Laborprotokoll

Distributed Computing GK9.3

Systemtechnik Labor
5BHIT 2017/18

Matthias Weiss

Version 0.1
Note:
Begonnen am 21. Januar 2018
Betreuer: M. Borko
Beendet am 24. Januar 2018

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 ßur 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; online: http://programmerguru.com/android-tutorial/android-restful-webservice-tutorial-part-1/
„REST with Java (JAX-RS) using Jersey - Tutorial“; Lars Vogel; Version 2.7; 27.09.2017; online: http://www.vogella.com/tutorials/REST/article.html
„Django REST framework“; Tom Christie; online: http://www.django-rest-framework.org/
„Eve. The Simple Way to REST“; Nicola Iarocci; online: http://python-eve.org/
„Heroku makes it easy to deploy and scale Java apps in the cloud“; online: https://www.heroku.com/

2 Ergebnisse

Die Aufgabe wurde in JavaEE mithilfe von **Spring Boot, Spring Security, Spring Data JPA** und **HSQL** realisiert. Dabei wurde mit dem folgenden Tutorial gearbeitet.

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

Dabei soll ein RESTful Webservice, bei welchem man sich registrieren und einloggen kann, als Ergebnis rauskommen.

2.1 Implementierung

2.1.1 Vorraussetzungen

- Zum ausführen des Projekts muss ein JDK 1.7 oder 1.8 (JDK 1.9 ist zu neu; siehe Probleme) und Maven 3 (oder neuer) installiert sein

2.1.2 Implementierung

Projekt Dependecies

Die verwendeten Dependencies (Frameworks) werden im **pom.xml** File definiert und Maven läd diese dann beim ersten ausführen herunter.

JPA Entities

Die @Entity Annotation repräsentiert eine Tabelle in der Datenbank. Dann wird als erstes eine Klasse erstellt, welche die Attribute für einen User enthält. Außerdem werden auch noch einige Setter und Getter implementiert.

```
1  @Entity
   @Table(name = "user")
3  public class User {
       private Long id;
5      private String username;
       private String password;
7      private String passwordConfirm;
       private Set<Role> roles;
9
       @Id
11     @GeneratedValue(strategy = GenerationType.AUTO)
       public Long getId() {
13         return id;
       }
15
       public void setId(Long id) {
17         this.id = id;
       }
19
...
```

Dadurch, dass die ID den Primary Key darstellt, werden die Annotationen @Id und @GenerateValue bei der Methode **getID()** benötigt.

JPA Repositories

Es werden 2 Repositories definiert, für User und Roles. Hier ist zu erwähnen, dass die gängigen Repository Funktionen **findOne(), findAll(), save(), ...** nicht extra impelemtiert werden müssen, da vom JPARepository geerbt wird und dieses diese Funktionen schon implementiert hat.

```
public interface UserRepository extends JpaRepository<User, Long> {
    User findByUsername(String username);
}
```

Spring Security's UserDetailService

Um Login bzw. Authentifizierung mithilfe von Spring Security zu implementieren muss das Interface **org.springframework.security.core.userdetails.UserDetailsService** implementiert werden.

In dieser Implementierung befindet sich eine Funktion, welche einen User holt und diesem eine Authority zuweist. Zurückgegeben wird ein User-Object mit **Username, Passwort** und **Authorities**.

```
@Service
public class UserDetailsServiceImpl implements UserDetailsService{
    @Autowired
    private UserRepository userRepository;

    @Override
    @Transactional(readOnly = true)
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByUsername(username);

        Set<GrantedAuthority> grantedAuthorities = new HashSet<>();
        for (Role role : user.getRoles()){
            grantedAuthorities.add(new SimpleGrantedAuthority(role.getName()));
        }

        return new org.springframework.security.core.userdetails.User(user.getUsername(), user.
            getPassword(), grantedAuthorities);
    }
}
```

Security Service

Als nächstes wird der Security Service implementiert, dieser stellt sicher, dass man sich automatisch einloggen lassen kann nachdem man sich registriert hat und gibt auch den Namen des zurzeit eingeloggten Users wieder.

Diese Funktion findet heraus ob ein User eingeloggt ist und welchen Namen dieser hat, falls ein User eingeloggt ist, dann wird dessen Name zurückgegeben, wenn keiner eingeloggt ist wird null returned.

```
@Override
public String findLoggedInUsername() {
    Object userDetails = SecurityContextHolder.getContext().getAuthentication().getDetails();
    if (userDetails instanceof UserDetails) {
        return ((UserDetails)userDetails).getUsername();
    }

    return null;
}
```

Mit der nächsten Funktion wird überprüft, ob das automatische Einloggen nach der Registration funktioniert hat.

```
@Override
public void autologin(String username, String password) {
    UserDetails userDetails = userDetailsService.loadUserByUsername(username);
    UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new
        UsernamePasswordAuthenticationToken(userDetails, password, userDetails.getAuthorities());

    authenticationManager.authenticate(usernamePasswordAuthenticationToken);

    if (usernamePasswordAuthenticationToken.isAuthenticated()) {
        SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);
        logger.debug(String.format("Auto login %s successfully!", username));
    }
}
```

User Service

Hier wird eine wichtige Funktion implementiert, welche bei einer Registrierung das Passwort verschlüsselt und die Rolle setzt.

```
@Override
public void save(User user) {
    user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
    user.setRoles(new HashSet<>(roleRepository.findAll()));
    userRepository.save(user);
}
```

Spring Validator

Diese Klasse überprüft, dass bei der Registrierung auch die vorgeschriebenen Bedingungen erfüllt werden. Diese Bedingungen sind zum Beispiel **Länge des Benutzernamen, Länge des Passworts, Passwortübereinstimmung, ...** und diese werden in der validation.properties Datei gespeichert.

```
@Override
public void validate(Object o, Errors errors) {
    User user = (User) o;

    ValidationUtils.rejectIfEmptyOrWhitespace(errors, "username", "NotEmpty");
    if (user.getUsername().length() < 6 || user.getUsername().length() > 32) {
        errors.rejectValue("username", "Size.userForm.username");
    }
    if (userService.findByUsername(user.getUsername()) != null) {
        errors.rejectValue("username", "Duplicate.userForm.username");
    }

    ValidationUtils.rejectIfEmptyOrWhitespace(errors, "password", "NotEmpty");
    if (user.getPassword().length() < 8 || user.getPassword().length() > 32) {
        errors.rejectValue("password", "Size.userForm.password");
    }

    if (!user.getPasswordConfirm().equals(user.getPassword())) {
        errors.rejectValue("passwordConfirm", "Diff.userForm.passwordConfirm");
    }
}
```

Controller

Der Controller kümmert sich um das „mappen“ der Links (**/registration, /login**).

Bei der Registrierungsseite wird überprüft ob das Formular korrekt ausgefüllt wurde, wenn ja dann wird man eingeloggt und wenn nicht, dann wird die Seite neu geladen.

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

    if (bindingResult.hasErrors()) {
        return "registration";
    }

    userService.save(userForm);

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

    return "redirect:/welcome";
}
```

Beim aufrufen der Loginseite wird überprüft, ob der User noch eingeloggt ist oder ob andere Fehler vorliegen, wenn keine vorliegen, dann wird die Loginseite aufgerufen.

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

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

    return "login";
}
```

Web Security Configuration

Diese Klasse kümmert sich um die Seitenaufrufe und besitzt auch noch eine Methode zum verschlüsseln für zum Beispiel das Passwort eines Users.

```
@Bean
public BCryptPasswordEncoder bCryptPasswordEncoder() {
    return new BCryptPasswordEncoder();
}

@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
        .antMatchers("/resources/**", "/registration").permitAll()
        .anyRequest().authenticated()
        .and()
    .formLogin()
        .loginPage("/login")
        .permitAll()
        .and()
    .logout()
        .permitAll();
}
```

2.1.3 Ausführen

Ausgeführt wird das Projekt mit **mvn clean spring-boot:run** und dann muss man im Browser auf **localhost:8080** gehen.

2.1.4 Probleme

Anfangs gab es ein paar „Startschwierigkeiten“, da die verwendete Java Version nicht mehr kompatibel mit dem ausgewählten Tutorial war und deshalb lange Zeit mit einem Versionierungsfehler gekämpft wurde. Es wurde versuch das Tutorial mit der **Java Version 9** kompatibel zu machen, jedoch ist dies nach langem bemühen nicht gelungen und als Lösung ist auf eine ältere Java Version umgestiegen worden.

Literatur

Tabellenverzeichnis

Listings

Abbildungsverzeichnis