

PSE Ausarbeitung

Phillip Wo
Benjamin Moser
Daniel Sommer

March 5, 2019

Contents

I	Pattern im Projekt	4
1	Layers Pattern	4
1.1	Erkläre die Funktionsweise + Skizze	4
1.1.1	3 Schichten Architektur:	4
2	Data Access Object (DAO) Pattern	5
2.1	Erkläre die Funktion + Skizze	5
2.2	Nenne die Konsequenzen der Anwendung	6
3	Service Layer Pattern (auch Session Fassade - in unserem Projekt im Domain Layer)	6
3.1	Erkläre die Funktion + Skizze	6
3.2	Nenne die Konsequenzen der Anwendung	9
4	Model-View-Controller (MVC) Pattern	9
4.1	Erkläre die Funktion + Skizze	9
4.1.1	Model	10
4.1.2	View	10
4.1.3	Controller	10
5	Front Controller	10
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
6	View Helper (/src/main/java/at/fhj/swd/psoe/web/*)	13
6.1	Erkläre die Funktion + Skizze	13
6.2	Nenne die Konsequenzen der Anwendung	13
7	Dependency Injection (CDI-Framework -> eingebunden im ./pom.xml)	14
7.1	Erkläre die Funktion + Skizze	14
7.2	Nenne die Konsequenzen der Anwendung	15
8	Data Transfer Object (DTO) Pattern	15
8.1	Erkläre die Funktion (Skizze - ein Grund für DTO)	15
8.2	Konsequenzen der Anwendung	17
9	Page-Object-Pattern	17
9.1	Page Object Pattern lt. Zahnlücke	18
9.2	Beispiel aus dem Projekt	18
9.2.1	Integration GUI Test mit Selenium	18
9.2.2	Durch Selenium getestetes Page Objekt	19

II	Exception Handling	21
10	Checked und Runtime Exceptions in Java	21
10.1	Checked Exceptions (z.B. SQL-Exception)	21
10.2	Unchecked Exceptions (z.B. NullPointerException)	21
11	Best Practice Beispiele beim Einsatz von Exceptions	21
12	Exception Handling Anti Pattern	22
13	Destructive Wrapping im Service Layer	22
14	Exception Testing	23
III	Allgemeines & Config	25
15	Logging	25
15.0.1	Vorteile Logging mittels Framework (z.B.: log4j)	25
16	Annotationen	25
16.1	Annotationen - Details	26
17	Konfigurationsdateien	26
17.1	standalone-psoe.xml	26
17.2	persistence.xml	27
17.3	web.xml	28
17.4	pom.xml	30
17.4.1	Aufbau pom.xml	31
18	Frageart Prüfung	31
19	Fehler im Projekt	31
19.1	Return null	31
19.2	Exception nicht gefangen	32
19.3	Destructive Wrapping - Logging fehlt - Information geht verloren	32
19.4	Logger mit falschem Parameter	33
19.5	Fehlendes Exception Handling	33
19.6	Exception werfen und gleich wieder fangen	34
20	Tests	35
20.1	Testpyramide	35
20.2	Unit	35
20.3	Integration	35
20.4	Selenium bzw. GUI	35

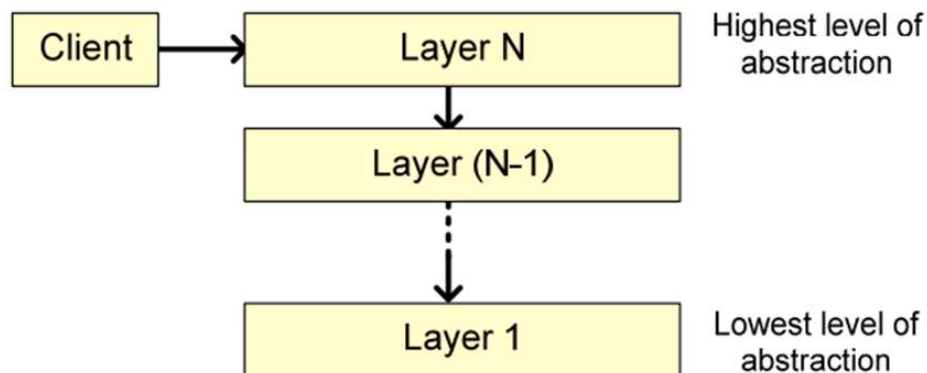
Part I

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:

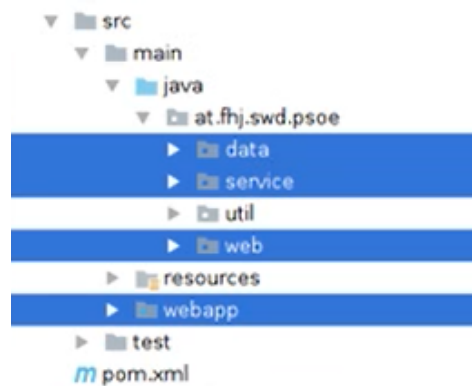
- Data Source Layer (data): Zugriff auf Daten, kümmert sich um Kommunikation mit anderen Systemen (z.B.: Datenbank)
 - enthält Entities -> Java Repräsentation vom DB Entity
 - * im Projekt wurde eine AbstractEntity erstellt, welche die id managed
 - beinhaltet DAO und DAOImpl » DocumentDAO, DocumentlibraryDAO
 - * 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.

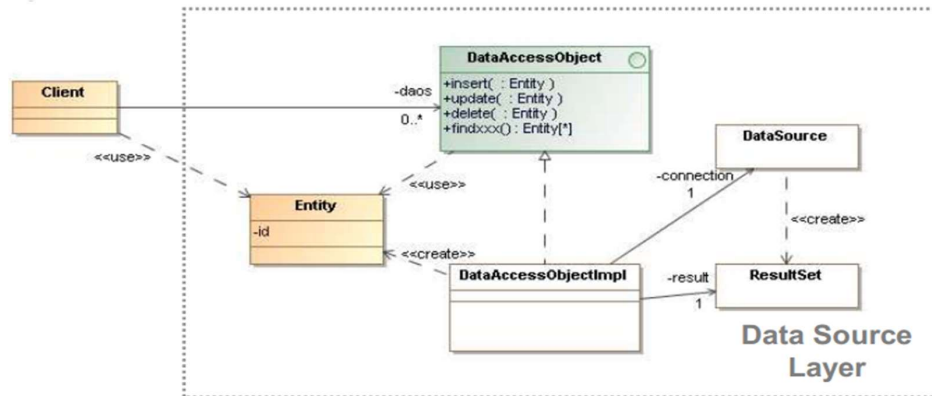
2.1 Erkläre die Funktion + Skizze

- 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

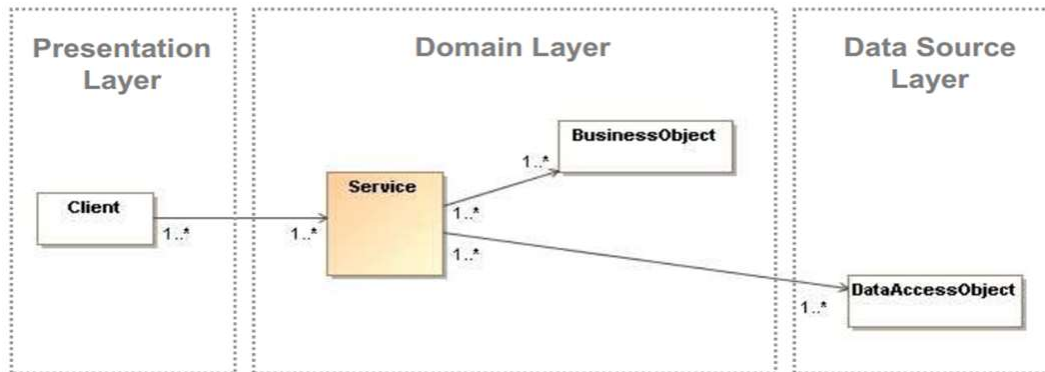
- Zugriff auf persistenten Speicher wird abstrahiert
- Details des Speichers werden versteckt
- ermöglicht einheitlichen Zugriff auf Daten
- entkoppelt Implementierung von Persistierung (Datenbank,...)
- ermöglicht Objektorientierte Ansicht des Speichers

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

```

21     User user;
22
23     // Validierungshandling gefolgt von Error Handling
24     try {
25         if (communityID <= 0) throw new
                ↪ IllegalArgumentException("community must not be empty");
26         if (userID == null) throw new IllegalArgumentException("user must
                ↪ not be empty");
27         if (data == null) throw new IllegalArgumentException("uploaded
                ↪ file must not be empty");
28         if (filename == null) throw new
                ↪ IllegalArgumentException("filename must not be empty");
29
30         Documentlibrary documentlibrary =
                ↪ documentlibraryDAO.findByCommunityId(communityID);
31
32         //create a document library, if there isn't already one in the
                ↪ database
33         if (documentlibrary == null) {
34             documentlibrary = addDocumentlibrary(communityID);
35         }
36
37         user = userDao.getByUserId(userID);
38
39         addedDocument = new Document(documentlibrary, user, filename,
                ↪ data);
40         documentDAO.insert(addedDocument); // Transaktionshandling
41         logger.info(String.format("Document %s saved in database",
                ↪ filename));
42     // Error Handling
43     } catch (IllegalArgumentException iaex) {
44         String errorMsg = "Uploading file failed (illegal argument)";
45         logger.error(errorMsg, iaex);
46         throw new ServiceException(errorMsg);
47
48     } catch (Exception ex) {
49         String errorMsg = String.format("Uploading file %s failed.",
                ↪ filename);
50         logger.error(errorMsg, ex);
51         throw new ServiceException(errorMsg);
52     }
53
54     String msgText = "Uploaded Document " + filename + " by user " +
                ↪ user.getUserId();
55     addMessageToStream(communityID, user, msgText, addedDocument);
56     return DocumentMapper.toDTO(addedDocument);
57 }
58
59
60 private void addMessageToStream(Long communityID, User user, String text,
    ↪ Document document) {...}
61
62 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
        ↪ DAO
68     community.setDocumentlibrary(documentlibrary); // Delegation zur
        ↪ Business Logik (Entity)
69     communityDAO.update(community); // Delegation zum DAO
70     return documentlibrary;
71 }
72
73 @Override
74 public List<DocumentDTO> getDocumentsFromCommunity(Long communityID)
        ↪ {...}
75
76 @Override
77 public List<DocumentDTO> getDocumentsFromUser(String userID) {...}
78
79 @Override
80 public void removeDocument(Long documentID) {...}
81
82 @Override
83 public DocumentDTO getDocumentById(Long documentID) {...}
84 }

```

3.2 Nenne die Konsequenzen der Anwendung

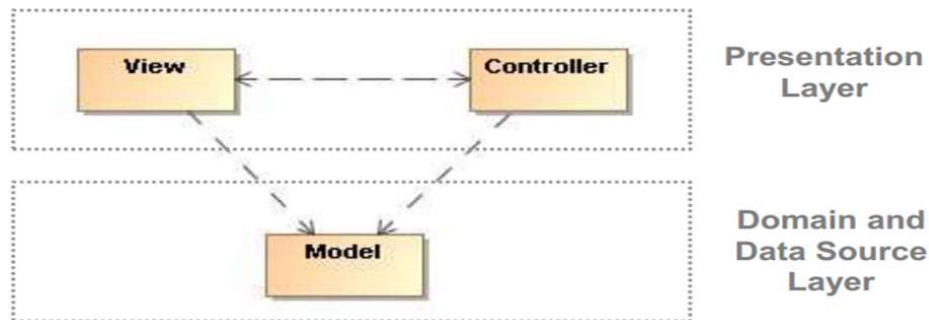
- Reduzierung der Abhängigkeiten zwischen Presentation und Domain Layer
- Zentralisiertes Sicherheits und Transaktionshandling
- verbirgt vor Client Komplexität der Business Logik
- stellt Client ein grobkörniges Interface zur Verfügung
- gut für Remote Aufrufe geeignet (weniger Aufrufe)

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

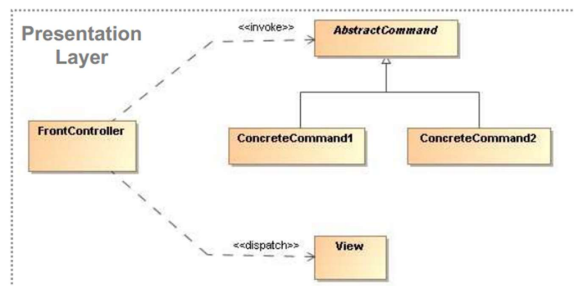
```

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"
19   ↪ actionListener="#{documentListController.loadDocumentsFromCommunity()}"
20   ↪ update="@form doclistform"></p:commandButton>
21 <p:dataTable id="doclisttable" value="#{documentListController.communityDocuments}"
22   ↪ var="docs">
23 <p:column class="documenttimecolumn"
24   ↪ headerText="#{msg.document_uploaded}">#{docs.createdTimestamp}</p:column>
25 <p:column class="documenttimecolumn"
26   ↪ headerText="#{msg.label_userid}">#{docs.user.userId}</p:column>
27 <p:column headerText="#{msg.label_filename}">#{docs.filename}</p:column>
28 <p:column headerText="" class="documentbuttoncolumn">
29 <p:commandButton value="#{msg.button_download}" ajax="false"
30   ↪ onclick="PrimeFaces.monitorDownload(start, stop);">
31 <p:fileDownload value="#{documentController.downloadDocument(docs.id)}/>
32 </p:commandButton>
33 </p:column>
34 <p:column headerText="" class="documentbuttoncolumn">
35 <p:commandButton id="btnDel" value="#{msg.button_delete}"
36   ↪ actionListener="#{documentController.removeDocument(docs.id)}"
37   ↪ update="@form doclistform">
38 </p:commandButton>
39 </p:column>
40 </p:dataTable>
41 </h:form>
42 <h:form id="formdocupload" enctype="multipart/form-data">
43 <p:fileUpload id="fileupload"
44   ↪ dragDropSupport="false"
45   ↪ update="@form doclistform"
46   ↪ fileUploadListener="#{documentController.uploadDocument}"
47   ↪ allowTypes="/(\\.|\\/) (pdf|jpe?g|docx)\\$/" sizeLimit="5000000"
48   ↪ mode="advanced" label="Add document (.pdf .jpg .docx)">
49 </p:fileUpload>
50 </h:form>
51 <p:messages id="feedbackBox" severity="info,error" showDetail="true"
52   ↪ showSummary="false">
53 <p:autoUpdate/>
54 </p:messages>
55 </ui:define>
56 </ui:composition>

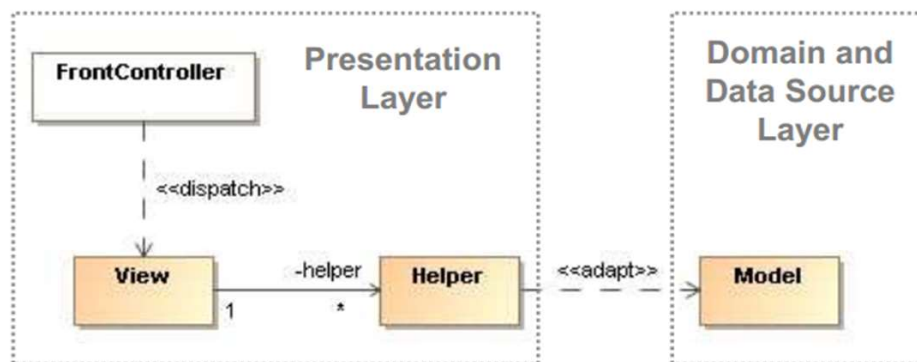
```

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



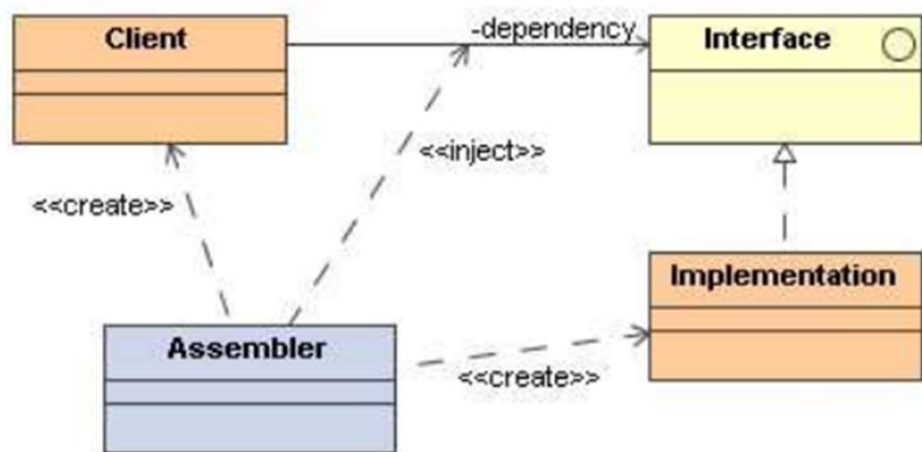
- View (xhtml-Dateien im Ordner */src/main/webapp/**) delegiert Aufgaben an Helper (z.B. DocumentController im Ordner web)
- Helper adaptieren View zu Model (Klassen in den Ordnern *src/main/java/at/fhj/swd/psoe/service/** und *src/main/java/at/fhj/swd/psoe/data/**)
- in View befindet sich HTML Code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



- Grundidee sind loose gekoppelte Objekte
- Objekte werden mittels externem Assembler verknüpft
- Abhängigkeiten bestehen nur auf Interfaces
- Assembler Objekt (Framework) erzeugt die Interface-Implementierungen (z.B.: durch Factory)
- 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 setIface(Interface iface)
12         {
13             this.iface = iface;
14         }
15     }
```

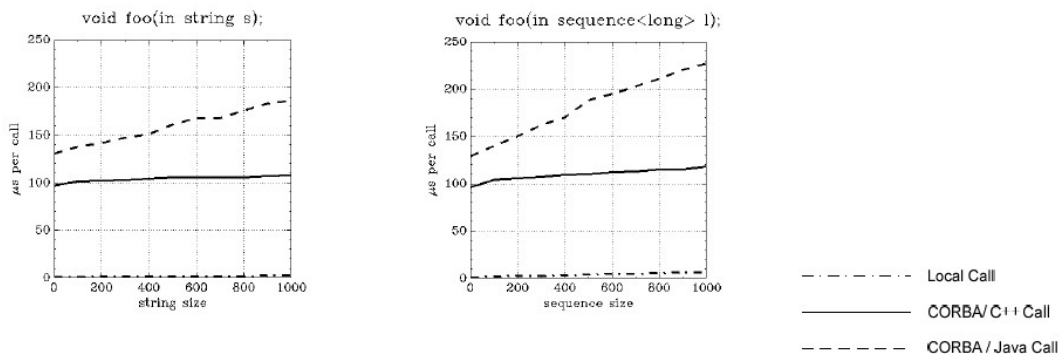
- Im Spring Context:
 - Dependency Injection mit XML-Datei
 - alle Beans sind dort gelistet und werden verknüpft
 - Context wird geladen damit alles verknüpft ist
 - erspart Factories

7.2 Nenne die Konsequenzen der Anwendung

- loose gekoppelte Objekte
- Referenzen nur noch auf Interfaces
- hohe Flexibilität (Strategy, Proxy,..)
- bessere Erweiterbarkeit und Testbarkeit
- bei Spring kann Dependency Injection mittels XML oder Annotation erfolgen
 - Vorteil Annotation: Typ-Sicherheit (Tippfehler passieren schnell im XML)
 - Nachteil Annotation: nicht so flexibel wie XML

8 Data Transfer Object (DTO) Pattern

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



- Transportiert Daten zwischen Prozessen um Remote Methodenaufrufe 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

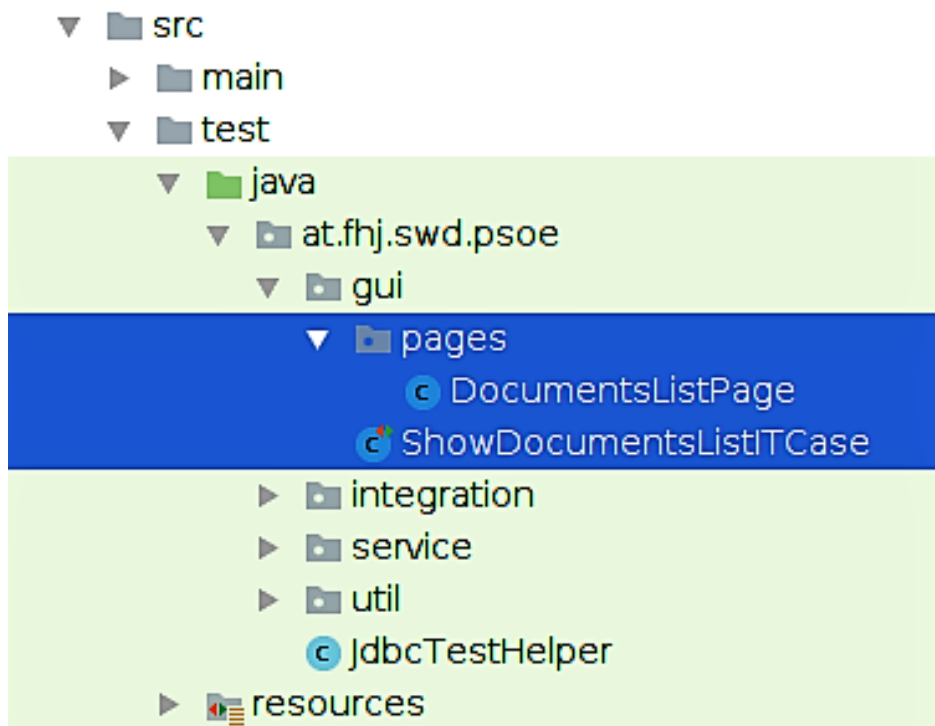
- kapselt und versteckt
- nimmt Komplexität
- steigert Effizienz da weniger Aufrufe über Remotegrenze

9 Page-Object-Pattern

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

9.1 Page Object Pattern lt. Zahnücke

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



9.2 Beispiel aus dem Projekt

9.2.1 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 DocumentsListPage extends AbstractPage {
9     private List<WebElement> list;
10
11     public DocumentsListPage(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;
18 }
19 }

```

9.2.2 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 =
20            ↪ "http://localhost:8080/chr-krenn-fhj-ws2018-swd16-pse";
21        private static final JdbcTestHelper JDBC_HELPER = new JdbcTestHelper();
22
23        @Before
24        @Override
25        public void setUp() {
26            super.setUp();
27            JDBC_HELPER.executeSqlScript(
28                ↪ "src/test/resources/sql/DocumentServiceITCase-addDocument.sql");
29            loginPage = new LoginPage(this.driver, baseUrl, 60);
30            welcomePage = loginPage.login("testdocument@swd.com", "admin");
31            CommunitiesPage communitiesPage =
32                ↪ welcomePage.openCommunitiesPage();
33            CommunityPage communityPage =
34                ↪ communitiesPage.openCommunityPage();
35            documentsListPage = communityPage.openDocumentListPage();
36        }
37
38        @Test
39        public void testTwoDocumentsListed() {
40            String content = documentsListPage.getList().stream().map(x
41                ↪ -> x.getText()).collect(Collectors.joining());
42            Assert.assertTrue(content.contains("documentuser123"));

```

```
39         Assert.assertTrue(content.contains("DocumentITCase1"));
40         Assert.assertTrue(content.contains("DocumentITCase2"));
41     }
42
43     @After
44     @Override
45     public void tearDown() {
46         super.tearDown();
47         JDBC_HELPER.executeSqlScript(
48
49 ↪ "src/test/resources/sql/DocumentServiceITCase-deleteDocument.sql");
50     }
```

Part II

Exception Handling

10 Checked und Runtime Exceptions in Java

10.1 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)

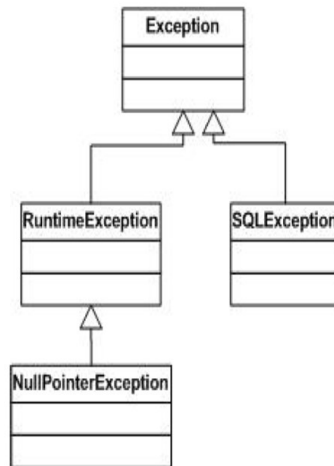
10.2 Unchecked Exceptions (z.B. NullPointerException)

- 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);}}
```

11 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

12 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 (proviziert an einer anderen Stelle eine NullPointerException)
- Catch and Ignore
- Unsupported Operation return Null (besser UnsupportedOperationException)

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

```

14 Exception Testing

```

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

```

27

}

28

...

Part III

Allgemeines & Config

15 Logging

15.0.1 Vorteile Logging mittels Framework (z.B.: log4j)

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

16 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
- @Stateless
 - * speichert den Client Status nicht
- @Entity
 - * Data Access Layer
- @Table
 - * Tabellenname im SQL
- @Column
 - * SQL-Spalten nullable=false
- @OneToMany
- @JoinColumn
 - * welche Spalten zusammen gehören FK

- @OneToMany
 - * auf anderen Seite
- @ApplicationScoped
 - * lebt die ganze Applikation lang, wird einmal gemacht.
- @PersistenceContext
 - * persistence.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.

16.1 Annotationen - Details

```

1 CascadeType anschauen
2
3 @Entity
4 @Table(name = "user")
5 public class User extends AbstractEntity {
6     private static final long serialVersionUID = -7060150053795176748L;
7
8     @OneToMany(
9         mappedBy = "communityAdminUser",
10        cascade = {CascadeType.PERSIST, CascadeType.REFRESH}, // Persist Operation des
           ↪ User werden auch auf dessen communityAdminUser persistiert - und auch
           ↪ update
11        orphanRemoval = true //
12    )

```

17 Konfigurationsdateien

17.1 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 1: 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"/>
29 </outbound-socket-binding>
30 </socket-binding-group>
31 </server>
```

17.2 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

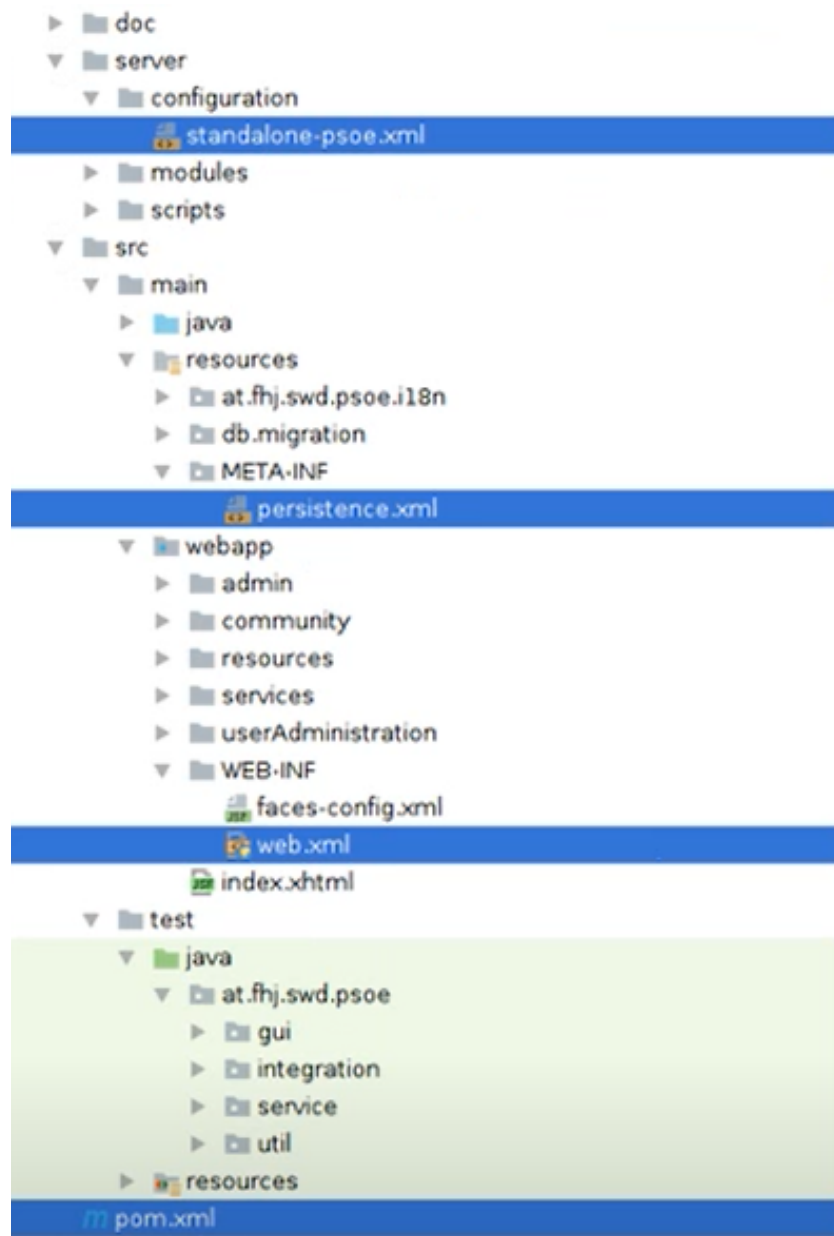
Listing 2: persistence.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence version="2.0"
3 xmlns="http://java.sun.com/xml/ns/persistence"
  ↪ xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4 xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  ↪ http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
5
6 <persistence-unit name="primary">
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 <property name="hibernate.enable_lazy_load_no_trans" value="true" />
12 <!--
13 SQL stdout logging
14 -->
15 <property name="hibernate.show_sql" value="true"/>
16 <property name="hibernate.format_sql" value="false"/>
17 <property name="use_sql_comments" value="true"/>
18
19 <!--
20 <property name="hibernate.hbm2ddl.auto" value="create-drop" />
21 -->
22 </properties>
23 </persistence-unit>
24
25 </persistence>
```

17.3 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"
  ↪ 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>
13
14 <!-- Security roles -->
15 <security-role>
16 <description>administrators</description>
17 <role-name>ADMIN</role-name>
18 </security-role>
19
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>

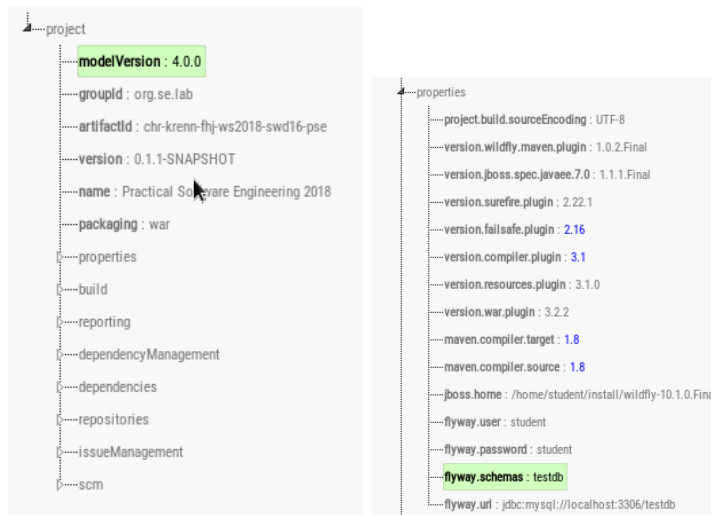
```

17.4 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 dependency injection

17.4.1 Aufbau pom.xml



18 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 weggeklappt 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.

19 Fehler im Projekt

19.1 Return null

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

19.2 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    }
```

19.3 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
5     ↪ communityName, boolean isEnabled) {
6         String errorText = "";
7         try {
8             boolean hasPermission = false;
9             User adminUser = userDao.getByUserId(adminUserId);
10            Community community = communityDAO.getByName(communityName);
11
12            for (Role r : adminUser.getRoles()) {
13                if (r.getName().equals("ADMIN") ||
14                ↪ r.getName().equals("PORTALADMIN")) {
15                    hasPermission = true;
16                }
17            }
18            if (hasPermission ||
19            ↪ adminUser.getUserId().equals(community.getCommunityAdminUser().getUserId()))
20            ↪ {
21                community.setIsEnabled(isEnabled);
22                communityDAO.update(community);
23            } else {
24                errorText = "No Permission to update community";
25                throw new AuthenticationException(errorText);
26            }
27            return CommunityMapper.toDTO(community);
28        } catch (DaoException e) {
29            errorText = "Error updating community enabled";
30            throw new ServiceException(errorText);
31        } catch (AuthenticationException e) {
32            throw new ServiceException(errorText);
33        }
34    }
```



```

30     } catch (Throwable e) {
31         errorText = "Unknown error updating community enabled";
32         throw new ServiceException(errorText);
33     }
34 }

```

19.4 Logger mit falschem Parameter

```

1     package at.fhj.swd.psoe.service.impl;
2     ...
3     @Stateless
4     public class DepartmentHierarchyServiceImpl implements
        ↳ DepartmentHierarchyService, Serializable {
5
6         private static final long serialVersionUID = -2467949382018996094L;
7         private static final Logger logger =
            ↳ LoggerFactory.getLogger(UserServiceImpl.class);
8     ...

```

19.5 Fehlendes Exception Handling

```

1     package at.fhj.swd.psoe.service.impl;
2     ...
3     @Local(MessageService.class)
4     @Remote(MessageServiceRemote.class)
5     @Stateless
6     public class MessageServiceImpl implements MessageService,
        ↳ MessageServiceRemote, Serializable {
7         ...
8     @Override
9         public MessageDTO getByMessageId(long id) {
10            Message message = messageDAO.getById(id);
11            if (message == null) {
12                return null;
13            }
14            return MessageMapper.toDTO(message);
15        }
16
17     @Override
18     public MessageDTO updateByAdmin(long id, String content, Date changed) {
19         Message message = messageDAO.getById(id);
20         message.setContent(content);
21         message.setEditedByAdmin(changed);
22         return MessageMapper.toDTO(messageDAO.update(message));
23     }
24
25     @Override
26     public MessageDTO updateByUser(long messageId, String content, Date
        ↳ changed) {
27         Message message = messageDAO.getById(messageId);
28         message.setContent(content);

```

```

29     message.setEditedByUser(changed);
30     return IMapper.toDTO(messageDAO.update(message));
31 }

```

19.6 Exception werfen und gleich wieder fangen

```

1     ...
2 public class MessageServiceImpl implements MessageService,
   ↳ MessageServiceRemote, Serializable {
3     private static final long serialVersionUID = 6768291437557855130L;
4     ...
5     // nicht optimal, da die IllegalArgumentException gleich wieder gefangen wird
6     // überdies wird alles andere nicht gefangen
7
8
9     @Override
10    public void deleteMessage(long id) {
11        try {
12            Message message = messageDAO.getById(id);
13            if (message == null) {
14                throw new IllegalArgumentException("Message cannot be empty");
15            }
16            messageDAO.delete(message);
17            logger.info("Message deleted successfully");
18        }
19        catch (IllegalArgumentException ex) {
20            String errorMsg = "Could not delete the message (illegal
   ↳ argument)";
21            logger.error(errorMsg, ex);
22            throw new ServiceException(errorMsg);
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 catch blocks
31    // erst wird die DaoException gefangen und anschließend alle anderen
32    // Stacktrace wird geloggt und jeweils die ServiceException weitergegeben
33    @Override
34    public void saveUser(UserDTO userDTO) {
35        logger.debug("UserService saveUser() called with parameter: '{}'",
   ↳ userDTO);
36
37        if (userDTO == null) {
38            throw new IllegalArgumentException("userDTO is null");
39        }
40
41        try {

```

```

42     User user = (userDTO.getId() == null) ? new User() :
        ↳ userDao.getById(userDTO.getId());
43     userDao.update(UserMapper.toEntity(userDTO, user));
44 } catch (DaoException e) {
45     logger.error("Error saving user", e);
46     throw new ServiceException("Error saving user");
47 } catch (Throwable e) {
48     logger.error("Unknown error saving user", e);
49     throw new ServiceException("Unknown error saving user");
50 }
51 }

```

20 Tests

20.1 Testpyramide

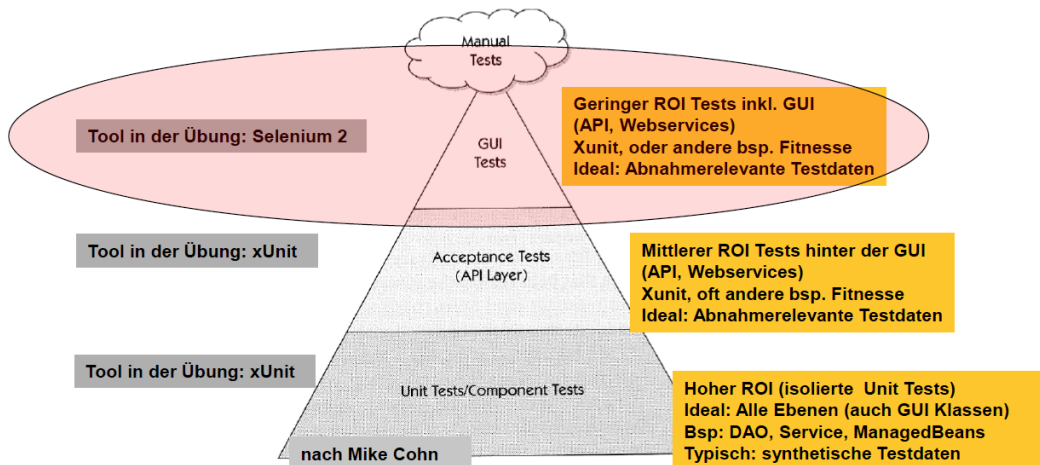


Figure 1

20.2 Unit

20.3 Integration

20.4 Selenium bzw. GUI