Mobilitätsprofil
Implementierung

Tetiana Lavrynovych
Thomas Röhl

https://github.com/ThomasRoehl/DBChallange.git

Projektaufbau

Das Android-Projekt ist in zwei Bereiche Unterteilt:
Der erste Teil besteht aus der graphischen Schnittstelle die im xml-Format definiert wird.
Den zweite Teil stellt der Backend-Teil der Anwendung dar, geschrieben in Java.

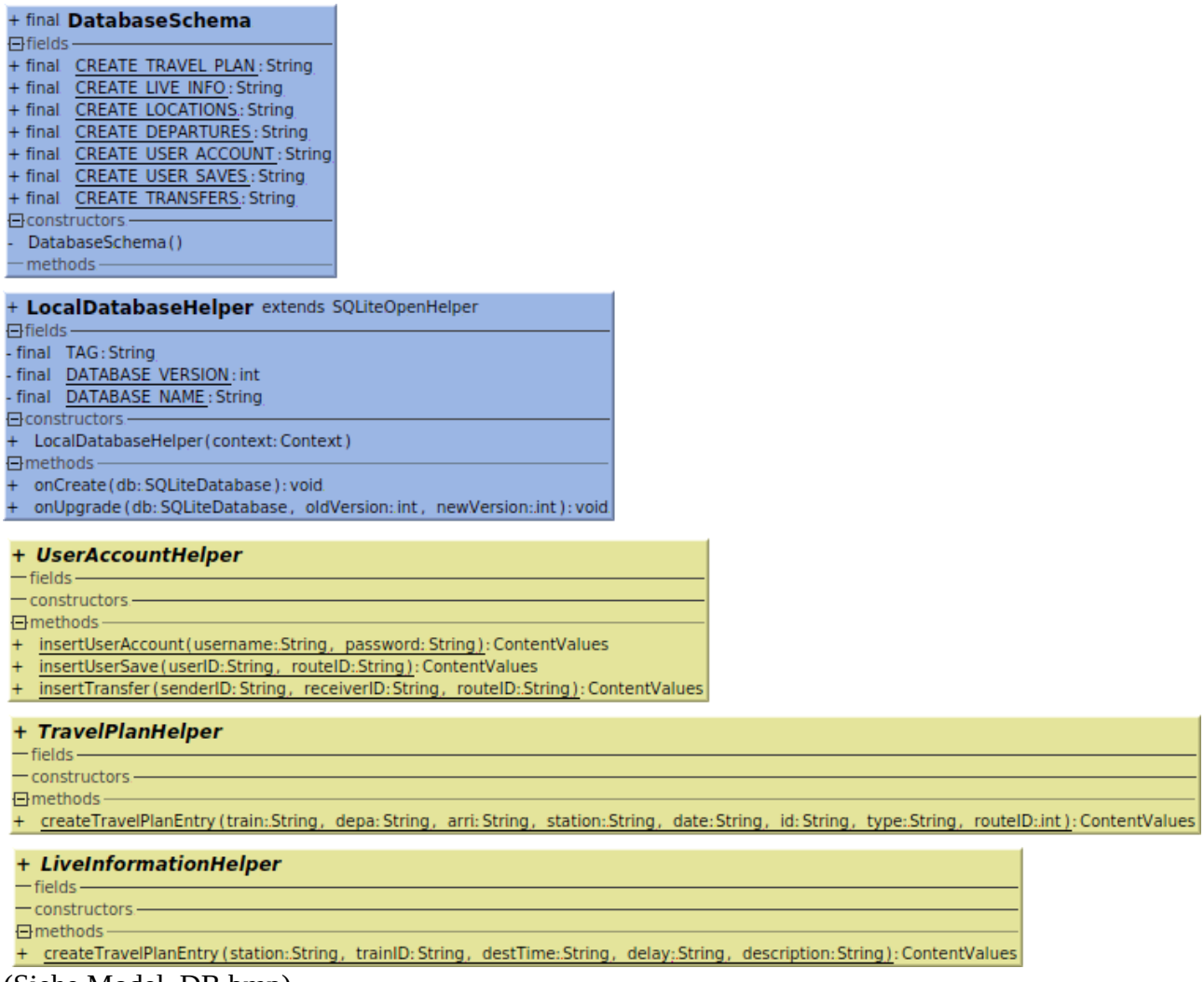
GUI

Die grafische Nutzerschnittstelle ist in Android im XML-Format gespeichert und wird in Java mit sogenannten Activities angesprochen (Siehe Backend:View).
Die XML-Dateien werden in Android in einem Ressourcen Ordner abgelegt.
Zu jeder Activity gehört eine XML-Datei, die alle Elemente der GUI beinhaltet.
Zusätzlich wurden einige Dateien erstellt um Inhalte wie Tabellen auszulagern und besser zu modifizieren.

Backend

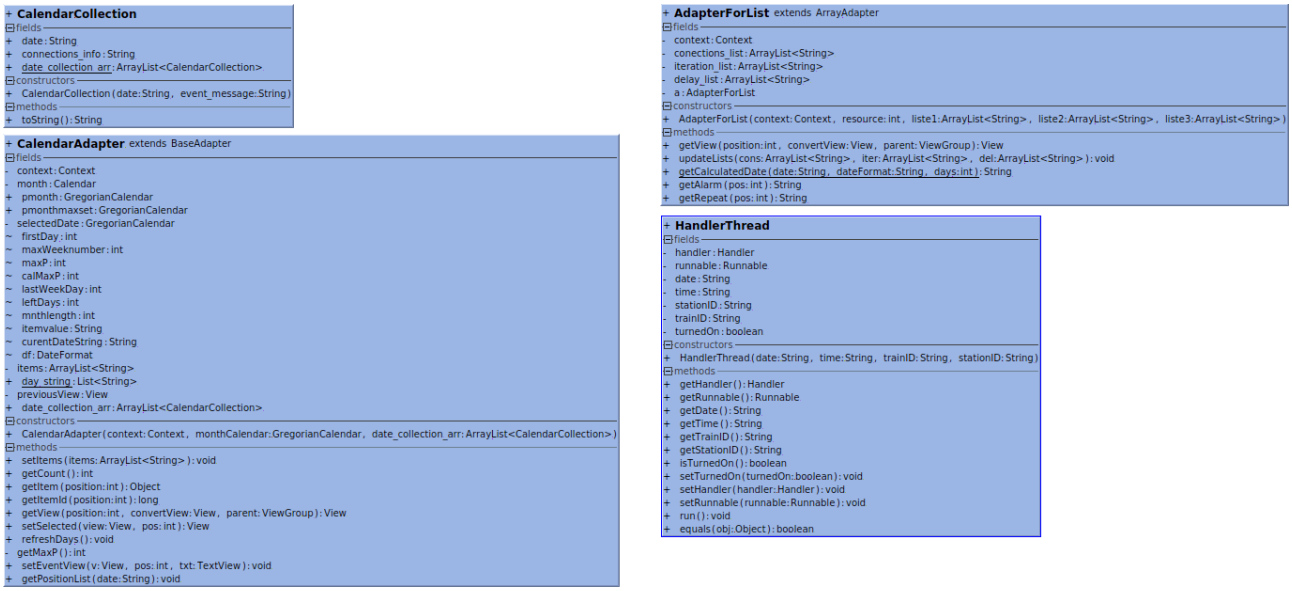
Für den Aufbau des Projekts wurde zur Strukturierung das “Model View Controller” Schema verwendet.
Daher ist das Projekt in vier Unterordner aufgeteilt, Model, View, Controller und ein Verzeichnis für die Testumgebung.

Model:



(Siehe Model_DB.bmp)

Im Model-Verzeichnis befindet sich das Java-Objektabbild der internen Datenbank, die im Kapitel Daten weiter beschrieben sind.
Dazu gibt es je eine Klasse zum Aufbau des Datenbankschemas, der Verwaltung der Datenbanktabellen und eine Hilfsklasse für die Verwaltung der Datenbank (“LocalDatabaseHelper”).

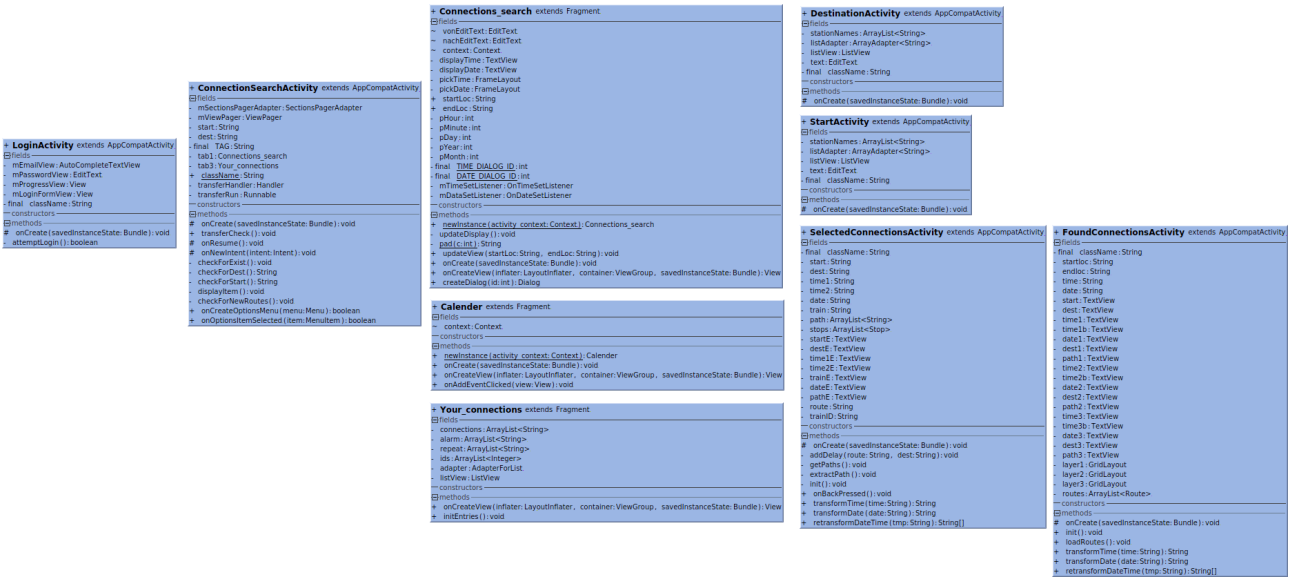


(Siehe Model_Adapter.bmp)

Als Abbild des Kalenders und der tabellarischen Auflistung der gespeicherten Verbindungen werden Adapterklassen benötigt ein “AdapterForList” und ein “CalendarAdapter” sowie eine modellarischen Darstellung des Kalenders in der “CalendarCollection” Klasse.

Um Lifeinformationen abfragen zu können werden in Android sogenannte Handler verwendet. Diese Handler werden durch “HandlerThread” Objekte verwaltet.

View:



(Siehe View-UML.bmp)

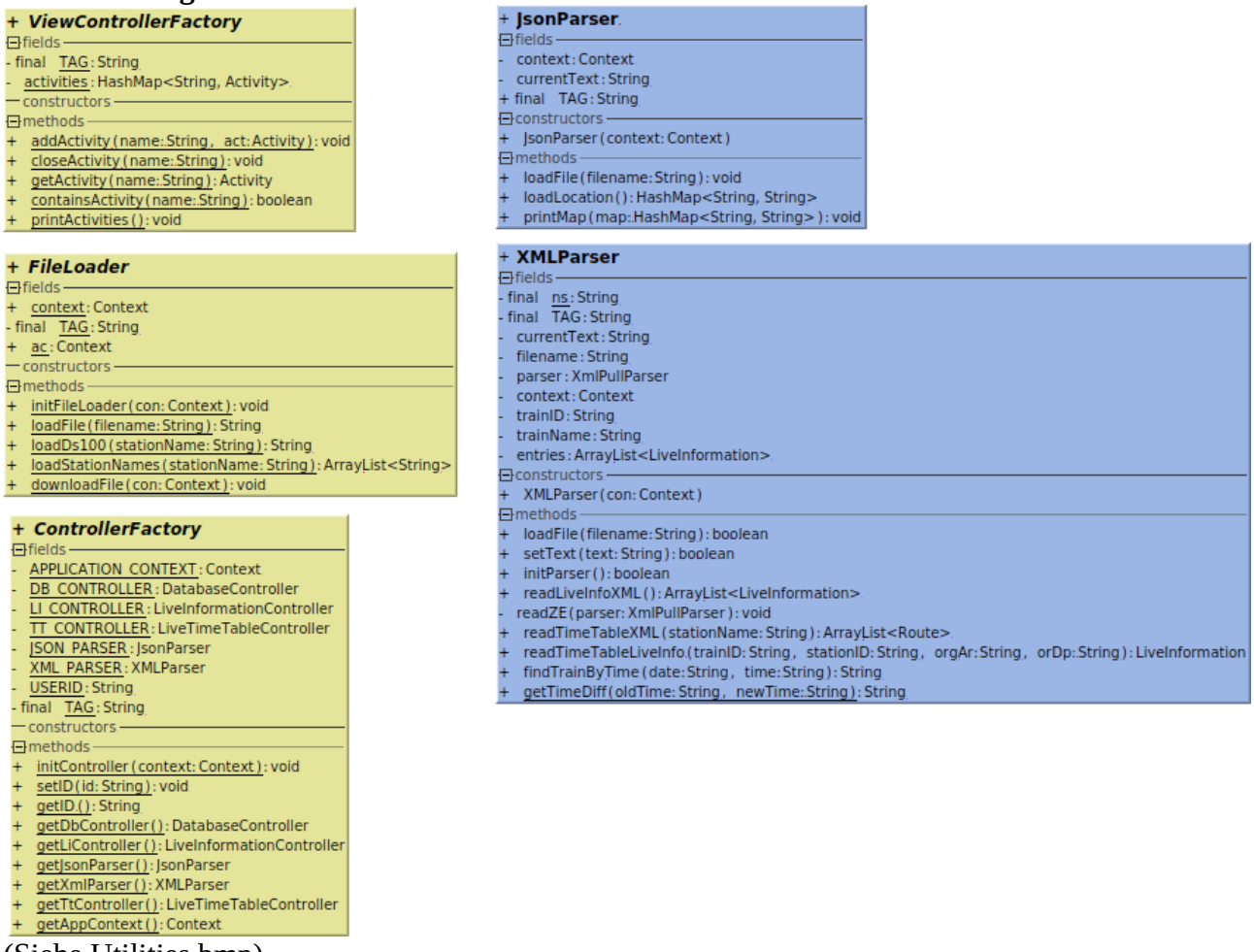
Die View Klassen sind in dieser Abbildung dargestellt in Form einer zeitlichen Abfolge.
Die Bedienung ist im Dokument Bedienungsanleitung beschrieben.
Die “LoginActivity” ist das Abbild des Login und Register Fensters und leitet den Nutzer weiter in

die zentrale “ConnectionSearchActivity”.
Die “ConnectionSearchActivity” besteht aus drei Fragmenten dem “ConnectionSearch”, “Calendar” und “Your_Connections” Fragment.
Diese drei Fragmente repräsentieren die drei Tabs des Hauptfensters.

Das erste Fenster dient zur Suche einer neuen Verbindung. Dazu kann der Start- bzw. Zielort festgelegt werden, der in den Fenstern “StartActivity” und “DestinationActivity” herausgesucht werden kann.
Die verschiedenen möglichen Verbindungen werden in der “SelectedConnectionsActivity” angezeigt. Der Nutzer hat hier die Möglichkeit weitere Details zu einer Verbindung zu betrachten und wird dabei weiter auf die “FoundConnectionsActivity” geleitet.
Hierbei werden alle Informationen aus den Online Zeittafeln (“http://iris.noncd.db.de/wbt/js/index.html?bhf=FF”) geladen und dargestellt.
Wenn die gewünschte Verbindung ausgewählt wurde, wird der Nutzer weiter auf das “Your_Connection” Fragment geleitet.

Controller:
Der Controllerbereich besteht aus zwei Teilen, den allgemeinen Unterstützungsklassen (“ViewControllerFactory”, “ControllerFactory”, “FileLoader”, “JsonParser” und “XMLParser”) und den einzelnen Controller-Klassen der Activities (“DatabaseController”, “LiveInformationController”, “LiveTimeTableController”).

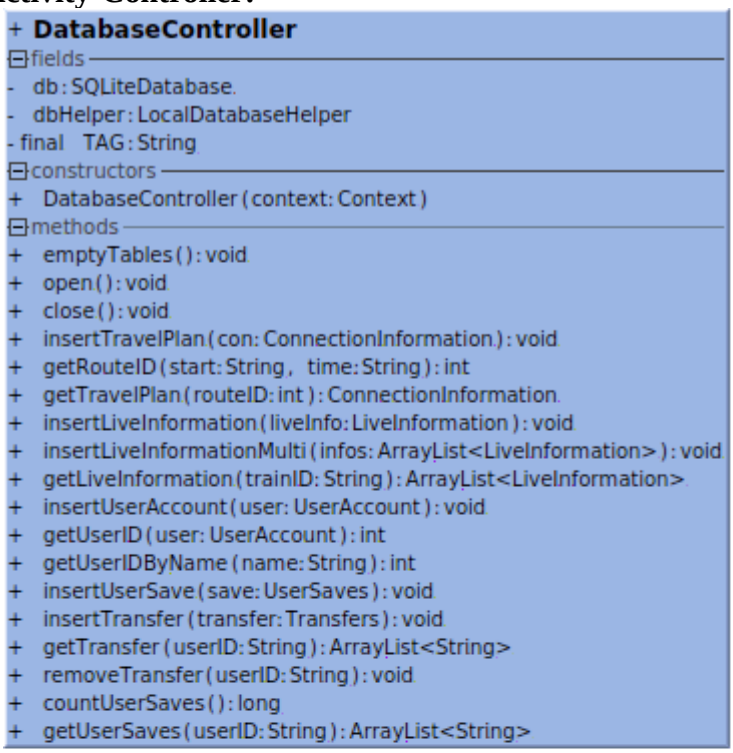
Unterstützungsklassen

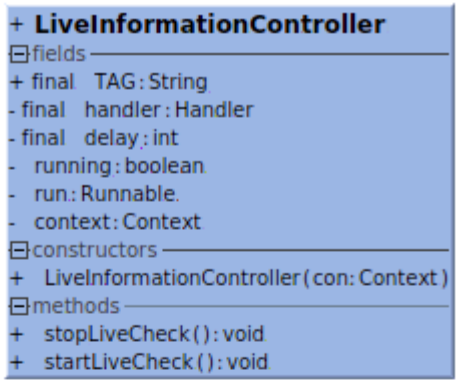


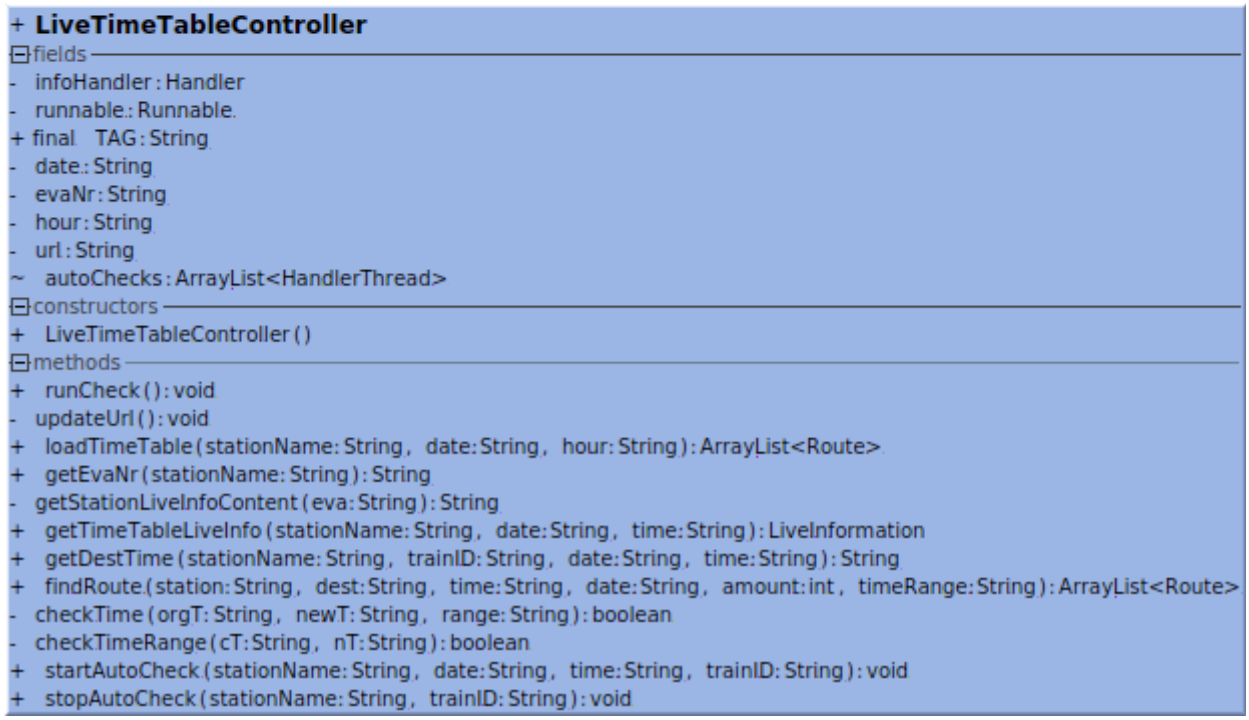
(Siehe Utilities.bmp)

Die Klassen “ViewControllerFactory” und “ControllerFactory” sind statische Klassen, die alle Informationen bereitlegen, die während der Laufzeit der Applikation benötigt werden.
Dazu gehören unter Anderem der “Context” der Applikation, der für verschiedene grafische Prozesse benötigt wird, und einer Liste über alle aktiven Fenster.
Mit dem FileLoader werden Daten aus dem internen Speicher geladen.
Zuletzt gibt es noch zwei Parser um die verschiedenen Onlinedaten zu extrahieren aus den Formaten Json und XML.

Activity-Controller:





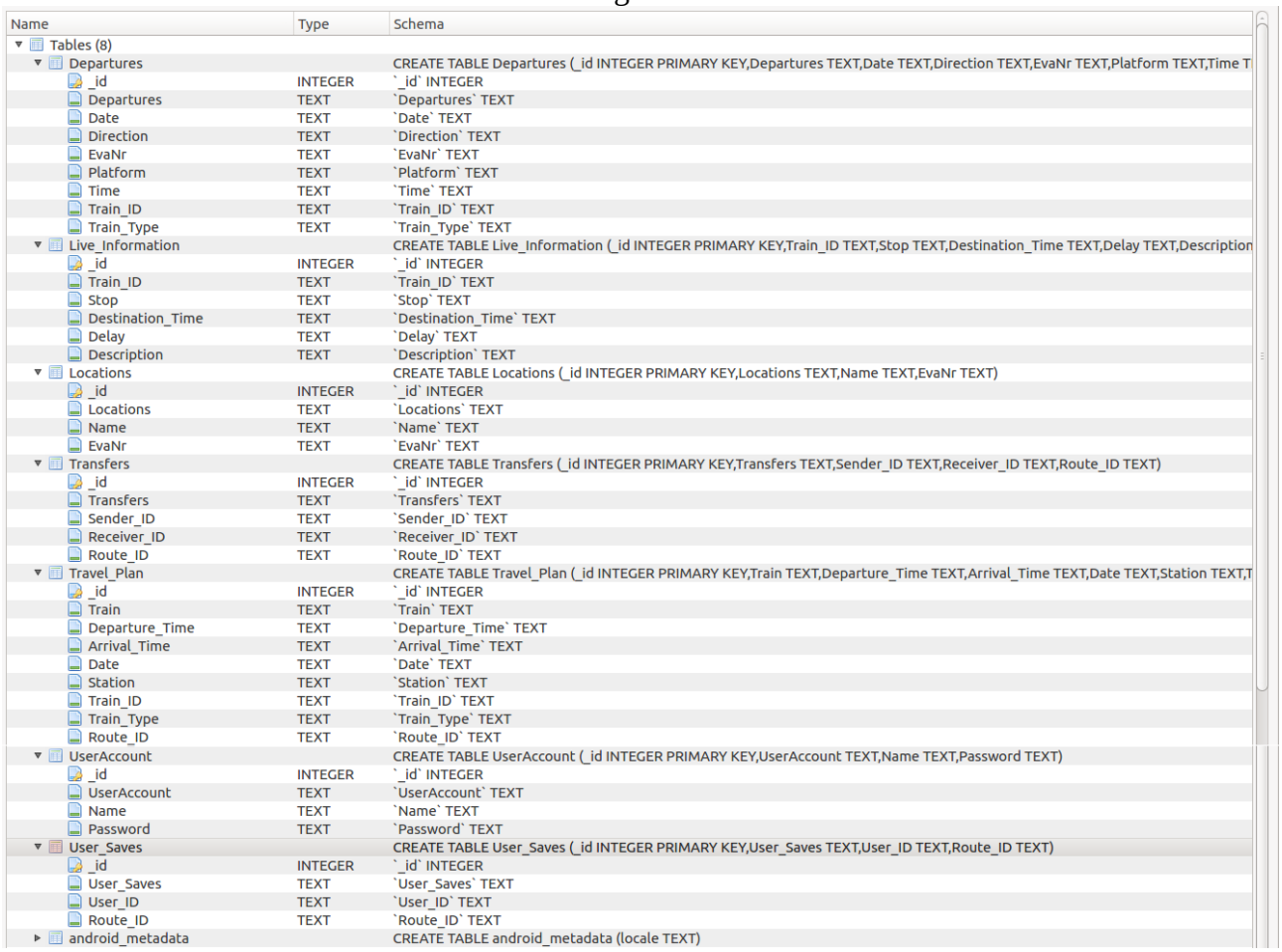


(Siehe ControllerActivities)

Der “DatabaseController” stellt die Schnittstelle zur internen Datenbank dar.
Er ermöglicht alle Datenbankzugriffe zur Laufzeit der Applikation.
Der “LiveInformationController” bietet die Möglichkeit zur Aktivierung und Deaktivierung der Onlineabfrage zu aktuellen Verspätungen der gespeicherten Verbindungen.
Der “LiveTimeTableController” stellt die Schnittstelle zur Online Zeittafel dar und bietet alle Funktionen zur Abfrage der aktuellen Daten.

Daten

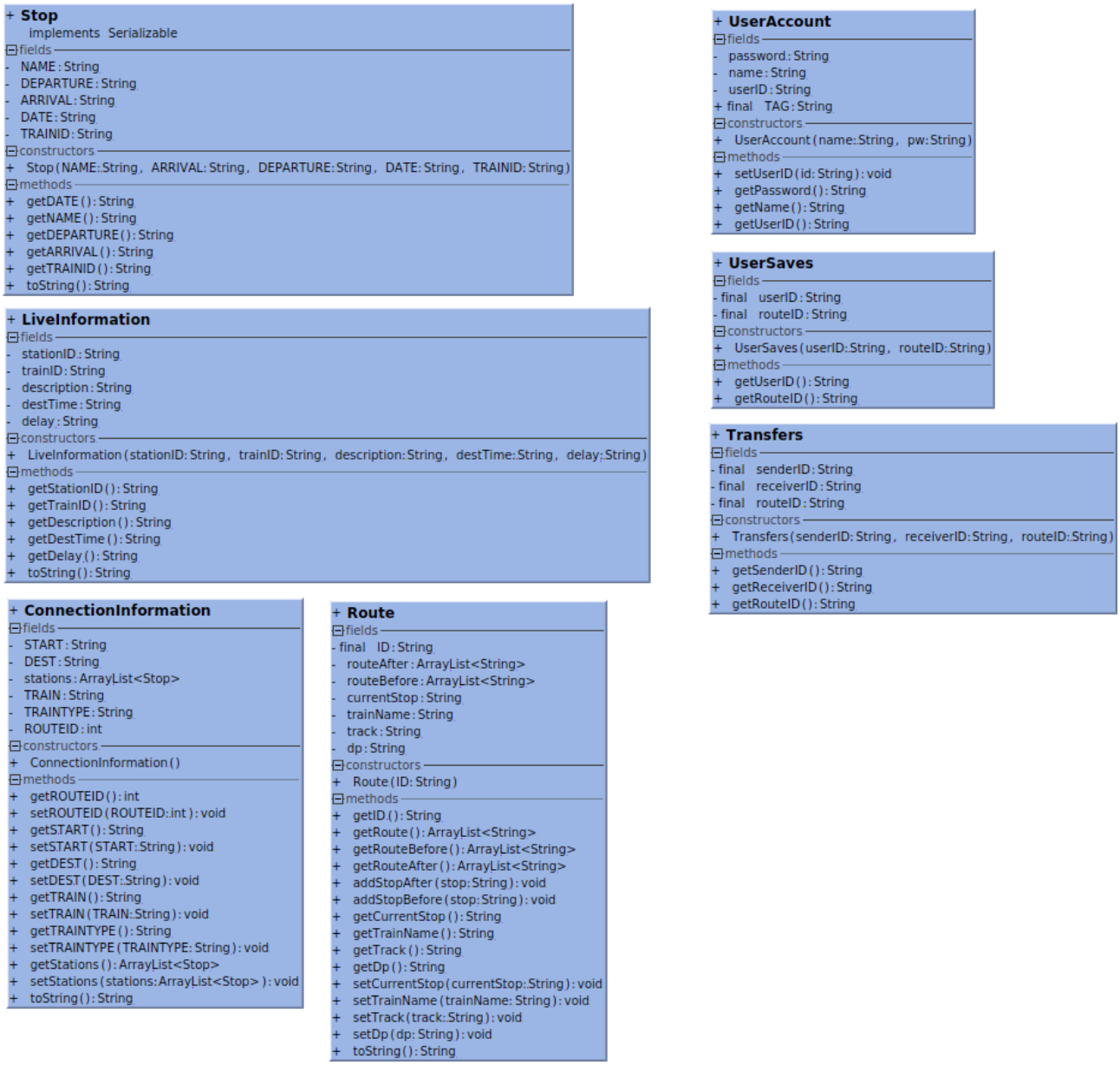
In diesem Projekt werden Daten in einer internen Datenbank oder in Dateien im internen Speicher abgelegt.
Die Datenbank besteht aus 7 Tabellen die alle Informationen zu Verbindungen, Stationen und Nutzern beinhaltet.
Der Aufbau der Datenbanktabellen ist in den folgenden Grafiken beschrieben:



Name	Type	Schema
Tables (8)		
Departures		CREATE TABLE Departures (_id INTEGER PRIMARY KEY,Departures TEXT,Date TEXT,Direction TEXT,EvaNr TEXT,Platform TEXT,Time T
_id	INTEGER	`_id` INTEGER
Departures	TEXT	`Departures` TEXT
Date	TEXT	`Date` TEXT
Direction	TEXT	`Direction` TEXT
EvaNr	TEXT	`EvaNr` TEXT
Platform	TEXT	`Platform` TEXT
Time	TEXT	`Time` TEXT
Train_ID	TEXT	`Train_ID` TEXT
Train_Type	TEXT	`Train_Type` TEXT
Live_Information		CREATE TABLE Live_Information (_id INTEGER PRIMARY KEY,Train_ID TEXT,Stop TEXT,Destination_Time TEXT,Delay TEXT,Description
_id	INTEGER	`_id` INTEGER
Train_ID	TEXT	`Train_ID` TEXT
Stop	TEXT	`Stop` TEXT
Destination_Time	TEXT	`Destination_Time` TEXT
Delay	TEXT	`Delay` TEXT
Description	TEXT	`Description` TEXT
Locations		CREATE TABLE Locations (_id INTEGER PRIMARY KEY,Locations TEXT,Name TEXT,EvaNr TEXT)
_id	INTEGER	`_id` INTEGER
Locations	TEXT	`Locations` TEXT
Name	TEXT	`Name` TEXT
EvaNr	TEXT	`EvaNr` TEXT
Transfers		CREATE TABLE Transfers (_id INTEGER PRIMARY KEY,Transfers TEXT,Sender_ID TEXT,Receiver_ID TEXT,Route_ID TEXT)
_id	INTEGER	`_id` INTEGER
Transfers	TEXT	`Transfers` TEXT
Sender_ID	TEXT	`Sender_ID` TEXT
Receiver_ID	TEXT	`Receiver_ID` TEXT
Route_ID	TEXT	`Route_ID` TEXT
Travel_Plan		CREATE TABLE Travel_Plan (_id INTEGER PRIMARY KEY,Train TEXT,Departure_Time TEXT,Arrival_Time TEXT,Date TEXT,Station TEXT,T
_id	INTEGER	`_id` INTEGER
Train	TEXT	`Train` TEXT
Departure_Time	TEXT	`Departure_Time` TEXT
Arrival_Time	TEXT	`Arrival_Time` TEXT
Date	TEXT	`Date` TEXT
Station	TEXT	`Station` TEXT
Train_ID	TEXT	`Train_ID` TEXT
Train_Type	TEXT	`Train_Type` TEXT
Route_ID	TEXT	`Route_ID` TEXT
UserAccount		CREATE TABLE UserAccount (_id INTEGER PRIMARY KEY,UserAccount TEXT,Name TEXT,Password TEXT)
_id	INTEGER	`_id` INTEGER
UserAccount	TEXT	`UserAccount` TEXT
Name	TEXT	`Name` TEXT
Password	TEXT	`Password` TEXT
User_Saves		CREATE TABLE User_Saves (_id INTEGER PRIMARY KEY,User_Saves TEXT,User_ID TEXT,Route_ID TEXT)
_id	INTEGER	`_id` INTEGER
User_Saves	TEXT	`User_Saves` TEXT
User_ID	TEXT	`User_ID` TEXT
Route_ID	TEXT	`Route_ID` TEXT
android_metadata		CREATE TABLE android_metadata (locale TEXT)

(Siehe DB.png)

Um die Informationen aus der Datenbank auf Javaseite nutzen zu können werden folgende Java Objekte verwendet:



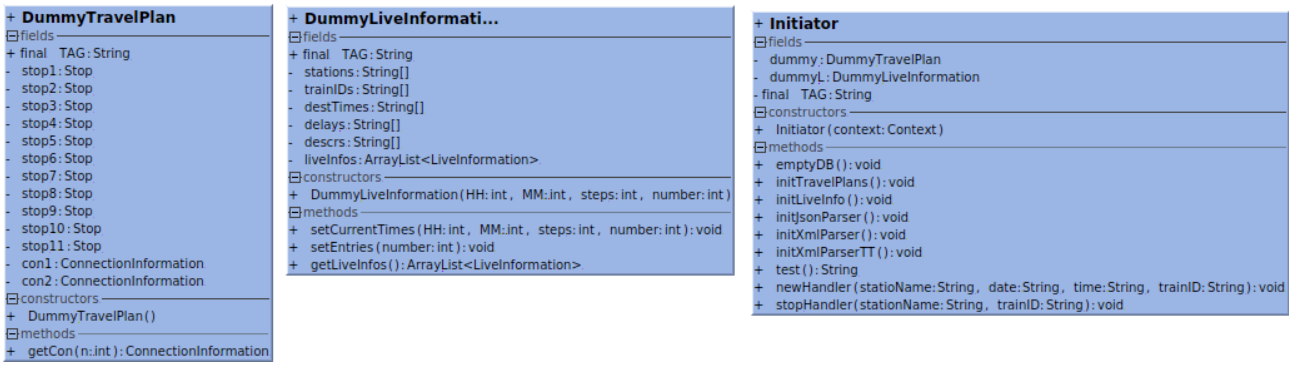
(Siehe DBObjects.bmp)

Das “Stops” Objekt beinhaltet Informationen über eine Haltestelle.
In “ConnectionInformationen” wird eine Verbindung gespeichert, die alle Informationen für die Darstellung in der Applikation beinhaltet.
Das “Route” Objekt stellt ähnlich zu der “ConnectionInformation” eine Verbindung dar, das für das speichern von Verbindungen aus den Online Zeittafeln verwendet wird.
Um aktuelle Informationen zu den gespeicherten Verbindungen zu speichern, wird das “Live-Information” Objekt verwendet.

Um Nutzerinformationen zu speichern werden die “UserAccount” Objekte verwendet.
Für Verbindungen, die vom Nutzer gespeichert werden, wird ein “UserSaves” Objekt verwendet.
Die Möglichkeit Verbindungen an andere Nutzer zu senden werden in “Transfers” Objekten gespeichert.

Tests

Zum Testen wurden verschiedene Testdaten generiert, die in den Klassen “DummyTravelPlan” und “DummyLiveInfromation” vorbereitet wurden.
Alle Testfunktionen sind in der Controllerklasse “Initiator” definiert und wurden im Anwendertest untersucht.



(Siehe Testing.bmp)

Mit diesen Tests wurde die Funktionalität evaluiert und eine Liste von bisher ungelösten Problemen erstellt, die im Kapitel Probleme erläutert werden.

Externe Daten

Im Rahmen der Entwicklung der Applikation war es nötig aus verschiedenen Quellen Daten zu extrahieren. Dazu wurden Parser für die folgenden Formate entwickelt:
Daten aus dem OpenDate Portal der Deutschen Bahn (“http://data.deutschebahn.com/dataset/api-fahrplan”) werden im Json Format versendet. Dazu wurde ein Json Parser entwickelt, der die Json Dateien in die entsprechende Datenbank Objekte konvertiert.
Die Life-Information der Deutschen Bahn sind im XML-Format gespeichert. Dazu wurde ein XML-Parser für die entsprechenden Einträge entwickelt.
Für Echtzeitinformationen wurden vor allem Daten aus den Online Zeittafeln der Bahnhöfe extrahiert. Diese Daten sind ebenfalls im XML-FORMAT gespeichert wofür auch ein Parser entwickelt wurde, der die Informationen in die Datenbank Objekte umwandelt.
Um die Daten zu erhalten wurde zusätzlich ein Schnittstelle für den Online-Zugriff auf den Webservice der Zeittafeln implementiert, der den Download der Informationen ermöglicht.
Zuletzt wurde noch ein dritter Anbieter von Informationen verwendet um die sogenannten DS100 Schlüssel der Haltestellen zu erhalten (“http://www.bahnseite.de/DS100/DS100_main.html”).

Probleme

Da die Daten des OpenDate Portals der Deutschen Bahn über keine Nähverkehrsstationen verfügen, war es nötig die Stationen aus den Online Zeittafeln herauszuziehen.
Ein Problem bei der Entwicklung waren die unterschiedliche Namensgebung der Stationen.
Da es teilweise notwendig war Stationen über den Namen zu identifizieren, um zum Beispiel die Verbindung der Online Zeittafeln zu analysieren, kam es zu Fehlern bei dem Namensvergleich.
Da die Online Zeittafel über die DS100 Schlüssel aufgerufen werden muss um die ID der Haltestelle zu erhalten war es nötig aus einer externen Quelle (Siehe externer Datenzugriff) eine Liste der DS100 Schlüssel zu extrahieren.
Aufgrund der unterschiedlichen Benennung kann nicht garantiert werden, dass alle Verbindungen korrekt erkannt werden.
Getestet wurde die Anwendung ausführlich auf der Strecke zwischen Hanau Hbf und Frankfurt (M) Hbf (tief).

Zukunft

Aus Performance-Gründen wäre es hilfreich auch Nähverkehrstationen in der OpenDate Datenbank der Deutschen Bahn zu erhalten, da ansonsten alle Verbindungen in einem definierten Zeitraum geladen werden müssen, um eine spezielle Verbindung zu finden.

Aus zeitlichen Gründen war es nicht mehr möglich Life-Information als Push-Nachrichten an das Handy zu senden, diese werden als Popup-Nachrichten in der Anwendung präsentiert.