Entwicklung einer Benutzerverwaltung inklusive Rollen- und Berechtigungskonzept

Abschlussprüfung
zum Fachinformatiker /zur Fachinformatikerin
Sommer 2016

Fachrichtung: Anwendungsentwicklung

eingereicht von: Thomas Köhler

Ausbildungsbetrieb: IT-Systemhaus der Bundesagentur für Arbeit
Regensburger Str. 104, 90478 Nürnberg

Abgabedatum: 15. Mai 2016

Prüflingsnummer: 158 23013

IT-Systemhaus

Inhaltsverzeichnis

1 Einleitung 1

1.1 Anmerkung zur Dokumentation 1

1.2 Der Ausbildungsbetrieb 1

1.3 Das Projektumfeld 1

1.4 Das Projektziel 2

2 Planung 2

2.1 Analyse 2

2.1.1 Projektdefinition 2

2.1.2 Anforderungsanalyse 3

2.2 Design 4

2.2.1 Verwendete Technologien 4

2.2.2 Planung und Entwurf der Oberflächen 5

2.2.3 Planung der Programmlogik 5

2.2.4 Planung des Rollen – und Berechtigungskonzepts 7

2.2.5 Planung und Entwurf der Datenbank 8

3 Realisierung 9

3.1 Implementierung der Datenbank 9

3.2 Implementierung der Oberflächen 9

3.3 Implementierung der Programmlogik 10

4 Qualitätssicherung 14

4.1 Code-Optimierung 14

4.2 Programmtest und Fehlerbehebung 15

5 Abschluss 15

5.1 Zeitaufwand 15

5.2 Fazit und Ausblick 16

6 Glossar 17

7 Quellenangaben 18

8 Abbildungsverzeichnis 18

1 Einleitung

1.1 Anmerkung zur Dokumentation

In der Dokumentation verwendete Fachbegriffe und Abkürzungen sind im Text bei erstmaliger Verwendung fett gedruckt und farbig hervorgehoben. Zusätzlich werden alle Fachbegriffe und Abkürzungen mit einer entsprechenden Beschreibung im Glossar in alphabetischer Reihenfolge aufgeführt.

1.2 Der Ausbildungsbetrieb

Die BA (Bundesagentur für Arbeit) ist mit rund 110.000 Mitarbeitern in zehn Regionaldirektionen, 156 Agenturen für Arbeit und rund 600 Geschäftsstellen einer der größten Dienstleister auf dem deutschen Arbeitsmarkt. Zu den Hauptaufgaben der BA zählen Leistungen zur Arbeitsvermittlung und Arbeitsförderung, sowie die Gewährung finanzieller Entgeltersatzleistungen.
Die hierarchische Struktur der BA, mit Sitz der Zentrale in Nürnberg, bringt neben den Regionaldirektion noch sieben weitere besondere Dienststellen hervor, worunter auch das IT-SYS (IT-Systemhaus) fällt. Das IT-SYS betreut als operativer IT-Dienstleister der BA eine der größten IT-Landschaften Deutschlands. Es besteht aus den drei Geschäftsführungsbereichen IT-P (IT-Produktion), SE (Systementwicklung) und dem IT-AS (IT-Anwenderservice).

1.3 Das Projektumfeld

Im Rahmen meiner Ausbildung zum Fachinformatiker in der Anwendungsentwicklung im Bereich SE bin ich während des dritten Lehrjahres dem Fachverfahren **DELTA.NT** (Dezentrales Testvorgabe – und Testauswertungssystem) unterstellt. Das Fachverfahren ist im Geschäftsbereich 1, Servicebereich **SEP12** (Systementwicklung Produkte 12 – Serviceübergreifende Dienste II) angesiedelt und stellt ein Anwendungssystem für den Berufspsychologischen Service der BA dar. Dieser führt unter anderem Tests, wie den Berufswahltest durch, welcher die Entscheidung für den Berufseinstieg erleichtern soll.

DELTA.NT ist in drei Einheiten **DELTA-U** (DELTA-Untersuchungssystem), **DELTA-SB** (DELTA-Sachbearbeitung), sowie **DELTA-Z** (Delta-Zusatzdaten) gegliedert. Das Untersuchungssystem ermöglicht Kunden die Durchführung diverser Tests und speichert die damit gewonnen Ergebnisse. Diese Daten werden zur Aufbereitung und Auswertung an die Sachbearbeitung übermittelt. DELTA-Z verwendet diese Daten zur Verbesserung bestehender und Erstellung neuer Tests.

1.4 Das Projektziel

Bisher gibt es im Servicebereich SEP12 keine Möglichkeit einer zentralen Speicherung und Verwaltung der systembezogenen technischen, sowie der verfahrensbezogenen Daten. Diese Daten umfassen unter anderem Benutzerkennungen und Passwörter der Server, aber auch Release-Pläne und andere versionsbezogene Daten. Aus Gründen des Datenschutzes und der Datensicherheit, ist eine Benutzerverwaltung, inklusive Rollen- und Berechtigungskonzept als Grundlage nötig. Daher habe ich den Auftrag bekommen, jene Benutzerverwaltung zu entwickeln.

2 Planung

2.1 Analyse

2.1.1 Projektdefinition

In Zusammenarbeit mit dem Kunden, erarbeitete ich mir eine genauere Definition des Projektes. Um die Benutzerverwaltung als Webapplikation fachlich und technisch umsetzen zu können, entwerfe und erstelle ich die Oberflächen, entwickle und implementiere die Programmlogik und entwerfe ein Datenbankmodell anhand dessen, ich die Datenbank modelliere. Darüber hinaus entwickle ich ein Rollen- und Berechtigungskonzept. Anhand der Projektdefinition und in Absprache mit dem Kunden erarbeitete ich mir im Anschluss die Anforderungsanalyse.

2.1.2 Anforderungsanalyse

Mittels der von mir zu erstellenden Benutzerverwaltung soll unter anderem das Schutzziel der Vertraulichkeit realisiert werden. Hierfür ist die zentrale Aufgabe der Benutzerverwaltung die Authentifizierung der Nutzer und die Bereitstellung ihrer Rechte innerhalb der darauf aufbauenden Webanwendung.
Zum Erreichen dieser Ziele ergaben sich mir folgende Anforderungen:

- Jeder Anwender soll nur eine Kennung und ein Passwort besitzen.
- Jeder Anwender soll mehreren Verfahren/ Produkten (z.B. DELTA-Z, **COAR**) zugeordnet werden können.
- Jedem Anwender sollen mehrere Rollen innerhalb eines Verfahrens zugeordnet werden können.
-

Außer den funktionalen Anforderungen, ergaben sich mir noch weitere technische Anforderungen:

- Möglichst modularer Aufbau der Anwendung, für eine gute Wartbarkeit und Gewährleistung einer möglichst guten Wiederverwertbarkeit.
- Stetige Erweiterbarkeit, z.B. der Rollen und Verfahren
- Hohe Datensicherheit durch :
    - Verschlüsselte Kommunikation über **HTTPS**
    - Schutz vor **SQL-Injections** und **XSS** (Cross-Site-Scripting)
    - Verschlüsselung der Benutzerkennungen und Passwörter

2.1.3 Zeitplanung

Vor der Projektdurchführung erstellte ich einen Zeitplan, welcher die Projektphasen mit deren Gesamtzeit, sowie die einzelnen Arbeitspakete mit dem geplanten Zeitaufwand darstellt.

Projektphasen	Soll
Analyse	**3h**
Projektdefinition	1h
Anforderungsanalyse	2h
Planung und Vorbereitung	**16h**
Entwurf der Oberflächen	3h
Planung und Entwurf des Rollen- & Berech-	3h

tigungskonzepts	
Planung der Programmlogik	5h
Planung und Entwurf der Datenbank	5h
Realisierung	**32h**
Implementierung der Datenbank	6h
Implementierung der Oberflächen	6h
Implementierung der Programmlogik	20h
Erstellung der graphischen Ausgabe	6h
Qualitätssicherung	**9h**
Code-Optimierung	2h
Programmtest und Fehlerbehebung	7h
Abschluss	**10h**
Dokumentation	8h
Vorstellung des Projektes	1h
Übergabe des Projektes	1h
Gesamt	**70h**

2.2 Design

2.2.1 Verwendete Technologien

Vor der eigentlichen Umsetzung, machte ich mir ausführlich Gedanken über die zu verwendenden Technologien. Bei der Wahl der Programmiersprache entschied ich mich für die Sprache **Java** in der Version 1.7, da sie zu den von der SE gesetzten Standards zur Softwareentwicklung gehört. Ein weiterer Grund ist die Erfahrung, welche ich während meiner Ausbildung bereits damit sammeln konnte. In Verbindung mit Java, benutze ich **JSF 2.2** (JavaServer Faces). JSF ist ein Framework zur Erstellung dynamischer Webanwendungen, welches unter anderem einen „Built-In"-Schutz vor „Cross-Site-Scripting" bietet.

Als Entwicklungsumgebung entschied ich mich für **Eclipse** in der Version 4.5.0, die aktuellste, mir zur Verfügung stehende, Version.

Als Webserver wird mir ein **Apache Tomcat** - Server bereitgestellt. Dieser hat die Spezifikation für **Java Servlets** und **JavaServer Pages** (JSP) implementiert und ermöglicht somit Webanwendungen auf Servlet-Basis auszuführen. Die Datenspeicherung erfolgt in einer **MySQL-DB** (Version 5.7)

Des Weiteren benötige ich für die Umsetzung, diverse **Libraries**. Für die Verschlüsselung der Passwörter verwende ich **jbcrypt**, für die Anbindung der Da-

tenbank **MySQL Connector/J** und ein auf JSF aufbauendes Komponenten-Framework namens **PrimeFaces** (Version 4.2).

2.2.2 Planung und Entwurf der Oberflächen

Als Webanwendung wird die Benutzerverwaltung dem Anwender in einem Webbrowser dargestellt. Das, bei JSF 2 –Webanwendungen, in der Regel verwendete Dateiformat zur Strukturierung und Darstellung der Inhalte, sind **XHTML**-Dateien. Die Anmeldung an die Anwendung realisiere ich über ein Formular.

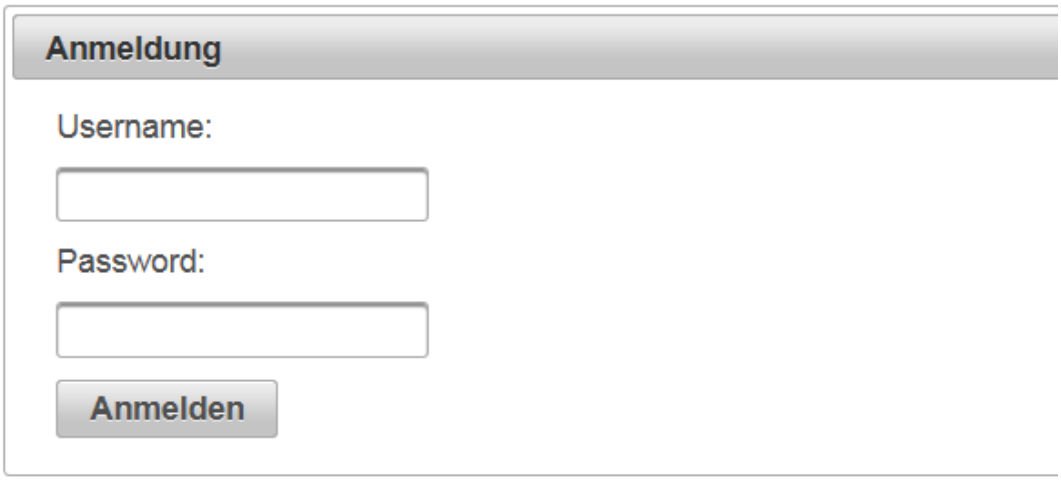

Abbildung 1: Formular zur Anmeldung an die Benutzerverwaltung

Eine zweite XHMTL-Datei, soll es ermöglichen Mitarbeiter, Rollen und Verfahren der Datenbank hinzuzufügen. Dies soll auch über Formulare gelöst werden.

2.2.3 Planung der Programmlogik

Bei der Planung der Programmlogik stellte sich mir zuallererst die Frage nach passenden Entwurfsmustern (auch als Design-Pattern bekannt). Ein weitverbreitetes und häufig genutztes Pattern heißt **MVC** (Model View Controller). Dieses wird mir durch die Verwendung des JSF 2 –Frameworks bereits vorausgesetzt, da es darauf aufbaut und eine Einhaltung des MVC-Patterns bedingt.

Das MVC-Entwurfsmuster zeichnet sich durch seine strikte Trennung der drei Bereiche, Model (Datenhaltung), View (Präsentation) und Controller (Steuerung/ Logik) aus.

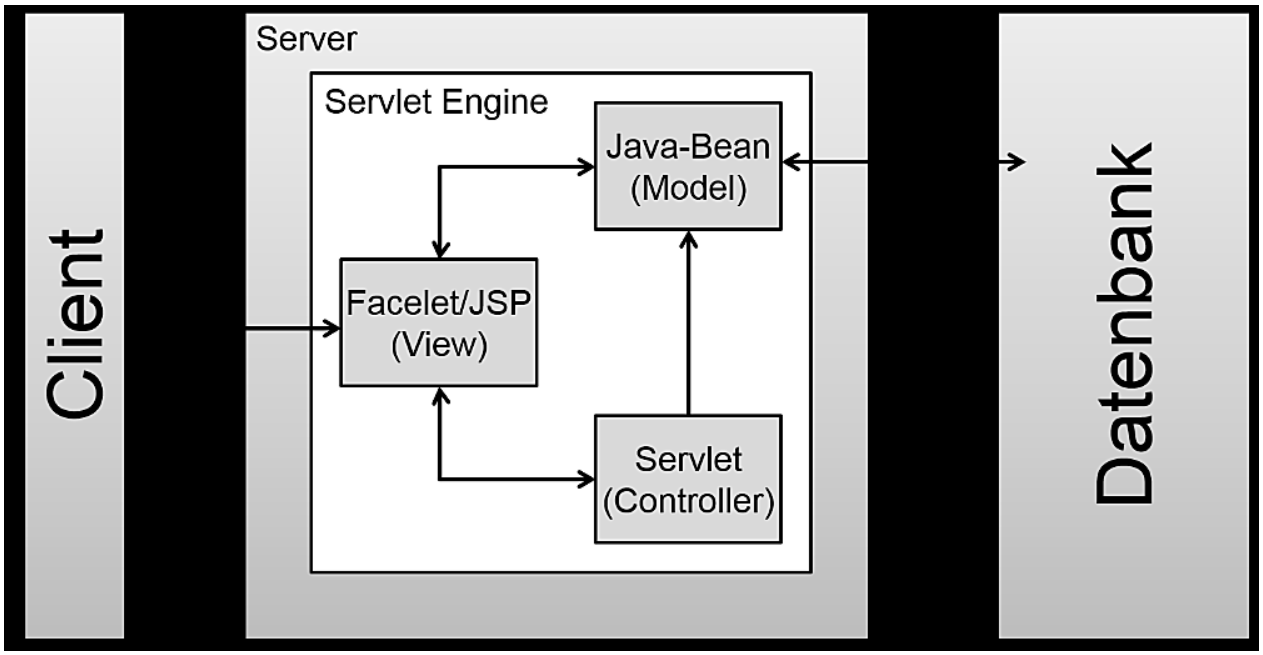

Abbildung 2 : Schema – Model View Controller bei JSF

Danach überlegte und notierte ich mir, welche **POJOs** (Plain Old Java Object) benötigt werden. POJOs einfache Klassen, welche für jedes Attribut eine „getter" und eine „setter" Methode- besitzen. Mit Hilfe der POJOs erarbeitete ich mir die benötigten **Managed Beans**. Diese sind Klassen, welche gewissen Spezifikationen einhalten müssen. Eine Spezifikation ist, dass zu jedem Attribut eine „getter" und eine „setter" Methode existieren muss. Man kann die Managed Beans als Model für die Oberflächenkomponenten betrachten. Des Weiteren ermöglichen sie es, über JSF-Seiten auf sie zuzugreifen. Darüber hinaus besitzen sie einen sogenannten **Scope**. Dieser gibt Aufschluss darüber, wie lange eine Instanz dieser Klasse, im Arbeitsspeicher des Servers existiert.

Im Anschluss stellte ich Überlegungen bezüglich der Anbindung an die Datenbank an und notierte mir welche SQL-Abfragen ich für die Realisierung der Anwendung benötigen werde.

2.2.4 Planung des Rollen – und Berechtigungskonzepts

Nach der Planung der Programmlogik, machte ich mich an den Entwurf eines Rollen und Berechtigungskonzepts. Nach ausgiebigen Überlegungen und Recherche entschied ich mich für das Modell einer Rollenbasierten Zugriffskontrolle.
Das Kernmodell enthält folgende Objekte:

- Benutzer
- Rollen
- Rechte
- Sessions

Unter einer Rolle ist eine Funktion innerhalb der Anwendung zu verstehen. Einer Rolle werden nun genau die Rechte zugeteilt, die sie zur Ausübung ihrer Funktionalität braucht. Die **Rolle** ersetzt somit quasi die Benutzergruppen, wobei diese gezwungener Maßen auf der Grundlage der gemeinsamen Rechte gebildet worden sein müssen, da eine Rolle immer die Gruppierung von Rechten und nicht von Benutzern darstellt. Jedem **Benutzer** werden nun auch entsprechend seiner Aufgaben eine, oder mehrere Rollen zugeteilt.
Allerdings sollen beim Login nicht automatisch alle zugeteilten Rollen und die damit verbundenen **Rechte** aktiviert werden, da ein Benutzer unterschiedliche Rollen in unterschiedlichen Verfahren/Bereichen besitzen können soll.
Vielmehr wird eine **Session** erzeugt, die die benötigte Rolle(n), des jeweiligen Verfahrens, aktiviert.
Zu beachten ist, dass ein Benutzer mehrere Sessions haben kann, allerdings kann jede Session nur einen Benutzer haben.

Die rollenbasierte Zugriffskontrolle zeichnet sich aus durch ihre enorme Flexibilität der Rechtevergab, und eignet sich somit gut als Authorisierungssystem für Systeme mit einem hohen Grad an Veränderung.

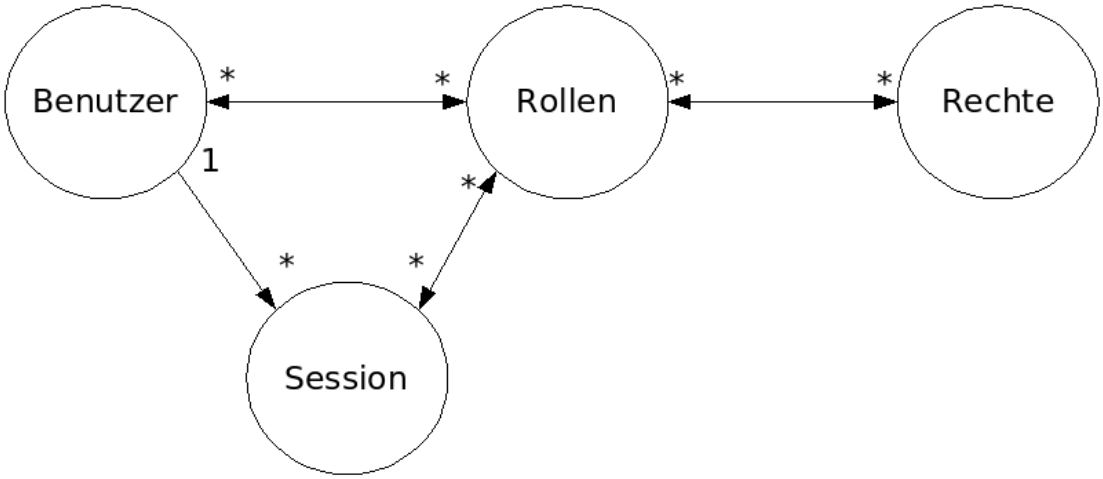

Abbildung 3: Kernmodell einer Rollenbasierten Zugriffskontrolle

2.2.5 Planung und Entwurf der Datenbank

Nach Absprache mit den Kollegen aus der Entwicklung wurden mir die zu verwendenden Technologien vorgegeben. Als Datenbank stand mir eine MySQL-Datenbank zur Verfügung. Im Anschluss entwarf ich, mit Hilfe der zuvor erstellten Anforderungsanalyse, folgendes **ER-Modell**, welches die Tabellen, sowie die Primär- und Fremdschlüsselbeziehungen enthält.

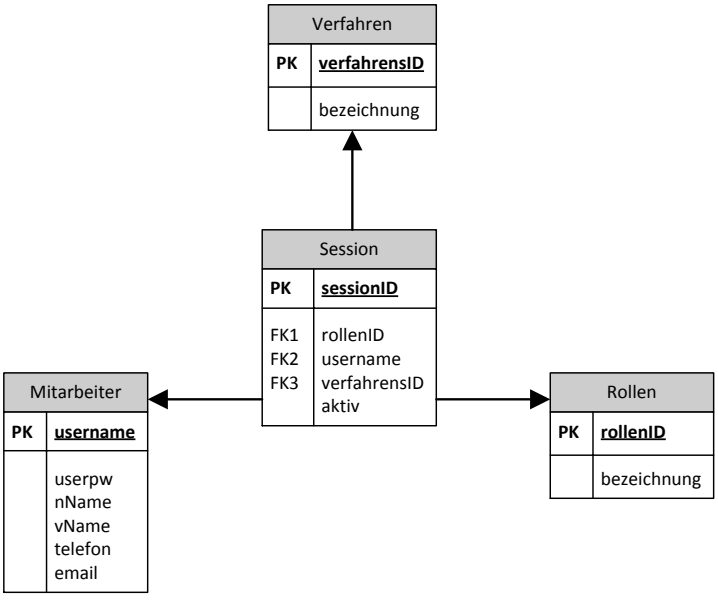

Abbildung 4: ER-Modell

3 Realisierung

3.1 Implementierung der Datenbank

Zum Erzeugen der Datenbanktabellen nahm ich mir das ER-Modell als Vorlage zu Hilfe. Zuallererst erstellte ich mir zum Anlegen jeder einzelnen der Tabellen ein entsprechendes SQL-Script, sodass selbst bei einem totalen Verlust der Datenbank, mir die Tabellenstrukturen erhalten bleiben.

Beispiel für die Tabelle ROLLEN:

```
mysql> create table rollen(
    -> rollenID INT UNSIGNED PRIMARY KEY NOT NULL,
    -> bezeichnung varchar(100) NOT NULL);
```

Nach dem ich alle Tabellen mit den entsprechenden Attributen erzeugte, verknüpfte ich diese über ihre Primär- und Schlüsselbeziehungen miteinander. Diese entnahm ich ebenfalls dem ER-Modell.

3.2 Implementierung der Oberflächen

Anhand der mir vorliegenden Skizzen, erstellte ich zwei XHTML-Dateien
Als Erstes realisierte ich die „Anmelde"-Seite. Hierfür verwendete ich ein sogenanntes Formular. Dieses enthält ein sogenanntes „panelGrid" , welches mir die Strukturierung der Inhalte des Formular ermöglicht. Jenes „panelGrid" besteht aus zwei Spalten und enthält zwei „Textlabels", sowie zwei „Input-Felder" und einen „Anmelde"-Button. Die beiden Inputfelder, wies ich jeweils einer Variable einer Managed Bean zu und den „Anmelde"-Button der Methode „*doIT*" zu, welche den Anmeldevorgang anstößt.

- „ ... *value="#{userService.user.username}* ..."
- „ ... *value="#{userService.user.password}* ..."
- „... *commandButton action="#{userService.doIT}* ..."

```
<h:form id="form">
    <p:panel header ="Anmeldung">
        <h:panelGrid colums="2" cellpadding="5">
            <h:outputLabel for="username" value="Username:"/>
            <p:inputText id="username" required="true" value="#{userService.user.username}"></p:inputText>

            <h:outputLabel for="password" value="Password:"/>
            <p:password id="password" required="true" value="#{userService.user.password}"></p:password>
            <p:commandButton action="#{userService.doIT}" value="Anmelden" update="@form"/>
        </h:panelGrid>
    </p:panel>
</h:form>
```

Abbildung 5 : Formular zur Anmeldung

Im Anschluss erstellte ich die XHTML-Seite, welche nach erfolgreichem Login angezeigt wird.

3.3 Implementierung der Programmlogik

Bei der Implementierung der Programmlogik begann ich damit, alle benötigten POJOs und somit alle ihre Variablen und dazugehörigen „getter-“ und „setter- Methoden“ zu erstellen. So entstand pro Tabelle meiner Datenbank je eine POJO- Klasse.

```
package pojos;

import java.io.Serializable;

public class User implements Serializable {

    private String password;
    private String username;
    private boolean aktiv;

    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public boolean isAktiv() {
        return aktiv;
    }
    public void setAktiv(boolean aktiv) {
        this.aktiv = aktiv;
    }
}
```

Abbildung 6: POJO „User.java“

Danach implementierte ich passend zu jedem POJO eine Managed Bean. Diese dienen zum einen als Schnittstelle zur View(Oberfläche), da sie Objekte und Methoden enthalten, welche von der View angesprochen werden können. Dies geschieht über das sogenannte Mapping, welches ich bereits in Abschnitt 3.2 Implementierung der Oberfläche beschreibe.
Um die Datenbankanbindung zu realisieren, waren mehrere Schritte nötig.
Zuallererst bearbeitete ich die Konfigurationsdatei „web.xml“ und hinterlegte dort eine Referenz auf eine Datenquelle.
Im Anschluss legte ich eine weitere Konfigurationsdatei „context.xml“ an. Diese enthält neben diversen Parametern, wie „username“ oder „password“, die URL der Datenbank, mit welcher es für die Anwendung möglich ist eine Verbindung zu jener Datenbank herzustellen.

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <Resource name="jdbc/benutzerverwaltung" auth="Container" type="javax.sql.DataSource"
              maxActive="50" maxIdle="30" maxWait="10000"
              username="root" password="root"
              driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost:3306/benutzerverwaltung"/>
</Context>
```

Abbildung 7: Auszug aus „context.xml“

Danach erstellte ich die Klasse „DBConnector“. Diese enthält unter anderem die Methode „connectToDB“, welche eine Connection(Verbindung) zur Datenbank herstellt und eine sogenannte „Datasource“, als Rückgabewert, an die jeweilige Managed Bean übergibt. Sollte der Verbindungsaufbau zur Datenbank nicht gelingen wird eine Fehlermeldung „Keine DB Verbindung!“ ausgegeben.

```
public DBConnectionDataSource connectToDB(){
        try {
            this.dbDataCon = new DBConnectionDataSource();
            Context ctx = new InitialContext();
            ds = (DataSource)ctx.lookup("java:comp/env/jdbc/benutzerverwaltung");
            dbDataCon.setDs(ds);
            try {
                 dbDataCon.setCon(ds.getConnection());
                 dbDataCon.setDs(ds);
            } catch (SQLException e) {
                System.out.println("Keine DB Verbindung!");
                e.printStackTrace();
            }
        } catch (NamingException e) {
            e.printStackTrace();
        }
    return dbDataCon;
}
```

Abbildung 8: Methode „connectToDB“ der Klasse „DBConnector“

Als nächstes implementierte ich die **PreparedStatements**. Diese SQL-Statements bieten den Vorteil, dass man sich mit ihnen vor SQL-Injections schützt, da nur die im Programmcode als PreparedStatements hinterlegten SQL-Abfragen getätigt werden können.
Folgende Funktionalitäten habe ich mittels PreparedStatements, gebündelt in der Klasse „PrepStatementController“ realisiert:

- Das Hinzufügen zur Datenbank von:
    - Rollen
    - Verfahren
    - Mitarbeitern
- Die Validierung von Username und Passwort → Enthält die Datenbank einen Mitarbeiter mit entsprechenden Zugangsdaten.

```
public void addRolle(Rolle rolle) throws SQLException {
    if(ds==null){
        throw new SQLException("Keine Verbindung zur Datenbank vorhanden");
    }
    PreparedStatement ps = con.prepareStatement("INSERT INTO rollen" + "(rollenID, bezeichnung) VALUES " + "(?,?)");
    ps.setInt(1, rolle.getRollenID());
    ps.setString(2, rolle.getRollenBezeichnung());
    ps.executeUpdate();
    System.out.println("Rolle hinzugefügt");
}
```

Abbildung 9: Auszug aus „PrepStatementController.java“

Die Methoden der Klasse PrepStatmentController, welche die Prepared-Statements enthalten, werden wiederum von den Managed Beans aufgerufen.

Als nächstes implementierte ich die Verschlüsselung der User-Passwörter.

Hierfür fügte ich zu Erst die API „jbcrypt“ meinem Projekt hinzu.

„bcrypt“ ist eine, speziell für das Hashen und Speichern von Passwörtern entwickelte kryptologische Hashfunktion.

Die Implementierung sieht z.B. folgendermaßen aus:

```
public String doIT(){

    //Holt das Userpassword des anmeldenen Users aus der DB und speichert es in originalPassword
     originalPassword = controller.getUserPW(username);

     // Hasht und Vergleicht die Hashes der Passwörter miteinander
     String generatedSecuredPasswordHash = BCrypt.hashpw(eingegebenesPasswort, BCrypt.gensalt(12));
     boolean matched = BCrypt.checkpw(originalPassword, generatedSecuredPasswordHash);

     //Falls der Login klappt -> redirect zu  neuer Seite -> loggedIn.xhtml
     if(matched){
        return "/adminSite/loggedIn.xhtml?faces-redirect=true";
     }

     //Falls Passwort nicht korrekt -> redirect zur Startseite -> index.xhtml
     else{
        return "/adminSite/index.xhtml?faces-redirect=true";
     }
}
```

Abbildung 10: Auszug aus „UserService.java“

Als letzten Teil der Implementierung, realisierte ich die Kommunikation über HTTPS. Zuallererst erstellte ich einen **Java Key Store** über folgenden Befehl:

➔ `"%JAVA_HOME%\bin\keytool" -genkey -alias tomcat -keyalg RSA`

Als nächstes musste ich die Serverkonfigurationsdatei „server.xml“ bearbeiten. Dort fügte ich folgendes hinzu:

```
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->
<Connector
           protocol="org.apache.coyote.http11.Http11NioProtocol"
           port="8443" maxThreads="200"
           scheme="https" secure="true" SSLEnabled="true"
           keystoreFile="${user.home}/.keystore" keystorePass="changeit"
            clientAuth="false" sslProtocol="TLS"/>
```

Abbildung 11: Auszug aus der Serverkonfigurationsdatei „server.xml“

Nun kommuniziert der Anwender über eine sichere Verbindung mit der Anwendung, welche sich auf dem Applikationsserver befindet, wie folgende Darstellung zeigt.

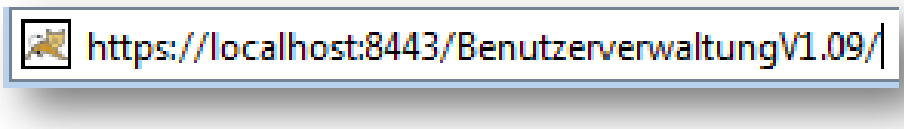

Abbildung 12: Kommunikation über HTTPS und Port 8443

4 Qualitätssicherung

4.1 Code-Optimierung

Nachdem ich die Implementierung abgeschlossen habe, begann ich damit, den Programmcode an gegebenen Stellen zu optimieren. Als Erstes, fügte ich an Stellen des Codes, die mir als Sinnvoll erschienen, Kommentare hinzu, um den Programmcode leichter verständlich zu machen und um die Wartbarkeit zu erhöhen.

Im Anschluss entfernte ich alle überflüssigen, meist bereits auskommentierten Codeabschnitte, sowie diverse „Sys-Out-Testausgaben“. Darüber hinaus, überarbeitete ich die Namensgebung bezüglich meiner Klassen und Methoden.

4.2 Programmtest und Fehlerbehebung

Um die jeweiligen Anforderungen zu gewährleisten, führte ich nach Abschluss einzelner Einheiten Tests durch. Hierfür nutzte ich das White-Box-Verfahren. Mithilfe der Debug-Funktion, die mir Eclipse bereitstellt, setzte ich Haltepunkte an bestimmten Positionen meines Programmcodes. Somit ist es mir möglich gewesen, den genauen Ablauf der Anwendung, sowie die aktuell zugewiesenen Werte der Variablen zu verfolgen. Aufgetretene Fehler im Programmcode, habe ich nach ihrer Entdeckung beseitigt.

5 Abschluss

5.1 Zeitaufwand

Projektphasen	Soll	Ist
Analyse	**3h**	**3h**
Projektdefinition	1h	1h
Anforderungsanalyse	2h	2h
Planung	**16h**	**15h**
Entwurf der Oberflächen	3h	2h
Planung und Entwurf des Rollen- & Berechtigungskonzepts	3h	4h
Planung der Programmlogik	5h	5h
Planung und Entwurf der Datenbank	5h	4h
Realisierung	**32h**	**33h**
Implementierung der Datenbank	6h	5h
Implementierung der Oberflächen	6h	5h
Implementierung der Programmlogik	20h	23h
Qualitätssicherung	**7h**	**7h**
Code-Optimierung	3h	3h
Programmtest und Fehlerbehebung	4h	4h
Abschluss	**10h**	**10h**
Dokumentation	8h	8h
Vorstellung des Projektes	1h	1h
Übergabe des Projektes	1h	1h
Gesamt	**70h**	**70h**

Nach Abschluss des Projekts stellte ich, wie in der Tabelle ersichtlich, die ursprüngliche Zeitplanung dem tatsächlichen Zeitaufwand gegenüber.
Im Vergleich zu meiner Planung, ergaben sich während der Durchführung wenige Abweichungen. Den Entwurf der Oberflächen konnte ich etwas

schneller fertigstellen, benötigte dafür bei der Ausarbeitung des Rollen- & Berechtigungskonzepts ein wenig mehr Zeit.
Bei der Implementierung der Datenbank, sowie der Implementierung der Oberflächen, benötigte ich weniger Zeit als angenommen, weswegen ich die IST-Analyse bei beiden Punkten um je eine Stunde zu hoch eingeschätzt hatte.
Allerdings benötigte ich dafür mehr Zeit bei der Implementierung der Programmlogik. Dies lag vor allem daran, dass die Wahl der Scopes, in Zusammenhang mit dem JSF-Life Cycle, ein neues Thema für mich darstellte, in welches ich mich, länger als vermutet, einarbeiten musste.
Trotz der kleineren Probleme waren das Projekt, sowie die Gesamtdauer von 70 Stunden nie gefährdet.

5.2 Fazit und Ausblick

Zusammenfassend kann ich sagen, dass ich mit dem Verlauf des Projektes sehr zufrieden bin. Die vom Auftraggeber gestellten Anforderungen an die Benutzerverwaltung konnte ich erfolgreich umsetzen. Die von mir entwickelte Benutzerverwaltung kann nun als Grundlage weiterer Webapplikation, wie z.B. bei der zentralen Verwaltung von sensiblen Daten, also Daten mit einem Hohen Schutzbedarf eingesetzt werden. Kriterien, wie „hohe Datensicherheit“ oder „modularer Aufbau“ konnte ich zur vollsten Zufriedenheit des Kunden umsetzen.
Hinsichtlich meiner fachlichen Kenntnisse, konnte ich mein bestehendes Wissen in vielen Teilbereichen der Softwareentwicklung erweitern. Darunter fallen der Umgang mit der Programmiersprache Java, das Entwerfen und Modellieren von Datenbanken, sowie sicherheitsspezifische Themen bezüglich der Verschlüsselung und Kommunikation.
Im Anschluss an dieses Projekt, sollen zukünftig, möglichst alle neu entwickelten und intern genutzten Webanwendungen des Bereichs SEP12, auf der von mir entwickelten Benutzerverwaltung aufbauen.

6 Glossar

Bezeichnung	Erläuterung
Apache Tomcat	Open-Source-Webserver und Webcontainer
API	Eine Programmbibliothek, die in ein eigenes Programm eingebunden und benutzt werden kann
BA	Bundesagentur für Arbeit
bcrypt	Kryptologische Hashfunktion
COAR	Computerunterstütze Agenturrevision
DELTA.NT	Dezentrales Testvorgabe- und Testauswertungssystem
DELTA-SB	Delta-Sachbearbeitung
DELTA-U	Delta-Untersuchung
DELTA-Z	Delta-Zusatzdaten
Eclipse	Integrierte Entwicklungsungebung
ER-Modell	Entity-Relationship-Modell
HTTPS	HyperText Transfer Protocol Secure
IT-AS	Geschäftsführungsbereich „IT-Anwenderservice“
IT-P	Geschäftsführungsbereich „IT-Produktion“
IT-SYS	IT-Systemhaus
Java	Objektorientierte Programmiersprache
Java Key Store	Verzeichnis von Sicherheitszertifikaten
Java Servlets	Java-Klassen, deren Instanzen Anfragen von Clients entgegennehmen
jbcrypt	Java API – ermöglicht die Verwendung der bcrypt-Hashfunktion
JSF	JavaServer Faces
JSP	JavaServer Pages
Library	Programmbibliothek
Managed Bean	Java-Klasse, welche gewissen Spezifikationen genügt
MVC	Model View Controller - Entwurfsmuster
MySQL Connector/J	Treiber zur MySQL Datenbankverbindung
MySQL DB	Relationales Datenbankverwaltungssystem MySQL
POJO	Plain Old Java Object
PreparedStatement	Vorbereitete Anweisung in Java für Datenbankabfrage
PrimeFaces	User Interface Framework für Java EE Entwicklung
Properties-Datei	Datei in der Schlüssel-Wert-Beziehungen stehen
Scope	Gültigkeitsdauer einer Managed Bean
SEP12	Systementwicklung Produkte; Geschäftsbereich 1, Service 12 – Serviceübergreifende Dienste II
SQL-Injection	Ausnutzen einer Sicherheitslücke in Zusammenhang mit SQL-Datenbanken
White-Box-Verfahren	Softwaretest mit Kenntnis über die innere Funktionsweise des Systems
XSS	Cross-Site-Scripting
XHTML	Extensible HyperText Markup Language

7 Quellenangaben

Internet (Zugriffe fanden zwischen März und Mai 2016 statt):

- http://www.torsten-horn.de/techdocs/jsf.htm
- http://openbook.rheinwerk-verlag.de/javainsel/
- https://dbis.ipd.kit.edu/download/SS07GroppRbac-Ausarbeitung.pdf
- https://tomcat.apache.org/tomcat-7.0-doc/ssl-howto.html
- http://www.w3schools.com/sql/
- http://www.mindrot.org/projects/jBCrypt/
- http://www.coreservlets.com/JSF-Tutorial/jsf2/
- http://www.primefaces.org/
- https://dev.mysql.com/doc/
- https://de.wikipedia.org/wiki/Model_View_Controller
- http://stackoverflow.com/

Literatur:

- „Entwurfsmuster von Kopf bis Fuß“ (Eric Freeman, Elisabeth Freeman – O`Reilly – 2006)

8 Abbildungsverzeichnis

Abbildung 1 : Formular zur Anmeldung an die Benutzerverwaltung 5
Abbildung 2 : Schema – Model View Controller bei JSF 6
Abbildung 3 : Kernmodell einer Rollenbasierten Zugriffskontrolle 8
Abbildung 4 : ER-Modell 8
Abbildung 5 : Formular zur Anmeldung 10
Abbildung 6 : POJO „User.java“ 10
Abbildung 7 : Auszug aus „context.xml“ 11
Abbildung 8 : Methode „connectToDB“ der Klasse „DBConnector“ 12
Abbildung 9 : Auszug aus „PrepStatementController.java“ 13
Abbildung 10: Auszug aus „UserService.java“ 13
Abbildung 11: Auszug aus der Serverkonfigurationsdatei „server.xml“ 14
Abbildung 12: Kommunikation über HTTPS und Port 8443 14