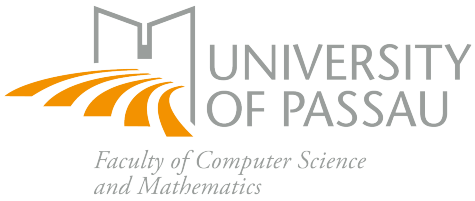

Feinspezifikation

TalentTauscher

Betreuer: Prof. Dr. Christian Bachmaier

Team 1

Phase	Verantwortlich
Pflichtenheft	Leon Föckersperger
Entwurf & Spezifikation	Jakob Edmaier
Implementierung & Validierung	David Sturm
Abschlusspräsentation	Leon Föckersperger

Version 2.0

29. Mai 2024

Inhaltsverzeichnis

1 Einleitung

Leon Föckersperger

Dieses Dokument stellt die Feinspezifikation für die Web-Applikation *TalentTauscher* dar. Es zielt darauf ab, eine Plattform zur Vermittlung und zum Austausch von Dienstleistungen zu schaffen. Die vorliegende detaillierte Spezifikation erweitert den initialen Entwurf und bietet eine tiefgehende und umfassende Darstellung der Funktionalitäten und Anforderungen des Systems.

2 Systemarchitektur

Jakob Edmaier

In diesem Kapitel wird die Architektur unseres Programms spezifizert. Dazu wird die verwendete Schichtenarchitektur graphisch veranschaulicht und die Ordnerstruktur des Projekts abgebildet.

2.1 Schichtenarchitektur

Im Vergleich zum Entwurf wurden die folgenden Änderungen an der Paketstruktur vorgenommen:

- Startup und Shutdown der Applikation werden im neuen Paket `model.backing.lifecycle` gesteuert. Dazu werden mithilfe von `@Observes`-Annotationen die Lifecycle-Klassen in den unteren Schichten aufgerufen.
- Ein neues Paket `model.persistence.config` enthält eine neue Klasse *AppConfig*, welche die Konfigurationsparameter der Anwendung (etwa für Datenbank und Email-Server) bereitstellt.
- Das Paket `model.persistence.connection` wurde aufgelöst und der bisher enthaltene *ConnectionPool* nach `model.persistence.dataaccess` verschoben.

Die Systemarchitektur ist in Abbildung 1 visualisiert.

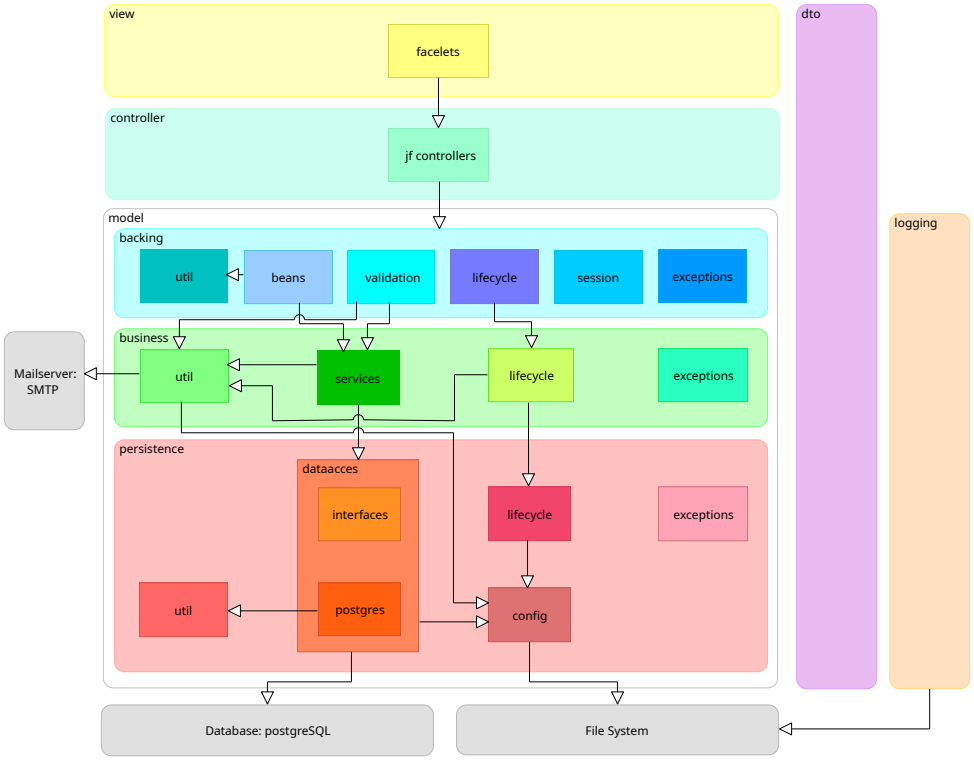

Abbildung 1: Schichtenmodell

2.2 Ordnerstruktur

Die Ornderstruktur der Entwicklungsumgebung ist in 2 abgebildet.

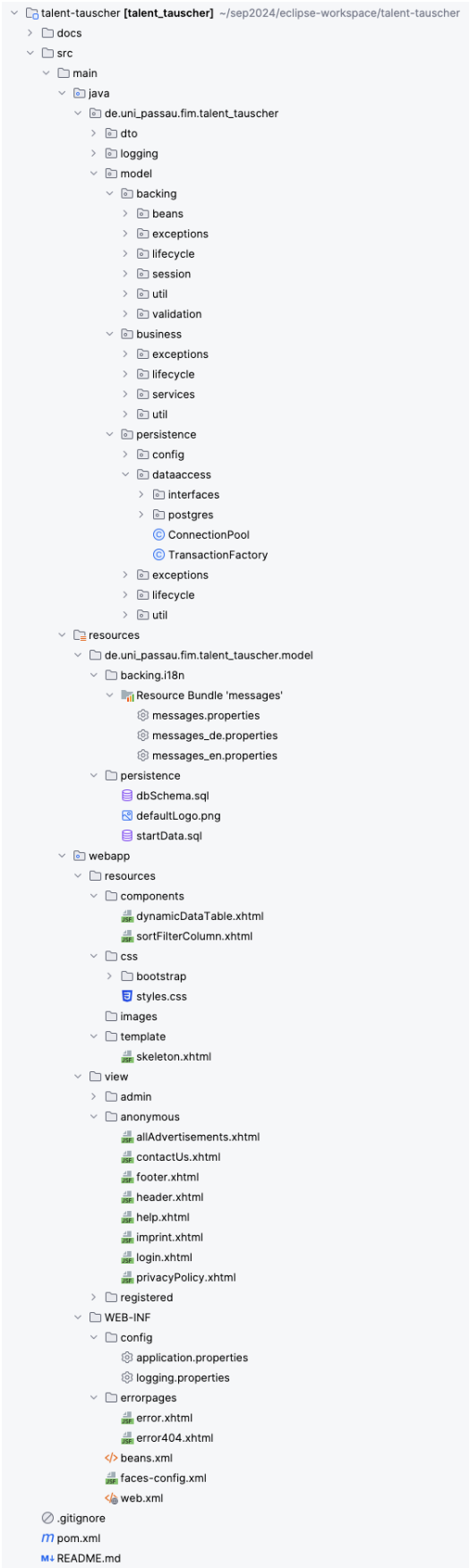

Abbildung 2: Ordnerstruktur

3 Klassen

Jakob Edmaier

Das folgende Kapitel enthält das Klassendiagramm. Für eine bessere Ansicht findet sich im Anhang eine alleinstehende Version des Klassendiagramms (Klassendiagramm.pdf).

3.1 Konventionen

Da fast jede Klasse DTOs benutzt, sind die entsprechenden uses-Beziehungen im Klassendiagramm im Sinne der Übersichtlichkeit nicht eingezeichnet. Klassen der `persistence`-Layer, die einen Logger benötigen, erstellen eine Instanz über eine Fabrik-Methode der Klasse *LoggerProducer*. Klassen der `backing` und `business`-Layer injecten bei Bedarf einen Logger und benutzen damit ebenfalls implizit den *LoggerProducer*. Diese use-Beziehungen sind im Klassendiagramm ebenfalls nicht eingezeichnet.

3.2 Klassendiagramm

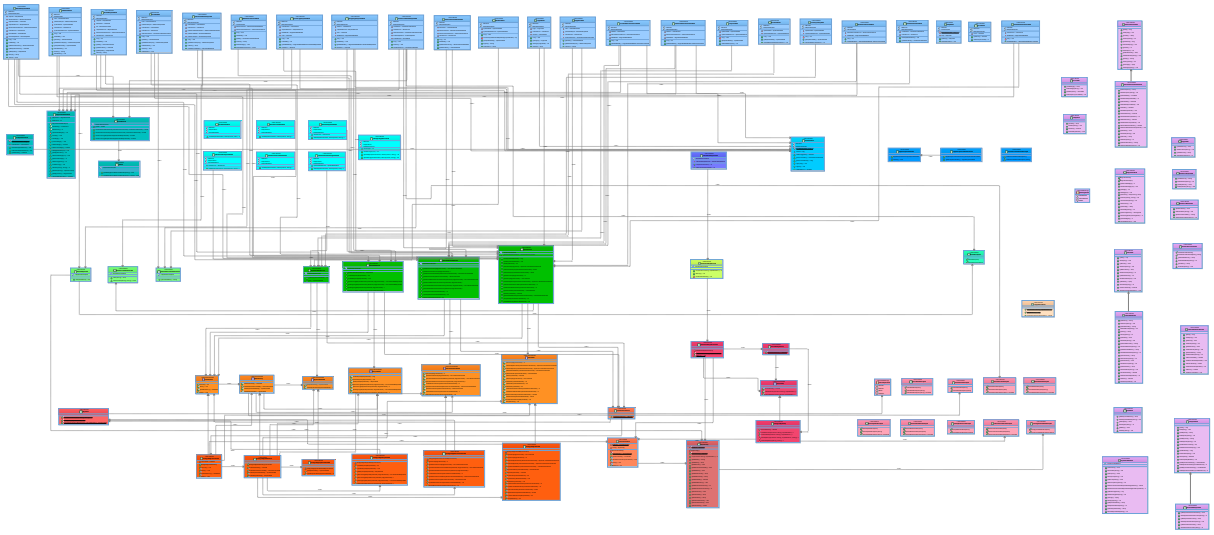

Abbildung 3: Klassendiagramm

3.3 JavaDoc

Sämtliche Klassen, Interfaces und Enums sind mithilfe von JavaDoc dokumentiert, einschließlich Methoden mit Sichtbarkeit `protected` oder `package`. Lediglich `private` Methoden und Attribute sind ausgenommen. Die JavaDoc-Kommentare finden sich als API-Dokumentation in HTML exportiert im Anhang (JavaDoc.zip).

4 Bibliotheken

Autor: Leon Föckersperger

Die Webapplikation ist auf mehrere Open-Source-Bibliotheken angewiesen, die essentiell für ihren Betrieb sind. Diese Bibliotheken sind kostenfrei und werden zusammen mit unserer Anwendung ausgeliefert, um zusätzliche Kosten zu vermeiden und die Installation zu vereinfachen. Im Folgenden ist eine Übersicht der verwendeten Bibliotheken samt Versionen und Lizenzen gegeben:

Name	Beschreibung	Version	Lizenz
JBoss Weld	Bietet Unterstützung für CDI (Contexts and Dependency Injection) in Java, entscheidend für die Geschäftslogik-Verwaltung.	5.1.2.Final	Apache 2.0
Mojarra	Ist die Referenzimplementierung von Jakarta Faces der Eclipse Foundation, zentral für die UI-Komponenten.	4.0.6	Eclipse Public License v2.0
PostgreSQL JDBC	Ermöglicht den Zugriff auf PostgreSQL Datenbanken, unerlässlich für die Datenverwaltung.	42.7.3	BSD 2-Clause
Bootstrap	Ein CSS-Framework, das das responsive Design für Webanwendungen unterstützt, essentiell für die Benutzeroberfläche.	5.3.3	Apache 2.0
Jakarta Mail API	Ermöglicht das Versenden von E-Mails aus Java-Anwendungen, wichtig für die Kommunikationsfähigkeiten.	2.1.3	Eclipse Public License v2.0
OmniFaces	Erweitert JavaServer Faces (JSF) um zusätzliche Komponenten, verbessert die Frontend-Entwicklung erheblich.	4.4.1	Apache 2.0

Diese Tabelle bietet eine klare und umfassende Darstellung der Technologien und Werkzeuge, die zur Unterstützung der Webapplikation benötigt und mitgeliefert werden. Die Aufstellung verdeutlicht auch die rechtlichen Rahmenbedingungen jeder Komponente.

5 Konfigurationsdateien

Leon Föckersperger

Unsere Softwarearchitektur bietet umfassende Möglichkeiten zur Feinkonfiguration mittels XML- und Eigenschaftsdateien, die in den Verzeichnissen `webapp/WEB-INF` und `webapp/WEB-INF/config` angeordnet sind. Darüber hinaus werden betriebskritische Einstellungen wie die Paginierung – also die Anzahl der Elemente pro Seite – dynamisch in der Datenbank gespeichert. Diese Methode ermöglicht eine flexible Anpassung der Systemparameter ohne die Notwendigkeit, die Anwendung neu starten zu müssen.

5.1 Detaillierter Überblick über die Konfigurationsdateien

Die Konfiguration unserer Anwendung wird durch verschiedene `.properties` Dateien gesteuert, die jeweils spezifische Aspekte abdecken:

- `application.properties`:
  - Diese Datei ist hauptverantwortlich für die Einstellungen der E-Mail-Dienste und der Datenbankverbindungen, was eine zentrale Rolle in der Netzwerkkommunikation und Datenmanagement spielt.
- `logger.properties`:
  - Diese Datei regelt die Protokollierungsstrategien innerhalb der Anwendung, eine essentielle Komponente für die Überwachung und Fehlerbehebung.

5.2 Zentrale Konfigurationsdatei: `application.properties`

Die `application.properties`-Datei ist das Herzstück der Konfiguration. Sie enthält Schlüsselparameter für den effizienten Betrieb der Anwendung:

- **E-Mail-Konfiguration**:
  - Hier sind die Parameter für die Absender-Adresse, den SMTP-Server, die Authentifizierungsmethoden und die Port-Nummern festgelegt, um eine sichere und effiziente E-Mail-Kommunikation zu gewährleisten.
- **Datenbank-Konfiguration**:

– Diese Sektion bietet detaillierte Informationen zur Anbindung an den Datenbankserver, zur SSL-Konfiguration und zu den Einstellungen des Verbindungspools, um eine optimale Leistung und Sicherheit zu gewährleisten.

```
########################################################################
# TalentTauscher Application Configuration                             #
# This file contains settings for email and database connectivity.     #
########################################################################

#------------------- EMAIL CONFIGURATION --------------------------#
# SMTP server settings for sending emails.

# Email address used as the sender in outgoing emails.
MAIL_ADDRESS_FROM = <email-address>  # Example: user@example.com

# SMTP server authentication details.
# IMPORTANT: Use environment variables to securely store sensitive information.
SMTP_USERNAME = ${ENV_SMTP_USERNAME}  # Set your SMTP username in an environment variable.
SMTP_PASSWORD = ${ENV_SMTP_PASSWORD}  # Set your SMTP password in an environment variable.

# SMTP server connection settings.
SMTP_HOST = <smtp-server-host>    # Example: smtp.example.com
SMTP_PORT = <smtp-port>           # Use 587 for STARTTLS or 465 for SMTPS/SSL.
SMTP_AUTH = <smtp-auth>           # Typically true; set to false if not required.

#------------------- DATABASE CONFIGURATION -----------------------#
# Database server connection settings.

# Connection string prefix.
DB_CONNECTION_PREFIX = <connection-prefix>  # Example: jdbc:postgresql

# Database server details.
DB_HOST = <db-server-host>   # Example: db.example.com
DB_PORT = <db-port>          # Default for PostgreSQL is 5432.

# Database driver class.
DB_DRIVER = <db-driver>  # Example: org.postgresql.Driver

# Database access credentials.
# IMPORTANT: Use environment variables to securely store sensitive credentials.
DB_NAME = <database-name>        # Example: myappdb
DB_USER = ${ENV_DB_USER}         # Set your database username in an environment variable.
DB_PASSWORD = ${ENV_DB_PASSWORD} # Set your database password in an environment variable.

# SSL configuration for secure connections.
DB_SSL = <db-ssl>                        # true or false.
DB_SSL_FACTORY = <ssl-factory-class>     # Example: org.postgresql.ssl.DefaultJavaSSLFactory

# Database connection pool settings.
DB_CONNECTION_COUNT = <connection-count>  # Recommended: 10
DB_TIMEOUT = <timeout-milliseconds>       # Example: 5000

################################
# Additional Setup Notes       #
################################
# Replace all placeholders with actual values.
# Use environment variables where sensitive information is required.
```

Listing 1: application.properties

5.3 Konfiguration und Verwaltung des Loggings

Die `logging.properties`-Datei ist ausschlaggebend für die Konfiguration und Verwaltung des Loggings. Sie definiert die Protokollierungsmethoden und den Speicherort der Log-Daten.

- **Allgemeine Logging-Einstellungen**:
  - Hier werden die Log-Handler und das Standard-Logging-Level festgelegt, um eine angemessene Protokollierung zu gewährleisten.
- **File Handler**:
  - Diese Einstellungen bestimmen den Speicherort und das Format der Log-Dateien sowie deren Lebensdauer, um eine effektive Log-Verwaltung zu sichern.

```
############################################
# Logging Configuration                    #
# Configures logging levels and handlers.  #
# Uses Apache Juli and Java util logging.  #
############################################

# List of handler classes used for logging.
handlers = org.apache.juli.FileHandler

```

```
# Default global logging level.
.level = INFO

############################################################################################
# Configuration of the File Handler                                                        #
# Documentation: https://tomcat.apache.org/tomcat-10.1-doc/api/org/apache/juli/FileHandler.html
############################################################################################

# Logging level for the file handler.
org.apache.juli.FileHandler.level = INFO

# Formatter for the file handler.
org.apache.juli.FileHandler.formatter = java.util.logging.SimpleFormatter

# Directory for log file output.
org.apache.juli.FileHandler.directory = ${catalina.base}/logs  # Set dynamically based on the server
    setup.

# Prefix for log files.
org.apache.juli.FileHandler.prefix = ${classloader.webappName}.

# Duration (in days) log files are retained.
org.apache.juli.FileHandler.maxDays = <max-days>  # Example: 60

################################
# Configuration of Formatters #
################################
# Specifies the output format for log entries.

java.util.logging.SimpleFormatter.format = %1$td-%1$tm-%1$tY %1$tk:%1$tM:%1$tS %4$s %2$s %5$s %6$s %n

################################
# Additional Logging Notes    #
################################
# Update 'maxDays' and other parameters as per your archival and audit requirements.
```

Listing 2: logging.properties

5.4 Sprachkonfigurationen

In unseren Java-basierten Webanwendungen werden sprachspezifische Ressourcen konsequent im Verzeichnis `src/main/resources` organisiert. Diese klare Trennung zwischen dem Quellcode und den sprachabhängigen Inhalten erleichtert die Lokalisierung unserer Softwarelösungen erheblich. Die Benutzeroberflächen unserer Anwendungen passen sich automatisch an die Spracheinstellungen der Benutzer an, indem sie dynamisch sprachspezifische Inhalte aus den Resource Bundles laden.

5.4.1 Struktur der Resource Bundles

Die Resource Bundles befinden sich in einem speziellen Unterverzeichnis: `src/main/resources/de/uni_passau/fim/talent_tausche`
Innerhalb dieses Verzeichnisses sind die `.properties`-Dateien nach Sprachen organisiert:

- `messages_de.properties` für Deutsch.
- `messages_en.properties` für Englisch.
- `messages.properties` für sprachneutrale Standardtexte.

5.4.2 Inhalt und Konventionen der Resource Bundles

Jedes Resource Bundle ist klar strukturiert und enthält spezifische Schlüssel für die Kategorisierung der Textelemente:

- **UI Labels**: Bezeichnungen für Elemente der Benutzeroberfläche wie Buttons, Menüs und Dialogfenster.
- **Hilfstexte**: Texte, die in Tooltips und auf Hilfeseiten genutzt werden und den Nutzern zusätzliche Informationen bieten.
- **Nachrichten**: Benutzerrückmeldungen wie Bestätigungen nach durchgeführten Aktionen.
- **Warnungen**: Texte für Validierungen und Fehlermeldungen.
- **Mail**: Vorlagen für automatisierte E-Mails.

5.4.3 Beispiele für Dateiinhalte

Im Folgenden werden Beispiele für Einträge in den jeweiligen `.properties`-Dateien gezeigt:

```
f_columnHeader_adCosts = Kosten
m_successMessage_login = Anmeldung erfolgreich
v_invalidMessage_emailForm = Ungültige E-Mail Adresse
h_helpText_loginPage = Wenn Sie bereits ein Nutzerkonto haben, können Sie sich hier anmelden.
Geben Sie dazu bitte Ihren Benutzernamen oder Ihre E-Mail sowie Ihr Passwort ein.
Falls Sie Ihr Passwort vergessen haben, können Sie es mit einem Klick auf "Passwort vergessen" zurü
    cksetzen.
Falls Sie noch kein Konto haben, können Sie durch Klick auf "Registrieren" eines erstellen.
ma_emailBody_verificationMail = Bitte klicken Sie auf folgenden Link, um Ihre E-Mail-Adresse zu
    verifizieren:
```

Listing 3: messages_de.properties

```
f_columnHeader_adCosts = Costs
m_successMessage_login = Login successful
v_invalidMessage_emailForm = Invalid email address
h_helpText_loginPage = If you already have a user account, you can log in here.
Please enter your username or your email address and your password.
If you have forgotten your password, you can reset it by clicking on "Forgot password".
If you do not yet have a user account, you can create one by clicking on "Register".
ma_emailBody_verificationMail = Please click on the following link to verify your email address:
```

Listing 4: messages_en.properties

```
f_appName_fileHeader = TalentTauscher
f_logoImage_placeHolder = Logo
```

Listing 5: messages.properties

5.4.4 Darstellung und Schlüsselkonventionen der Resource Bundles

Die Resource Bundles verwenden das einfache Zuweisungsformat `<Schlüssel> = <Wert>` und folgen einer klaren Konvention zur Schlüsselbenennung, was das Auffinden und Verwalten der Nachrichten erleichtert:

`<Art_der_Message><Id_oder_Art_des_Elements><Ort_des_Elements>`

Die verwendeten Präfixe stehen für:

- `f_` für **Features** oder funktionale Aspekte.
- `m_` für **Meldungen** oder Benachrichtigungen.
- `v_` für **Validierungen** oder Fehlermeldungen.
- `h_` für **Hilfstexte**.
- `ma_` für automatisierte **Mail**texte.

5.4.5 Automatische Sprachanpassung

Für die automatische Anpassung der Spracheinstellungen nutzen wir die Java-Klassen `Locale` und `ResourceBundle`, die es unserer Anwendung ermöglichen, basierend auf den Browsereinstellungen des Benutzers die geeignete Sprachdatei auszuwählen. Dies gewährleistet einen nahtlosen Sprachwechsel während des Betriebs.

Diese strukturierte Vorgehensweise gewährleistet eine effiziente und wartungsarme Lokalisierung unserer Java-Webapplikationen.

6 Facelets

Leon Föckersperger

Dieses Kapitel widmet sich der detaillierten Darstellung der Implementierung und Konfiguration von Facelets sowie den zugehörigen UI-Komponenten. Es wird besonders auf die Anpassung der Zugriffskontrollen eingegangen, die je nach Rolle des Benutzers und den spezifischen Anforderungen des Kontexts variieren. Ein vorangestelltes Symbol "|" kennzeichnet dabei, dass das entsprechende Element ein untergeordnetes Element des zuvor genannten Elements ist.

Legende:

AN = Alle Nutzer
AY = Anonyme Nutzer
N = Authentifizierte Nutzer
A = Administratoren

6.1 Template

skeleton.xhtml

Das Template stellt das Grundgerüst für die Darstellung aller Seiten in der Anwendung bereit.

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<ui:include>	header	Integriert den Kopfbereich (Header) der Anwendung aus einer externen XHTML-Datei.	-	src="/view/anonymous/header.xhtml"	AN
<ui:include>	footer	Integriert den Fußbereich (Footer) der Anwendung aus einer externen XHTML-Datei.	-	src="/view/anonymous/footer.xhtml"	AN
<ui:insert>	content	Ein Platzhalter, der durch spezifischen Inhalt auf verschiedenen Seiten der Anwendung ersetzt wird.	-	name="childContent"	AN
<ui:insert>	childMetadata	Platzhalter für View-Parameter und View-Actions der Seiten.	-	name="metadata"	AN
<h:messages>	message	Zeigt systemweite Benachrichtigungen und Fehlermeldungen in einer konsolidierten Form an.	-	globalOnly="true"	AN
<ui:include>	help	Integriert den Helptext und Helpbutton der Anwendung aus einer externen XHTML Datei.	-	src="/view/anonymous/help.xhtml"	AN

Tabelle 1: Beschreibung der Template-Komponenten im Facelet

6.2 Custom Components

6.2.1 dynamicDataTable.xhtml

Die **DynamicDataTable**-Komponente ermöglicht eine dynamische Darstellung von Daten, die direkt aus einer Datenquelle geladen werden. Nutzer können durch Daten navigieren und bestimmte Einträge gezielt auswählen. Sie unterstützt außerdem Paginierung, um die Benutzerfreundlichkeit beim Durchsuchen großer Datenmengen zu verbessern.

Name	Type	Required
dataModel	de.uni_passau.fim.talent_tauscher.model.backing.util.PaginatedDataModel	true

Tabelle 2: Attribute des Composite Components

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:dataTable>	dataTable	Stellt die Daten in einer tabellarischen Ansicht dar.	-	value="#{cc.attrs.dataModel.fetchData()}" var="item"	AN
<h:commandButton>	filterButton	Button zur Anwendung von Filterkriterien auf die angezeigten Daten.	-	action="#{cc.attrs.dataModel.refresh()}"	AN
<h:commandButton>	previousButton	Button für den Wechsel zur vorherigen Seite.	disabled="#{!cc.attrs.dataModel.hasPreviousPage()}"	value="messages.f_previousButton_dataTable" action="#{cc.attrs.dataModel.previousPage()}"	AN
<h:inputText>	pageInput	Eingabefeld zur manuellen Eingabe einer Seitenzahl, um direkt zu dieser Seite zu springen.	required="true"	value="#{cc.attrs.dataModel.pagination.page}"	AN
<h:outputText>	totalPages	Zeigt die aktuelle Seitenzahl und die Gesamtzahl der Seiten.	-	value="of #{cc.attrs.dataModel.pagination.totalPages}"	AN
<h:commandButton>	nextButton	Button für den Wechsel zur nächsten Seite.	disabled="#{!cc.attrs.dataModel.hasNextPage()}	action="#{cc.attrs.dataModel.nextPage()}"	AN

Tabelle 3: Beschreibung der Komponenten in der **dynamicDataTable** Composite Component

6.2.2 sortFilterColumn.xhtml

Die **sortFilterColumn** ermöglicht das Sortieren und Filtern von Tabellendaten nach spezifischen Kriterien. Nutzer können Spalten auf- oder absteigend sortieren und Filter anwenden, um die angezeigten Daten zu verfeinern.

Name	Type	Required
dataModel	de.uni_passau.fim.talent_tauscher.model.backing.util.PaginatedDataModel	true
columnName	java.lang.String	true
columnIdentifier	java.lang.String	true

Tabelle 4: Attribute des Composite Components

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:commandLink>	sortLink	Schaltfläche zum Sortieren der Spalte, zeigt den Spaltennamen an und ändert das Icon basierend auf dem Sortierzustand.	-	action="#{cc.attrs.dataModel. setSorting(cc.attrs.columnIdentifier)}"	AN
| <h:outputText>	columnName	Headername der Spalte wird vom Facelet übergeben.	-	value="#{cc.attrs.columnName}"	AN
| <h:outputText>	sortIconAsc	Zeigt einen aufsteigenden Pfeil, wenn die Spalte aufsteigend sortiert ist.	rendered="#{cc.attrs.dataModel. pagination.sortBy eq cc.attrs.columnIdentifier and cc.attrs.dataModel.pagination. sortAscending}"	value="&#9650"	AN
| <h:outputText>	sortIconDesc	Zeigt einen absteigenden Pfeil, wenn die Spalte absteigend sortiert ist.	rendered="#{cc.attrs.dataModel. pagination.sortBy eq cc.attrs.columnIdentifier and cc.attrs.dataModel.pagination. sortDescending}"	value="&#9660"	AN
<h:inputText>	filterInput	Eingabefeld für den Filterwert, das den aktuellen Filterwert an die Spalte bindet und die Daten entsprechend filtert.	-	value="#{cc.attrs.dataModel. pagination.filters[cc.attrs. columnIdentifier]}"	AN

Tabelle 5: Beschreibung der Komponenten in der **sortFilterColumn** Composite Component

6.2.3 deleteWarnMessage.xhtml

Dieser Abschnitt beschreibt das Composite Component **deleteWarnMessage**, das noch in der Konzeptphase ist und daher möglicherweise nicht vollständig konsistent mit den übrigen Kapiteln ist. Aufgrund zeitlicher Einschränkungen konnte bisher kein vollständiger Abgleich mit anderen Komponenten erfolgen. Trotzdem ist geplant, dieses Component in der Implementierungsphase zu integrieren. **deleteWarnMessage** soll dem Benutzer eine kritische Warnmeldung anzeigen, bevor eine Löschaktion bestätigt wird. Es findet vor allem in Verwaltungsinterfaces Verwendung, um sicherzustellen, dass Nutzer über die möglichen Konsequenzen einer Löschung, einschließlich der Anzahl betroffener Elemente, umfassend informiert sind. Diese Funktionalität dient dazu, unbeabsichtigte Löschungen zu verhindern und eine bewusste Interaktion mit der Anwendung zu fördern.

Name	Type	Required
backingBean	de.uni_passau.fim.talent_tauscher.model.backing.beans.AbstractDeleteBean	true

Tabelle 6: Attribute des Composite Components

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:outputText>	warnMessageDeletion	Zeigt eine Warnmeldung an, die den Nutzer darüber informiert, wie viele Elemente gelöscht werden, wenn diese Entität entfernt wird.	-	value="messages.v_warnMessageDeletion1_AbstractDeleteBean #{cc.attrs.backingBean.deletedItems} messages.v_warnMessageDeletion2_AbstractDeleteBean"	N
<h:commandButton>	confirmDeletionButton	Ein Button, der die Löschung bestätigt und die entsprechende Aktion im Backing Bean ausführt.	-	value="messages.f_confirmDeletionButton_AbstractDeleteBean" action="#{cc.attrs.backingBean.delete()}"	N
<h:commandButton>	cancelDeletionButton	Ein Button, der die Löschaktion abbricht und den Nutzer zur vorherigen Ansicht zurückführt.	-	value="messages.f_cancelDeletionButton_AbstractDeleteBean" action="#{cc.attrs.backingBean.cancel()}"	N

Tabelle 7: Beschreibung der Komponenten in der **deleteWarnMessage** Composite Component

6.3 Seiten von Leon Föckersperger

6.3.1 header.xhtml

Der Header wird konsistent über alle Seiten hinweg eingefügt und bietet Zugang zu Authentifizierungsdialogen sowie weiteren wichtigen Bereichen der Webseite wie Nutzerverwaltung und persönliche Benutzeranzeigen. Jede Schaltfläche im Header ist bedingungsabhängig gerendert, basierend auf dem Login-Status und den Berechtigungen des Nutzers.

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:link>	logoLink	Logo-Link, zurück zur Startseite	-	outcome="/view/anonymous/ allAdvertisements"	AN
| <o:graphicImage>	headerLogo	Logo-Bild	-	value="#{headerBean. systemData.logo} dataURI="true"	AN
<h:form>	-		rendered="#{userSession. isUserLoggedIn()}"	-	N
| <h:link>	profileLink	Link zur Profilseite des angemeldeten Nutzers.	-	outcome="/view/registered/profile" value="messages.f_profileLink _header"	N
<h:form>	-		rendered="#{userSession. isUserLoggedIn()}"	-	N
| <h:link>	createdAdsLink	Link zur Seite mit den vom Nutzer erstellten Anzeigen.	-	outcome="/view/registered /createdAdvertisements" value="messages.f_createdAdsLink _header"	N
<h:form>	-		rendered="#{userSession. isUserLoggedIn()}"	-	N
| <h:link>	claimedAdsLink	Link zur Seite mit den vom Nutzer beanspruchten Anzeigen.	-	outcome="/view/registered /claimedAdvertisements" value="messages.f_claimedAdsLink _header"	N
<h:form>	-		rendered="#{userSession. isUserLoggedIn()}"	-	AY
| <h:link>	userManagement Link	Link zur Seite mit der User Verwaltung.	-	outcome="/view/admin/ userAdministration" value="messages. f_userManagementLink_header"	AY
<h:commandButton>	loginButton	Login Button welcher zur Login Seite weiterleitet.	rendered="#{!userSession. isUserLoggedIn()}"	value="messages.f_loginButton _header" action="#{headerBean.goToLogin()}"	AY
<h:commandButton>	logoutButton	Logout Button welcher den angemeldeten User wieder ausloggt.	rendered="#{userSession. isUserLoggedIn()}"	value="messages.f_logoutButton _header" action="#{headerBean.logout()}"	N

Tabelle 8: Beschreibung der Komponenten des **header** Facelets

6.3.2 footer.xhtml

Der Footer bietet Zugriff auf wichtige organisatorische Informationen wie Datenschutzerklärung, Kontaktseite und Impressum.

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:link>	privacyPolicyLink	Link zur Datenschutzerklärung	-	outcome="/view/public/privacyPolicy" value="messages.f_privacyPolicyLink_footer"	AN
<h:link>	contactUsLink	Link zur Kontaktseite	-	outcome="/view/public/contactUs" value="messages.f_privacyPolicyLink_footer"	AN
<h:link>	imprintLink	Link zum Impressum	-	outcome="/view/public/imprint" value="messages.f_privacyPolicyLink_footer"	AN

Tabelle 9: Beschreibung der Komponenten des **footer** Facelets

6.3.3 help.xhtml

Die Help-Komponente bietet Nutzern Unterstützung durch Hilfetexte und -dialoge, die auf Anfrage sichtbar gemacht werden können.

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<f:form>	-		rendered="#{helpBean.help.visible}"	-	AN
| <h:outputText>	helpText	Zeigt Hilfetext basierend auf dem aktuellen Kontext der Anwendung an.	-	value="#{helpBean.help.text}"	AN
<h:commandButton>	helpButton	Schaltfläche zum Anzeigen oder Verbergen des Hilfetextes.	-	action="#{helpBean.toggleHelpText}" value="#{messages.f_helpButton_help}"	AN

Tabelle 10: Beschreibung der Komponenten des **help** Facelets

6.4 Seiten

David Sturm

Für alle Seiten wird skeleton.xhtml benutzt um das Layout der Seite aufzubauen. Da Header und Footer damit bereits eingebaut sind, sind folgende Bedienelemente also als Zusatz im Main Content zu betrachten. Die Seiten werden aufsteigend nach den benötigten Zugriffsrechten gruppiert, somit kann jede Seite, außer explizit angegeben, ebenfalls von den Nutzern der folgenden Sektionen aufgerufen werden.
Elemente mit dem Elementtyp "<h:inputText>" so wie <o:inputFile> implizieren immer die existenz eines Elements des Typs <h:message> mit dem Binding for="[Id des Inputfeldes]", welche ggf. Facesmessages für das ihnen zugehörige Element wiedergeben.

Per Konvention haben alle Elemente mit den folgenden ebenfalls die dazugehörigen vordefinierten Validatoren bzw. Converter welche für die Übersichtlichkeit in den Tabellen weggelassen wurden:
required="true" $\Longrightarrow$ <f:validateRequired>

maxlength=“x” $\Longrightarrow$ <f:validateLength maximum=“x”/>
converter=“convertDateTime” $\Longrightarrow$ <f:convertDateTime type=“localDate” pattern=“dd.MM.yyyy”/>

Außerdem sind Elemente des Typs “<f:viewParam>” so wie “<f:viewAction>” wie folgt in einem “<ui:define name="metadata»” deklariert:

```
<ui:define name=”metadata”>
  <f:metadata>
    [ Block mit allen “<f:viewParam>”/“<f:viewAction>” ]
  </f:metadata>
</ui:define>
```

Bei den Beschreibungstabellen für DynamicDataTables beschreibt die vorangetellte Anzahl der “|” ob es sich um Elemente innerhalb des Vorangegangenen Elements handelt oder um ein neues. Des weiteren werden die einzelnen Spalten für die Übersichtlichkeit mit einer Doppellinie getrennt. Beispielhaft wir hier eine Spalte einer Tabelle mit der dazugehörigen Beschreibung angegeben:

```
<h:column scope=“col”>
  <f:facet name=“header”>
    <h:outputText value=“#{messages.image}”/>
  </f:facet>
  <o:graphicImage dataURI=“true” value=“#{item.image}”/>
</h:column>
```

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	allAdvertisements: image	Spalte für die Titelbiler.	scope="col"		AN
| <f:facet>	allAdvertisements: imageHeader	Titel der Spalte der Titelbilder.		name="header"	AN
| | <h:outputText>		Titel der Spalte der Titelbilder.		value=“#{messages.image}”	AN
| <o:graphicImage>	allAdvertisements: imageImage	Bild der Anzeige.	dataURI=“true”	value=“#{item.image}”	AN

6.4.1 Anonyme Nutzer

Die folgenden Seiten können von allen Nutzern, somit insbesondere von anonymen Nutzern, aufgerufen werden.

Impressum Diese Seite ermöglicht dem Nutzer das Impressum einzusehen. Der Admin hat hier die Möglichkeit das Impressum zu bearbeiten.

imprint.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:outputText>	imprint:imprint	Das vom Admin als in HTML festgelegte Impressum.	escape="false" rendered="#{not userSession .isUserAdmin()}"	value="#{imprintBean.imprint}"	AY /N
<h:inputText>	imprint:new-Imprint	Editierfeld für den Admin um das Impressum zu ändern.	escape="false" rendered= "#{userSession.isUserAdmin()}"	value="#{imprintBean.imprint}"	A
<h:commandButton>	imprint:update	Updated das gespeicherte Impressum	rendered= "#{userSession.isUserAdmin()}"	action="#{imprintBean .submitNewImprint()}" value="#{messages.submit}"	A

Kontakt Diese Seite ermöglicht dem Nutzer die Kontaktdaten des Admins einzusehen. Der Admin hat hier die Möglichkeit die Daten zu bearbeiten.

contactUs.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:outputText>	contact:contact	Das vom Admin als in HTML festgelegte Kontaktformular.	escape="false" rendered="#{not userSession .isUserAdmin()}"	value="#{contactBean.contact}"	AY /N
<h:inputText>	imprint:new-Contact	Editierfeld für den Admin um das Kontaktformular zu ändern.	escape="false" rendered= "#{userSession.isUserAdmin()}"	value="#{contactBean.contact}"	A
<h:commandButton>	contact:update	Updated das gespeicherte Kontaktformular	rendered= "#{userSession.isUserAdmin()}"	action="#{contactBean .submitNewContact()}" value="#{messages.submit}"	A

Datenschutz Diese Seite ermöglicht dem Nutzer den Datenschutzeintrag einzusehen. Der Admin hat hier die Möglichkeit den Datenschutzeintrag zu bearbeiten.

privacyPolicy.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:outputText>	privacyPolicy: privacyPolicy	Die vom Admin als in HTML festgelegte Datenschutzerklärung.	escape="false" rendered="#{userSession .isUserAdmin()}"	value="#{privacyPolicyBean .privacyPolicy}"	AY /N
<h:inputText>	privacyPolicy: newprivacyPolicy	Editierfeld für den Admin um die Datenschutzerklärung zu ändern.	escape="false" rendered= "#{userSession.isUserAdmin()}"	value="#{privacyPolicyBean .privacyPolicy}"	A
<h:commandButton>	privacyPolicy: update	Updated die gespeicherte Datenschutzerklärung	rendered= "#{userSession.isUserAdmin()}"	action="#{privacyPolicyBean .submitNewprivacyPolicy()}" value="#{messages.submit}"	A

Login Die Login Seite bietet dem Nutzer die Möglichkeit seine Anmeldedaten einzugeben oder zur Registrieren Seite zu wechseln. Optional kann auch das Passwort zurückgesetzt werden. Diese Seite kann nur von anonymen Nutzern aufgerufen werden.

login.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:inputText>	login:nick	Inputfeld für den Nutzernamen oder die Email.	required="true" maxlength="255"	value="#{loginBean.login.nick}"	AY
<h:inputSecret>	login:password	Inputfeld für das Passwort.	required="true" maxlength="255"	value="#{loginBean .loginBean.password}"	AY
<h:commandButton>	login:submit	Löst den Loginvorgang aus. Bei Erfolg wird auf die Seite "Eigene Anzeigen" weitergeleitet. Bei Misserfolg wird dem Benutzer eine entsprechende Fehlermelung angezeigt.		action="#{loginBean.submit()}"	AY
<h:link>	login:linkRegister	Link welcher zur Registrierungsseite weiterleitet.		outcome="/view/anonymous/ register" value="#{messages.loginLink}"	AY
<h:link>	login: linkResetPassword	Link welcher zur Seite für die Passwort-Rücksetzung weiterleitet.		outcome="/view/anonymous/ passwordResetRequest" value="#{messages .passwordResetLink}"	AY

Passwort Rücksetzung Anfrage Diese Seite ermöglicht dem Nutzer eine Email anzugeben, deren Account eine Passwort Rücksetzung benötigt. Diese Seite kann nur von anonymen Nutzern aufgerufen werden.

passwordResetRequest.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:inputText>	passwordReset-Request:email	Inputfeld für den die Email.	required="true" maxlength="70" validator= "emailFormatValidator"	value="#{passwordResetRequest-Bean.verification.emailAddress}"	AY
<h:commandButton>	passwordReset-Request:submit	Löst den Passwort Rücksetzungs-vorgang aus.		action="#{passwordReset-RequestBean.submit()}" value="#{messages.submit}"	AY

Passwort Rücksetzung Passworteingabe Diese Seite ermöglicht dem Nutzer ein neues Passwort anzugeben, sofern eine Passwort Rücksetzung läuft. Diese Seite kann nur von anonymen Nutzern mittels einem per E-Mail erhaltenen Link aufgerufen werden.

passwordResetInput.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<f:viewParam>		Secret aus dem Link für die Authentifizierung		name="secret" value="#{passwordReset-InputBean.passwordReset .verificationToken}"	
<h:inputSecret>	passwordReset-Input:password	Inputfeld für das neue Passwort.	required="true"	value="#{passwordResetInput-Bean.passwordInput.password}"	AY
<h:inputSecret>	passwordReset-Input:repetition	Inputfeld für die Wiederholung des neuen Passworts.	required="true"	value="#{passwordResetInput-Bean..passwordInput.repetition}"	AY
<h:commandButton>	passwordReset-Input:submit	Speichert das neue Passwort und leitet auf die Seite "Login" weiter.		action="#{passwordReset-RequestBean.submit()}" value="#{messages.submit}"	AY

Registrieren Die Registrieren Seite bietet dem Nutzer die Möglichkeit sich zu registrieren oder zur Login Seite zu wechseln. Diese Seite kann nur von anonymen Nutzern aufgerufen werden.

register.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:inputText>	register:nick	Inputfeld für den Nutzernamen.	required="true" maxlength="255" validator= "nicknameUniqueValidator"	value="#{registerBean.user .nickname}"	AY
<h:inputSecret>	register:password	Inputfeld für das Passwort.	required="true" maxlength="255"	value="#{registerBean .passwordInput.password}"	AY
<h:inputSecret>	register:repetition	Inputfeld für die Wiederholung des Passworts.	required="true" maxlength="255"	value="#{registerBean .passwordInput.repetition}"	AY
<h:inputText>	register:email	Inputfeld für die E-Mail.	required="true" maxlength="70" validator= "emailFormatValidator" validator= "emailUniqueValidator"	value="#{registerBean.user .email}"	AY
<h:inputText>	register:firstname	Inputfeld für den Vornamen.	required="true" maxlength="50"	value="#{registerBean.user .firstname}"	AY
<h:inputText>	register:lastname	Inputfeld für den Nachnamen.	required="true" maxlength="50"	value="#{registerBean.user .lastname}"	AY
<h:inputText>	register:street	Inputfeld für die Straße.	required="true" maxlength="150"	value="#{registerBean.user .street}"	AY
<h:inputText>	register: houseNumber	Inputfeld für die Hausnummer.	required="true" maxlength="10"	value="#{registerBean.user .houseNumber}"	AY
<h:inputText>	register:city	Inputfeld für die Stadt.	required="true" maxlength="150"	value="#{registerBean.user .city}"	AY
<h:inputText>	register: postalCode	Inputfeld für die Postleitzahl.	required="true" maxlength="25"	value="#{registerBean.user .postalCode}"	AY
<h:inputText>	register:phone	Inputfeld für die Telefonnummer.	maxlength="20"	value="#{registerBean.user .phone}"	AY
<h:commandButton>	register:submit	Startet den Registriervorgang. Bei Erfolg wird auf die Seite "Übersicht" weitergeleitet. Bei Misserfolg wird dem Benutzer eine entsprechende Fehlermelung angezeigt.		action="#{registerBean.submit()}" value="#{messages.submit}"	AY
<h:link>	register:loginLink	Link welcher zur Seite für den Login weiterleitet.		outcome="/view/anonymous/login" value="#{messages.loginLink}"	AY

Email Verifikation Diese Seite ermöglicht dem Nutzer den Code aus der Registrierungsemail einzugeben, um die Registrierung abzuschließen. Das InputField wird dabei falls möglich mit einem Code aus den View Parametern vorbefüllt.

emailVerification.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<f:viewParam>		Secret aus dem Link für die Email Verifikation.		name="secret" value="#{EmailVerification-Bean.token.token}"	
<f:viewAction>		Führt die Athentifizierung des Tokens aus.		action="#{EmailVerification-Bean.loadData()}"	
<h:link>	emailVerification: loginLink	Link welcher zur Seite für den Login weiterleitet.		outcome="/view/anonymous/login" value="#{messages.loginLink}"	AY

Übersicht/Suche Diese Seite ermöglicht dem Nutzer alle Anzeigen einzusehen und zu durchsuchen.

allAdvertisements.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<comp: dynamicDataTable>	allAdvertisements: table	Tabelle mit Paginierung für die Anzeigen.		dataModel= "#{allAdvertisementsBean .dataModel}"	AN

Beschreibungstabelle für die DynamicDataTable:

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	allAdvertisements: image	Spalte für die Titelbiler.	scope="col"		AN
| <f:facet>	allAdvertisements: imageHeader	Kopf der Spalte der Titelbilder.		name="header"	AN
| | <h:outputText>		Titel der Spalte der Titelbilder.		value="#{messages.image}"	AN
| <o:graphicImage>		Bild der Anzeige.	dataURI="true"	value="#{item.image}"	AN
<h:column>	allAdvertisements: adTitle	Spalte für die Anzeigentitel.	scope="col"		AN
| <f:facet>	allAdvertisements: titleHeader	Kopf der Spalte der Anzeigentitel mit Filterfunktion.		name="header"	AN
| | <comp: sortFilterColumn>	allAdvertisements: ad-title			datamodel= "#{allAdvertisementsBean .dataModel}" columnName="#{messages.title}" columnIdentifier="title"	AN
| <h:outputText>		Titel der Anzeige.		value="#{item.title}"	AN
<h:column>	allAdvertisements: adCosts	Spalte für die benötigten Talentpunkte.	scope="col"		AN
| <f:facet>	allAdvertisements: costsHeader	Kopf der Spalte die benötigten Talentpunkte mit Filterfunktion.		name="header"	AN
| | <comp: sortFilterColumn>	allAdvertisements: ad-costs			datamodel= "#{allAdvertisementsBean .dataModel}" columnName="#{messages.costs}" columnIdentifier= "cost_in_talentPoints"	AN
| <h:outputText>		Kosten der Anzeige in Talentpunkten.		value="#{item.cost}"	AN
<h:column>	allAdvertisements: adPostal	Spalte für die Postleitzahl.	scope="col"		AN
| <f:facet>	allAdvertisements: costsHeader	Kopf der Spalte die Postleitzahl mit Filterfunktion.		name="header"	AN
| | <comp: sortFilterColumn>	allAdvertisements: ad-postal-code			datamodel= "#{allAdvertisementsBean .dataModel}" columnName= "#{messages.postalCode}" columnIdentifier="postal_code"	AN
| <h:outputText>		Postleitzahl der Anzeige.		value="#{item.postalCode}"	AN

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	allAdvertisements: adCity	Spalte für die Stadt der Anzeige.	scope="col"		AN
| <f:facet>	allAdvertisements: adCityHeader	Kopf der Spalte für die Stadt der Anzeige mit Filterfunktion.		name="header"	AN
| | <comp: sortFilterColumn>	allAdvertisements: ad-city			datamodel= "#{allAdvertisementsBean .dataModel}" columnName="#{messages.city}" columnIdentifier="city"	AN
| <h:outputText>		Postleitzahl der Anzeige.		value="#{item.city}"	AN
<h:column>	allAdvertisements: editAdvertisement	Spalte mit Editieren-Buttons für den Administrator.	scope="col" rendered= "#{userSession.isUserAdmin}"		A
| <f:facet>	allAdvertisements: editAdvertisement Header	Kopf der Spalte mit Editieren-Buttons für den Administrator.		name="header"	A
| | <h:outputText>		Titel der Spalte mit Editieren-Buttons für den Administrator.		value="#{messages.edit}"	A
| <h:button>		Button zum Editieren der Anzeige.		outcome="/view/registered /editAdvertisement" includeViewParams="true" <f:param name="id" value= "#{item.id}" /> value="#{messages.edit}"	A

Anzeigen Details Diese Seite ermöglicht dem Nutzer eine Anzeigen genauer anzusehen. Von hier aus werden auch Anfragen getätigt.

advertisementDetails.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<f:viewParam>		Id der Anzeige.		name="id" value="#{advertisementDetails-Bean.advertisement.id}"	
<f:viewAction>		Lädt die Anzeige..		action="#{advertisementDetails-Bean.loadData()}"	
<o:graphicImage>	advertisement-Details:image	Titelbild der Anzeige.	dataURI="true"	value="#{advertisementDetails-Bean.advertisement.image}"	AN
<o:graphicImage>	advertisement-Details:avatar	Profilbild des Anzeigenerstellert.	dataURI="true"	value="#{advertisementDetails-Bean.adCreator.avatar}"	AN
<h:outputText>	advertisement-Details:user	Nutzername des Anzeigenerstellers.		value="#{advertisementDetails-Bean.adCreator.nickname}"	AN
<h:outputText>	advertisement-Details:name	Voller Name des Anzeigenerstellers.	rendered="#{advertisement-DetailsBean.advertisement .nameShown}"	value="#{advertisementDetails-Bean.adCreator.firstname} Bean.adCreator.lastname}"	AN
<h:outputText>	advertisement-Details:title	Titel der Anzeige.		value="#{advertisementDetails-Bean.advertisement.title}"	AN
<h:outputText>	advertisement-Details:description	Beschreibung der Anzeige.		value="#{advertisementDetails-Bean.advertisement.description}"	AN
<h:outputText>	advertisement-Details:city	Stadt der Anzeige.		value="#{advertisementDetails-Bean.advertisement.city}"	AN
<h:outputText>	advertisement-Details:postalCode	Postleitzahl der Anzeige.		value="#{advertisementDetails-Bean.advertisement.postalCode}"	AN
<h:outputText>	advertisement-Details:endDate	Ablaufdatum der Anzeige.	converter="convertDateTime"	value="#{advertisementDetails-Bean.advertisement.endDate}"	AN
<h:outputText>	advertisement-Details:street	Straße der Anzeige.	rendered="#{advertisement-DetailsBean.advertisement .streetShown}"	value="#{advertisementDetails-Bean.advertisement.street}"	N / A
<h:outputText>	advertisement-Details:email	E-mail-Adresse des Anzeigenerstellers.		value="#{advertisementDetails-Bean.advertisement.email}"	N / A
<h:outputText>	advertisement-Details:phone	Telefonnummer des Anzeigenerstellers.	rendered="#{advertisement-DetailsBean.advertisement .phoneNumberShown}"	value="#{advertisementDetails-Bean.advertisement.phonenumber}"	N / A
<h:commandButton>	advertisement-Details:request	Button für das schreiben einer Anfrage.		action="#{advertisementDetails-Bean.goToWriteRequest()}"	N / A
<h:link>	advertisement-Details:allAdsLink	Link zurück zu allen Anzeigen		outcome="/view/anonymous /allAdvertisements" value="#{messages .allAdvertisementsLink}"	AN

6.4.2 Angemeldete Nutzer

Die folgenden Seiten können von angemeldeten Nutzern aufgerufen werden.

Mein Profil Auf dieser Seite befinden sich alle notwendigen Bedienelemente, um einen Benutzer zu bearbeiten oder zu löschen. Alle Felder sind zu beginn mit den aktuellen Werten vorbefüllt.

profile.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<o:graphicImage>	profile:avatar	Aktuelles Profilbild/Preview für das neue Profilbild.	dataURI="true"	value="#{profileBean.user.avatar}"	N / A
<o:inputFile>	profile:newAvatar	File input für neues Profilbild. Bei Auswahl einer passenden Datei wird das aktuelle Profilbild mittels eines ajax calls mit einer Vorschau des neuen ersetzt.	validator="avatarValidator" accept="image/png,image/jpeg"	value="#{profileBean.user.avatar}" <f:ajax render="@form"/>	N / A
<h:commandButton>	profile: deleteAvatar	Löscht das aktuelle Profilbild.		action="#{profileBean .deleteAvatar()}" value="#{messages.deleteAvatar}"	N / A
<h:inputText>	profile:nick	Inputfeld für den Nutzernamen.	required="true" maxlength="255" validator= "nicknameUniqueValidator" validator= "nicknameFormatValidator"	value="#{profileBean.user .nickname}"	N / A
<h:inputSecret>	profile:password	Inputfeld für das Passwort.	required="true" maxlength="255"	value="#{profileBean .passwordInput.password}"	N / A
<h:inputSecret>	profile:repetition	Inputfeld für die Wiederholung des Passworts.	required="true" maxlength="255"	value="#{profileBean .passwordInput.repetition}"	N / A
<h:inputText>	profile:email	Inputfeld für die E-Mail.	required="true" maxlength="70" validator= "emailFormatValidator" validator= "emailUniqueValidator"	value="#{profileBean.user .email}"	N / A
<h:inputText>	profile:firstname	Inputfeld für den Vornamen.	required="true" maxlength="50"	value="#{profileBean.user .firstname}"	N / A
<h:inputText>	profile:lastname	Inputfeld für den Nachnamen.	required="true" maxlength="50"	value="#{profileBean.user .lastname}"	N / A
<h:inputText>	profile:street	Inputfeld für die Straße.	required="true" maxlength="150"	value="#{profileBean.user .street}"	N / A
<h:inputText>	profile: houseNumber	Inputfeld für die Hausnummer.	required="true" maxlength="10"	value="#{profileBean.user .houseNumber}"	N / A

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:inputText>	profile:city	Inputfeld für die Stadt.	required="true" maxlength="150"	value="#{profileBean.user .city}"	N / A
<h:inputText>	profile: postalCode	Inputfeld für die Postleitzahl.	required="true" maxlength="25"	value="#{profileBean.user .postalCode}"	N / A
<h:inputText>	profile: addressAddition	Inputfeld für den Addresszusatz.	maxlength="40"	value="#{profileBean.user .addressAddition}"	N / A
<h:inputText>	profile:phone	Inputfeld für die Telefonnummer.	maxlength="20"	value="#{profileBean.user .phone}"	N / A
<h:commandButton>	profile:submit	Speichert die aktuellen Daten.		action="#{profileBean.submit()}" value="#{messages.submit}"	N / A
<h:commandButton>	profile:delete	Löscht das Profil.		action="#{profileBean.deleteUser()}" value="#{messages.delete}"	N / A
<h:link>	profile: createdAdsLink	Link, welcher zur Seite für die selbst erstellten Anzeigen weiterleitet.		outcome="/view/registered/ createdAdvertisements" value="#{messages .createdAdvertisementsLink}"	N / A

Anzeige erstellen/bearbeiten/löschen Auf dieser Seite befinden sich alle notwendigen Bedienelemente, um eine Anzeige zu erstellen, zu bearbeiten oder zu löschen. Sollte eine Anzeige erstellt werden, sind alle Felder zu beginn leer. Sollte eine Anzeige bearbeitet werden, sind alle Felder zu beginn mit den aktuellen Werten vorbefüllt.

editAdvertisement.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<f:viewParam>		Id der Anzeige.		name="id" value="#{editAdvertisementBean .advertisement.id}"	
<f:viewAction>		Lädt die Anzeige.		action="#{editAdvertisementBean .loadData()}"	
<o:graphicImage>	editAdvertisement: Image	Aktuelles Titelbild/Preview für das hochgeladene Titelbild.	dataURI="true" rendered="#{not empty editAdvertisementBean .advertisement.image}"	value="#{editAdvertisementBean .advertisement.image}"	N / A
<o:inputFile>	editAdvertisement: newImage	File input für neues Titelbild. Bei Auswahl einer passenden Datei wird das aktuelle Titelbild mittels eines ajax calls mit einer Vorschau des neuen ersetzt.	validator= "advertisementImageValidator" accept="image/png,image/jpeg"	value="#{editAdvertisementBean .advertisement.image}" <f:ajax render="@form"/>	N / A
<h:commandButton>	editAdvertisement: deleteImage	Löscht das aktuelle Titelbild.		action="#{editAdvertisementBean .deleteImage()}" value="#{messages.deleteImage}"	N / A

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:inputText>	editAdvertisement: titel	Inputfeld für den Titel.	required="true" maxlength="255"	value="#{editAdvertisementBean .advertisement.title}"	N / A
<h:inputText>	editAdvertisement: cost	Inputfeld für die Kosten.	required="true" maxlength="15" onkeypress="if(event.which < 48 || event.which > 57) return false;" <f:convertNumber integerOnly="true" type="number" />	value="#{editAdvertisementBean .advertisement.cost}"	N / A
<h:inputText>	editAdvertisement: description	Inputfeld für die Beschreibung.	required="true" maxlength="2047"	value="#{editAdvertisementBean .advertisement.description}"	N / A
<h:inputText>	editAdvertisement: city	Inputfeld für die Stadt.	required="true" maxlength="150"	value="#{editAdvertisementBean .advertisement.city}"	N / A
<h:inputText>	editAdvertisement: postalCode	Inputfeld für die Postleitzahl.	required="true" maxlength="25"	value="#{editAdvertisementBean .advertisement.postalCode}"	N / A
<h:inputText>	editAdvertisement: street	Inputfeld für die Straße.	maxlength="150"	value="#{editAdvertisementBean .advertisement.street}"	N / A
<h:inputText>	editAdvertisement: email	Inputfeld für die E-Mail.	required="true" maxlength="70" validator= "emailUniqueValidator"	value="#{editAdvertisementBean .advertisement.email}"	N / A
<h:inputText>	editAdvertisement: phoneNumber	Inputfeld für die Telefonnummer.	maxlength="20"	value="#{editAdvertisementBean .advertisement.phoneNumber}"	N / A
<h:inputText>	editAdvertisement: begin	Inputfeld für das Startdatum.	maxlength="10" converter="convertDateTime"	value="#{editAdvertisementBean .advertisement.startDate}"	N / A
<h:inputText>	editAdvertisement: end	Inputfeld für das Enddatum.	maxlength="10" converter="convertDateTime"	value="#{editAdvertisementBean .advertisement.endDate}"	N / A
<h:selectBoolean Checkbox>	editAdvertisement: showPhoneNumber	Checkbox, ob die Telefonnummer in der Anzeige angezeigt wird.		value="#{editAdvertisementBean .advertisement .phoneNumberShown}"	N / A
<h:selectBoolean Checkbox>	editAdvertisement: showName	Checkbox, ob der Name des Erstellers in der Anzeige angezeigt wird.		value="#{editAdvertisementBean .advertisement .nameShown}"	N / A
<h:selectBoolean Checkbox>	editAdvertisement: showStreet	Checkbox, ob die Straße in der Anzeige angezeigt wird.		value="#{editAdvertisementBean .advertisement .streetShown}"	N / A
<h:selectBoolean Checkbox>	editAdvertisement: hidden	Checkbox, ob die Anzeige anderen Nutzern angezeigt wird.		value="#{editAdvertisementBean .advertisement .hidden}"	N / A

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:commandButton>	editAdvertisement: submit	Erstellt/Ändert die Anzeige.		action="#{editAdvertisementBean .submit()}" value="#{messages.submit}"	N / A
<h:commandButton>	editAdvertisement: delete	Löscht die Anzeige.		action="#{editAdvertisementBean .deleteAdvertisement()}" value="#{messages.delete}"	N / A
<h:link>	editAdvertisement: :created-AdsLink	Link, welcher zur Seite für die selbst erstellten Anzeigen weiterleitet.		outcome="/view/registered/ createdAdvertisements" value="#{messages .createdAdvertisementsLink}"	N / A

Anfrage beantworten Diese Seite ermöglicht dem Nutzer auf eine Anfrage zu antworten.

writeResponse.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<f:viewParam>		Id der Anfrage.		name="id" value="#{writeResponseBean .request.id}"	
<f:viewAction>		Lädt die Anfrage.		action="#{writeResponseBean .loadData()}"	
<h:outputText>	writeResponse: request	Text der Anfrage.		value="#{writeResponseBean .request.request}"	N / A
<h:inputText>	writeResponse: response	Inputfeld für den Freitext der Antwort.		value="#{writeResponseBean .request.response}"	N / A
<h:commandButton>	writeResponse: accept	Akzeptiert die Anfrage und schickt die Nachricht.		action="#{writeResponseBean .accept()}" value="#{messages.accept}"	N / A
<h:commandButton>	writeResponse: reject	Lehnt die Anfrage ab und schickt die Nachricht.		action="#{writeResponseBean .reject()}" value="#{messages.reject}"	N / A
<h:link>	writeResponse: incomingRequests-Link	Link, welcher zur Seite für die eingegangenen Anfragen weiterleitet.		outcome="/view/registered/ incomingRequests" value="#{messages .incomingRequestsLink}"	N / A

Eingegangene Anfragen + Antworten Diese Seite ermöglicht dem Nutzer, einkommende Anfragen und die gegebenen Antworten einzusehen.

incomingRequests.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<f:viewParam>		Id des Nutzers.		name="id" value="#{incomingRequestsBean .user.id}"	
<f:viewAction>		Lädt die eingegangenen Anfragen des Nutzers.		action="#{incomingRequestsBean .loadData()}"	
<comp: dynamicDataTable>	incomingRequests: table	Tabelle mit Paginierung für die Anfragen.		dataModel= "#{incomingRequestsBean .dataModel}"	N / A
<h:link>	incomingRequests: createdAdsLink	Link, welcher zur Seite für die selbst erstellten Anzeigen weiterleitet.		outcome="/view/registered/ createdAdvertisements" value="#{messages .createdAdvertisementsLink}"	N / A
<h:link>	incomingRequests: user Administration Link	Link, welcher zur Seite für die Benutzerverwaltung weiterleitet.	rendered= "#{userSession.isUserAdmin}"	outcome="/view/admin/ userAdministration" value="#{messages .userAdministrationLink}"	A

Beschreibungstabelle für die DynamicDataTable:

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	incomingRequests: user	Spalte für den Benutzername des Anzeigenstellers.	scope="col"		N / A
| <f:facet>	incomingRequests: userHeader	Kopf der Spalte für den Benutzername des Anzeigenstellers mit Filterfunktion.		name="header"	N / A
| | <comp: sortFilterColumn>	incomingRequests: request-user-nick			datamodel= "#{incomingRequestsBean .dataModel}" columnName="#{messages.user}" columnIdentifier="user"	N / A
| <h:outputText>		Benutzername des Anzeigenstellers.		value= "#{item.requestCreatorUsername}"	N / A

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	incomingRequests: ad	Spalte für den Titel der Anzeige.	scope="col"		N / A
| <f:facet>	incomingRequests: adHeader	Kopf der Spalte für den Titel der Anzeige mit Filterfunktion.		name="header"	N / A
| | <comp: sortFilterColumn>	incomingRequests: request-ad-title			datamodel= "#{incomingRequestsBean .dataModel}" columnName= "#{messages.advertisement}" columnIdentifier="ad"	N / A
| <h:outputText>		Titel der Anzeige.		value="#{item.advertisementTitle}"	N / A
<h:column>	incomingRequests: requestText	Spalte für den Anfragetext. Falls keiner vorhanden ist, ist diese leer.	scope="col"		N / A
| <f:facet>	incomingRequests: requestTextHeader	Kopf der Spalte für den Anfragentext.		name="header"	N / A
| | <comp: sortFilterColumn>	incomingRequests: request-text			datamodel= "#{incomingRequestsBean .dataModel}" columnName= "#{messages.requestText}" columnIdentifier="requestText"	N / A
| <h:outputText>		Anfang des Anfragetextes.	<f:stringLengthConverter maxLength="100"/>	value="#{item.request}"	N / A
<h:column>	incomingRequests: response	Spalte für die Antwort auf die Anfrage. Falls keine vorhanden ist, ist diese leer.	scope="col"		N / A
| <f:facet>	incomingRequests: responseHeader	Kopf der Spalte für die Antwort auf die Anfrage mit Filterfunktion.		name="header"	N / A
| | <comp: sortFilterColumn>	incomingRequests: request-response			datamodel= "#{incomingRequestsBean .dataModel}" columnName= "#{messages.response}" columnIdentifier="response"	N / A
| <h:outputText>		Anfang des Antworttextes.	<f:stringLengthConverter maxLength="100"/>	value="#{item.response}"	N / A

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	incomingRequests: answer	Spalte für die Antwort-Buttons.	scope="col"		N / A
| <f:facet>	incomingRequests: answerHeader	Kopf der Spalte für die Antwort-Buttons.		name="header"	N / A
| | <h:outputText>		Titel der Spalte für die Antwort-Buttons.		value="#{messages.answer}"	N / A
| <h:button>		Button um auf die Anzeige zu antworten.		outcome="/view/registered /writeResponse" includeViewParams="true" <f:param name="id" value= "#{item.advertisementId}" /> value="#{messages.answer}"	N / A

Erstellte Anzeigen Diese Seite ermöglicht dem Nutzer eigens erstellte Anzeigen einzusehen.

createdAdvertisements.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<comp: dynamicDataTable>	created-Advertisements: table	Tabelle mit Paginierung für die selbst erstellten Anzeigen.		dataModel= "#{createdAdvertisementsBean .dataModel}"	N / A
<h:Button>	advertisement-Details:newAd	Button, für das Erstellen einer neuen Anzeige.		outcome="/view/registered /editAdvertisement" value="#{messages.createAd}	N / A
<h:Link>	advertisement-Details: incomingRequests Link	Link zu den eingegangenen Anfragen.		outcome="/view/registered /incomingRequests" value= "#{messages.incomingRequests}	N / A

Beschreibungstabelle für die DynamicDataTable:

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	created-Advertisements: image	Spalte für die Titelbiler.	scope="col"		N / A
| <f:facet>	created-Advertisements: imageHeader	Kopf der Spalte der Titelbilder.		name="header"	N / A
| | <h:outputText>		Titel der Spalte der Titelbilder.		value="#{messages.image}"	N / A
| <o:graphicImage>		Bild der Anzeige.	dataURI="true"	value="#{item.image}"	N / A
<h:column>	created-Advertisements: adTitle	Spalte für die Anzeigentitel.	scope="col"		N / A
| <f:facet>	created-Advertisements: titleHeader	Kopf der Spalte der Anzeigentitel mit Filterfunktion.		name="header"	N / A
| | <comp: sortFilterColumn>	created-Advertisements: ad-title			datamodel= "#{createdAdvertisementsBean .dataModel}" columnName="#{messages.title}" columnIdentifier="title"	N / A
| <h:Link>		Titel der Anzeige. Durch drücken des Titels gelankt der Nutzer zur Bearbeitungsseite für die Anzeige	created-Advertisements: adLink	outcome="/view/registered/ editAdvertisement" includeViewParams="true" <f:param name="id" value= "#{item.id}" /> value="#{item.title}"	N / A
<h:column>	created-Advertisements: adCosts	Spalte für die benötigten Talentpunkte.	scope="col"		N / A
| <f:facet>	created-Advertisements: costsHeader	Kopf der Spalte die benötigten Talentpunkte mit Filterfunktion.		name="header"	N / A
| | <comp: sortFilterColumn>	created-Advertisements: ad-costs			datamodel= "#{createdAdvertisementsBean .dataModel}" columnName="#{messages.costs}" columnIdentifier= "cost_in_talentPoints"	N / A
| <h:outputText>		Kosten der Anzeige in Talentpunkten.		value="#{item.cost}"	N / A

Ausgehende Anfragen + Antworten Diese Seite ermöglicht dem Nutzer ausgehende Anfragen und die gegebenen Antworten einzusehen.

outgoingRequests.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<f:viewParam>		Id des Nutzers.		name="id" value="#{outgoingRequestsBean .user.id}"	
<f:viewAction>		Lädt die eingegangenen Anfragen des Nutzers.		action="#{outgoingRequestsBean .loadData()}"	
<comp: dynamicDataTable>	outgoingRequests: table	Tabelle mit Paginierung für die Anfragen.		dataModel= "#{outgoingRequestsBean .dataModel}"	N / A
<h:link>	outgoingRequests: claimedAdsLink	Link welcher zur Seite für die selbst beanspruchte Anzeigen weiterleitet.		outcome="/view/registered/ claimedAdvertisements" value="#{messages .claimedAdvertisementsLink}"	N / A
<h:link>	outgoingRequests: user- Administration- Link-	Link welcher zur Seite für die Benutzerverwaltung weiterleitet.	rendered= "#{userSession.isUserAdmin}"	outcome="/view/admin/ userAdministration" value="#{messages .userAdministrationLink}"	A

Beschreibungstabelle für die DynamicDataTable:

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	outgoingRequests: user	Spalte für den Benutzername des Anzeigenstellers.	scope="col"		N / A
| <f:facet>	outgoingRequests: userHeader	Kopf der Spalte für den Benutzername des Anzeigenstellers mit Filterfunktion.		name="header"	N / A
| | <comp: sortFilterColumn>	outgoingRequests: request-user-nick			datamodel= "#{outgoingRequestsBean .dataModel}" columnName="#{messages.user}" columnIdentifier="user"	N / A
| <h:outputText>		Benutzername des Anzeigenstellers.		value= "#{item.requestCreatorUsername}"	N / A

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	outgoingRequests: ad	Spalte für den Titel der Anzeige.	scope="col"		N / A
| <f:facet>	outgoingRequests: adHeader	Kopf der Spalte für den Titel der Anzeige mit Filterfunktion.		name="header"	N / A
| | <comp: sortFilterColumn>	outgoingRequests: request-ad-title			datamodel= "#{outgoingRequestsBean .dataModel}" columnName= "#{messages.advertisement}" columnIdentifier="ad"	N / A
| <h:outputText>		Titel der Anzeige.		value="#{item.advertisementTitle}"	N / A
<h:column>	outgoingRequests: requestText	Spalte für den Anfragetext. Falls keiner vorhanden ist, ist diese leer.	scope="col"		N / A
| <f:facet>	outgoingRequests: requestTextHeader	Kopf der Spalte für den Anfragentext.		name="header"	N / A
| | <comp: sortFilterColumn>	outgoingRequests: request-text			datamodel= "#{outgoingRequestsBean .dataModel}" columnName= "#{messages.requestText}" columnIdentifier="requestText"	N / A
| <h:outputText>		Anfang des Anfragetextes.	<f:stringLengthConverter maxLength="100"/>	value="#{item.request}"	N / A
<h:column>	outgoingRequests: response	Spalte für die Antwort auf die Anfrage. Falls keine vorhanden ist, ist diese leer.	scope="col"		N / A
| <f:facet>	outgoingRequests: responseHeader	Kopf der Spalte für die Antwort auf die Anfrage mit Filterfunktion.		name="header"	N / A
| | <comp: sortFilterColumn>	outgoingRequests: request-response			datamodel= "#{outgoingRequestsBean .dataModel}" columnName= "#{messages.response}" columnIdentifier="response"	N / A
| <h:outputText>		Anfang des Antworttextes.	<f:stringLengthConverter maxLength="100"/>	value="#{item.response}"	N / A

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	outgoingRequests: accepted	Spalte für das Anfrageergebnis.	scope="col"		N / A
| <f:facet>	outgoingRequests: acceptedHeader	Kopf der Spalte für das Anfrageergebnis.		name="header"	N / A
| | <comp: sortFilterColumn>	outgoingRequests: request-accepted			datamodel= “#{outgoingRequestsBean .dataModel}” columnName= “#{messages.accepted}” columnIdentifier=“accepted”	N / A
| <h:outputText>		Ergebnis der Anfrage		value=“#{messages[item.accepted]}”	N / A

Beanspruchte Anzeigen Diese Seite ermöglicht dem Nutzer eigens beanspruchte Anzeigen einzusehen.

claimedAdvertisements.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<comp: dynamicDataTable>	claimed-Advertisements: table	Tabelle mit Paginierung für die beanspruchten Anzeigen.		dataModel= “#{claimedAdvertisementsBean .dataModel}”	N / A
<h:Link>	claimed-Advertisements: outgoingRequests Link	Link zu den ausgehenden Anfragen.		outcome=“/view/registered /outgoingRequests” value= “#{messages.outgoingRequests}	N / A

Beschreibungstabelle für die DynamicDataTable:

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	claimed-Advertisements: image	Spalte für die Titelbiler.	scope="col"		N / A
| <f:facet>	claimed-Advertisements: imageHeader	Kopf der Spalte der Titelbilder.		name="header"	N / A
| | <h:outputText>		Titel der Spalte der Titelbilder.		value=“#{messages.image}”	N / A
| <o:graphicImage>		Bild der Anzeige.	dataURI=“true”	value=“#{item.image}”	N / A

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	claimed-Advertisements: adTitle	Spalte für die Anzeigentitel.	scope="col"		AN
| <f:facet>	claimed-Advertisements: titleHeader	Kopf der Spalte der Anzeigentitel mit Filterfunktion.		name="header"	AN
| | <comp: sortFilterColumn>	claimed-Advertisements: ad-title			datamodel= "#{claimedAdvertisementsBean .dataModel}" columnName="#{messages.title}" columnIdentifier="title"	AN
| <h:outputText>		Titel der Anzeige.	claimed-Advertisements: ad	value="#{item.title}"	N / A
<h:column>	claimed-Advertisements: adCosts	Spalte für die benötigten Talentpunkte.	scope="col"		N / A
| <f:facet>	claimed-Advertisements: costsHeader	Kopf der Spalte die benötigten Talentpunkte mit Filterfunktion.		name="header"	N / A
| | <comp: sortFilterColumn>	created-Advertisements: ad-costs			datamodel= "#{claimedAdvertisementsBean .dataModel}" columnName="#{messages.costs}" columnIdentifier= "cost_in_talentPoints"	N / A
| <h:outputText>		Kosten der Anzeige in Talentpunkten.		value="#{item.cost}"	N / A

Anfrage schreiben Diese Seite ermöglicht dem Nutzer eine Anfrage für eine Anzeige zu schreiben.

writeRequest.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<f:viewParam>		Id der Anzeige.		name="id" value="#{writeRequestBean .ad.id}"	
<f:viewAction>		Lädt die Anzeige.		action="#{writeRequestBean .loadData()}"	
<o:graphicImage>	writeRequest: image	Titelbild der Anzeige.	dataURI="true"	value="#{writeRequestBean .ad.image}"	N / A
<o:graphicImage>	writeRequest: image	Profilbild des Anzeigenerstellers.	dataURI="true"	value="#{writeRequestBean .adCreator.avatar}"	N / A
<h:outputText>	writeRequest: AdTitle	Titel der Anzeige.		value="#{writeRequestBean .ad.title}"	N / A
<h:outputText>	writeRequest: creatorNick	Benutzername des Anzeigenerstellers.		value="#{writeRequestBean .adCreator.nickname}"	N / A
<h:inputText>	writeRequest: request	Inputfeld für die Anfrage.		value="#{writeRequestBean .request.request}"	N / A
<h:commandButton>	writeRequest: submit	Schickt die Anfrage		action="#{writeRequestBean .submit()}" value="#{messages.submit}"	N / A
<h:link>	writeRequest: adDetailsLink	Link welcher zur Detailseite der Anzeige zurückleitet.		outcome="/view/anonymous/ advertisementDetails" includeViewParams="true" <f:param name="id" value= "#{writeRequestBean.ad.id}" /> value="#{messages .advertisementDetailsLink}"	N / A

6.4.3 Administrator

Die folgenden Seiten können nur von einem Administrator aufgerufen werden.

Nutzerverwaltung Diese Seite ermöglicht dem Admin alle Nutzer einzusehen und zu bearbeiten.

userAdministration.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<comp: dynamicDataTable>	user-Administration: table	Tabelle mit Paginierung für die registrierten Benutzer.		dataModel= “#{userAdministrationBean .dataModel}”	A
<h:Button>	user-Administration: newUser	Button für das Erstellen eines neuen Benutzers.		outcome=“/view/admin /editUser” value=“#{messages.createUser}	A

Beschreibungstabelle für die DynamicDataTable:

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	user-Administration: avatar	Spalte für die Avatare der Benutzer.	scope="col"		A
| <f:facet>	user-Administration: avatarHeader	Kopf der Spalte für die Avatare der Benutzer.		name="header"	N / A
| | <h:outputText>		Titel der Spalte für die Avatare der Benutzer.		value=“#{messages.avatar}”	A
| <o:graphicImage>		Avatar des Benutzers	dataURI=“true”	value=“#{item.avatar}”	A

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	user-Administration: nick	Spalte für die Benutzernamen.	scope="col"		A
| <f:facet>	user-Administration: nickHeader	Kopf der Spalte für die Benutzernamen mit Filterfunktion.		name="header"	A
| | <comp: sortFilterColumn>	user-Administration: user-nick			datamodel= “#{userAdministrationBean .dataModel}” columnName=“#{messages.nick}” columnIdentifier=“nick”	A
| <h:Link>		Benutzername des Nutzers. Durch drücken des Nutzernames gelankt der Admin zur Bearbeitungsseite für den Nutzer		outcome=“/view/admin/ editUser” includeViewParams=“true” <f:param name=“id” value= “#{item.id}” /> value=“#{item.nick}”	A
<h:column>	user-Administration: talentPoints	Spalte für die Talentpunkte des Benutzers.	scope="col"		A
| <f:facet>	user-Administration: talentPointsHeader	Kopf der Spalte für die Talentpunkte des Benutzers mit Filterfunktion.		name="header"	A
| | <comp: sortFilterColumn>	user-Administration: talent-points			datamodel= “#{userAdministrationBean .dataModel}” columnName= “#{messages.talentPoints}” columnIdentifier=“talentPoints”	A
| <h:outputText>		Talentpunkte des Nutzers.		value=“#{item.talentPoints}”	A

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:column>	user-Administration: verified	Spalte mit dem Verifikationsstatus durch den Admin. Wird nur angezeigt wenn die Verifikation durch den Admin im System aktiviert ist.	scope="col" rendered="#{systemData .adminConfirmationRequired-ForRegistration}"		A
| <f:facet>	user-Administration: verifiedHeader	Kopf der Spalte für den Verifikationsstatus durch den Admin mit Filterfunktion.		name="header"	A
| | <comp: sortFilterColumn>	user-Administration: admin-verified			datamodel= "#{userAdministrationBean .dataModel}" columnName= "#{messages.adminVerified}" columnIdentifier="verified"	A
| <h:outputText>		Verifikationsstatus des Benutzers durch den Admin.		value="#{item.adminVerified}"	A

Nutzer erstellen/bearbeiten/löschen In diesem Form befinden sich alle notwendigen Bedienelemente, um einen Benutzer zu erstellen, zu bearbeiten oder zu löschen. Sollte ein Benutzer erstellt werden, sind alle Felder zu beginn leer. Sollte ein Nutzer bearbeitet werden, sind alle Felder zu beginn mit den aktuellen Werten vorbefüllt.

editUser.xhtml

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<f:viewParam>		Id des Benutzers.		name="id" value="#{editUser .user.id}"	
<f:viewAction>		Lädt den Nutzer.		action="#{editUser .loadData()}"	
<o:graphicImage>	editUser:avatar	Aktuelles Profilbild/Preview für das neue Profilbild.	dataURI="true"	value= "#{editUserBean.user.avatar}"	A
<o:inputFile>	editUser: newAvatar	File input für neues Profilbild. Bei Auswahl einer passenden Datei wird das aktuelle Profilbild mittels eines ajax calls mit einer Vorschau des neuen ersetzt.	validator="avatarValidator" accept="image/png,image/jpeg"	value= "#{editUserBean.user.avatar}" <f:ajax render="@form"/>	A
<h:commandButton>	editUser: deleteAvatar	Löscht das aktuelle Profilbild.		action="#{editUserBean .deleteAvatar()}" value="#{messages.deleteAvatar}"	A
<h:inputText>	editUser:nick	Inputfeld für den Nutzernamen.	required="true" maxlength="255" validator= "nicknameUniqueValidator"	value="#{editUserBean.user .nickname}"	A
<h:inputText>	editUser:password	Inputfeld für das Passwort.	required="true" maxlength="255"	value="#{editUserBean .passwordInput.password}"	A
<h:inputText>	editUser:email	Inputfeld für die E-Mail.	required="true" maxlength="70" validator= "emailFormatValidator" validator= "emailUniqueValidator"	value="#{editUserBean.user .email}"	A
<h:inputText>	editUser:firstname	Inputfeld für den Vornamen.	required="true" maxlength="50"	value="#{editUserBean.user .firstname}"	A
<h:inputText>	editUser:lastname	Inputfeld für den Nachnamen.	required="true" maxlength="50"	value="#{editUserBean.user .lastname}"	A
<h:inputText>	editUser: talentPoints	Inputfeld für die Talentpunkte des Nutzers.	required="true" onkeypress="if(event.which < 48 || event.which > 57) return false;" <f:convertNumber integerOnly="true" type="number" />	value="#{editUserBean .user.talentPoints}"	A
<h:inputText>	editUser:street	Inputfeld für die Straße.	required="true" maxlength="150"	value="#{editUserBean.user .street}"	A
<h:inputText>	editUser: houseNumber	Inputfeld für die Hausnummer.	required="true" maxlength="10"	value="#{editUserBean.user .houseNumber}"	A

Elementtyp	Identifier	Beschreibung	Constraints	Binding	Sichtbarkeit
<h:inputText>	editUser:city	Inputfeld für die Stadt.	required="true" maxlength="150"	value="#{editUserBean.user .city}"	A
<h:inputText>	editUser: postalCode	Inputfeld für die Postleitzahl.	required="true" maxlength="25"	value="#{editUserBean.user .postalCode}"	A
<h:inputText>	editUser: addressAddition	Inputfeld für den Addresszusatz.	maxlength="40"	value="#{editUserBean.user .addressAddition}"	A
<h:inputText>	editUser:phone	Inputfeld für die Telefonnummer.	maxlength="20"	value="#{editUserBean.user .phone}"	A
<h:selectBoolean Checkbox>	editUser: emailVerified	Checkbox für die Emailverifikation.		value="#{editUserBean .user .emailVerified}"	A
<h:selectBoolean Checkbox>	editUser: adminVerified	Checkbox für die Adminverifikation.	rendered="#{systemData .adminConfirmationRequired-ForRegistration}"	value="#{editUserBean .user .adminVerified}"	A
<h:selectBoolean Checkbox>	editUser: setAdmin	Checkbox für den Adminstatus.		value="#{editUserBean .user .admin}"	A
<h:commandButton>	editUser:submit	Speichert die aktuellen Daten.		action="#{editUserBean.submit()}" value="#{messages.submit}"	A
<h:commandButton>	editUser:delete	Löscht das Profil.		action= "#{editUserBean.deleteUser()}" value="#{messages.delete}"	A
<h:link>	editUser:user-Administration-Link	Link welcher zur Seite für die Nutzerverwaltung weiterleitet.		outcome="/view/admin/ userAdministration" value="#{messages .userAdministrationLink}"	A

7 Datenbank

Leon Föckersperger

7.1 Konvertierung des ER-Diagramms

Dieser Abschnitt beschreibt detailliert die Umwandlung des ER-Diagramms in die *Data Definition Language* (DDL) und erläutert jeden Schritt des Prozesses.

7.2 SQL-Standardkonformität und PostgreSQL-Erweiterungen

SQL ist standardisiert, doch PostgreSQL erweitert diesen durch spezifische Funktionen, die unser Datenbankschema nutzt:

- **ENUM-Typen:** Definiert durch `CREATE TYPE ... AS ENUM` zur Beschränkung von Werten, z.B. `USER` und `ADMIN`.
- **BYTEA:** Speichert binäre Daten wie Bilder direkt in der Datenbank, wodurch externe Speichersysteme unnötig werden.
- **SERIAL:** Automatisch inkrementierende Ganzzahlen für eindeutige Identifikatoren, eingesetzt als Primärschlüssel.
- **TIMESTAMP WITHOUT TIME ZONE:** Vereinfacht die Zeitstempelverwaltung ohne Berücksichtigung der Zeitzone.
- **LOWER-Funktion in Indexen:** Ermöglicht case-insensitive Indizes zur Sicherstellung der Einzigartigkeit von E-Mail-Adressen und Nicknamen.

Diese spezifischen PostgreSQL-Erweiterungen verbessern die Effizienz und Sicherheit unseres Datenbankschemas deutlich über den allgemeinen SQL-Standard hinaus.

7.2.1 Normalisierung

Das Datenbankschema entspricht der *Boyce-Codd-Normalform* (BCNF). Surrogatschlüssel in den Entitäten eliminieren die meisten funktionalen Abhängigkeiten, da sie als Superschlüssel fungieren. Alle `UNIQUE`-Beschränkungen stellen ebenfalls funktionale Abhängigkeiten dar und wirken als Superschlüssel.

7.2.2 Namenskonventionen

Tabellennamen sind im Singular formuliert, z.B. ‘user‘ statt ‘users‘, und Bezeichner werden in *snake_case* geschrieben, um sie visuell von Java’s *camelCase* zu unterscheiden. Enumerationen werden in *UPPER_CASE* notiert.

7.2.3 Beziehungshandhabung

1:n Beziehungen Bei 1:n Beziehungen ist die Implementierung direkt und effizient durch den Einsatz eines Fremdschlüssels auf der ’n’-Seite möglich. Dieser Fremdschlüssel verweist auf den Primärschlüssel der ’1’-Seite. So kann jede Entität auf der ’n’-Seite eindeutig einer Entität auf der ’1’-Seite zugeordnet werden.

n:m Beziehungen Für n:m Beziehungen ist die Einrichtung einer Verbindungstabelle notwendig, die die Schlüsselpaare aller beteiligten Entitäten speichert. Diese Tabelle enthält Fremdschlüssel, die auf die Primärschlüssel der beiden ursprünglich verbundenen Tabellen verweisen.

Bezüglich der Abhängigkeiten, wie zum Beispiel bei einer Anfrage-Antwort-Beziehung, die zu einer Anzeige gehört, wird das `ON DELETE CASCADE` Prinzip angewendet. Wenn die Hauptentität (zum Beispiel `Ad`) gelöscht wird, führt dies dazu, dass alle zugehörigen abhängigen Daten (wie `request_response`) automatisch gelöscht werden. Für Fremdschlüssel, die nicht direkt von anderen Entitäten abhängig sind, wie beispielsweise zwischen Anfragen-Erstellern und den Anfragen/Antworten, wird die Aktion `ON DELETE SET NULL` angewendet. Dies bedeutet, dass im Falle der Löschung der referenzierten Entität die entsprechenden Fremdschlüssel in den abhängigen Datensätzen auf `NULL` gesetzt werden. Somit bleibt die Integrität der Datenbank erhalten, indem die abhängigen Datensätze ohne Verbindung zur ursprünglichen Referenz weiterhin existieren.

7.2.4 Vererbung

In ER-Diagrammen spiegeln *IS-A*-Beziehungen Vererbungsbeziehungen zwischen Entitäten wider. Zur Umsetzung dieser Vererbung in der Datenbank verwenden wir den PostgreSQL-spezifischen `ENUM`-Typ. Beispielsweise definieren wir einen `ENUM`-Typ `user_role`, der die Werte `USER` und `ADMIN` umfasst, um unterschiedliche Benutzerrollen klar zu differenzieren.

7.3 Schema-Erstellung und Verwaltung

Bevor wir die einzelnen Entitäten und deren Beziehungen definieren, müssen wir das Schema in PostgreSQL erstellen, das als Container für unsere Tabellen und Typen dient. Ein Schema in PostgreSQL ist eine logische Sammlung von Datenbankobjekten, die organisiert verwaltet werden können. Hier ist der Befehl zur Erstellung des Schemas:

```
CREATE SCHEMA IF NOT EXISTS talent_tauscher;
```

Listing 6: Erstellen des Schemas

Dieser Befehl überprüft, ob das Schema `talent_tauscher` bereits existiert, und erstellt es nur, wenn es noch nicht vorhanden ist. Dies vermeidet Fehler beim wiederholten Ausführen des Skripts und sorgt dafür, dass alle folgenden Datenbankobjekte (wie Tabellen und Typen) korrekt in diesem Schema organisiert werden.

7.4 Definition der Entitäten

Hier werden die einzelnen Entitäten in der Data Definition Language beschrieben. Durch die Verwendung von `SERIAL` werden die Surrogatschlüssel der Relationen automatisch von PostgreSQL erstellt und die Eigenschaften der Primärschlüssel sichergestellt.

7.4.1 Benutzerentität

```
CREATE TABLE IF NOT EXISTS talent_tauscher.user
(
    id                          SERIAL PRIMARY KEY,
    first_name                  VARCHAR(50) NOT NULL,
    last_name                   VARCHAR(50) NOT NULL,
    nickname                    VARCHAR(255) UNIQUE,
    user_role                   talent_tauscher.user_role NOT NULL,
    avatar                      BYTEA,
    talent_points               INT DEFAULT 50,
    email_address               VARCHAR(70) UNIQUE NOT NULL,
    -- <salt> <hash> : 16 Byte Salt + 16 Byte Hash Base64 encoded
    auth_token                  VARCHAR(512),
    hash_method                 VARCHAR(20),
    secret_for_email_verification VARCHAR(512),
    secret_for_password_reset   VARCHAR(512),
    has_admin_verified          BOOLEAN DEFAULT FALSE,
    is_email_verified           BOOLEAN DEFAULT FALSE,
    phone_number                VARCHAR(20),
    created_at                  TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW(),
    updated_at                  TIMESTAMP WITHOUT TIME ZONE,
    country                     VARCHAR(40) NOT NULL,
    city                        VARCHAR(150) NOT NULL,
    postal_code                 VARCHAR(25) NOT NULL,
    street                      VARCHAR(150),
    address_addition            VARCHAR(40),
    house_number                VARCHAR(10),
    CONSTRAINT name_check CHECK (length(first_name) > 0 AND length(last_name) > 0)
);
```

Listing 7: DDL für Benutzerentität

7.4.2 Systemeinstellungen

```
CREATE TABLE IF NOT EXISTS talent_tauscher.system_settings
(
    id                                      SERIAL PRIMARY KEY,
    css_name                                VARCHAR(255),
    logo                                    BYTEA,
    data_protection                         TEXT,
    max_pic_size                            INT,
    contact_info                            TEXT
    is_admin_confirmation_needed_registration BOOLEAN,
    imprint                                 TEXT,
    sum_paginated_items                     INT
);
```

Listing 8: DDL für Systemeinstellungen

7.4.3 Anzeigenentität

```
CREATE TABLE IF NOT EXISTS talent_tauscher.ad
(
    id                  SERIAL PRIMARY KEY,
    creator_id          INT NOT NULL REFERENCES talent_tauscher.user (id) ON DELETE CASCADE,
    title               VARCHAR(255),
    image               BYTEA,
    free_text           VARCHAR(2047),
```

```
    cost_in_talent_points INT NOT NULL CHECK (cost_in_talent_points >= 0),
    date_of_publication   TIMESTAMP WITHOUT TIME ZONE NOT NULL,
    date_of_completion    TIMESTAMP WITHOUT TIME ZONE,
    is_active             BOOLEAN DEFAULT TRUE,
    country               VARCHAR(40) NOT NULL,
    city                  VARCHAR(150) NOT NULL,
    postal_code           VARCHAR(25) NOT NULL,
    street                VARCHAR(150),
    address_addition      VARCHAR(40),
    house_number          VARCHAR(10),
    street_shown          BOOLEAN,
    name_shown            BOOLEAN,
    phone_number_shown    BOOLEAN
);
```

Listing 9: DDL für Anzeigenentität

7.4.4 Anfrage/Antwort Entität

```
CREATE TABLE IF NOT EXISTS talent_tauscher.request_response
(
    id                 SERIAL PRIMARY KEY,
    ad_id              INT NOT NULL REFERENCES talent_tauscher.ad (id) ON DELETE CASCADE,
    request_creator_id INT REFERENCES talent_tauscher.user (id) ON DELETE SET NULL,
    timestamp_request  TIMESTAMP WITHOUT TIME ZONE DEFAULT NOW(),
    timestamp_response TIMESTAMP WITHOUT TIME ZONE,
    free_text_request  TEXT,
    free_text_response TEXT,
    result             BOOLEAN
);
```

Listing 10: DDL der Anfrage/Antwort Entität

7.5 Indizierung für Einzigartigkeit

Zur Sicherstellung der Einzigartigkeit von E-Mail-Adressen und Nicknamen, unabhängig von der Groß- oder Kleinschreibung, sind in PostgreSQL spezielle Indizes erforderlich, da das UNIQUE Schlüsselwort selbst case-sensitive ist.

```
CREATE UNIQUE INDEX email_unique_case_insensitive
ON talent_tauscher.user (LOWER(email_address));

CREATE UNIQUE INDEX nickname_unique_case_insensitive
ON talent_tauscher.user (LOWER(nickname));
```

Listing 11: DDL für eindeutige Indizes

7.6 Enum-Typen

In PostgreSQL ermöglichen Enum-Typen die Definition von Spalten, die ausschließlich einen von vordefinierten Werten annehmen. Diese Funktion stellt sicher, dass Daten immer gültig und konsistent sind, indem sie nur spezifische Werte für ein Attribut zulassen. Enums bieten auch verbesserte Leistung und Klarheit gegenüber äquivalenten Textspalten, da sie die Verarbeitung und Speicherung von Daten optimieren.

Hier ist ein Beispiel für einen Enum-Typ, den wir in unserer Datenbank nutzen:

```
CREATE TYPE talent_tauscher.user_role AS ENUM ('USER', 'ADMIN');
```

Listing 12: DDL für Nutzerrollen

Dieser Typ `user_role` hilft dabei, die Benutzerrollen klar zu definieren und sicherzustellen, dass nur gültige und erwartete Werte in die Datenbank eingefügt werden.

7.7 Beziehungen

Wir benötigen keine Tabellen für Beziehungen , da wir für die Darstellung der ‘benefits‘ Beziehung, die eine bestätigte Anfrage reflektiert, stattdessen diese Beziehung über Fremdschlüsselattribute und das ‘result‘ Attribut in der Anfrage/Antwort Entität modellieren.

8 Datenfluss

David Sturm

8.1 Erfolgreiches Bearbeiten einer Anzeige

Hier folgend wird beispielhaft für den Datenfluss im Programm ein Sequenzdiagramm für das Bearbeiten einer Anzeige. Es wird angenommen, dass ein ein angemeldeter Nutzer eine bereits zuvor von ihm erstellte Anzeige aufruft um diese zu bearbeiten.

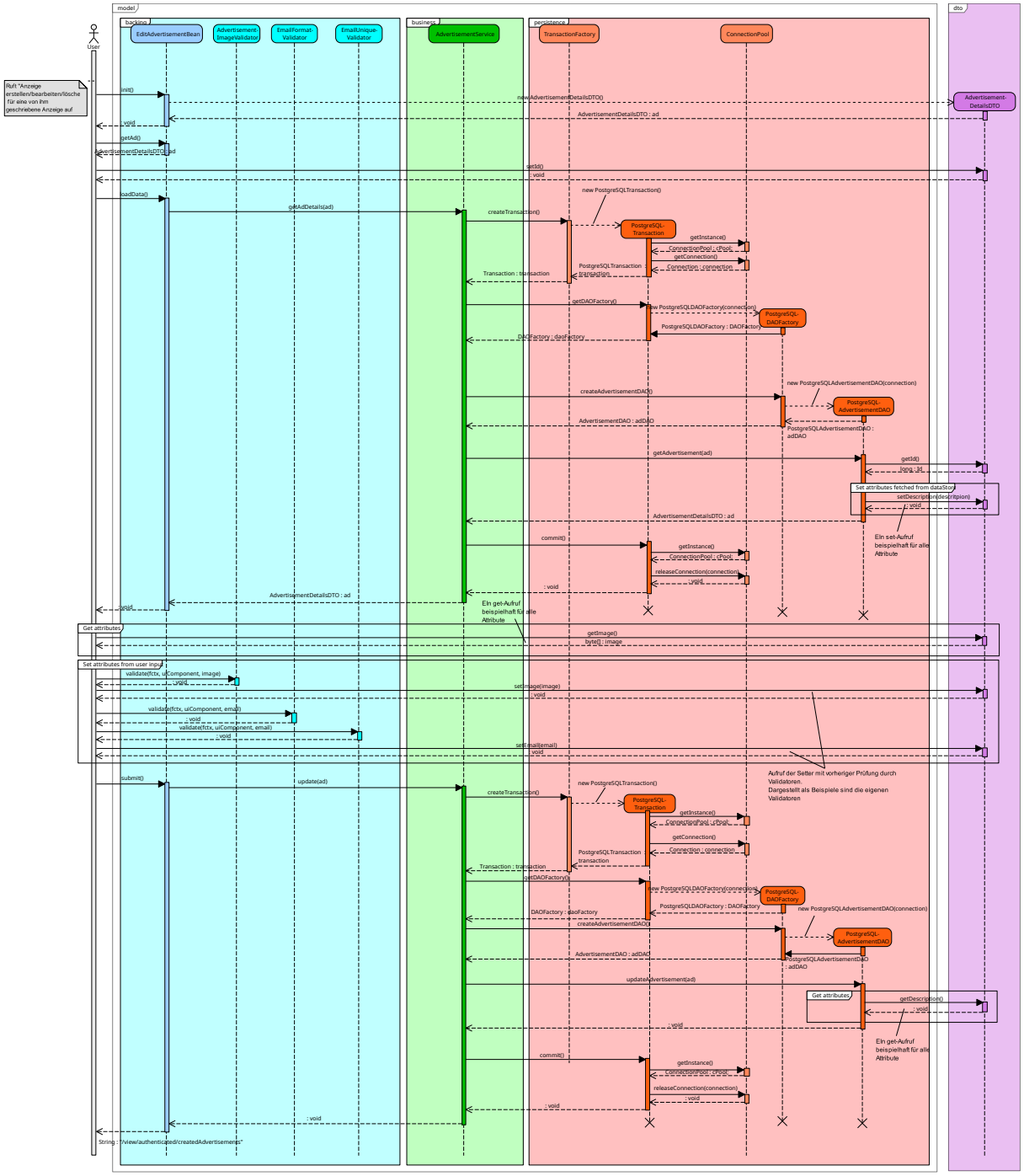

Abbildung 4: Sequenzdiagramm für das erfolgreiche Bearbeiten einer Anzeige

8.2 Fehlgeschlagenes Bearbeiten einer Anzeige

Hier folgend wird beispielhaft die Fehlerbehandlung bei einem auftretenden fatalen Fehler bei der Verbindung mit der Datenbank unter dem selber Szenerio wie zuvor.
Bei dem Paket ”java.util.logging” handelt es sich um ein Java eigenes Paket. Dieses wird zur verständlicheren Darstellung der Fehlerbehandlung mit aufgeführt, da das Programm auf die von Java bereitgestellten Logger Klasse zurückgreift.

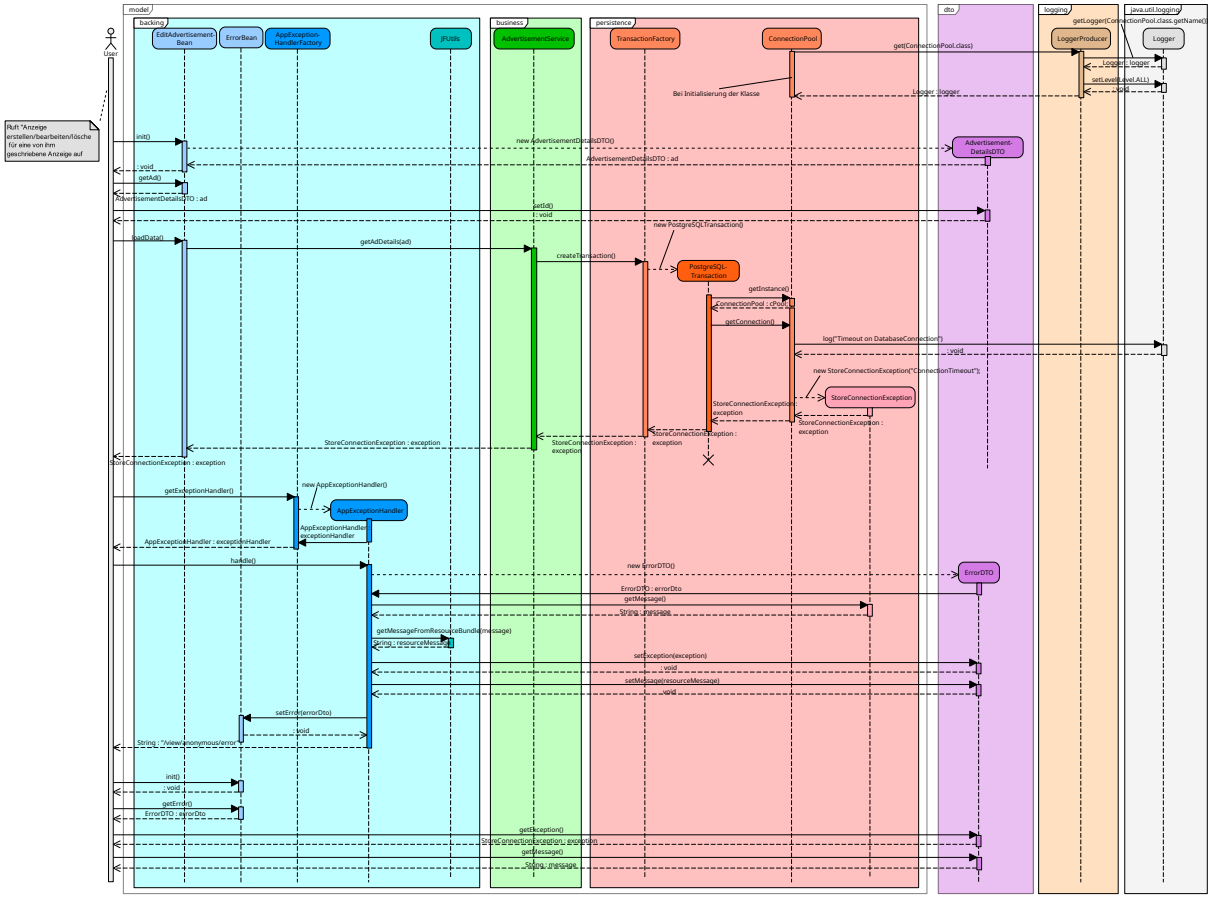

Abbildung 5: Sequenzdiagramm für das Auftreten eines fatalen Fehlers

9 Sicherheit

Leon Föckersperger

In diesem Abschnitt werden Strategien zur Verhinderung gängiger Sicherheitsangriffe und zur Minimierung von Sicherheitsrisiken vorgestellt. Die vorgeschlagenen Maßnahmen basieren auf den OWASP Top Ten Sicherheitsrisiken.

9.1 Zugriffskontrolle

Zur Abwehr von Bedrohungen durch fehlerhafte Zugriffskontrolle implementiert unsere Webanwendung eine strenge URL-basierte Zugriffssteuerung. Die Klasse `TrespassListener` validiert die Berechtigungen des Benutzers im Verhältnis zu der angeforderten URL. Die Verzeichnisstruktur unter `webapp/view` ist in `admin`, `registered`, und `anonymous` unterteilt, was eine rollenbasierte Zugriffssteuerung ermöglicht. Falls die Rolle in der `UserSession` nicht mit dem erforderlichen Zugriffslevel des angeforderten Verzeichnisses übereinstimmt, wird der Zugriff verwehrt und der Benutzer auf eine 404-Fehlerseite umgeleitet. Dies hilft nicht nur, unberechtigte Zugriffe zu verhindern, sondern verschleiert auch die Existenz gültiger Seiten.

Das `rendered`-Attribut in Facelets wird genutzt, um UI-Komponenten rollenspezifisch anzuzeigen. Die Beziehung des Benutzers zur angefragten Ressource wird zusätzlich durch Backing Beans in ihrer `init()`-Methode überprüft, welche mit `@PostConstruct` annotiert ist. Bei festgestellten Unstimmigkeiten erfolgt eine Weiterleitung auf eine Fehlerseite, wodurch das Risiko einer unsicheren direkten Objektreferenzierung minimiert wird.

Die Verarbeitung aller Anfragen an Facelet-Dateien wird durch folgendes Servlet-Mapping in der `web.xml` sichergestellt:

```
<servlet>
    <servlet-name>facesServlet</servlet-name>
    <servlet-class>jakarta.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>facesServlet</servlet-name>
    <url-pattern>/view/*</url-pattern>
</servlet-mapping>
```

Listing 13: Servlet-Mapping

Diese Konfiguration ermöglicht es dem `TrespassListener`, entweder das korrekte Rendering der Facelet-Datei als HTML oder die Umleitung des Nutzers auf die 404-Fehlerseite zu steuern.

Weiterhin wird durch die nachstehende Sicherheitsrichtlinie in der `web.xml` das direkte Herunterladen von auf dem Server gespeicherten Dateien eingeschränkt:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>downloadProtectedArea</web-resource-name>
        <url-pattern>/*/</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Listing 14: Security constraint

Diese Maßnahmen dienen dazu, die Sicherheit der Anwendung durch effektive Zugriffskontrollen und Datenintegrität zu gewährleisten.

9.2 Kryptographische Schwachstellen

Die Sicherung von Passwörtern erfolgt durch den Einsatz der kryptografischen Hashfunktion SHA-512, kombiniert mit einem individuell für jedes Passwort generierten zufälligen Salt. Dieses Verfahren verbessert die Widerstandsfähigkeit gegenüber Rainbow-Table-Angriffen. Sowohl der Hash als auch der Salt werden gemeinsam in der Datenbank gespeichert. Die Passwörter werden unmittelbar nach ihrer Eingabe gehasht, um die Dauer ihrer Existenz in Klartextform auf dem Server zu minimieren und potenzielle Risiken zu reduzieren. Die Implementierung erfolgt durch die Klasse `PasswordAuthentication` im Paket `business.util`, welche die `java.security`-Bibliothek nutzt, um sowohl das Hashen als auch das Generieren von Salts zuverlässig durchzuführen.

Zusätzlich sind Sicherheitsrichtlinien für die Passwortkomplexität implementiert, um das Erraten von Passwörtern durch Brute-Force-Attacken zu erschweren. Passwörter müssen eine Länge von 8 bis 100 Zeichen haben und eine Kombination aus Groß- und Kleinbuchstaben, Zahlen sowie Sonderzeichen enthalten. Diese Anforderungen werden von der Klasse `PasswordValidator` überprüft, die sicherstellt, dass alle Passwörter den festgelegten Sicherheitsstandards entsprechen.

9.3 Injection

9.3.1 Cross Site Scripting (XSS)

Die Anwendung verwendet ausschließlich JF-Komponenten zur HTML-Ausgabe, was automatisch eine Reinigung aller Nutzereingaben umfasst. Kritische Zeichen wie '<' und '>' werden beispielsweise zu '<' und '>' umgewandelt, was die Erstellung von HTML-Tags durch Nutzereingaben verhindert. Dies eliminiert die Möglichkeit von XSS bzw. HTML-Injection, da selbst eingefügter JavaScript-Code inaktiv bleibt, solange er sich innerhalb eines HTML-Tags (wie z.B. `<script>`) befinden muss. Ausnahmen werden für Inhalte gemacht, die von vertrauenswürdigen Administratoren stammen, wie das Impressum und der Datenschutz, wo das Attribut `escape="false"` eine formatierte Ausgabe ermöglicht.

9.3.2 SQL-Injection

SQL-Injection-Angriffe treten auf, wenn unautorisierter SQL-Code durch Angreifer in Benutzereingaben integriert und vom System fälschlicherweise als Teil einer legitimen SQL-Abfrage behandelt und ausgeführt wird. Zur Minimierung dieses Risikos erfolgt die Übermittlung aller Eingabeparameter strikt über `PreparedStatements`. Diese Methode trennt den SQL-Befehl von den Nutzereingaben, indem sie den SQL-Code vordefiniert und Daten erst zur Ausführungszeit einbindet. Dies verhindert die Ausführung von eingeschleustem Schadcode, da Eingabedaten nur als Parameter behandelt und nicht als Teil des SQL-Codes ausgeführt werden. Die Integration von Benutzertext in SQL-Anfragen bleibt auf solche Daten beschränkt, die nicht direkt von Benutzern stammen. Bestimmte vertrauliche Informationen, wie etwa der Datenbankhost aus Konfigurationsdateien, werden direkt in SQL-Statements übernommen. Da jedoch nur Administratoren Zugriff auf diese Konfigurationen haben und ihnen vertraut wird, wird das damit verbundene Risiko als gering eingestuft.

9.3.3 Session Hijacking & Fixation

In der `web.xml`-Datei kann ein Timeout für Sessions konfiguriert werden. Dieses Timeout hilft, sowohl Session Hijacking als auch unerwünschten physischen Zugriff auf den Browser des Benutzers zu verhindern. Jakarta Faces (JF) setzt standardmäßig das `HttpOnly`-Attribut für das Session-Cookie. Dadurch ist kein Zugriff über JavaScript auf das Cookie möglich, was das Risiko eines Hijackings deutlich reduziert.

Um die Sicherheit weiter zu erhöhen, wird nach jeder erfolgreichen Anmeldung die Session-ID verworfen und neu generiert. Dieser Prozess erfolgt durch den Aufruf der Methode `changeSessionId()`, wie im folgenden Codeausschnitt gezeigt:

```
1    FacesContext.getCurrentInstance().getExternalContext().getRequest().changeSessionId()
```

Listing 15: Verhinderung von Session Hijacking

Durch das Neugenerieren der Session-ID nach jeder Authentifizierung werden bestehende Sessions ungültig gemacht, was *Session-Fixation*-Attacken effektiv verhindert.

9.3.4 Vergabe von Administratorrechten

Das System ermöglicht es bestehenden Administratoren, Nutzerkonten Administratorrechte zu verleihen oder diese zu entziehen. Aufgrund der hohen Sicherheitsrelevanz solcher Berechtigungsänderungen ist ein zusätzlicher Authentifizierungsschritt erforderlich. Administratoren müssen ihre Identität durch die erneute Eingabe ihres Passworts bestätigen, bevor sie Änderungen an den Rechten eines Nutzers vornehmen können. Dies stellt sicher, dass solche kritischen Aktionen nur von verifizierten und autorisierten Personen durchgeführt werden können.

9.4 Software and Data Integrity Failures

Die Sicherstellung der Integrität von Software und Daten ist für die Zuverlässigkeit und Sicherheit unserer Systeme entscheidend. Wie im Sicherheitsmechanismus des Listings 14 dargestellt, erfolgt die Kommunikation zwischen Client und Server ausschließlich über gesicherte HTTPS-Verbindungen. Dies schützt die übertragenen Daten vor Abhör- und Manipulationsversuchen. Zusätzlich verwenden wir das Versionskontrollsystem GitLab, das auf einem Server unserer Universität gehostet wird. Durch den Einsatz von GitLab reduzieren wir das Risiko von Datenverlusten und unautorisierten Änderungen am Quellcode erheblich. Der Einsatz solcher Werkzeuge ist essenziell, um die Konsistenz und Unversehrtheit des Codes über den gesamten Entwicklungsprozess hinweg zu gewährleisten.

9.5 Security Logging and Monitoring Failures

Effektive Sicherheitsüberwachung umfasst weit mehr als nur die Implementierung von Sicherheitsmechanismen. Sie beinhaltet die kontinuierliche Überwachung der Systemaktivitäten und das detaillierte Protokollieren von sicherheitsrelevanten Ereignissen. Unsere Strategie schließt das Erfassen von Logdaten ein, und wir empfehlen den Administratoren, diese Daten regelmäßig zu analysieren. Zudem ist das Implementieren von Alarmierungssystemen für potenzielle Sicherheitsbedrohungen entscheidend. Die Protokolldaten, die auf den Tomcat-Servern gespeichert werden, spielen eine wesentliche Rolle beim frühzeitigen Erkennen und Reagieren auf ungewöhnliche oder potenziell schädliche Aktivitäten im System. Es ist essenziell, dass diese Daten systematisch analysiert werden, um Sicherheitsvorfälle präventiv zu erkennen und deren Auswirkungen effektiv zu minimieren.

9.6 Information Leakage

Die Offenlegung von internen Systeminformationen kann potenzielle Angriffspunkte für Angreifer bieten. Zu diesen Informationen gehören technische Details der Systemumgebung, die verwendete Software und interne Fehlermeldungen sowie spezifische Implementierungsdetails. Um derartige Informationslecks zu vermeiden, haben wir gezielte Maßnahmen implementiert.

Im Produktionsmodus werden Fehlermeldungen so konfiguriert, dass sie keine Stacktraces preisgeben. Anstelle detaillierter technischer Daten werden benutzerfreundliche Nachrichten angezeigt, die lediglich für Endbenutzer relevante Informationen beinhalten. Zudem haben wir standardmäßige Fehlerseiten, wie die für HTTP 404- und 500-Fehler, durch spezifisch gestaltete Seiten ersetzt, um die Verwendung von Jakarta Faces (JF/Mojarra) zu verschleiern. Dies wird in der `web.xml`-Datei wie folgt konfiguriert:

```
1 <error-page>
2   <error-code>404</error-code>
3   <location>/WEB-INF/errorpages/error404.xhtml</location>
4 </error-page>
```

Listing 16: Angepasste Fehlerseite

Weiterhin tragen wir zur Verschleierung bei, indem wir .xhtml-Dateiendungen für unsere Facelets nutzen und den Standardbezeichner `JSESSIONID` für Session-Cookies und URL-Rewriting durch einen eigenen Bezeichner ersetzen. Dies ist in der `web.xml`-Datei wie folgt konfiguriert:

```
1 <session-config>
2   <cookie-config>
3     <name>sessionid</name>
4   </cookie-config>
```

```
5 </session-config>
```

Listing 17: Eigene Session-ID Konfiguration

Diese Maßnahmen dienen dazu, die tatsächliche Nutzung von JF/Mojarra zu verbergen und das Risiko von Informationslecks zu minimieren.