Internationaler Studiengang Medieninformatik

Datenbankbasierte Web-Anwendungen

(SoSe 2015)

Autovermietung

Gruppenmitglied	Matrikelnummer
Fabian Redecker	375750
Marco Geils	375736
Mike Hüsing	344089
Oliver Bammann	360330
Konrad Mumm	296179

Inhaltsverzeichnis

1. Spezifikation

In dem Modul „Datenbankbasierte Web-Anwendungen“ wurde die Anwendung „Autovermietung“ entwickelt. Es handelt sich hierbei um eine Autovermietung, dessen Anforderungen über einen Browser abgerufen werden.

Im folgenden Kapitel wird zunächst erläutert, was die Webanwendung ausmacht und welche Anforderungen das Projekt erfüllen soll. Im weiteren Verlauf dieser Dokumentation wird näher auf die Entwicklung und weitere Aspekte dieser Projektarbeit eingegangen.

Mit dieser Webanwendung lassen sich Daten einer potentiellen Autovermietungsfirma einsehen, bearbeiten oder hinzufügen. Dabei werden die Daten in einer Datenbank gelesen bzw. geschrieben. Bevor über die Browseranwendung auf die Daten zugegriffen werden kann, erfolgt zunächst ein Login oder eine Registrierung mit E-Mail und Passwort. Weitere Anforderungen werden in Punkt 1.1 beschrieben.

Die Webanwendung bietet den Kunden und Mitarbeitern dieser Autovermietung den Mehrwert, dass sie sich im System registrieren können und verschiedene Aktionen ausführen können, wie z.B. das Einsehen von Fahrzeugen, ohne, dass dabei Kenntnisse über ein Datenbanksystem oder die jeweilige Software vorhanden sein müssen. All dies ist über die Webanwendung möglich, welche sehr intuitiv zu bedienen ist.

1.1 Anforderungen

Zu Anfang der Projektarbeit wurden Anforderungen an die Anwendung festgelegt, die jedoch im Laufe des Projektes etwas angepasst wurden. Im folgenden Abschnitt werden die vorgesehenen Anforderungen aufgezeigt. Anschließend werden die tatsächlich umgesetzten Anforderungen mit Hilfe von Use-Case Diagrammen dargestellt.

Vorgesehene Anforderungen:

1) „Als Mitarbeiter möchte ich Fahrzeuge in die Datenbank eintragen und löschen können, um den Datenbankbestand der Fahrzeuge aktualisieren zu können."

2) „Als Mitarbeiter möchte ich für die Autovermietung die Kundenliste einsehen und bearbeiten können, um neue Kunden eintragen zu können und Informationen über diese zu erhalten."

3) „Als Kunde möchte ich Fahrzeuge buchen können, um mobil zu sein."

4) „Als Kunde möchte ich abrufen können, welche Fahrzeuge nicht vermietet sind, um zu prüfen, ob das gewünschte Fahrzeug für meinen Termin verfügbar ist."

5) „Als Kunde möchte ich sehen, welche Fahrzeuge im Fuhrpark vorhanden sind um zu sehen ob die Autovermietung mein gewünschtes Fahrzeug besitzt."

Umgesetzte Anforderungen:

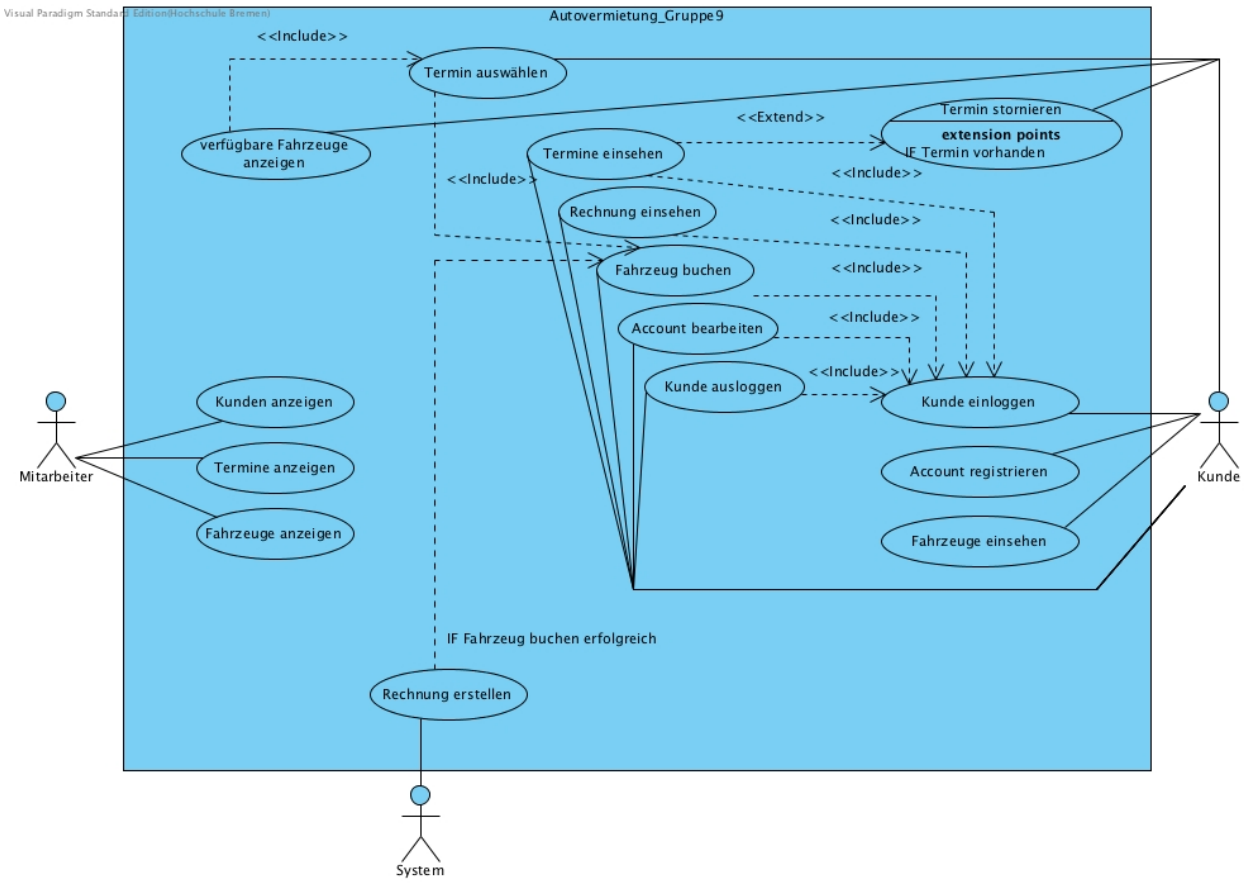

Abbildung 1 Use-Case Diagramm umgesetzte Anforderungen

Die im Folgenden beschriebenen umgesetzten Anforderungen können im Use-Case Diagramm (Abbildung 1) betrachtet werden.

Anders als die vorgesehenen Anforderungen eins und zwei sind von den Mitarbeiterfunktionen lediglich die Use-Cases „Kunden anzeigen“, „Termine anzeigen“ und „Fahrzeuge anzeigen“ in der Anwendung umgesetzt. Dazu kann der eingeloggte Kunde in eine Mitarbeitersicht wechseln, um auf diese Funktionen zuzugreifen. Alle CRUD-Funktionen bis auf READ sind dabei entfallen. Zusätzlich können sich Kunden selbst registrieren.

Die Anforderung fünf ist über den Use-Case „Fahrzeuge einsehen“ realisiert. Dabei können auch nicht eingeloggte Kunden darauf zugreifen.

Um die Anforderungen drei und vier zu erfüllen kann der Kunde nach erfolgtem Login in der Anwendung Fahrzeuge buchen. Dazu wählt er zunächst den Zeitraum des Termins aus, woraufhin das System die für diesen Zeitraum verfügbaren Fahrzeuge anzeigt. Danach kann der Kunde eines dieser Fahrzeuge auswählen und mit diesem den Termin buchen. Das System erstellt dann automatisch eine Rechnung für diesen Termin und weist sie dem Kunden zu.

Um die Usability und die Kundenzufriedenheit zu erhöhen, wurden zusätzlich die Use-Cases „Account bearbeiten“, „Kunde ausloggen“, „Rechnung einsehen“, „Termine einsehen“ und „Termin stornieren“ implementiert.

2. Architektur

In diesem Kapitel wird die Architektur des Projektes erläutert und der Projektablauf geschildert. Daraufhin folgt die Erläuterung der Persistenzschicht und als letzter Punkt wird auf die Präsentationsschicht eingegangen.

2.1 Gesamtarchitektur

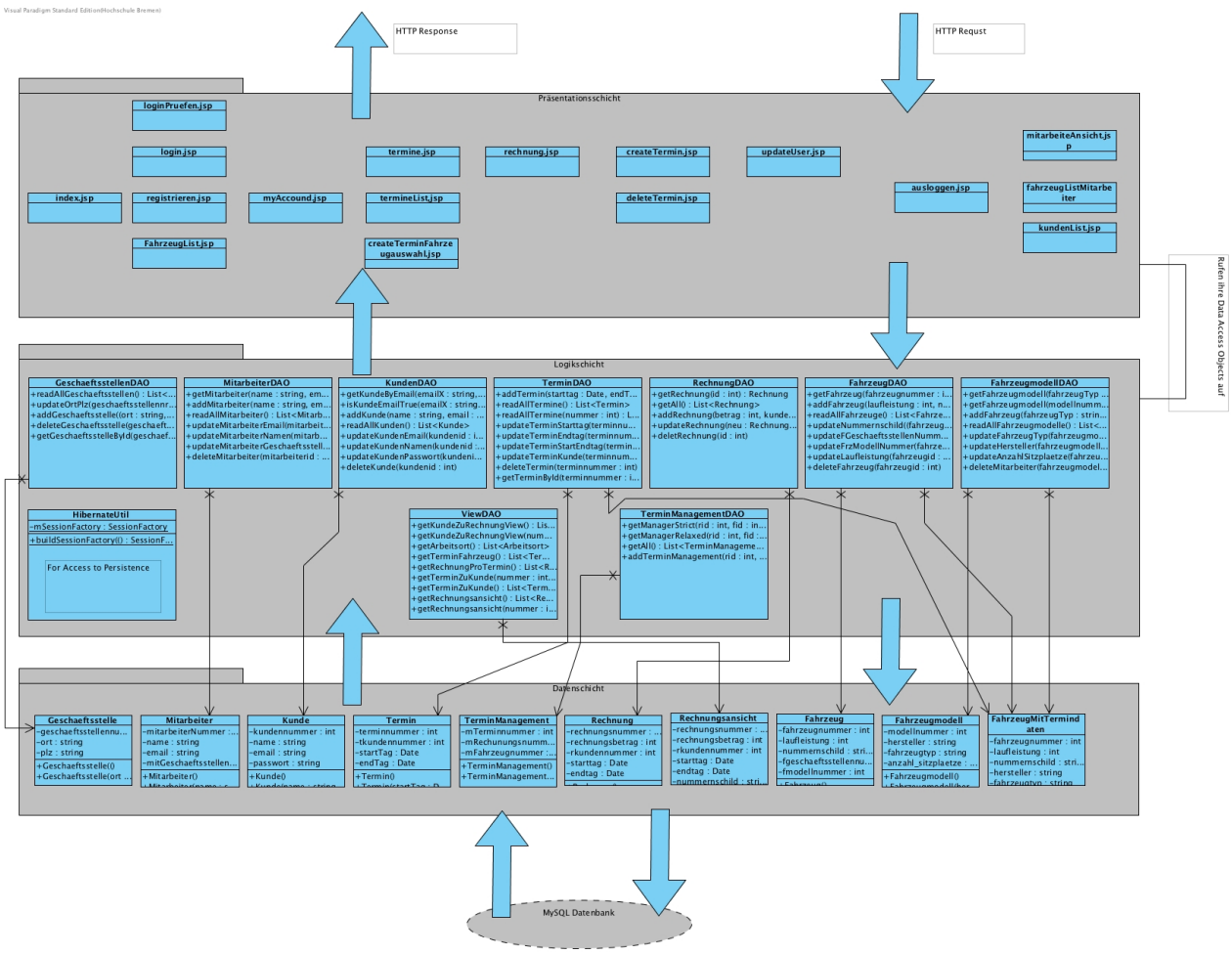

Abbildung 2 Klassendiagramm

Das Projekt ist in einer Drei-Schichten-Architektur (Abbildung 2) organisiert. Die Schichten sind Datenschicht, Logikschicht und Präsentationsschicht.

- Datenschicht

  In der Datenschicht finden mittels Hibernate(OR-Mapping) Zugriffe auf die MySQL Datenbank statt.
- Logikschicht

  Hier befinden sich die DAOs(Data Access Object). Diese generieren mittels query-statements die CRUD(Create Read Update Delete) Funktionen.
- Präsentationsschicht

  Hier sind die JSPs (Java Server Pages) zu finden. Die JSPs beinhalten HTML Code gepaart mit Java Code. In diesem Code werden die DAOs aus der Logikschicht abgerufen.

Die Präsentationsschicht nimmt HTTP-Requests entgegen. Dieser Request wird dann von der Logikschicht verarbeitet und Daten aus der Datenschicht abgerufen. Die Java Server Pages, die sich in der Präsentationschicht befinden, generieren ein Servlet das HTML ausliefert. So gibt die Präsentationschicht HTTP-Responses zurück.

2.2 Programmabläufe

Als Nächstes werden die Datenbankzugriffe exemplarisch beschrieben. Typische Abläufe sind die CRUD (Create, Read, Update & Delete) Operationen. Der Zugriff auf die Datenbank findet mittels Hibernate (OR- Mapping) statt.

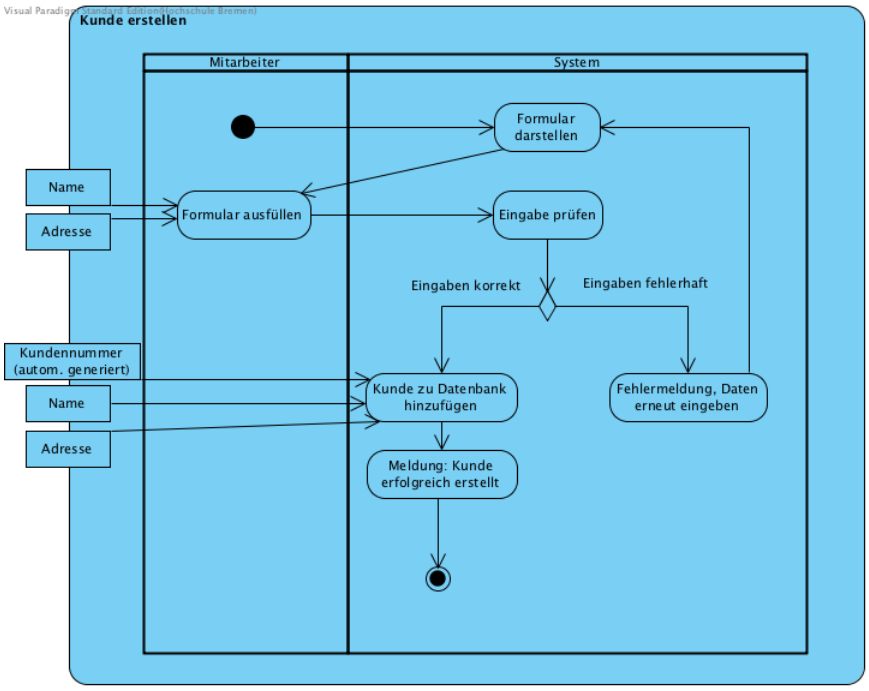

Abbildung 3 Aktivitätsdiagramm "Kunde erstellen"

Zum Erstellen (Create) einer neuen Column in einer Entität wird in den DAOs die add-Methode aufgerufen. Die Parameter entsprechen dabei den Eingabefeldern des Formulars der HTML Seite. Um einen neuen Kunden zu erstellen (Abbildung 3) muss der User seinen Namen, Email-Adresse und ein Passwort angeben. Der Primärschlüssel, die Kundennummer, wird automatisch generiert. Die Richtigkeit der Eingabe wird innerhalb eines HTML <form> Tags geprüft. Wenn alle Eingaben korrekt sind wird der User als Kunde in der entsprechenden Entität eingetragen und darüber informiert, dass er nun Kunde ist. Sollten Eingaben nicht korrekt sein, wird der User aufgefordert falsche Eingaben zu korrigieren.

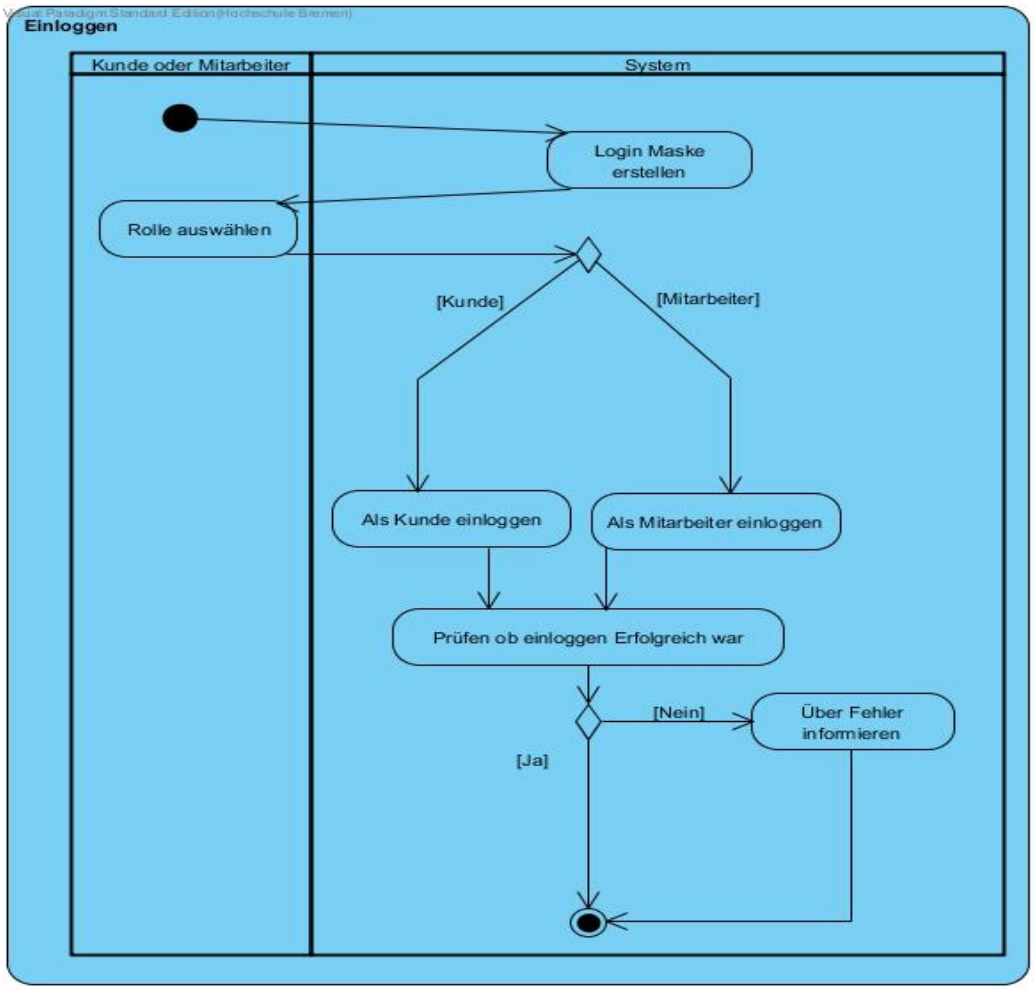

Abbildung 4 Aktivitätsdiagramm (Einloggen)

Eine Login Funktion (Abbildung 4) war zu Anfang des Projektes nicht geplant. Aufgrund sich veränderter Anforderungen wurde diese Funktion in der finalen Abgabe doch implementiert. Diese Funktion steht für den Kunden zu Verfügung. In der login.jsp wird der Kunde aufgefordert seine Email-Adresse und sein Passwort einzugeben. Nach einem Submit wird der Kunde auf die loginPruefen.jsp weitergeleitet. Wird die Email-Adresse und das Passwort in der Datenbank gefunden, wird ein Bean erstellt und die Kundendaten dort über die Dauer der Session gespeichert (Abbildung 5).

```
<jsp:useBean id="loggedKunde" class="Tables.Kunde" scope="session"/>
<jsp:setProperty name="loggedKunde" property="id" value="<%=kunde.getId()%>"/>
<jsp:setProperty name="loggedKunde" property="name" value="<%=kunde.getName()%>"/>
<jsp:setProperty name="loggedKunde" property="email" value="<%=kunde.getEmail()%>"/>
<jsp:setProperty name="loggedKunde" property="passwort" value="<%=kunde.getPasswort()%>"/>
<jsp:forward page="myAccount.jsp"/>
```

Abbildung 5 Programmcode Ausschnitt "Beanerstellung)

Der Kunde wird auf die myAccount.jsp weitergeleitet. In dieser JSP stehen dem Kunden alle Implementierungen der Use-Cases zur Verfügung.

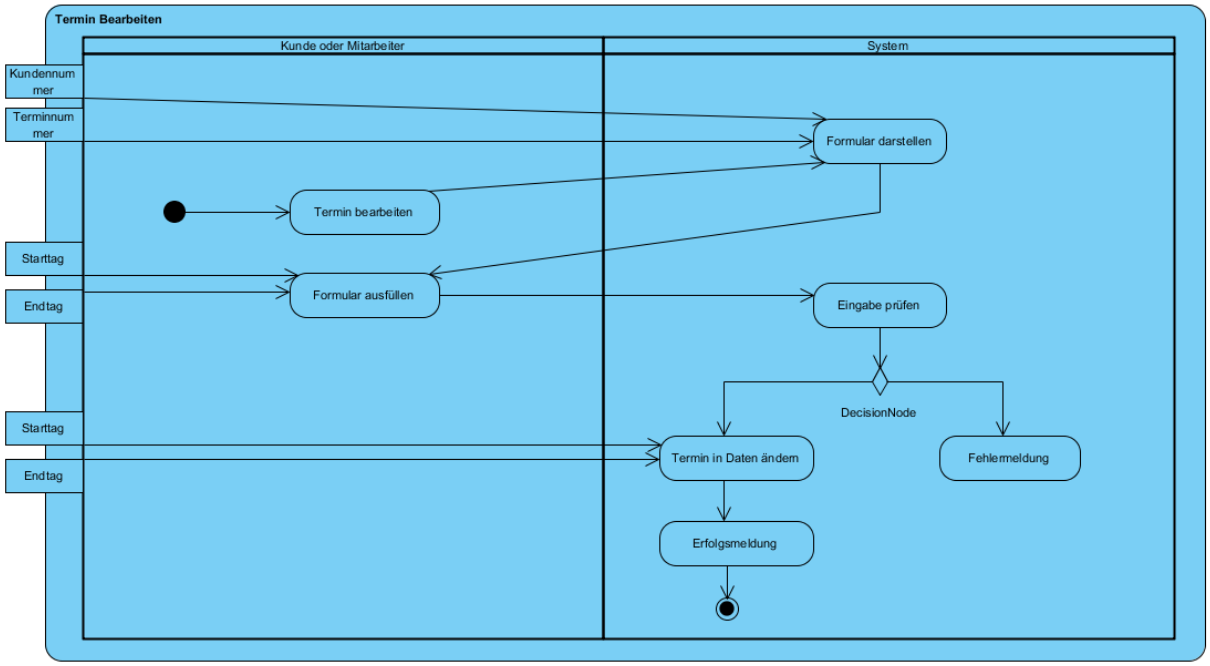

Abbildung 6 Aktivitätsdiagramm "Termin bearbeiten"

Zum Verändern einer Column wird die update-Methode (Abbildung 7) der jeweiligen DAO Klasse aufgerufen. Sie enthält als Parameter immer den Primärschlüssel der Entität und als zweiten Paramater den Wert der verändert werden soll. In der Präsentationschicht kann über die Bean der Kunde abgerufen werden. Mit den Kundeninformationen ist es möglich über Foreign-Keys an weitere Informationen z.B. eine Terminnummer zu kommen. Nun muss der Kunde in einem HTML Formular die Werte eingeben, die verändert werden sollen. Danach ist der Aufruf einer update-Funktion möglich (Abbildung 6).

```
public void updateTerminStartEndtag(int terminnummer, Date starttag, Date endtag){
    Session session = HibernateUtil.getSessionFactory().openSession();
    Transaction tx = null;
    try{
        tx = session.beginTransaction();
        Termin termin = (Termin)session.get(Termin.class, terminnummer);
        termin.setStarttag(starttag);
        termin.setEndtag(endtag);
        session.update(termin);
        tx.commit();
    }catch (HibernateException e) {
        if (tx!=null) tx.rollback();
        e.printStackTrace();
    }finally {
        session.close();
    }
}
```

Abbildung 7 Programmcode Ausschnitt "updateTerminStartEndtag"-Methode

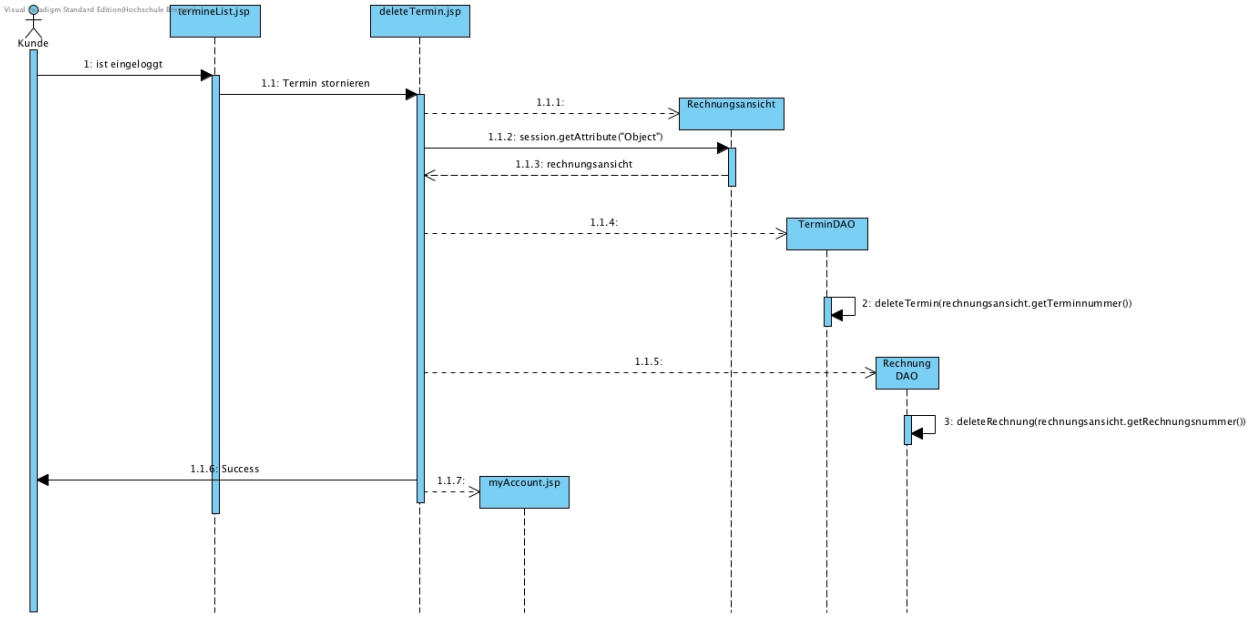

Abbildung 8 Sequenzdiagramm "Termin löschen"

Um einen Termin zu löschen (Abbildung 8) muss sich der Kunde in der terminList.jsp befinden. Es wird die Rechnungsansicht (View) benötigt. Mit den Informationen der Rechnungsansicht kann über das TerminDAO und das RechnungsDAO ein Termin gelöscht werden. Die ausgeführte Methode ist in beiden Klassen die delete-Methode (Abbildung 9). Die TerminManagement-Table wird automatisch angepasst(Mapping-Tabelle). Nachdem die Methoden ausgeführt wurden wird der Kunde darüber informiert, dass der ausgewählte Termin storniert wurde. Im Anschluss wird er auf die myAccount.jsp weitergeleitet.

```
public void deleteTermin(int terminnummer){
    Session session = HibernateUtil.getSessionFactory().openSession();
    Transaction tx = null;
    try{
        tx = session.beginTransaction();
        Termin termin = (Termin)session.get(Termin.class, terminnummer);
        session.delete(termin);
        tx.commit();
    }catch (HibernateException e) {
        if (tx!=null) tx.rollback();
        e.printStackTrace();
    }finally {
        session.close();
    }
}
```

Abbildung 9 Programmcode Ausschnitt "deleteTermin"-Methode

2.3 Persistenz

In der Anwendung wird eine MySQL Datenbank benutzt, die auf einem Server der Hochschule Bremen gehostet wird.

Für das Projekt wurde ein Account zur Verfügung gestellt. Mit dem Benutzer „dbweb_user_09“ können Zugriffe auf die Datenbank „dbweb_sose15_09“ durchgeführt werden.

Als Zugriffstool wird MySQL Workbench 6.3 verwendet, das offizielle Tool von Oracle.

In der Endanwendung werden keine Datenbankzugriffe mehr über das Tool erfolgen, da alle benötigten Funktionen in der Webanwendung realisiert sind und der Zugriff von dort über JDBC und Hibernate (siehe 2.3.3) umgesetzt wird.

2.3.1 Konzeption

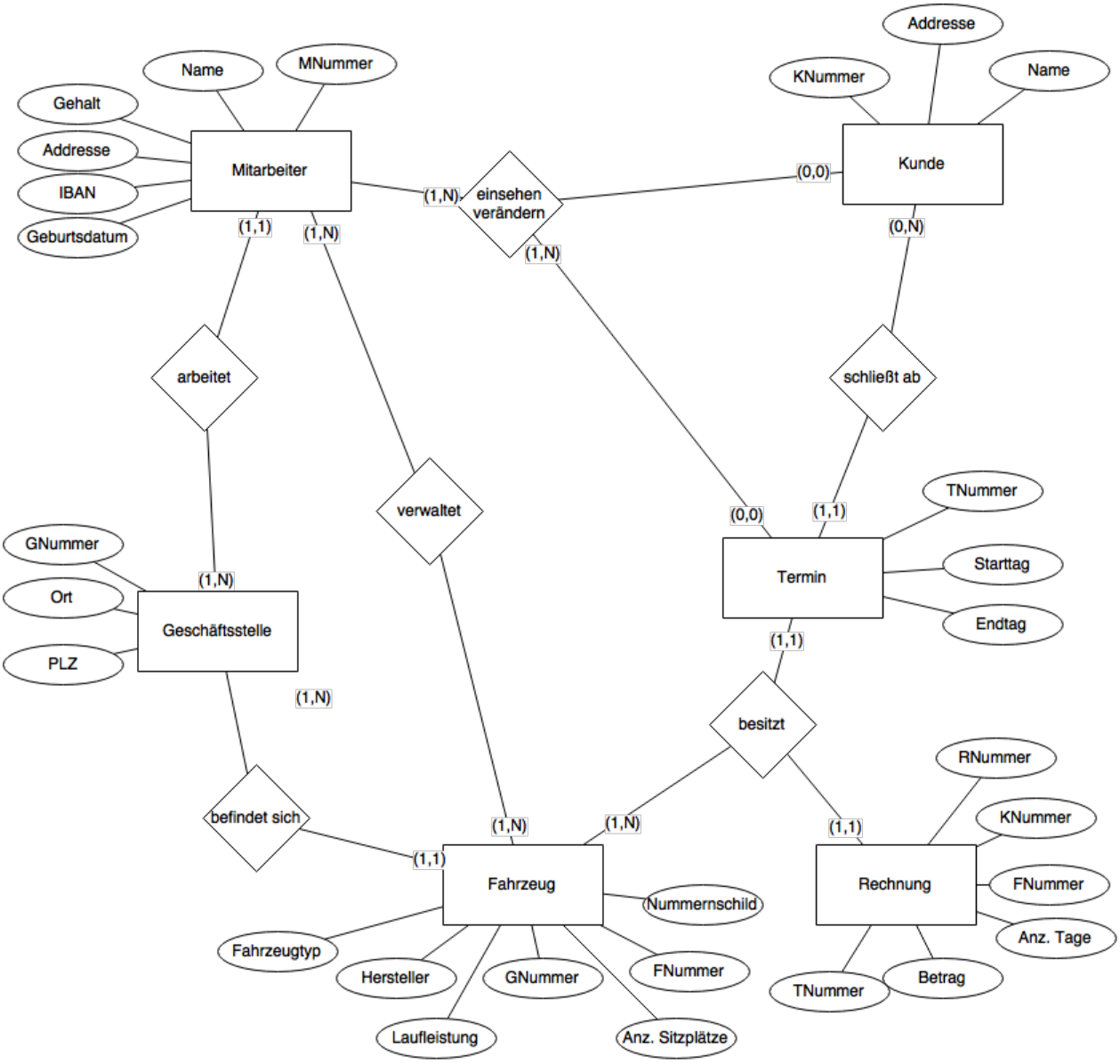

Abbildung 10 ER-Diagramm

Das Entity Relationship Diagramm (Abbildung 10) war der erste Anhaltspunkt über die Struktur der Datenbank. Es spiegelt jedoch nicht den aktuellen Stand wieder, da es für die spätere Entwicklung nicht mehr benötigt wurde.

Mitarbeiter:

- Arbeitet in genau einer Geschäftsstelle und verwaltet dort die Fahrzeuge.
- Kann die Kundeninformationen und Termine einsehen und verändern und somit Wünsche der Kunden (z.B. Termin ändern) umsetzen.

Geschäftsstelle:

- Besitzt einen oder mehrere Mitarbeiter und ist der Standort der Fahrzeuge.

Fahrzeug, Termin und Rechnung:

- Stellen zusammen dar, wie ein Kunde sich ein Fahrzeug mieten kann.
- Ein Fahrzeug kann zu verschiedenen Terminen gemietet werden

Kunde:

- Kann keine oder beliebig viele Termine abschließen und seine Daten können auf Wunsch von einem Mitarbeiter verändert werden.

Termin:

- Zu jedem Termin gehört genau eine Rechnung, ein Kunde und ein Fahrzeug.
- Termine können auf Kundenwunsch von Mitarbeitern verändert werden.

2.3.2 Umsetzung

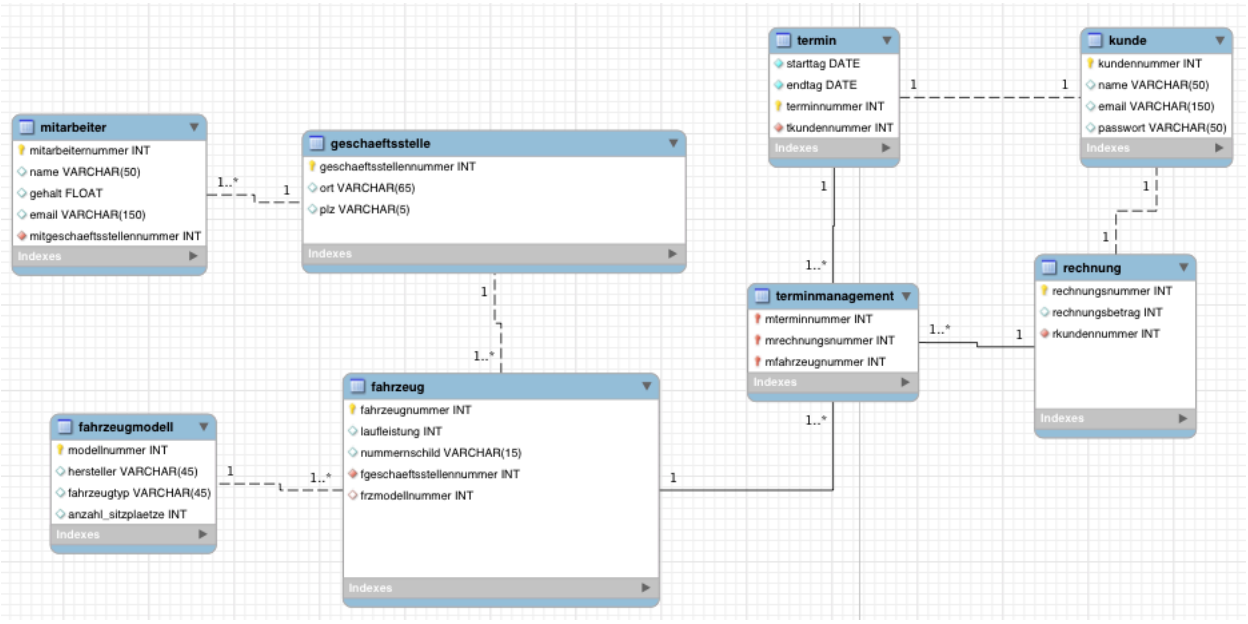

Abbildung 11 Relationales Modell

Das Relationale Modell (Abbildung 11) unterscheidet sich vom Entity Relationship Modell, da nach Normalisierungen und datenbankspezifischen Eigenheiten einige Daten geändert werden mussten. So ist z.B. das Terminmanagement als Mapping-Tabelle entstanden.

Mitarbeiter:

- Mitarbeiter arbeitet an einer Geschäftsstelle und kann immer noch administrativ Daten verwalten, allerdings ist es nicht nötig dies hier darzustellen.

Geschäftsstelle:

- Eine Geschäftsstelle hat ein oder mehr Mitarbeiter und ebenso ein oder mehr Fahrzeuge, die real dort arbeiten bzw stehen würden.

Fahrzeug:

- Jedes Fahrzeug ist einem Fahrzeugmodell zugeordnet, welches wichtige Dinge repräsentiert.
- Ein Fahrzeug kann beliebig vielen Terminmanagements zugeordnet werden, welches ausgeliehene Fahrzeuge darstellt.

Fahrzeugmodell:

- Die Fahrzeugmodelltabelle ist durch Normalisierung entstanden, da sich Daten oft wiederholen würde z.B. haben die meisten PKW 5 Sitze
- Ebenfalls kann man so einfacher mehrere Fahrzeuge vom gleichen Typ verwalten, da sie sich nur in Nummernschild und Laufleistung unterscheiden würden.

Terminmanagement:

- Dies ist die Mapping-Tabelle zwischen Terminen, Fahrzeugen und Rechnungen, ähnlich wie sie im ER Modell schon abgebildet wurde.

Kunde:

- Kunden können Termine buchen, welche ihm dann zugeordnet werden. Zusätzlich wird eine Rechnung erstellt und das ganze im Terminmanagement mit dem Fahrzeug verknüpft.

Rechnung:

- Die Rechnung wird erstellt, nachdem ein Termin gebucht wurde und kann dann direkt bezahlt werden. Der Starttag und Endtag wurden durch Normalisierung in Termin ausgelagert.

Anwendungsspezifische Constraints:

- In Termin darf Startdatum nicht vor dem Enddatum liegen.
- Passwörter müssen beim Registrieren und Ändern wiederholt werden und übereinstimmen.
- E-Mailadressen müssen ein @-Zeichen enthalten.

2.3.3 Zugriff

Für die Verbindung zur Datenbank wird die Java API JDBC (Java Database Connectivity) verwendet. Diese Datenbankschnittstelle ist im JDK seit Version 1.1 enthalten und baut Verbindungen zu einer Datenbank auf und verwaltet diese. Zudem werden SQL-Anfragen an die Datenbank weitergeleitet und die Ergebnisse in ein für Java nutzbares Format umgewandelt.

Da JDBC das relationale Datenmodell verwendet und somit nicht objektorientiert ist, wird zusätzlich ein OR-Mapper Framework (Object-Relational Mapper), in diesem Fall Hibernate, genutzt. Da Hibernate JPA (Java Persistence API) implementiert, kann es problemlos mit Java genutzt werden.

Die Datenbankschnittstelle JDBC wird genutzt, da dies die einzige Datenbankschnittstelle ist die in Java implementiert ist. Als OR-Mapper Framework wurde Hibernate gewählt. Dies ist am besten für dieses Projekt geeignet. Es besitzt eine große Community, d.h. es gibt viele Beispiele an denen man sich orientieren und lernen kann. So findet man sich selbst beim ersten Benutzen dieser Technologie sehr gut zurecht.

Zudem war das Arbeiten mit Annotations ebenfalls ein positiver Aspekt, der bei Hibernate aufgefallen ist. In vielen anderen Frameworks lässt sich nur mit xml-Dateien arbeiten, was für den Zweck dieser Anwendung zu aufwendig war.

2.4 Präsentation

Um die Daten der Datenbank im Browser darzustellen und neue Daten durch Nutzer speichern zu können, wird die Präsentationsschicht benötigt. Diese ist mit JSPs (Java Server Pages) nach Model 1 realisiert. Dabei enthalten die JSPs nicht die komplette Logik der Anwendung, sondern rufen die DAOs (Data Access Object) der Logikschicht auf, so wie es in der Drei-Schichten-Architektur (2. Architektur) dargestellt ist.

Ein großer Vorteil der Entwicklung nach Model 1 ist, dass es nicht so komplex ist wie die Model 2 Architektur, da sich die gesamte Präsentationsschicht bei Model 1 in den JSPs befindet und nicht zusätzliche Servlets programmiert werden müssen. Es ist somit für kleinere Anwendungen mit relativ geringem Umfang und einfacher Logik, so wie bei dieser Autovermietung, ideal geeignet.

Dabei wird allerdings auch ein großer Nachteil der Model 1 Architektur ersichtlich. Wenn die Anwendungen umfangreicher und komplexer sind, werden die JSP Dateien schnell sehr voll und unübersichtlich, da der Umfang des Logikanteils deutlich zunehmen kann.

Somit ist JSP nach Model 1 eher für kleinere Anwendungen geeignet, wohingegen größere und komplexere Projekte nach Model 2 oder anderen Mustern programmiert werden sollten.

3. Installation

Im folgenden Kapitel wird beschrieben, welche Programmierwerkzeuge benutzt wurden und wie die Anwendung aufgesetzt und daraufhin auch konfiguriert werden kann.

3.1 Programmierwerkzeuge

3.1.1 Entwicklungsumgebung: *Intellij*

Der Hauptgrund, dass IntelliJ verwendet wurde, war, dass eine Person dieser Gruppe Erfahrung mit dieser Entwicklungsumgebung hatte und in vorherigen Projekten die Einfachheit des Einbindens und Benutzens von Git kennengelernt hatte. Ein weiterer Vorteil von IntelliJ ist die, wie der Name schon verrät, Eigenintelligenz des Programmes. Autovorschläge und Code-Generierung sind in der Entwicklungsumgebung sehr gut umgesetzt. Negativ aufgefallen sind allerdings das Fehler-Highlighting und das teilweise unübersichtliche Design.

3.1.2 Versionskontrollsystem: *Git*

Beim Versionskontrollsystem wurde sich für Git entschieden, da dort schon angesammelte Erfahrungen vorhanden sind. Bereits früh im Studium wurde den Studenten nahegelegt Git zu verwenden, da es sehr gut im Mergen ist und kein eigener Server benötigt wird, wie z.B. für SVN. Ein Nachteil von Git ist, dass es sehr kompliziert ist zu lernen und auch nur wenig gute GUI's hat.

3.1.3 Buildautomatisierungswerkzeug: *Gradle*

Nach ausgiebiger Recherche über die Buildtools wurde sich für Gradle entschieden, da es die Vorteile von Maven und Ant bietet und dabei sehr einfach zu installieren ist. Diese Vorteile entstehen auch teilweise dadurch, dass Gradle das modernste der drei Systeme ist und von den anderen Tools lernen konnte.

3.1.4 Komponententest-Framework:

Da im Modul viele neue Techniken erlernt werden mussten, hätte die Hereinnahme eines Komponententest-Frameworks den Rahmen des Moduls gesprengt. Dennoch ist anzumerken, dass „Test-First" ein wichtiger Aspekt der agilen Programmierung ist.

3.2 Installation der Entwicklungsumgebung

Zum Einrichten der Entwicklungsumgebung gehört das Installieren von Gradle, IntelliJ und vor allem einer aktuellen Java Runtime dazu. Um dann das Projekt einzubinden muss Git installiert und eingerichtet werden. Das Projekt kann nun in IntelliJ eingebunden werden.

3.3 Installation der Anwendung

Für die lokale Anwendung gibt es verschiedene Möglichkeiten, wie man das ganze auf einem Tomcat-Server starten kann.

In dieser Projektarbeit wird ein Tomcat-Server für die Darstellung auf dem Webbrowser genutzt.

Zuerst muss in der Entwicklungsumgebung Tomcat richtig konfiguriert werden. In den Optionen kann man unter Build -> Execution und Deployment bei dem Punkt Application Servers den Tomcat Server hinzufügen.

Dieser sollte bereits automatisch erkannt werden, sofern es richtig Installiert wurde.

Nun muss der Tomcat-Server als Run Configuration eingestellt werden.

Dazu wird unter Run Configuration -> Defaults der Tomcat Server -> Local gewählt und dort die neue Konfiguration erstellt.

Hierbei muss bei dem Punkt Deployment das Gradle Artefakt hinzugefügt werden, z.B. „Gradle: hsb Graddle test: Graddle test -.0: SNAPSHOT.war“.

Nun kann das gesamte Projekt auf dem Tomcat-Server mit einem Klick auf Run gestartet werden. Es öffnet sich automatisch die Localhost-Seite im Browser.

3.4 Konfiguration der Datenbank

Für die Konfiguration der Datenbank die der Anwendung zugrunde liegt müssen folgende Schritte befolgt werden:

- MySQL Workbench muss installiert werden.
- In der MySQL Workbench muss eine neue SQL Connection und Server erstellt werden.

Um die Verbindung zum lokalen Datenbankserver zu gewährleisten muss ein lokaler MySQL Server auf dem Rechner laufen. Bei Windows kann es wichtig sein die "mysqld.exe" mit Administratorrechten zu starten. Diese Datei findet man unter der MySQL Ordnerstruktur in "MySQL\MySQL Server 5.6\bin".

- In der Verbindung kann dann eine Query geöffnet werden
- In dieser Query muss lediglich der gesamte Inhalt der Textdatei "SQLStatementsZusammengefasst.txt" kopiert und ausgeführt werden. Diese Query setzt dann die gesamte Datenbank auf und füllt diese mit Beispieldatensätzen.

Nachdem diese Schritte erfolgreich durchgeführt werden kann die Anwendung installiert und gestartet werden.