JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Cloud-Datenmanagement

Systemtechnik Labor
5BHIT 2017/18

Martin Wölfer

Version 2.1
Note:
Begonnen am 16. November 2017
Betreuer: Michael Borko
Beendet am 22. November 2017

Inhaltsverzeichnis

1 Einführung

Diese Übung zeigt die Anwendung von mobilen Diensten.

1.1 Ziele

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

Die Kommunikation zwischen Client und Service soll mit Hilfe einer REST Schnittstelle umgesetzt werden.

1.2 Voraussetzungen

- Grundlagen einer höheren Programmiersprache
- Verständnis über relationale Datenbanken und dessen Anbindung mittels ODBC oder ORM-Frameworks
- Verständnis von Restful Webservices

1.3 Aufgabenstellung

Es ist ein Webservice 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 (Acceptance-Tests bez. aller funktionaler Anforderungen mittels Unit-Tests) dokumentiert werden. Verwenden Sie auf jeden Fall ein gängiges Build-Management-Tool (z.B. Maven oder Gradle). Dabei ist zu beachten, dass ein einfaches Deployment möglich ist (auch Datenbank mit z.B. file-based DBMS).

2 Ergebnisse

Da der erste Versuch nicht funktionierte, wurde auf eine Realisierung in JavaEE gesetzt mit Unterstützung von `Spring Boot`, `Spring Security`, `Spring Data JPA` und der Datenbank `HSQL`. Es wurde mit folgendem Tutorial gearbeitet:

https://hellokoding.com/registration-and-login-example-with-spring-security-spring-boot-spring-data-jpa-hsql-jsp/

Das Endergebnis ist ein RESTful Webservice bei welchem man sich einloggen und registrieren kann und mit einer Willkommensnachricht begrüßt wird.

2.1 Allgemein

Zuerst mussten gewisse Themengebieten recherchiert werden. Vor allem das Themengebiet `Hibernate` musste von Grund auf gelernt und verstanden werden.

2.1.1 Hibernate

src: https://howtoprogramwithjava.com/hibernate-persistence-beginners/

Hibernate "steht" zwischen objektorientiertem Java und einem **R**elational **D**ata**B**ase **M**anagement **S**ystem.

Grundsätzlich dient Hibernate dazu, Java Objekte zu *persistieren*, also diese "permanent" erhältlich zu machen.

2.1.2 Spring

src: https://howtoprogramwithjava.com/podcast-episode-33-intro-to-spring-framework/

Hibernate wird sehr oft in Verindung mit `Spring` verwendet, Spring kümmert sich um die Kernfunktion einer Rest-Applikation, und zwar dem `Controller`. Mit Spring können auf bestimmte Links oder Link-patterns Funktionen ge`mapped` werden, welche wiederum beispielsweise eine `.jsp` page aufrufen.

2.1.3 Beans

Zwar ist dieses Thema Grundwissen von JavaEE, trotzdem hatte ich große Probleme Erklärungen zu verstehen, da ich nicht wusste im Kontext von Java was eine `Bean` ist.

Eine Bean ist lediglich ein Standard, welcher 3 folgende Eigenschaften vorschreibt:

1. Alle Attribute sind private (nur Getter/Setter)
2. Ein public Konstruktor ohne Parameter
3. Muss `Serializable` implementieren

Serializable: Beschreibt die Eigenschaft dass das Objekt in ein String umgewandelt werden kann

2.2 Dependencies definieren

Die dependencies, d.h. welche Framework verwendet wird. Normalerweise würde das bedeuten die jeweiligen `.jar` Files herunterzuladen und dem Build-Path hinzuzufügen, aber dank Maven kann man dies ganz einfach im `pom.xml` File definieren:

```
<dependencies>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
<groupId>org.hsqldb</groupId>
<artifactId>hsqldb</artifactId>
<scope>runtime</scope>
</dependency>


<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-tomcat</artifactId>
<scope>provided</scope>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>

<dependency>
<groupId>org.apache.tomcat.embed</groupId>
<artifactId>tomcat-embed-jasper</artifactId>
<scope>provided</scope>
</dependency>

<dependency>
<groupId>javax.servlet</groupId>
<artifactId>jstl</artifactId>
</dependency>

<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
</dependency>

</dependencies>
```

Zusätzlich wird noch das `springframework` plugin defniert:

```
<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>
```

2.3 Entities definieren

Mit `hsql` wird eine table durch die annotation `@Entitity` definiert. Es wird eine Klasse User erstellt, welche folgende Attribute besitzt:

- Long id
- String username
- String password
- String passwordConfirm
- Set<Role> roles

Danach werden Getter- und Settermethoden definiert. Zu beachten ist, dass id den eindeutigen Primary Key repräsentiert und somit bei `getId()` die Annotations `@Id` und `@GenerateValue(strategy = Genera` benötigt werden.

```
package com.hellokoding.auth.model;

import javax.persistence.*;
import java.util.Set;

@Entity
@Table(name = "user")
public class User {
private Long id;
private String username;
private String password;
private String passwordConfirm;
private Set<Role> roles;

@Id
@GeneratedValue(strategy = GenerationType.AUTO)
public Long getId() {
return id;
}

public void setId(Long id) {
this.id = id;
}

public String getUsername() {
return username;
}

public void setUsername(String username) {
this.username = username;
}

public String getPassword() {
return password;
}

public void setPassword(String password) {
this.password = password;
}

```

```
@Transient
public String getPasswordConfirm() {
return passwordConfirm;
}

public void setPasswordConfirm(String passwordConfirm) {
this.passwordConfirm = passwordConfirm;
}

@ManyToMany
@JoinTable(name = "user_role", joinColumns = @JoinColumn(name = "user_id"), inverseJoinColumns =
    @JoinColumn(name = "role_id"))
public Set<Role> getRoles() {
return roles;
}

public void setRoles(Set<Role> roles) {
this.roles = roles;
}
}
```

Abbildungsverzeichnis