Laborprotokoll

DEZSYS-09: Webservices in Java

Systemtechnik
5BHITT 2015/16

Rene Hollander

Version 1.0

Note:

Betreuer: Borko

Begonnen am 12. Februar 2016

Beendet am 18. Februar 2016

Inhaltsverzeichnis

Einf hrung

Diese Übung zeigt die Anwendung von mobilen Diensten in Java.

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.

2 Voraussetzungen

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

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..

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

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)

Ergebnisse

1 Projekt Setup

Mithilfe des Tutorials Bootiful Java EE in Spring[1] wurde ein Projekt mit Maven als Buildtool aufgesetzt. Das veränderte POM File sieht wie folgt aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>at.renehollander</groupId>
    <artifactId>webservices</artifactId>
    <version>0.0.1</version>
    <packaging>jar</packaging>
    <repositories>
        <repository>
            <id>spring-snapshots</id>
            <name>Spring Snapshots</name>
            <url>https://repo.spring.io/snapshot</url>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </repository>
        <repository>
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>https://repo.spring.io/milestone</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
            <id>spring-snapshots</id>
            <name>Spring Snapshots</name>
            <url>https://repo.spring.io/snapshot</url>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </pluginRepository>
        <pluginRepository>
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>https://repo.spring.io/milestone</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </pluginRepository>
    </pluginRepositories>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.2.0.RELEASE</version>
    </parent>
    <dependencies>
        <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-jersey</artifactId>
        </dependency>
        <dependency>
            <groupId>org.glassfish.jersey.media</groupId>
            <artifactId>jersey-media-json-jackson</artifactId>
            <version>${jersey.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-undertow</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <exclusions>
                <exclusion>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <start-class>at.renehollander.webservices.Application</start-class>
        <java.version>1.8</java.version>
    </properties>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

2 Code

In Application.java wird die eigentliche Spring Applikation gestartet:

```
@SpringBootApplication
public class Application {
     public static void main(String[] args) {
          SpringApplication.run(Application.class, args);
     }
}
```

Damit Jersey verwendet wird um die REST APIs zur Verfügung zu stellen ist Folgende Konfiguration nötig:

```
@Named
public class JerseyConfig extends ResourceConfig {
     public JerseyConfig() {
          this.register(UserEndpoint.UserRegisterEndpoint.class);
          this.register(UserEndpoint.UserLoginEndpoint.class);
          this.register(JacksonFeature.class);
     }
}
```

Auch wurde eine Klasse User erstellt die Persistiert wird. Die Klasse wurde so konstruiert, dass Jersey den eingehenden HTTP JSON Request auf die User Klasse mappt und einen SHA256 Hash erstellt. Persistiert wird nur der Passwort Hash, nicht das Passwort im Klartext.

```
@Entity
public class User {
     @Id
     private String email;
     private byte[] passwordHash;
     @JsonCreator
     public User(@JsonProperty("email") String email, @JsonProperty("password") String
password) {
          this(email, hash(password));
     }
     public User(String email, byte[] passwordHash) {
          this.email = email;
          this.passwordHash = passwordHash;
     }
     public User() {
     }
     public String getEmail() {
          return email;
     }
     public byte[] getPasswordHash() {
          return passwordHash;
     }
     private static byte[] hash(String password) {
          try {
               MessageDigest md = MessageDigest.getInstance("SHA-256");
               md.update(password.getBytes("UTF-8"));
               return md.digest();
          } catch (Exception e) {
               throw new RuntimeException(e);
          }
     }
}
```

Um einen User persistieren zu können muss noch ein UserRepository angelegt werden.

```
public interface UserRepository extends CrudRepository<User, String> {
}
```

Zuletzt können die eigentlichen REST Endpoints erstellt werden:

Über /register wird ein User angelegt. Sollte der User bereits existieren, wird ein Fehlercode und eine Fehlernachricht als JSON zurückgegeben. Wenn der User noch nicht existiert, wird er angelegt und eine Bestätigung an den User zurückgesendet

```
@Named
@Path("/register")
@Produces({MediaType.APPLICATION_JSON})
public static class UserRegisterEndpoint {
     @Autowired
     private UserRepository userRepository;
     @POST
     public Response post(User user) {
          if (userRepository.exists(user.getEmail())) {
               return Response.status(400).entity(Maps.of("success", false)).build();
          } else {
               userRepository.save(user);
               return Response.status(201).entity(Maps.of("success", true)).build();
          }
     }
}
```

Über den /login Endpoint werden die Daten im POST Request überprüft und ein success: true zurückgeliefert wenn eMail und Passwort mit dem aus der Datenbank übereinstimmen.

3 Acceptance Tests

Registrierung von einem neuen User

Request: http://localhost:8080/register, POST, application/json

```
{
    "email": "rene.hollander@hotmail.de",
    "password": "1234"
}
```

Response: application/json, Code 201

```
{
  "success": true
}
```

Registrierung von einem User der bereits registriert ist

Request: http://localhost:8080/register, POST, application/json

```
{
    "email": "rene.hollander@hotmail.de",
    "password": "5678"
}
```

Response: application/json, Code 400

```
{
  "success": false
}
```

Login von einem User

Request: http://localhost:8080/login, POST, application/json

```
{
    "email": "rene.hollander@hotmail.de",
    "password": "1234"
}
```

Response: application/json, Code 200

```
{
  "success": true
}
```

Login von einem User mit falschem Passwort

Request: http://localhost:8080/login, POST, application/json

```
{
    "email": "rene.hollander@hotmail.de",
    "password": "5678"
}
```

Response: application/json, Code 403

```
{
  "success": false
}
```

Login von einem User mit falscher eMail

Request: http://localhost:8080/login, POST, application/json

```
{
    "email": "michael@borko.at",
    "password": "1234"
}
```

Response: application/json, Code 403

```
{
  "success": false
}
```

Login von einem User mit falscher eMail und Passwort

Request: http://localhost:8080/login, POST, application/json

```
{
    "email": "michael@borko.at",
    "password": "YAgTbhmKX7usPPYp"
}
```

Response: application/json, Code 403

```
{
  "success": false
```

4 Source Code

Der Code ist Verfügbar unter: https://github.com/ReneHollander/dezsys09-webservices

5 Kompilieren und Ausf hren

Als Build Tool wurde Maven verwendet. Zum Kompilieren der Software wird folgender Befehl verwendet: `mvn package`
Im `target/` Ordner befindet sich dann ein `webservices-0.0.1.jar`. Dieses Jar kann mit `java -jar webservices-0.0.1.jar` ausgeführt werden. Unter der Adresse `http://localhost:8080/` steht dann der Webservice zur verfügung.
Im Ordner db/ im CWD wird die Datenbank abgelegt.

Probleme

Wegen ein paar Schwierigkeiten Jersey mit Spring zum Laufen zu bekommen wurde etwas Zeit verschwendet. Die Implementierung der Datenbankanbindung und der Endpoints konnte recht flott erledigt werden.

Zeitaufwand

Zusätzlich zu den 3 Stunden musste eine weitere Stunde für die Vervollständigung des Protokolls, sowie für die Fehlerbehebung der Datenbank Speicherung genutzt werden.

Quellen

[1] „Bootiful“ Java EE Support in Spring Boot 1.2
Abrufbar unter: http://spring.io/blog/2014/11/23/bootiful-java-ee-support-in-spring-boot-1-2
zuletzt abgerufen: 12.02.2016