DEZSYS09

Web Services in Java

Systemtechnik Labor
5BHITT 2015/16, Gruppe X

Andreas Ernhofer

Version 1.0

Note:

Betreuer: M. Borko

Begonnen am 12. Februar 2016

Beendet am 18. Februar 2016

Inhaltsverzeichnis

Abbildungsverzeichnis

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 Datenbanktabelle 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 Bewertung

- 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

2.1 Info

Dieses Projekt ist auf Github verfügbar. Dafür muss der angegebene Link geklont werden [REPD]. Der Branch *next-try-maven* ist dabei der aktuelle Branch.

Um das Programm starten zu können muss zuerst in dem Ordner *vagrant*

```
vagrant up
```

ausgeführt werden. Wurde dadurch eine VM erstellt, so kann man durch das ausführen des folgenden Befehls, im Hauptverzeichnis, das Programm starten.

```
mvn clean install tomcat7:run
```

Das Programm ist dann unter ip:port/WebServicesInJava erreichbar. Standardmäßig ist das unter localhost:19090/WebServicesInJava/, muss jedoch an die eigenen Einstellungen angepasst werden.

2.2 Datenbank

Für mein Webservice benötige ich eine Datenbank, um meine Userdaten persistent zu speichern. Dabei habe ich mich für eine MySQL-Datenbank entschieden. Zur Verfügung stelle ich diese Datenbank mittels vagrant. Dazu habe ich in meinem Verzeichnis einen vagrant Ordner angelegt. In diesem Ordner habe ich mittels *vagrant init* ein Vagrantfile erzeugt. Dieses habe ich dann, ähnlich einem zur Verfügung Gestelltem [VAGM], angepasst:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box =  "hashicorp/precise64"
  config.vm.provider "virtualbox" do |v|
    v.name = "dezsys09_mysql"
  end
  config.vm.network :forwarded_port, guest: 3306, host: 12345
  config.vm.provision :shell, :path => "install.sh"
  config.vm.synced_folder ".", "/vagrant", :mount_options =>
["dmode=777", "fmode=666"]
  config.vm.network "private_network", ip: "192.168.56.56"
end
```

Beim Starten wird ein File namens *install.sh* ausgeführt um ein Update auszuführen, sowie mysql zu installieren und zu konfigurieren. Dieses beinhaltet folgenden Inhalt:

```
#!/usr/bin/env bash

sudo apt-get update
sudo debconf-set-selections <<< 'mysql-server mysql-server/root_password
password root'
sudo debconf-set-selections <<< 'mysql-server mysql-
server/root_password_again password root'
sudo apt-get install -y vim curl python-software-properties
sudo apt-get update
sudo apt-get -y install mysql-server
sed -i "s/^bind-address/#bind-address/" /etc/mysql/my.cnf
mysql -u root -proot < /vagrant/create.sql
mysql -u root -proot -e "GRANT ALL PRIVILEGES ON users.* TO 'user'@'%'
IDENTIFIED BY 'user' WITH GRANT OPTION; FLUSH PRIVILEGES;"
sudo /etc/init.d/mysql restart
```

In die Datenbank wird eine Datei namens *create.sql* geladen. Ihr Inhalt sieht folgendermaßen aus:

```
USE mysql;
CREATE USER user IDENTIFIED BY 'user';
CREATE DATABASE IF NOT EXISTS users;
USE users;
CREATE TABLE IF NOT EXISTS `user` (
  `name` varchar(50) NOT NULL,
  `username` varchar(50) NOT NULL,
  `password` varchar(50) NOT NULL,
  `register_dt` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`username`)
)
```

Mit dem Befehl *vagrant up* kann nun eine VM erzeugt/gestartet werden. Diese installiert automatisch eine MySQL Datenbank mit dem Namen *users*. Es ist erlaubt mit dem Benutzer *user* auf diese Datenbank zuzugreifen. Dazu muss folgender Befehl verwendet werden:

```
mysql -h 192.168.56.56 -u user -puser
```

2.3 Webservice-Schnittstelle

Zu Beginn habe ich in IntelliJ ein neues maven-Projekt angelegt. Dabei habe ich meine aktuelle Java jdk-Installation und das zu verwendende Archtype angegeben. Dabei habe ich folgende Einstellungen getroffen: Meine Java Version ist 1.8 und Archtype habe ich ein neues hinzugefügt.
GroupId: org.codehause.mojo.archetype
ArtifactId: webapp-j2ee14,
Version: 1.1.

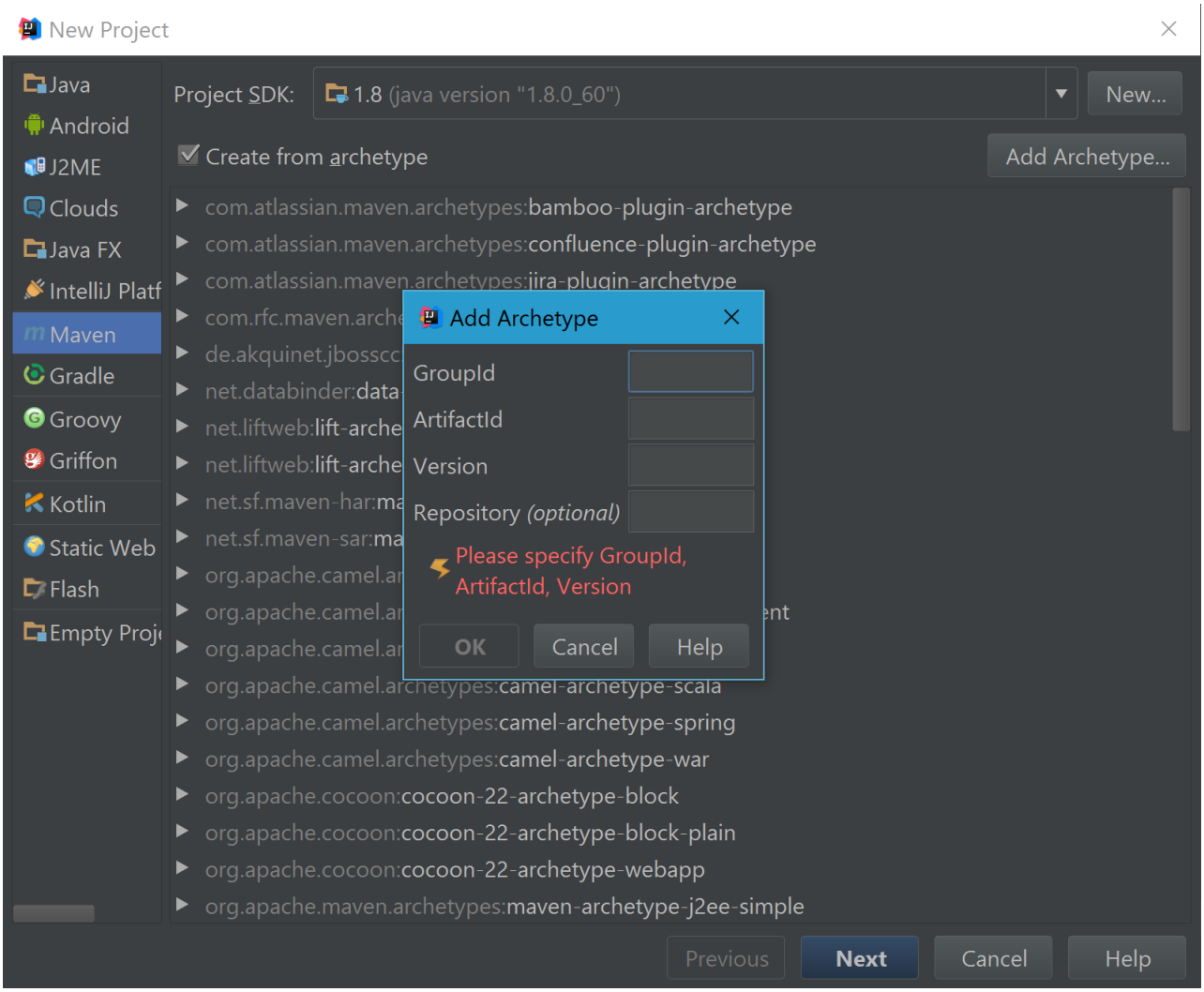


Abbildung 1: Anlegen eines neuen Projektes

Anschließend habe ich noch eine GroupId, sowie eine ArtefactId für mein Projekt vergeben und habe es erzeugt.

Danach mit Hilfe des Tutorials[ANDR2] ein Restful Webservice erstellt. Zuerst habe ich meine *web.xml* Datei geöffnet und den Inhalt durch den Folgenden ersetzt. Web.xml befindet sich unter src/main/webapp/WEB-INF/web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>WebServicesInJava</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <servlet>
        <servlet-name>Jersey REST Service</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>DEZSYS09.WebServicesInJava.ernhofer</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Jersey REST Service</servlet-name>
        <url-pattern>/ernhofer/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Anschließend habe ich in dem Package DEZSYS09.WebServicesInJava.ernhofer meine ersten Java Dateien angelegt Dabei habe ich mich zuerst bei dem Tutorial[ANDR2] von Android Guru erkundigt und anschließend ähnliche Klassen erstellt. Alle diese Dateien finden sich in meinem Repository wieder. [REPD] Begonnen habe ich mit der *Constants.java*. Diese Datei beinhaltet alle meine Konstanten. Im Konkreten, meine Werte um auf die Datenbank zugreifen zu können.
Allein durch das Festlegen der Verbindungsdaten besteht jedoch noch keine Verbindung. Daher habe ich eine Datei *DBConnection.java* erstellt. Diese ist dafür zuständig eine Verbindung mit der Datenbank herzustellen und die Daten zu verwalten.
Die dritte Datei trägt den Namen *Utility.java*. Diese ist dafür zuständig JSON-Objekte zu erzeugen und zu verwalten.
Die nächsten beiden Klassen, Register und Login, beinhalten den wichtigsten Teil für das Web Service. Sie kümmern sich um das Layout der URL und was anschließend passieren soll.

Als Erste der Beiden habe ich die Register Klasse erzeugt, da Benutzer logischerweise zuerst registriert werden müssen, bevor sie sich einloggen können.

```
//Path: http://localhost/<appln-folder-name>/register
@Path("/register")
public class Register {
   // HTTP Get Method
   @GET
   // Path: http://localhost/<appln-folder-name>/register/doregister
   @Path("/doregister")
   // Produces JSON as response
   @Produces(MediaType.APPLICATION_JSON)
   // Query parameters are parameters: http://localhost[:port]/<appln-folder-
name>/register/doregister?name=pqrs&username=abc&password=xyz
   public String doLogin(@QueryParam("name") String name,
@QueryParam("username") String uname, @QueryParam("password") String pwd){
      String response = "";
      int retCode = registerUser(name, uname, pwd);
      if(retCode == 0){
         response = Utility.constructJSON("register",true);
      }else if(retCode == 1){
         response = Utility.constructJSON("register",false, "You are already
registered");
      }else if(retCode == 2){
         response = Utility.constructJSON("register",false, "Special Characters
are not allowed in Username and Password");
      }else if(retCode == 3){
         response = Utility.constructJSON("register",false, "Error occured");
      }
      return response;
   }
   private int registerUser(String name, String uname, String pwd){
      int result = 3;
      if(Utility.isNotNull(uname) && Utility.isNotNull(pwd)){
         try {
            if(DBConnection.insertUser(name, uname, pwd)){
               result = 0;
            }
         } catch(SQLException sqle){
       //When Primary key violation occurs that means user is already registered
            if(sqle.getErrorCode() == 1062){
               result = 1;
            }
            //When special characters are used in name,username or password
            else if(sqle.getErrorCode() == 1064){
               System.out.println(sqle.getErrorCode());
               result = 2;
            }
         }
         catch (Exception e) {
            result = 3;
         }
      }else{
         result = 3;
      }
      return result;
   }
}
```

Diese Klasse kümmert sich nun darum einen Benutzer zu registrieren Die URL um einen Benutzer anzumelden stellt sich nun wie folgt zusammen, bzw. wird sie in dieser Reihenfolge abgearbeitet. Gut veranschaulicht wird dies in einem Bild des Tutorials von Android Guru, auch wenn der Code nicht zu 100% meinem Entspricht.


```
1  package com.prgguru.jersey;
2
3  import java.sql.SQLException;
10 //Path: http://localhost/<appln-folder-name>/register
11 @Path("/register")
12 public class Register {
13     // HTTP Get Method
14     @GET
15     // Path: http://localhost/<appln-folder-name>/register/doregister
16     @Path("/doregister")
17     // Produces JSON as response
18     @Produces(MediaType.APPLICATION_JSON)
19     // Query parameters are parameters: http://localhost/<appln-folder-name>/register/doregister?name=pqrs&username=abc&password=xyz
20     public String doLogin(@QueryParam("name") String name, @QueryParam("username") String uname, @QueryParam("password") String pwd){
21         String response = "";
22         //System.out.println("Inside doLogin "+uname+"  "+pwd);
23         int retCode = registerUser(name, uname, pwd);
24         if(retCode == 0){
25             response = Utitlity.constructJSON("register",true);
26         }else if(retCode == 1){
27             response = Utitlity.constructJSON("register",false, "You are already registered");
28         }else if(retCode == 2){
29             response = Utitlity.constructJSON("register",false, "Special Characters are not allowed in Username and Password");
30         }else if(retCode == 3){
31             response = Utitlity.constructJSON("register",false, "Error occured");
32         }
33         return response;
34
35     }
```

Abbildung 2: URL zusammensetzung beim Registrieren [ANDR2]

Als Antwort auf die Anfrage wird von der Klasse ein JSON-Objekt erzeugt. Dieses beinhaltet jedenfalls den Inhalt *„register*". Zusätzlich wird *true* angehängt, sollte die Registrierung erfolgreich gewesen sein. Sollte die Registrierung fehlschlagen, so wird dem JSON-Objekt *false* angehängt, sowie eine Beschreibung des aufgetretenen Fehlers. Wenn der Benutzer beispielsweiße schon registriert ist und versucht sich erneut zu registrieren, ist dies nicht möglich und es wird folgendes JSON-Objekt zurückgegeben:

```
{"tag":"register","status":false,"error_msg":"You are already registered"}
```

Sollte alles in Ordnung, und die Registrierung gelungen sein, so wird folgendes JSON zurückgegeben:

```
{"tag":"register","status":true}.
```

Die URL um einen Benutzer zu registrieren sieht beispielsweiße wie folgt aus:

http://localhost:19090/WebServicesInJava/ernhofer/register/doregister?name=Admin&username=admin@example.com&password=password.

Bei der Anwendung dieses Links ist es wichtig, dass die IP-Adresse sowie der Port an die Verwendeten Werte angepasst werden.

Um das Einloggen von Benutzern kümmert sich eine Klasse namens *Login*. Der Inhalt der Datei sieht folgendermaßen aus:

```
//Path: http://localhost/<appln-folder-name>/login
@Path("/login")
public class Login {
   // HTTP Get Method
   @GET
   // Path: http://localhost/<appln-folder-name>/login/dologin
   @Path("/dologin")
   // Produces JSON as response
   //@Produces(MediaType.APPLICATION_JSON)
    @Produces(MediaType.TEXT_PLAIN)
    // Query parameters are parameters: http://localhost/<appln-folder-
name>/login/dologin?username=abc&password=xyz
   public String doLogin(@QueryParam("username") String uname,
@QueryParam("password") String pwd){
      String response = "";
      if(checkCredentials(uname, pwd)){
         //response = Utility.constructJSON("login",true);
            response = "Login erfolgreich! Willkommen, sie sind als " + uname +
" angemeldet!";
      }else{
         //response = Utility.constructJSON("login", false, "Incorrect Email or
Password");
            response = "Login fehlgeschlagen! Email und Password stimmen nicht
ueberein!";
      }
   return response;
   }

   /**
    * Method to check whether the entered credential is valid
    *
    * @param uname
    * @param pwd
    * @return
    */
   private boolean checkCredentials(String uname, String pwd){
      System.out.println("Inside checkCredentials");
      boolean result = false;
      if(Utility.isNotNull(uname) && Utility.isNotNull(pwd)){
         try {
            result = DBConnection.checkLogin(uname, pwd);
            //System.out.println("Inside checkCredentials try "+result);
         } catch (Exception e) {
            //System.out.println("Inside checkCredentials catch");
            result = false;
         }
      }else{
         //System.out.println("Inside checkCredentials else");
         result = false;
      }

      return result;
   }
}
```

Die URL um einen Benutzer anzumelden stellt sich nun wie folgt zusammen, bzw. wird sie in dieser Reihenfolge abgearbeitet. Gut veranschaulicht wird dies in einem Bild des Tutorials von Android Guru, auch wenn der Code nicht zu 100% meinem Entspricht.

```
1  package com.prgguru.jersey;
2
3  import javax.ws.rs.GET;  http://localhost/<appln-folder-name>/login/dologin?username=hello@domain.com&password=welcome123
4  import javax.ws.rs.Path;
5  import javax.ws.rs.Produces;
6  import javax.ws.rs.QueryParam;
7  import javax.ws.rs.core.MediaType;
8  //Path: http://localhost/<appln-folder-name>/login
9  @Path("/login")
10 public class Login {
11     // HTTP Get Method
12     @GET
13     // Path: http://localhost/<appln-folder-name>/login/dologin
14     @Path("/dologin")
15     // Produces JSON as response
16     @Produces(MediaType.APPLICATION_JSON)
17     // Query parameters are parameters: http://localhost/<appln-folder-name>/login/dologin?username=abc&password=xyz
18     public String doLogin(@QueryParam("username") String uname, @QueryParam("password") String pwd){
19         String response = "";
20         if(checkCredentials(uname, pwd)){
21             response = Utitlity.constructJSON("login",true);
22         }else{
23             response = Utitlity.constructJSON("login", false, "Incorrect Email or Password");
24         }
25     return response;
26     }
27
28     /**
29      * Method to check whether the entered credential is valid
30      *
31      * @param uname
32      * @param pwd
33      * @return
34      */
```

Abbildung 3: URL Zusammensetzung beim Login [ANDR2]

Ich habe den Inhalt im Gegensatz zur Vorlage folgendermaßen Angepasst. Um der Aufgabenstellung „Bei einem erfolgreichen Login soll eine einfache Willkommensnachricht angezeigt werden.“ Folge zu leisten, habe ich die Art des Producers von `MediaType.APPLICATION_JSON` auf `MediaType.TEXT_PLAIN` geändert. Dies ermöglicht es mir anstatt eines JSON-Objektes einen Einfachen String zurückzugeben. Dadurch kann ich nach einem erfolgreichen Login eine Willkommensnachricht ausgeben. Diese sieht wie folgt aus:

Login erfolgreich! Willkommen, sie sind als [username] angemeldet!

Sollte das einloggen Fehlschalgen erscheint folgende Ausgabe:

Login fehlgeschlagen! Email und Password stimmen nicht ueberein!

Die URL um einen Benutzer zu registrieren sieht beispielsweiße wie folgt aus: http://localhost:19090/WebServicesInJava/ernhofer/login/dologin?username=admin@example.com&password=password

Bei der Anwendung dieses Links ist es wichtig, dass die IP-Adresse sowie der Port an die Verwendeten Werte angepasst werden.

Um das Projekt Compilier- und Ausführbar zu machen, musste ich anschließend noch Änderungen in der *pom.xml* Datei vornehmen. Wichtig war es dabei, keine der verwendeten Ressourcen zu vergessen. Wenn auch nur Eine fehlt funktioniert das Programm entweder gar nicht, oder nicht zuverlässig. Die wichtigsten Inhalte sind die Jersey Abhängigkeiten, sowie das automatische ausführen auf einem Tomcat-Server. Zuletzt genanntes wird durch folgende Codezeilen realisiert. Es handelt sich dabei aber nur um einen Ausschnitt aus der *pom.xml* Datei und nicht um den gesamten Inhalt!

```
<build>
    <defaultGoal>clean install tomcat:run</defaultGoal>
    <plugins>
        <plugin>
            <groupId>org.apache.tomcat.maven</groupId>
            <artifactId>tomcat7-maven-plugin</artifactId>
            <version>2.0</version>
        </plugin>
        <plugin>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.3</version>
        </plugin>
    </plugins>
</build>
```

Um eine Verbindung mit der Datenbank aufzubauen war weiter Folgende Ergänzung notwendig:

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.17</version>
</dependency>
```

Die gesamte *pom.xml* Datei findet sich in meinem Repository wieder. [REPD]

Nach diesen Schritten habe ich in IntelliJ unter Run -> *edit configurations* eine neue Konfiguration zum Ausführen eines Mavenprojektes angelegt. Anschließend habe ich diese gestartet. Wenn alles geklappt hat kann man unter *localhost:8080/WebServicesInJava* folgendes finden:

Abbildung 4: Startseite des Webservice

Das Registrieren und Einloggen kann mit folgenden Links durchgeführt werden:

http://localhost:8080/WebServicesInJava/ernhofer/register/doregister?name=Admin&username=admin@example.com&password=password

http://localhost:8080/WebServicesInJava/ernhofer/login/dologin?username=admin@example.com&password=password

Um zu testen, ob das Projekt Umfeld unabhängig ist, habe ich es, auf einer Debian VM, von Github gecloned. Zu Beginn hat es leider nicht gleich funktioniert. Da auf dieser VM der Port 8080 bereits von Jenkins verwendet wird bin ich auf den Port 19090 ausgewichen. Dies habe ich in der pom.xml festgelegt. Anschließend habe ich den Befehl

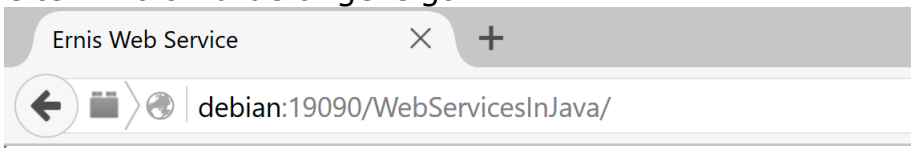
```
mvn clean install tomcat7:run
```

ausgeführt. Dadurch wurde das Projekt gestartet.

Auf meinem Host System habe ich anschließend die Startseite des auf der VM laufenden Webservice aufgerufen

debian:19090/WebServicesInJava/

Folgender Seiteninhalt wurde angezeigt:



Web Service by Andi Ernhofer

Abbildung 5: Startseite des Webservice auf der Debian VM

debian verlinkt dabei auf 192.168.48.234

Das Programm konnte somit auch auf Debian-Maschinen erfolgreich gestartet und verwendet werden.

2.4 Acceptance Tests

Startseite

ID: S01

Situation:
Ein Benutzer möchte die Startseite des Webservice aufrufen.

URL, mit der sich der Benutzer einloggen kann:
http://localhost:8080/WebServicesInJava

Erwartetes Ergebnis:
Im Browser erscheint die Startseite des Webservice. Diese beinhaltet, mit fetter Schrift, „Web Service by Andi Ernhofer".

Tatsächliches Ergebnis:
Im Browser erscheint eine leere Seite.

Ergebnis des Tests:
Der Test ist fehlgeschlagen, da die gewünschte Seite nicht geladen wurde. Die Lösung des Problems ist unter Probleme zu finden. Nach erfolgreicher Behebung des Fehlers ist ein neuer Test durchzuführen.

ID: S02

Situation:
Ein Benutzer möchte die Startseite des Webservice aufrufen.

URL, mit der sich der Benutzer einloggen kann:
http://localhost:8080/WebServicesInJava

Erwartetes Ergebnis:
Im Browser erscheint die Startseite des Webservice. Diese beinhaltet, mit fetter Schrift, „Web Service by Andi Ernhofer".

Tatsächliches Ergebnis:
Im Browser erscheint die gewünschte Startseite des Webservice.

Ergebnis des Tests:
Der Test ist erfolgreich.

Registrieren

ID: R01

Situation:
Ein Benutzer will sich registrieren.
Er gibt alle angeben an und er ist ein neuer Benutzer.

Angaben:
Name: David
Username: david@gmail.com
Password: david

URL um den Benutzer registrieren:
http://localhost:8080/WebServicesInJava/ernhofer/register/doregister?name=David&username=david@gmail.com&password=david

Erwartetes Ergebnis:
Da der Benutzer noch nicht in der Datenbank eingetragen ist, und alle erforderlichen Angaben getätigt hat soll ein neuer Benutzer in der Datenbank angelegt werden und eine Erfolgsmeldung im Browser erscheinen.

Tatsächliches Ergebnis:
Im Webbrowser erscheint folgende Ausgabe:

```
{"tag":"register","status":true}
```

In der Datenbank wurde außerdem folgender Eintrag angelegt:

```
+-------+-----------------+----------+---------------------+
| name  | username        | password | register_dt         |
+-------+-----------------+----------+---------------------+
| David | david@gmail.com | david    | 2016-02-12 22:35:12 |
+-------+-----------------+----------+---------------------+
```

Ergebnis des Tests:
Der Test kann als erfolgreich gewertet werden, da alle Angaben erfolgreich in einen neuen Datenbankeintrag übernommen wurden. Das einloggen dieses Benutzers wird im Test L05 getestet.

ID: R02

Situation:
Ein Benutzer will sich registrieren.
Es handelt sich um einen neuen Benutzer, jedoch gibt er keinen Namen an.

Angaben:
Name:
Username: florian@live.de
Password: esel

URL um den Benutzer registrieren:
http://localhost:8080/WebServicesInJava/ernhofer/register/doregister?username=florian@live.de&password=esel

Erwartetes Ergebnis:
Der Benutzer ist in der Datenbank noch nicht vorhanden. Von dieser Seite sollte es also keine Probleme geben. Jedoch hat der Benutzer keinen Namen angegeben. Daher sollte er nicht registriert werden und es soll kein Datenbankeintrag für ihn angelegt werde.

Tatsächliches Ergebnis:
Im Browser wird folgende Meldung angezeigt und es wird kein neuer Eintrag in der Datenbank angelegt:

```
{"tag":"register","status":false,"error_msg":"Error occured"}
```

Ergebnis des Tests:
Da der Benutzer nicht angelegt wurde sind die Erwartungen erfüllt worden und der Test kann als gelungen gewertet werden.

ID: R03

Situation:
Ein Benutzer möchte sich registrieren.
Der Benutzer ist noch nicht vorhanden, jedoch gibt er keine Benutzernamen an.

Angaben:
Name: Ferdinand
Username:
Password: fahrrad

URL um den Benutzer registrieren:
http://localhost:8080/WebServicesInJava/ernhofer/register/doregister?name=Ferdinand@live.de&password=fahrrad

Erwartetes Ergebnis:
Da der Benutzer keine Angaben zu seinem Benutzernamen gemacht hat, soll kein Eintrag in der Datenbank, für ihn, erzeugt werden.

Tatsächliches Ergebnis:
Im Browser wird folgende Meldung angezeigt und es wird kein neuer Eintrag in der Datenbank angelegt:

```
{"tag":"register","status":false,"error_msg":"Error occured"}
```

Ergebnis des Tests:
Da der Benutzer nicht angelegt wurde sind die Erwartungen erfüllt worden und der Test kann als gelungen gewertet werden.

ID: R04

Situation:
Ein Benutzer möchte sich registrieren.
Der Benutzer ist noch nicht vorhanden, jedoch gibt er kein Passwort an.

Angaben:
Name: Michael
Uname: michi@gmx.at
Password:

URL um den Benutzer registrieren:
http://localhost:8080/WebServicesInJava/ernhofer/register/doregister?name=Michael&username=florian@live.de

Erwartetes Ergebnis:
Da der Benutzer kein Passwort angegeben hat, soll kein Eintrag in der Datenbank erzeugt werden.

Tatsächliches Ergebnis:
Im Browser wird folgende Meldung angezeigt und es wird kein neuer Eintrag in der Datenbank angelegt:

```
{"tag":"register","status":false,"error_msg":"Error occured"}
```

Ergebnis des Tests:
Da der Benutzer nicht angelegt wurde sind die Erwartungen erfüllt worden und der Test kann als gelungen gewertet werden.

ID: R05

Situation:
Ein Benutzer möchte sich registrieren.
Er gibt alle erforderlichen Angaben an, jedoch hat er sich bereits registriert.

Angaben:
Name: David
Uname: david@gmail.com
Password: david

URL um den Benutzer registrieren:
http://localhost:8080/WebServicesInJava/ernhofer/register/doregister?name=David&username=david@gmail.com&password=david

Erwartetes Ergebnis:
Die Angaben des Benutzers sind korrekt, jedoch hat er sich bereits registriert und es soll daher nicht noch ein Eintrag von ihm erzeugt werde. Im Browser soll ein Text erscheinen der Angibt, dass sich der Benutzer bereits registriert hat.

Tatsächliches Ergebnis:
Es wird kein neuer Datenbankeintrag angelegt und im Browser erscheit folgendes:

```
{"tag":"register","status":false,"error_msg":"You are already registered"}
```

Ergebnis des Tests:
Da der Benutzer schon registriert ist, kann er sich nicht erneut registrieren.
Der Test ist erfolgreich.

ID: R06

Situation:
Ein Benutzer möchte sich registrieren.
Er gibt alle erforderlichen Angaben an. Das Webservice läuft, jedoch besteht keine Verbindung zu der Datenbank

Angaben:
Name: David
Uname: david@gmail.com
Password: david

URL um den Benutzer registrieren:
http://localhost:8080/WebServicesInJava/ernhofer/register/doregister?name=David&username=david@gmail.com&password=david

Erwartetes Ergebnis:
Die Angaben des Benutzers sind korrekt und es soll versucht werden einen neuen Eintrag in der Datenbank angelegt werden. Da jedoch keine Verbindung zur Datenbank besteht soll der Versuch abgebrochen werden.

Tatsächliches Ergebnis:
Nach einiger Zeit erscheint folgende Meldung im Browser und es wird kein neuer Benutzer angelegt:
{"tag":"register","status":false,"error_msg":"Error occured"}
Ergebnis des Tests:
Der Test war erfolgreich, da kein neuer Benutzer angelegt wurde.

ID: R07

Situation:
Ein Benutzer möchte sich registrieren.
Er gibt alle erforderlichen Angaben an. Den Namen, welchen er verwenden möchte gibt es schon. Benutzername und Passwort sind allerdings unterschiedlich zu dem Eintrag, welcher den gleichen Namen aufweist.

Angaben:
Name: David
Uname: d.klammer@gmail.com
Password: ichmagzuege

URL um den Benutzer registrieren:
http://localhost:8080/WebServicesInJava/ernhofer/register/doregister?name=David&username=d.klammer@gmail.com&password=ichmagzuege

Erwartetes Ergebnis:
Die Angaben des Benutzers sind korrekt und vollständig. Obwohl ein Eintrag mit dem gleichen Namen schon vorhanden ist soll ein neuer Eintrag für den aktuellen Benutzer angelegt werden. Dies liegt daran, dass sich Name und Passwort durchaus mit anderen Einträgen überschneiden dürfen. Lediglich der Benutzername muss einmalig sein!

Tatsächliches Ergebnis:
Die Registrierung war Erfolgreich! Die Datenbank weißt danach folgenden Inhalt auf:

```
+-------+--------------------+------------+---------------------+
| name  | username           | password   | register_dt         |
+-------+--------------------+------------+---------------------+
| David | d.klammer@gmail.com | ichmagzuege | 2016-02-12 23:11:31 |
| David | david@gmail.com     | david      | 2016-02-12 23:03:47 |
+-------+--------------------+------------+---------------------+
```

Login

ID: L01

Situation:
Ein Benutzer möchte sich einloggen.
Er gibt alle Angeben vollständig und korrekt an. Der Benutzer ist jedoch noch nicht registriert.

Angaben:
Name: David
Username: david@gmail.com
Password: david

URL, mit der sich der Benutzer einloggen kann:
http://localhost:8080/WebServicesInJava/ernhofer/login/dologin?name=David&username=david@gmail.com&password=david

Erwartetes Ergebnis:
Der Benutzer soll sich nicht einloggen können, da er noch nicht registriert ist.

Tatsächliches Ergebnis:
Im Browser Erscheint folgende Fehlermeldung:

```
Login fehlgeschlagen! Email und Password stimmen nicht ueberein!
```

Ergebnis des Tests:
Der Benutzer konnte sich nicht einloggen. Der Test ist daher erfolgreich.

ID: L02

Situation:
Ein Benutzer möchte sich einloggen.
Er ist bereits registriert und gibt seinen Benutzernamen sowie sein Passwort an.

Angaben:
Username: david@gmail.com
Password: david

URL, mit der sich der Benutzer einloggen kann:
http://localhost:8080/WebServicesInJava/ernhofer/login/dologin?username=david@gmail.com&password=david

Erwartetes Ergebnis:
Der Benutzer kann sich einloggen und erhält eine einfache Willkommensmeldung

Tatsächliches Ergebnis:
Der Benutzer konnte sich einloggen und folgende Nachricht erschien:
```
Login erfolgreich! Willkommen, sie sind als david@gmail.com angemeldet!
```

Ergebnis des Tests:
Der Test kann als erfolgreich gewertet werden.

ID: L03

Situation:
Ein Benutzer möchte sich einloggen.
Er ist bereits registriert und gibt seinen Namen sowie sein Passwort an.

Angaben:
Name: David
Password: david

URL, mit der sich der Benutzer einloggen kann:
http://localhost:8080/WebServicesInJava/ernhofer/login/dologin?Name=David@gmail.com&password=david

Erwartetes Ergebnis:
Der Benutzer kann sich nicht einloggen, da er seinen Benutzernamen nicht angegeben hat.

Tatsächliches Ergebnis:
Der Benutzer konnte sich nicht einloggen und folgende Nachricht erschien:

```
Login fehlgeschlagen! Email und Password stimmen nicht ueberein!
```

Ergebnis des Tests:
Der Test kann als erfolgreich gewertet werden.

ID: L04

Situation:
Ein Benutzer möchte sich einloggen.
Er ist bereits registriert und gibt seinen Namen, sowie seinen Benutzernamen an.

Angaben:
Name: David
Username: david@gmail.com

URL, mit der sich der Benutzer einloggen kann:
http://localhost:8080/WebServicesInJava/ernhofer/login/dologin?name=David&username=david@gmail.com

Erwartetes Ergebnis:
Der Benutzer kann sich einloggen und erhält eine einfache Willkommensmeldung

Tatsächliches Ergebnis:
Der Benutzer kann sich nicht einloggen, da er sein Passwort nicht angegeben hat.

Tatsächliches Ergebnis:
Der Benutzer konnte sich nicht einloggen und folgende Nachricht erschien:

```
Login fehlgeschlagen! Email und Password stimmen nicht ueberein!
```

Ergebnis des Tests:
Der Test kann als erfolgreich gewertet werden.

ID: L05

Situation:
Ein Benutzer möchte sich einloggen.
Er ist bereits registriert und gibt seinen Namen, Benutzernamen sowie sein Passwort an.

Angaben:
Name: David
Username: david@gmail.com
Password: david

URL, mit der sich der Benutzer einloggen kann:
http://localhost:8080/WebServicesInJava/ernhofer/login/dologin?name=David&username=david@gmail.com&password=david

Erwartetes Ergebnis:
Der Benutzer kann sich einloggen und erhält eine einfache Willkommensmeldung

Tatsächliches Ergebnis:
Der Benutzer konnte sich einloggen und folgende Nachricht erschien:
```
Login erfolgreich! Willkommen, sie sind als david@gmail.com angemeldet!
```

Ergebnis des Tests:
Der Test kann als erfolgreich gewertet werden.

ID: L06

Ein Benutzer möchte sich einloggen.
Er ist bereits registriert und gibt seinen Benutzernamen sowie sein Passwort an. Es besteht jedoch keine Verbindung zur Datenbank.

Angaben:
Username: david@gmail.com
Password: david

URL, mit der sich der Benutzer einloggen kann:
http://localhost:8080/WebServicesInJava/ernhofer/login/dologin?username=david@gmail.com&password=david

Erwartetes Ergebnis:
Der Benutzer kann sich nicht einloggen, weil keine Verbindung mit der Datenbank hergestellt werden kann.

Tatsächliches Ergebnis:
Der Benutzer konnte sich einloggen, weil seine Angaben nicht mit den bereits registrierten Benutzern verglichen werden konnte.

Ergebnis des Tests:
Der Test kann als erfolgreich gewertet werden.

Testergebnisse:

Gescheitert
Erfolgreich

S01
S02
R01
R02
R03
R04
R05
R06
R07
L01
L02
L03
L04
L05
L06

3 Probleme

Eclipse

Zu Beginn des Projektes habe ich mit Eclipse gearbeitet. Genauer gesagt mit der JEE Version von Eclipse. Dass ich mich für diese Programmierumgebung entschieden habe, lag daran, dass es in dem Tutorial von Androidguru so empfohlen war. Da das Tutorial sehr vielversprechend aussah habe ich diese Empfehlung auch befolgt. Ich habe das gesamte Tutorial Schritt für Schritt abgearbeitet und war überrascht, wie Problemlos alles funktionierte. Ich konnte anschließend Benutzer registrieren sowie einloggen. So weit so gut, jedoch musste ich, um die Aufgabenstellung zu erfüllen, ein Build Tool verwenden. Dieses war bis zu diesem Punkt nicht eingebunden. Genau an diesem Punkt begannen jedoch meine Probleme. Nach Stundenlanger Arbeit bin ich von Fehler zu Fehler gewandert und es war kein Licht am Ende des Tunnels in Sicht. Ich hatte es geschafft ein Projekt anzulegen, welches alle benötigten Daten mittels Maven ladet. Das Einzige, was nicht möglich war, war es das Programm automatisch auf einem Tomcat Server auszuführen. Es war somit also nicht Zufriedenstellend. Nach weiterer Arbeit hatte ich es geschafft ein neues Projekt anzulegen, welches automatisch mittels maven auf einem Tomcat-Server ausgeführt wurde. Sobald ich jedoch mein ursprüngliches Programm, mit Registrier- und Anmeldefunktion, auf diesem Server ausführen wollte gab es wieder Probleme. Insgesamt bin ich über Probleme wie beispielsweise Verbindungsschwierigkeiten bedingt durch Berechtigungsfehler auf localhost:8080 gestoßen. Ein weiteres Schwerwiegendes Problem war es, dass Eclipse fortan immer meine Java jre Installation verwendet hat. Nach Recherchen im Internet und vielen Einstellungsänderungen in Eclipse und in meinen Umgebungsvariablen war das Problem jedoch immer noch nicht beseitigt. Das konkrete Problem war es, dass Eclipse mitteilte, dass es mit jre meinen Code nicht kompilieren kann, sondern ein jdk dazu braucht. Es war mir nicht geläufig, warum vorherige Projekte, welche ich ausführen konnte, diesen Fehler nicht erzeugten. Schlussendlich habe ich keine Lösung gefunden und nach stundenlanger Research auch keine Ideen mehr gehabt, was ich noch ausprobieren könnte.

Eine andere Lösung musste her und so bin ich zu dem Entschluss gekommen eine andere Programmierumgebung zu verwenden. Meine Wahl fiel dabei auf IntelliJ. Mittels diesem Programm habe ich es anschließend geschafft alle Probleme, welche ich durch die Verwendung von Eclipse erhalten habe zu beseitigen und ein lauffähiges Programm zu erzeugen.

Vagrant

Ich habe das Problem, dass ich mittels vagrant kein privates Netzwerk konfigurieren kann. Wenn ich im *Vagrantfile* eine Zeile wie Beispielsweiße `config.vm.network "private_network", ip: "192.168.56.56"` schreibe wird automatisch ein neues Hostonly Netzwerk erstellt, welches eine Adresse von 169.153.*.* besitzt. Anschließend erfolgt eine Meldung, dass vagrant keinen privaten Adapter erzeugen konnte. Folgende Fehlermeldung wird ausgegeben:

```
==> default: Clearing any previously set network interfaces...
There was an error while executing `VBoxManage`, a CLI used by Vagrant
for controlling VirtualBox. The command and stderr is shown below.

Command: ["hostonlyif", "create"]

Stderr: 0%...
Progress state: E_FAIL
VBoxManage.exe: error: Failed to create the host-only adapter
VBoxManage.exe: error: Operation canceled by the user
VBoxManage.exe: error: Details: code E_FAIL (0x80004005), component
VirtualBox,
interface IVirtualBox
VBoxManage.exe: error: Context: "int __cdecl handleCreate(struct
HandlerArg *,int,int *)" at line 66 of file VBoxManageHostonly.cpp
```

Im Internet gibt es dazu jede Menge Tickets. Es scheint als ob es sich dabei um ein bekanntes Problem, bei der Kombination von Vagrant, VirtualBox und Windows10, handelt. Als Lösung wird eine .exe Datei angeboten, welche angeblich alle Probleme beseitigt. Aus Gründen der Ungewissheit, was diese Datei wirklich ausführt und weil ich meinen Rechner noch länger brauche, habe ich auf diesen Lösungsvorschlag verzichtet. Stattdessen hatte ich einen anderen Weg gefunden dieses Problem zu umgehen. Dazu habe ich *vagrant up* ausgeführt. Wie erwartet kam die Fehlermeldung. Die Instanz wird jedoch erstellt. Händisch habe ich einen zusätzlichen Netzwerkadapter hinzugefügt, welcher die VM mittels Host-only Netzwerks mit meinem Host-System verbinden soll.

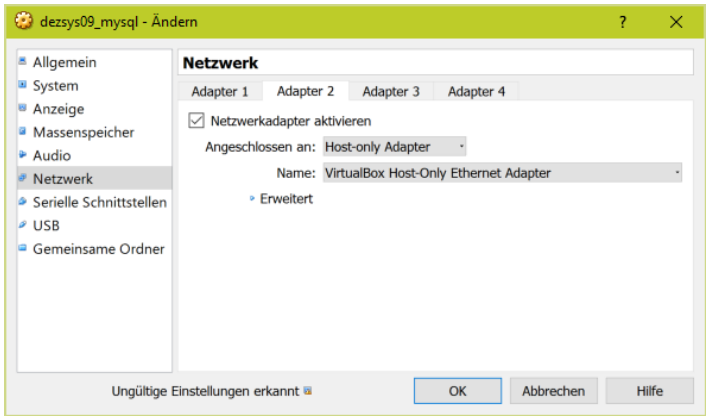


Abbildung 6: Manuelles einfügen eines Adapters

Anschließend habe ich die VM manuell gestartet, wieder beendet und erneut den Befehl *vagrant up* ausgeführt. Obwohl die Netzwerkschnittstellen beim Ausführen dieses Befehls erneut erzeugt werden funktioniert nach dem zweiten Mal alles und die VM wird wie gewünscht gestartet.
Dies ist zwar nicht die optimale Lösung, jedoch funktioniert sie. Zufriedenstellend war sie deshalb aber nicht und so habe ich mich wieder auf die Suche nach anderen Lösungen gemacht. Immerhin war es genaugenommen keine Lösung, sondern nur ein Weg das Problem zu umgehen. Etwas Besseres habe ich jedoch auch nach weiteren Recherchen nicht gefunden.
Mit dem Stand, welches das Projekt mit dem Datum 16.02.2016 hat, ist es ohne Probleme Möglich die VM mittels *vagrant up* zu starten. Dabei ist es egal ob die Instanz schon vorhanden ist, oder ob sie zum ersten Mal erzeugt wird. Meiner Ansicht nach gibt es zwei Möglichkeiten, weshalb es zu Beginn Probleme gab und nun reibungslos Funktioniert. Hypothese 1 ist, dass es durch das einmalige händische Eingeben des zusätzlichen Netzwerkadapters nun immer wieder Funktioniert. Gegen diese Behauptung spricht jedoch, dass ich Anfang jedes Mal, nachdem ich die VM mittels *vagrant destroy* beendet habe und sie mittels *vagrant up* wieder erzeugen wollte, immer beim ersten Mal händische Änderungen vornehmen musste. Die zweite Hypothese, so unglaublich sie auch klingt, besteht darin, dass durch das Hinzufügen von

```
config.vm.provider "virtualbox" do |v|
  v.name = "dezsys09_mysql"
end
```

dieses Problem gelöst wurde. Vermutlich handelt es sich um einen Zufall und etwas Anderes hat das Problem gelöst, jedoch taucht der Fehler nicht mehr auf seit ich diese Zeilen dem Vagrantfile hinzugefügt habe. Alleine diese Zeilen können es nämlich nicht sein, da, als ich bei einem Anderen Vagrantfile diese Zeilen eingefügt habe, immer noch das Problem mit dem privaten Netzwerk bestand. Die Lösung hält sich somit immer noch versteckt, jedoch bin ich froh mit den aktuellen Einstellungen eine lauffähige Version zu haben.

Startseite

Nachdem das Projekt erstellt war und gestartet werden konnte, wurde meine Startseite nicht angezeigt. Anstatt der gewünschten Anzeige war nur eine leere Seite zu sehen. Die Lösung des Problems bestand darin das url-Pattern in der *web.xml* Datei von `/*` auf `/ernhofer/*` zu ändern.

Login

Nach dem fertigen Konfigurieren des Projektes war es möglich Benutzer anzumelden, jedoch bekam ich jedes Mal einen Fehler, wenn ich einen Benutzer anmelden wollte. Nach langer Fehlersuche in meinem Code hatte ich nicht verstanden, weshalb beim Registrieren alles klappt, beim Login jedoch immer Fehler auftreten. Nach mehreren Kontrollen in der Datenbank und bei den Werten des anzumeldenden Benutzers hatte ich den Fehler gefunden. Aus einem Vermutlichen Copy-Paste Fehlers hatte sich bei dem Passwort-Eintrag in der Datenbank am Ende des Passwortes ein „." angefügt. Nach Löschen und erneut anlegen des Benutzers war das Problem behoben und das Registrieren sowie Einloggen von Benutzern war möglich.

Maven-Vagrant

Ich wollte es ermöglichen, dass automatisch eine VM erzeugt/gestartet wird, wenn das Projekt gestartet wird. Dazu habe ich ein maven-plugin gefunden. [MAVV] Dieses habe ich in meine pom.xml aufgenommen und zu den default Goals *vagrnat:up* hinzugefügt. Dabei sind mir anschließend gleich 2 Probleme aufgetreten. Ich fand keine Möglichkeit diesem Plugin meinen Pfad zu meinem Vagrantfile anzugeben. Dieses Problem war jedoch für Probezwecke nicht wirklich schwerwiegend, und konnte durch einfaches Kopieren der Vagrant Daten in den Root Ordner des Projektes behoben werden. Das zweite Problem war jedoch nicht so leicht zu Lösen. Beim ausführen des Befehls *vagrant:up* erschien folgende Fehlermeldung:

```
Failed to execute goal net.ju-n.maven.plugins:vagrant-maven-plugin:1.0.1:up.
```

Der Grund, dass es nicht ausgeführt werden konnte war:

```
NoMethodError: undefined method `configure' for Vagrant:Module
```

Nach Recherchen im Internet habe ich folgende Lösung für dieses Problem gefunden[LOEV] Diese besagt, dass mit der Vagrant Version 1.0.1 kein vagrant.config möglich ist. Als Lösung wird geraten eine neuere Vagrat Version zu verwenden. Normalerweise kein Problem. Einfach eine neuere Version installieren und die Sache ist erledigt. Da ich jedoch mit dem Maven Plugin arbeite ist es nicht so einfach. Ich bin an die Version des Plugins angewiesen. Eine neuere (genau genommen gar keine andere) Version habe ich jedoch nicht gefunden und somit konnte das Problem nicht gelöst werden. Die einzige Möglichkeit, welche halbwegs akzeptabel ist, ist es zuerst manuell die vagrant VM zu erzeugen/starten und anschließend das Projekt mittels maven zu generieren.

4 Zeitaufzeichnung

Tätigkeit	Dauer
Mittels Vagrant Datenbank zur Verfügung Stellen	120min
Aufsetzen einer Webservice-Schnittstelle	120min
Registrierung von Benutzern mit entsprechender Persistierung	30min
Login und Rückgabe einer Willkommensnachricht	60min
AcceptanceTests	180min
Problembearbeitung und Fehlerbehebung	720min
Protokollierung	270min
Gesamt	25h

5 Quellen

[ANDR1]"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/
zuletzt besucht 15.02.2016

[ANDR2]"Android Restful Webservice Tutorial – Introduction to RESTful webservice – Part 2"; Posted By Android Guru on May 11, 2014; online: http://programmerguru.com/android-tutorial/android-restful-webservice-tutorial-how-to-create-restful-webservice-in-java-part-2/
zuletzt besucht 17.02.2016

[RESJ] "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
zuletzt besucht 15.02.2016

[HERM] "Heroku makes it easy to deploy and scale Java apps in the cloud"; online: https://www.heroku.com/
zuletzt besucht 15.02.2016

[VAGM] "Vagrant Vorlage für eine VM mit MySQL" https://github.com/AlexDisler/mysql-vagrant
zuletzt besucht 16.02.2016

[DOCJ] „Jersey Dokumentation" https://jersey.java.net/documentation/latest/user-guide.html
zuletzt besucht 16.02.2016

[REPD] „Git-Repository des Projektes" https://github.com/aernhofer-tgm/DEZSYS09/
zuletzt besucht 17.02.2016

[MAVV] „Vagrant Maven plugin" http://nicoulaj.github.io/vagrant-maven-plugin/#
zuletzt besucht 17.02.2016

[LOEV]"undefined method configure for Vagrant:Module" http://www.fvue.nl/wiki/Vagrant:_Vagrantfile:4:_undefined_method_configure_for_Vagrant:Module
zuletzt besucht 17.02.2016