

**Laborprotokoll**

**Web Services in Java**

**Systemtechnik Labor**

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

**Andreas Ernhofer**

|  |  |
| --- | --- |
|  | **Version 1.0** |
| **Note:** | **Begonnen am 12. Februar 2016** |
| **Betreuer: M. Borko** | **Beendet am 19. Februar 2016** |

Inhaltsverzeichnis

[1 Einführung 3](#_Toc443458980)

[1.1 Ziele 3](#_Toc443458981)

[1.2 Voraussetzungen 3](#_Toc443458982)

[1.3 Aufgabenstellung 3](#_Toc443458983)

[1.4 Bewertung 3](#_Toc443458984)

[2 Ergebnisse 4](#_Toc443458985)

[2.1 Datenbank 4](#_Toc443458986)

[2.2 Jersey – RESTful Webservice 6](#_Toc443458987)

[3 Probleme 11](#_Toc443458988)

[4 Zeitaufzeichnung 13](#_Toc443458989)

[5 Quellen 14](#_Toc443458990)

# Einführung

Diese Übung zeigt die Anwendung von mobilen Diensten in Java.

## 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.

## Voraussetzungen

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

## 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.

## 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)

# Ergebnisse

## Datenbank

Für mein Webservice benötige ich eine Datenbank um meine Userdaten zu speichern. Dabei habe ich mich für eine MySQL-Datenbank entschieden. Um diese Datenbank zur 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 an ein 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 instll.sh ausgeführt um ein Update auszuführen, sowie mysql zu installieren und 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

## Jersey – RESTful Webservice

Zu Beginn habe ich in IntelliJ ein neues maven-Projekt angelegt. 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 einen neuen hinzugefügt.

GroupId: org.codehause.mojo.archetype

ArtifactId: webapp-j2ee14,

Version: 1.1.

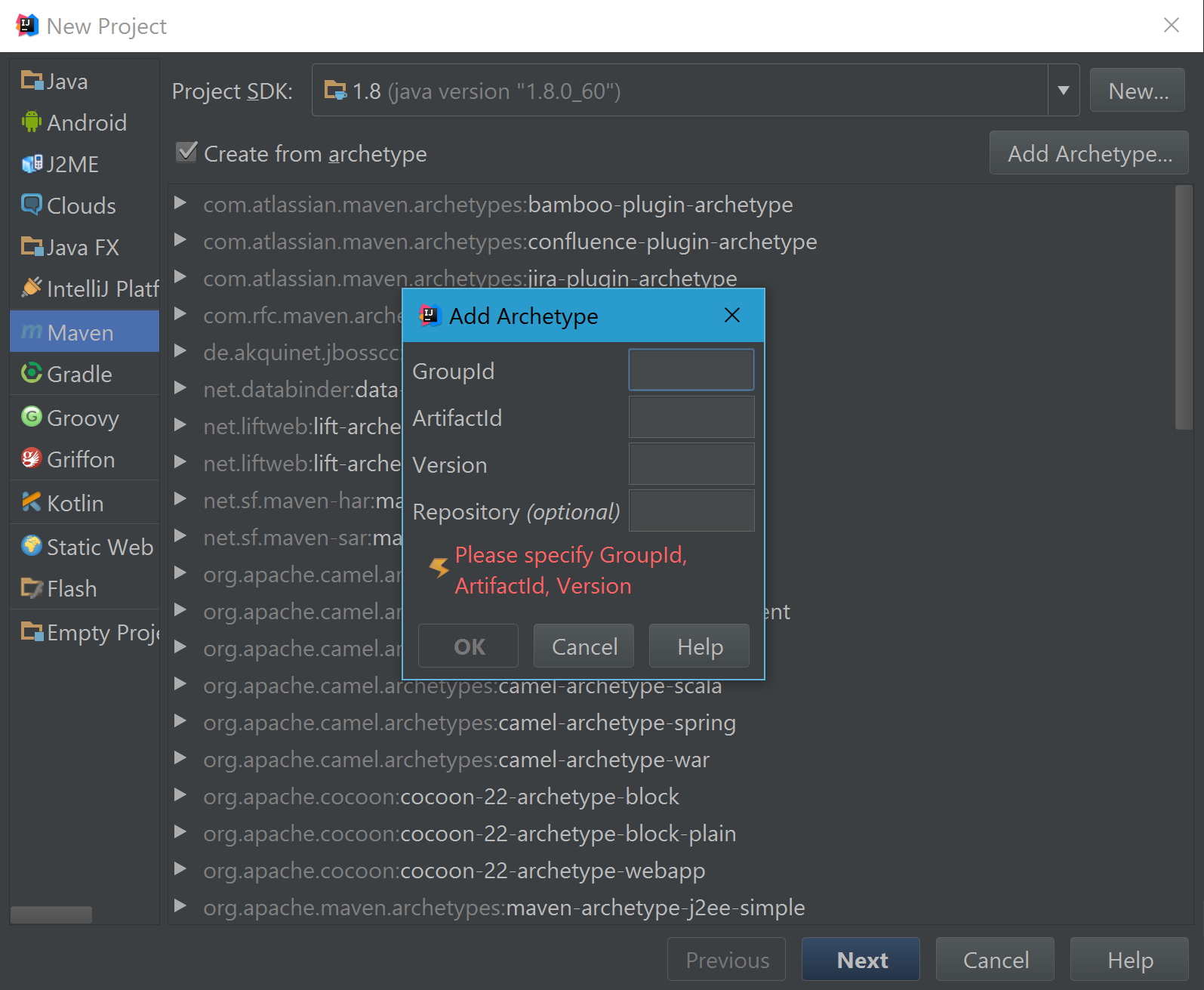


Abbildung 1: Anlegen eines neuen Projektes

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. 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.



Als Antwort auf die Anfrage wird von der Klasse ein JSON 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 *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 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:8080/Web-Services-in-Java/ernhofer/register/doregister?  
name=Admin&username=admin@example.com&password=password](http://localhost:8080/Web-Services-in-Java/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.

TODO: Ev. Statt JSON antwort eine einfach text/plain Antwort…

Register & Login

Links zum login/register (Bild androidguru…)

Pom.xml

Code Dissection (Tabelle android Guru)

# Probleme

**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, welcher 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 wir 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.

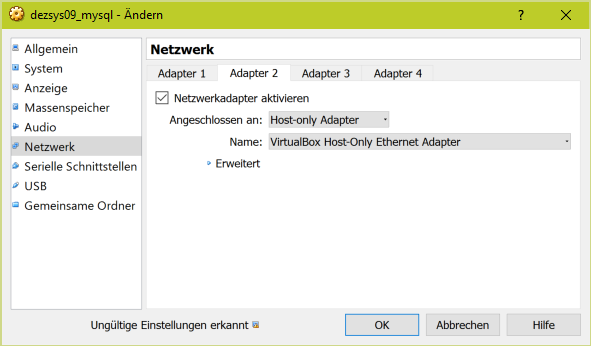


Abbildung 2: Manuelles einfügen eines Adapters

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 erzeugt und gestartet.

Dies ist zwar nicht die optimale Lösung, jedoch funktioniert sie. Zufriedenstellend war sie deshalb aber nicht für mich und so habe ich mich wieder auf die Suche nach anderen Lösungen gemacht. Immerhin war meine Lösung meinerseits und genaugenommen war es 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 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. Möglicherweise 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 jedoch nicht die Lösung 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.

Eclipse

Maven - tomcat

IntelliJ

Index.jsp wird nicht angezeigt

* Lösung url pattern in web.xml

# Zeitaufzeichnung

|  |  |
| --- | --- |
| Tätigkeit | Dauer |
| Vagrant | 120min |
|  |  |
|  |  |
|  |  |
| Gesamt |  |

# 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/>

[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/>

"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/>

[VAGM]

<https://github.com/AlexDisler/mysql-vagrant>

zuletzt besucht 16.02.2016

<https://jersey.java.net/documentation/latest/user-guide.html>