Laborprotokoll

DEZSYS-11: Mobile Access to Web Services

Systemtechnik
5BHITT 2015/16

Rene Hollander

Version 1.0

Note:

Betreuer: Borko

Begonnen am 19. Februar 2016

Beendet am 10. März 2016

Inhaltsverzeichnis

Einführung

Diese Übung gibt einen Einblick in Entwicklungen von mobilen Applikationen.

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.

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

Quellen

"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/

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

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)

Zeitschätzung

3 Stunden App implementieren
1 Stunde Heroku zum laufen bringen
1 Stunde Protokoll
→ 5 Stunden Zeitaufwand

Tatsächlich:
2.5 Stunden App implementieren
1 Stunde Heroku zum laufen bringen
45 Minuten Protokoll
→ 4.5 Stunden

Ergebnisse

Aufbauend auf der Übung DezSys09 Web Services in Java wurde eine Mobile Applikation geschaffen, mit der man sich auf dem Web Service registrieren und anmelden kann.

Java 8 Lambda Support

Standardmäßig unterstützt Android keine Lambda Funktionen. Da ich diese aber sehr praktische finde wurde auf das retrolambda Projekt zurückgegriffen.

Die Nutzung ist einfach und unkompliziert. Nur das build.gradle File muss angepasst werden. Folgende Zeilen werden hinzugefügt:

```
buildscript {
    repositories {
        mavenCentral()
    }
    dependencies {
        classpath 'me.tatarka:gradle-retrolambda:3.2.5'
    }
}
apply plugin: 'me.tatarka.retrolambda'
```

Http Client

Um eine Verbindung zum Webservice aufbauen zu können, wird der AsyncHttpClient [2] genutzt um HTTP Requests auszuführen.

Beispiel anhand der Registrierung:

Zuerst muss das Request JSON generiert werden:

```
JSONObject body = new JSONObject();
body.put("email", email);
body.put("password", password);
```

Danach kann mit einem Instanzierten HTTP Client ein POST Request ausgeführt werden:

```
client.post(this, API_URL + "/register", new StringEntity(body.toString()),
ContentType.APPLICATION_JSON.getMimeType(), new JsonHttpResponseHandler() {
    ...
});
```

Im Response Handler werden die folgenden Methoden überschrieben:

onSuccess:

Wenn der Request funktioniert hat wird ein Status Code im 200 Bereich zurückgegeben. Dieser wird zusätzlich ausgewertet dann der Callback ausgeführt.

```
public void onSuccess(int statusCode, Header[] headers, JSONObject response) {
     if (statusCode == 201) {
          cb.call(null, true);
     } else {
          cb.call(new Exception("invalid status code for success: " + statusCode), null);
     }
}
```

onFailure:

Um eventuelle Fehler des AsyncHttpClients, sowie Fehler des Requests abzufangen wird der onFailure Handler überschrieben:

```
public void onFailure(int statusCode, Header[] headers, String string, Throwable
throwable) {
     cb.call(throwable, null);
}
public void onFailure(int statusCode, Header[] headers, Throwable throwable, JSONObject
object) {
     if (statusCode == 400) {
          cb.call(null, false);
     } else {
          cb.call(throwable, null);
     }
}
```

Der Callback wird mit dem Resultierendem Fehler bzw Fehlercode versorgt.

Der Login wurde wie die Register Methode implementiert, der URL Endpoint lautet logischerweise /login.

GUI

Die Login GUI wurde mit dem Template, dass bei der Erstellung des Projektes ausgewählt werden kann realisiert. Einzig und alleine die attemptLogin() Methode wurde angepasst, um den zuvor erstellen HTTP Client zu nutzen:

```
((Application) getApplication()).login(email, password, (err, res) -> {
    showProgress(false);
    if (err == null) {
        if (res) {
            Util.messageDialogRunLater(this, "Success", "You successfully logged in!");
            Log.d("login", "Successfull login");
        } else {
            Util.messageDialogRunLater(this, "Error", "Wrong email and or password!");
            Log.e("login", "Error logging in");
        }
    } else {
        Util.messageDialogRunLater(this, "Error", "An error occured while logging in: "
+ err.getMessage());
        Log.e("login", "Error logging in", err);
    }
});
```

Neben eines Message Dialog wird auch der vollständige Fehler mitgeloggt, um die Entwicklung und Fehlersuche zu vereinfachen.

Für den Register Dialog wurde das Layout XML File sowie der Code für die Activity dupliziert und der Text angepasst.

Wenn die Registrierung erfolgreich war, wird ein neues Intent gestartet um auf die Login Activity zu wechseln:

```
Util.messageDialogRunLater(this, "Success", "You successfully registered!");
Log.d("register", "Successfull register");
startActivity(new Intent(this, LoginActivity.class));
```

Auch wurden Buttons hinzugefügt um zwischen den Activities zu wechseln:

```
<Button
    android:id="@+id/go_to_register_button"
    style="?android:textAppearanceSmall"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:text="@string/action_go_to_register"
    android:textStyle="bold" />
```

Damit der Button eine Funktion hat wird in der onCreate ein Listener erstellt. Auf Knopfdruck wird ein neues Intent gestartet und auf die RegisterActivity weiter geleitet:

```
Button mGoToRegisterButton = (Button) findViewById(R.id.go_to_register_button);
mGoToRegisterButton.setOnClickListener(view -> startActivity(new Intent(this,
RegisterActivity.class)));
```

Heroku

Heroku wird genutzt um die Webapplikation auf einem Server auszuführen damit die Webapp nicht lokal gestartet werden muss um die App zu verwenden.

Dazu wurde zuerst ein Accout bei Heroku angelegt, dann eine neue Applikation mit dem Namen dezsys11-mobileapp. Um Heroku nutzen zu können musste noch der Heroku Toolbelt installiert werden.

Im Anschluss wurde ein Procfile und ein system.properties File erstellt:

Procfile:

```
web: java -Dserver.port=$PORT -jar target/mobileapp-0.0.1.jar
```

system.properties:

```
java.runtime.version=1.8
```

Dann konnte das Heroku Repository zu Git hinzugefügt werden: Befehl:

```
heroku git:remote -a dezsys11-mobileapp
```

Mit dem Befehl git push heroku master wurde das Repository auf Heroku gepusht. Nun läuft die Webapplikation und ist unter der URL https://dezsys11-mobileapp.herokuapp.com/ verfügbar.

Source Code

Der Code ist Verfügbar unter: https://github.com/ReneHollander/dezsys11-mobileapp

Kompilieren und Ausführen

Die Webapplication kann wie beschrieben in Dezsys09 [1] lokal gestartet werden. Um die Lokale Installation der Webapplication zu nutzen, muss die IP in der Application Class der App angepasst werden:

```
private static final String API_URL = "http://10.0.0.51:8080";
```

Durch Verwendung von Heroku wird aber die Webapplikation auf einem Server ausgeführt um darauf zugreifen zu können. Die dazu benötigte Adresse ist Standardmäßig eingetragen und kann jederzeit verwendet werden. Also muss das Projekt nur kompiliert und auf ein Gerät übertragen werden, beziehungsweise im Emulator ausgeführt werden.

Quellen

[1] DEZSYS09-Webservices, Rene Hollander
Abrufbar unter: https://github.com/ReneHollander/dezsys09-webservices
zuletzt abgerufen: 19.02.2016

[2] Android Asynchronous Http Client
Abrufbar unter: http://loopj.com/android-async-http/
zuletzt abgerufen: 10.03.2016

[3] Retrolambda Gradle Github
Abrufbar unter: https://github.com/evant/gradle-retrolambda
zuletzt abgerufen: 10.03.2016