JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

MOBILE ACCESS TO WEB SERVICES

Systemtechnik Labor
5BHITT 2015/16, Gruppe A

Jakub Kopec

Version 0.2

Note:

Betreuer: Michael Borko

Begonnen am 15. April 2016

Beendet am 22. April 2016

Inhaltsverzeichnis

1 Einführung

Diese Übung gibt einen Einblick in Entwicklungen von mobilen Applikationen.

1.1 Ziele

Das Ziel dieser Übung ist eine Anbindung einer mobilen Applikation an ein Webservices.

Die Anbindung soll mit Hilfe eines RESTful Webservice (Gruppe1) umgesetzt werden.

1.2 Voraussetzungen

- Grundlagen Java und XML
- Grundlegendes Verständnis über Entwicklungs- und Simulationsumgebungen
- Verständnis von RESTful Webservices

1.3 Aufgabenstellung

Es ist eine mobile Anwendung zu implementieren, die sich an das Webservice aus der Übung DezSysLabor-09 "Web Services in Java" anbinden soll. Dabei müssen die entwickelten Schnittstellen entsprechend angesprochen werden.

Es ist freigestellt, welche mobile Implementierungsumgebung dafür gewählt wird. Empfohlen wird aber eine Implementierung auf Android.

1.4 Bewertung: 16 Punkte

- Anbindung einer mobilen Applikation an die Webservice-Schnittstelle (6 Punkte)
- Registrierung von Benutzern (3 Punkte)
- Login und Anzeige einer Willkommensnachricht (3 Punkte)
- Simulation bzw. Deployment auf mobilem Gerät (2 Punkte)
- Protokoll (2 Punkte)

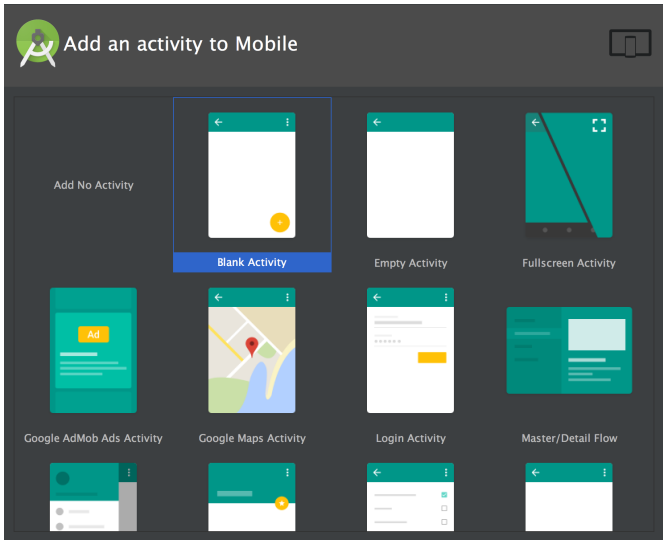
2 Ergebnisse

2.1 Entwicklungsumgebung

Als Entwicklungsumgebung wurde für diese Laborübung IntelliJ [3] verwendet. Es bestand die Möglichkeit Android Studio [4] zu verwenden, ein Tool welches extra für das Android Development entwickelt wurde. Da IntelliJ aber Android Studio implementiert und alle Android Studio-Features in IntelliJ enthalten sind, ist die Entscheidung auf IntelliJ gefallen. [5] Das Einzige, was in IntelliJ konfiguriert werden muss, ist die Einbindung einer Android SDK [6].

2.2 Einlesen in die Thematik

Da es etwas länger her ist, dass ich meine letzte Android App geschrieben habe, musste ich mich damit auseinandersetzen. Dazu habe ich mir die von IntelliJ vorgeschlagenen „default"-Projekte angesehen, welche man in der Folgenden Abbildung sieht.



Informationen die ich aus der Auseinandersetzung gewinnen konnte:

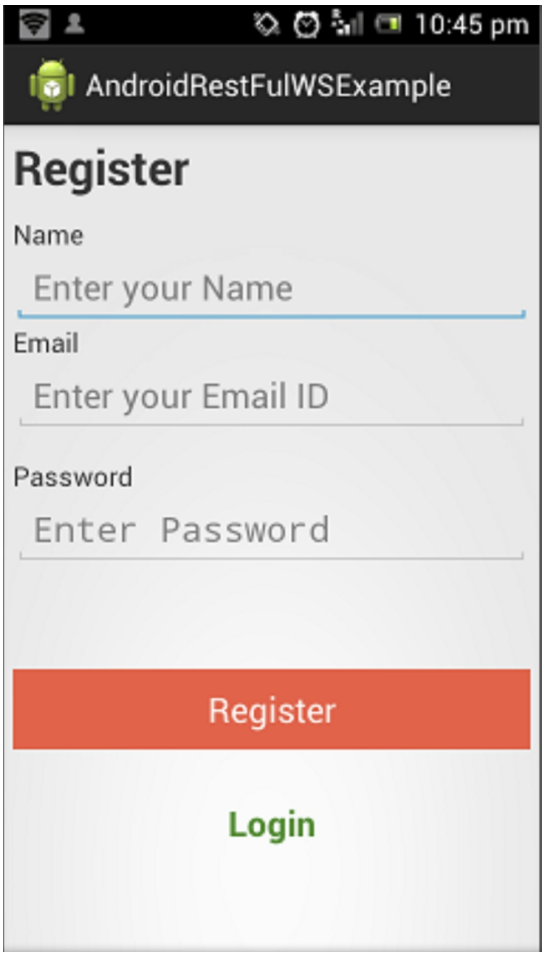
- Gradle Dependencies in Erinnerung gerufen
- Statische Strings gehören in die string.xml
- Graphische Oberflächen gehören in XML-Files unter app/res/layout/

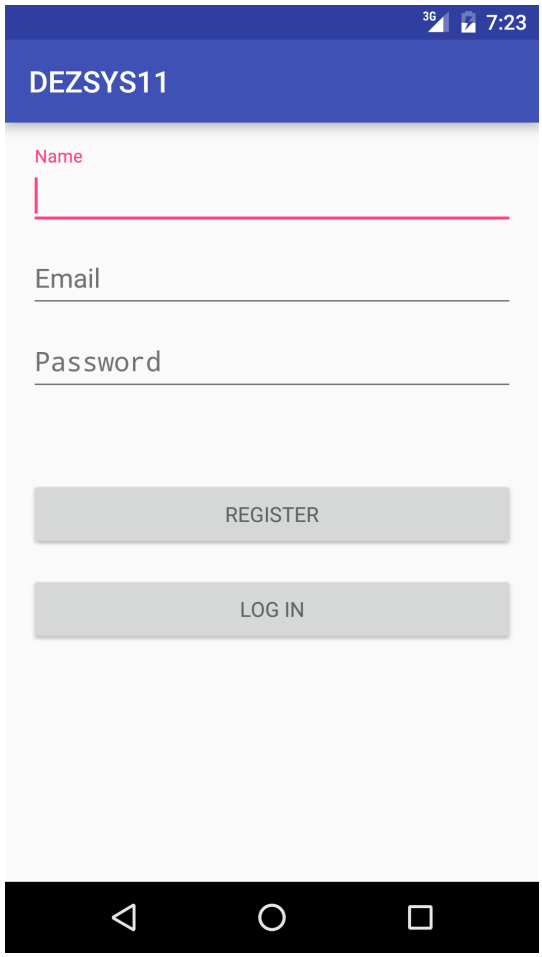
2.3 Umsetzung

Für die Umsetzung wurde im Großen und Ganzen das vom Herrn Professor Borko zur Verfügung gestellte Tutorial befolgt. [1] Respekt an dieser Stelle, das war ein sehr gut erklärtes und aufschlussreiches Tutorial welches zur Abwechslung (fast) Reibungslos funktioniert hat.

Verbesserung des GUI

Damit es nicht zu leicht ist, wurde das Layout aus dem Tutorial wesentlich verbessert und auf den heutigen Stand der Technik gebracht. Hier ein Vorher-Nachher-Vergleich:





Um dieses Layout zu erreichen, musste ziemlich viel geändert werden. Zum einen natürlich die Layout-Files selbst, zum anderen aber auch die Java-Files.

Bei den Layout-Files wurden vor allem die Textfelder und Buttons geändert. Die für die Textfelder zuständige Codesequenz:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/name" />
<!-- Name TextField -->

<EditText
    android:id="@+id/registerName"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:hint="Enter your
Name"/>
<!-- Email Label →
```

wurde durch folgende ersetzt:

```
<android.support.design.widget.TextInputLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

<EditText
        android:id="@+id/registerName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/prompt_name"
        android:imeActionId="@+id/login"
        android:imeActionLabel="@string/prompt_name"
        android:imeOptions="actionUnspecified"
        android:inputType="textPersonName"
        android:maxLines="1"
        android:singleLine="true"/>
</android.support.design.widget.TextInputLayout>
```

Ähnlich wurden auch die Buttons ersetzt.
Hier die alten Buttons:

```
<Button
    android:id="@+id/btnRegister"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dip"
    android:background="#ff6347"
    android:onClick="registerUser"
    android:text="@string/btnRegister"
    android:textColor="#fff" />
```

welche durch weit weniger störende ersetzt wurden:

```
<Button
    android:id="@+id/btnRegister"
    style="?android:textAppearanceSmall"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="@dimen/activity_vertical_margin"
    android:onClick="registerUser"
    android:text="@string/action_register"
     />
```

Bei allen Layout-Files wurde zusätzlich die `<ScrollView>` mit einem Weiteren `<LinearLayout>` umrandet. In den Layouts wurden noch weitere kleine Änderungen durchgeführt, auf welche hier aber nicht mehr eingegangen wird und welche aus dem Sourcecode zu entnehmen sind.

Auf der Seite der Java-Files wurde bei jeder Klasse anstatt von `Activity`, von `AppCompatActivity` geerbt, wodurch das GUI eine „Title-Bar" erhalten hat.

Funktionalität

Das GUI wurde verbessert. Nun galt es die Funktionalität zum laufen zu bekommen, wobei es mehrere Probleme gab.

Diese Probleme unterteilen sich in Probleme des Tutorials an sich und Kompatibilitätsproblemen mit der letzten Aufgabe.

Zuerst die Probleme des Tutorials an sich. Das Tutorial verwendet einen `AsyncHttpClient` zur Kommunikation mit der REST-Schnittstelle. Folgender ist im `com.loopj.android.http` Package zu finden, was zur Folge hat, dass ein Gradle-Dependency-Eintrag notwendig ist. Hier muss man jedoch aufpassen. Das Tutorial ist doch schon etwas älter und funktioniert mit einer älteren version des `com.loopj.android.http` Packages. Soll jedoch die neueste Version verwendet werden, ändern sich die Parameter der `.get()` oder `.post()` Methode. Im Falle dieser Aufgabe wurde entschieden die neueste version zu verwenden, womit folgender Gradle-Dependency-Eintrag hinzugefügt werden muss:

```
dependencies {
    compile 'com.loopj.android:android-async-http:1.4.9'
}
```

Durch die Verwendung der neuesten Version, müssen wie bereits erwähnt, die Parameter angepasst werden. Sowohl Anzahl als auch Reihenfolge sind neu. Diese Änderungen sind aus dem Code zu entnehmen.

Nun zu den Kompatibilitätsproblemen. Die REST-Schnittstelle aus der Aufgabe DEZSYS09 empfängt und verarbeitet JSON-Objekte. In diesem Tutorial jedoch, werden an die REST-Schnittstelle, Parameter in Form von `RequestParams` verschickt. Damit die Kommunikation funktioniert musste das geändert werden. Folgende Schritte wurden dazu benötigt. Als erstes wurde für die Parameter an Stelle des `RequestParams` ein `JSONObject` erstellt.

```
JSONObject params = new JSONObject();
```

Diesem Objekt wurden alle benötigten Daten hinzugefügt.

```
params.put("name", name);
params.put("email", email);
params.put("password", password);
```

Daraufhin wird eine Methode zur Kommunikation mit dem Server aufgerufen.

```
invokeWS(params);
```

In dieser Methode muss eine StringEntity daraus gebildet werden um die Daten versenden zu können.

```
StringEntity request = null;
try {
   request = new StringEntity(params.toString());
   request.setContentType(new BasicHeader(HTTP.CONTENT_TYPE,
"application/json"));
} catch (UnsupportedEncodingException e) {
   e.printStackTrace();
}
```

Nun wird ein `AsyncHttpClient` erstellt.

```
AsyncHttpClient client = new AsyncHttpClient();
```

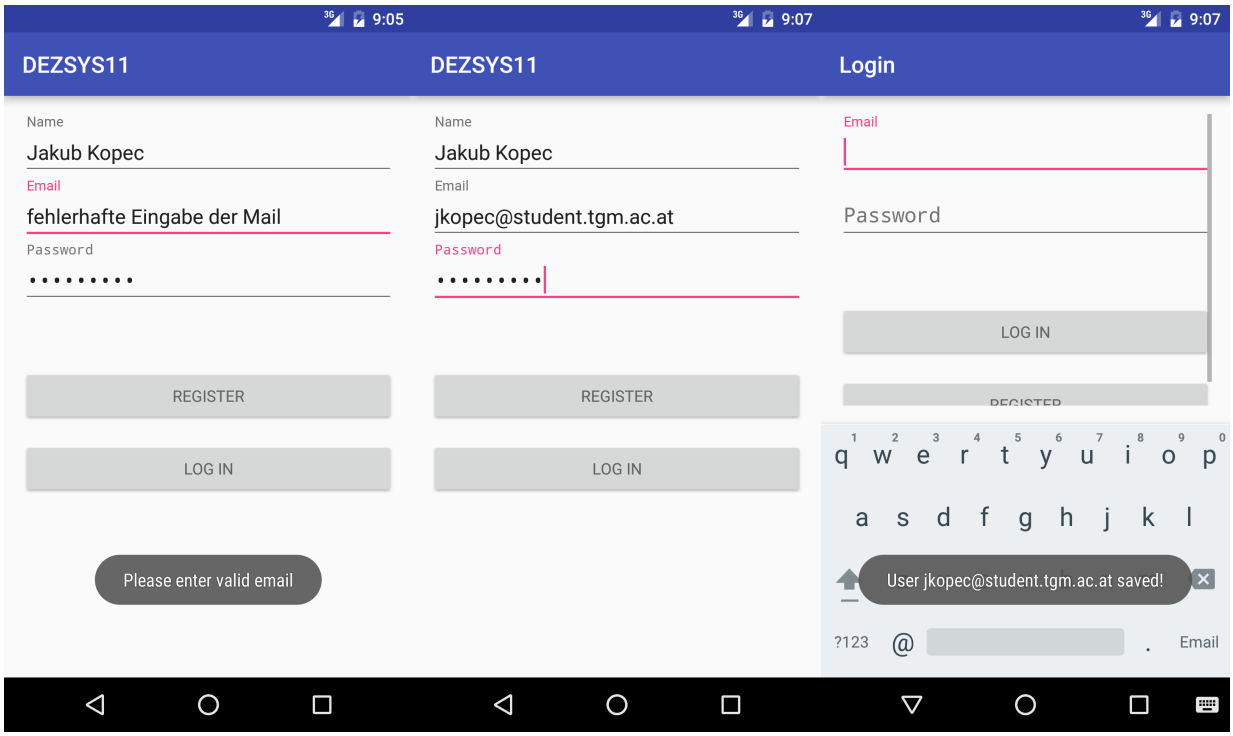
Mit Hilfe dessen können die Daten nun versendet werden. Hierbei ist zu beachten, dass die IP-Adresse des Servers nicht `127.0.0.1` ist, obwohl der Server lokal läuft.

```
client.post(this.getApplicationContext(),
"http://10.0.2.2:8080/register", request, "application/json", new
TextHttpResponseHandler() {
     //onSuccess & onFailure Methoden
}
```

2.4 Programmablauf

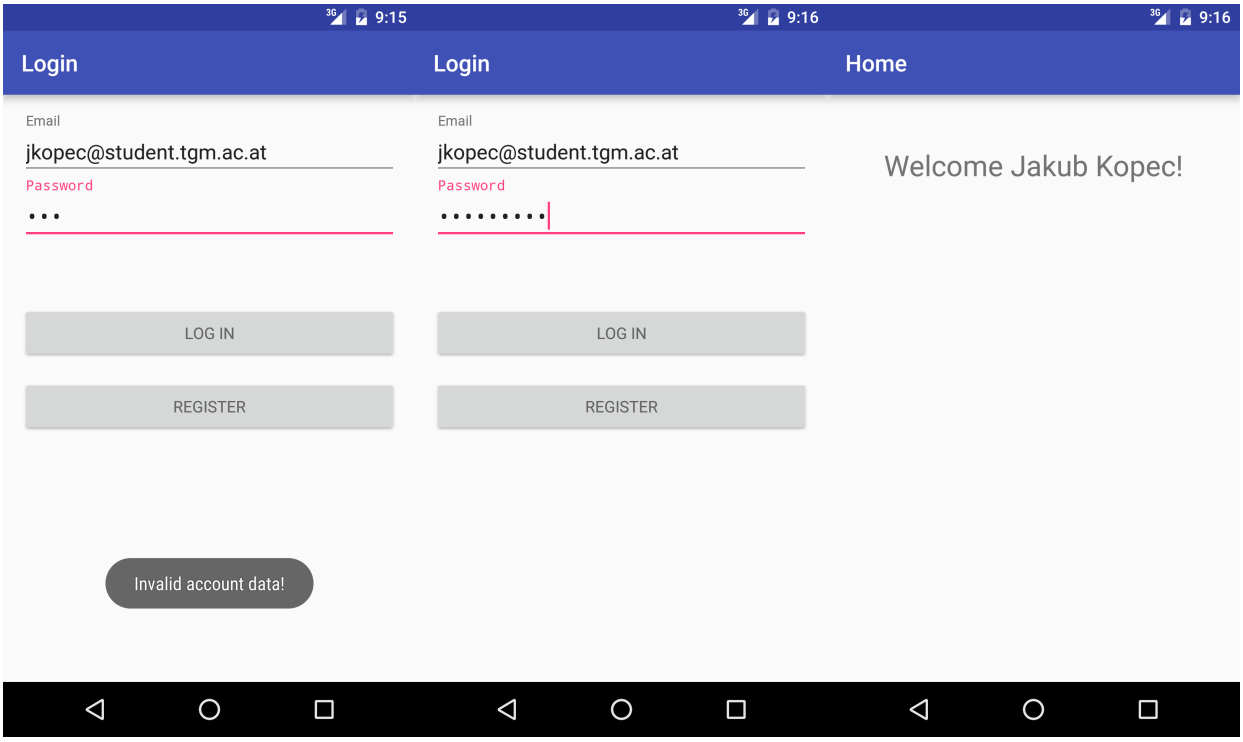
Als erstes muss die REST-Schnittstelle gestartet werden. Nun kann die App gestartet werden.

Sobald die App gestartet wird, kommt man zum Registrierungsformular.



So könnte beispielsweise die Registrierung eines neuen Users aussehen. In der ersten Abbildung ist zu sehen, dass eine fehlerhafte Email angegeben wurde. Auf diesen Fehler wird man aufmerksam gemacht. Daraufhin wird in Abbildung 2 eine valide Email eingegeben und abermals auf den „REGISTER"-Button geklickt. Sobald die Registrierung erfolgreich verlaufen ist, wird man zum Login-Frame weitergeleitet und erhält als „Toast" die Antwort vom Server.

Im Falle des Logins funktioniert das ganz ähnlich.



Gibt man falsche Accountdaten an, wie in der ersten Abbildung zu sehen,wird man darauf hingewiesen. Gibt man in weiterer Folge die richtigen Accountdaten an, wird man zum Willkommens-Layout weitergeleitet.

3 Zeitaufwand

Task	Zeit
Einlesen in die Thematik	2h
Verbesserung des GUI	1h
Funktionalität Implementieren	2h
Debugging	1h
Summe	**6h**

4 Quellen

[1] "Android Restful Webservice Tutorial – How to call RESTful webservice in Android – Part 3"; Posted By Android Guru on May 27, 2014; online: http://programmerguru.com/android-tutorial/android-restful-webservice-tutorial-how-to-call-restful-webservice-in-android-part-3/

[2] "Referenzimplementierung von DezSys09"; Paul Kalauner; online: https://github.com/pkalauner-tgm/dezsys09-java-webservices

[3] „IntelliJ"; online: https://www.jetbrains.com/idea/

[4] „Android Studio"; online: http://developer.android.com/sdk/index.html

[5] „IntelliJ Idea and Android Studio FAQ"; online: http://blog.jetbrains.com/idea/2013/05/intellij-idea-and-android-studio-faq/

[6] „Download Android SDK for Mac OS X"; online: http://dl.google.com/android/android-sdk_r24.4.1-macosx.zip