

# 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.

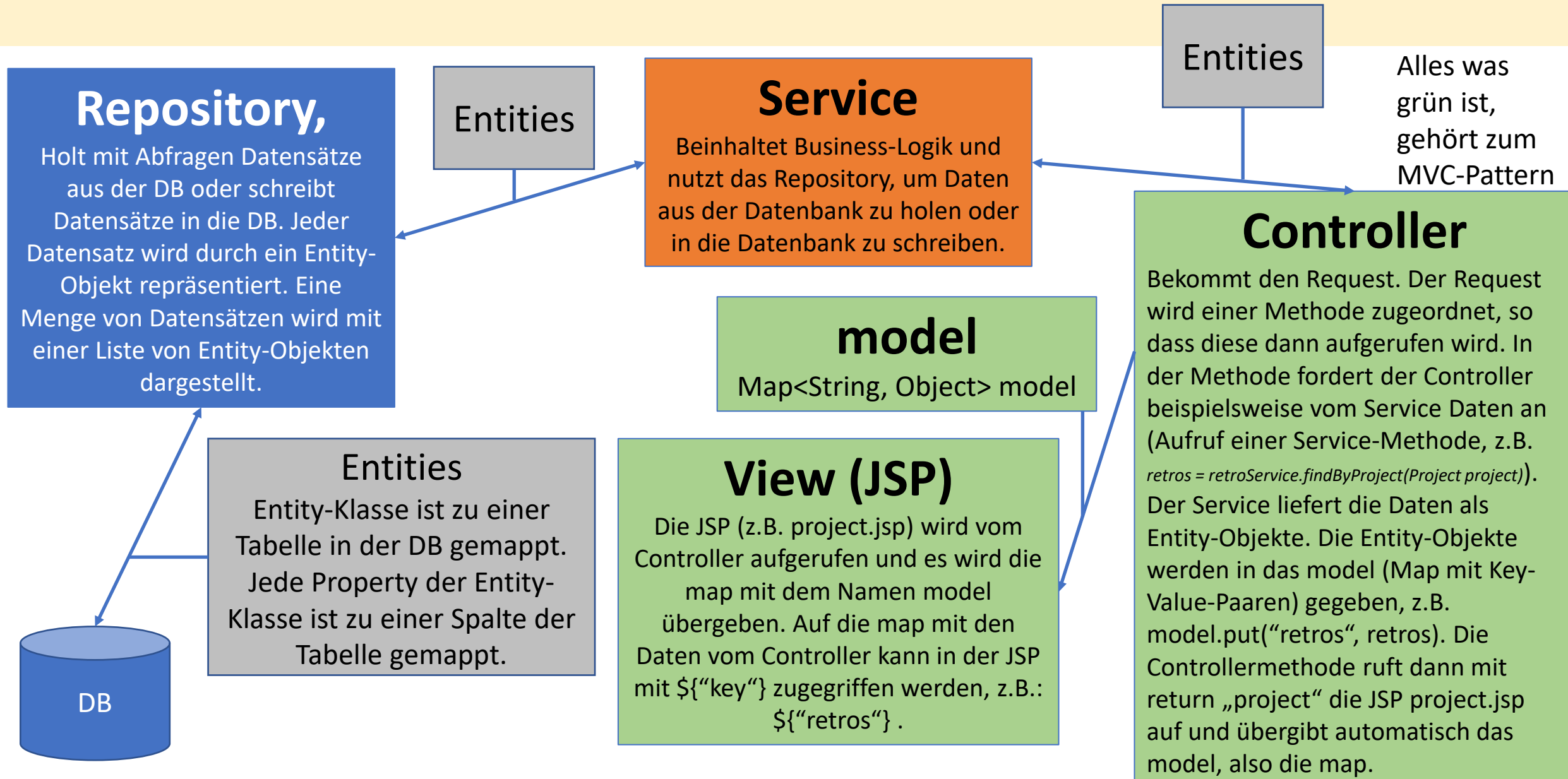
Verifiziere, dass

- 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

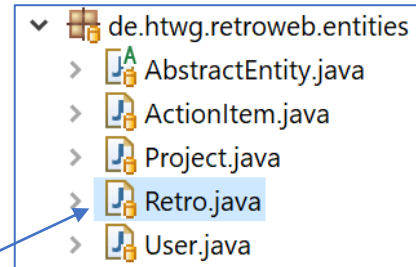
### Technische Anmerkungen:

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 getById(Long id)* an, wobei der Parameter `id` die `id` des Projekts ist, das der Service zurückliefern soll.

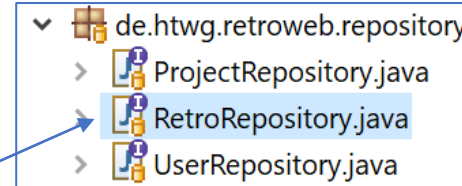
# Prinzipielles Bild für Spring-Boot



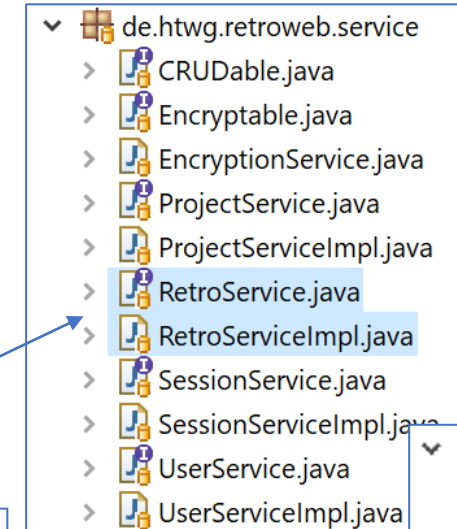
# Reihenfolge des Codings



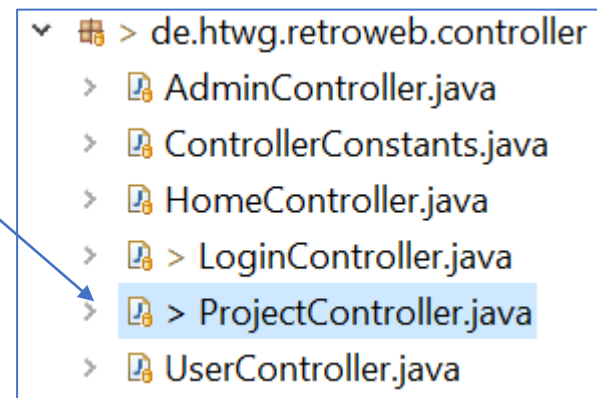
1. Entityklasse: Retro (grau)



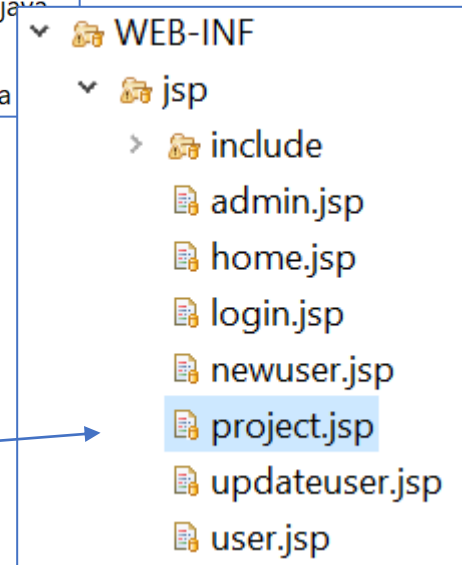
2. Repository: RetroRepository (blau)



3. Service: RetroService und RetroServiceImpl (orange)



4. Controller: ProjectController (grün)



5. JSP: project.jsp (grün)

```
package de.htwg.retroweb.entities;
```

```
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;
```

```
@Id
@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 = "project_id", nullable = false)
private Project project;
```

```
public Project getProject() {
    return project;
}
```

```
public void setProject(Project project) {
    this.project = project;
}
```

```
@Override
public boolean equals(Object o) {
    if (o == null) { return false; }
    if (o.getClass().equals(getClass())) {
        Retro r = (Retro) o;
        //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;
    }
}
```

```
@Override
public int hashCode() { //always override hashCode, when overriding equals!!!
    return new HashCodeBuilder(19, 37). // two randomly chosen prime numbers
        // if deriving: appendSuper(super.hashCode()).
        append(getScheduled()).
        append(getProject().getId()).toHashCode();
}
```

# 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.

equals-Methode:  
Immer mit eindeutigen  
Properties arbeiten!!!  
(Hier scheduled +  
project\_id)

```
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;

import java.util.List;


import de.htwg.retroweb.entity.Project;
import de.htwg.retroweb.entity.Retro;

public interface RetroService extends CRUDable<Retro> {

    public List<Retro> findByProject(Project project);

}
```

Von CRUDable müsste man eigentlich für diese Aufgabe nicht ableiten. Wenn man das nicht macht, spart man sich, einige Methoden bei der Implementierung zu schreiben.



---

```
@Override
public List<Retro> findByProject(Project project) {
    LOG.debug("findByProject");
    return retroRepository.findByProject(project);
}
```

Implementierung der Methode  
findByProject in RetroServiceImpl



## 4. ProjectController (Controller)

```
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";
```

```
            }
```

```
        }
```

**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">
  <ul class="navbar-nav mr-auto">
    <li class="nav-item active">
      <a class="nav-link" href="home">Home</a>
    </li>
  </ul>
</nav>

<div class="page">
  <h5>Project: ${project.name}</h5>
  <table class="table table-striped">
    <tr><th>Name</th><th>Scheduled</th></tr>
    <c:forEach items="${retros}" var="retro">
      <tr><td><a href="retro?id=<c:out value='${retro.id}'"/>"><c:out value='${retro.name}'"/></a></td><td><c:out value='${retro.scheduled}'/></td>
    </c:forEach>
  </table>
</div>
<c:import url="include/footer.jsp"/>
</body>
</html>
```

Zugriff auf Model mit dem Key *project* und dem Projekt-Attribut name.

Zugriff auf Model mit dem Key *retros* auf die Liste der Retros.