JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

SYT

Distributed Computing GK9.3

SYT
5BHIT 2017/18

Zeljko Colakovic

Version 1.0
Begonnen am 30:11.2017
Beendet am 25.01.2018

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

2 Quellen

Ändroid 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/

"Registration and Login Example with Spring Boot, Spring Security, Spring Data JPA, and HSQL"; Giau Ngo; 5.7.2016; online: https://hellokoding.com/registration-and-login-example-with-spring-security-spring-boot-spring-data-jpa-hsql-jsp/

"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

"Creating a 'hello world' RESTful web service with Spring."; Spring examples; online: https://github.com/spring-guides/gs-rest-service

"Django REST framework"; Tom Christie; online: http://www.django-rest-framework.org/

Ëve. 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

3 Ergebnisse

Die Abgabe befindet sich auf *github* unter folgendem Link:

3.1 Maven

Maven ist ein Buildtool und basiert auf POM - *ProjekctObject Model*.

3.1.1 POM

Die pom.xml in einem Maven Projekt beinhaltet alle Informationen die Maven benötigt um das Projekt zu builden. Wenn man einen Task (Goal) mittels Maven ausführt, schaut diese in der pom.xml nach allen nötigen Informationen die es benötigt um das Ziel zu erreichten, und führt dann das Goal aus. Typische Dinge die festgelegt werden:

- Project Dependencies
- Plugins
- Goals
- Build Proles

3.1.2 Features

Die wohl wichtigsten Features von Maven sind folgende:

- Simple project setup
- Dependency management
- Release Management (z.B. git)

3.2 Spring

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 gemapped werden, welche wiederum beispielsweise eine .jsp page aufrufen.

3.2.1 Funktionsweise

In Spring werden statt echten Instantiierungen so gennante Beans verwendet. Wenn ein Bean erstelltwird,handelt es sich eigentlich nur um ein Vorgehen um Instanzen der zu erstellen Spring übernimmt dabei die Aufgaben der Instantiierung, Initialisierung und des injecten an den richtigen Stellen zur Laufzeit.

3.3 Beans

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

- Alle Attribute sind private (nur Getter/Setter)
- Ein public Konstruktor ohne Parameter
- Muss Serializable implementieren

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

3.4 Annotationen

- **@RequestMapping**

  Ist die eigentliche Verknüpfung von Methoden und URLs. Hier wird auch festgelegt ob HTTP GET oder POST verwendet werden soll.

- **@Autowired**

  Heißt, dass Spring für die Erstellung zuständig ist. - Ermöglicht das skippen von Konfiguration welche wo anders inizialisiert wird. Bei dieser Annotation findet eine Spring initialisierung statt.

- **@Service**

  WennUserDetailsServiceImpgestartetwird,wirddasAttributuserRepositoryvomTypUserRepository deniert, und mit der @Autowired Annotation versehen. Dies bedeutet, dass nach der Bean UserRepository gesucht wird, und diese dann anschließend in dieses Attribut ”injected” wird. Ich habe noch immer nicht ganz verstanden was ”injecten” bedeutet in dem Kontext.

- **@Configuration**

  Verweißt darauf, dass in der Klasse eine oder mehrere Beans vorhanden ist.

- **@EnableWebSecurity**

  Rolle eines Makers. Es erlaubt Spring zu finden und wendet es automatisch auf den globalen WebSecurity.

- **@Entity**

  Eine DB-Tabelle wird hiermit deniert

- **@Table**

  Bestimmt den namen der Tabelle in der DB (default ist Name der Klasse)

- **@Id**

  Dient zur festlegung des PK (Primary Key)

- **@GeneratedValue** JPA, dass sie sich selbst um die festlegung der ID kümmern muss

- **@Transient**

  Dient um ein Feld als nicht persistent zu setzten (Runtime-Use)

- **@ManyToMany**

  Viele- zu viele Beziehung zwischen zwei Entitäten @JoinTable @JoinColumn deniert das die Attribute welche die Fremdschlüssel beinhalten

- **@mappedBy**

  Ist das Gegenteil von @JoinColumn

3.5 Implementation und Aufbau

Folgendes Beispiel wurde als Basis genutzt: *https://hellokoding.com/registration-and-login-example-with-spring-security-spring-boot-spring-data-jpa-hsql-jsp/*

3.5.1 Projektstruktur

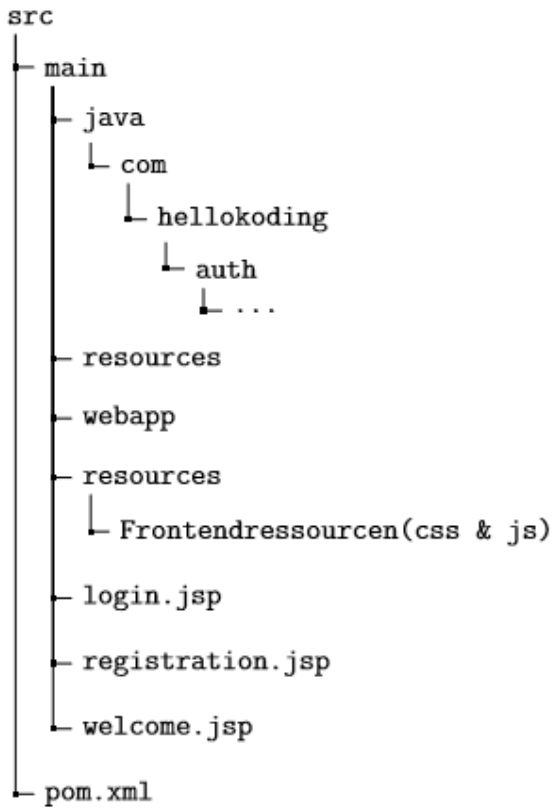

Abbildung 1: Aufbau - Projektstruktur

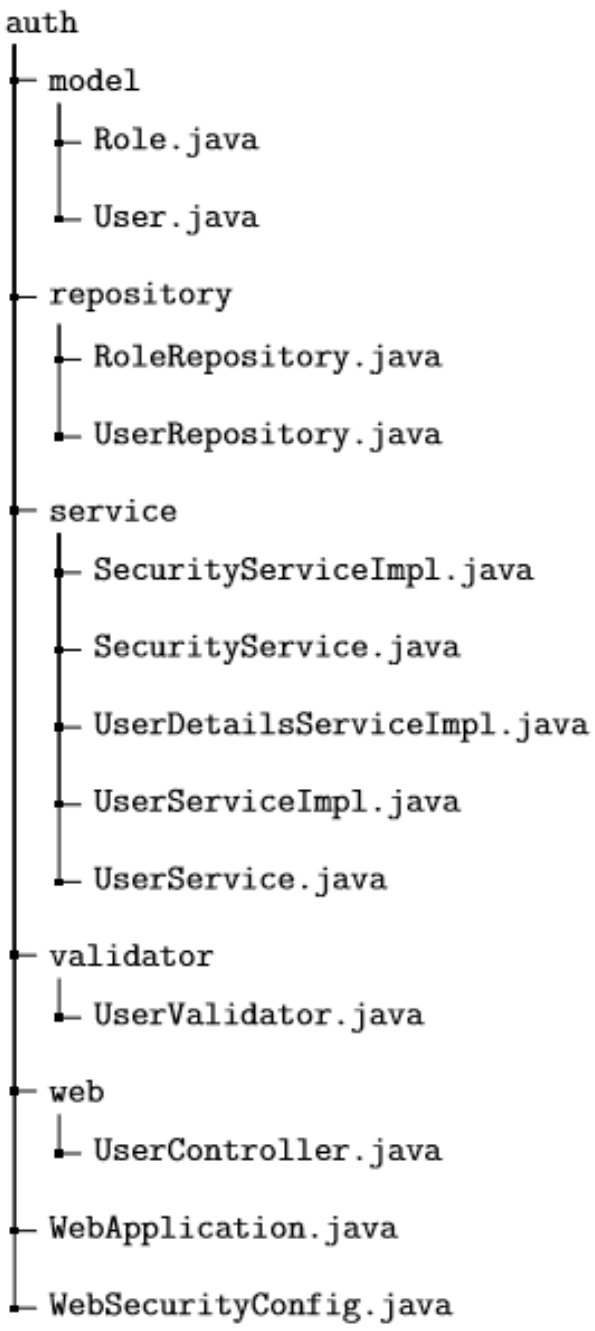

Abbildung 2: Aufbau - Klassenstruktur

3.5.2 Lauffähigkeit

Das Projekt wurde zuerst aus dem Repository geclont. Darauf hin mittels -> *File* -> *Import* -> *Exisiting Maven Projekt*, in Eclipse (Oxygen) importieren.

In der Running Config (*RunAS* -> *maven build* als Goal *spring-boot:run* setzen.

3.5.3 Dependencies

Die Abhängigkeiten für das Projekt (Maven beschrieben) sind in der pom.xml (Project Object Model) eingespeichert.

```
<?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>
    <artifactId>auth</artifactId>
    <name>auth</name>
    <description>auth</description>
    <packaging>war</packaging>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.3.5.RELEASE</version>
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <java.version>1.7</java.version>
    </properties>

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

        <dependency>
```

```
                <groupId>javax.servlet</groupId>
                <artifactId>jstl</artifactId>
            </dependency>
        </dependencies>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
            </plugins>
        </build>
    </project>
```

Listing 1: pom.xml

3.5.4 Model

Im Model befinden sich 2 JPA-Einträge (Java Persistence API) für dieses Projekt. In diesen Klassen werden Elemente deniert die in der Datenbank gespeichert werden sollen. Dies passiert mittels object-relational mapping. Was soviel bedeutet, wie das eine Datenbank wie normale Javaobjekte verwendet werden können (POJO). Es können auch hier schon Methoden wie Setter und Getter deniert werden, welche oft gebraucht. Bei dieser Klasse handelt es sich um die Denition für Benutzer des Projektes. Annotations (@) dienen zur denition von Eigenschaften.

```
@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;
    }
}
```

Listing 2: User JPA Klasse

3.5.5 Repository

Spring Data JPA besitzt eingebaute Funktionalitäten. in Repository werden einige davon implementiert um Datenbankaufgaben zu übernehmen (zB ndOne, ndAll, save)

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

Listing 3: User Repository Klasse

Im folgendem Beispiel wird ein ndByUsername aktiviert:

```
@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);
    }
}
```

Listing 4: UserDataService

UserDetailsServiceImpl.java dient als Klasse welche einem anhand des Usernames den gesamten Nutzer mit seinen Daten zurück gibt.

SecurityService.java wird verwendet um den aktuell angemeldeten Benutzer die Accounts zu übergeben User die sich gerade angelegt haben automatisch einzuloggen.

UserServiceImpl.java dient für die Erstellung von Benutzern. Dabei wird der anzulegende Benutzer als Parameter mitgegeben, aus diesem werden dann alle Attribute ausgelesen und zur DB hinzugefügt.

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

Listing 5: Benutzer Erstellung

3.5.6 Validator

Dient zur Überprüfung der Gültigkeit der Daten die der Benutzer angibt beim Erstellen seines Accountes.

```
@Component
public class UserValidator implements Validator {
    @Autowired
    private UserService userService;

    @Override
    public boolean supports(Class<?> aClass) {
        return User.class.equals(aClass);
    }

    @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");
        }
    }
}
```

Listing 6: UserValidator

Es werden die klassischen Überprüfungen wie Länge des Benutzernamens oder des Passwort vorgenommen. Es kontrolliert außerdem ob die Felder überhaupt befüllt wurden. Fehlermeldungen wurden in validation.properties festgelegt.

3.5.7 Web

Die Aufgabe des UserController ist es die Methoden mit URLs zu verknüpfen. So benötigt es eine bestimmte URL um eine bestimmte Methode aufzurufen.

```
@Controller
public class UserController {
    @Autowired
    private UserService userService;

    @Autowired
    private SecurityService securityService;

    @Autowired
    private UserValidator userValidator;

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

        return "registration";
    }

    @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";
    }

    @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";
    }

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

Listing 7: UserController

Listings

Abbildungsverzeichnis