JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Web Services in Java

Systemtechnik Labor
5BHITT 2015/16, GruppeA

Klaus Ableitinger

Version 0.2
Begonnen am 18. Februar 2016
Beendet am 18. Februar 2016

Note:
Betreuer: Mi. Borko

Inhaltsverzeichnis

1 Einführung

Diese Übung zeigt die Anwendung von mobilen Diensten in Java.

1.1 Ziele

Das Ziel dieser Übung ist eine Webanbindung zur Benutzeranmeldung in Java umzusetzen. Dabei soll sich ein Benutzer registrieren und am System anmelden können.

Die Kommunikation zwischen Client und Service soll mit Hilfe von JAX-RS (Gruppe1) umgesetzt werden.

1.2 Voraussetzungen

- Grundlagen Java und Java EE
- Verständnis über relationale Datenbanken und dessen Anbindung mittels JDBC oder ORM-Frameworks
- Verständnis von Restful Webservices

1.3 Aufgabenstellung

Es ist ein Webservice mit Java zu implementieren, welches eine einfache Benutzerverwaltung implementiert. Dabei soll die Webapplikation mit den Endpunkten /register und /login erreichbar sein.

Registrierung Diese soll mit einem Namen, einer eMail-Adresse als BenutzerID und einem Passwort erfolgen. Dabei soll noch auf keine besonderen Sicherheitsmerkmale Wert gelegt werden. Bei einer erfolgreichen Registrierung (alle Elemente entsprechend eingegeben) wird der Benutzer in eine Datebanktabelle abgelegt.

Login Der Benutzer soll sich mit seiner ID und seinem Passwort entsprechend authentifizieren können. Bei einem erfolgreichen Login soll eine einfache Willkommensnachricht angezeigt werden.

Die erfolgreiche Implementierung soll mit entsprechenden Testfällen dokumentiert werden. Es muss noch keine grafische Oberfläche implementiert werden! Verwenden Sie auf jeden Fall ein gängiges Build-Management-Tool..

1.4 Quellen

Android Restful Webservice Tutorial – Introduction to RESTful webservice – Part 1; Posted By Android Guru on May 1, 2014; online: http://programmerguru.com/android-tutorial/android-restful-webservice-tutorial-part-1/
REST with Java (JAX-RS) using Jersey - Tutorial; Lars Vogel; Version 2.5; 15.12.2015; online: http://www.vogella.com/tutorials/REST/article.html
O Java EE 7 Application Servers, Where Art Thou? Learn all about the state of Java EE app servers, a rundown of various Java EE servers, and benchmarking.; by Antonio Goncalves; Java Zone; Feb. 10, 2016; online: https://dzone.com/articles/o-java-ee-7-application-servers-where-art-thou
Heroku makes it easy to deploy and scale Java apps in the cloud; online: https://www.heroku.com/

Bewertung: 16 Punkte

- Aufsetzen einer Webservice-Schnittstelle (4 Punkte)
- Registrierung von Benutzern mit entsprechender Persistierung (4 Punkte)
- Login und Rückgabe einer Willkommensnachricht (3 Punkte)
- AcceptanceTests (3 Punkte)
- Protokoll (2 Punkte)

2 Ergebnisse

GitHub Repository Link: https://github.com/Pwngu/Java-WebApp

2.1 Projekt Setup & Deployment

Da ich das Projekt mit Gradle umsetzen wollte, habe ich ein Github Repository mit Tutorial gefunden, welches das aufsetzen eines einfaches Projektes gut erklärt. [1].
Für das Tutorial wurde der Tomcat Application Server [2] verwendet.

2.1.1 Probleme beim Deployment

Beim deployment auf den Tomcat Server ist ein ziemliches Problem aufgetreten; der Tomcat Server scheint den konfigurierten Pfad nicht zu registrieren. Bei folgender Konfiguration (mit Projektnamen `java-web`):

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.
             sun.com/xml/ns/javaee/web-app_2_5.xsd"
         id="WebApp_ID" version="2.5">
    <display-name>Java Web</display-name>
    <servlet>
        <servlet-name>jersey</servlet-name>
        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer<
            /servlet-class>
        <init-param>
            <param-name>com.sun.jersey.config.property.packages</param-name>
            <param-value>at.tgm.ablk</param-value>
        </init-param>

        <init-param>
            <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-
                name>
            <param-value>true</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>jersey</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Listing 1: web.xml


```java
package at.tgm.ablk;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("login")
public class LoginResource {

    @GET
    @Path("/do")
    @Produces(MediaType.TEXT_PLAIN)
    public String getLogin() {
        return "hello";
    }
}
```

Listing 2: Java Testfile

sollte das Interface unter `localhost:8080/java-web/rest/login/do` erreichbar sein, jedoch bekomme ich immer, auch bei mehreren Variationen des Pfades eine 404 Response.
Auch in den Tomcat log files ließ sich nichts auffälliges finden.
Aufgrund dieses Fehlers konnte ich das Programm nicht wirklich testen.

2.2 Datenbank

Für die Aufgabe habe ich mich für eine *SQLite* Datenbank entschieden und verwende einen dazugehörigen JDBC Treiber [3]. Dieser erstellt automatisch ein Datenbankfile an der gegebenen Stelle.

```java
try {

    Class.forName("org.sqlite.JDBC");
    connection = DriverManager.getConnection("jdbc:sqlite:user.db");
    connection.setAutoCommit(false);
} catch(Exception ex) {

    LOG.fatal("Error while connecting to Database", ex);
    System.exit(
}

LOG.info("Opened database connection successfully");
```

Listing 3: Aufbau der SQLite Datenbenkverbindung

2.3 REST Interface

Sämtliche Kommunikation über das REST-Interface verläuft über JSON. Um JSON Requests akzeptieren zu können, ist es notwendig zuerst eine Klasse, welches das zu empfangende JSON repräsentiert, erstellt werden (Hier `at.tgm.ablk.rest.User`).

2.3.1 Registrierung

Bei der Registrierung muss eine JSON - POST request an den Server geschickt werden, jeweils mit der E-Mail und dem Passwort. Wenn die Registrierung erfolgreich war, wird mit einer 201 response

geantwortet, ansonsten mit 401.
Derzeit wird noch nicht überprüft, ob die übergebene E-Mail bereits registriert ist.

```
@POST
@Consumes({"application/json"})
@Path("/register")
public Response register(final User user) {

    if(user.email == null || user.password == null)
        return Response.status(401).header("message", "Invalid registration data").build();

    try(Statement stmt = database.getConnection().createStatement();
        ResultSet res = stmt.executeQuery(String.format("INSERT INTO web_user (email, password)" +
                                                        "VALUES (%s, %s)", user.email, user.password)))
                                                            {

        database.getConnection().commit();
        return Response.status(201).header("message", "Successfully registered user").build();

    } catch(SQLException ex) {
        LOG.error("Database error while registering", ex);
        return Response.status(500).header("message", "SQL Error, check server log").build();

    } catch(Exception ex) {
        LOG.error("Error while registering", ex);
        return Response.status(500).header("message", "Exception, check server log").build();
    }
}
```

Listing 4: REST User Registration

2.3.2 Login

Beim Login wird, genauso wie bei der Registrierung, eine JSON - POST request erwartet. Hier muss das JSON file auch eine E-Mail Adresse und ein Passwort enthalten, danach wird überprüft, ob der angegebene Nutzer existiert, wenn ja, wird das Passwort überprüft und bei Erfolg ein 200 response mit Willkommensnachricht verschickt.

```
@POST
@Consumes({"application/json"})
@Path("/login")
public Response login(final User user) {

    if(user.email == null || user.password == null)
        return Response.status(403).header("message", "Invalid credentials").build();

    try(Statement stmt = database.getConnection().createStatement();
        ResultSet res = stmt.executeQuery(String.format("SELECT email, password FROM web_user " +
                                                        "WHERE email = %s", user.email))) {

        while(res.next()) {

            if(res.getString("password").equals(user.password))
                return Response.status(200).header("message", "Successfully logged in").build();
        }

        // not registered
        return Response.status(403).header("message", "Invalid credentials").build();

    } catch(SQLException ex) {
        LOG.error("Database error while logging in", ex);
        return Response.status(500).header("message", "SQL Error, check server log").build();

    } catch(Exception ex) {
        LOG.error("Error while logging in", ex);
        return Response.status(500).header("message", "Exception, check server log").build();
```

```
    }
}
```

Listing 5: REST User Login

Literatur

[1] Jason Ray, 2016.

[2] 1999-2016.

[3] Oct 2015.

Tabellenverzeichnis

Listings

Abbildungsverzeichnis