

# PSE Ausarbeitung

Phillip Wo  
Benjamin Moser  
Daniel Sommer

March 7, 2019

# Contents

|  |           |
|--|-----------|
| <b>I Pattern im Projekt</b>  | <b>4</b>  |
| <b>1 Layers Pattern</b>  | <b>4</b>  |
| 1.1 Erkläre die Funktionsweise + Skizze . . . . .  | 4         |
| 1.1.1 3 Schichten Architektur: . . . . .   | 4         |
| <b>2 Data Access Object (DAO) Pattern</b>  | <b>5</b>  |
| 2.1 Erkläre die Funktion + Skizze . . . . .  | 5         |
| 2.2 Nenne die Konsequenzen der Anwendung . . . . .   | 6         |
| <b>3 Service Layer Pattern (auch Session Fassade - in unserem Projekt im Domain Layer)</b> | <b>6</b>  |
| 3.1 Erkläre die Funktion + Skizze . . . . .  | 6         |
| 3.2 Nenne die Konsequenzen der Anwendung . . . . .   | 9         |
| <b>4 Model-View-Controller (MVC) Pattern</b>   | <b>9</b>  |
| 4.1 Erkläre die Funktion + Skizze . . . . .  | 9         |
| 4.1.1 Model . . . . .  | 10        |
| 4.1.2 View . . . . .   | 10        |
| 4.1.3 Controller . . . . .   | 10        |
| <b>5 Front Controller</b>  | <b>10</b> |
| 5.1 Erkläre die Funktion + Skizze . . . . .  | 10        |
| 5.2 Servlet . . . . .  | 10        |
| 5.2.1 Java Server Faces (JSF) . . . . .  | 11        |
| 5.3 Nenne die Konsequenzen der Anwendung . . . . .   | 13        |
| <b>6 View Helper (<i>/src/main/java/at/fhj/swd/psoe/web/*</i>)</b>                         | <b>13</b> |
| 6.1 Erkläre die Funktion + Skizze . . . . .  | 13        |
| 6.2 Nenne die Konsequenzen der Anwendung . . . . .   | 13        |
| <b>7 Dependency Injection (CDI-Framework -&gt; eingebunden im <i>./pom.xml</i>)</b>        | <b>14</b> |
| 7.1 Erkläre die Funktion + Skizze . . . . .  | 14        |
| 7.2 Nenne die Konsequenzen der Anwendung . . . . .   | 15        |
| <b>8 Data Transfer Object (DTO) Pattern</b>  | <b>15</b> |
| 8.1 Erkläre die Funktion (Skizze - ein Grund für DTO) . . . . .                            | 15        |
| 8.2 Konsequenzen der Anwendung . . . . .   | 17        |
| <b>II Exception Handling</b>   | <b>18</b> |
| <b>9 Checked und Runtime Exceptions in Java</b>  | <b>18</b> |
| 9.1 Checked Exceptions (z.B. SQL-Exception) . . . . .                                      | 18        |
| 9.2 Unchecked Exceptions (z.B. NullPointerException) . . . . .                             | 18        |

|   |           |
|---|-----------|
| <b>10 Best Practice Beispiele beim Einsatz von Exceptions</b>                   | <b>18</b> |
| <b>11 Exception Handling Anti Pattern</b>                                       | <b>19</b> |
| <b>12 Destructive Wrapping im Service Layer</b>                                 | <b>19</b> |
| <b>13 Exception Testing</b>   | <b>20</b> |
| <br>  |           |
| <b>III Allgemeines &amp; Config</b>   | <b>21</b> |
| <b>14 Logging</b>   | <b>21</b> |
| 14.0.1 Vorteile Logging mittels Framework (z.B.: log4j) . . . . .               | 21        |
| <b>15 Annotationen</b>  | <b>21</b> |
| 15.1 Annotationen - Details . . . . .   | 22        |
| <b>16 Konfigurationsdateien</b>   | <b>23</b> |
| 16.1 persistence.xml . . . . .  | 23        |
| 16.2 web.xml . . . . .  | 24        |
| 16.3 pom.xml . . . . .  | 26        |
| 16.3.1 Aufbau pom.xml . . . . .   | 26        |
| 16.4 standalone-psoe.xml . . . . .  | 26        |
| 16.5 log4j.properties . . . . .   | 28        |
| <b>17 Fehler im Projekt</b>   | <b>28</b> |
| 17.1 Return null . . . . .  | 28        |
| 17.2 Exception nicht gefangen . . . . .   | 28        |
| 17.3 Destructive Wrapping - Logging fehlt - Information geht verloren . . . . . | 28        |
| 17.4 Logger mit falschem Parameter . . . . .                                    | 29        |
| 17.5 Fehlendes Exception Handling . . . . .                                     | 29        |
| 17.6 Exception werfen und gleich wieder fangen . . . . .                        | 30        |
| <b>18 Tests</b>   | <b>31</b> |
| 18.1 Testpyramide . . . . .   | 32        |
| 18.2 Unit . . . . .   | 32        |
| 18.3 Integration Tests . . . . .  | 33        |
| 18.4 GUI-Test . . . . .   | 33        |
| 18.4.1 Page Object Pattern . . . . .  | 33        |
| 18.5 Page Object Pattern lt. Zahnlücke . . . . .                                | 34        |
| 18.6 Beispiel aus dem Projekt . . . . .   | 35        |
| 18.6.1 Integration GUI Test mit Selenium . . . . .                              | 35        |
| 18.6.2 Durch Selenium getestetes Page Objekt . . . . .                          | 36        |
| <b>19 Toni FRAAGNAA</b>   | <b>37</b> |
| <b>20 Frageart Prüfung</b>  | <b>39</b> |

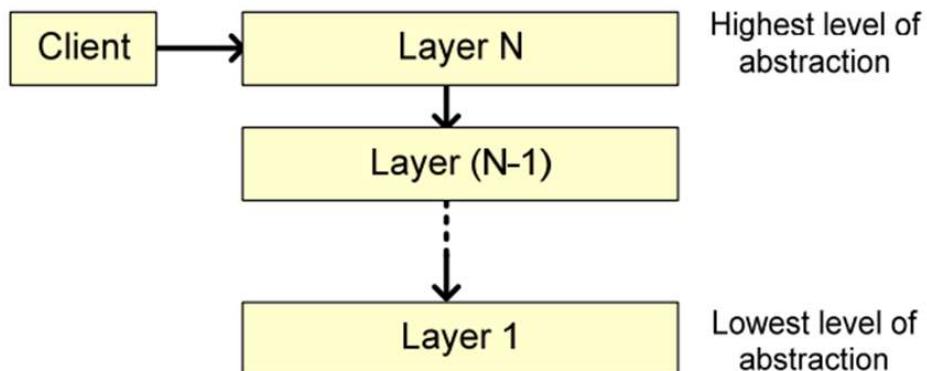
# Part I

## Pattern im Projekt

### 1 Layers Pattern

#### 1.1 Erkläre die Funktionsweise + Skizze

- Client schickt eine Anfrage an Layer N
- Layer N reicht da er nicht vollständig alleine beantworten kann, Anfragen an darunterliegenden Layer weiter
- Eine Anfrage kann bei Bedarf auch in mehrere Anfragen an darunterliegende Layer geteilt werden
- dies wird immer weiter fortgesetzt bis Layer 1 erreicht ist
- dabei gehen Abhängigkeiten nur von oben nach unten

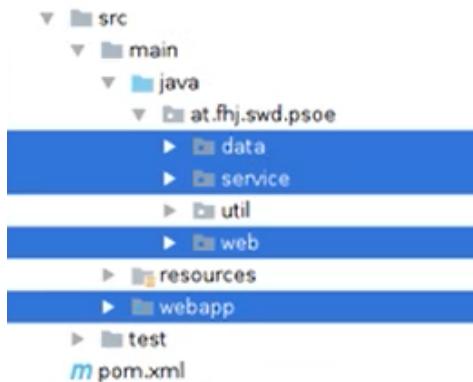


#### 1.1.1 3 Schichten Architektur:

- Data Source Layer (data): Zugriff auf Daten, kümmert sich um Kommunikation mit anderen Systemen (z.B.: Datenbank)
  - enthält Entities -> Java Repräsentation vom DB Entity
    - \* im Projekt wurde eine AbstractEntity erstellt, welche die id managed
  - beinhaltet DAO und DAOImpl » DocumentDAO, DocumentlibraryDAO
    - \* damit man auf die Entities zugreifen kann.
    - \* um die DB zu abstrahieren.
    - \* enthält Methoden mit denen auf die DB zugegriffen wird
    - \* eine DAOException kontrolliert den Input
      - . der EntityManager Aufruf in DAOImpl befindet sich innerhalb eines Try Blocks
      - . im catch wird der Cause in die DaoException gewrapped

- Domain Layer(service): enthält Business Logik (Berechnungen, Datenvalidierung, ...)
  - beinhaltet
    - \* **Service Layer Pattern** (aka Session Fassade - siehe 3)
    - \* DTO » DocumentDTO
    - \* Mapper » DocumentMapper

```
public static Document toEntity(DocumentDTO documentDTO, Document
→ document){};
public static DocumentDTO toDTO(Document document){};
```
- Presentation Layer(web): serverseitig, kümmert sich um Benutzerinteraktion
  - Controller (ViewHelper) » DocumentController, DocumentListController
  - View (WebApp)



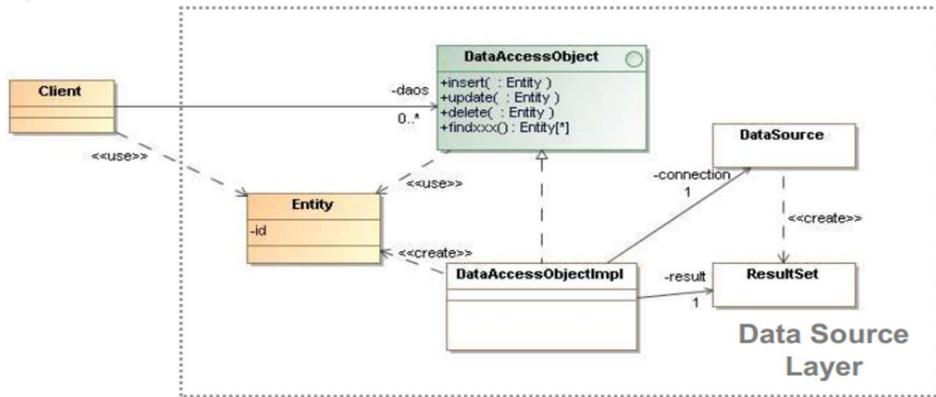
## 2 Data Access Object (DAO) Pattern

Befindet sich im Projekt in data und damit innerhalb des Data Layer.

### 2.1 Erkläre die Funktion + Skizze

- Client erstellt ein DAO Object und kann nach Entitäten suchen, einfügen, löschen, etc.
- das DAO selbst soll keine spezifischen Elemente enthalten (Entity Manager, SQL Exception -> stattdessen DAOException)
- dadurch entsteht eine Kapselung bei der die DAOImpl ohne den Client zu verändern ausgetauscht werden kann

```
@ApplicationScoped
public class DocumentDAOImpl implements DocumentDAO, Serializable {
  private static final long serialVersionUID = 1L;
  private static final Logger logger =
    LoggerFactory.getLogger(DocumentDAOImpl.class);
```



```
@PersistenceContext  
private EntityManager entityMangaer;  
  
@Override  
public List<Document> findByCommunity(Community community) {...}  
  
@Override  
public List<Document> findByUser(User user) {...}  
  
@Override  
public void insert(Document document) {...}  
  
@Override  
public void delete(Document document) {...}  
  
@Override  
public Document findById(Long id) {...}  
}
```

## 2.2 Nenne die Konsequenzen der Anwendung

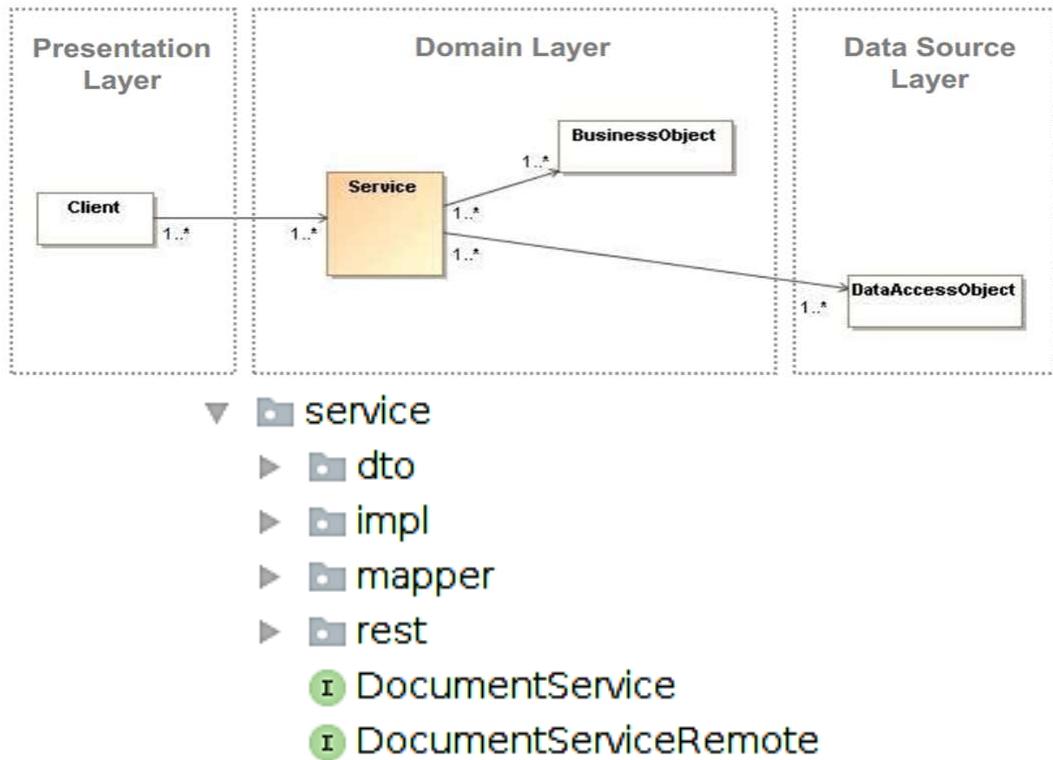
- Zugriff auf persistenten Speicher wird abstrahiert
  - Details des Speichers werden versteckt
    - ermöglicht einheitlichen Zugriff auf Daten
    - entkoppelt Implementierung von Persistierung (Datenbank,...)
    - ermöglicht Objektorientierte Ansicht des Speichers

### 3 Service Layer Pattern (auch Session Fassade - in unserem Projekt im Domain Layer)

### 3.1 Erkläre die Funktion + Skizze

- Der Service Layer (Ordner "service" im Projekt) delegiert auf die Business Logik (Zeile 68 `community.setDocumentlibrary`) und zum DAO (z.B. Zeile 66)

- Bei wenig Logik wird zumindest Transaktions (Zeile 40), Error (ab Zeile 42) und Validierungshandling (ab Zeile 23) im Service erledigt



```

1  @Local(DocumentService.class)
2  @Remote(DocumentServiceRemote.class)
3  @Stateless
4  public class DocumentServiceImpl implements DocumentService,
5      DocumentServiceRemote, Serializable {
6      private static final long serialVersionUID = -1L;
7      private static final Logger logger =
8          LoggerFactory.getLogger(DocumentServiceImpl.class);
9
10     @Inject
11     private DocumentDAO documentDAO;
12     @Inject
13     private DocumentlibraryDAO documentlibraryDAO;
14     @Inject
15     private CommunityDAO communityDAO;
16     @Inject
17     private UserDao userDao;
18     @Inject
19     private MessageDAO messageDAO;
20     @Override
21     public DocumentDTO addDocument(Long communityID, String userID, byte[]
22         data, String filename) {
23         Document addedDocument;
  
```

```

21         User user;
22
23     // Validierungshandling gefolgt von Error Handling
24     try {
25         if (communityID <= 0) throw new
26             ↪ IllegalArgumentException("community must not be empty");
27         if (userID == null) throw new IllegalArgumentException("user must
28             ↪ not be empty");
29         if (data == null) throw new IllegalArgumentException("uploaded
30             ↪ file must not be empty");
31         if (filename == null) throw new
32             ↪ IllegalArgumentException("filename must not be empty");
33
34         Documentlibrary documentlibrary =
35             ↪ documentlibraryDAO.findById(communityID);
36
37         //create a document library, if there isn't already one in the
38             ↪ database
39         if (documentlibrary == null) {
40             documentlibrary = addDocumentlibrary(communityID);
41         }
42
43         user = userDao.getById(userID);
44
45         addedDocument = new Document(documentlibrary, user, filename,
46             ↪ data);
47         documentDAO.insert(addedDocument); // Transaktionshandling
48         logger.info(String.format("Document %s saved in database",
49             ↪ filename));
50
51         // Error Handling
52     } catch (IllegalArgumentException iaex) {
53         String errorMsg = "Uploading file failed (illegal argument)";
54         logger.error(errorMsg, iaex);
55         throw new ServiceException(errorMsg);
56
57     } catch (Exception ex) {
58         String errorMsg = String.format("Uploading file %s failed.",
59             ↪ filename);
60         logger.error(errorMsg, ex);
61         throw new ServiceException(errorMsg);
62     }
63
64     String msgText = "Uploaded Document " + filename + " by user " +
65         ↪ user.getUserId();
66     addMessageToStream(communityID, user, msgText, addedDocument);
67     return DocumentMapper.toDTO(addedDocument);
68 }
69
70
71 private void addMessageToStream(Long communityID, User user, String text,
72     ↪ Document document) {...}
73
74 private Documentlibrary addDocumentlibrary(Long communityID) {

```

```

63         logger.info("Create missing documentlibrary");
64         Community community;
65         Documentlibrary documentlibrary = new Documentlibrary();
66         documentlibraryDAO.insert(documentlibrary); // Delegation zum DAO
67         community = communityDAO.findById(communityID); // Delegation zum
68         ↪ DAO
69         community.setDocumentlibrary(documentlibrary); // Delegation zur
70         ↪ Business Logik (Entity)
71         communityDAO.update(community); // Delegation zum DAO
72         return documentlibrary;
73     }
74
75     @Override
76     public List<DocumentDTO> getDocumentsFromCommunity(Long communityID)
77     ↪ {...}
78
79     @Override
80     public List<DocumentDTO> getDocumentsFromUser(String userID) {...}
81
82     @Override
83     public void removeDocument(Long documentID) {...}
84 }
```

### 3.2 Nenne die Konsequenzen der Anwendung

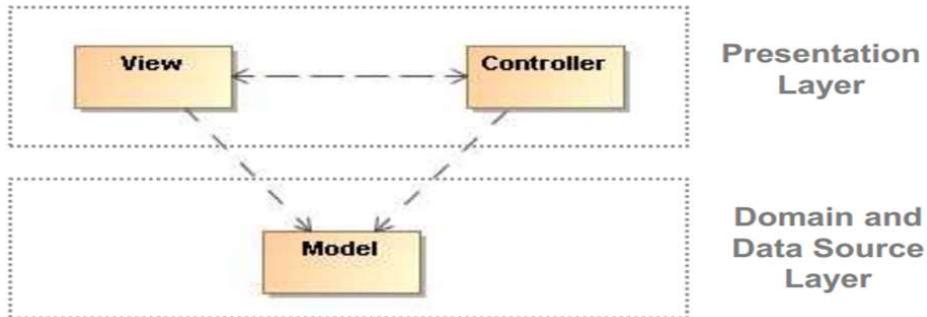
- Reduzierung der Abhängigkeiten zwischen Presentation und Domain Layer
- Zentralisiertes Sicherheits und Transaktionshandling
- verbirgt vor Client Komplexität der Business Logik
- stellt Client ein grobkörniges Interface zur Verfügung
- gut für Remote Aufrufe geeignet (weniger Aufrufe)

## 4 Model-View-Controller (MVC) Pattern

### 4.1 Erkläre die Funktion + Skizze

MVC unterteilt eine interaktive Applikation in drei Teile: Model, View und Controller.

- Controller und View befinden sich im Presentation Layer und haben gegenseitig Abhängigkeiten
- Das Model darf keine Abhängigkeiten haben (Controller und View hängen vom Model ab)



#### 4.1.1 Model

- Es befinden sich Teile im Domain und Data Source Layer.
- Das Model enthält die Kernfunktionalität und Daten. (z.B.: Datenbankzugriff)
- Im Projekt wird dies durch die Ordner *service* und *data* repräsentiert

#### 4.1.2 View

- Im Projekt im Ordner *webapp* zu finden.
- Enthält im Projekt *xhtml* Dateien zur Darstellung und User Interaktion

#### 4.1.3 Controller

- Im Projekt sind Controllerklassen im Ordner *web* zu finden.
- Sie enthalten die Logik und behandeln Benutzereingaben

## 5 Front Controller

### 5.1 Erkläre die Funktion + Skizze

- Client schickt Request an Front Controller
- FC erfasst nur Infos die er für die weiter Delegation braucht
- FC gibt Request an entsprechenden ConcreteCommand oder View weiter
- es gibt zwei Implementierungsvarianten des Controller
  - Servlet
  - ConcreteCommand

### 5.2 Servlet

- Im Projekt wurde der Front Controller in Form eines Servlet realisiert,
- die Einbindung erfolgt in der Konfigurationsdatei *src/main/webapp/WEB-INF/web.xml*,
- Servlet ist eine Java-API, welche auf einem Server betrieben wird,

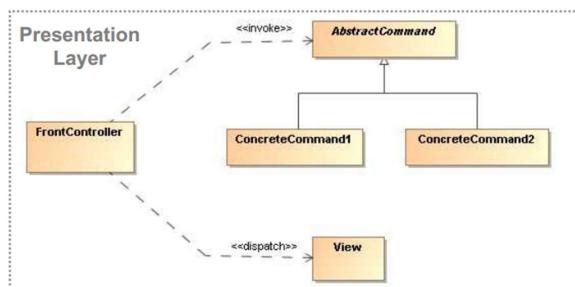
- die Verarbeitung von Requests und Responses wird ermöglicht,
- JSF und JSP können darauf aufsetzen, in unserem Projekt wurde JSF verwendet

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   ↳ xmlns="http://xmlns.jcp.org/xml/ns/javaee"
   ↳ xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
   ↳ http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
3 ...
4 <servlet>
5 <servlet-name>Faces Servlet</servlet-name>
6 <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
7 <load-on-startup>1</load-on-startup>
8 </servlet>
9 <servlet-mapping>
10 <servlet-name>Faces Servlet</servlet-name>
11 <url-pattern>*.xhtml</url-pattern>
12 </servlet-mapping>
```

### 5.2.1 Java Server Faces (JSF)

- JSF basiert auf dem MVC-Pattern
- JSF-View-Code ist im Projekt im Ordner `src/main/webapp/*` zu finden
- JSF-Logik befindet sich in den Java-Beans (im Projekt `/src/main/java/at/fhj/swd/psoe/web/*`)
- in unserem Projekt gibt es zu jeder xhtml-View eine eigene Controller-Klasse, welche dem ViewHelper-Pattern entspricht
- in unserem Projekt kommt PrimeFaces zum Einsatz (eine konkrete Implementierungsart von JSF => Einbindung in pom.xml)



```

1 <!-- Pfad: /src/main/webapp/community/documentManagement.xhtml -->
2
3 <?xml version='1.0' encoding='UTF-8' ?>
4 <!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   ↳ "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
5 <ui:composition xmlns="http://www.w3.org/1999/xhtml"
6 xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
```

```

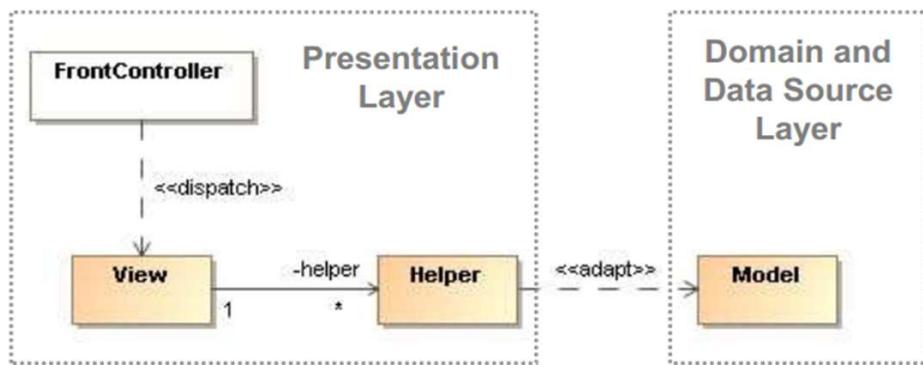
7  xmlns:h="http://xmlns.jcp.org/jsf/html"
8  xmlns:p="http://primefaces.org/ui"
9  xmlns:f="http://xmlns.jcp.org/jsf/core"
10 template="communityTemplate.xhtml">
11 <ui:define name="communityContent">
12 <h1>#{msg.document_manage_title}</h1>
13 <f:metadata>
14 <f:viewAction action="#{documentListController.loadDocumentsFromCommunity()}" />
15 </f:metadata>
16
17 <h:form id="doclistform">
18 <p:commandButton value="Refresh list"
   ↳  actionListener="#{documentListController.loadDocumentsFromCommunity()}"
   ↳  update="@form doclistform"></p:commandButton>
19 <p:dataTable id="doclisttable" value="#{documentListController.communityDocuments}"
   ↳  var="docs">
20 <p:column class="documenttimecolumn"
   ↳  headerText="#{msg.document_uploaded}">#{docs.createdTimestamp}</p:column>
21 <p:column class="documenttimecolumn"
   ↳  headerText="#{msg.label_userid}">#{docs.user.userId}</p:column>
22 <p:column headerText="#{msg.label_filename}">#{docs.filename}</p:column>
23 <p:column headerText="" class="documentbuttoncolumn">
24 <p:commandButton value="#{msg.button_download}" ajax="false"
25 onclick="PrimeFaces.monitorDownload(start, stop);">
26 <p:fileDownload value="#{documentController.downloadDocument(docs.id)}"/>
27 </p:commandButton>
28 </p:column>
29 <p:column headerText="" class="documentbuttoncolumn">
30 <p:commandButton id="btnDel" value="#{msg.button_delete}"
31 actionListener="#{documentController.removeDocument(docs.id)}"
32 update="@form doclistform">
33 </p:commandButton>
34 </p:column>
35 </p:dataTable>
36 </h:form>
37 <h:form id="formdocupload" enctype="multipart/form-data">
38 <p:fileUpload id="fileupload"
39 dragDropSupport="false"
40 update="@form doclistform"
41 fileUploadListener="#{documentController.uploadDocument}"
42 allowTypes="/(.|\|/)(pdf|jpe?g|docx)\$/" sizeLimit="5000000"
43 mode="advanced" label="Add document (.pdf .jpg .docx)">
44 </p:fileUpload>
45 </h:form>
46 <p:messages id="feedbackBox" severity="info,error" showDetail="true"
   ↳  showSummary="false">
47 <p:autoUpdate/>
48 </p:messages>
49 </ui:define>
50 </ui:composition>
```

### 5.3 Nenne die Konsequenzen der Anwendung

- es muss nur EIN (Front) Controller konfiguriert werden
- da bei jedem Request ein neues Command Objekt erzeugt wird ist Thread-Safety nicht notwendig
- da nur EIN Controller sind auch Erweiterungen durch z.B.: Decorator einfach (auch zur Laufzeit)

## 6 View Helper (*/src/main/java/at/fhj/swd/psoe/web/\**)

### 6.1 Erkläre die Funktion + Skizze



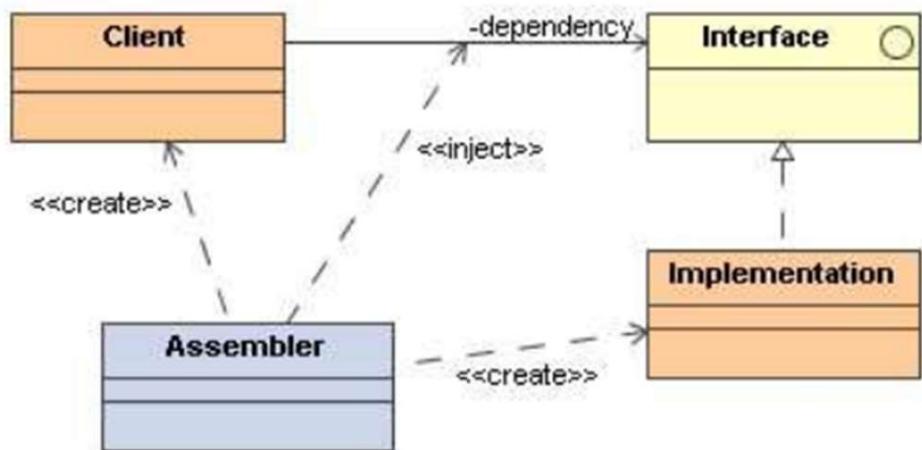
- View (xhtml-Dateien im Ordner */src/main/webapp/\**) delegiert Aufgaben an Helper (z.B. DocumentController im Ordner web)
- Helper adaptieren View zu Model (Klassen in den Ordnern *src/main/java/at/fhj/swd/psoe/service/\** und *src/main/java/at/fhj/swd/psoe/data/\**)
- in View befindet sich HTML Code im ViewHelper Java Code zur Aufbereitung der Daten (+ wenig HTML)

### 6.2 Nenne die Konsequenzen der Anwendung

- kapselt Design-Code in View und View-Processing-Code Logik in Helper
- steigert Wiederverwendbarkeit, Wartbarkeit und Strukturierungsqualität der Anwendung
- vereinfacht Tests (Helperfunktionen ohne View)
- bessere Trennung zwischen
  - Presentation und Data Source Layer
  - Entwickler und Designer

## 7 Dependency Injection (CDI-Framework -> eingebunden im ./pom.xml)

### 7.1 Erkläre die Funktion + Skizze



- ein Hauptgrund für Dependency Injection ist die Trennung zwischen Business- und Instanzierungslogik
- Klassen sollen nur noch Abhängigkeiten auf Interfaces haben
- die Verwendung des new Operators stattdessen würde eine Abhängigkeit zur Implementierungsklasse herstellen
- Implementierungsklassen sollen auf Interfaces referenzieren, wobei es ihnen egal wie diese Referenz hergestellt wird
- Die Instanziierung wird in die sogenannte Assembler Klasse "herausgezogen"
- Objekte werden mittels des Assembler "gebaut".
- Zudem instanziert er die Objekte und injected die Referenzen
- So hat er Client nur mehr eine Abhängigkeit auf das Interface
- wird die Implementierung getauscht, so geschieht das direkt im Assembler ohne den Client zu beeinflussen
- der Container erledigt DI über @Inject indem er prüft welche Klasse das Interface implementiert und entsprechende Referenz bei der Annotation einhängt
- Es wird zwischen Constructor Injection und Setter Injection unterschiedlichen

```
1 // Constructor Injection
2 public class Client {
3     private Interface iface;
4     public Client(Interface iface) {
```

```

5         this.iface = iface;
6     }
7
8     // Setter Injection
9     public class Client {
10         private Interface iface;
11         public setInterface(Interface iface)
12         {
13             this.iface = iface;
14         }

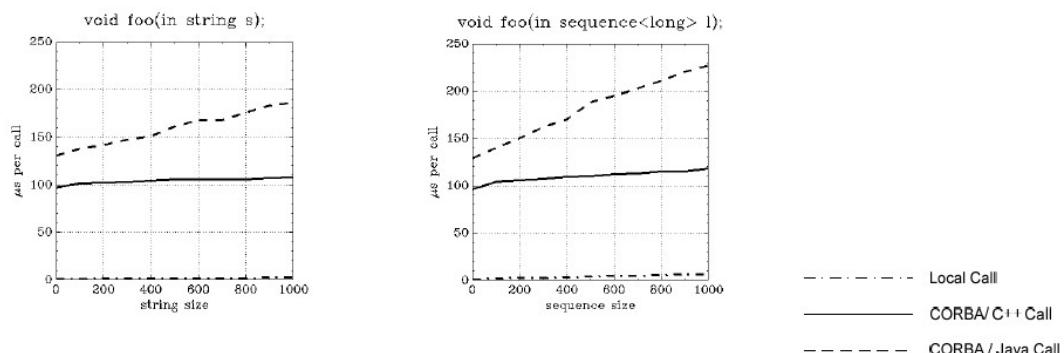
```

## 7.2 Nenne die Konsequenzen der Anwendung

- lose gekoppelte Objekte
- Referenzen nur noch auf Interfaces
- hohe Flexibilität (Strategy, Proxy,..)
- bessere Erweiterbarkeit und Testbarkeit

# 8 Data Transfer Object (DTO) Pattern

## 8.1 Erkläre die Funktion (Skizze - ein Grund für DTO)



- Transportiert Daten zwischen Prozessen um Remote Methodaufrufe zu minimieren
- besteht aus Fields, Getter und Setter
- fasst Daten verschiedener Objekte zusammen die vom Remote Objekt benötigt werden
- ev. Map, Record Set, ... -> um Anzahl der Aufrufe zu minimieren

```

1 package at.fhj.swd.psoe.service.dto;
2
3 import java.io.Serializable;
4 import java.util.Date;
5

```

```

6  public class DocumentDTO implements Serializable {
7      private static final long serialVersionUID = 4016557982897997689L;
8
9      private Long id;
10     private Long documentlibraryID;
11     private String filename;
12     private UserDTO user;
13     private byte[] data;
14     private Date createdTimestamp;
15
16     public DocumentDTO() {}
17
18     public Long getId() {
19         return id;
20     }
21
22     public void setId(Long id) {
23         this.id = id;
24     }
25
26     public Long getDocumentlibraryID() {
27         return documentlibraryID;
28     }
29
30     public void setDocumentlibraryID(Long documentlibraryID) {
31         this.documentlibraryID = documentlibraryID;
32     }
33
34     public String getFilename() {
35         return filename;
36     }
37
38     public void setFilename(String filename) {
39         this.filename = filename;
40     }
41
42     public UserDTO getUser() {
43         return user;
44     }
45
46     public void setUser(UserDTO user) {
47         this.user = user;
48     }
49
50     public byte[] getData() {
51         return data;
52     }
53
54     public void setData(byte[] data) {
55         this.data = data;
56     }
57
58     public Date getCreatedTimestamp() {

```

```

59             return createdTimestamp;
60     }
61
62     public void setCreatedTimestamp(Date createdTimestamp) {
63         this.createdTimestamp = createdTimestamp;
64     }
65
66     @Override
67     public String toString() {
68         return "DocumentDTO{" +
69             "id=" + id +
70             ", documentlibraryID=" + documentlibraryID +
71             ", filename='" + filename + '\'' +
72             '}';
73     }
74
75     @Override
76     public boolean equals(Object o) {
77         if (this == o) return true;
78         if (!(o instanceof DocumentDTO)) return false;
79
80         DocumentDTO that = (DocumentDTO) o;
81
82         return id.equals(that.id);
83     }
84
85     @Override
86     public int hashCode() {
87         return id.hashCode();
88     }
89 }
```

## 8.2 Konsequenzen der Anwendung

- kapselt und versteckt
- nimmt Komplexität
- steigert Effizienz da weniger Aufrufe über Remotecgrenze

# Part II

## Exception Handling

### 9 Checked und Runtime Exceptions in Java

#### 9.1 Checked Exceptions (z.B. SQL-Exception)

- leiten von Exception Klasse ab und müssen behandelt werden (throws - catch)
- Verwendung für Probleme die durch User behoben werden können (alternative Aktion)

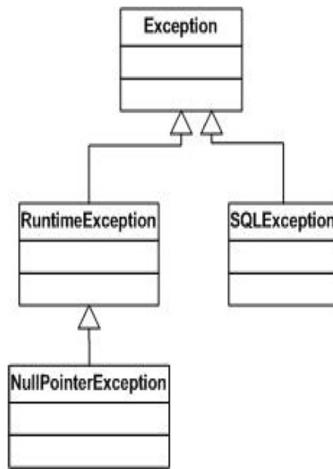
#### 9.2 Unchecked Exceptions (z.B. NullPointerException)

- leiten von RuntimeException ab
- Verwendung für technische Probleme (User kann nichts machen außer neu starten)
  - beschädigte Datenbank - die Exception geht durch alle Layer
    - \* erst mit Implementierungsspezifischer Exception
    - \* später mit Runtime **ohne Stacktrace** bis zum User (-> Destructive wrapping mit Log and Throw im ServiceLayer)
    - \* im Projekt: ServiceException, DaoException, AuthenticationException und SerializerException

```
1 package at.fhj.swd.psoe.service;
2
3 public class ServiceException extends RuntimeException {
4     private static final long serialVersionUID = -1109707847007116930L;
5
6     public ServiceException(String message) {super(message);}
7
8     package at.fhj.swd.psoe.data;
9
10    public class DaoException extends RuntimeException {
11        private static final long serialVersionUID = -2712863481296295032L;
12
13        public DaoException(String message, Throwable cause) {
14            super(message, cause);
15        }
16        public DaoException(Throwable cause) {super(cause);}
17    }
18}
```

### 10 Best Practice Beispiele beim Einsatz von Exceptions

- Exceptions nicht für Programmflusskontrolle verwenden (schlechte Performance)
- offene Ressourcen schließen (try-with-resources bzw. close im finally)
- selbst erstellte Exceptions auch mit nützlichen Infos ausstatten



- Implementierungsspezifische Exceptions nicht bis zum User durchwerfen (stattdessen catch + throw RuntimeException)
- dokumentieren mit @throws im DOC, testen mit JUnit

## 11 Exception Handling Anti Pattern

- Log and Throw (nie beides: entweder, oder)
- Throwing Exception bzw. catch Exception (spezifischere anstatt Basisklasse verwenden)
- Destructive Wrapping (wenn bei catch + throw = wrapping nicht die Original Exception weitergegeben wird)
- Log and return Null (provoziert an einer anderen Stelle eine NullPointerException)
- Catch and Ignore
- Unsupported Operation return Null (besser UnsupportedOperationException)

## 12 Destructive Wrapping im Service Layer

- im Codebeispiel wird in Zeile 12 eine `IllegalArgumentException` (Runtime) gefangen in Zeile 18 die Exception allgemein
- beide werden in Zeile 14 bzw. 18 inklusive Stacktrace geloggt
- ausnahmsweise muss hier zusätzlich auch eine neue `ServiceException` geschmissen werden, jedoch **ohne Stacktrace** (siehe Zeile 15 und 19)

```

1 package at.fhj.swd.psoe.service.impl;
2 ...
3 @Override
4 public List<DocumentDTO> getDocumentsFromCommunity(Long communityID) {
5     try {
  
```

```

6         List<Document> documents;
7         if (communityID <= 0) throw new IllegalArgumentException("community
8             → must not be empty");
9         Community community = communityDAO.findById(communityID);
10        if (community == null) throw new IllegalStateException("community "
11            → + communityID + " not found");
12        documents = documentDAO.findByCommunity(community);
13        return
14            → documents.stream().map(DocumentMapper::toDTO).collect(Collectors.toList());
15    } catch (IllegalArgumentException iaex) {
16        String errorMsg = "Could not load docs from community (illegal
17            → argument)";
18        logger.error(errorMsg, iaex);
19        throw new ServiceException(errorMsg);
20    } catch (Exception ex) {
21        String errorMsg = "Could not load docs for community.";
22        logger.error(errorMsg + " id " + communityID, ex);
23        throw new ServiceException(errorMsg);
24    }
25
26    ...

```

## 13 Exception Testing

```

1             package at.fhj.swd.psoe.service;
2
3                 import java.util.ArrayList;
4
5                 ...
6
7                 @RunWith(MockitoJUnitRunner.Silent.class)
8                 public class DocumentServiceTest {
9                     @Mock
10                    private DocumentDAO documentDAO;
11
12
13                     @Test(expected = ServiceException.class)
14                     public void getDocumentsFromCommunity_WithId0_ShouldFail() {
15                         documentService.getDocumentsFromCommunity(0L);
16                     }
17
18                     @Test(expected = ServiceException.class)
19                     public void getDocumentsFromCommunity_NoDocuments_ShouldFail() {
20                         Community community = Mockito.mock(Community.class);
21                         Mockito.when(community.getId()).thenReturn(COMMUNITYID);
22                         Documentlibrary doclib = Mockito.mock(Documentlibrary.class);
23                         Mockito.when(community.getDocumentlibrary()).thenReturn(doclib);
24                         Mockito.when(doclib.getCommunity()).thenReturn(community);
25                         Mockito.when(communityDAO.findById(community.getId())).thenReturn(null);
26
27                         documentService.getDocumentsFromCommunity(COMMUNITYID);
28                     }
29
30                     ...

```

# Part III

## Allgemeines & Config

### 14 Logging

#### 14.0.1 Vorteile Logging mittels Framework (z.B.: log4j)

- Nutzt ein einheitliches Format / Konventionen
- logging kann optional an und ausgeschalten werden
- durch verschiedene Log-level können Logs gefiltert erstellt werden
- Layout für Ausgabe kann zentral definiert/geändert werden

### 15 Annotationen

- @MappedSuperclass
  - ist im Hybernate Framework
  - eine Klasse durch die gemeinsame Felder definiert werden.
  - definiert eine abstrakte Superklasse
- @Produces
  - kommt während deployment, markiert Factory Method damit man nicht direkt auf die Klasse zugreifen muss
- @Typed
  - zeigt die Vererbung Wieso bei uns allein stehend?
- @Named
  - Zeigt bei Mehrdeutigkeit das richtige Objekt mit dem Namen
- @Resource
  - fast wie Dependency Injection
  - damit weiß der Container wie er das Annotierte Feld instanzieren muss
- @Stateless
  - speichert den Client Status nicht
- @Entity
  - Data Access Layer
- @Table

- Tabellenname im SQL
- @Column
  - SQL-Spalten nullable=false
- @OneToMany
- @JoinColumns
  - welche Spalten zusammen gehören FK
- @ManyToOne
  - auf anderen Seite
- @ApplicationScoped
  - lebt die ganze Applikation lang, wird einmal gemacht.
- @PersistenceContext
  - persistance.xml auslesen für Treiber und andere JPA Geschichten + Data Source. Entity Manager
- @Id
  - das ist die id
- @GeneratedValue
  - Wert kommt aus der DB
- @Local
  - Klasse für lokale Aufrufe.
- @Remote
  - interprozessaufrufe. RMI
- @ApplicationException
  - Rollback wenn so eine Exception kommt, Nachricht zum Client.

### 15.1 Annotationen - Details

- CascadeType.Remove löscht die damit Annotierte Verknüpfung mit
- dies geht auch rekursiv in der kompletten Datenbank
- CascadeType.Remove und orphanRemoval=true ist equivalent

```

1     ...
2     @OneToOne(fetch = FetchType.EAGER,
3             cascade = CascadeType.ALL,
4             orphanRemoval = true)
5     @JoinColumn(name = "documentlibrary_id")
6     private Documentlibrary documentlibrary;
7
8     @Column(nullable = false, unique = true)
9     private String name;
10    ...
11    public enum CascadeType {
12        /* Cascade all operations /
13        ALL,
14        /* Cascade persist operation /
15        PERSIST,
16        /* Cascade merge operation /
17        MERGE,
18        /* Cascade remove operation /
19        REMOVE,
20        /* Cascade refresh operation /
21        REFRESH,
22        */
23        DETACH
24        ...
25    }
26 )

```

## 16 Konfigurationsdateien

### 16.1 persistence.xml

Die Datei *persistence.xml* ist der zentrale Bestandteil der Persistierungs-Konfiguration. Folgende Dinge können konfiguriert werden:

- SQL dialect
- the persistence provider that shall be used at runtime
- the data source you want to use to connect to your database
- several provider-specific configuration parameters

Listing 1: persistence.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.0"
3 xmlns="http://java.sun.com/xml/ns/persistence"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
6   http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
7
8 <persistence-unit name="primary">
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
312
313
313
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
```

```

7 <jta-data-source>java:jboss/datasources/psoeDS</jta-data-source>
8 <properties>
9   <!-- Properties for Hibernate -->
10  <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect"
11    />
12  <property name="hibernate.enable_lazy_load_no_trans" value="true" />
13  <!--
14    SQL stdout logging
15    -->
16  <property name="hibernate.show_sql" value="true"/>
17  <property name="hibernate.format_sql" value="false"/>
18  <property name="use_sql_comments" value="true"/>
19  <!--
20  <property name="hibernate.hbm2ddl.auto" value="create-drop" />
21  -->
22 </properties>
23 </persistence-unit>
24
25 </persistence>

```

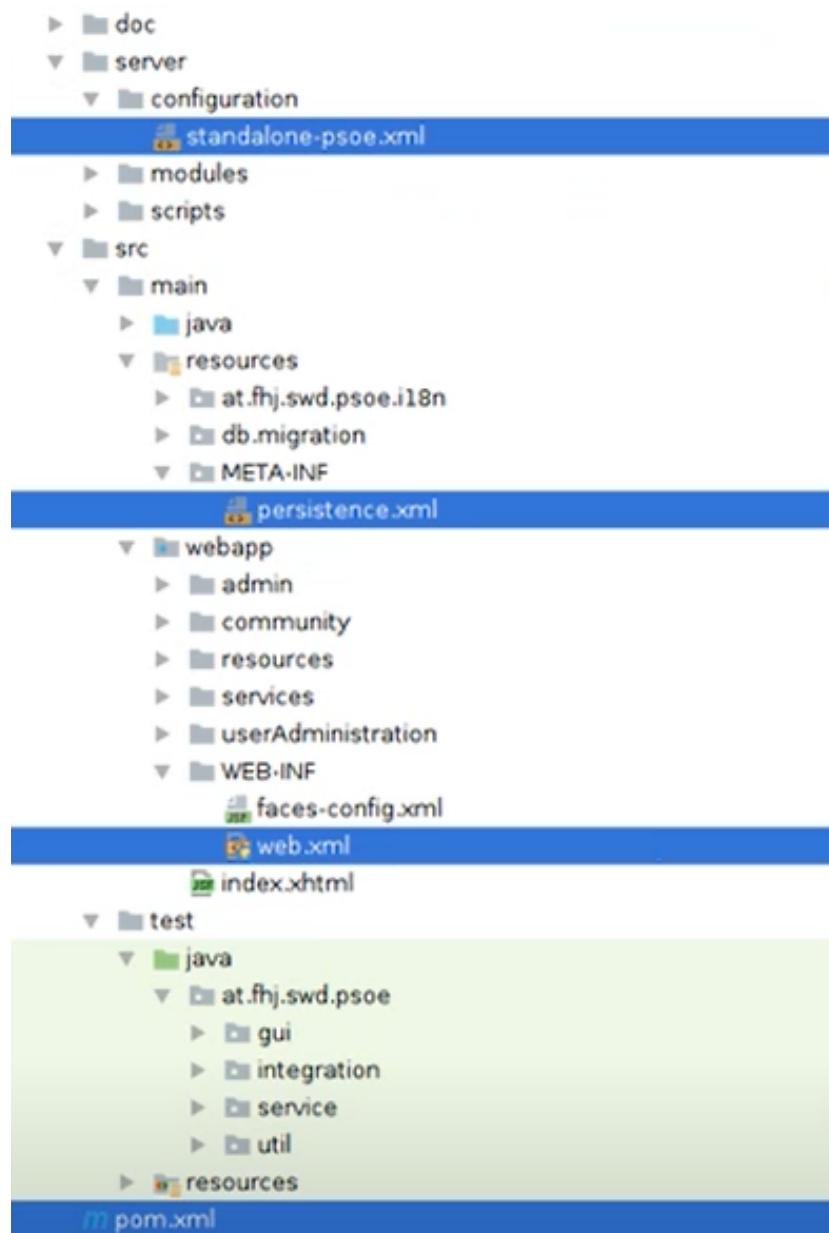
## 16.2 web.xml

- konfiguriert den Java Webserver (Wildfly - JBOSS)
- Einbindung des Faces-Servlet (FrontController - Implementierung, Zugriffskontrolle, Rollenkonfiguration)
- befindet sich im Ordner **src/main/webapp/WEB-INF/web.xml**

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
5   http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" version="3.1">
6
7 <servlet>
8   <servlet-name>Faces Servlet</servlet-name>
9   <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
10  <load-on-startup>1</load-on-startup>
11 </servlet>
12 <servlet-mapping>
13   <servlet-name>Faces Servlet</servlet-name>
14   <url-pattern>*.xhtml</url-pattern>
15 </servlet-mapping>
16
17 <!-- Security roles -->
18 <security-role>
19   <description>administrators</description>

```



```

20 <!-- Security constraints -->
21 <security-constraint>
22 <web-resource-collection>
23 <web-resource-name>admin area</web-resource-name>
24 <url-pattern>/admin/*</url-pattern>
25 </web-resource-collection>
26 <auth-constraint>
27 <role-name>ADMIN</role-name>
28 </auth-constraint>
29 </security-constraint>
30
31 <login-config>
32 <auth-method>FORM</auth-method>
33 <realm-name>pse</realm-name>
34 <form-login-config>
35 <form-login-page>/login.xhtml</form-login-page>
36 <form-error-page>/login.xhtml</form-error-page>
37 <!--          <form-error-page>/loginerror.xhtml</form-error-page> -->
38 </form-login-config>
39 </login-config>
40 </web-app>

```

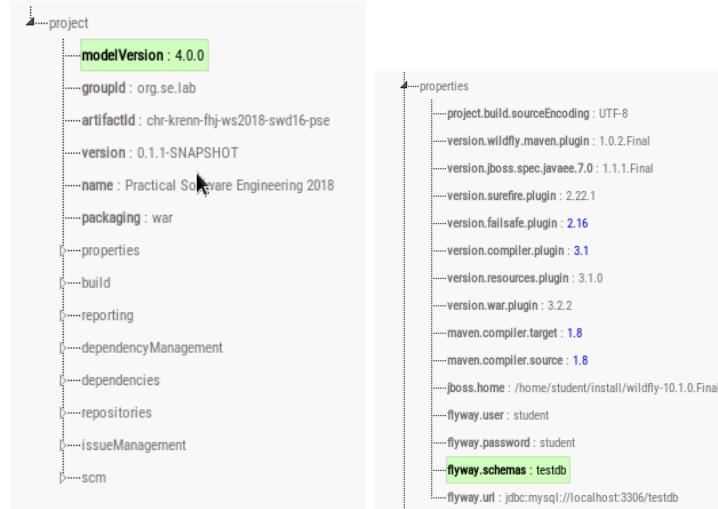
### 16.3 pom.xml

- resources-plugin (bindet die Serverressourcen ein - Ordner *configuration* im Projekt - z.B. *standalone-psoe.xml*)
- Wildfly (JBoss)- Webserver
  - 1. Compile
  - 2. Surefire (unitTests)
  - 3. Packaging - war file erstellen
  - 4. Wildfly - fressen und deployen
  - 5. Failsafe IT-test
  - 6. MVN site
  - 7. Gui test
- Primeface = jsf Framework
- Jacoco = test Coverage
- Slf4j = logger
- Jaxb – xml
- Cdi = context dependancy injection

#### 16.3.1 Aufbau pom.xml

### 16.4 standalone-psoe.xml

Wird ein JBoss Applikationsserver im *standalone*-Modus betrieben, läuft jede Instanz in einem eigenen Prozess. Diese Datei ist eine Java Enterprise Edition 6 zertifizierte Web-Profil Konfiguration welche alle benötigten Technologien (z.B. Extensions von JBoss, Datasources etc.) definiert.



JBoss EAP benutzt standardmäßig die standalone.xml Konfigurationsdatei, kann aber auch unter Verwendung einer anderen gestartet werden.

#### Abschnitte der standalone.xml

- extensions (z.B. diverse Wildfly Module)
- management (z.B. Access Control -> role-mapping)
- profile (z.B. JPA Subsystem)
- interfaces (z.B. \${jboss.bind.address:127.0.0.1})
- socket-binding-group (z.B \${jboss.http.port:8080})
- Rechte (Management-Realm)
- Datenbankzugriffsparameter

Listing 2: standalone.xml (auszugsweise)

```

1      .....
2
3          </endpoint-config>
4      <client-config name="Standard-Client-Config"/>
5  </subsystem>
6  <subsystem xmlns="urn:jboss:domain:weld:3.0"/>
7  </profile>
8  <interfaces>
9      <interface name="management">
10         <inet-address value="${jboss.bind.address.management:127.0.0.1}" />
11     </interface>
12     <interface name="public">
13         <inet-address value="${jboss.bind.address:127.0.0.1}" />
14     </interface>
15 </interfaces>
```

```

16 <socket-binding-group name="standard-sockets" default-interface="public"
17   ↳ port-offset="${jboss.socket.binding.port-offset:0}">
18 <socket-binding name="management-http" interface="management"
19   ↳ port="${jboss.management.http.port:9990}" />
20 <socket-binding name="management-https" interface="management"
21   ↳ port="${jboss.management.https.port:9993}" />
22 <socket-binding name="ajp" port="${jboss.ajp.port:8009}" />
23 <socket-binding name="http" port="${jboss.http.port:8080}" />
24 <socket-binding name="https" port="${jboss.https.port:8443}" />
25 <socket-binding name="txn-recovery-environment" port="4712" />
26 <socket-binding name="txn-status-manager" port="4713" />
27 <outbound-socket-binding name="mail-smtp">
28   <remote-destination host="localhost" port="25" />
</outbound-socket-binding>
</socket-binding-group>
</server>

```

## 16.5 log4j.properties

textitsrc/test/resources/log4j.properties

# 17 Fehler im Projekt

## 17.1 Return null

Anstatt von Null einfach eine Leere Liste bzw. ein default Objekt (oder new <Object>) zurückgeben

## 17.2 Exception nicht gefangen

```

1 package at.fhj.swd.psoe.service.impl;
2 ...
3 @Override
4 public void removeCommunityByAdmin(CommunityDTO communityDTO) {
5     Community community = communityDAO.findById(communityDTO.getId());
6     String errorText;
7     try {
8         communityDAO.removeCommunityByAdmin(community);
9     } catch (DaoException e) {
10         errorText = "Error removing community";
11         logger.error(errorText, e);
12         throw new ServiceException(errorText);
13     }
14 }

```

## 17.3 Destructive Wrapping - Logging fehlt - Information geht verloren

```

1 package at.fhj.swd.psoe.service.impl;
2 ...
3 @Override
4 public CommunityDTO updateCommunityEnabled(String adminUserId, String communityName,
→   boolean isEnabled) {

```

```

5     String errorText = "";
6     try {
7         boolean hasPermission = false;
8         User adminUser = userDao.getByUserId(adminUserId);
9         Community community = communityDAO.getByName(communityName);
10
11
12        for (Role r : adminUser.getRoles()) {
13            if (r.getname().equals("ADMIN") || r.getname().equals("PORTALADMIN")) {
14                hasPermission = true;
15            }
16        }
17        if (hasPermission ||
18            → adminUser.getUserId().equals(community.getCommunityAdminUser().getUserId()))
19            → {
20                community.setIsEnabled(isEnabled);
21                communityDAO.update(community);
22            } else {
23                errorText = "No Permission to update community";
24                throw new AuthenticationException(errorText);
25            }
26        return CommunityMapper.toDTO(community);
27    } catch (DaoException e) {
28        errorText = "Error updating community enabled";
29        throw new ServiceException(errorText);
30    } catch (AuthenticationException e) {
31        throw new ServiceException(errorText);
32    } catch (Throwable e) {
33        errorText = "Unknown error updating community enabled";
34        throw new ServiceException(errorText);
35    }
36}

```

## 17.4 Logger mit falschem Parameter

```

1 package at.fhj.swd.psoe.service.impl;
2 ...
3 @Stateless
4 public class DepartmentHierarchyServiceImpl implements
5     → DepartmentHierarchyService, Serializable {
6
7     private static final long serialVersionUID = -2467949382018996094L;
8     private static final Logger logger =
9         → LoggerFactory.getLogger(UserServiceImpl.class);
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
311
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
142
```

```

6  public class MessageServiceImpl implements MessageService, MessageServiceRemote,
7      → Serializable {
8
9      ...
10     @Override
11     public MessageDTO getById(long id) {
12         Message message = messageDAO.getById(id);
13         if (message == null) {
14             return null;
15         }
16         return MessageMapper.toDTO(message);
17     }
18
19     @Override
20     public MessageDTO updateByAdmin(long id, String content, Date changed) {
21         Message message = messageDAO.getById(id);
22         message.setContent(content);
23         message.setEditedByAdmin(changed);
24         return MessageMapper.toDTO(messageDAO.update(message));
25     }
26
27     @Override
28     public MessageDTO updateByUser(long messageId, String content, Date changed) {
29         Message message = messageDAO.getById(messageId);
30         message.setContent(content);
31         message.setEditedByUser(changed);
32         return MessageMapper.toDTO(messageDAO.update(message));
33     }

```

## 17.6 Exception werfen und gleich wieder fangen

```

1      ...
2  public class MessageServiceImpl implements MessageService, MessageServiceRemote,
3      → Serializable {
4      private static final long serialVersionUID = 6768291437557855130L;
5
6  // nicht optimal, da die IllegalArgumentException gleich wieder gefangen wird
7  // überdies wird alles andere nicht gefangen
8
9
10     @Override
11     public void deleteMessage(long id) {
12         try {
13             Message message = messageDAO.getById(id);
14             if (message == null) {
15                 throw new IllegalArgumentException("Message cannot be empty");
16             }
17             messageDAO.delete(message);
18             logger.info("Message deleted successfully");
19         }
20         catch (IllegalArgumentException ex) {
21             String errorMsg = "Could not delete the message (illegal argument)";
22             logger.error(errorMsg, ex);
23             throw new ServiceException(errorMsg);
24         }

```

```

23         }
24     }
25
26     //----- besser wäre
27     package at.fhj.swd.psoe.service.impl;
28 ...
29     // erst loggen, dass man in die Methode gekommen ist
30     // wenn userDTO null ist wird IllegalArgumentException geworfen und das außerhalb des try
31     // catch blocks
32     // erst wird die DaoException gefangen und anschließend alle anderen
33     // Stacktrace wird geloggt und jeweils die ServiceException weitergegeben
34     @Override
35     public void saveUser(UserDTO userDTO) {
36         logger.debug("UserService saveUser() called with parameter: '{}'", userDTO);
37
38         if (userDTO == null) {
39             throw new IllegalArgumentException("userDTO is null");
40         }
41
42         try {
43             User user = (userDTO.getId() == null) ? new User() :
44                 userDAO.getById(userDTO.getId());
45             userDAO.update(UserMapper.toEntity(userDTO, user));
46         } catch (DaoException e) {
47             logger.error("Error saving user", e);
48             throw new ServiceException("Error saving user");
49         } catch (Throwable e) {
50             logger.error("Unknown error saving user", e);
51             throw new ServiceException("Unknown error saving user");
52         }
53     }

```

## 18 Tests

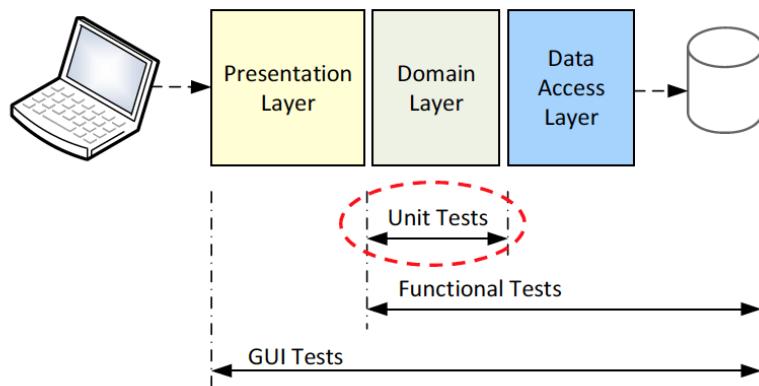


Figure 1

- Ziel der Tests ist es, 100% Codecoverage im Service-Layer zu erreichen (Anforderung für unser Projekt),
- Private und Protected-Methoden müssen per se nicht getestet werden (Rule of Thumb), Wege zum erfolgreichen Test:
  - Access-Modifier der Methoden auf public (public static) ändern,
  - Extrahieren der Methoden in neue Klassen,
  - Notfalls Package-Sichtbarkeit in Methode, wenn diese privat bleiben muss, Tests aber unbedingt notwendig sind.

## 18.1 Testpyramide

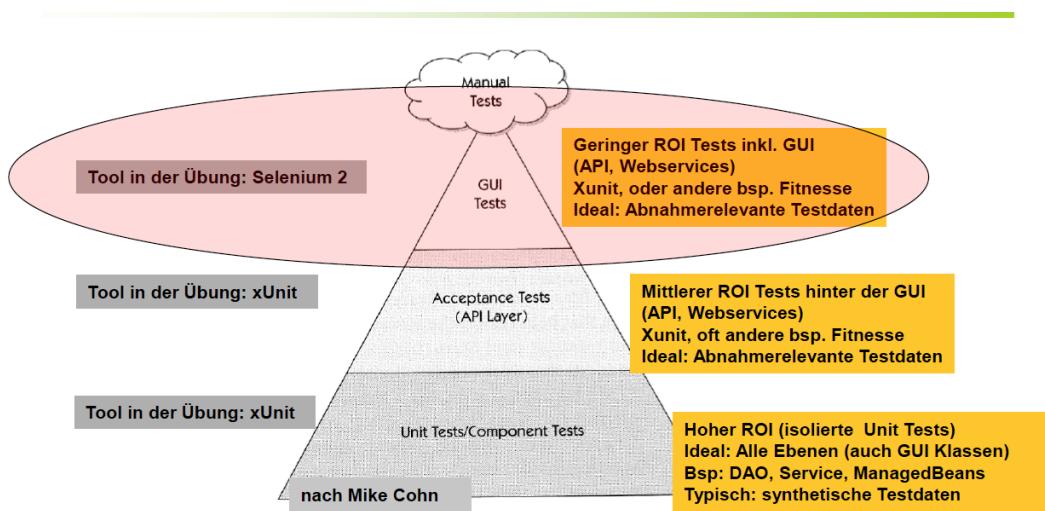


Figure 2

## 18.2 Unit

- Im Build-Cycle werden zuerst Unit-Tests ausgeführt,
- sie befinden sich im Projekt im Ordner `src/test/at/fhj/swd/psoe/service/*Test.java`,
- ein Unit-Test muss jede Methode der Service-Klasse testen
- Mockito imitiert die Methoden, die von der Datenbank abhängen, da diese zu diesem Zeitpunkt noch nicht zur Verfügung steht (Build-Cycle => Compile - Unit-Tests)
- Im Unit-Test wird jeweils die kleinste Einheit getestet
- Exceptions einfach mit Mockito werfen und testen, ob sie geworfen wurden

```

1  @Test(expected = ServiceException.class)
2  public void testGetCommunityByUserMessagesDAOException()
3  {
4      Mockito.when(this.userPrincipal.getId()).thenReturn(-1L);
5      Mockito.when(this.communityService.loadAllCommunitiesByUser(-1L)).thenThrow(new
6          → DaoException(new RuntimeException()));
7
8      this.activitystreamService.getCommunityByUserMessages();
9  }
10
11
12
13
14
15
16
17
18
19
20
21 }
```

### 18.3 Integration Tests

- testen alle Komponenten mit (Datenbank)
- werden im Build-Cycle erst nach dem Deployen des WAR-Files ausgeführt (laufende Applikation)
- In unserem Projekt kommt das Plugin *failsafe* zum Einsatz
- Datenbankscripts werden separat zu den anderen Scripts ausgeführt (Datenbank vorbereiten und auf den alten Stand zurückbringen)
- 

### 18.4 GUI-Test

#### 18.4.1 Page Object Pattern

- stellt Screens der Web-App als Reihe von Objekten dar

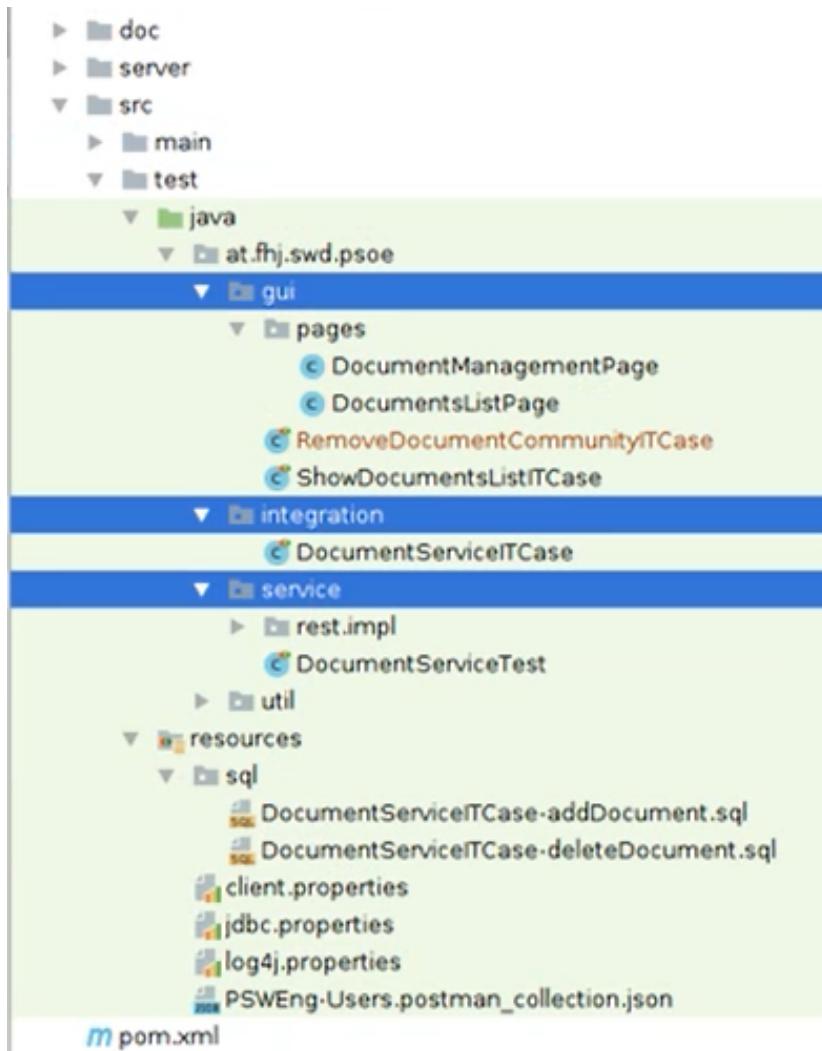


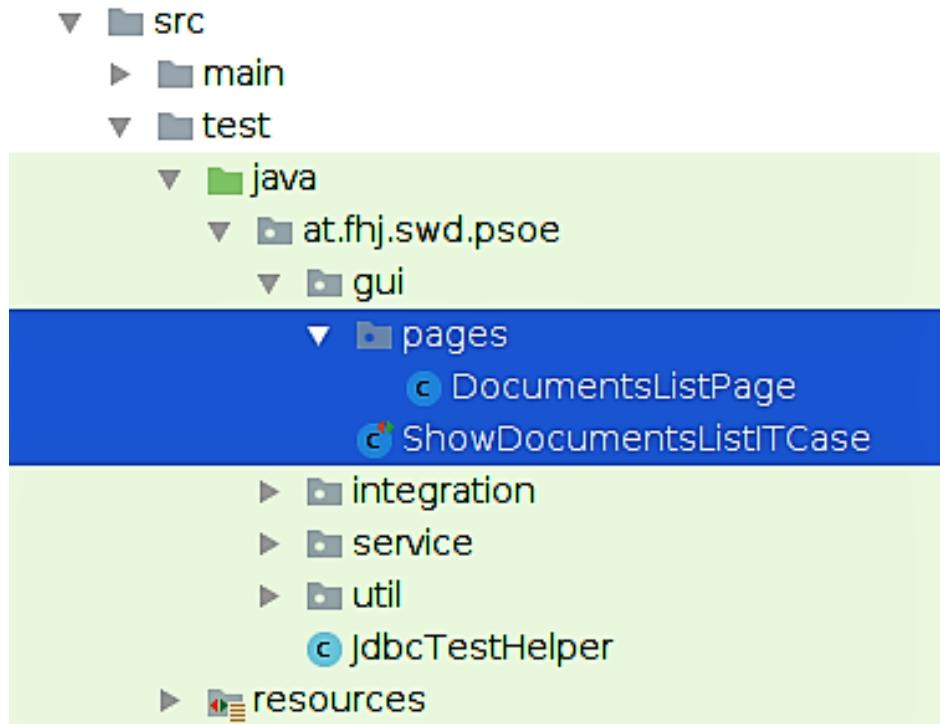
Figure 3

- tatsächlich werden nicht alle Seiten sondern wesentliche Elemente in Objekte gekapselt
- eine HTML Seite wird so mitunter mit mehreren Objekten dargestellt (z.B. Header und Footer Page Object)
- Das Page Objekt Design eignet sich besonders gut um Selenium Tests umzusetzen
- Mittels der Page Objekte kann HTML Code verändert werden (Verkapselung)
- ermöglichen die Modellierung der Benutzeroberfläche für Tests
- reduziert Code duplication
- verbessert Testwartbarkeit und macht Tests robuster

## 18.5 Page Object Pattern lt. Zahnlücke

- Trennung zwischen Testmethode und Page Code

- Je Page eine Klasse mit Services / Operationen
- Return einer Operation ist ein PageObject
- Einfachere Wartbarkeit (Kapselung in PageObject)



## 18.6 Beispiel aus dem Projekt

### 18.6.1 Integration GUI Test mit Selenium

```

1 package at.fhj.swd.psoe.gui.pages;
2
3 import org.openqa.selenium.By;
4 import org.openqa.selenium.WebDriver;
5 import org.openqa.selenium.WebElement;
6 import java.util.List;
7
8 public class DocumentsLoginPage extends AbstractPage {
9     private List<WebElement> list;
10
11     public DocumentsLoginPage(WebDriver driver) {
12         super(driver);
13     }
14     public List<WebElement> getList() {
15         list =
16             this.getDriver().findElements(By.xpath("//*[@id=\"documents:comdoctable_data\"]"));
17         return list;
  
```

```
17 }
18 }
19 }
```

### 18.6.2 Durch Selenium getestetes Page Objekt

```
1 package at.fhj.swd.psoe.gui;
2
3 import at.fhj.swd.psoe.JdbcTestHelper;
4 import at.fhj.swd.psoe.gui.pages.*;
5 import org.junit.After;
6 import org.junit.Assert;
7 import org.junit.Before;
8 import org.junit.Test;
9
10 import java.io.IOException;
11 import java.sql.SQLException;
12 import java.util.stream.Collectors;
13
14 public class ShowDocumentsListITCase extends AbstractChromeTest {
15     final private String roleName = "ADMIN";
16     private LoginPage loginPage;
17     private WelcomePage welcomePage;
18     private DocumentsListPage documentsListPage;
19     private String baseUrl = "http://localhost:8080/chr-krenn-fhj-ws2018-swd16-pse";
20     private static final JdbcTestHelper JDBC_HELPER = new JdbcTestHelper();
21
22     @Before
23     @Override
24     public void setUp() {
25         super.setUp();
26         JDBC_HELPER.executeSqlScript(
27             "src/test/resources/sql/DocumentServiceITCase-addDocument.sql");
28         loginPage = new LoginPage(this.driver, baseUrl, 60);
29         welcomePage = loginPage.login("testdocument@swd.com", "admin");
30         CommunitiesPage communitiesPage = welcomePage.openCommunitiesPage();
31         CommunityPage communityPage = communitiesPage.openCommunityPage();
32         documentsListPage = communityPage.openDocumentListPage();
33     }
34
35     @Test
36     public void testTwoDocumentsListed() {
37         String content = documentsListPage.getList().stream()
38             .map(x -> x.getText())
39             .collect(Collectors.joining());
40         Assert.assertTrue(content.contains("documentuser123"));
41         Assert.assertTrue(content.contains("DocumentITCase1"));
42         Assert.assertTrue(content.contains("DocumentITCase2"));
43     }
44
45     @After
46     @Override
47     public void tearDown() {
48         super.tearDown();
49     }
50 }
```

```

47 JDBC_HELPER.executeSqlScript(
48 "src/test/resources/sql/DocumentServiceITCase-deleteDocument.sql");
49 }
50 }

```

## 19 Toni FRAAGNAA

Den Code durchgehen - was statt null - könnte leere Liste sein, return null sollte aber auch okay sein welche Exception - logger ok? ob ein Throw im try Block ok ist - sollte okay sein, da wenn nicht im try-Block, erfolgt kein Mapping als DAO-Exception.

```

1   @Override
2   public List<Document> findByUser(User user) {
3       logger.debug("dao: documents findByUser");
4       try {
5           if(user == null) throw new IllegalArgumentException("user must not be
6               → null");
6           return entityMangaer.createQuery("from Document d where d.user.userId =
7               → :userid", Document.class).setParameter("userid",
7               → user.getUserId()).getResultList();
8       } catch(NoResultException noresex) {
9           logger.info("dao: findByUser no documents found");
10          return null;
11      }
12      catch (Exception e) {
13          throw new DaoException("error finding documents of user",e);
14      }
15
16 // throw in try - passt
17 @Override
18 public void delete(Document document) {
19     logger.debug("dao: delete document");
20
21     try {
22         if(document == null) throw new IllegalArgumentException("document must
23             → not be null");
23         if(!entityMangaer.contains(document)) {
24             document = entityMangaer.merge(document);
25         }
26         entityMangaer.remove(document);
27     }
28     catch (Exception e) {
29         throw new DaoException("error deleting document",e);
30     }
31 }
32 // ist loggen ohne stacktrace ok? - Stack-Trace gehört eigentlich dazu
33 @Override
34 public TimeRecording getTimeRecordingByTask(Task task) {
35     TimeRecording result = new TimeRecording();
36     try {

```

```

37         result = entityManager.createQuery("select t from TimeRecording t where
38             → t.task.id = :id", TimeRecording.class)
39                 .setParameter("id", task.getId()).getSingleResult();
40     } catch (NoResultException e) {
41         logger.warn("Not Possible to find TimeRecording with task id " +
42             → task.getId());
43     }
44     return result;
45 }
46 // UserDTO - @XmlElement
47 @XmlElement - sichtbar in XmlSerializerTest - für den automatischen Import der
48     → User via XML
49 public void setPhoneNumber(String phoneNumber) {
50     this.phoneNumber = phoneNumber;
51 }
52 // wos is produces
53 @Override
54 @Produces - ist eine Factory-Methode - kommt während dem Deployment und erstellt
55     → das Objekt
56 @Named("userPrincipal")
57 @SessionScoped
58 public UserPrincipal getUserPrincipal() {
59     String principalName = sessionContext.getCallerPrincipal().getName();
60     logger.info(principalName);
61     try {
62         User user = userDao.getByEmail(principalName);
63         return new UserPrincipal(user.getId(), user.getUserId(),
64             → user.getEmail(), user.getFirstname(), user.getLastname());
65     } catch (DaoException e) {
66         logger.error("Error loading user '{}'", principalName, e);
67         throw new ServiceException("Error loading user");
68     } catch (Throwable e) {
69         logger.error("Unknown error loading user '{}'", principalName, e);
70         throw new ServiceException("Unknown error loading user");
71     }
72 // @Typed - zeigt bei Mehrdeutigkeit die Vererbung (z.B. implementiert Interface
73     → und leitet von Klasse ab - muss aber in Klammer immer mitgegeben werden,
74     → wovon Java Server Beans dann die Ableitung macht)
75
76 // warum immer mappedBy Mehrzahl
77 @ManyToMany(mappedBy = "businessTrips") - Name vom Feld in der Entity, die
78     → verbunden wird, da ManyToMany, ist es egal, bei welcher Tabelle
79
80 // dependency Injection - wird schon im Skript erklärt
81
82 // braucht man im Controller (ViewHelper) überhaupt noch Exception Handling - es
83     → passiert nur Nutzereingabenväldierung und die Fehlermeldung kann nicht
84     → weitergeworfen werden - es wird nur mehr in den Logger geschrieben
85
86 // müssen wir die Folien genau beherrschen (Stubs vs. Mocks?) Nein
87
88 // Bei welchem Goal wird was mitausgeführt? IT-Test bei mvn wildfly:run?
89

```

```

80 //Woher weiß PrimeFaces, wie es zum Ordner web mit den Controllern kommt?
81
82 //MessagePrincipal - @Typed - bereits oben erklärt
83
84 //Wie löst Maven Abhängigkeiten zu Libraries auf? Es wird im Maven-Repository
→ nach dem Package gesucht und dort nach der Version, die in der
→ Dependency angegeben wurde. Wird die Library nicht gefunden, muss ein
→ alternatives Repo angegeben werden. WICHTIG: niemals Libraries mit
→ Plugins verwechseln (Plugin ist eine Erweiterung der
→ Maven-Funktionalität, Library ist ein bestehender Java-Code, der
→ verwendet werden kann)

```

## 20 Frageart Prüfung

Welche Fehler können bei Exception-Handling vorkommen in unserem Projekt?? – wie funktioniert es grundsätzlich in unserem Code

DocumentDAO – DocumentService – DocumentController – so sollte Exception-Handling implementiert werden

DAO wirft Exception – im ServiceLayer wird dies gefangen und der Stack-Trace wird im weggeloggt und eine benutzerfreundliche Fehlermeldung wird ausgegeben (Destructive Wrapping).

Alle Patterns, die vorkommen – praktische Beispiele aus dem Code

Was sind JavaBeans? Wie funktioniert das Konzept? Wie wird es genau implementiert? NamedBean, TypedBean etc.