

**Laborprotokoll**

**Mobile access to web services**

**Systemtechnik Labor**

**5BHITT 2015/16, Gruppe Z**

**Manuel Reiländer**

|  |  |
| --- | --- |
|  | **Version 1.0** |
| **Note:** | **Begonnen am 20. April 2016** |
| **Betreuer: Michael Borko** | **Beendet am 22. April 2016** |

Inhaltsverzeichnis

[1Einführung 3](#__RefHeading___Toc105_1963609346)

[1.1Ziele 3](#__RefHeading___Toc107_1963609346)

[1.2Voraussetzungen 3](#__RefHeading___Toc109_1963609346)

[1.3Aufgabenstellung 3](#__RefHeading___Toc111_1963609346)

[2Ergebnisse 4](#__RefHeading___Toc113_1963609346)

# Einführung

Diese Übung gibt einen Einblick in Entwicklungen von mobilen Applikationen.

## Ziele

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

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

## Voraussetzungen

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

## 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

# Ergebnisse

## Installation der Android SDK und System Images

### Android SDK

Herunterladen der Android SDK von der offiziellen Seite ohne Android Studio (command line tools only)

* <http://developer.android.com/sdk/index.html>

Dies installiert einen AVD Manager (Android Virtual Device Manager) und einen SDK Manger. Mit dem SDK Manager können APIs und System Images heruntergeladen werden. APIs werden benötigt, um Zugriff auf Features einer Android Version zu erhalten. Die System Images werden benötigt um AVDs zu erstellen und zu starten.

### APIs

Je nachdem auf welcher Android Version man die zu programmierende Applikation installieren will, muss man natürlich die entsprechende Android API auswählen.

### System Images

Die System Images dienen dazu einen Emulator zu starten, der ein Gerät simuliert auf dem Android läuft, um die Applikation testen zu können. Hier sollten Images der zuvor gewählten API Version installiert werden. D.h. wählt man die API Version 23, sollten auch Images dieser API installiert und für den Test der Applikation verwendet werden (siehe Abbildung 1 Android SDK Manager).

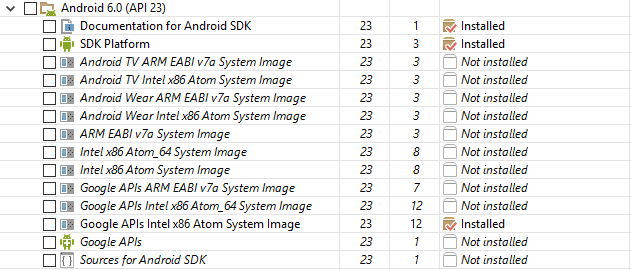


Abbildung 1 Android SDK Manager

### AVD Manager (Android Virtual Device Manager)

Um ein virtuelles Android Gerät zu erstellen, öffnet man den AVD, erstellt ein neues AVD mit dem Button „Create“ (siehe Abbildung 2 AVD) und trägt die benötigten Parameter ein.

AVD Name … Name des Gerätes (beliebig)

Device … Version des Gerätes (z.B. Nexus 6)

Target … Android Version, die auf dem Gerät läuft

CPU … Prozessor des AVD

Skin … k.A.

Alle anderen Parameter sind optional, jedoch sollte der Hacken, bei „Use Host GPU“ angekreuzt werden, ansonsten benutzt der Emulator den integrierten Grafikchip des Hostsystems.

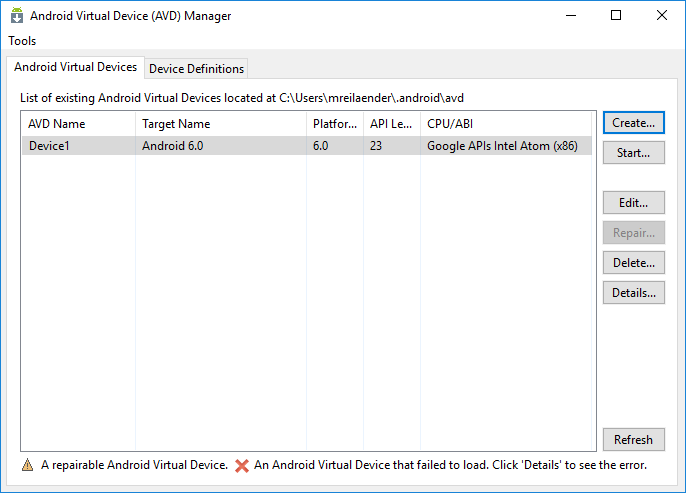


Abbildung 2 AVD

## Kofiguration von IntelliJ

Es ist nicht viel notwendig, um Intellij so zu konfigurieren, dass die Applikation auf Knopfdruck auf das AVD deployed wird. Zuerst erstellt man ein neues Projekt von einem Android Template (Gradle: Android Module), das von IntelliJ angeboten wird (siehe Abbildung 3 IntelliJ Android Template).

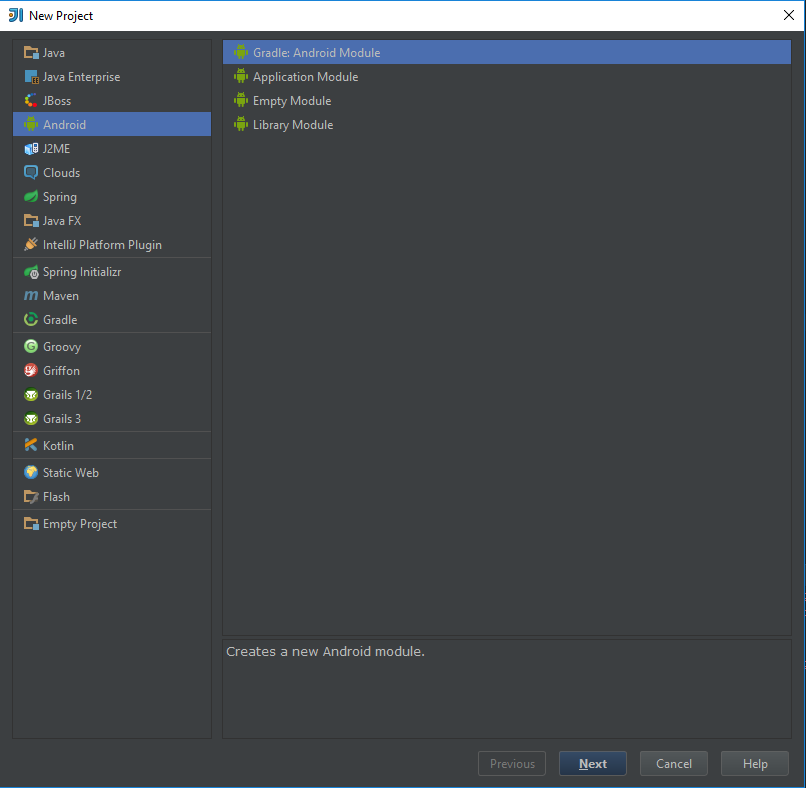


Abbildung 3 IntelliJ Android Template

Hier ist nur zu beachten, beim Parameter „Target SDK“ dieselbe Android Version anzugeben, wie beim zuvor Erstellten AVD. Die restlichen Parameter können beliebig gewählt werden.

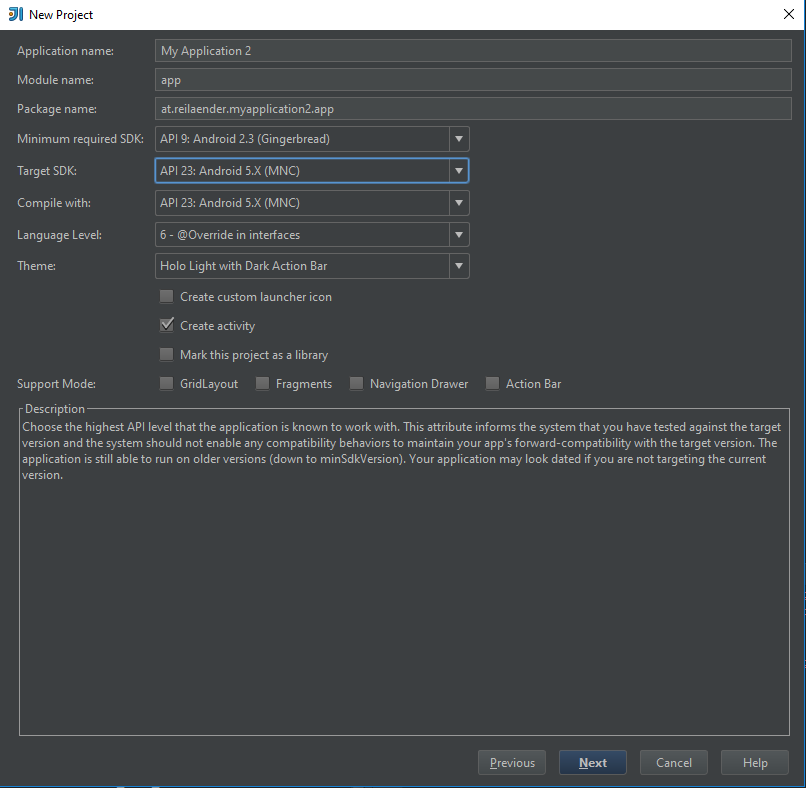


Abbildung 4 Android Project settings

# Troubleshooting

## failed to find target xy



Hier muss man zuerst herausfinden, welchen Namen die installierten „*system images*“ haben. Hierzu geht man in den Installationsordner der Android sdk (unter windows *C:\Users\userxy\AppData\Local\Android\sdk*) und im Verzeichnis *system-images* findet man nun die Namen der installierten system images. Nun öffnet man im IntelliJ Projekt die *build.gradle* (die build.gradle der Applikation, nicht die die im Root-Verzeichnis des Projektes liegt!) und trägt nun unter dem Parameter compileSdkVersion den Namen des Ordners unter *system-images* ein. Je nachdem welche Android Version man mit dem SDK Manager installiert hat (siehe Punkt APIs), hat dieser Ordner einen anderen Namen. Es können auch mehrere system-images installiert sein. Verwendet man z.B. Version 23 der API so ist der Name des Ordners entsprechen *android-23*.

## Android.os.NetworkOnMainThreadException

Wie der Name schon sagt, erlaubt Android standardmäßig keine Netzwerk Operationen im Main Thread. Nun gibt es 2 Varianten dies zu umgehen. Einerseits kann man dies in der *AndroidManifest.xml* mit einem entsprechenden Eintrag erlauben, andererseits, was die wesentlich schönere Methode ist, kann man die Operationen in einen *AsyncTask* auslagern. Dies sieht dann wie folgt aus.

public class RegisterButtonHandler extends AsyncTask<HttpEntity, Void, ResponseEntity> {  
 private RestTemplate restTemplate;  
 private HttpEntity httpEntity;  
 private String url;  
  
 public RegisterButtonHandler(HttpEntity httpEntity, String url) {  
 this.httpEntity = httpEntity;  
 this.url = url;  
 this.restTemplate = new RestTemplate();  
 }  
  
 @Override  
 protected ResponseEntity doInBackground(HttpEntity... httpEntities) {  
 try {  
 return this.restTemplate.postForEntity(this.url + "/register", this.httpEntity, String.class);  
 } catch (HttpClientErrorException e) {  
 return null;  
 }  
 }  
}

AsyncTask ist eine generische Klasse mit 3 Parametern, *AsyncTask<Params, Progress, Result>*, die es erlaubt ein beliebiges Objekt als *Params* zu übergeben. Progress wird nicht benötigt und kann ignoriert werden und ResponseEntity ist das, was die Methode doInBackground zurückgibt. Dies muss so gestaltet sein, da der AsyncTask folgendermaßen aufgerufen wird:

new RegisterButtonHandler(httpEntity, ServerConfig.*url*).execute()

Nun wird die Methode doInBackground aufgerufen, die ihre Arbeit erledigen kann. Will man nun das Ergebnis, also in diesem Fall die ResponseEntity, führt man *get()* aus und bekommt diese zurück.

ResponseEntity responseEntity =

new RegisterButtonHandler(httpEntity, ServerConfig.*url*).execute().get();