Entwurfsbeschreibung

mokl17

24. Mai 2017

Inhaltsverzeichnis

1 Allgemeines **1**

2 Produktübersicht **1**

3 Grundsätzliche Struktur- und Entwurfsprinzipien **2**
3.1 Modulkonzept . . . 2
3.2 Prepared Statements . . . 2
3.3 Benötigte Architektur und Schnittstellen . . . 2
3.3.1 Application Server . . . 2
3.3.2 Client-/Serverarchitektur . . . 2
3.3.3 MVC-Architektur . . . 3
3.3.4 JDBC . . . 3
3.3.5 Representational State Transfer . . . 3

4 Struktur- und Entwurfsprinzipien einzelner Pakete **4**
4.1 UserAdministrationModule . . . 4
4.2 KpiModule . . . 6
4.3 RulesModule . . . 7
4.4 StatusModule . . . 7
4.5 IncidentsModule . . . 9
4.6 DbConnectionModule . . . 10
4.7 Restservice-Module . . . 11
4.8 Mobile App . . . 12

5 Datenmodell **13**
5.1 Speicherkonzept . . . 13
5.2 Entity-Relationship Modell . . . 14

6 Benötigte Technologien und Frameworks **15**
6.1 Ionic . . . 15
6.2 Java . . . 15
6.3 Maven . . . 15
6.4 Vaadin . . . 15
6.5 Drools . . . 15

7 Glossar **16**
7.1 Client-Server . . . 16
7.2 JDBC . . . 16
7.3 Kryptographische Hashfunktion . . . 16
7.4 MVC . . . 16
7.5 Prepared Statements . . . 16
7.6 ReST . . . 16
7.7 SQL-Injection . . . 16
7.8 DML . . . 16
7.9 DQL . . . 17
7.10 URI . . . 17

1 Allgemeines

In dem vorliegende Dokument soll die Realisierung einer Software für Logistikunternehmen dargelegt werden. Insbesondere soll dem Aufbau und den Funktionalitäten, sowie der geplanten technischen Umsetzung Einblick gewährt werden. Ziel ist es, projektfernen Entwickler einen guten Überblick über die Software und ihre Bestandteile zu geben, so dass diese sich möglichst problemlos in das Projekt einarbeiten können. Dafür werden anfangs im Punkt Produktübersicht Features der Software erläutert. Darauf folgt eine Übersicht über Entwurfsprinzipien, welche das Softwareprojekt als ganzes betreffen. Im Punk 4, den Struktur- und Entwurfsprinzipien einzelner Pakete wird Modul für Modul erläutert anhand von Klassendiagrammen. Danach folgt ein Überblick über das Datenmodell anhand des Speicherkonzepts und eines Entity-Relationship-Modells. Zum Schluss werden noch kurz verschiedene verwendete Technologien und Frameworks genannt.

2 Produktübersicht

Das Projekt hat im wesentlichen die Funktion Informationen zu sammeln, zu berechnen sowie diese übersichtlich darzustellen, um so gezielter und schneller auf spezifische Störfälle im Logistikunternehmen reagieren zu können.
Das Back Office besitzt mehrere Views die mithilfe des Frameworks Vaadin umgesetzt werden. Diese Views sind im wesentlichen das Benutzerverwaltungsmodul, das Störfällemodul, das StatusModule, das KpiModule, sowie das RulesModule (s. Arbeitsplan).
Bei der Benutzerverwaltung wird es ausschließlich dem Administrator möglich sein, neue Benutzer anzulegen, sowie bereits vorhandene zu bearbeiten, zu löschen oder in Gruppen einzuteilen.
In der View der Regeldefinition ist es dem Administrator und Geschäftsführer möglich, Regeln zur Berechnung der Kennzahlen festzulegen. Außerdem wird es die Funktion geben Wunschkennzahlen zu definieren zur Überprüfung ob sich das Unternehmen seinen vorgegebenen Zielen annähert oder gar entfernt.
Bei der KPI Berechnung kann der Administrator,Geschäftsführer,Disponent neue Touren hinzufügen sowie verschiedene Kennzahlen auf Basis der definierten Regeln errechnen lassen. Diese Kennzahlen stellen die Anzahl und den Gewinn abgeschlossener Touren, die Anzahl und den voraussichtlichen Gewinn der sich in Bearbeitung befindlichen Touren und die Anzahl der sich in Planung befindlichen Touren dar.
In der Statusanzeige wird der aktuelle Status jeder Tour errechnet und übersichtlich im Ampelsystem in Listen dargestellt.
Bei der Störfälleanzeige werden alle Touren die von der ursprünglichen Planung zu stark abweichen, also den Status Gelb oder Rot haben, aufgelistet.
Desweiteren können die Nutzer auswählen wie sie im Falle von Störfällen benachrichtigt werden möchten. Dabei werden Push-Benachrichtigung sowie Tonbenachrichtigung zur Auswahl stehen.

Neben dem Back Office stellt das Projekt auch eine mobile App zur Verfügung. Diese dient ausschließlich zur Informationsanzeige der im Back Office gesammelten und errechneten Daten. Auch hier wird die Anzeige von den Rechten des jeweiligen Nutzers abhängen.
Der Disponent hat das Recht sich eine Liste von Touren und deren Status, in Ampelform, sowie den zugehörigen Tour-Details anzeigen zu lassen. Desweiteren hat er die Möglichkeit sich Listen von Störfällen und Nachrichten darstellen zu lassen.
Der Disponent hat auch das Recht sich bestimmte Kennzahlen anzeigen zu lassen. Während Disponenten nur die Anzahl(Kennzahlen) abgeschlossener,aktuell in Bearbeitung befindlicher, sowie zukünftiger Touren sieht, hat der Geschäftsführer auch die Rechte zum Einsehen des (voraussichtlichen) Gewinns der Touren.

Zusätzlich zu all diesen Ansichten hat der Administrator die Möglichkeit sich alle angemeldeten Benutzer und deren Rechte und Details anzeigen zu lassen. Jeder Nutzer wird von der App, je nach Einstellung im Back Office, mit Push- und/oder Tonbenachrichtigung bei bestimmten Störfällen alarmiert.

3 Grundsätzliche Struktur- und Entwurfsprinzipien

In diesem Softwareprojekt kommen mehrere Struktur- und Entwurfsprinzipien zum Einsatz, die sich durch das gesamt Projekt ziehen und nicht bestimmten Modulen zuordnenbar sind. Diese sollen hier kurz erläutert werden und die Entscheidung für deren Anwendung nachvollziehbar gemacht werden.

3.1 Modulkonzept

Das Projekt folgt dem Modularitätsprinzip und damit einem komponentenbasierte Ansatz. Die Software ist gegliedert in Module, die miteinander kommunizieren. Das Modularitätsprinzip wird gewählt, da es flexible Entwicklungszyklen ermöglicht und die Komponenten einfach austauschbar durch die jeweiligen nachfolgenden Versionen sind. Das komplexe Gesamtsystem wird durch einen Modularen Ansatz einfacher verständlich. Außerdem ermöglicht es später eine einfachere Wartbarkeit der Software als beispielsweise ein stark integrierter Ansatz.

3.2 Prepared Statements

Da eine Weboberfläche mit Eingabedaten entwickelt wird, muss die Software gegen SQL-Injections (siehe 7.7) abgesichert werden. Aus diesem Grund verwenden wir Prepared Statements (siehe 7.5). mit später Paramenterbindung (nur bestimmte Eingabeformate werden zugelassen). Diese PreparedStatements werden vor allem im RulesModule zur Anwendung kommen um Daten aus der Datenbank dem Drools Regelsystem zur Verfügung zu stellen.

3.3 Benötigte Architektur und Schnittstellen

Im nachfolgenden Abschnitt soll dem projektfernen Entwickler ein Gesamtüberblick über die verwendete Softwarearchitektur gewährt werden, sowie Prinzipien der Entwicklung, die sich über das gesamte Projekt strecken. Details zu einzelnen Modulen werden in Abschnitt 4 behandelt.

3.3.1 Application Server

Für das Projekt kommt ein Application Server zum Einsatz. Dieser bildet das Rahmenwerk für die Backoffice-Anwendung, welche den Kern des Softwareprojekts darstellt. Auf dem Application Server werden Dienste zur Verwaltung und Speicherung der Daten, Anbindung an die Datenbank (4.6) sowie Authentifizierung (4.1) bereitgestellt. Für das Projekt kommt Apache Tomcat 8.5 zum Einsatz.

3.3.2 Client-/Serverarchitektur

Die Vaadin-Internetanwendung basiert auf dem Client-/Server Prinzip. Das bedeutet, dass alle Daten auf dem Server persistieren und auch dort verarbeitet werden. Der Server stellt also die gesamte Anwendungsfunktionalität zur Verfügung. Der Client stellt Anfragen an den Server, der diese bearbeitet und dem Client das Ergebnis der verarbeiteten Anfrage zurückliefert. Als Application-Server wird Tomcat verwendet (siehe Abbildung 1). Clients greifen mit einem Browser oder von einem mobilen Endgerät auf diesen Server, auf dem Vaadin läuft, zu.

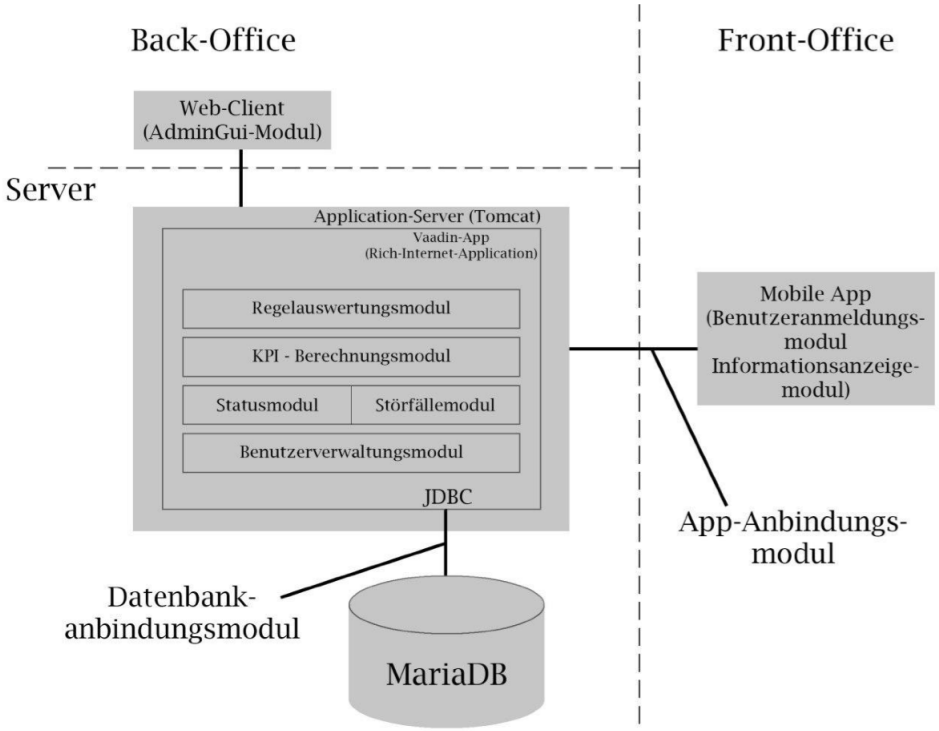

Abbildung 1: Client-/Server-Architektur

3.3.3 MVC-Architektur

Das Vaadin-Framework basiert auf der MVC-Architektur. Das heißt, dass die Programmlogik von der grafischen Benutzeroberfläche getrennt ist. Die logischen Komponenten werden zusammen mit den Views der jeweiligen Module in den einzelnen Packages gespeichert.
In den Views ist nur das Aussehen und Verhalten der grafischen Benutzeroberfläche definiert (View+Controll). Die logischen Komponenten (Model) beinhalten ausschließlich die Programmlogik ohne Bezug auf die grafische Benutzeroberfläche.
Da durch dieses Konzept die Logik strikt von der Oberfläche und ihrem Verhalten getrennt ist, ist es möglich, Änderungen in der Logik durchzuführen, ohne den Code der View anpassen zu müssen. Umgedreht kann auch die grafische Oberfläche angepasst werden, ohne die Programmlogik ändern zu müssen.

3.3.4 JDBC

Im Projekt wird für die Verbindung zur Datenbank die Datenbankschnittstelle JDBC angewendet. JDBC bietet eine einheitliche Schnittstelle zu den Datenbanken verschiedener Hersteller, sodass beim Kunden bei Austausch des Datenbankherstellers nur wenige Anpassungen in der Software nötig sind. Weitere Informationen zur Verbindung der Anwendung mit der Datenbank finden sich in der Erklärung zu dem Modul (siehe 4.6).

3.3.5 Representational State Transfer

Das Programmierparadigma der ReSTful-Webservices wird für die Schnittstelle zwischen Server und Client (Mobile Applikation) verwendet. ReST bietet sich für das umzusetzende Softwareprojekt an, da es sich für verteilte Systeme eignet und einen modernen Ansatz für Datenverfügbarkeit durch einen Webservice darstellt. Auch lässt sich ReST günstig in die bestehende

Architektur einbauen und setzt hauptsächlich das Vorhandensein eines Application-/Webservers voraus.

4 Struktur- und Entwurfsprinzipien einzelner Pakete

In diesem Punkt werden die einzelnen Module aus technischer Sicht kurz erläutert und eventuelle Entwurfsprinzipien und Konzepte, welche speziell in den jeweiligen Modulen zur Anwendung kommen, benannt. Für die Module befindet sich zur besseren Übersicht ein Klassendiagramm vor den Erläuterungen.

4.1 UserAdministrationModule

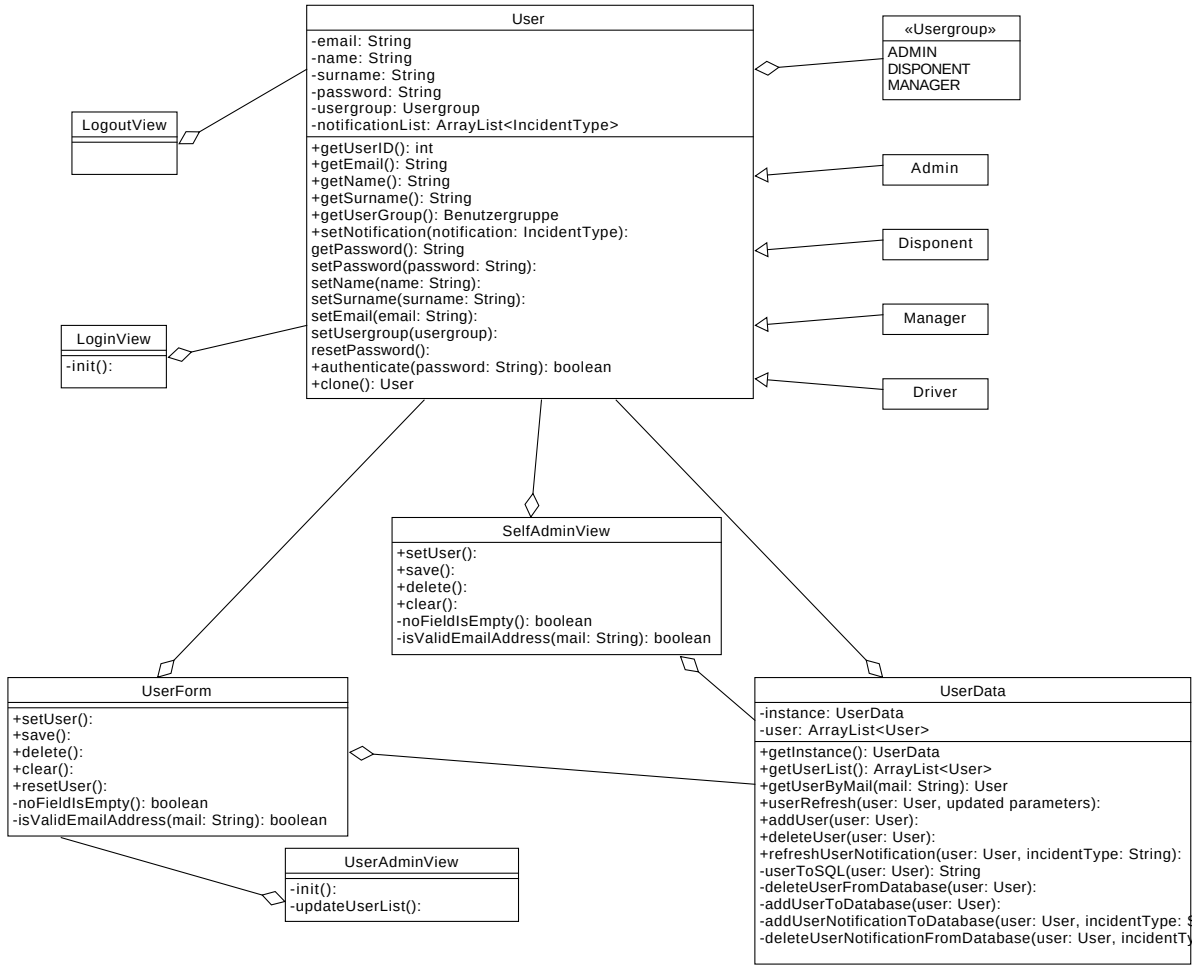

Abbildung 2: Benutzerverwaltung

Das Benutzerverwaltungsmodul ist für die Authentifizierung und Verwaltung aller Benutzer zuständig. Jedem Benutzer wird eine Gruppe zugewiesen, sodass eine Rechteverwaltung möglich ist. Es gibt die Gruppen Admin, Manager, Driver und Disponent. Benutzer der Gruppe Admin sind für das Anlegen, Bearbeiten und Löschen der Benutzer zuständig. Manager können Regeln anlegen, bearbeiten und löschen. Außerdem bekommen sie die Kennzahlen für die jeweiligen Touren angezeigt. Disponenten bekommen den aktuellen Tourenstatus angezeigt. In der Klassenstruktur gibt es eine abstrakte Klasse User, in der die grundlegenden Informationen,

wie Name, Mailadresse, Nutzergruppe, Passwort sowie die gewünschten Benachrichtigungen bei bestimmten Störfällen gespeichert sind. Diese Struktur ist deckungsgleich mit den Tabellen user join usernotification in der Datenbank, in der die Benutzer persistieren (siehe hierzu Kap. 5.1). Diese Klasse stellt außerdem die Grundfunktionalität für Benutzer bereit. Sie beinhaltet also Setter und Getter für alle Instanzvariablen, die Möglichkeit, das Passwort auf den Standartwert 'password' zurückzusetzen und den Benutzer zu authentifizieren. Von der Klasse User erben die Klassen Admin, Manager und Disponent, die sich durch die Instanzvariable usergroup unterscheiden.
Das dem Benutzer zugehörige Passwort wird nicht im Klartext in der Datenbank gespeichert. Bei der Anmeldung eines Benutzers wird aus dem String aus dem Textfeld passwort der Klasse LoginView der Hashwert mit dem SHA-256 berechnet zur Authentifizierung benutzt. In der Spalte password in der Tabelle user der Datenbank wird demzufolge ebenso nur der Hashwert des Passwortes für jeden Benutzer gespeichert. Durch das Speichern des Hashwertes des Passworts wird verhindert, dass Personen, die sich unberechtigten Zugang zur Datenbank verschafft haben, zwar Zugriff auf den Hashwert, aber nicht auf das Passwort des Benutzers haben.
Die logische Einteilung der Benutzer in Gruppen dient zur vereinfachten Rechtevergabe. Je nach dem welcher Gruppe ein Benutzer zugeordnet ist, wird ihm eine für seine Gruppe angepasste View angezeigt. Benutzern der Gruppe Admin bekommen so zum Beispiel die Benutzerverwaltung angezeigt, Benutzer anderen Typs jedoch nur ein Tool zur Selbstverwaltung.

4.2 KpiModule

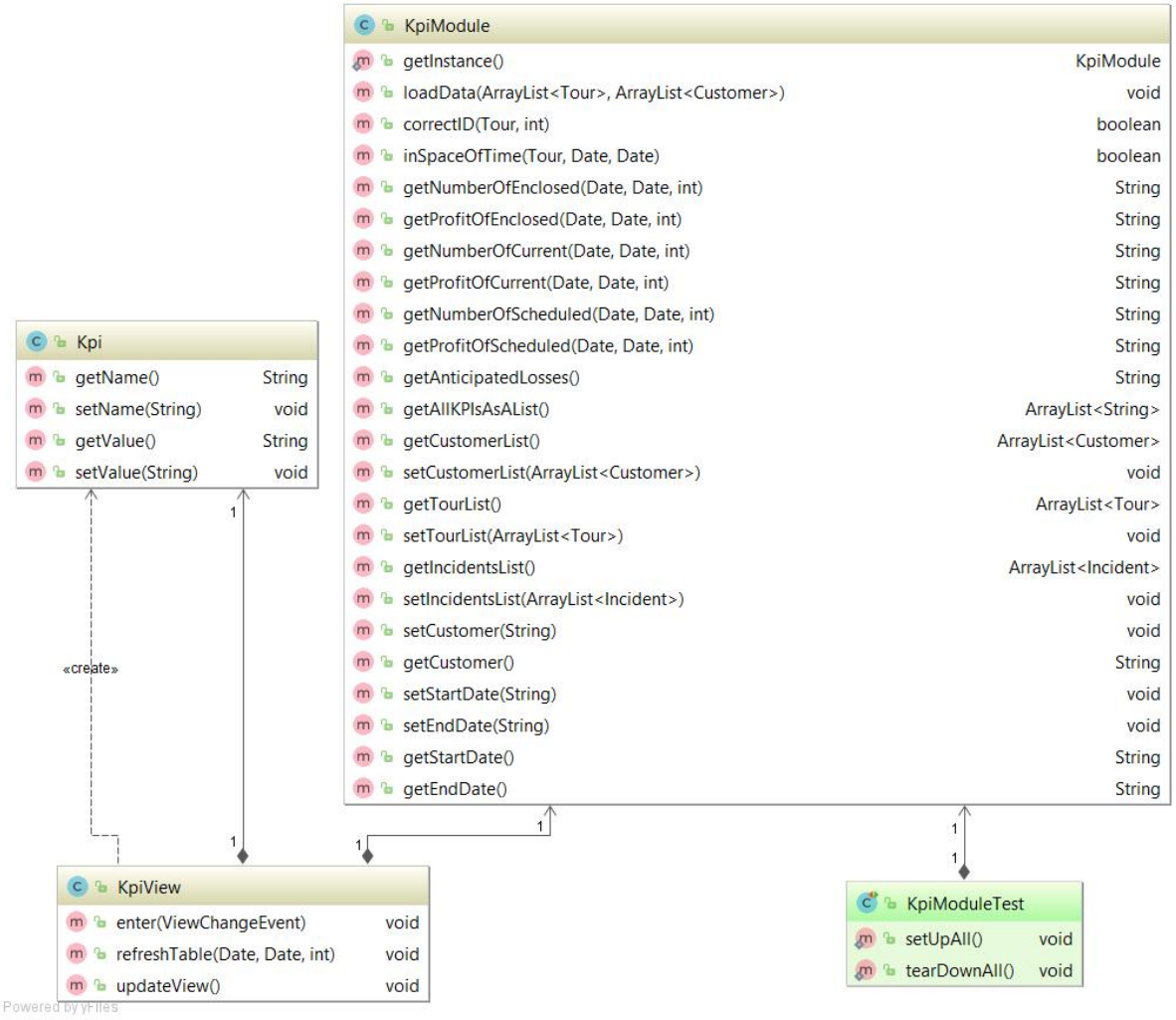

Abbildung 3: KpiModule

Das Modul ist grundsätzlich unterteilt in die Anzeige(KpiView) und die Auswertung(KpiModule) der Daten. In diesem Modul werden Informationen zum Status aller Touren gegeben. Das beinhaltet eine Tabelle, mit allen jemals gefahrenen Touren, die den Gewinn und Anzahl einzelner Tourgruppen(abgeschlossen,aktuell,geplant) auflistet. Die Anzeige der Daten erfolgt dabei über KPIView. Dieses Modul bekommt die Daten über verschiedene get-Methoden von KPIModule. Set-Methoden sind nicht wichtig, da es nur Daten anzeigt. Die Berechnung der nötigen Daten erfolgt in KPIModule. Dort erfolgt die Berechnung der KPIs, die über die Funktion loadData() neu gesetzt werden. Dazu gehört nicht nur die Tabelle, sondern auch eine ArrayList der KPIs, auf die andere Module zugreifen können. Über get-Methoden werden Daten über Kunden,Touren etc. gesammelt. Die über die set-Methoden wieder zurück geschickt werden. Wichtig ist hier, dass der Admin auch Einfluss auf die Daten nehmen können soll. Dazu steht ein ein DropDown-Menü zur Verfügung, um einzelne Kunden auszuwählen, sowie zwei Kalender, in denen Start- und Enddatum, der zu berechnenden KPIs, eingegeben werden kann. Sofern der Admin die eingegeben Daten laden oder zurücksetzen möchte, stehen ihm dazu zwei Buttons zur Verfügung. Zusätzlich werden dem Admin drohende Verluste angezeigt.

4.3 RulesModule

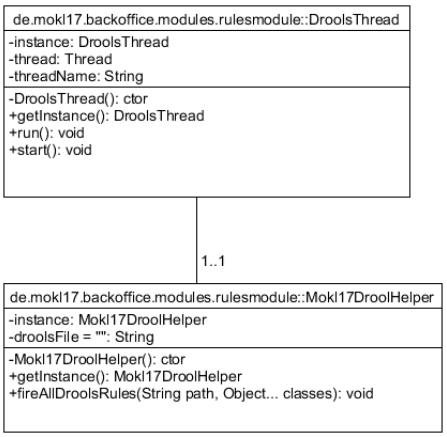

Abbildung 4: Rulesmodul

Durch die Einbindung eines regelbasierten Systems mit Hilfe von Drools, erreichen wir eine höhere Flexibilität unserer Logik. So werden die Stati und die KPIs vom Admin aus gesteuert. Die Regeln werden in einer externer Datei gespeichert auf die der Admin zugreifen kann. Die Bearbeitung der Regeln erfolgt genauso über die Datei. Wird die Anwendung gestartet so startet auch ein Thread der die Regeln, welche in der mokl17.drl Datei definiert sind periodisch auswertet und daraufhin Stati, Stöerfalllisten, Kennzahlen und Pushbenachrichtigungen steuert.

4.4 StatusModule

Das StatusModule hat 3 grundlegende Funktionen.

/FA 40/ Jeder Tour wird der aktuelle Status errechnet (rot,gelb oder grün)

/FA 41/ Jeder Tour wird die Information angefügt, ob diese von der Planung abweicht

/FA 42/ Anzeige des Gesamtstatus aller in Bearbeitung befindlichen Touren

Dieses Modul hält alle entscheidenden Daten vor und hat die entsprechenden Aufrufe um die Customer-,Tour- und Lkw-Objekte aus der Datenbank zu erstellen. Diese werden in den jeweiligen *Data-Klassen in Listen abgelegt und können von der RuleEngine aufgerufen werden um Operationen auf den Listen vorzunehmen und die Informationen der Listen auszuwerten. Um Threadsicherheit zu gewährleisten wird das sogenannte Initialize-On-Demand-Holder-Pattern verwendet, welches durch die Reihenfolge der Instanzierung von Klassen in der Java-Vm eine threadsichere Anwendung der Listen in den *Data-Klassen garantiert. Außerdem liegt in diesem Modul der Aufzählungstyp für die entsprechenden ausgewerteten Statianzeigen der Touren(CalculatedState).

Status Grün

- erwartete Ankunftszeit liegt vor der geplanten Zustellzeit
- Zustelltag ist der richtige
- Wareneingang des Empfängers ist zur erwarteten Ankunftszeit geöffnet

Status Gelb

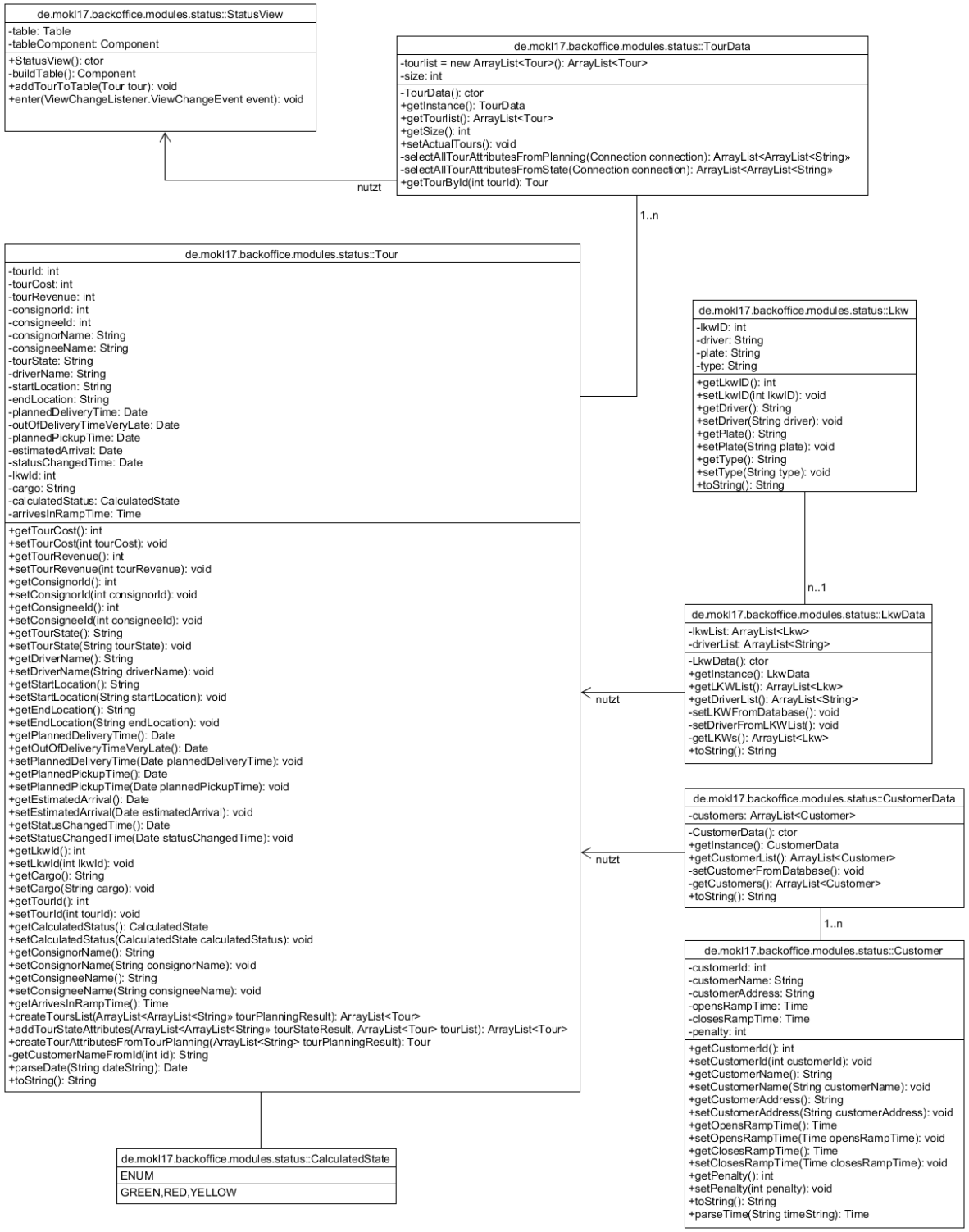

Abbildung 5: Statusmodul

- letzte erwartete Ankunftszeit liegt max. 15 Minuten nach der geplanten Zustellzeit
- Wareneingang des Empfängers ist zur erwarteten Ankunftszeit geöffnet

Status Rot

- letzte erwartete Ankunftszeit liegt über 15 Minuten nach der geplanten Zustellzeit
- letzte erwartete Ankunftszeit weicht vom geplanten Zustelltag ab
- Wareneingang des Empfängers ist zur erwarteten Ankunftszeit geschlossen

Um den Zugriff von statusrelevanten Daten herstellen zu können, gibt es eine DBHelper-Klasse für dieses Modul. Diese hat die Aufgabe schnell und einfach die Daten aus der Datenbank zu extrahieren und wie gewünscht zu formatieren.

4.5 IncidentsModule

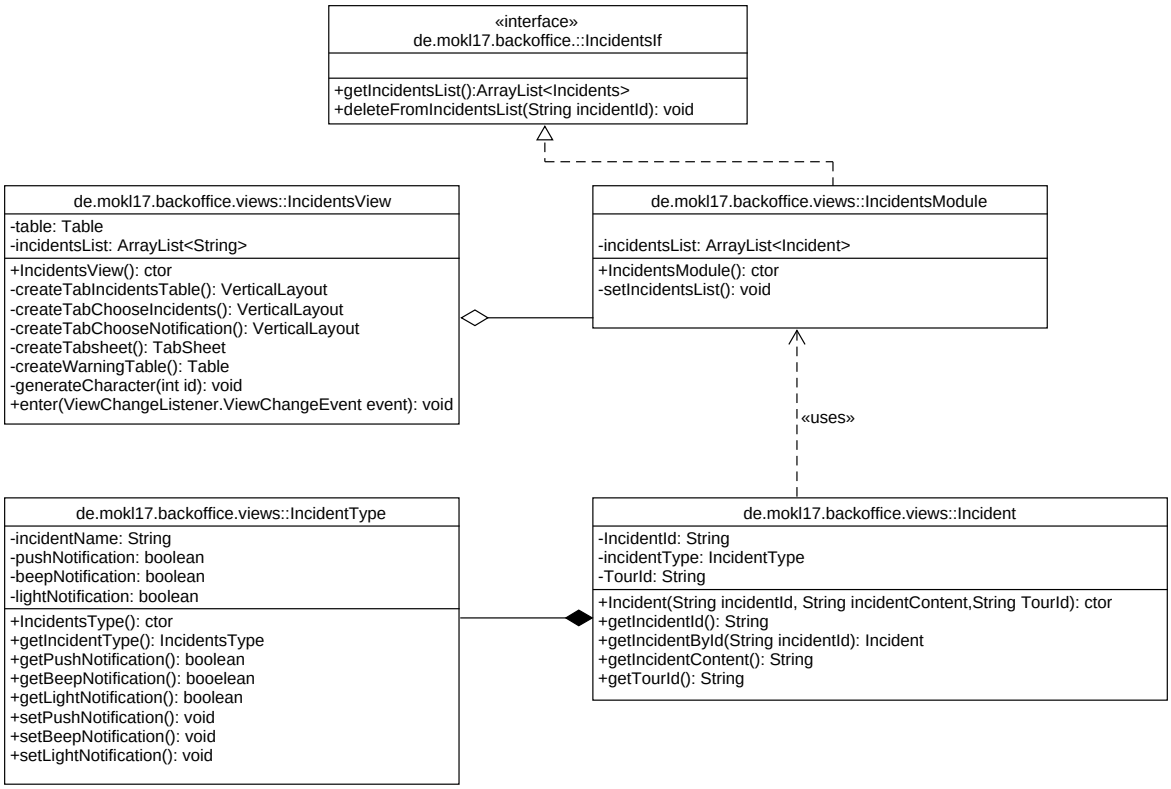

Abbildung 6: IncidentsModule

Das IncidentsModule stellt Methoden für das Regelmodul bereit. Greift in Drools eine Regel zur Identifikation eines Incidents, so wird aus dem Regelmodul heraus der Konstruktor von Incidents aufgerufen und ein Störfall für die entsprechende Tour erzeugt. Dazu wird beispielsweise im Drools der voraussichtliche Lieferzeitpunkt mit geplantem Lieferzeitpunkt einer Tour verglichen. Voraussichtliche Lieferzeitpunkte kommen dabei vom LogistikSystem des Kunden. Die Schätzung dieser Zeitpunkte ist nicht Teil der Applikation die hier entwickelt wird. Falls ein voraussichtlicher Lieferzeitpunkt zeitlich hinter dem geplanten Termin liegt wäre diese Tour als „verspätet“ identifiziert und würde daraufhin als ein entsprechendes Objekt Incident erzeugt. Dieses wird dann in einer IncidentsList abgelegt, welche bspw. für die RestSchnittstelle zugänglich gemacht wird. Weiterhin stellt das IncidentsModule Methoden bereit um bspw. die Benachrichtigungsart für eine bestimmte Störfallart festzulegen und im Incident zu hinterlegen.

4.6 DbConnectionModule

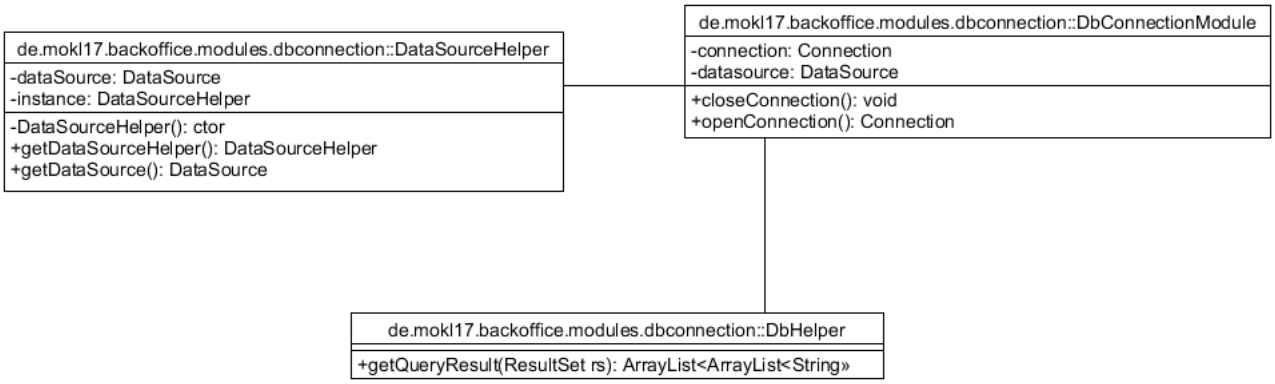

Abbildung 7: DbConnectionModule

Die Datenbank wird über das DbConnectionModule mit der Backoffice-Anwendung verbunden. Für die Verbindung zur Datenbank wird wie bereits erwähnt die JDBC-Schnittstelle der Java-Plattform genutzt. Es wir der standard mysql-jdbc-driver verwendet. In Datasource wird die entsprechende Verbindung zum Connection-Pool des Tomcat-Application Server definiert, welcher die jdbc-connections verwaltet. Nach absetzen der jeweiligen Statements wird die Verbindung zur Datenbank über die closeDbConnection-Methode wieder getrennt und die Connection geht wieder in den Connection-Pool über und steht bereit.

4.7 Restservice-Module

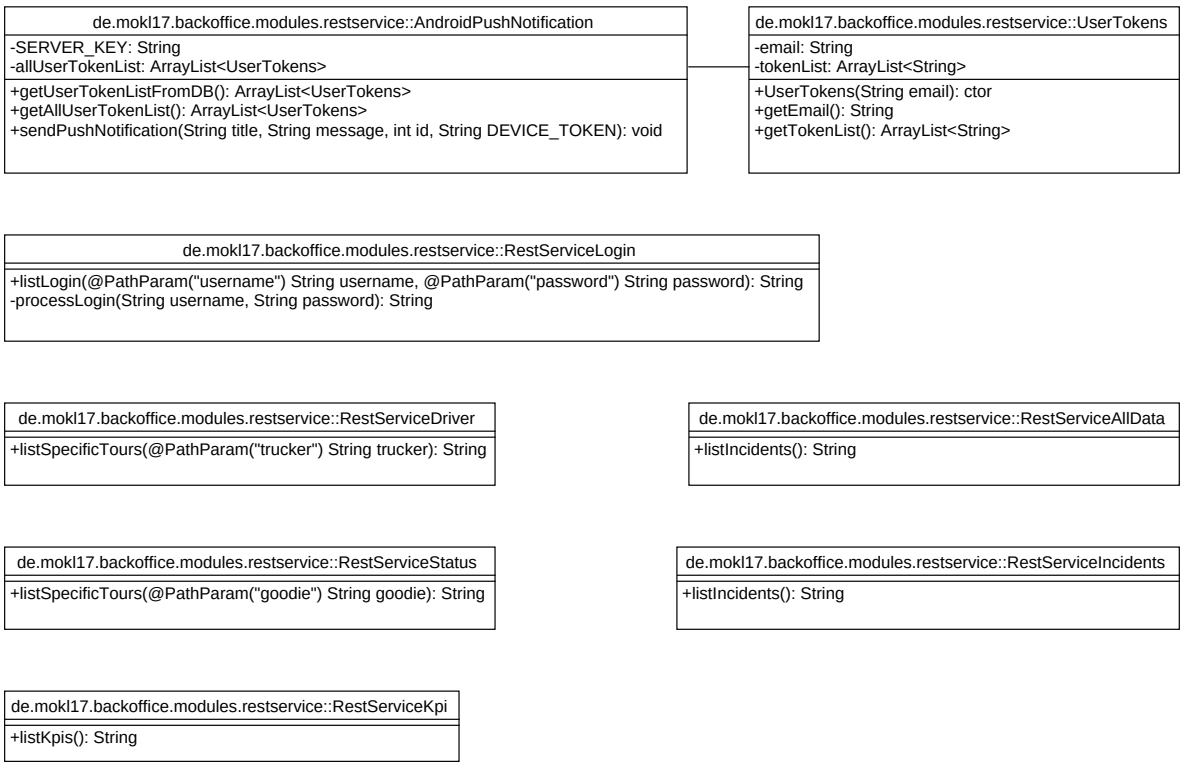

Abbildung 8: RestserviceModule

Dabei gibt es in der Backoffice-Anwendung ein Modul Restservice in dem für jedes andere Modul eine Klasse mit den entsprechend angebotenen ReST-Methoden existiert. Diese greifen auf die Daten der anderen Module zurück und bringen diese in das zur Übertragung an die Mobile App genutzte JSON-Format und stellen sie unter einem URI(7.10) bereit. Weiterhin gibt es die Klasse AndroidPushNotification, die dafür zuständig ist, Pushnachrichten an die mobile App zu senden. Die Klasse UserTokens hat dabei die Funktion, alle Tokens aller Nutzer sinnvoll in AndroidPushNotification abzuspeichern.

4.8 Mobile App

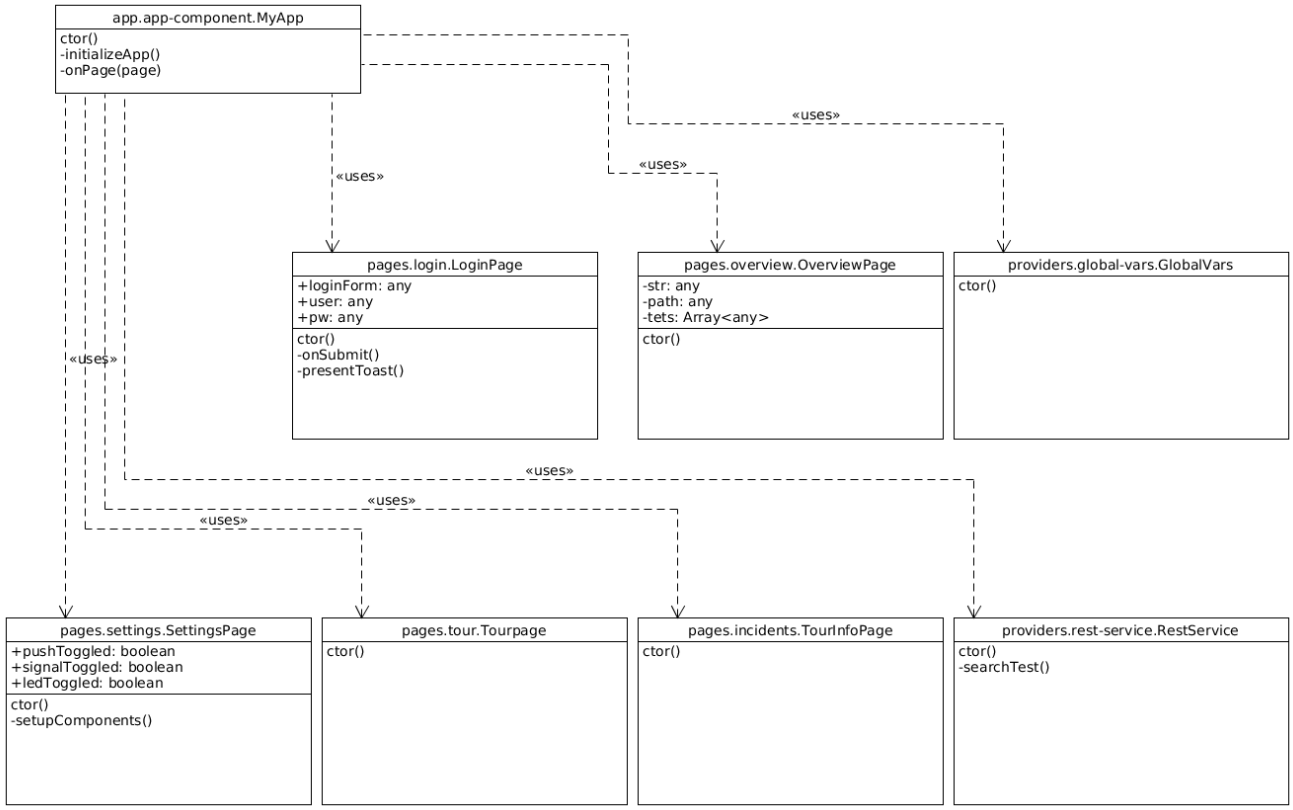

Abbildung 9: Ionic App

Die Mobile App wird mit dem ionic2-Framework entwickelt, welches sich eignet um für verschiedene Plattformen Apps zu erzeugen und dabei dieselbe Codebase zu nutzen. Dabei wird das sogenannte Page Object Pattern verwendet, welches HTML und CSS/SASS inklusive TypeScript und JavaScript wrapped. Zusätzlich wird eine RestClient-Klasse (aktuell noch RestTest) implementiert, welche für die Kommunikation zur Rest-Schnittstelle der Backoffice-Anwendung genutzt wird.

5 Datenmodell

Alle Daten sollen grundsätzlich in der Datenbank gespeichert werden, sodass die Daten für die einzelnen Modulen dauerhaft verfügbar sind bzw. auch nach einem Neustart wieder abgerufen werden können. Die Mobile App schickt als Client über die mobile App eine Anfrage an den Server. Der Server reagiert auf diese Anfrage und sendet die ausgewerteten Daten zurück an den Client. Im Hintergrund werden die Daten über die REST-API (weiter dann über das HTTP-Protokoll) zum Client übertragen.

5.1 Speicherkonzept

Im Folgenden gibt es einen Überblick über alle genutzten Tabellen in der mokl17-Datenbank der MariaDB:
In jeder Tabelle werden Daten durch einen Primärschlüssel identifiziert, welcher aber nicht zwingend unten aufgeführt ist. In den meisten Modulen werden die Daten aus der Datenbank geladen und zwischenzeitlich in eine ArrayList gespeichert. Das dient dazu, dass schnellere Zugriffszeiten ermöglicht werden und nicht immer in der Datenbank nach Daten gesucht werden muss. Der Nachteil ist allerdings, dass es auch zu Problemen kommen kann, wenn zwischenzeitlich Daten in der Datenbank bearbeitet werden.
Primärschlüssel sind unterstrichen.
Fremdschlüssel sind durch eine gestrichelte Linie markiert.

Tabelle 1: user

mail-address VARCHAR	forename VARCHAR	surname VARCHAR	password VARCHAR	user_group VARCHAR

Tabelle 2: tour

TourID INT	start_location VARCHAR	end_location VARCHAR	driver VARCHAR	status FOREIGN KEY	timestamp TIMESTAMP	deviation BOOLEAN

Tabelle 3: notification

notification_ID INT	mail FOREIGN KEY	push_notification BOOLEAN	sound_notification BOOLEAN

Tabelle 4: kpi

kpiID INT	name VARCHAR	logic VARCHAR	dependentKpi VARCHAR	value INT

5.2 Entity-Relationship Modell

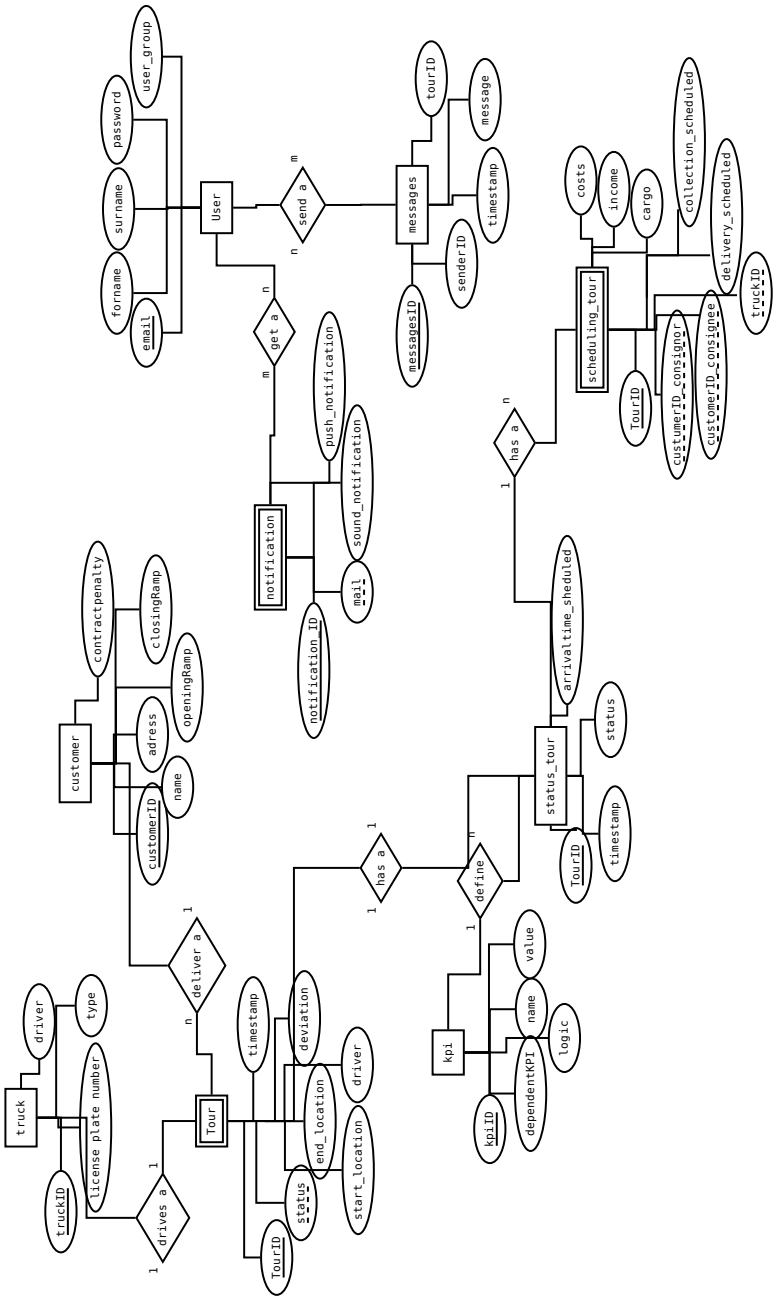

Abbildung 10: Entity-Relationship Modell

6 Benötigte Technologien und Frameworks

6.1 Ionic

Ionic ist ein Framework, das die Erstellung von plattformübergreifenden Apps (Android, iOS) unterstützt. Grundsätzlich liefert das Framework Apps nach einer MVC-Architektur (Model View Controller). Dies bietet den Vorteil, dass Verarbeitung und Design voneinander getrennt werden können.

6.2 Java

Als Programmiersprache wird Java 1.8 verwendet. Da wir uns für Vaadin (6.4) als Framework für dieses Projekt entschieden haben, ist Java unverzichtbar.

6.3 Maven

Maven ist ein Build-Tool und dient der Sicherstellung von Abhängigkeiten (Klassen, Pakete). Dies ist in der Kombination mit Vaadin unausweichlich, da viele Abhängikeiten so Plattformübergreifend abgerufen werden könnnen. Maven stellt hier eine simple Ordnerstruktur zur Verfügung (java für alle .java-Dateien, webapp für alle .html- und .css-Datein, so wie resources).

6.4 Vaadin

Vaadin ist ein freies Framework und wurde speziell für serverseitige Lösungen entwickelt. Der Einsatz dieses Frameworks bietet sich daher sehr gut an: Die Weboberfläche und App kann jeweils in Java entwickelt werden. Zudem werden viele Steuerelemente von Haus aus bereitgestellt, so dass elementare Funktionen, wie Navigationsleisten, Buttons, etc. pp. nicht selbst entwickelt werden müssen. Hier sind eine hohe Wiederverwendbarkeit und gute Portierungsmöglichkeiten (Mobile Website, App UI) gewährleistet. Vaadin bietet hier viele Elemente, die über das Tool Maven hervoragend eingebunden werden können. (siehe 6.3)

6.5 Drools

Drools ist ein Open-Source Projekt von JBoss für die Umsetzung eines Business Rule Management System (BRMS). Die Engine bietet uns die Möglichkeit Regeln flexibel, vom restlichen System getrennt, Regeln zu erstellen. Die auf Java basierende App ist leicht über Maven zu implementieren.

7 Glossar

7.1 Client-Server

Das Client-Server-Modell ist eine Möglichkeit Aufgaben und Dienste auf andere Rechner auszulagern. Dabei gibt es einen Rechner, der Server, der die Dienste bereitstellt. Ein Client stellt eine Anfrage mit der zu bearbeitenden Aufgabe an den Server. Nun verarbeitet der Server diese Anfrage und sendet dem Client die Antwort. Dies führt dazu, dass zeit- oder speicherintensive Aufgaben ausgelagert werden können.

7.2 JDBC

Java Database Connectivity: JDBC ist eine Datenbankschnittstelle der Java-Plattform, die auf relationale Datenbanken ausgelegt ist.

7.3 Kryptographische Hashfunktion

Eine kryptographische Hashfunktion ist eine Hashfunktion, die über eine komplexe Berechnungsmethode eine Zeichenkette mit fester Länge generiert. Eine solche Hashfunktion muss eindeutig (identische Zeichenfolge ergibt immer die selbe Hash-Zeichenkette), reversibel (nicht zurückrechenbar) und kollisionsresistent (zwei unterschiedliche Zeichenfolgen düfen nicht die selbe Hash-Zeichenkette ergeben).

7.4 MVC

MVC steht für Model Viev Controller. In diesem Design-Pattern wird die Programmlogik (Model) von der grafischen Benutzeroberfläche (GUI) getrennt und ist komplett unabhängig von ihr. Außerdem wird die Ansicht der GUI (View) von ihrem Verhalten (Controll) getrennt. Diese Unabhängigkeit ermöglicht, dass einzelne Teile zum Beispiel der Logik überarbeitet werden können, ohne, dass die anderen Teile von den Änderungen betroffen sind.

7.5 Prepared Statements

Eine SQL-Anweisung muss vor der Ausführung im Datenbanksystem einige Tests und Umwandlungen erfahren (Syntax, etc). Anschließend werde diese in einem internen Ausführungsplan der Datenbank übersetzt und die Transaktionen verarbeitet. Prepared Statements ermöglichen es, der Datenbank bereits im Vorfeld zu übermitteln, dass Daten gespeichert werden. Dies erhöht sowohl die Performance, als auch die Sicherheit.

7.6 ReST

Abkürzung für Representational State Transfer. Programmierparadigma für Webservices und Clouddienste. Bietet Schnittstellen (APIs) um Services online verfügbar zu machen.

7.7 SQL-Injection

SQL-Injection (engl. für SQL-Einschleusung) bezeichnet eine Sicherheitslücke bei SQL-Datenbanken. Hierbei wird ausgenutzt, dass die Überprüfung der Eingabedaten über SQL-Formulare mangelhaft ist, und darüber schadhafte SQL-Anweisungen 'eingeschleust' werden können.

7.8 DML

Data Manipulation Language: SQL Sprachelemente zur Veränderung von Daten, wie bspw. INSERT,UPDATE, DELETE !

7.9 DQL

Data Query Language: SQL Sprachelemente zur Datenabfrage, wie bspw. SELECT, WHERE, JOIN !

7.10 URI

Uniform Ressource Identifier: URIs werden zur Bezeichnung und Identifikation von Ressourcen eingesetzt