Software Engineering 2 Lösungen der Übung

Dr. F.-K. Koschnick, Sybit GmbH

Aufgabe (Story): Projektseite für RetroWeb bauen Teil 2

Als User von RetroWeb möchte ich auf einer Webseite die Details eines Projekts mit den durchgeführten und geplanten Retros sehen, um mich über das Projekt, dessen Retros und über die Termine der geplanten Retros zu informieren.

Teil 1: Auf der Webseite soll nur der Projektname als Überschrift stehen, die Retros lassen wir in dieser Übung außen vor.

Verifiziere, dass

- sich die Projektdetailseite öffnet, wenn der User auf ein Projekt in der Home-Page klickt
- als Überschrift auf der Seite der Projektname steht
- es einen Link zurück auf die Home-Page gibt
- Die Seite den Header und Footer hat, der auf der Login-Seite schon besteht.
- nur ein eingeloggter User die Seite erreichen kann

Technische Anmerkungen:

Es ist dabei die bestehende Spring-Boot-Applikation retroweb weiterzuentwickeln. Header und Footer sollen eingebunden werden (von der Aufgabe Design der JSP der Login-Seite von retroweb). Der View der Projektseite soll project.jsp heißen, der zugehörige Controller soll ProjectController.java heißen. Es soll der bestehende *ProjectService* genutzt werden. Dieser bietet für diesen Zweck die Methode *public Project getByld(Long id)* an, wobei der Parameter id die id des Projekts ist, das der Service zurückliefern soll.

Prinzipielles Bild für Spring-Boot

Repository,

Holt mit Abfragen Datensätze
aus der DB oder schreibt
Datensätze in die DB. Jeder
Datensatz wird durch ein EntityObjekt repräsentiert. Eine
Menge von Datensätzen wird mit
einer Liste von Entity-Objekten
dargestellt.

DB

Entities

Service

Beinhaltet Business-Logik und nutzt das Repository, um Daten aus der Datenbank zu holen oder in die Datenbank zu schreiben.

model

Map<String, Object> model

Entities

Entity-Klasse ist zu einer
Tabelle in der DB gemappt.
Jede Property der EntityKlasse ist zu einer Spalte der
Tabelle gemappt.

View (JSP)

Die JSP (z.B. project.jsp) wird vom
Controller aufgerufen und es wird die
map mit dem Namen model
übergeben. Auf die map mit den
Daten vom Controller kann in der JSP
mit \${"key"} zugegriffen werden, z.B.:
\${"retros"}.

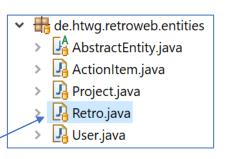
Entities

Alles was grün ist, gehört zum MVC-Pattern

Controller

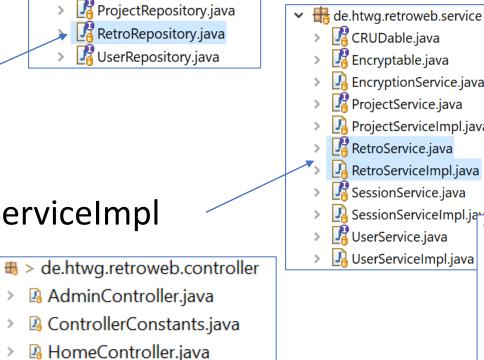
Bekommt den Request. Der Request wird einer Methode zugeordnet, so dass diese dann aufgerufen wird. In der Methode fordert der Controller beispielsweise vom Service Daten an (Aufruf einer Service-Methode, z.B. retros = retroService.findByProject(Project project)). Der Service liefert die Daten als

Entity-Objekte. Die Entity-Objekte werden in das model (Map mit Key-Value-Paaren) gegeben, z.B. model.put("retros", retros). Die Controllermethode ruft dann mit return "project" die JSP project.jsp auf und übergibt automatisch das model, also die map.



Reihenfolge des Codings

- 1. Entityklasse: Retro (grau)
- 2. Repository: RetroRepositiory (blau)
- 3. Service: RetroService und RetroServiceImpl (orange)
- 4. Controller: ProjectController (grün)
- 5. JSP: project.jsp (grün)



de.htwg.retroweb.repository

B > LoginController.java

ProjectController.java

UserController.java

Encryptable.java 🛂 EncryptionService.java ProjectService.java 🛂 ProjectServiceImpl.java RetroService.java RetroServiceImpl.java Kara Session Service. java SessionServiceImpl.ja~ ₩EB-INF **U**serService.java UserServiceImpl.java Y 🔊 jsp include admin.jsp home.jsp login.jsp newuser.jsp project.jsp 🔒 updateuser.jsp 🔒 user.jsp

```
import java.util.Date;
import javax.persistence.*;
import javax.validation.constraints.Size;
import org.apache.commons.lang3.builder.HashCodeBuilder;
@Entity
@Table(name = "retros")
public class Retro extends AbstractEntity {
    private static final long serialVersionUID = -8470435536181413339L;
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(name = "name", nullable = false)
    @Size(max = 255)
    private String name;
    @Column(name = "isactive", nullable = false)
    private boolean active;
    @Column(name = "scheduled", nullable = false)
    private Date scheduled;
    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "projectid", nullable = false)
    private Project project;
   public Project getProject() {
   public void setProject(Project project) {
   public boolean equals(Object o) {
      if( o == null) { return false; }
      if(o.getClass().equals(getClass())) {
         //do not use id (of retro), id is null when not saved, after saving id is a number, -> hashcode would be changed
         return r.getScheduled()== getScheduled() && r.getProject().getId() == getProject().getId();
      } else {
         return false;
   public int hashCode() { //always override hashCode, when overriding equals!!!
      return new HashCodeBuilder(19, 37). // two randomly chosen prime numbers
            append(getScheduled()).
            append(getProject().getId()).toHashCode();
```

equals-Methode:

(Hier scheduled +

project id)

Immer mit eindeutigen Properties arbeiten!!!

package de.htwg.retroweb.entities;

1. Retro (Entityklasse)

Bemerkungen:

Zusätzlich Getter und Setter, die hier nicht gezeigt sind.

In AbstractModel sind die Properties created und updated definiert und zu den Spalten gemappt. Da diese Properties überall auftauchen, wurden diese in die abstrakte Klasse ausgelagert.

```
CREATE TABLE retros (
   id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(255) NOT NULL,
   scheduled DATETIME NOT NULL,
   isactive BOOLEAN NOT NULL,
   projects_id BIGINT NOT NULL,
   foreign key (projects_id) references projects(id),
   created DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
   updated DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP);
```

2. RetroRepository (Repository)

```
package de.htwg.retroweb.repository;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.transaction.annotation.Transactional;
import de.htwg.retroweb.entities.Project;
import de.htwg.retroweb.entities.Retro;
import java.util.Date;
import java.util.List;
public interface RetroRepository extends JpaRepository<Retro, Long> {
    List<Retro> findByProject(Project project);
Hintergrund: Der Name der Suchmethode findByProject wird
automatisch interpretiert und es wird dann intern mit folgender
Query gesucht: Select * FROM retros WHERE projects id = ...
```

3. RetrosService (Service, Interface+Implementierung)

```
package de.htwg.retroweb.service;
                                                  Von CRUDable müsste man eigentlich für
                                                  diese Aufgabe nicht ableiten. Wenn man
import java.util.List;
                                                  das nicht macht, spart man sich, einige
                                                  Methoden bei der Implementierung zu
import de.htwg.retroweb.entity.Project;
                                                  schreiben.
import de.htwg.retroweb.entity.Retro;
public interface RetroService extends CRUDable<Retro> {
  public List<Retro> findByProject(Project project);
                                                          Implementierung der Methode
                                                          findByProject in RetroServiceImpl
@Override
  public List<Retro> findByProject(Project project) {
  LOG.debug("findByProject");
  return retroRepository.findByProject(project);
```

```
package de.htwg.retroweb.controller;
import ....;
@Controller
public class ProjectController {
  @Autowired
  RetroService retroService;
  @Autowired
  ProjectService projectService;
  @Autowired
  SessionService sessionService;
  @RequestMapping(value = "/project", method = RequestMethod.GET)
  public String showProject(@RequestParam long id, Map<String, Object> model, HttpServletRequest request) {
    HttpSession session = request.getSession();
    if (sessionService.isLoggedIn(session)) {
      try {
        Projects project = projectService.getById(id);
        model.put("project", project);
        List<Retro> retros = retroService.findByProject(project);
        model.put("retros", retros);
      } catch(ResourceNotFoundException e) { //Fehlerbehandlung }
      model.put("userName", sessionService.getUserName(session));
      return"project";
    } else {
     return "redirect:/login";
```

4. ProjectController (Controller)

model.put(String key, Object value)

Model wird mit Daten gefüllt. Es gibt also in der Model-Map jetzt die Keys: project, retros und userName, mit denen im View (JSP) auf die Daten zugegriffen werden kann.

5. project.jsp (View)

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html>
<html>
<c:import url="include/head.jsp"/>
<body>
<c:import url="include/header.jsp"/>
<nav class="navbar navbar-expand-lg navbar-light bg-light">
 Zugriff auf Model mit dem Key project
   <a class="nav-link" href="home">Home</a>
                                           und dem Projekt-Attribut name.
  </nav>
                                                      Zugriff auf Model mit dem Key retros
<div class="page">
                                                      auf die Liste der Retros.
 <h5>Project: ${project.name}</h5>
 NameScheduled
   <c:forEach items="${retros}" var = "retro">
     <tr><a href="retro?id=<c:out value = "${retro.id}"/>"><c:out value = "${retro.name}"/></a><c:out value = "${retro.scheduled}"/>
   </c:forEach>
 </div>
<c:import url="include/footer.jsp"/>
</body>
</html>
```