Feinspezifikation

DocThesisTracker

Client: Dr. Armin Größlinger

Team 2

Hadi Abou Hassoun
Alaa Qasem
Johannes Silvennoinen
Stefan Witka

Inhaltsverzeichnis

1 Einleitung

Stefan Witka

Dieses Dokument erweitert den ersten Entwurf des Online Umlaufprogramms für Dissertationen DocThesisTracker.

2 Systemarchitektur

Johannes Silvennoinen

2.1 Architektur-Überblick

Unsere Architektur basiert auf dem Model-View-Controller (MVC) Pattern, das eine Trennung von Anwendungslogik, Daten und Darstellung ermöglicht. Die Model-Schicht enthält die Geschäftslogik und den Datenzugriff, während die View-Schicht für die Präsentation der Benutzeroberfläche zuständig ist. Der Controller verbindet die beiden Schichten und koordiniert die Aktionen des Benutzers.

Um eine effiziente und sichere Verbindung zur Datenbank herzustellen, verwenden wir einen Connection Pool als Singleton. Dies stellt sicher, dass mehrere Verbindungen zur Datenbank gleichzeitig genutzt werden können, ohne dass es zu Konflikten kommt. Der Pool wird auch verwendet, um Verbindungsfehler zu vermeiden und die Ressourcen effektiv zu nutzen.

Um Daten zwischen den Schichten zu übertragen, verwenden wir Data Transfer Objects (DTOs). Diese sind einfache POJOs (Plain Old Java Objects), die nur Daten und keine Geschäftslogik enthalten. Sie dienen als Schnittstelle zwischen der Model-Schicht und der View-Schicht und erleichtern die Datenübertragung und -manipulation.

2.2 Paketaufteilung

Die gesamte Anwendung ist in drei Hauptpakete unterteilt: View, Controller und Model.

Im View-Paket befinden sich alle XHTML-Seiten, die der Benutzer sehen und mit denen er interagieren kann. Diese Seiten sind für die Darstellung von Informationen und die Benutzereingabe verantwortlich.

Der Controller wird von Jakarta Server Faces (JSF) verwaltet und ist für die Steuerung der Interaktionen zwischen der View und dem Model zuständig.

Das Model-Paket ist in zwei Schichten unterteilt: Business Layer und Data Access Layer. Der Business Layer enthält die Backing Beans und ist für die Implementierung der Geschäftslogik zuständig. Diese Schicht enthält auch Validatoren

für die Eingabevalidierung und die Fehlerbehandlung.

Der Data Access Layer ist für die Datenbankanbindung zuständig und ist in Unterpakete für jede unterstützte Datenbankplattform unterteilt. Das PostgreSQL-Unterpaket enthält alle Klassen und Schnittstellen, die für die Interaktion mit einer PostgreSQL-Datenbank erforderlich sind. Es ist jedoch möglich, in Zukunft Unterpakete für andere Datenbankplattformen hinzuzufügen, um die Kompatibilität der Anwendung zu erweitern.

Durch diese Paketaufteilung wird eine klare Trennung von Aufgaben und Verantwortlichkeiten erreicht, was die Wartbarkeit und Erweiterbarkeit der Anwendung verbessert.

2.3 Datenübertragung

Zur Übertragung von Daten zwischen den Schichten werden Data Transfer Objects (DTOs) genutzt. Diese sind einfache POJOs (Plain Old Java Objects), die keine JEE- oder JPA-Managed Beans sind und außerhalb der Data Access Schicht und Business Schicht liegen.

2.4 Verbindungspool

Für die Verbindung zur Datenbank wird ein Connection Pool als Singleton (threadsicher) verwendet. Dadurch wird sichergestellt, dass die Verbindung zur Datenbank effizient genutzt wird und keine unnötigen Verbindungen aufgebaut werden.

2.5 Fehlerbehandlung

Die Fehlerbehandlung in der Systemarchitektur wird mithilfe von Validatoren implementiert. Dabei wird für jedes Eingabeformular ein eigener Validator erstellt, der die Eingaben des Nutzers auf Plausibilität und Vollständigkeit prüft. Falls ein Fehler auftritt, wird dem Nutzer eine entsprechende Fehlermeldung angezeigt und er wird aufgefordert, die fehlerhaften Eingaben zu korrigieren.

Die Validatoren sind in den Backing Beans für den Business Code integriert. Diese Beans sind für die Verarbeitung der Geschäftslogik und der Nutzerinteraktion zuständig. Wenn eine Eingabe vom Nutzer empfangen wird, ruft das entsprechende Backing Bean den passenden Validator auf.

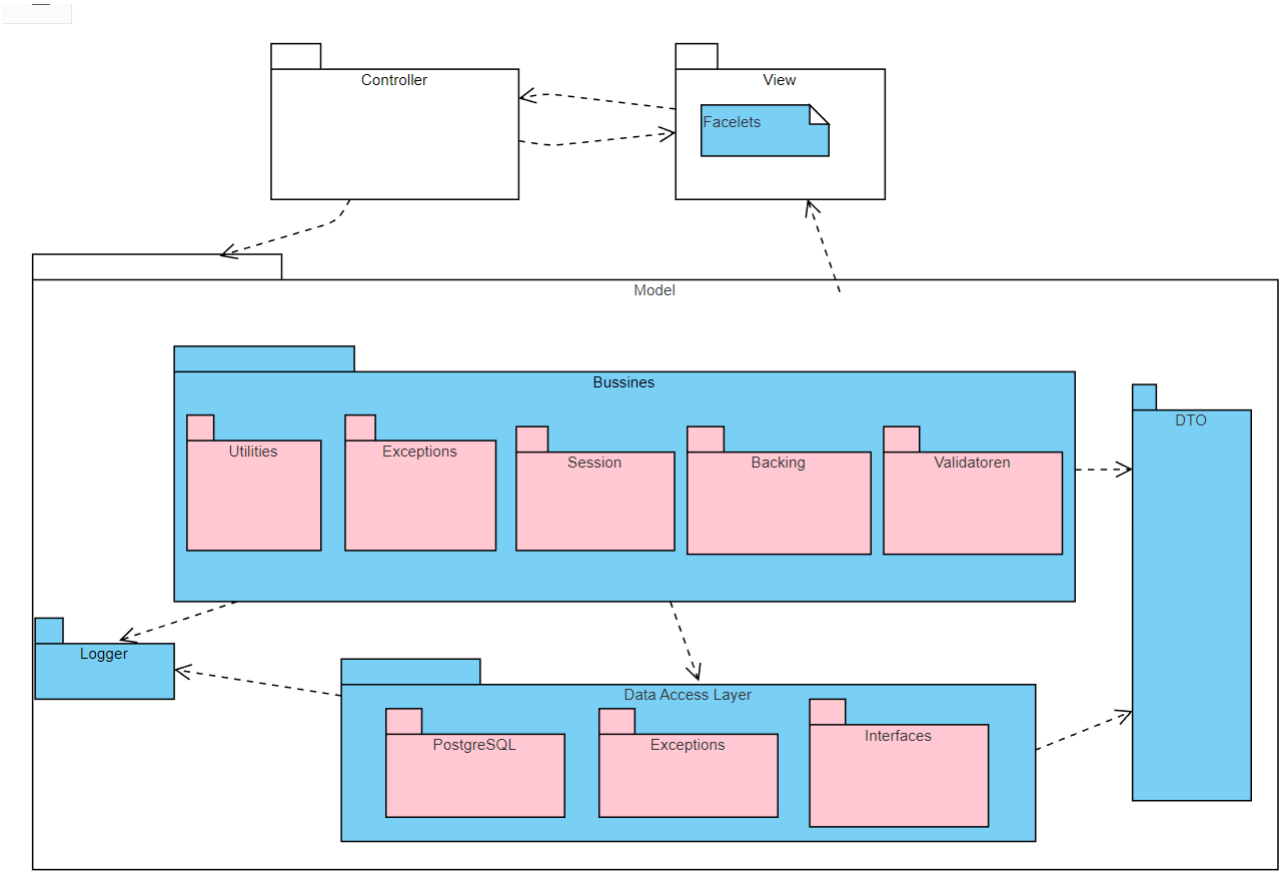
Die Validatoren werden auch für die Datenüberprüfung verwendet, bevor sie in die Datenbank eingefügt werden. Dadurch wird sichergestellt, dass die Daten in

der richtigen Form vorliegen und keine Datenbankfehler auftreten.

Zusätzlich werden auch systemweite Fehlerbehandlungen implementiert, um unerwartete Fehler abzufangen und dem Nutzer eine angemessene Fehlermeldung anzuzeigen. Dabei wird ein Globaler-Fehler-Handler eingesetzt, der alle unerwarteten Fehler abfängt und eine Fehlerseite mit einer entsprechenden Fehlermeldung anzeigt. Die Fehlermeldungen sind dabei für den Nutzer verständlich formuliert und geben Hinweise auf die Ursache des Fehlers.

Insgesamt gewährleistet die Verwendung von Validatoren eine zuverlässige und benutzerfreundliche Fehlerbehandlung in der Systemarchitektur.

2.6 Diagramme



2.7 Pakete

Hadi Abou Hassoun

2.7.1 dtt.business

Dieses Paket enthält alle Klassen der Anwendungslogikschicht.

dtt.business.backing Dieses Paket enthält alle Managed Beans, die verwendet werden, um die Facelets in der View darzustellen.

dtt.business.validation Dieses Paket umfasst alle Klassen, die speziell für die Validierung von Benutzereingaben in Facelets entwickelt wurden.

dtt.business.exception Dieses Paket enthält eine Reihe von Exception-Klassen, die spezifische Arten von Fehlverhalten im System repräsentieren. Jede Exception-Klasse ist darauf ausgerichtet, einen bestimmten Fehlerfall abzubilden. Sie dienen dazu, ungewöhnliche oder fehlerhafte Zustände im System zu signalisieren und eine entsprechende Reaktion oder Behandlung zu ermöglichen.

dtt.business.utilities Dieses Paket enthält Hilfsklassen, die Dienste zur Verfügung stellen, welche unabhängig von der Anwendungslogik genutzt werden können

2.7.2 dtt.dataAccess

Dieses Paket enthält alle Klassen der Persistenzschicht.

dtt.dataAccess.Repository Dieses Paket enthält alle Klassen, die speziell für den direkten Zugriff auf die Datenbank entwickelt wurden. Diese Klassen stellen Methoden und Funktionen bereit, um Datenbankoperationen durchzuführen.

dtt.dataAccess.utilities Dieses Paket enthält Hilfsklassen, die zur Datenbankverwaltung dienen.

2.7.3 dtt.global

Dieses Paket enthält eine Sammlung von Klassen, die über mehrere Schichten hinweg verwendet werden.

dtt.global.transport Dieses Paket enthält alle POJO-Klassen, die Daten in der Anwendung repräsentieren und für deren Transport verwendet werden. Diese Klassen dienen als einfache Datenhalter und enthalten normalerweise private Felder mit entsprechenden Getter- und Setter-Methoden. Sie enthalten keine Anwendungslogik oder komplexe Verarbeitungslogik, sondern dienen ausschließlich der Strukturierung und Organisation von Daten.

2.8 Ordnerstruktur der Entwicklungsumgebung

Wir liegen hier die Organisation und Platzierung von Dateien fest, um den Entwicklungsprozess zu strukturieren

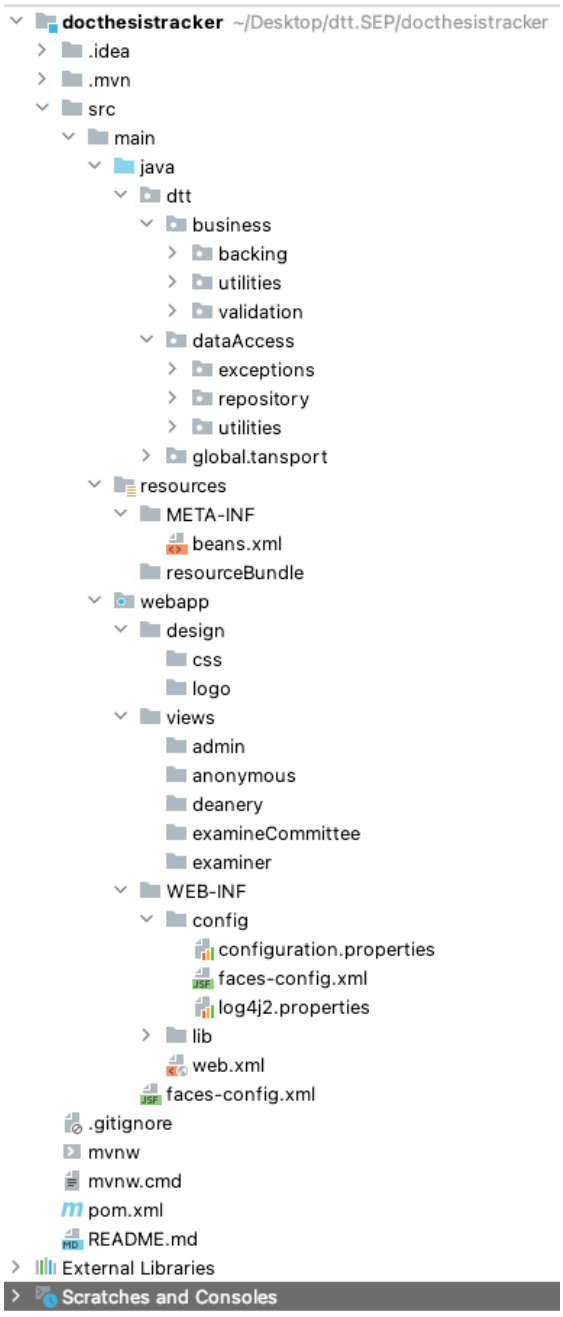


Abbildung 1: Pakete die DocTheisesTracker verwendet

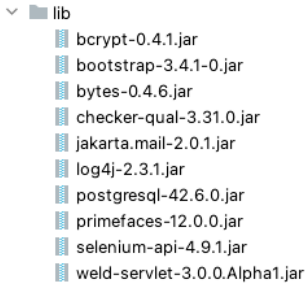


Abbildung 2: Bibliotheken

2.9 Fazit

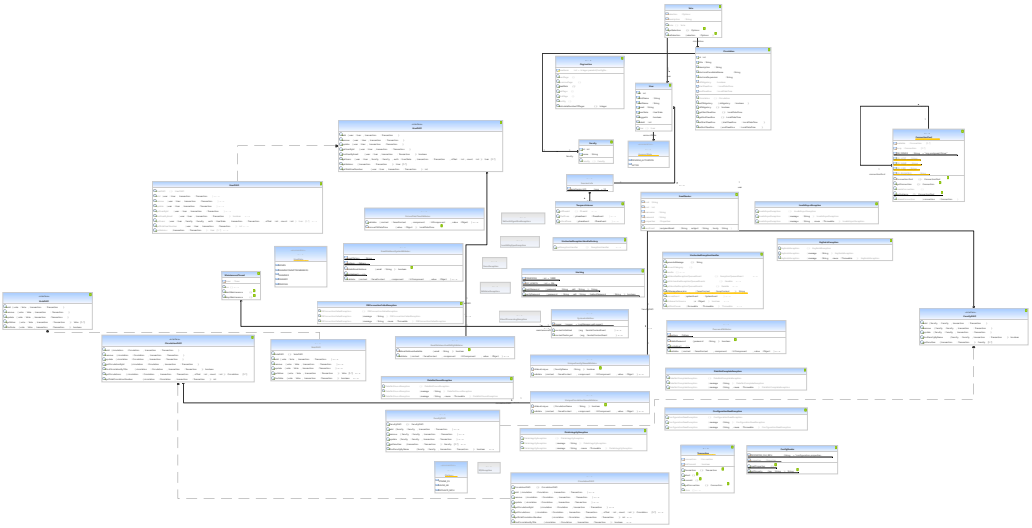
Johannes Silvennoinen

Die Systemarchitektur basiert auf bewährten Entwurfsmustern und ermöglicht eine effiziente und flexible Entwicklung. Die klare Trennung von Aufgaben und die Verwendung von DTOs und DAOs erleichtert die Wartung und Erweiterung des Systems.

3 Statisches Klassendiagramm in UML

Johannes Silvennoinen

Das UML diagramm generiert von Yatta UML Lab.



4 Bibliotheken

Hadi Abou Hassoun

Um die Entwicklung und Verwendung von DocTheisisTracker durchzuführen, sind bestimmte Softwarebibliotheken erforderlich. Hier ist ein Überblick über die Bibliotheken und ihre Versionen, die wir verwenden werden

Name	Beschreibung	version	Lizenz
Mojarra	Ist eine JavaServer Faces (JSF)-Implementierung, die ein serverseitiges Framework für den Aufbau von komponentenbasierten Webanwendungen in Java bereitstellt	v3.0.3	EPL v2.0
PostreSQL JDBC	Der PostreSQL JDBC-Treiber ermöglicht die Verbindung und Interaktion zwischen einer Java-Anwendung und einer PostgreSQL-Datenbank.	v42.6.0	BSD 2-clause
JBoss Weld	JBoss Weld ist ein CDI (Contexts and Dependency Injection)-Implementierungsframework für Java EE.	v3.0.0	Apache 2.0
Primefaces	Bibliothek von JSF-Komponenten und Erweiterungen für die Erstellung ansprechender Benutzeroberflächen.	v12.0.0	Apache 2.0
Jakarta-mail	Automatisierte E-Mail Versendung	v2.0.1	EPL v2.0
Log4j	Ist eine leistungsstarke und flexible Java-Logging-Bibliothek, dabei hilft, Protokollnachrichten zu generieren, zu verwalten und auszugeben.	v2.3.1	Apache 2.0
Bcrypt	Eine Bcrypt-Bibliothek ist eine Softwarebibliothek, die Funktionen und Methoden zur Verwendung des Bcrypt-Hashing-Algorithmus bereitstellt. Diese Bibliotheken ermöglichen die einfache Integration von Bcrypt in Anwendungen und erleichtern die sichere Speicherung von Passwörtern oder anderen sensiblen Daten.	v4.1.1	Apache 2.0
Bootstrap	CSS Framework	v3.4.1	MIT

4.1 Bibliotheken für die Testphase:

Name	Beschreibung	version	Lizenz
JUnit	JUnit ist ein Java-Testframework	V5.9.1	EPL v2.0
Selenium	Selenium ist ein Open-Source-Framework für die Automatisierung von Webanwendungen, es ermöglicht, Aktionen in einem Webbrowser zu simulieren und Tests für Webanwendungen durchzuführen.	V4.9.1	Apache 2.0

5 JF-Seiten

Alaa Qasem

Im Folgendem Abschnitt werden Abkürzungen für die verschiedenen Rollen eingeführt: **A** steht für Administrator, **D** für Dekanat, **PK** für einen Prüfungskommissionsmitglieder , **P** für Prüfungsberechtigte und **AN** für anonymen Nutzer. Ist eine Funktion für alle Benutzerrollen ,so werden diese Außer anonymen Nutzer unter dem Begriff **Alle** zusammengefasst.

5.1 Namenskonvention

Hadi Abou Hassoun

Unsere Namenskonvention für Komponenten in den Facelets folgt dem kebab-case-Prinzip, bei dem die Wörter in Kleinbuchstaben geschrieben und durch Bindestriche getrennt werden. Um den genauen Typ der Komponente in der ID zu kennzeichnen, fügen wir ein entsprechendes Suffix hinzu. Auf diese Weise erhalten die Komponenten IDs, die ihren Typ eindeutig anzeigen.

Typ	**Suffix**
link	link
inputtext	itxt
output	otxt
inputfile	ifile
datatable	dtbl
graphicImage	gimg
commandButton	cbtn
message	msg
selectOneMenu	slctom
inputSecret	iscrt

5.2 Login-Seite

Alaa Qasem

login.xhtml Auf der Login-Seite wird das System vorgestellt. Es gibt ein Login-Formular zur Anmeldung im System. Für nicht registrierte Nutzer wird man über einen Button zu registration.xhtml weitergeleitet.

login.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
loginHead-otxt	<h:outputText>	value="#{loginBacking. userDTO.firstName}"	Überschrift der Login-Seite.	/
email-itxt	<h:inputText>	value="#{loginBacking. userDTO.email}"	Hier kann der Benutzer seine Email eingeben.	EmailAddress AvailbilityValidator ,required=true"
password-iscrt	<h:inputSecret>	value ="#{loginBacking.userDTO.passwordSalt}"	Hier kann der Benutzer seine Passwort eingeben.	password Validator
forgetPass-link	<h:link>	value="Go to forget-password Page" outcome=/view/forgetPass"	Weiterletiung zur PasswortSetzen-Seite	/
login-cbtn	<h:commandButton>	action ="#{loginBacking.login}.	Ausführen des Login-Prozesses.	/
register-cbtn	<h:commandButton>	action ="#{loginBacking.goToRegister}	Weitergeleitung zur Registrierung-Seite.	/

5.3 PasswortVergessen-Seite

forgetPass.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
forgetPass-head-itxt	<h:outputText>	value="#{resetPasswordBacking.}"	Überschrift der forgetPass-Seite.	/
email-itxt	<h:inputText>	value ="#{resetPassword-Backing.email}	Angabe der Email.	EmailAddress AvailbilityValidator ,required=true"
forgetPass-cbtn	<h:commandButton>	action ="#{reset-Backing.sendResetPasswordEmail}	wird der Prozess zum Zurücksetzen des Passworts gestartet	/

5.4 reset-Seite

setNew.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
password-iscrt	<h:inputSecret>	action = "#{setNewPassword-Backing. password}"	Eingabe des neuen Passworts des Benutzers.	passwordValidator lenght mustBe at least 8 characters long. and contain at least one digit ,one lowercase letter, one uppercase letter, one special character .
savePass-cbtn	<h:commandButton>	action ="#{setNewPassword-Backing. save}"	Speichert das neue Passwort	/
succsess-faildMessage-msg	<h:message>	/	Ob das Password erfolgreich geändert	/
loginPage-link	<h:link>	value="Go to login Page" outcome=/view/anonymous/login"	Weiterleitung zur Login Seite.	/

5.5 Registrierung-Seite

registration.xhtml Seite zur Registrierung anonymer Nutzer.

registration.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
firstname-itxt	<h:inputText>	value="#{RegistrationBacking. userDTO.firstName}."	Angabe des Vornamens	/
name-itxt	<h:inputText>	value="#{RegistrationBacking. userDTO.lastName}."	Nachname eingeben.	/
email-itxt	<h:inputText>	value="#{RegistrationBacking. userDTO.email}."	Email eingeben.	EmailAddress AvailbilityValidator
faculty-slctom	<h:selectOneMenu>	value="#{RegistrationBacking. userDTO.faculty}."	Fakultät auswählen.	/
register-cbtn	<h:commandButton>	action="#{RegistrationBacking. register}."	wird der Registrierungsprozess ausgeführt.	/
loginPage-link	<h:link >	value="Go to login Page" outcome=/view/anonymous/login"	Weiterleitung zur Login Seite.	/

5.6 Profile

profile.xhtml Die Profil-Seite eines angemeldeten Nutzers kann nur vom Nutzer selbst oder vom Administrator bearbeitet werden. Alle zuvor gespeicherten Daten werden in den entsprechenden Feldern angezeigt.

profile.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
firstname-itxt	<h:inputText>	value="#{ProfileBacking. userDTO.firstName}."	Vorname sehen bzw. ändern.	/
name-itxt	<h:inputText>	value="#{ProfileBacking. userDTO.lastName}."	Name sehen bzw. ändern.	/
password-iscrt	<h:inputSecret>	value="#{ProfileBacking. userDTO.passwordSalt}."	Eingabe des neuen Passworts.	passwordValidator lenght mustBe at least 8 characters long. and contain at least one digit ,one lowercase letter, one uppercase letter, one special character .
email-itxt	<h:inputText>	value="#{ProfileBacking. userDTO.email}."	Email ändern.	EmailAddress-AvailbilityValidator
faculty-otxt	<h:outputText>	value="#{ProfileBacking. userDTO.faculty}	Überblick über die angehörte Fakultäten.	/
save-cbtn	<h:commandButton>	action="#{ProfileBacking.save}."	Wird die neue veränderte Daten gespeichert.	/
delete-cbtn	<h:commandButton>	action ="#{Profile-Backing.deleteProfile}."	löscht das Profil.	/

5.7 Error-Seite

errorPage.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
errorMsg-otxt	<h:outputText>	value ="#{errorPage-Backing.errorMessage}"	Was ist fehlgeschlagen.	/

5.8 Header

Header.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
logo-gimg	<h:graphicImage>	value="#{/design/logo}	Logo der Applikation	/
logout-cbtn	<h:commandButton>	action="#{NavigationBacking.logout}	Loggt den Nutzer aus dem System	/
help-link	<h:link>	value="Go to the help page. " outcome=/view/examiner/help"	link zur Hilfseite	/
profil-link	<h:link>	value="Go to the profile page." outcome=/view/examiner/profile"	link zur Profilübersicht	/
homepage-link	<h:link>	value="Go to the homepage." outcome=/view/examiner/homepage"	Link zur Hauptseite	/
user-list-link	<h:link>	value=ßhow the list of users" outcome=/view/deanery/Userlist"	Link zur Übersichtsseite aller Nutzer	/
create-circulation-link	<h:link>	value=create a Circulation " outcome=/view/examineCommittee/createCirculation"	Link zur erstellung einem Umlauf	/

5.9 Template

Main.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
//	<ui:include>	src=”header.xhtml”	kopfzeile	/
//	<ui:include>	/	Inhalt der Seite.	/
//	<ui:include>	src=”footer.xhtml”	Fußzeile.	/

5.10 footer.xhtml

footor.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
imprint-link	<h:link>	value="Go to the imprint page " outcome=/view/anonymous"	Link zur Seite des Impressums.	/

imprint.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
imprint-otxt	<h:outputText>	/	impressum des Betreibers.	/

5.11 Pagination.xhtml

Pagination.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
circulaiton-dtbl	<h:dataTable>	value="#{dtt.business.utilities. pagination. }."var=”entry”	Hier werden die Spalten eingefügt.	/
user-dtbl	<h:dataTable>	value="#{dtt.business.utilities. pagination. }."var=”entry”	Hier werden die Spalten eingefügt.	/
firstpage-cbtn	<h:commandButton>	action="#{dtt.business.utilities. Pagination.firstPage}	Zurück zur ersten Seite.	/
prevpage-cbtn	<h:commandButton>	action="#{dtt.business.utilities. Pagination.previousPage}	Eine Seite zurück.	/
currentpage-otxt	<h:outputText>	value="#{dtt.business.utilities. Pagination.currentPage}."	aktuelle Seite	/
nextpage-cbtn	<h:commandButton>	action="#{dtt.business.utilities. Pagination.nextPage}	Eine Seite weiter.	/
lastpage-cbtn	<h:commandButton>	action="#{dtt.business.utilities. Pagination.lastPage}	Zur letzten Seite.	/

5.12 Benutzer Verwaltung

userlist.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
search-field-itxt	<h:inputText>	value="#{UserlistBacking. searchfield }."	such Textfeld	/
search-cbtn	<h:commandButton>	action="#{UserListBacking. search}	Suche ausführen.	/
search-for-slcom	<h:selectOneMenu>	value="#{ UserlistBacking. searchItem }."	werden die Suchkriterien definiert, um herauszufinden, wonach genau gesucht wird	/
user-filter-slcom	<h:selectOneMenu>	value="#{UserlistBacking. filterItem }."	werden die filterkriterien definiert, um herauszufinden, wonach genau gefiltert wird	/
User-list-pg	<h:dataTable>	value="#{UserlistBacking. user-Paginaition.loadData	Tabelle mit Paginierung	/
user-state-slcom	<h:selectOneMenu>	value="#{UserControllBacking. setUserState }."	Festlegung der Nutzer im system	/
save-cbtn	<h:commandButton>	value="#{UserControllBacking. save }."	speicher die Änderungen	/

Data Table for User List

```
<h:dataTable value="#{userListBacking.users}"var=üser»
 <h:column rendered="#{userListBacking.sessionInfo.roleId>3}»
    <f:facet name="header»
      <h:outputText value=Fakultät/>
    </f:facet>
    <h:inputText value="#{user.faculty}»
      <f:ajax event="keyuplistener="#{userListBacking.filterUsers}render="dataTable/>
    </h:inputText>
  </h:column>
  <h:column>
    <f:facet name="header»
      <h:outputText value="Berechtigungen/>
    </f:facet>
    <h:inputText value="#{user.userState}»
      <f:ajax event="keyuplistener="#{userListBacking.filterUsers}render="dataTable/>
    </h:inputText>
  </h:column>
  <h:column>
    <f:facet name="header»
      <h:outputText value=Ë-Mail/>
    </f:facet>
    <h:inputText value="#{user.email}»
      <f:ajax event="keyuplistener="#{userListBacking.filterUsers}render="dataTable/>
    </h:inputText>
  </h:column>
  <h:column>
    <f:facet name="header»
      \href{profile.xhtml?id=#{user.id}       <h:outputText value="Vorname/>
    </f:facet>
    <h:inputText value="#{user.firstName}»
      \href{profile.xhtml?id=#{user.id}
      <f:ajax event="keyuplistener="#{userListBacking.filterUsers}render="dataTable/>
    </h:inputText>
  </h:column>
  <h:column>
    <f:facet name="header»
      \href{profile.xhtml?id=#{user.id}}{user.id}       <h:outputText value=Name/>
    </f:facet>
    <h:inputText value="#{user.lastName}»
      <f:ajax event="keyuplistener="#{userListBacking.filterUsers}render="dataTable/>
    </h:inputText>
  </h:column>
</h:dataTable>
```

5.13 Circulation

Circulationlist.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
search-field-itxt	<h:inputText>	value="#{CirculationlistBacking.searchField }."	such Textfeld	/
search-cbtn	<h:commandButton>	action="#{CirculationListBacking.search}	suchefunktion	/
search-for-slcom	<h:selectOneMenu>	value="#{CirculationlistBacking.searchItem }."	werden die Suchkriterien definiert, um herauszufinden, wonach genau gesucht wird	/
circulation-filter-slcom	<h:selectOneMenu>	value="#{Circulationlist Backing.filterItem }."	werden die filterkriterien definiert, um herauszufinden, wonach genau gefiltert wird	/
circulation-link	<h:link>	value=”Show the datails of the circulation” outcome=/view/examiner/circulationdatails"	link zur Umlaufdetails	/
Circulations-pg	<h:dataTable>	”#{CirculationlistBacking.users Pagination.loadData”	list alle Umläufe	/

Code

```
<h:dataTable value="#{circulationListBacking.circulations}"var=circ»
 <h:column rendered="#{CirculationListBacking.sessionInfo.roleId>3}»
    <f:facet name="header»
      <h:outputText value=Fakultät/>
    </f:facet>
    <h:inputText value="#{circulationListBacking.filter.faculty}»
      <f:ajax event="keyuplistener="#{circulationListBacking.filterCirculations}render="dataTable/>
    </h:inputText>
  </h:column>
  <h:column>
    <f:facet name="header»
      <h:outputText value=Titel/>
      \href{circulationdetails.xhtml?id=#{user.id}     </f:facet>
    <h:inputText value="#{circulationListBacking.filter.title}»
      <f:ajax event="keyuplistener="#{circulationListBacking.filterCirculations}render="dataTable/>
    </h:inputText>
  </h:column>
  <h:column>
    <f:facet name="header»
      <h:outputText value=Name des Doktoranden/>
    </f:facet>
    <h:inputText value="#{circulationListBacking.filter.doctoralCandidate}»
      <f:ajax event="keyuplistener="#{circulationListBacking.filterCirculations}render="dataTable/>
    </h:inputText>
  </h:column>
  <h:column>
    <f:facet name="header»
      <h:outputText value="Doktorvater/>
    </f:facet>
    <h:inputText value="#{circulationListBacking.filter.doctoralSupervisor}»
      <f:ajax event="keyuplistener="#{circulationListBacking.filterCirculations}render="dataTable/>
    </h:inputText>
  </h:column>
 <h:column>
    <f:facet name="header»
      <h:outputText value=''Startzeit'' />
    </f:facet>
 <h:column>
    <f:facet name="header»
      <h:outputText value=''Endzeit''/>
    </f:facet>
```

Tabelle 2: Data Table for a Circulation List

createcirculation.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
title-itxt	<h:inputText>	value="#{CreateCirculationBacking.circulation.titel}."	Titel der Dissertation	Eindeutigkeit eines gegebenen Circulation Name"überprüfen
doctoral-candidate-name-itxt	<h:inputText>	value="#{CreateCirculationBacking.circulation.doctoralCandidateName}."	Neme der Doktorand	/
doctoral-supervisoror-itxt	<h:inputText>	value="#{CreateCirculationBacking.circulation.doctoralSupervisoror }."	Neme der Doktorvater	/
description-itxt	<h:inputText>	value="#{CreateCirculation Backing.circulation.description }."	kurze Beschreibung über die Dissertation	/
start-deedline-itxt	<h:inputText>	value="#{CreateCirculationBacking. circulation.startDeedline }."	Starttermin für die Begutachtung	prüfen, ob ein angegebener Datumswert in der Zukunft liegt und ein gültiges zukünftiges Datum
end-deedline-itxt	<h:inputText>	value="#{CreateCirculationBacking. circulation.etartDeedline}."	Ende der First für die Begutachtung	prüfen, ob ein angegebener Datumswert in der Zukunft liegt und ein gültiges zukünftiges Datum
save-cbtn	<h:commandButton>	action="#{CirculationCreating Backing.create}	um den Umlauf zu speichern	/

Circulationdetails.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator/Constraints**
/	<f:viewParam>	value="#{CirculationdetailsBacking. circulation.id}	Id des Umlauf	/
/	<f:viewAction>	action="#{CirculationdetailsBacking. loadCirculation}	Laden des entsprechenden umlauf.	/
title-otxt	<h:outputText>	value="#{CirculationdetailsBacking. circulation.titel}	Titel der Dissertation	/
doctoral-candidate-name-otxt	<h:outputText>	value="#{CirculationdetailsBacking. circulation.doctoralCandidateName}."	Neme der Doktorand	/
doctoral-supervisoror-otxt	<h:outputText>	value="#{CirculationdetailsBacking. circulation.doctoralSupervisoror }."	Neme der Doktorvater	/
description-otxt	<h:outputText>	value="#{CirculationdetailsBacking. circulation.description }."	kurze Beschreibung über die Dissertation	/
start-deedline-otxt	<h:outputText>	value="#{CirculationdetailsBacking. circulation.startDeedline }."	Starttermin	/
end-deedline-otxt	<h:outputText>	value="#{CirculationdetailsBacking. circulation.endDeedline }."	Ende der First für die Begutachtung	/
votes-out	<h:outputText>	value="#{CreateCirculationBacking. loadVotes}."	Show the Votes for a Circulaiton	rendered="#{Circulationdetails Backing.sessionInfo.user.role Id>=2}."
remove-cbtn	<h:commandBut ton>	action="#{CirculationDetailsBacking. remove}	Umlauf löschen	rendered="#{Circulationdetails Backing.sessionInfo.user.role Id>=2}."
description-itxt	<h:inputText>	value="#{CirculationDetailsBacking. vote.description}	kurze Beschreibung hinzufugen	/
vote-options-slcom	<h:selectOn Menu>	value="#{CirculationDetailsBacking. vote.options}	verfügbare Optionen für eine Abstimmung oder Wahl in einer Zirkulation.	/
vote-btn	<h:commandBut ton>	action="#{CirculationDetailsBacking. vote}	Umlauf abstimmen	/
modify-cbtn	<h:commandBut ton>	outcome=”createCirculation” <f:param name=”circulationId” value=”#circulation.id” />	umlauf bearbeiter	rendered="#{Circulationdetails Backing.sessionInfo.user.role Id>=2}."

5.14 Hilfe

Hilfe.xhtml				
ID	**Typ**	**value/action**	**Beschreibung**	**Validator**
help-otxt	<h:outputText>	/	help texts.	/

6 Konfiguration

Johannes Silvennoinen

6.1 Überblick

In der DocThesisTracker Anwendung werden benutzerdefinierte Konfigurationsmöglichkeiten bereitgestellt. Die entsprechenden Konfigurationsdateien befinden sich im Verzeichnis "webapp/WEB-INF/config". Es gibt zwei separate Konfigurationsdateien: "configuration.properties" für allgemeine Systemeinstellungen, E-Mail Konfiguration und Datenbankinformationen, sowie "log4j2.properties" für die Konfiguration des Logging-Mechanismus der Anwendung.

6.2 configuration.properties

Die Konfigurationsdatei wird mithilfe der ReadConfig Klasse gelesen.

```
# The configuration for the DTT app.
# General Configuration

PAGINATION_MAX_ITEMS = 50
ROOT_ADMIN = silven01@uni-passau.de
COLOR_SCHEME = default
IMPRINT = Impressum
LOGO_PATH = LogoSeite.png

# E-Mail Configuration

EMAIL_PATTERN = ^[a-zA-Z0-9._%+-]+@uni-passau\\.de$

# Database Configuration

DATABASE_URL = http://bueno.fim.uni-passau.de/
DATABASE_USER = sep23g02
DATABASE_PASSWORD = PiewoiMahs2o
DATABASE_DRIVER = org.postgresql.Driver
DATABASE_SIZE = 50
SSL = true
SSL_FACTORY = org.postgresql.ssl.DefaultJavaSSLFactory

# Password Configuration

PASSWORD_PATTERN = ^.*(?=.{8,})(?=..*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=]).*$
```

6.3 log4j2.properties

Die Konfigurationsdatei für den Logger wird automatisch von log4j erkannt und gelesen.

```
# The configuration for the logger
rootLogger.level = debug
rootLogger.appenderRef.file.ref = File

appender.file.type = File
appender.file.name = File
appender.file.fileName = /logs/Logfile.log
appender.file.layout.type = PatternLayout
appender.file.layout.pattern = %d{HH:mm::ss:SSS} [%t] %5level %logger{36} - %msg%n

```

6.4 Resource Bundles

Die Nutzung von Resource Bundles in dieser JSF-Webanwendung ermöglicht vorerst die Unterstützung der deutschen Sprache und die Lokalisierung der Anwendung. Durch die Speicherung von Texten in separaten Ressource-Dateien erleichtert es die Internationalisierung und Lokalisierung der Anwendung, sodass es leichter ist später andere Sprachen hinzufügen zu können. Resource Bundles ermöglichen die einfache Aktualisierung von Texten ohne erneutes Kompilieren der Anwendung. Sie verbessern die Wartbarkeit und bieten eine hohe Wiederverwendbarkeit von Texten. Die Resource Bundles werden in dem 'bundles' ordner gespeichert.

7 Systemfunktionen

Johannes Silvennoinen

7.1 Systemstart

Der Start des Systems umfasst mehrere Schritte:

1. Starten von Tomcat: Tomcat ist ein Webserver und Servlet-Container, der das System ausführt. Der Server wird durch Ausführen der Start-Skripte gestartet.

2. Weblistener: Der Weblistener ist eine Komponente des Tomcat-Servers, die bestimmte Ereignisse im Lebenszyklus des Servers verfolgt. Bem Start des Systems wird der Weblistener verwendet, um die Verbindung zur Datenbank initialisieren die Konfiguration zu lesen und den Logger zu starten durch die methode `contextInitalized()`.

3. Logger initialisieren: Als erstes wird der Logger initialisiert, um sicherzustellen, dass wir Log-Meldungen ausgeben können, falls bei dem Systemstart Probleme auftreten. Der Logger wird über die Methode `init()` initalisiert.

4. Konfiguration lesen: Das System muss beim Start die Konfigurationsdatei config.properties lesen, um die Einstellungen und Parameter zu erhalten, die für das System erforderlich sind. Die Konfigurationsdatei wird als Java-Properties gespeichert und enthalten Informationen wie Datenbankverbindungsparameter, Sicherheitseinstellungen, Zugriffsrechte und andere Systemeinstellungen. Dies erfolgt durch die methode `loadConfig()`.

5. Verbindung zur Datenbank: Beim Start des Systems muss eine Verbindung zur Datenbank hergestellt werden. Hierfür wird der Connection Pool verwendet. Der Connection Pool ist als Singleton implementiert und threadsicher, so dass mehrere Anfragen gleichzeitig verarbeitet werden können. Für die initialisierung der Verbindungen wird die Methode `initalizeConnections()` aufgerufen.

6. Wartungsthread: Beim Start des Systems wird auch ein Wartungsthread gestartet, der die Systemwartung und -überwachung durchführt. Die Methode `startMaintenanceThread()` wird hierfür aufgerufen.

7.2 Systemstop

Der Stop des Systems umfasst auch mehrere Schritte:

1. Weblistener: Beim Herunterfahren des Systems wird der Weblistener verwendet, um die Verbindung zur Datenbank zu schließen, den Logger zu stoppen und den Wartungsthread zu beenden. Dies erfolgt durch die Methode `contextDestroyed()`.

2. Wartungsthread: Beim Herunterfahren des Systems wird dieser Thread gestoppt. Die Methode `stopMaintenanceThread()` wird hierfür aufgerufen.

3. Datenbankverbindung: Beim Stoppen des Systems wird die Verbindung zur Datenbank geschlossen. Der Connection Pool wird verwendet, um sicherzustellen, dass alle offenen Verbindungen geschlossen werden. Dafür ist die methode `closeConnections()` zuständig.

4. Logger: Beim Herunterfahren des Systems muss auch der Logger gestoppt werden, um sicherzustellen, dass alle ausstehenden Log-Einträge geschrieben werden. Hierfür wird die Methode `stopLogger()` aufgerufen.

5. Tomcat-Server herunterfahren: Schließlich muss der Tomcat-Server heruntergefahren werden, um das System vollständig zu beenden.

7.3 Wartungsthread

Das System verfügt über einen Wartungsthread, der zuständig ist um erinnerungs E-Mails zu Schicken an Prüfungsberechtigte die noch nicht abgestimmt haben, 1 tag vor der Ablauffrist.

7.4 Zugriffsrechte

Das System ist mit verschiedenen Benutzerrollen ausgestattet. Jede Rolle hat unterschiedliche Zugriffsrechte auf die Funktionen des Systems. Administratoren haben die höchsten Zugriffsrechte und können alle Funktionen des Systems nutzen. Andere Benutzerrollen haben eingeschränkte Zugriffsrechte. Das System verwendet eine Kombination von Techniken, um sicherzustellen dass nur autorisierte Benutzer auf das System zugreifen können. Zu diesen Techniken gehören Trespass-Listener sowie die Überprüfung von Zugangsrechten in Backing-Beans.

Der Trespass-Listener überwacht die Anforderungen an das System und überprüft, ob der Benutzer über die efroderlichen Berechtigungen verfügt, um auf die angeforderte Ressource zugreifen. Wenn der Benutzer nicht über die erforderlichen Berechtigungen verfügt, wird eine Zugriffsverletzung ausgelöst.

Darüber hinaus werden die Zugriffsrechte auch in Backing-Beans überprüft. Backing-Beans enthalten den Business-Code des Systems und stellen eine Verbindung zwischen den xhtml Seiten und der Datenzugangs-Schicht her. Bevor ein Benutzer auf eine Ressource zugreifen kann, wird in den Backing-Beans überprüft, ob der Benutzer über die erforderlichen Berechtigungen verfügt. Wenn der Benutzer nicht über die erforderlichen Berechtigungen verfügt, wird ebenso eine Zugriffsverletzung ausgelöst.

Durch die Kombination dieser Techniken wird sichergestellt, dass nur autorisierte Benutzer auf das System zugreifen können und dass die Integrität und Sicherheit des Systems gewahrt bleibt.

7.5 Logging

Das System nutzt die Java Logging API um eine umfassende Protokollierung aller Aktivitäten durch zu führen. Dabei werden sämtliche Aktionen, die von Benutzern und dem System ausgeführt werden, erfasst. Die Protokolldateien werden auf dem Server gespeichert und sind nur für berechtigte Personen einsehbar. Das Logging dient der Überwachung der Systemaktivitäten und der Fehlerbehebung. Die Loglevels sind auf folgende 4 Level aufgeteilt:

- Severe
- Warning
- Info
- Debug

7.6 Sicherheitsaspekte

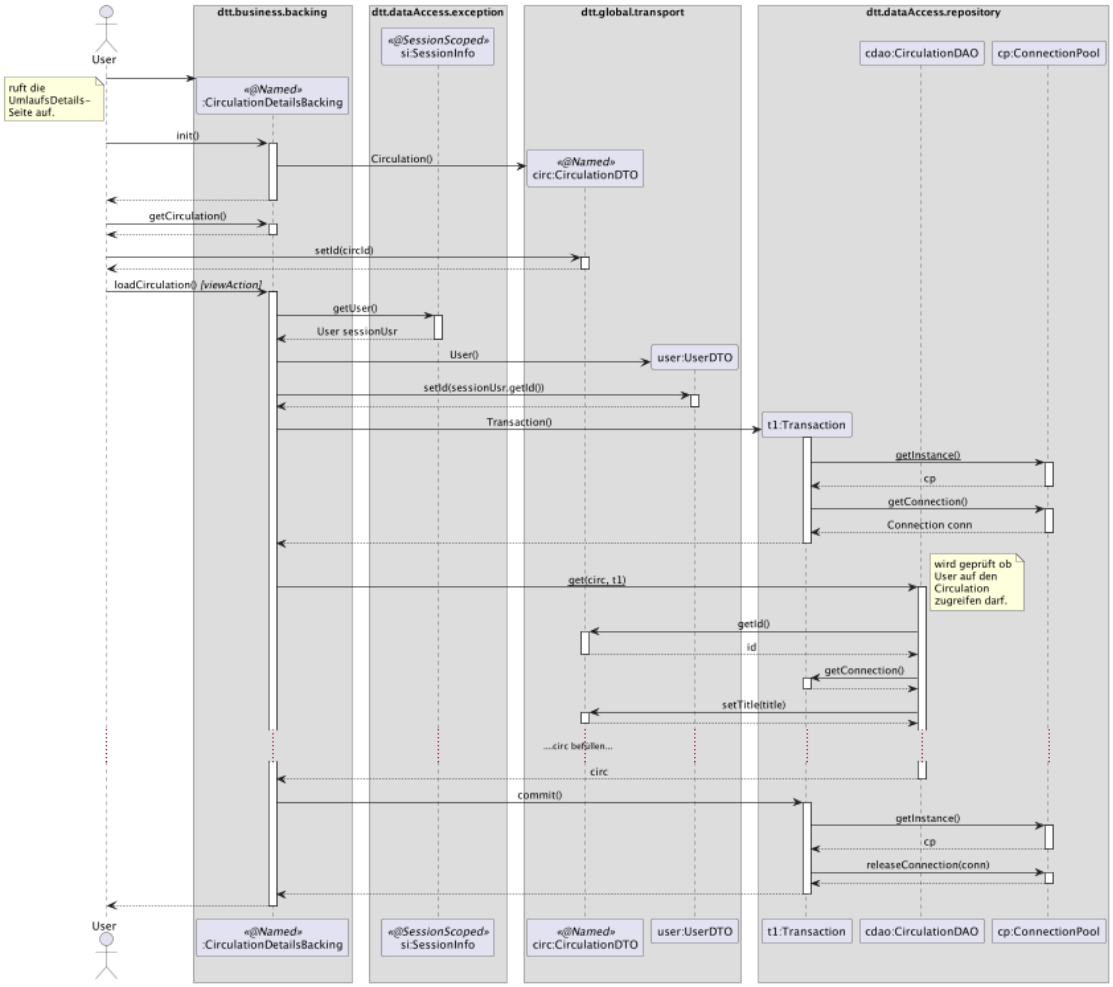
Das System ist gegen Angriffe von außen abgesichert. Der Datenverkehr zwischen dem Server und dem Client erfolgt über eine SSL-verschlüsselte Verbindung, um die Übertragung von Daten abzusichern. Die Verbindung zur Datenbank ist ebenfalls über SSL gesichert, um die Sicherheit der Datenübertragung zu gewährleisten. Das System verfügt über eine sinnvolle Fehlerbehandlung bei fehlerhaften Eingaben, um mögliche Schwachstellen im System zu minimieren.

8 Datenfluss

Alaa Qasem

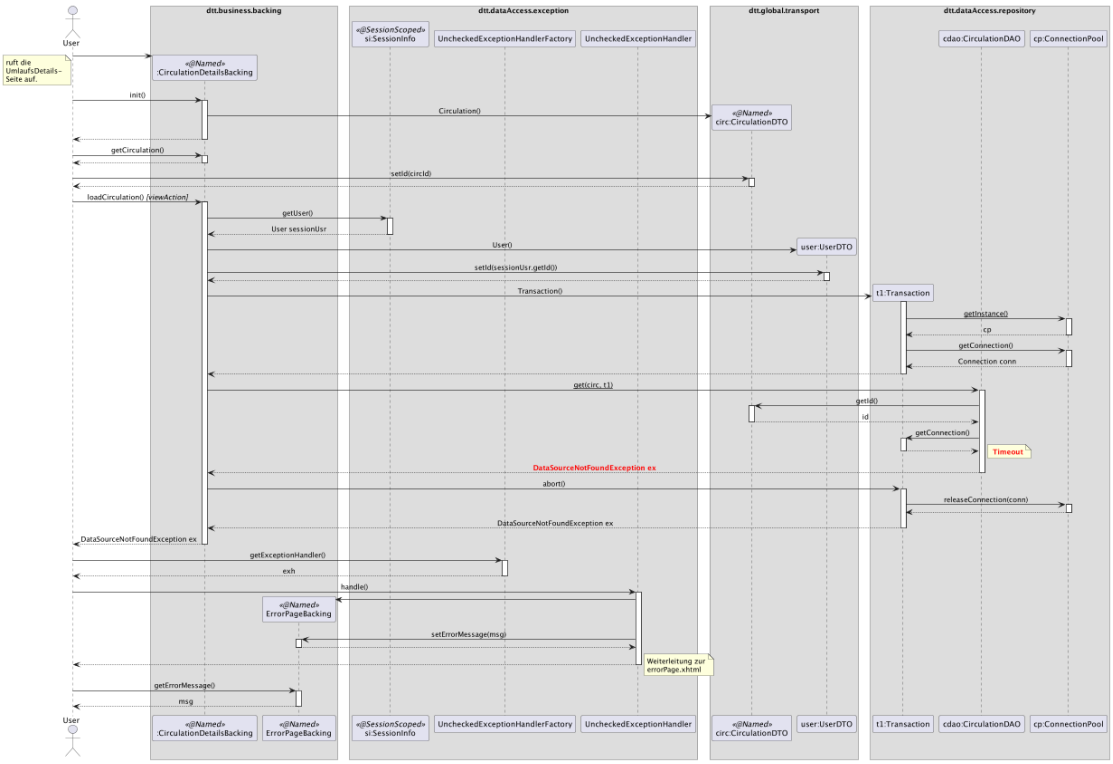
8.1 Anzeigen eines Umlaufs

Ein angemeldeter Nutzer ruft eine UmlaufsDetails-Seite.



8.2 Anzeigen eines Umlaufs (Fehlerfall)

Ein Timeout Fall



9 DB Schema

Stefan Witka

9.1 ER-Diagramm

ER Diagramm an neue Anforderungen angepasst.

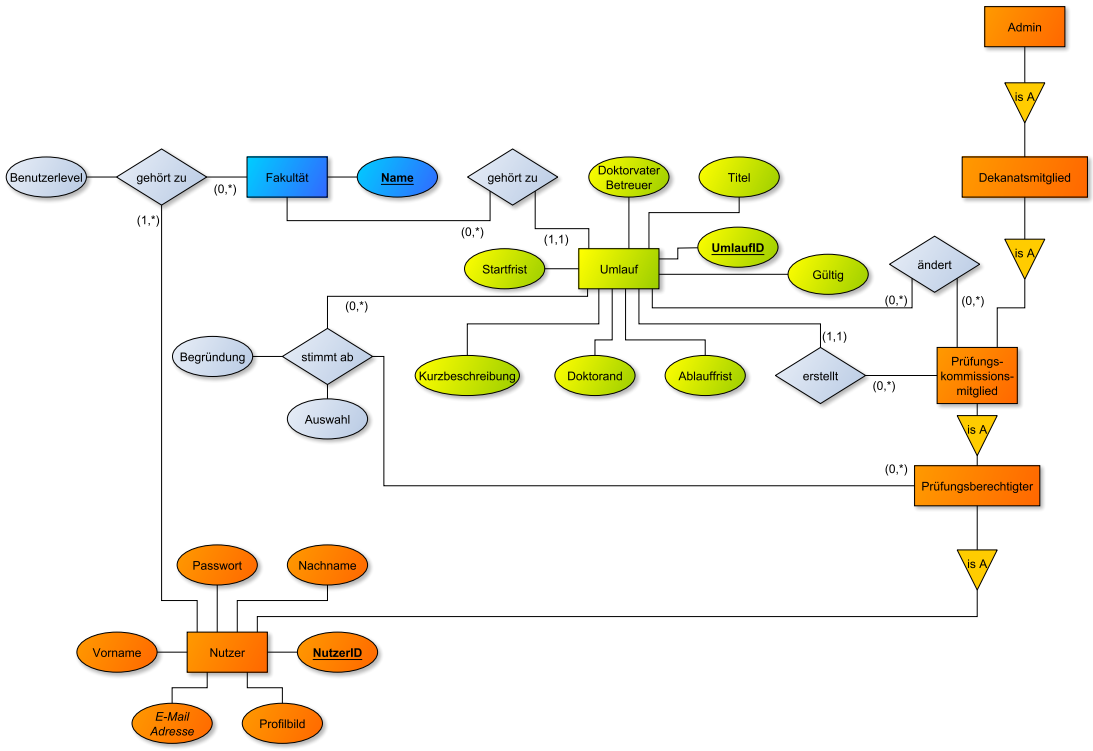


Abbildung 3: Das ER Diagramm zu DocThesisTracker

9.2 DDL

```
CREATE TABLE "user" (
user_id SERIAL PRIMARY KEY NOT NULL,
email_address VARCHAR(70) UNIQUE NOT NULL,
first_name VARCHAR(50) NOT NULL,
last_name VARCHAR(50) NOT NULL,
birth_date DATE,
password_hash VARCHAR(50),
```

```
password_salt VARCHAR(24),
CONSTRAINT valid_birthdate CHECK (birthdate < CURRENT_DATE),
CONSTRAINT email_address_pattern CHECK (email_address LIKE '_%@_%')
);
```

```
CREATE TABLE faculty (
faculty_id SERIAL PRIMARY KEY NOT NULL,
faculty_name VARCHAR(50) UNIQUE NOT NULL
);
```

```
CREATE TABLE circulation (
circulation_id SERIAL PRIMARY KEY NOT NULL,
title VARCHAR(50) UNIQUE NOT NULL,
doctoral_candidate_name VARCHAR(50) NOT NULL,
doctoral_supervisor_name VARCHAR(50) NOT NULL,
description VARCHAR(500) NOT NULL,
start_date DATE,
end_date DATE,
is_obligatory BOOLEAN,
created_by INTEGER,
faculty_id INTEGER NOT NULL,
is_valid BOOLEAN,

FOREIGN KEY (created_by) REFERENCES "user"(user_id) ON DELETE SET NULL,
FOREIGN KEY (faculty) REFERENCES faculty(faculty_id) ON DELETE CASCADE
);
```

```
CREATE TYPE user_state AS ENUM (
   'ADMIN',
    'EXAMINCOMMITTEEMEMBERS',
    'EXAMINER',
    'DEANERY',
    'PENDING'
)

CREATE TABLE authentication (
  authentication_id SERIAL PRIMARY KEY NOT NULL,
  user_id INTEGER NOT NULL,
  faculty_id INTEGER,
  user_level user_state,
  UNIQUE (user_id, faculty_id),
  FOREIGN KEY (user_id) REFERENCES "user"(user_id) ON DELETE CASCADE,
```

```
  FOREIGN KEY (faculty_id) REFERENCES faculty(faculty_id) ON DELETE
      CASCADE
);
```

```
CREATE TABLE "admin" (
user_id INTEGER NOT NULL,

PRIMARY KEY (user_id),
FOREIGN KEY (user_id) REFERENCES "user"(user_id) ON DELETE CASCADE
);
```

```
CREATE TABLE vote (
vote_id SERIAL PRIMARY KEY NOT NULL,
user_id INTEGER,
circulation_id INTEGER NOT NULL,
choice INTEGER NOT NULL,
reason VARCHAR(500),

UNIQUE (user_id, circulation_id),
FOREIGN KEY (user_id) REFERENCES "user"(user_id) ON DELETE SET NULL,
FOREIGN KEY (circulation_id) REFERENCES circulation(circulation_id) ON
    DELETE CASCADE
);
```

9.3 Vererbung

Die im ER-Diagramm durch „is A“ gekennzeichneten Nutzerrollen werden durch eine einzige „user“ Tabelle implementiert.
Die tatsächlichen Nutzerrollen werden in der „authorization“ Tabelle pro Fakultät festgelegt. Ausnahme bilden die Administratoren, welche in einer eigenen „admin“ Tabelle angelegt sind.

9.4 Normalform

Das Datenbankschema befindet sich in Boyce-Codd-Normalform.

9.5 Namenskonventionen

Tabellennamen sind im englischen Singular.
Identifikatoren (Tabellen, Spalten, etc) Sind in snake case,
kleinbuchstaben_mit_unterstrichen

Johannes Silvennoinen

10 Sicherheit

10.1 SQL Injections

Um potenzielle SQL-Injection-Angriffe zu verhindern, wird in der Webanwendung die Verwendung von Prepared Statements konsequent angewendet. Prepared Statements ermöglichen die vorherige Kompilierung von SQL-Statements, wodurch die Trennung von SQL-Code und Nutzereingaben gewährleistet wird. Bei der Ausführung des Statements werden die Nutzereingaben als separate Parameter behandelt und nicht direkt in das SQL-Statement eingefügt. Dies stellt sicher, dass die Nutzereingaben nicht dazu führen können, dass das SQL-Statement modifiziert oder manipuliert wird.

Durch die Verwendung von Prepared Statements wird die Sicherheit der Datenbankabfragen erhöht, da potenziell schadhafter SQL-Code durch die Trennung von Nutzereingaben und SQL-Statement verhindert wird. Die Parameterwerte werden sicher an die Datenbank übermittelt, ohne dass sie als Teil des SQL-Codes interpretiert werden. Dies bietet einen effektiven Schutz vor SQL-Injection-Angriffen und gewährleistet die Integrität der Datenbankabfragen.

In der Umsetzung der Webanwendung wird daher konsequent auf die Verwendung von Prepared Statements gesetzt, um sicherzustellen, dass SQL-Injection-Angriffe effektiv verhindert werden und die Datenbank vor potenziellen Sicherheitslücken geschützt ist.

10.2 Cross Site Scripting

Für die Verhinderung von Cross-Site-Scripting (XSS) werden in der Webanwendung ausschließlich JSF-Komponenten für die HTML-Ausgabe verwendet. JSF bereinigt automatisch alle Nutzerausgaben, um potenzielle XSS-Angriffe zu verhindern. Nutzerausgaben, einschließlich spezieller Zeichen wie '<' und '>', werden sicher in der Ausgabe verarbeitet, sodass sie nicht als HTML-Tags interpretiert werden können.

Auch das Einfügen von JavaScript-Code wird durch diese Maßnahmen verhindert. JavaScript-Code müsste sich in einem HTML-Tag wie beispielsweise <script> befinden, um ausgeführt zu werden. Da JSF die Nutzerausgaben sicher verarbeitet und sie nicht als Teil eines HTML-Tags interpretiert, ist das Einfügen von

JavaScript-Code nicht möglich.

Durch diese Vorgehensweise wird sichergestellt, dass XSS-Angriffe effektiv verhindert werden und die Ausgabe der Webanwendung sicher ist.

10.3 Insecure Direct Object Reference

Um unautosrisierten Zugriff auf bestimmte Seiten, Information und Aktionen zu verhindern, wird standardmäßig der Zugriff auf nicht-öffentliche Ressource verweigert. Ausgenommen sind lediglich Login- und Registrierungsseiten. Die Entscheidung, ob ein Nutzer die erforderlichen Berechtigungen hat, wird durch die Klasse "TrespassListener" getroffen. Eine anfrage wird nur akzeptiert, wenn die erforderlichen Rechte vorliegen, ansonsten wird der Nutzer auf eine 404-Fehlerseite weitergeleitet oder es wird eine enstsprechende Fehlermeldung angezeigt.

Dies geschieht, indem die angeforderte URL analysiert wird, um das zugehörige Facelet zu ermitteln. Anschließend wird überprüft, ob der Zugriff auf das Facelet gemäß der Rolle erlaubt ist. Der TrespassListener kann über die Sitzungsinforamtionen auf die Rolle des Nutzers zugreifen. Darüber hinaus hängt der Zugriff auf die angeforderte Ressource von der Beziehung des Nutzers dazu ab. Zum Beispiel kann ein Prüfungsberechtigter nur sein eigenes Profil sehen. Aus diesem Grund überprüft das jeweilige Backing Bean für die relevanten Seiten diese Beziehung und leitet gegebenfalls auf eine Fehlerseite weiter. Dadurch wird eine "Insecure Direct Object Refrence" vermieden, da IDs von Benutzern und Umläufen nicht über URL-Parameter offengelegt werden.

Die Konfiguration in der "web.xmlDatei mit dem Eintrag des "Faces Servlet" leitet Anfragen an alle unsere Facelet-Dateien an das Faces Servlet weiter. Dadurch wird basierend auf der Entscheidung des TrespassListeners entweder das Facelet korrekt als HTML gerendert oder der Nutzer zur richtigen Seite weitergeleitet oder es wird eine Fehlermeldung angezeigt. Es ist jedoch nicht möglich, die eigentliche Facelet-Datei, die auf dem Server gespeichert ist, herunterzuladen.

10.4 Passwörter

Die Passwörter in der Webanwendung werden mithilfe der sicheren PBKDF2-Funktion gehasht. Jedes Passwort erhält einen eigenen zufälligen generierten Salt, der zusammen mit dem Hash in der Datenbank gespeichert wird. Um eine maxmimale Sicherheit zu gewährleisten, werden die Passwörter so früh wie möglich gehasht, um die Lebensdauer der Klartextpasswörter auf dem Server zu miminimieren. Die Klasse Hashing in dem Utilities Paket in der Bussiness schicht bietet

Methoden zum Generieren eines Salts und zum Durchführen des Hashings. Für diese Funktion wird die javax.cryptop-Bibliothek verwendet.

Um das Erraten von Passwörtern durch Brute-Force-Attacken und ähnliche Methoden zu erschweren, müssen die Passwörter eine Länge von 8 bis 100 Zeichen haben und aus Großbuchstaben, Kleinbuchstaben, Zahlen und Sonderzeichen bestehen. Diese Anforderungen werden vom PasswordValidator.java geprüft.

10.5 Session Fixation

In der web.xml-Datei kann ein Timeout für Sitzungen konfiguriert werden. Dadurch wird Session-Hijacking sowie unerwünschter physischer Zugriff auf den Browser des Clients verhindert. JSF setzt standardmäßig das "http-only"-Flag für das Session-Cookie, wodurch ein Zugriff über JavaScript verhindert wird und Hijacking erschwert wird.

Nach jeder Anmeldung wird die Sitzung verworfen und neu generiert. Auf diese Weise bleibt keine alte Session nach der Authentifizierung bestanden, und Session-Fixation-Angriffe werden verhindert.

10.6 Vermeidung von Informationen-Enthüllung

Um sicherzustellen, dass sensible Informationen nicht nach außen gelangen, werden verschiedene Maßnahmen ergriffen. Im Produktionsmodus werden Fehlermeldungen so konfiguriert, dass sie keine detaillierten technischen Informationen enthalten. Stattdessen erhalten die Benutzer nur relevante und benutzerfreundliche Fehlermeldungen.

Zusätzlich werden eigene Fehlerseiten für Ausnahmesituationen verwendet. Die standardmäßigen Fehlerseiten für die HTTP-Fehlercodes 404 (Seite nicht gefunden) und 500 (Interner Serverfehler) von JSF werden durch benutzerdefinierte Fehlerseiten ersetzt. Dadurch wird die Verwendung von JSF nicht offensichtlich.

Um weitere Informationen zu verschleiern, verwenden wir spezifische Dateiendungen für unsere Facelets, beispielsweise .xhtml. Zudem wird der Bezeichner "JSESSIONID" für Session-Cookies und URL-Rewriting in der Konfigurationsdatei verändert.

Diese Maßnahmen dienen dazu, Informationen über die technische Umgebung, verwendete Software, interne Fehlermeldungen und Implementierungsdetails vor unbefugtem Zugriff zu schützen.