind Komponenten, die auf systemweit versandte Nachrichten reagieren. Android verschickt solche Mitteilungen zum Beispiel, wenn der Batteriestand niedrig ist oder der Bildschirm ausgeschaltet wurde [2, s.S.137]. Apps können ebenfalls derartige Nachrichten an das Betriebssystem oder an andere Apps versenden, welche nach Erhalt der Nachricht reagieren können.

Services

Services sind Anwendungsbausteine, die ohne Benutzeroberfläche auskommen. Anders als beispielsweise Broadcast Receiver werden sie aber nicht nur beim Eintreten eines Ereignisses aktiviert. Auch sind Services - im Gegensatz zu Broadcast Receivern - gerade für länger andauernde Tätigkeiten gedacht [2, s.S. 238].
Services können beispielsweise im Hintergrund Netzwerk-Transaktionen verarbeiten, Musik abspielen oder Daten einlesen oder Schreiben [15].

Services-Typen

Der Android API Guide unterscheidet drei Typen von Services [15]:

- Foreground Services führen Operationen aus, welche durch den Anwender wahrgenommen werden können. Forground Services müssen währenddessen ein Icon in der Statusbar anzeigen. Ein Foreground Service könnte beispielsweise eine Audiodatei abspielen.
- Background Services führen Operationan aus, welche nicht unbedingt durch den Anwender wahrgenommen werden können. Ein Background Services könnte beispielsweise eine Datei komprimieren.
- Bound Services bieten für andere Komponenten eine Schnittstelle an, damit diese mit dem Bound Service interagieren können. Ein Bound Service ist so lange aktiv, so lange eine andere Komponente mit diesem gekoppelt ist.

**1.3.2.3.1 Service implementieren** Um eine eigene Serviceklasse zu erstellen, wird diese von der Klasse android.app.services abgeleitet. Sofern keine Nebenläufigkeit notwendig ist, sondern Anfragen an den Service nacheinander abgearbeitet werden können, kann der Service auch von android.app.intendservice abgeleitet werden.

Intent Service Implementierung

Listing 1.3: Beispiel für einen Intent Service

```
import android.app.IntentService;
import android.content.Intent;
import android.widget.Toast;

public class HelloIntentService extends IntentService {

    public HelloIntentService() {
        super("HelloIntentService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        try {
            Toast.makeText(this, "Intent␣Service␣gestartet",
                      Toast.LENGTH_SHORT).show();
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            // Restore interrupt status.
            Thread.currentThread().interrupt();
        }
    }

    @Override
    public void onDestroy() {
        Toast.makeText(this, "Intent␣service␣done", Toast.LENGTH_SHORT).show();
    }
}
```

**1.3.2.3.2 Service deklarieren** Damit ein Service innerhalb der Anwendung verwendet werden kann, ist dieser in der Manifestdatei zu deklarieren:

Intent Service deklarieren

Listing 1.4: Services in Manifestdatei

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.allan.myapplication">
    <application
        ...
        <activity android:name=".MainActivity">
            ...
        </activity>
        <service android:name="service.HelloIntentService"></service>
    </application>
</manifest>
```

**1.3.2.3.3 Service starten** Um einen Service starten zu können, wird die Methode startService aus dem ContextWrapper verwendet. Die Methode erwartet als Eingabeparameter ein Intentobjekt, welches zuvor mit der, zu startenden, Serviceklasse zu instanzieren ist. Der Services wird nun zum gewünschten Zeitpunkt ausgeführt.

Intent Service starten

Listing 1.5: Service innerhalb einer Aktivität starten

```
    private void startIntentService() {
        Intent intent = new Intent(this, HelloIntentService.class);
        startService(intent);
    }
```

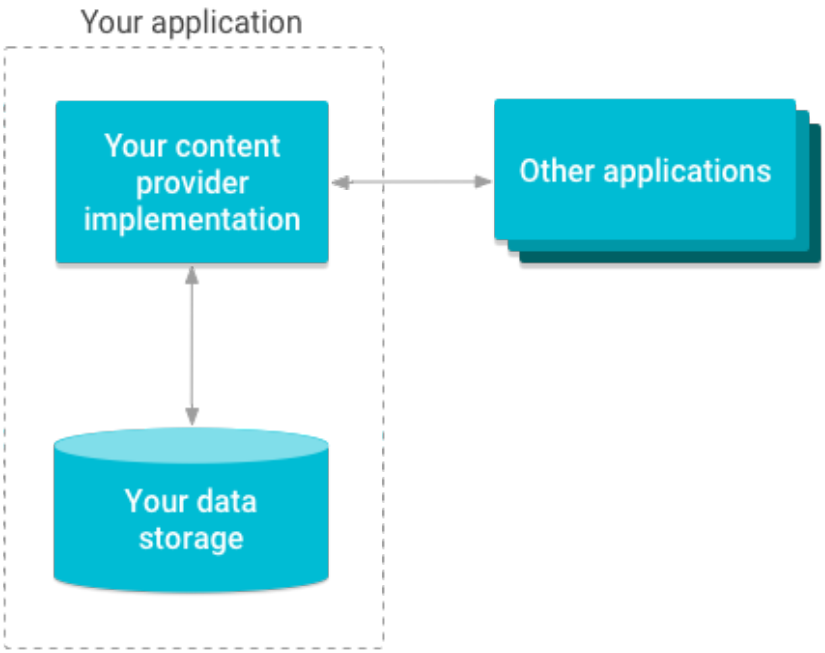


Abbildung 1.4: Datenzugriff mit Content Provider [16]

#### 1.3.2.4 Content Provider

Content Provider unterstützen Anwendungen bei der Verwaltung von Datenzugriffen, die sowohl durch andere Anwendungen erfolgen, als auch den Zugriff auf Daten anderer Anwendungen. Content Provider sind eine Standard-Schnittstelle, welche Daten in einem Prozess anfordern und in einem anderen Prozess verarbeiten können [16]. Damit eine Anwendung Daten bereitstellt, wird die Content Provider Schnittstelle implementiert, sodass andere Anwendungen über einen Schnittstellenaufruf einen geschützten Zugriff auf die bereitgestellten Daten erhalten können. Der Datenzugriff über Content Provider ist in der Abbildung 1.4 veranschaulicht.

Standard-Schnittstelle

Das Android Framework enthält Content Provider, um beispielsweise auf Audio- und Videodaten, auf Bilder oder auf persönliche Kontaktinformationen zugreifen zu können. Hierbei ermöglicht die Content Provider Schnittstelle eine granulare Vergabe und Kontrolle von Zugriffsberechtigungen.

Ein Content Provider repräsentiert die Daten, aus anderen Anwendungen, in einer oder mehreren Tabellen, vergleichbar mit den Tabellen in einem relationalen Datenbanksystem. Eine Zeile in dieser Tabelle entspricht hierbei einer Instanz eines Datentyps, welcher über den Content Provider verwaltet wird [17].

### 1.3.3 App Ressourcen

Die Ressourcen einer App sind die Quelldateien der Benutzeroberfläche sowie die darin verwendeten Bilder, Farben sowie Beschriftungen und Texte. Ressourcen sollten vom Quellcode der Anwendungen getrennt werden, um diese unabhängig von den Funktionen der Anwendung anpassen zu können. Das Verwenden von Ressourcen erleichtert desweiteren die Anpassung der

Benutzeroberfläche an verschiedenen Bildschirmgrößen unterschiedlicher Geräte oder an sprach- und länderspezifische Gegebenheiten [18].

#### 1.3.3.1 Ressourcen bereitstellen

Für jede Art von Ressourcedateien wird üblicherweise ein Unterordner angelegt, welcher die verwendeten Dateien enthält. Der Ablageort für Ressourcedateien ist das /res-Verzeichnis des Projektordners. Ein Beispiel für ein Ressourcenverzeichnis ist nachfolgend ersichtlich:

Beispiel für res-Verzeichnis

Listing 1.6: Ressourcen in Android Projekt[19]

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            graphic.png
        layout/
            main.xml
            info.xml
        mipmap/
            icon.png
        values/
            strings.xml
```

Es gibt verschiedene Arten von Ressourcedateien, welche in folgenden Ordnern abgelegt werden:

Ressource-Verzeichnisse

- drawable - enthält Bilddateien und Zeichnungen
- layout - enthält Quelldateien der Benutzeroberfläche
- mipmap - enthält Bilddateien in verschiedenen Größen für skalierbare Oberflächen
- values - enthält verschiedene XML-Dateien, welche Werte für die Anwendung enthalten:
  - arrays.xml - enthält Datenfelder mit Werten
  - colors.xml - enthält Farbcodes für Texte oder weitere Oberflächenelemente
  - strings.xml - enthält Texte und Beschriftungen
  - styles.xml - enthält Parameter für den Style der Benutzeroberfläche, bspw. Schriftart oder Abstände

Zum besseren Verständnis folgt nun ein Beispiel einer strings.xml-Ressourcedatei:

strings.xml

Listing 1.7: Beispiel für strings.xml [2]

```
<resources>
    <string name="app_name">Hallo Android!</string>

    <string name="willkommen">Guten Tag. Vielen Dank dass Sie mich
    gestartet haben.
    Bitte verraten Sie mir Ihren Namen.</string>

    <string name="hallo">
        Hallo %1$s. Ich freue mich, Sie kennenzulernen.
    </string>
    <string name="weiter">Weiter</string>
    <string name="fertig">Fertig</string>
    <string name="vorname_nachname">Vorname Nachname</string>
</resources>
```

Diese Ressourcendatei enthält u. A. Beschriftungen für Eingabekomponenten und Texte für anzuzeigenden Meldungen. Die Elemente der Ressourcedatei werden über den Namen identifiziert und können im Quellcode der Anwendung mit den gewünschten Oberflächenelementen verknüpft werden. Die Datei enthält beispielsweise einen Stringwert mit dem Namen app_name , welcher in einer Anwendung ausgelesen werden könnte, um den anzuzeigenden Namen der Anwendung zu definieren.
Bei dem dritten Eintrag aus der Ressourcedatei handelt es sich um einen Begrüßungstext, welcher einen Platzhalter für den Namen des Anwenders enthält. Dieser Platzhalter kann zur Laufzeit mit dem eingegebenen Namen des Anwenders ersetzt werden.

#### 1.3.3.2 Ressourcen verwenden

Um innerhalb des Quelltextes auf Ressourcen zugreifen zu können, wird die Klasse R verwendet. Die Abbildung 1.5 zeigt, wie unter Verwendung der Klasse R der Zugriff auf den in der Ressourcedatei hinterlegten Begrüßungstext möglich ist. Die Entwicklungsumgebung Android Studio bietet hier bei der Quelltextanzeige den Komfort, dass nach Deklaration des Textes R.string.willkommen dieser bereits angezeigt wird, obwohl der Text erst zur Laufzeit in die Komponente eingesetzt wird.

```
nachricht.setText("Guten Tag. Schön dass Sie mich gestartet haben. Bitte v...");
nachricht.setText(R.string.willkommen);
```

Abbildung 1.5: Klasse R für Zugriff auf Ressourcen

### 1.3.4 Benutzeroberfläche

UI Komponenten und Layout

Die Benutzeroberfläche einer App ist alles, was dem Anwender auf dem Bildschirm angezeigt wird und mit dem der Anwender interagieren kann. Das Android SDK enthält die gebräuchlichsten UI Komponenten, sowie Objekte für ein strukturiertes Layout, um daraus die graphische Benutzeroberfläche möglichst komfortabel zu konstruieren. Android enthält darüberhinaus UI Module für spezielle Interaktionen wie beispielsweise Dialoge, Benachrichtungen oder Menüs [20].

#### 1.3.4.1 Viewsystem

ViewGroup und View

Unter Android sind alle Bedienungselemente direkte oder indirekte Unterklassen der Klasse android.view.View. Jeder View belegt einen rechteckigen Bereich des Bildschirms. Seine Position und Größe wird durch Layouts bestimmt, die wiederum von android.view.ViewGroup erben, welche ebenfalls ein Kind von View ist[...] Die Text- und Eingabefelder sowie die Schaltflächen[...]sind also Views [2, s.S. 63]. Die Abbildung 1.6 zeigt die Hierarchie zwischen Viewgroup- und View-Instanzen auf.

In dem nachfolgenden Listing wird die Deklaration mehrerer View-Komponenten innerhalb einer XML-Datei aus dem Resourcen-Verzeichnis angezeigt:

Verwendung von Views

Listing 1.8: Definition von Views in einer layout-Datei [2]

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
```

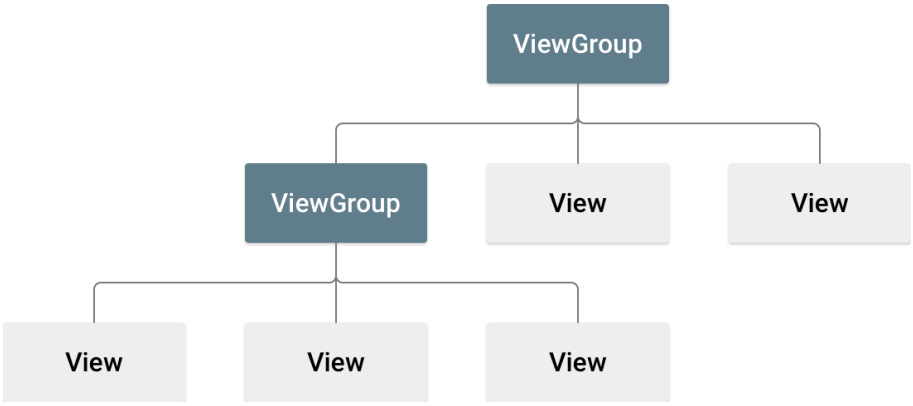


Abbildung 1.6: View Hierarchie [21]

```
    android:layout_height="match_parent"
    android:padding="10dp">

    <TextView android:id="@+id/nachricht"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:lines="2"/>

    <EditText android:id="@+id/eingabe"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/vorname_nachname"
        android:lines="1"
        android:inputType="textCapWords"
        android:imeOptions="actionNext"/>

    <Button android:id="@+id/weiter_fertig"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="end"
        android:layout_marginTop="16dp"/>
</LinearLayout>
```

Diese XML-Datei wird wiederum im Quelltext einer Aktivität als Content-View verknüpft. Der nachfolgende Quelltext zeigt, wie die benötigten Oberflächenobjekte mit den Einträgen dieser XML-Datei instanziert werden:

Listing 1.9: ContentView setzen und Komponenten instanziieren

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        nachricht = (TextView) findViewById(R.id.nachricht);
        weiterFertig = (Button) findViewById(R.id.weiter_fertig);
        eingabe = (EditText) findViewById(R.id.eingabe);
        ...
```

Bei den verwendeten Komponenten handelt es sich um eine Komponente zur Ausgabe eines Textes, eine Komponente zur Eingabe eines Textes, sowie um eine Schaltfläche.

#### 1.3.4.2 Layout

Ein Layout definiert die visuelle Struktur einer Benutzeroberfläche, beispielsweise einer Oberfläche für eine Aktivität. Layouts können, wie bereits gezeigt, über eine XML-Datei definiert werden oder zur Laufzeit über den Quelltext erstellt und hinzugefügt werden. Der Vorteil an der Verwendung von Layoutdateien ist die strikte Trennung zwischen der Darstellung der Anwendung und dem Kontrollfluss. Dies erleichtert die Anpassung des Layouts einer Oberflächenkomponente, ohne den eigentlichen Quellcode der Anwendung verändern zu müssen [22].

Android bietet verschiedene Layout-Typen, welche die Gestaltung von Benutzeroberflächen erleichtern. Die gebräuchlichsten Layouttypen werden hier nun vorgestellt.

**1.3.4.2.1 LinearLayout** Ein LinearLayout ist, wie der Name andeutet, ein View, der seine Kinder in einer Reihe oder einer Spalte anzeigt. Wie die Anzeige erfolgt, wird durch eine Eigenschaft festgelegt, die die Orientierung steuert. Die Kind-Views werden in der Reihenfolge angezeigt, in der sie dem LinearLayout hinzugefügt wurden (unabhängig von der Reihenfolge, in der sie erstellt wurden), und in der Richtung, die westlichen Lesern vertraut ist: von links nach rechts und von oben nach unten [12, s.S. 198]. Das wesentliche Konzept des LinearLayout zeigt die Abbildung 1.7.

Abbildung 1.7: LinearLayout [22]

**1.3.4.2.2 RelativeLayouts** RelativeLayouts beschreiben die Lage von Views in Relation zu anderen Bedienelementen. Hierfür werden Positionsangaben wie android:layout_below oder auch android:layout_toendOf verwendet. Entsprechende Oberflächenbeschreibungen sind nicht ganz so leicht zu lesen wie die Ihnen bereits bekannten LinearLayouts, benötigen aber zur Laufzeit weniger Speicher und werden schneller verarbeitet [2, s. S. 85]. Die Abbildung 1.8 zeigt ein Beispiel für ein relatives Layout an.

Abbildung 1.8: RelativeLayout [22]

**1.3.4.2.3 List View** Eine List View ist eine Komponente zur Darstellung einer Liste von Elementen. Die aufgelisteten Elemente werden über einen Adapter mit der List View verknüpft und angezeigt. Die Elemente werden mithilfe eines Adapters aus einem Datenfeld oder aus einer Datenbankabfrage geladen und für die Anzeige in der List View konvertiert [22]. Die Abbildung 1.9 zeigt das Konzept der List View auf.

Abbildung 1.9: List View [22]

**1.3.4.2.4 Grid View** Eine Grid View zeigt Oberflächenelemente in einem zweidimensionallen Gitter an. Die Elemente werden über einen ListAdapter eingefügt [22]. Die Abbildung 1.10 zeigt das Konzept der Grid View auf.

### 1.3.4.3 Eingabeelemente

Das Android SDK enthält eine Reihe vorgefertigter Eingabe- und Steuerkomponenten, welche die Gestaltung von Benutzeroberflächen erleichtern. Die Android API unterscheidet sieben Typen von Eingabeelementen. Diese werden in den nachfolgenden Kapiteln kurz vorgestellt [23].

**1.3.4.3.1 Buttons** Ein Button oder auch Schaltfläche, ist mit einem Text oder einem Icon beschriftet und wird durch den Anwender betätigt, um eine bestimmte Aktion zu starten. Die

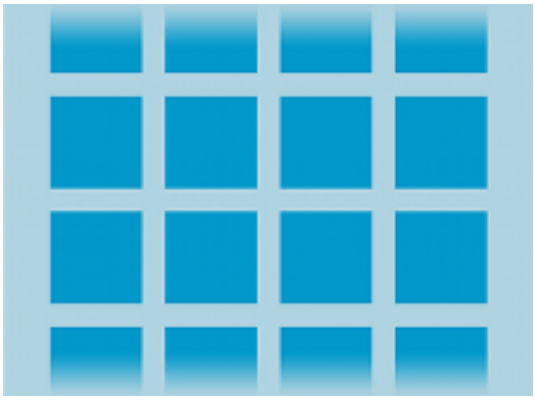

Abbildung 1.10: Grid View [22]


Abbildung 1.11: Button types [23]

Abbildung 1.11 zeigt Typen dieser Schaltflächen an. In dem Listing 1.8 auf Seite 16 wurde gezeigt, wie das Layout eines Buttons konfiguriert werden kann. In diesem Beispiel wurde zunächst eine ID festgelegt, um die Komponente im Quelltext identifzieren zu können. Desweiteren wurde die Höhe und Breite der Komponente auf den Wert 'wrap_content' gesetzt.
Der Wert wrap_content [...] bedeutet, dass sich die Größe aus dem Inhalt der View ergibt, beispielsweise aus der Beschriftung einer Schaltfläche. Die Zeile android:layout_gravity='end' sorgt dafür, dass die Schaltfläche rechtsbündig angeordnet wird [2, s.S. 65]. Die Angabe margin_Top bezieht sich auf den Abstand oberhalb der Komponente und ist in diesem Beispiel in der Einheit dp angegeben. Werte, die auf dp enden, geben [...] geräteunabhängige Pixelgrößen an. Sie beziehen die Auflösung der Anzeige eines Gerätes mit ein [2, s.S. 74].

Die Aktion nach Betätigung der Schaltfläche wird über einen OnClickListener im Quelltext implementiert. Die Abbildung 1.10 zeigt eine mögliche Definition eines solchen Listeners an. Hierbei wird die Methode onClick überschrieben, um mithilfe einer boolschen Variable zu überprüfen, ob dieser Button zum ersten Mal betätigt wurde. Sofern der Button zum ersten Mal betätigt wurde, so wird ein Begrüßungstext angezeigt.

Button-Listener

Listing 1.10: OnClickListener auf Button

```
weiterFertig.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (ersterKlick) {
            nachricht.setText(getString(R.string.hallo, eingabe.getText()));
            eingabe.setVisibility(View.INVISIBLE);
            weiterFertig.setText(R.string.fertig);
            ersterKlick = false;
        } else {
            finish();
        }
    }
});
```

**1.3.4.3.2 Textfeld** Textfelder ermöglichen dem Anwender die Eingabe und Bearbeitung von Zeichen und Wörtern. Wenn ein Textfeld berührt wird, zeigt Android für die Eingabe des gewünschten Textes eine Bildschirmtastatur an. In der Abbildung 1.12 wird der Anwender zur Eingabe seines Namens aufgefordert. Die Beschriftung der Textfelder erleichtert die Eingabe an den hierfür vorgesehenen Stellen der Benutzeroberfläche. Darüberhinaus kann die Eingabe des Textes auf bestimmte Zeichen oder auf eine bestimmte Länge eingeschränkt werden [23].

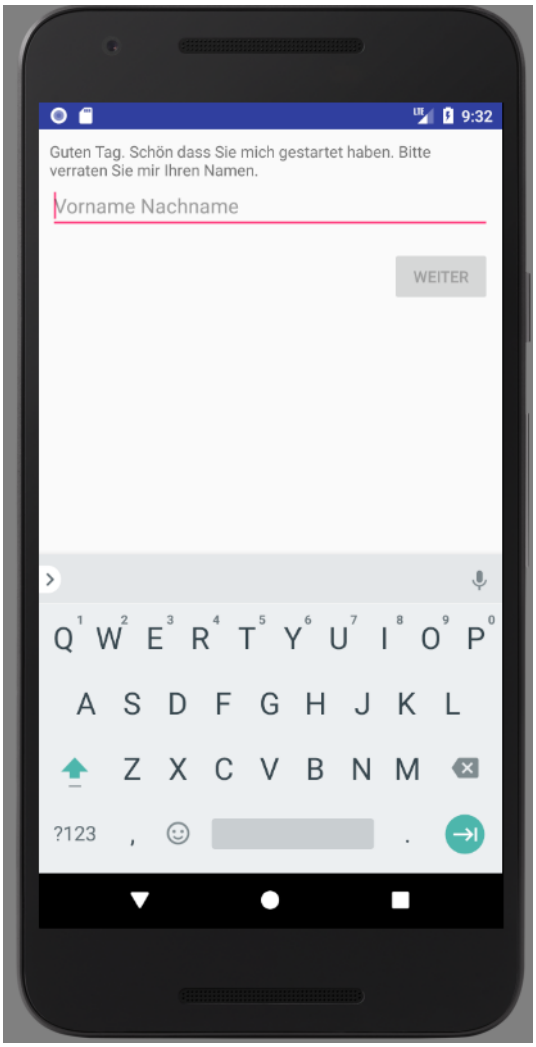


Abbildung 1.12: Eingabe über Textfelder [2, s.S. 75]

**1.3.4.3.3 Checkboxes** Checkboxen eignen sich zur Abfrage von Eigenschaften mit genau zwei Ausprägungen. Das sind in der Regel Ja/Nein- oder An/Aus-Fragen. Die Abbildung 1.13 zeigt exemplarisch drei Optionen für eine Datensyncronisation, die jeweils mit Checkboxen ein- oder ausgestellt werden.

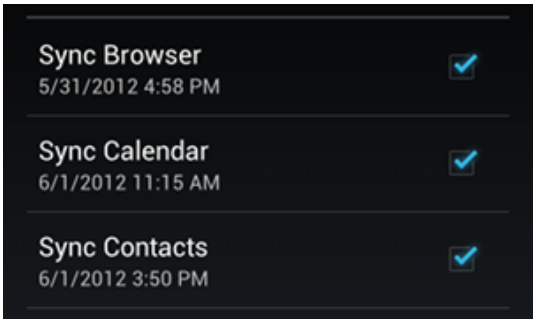


Abbildung 1.13: Eingabe über Textfelder [23]

**1.3.4.3.4 Radio Buttons** Radiobuttons werden verwendet, um aus einer Menge von Möglichkeiten genau eine auszuwählen. Ein Beispiel wird in der Abbildung 1.14 angezeigt.

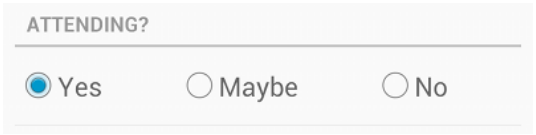


Abbildung 1.14: Auswahl über Radiobuttons [23]

**1.3.4.3.5 Toggle Buttons** Toggle Buttons eignen sich, ähnlich wie Checkboxen, für Ja/Nein- bzw. An/Aus-Fragen. Die wesentlichen Unterschiede sind die Art der Darstellung und die Bedienung. So wird der Zustand dieser Eigenschaft i.d.R. mit einer Beschriftung oder einer Farbe angezeigt. Um den Wert zu verändern, wird die Komponente wie ein Schalter in die gewünschte Richtung bewegt. Die Abbildung 1.15 zeigt einen On/Off-Schalter, welcher zu den moderneren Varianten des Toggle Buttons zählt [23].


Abbildung 1.15: Auswahl über Toggle Buttons [23]

**1.3.4.3.6 Spinners** Mithilfe von Spinner-Komponenten wird ein Wert aus einer Liste von möglichen Werten ausgewählt. Der aktuell ausgewählte Zustand wird in der Komponente angezeigt. Wenn die Komponente berührt wird, so öffnet sich eine Liste mit möglichen Werten, aus welchen ein Wert ausgewählt werden kann. Die Abbildung 1.16 zeigt ein Beispiel für eine solche Komonenten [23].

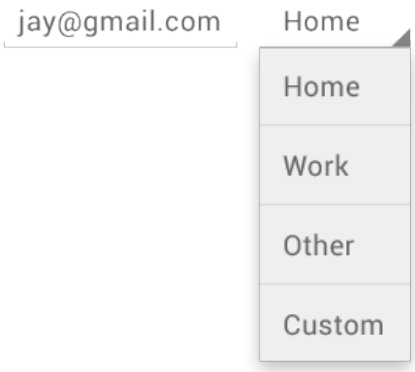


Abbildung 1.16: Auswahl über Spinner [23]

**1.3.4.3.7 Pickers** Picker-Komponenten erleichtern die Eingabe von gültigen Datums- oder Zeitangaben. Die Komponenten ermöglichen einerseits mithilfe eindeutiger Bedienelemente eine komfortable Eingabe, vermeiden andererseits Konvertierungsprobleme, die in Freitextfeldern beispielsweise durch ungültige Angaben oder durch andere Zeit- oder Datumsformate verursacht werden.

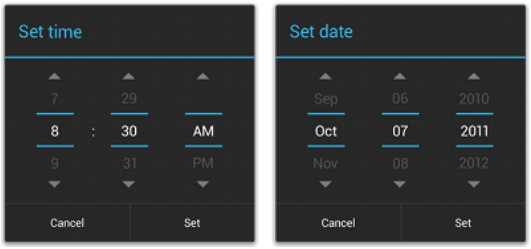


Abbildung 1.17: Auswahl über DateTimePicker [23]

### 1.3.4.4 Menüs

Menüs präsentieren dem Benutzer Funktionen und Aktionen, die er zu einem bestimmten Zeitpunkt ausführen kann [2, s. S.214]. Damit ein Menü überhaupt verwendet werden kann, ist sicherzustellen, dass die Aktivität die app bar bzw. die Action Bar verwenden kann. Die Action Bar ist eines der wichtigsten Designelemente einer Aktivität, da diese eine visuelle Struktur und interaktive Elemente verwendet, welche den meisten Android Anwendern geläufig ist. Durch Verwendung der Action Bar ist die Anwendung mit den meisten anderen Apps konsistent [25]. Die Action Bar enthält folgende Schlüsselfunktionen [25]: Action Bar

- ein fester Ort auf dem Bildschirm, an dem der Name der Anwendung angezeigt wird
- Zugriff auf gebräuchliche Aktionen wie bspw. die Suchfunktion
- Unterstützung zur Navigation und Sichtenwechsel über Reiter oder Drop-/Downlisten

Die Abbildung 1.18 zeigt ein Beispiel einer Action Bar. Diese zeigt den Namen der Anwendung 'Sheets' sowie Icons zum Aufrufen der Suchfunktion und weiterer Funktionen.

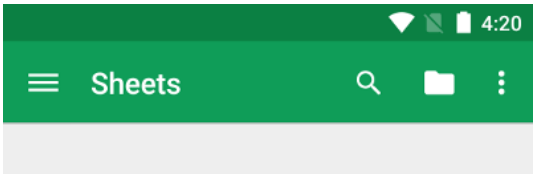


Abbildung 1.18: Action Bar [25]

Damit eine Aktivität eine Actionbar verwendet, muss diese von der Klasse AppCompatActivity abgeleitet werden.

Der Android API Guide unterscheidet zwischen drei fundamentalen Typen von Menüs [24]:

- Optionsmenü
- Contextmenü
- Popupmenü

Diese Menüs werden in den nachfolgenden Unterkapiteln näher vorgestellt.

**1.3.4.4.1 Optionsmenü** Das Optionsmenü ist prinzipiell mit einer klassischen Menüleiste vergleichbar, sollte aber weitaus weniger Elemente enthalten und nur solche Funktionen anbieten, die für die aktuelle Activity sinnvoll sind. Üblicherweise kann der Benutzer eine Einstellungsseite aufrufen oder sich Informationen über die App anzeigen lassen [24]. Wenn eine Aktivität ein Optionsmenü anbieten möchte, so wird innerhalb der Aktivität die Methode onCreateOptionsMenu() überschreiben. Die Struktur des Optionsmenüs wird in einer XML-Datei definiert, welche im Ressourcen-Ordner abgelegt wird und über die R Klasse in den Quelltext eingelesen wird. Mithilfe der XML-Datei werden die Einträge des Optionsmenüs festgelegt, die einzelnen Einträge erhalten eine eindeutige ID und die Beschriftung kann festgelegt werden. Der nachfolgende Quellcode zeigt exemplarisch eine XML-Datei eines Optionsmenüs:

Definition eines Optionsmenüs

Listing 1.11: Defintion eines Optionsmenüs in XML

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/item1"
        android:title="@string/item1" />
    <item android:id="@+id/item2"
        android:title="@string/item2" />
</menu>
```

Die XML-Datei enthält zwei Menüeinträge mit den IDs 'item1' und 'item2'. Die eigentlichen Beschriftungen werden über die Ressourcedatei strings.xml verknüpft.

Die Abbildung 1.19 zeigt ein Optionsmenü mit zwei Menüeinträgen. Diese starten den weiter oben bereits vorgestellten Service sowie den Intentservice.

Damit die Einträge des Optionsmenüs die gewünschte Funktion aufrufen, ist im Quellcode der Aktivität die Methode onOptionsItemSelected zu überschreiben. Die Methode erhält ein MenuItem-Objekt, aus dem die ID des selektierten Items ausgelesen wird. Mithilfe der R-Klasse wird nun geprüft, welches Item gewählt wurde, sodass die gewünschte Funktion aufgerufen wird. Ein Beispiel dieser Funktionszuweisung ergibt sich nachfolgend:

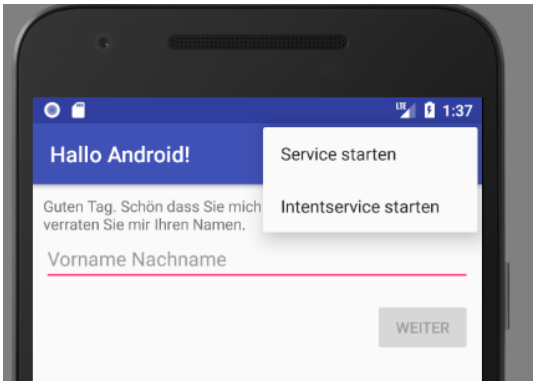


Abbildung 1.19: Optionsmenue in Action Bar

Listing 1.12: onOptionsItemSelected

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.item1:
            startService();
            return true;
        case R.id.item2:
            startIntentService();
            return true;
    }
    return true;
}
```

**1.3.4.4.2 Contextmenü** Ein Contextmenü liefert Optionen, die sich auf ein bestimmtes Element der Benutzeroberfläche oder auf eine bestimmte Aktion beziehen. Contextmenüs können für jedes beliebige View-Element implementiert werden, stehen aber für gewöhnlich in List- oder Gridviews zur Verfügung [24]. Es gibt zwei Möglichkeiten, um ein Contextmenü aufzurufen:

- floating context menu: durch einen langen Klick (Drücken und Halten) erscheint eine floating list und die gewünschte Option kann angetippt werden
- contextual action mode: in diesem Modus enthält die Action Bar Einträge zum jeweiligen Kontext

Das Kontextmenü für eine bestimmte View-Komponente wird mithilfe der Methode `registerForContextMenu(eingabe)` aktiviert. Hierbei ist die Variable `eingabe` eine Instanz des Viewelements, für welche ein Kontextmenü erscheinen soll. Weiterhin ist die Methode `onCreateContextMenu` zu überschreiben. Die Abbildung 1.13 demonstriert, wie das Kontextmenü aus den Ressourcedateien verknüpft werden kann. Um den Einträgen des Kontextmenüs Funktionen zuzuordnen, ist die Methode `onContextItemSelected` zu überschrieben. Es handelt sich um das gleiche Vorgehen, welches bereits bei der Erstellung des Optionsmenüs gezeigt wurde.

Listing 1.13: onCreateContextMenu

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.
    ContextMenuInfo menuInfo) {
```

```
        super.onCreateContextMenu(menu, v, menuInfo);
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.optionsmenu, menu);
    }
```

**1.3.4.4.3 Popupmenü** Ein Popupmenü ist ein modales Menü, welches in Relation zu einer View-Komponente angezeigt wird. Wenn Platz vorhanden ist, dann wird das Menü unterhalb der View-Komponente angezeigt, ansonsten erscheint es oberhalb [24]. Ein Popupmenü wird vorwiegend verwendet, um

- ein Auswahlmenü über den aktuellen Bereich der Benutzeroberfläche zu legen, um eine zum relevanten Kontext passende Aktion auszuwählen,
- eine zweite Auswahlstufe zu implementieren (bspw. erscheint nach Auswahl der Hinzufüge-Schaltfläche ein Popup-Menü mit spezifischen Hinzufüge-Optionen),
- eine DropDown-Auswahl, ähnlich einem Spinner, zu ermöglichen.

Um in einer Aktivität ein Popup-Menü zu verwenden, ist das Interface `OnMenuItemClickListener` zu implementieren, damit die Methode `onMenuItemClick` definiert werden kann. Alternativ kann dieser Listener in einer separaten Klasse implementiert werden.

Das Listing 1.14 zeigt die Implementierung eines Popup-Menüs. Die Methode `showPopup` erhält als Eingabeparameter die Viewkomponente, auf welche sich das Popup-Menü bezieht. Das Popupmenu wird instanziert und mit dem zuvor implementierten Listener verknüpft. Die Methode `show` zeigt schließlich das Popupmenü an.

Popupmenü aufrufen

Listing 1.14: showPopup

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    popup.setOnMenuItemClickListener(this);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.popupmenu, popup.getMenu());
    popup.show();
}
```

In der Abbildung 1.20 wird das Popupmenü unterhalb des Eingabefeldes angezeigt. Die Einträge aus dem Popupmenü können nun gewählt werden, um den gerade benötigten Service zu starten.

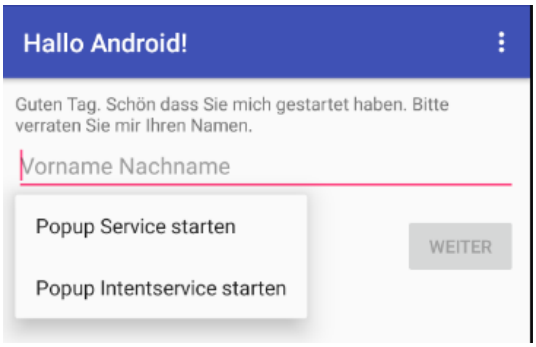


Abbildung 1.20: Popupmenü unter Eingabemaske

#### 1.3.4.5 Dialoge

Ein Dialog ist ein schmales Fenster, welches den Anwender zu einer Entscheidung oder zur Eingabe zusätzlicher Informationen auffordert. Ein Dialog füllt den Bildschirm nicht komplett und wird für gewöhnlich für modale Events verwendet, welche durch den Anwender bearbeitet werden müssen, um die Bearbeitung fortzusetzen [26]. Das Listing 1.15 zeigt ein Beispiel für die Implementierung eines Dialogs. Hierbei wird eine Builderklasse dazu verwenden, um den zu konstruierenden Dialog einer Aktivität zuzuordnen und um die gewünschten Nachrichten und Methoden zu verknüpfen. Um den Dialog aufzurufen, ist dieser zu instanzieren und mit der `show`-Methode anzuzeigen. Die `show`-Methode benötigt als Parameter den FragmentManager, welcher im Kontext einer Aktivität mit der Methode `getFragmentManager()` aufgerufen werden kann. Der zweite Eingabeparameter ist ein Tag, welches aus der Klasse `DialogFragment` gewählt werden kann. In der Abbildung 1.21 wird das fertiggestellte Beispiel eines Dialogs angezeigt.

Implementierung eines Dialogs

Listing 1.15: DialogFragment

```
public class FireMissilesDialogFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
        builder.setMessage(R.string.dialog_fire_missiles)
                .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener
                    () {
                    public void onClick(DialogInterface dialog, int id) {  }
                })
                .setNegativeButton(R.string.cancel, new DialogInterface.
                    OnClickListener() {
                    public void onClick(DialogInterface dialog, int id) {  }
                });
        return builder.create();
    }
}
```

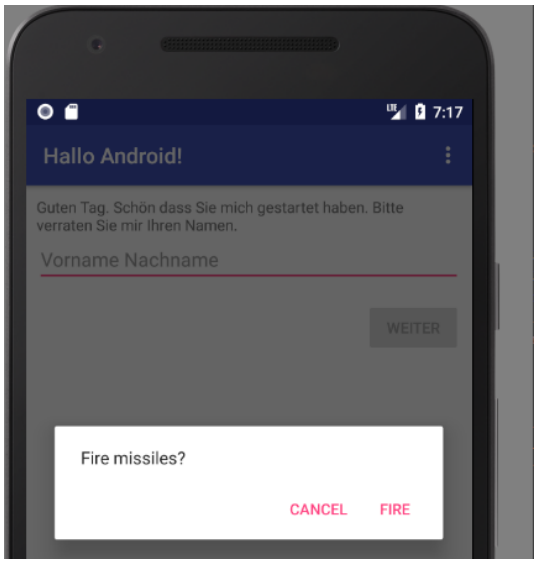


Abbildung 1.21: Anzeige eines Dialogs [2]

# Teil II

# Entwicklungsprozess mit Android Studio

## 2.1 Entwicklungsprozess mit Android Studio

Im zweiten Teil der Seminararbeit wird die Entwicklungsumgebung Android Studio näher vorstellt und ein möglicher Entwicklungsprozess für eine Android App beschrieben.

### 2.1.1 Android Studio

ADT und Android Studio

Google bot mit dem Android Development Tools (ADT) ein äußerst mächtiges Plug-in für Eclipse an, sodass diese IDE lange Zeit der De-facto-Standard für die App-Entwicklung unter Android war. Allerdings hat sich der Suchmaschinenprimus schon vor geraumter Zeit von Eclipse abgewandt und pflegt mit Android Studio eine eigene Android-Entwicklungsumgebung, die auf der IntelliJ IDEA Community Edition der Firma JetBrains basiert [2, s. S. 31]. Die Entwicklungsumgebung Android Studio liefert, gemäß [27], zusätzlich zur IntelliJ-basierten Entwicklungsumgebung, die nachfolgenden Funktionen, um die Entwicklung von Android Apps zu unterstützen:

- ein flexibles Buildsystem, welches auf Gradle basiert [30]
- schneller und featurereicher Emulator für Android-Geräte
- eine einheitliche Umgebung, um Anwendungen für alle Android-Geräte zu entwickeln
- Test-Werkzeuge
- Code-Vorlagen und GitHub-Integration

### 2.1.2 Überblick Android Studio

#### 2.1.2.1 Struktur eines Projektes

Jedes Projekt in Androd Studio besteht aus einem oder mehreren Modulen mit Quell- und Ressourcendateien. Folgende Modultypen sind zu unterscheiden:

Modultypen

- Android App Module - die eigentlichen Apps
- Bibliothek Module - enthalten wiederverwendbaren Code für andere Module
- Google Cloud Module - enthalten Container, um Backend-Code in der Google Cloud bereitzustellen [28]

Die Abbildung 2.22 zeigt beispielhaft die vorgesehende Struktur eines, mit Android Studio entwickelten, Android Projektes an. Im Hauptverzeichnis liegt die Builddatei build.gradle. Die Entwicklungsumgebung unterstützt beim Anlegen des Projektes durch die Erstellung dieses Buildfiles. Das Buildfile enthält u. A. die Angabe der verwendeten SDK-Version, die ApplikationsID sowie die Versionsnummer der App. Desweiteren werden im Buildfile Abhängigkeiten zu Bibliotheken deklariert, der Ablageort für die übersetzten Binärdateien festgelegt und verschiedene Buildtasks angelegt, um temporäre Dateien zu löschen oder Unittests auszuführen.

Jedes Modul enthält folgende Ordner:

- manifests - enthält die im Abschnitt 1.3.1 vorgestellte Manifestdatei AndroidManifest.xml
- java - enthält den Java-Quellcode des Moduls
- res - enthält die im Abschnitt 1.3.3 vorgestellten Ressourcedateien

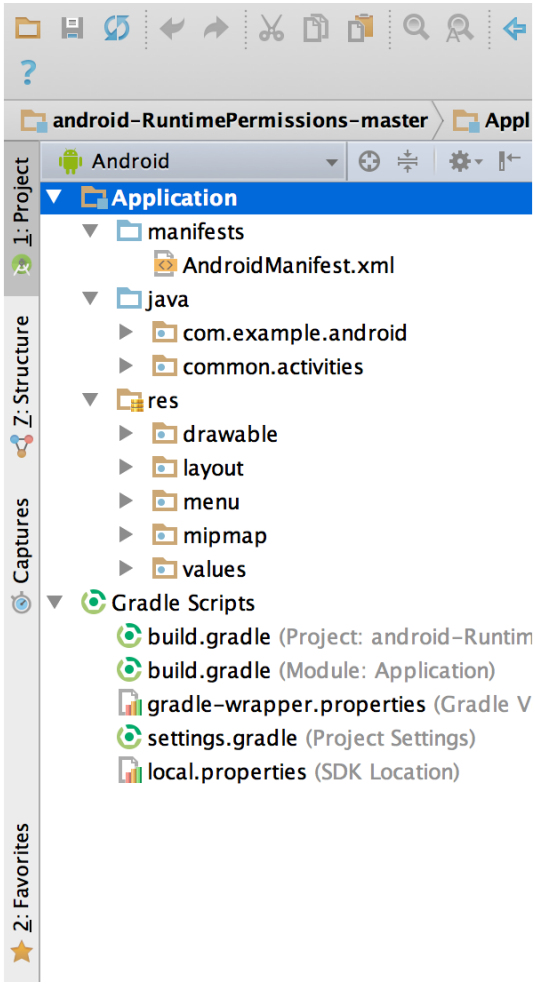


Abbildung 2.22: Android Projektstruktur [27]

#### 2.1.2.2 Android Studio Benutzeroberfläche

Die Abbildung 2.23 aus dem Android API Guide [27] zeigt die einzelnen Bereiche der Benutzeroberfläche von Android Studio. Diese sind nach dem API Guide wie folgt beschrieben:

1. Toolbar - um die wichtigsten Aktionen schnell auszuführen, bspw. den Start der Anwendung
2. Navigationsbar - zeigt den Ordnerpfad der Datei an, die gerade geöffnet ist
3. Quellcodeeditor - zur Anzeige und Bearbeitung von Quelldateien
4. Toolwindowbar - ermöglicht das Auf- und Zuklappen verschiedener Tools, bspw. die Projektsicht zur Anzeige der Projektstruktur oder die Struktursicht, um in der Struktur der geöffneten Quelldatei zu navigieren
5. Tool window - ermöglicht den Zugriff auf weitere Funktionen wie die Versionsverwaltung, verschiedene Funktionen zum Debuggen oder das Aufrufen der Kommandozeile
6. Status bar - zeigt Informationen zum Projekt oder zur Entwicklungsumgebung an, bspw. Warnungen und Fehlermeldungen

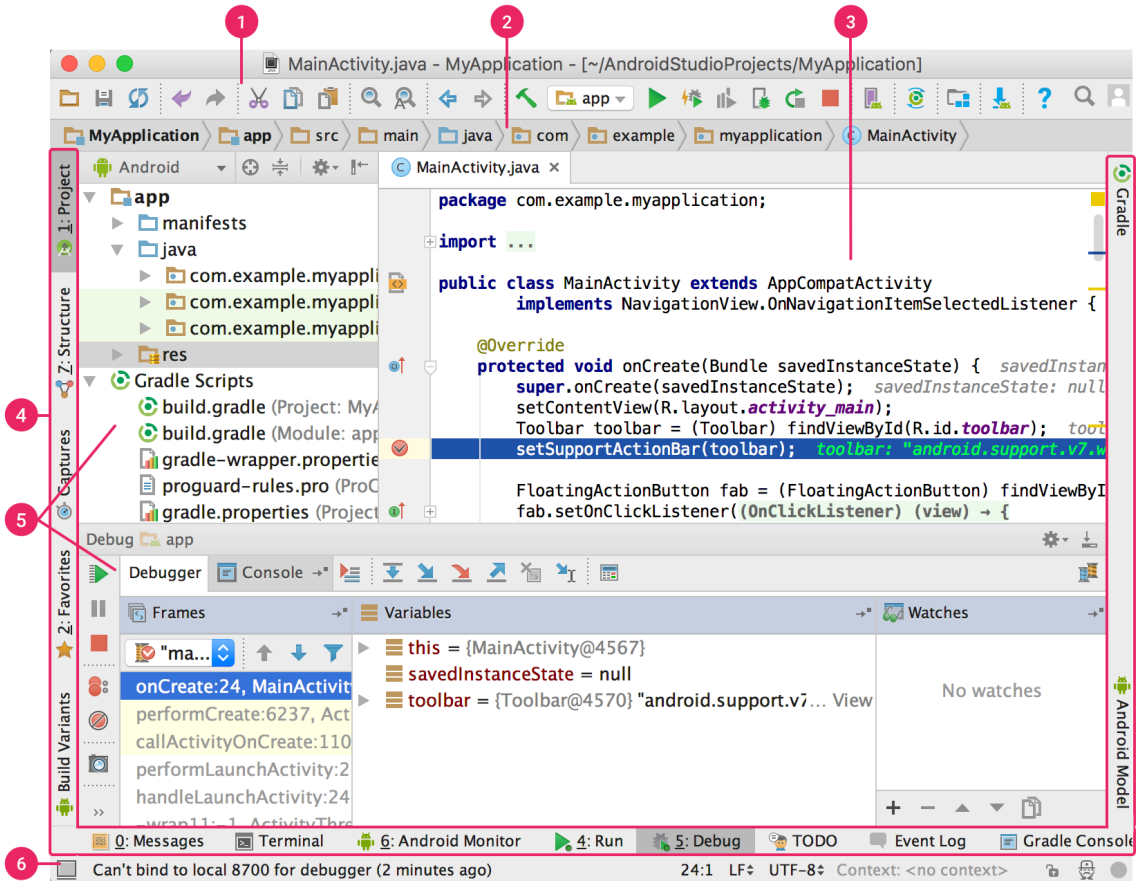


Abbildung 2.23: Benutzeroberfläche [27]

#### 2.1.2.3 Android Projekt einrichten

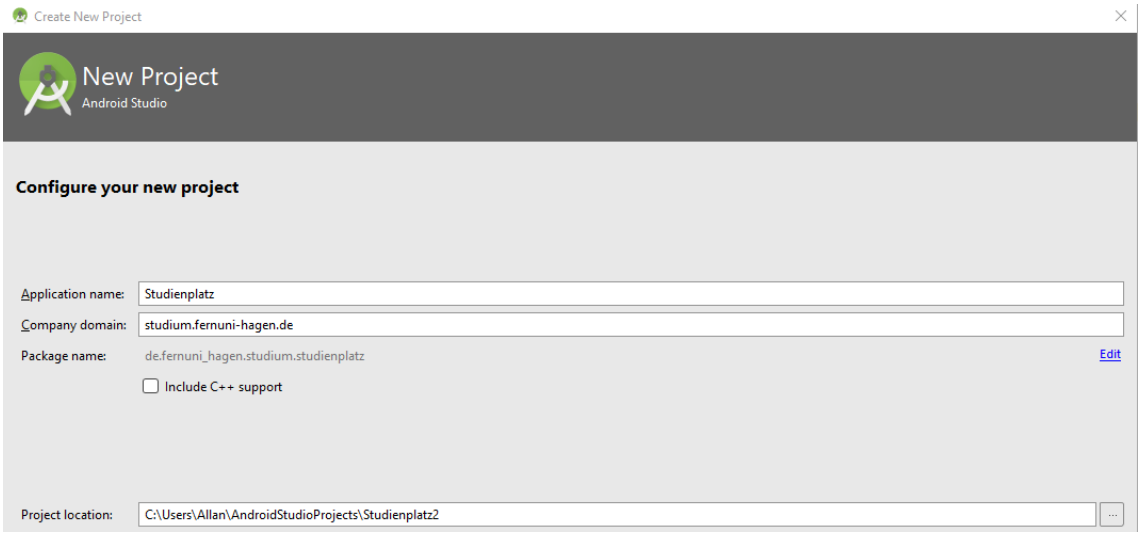


Abbildung 2.24: Projekt anlegen

Für die zu entwickelnde Anwendung wird über die Funktion 'Create New Project' ein Projekt angelegt. Die Abbildung 2.24 zeigt den Erfassungsdialog zur Eingabe des App-Namens sowie der Domäne. Mit diesen Angaben wird ebenfalls der Package-Name der Java-Klassen erstellt.

Im nächsten Schritt des Erfassungsdialogs wird festgelegt, für welchen Gerätetyp die App vorgesehen ist, und welche Version des Android SDK auf dem Gerät mindestens installiert sein muss, um die App auf dem Gerät benutzen zu können, siehe Abbildung 2.25.

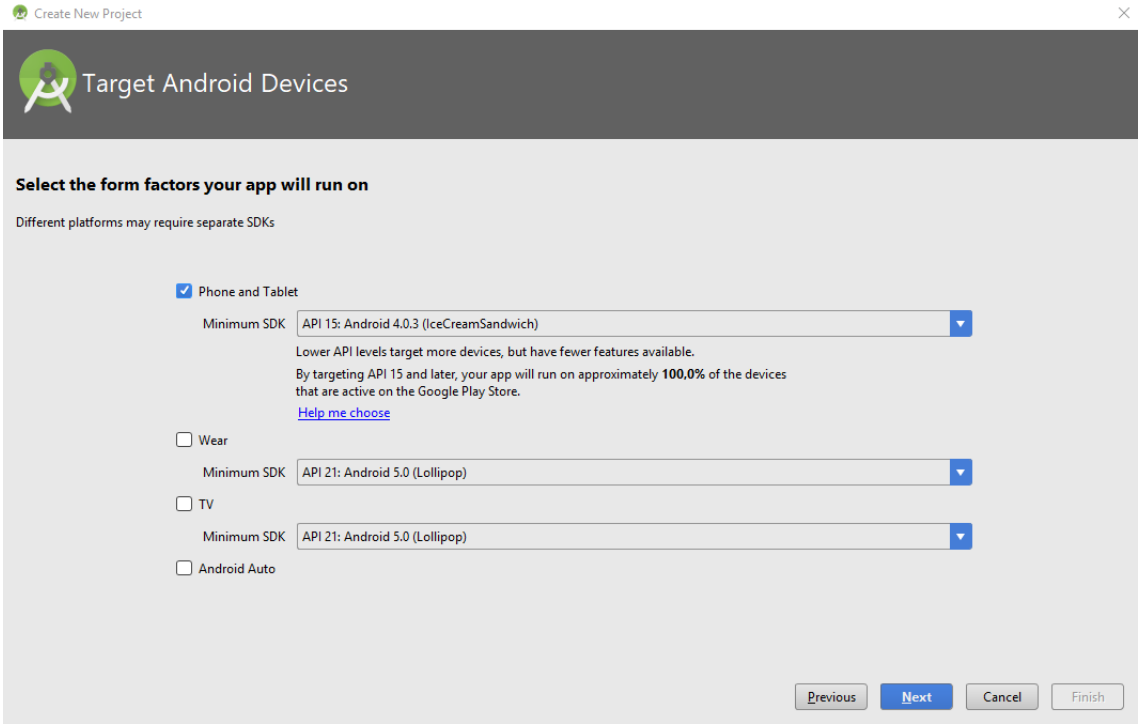


Abbildung 2.25: Gerätetyp wählen

Im nächsten Schritt, siehe Abbildung 2.26, bietet Android Studio eine Auswahl typischer Aktivitäten an. Für die Beispielsanwendung wird die Aktivität 'Login Activity' ausgewählt. Das Projekt wird nun eingerichtet und die erstellten Quelldateien werden kompiliert.

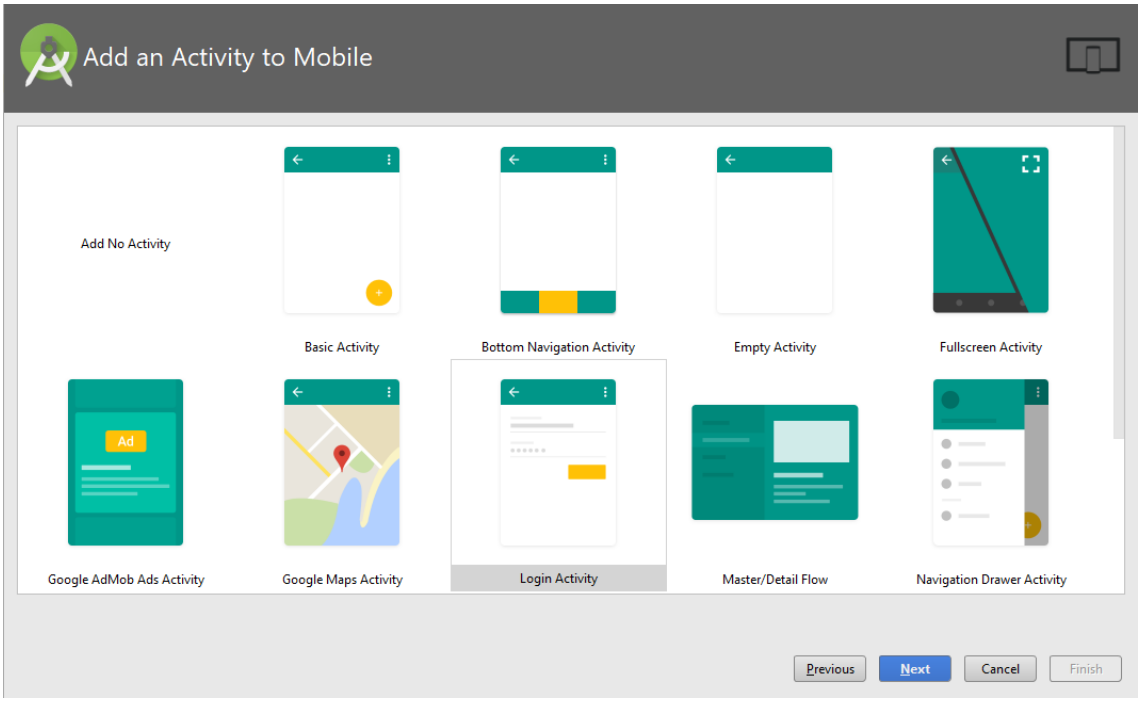


Abbildung 2.26: Activity wählen

Sobald dieser Vorgang abgeschlossen ist, kann die Anwendung über den 'Run'-Button auf einem Android-Gerät emuliert werden, siehe Abbildung 2.27.

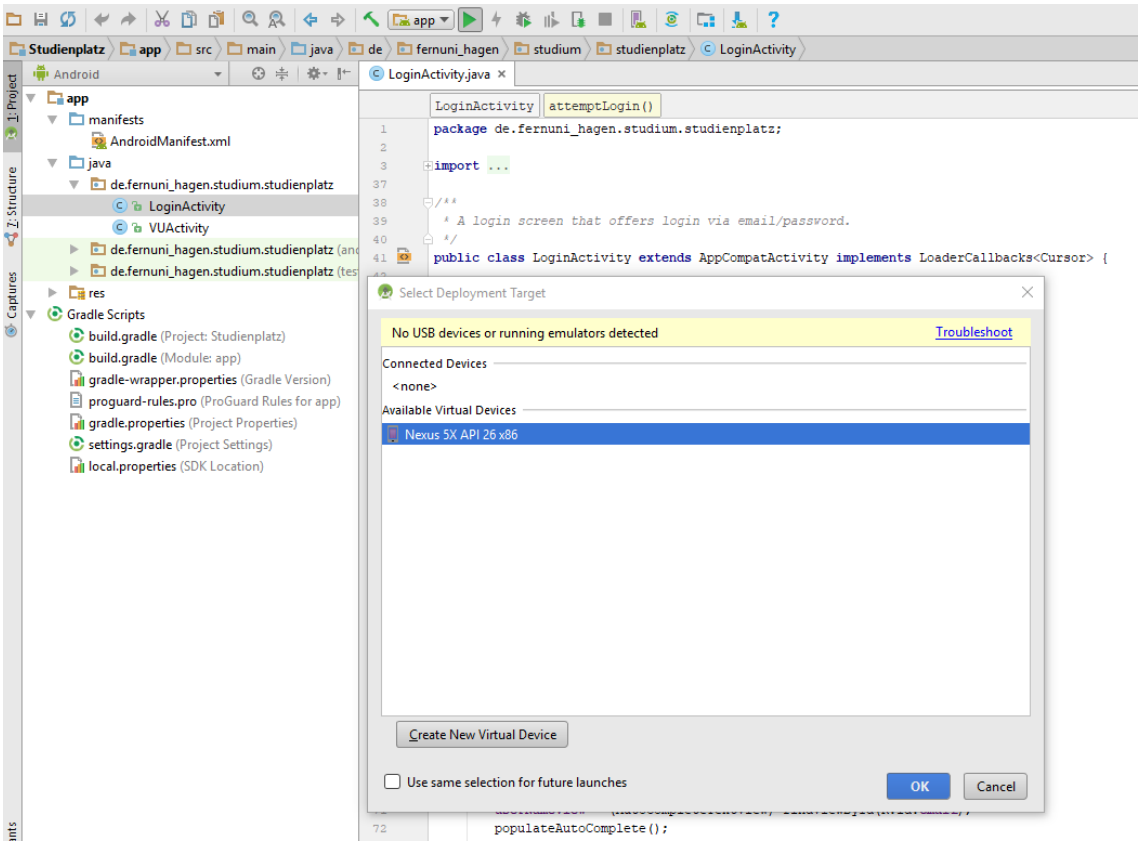


Abbildung 2.27: Anwendung starten

Nach Start des Android-Emulators sollte die ausgewählte Aktivität angezeigt werden. Dies zeigt die Abbildung 2.28.
Zu der eingerichteten Login-Aktivität gehört die Java-Klasse `LoginActivity` sowie die Layout-Datei `activity_login.xml`. Auf Basis dieses Projektes werde ich Ihnen den Entwicklungsprozess einer Beispielanwendung näher erläutern.

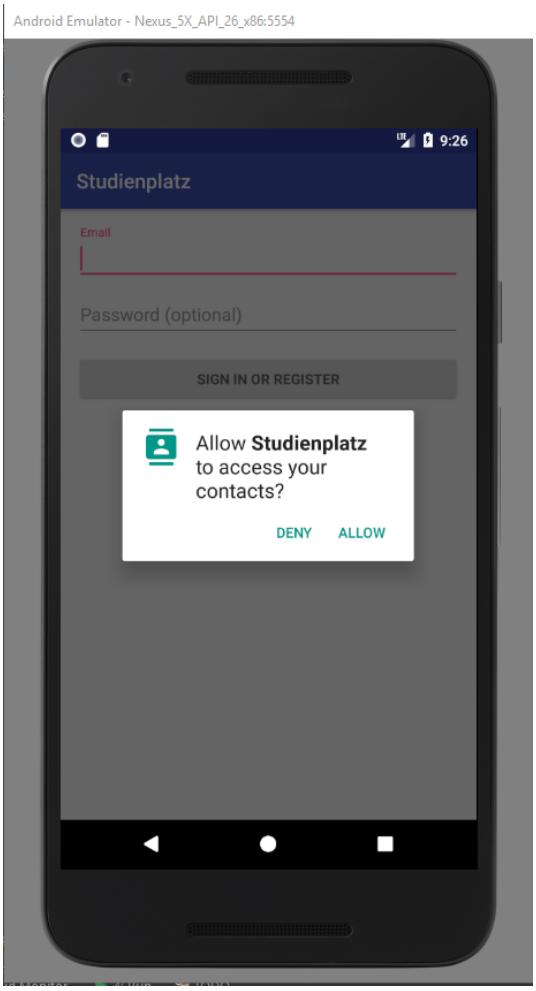


Abbildung 2.28: Anzeige der Anwendung

## 2.2 Konzept Studienplatz App

### 2.2.1 Beschreibung der Studienplatz App

Die Beispielanwendung 'Studienplatz App' zeigt den Virtuellen Studienplatz der Fernuniversität Hagen an [29]. Um den virtuellen Studienplatz zu öffnen, gibt der Anwender zuvor den Benutzernamen und das Kennwort seines Accounts ein, damit eine Anmeldung am virtuellen Studienplatz durchgeführt wird und die Inhalte angezeigt werden. Nach erfolgreicher Anmeldung kann innerhalb der App der virtuelle Studienplatz uneingeschränkt verwendet werden, um bspw. belegte Kurse durch Antippen der Links zu öffnen.

### 2.2.2 Layout

Die Beispielanwendung besteht aus zwei Aktivitäten: Die erste Aktivität dient zur Eingabe des Benutzernamens und des Kennworts und basiert auf der bereits vorhandene LoginActivity. Die zweite Aktivität ist eine WebView und zeigt den virtuellen Studienplatz nach erfolgreichem

Login an.

## 2.3 Benutzeroberfläche

### 2.3.1 LoginActivity

Das Layout der Login-Aktivität kann durch Öffnen der Datei activty_login.xml in Android Studio angepasst werden. Hierzu steht im Reiter Design ein Editor zur Verfügung, um beispielsweise neue Eingabeelemente hinzuzufügen. Im Reiter `Text` wird der erstellte XML-Text angezeigt und kann geändert werden.
Für die benötigte Login-Aktivität wird die Eingabe des Benutzernamens benötigt. Die vorhandene Aktivität sieht jedoch die Eingabe einer eMailadresse vor. Aus diesem Grund wird in die Ressourcedatei `strings.xml` ein neuer Eintrag mit dem Namen `prompt_user` eingegeben, welcher den Text 'Benutzernamen' erhält. Desweiteren wird in dieser Datei die Beschriftung des Buttons in 'Anmeldung am virtuellen Studienplatz' geändert. Die Abbildung 2.29 zeigt die Bearbeitung des Layouts in Android Studio.

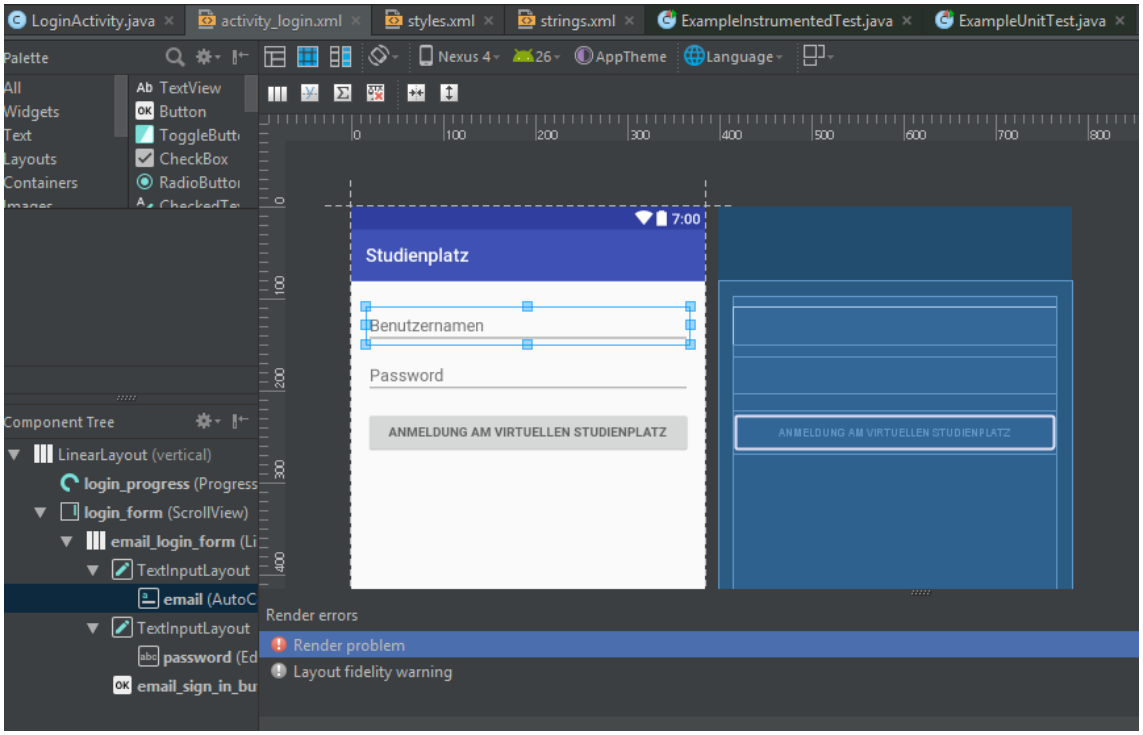


Abbildung 2.29: Anpasung der Login-Oberfläche

### 2.3.2 VUActivity

Für die Anzeige des virtuellen Studienplatz wird eine WebView-Komponente verwendet. Diese zeigt den Inhalt einer Webseite innerhalb einer Android-App an. Die WebView füllt den kompletten Bildschirm aus und zeigt den virtuellen Studienplatz nach erfolgreichem Login an. Für

die Aktivität wird eine weitere Ressourcedatei benötigt, in welcher das LinearLayout festgesetzt wird und die benötigte WebView definiert wird. Diese Layoutdatei könnte wie folgt aussehen:

activity_vu.xml

Listing 2.16: Layout-Datei für VUActivity

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <WebView
        android:id="@+id/webview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

## 2.4 Entwicklung der Studienplatz App

### 2.4.1 LoginActivity

Die LoginActivity wurde bereits angelegt und das Layout der Aktivität wurde bereits im vorherigen Abschnitt an die Anforderungen der Beispielanwendung angepasst. Um die LoginActivity für die Studienplatz App nun fertigzustellen, muss noch die Validierung der Benutzereingaben angepasst werden. Desweiteren muss nach Betätigung der Login-Schaltfläche die zweite Aktivität gestartet werden.

#### 2.4.1.1 Prüfung der Benutzereingaben

Für die Eingabe des Benutzernamens werden folgende Anforderungen festgesetzt:

- der Benutzername darf nicht leer sein
- der Benutzername muss, unabhängig von Groß- und Kleinschreibung, mit Q beginnen
- der Benutzername muss mindestens zweistellig sein
- der Benutzername muss ab der zweiten Stelle vom Typ Ganzzahl sein

Der nachfolgende Quelltext zeigt eine Prüfmethode, welche den genannten Anforderungen genügt:

Überprüfung des Benutzernamens

Listing 2.17: Prüfung des Benutzernamens

```
private boolean isUserValid(String user) {
    if (!user.toUpperCase().startsWith("Q")) {
        return false;
    }
    if (user.length()<3) {
        return false;
    }
    String number = user.substring(1, user.length());
    try {
        int parse = Integer.parseInt(number);
    } catch (NumberFormatException e) {
        return false;
    }
    return true;
}
```

Diese Prüfmethode wird aufgerufen, sobald der Login-Button betätigt wurde und sofern ein Benutzername eingegeben wurde, d. h. eine Nichteingabe des Benutzernamens wurde zuvor bereits abgeprüft. Damit die Prüfung stattfindet, wird in der LoginActivity die bereits vorhandene Methode `attemptLogin()` angepasst, sodass der eigentliche Login-Prozess nur ausgeführt wird, sofern die Prüfmethode die Gültigkeit des Benutzernamens bestätigt.

#### 2.4.1.2 Start der zweiten Aktivität

Nach Betätigung der Anmeldeschaltfläche wird eine zweite Aktivität gestartet, welche den virtuellen Studienplatz in einer WebView anzeigt. Diese Aktivität erhält den Klassennamen `VUActivity.java`. Um innerhalb einer Aktivität eine weitere Aktivität zu starten, wird ein Intent-Objekt benötigt. Das Intent-Objekt erhält als Eingabeparameter die Instanz der aufrufenden Aktivität, sowie den Klassennamen der zu startenden Aktivität. Das Intent-Objekt wird beim Aufruf der Methode `startActivity` als Parameter übergeben. Mit diesem Methodenaufruf wird die Aktivität instanziert und auf dem Bildschirm angezeigt. Damit in der Aktivität `VUActivity` die Anmeldeinformationen zur Verfügung stehen, werden diese mithilfe eines Bundle-Objekts an die Aktivität übertragen. Der Quelltext zum Start dieser Aktivität könnte wie folgt lauten:

Listing 2.18: Start der Aktivität VUActivity

```
showProgress(true);
Bundle param = new Bundle();
param.putString(VUActivity.USERKEY, username);
param.putString(VUActivity.PASSWORDKEY, password);
Intent i = new Intent(LoginActivity.this, VUActivity.class);
i.putExtras(param);
startActivity(i);
```

### 2.4.2 VUActivity

#### 2.4.2.1 Klasse VUActivity.java

Die zweite Aktivität VUActivity.java erbt von der Klasse Activity und enthält, für die Anzeige des virtuellen Studienplatz, eine Instanz vom Typ WebView. Innerhalb der `onCreate`-Methode wird die webView-Variable mit der Layout-Datei verknüpft, sodass Änderungen im Layoutdateien möglich sind, ohne den Quellcode der Aktivität ändern zu müssen. Für die Anzeige des virtuellen Studienplatzes ist es von Vorteil, wenn der Bildschirm rotiert, sofern das Telefon gedreht wird, um den virtuellen Studienplatz komplett und in akzeptabler Größe angezeigt zu bekommen. Um dies zu erreichen werden die Methoden `onSaveInstanceState` sowie `onRestoreInstanceState` überschreiben, damit der aktuellen Zustand der Aktivität zwischengespeichert und nach Rotation des Smartphones wiederhergestellt wird:

Anpassen der Anzeige nach Rotation des Gerätes

Listing 2.19: onSaveInstanceState und onRestoreInstanceState

```
@Override
protected void onSaveInstanceState(Bundle outState )
{
    super.onSaveInstanceState(outState);
    webView.saveState(outState);
}

@Override
protected void onRestoreInstanceState(Bundle savedInstanceState)
{
    super.onRestoreInstanceState(savedInstanceState);
    webView.restoreState(savedInstanceState);
}
```

#### 2.4.2.2 Parameter in Aktivität laden

Damit der eingegebene Username und das Kennwort für den Login-Vorgang zur Verfügung stehen, werden diese mithilfe der Methode `getIntent()` eingelesen. Die benötigten Werte wurden mithilfe von Zugriffsschlüssel in das Bundle-Objekt `Extras` eingetragen und können mithilfe dieser Zugriffsschlüssel wieder ausgelesen werden. Das Listing 2.20 zeigt, wie die benötigten Werte ausgelesen werden, bei den Variablen USERKEY und PASSWORDKEY handelt es sich um Konstanten, welche die Zugriffsschlüssel enthalten.

Auslesen aus Bundle

Listing 2.20: Username und Kennwort auslesen

```
String user = (String)getIntent().getExtras().get(USERKEY);
String password = (String) getIntent().getExtras().get(PASSWORDKEY);
```

#### 2.4.2.3 WebView verwenden

Das WebView-Objekt enthält die Methode `loadUrl`, um den Inhalt einer Webseite in die WebView-Komponente zu laden. Vor dem Aufruf ist zu prüfen, ob bereits eine Instanz der Aktivität vorhanden ist, damit die WebView-Komponente nicht mehrfach neu geladen wird. Nach dem Laden oder Wiederherstellen der WebView-Komponente wird noch der Fokus auf diese Komponente gesetzt, sodass der Anwender mit der eigentlichen Webseite interagieren kann. Der Quellcode zum Laden des virtuellen Studienplatz innerhalb der WebView-Komponente lautet wie folgt:

Webseite in WebVIEW laden

Listing 2.21: Virtueller Studienplatz in WebView laden

```
if (savedInstanceState!=null) {
    webView.restoreState(savedInstanceState);
} else {
    webView.loadUrl
("https://vu.fernuni-hagen.de/lvuweb/lvuauth/app/MyVU?function=Info");
}
webView.requestFocus();
```

#### 2.4.2.4 Anmeldung am virtuellen Studienplatz

Damit eine Anmeldung am virtuellen Studienplatz durchgeführt werden kann, müssen für die WebView-Komponente verschiedene Einstellungen durchgeführt werden. Die WebView Komponente verfügt über ein WebSettings-Objekt, über welches JavaScript eingeschaltet werden kann. Desweiteren wird für die Anmeldung am virtuellen Studienplatz ein Cookie eingerichtet, weshalb, mithilfe der Klasse `CookieManager`, die Einstellungen `AcceptCookie`, sowie `AcceptThirdPartyCookies`, gesetzt werden müssen. Für den virtuellen Studienplatz wird eine sog. Basisauthentifzierung durchgeführt, in welcher, mithilfe eines `HttpAuthHandlers` der Benutzername und das Kennwort einzugeben ist. Damit diese Basisauthentifzierung durchgeführt wird, muss für die WebView ein WebViewClient implementiert werden, in dessen Methode `onReceivedHttpAuthRequest`, der Username und das Kennwort einzutragen sind. Der nachfolgende Quelltext demonstriert, wie die Anmeldung am virtuellen Studienplatz mithilfe des WebViewClients durchgeführt werden kann:

Basisauthentifzierung

Listing 2.22: Anmeldung am virtuellen Studienplatz

```
WebSettings webSettings = webView.getSettings();
        webSettings.setJavaScriptEnabled(true);
        webView.setWebChromeClient(new WebChromeClient());
        CookieManager.getInstance().setAcceptCookie(true);
        CookieManager.getInstance().setAcceptThirdPartyCookies(webView, true);
        webView.getSettings().setJavaScriptCanOpenWindowsAutomatically(true);
```

```
webView.setWebViewClient(new WebViewClient() {

    @Override
    public void onReceivedHttpAuthRequest
 (WebView view, HttpAuthHandler handler, String host, String realm) {

     String user = (String)getIntent().getExtras().get(USERKEY);
        String password = (String) getIntent().getExtras().get(PASSWORDKEY);
        handler.proceed(user, password);
    }

    @Override
    public boolean shouldOverrideUrlLoading
 (WebView view, WebResourceRequest r) {
        view.loadUrl(r.getUrl().toString());
        return true;
    }

    @Override
    public void onPageFinished(WebView view, String url) {
        super.onPageFinished(view, url);
    }
});
```

### 2.4.3 Ergebnispräsentation der Studienplatz App

Nach Fertigstellung der Anwendung bietet die Studienplatz App nun einen Anmeldebildschirm für den virtuellen Studienplatz der Fernuniversität Hagen. Der Anmeldebildschirm wird noch einmal in der Abbildung 2.30 gezeigt. Für die abschließenden Tests wurden die korrekten Anmeldedaten des Accounts Q9502998 eingegeben.

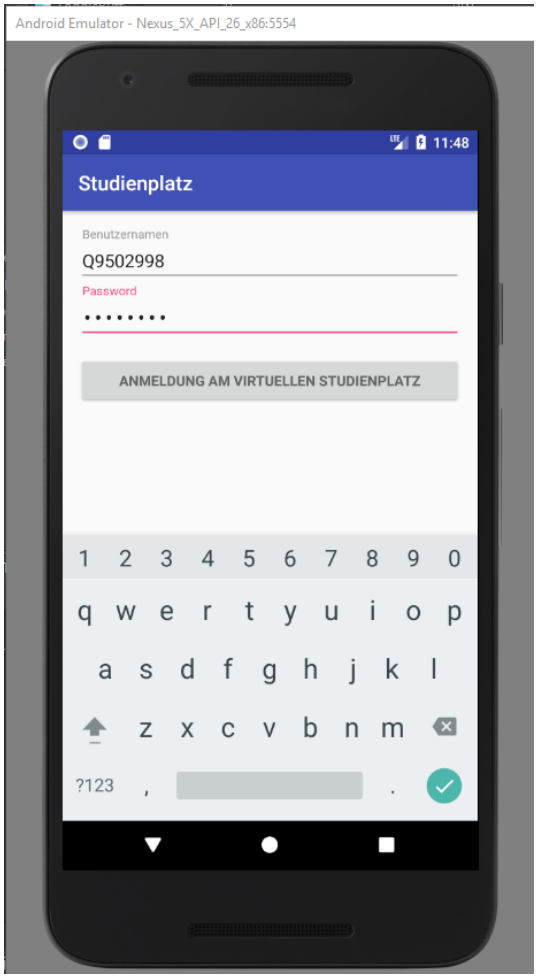


Abbildung 2.30: Login Studienplatz App

Nach Eingabe der Anmeldedaten, sowie nach Betätigung der Schaltfläche 'Anmeldung am virtuellen Studienplatz', wird der virtuelle Studienplatz innerhalb der App geöffnet. Die Anwendung zeigt nun den gewünschten Inhalt an. Wenn das Gerät gedreht wird, so wird der virtuelle Studienplatz entsprechend ausgerichtet, sodass die gewünschten Informationen komfortabel eingesehen werden können. Dies zeigt die Abbildung 2.31.


Abbildung 2.31: Virtueller Studienplatz in WebView [29]

Darüberhinaus können nun die verschiedenen Bereiche des virtuellen Studienplatz verwendet werden, um beispielsweise die aktuelle Belegung zu prüfen oder um weitere Kurse nachzubelegen. Dies zeigt die Abbildung 2.32.

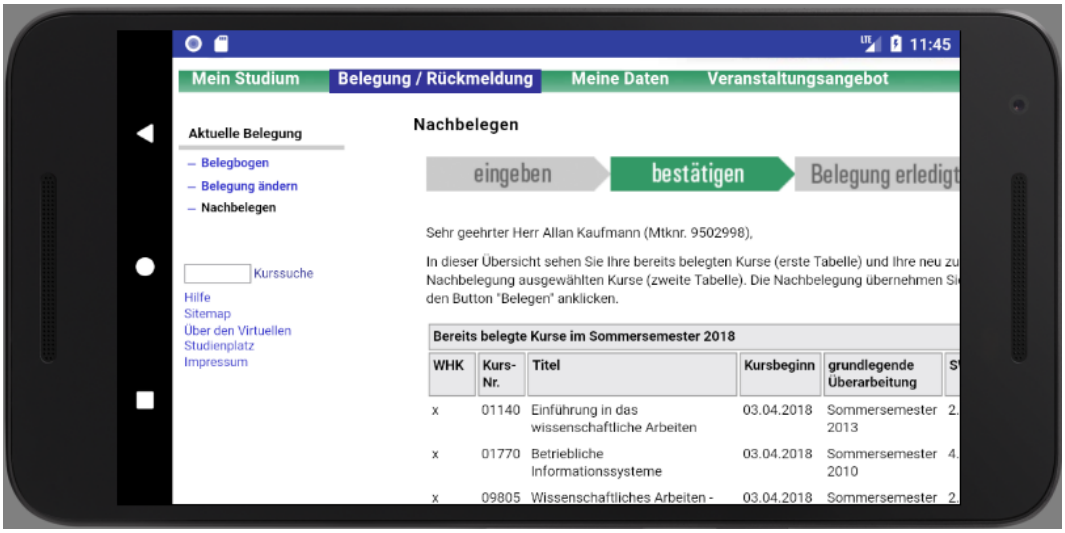


Abbildung 2.32: Kurse nachbelegen [29]

### 2.4.4 Fazit: Android Studio

Die Entwicklungsumgebung Android Studio ermöglicht einen sehr einfachen Einstieg in die Entwicklung von Android Apps, da die Entwicklungsumgebung bereits vorgefertigte Komponenten mitbringt und die Einrichtung von Projekten erleichtert. Die Entwicklungsumgebung unterstützt alle gängigen Android SDK-Versionen sowie eine Vielzahl von Geräten, sodass die zu entwickelnde App praktisch für jedes Gerät getestet werden kann. Die Umstieg der Entwicklungsumgebung von Eclipse auf IntelliJ war, aus meiner Sicht, der richtiger Schritt, da IntelliJ in den vergangenen Jahren unter Java-Entwicklern immer mehr Zuspruch erhalten hat. Dies liegt, meiner Ansicht nach, vorallem an der höheren Benutzerfreundlichkeit, sowie an der besseren Unterstützung moderner Techniken, wie beispielsweise die komfortable Integration der Buildsysteme Maven und Gradle. Aus diesen Gründen kann ich die Entwicklungsumgebung Android Studio uneingeschränkt empfehlen!

# Literaturverzeichnis


[1] ZDNet / Mobil *Android steigert Marktanteil auf fast 88 Prozent*
http://www.zdnet.de/88282259/android-steigert-marktanteil-auf-fast-88-prozent/

[2] Thomas Künnerth *Android 7: Das Praxisbuch für Entwickler Rheinwerk Verlag GmbH 4., aktualiserte Auflage 2017*

[3] Android Developers Blog *This is the droid you're looking for*
https://android-developers.googleblog.com/2007/11/posted-by-jason-chen-android-advocat

[4] Prof. Dr. Lars Mönch *Betriebliche Informationssysteme, Kurseinheit 4: Konstruktion von dienstorientierten Informationssystemen*

[5] Android Source *ART and Dalvik*
https://source.android.com/devices/tech/dalvik/

[6] Android Source *Verifying App Behavior on the Android Runtime (ART)*
https://developer.android.com/guide/practices/verifying-apps-art.html/

[7] Android API Guides *Platform Architecture*
https://developer.android.com/guide/platform/index.html

[8] Stephan Elter, Sven Haiges *Android Schnelleinstieg entwickler.press, 2014*

[9] App Manifest
https://developer.android.com/guide/topics/manifest/manifest-intro.html

[10] Resources Overview
https://developer.android.com/guide/topics/resources/overview.html

[11] App components
https://developer.android.com/guide/components/index.html

[12] *Zigurd Mednieks, Laird Dornin, G.Blake Meike & Masumi Nakamura*
O'Reilly Verlag GmbH & Co. KG, 2013

[13] Introduction to Activities *https://developer.android.com/guide/components/activities/intro-activities.html*

[14] Fragments *https://developer.android.com/guide/components/fragments.html*

[15] Services *https://developer.android.com/guide/components/services.html*

[16] Content Providers *https://developer.android.com/guide/topics/providers/content-providers.html*

[17] Content Provider Basics *https://developer.android.com/guide/topics/providers/content-provider-basics.html*

[18] Resources Overview *https://developer.android.com/guide/topics/resources/overview.html*

[19] Providing Resources *https://developer.android.com/guide/topics/resources/providing-resources.html*

[20] User Interface *https://developer.android.com/guide/topics/ui/index.html*

[21] UI overview *https://developer.android.com/guide/topics/ui/overview.html*

[22] Layouts *https://developer.android.com/guide/topics/ui/declaring-layout.html*

[23] Input controls *https://developer.android.com/guide/topics/ui/controls.html*

[24] Menus *https://developer.android.com/guide/topics/ui/menus.html*

[25] Adding the App Bar *https://developer.android.com/training/appbar/index.html*

[26] Dialogs *https://developer.android.com/guide/topics/ui/dialogs.html*

[27] Meet Android Studio *https://developer.android.com/studio/intro/index.html*

[28] Project Overview *https://developer.android.com/studio/projects/index.html*

[29] Willkommen im Virtuellen Studienplatz *https://vu.fernuni-hagen.de/lvuweb/lvu*

[30] Configure Your Build *https://developer.android.com/studio/build/index.html*