****

**Internationaler Studiengang Medieninformatik**

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

**Autovermietung**

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

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 sollen die folgenden Punkte erläutert werden:*

* *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.“

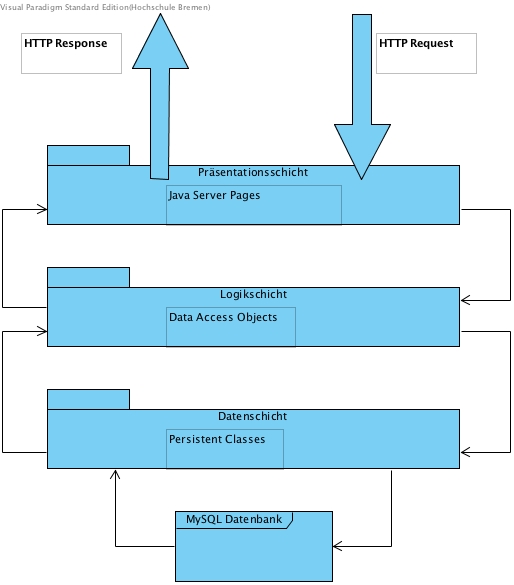
„Als Kunde möchte ich abrufen können, welche Fahrzeuge nicht vermietet sind, um zu prüfen, ob das gewünschte Fahrzeug für meinen Termin verfügbar ist.“

„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.*

## 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

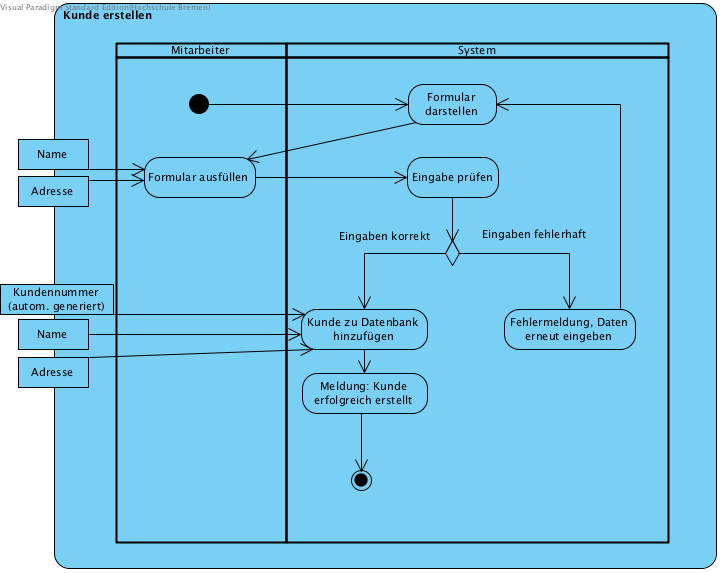
Hier sind die JSPs (Java Server Pages) zu finden. Die JSPs beinhalten HTML Code gepaart mit Java Code. In diesem Code werden die DAOs aus der Logikschicht abgerufen.

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

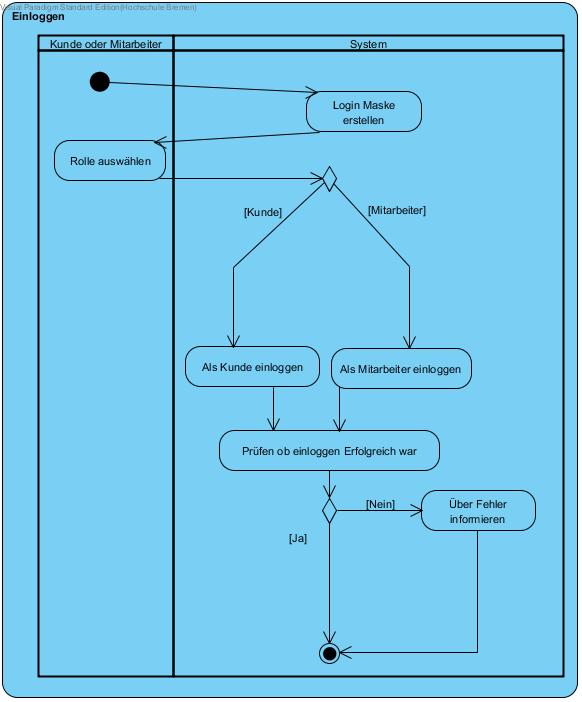
Im folgendem werden die Datenbankzugriffe exemplarisch beschreiben. Typische Abläufe sind die CRUD (Create, Read, Update & Delete) Operationen. Der Zugriff auf die Datenbank findet mittels Hibernate (OR- Mapping) statt.

CREATE (Kunde erstellen)

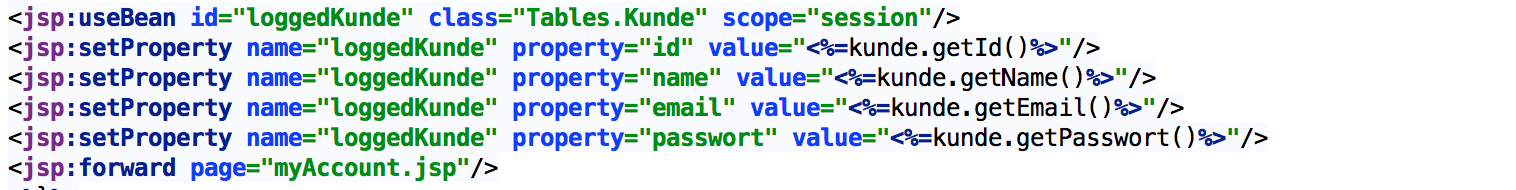


Zum erstellen (Create) einer neuen Column in einer Entität wir in unseren DAOs die add Methode aufgerufen. Die Parameter entsprechen dabei den Eingabefeldern des Formulars der HTML Seite. Um einen neuen Kunden zu erstellen muss der User seinen Namen, Email-Adresse und ein Passwort angeben. Der Primärschlüssel, 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)

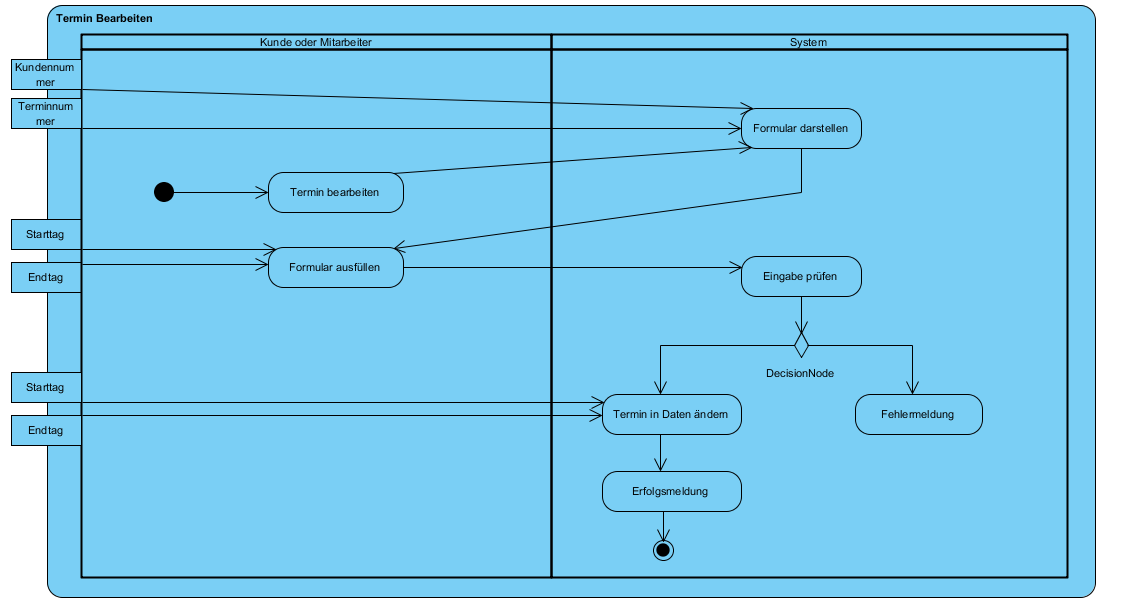


Eine Login Funktion war von uns zu Anfang des Projekts nicht geplant. Aufgrund sich veränderter Anforderungen haben wir diese Funktion in der finalen Abgabe doch implementiert. Diese Funktion steht für den Kunden zu Verfügung. In der login.jsp wird der Kunde aufgefordert seine Email-Adresse und sein 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 useBean benutz um den Kunden über die Dauer der Session verfügbar zu machen.

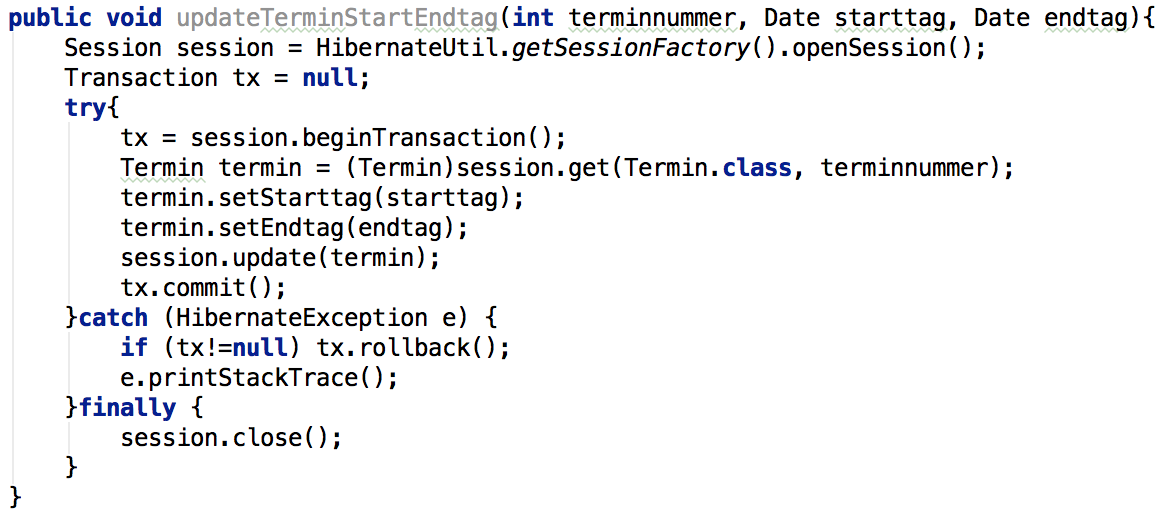


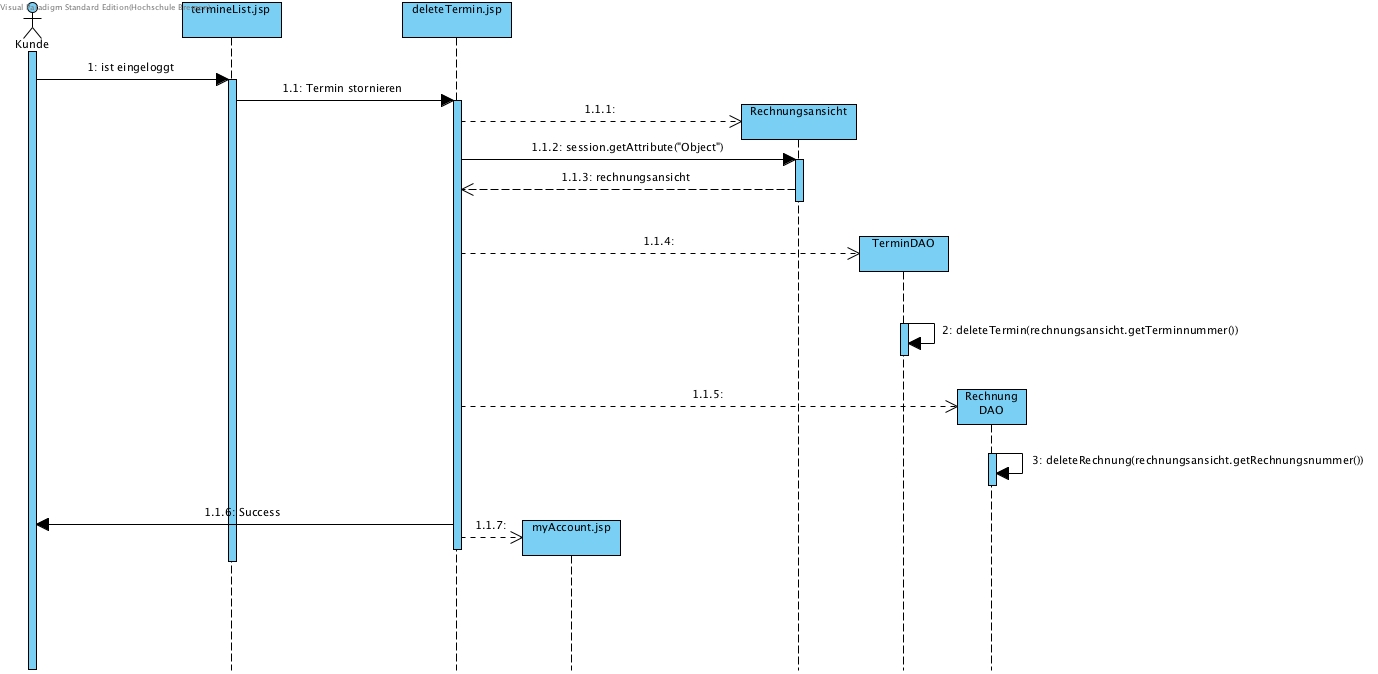
Der Kunde wird auf die myAccount.jsp weitergeleitet. In dieser JSP stehen dem Kunden alle Use Cases zur Verfügung.

UPDATE (Termin Bearbeiten)

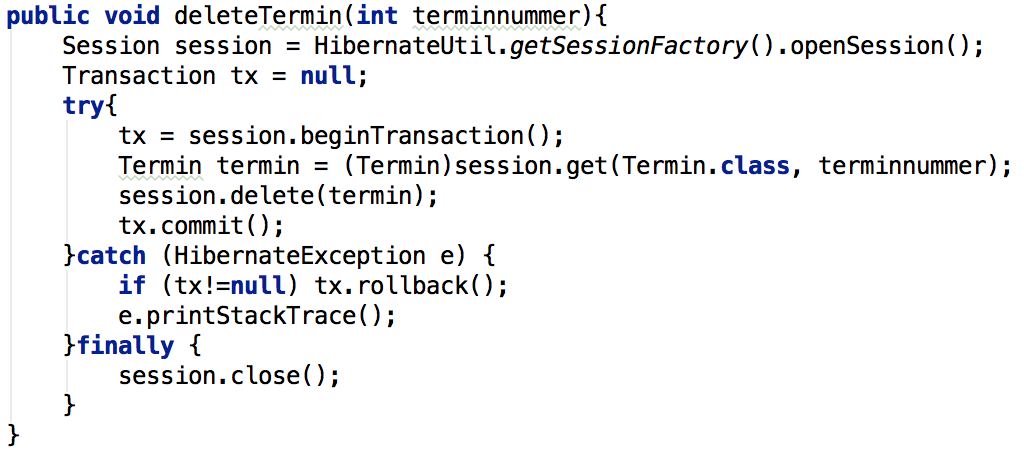


Zum Verändert einer Column wird die jeweilige update Methode der jeweiligen DAO Klasse aufgerufen. Sie enthält als Parameter immer den Primärschlüssel der Entität und als zweiten Paramater der Wert der verändert werden soll. In der Präsentationschicht kann über den useBean der Kunde abgerufen werden.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

### 2.3.1 Konzeption

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

### 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.*

### 2.3.3 Zugriff

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

## 2.4 Präsentation

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

# 3. Installation

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

## 3.4 Konfiguration der Datenbank

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