****

**Internationaler Studiengang Medieninformatik**

**Datenbankbasierte Web-Anwendungen (SoSe 2015)**

**Autovermietung**

|  |  |
| --- | --- |
| Fabian Redecker | 375750 |
| Marco Geils | 375736 |
| Mike Hüsing | 344089 |
| Oliver Bammann | 360330 |
| Konrad Mumm | 296179 |

Inhaltsverzeichnis

1. Spezifikation 3

1.1 Anforderungen 3

2. Architektur 4

2.1 Gesamtarchitektur 4

2.2 Programmabläufe 5

2.3 Persistenz 9

2.3.1 Konzeption 9

2.3.2 Umsetzung 9

2.3.3 Zugriff 9

2.4 Präsentation 9

3. Installation 9

3.1 Programmierwerkzeuge 9

3.1.1 Entwicklungsumgebung: *Intellij* 9

3.1.2 Versionskontrollsystem: *Git* 10

3.1.3 Buildautomatisierungswerkzeug: *Gradle* 10

3.1.4 Komponententest-Framework: 10

3.2 Installation der Entwicklungsumgebung 10

3.3 Installation der Anwendung 10

3.4 Konfiguration der Datenbank 10

# 1. Spezifikation

In diesem Kapitel wird erläutert, was die Webanwendung ausmacht und wie Anforderungen an das Projekt sind.

* *Was leistet die Webanwendung?*

Mit der Webanwendung „Autovermietung“ lassen sich Daten einer potentiellen Autovermietungsfirma einsehen, bearbeiten oder hinzufügen. Dabei werden die Daten in einer Datenbank gelesen bzw. geschrieben. Bevor über einen Browser auf die Daten zugegriffen werden kann, erfolgt zunächst ein Login oder eine Registrierung mit Name/E-Mail und Passwort.

* *Welchen Mehrwert bietet die Webanwendung für dessen Nutzer?*

Die Webanwendung bietet den Mehrwert, dass nun Kunden und Mitarbeiter angelegt werden können und diese verschiedene Aktionen betreiben können wie zum Beispiel das Einsehen von Fahrzeugen, ohne, dass dabei Kenntnisse über ein Datenbanksystem oder die jeweilige Software vorhanden sein muss. All dies ist sehr intuitiv über die Webanwendung möglich.

## 1.1 Anforderungen

*In diesem Unterkapitel sollen die vorgesehenen und die tatsächlich umgesetzten Anforderungen beschrieben und mit Hilfe von Use-Case- und Aktivitätsdiagrammen konkretisiert werden.*

**Vorgesehene Anforderungen:**

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

„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.“

„Als Kunde möchte ich Fahrzeuge buchen können, um mobil zu sein.“

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

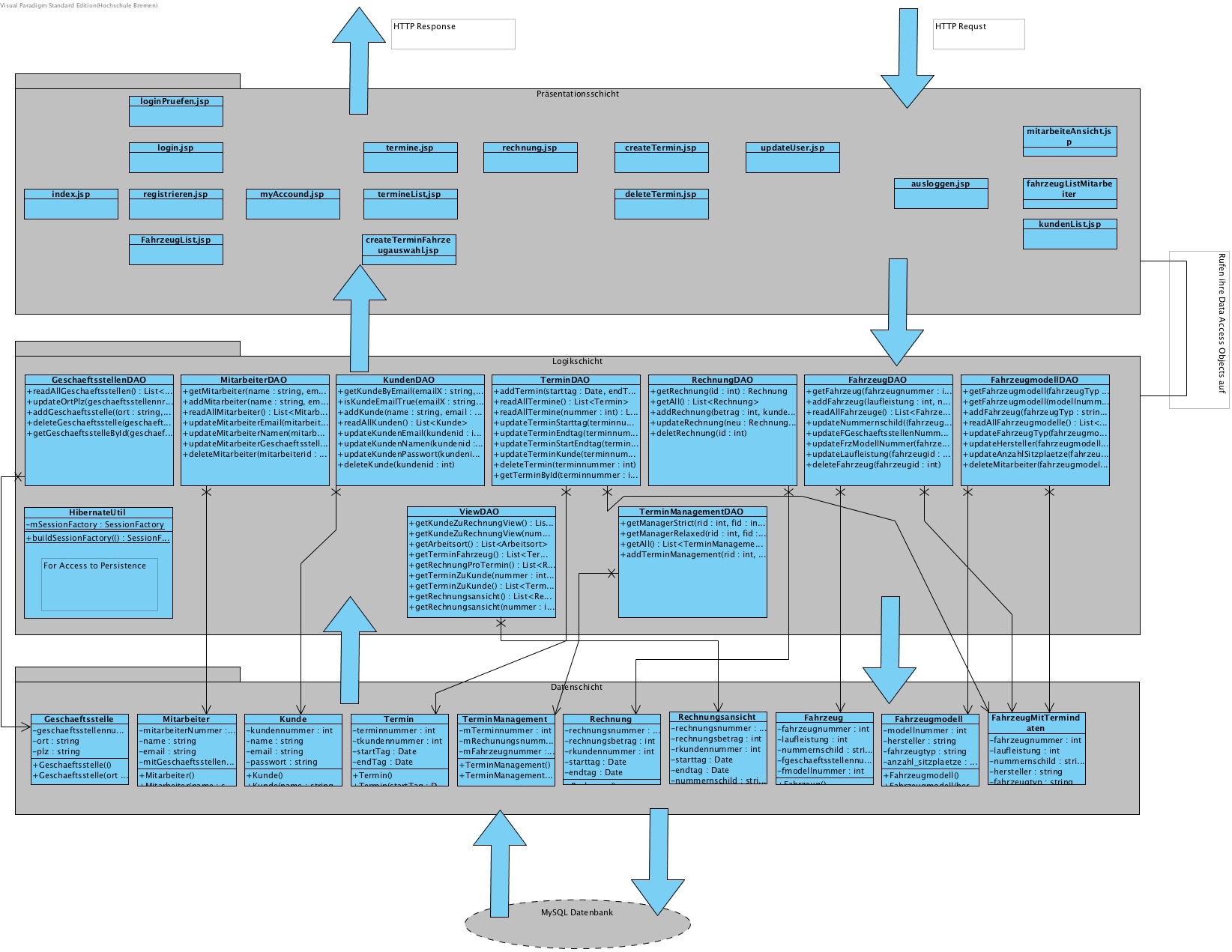
„Als Kunde möchte ich sehen, welche Fahrzeuge im Fuhrpark vorhanden sind um zu sehen ob die Autovermietung mein gewünschtes Fahrzeug besitzt.“

# 2. Architektur

*In diesem Kapitel soll beschrieben werden, wie das Projekt und die Datenbank strukturiert ist und welche Technologien und Frameworks verwendet werden.*

Im Folgenden wird die Architektur des Projektes erläutert und gezeigt, wie der Programmablauf ist. Daraufhin folgt die Erklärung der Persistenzschicht und als letzter Punkt wird auf die Präsentationsschicht eingegangen.

## 2.1 Gesamtarchitektur



Das Projekt ist in einer Drei-Schichten-Architektur 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 querry statements die CRUD(Create Read Update Delete) Funktionen.

* Präsentationsschicht

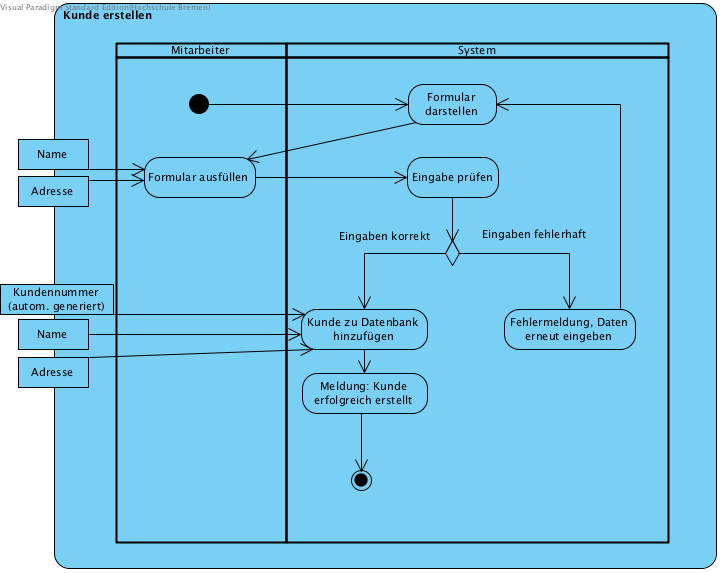
Hier sind die JSPs (Java Server Pages) zu finden. Die JSPs beinhalten HTML Code gepaart mit Java Code. In diesem Code werden die DAOs aus der Logikschicht abgerufen.

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 in der Präsentationschicht befinden generieren ein Servlet welches HTML ausliefert. Somit gibt die Präsentationschicht HTTP-Responses zurück.

## 2.2 Programmabläufe

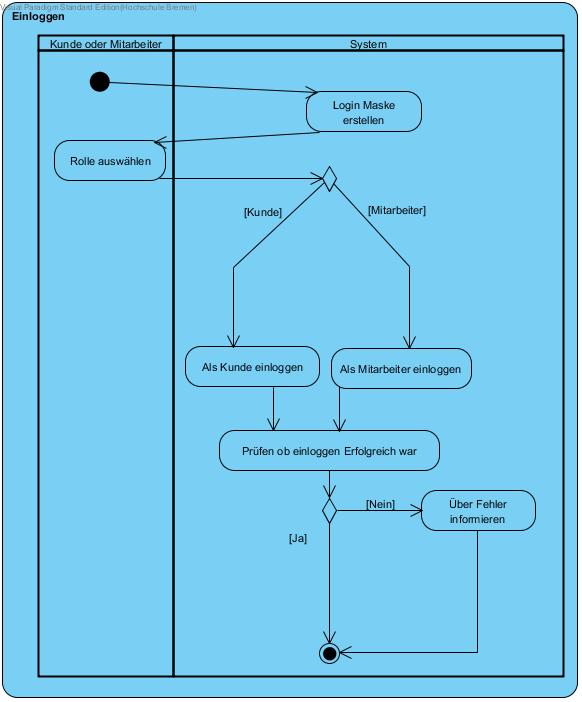
Im folgendem werden die Datenbankzugriffe exemplarisch beschreiben. Typische Abläufe sind die CRUD (Create, Read, Update & Delete) Operationen. Der Zugriff auf die Datenbank findet mittels Hibernate (OR- Mapping) statt.

CREATE (Kunde erstellen)

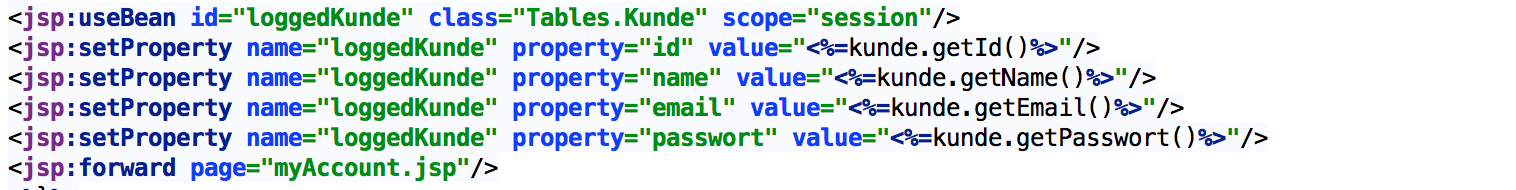


Zum erstellen (Create) einer neuen Column in einer Entität wird in den DAOs die add Methode aufgerufen. Die Parameter entsprechen dabei den Eingabefeldern des Formulars der HTML Seite. Um einen neuen Kunden zu erstellen muss der User seinen Namen, Email-Adresse und ein Passwort angeben. Der Primärschlüssel, 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 eigetragen und darüber informiert, dass er nun Kunde ist. Sollten Eingaben nicht Korrekt sein wird der User aufgefordert falsche Eingaben zu korrigieren.

READ (Kunde einloggen)

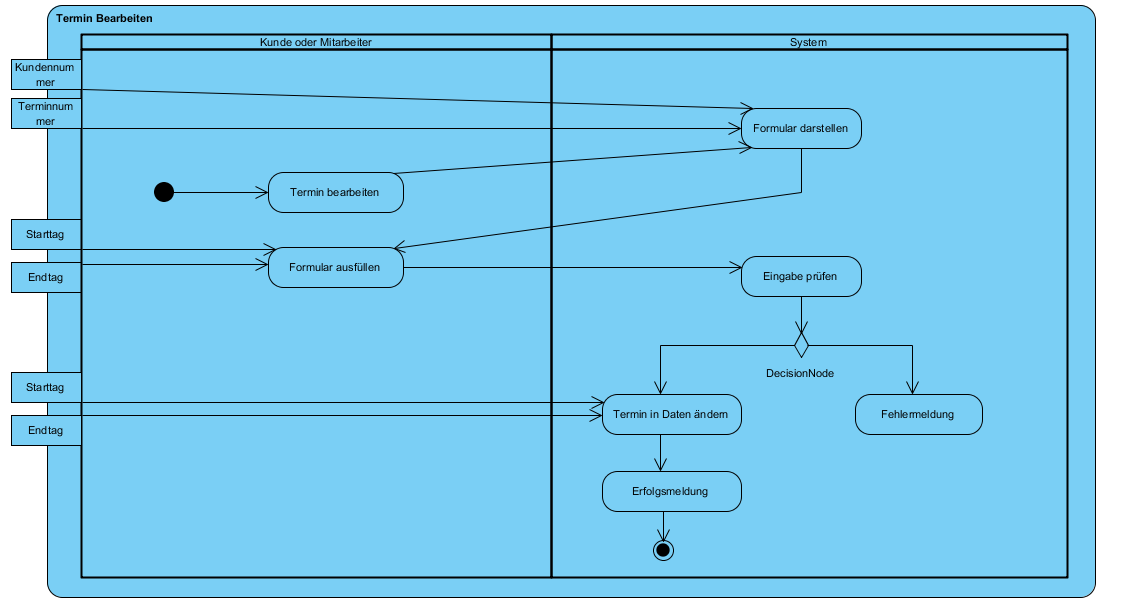


Eine Login Funktion war zu Anfang des Projekts nicht geplant. Aufgrund sich veränderter Anforderungen wurde diese Funktion in der finalen Abgabe implementiert. Diese Funktion steht für den Kunden zu Verfügung. In der login.jsp wird der Kunde aufgefordert seine Email-Adresse und sein Passwort einzugeben. Nach einem Submit wird der Kunde auf die loginPruefen.jsp weitergeleitet. Wird die Email-Adresse und das Passwort in der Datenbank gefunden wird ein useBean benutz um den Kunden über die Dauer der Session verfügbar zu machen.

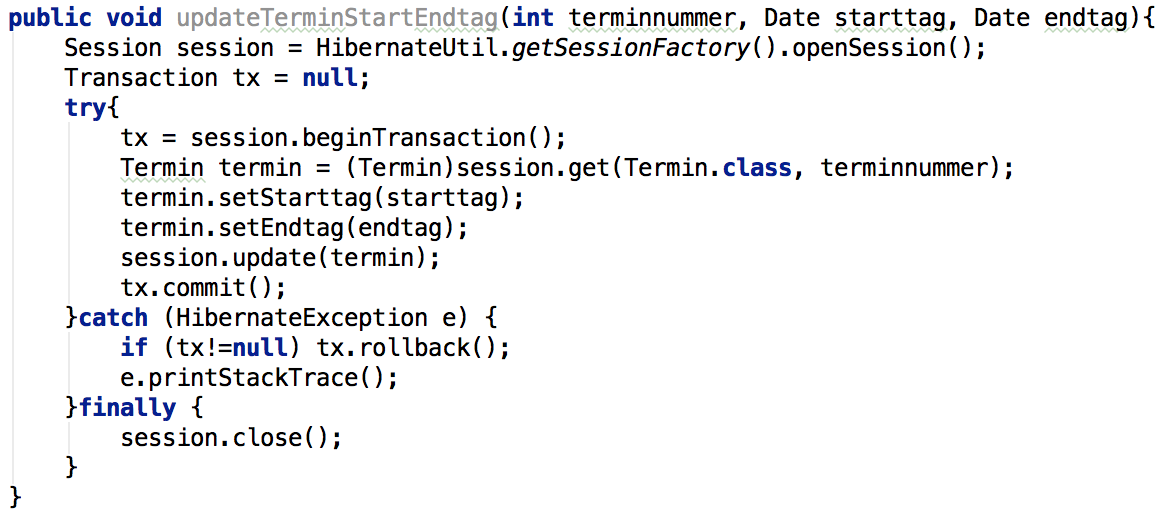


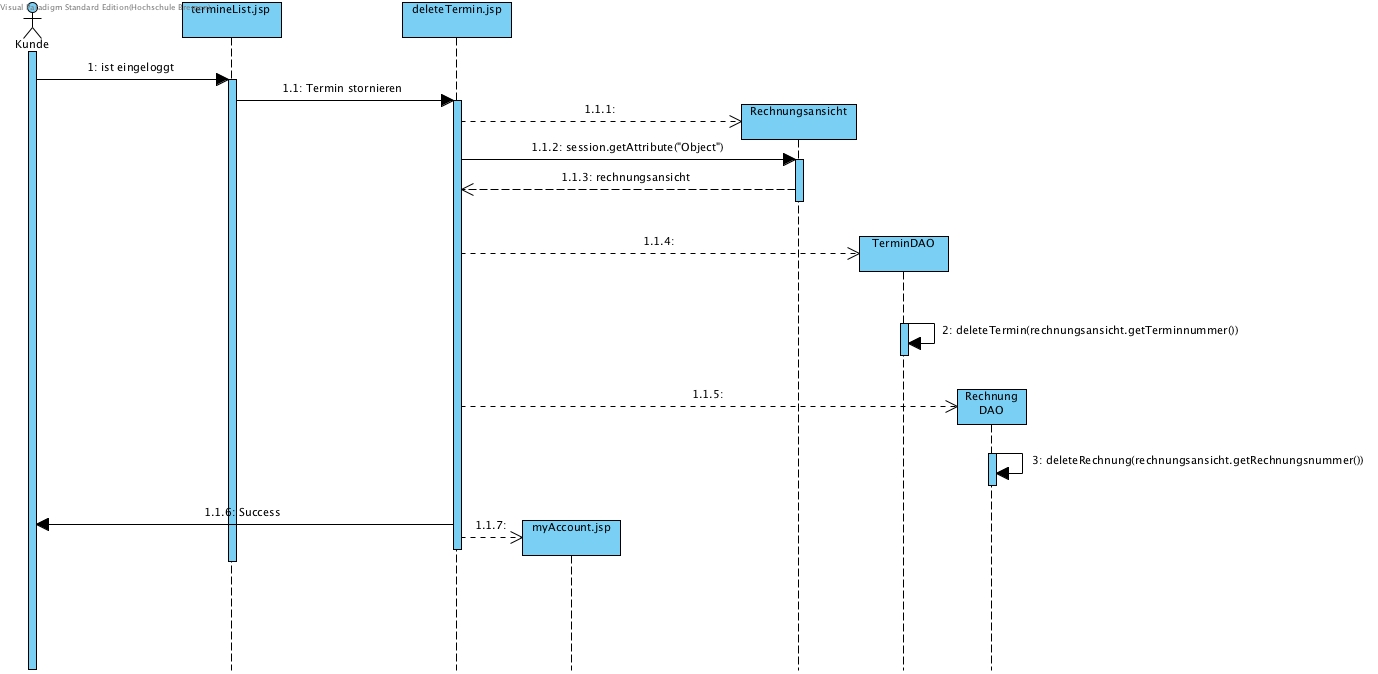
Der Kunde wird auf die myAccount.jsp weitergeleitet. In dieser JSP stehen dem Kunden alle Use Cases zur Verfügung.

UPDATE (Termin Bearbeiten)

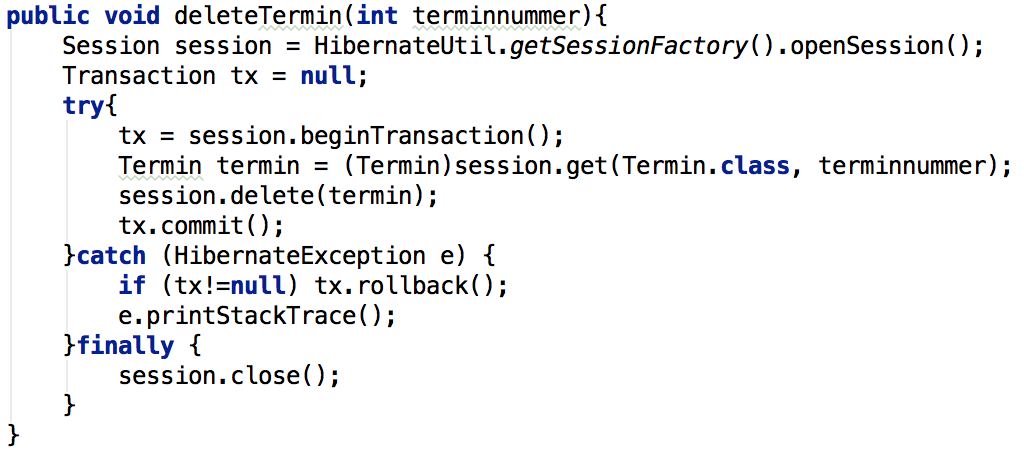


Zum Verändert einer Column wird die update Methode der jeweiligen DAO Klasse aufgerufen. Sie enthält als Parameter immer den Primärschlüssel der Entität und als zweiten Paramater der Wert der verändert werden soll. In der Präsentationschicht kann über den useBean der Kunde abgerufen werden. Mit den Kundeninformationen 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.



DELETE (Termin löschen)

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



## 2.3 Persistenz

*In diesem Unterkapitel soll die genutzte Datenbank und der Zugriff auf ihre Daten erläutert werden.*

In unserer Anwendung benutzen wir eine MySQL Datenbank, welche auf einem Server der Hochschule Bremen gehostet wird.

Für unser Projekt haben wir einen Account bekommen, den „dbweb\_user\_09“ mit dem wir Zugriffe auf die Datenbank „dbweb\_sose15\_09“ realisieren können.

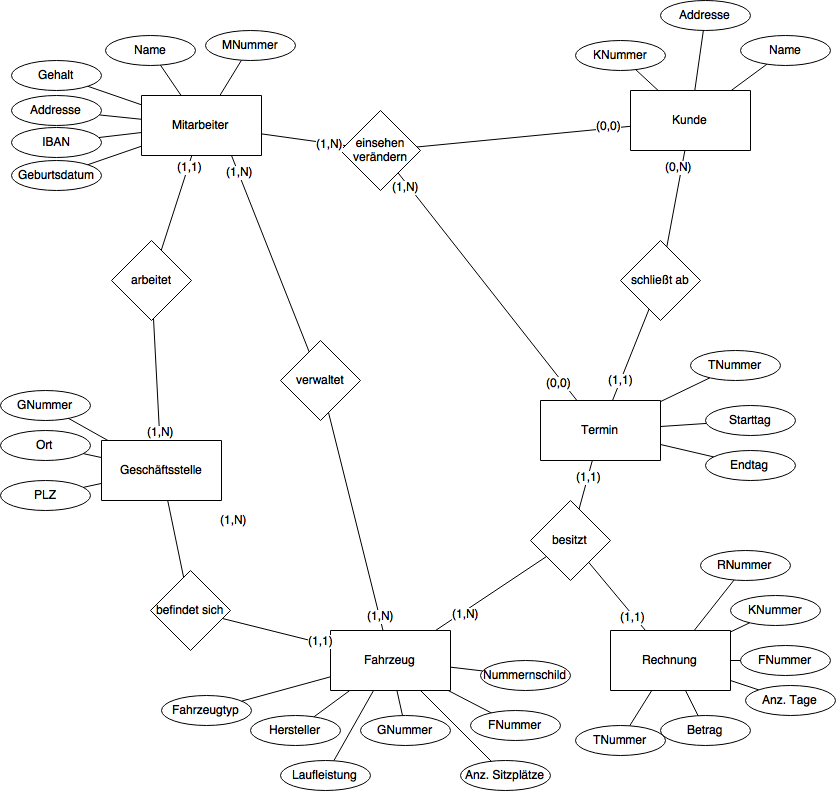
Als Zugriffstool verwenden wird MySQL Workbench 6.3, das offizielle Tool von Oracle.

In der Endanwendung werden von uns 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.

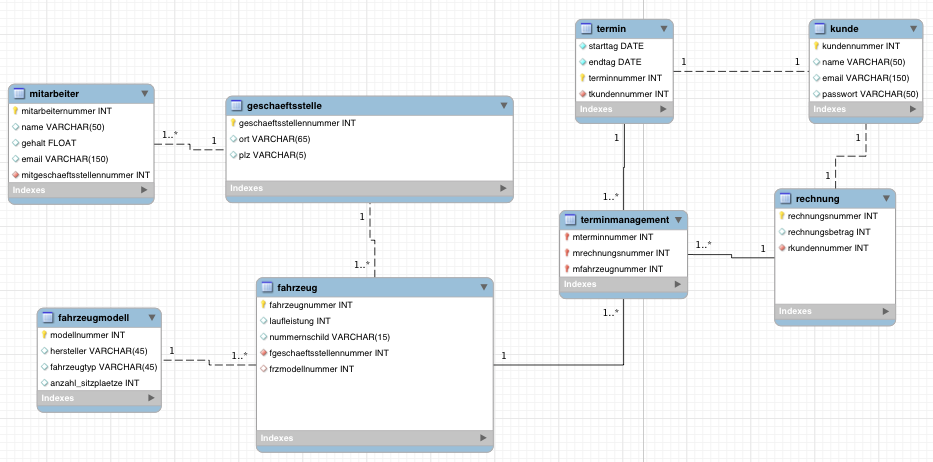
### 2.3.1 Konzeption

*In diesem Unterkapitel soll die konzepierte Datenbank nach ER-Diagramm beschrieben werden*

Zu Beginn des Projektes stand die Konzeption der Datenbank mithilfe eines Entity-Relationship-Diagramm. In diesem Diagramm sind Entities und deren Beziehungen miteinander eingezeichnet. Das ER-Diagramm der Autovermietung besitzt 6 Entities( Mitarbeiter, Kunde, Geschäftsstelle, Termin, Fahrzeug, Rechnung) und 6 Beziehungen untereinander, wobei zu beachten ist, dass eine der Beziehungen (zwischen Termin, Fahrzeug und Rechnung) eine Dreiecksbeziehung ist. Die drei genannten Entities besitzen jeweils die andere Entity. Eine weitere, nicht einfache Beziehung besteht aus der Entity Mitarbeiter, welche die Entities Kunde und Termin einsehen, beziehungsweise verändern kann. Diese Beziehung ist dabei einseitig, denn Termin und Kunde können nicht, den Mitarbeiter einsehen oder verändern. Jedoch haben Kunde und Termin eine Beziehung untereinander, denn ein Kunde kann Termine abschließen. Die Entity Mitarbeiter besitzt aber noch zwei weitere Beziehungen. Die eine Beziehung ist, dass ein Mitarbeiter bei einer Geschäftsstelle arbeitet. Die andere Beziehung besteht zum Fahrzeug. Ein Mitarbeiter kann mehrere Fahrzeuge verwalten, welche einen Termin besitzt und somit über den Termin mit dem Kunden verbunden ist.



### 2.3.2 Umsetzung

*In diesem Unterkapitel soll die Datenbank nach relationalem Modell beschrieben werden. Außerdem sollten anwendungsspezifische Constraints aufgelistet werden, die bei der Implementierung bedacht werden müssen.*

Das Relationale Modell unterscheidet sich vom Entity Relationship Modell, da wir nach Normalisierungen und datenbankspezifischen Eigenheiten einige sachen ändern 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 enstanden, da sich Daten oft widerholen 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 Endtatum liegen

### 2.3.3 Zugriff

*In diesem Unterkapitel soll beschrieben werden mit welchen Technologien und Frameworks von der Webanwendung auf die Datenbank zugegriffen wird.*

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.

Wir haben uns für JDBC entschieden, da es die einzige Datenbankschnittstelle für Java ist, die uns bekannt ist. Als OR-Mapper Framework haben wir Hibernate gewählt. Da wir vorher kein OR-Mapper Framework kannten, mussten wir uns erst einmal informieren, um uns für ein Framework zu entscheiden. Nach ausgiebiger Recherche hat Hibernate den besten Eindruck gemacht hat. Die Community ist sehr groß, d.h. es gibt viele Beispiele an denen man sich orientieren und lernen kann. 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 in unseren Augen für unseren Zweck zu aufwendig war.

## 2.4 Präsentation

*In diesem Unterkapitel soll die Präsentationsschicht der Webanwendung beschrieben werden. Hierzu u.a. die genutzte Technologie/Framework.*

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 nur die DAOs (Data Access Object) der Logikschicht auf, so wie es in der Drei-Schichten-Architektur üblich 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 unsere Anwendung, die vom Umfang und von der Komplexität eher gering ist, 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 Logikanteil deutlich größer werden kann.

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

# 3. Installation

*In diesem Kapitel soll beschrieben werden, welche Programmierwerkzeuge verwendet wurden und wie die Webanwendung lokal aufgesetzt werden kann*

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

## 3.1 Programmierwerkzeuge

*Auflistung der verwendeten Programmierwerkzeugen mit Vor- und Nachteilen*

### 3.1.1 Entwicklungsumgebung: *Intellij*

Der Hauptgrund, dass wir IntelliJ verwendeten, war, dass eine Person bei uns Erfahrung damit 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 sehr gut in der Entwicklungsumgebung. Negativ aufgefallen ist uns allerdings das Fehler Highlighting und das teils unübersichtliche Design.

### 3.1.2 Versionskontrollsystem: *Git*

Sowie bei IntelliJ war der Hauptgrund, dass wir Git benutzten unsere angesammelte Erfahrungen mit Git, die sogar noch mehr sind, als die von IntelliJ. Uns wurde früh im Studium nahegelegt Git zu verwenden, da es sehr gut im Mergen ist und wir keinen eigenen Server benötigen, wie für z.B. 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*

Nachdem wir uns über die Buildtools erkundigt hatten, haben wir uns für Gradle entschieden, da es die Vorteile von Maven und Ant bietet und dabei sehr einfach zu installieren ist. Diese Vorteile sind auch teils dadurch, dass Gradle das modernste der drei Systeme ist und von den anderen Tools lernen konnte.

### 3.1.4 Komponententest-Framework:

Wir haben uns gegen die Nutzung eines Komponententest-Frameworks entschieden.

Aus unserer Sicht haben wir uns in dem Modul mit vielen neuen Sachen beschäftigt. Wir wissen das „Test-First“ ein wichtiger Aspekt der Agilen Programmierung ist. Allerdings haben wir uns im letzten Semester nur kurz und theoretisch mit dem Thema beschäftigt.

## 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

*Beschreibung wie die Anwendung auf dem lokalen Rechner gestartet werden kann*

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

Wir benutzen alle einen lokalen Tomcat Server, also muss dieser als erstes installiert werden.

Als Entwickler kann man nachdem man das Projekt erfolgreich importiert am einfachsten arbeiten.

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

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

Nun muss man den Tomcat Server als Run Configuration einstellen. Wenn man einfach auf Run Klickt und noch keine Konfiguration hat, kann man diese erstellen.

Dazu wählt man unter der Run Configuration -> Defaults den Tomcat Server -> Local und erstellt dort die neue Konfiguration.

Hierbei muss man noch bei dem Punkt Deployment das Gradle Artefakt hinzufügen z.B. „Gradle: hsb Graddle test: Graddle test -.0: SNAPSHOT.war“

Dann erscheint unten in dem Feld before Launch: Make, Build Artefact.

Nun kann man das gesamte Projekt mit einem klick aur Run laufen lassen und der Tomcat server wird gestartet, und es öffnet sich automatisch eine Seite im Browser, wenn das ganze fertig ist.

Alternativ, wenn man nur die XXX.war Datei hat, kann man diese einfach in das Tomcat/Webapps Verzeichnis legen, seinen Server starten und dann im Browser über localhost:8080/XXX darauf zugreifen.

## 3.4 Konfiguration der Datenbank

*Beschreibung welche Schritte durchgeführt werden müssen, um die Datenbank mitsamt ihrer Struktur und Beispieldaten auf einem lokalen System aufzusetzen*