
Protokoll GK9.3

Labor 19. Oktober 2017

Systemtechnik Labor
5BHIT 2017/18

Maximilian Müller

Note:
Betreuer: BORM

Version 1.0
Begonnen am 12. Oktober 2017
Beendet am 19. Oktober 2017

Inhaltsverzeichnis

1	Einführung	1
1.1	Ziele	1
1.2	Voraussetzungen	1
1.3	Aufgabenstellung	1
1.4	Bewertung	1
1.5	Quellen	2
2	Einstieg	3
3	Erklärung der Codebase	3
3.1	pom.xml	3
3.2	src/main/java/com/hellokoding/auth/model/User.java	3
3.3	src/main/java/com/hellokoding/auth/model/Role.java	3
3.4	Spring Security	3
3.5	Spring Validator	3
3.6	Controller (Endpunkte)	4
4	Unittests	5
4.0.1	Ansprechen der REST Schnittstelle	5
4.0.2	Registrieren eines Benutzers	5
4.0.3	test_create_user	5
4.0.4	test_create_user_short_pw	5
4.0.5	test_create_user_short_un	6
4.0.6	test_login	6
4.0.7	test_login_false	7

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 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 (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 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 Tutorial¹ 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.

3 Erklärung der Codebase

3.1 pom.xml

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

3.2 src/main/java/com/hellokoding/auth/model/User.java

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

3.3 src/main/java/com/hellokoding/auth/model/Role.java

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

3.4 Spring Security

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

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

3.5 Spring Validator

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

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

3.6 Controller (Endpunkte)

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

```
1 package com.hellokoding.auth.web;

3 import com.hellokoding.auth.model.User;
import com.hellokoding.auth.service.SecurityService;
5 import com.hellokoding.auth.service.UserService;
import com.hellokoding.auth.validator.UserValidator;
7 import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
9 import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
11 import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
13 import org.springframework.web.bind.annotation.RequestMethod;

15 @Controller
public class UserController {
17     @Autowired
    private UserService userService;

19     @Autowired
21     private SecurityService securityService;

23     @Autowired
25     private UserValidator userValidator;

27     @RequestMapping(value = "/register", method = RequestMethod.GET)
    public String register(Model model) {
        model.addAttribute("userForm", new User());

29         return "register";
31     }

33     @RequestMapping(value = "/register", method = RequestMethod.POST)
    public String register(@ModelAttribute("userForm") User userForm, BindingResult bindingResult, Model
        model) {
35         userValidator.validate(userForm, bindingResult);

37         if (bindingResult.hasErrors()) {
39             return "register";
41         }

43         userService.save(userForm);

45         securityService.autologin(userForm.getUsername(), userForm.getPasswordConfirm());

47         return "redirect:/welcome";
49     }

51     @RequestMapping(value = "/login", method = RequestMethod.GET)
    public String login(Model model, String error, String logout) {
53         if (error != null)
55             model.addAttribute("error", "Your username and password is invalid.");

57         if (logout != null)
59             model.addAttribute("message", "You have been logged out successfully.");

61         return "login";
63     }

65     @RequestMapping(value = {"/", "/welcome"}, method = RequestMethod.GET)
    public String welcome(Model model) {
67         return "welcome";
69     }
}
```

4 Unittests

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

4.0.1 Ansprechen der REST Schnittstelle

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

4.0.2 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)
```

4.0.3 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)
```

4.0.4 test_create_user_short_pw

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

```
user="test_%s"%'.join(random.sample('0123456789',5))
2 response = self.create_user(user=user, passwd='1234567')
  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':
        self.assertTrue(False, "Should not have been able to login.")
        return
10
12 for z in y2:
    if z['id'] == 'passeord.errors':
        self.assertTrue(True)
        return
14
16 if z['id'] == 'username.errors':
    self.assertTrue(False, "Should not have username errors:")

```

4.0.5 test_create_user_short_un

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

```

user="test_"
2 response = self.create_user(user=user, passwd='12345678')
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':
        self.assertTrue(False, "Should not have been able to login.")
        return
10
12 for z in y2:
    if z['id'] == 'passeord.errors':
        self.assertTrue(False, "Password should be long enough.")
        return
14
16 if z['id'] == 'username.errors':
    if z.next == "Please use between 6 and 32 characters.":
        self.assertTrue(True, "Should not have username errors:")
18
    else:
        self.assertTrue(False, "UN should be unique")

```

4.0.6 test_login

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

```

1 user = "test_%s" % ''.join(random.sample('0123456789', 5))
passwd = "123456789"
3 self.create_user(user, passwd=passwd)
x = requests.get("%slogin"%self.server)
5 id = x.cookies
x = BeautifulSoup(x.text, 'html.parser')
7 x = x.find_all("input")
payload={"username":user, "password":passwd}
9 for y in x:
    if y["name"] == "_csrf":
        payload[y["name"]] = y["value"]
11

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

```


4.0.7 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.

```
1 user = "test_%s" % ''.join(random.sample('0123456789', 5))
2 passwd = "123456789"
3 self.create_user(user, passwd=passwd)
4 x = requests.get("%slogin"%self.server)
5 id = x.cookies
6 x = BeautifulSoup(x.text, 'html.parser')
7 x = x.find_all("input")
8 payload={"username":user, "password":passwd+"x"}
9 for y in x:
10     if y["name"] == "_csrf":
11         payload[y["name"]] = y["value"]
12
13 response = requests.post("%slogin"%self.server, payload, cookies=id)
14 response = BeautifulSoup(response.text, 'html.parser')
15 y = BeautifulSoup(response.text, 'html.parser').find_all("form")
16
17 for z in y:
18     if z['id'] == 'logoutForm':
19         self.assertTrue(False, "Used Wrong PW, should not have logged in")
20         return
21 self.assertTrue(True)
```