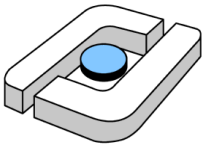

Hochschule Osnabrück

University of Applied Sciences

Fakultät

Ingenieurwissenschaften und Informatik

Schriftliche Ausarbeitung zum Thema:

Erstellung der Java Webanwendung Supercar zur Verwaltung eines Autoverleihs

im Rahmen des Moduls
Komponentenbasierte Softwareentwicklung,
des Studiengangs Informatik-Medieninformatik

Autor:	Lukas Bernhold
Matr.-Nr.:	740592
E-Mail:	lukas.bernhold@hs-osnab-rueck.de

Autor:	Maximilian Nussbaum
Matr.-Nr.:	732386
E-Mail:	maximilian.nussbaum@hs-osn-abrueck.de

Autor:	Patrick Wiethoff
Matr.-Nr.:	738195
E-Mail:	patrick.wiethoff@hs-osnab-rueck.de

Themensteller: Prof. Dr. Rainer Roosmann

Abgabedatum: 20.07.2018

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Source-Code Verzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
CDI	Context and Dependency Injection for the Java EE Plattform
CRUD	Create Read Update Delete
ECB	Entity-Controller-Boundary Pattern
EJB	Enterprise Java Beans
JAAS	Java Authentication and Authorization Service
JSON	JavaScript Object Notation
Java EE	Java Enterprise Edition, in der Version 8
JPA	Java Persistence API
JSF	Java Server Faces
JSP	JavaServer Pages
KBSE	Komponentenbasierte Softwareentwicklung
MVC	Model View Controller
ORM	Object-relational mapping
PDF	Portable Document Format
SFLB	Statefull Session Bean
SLSB	Stateless-Session Bean
UI	User Interface

1 Einleitung

Diese Arbeit beschäftigt sich mit dem Thema der Ausarbeitung einer Java EE Web Applikation zur Verwaltung von Autos eines Autoverleihs unter Verwendung von JSF, JPA und weiteren Java EE Komponenten. Ziel dieser Arbeit ist die effektive Verwendung fortgeschrittener Programmierkonzepte wie zum Beispiel Software Design-Pattern.

1.1 Vorstellung des Themas

1.1.1 Funktionale Anforderungen

Die funktionalen Anforderungen dieser Arbeit sind die Erstellung der Webseite “Supercar” mit folgenden Features. Auf der Webseite können Benutzer sich Anmelden und dann aus einer Liste von verfügbaren Mietwagen auswählen. Durch eine Suchfunktion wird das Finden der Autos vereinfacht. Wenn der Benutzer ein Auto ausgewählt hat, kann er dieses mieten und später zurückgeben, wobei er bei der Rückgabe Einträge in ein Fahrtenbuch machen muss und zusätzlich Angaben zu Mängeln am Fahrzeug angeben kann. Nachdem ein Ausleihvorgang abgeschlossen ist, findet der Benutzer seine Rechnungen unter seinem Account und kann diese als PDF Datei herunterladen.
Neben den Benutzern gibt es noch die Rollen Admin und Mitarbeiter, wobei ein Administrator neue Fahrzeuge inklusive ihrer Metadaten anlegen darf und alle Fahrzeuge in die Werkstatt schicken kann. Mitarbeiter können Nutzer sperren und für die Ausleihe freischalten sowie neue Werkstätten hinzufügen.

1.1.2 Nicht Funktionale Anforderungen

Das Projekt ist mit Java EE 8 unter der Verwendung von JSF 2.2 bzw. Primefaces und Bootsfaces entwickelt, wobei mit Hilfe von CSS ein paar Änderungen vorgenommen wurden. Die Anmeldung ist eine Eigenentwicklung, da die zur Verfügung stehenden Frameworks den Umfang der benötigten Funktionalität übersteigen. Für die Datenbankverwaltung wird JPA in Verbindung mit JTA genutzt. Zusätzlich werden alle Benutzereingaben mit Hilfe von BeanValidation überprüft. Die Objekte werden durch CDI injiziert und die Datenbankabfragen werden durch mit EJB als Stateless annotierte Klassen übernommen.

1.2 Ziel der Ausarbeitung

Das Ziel diese Ausarbeitung ist die Darstellung der Umsetzung dieses Projektes. Der Leser soll einen Eindruck bekommen, wie das Projekt aufgebaut ist und welche Probleme bei der Entwicklung aufgetreten sind. Es soll dargestellt werden, welche Erfahrungen im Laufe des Projektes gemacht wurden und welche Vor- und Nachteile die Entwicklung einer Webanwendung mit Java EE 8 mit sich bringt. Zusätzlich wurde eine ähnliche Anwendung bereits in einem anderen Zusammenhang mit Hilfe von PHP und SQL umgesetzt, daher wird diskutiert, welche wesentlichen Unterschiede vorhanden sind und welche Technologie wofür besser geeignet ist.

1.3 Aufbau der Hausarbeit

Im Folgenden werden zuerst die allgemeinen und die technischen Grundlagen erläutert, danach wird auf den Aufbau des Projektes und die einzelnen Komponenten eingegangen. Nachdem der Aufbau dargestellt wurde, werden einige Fehler beschrieben, die zu Beginn häufig aufgetreten sind und an die man sich erst gewöhnen muss. Zusätzlich wird auf einige größere Probleme eingegangen und wie diese gelöst oder umgangen wurden. Am Ende wird das Ergebnis des Projektes noch einmal zusammengefasst und im Fazit wird noch einmal beleuchtet, welche Erfahrungen gemacht wurden.

2 Darstellung der Grundlagen

Die Grundlagen der Projektarbeit basieren auf Java EE, JPA, CDI, JSF, GlassFish und Docker. Im Folgenden werden die Grundlagen der einzelnen Technologie kurz erläutert.

2.1 Java EE

Java EE ist eine Middleware Plattform, welche den Fokus hauptsächlich auf transaktionsbasierte in Java programmierbare Webanwendungen legt. Um dies zu realisieren, wird an einer Spezifikation gearbeitet, welches das Verwenden von modularen Komponenten definiert, welche ebenfalls hauptsächlich in Java erstellt wurden. Dadurch wird ein System mit klar definierten Schnittstellen zwischen den Komponenten geschaffen, welches zwischen unterschiedlichen Herstellern interoperabel ist.

2.2 JPA

JPA ist eine Schnittstelle um die Anbindung an die Datenbank unter anderem in einem objektorientierten Stil zu vereinfachen. Zusätzlich dazu lassen sich Relationen im Code über Annotationen abbilden, welche von JPA zur Laufzeit aufgelöst werden. Es lassen sich aber auch normale SQL-Abfragen durchführen, diese werden als native Query bezeichnet. Bei diesen ist allerdings darauf zu achten, valide Abfragen zu erzeugen.

Die Open-Source-API, wird von verschiedenen Anbietern wie z.B. Oracle oder Eclipse implementiert. [@tutorialspoint] Neben der Standardspezifikationen von JPA werden, von den Anbietern, noch eigene Spezifikationen in die Schnittstelle integriert. Die bekanntesten Produkte sind Hibernate, Eclipselink, Toplink und Spring Data JPA.

2.3 CDI

Oracle schreibt über CDI, dass es eine Zusammenstellung von Services ist. Diese Services erleichtern es den Entwicklern Enterprise-Beans zusammen mit der JSF-Technologie in Webanwendungen zu verwenden. Außerdem schreibt Oracle, dass CDI für die Verwendung mit zustandsbehafteten Objekten entwickelt wurde und ermöglicht so eine große Flexibilität bei der Integration verschiedener Arten von Komponenten, in einer lose gekoppelten aber Typen sicheren Weise.[@javaee]

2.4 JSF

JSF ist ein Frameworkstandard um eine grafische Benutzeroberfläche für Java Webanwendungen zu gestalten. Durch Verwendung von Servlets und JSP Technik gehört das Framework zu den Webtechnologien, die in Java EE verwendet werden. Mit Hilfe von JSF kann der Entwickler auf eine einfache Art und Weise Webseiten zur Benutzerinteraktion oder andere Komponenten definieren. [@oracle]

JSF wird bei der Ausführung in HTML- und JavaScript-Code umgewandelt und ermöglicht dadurch eine Anzeige der Webseite in jedem gängigen Browser. Außerdem kann das Standardframework noch durch andere Frameworks, wie z.B. PrimeFaces oder BootsFaces, erweitert werden. Die Erweiterungen bieten mehr Gestaltungsmöglichkeiten für die Webseite an, als eine reine JSF Webseite.

2.5 GlassFish

GlassFish ist eine Applikationsserver der auf Java EE basiert und welcher Java um Funktionen bezüglich Webentwicklung erweitert. Mit der Hilfe von GlassFish können Java EE Webanwendung ausgeführt und veröffentlicht werden.

2.6 Docker

Docker ist eine Software zur Isolierung von Programmen, ähnlich wie in einer virtuellen Maschine. Es lassen sich sogenannte Container erstellen, welche von Docker auf dem Host System ausgeführt werden. Der Vorteil von Docker gegenüber virtuellen Maschinen ist ein geringerer Memory Footprint und somit eine schnelle Startzeit. Man kann davon ausgehen, dass die Container auf egal welchem Host System gleich ausgeführt werden, wodurch sich ein reibungsloses Deployment realisieren lässt. [@Docker]

3 Anwendung

3.1 MVC Pattern in Java

Das Model-View-Controller Pattern unterteilt die Software in drei Klassen. Die Models, die die Bearbeitung der Business Logic übernehmen. Die Controller, die für die Datenbankverbindung zuständig sind und die Views, die die Darstellung realisieren. In diesem Projekt ist ein Repository vergleichbar mit einem Controller, ein Model bleibt ein Model und eine XHTML-Datei entspricht einer View. Dadurch kann sichergestellt werden, dass alle Datenbankzugriffe durch ein Repository übernommen werden und die Validierung von Nutzerdaten in einer Model Klasse passiert. Diese Unterteilung verringert insbesondere Code Duplizierung und vereinfacht die Verhinderung von SQL-Injektions und ähnlichen Manipulationsversuchen.

3.2 Datenbankstruktur

Die Datenbankstruktur besteht, wie in der Abbildung auf der rechten Seite zu sehen, aus acht Tabellen. Abzüglich der SEQUENCE Tabelle die automatisch generiert wurde. Die Anderen wurden von JPA aus den Entity Klassen generiert.

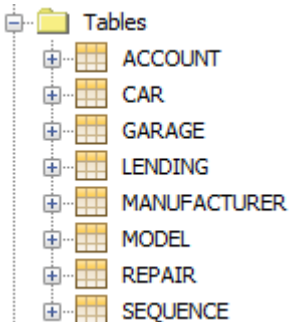

Abbildung 1: Entitäten

3.2.1 Entities

Jede Entity Klasse erbt von der abstrakten Klasse IUniqueEntity und erweitert diese. Diese implementiert die grundlegenden Entity Eigenschaften wie eine einzigartige ID vom Typ Long, welche durch die Annotation @GeneratedValue mit der Strategie GenerationType.AUTO automatisch generiert wird und welche der Entity als Primärschlüssel dient. Außerdem implementiert diese abstrakte Klasse das Interface Serializable und stellt dadurch die Methoden hashCode() und equals() zur Verfügung. Die toString() Methode stellt eine Standard Implementierung zu Verfügung an der man den Klassennamen und die Entity ID erkennen kann.

In den eigentlichen Entities sind dann die entsprechenden Attribute definiert und durch Bean Validation Annotationen wie beispielsweise @NotNull oder andere selbst geschriebene Validatoren überprüft. Zusätzlich zu dem Standard Konstruktor gibt es einen Konstruktor, der zum Erstellen eines neuen Objektes verwendet werden kann. Die Getter und Setter sowie der erwähnte Standard Konstruktor werden von JPA für die Persistierung benötigt und werden automatisch generiert.

Die Kardinalitäten werden ebenfalls durch Annotationen wie beispielsweise @OneToMany definiert, zusätzlich wird der entsprechende Fremdschlüssel des anderen Attributs, falls notwendig durch die Annotation @JoinColumn angegeben, für diese zusätzliche Spalte kann über die Annotation ein benutzerdefinierter Name gesetzt werden. Die Annotation @JoinColumn ist nur bei einer OneToMany Beziehung notwendig, da in diesem Fall ein Fremdschlüssel benötigt wird. In JPA wird eine solche Beziehung durch eine Liste mit Objekten des gegenüberliegenden Entities realisiert. Dies muss allerdings in einer Datenbank durch das Eintragen des Primärschlüssels als Fremdschlüssel in das gegenüberliegende Entity realisiert werden.

Das Entity Account besitzt ein Attribut AccountType, welches später im LoginHandler benötigt wird, dieses kann Werte aus dem Enum AccountType enthalten wodurch die Rollenverwaltung gehandhabt wird. Dazu besitzt ein Account noch die Methode isAtLeast() die true zurückgibt, falls der AccountType größer oder gleich dem Übergebenen AccountType ist.

3.3 Datenbankanbindung

3.3.1 Repository

Die Repository Klassen sind für die Anbindung an die Datenbank zuständig und stellen verschiedene Methoden zur Datenbankabfrage zur Verfügung. Jede Repository Klasse erbt von der abstrakten Klasse IRepository, dieses stellt die grundlegenden CRUD Funktionalitäten bereit und hält ein Objekt des Entity Managers der, über den Persistence Context bekannt gemacht wurde. IRepository ist mit @Transactional Annotiert, dadurch werden alle Operationen auf der Datenbank in einer Transaktion durchgeführt, da auch alle anderen Klassen über die query() Methode dieser Klasse auf die Datenbank zugreifen. Die query() Methode gibt ein Objekt vom Type IQuery zurück und kann zum Ausführen von Benutzerdefinierten Queries genutzt werden.

Die abstrakte Klasse IQuery wird verwendet um Queries zu erstellen und zu verarbeiten. Dabei gibt es die Methode put() um in einer Query eine mit ":name" angegebene Variable durch einen übergebenen String zu ersetzen. Die Methoden one() und all() liefern ein einzelnes oder eine Liste von Ergebnissen und geben zusätzlich die ausgeführte Query in der Konsole aus. Auch die execute() Methode gibt die ausgeführte Query in der Konsole aus und übernimmt ansonsten dieselbe Funktionalität wie die normale executeUpdate() Methode, ist also für alle benutzerdefinierten Update, Insert oder Delete Operationen zuständig. Wie man sehen kann, unterscheidet sich die IQuery Klasse nicht sonderlich von der normalen Query Klasse, allerdings ist diese essentiell für das Ausgeben der Abfragen in die Konsole und das Abfangen von eventuell auftretenden Fehlern. Außerdem erlaubt es dieser Wrapper eigene Standardwerte für die Methoden one() und all() zu definieren, dadurch werden in der Model Klasse, zumindest beim Benutzen der all() Methode, einige Überprüfungen auf null gespart, da diese Methode, falls sie keine Ergebnisse in der Datenbank findet, eine leere Liste anstatt null zurückgibt.

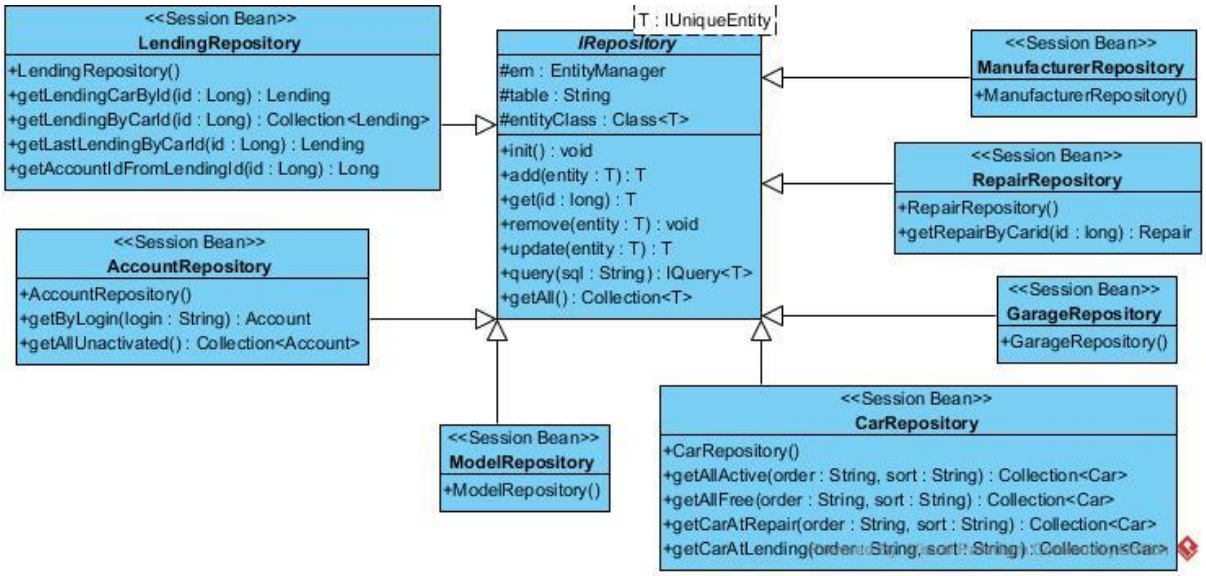

Abbildung 2: Repository Klassendiagramm

3.4 Login System

Das Login System wurde durch eine Eigenentwicklung umgesetzt. Anfangs wurde überlegt JAAS, Java EE Security oder Keycloak zu verwenden, allerdings schien eine Eigenentwicklung unter Verwendung von Filtern als beste Lösung für den gewünschten Anwendungszweck. Das dies allerdings zu Problemen führen würde, da der ExternalContext zu dem Zeitpunkt der Filterung noch nicht gesetzt ist, war zu der Zeit nicht ersichtlich. JAAS ist eine Lösung welche sich hauptsächlich über XML-Dateien konfigurieren und über Annotationen verwenden lässt. Eine Einbindung in die XHTML-Dateien bzw. in die Views wäre dadurch erschwert worden. Java EE Security, oder auch JSR-375, erschien sehr repetitiv, allerdings ließ sich vergleichsweise wenig Literatur zu diesem Thema finden, vor allem in Verbindung mit JSF. Keycloak hingegen erschien sehr gut dokumentiert, ist aber eine zentralisierte Lösung welche sich projektübergreifend einsetzen lässt. Allerdings benötigt man dafür einen Authentifikations-Server welcher für eine simple Anwendung wie dieses Projekt mehr Overhead verursacht, als dass er wirklich hilfreich wäre. Vor allem da die Konfiguration der Filter in der web.xml dieselbe wie die der Eigenentwicklung ist.

Die Klasse LoginHandler hält eine boolean Variable, die angibt ob der Benutzer angemeldet ist und eine Variable in der die AccountId gespeichert wird. Wichtig ist, dass diese Klasse mit @SessionScoped Annotiert und dadurch, während die Session des Benutzers besteht, gespeichert bleibt.

Der LoginHandler ist für den Login Prozess verantwortlich, er hält Variablen die die aktuelle Session definieren. Bei jedem Aufruf der getAccount() Methode wird der aktuelle Account des Benutzers aus der Datenbank abgerufen und in der Variable account gespeichert. Die Methode login() prüft, ob die Anmeldedaten korrekt sind und ob der Account deaktiviert oder gebannt ist, falls der Account valide ist wird die account Variable erstmals gesetzt und loggedIn auf true geändert, sowie die entsprechende AccountId eingetragen. Die logout() Methode setzt die Werte loggedIn und accountID auf die Default-Werte zurück und setzt die account Variable auf null. Die Methode hasAccess() kontrolliert, ob der Nutzer eingeloggt ist und mindestens dem übergebenen AccountType entspricht. Die Account Typen geben an, welche Rechte der jeweilige Nutzer hat und werden in der folgenden Tabelle dargestellt.

Account Typ	**Rechte**
Nicht angemeldet	Auto Liste ansehen;
User	Nicht angemeldet+ Autos ausleihen und zurückgeben; Rechnungen herunterladen; Profil bearbeiten;
Mitarbeiter	User+ Werkstätten anlegen und bearbeiten; User zur Ausleihe freischalten oder bannen;
Administrator	Mitarbeiter+ Autos, Hersteller, Modelle anlegen und bearbeiten; Autos deaktivieren; Autos in Werkstatt geben;

Tabelle 1: Rechte

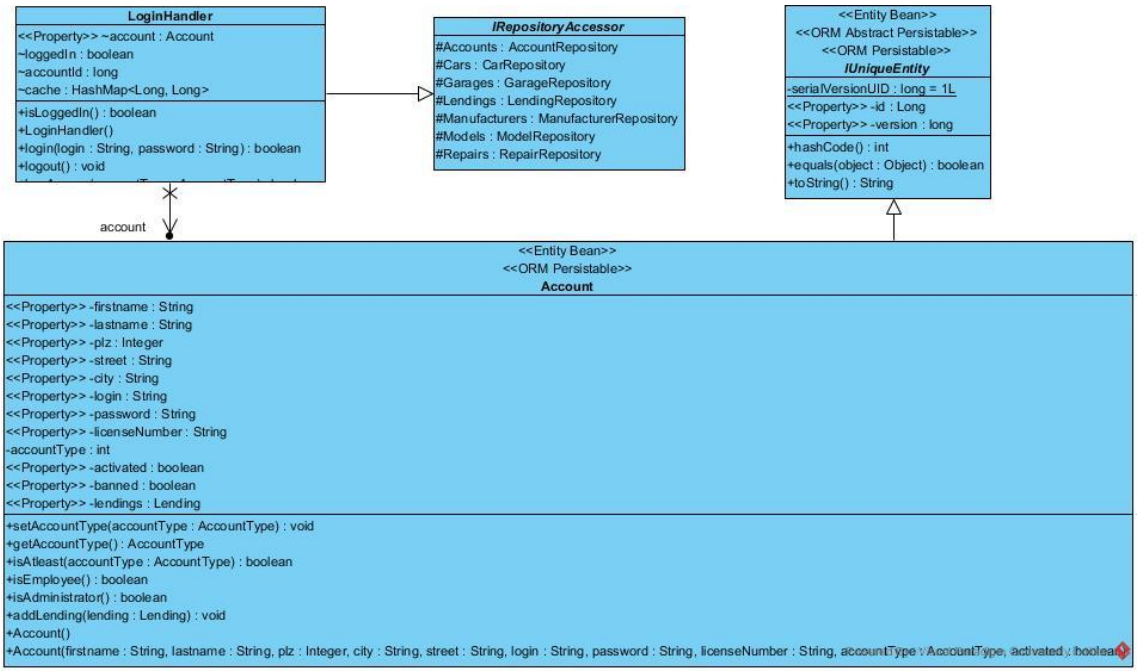

Abbildung 3: Aufbau des Logins

3.5 Filter in Javax Servlet

Die Filter von Javax Servlet sind eine einfache Methode, um den Zugriff auf Seiten zu beschränken. Man kann in der web.xml definieren welche Filter Typen es gibt und durch welche Methoden diese überprüft werden.

```
<filter>
        <filter-name>employee</filter-name>
        <filter-class>supercar.filters.EmployeeFilter</filter-class>
</filter>
```

Snippet 1: Definition eines Filters

Einem Filter wird ein Name zugewiesen und eine Klasse, die eine Variable vom Typ Supplier<Boolean> besitzt in der angegeben wird, ob der Zugriff gefiltert werden soll. Optional kann man ebenfalls angeben, auf welche Seite im Falle von nicht erlaubtem Zugriff weitergeleitet werden soll.

```
<filter-mapping>
        <filter-name>employee</filter-name>
        <url-pattern>/faces/user_management.xhtml</url-pattern>
        <url-pattern>/faces/add_garage.xhtml</url-pattern>
</filter-mapping>
```

Snippet 2: Filter Mapping

Anschließend muss angegeben werden, wann welcher Filter aufgerufen wird. Daher muss ein Filter Mapping angegeben werden, bei dem jedem Filter eine Anzahl an URL-Patterns zugewiesen wird, die in diesem Fall angeben, welche Rolle ein Account mindestens haben muss, um auf die Seite zuzugreifen.

Die Filter Klassen erben von der abstrakten Klasse IFilter, die angibt auf welche Seite weitergeleitet wird, falls eine Seite nicht aufgerufen werden darf. Dies wird durch die Funktion doFilter() implementiert, die falls die Seite gefiltert wird, auf die angegebene Seite weiterleitet oder ansonsten die Seite normal aufruft. Filter lassen sich auch rekursiv nutzen, allerdings wird dieses Feature in diesem Fall nicht benötigt.

3.6 GSON

GSON ist eine von Google entwickelte Library, die es ermöglicht Java Objekte in einen JSON String umzuwandeln und umgekehrt. Der Vorteil von GSON ist, dass keine weiteren Annotationen benötigt werden, um eine Klasse zu Serialisieren bzw. zu Deserialisieren. Die einzige Voraussetzung ist, dass die entsprechende Klasse einen Default Konstruktor besitzt und Serializable implementiert.

In diesem Projekt wird GSON genutzt, um die JSON Strings für die RESTfull-API zur Verfügung zu stellen und die Daten der Postleitzahl-API zu Deserialisieren.

3.7 Validators

Validator wie @NotNull werden genutzt, um den Wert von Attributen zu validieren. Falls ein Wert nicht gültig ist, gibt der Validator false zurück. Zusätzlich besitzen Validators einen message String, in der eine passende Fehlermeldung definiert werden kann. Zur Überprüfung der Nummernschilder und der Postleitzahl wurden eigene Validator definiert.

Der Validator für die Nummernschilder besteht aus den Klassen LicensePlate und LicensePlateValidator, wobei in der LicensePlate Klasse definiert ist, dass der LicensePlateValidator genutzt werden soll und welche Fehlermeldung abgerufen werden kann. Die Validator Klasse übernimmt die eigentliche Verifizierung der Daten durch die Methode isValid(), diese überprüft im Fall eines Nummernschildes, ob der übergebene String dem Format eines Deutschen Nummernschildes entspricht und ob es maximal zehn Zeichen lang ist bzw. nicht Null ist.

3.8 Docker

Das Erstellen eines funktionierenden Docker-Containers war schwieriger als erwartet. Der erste Ansatz war, einen eigenen Ubuntu Container zu erstellen und dort die Installation von Grund auf zu Implementieren. Allerdings entstanden dort Probleme mit dem Speichern bzw. dem Überreichen des asadmin GlassFish Passworts an weitere Befehle. Ebenso wie Versionierungs-Probleme zwischen Java und GlassFish. Zum Lösen dieser Probleme wurde ein offizielles GlassFish Image von Oracle verwendet. Dieses wurde allerdings noch so erweitert, dass es beim Starten die Konfiguration des Connection-Pools und die war-Datei aus dem Projekt importiert. Beim Erstellen des GlassFish Containers wird außerdem die Java SDK Version angepasst, da der vorgefertigte GlassFish Container von Oracle eine nicht länger existierende Java Version installieren will. Die Dockerfile gibt anschließend noch die benötigten Ports frei. Zum Starten des Containers wird Docker-Compose benutzt, um das target-Verzeichnis für den Container freizugeben, dadurch wird erreicht, jedes Mal die aktuellste Version des Projekts im Docker-Container zu starten.

3.9 Rechnungen

Für die Erstellung der Rechnungen wird die Library IText verwendet. IText ist eine Library, die es ermöglicht PDF-Files zu erstellen oder zu manipulieren. Diese Library kann durch Maven eingebunden werden und wird in der Klasse BillCreator im Core Package verwendet.

Um die Formatierung der PDF zu erzeugen, können einfach neue Paragraphen erzeugt werden, die mit Hilfe der Methode document.add() zu der PDF hinzugefügt werden. Diese Paragraphen stehen standardmäßig untereinander, lassen sich aber durch die Methode setSpacingAfter() bzw. setSpacingBefore() verschieben und durch setAlignment(2) rechtsbündig einfügen. Außerdem wird eine Tabelle erstellt, der bei der Erzeugung die Spaltenanzahl übergeben wird. Mit der Methode table.addCell() kann eine neue Zelle hinzugefügt werden, wobei nach der angegebenen Spaltenanzahl automatisch eine neue Zeile begonnen wird. Einer Zelle wird bei der Erzeugung oder im Nachhinein über eine Methode eine Phrase zugewiesen, die aus einem String und optional einer Schriftart besteht. Auf einer Zelle der Tabelle können verschiedene Methoden zur Formatierung aufgerufen werden. Durch die Methode setUseVariableBorders(true) wird ermöglicht, die Umrandung für jede Seite einer Zelle individuell zu setzen. Die Funktion setBorderWidth(0) setzt die Umrandung aller Zellen auf deaktiviert und nur für die obere und untere Seite einiger Zellen, wird die Umrandung durch die Methoden setBorderWidthTop(2) bzw. setBorderWidthBottom(2) wieder aktiviert, um die horizontalen Linien im PDF zu erzeugen. Außerdem wird die Höhe einer Zelle angepasst.

```
Chunk accountOwner = new Chunk("Account Owner: ", bold);
Chunk accountOwnerValue = new Chunk("Supercar Confederation", normalNormal);
Paragraph accountOwnerEntry = new Paragraph(accountOwner);
accountOwnerEntry.setFont(normal);
accountOwnerEntry.add(accountOwnerValue);
```

Snippet 3: Formatierung einer PDF

Um in einem Paragraphen sowohl normalen als auch fetten Text zu verwenden, müssen, wie im Quellcode zu sehen, zuerst zwei Chunks erstellt werden, wobei der zweite Chunk eine Schriftart zugewiesen bekommt, in der die Betonung, also bold oder normal, UNDEFINED angegeben wurde. Danach kann ein neuer Paragraph mit dem ersten Chunk erstellt werden, dem danach die für den zweiten Chunk gewünschte Schriftart zugewiesen wird. Erst dann darf der zweite Chunk hinzugefügt werden, um in diesem Fall die Ausgabe “**Account Owner:** Supercar Confederation” zu erzeugen.

Die Methode createPDF() in der BillCreator Klasse übernimmt neben der Formatierung der PDF auch die Berechnungen und den Output als Bitstream, damit die PDF direkt heruntergeladen werden kann und nicht erst auf dem Server gespeichert werden muss. Um die Daten der Rechnung zu berechnen, wird aus der Differenz zwischen dem ReturnDate und dem RentDate eine Duration in Sekunden erstellt. Aus dieser Duration werden dann die Tage, Stunden und Minuten berechnet und zwischengespeichert. Diese Werte werden dann in der Tabelle verwendet, um die Leihdauer anzuzeigen und den Gesamtpreis der Rechnung zu berechnen. Wichtig ist, dass falls eine Ausleihe keinen ganzen Tag dauert, trotzdem ein Tag berechnet wird, aber wenn das Auto beispielsweise genau bei zwei Tagen zurückgegeben wird, dürfen auch nur zwei Tage berechnet werden.

Um die PDF direkt herunterzuladen, wird die Datei in einem ByteArrayOutputStream gespeichert und am Ende in einen DefaultStreamedContent konvertiert und zurückgegeben. Dieser

Rückgabewert wird dann auf der Bills.xhtml Seite über einen fileDownload Tag in einem Command Button heruntergeladen.

3.10 Design

Die CSS Anpassungen am Design beziehen sich hauptsächlich auf kleinere Änderungen an Komponenten von Bootsfaces und Primesfaces, sowie das Designen eines eigenen Footers und das Erstellen eines globalen Seitenlayouts. Eine dieser kleinen Anpassungen war zum Beispiel das Ändern der Farbe des aus dem Bootstrap stammenden ‘btn-danger’-Buttons an die Farbe des Seitenlogos. Eine weitere wichtige Änderung war das Anpassen des Footers auf position: absolute damit dieser, auf jeder Seite bündig, mit der Seite abschließt. Ebenfalls relevant dafür war das Hinzufügen der position: relative zum html-Tag. Im Großen und Ganzen zielen die Änderungen hauptsächlich darauf ab die Seite etwas übersichtlicher zu gestalten und zumindest ein wenig fürs Auge herzugeben.

3.11 RESTfull API mit Jax-RS

Die Restfull-Api wird unter Verwendung von Jax-RS realisiert. Jede Resource Klasse erbt von IResource, welche grundsätzliche Funktionen zur Ausgabe von Status Codes und JSON zur Verfügung stellt. Außerdem erbt diese wiederum von IRestrictableRepositoryAccessor, was den Zugriff zur Datenbank ermöglicht. Die Verfügbarkeit im Web der Resource Klassen wird per @Path Annotation definiert, die Art und Weise Daten zu verarbeiten per @Produces und @Consumes. In diesem Fall wird komplett mit JSON gearbeitet, weshalb in beiden Fällen der Media-Type “application/json” benutzt wird. Die API ermöglicht den Zugriff auf die grundlegenden Funktionen welche einem User ohne API auf der Webseite zur Verfügung stehen.

3.12 PlzAPI

Die PLZ-API wird genutzt, um aus einer externen Datenbank den Namen einer Stadt zu einer bestimmten Postleitzahl abzufragen. Wenn z.B. der Nutzer bei der Registrierung eine PLZ angibt, wird automatisch auch die entsprechende Stadt eingetragen.

An die Methode getName() wird eine PLZ übergeben, zu der der Name einer Stadt gesucht wird. Anschließend wird mithilfe einer GET-Anfrage an die API nach den Namen der Stadt gefragt. Wird in der Datenbank ein entsprechender Eintrag gefunden, liefert die API einen entsprechenden JSON Text zurück und dieser Text wird mithilfe von GSON in ein JsonObject umgewandelt. Aus dem JsonObject wird dann der Name der Stadt herausgefiltert und entsprechend zurückgegeben.

Wenn die PLZ nicht in der Datenbank gefunden wird, liefert die API einen null String zurück und dementsprechend gibt die Methode einen leeren String zurück.

```
public String getName(int plz) {
   String json = WebHelper.getHTML("http://api.zippopotam.us/de/" + plz);
   if (json == null) {
       return "";
   }
   JsonObject obj = gson.fromJson(json, JsonObject.class);
   if (obj == null) {
       return "";
   }
   JsonArray arr = obj.getAsJsonArray("places");
   String ret = arr.size() > 0 ? arr.get(0).getAsJsonObject()
    .get("place name").getAsString() : null;
       return ret;
   }
```

Snippet 4: Abruf der PLZ API

3.13 Versionen

Name	Ver-sion	Beschreibung
Eclipslink	2.5.2	ORM um Java-Objekte in einer spalten-basierten Datenbank abzulegen
javaee-api	8.0	Sammlung von Programmiermustern und APIs für Java
primfaces	6.2	“Open-Source-Framework für JavaServer Faces mit über 100 Komponenten, Touch-optimiertem Mobilekit, Client-Validierung, Theme-Engine und mehr” [@primfaces]
primeface-ex-tensions	6.2.4	“Open-Source-Komponentenbibliothek für Java Server Faces 2.0“ [@primfaces]
Gson	2.8.5	Open-Soure-Java-Bilbliothek zum Serialisieren und Deserialisie-ren von Java-Objekten zu und von JSON.
bootsfaces	1.2.0	“leistungsfähiges und leichtgewichtiges JSF-Framework basie-rend auf Bootstrap 3 und jQuery UI” [@bootsfaces]
Primeface Theme Bootstrap	1.0.10	Primeface Design angelehnt an Bootstrap
itextpdf	5.5.13	“Einfache PDF-Generierung und Bearbeitung für Java- und .NET-Entwickler” [@itext]

Tabelle 2: Versionen

3.14 Probleme

Ein Projekt in diesen Dimensionen geht im Normalfall mit einigen Problemen einher. Im Folgenden werden ein paar Probleme und die entsprechende Lösung dargestellt.

3.14.1 Login Handler

Der LoginHandler sollte eigentlich in LoginHandler und LoginSession unterteilt werden, wobei der LoginHandler mit RequestScoped und die LoginSession mit SessionScoped annotiert werden sollten. Dadurch entstand das Problem, dass obwohl die LoginSession SessionScoped war immer dasselbe Objekt in den LoginHandler Injiziert wurde. Dadurch wurde auch bei mehreren Zugriffen immer derselbe Benutzer verwendet. Das Problem wurde dadurch behoben, die Variablen aus der ehemaligen LoginSession in den LoginHandler zu integrieren und den Scope auf SessionScoped zu erweitern.

3.14.2 PDF download

Die Erstellung der PDF findet im BillModel statt, da die Methode createPDF() den zur Rechnung passenden Account als Übergabeparameter erwartet, wurde zuerst die ID aus dem Login Handler verwendet. Allerdings kann dadurch nur der Nutzer dem die Rechnung gehört die Datei herunterladen. Falls die Anwendung aber noch erweitert werden soll, sodass auch ein Administrator die PDF eines Kunden herunterladen kann, muss der Account auf eine andere Weise abgefragt werden. Die AccountID ist in der Datenbank als Fremdschlüssel in der Ausleihe Tabelle eingetragen, allerdings kennt die JPA Entity Lending die ID nicht. Um trotzdem an die ID zu kommen, wurde im Lending Repository die Funktion getAccountIdFromLendingId() erstellt, die mit Hilfe einer Native Query, die es ermöglicht eine SQL Query direkt auf der Datenbank auszuführen, die die AccountID aus der Lending Tabelle zurückgibt. Dadurch könnte die Anwendung auch so erweitert werden, dass ein Administrator für einen User die Rechnungen herunterlädt und dann auch in der Rechnung der korrekte Benutzer steht.

```
public Long getAccountIdFromLendingId(Long id) {
    return (Long) em.createNativeQuery(
      "select ACCOUNT_ID from LENDING where ID = " + id)
      .getSingleResult();
    }
```

Snippet 5: Native Query

3.14.3 Responsivität

Die Responsivität stand zwar nicht im Vordergrund der Entwicklung, sollte aber dennoch zumindest grundlegend implementiert werden. Da Primefaces und Bootsfaces schon recht gute Responsivität zur Verfügung stellen, erleichterte dies die Umsetzung. Die DataTables von Bootsfaces stellen auch den Tag responsive=”true” zur Verfügung, dieser scheint allerdings die Auflösung des Gerätes zu verwenden. Dadurch gibt es das Problem, dass die DataTables

nur im Vollbild vollständig Responsiv sind und die Geräte Vorschau der Chrome Entwicklertools nicht richtig funktioniert. Mit anderen Worten, auf einem Smartphone sind die Tabellen Responsiv, auf einem genauso großen Fenster auf einem größeren Bildschirm allerdings nicht.

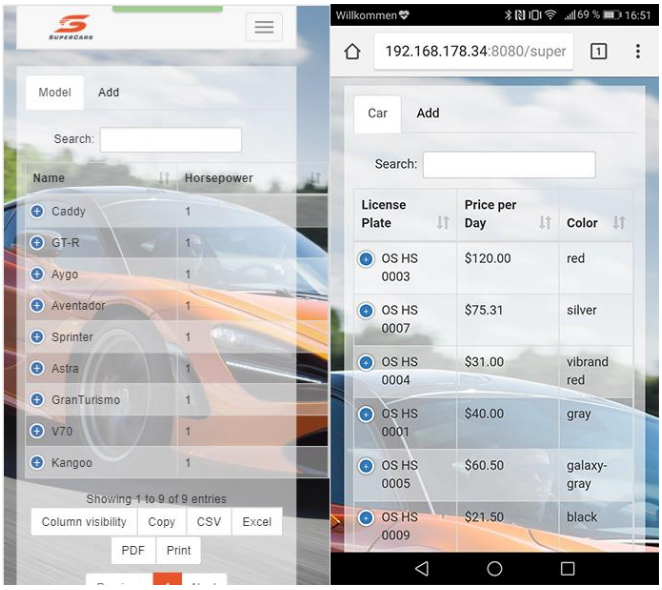

Abbildung 4: Vergleich Entwicklertools und Handy

4 Zusammenfassung und Fazit

Das Projekt wurde erfolgreich durchgeführt und die aufgetretenen Probleme konnten zufriedenstellend gelöst werden. Alle funktionalen und nicht funktionalen Anforderungen konnten Erfolgreich implementiert werden.

4.1 Erfahrungen

Im Laufe des Projektes, haben wir viel im Zusammenhang mit Java EE gelernt, allerdings auch zu allgemeinen Programmierkonzepten und der Umsetzung verschiedener Pattern. Da wir das Projekt vor einiger Zeit bereits mit PHP und SQL umgesetzt haben, können wir die Umsetzung beider Projekte miteinander vergleichen. Die Vorteile einer Umsetzung mit Java sind definitiv die nutzbaren Komponenten wie Beispielsweise die Filter. Außerdem vereinfacht JPA den Umgang mit einer Datenbank sehr, da man sowohl die Business Logic als auch die Datenhaltung objektorientiert betrachten kann. In einem reinen PHP und MySQL Projekt muss man immer zwischen dem Objektorientierten Ansatz und dem relationalen Datenbanksystem umdenken. Ein großer Nachteil der Entwicklung mit Java ist die lange Build-Zeit, dadurch hat es zum Beispiel sehr lange gedauert die PDF zu formatieren, da man für jede Änderung das Projekt neu Builden muss. Hinzu kommt noch, dass wenn man in der Startup Bean neue Objekte mit einer Postleitzahl hinzufügt und jede Postleitzahl über eine API Anfrage validiert, dies die Startzeit noch stärker erhöht.
Alles in allem ist eine Webentwicklung mit Java eher für große Projekte geeignet, es lohnt sich nicht ein komplexes Java Projekt aufzubauen um nur eine kleine Anwendung zu realisieren. Bei großen Projekten hat Java den Vorteil, dass viele Komponenten wiederverwendet werden können und man diese Komponenten auch sehr generisch aufbauen kann, auch wenn PHP mittlerweile in eine ähnliche Richtung geht. Ein Nachteil von Java war, dass wir oft Build Fehler hatten, oder Filter nicht funktionierten was sich aber durch erneutes Builden oder einen Neustart des GlassFish Servers beheben ließ. Dadurch entstand aber das Problem, dass man sich nie sicher sein konnte, ob ein Fehler im Code vorlag, oder ob es wieder ein Build Problem war.

5 Referenzen

Web-Seiten zuletzt am 17.07.2018 abgerufen.

[@tutorialspoint] Tutorialspoint, https://www.tutorialspoint.com/de/jpa/jpa_introduction.htm

[@javaee] Java Platform, Enterprise Edition (Java EE) 8 The Java EE Tutorial, https://javaee.github.io/tutorial/cdi-basic002.html#GIWHL

[@oracle] JavaServer Faces Technology Overview, http://www.oracle.com/technetwork/java/javaee/overview-140548.html

[@Docker] Docker, https://www.docker.com/

[@primfaces] Primefaces, https://www.primefaces.org/

[@bootsfaces] Bootsfaces, https://www.bootsfaces.net/

[@itext] iText, https://itextpdf.com/

Eidesstattliche Erklärung

Hiermit erkläre ich/ Hiermit erklären wir an Eides statt, dass ich/ wir die vorliegende Arbeit selbständig und ohne fremde Hilfe angefertigt habe/ haben. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche einzeln kenntlich gemacht. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

....
Ort, Datum Unterschrift

....
Ort, Datum Unterschrift

....
Ort, Datum Unterschrift

Urheberrechtliche Einwilligungserklärung

Hiermit erkläre ich/ Hiermit erklären wir, dass ich/wir damit einverstanden bin/sind, dass meine/ unsere Arbeit zum Zwecke des Plagiatsschutzes bei der Fa. Ephorus BV bis zu 5 Jahren in einer Datenbank für die Hochschule Osnabrück archiviert werden kann. Diese Einwilligung kann jederzeit widerrufen werden.

....
Ort, Datum Unterschrift

....
Ort, Datum Unterschrift

....
Ort, Datum Unterschrift