Martina Dodmasej
If19b159
BIF4-SWE2

SWE 2 Tour Planner Dokumentation

GitHub

https://github.com/martinadodmasejj/TourPlanner.git

App Architechture

Der TourPlanner App ist nach die MVVM Pattern entwickelt, d.h es gibt ein DataAccessLayer, BusinessLayer, MainViewModel und Views.

Key Architechtural Decisions

Maven

Maven wurde als Libraryverwaltung Tool verwendet, um neue externe Libraries hinzufügen.

JSON Jackson

JSON Jackson wurde verwendet, um die Daten aus der JSON Datenbank Config File auszulesen.

JSON in Java

JSON in Java wurde verwendet, um den http JSON Response Daten vom String Format im JSON Objekte umzuwandeln damit sie weiterverwendet werden können.

iText PDF

Die iTextPDF Library wurde verwendet, um die TourReports und TourReportSummaries zu generieren.

JUNIT

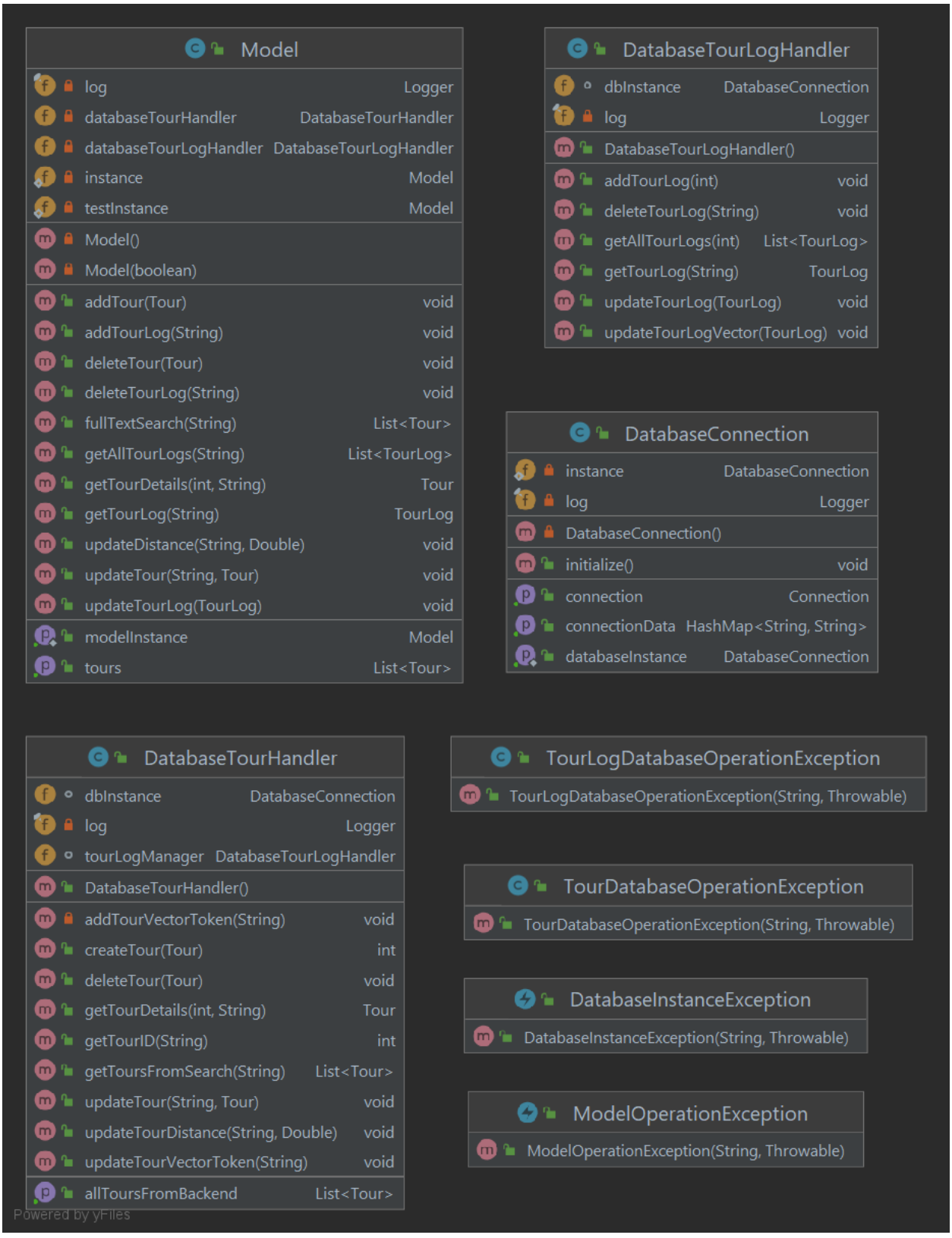
Die JUNIT Library wird verwendet, um Unit Tests umzusetzen.

Tour Images

Tour Images werden in eine lokale Verzeichnis im Projektverzeichnis gespeichert. Wenn ein Tour neu erstellt oder aktualisiert wird, dann wird das neue Tour Image in dem Verzeichnis gespeichert und das alte Tour Image gelöscht. Somit werden http Requests und Performance erspart, und anstatt jedes Mal eine neue Request zu schicken wird das gespeicherte Bild dafür verwendet.

Martina Dodmasej
If19b159
BIF4-SWE2

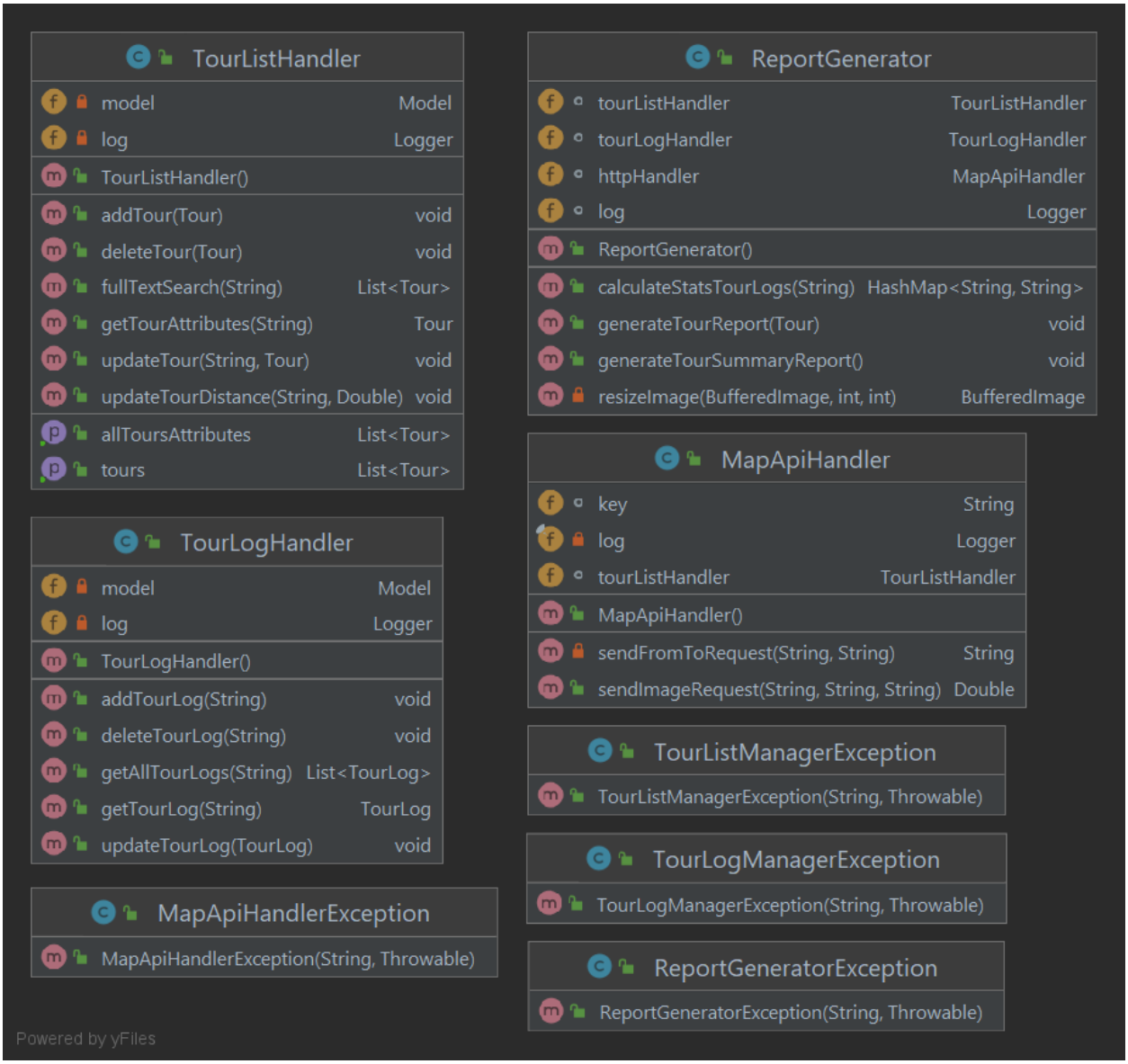
DAL – DataAccessLayer



Martina Dodmasej
If19b159
BIF4-SWE2

Der DataAccessLayer ist von mehreren Packages aufgebaut. Die Database Package kümmert sich um die Tour und Tour Logs Datenbankperationen wie z.b Inserts,Update,Delete,Search. Die Klassen können einen DatabaseInstance von die DatabaseConnection Klasse anfordern, um Operationen durchzuführen. Die Connectiondaten werden aus die config.json File gelesen. Die Operationen auf Datenbankebene sind auf zwei Klassen geteilt DatabaseTourHandler und DatabaseTourLogHandler um den single-responsibility principle zu erfüllen. Als Interface dient die Model Klasse, die alle Funktionalitäten an die andere Layer zur Verfügung stellt. Zusätzlich hat jede Klasse seine eigene Exceptions, um detaillierte Auskunft über Errors möglich zu machen.

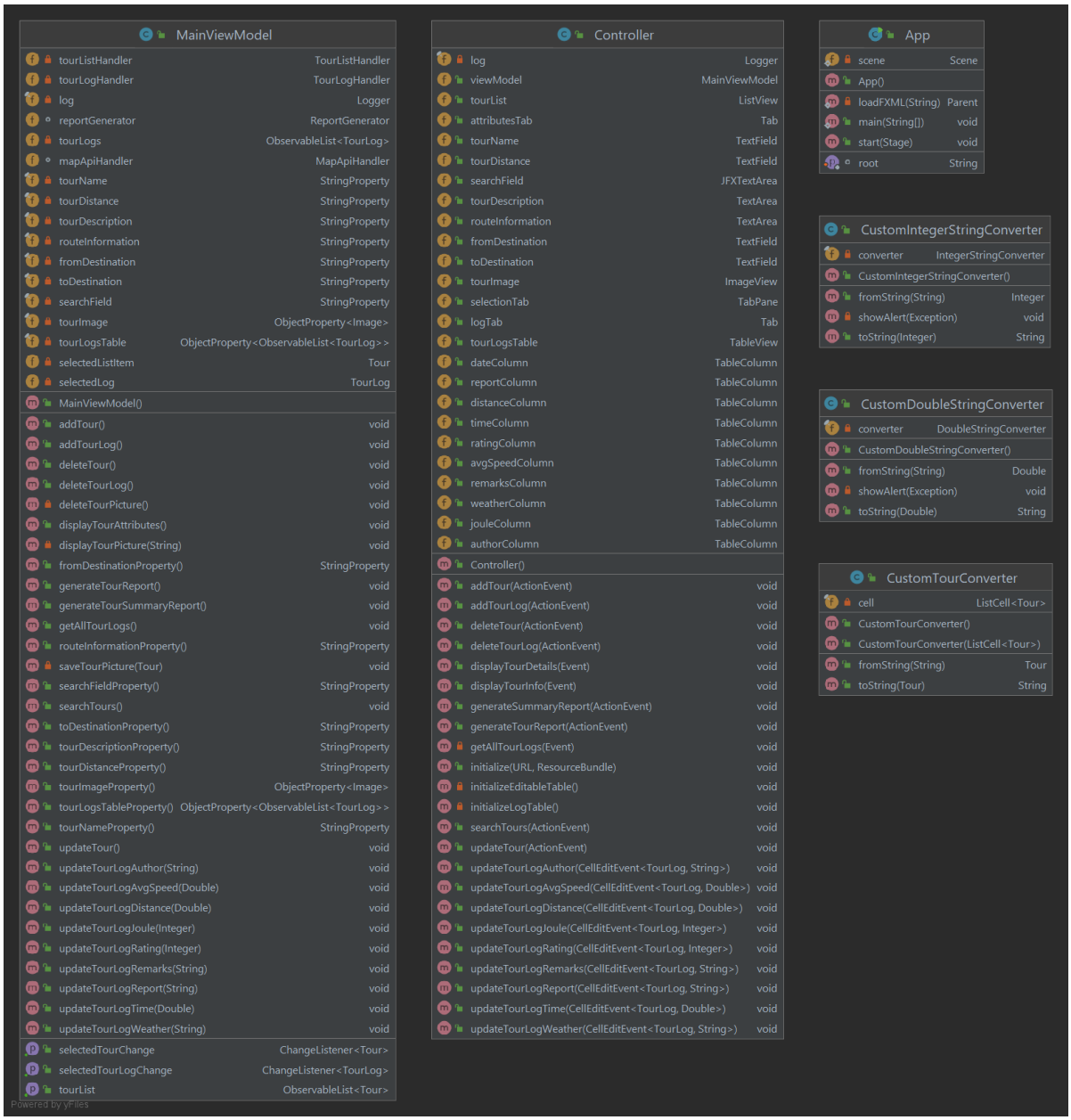
BusinessLayer



Martina Dodmasej
If19b159
BIF4-SWE2

Der BusinessLayer kümmert sich um das Verwalten und Bearbeiten der Daten, die von Model ausgeliefert werden. Diese Layer entsteht aus der ReportGenerator, MapApiHandler, und die zuständige TourList und TourLog Handlers. Die TourList und TourLog Handlers kommunizieren mit den Model, um Daten von Backend rauszuholen. Die Daten werden dann bearbeitet und stehen bereit für den ViewModel zum Holen. Der Report Generator verwendet die Handler, um Daten von Backend zu lesen und um Pdfs entsprechend zu generieren. Zusätzlich werden Statistiken berechnet für den gesamten TourSummaryReport. Der MapApiHandler kümmert sich um das Schicken und Empfangen von http Requests and den MapApi. Dort werden die JSON Daten behandelt und weitergeleitet. Jede Klasse definiert seine eigene Exceptions zusätzlich.

View / ViewModel

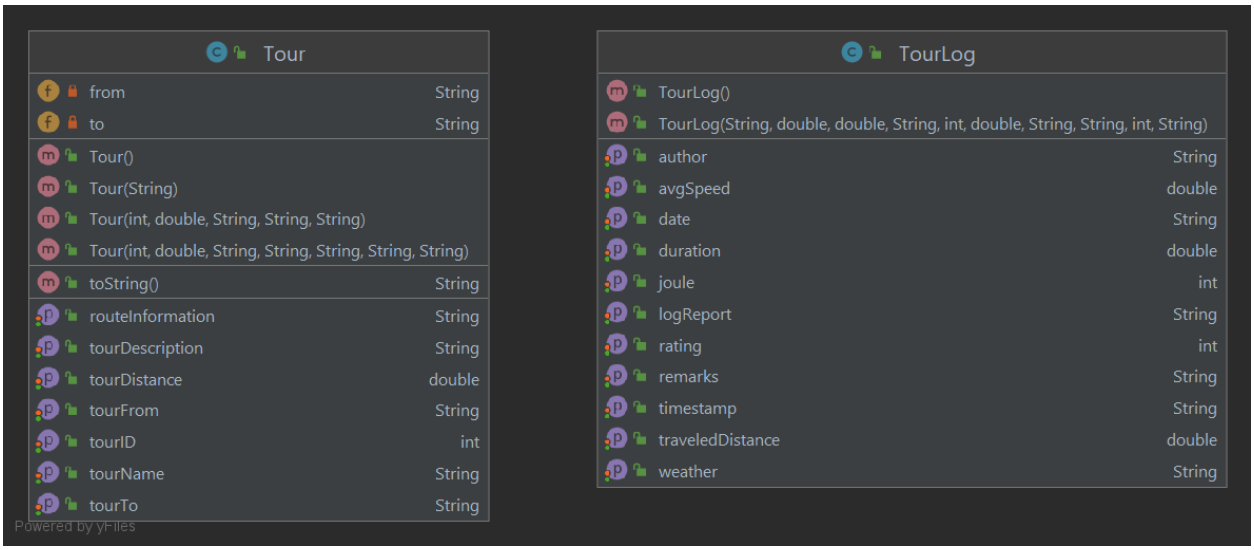


Martina Dodmasej
If19b159
BIF4-SWE2

Die View Layer kümmert sich um das Darstellen aller Daten um UI und um das Lesen alle Benutzer Inputs und Anforderungen. Um den MVVM Pattern zu erfüllen gibt's ein MainViewModel Klasse, die mit den JavaFx Controller verbunden ist. Die MainViewModel Klasse ist bidirektional mit dem Controller verbunden, d.h bidirektional gebundene Werte werden auch beim MainViewModel automatisch erneuert. Zusätzlich gibt es 3 ConverterKlassen: CustomIntegerConverter, CustomDoubleConverter, CustomTourConverter. Die dienen, um Objekte in dem UI darstellen zu können beispielsweiße: in dem TourList wird nicht der TourObjekt selbst dargestellt, sondern nur der TourName.

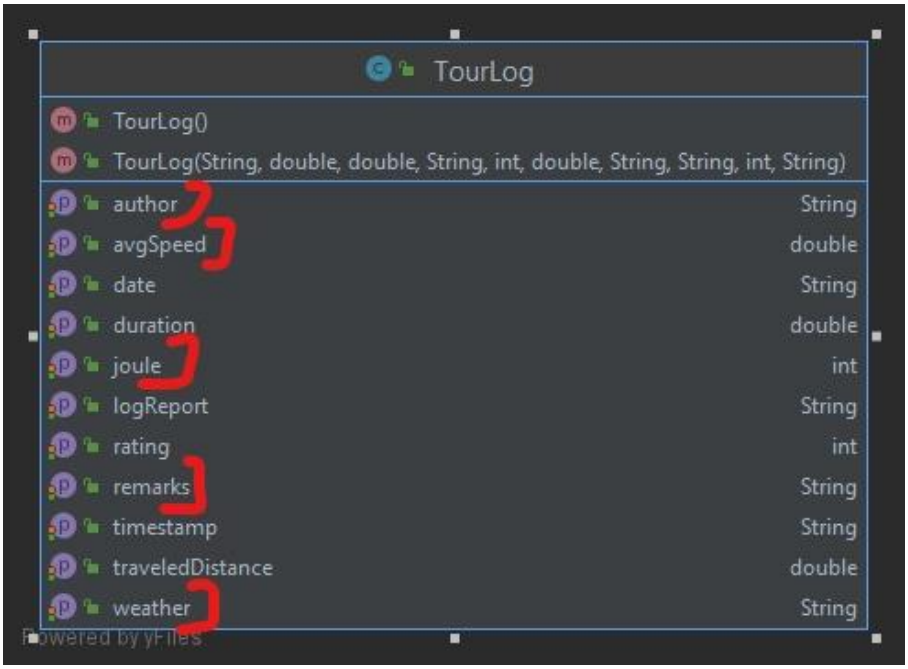
Custom Objects

Es gibt 3 neu definierten Datentypen, um Daten einzupacken und durch die Layer korrekt zu übertragen.



Tour Log Properties

Ein TourLog beeinhaltet fünf neue Attribute zusätzlich zum Spezifikation



Martina Dodmasej
If19b159
BIF4-SWE2

Design Patterns

Singleton Pattern

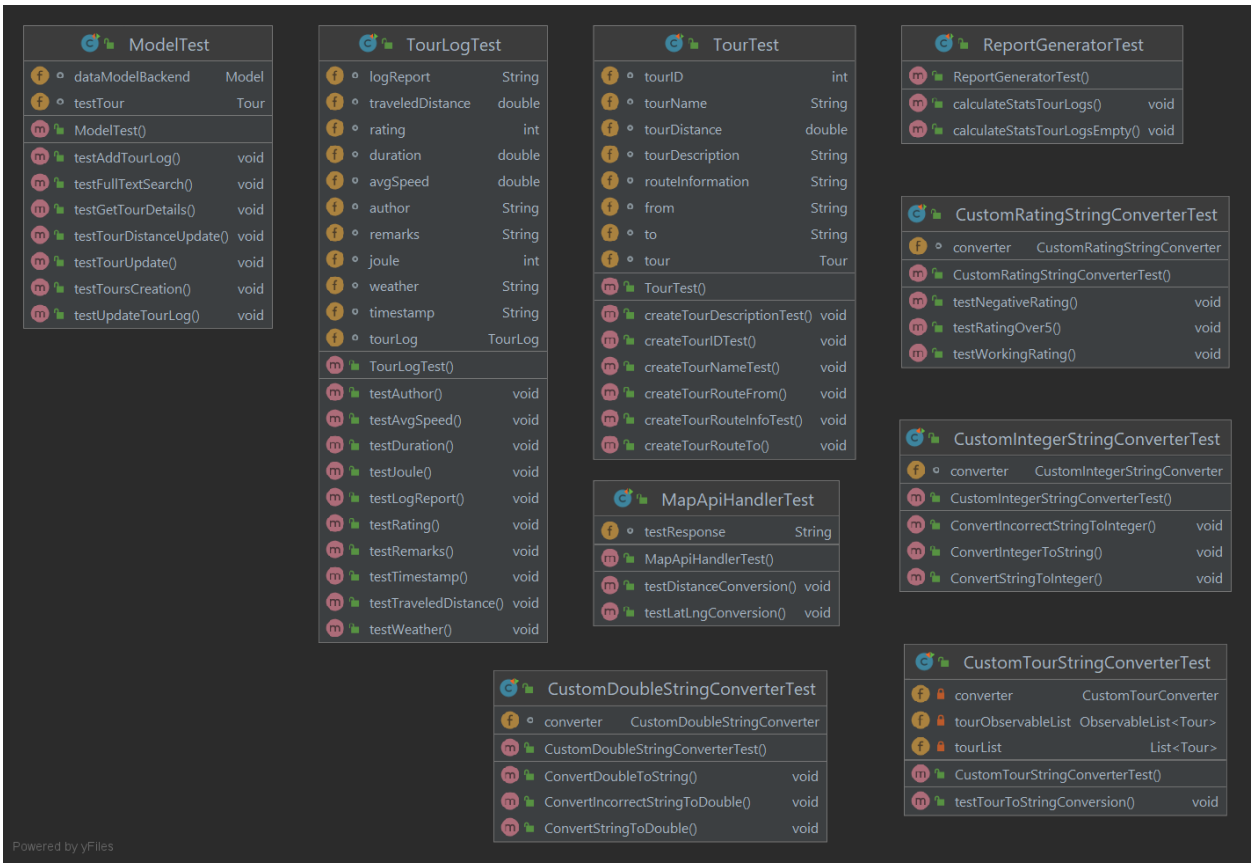
```
public static DatabaseConnection getDatabaseInstance() throws DatabaseInstanceException {
    if(instance==null){
        instance=new DatabaseConnection();
    }
    return  instance;
}
```

```
private DatabaseConnection() throws DatabaseInstanceException {

    log = LogManager.getLogger(DatabaseConnection.class);
    connectionData = new HashMap<String, String>();
    ObjectMapper mapper = new ObjectMapper();
    Map<?, ?> readValues = null;

    try {
        readValues = mapper.readValue(Paths.get( first: "config.json").toFile(), Map.class);
        for (Map.Entry<?, ?> entry : readValues.entrySet()) {
            connectionData.put(entry.getKey().toString(), entry.getValue().toString());
        }
        log.info("Read Connection Attributes from config file");
        initialize();
    } catch (JsonMappingException jsonMappingException) {
        throw new DatabaseInstanceException("could not convert JsonMapping into Java Logic",jsonMappingException);
    } catch (JsonParseException jsonParseException) {
        throw new DatabaseInstanceException("could not parse JsonData into Java",jsonParseException);
    } catch (IOException ioException) {
        throw new DatabaseInstanceException("could not read config file",ioException);
    }
}
```

Die DatabaseConnection Klasse ist eine Singleton. D.h es gibt nur eine einzige globale Instanz von die DatabaseConnection Klasse. Durch das Aufrufen von getDatabaseInstance() wird eine neue Instanz erzeugt, wenn es keines gibt oder die existierende Instanz zurückgegeben. Die Instanz lest dann die Daten aus der config.json Datei und baut die einzelne Datenbankverbindung ein.

Martina Dodmasej
If19b159
BIF4-SWE2

Testing Decisions



CustomInteger/Double/TourStringConverterTest/RatingConverterTest

Die Klassen dienen, um zu testen ob die Objekte vom Backend korrekt in ein passendes Format für den UI umgewandelt werden. Daher ist es wichtig zu testen, ob das der Fall ist damit es nicht rohe Objektdaten in den UI dargestellt werden.

ReportGeneratorTest

Die ReportGeneratorTest Klasse testet, ob der ReportGenerator Tourdaten richtig einsammelt und ob die Daten für die Reports korrekt behandelt werden. Also hier wird nicht der iText Library getestet, sondern nur die Datensammlung für einzelne Tourreports und die Berechnung von die Averages für den Tour Summary Report. Daher ist es wichtig zu bestimmen, ob die Daten für die PDF-Generierung korrekt gesammelt werden

MapApiHandlerTest

In dieser Instanz wird getestet, ob die JSON String Daten korrekt im JSON Objekte umgewandelt werden. In den MapApiHandler werden 2 http Requests nacheinander geschickt und die Daten dazwischen müssen korrekt behandelt werden. Die Klasse testet die Datenbehandlung von die MapApi Klasse und nicht die MapApi selbst.

Martina Dodmasej
If19b159
BIF4-SWE2

ModelTest

Die ModelTest ist besonders wichtig, weil die Modelklasse als Hauptinterface für die Datenmanipulation dient. D.H es muss sichergestellt werden, dass Daten richtig gespeichert und geliefert werden. Die Klasse führt Test auf mögliche Datenbankoperationen.

TourTest / TourLogTest

Daher die Tour und TourLog Klassen als Datentypen verwendet werden, ist es kritisch, dass die Attribute korrekt gespeichert werden.

Lessons Learned

Architecture Planning

Durch ein genaueres Plan und UML Diagramm könnte ein bisschen mehr Zeit sparen. Also das Trennung von die Layers musste neu gemacht werden, um die MVVM Pattern zu halten. Durch besseres Planung könnte man die Klassen sauber trennen und diese Refractoring vermeiden.

Mehr Git Branches

Es war öfters der Fall, dass manche Git Branchen übersprungen wurden und das war natürlich nicht der beste Ansatz, wenn es zu Fehler kam. Wenn da mehr Branchen erstellt wurden, wurde die Arbeit beim Fehlerkorrektur und Finden viel erleichtert, weil da mehrere genauere Kontrollpunkte gibt’s, um zurückzuspringen beim Fehler.

Martina Dodmasej
If19b159
BIF4-SWE2

Time Tracking

Singleton database 15 min

UI Creation 5 Std

Reading from config file 1 Std

Show data from tour database in list 1std

Custom Converters for the UI Lists and Tables 3 Std

Custom Datatypes for Lists and Tables 1 Std

Selecting tour from list 1std

Create new Tour 2 Std

Delete Tour 0.5 Std

Update Tour 1 Std

Image file management 1 Std

singleton Model 10 min

MAP API Images -- 5 Std

FROM TO in tour database -- 2Std

check from to loading when they dont exists 1Std

Logging - 3std

Reporting - PDF of Tour generation (Single Tour) - 4std

Reporting – PDF Summary Report – 2 Std

Fulltextsearch 3 Std

http requests tune up 0.5 Std

Distance von MapQuest - pointers 1 Std

Unit Testing done – 10 Std

Tour Logs add - delete - update - select 7Std

Search for Tour Log Details - 2Std

Exception development -- 5 Std **Total: ~ 44 Std**