JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

# Protokoll GK9.3

# Labor 9. November 2017

Systemtechnik Labor
5BHIT 2017/18

Maximilian Müller

**Version 1.0**
**Begonnen am 19. Oktober 2017**
**Beendet am 9. November 2017**

**Note:**
**Betreuer: BORM**

# 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).

## 1.4 Bewertung

- Gruppengrösse: 1 Person
- Anforderungen ”überwiegend erfüllt”
  - Dokumentation und Beschreibung der angewendeten Schnittstelle

 

  - Aufsetzen einer Webservice-Schnittstelle
  - Registrierung von Benutzern mit entsprechender Persistierung

- Anforderungen "zur Gänze erfüllt"
  - Login und Rückgabe einer Willkommensnachricht
  - Überprüfung der funktionalen Anforderungen mittels Regressionstests

## 1.5 Quellen

"Android Restful Webservice Tutorial – Introduction to RESTful webservice – Part 1"; Posted By Android Guru on May 1, 2014;
"REST with Java (JAX-RS) using Jersey - Tutorial"; Lars Vogel; Version 2.7; 27.09.2017;
"Creating a 'hello world' RESTful web service with Spring."; Spring examples;
"Registration and Login Example with Spring Boot, Spring Security, Spring Data JPA, and HSQL"; Giau Ngo; 5.7.2016;
"Django REST framework"; Tom Christie;
"Eve. The Simple Way to REST"; Nicola Iarocci;
"Heroku makes it easy to deploy and scale Java apps in the cloud";

# 2 Code

Der Code ist bei https://github.com/mmueller-tgm/syt5 bereitgestellt.

# 3 Einstieg

Am Anfang wollte ich, durch verschiedene Tutorials selber in Java/Node den REST Service implementieren aber nach mehreren Problemen mit Packages wurde ich auf ein Tutorial[1] hingewiesen, welches mehr oder weniger die Aufgabe abbildet.

Voraussetzung ist JDK1.7 oder später und Maven 3 zum Builden und Deployen. Den Sourcecode erhält man mit GIT „git clone git@github.com:hellokoding/ registration-login-spring-hsql.git“.

Per default ist „/login“ schon als Endpunkt konfiguriert ist aber „/register“ nicht. Um „/registration“ auf „/register“ umzuleiten habe ich folgendes Command verwendet:

```
1    find . -type f -exec sed -i 's/registration/register/g' {} +
```

Es sucht in jedem File in dem Ordner und tauscht „registration“ mit „register“ aus.

# 4 Ausführen des Codes

Deployed wird mit „mvn clean spring-boot:run“, Test cases mit „python3.5 tests.py“. Der Service rennt auf „localhost:8080“.

# 5 Erklärung der Codebase

## 5.1 pom.xml

Da die Buildumgebung Maven ist, werden in „pom.xml“ die benötigten Packages angegeben.

## 5.2 src/main/java/com/hellokoding/auth/model/User.java

Hier wird die Struktur der User Tabelle bzw. das User Objekt definiert.

## 5.3 src/main/java/com/hellokoding/auth/model/Role.java

Hier wird die Struktur der Role Tabelle bzw. das Role Objekt definiert.

## 5.4 Spring Security

In „src/main/java/com/hellokoding/auth/service/SecurityServiceImpl.java“ werden eingeloggte User und neu erstellte User authentifiziert.

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

In „src/main/java/com/hellokoding/auth/service/UserServiceImpl.java“ wird das Service zum registrieren von Usern bereitgestellt.

## 5.5 Spring Validator

In „src/main/java/com/hellokoding/auth/validator/UserValidator.java“ werden angegebene Userdaten Validiert, die bei dem Registrieren angegeben werden.

## 5.6 Controller (Endpunkte)

„src/main/java/com/hellokoding/auth/web/UserController.java“ handeld alle POST und GET Requests.

```
1  package com.hellokoding.auth.web;
2
3  import com.hellokoding.auth.model.User;
4  import com.hellokoding.auth.service.SecurityService;
5  import com.hellokoding.auth.service.UserService;
6  import com.hellokoding.auth.validator.UserValidator;
7  import org.springframework.beans.factory.annotation.Autowired;
8  import org.springframework.stereotype.Controller;
9  import org.springframework.ui.Model;
10 import org.springframework.validation.BindingResult;
11 import org.springframework.web.bind.annotation.ModelAttribute;
12 import org.springframework.web.bind.annotation.RequestMapping;
13 import org.springframework.web.bind.annotation.RequestMethod;
14
15 @Controller
16 public class UserController {
17     @Autowired
18     private UserService userService;
19
20     @Autowired
21     private SecurityService securityService;
22
23     @Autowired
24     private UserValidator userValidator;
25
26     @RequestMapping(value = "/register", method = RequestMethod.GET)
27     public String register(Model model) {
28         model.addAttribute("userForm", new User());
29
30         return "register";
31     }
32
33     @RequestMapping(value = "/register", method = RequestMethod.POST)
34     public String register(@ModelAttribute("userForm") User userForm, BindingResult bindingResult, Model
           model) {
35         userValidator.validate(userForm, bindingResult);
36
37         if (bindingResult.hasErrors()) {
38             return "register";
39         }
40
41         userService.save(userForm);
42
43         securityService.autologin(userForm.getUsername(), userForm.getPasswordConfirm());
44
45         return "redirect:/welcome";
46     }
47
48     @RequestMapping(value = "/login", method = RequestMethod.GET)
49     public String login(Model model, String error, String logout) {
50         if (error != null)
```

```
51              model.addAttribute("error", "Your username and password is invalid.");

53          if (logout != null)
                model.addAttribute("message", "You have been logged out successfully.");
55
            return "login";
57      }

59      @RequestMapping(value = {"/", "/welcome"}, method = RequestMethod.GET)
        public String welcome(Model model) {
61          return "welcome";
        }
63  }
```

# 6 Unittests

Unittests wurden mit den Python Bibliotheken „unittests“ und „requests“ durchgeführt.

### 6.0.1 Ansprechen der REST Schnittstelle

Um mit dem Service zu kommunizieren müssen, per Definition, HTTP Requests gesendet werden. Dazu eignet sich besonders die Bibliothek „requests“.

### 6.0.2 Registrieren eines Benutzers

Um einen Benutzer zu registrieren muss zuerst ein Request zu /register gemacht werden, um einen Sessionkey und einen csrf-Key zu bekommen.

```
1  x = requests.get("%sregister"%self.server)
   cookies = x.cookies
3  x = BeautifulSoup(x.text, 'html.parser')
   x = x.find_all("input")
5  payload={}
   for y in x:
7      if y["name"] == "_csrf":
           payload[y["name"]] = y["value"]
```

Mit diesen Keys kann man dann mit POST einen User erstellen.

```
   payload["username"] = "test_%s"%''.join(random.sample('0123456789',5))
2  payload["password"] = '123456789'
   payload["passwordConfirm"] = '123456789'
4  response = requests.post("%sregister"%self.server, payload, cookies=id)
```

### 6.0.3 test_create_user

Dieser Testfall erstellt einen Benutzer und checkt ob dieser eingeloggt wird.

```
   user="test_%s"%''.join(random.sample('0123456789',5))
2  response = self.create_user(user=user)
   y = BeautifulSoup(response.text, 'html.parser').find_all("form")
4  y2 = BeautifulSoup(response.text, 'html.parser').find_all("span")

6  for z in y:
       if z['id'] == 'logoutForm':
8          self.assertTrue(True, "Logged in successfully")
           return
10 self.assertTrue(False)
```

### 6.0.4 test_create_user_short_pw

Dieser Testfall erstellt einen Benutzer mit zu kurzem Passwort und checkt, ob dieser eingeloggt wird, wenn ja, dann failt der Test.

```
user="test_%s"%''.join(random.sample('0123456789',5))
response = self.create_user(user=user, passwd='1234567')
y = BeautifulSoup(response.text, 'html.parser').find_all("form")
y2 = BeautifulSoup(response.text, 'html.parser').find_all("span")

for z in y:
    if z['id'] == 'logoutForm':
        self.assertTrue(False, "Should not have been able to login.")
        return

for z in y2:
    if z['id'] == 'passeord.errors':
        self.assertTrue(True)
        return
    if z['id'] == 'username.errors':
        self.assertTrue(False, "Should not have username errors:")
```

### 6.0.5 test_create_user_short_un

Dieser Testfall erstellt einen Benutzer mit zu kurzem Usernamen und checkt, ob dieser eingeloggt wird, wenn Ja, dann failt der Test.

```
user="test_"
response = self.create_user(user=user, passwd='12345678')
y = BeautifulSoup(response.text, 'html.parser').find_all("form")
y2 = BeautifulSoup(response.text, 'html.parser').find_all("span")

for z in y:
    if z['id'] == 'logoutForm':
        self.assertTrue(False, "Should not have been able to login.")
        return

for z in y2:
    if z['id'] == 'passeord.errors':
        self.assertTrue(False, "Password should be long enough.")
        return
    if z['id'] == 'username.errors':
        if z.next == "Please use between 6 and 32 characters.":
            self.assertTrue(True, "Should not have username errors:")
        else:
            self.assertTrue(False, "UN should be unique")
```

### 6.0.6 test_login

Dieser Testfall erstellt einen Benutzer. In einer anderen Session loggt er sich dann ein. Bei erfolgreichem einloggen ist auch der Testfall erfolgreich.

```
user = "test_%s" % ''.join(random.sample('0123456789', 5))
passwd = "123456789"
self.create_user(user, passwd=passwd)
x = requests.get("%slogin"%self.server)
id = x.cookies
x = BeautifulSoup(x.text, 'html.parser')
x = x.find_all("input")
payload={"username":user, "password":passwd}
for y in x:
    if y["name"] == "_csrf":
```

```
        payload[y["name"]] = y["value"]

response = requests.post("%slogin"%self.server, payload, cookies=id)
response = BeautifulSoup(response.text, 'html.parser')
y = BeautifulSoup(response.text, 'html.parser').find_all("form")
for z in y:
    if z['id'] == 'logoutForm':
        self.assertTrue(True, "Logged in successfully")
        return
    self.assertTrue(False)
```

### 6.0.7 test_login_false

Dieser Testfall erstellt einen Benutzer. In einer anderen Session loggt er sich dann mit falschem Passwort ein. Bei erfolgreichem Einloggen ist der Testfall nicht erfolgreich.

```
user = "test_%s" % ''.join(random.sample('0123456789', 5))
passwd = "123456789"
self.create_user(user, passwd=passwd)
x = requests.get("%slogin"%self.server)
id = x.cookies
x = BeautifulSoup(x.text, 'html.parser')
x = x.find_all("input")
payload={"username":user, "password":passwd+"x"}
for y in x:
    if y["name"] == "_csrf":
        payload[y["name"]] = y["value"]

response = requests.post("%slogin"%self.server, payload, cookies=id)
response = BeautifulSoup(response.text, 'html.parser')
y = BeautifulSoup(response.text, 'html.parser').find_all("form")

for z in y:
    if z['id'] == 'logoutForm':
        self.assertTrue(False, "Used Wrong PW, should not have logged in")
        return
    self.assertTrue(True)
```