Projektbericht

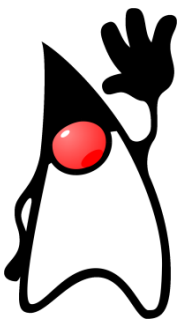
Hendrik Moschall

Fachwahlpflichtfach
Internetprogrammierung

Team HorstWare

Schwerpunkt
Smartphone-Anbindung

Hochschule **Rosenheim**
University of Applied Sciences



Inhalt

1 Planung und Initialisierung

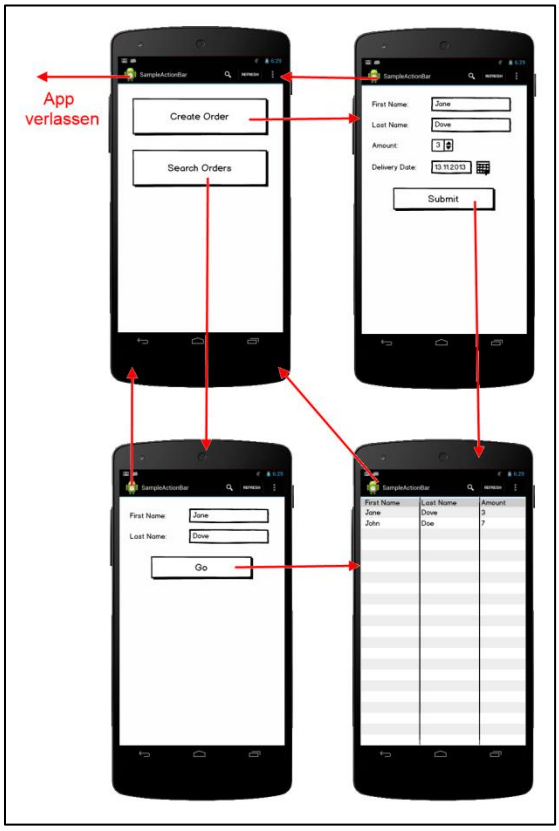
1.1 Einrichten des Repositories bei GitHub

Als erstes haben wir uns auf GitHub, ein Hosting-Dienst für die verteilte Versionsverwaltung Git, ein Repository erstellt. Für dieses musste man dort einen Account erstellen und den zugehörigen Desktop-Client installieren.

1.2 Erstellen der Projektbeschreibung

Die folgende Arbeit bestand darin die Projektbeschreibung zu schreiben, wofür ich als erstes das Dokument mit Titelseite, Kopf- und Fußzeilen, Zusammenfassung, Historie und Inhaltsverzeichnis mit Gliederung erstellt.

Inhaltlich war ich für die Erstellung der GUI-Mockups und der GUI-Landkarte für die Android Client-App verantwortlich, welche eine erste Version der Benutzeroberfläche darstellen sollte:



Des Weiteren habe ich eine kurze Beschreibung des Projektschwerpunktes geschrieben.

Zuletzt haben wir gemeinsam aus den einzelnen Kapiteln die Version für den Projektstart am 21.11.2013 zusammengestellt. Diese wurde im Anschluss von Herrn Schneider und mir noch etwas überarbeitet. Die aktuelle Version habe ich damals Herrn Peröbner als PDF per E-Mail gesendet.

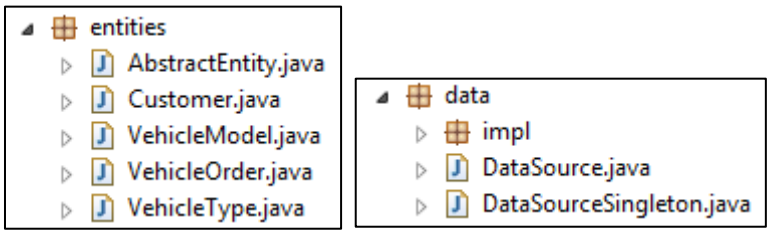
2 Implementierung

2.1 Erstellen der Client-App

Meine Hauptaufgabe in den Wochen vor Weihnachten und über die Feiertage war das Erstellen der Android Client-App.

2.1.1 Übersicht

Die Datenschicht besteht aus dem Paket **entities** und dem Paket **data**.:

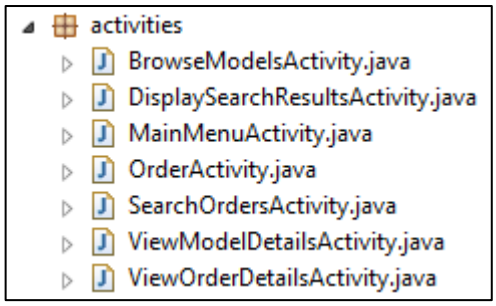


Die Entitäten spiegeln Die aus der Server-Anwendung wieder und leiten alle von **AbstractEntity** ab, welche eine ID vererbt. **Datasource** ist die Schnittstelle, welche letztendlich durch die von Herrn Schneider programmiert REST-Serviceanbindung implementiert wird. Das **DataSourceSingleton** entkoppelt die Instanzierung der **DataSource** von deren Verwendung.

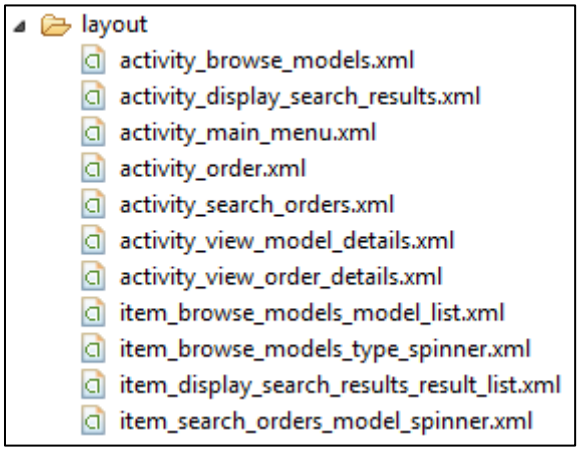
```
public interface DataSource
{
    public VehicleType retrieveVehicleType (long id);
    public List<VehicleType> retrieveVehicleTypes ();

    public VehicleModel retrieveVehicleModel (long id);
    public List<VehicleModel> retrieveVehicleModels ();
    public List<VehicleModel> retrieveVehicleModels (VehicleType filterByType);

    public Customer createCustomer (String firstName, String lastName);
    public Customer retrieveCustomer (long id);
    public List<Customer> retrieveCustomers ();
    public List<Customer> findCustomers (String firstName, String lastName);

    public VehicleOrder createVehicleOrder (Customer customer, VehicleModel model,
    public VehicleOrder retrieveVehicleOrder (long id);
    public List<VehicleOrder> retrieveVehicleOrders ();
    public List<VehicleOrder> retrieveVehicleOrders (String filterByCustomersFirstN
}
```

Abbildung 1: DataSource Schnittstelle

Die Programmlogik wird in Android in erster Linie durch sogenannte Activities umgesetzt, welche in der Regel mit einer Benutzeroberfläche korrespondieren. Diese sind im Paket **activities** enthalten:



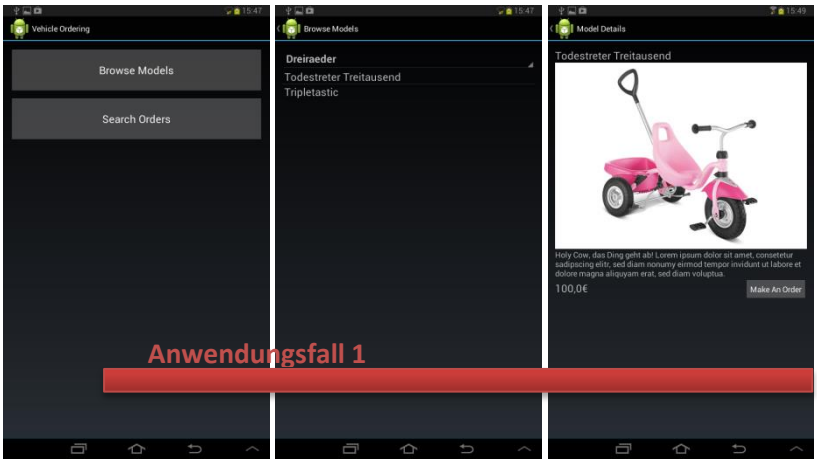
Die Darstellung wird in Form von in XML deklarierten **Layouts** umgesetzt. Dabei können Activities über IDs auf die einzelnen Elemente, z.B. einem Button, der ihnen zugeordneten Layouts zugreifen. Diese XML-Dateien finden sich im Projektverzeichnis „res/layout“:



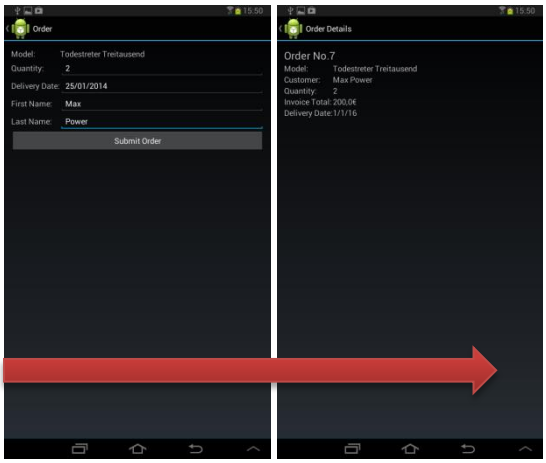
Die App auf diesem Stand findet man als „Android Client App Prototyp“ auf der CD. Er hat volle Funktionalität, bezieht seine Daten jedoch aus einem lokalen **DataSource**-Mockup und noch nicht aus der im weitern Verlauf integrierten REST-Anbindung.

2.1.2 Programmablauf

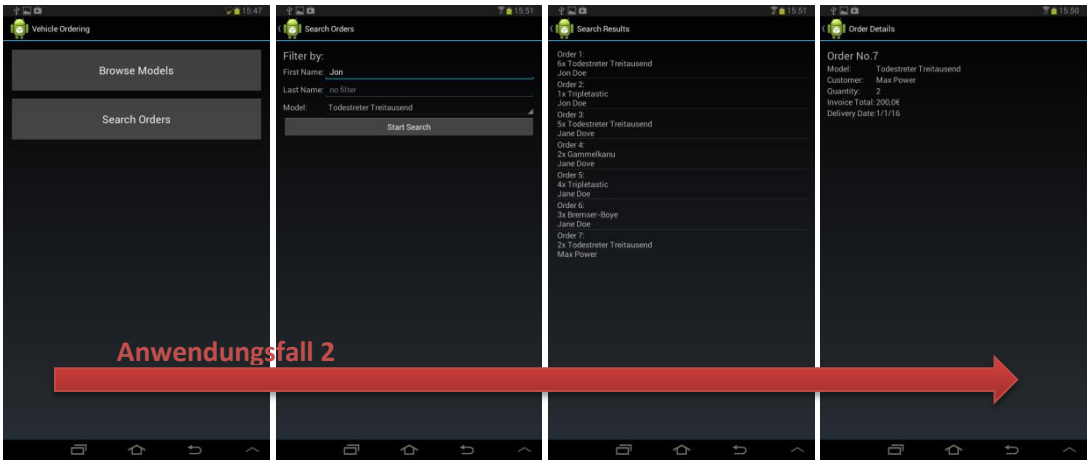
Der erste Anwendungsfall ist das browsen der angebotenen Vehikel und deren Bestellung. Er wird im Hauptmenü durch betätigen von ‚Browse Models‘ gestartet. Anschließen kann in einem Dropdownmenü der Typ (z.B. Dreiräder) gewählt und in einer dann erscheinenden Liste das Model (z.B. Todestreter Treitausend) ausgesucht werden. Durch das Klicken auf einen Listeneintrag gelangt man in eine Detailansicht mit dem Namen, Beschreibung, Preis und, falls eine Internetverbindung besteht, Bild des Models.



Drückt man unten in der Detailansicht auf „Make An Order“ kann man das gewählte Model bestellen. Dazu muss man in der nun erscheinenden Maske Anzahl, Lieferdatum und Name angeben und mit „Submit Order“ bestätigen. Die Eingaben werden validiert und bei erfolgreicher Bestellung werden dessen Details in einer abschließenden Ansicht präsentiert.



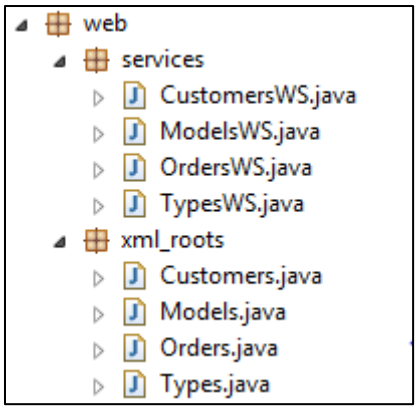
Der zweite Anwendungsfall ist das Suchen nach bereits eingegangenen Bestellungen. Dazu wählt man im Hauptmenü „Search Orders“. In der folgenden Maske kann angegeben werden nach welchen Kriterien (Name oder Model) die Bestellungen gefiltert werden soll. Gibt man keine Filter an, werden alle Bestellungen in den Suchergebnissen aufgelistet. Startet man die Suche durch klicken auf „Start Search“ werden die Suchergebnisse in der folgenden Ansicht aufgelistet. Wählt man einen Listeneintrag wird eine Detailansicht über die gewählte Bestellung angezeigt.



2.2 Erstellung der JEE Server-Anwendung

Um Herrn Schneider eine schnelle Möglichkeit zu geben seine REST-Anbindung zu programmieren, musste der Server, insbesondere die REST-Schnittstelle, frühzeitig implementiert werden. Dazu habe ich den Business-Backend des Servers um die Entitäten Type und Model sowie zugehörige DAOs erweitert.

Danach habe ich dann direkt die REST-Schnittstelle definiert und implementiert.



ID	Methode	URL
1	GET	vehicle-ordering/services/types
2	GET	vehicle-ordering/services/types/id
3	GET	vehicle-ordering/services/models?type=id
4	GET	vehicle-ordering/services/models/id
5	GET	vehicle-ordering/services/customers?first=name&last=name
6	GET	vehicle-ordering/services/customers/id
7	GET	vehicle-ordering/services/orders?first=name&last=name&model=id
8	GET	vehicle-ordering/services/orders/id
9	POST	vehicle-ordering/services/customers
10	POST	vehicle-ordering/services/orders

Diese Implementierung des Servers war rudimentär und besitzt deswegen keine BA- oder BS-Klassen, Fehlerbehandlung, neue oder erweiterte JSFs, weitere Dokumentation usw. Die aktuelle, vollständige Version ist das Ergebnis von Herrn Wörndls Arbeit.

2.3 Integration der REST-Anbindung

Als letzte Aufgabe im Rahmen der Implementierung musste noch die REST-Anbindung integriert werden. Diese gestaltete sich schwieriger als erwartet, da ich ursprünglich nicht daran gedacht habe, dass die REST-Aufrufe asynchron ablaufen.

Unser erster Ansatz, um weiterhin die **DataSource**-Schnittstelle nutzen zu können, war es die Aufrufe in einer Zwischenschicht wieder zu synchronisieren. Dies ist zwar nicht ideal, hätte uns aber viel Arbeit an anderer Stelle gespart. Letztendlich wurde dieser Ansatz jedoch aufgegeben, da sämtliche Bemühung die REST-Aufrufe zu synchronisieren vergebens waren - selbst triviale Ansätze wie Busy-Waiting scheiterten. Wir denken dies lag am fehlenden Verständnis für die Android-Umgebung und hätten wir mehr Zeit gehabt, wären wir vielleicht sogar Erfolgreich gewesen.

Letztendlich beschlossen wir, um keine weitere Zeit zu verlieren, eine asynchrone Schnittstelle zu verwenden. Diese besteht aus den Klassen **VehicleOrderingAPI** und **OnRestResponse** im Paket **network**. Ich habe dafür nochmal sämtliche Activities zügig überarbeitet, da die Fertigstellung Priorität hatte.

```
public static VehicleOrderingAPI getInstance() {
public void getOrders (OnRestResponse listener) {
public void getOrders (String customerFirstName, String customerLastName, Long id, OnRestResponse listener) {
public void getOrder (long id, OnRestResponse listener) {
public void getCustomers(OnRestResponse listener) {
public void getCustomers(String firstName, String lastName, OnRestResponse listener) {
public void getCustomer (long id, OnRestResponse listener) {
public void getModels (OnRestResponse listener) {
public void getModels (Long typeId, OnRestResponse listener) {
public void getModel (long id, OnRestResponse listener) {
public void getTypes (OnRestResponse listener) {
public void getType (long id, OnRestResponse listener) {
```

Leider konnten wir die REST-Schnittstelle nicht vollständig implementieren, weswegen es in der aktuellen Version nicht möglich ist neue Bestellungen anzulegen. Dies liegt daran, dass zwar neue Kunden und Bestellungen am Server angelegt werden können, der REST-Client aber nicht mitgeteilt bekommt welche dies sind. Konkret müsste der Server die ID des neuen Kunden bzw. Bestellung als Bestätigung in der Response zurückschicken.

3. Abschluss

Zum Ende des Projektes musste noch dessen Präsentation am 16.01.2014 vorbereitet, dieser Projektbericht geschrieben und der Inhalt der CD zusammengestellt werden, welche nach der Präsentation abgegeben werden soll.

Für die Präsentation habe ich die erste Version der Folien erstellt, welche wir anschließend gemeinsam vervollständigt und deren Vortragung abgesprochen haben. Ebenso musste überlegt werden, wie wir die Software am besten vorführen.

Für die CD habe ich die benötigten Dateien aus dem Repository herausgesucht, die Projektberichte gesammelt und letztendlich alles auf CD gebrannt.