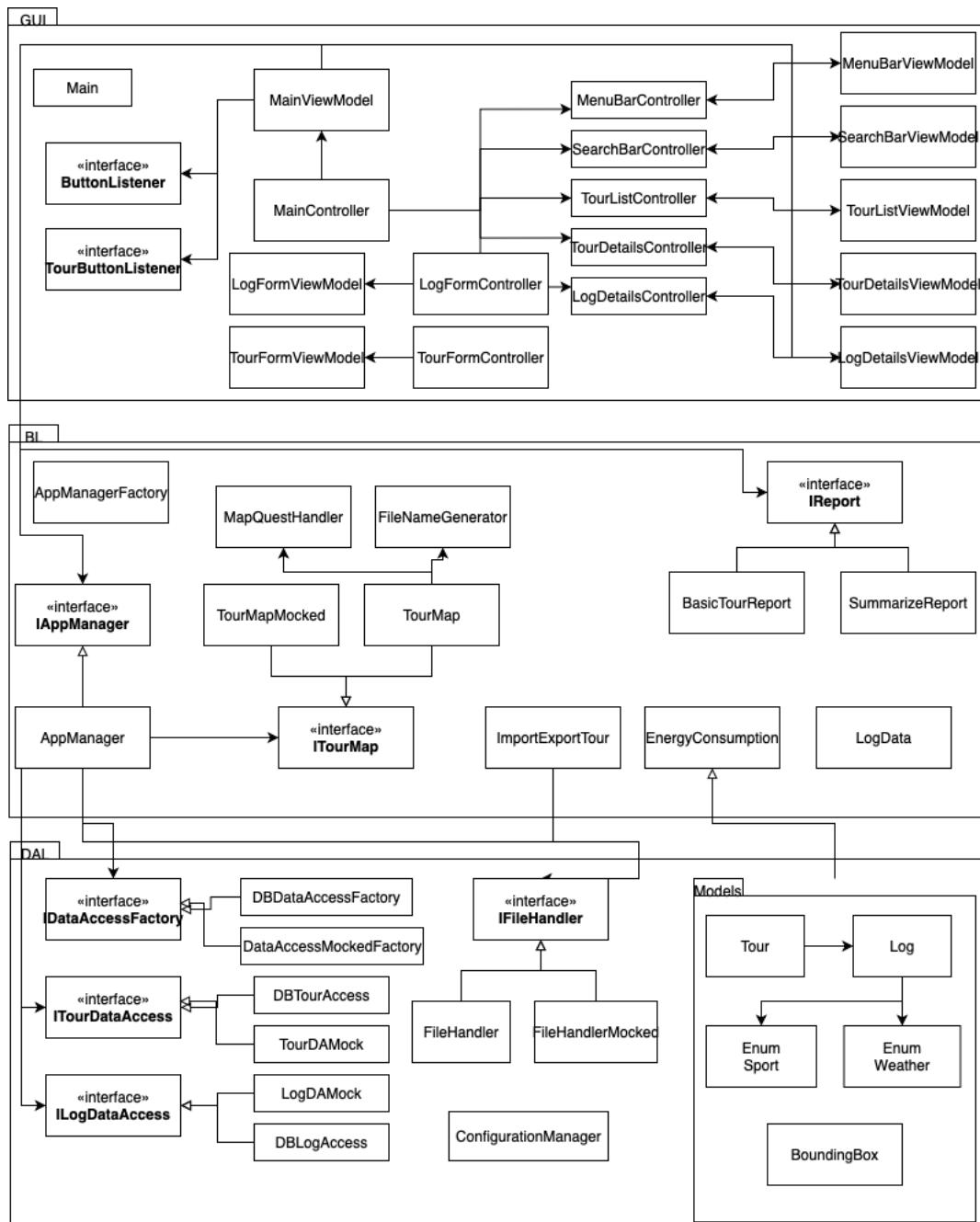


## Dokumentation – Tourplanner

### Architektur / Layers

Das Projekt wurde in drei Module aufgeteilt, ganz unten befindet sich der DataAccessLayer, darauf folgt der BusinessLayer, und zum Schluss folgt noch die UI-Ebene.

### Klassendiagramm



## Data-Access Layer

Der Modul DAL befinden sich zwei große Bereiche, zum einen der DataAccess, zum anderen die Models.

### Models

- Im Bereich Models befinden sich simple Datenhalter-Klassen, angefangen von Tour, welche die ID, Name der Tour, Beschreibung, Start- und Endpunkt, Distanz, den Dateinamen des dazugehörigen Bildes der Karte und eine Liste von dazugehörigen Logs.
- Die Klasse Log enthält die vorgegebenen Attribute Datum, Zeit, Distanz und Rating und wird erweitert um die Attribute Wetter (Enum z.B. Sonnig), Gewicht und Größe (zum Berechnen der verbrauchten Kalorien, siehe BL), Sportart (Enum z.B. Laufen) und die Anzahl der Schritte. Beide Enums befinden sich auch im Models-Package.
- Zu guter Letzt gibt es noch eine Klasse Bounding-Box, welche Koordinaten und Distanz einer Tour erhält (wird bei der Bearbeitung eines Mapquest-Requests benötigt).

### Data-Access

- Im Data-Access befindet sich der ConfigurationManager, welcher Daten (z.B. Connection-String für die Datenbank) aus einem Config-File holt und für einen Schlüssel den passenden Wert zurückliefert.
- Die Interfaces ITourDataAccess und ILogDataAccess definieren Schnittstellen zum Daten persistieren, es gibt jeweils zwei Implementierungen, eine ist eine in-memory gespeicherte Liste von Daten (gedacht als Mock zum Testen), die andere arbeitet mit einer PostgreSQL Datenbank. Außerdem gibt es eine Hilfsklasse, die die Verbindung zur Datenbank verwaltet.
- Das Interface I FileAccess definiert eine Schnittstelle zum Lesen und Schreiben von Dateien einer „leeren Implementierung“, welche zum Testen der AppManager-Klasse verwendet wird, und eine tatsächliche Implementierung.

### Business-Layer

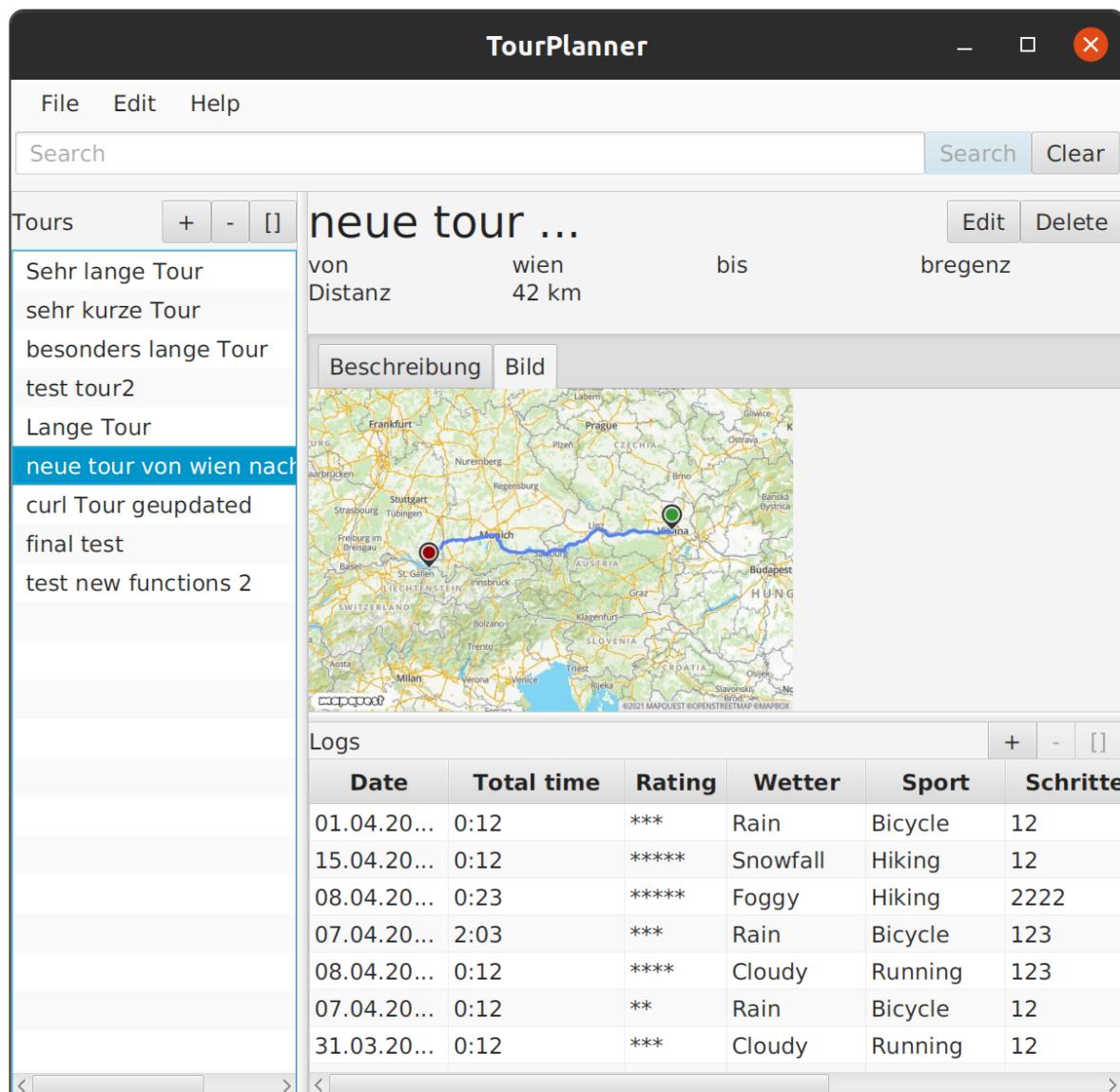
- Die AppManager Klasse implementiert das AppManager-Interface, welches Schnittstellen für das Viewmodel bereitstellt. Es überprüft auch die Parameter bei Save/Update auf Richtigkeit und holt sich dabei auch das Bild von mapquest bzw löscht das Bild sollte es nicht mehr benötigt werden.
- Die Interface IReport definiert eine Schnittstelle, die ein PDF als byte-Array zurückgibt. Die Klasse BasicTourReport erstellt mithilfe der iText Bibliothek ein simples PDF mit Name, Beschreibung, Distanz, dem Bild und einer Tabelle aller Logs. Die Klasse SummarizeReport erstellt ein PDF mit statistischen Werten wie durchschnittliche Geschwindigkeit, Gesamtdistanz, durchschnittliche Energieverbrauch und noch mehr.
- Im Package TourMap befindet sich das Interface ITourMap, welches eine Funktion definiert, die sich das Bild einer Route von MapQuest holt, es abspeichert und den FileNamen zurückgibt. Es gibt wieder zwei Implementierungen, eine um das Holen zum Mocken zwecks Testen und eine, die tatsächlich das Bild holt. Dazu erstellt sie mittels der Hilfsklasse MapQuestHandler ein http-Request und liest daraus die

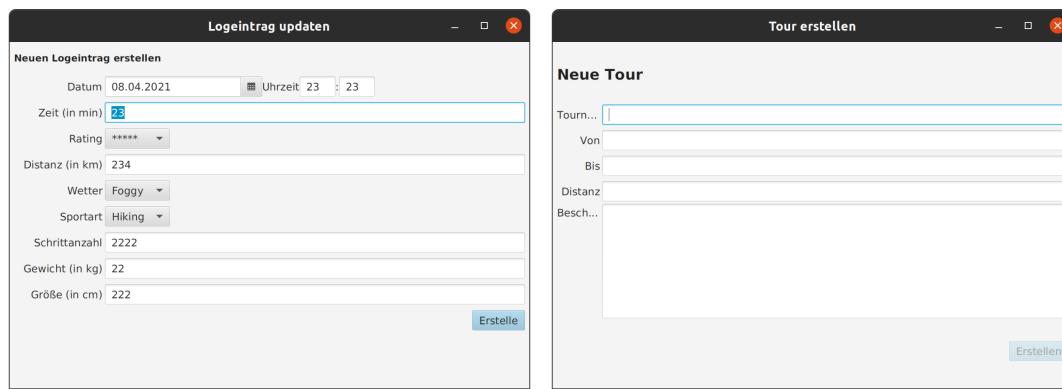
benötigten Koordinaten raus. Dann schickt sie ein weiteres Request um das Bild zu bekommen. Die Klasse FileNameGenerator generiert einen Filenamen für das Bild, das Bild wird abgespeichert und der Filename wird zurückgegeben.

- Die Klasse ImportExportTour besitzt zwei Methoden, eine erstellt aus einer Tour ein JSON und speichert es ab, die andere geht den umgekehrten Weg und liest aus einem gegebenen File die Daten wieder heraus und wandelt sie in ein Tour-Objekt um.
- Die Klassen LogData und EnergyConsumption berechnen zu einem gegebenen Log Durchschnittsgeschwindigkeit, Pace und Energieverbrauch.

## GUI

Die Applikation enthält drei verschiedenen Fenster, das Hauptfenster mit allen Touren und Detailinformationen zu einer einzelnen Tour, ein Formular zum Erstellen/Update eines Logs zu einer Tour und eins zum Erstellen/Update eines Logs zu einer Tour.





## Views

Das Mainwindow wird auf mehrere FXML-Files (MenuBar, SearchBar, TourList, TourDetails und LogDetails) aufgeteilt. Beide Formulare besitzen jeweils nur ein FXML-File.

## Controller

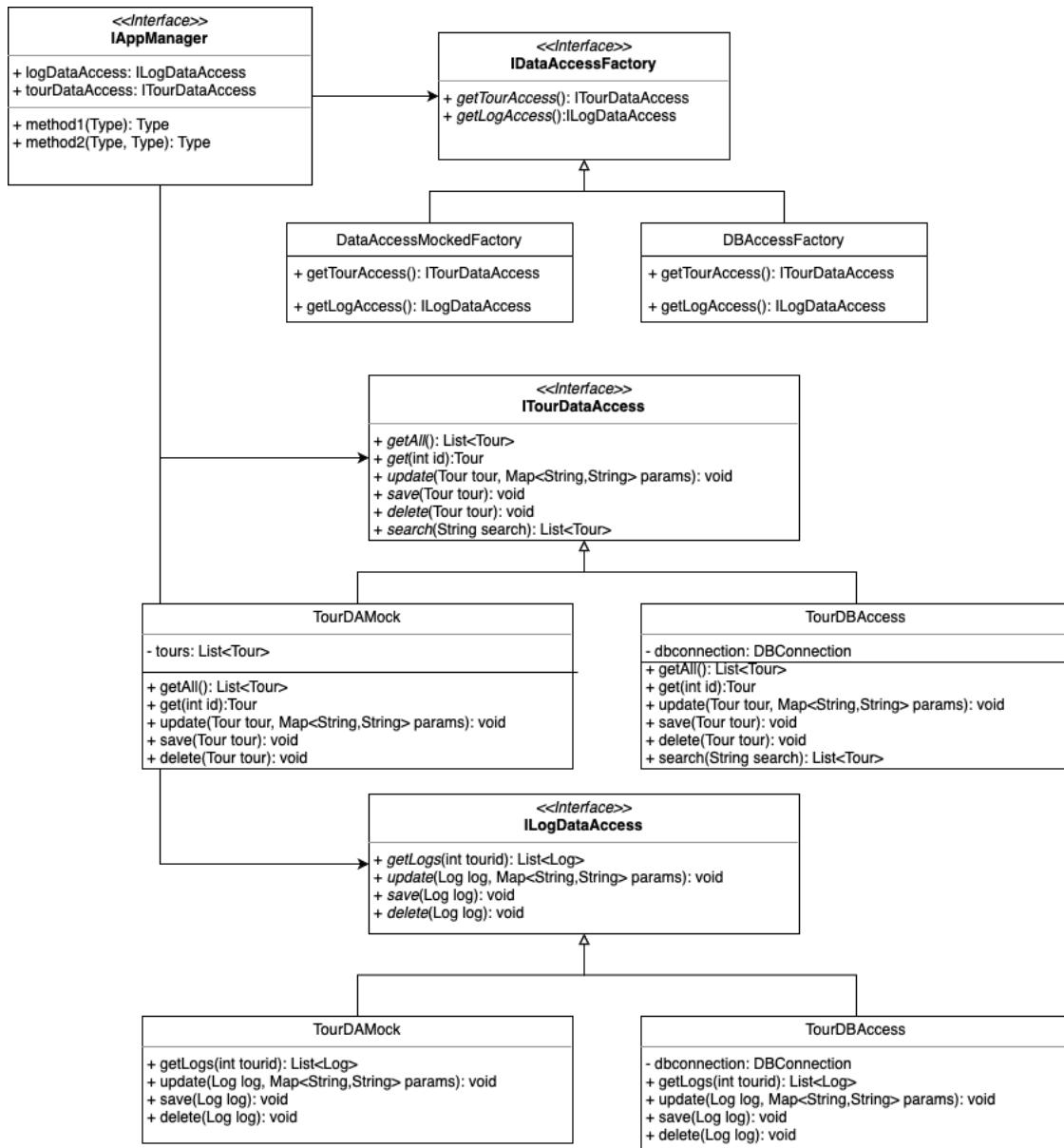
Im MainController werden alle Subcontroller (der inkludierten FXMLs) injected und in der Initialisierungs-Methode beim initialisieren des Konstruktors an diesen übergeben. Des Weiteren finden in allen Controllern die Bindings von ObjectProperties, das Setzen von ObservableLists sowie das Hinzufügen von Listeners.

## ViewModels

Im MainViewModel wird über den AppManager mit der DB kommuniziert und die Daten anschließend gesetzt. Alle Funktionen (z.B. der ImportListener) werden gesetzt. Sollte z.B. der User dann in der Menubar auf „Import“ klicken, wird dadurch das MainViewModel „informiert“, im MainViewModel wird dann von der Klasse ImportExportTour die Methode import aufgerufen, die dadurch zurückgegebene Tour wird dann dem Appmanager übergeben und gespeichert. Anschließend werden die Tourlisten aktualisiert und eine Tour gesetzt.

## Design Pattern

### Abstrakte Fabrik



Als Designpattern wurde die abstrakte Fabrik gewählt. Der Grund ist, da LogDataAccess und TourDataAccess als eine „Produktfamilie“ angesehen wurden und sich so die die abstrakte Fabrik angeboten hat. Es gibt zwei unterschiedliche Implementierungen, einmal eine in-memory gespeicherte Liste von Touren, welche fürs Testen verwendet wird, und einmal eine, die die Daten aus einer PostgreSQL-Datenbank holt. Der **IAppManager** (Klient) bekommt im Konstruktor eine **IDataAccessFactory** übergeben und kann sich so die einzelnen DAOs holen, ohne abhängig von einer speziellen Implementierung abhängig zu sein. Alle DataAccess-Klassen wurden zudem als Singleton (privater Konstruktor, privates statische Objekt der Klasse und eine statische Methode zum Bekommen des Objekts) implementiert, da über die ganze Applikation von jeder dieser Klassen nicht mehr als ein Objekt notwendig ist.

## Unit Tests

Zum Testen wurde die Klassen AppManager sowie LogData und EnergyConsumption hergenommen.

### AppManagerTest

Zum Testen des AppManagers wird die gemockte Datenbank (siehe Designpattern – Abstrakte Fabrik) verwendet, zudem wird das Holen der Tourkarte und Abspeichern von Files gemockt.

- Bei der Methode getSearch() wurde getestet, ob die richtige Anzahl von Ergebnissen zurückgeliefert wird.
- Bei der Methode updateTour() wurden bei den Parametern auf Gültigkeit (nicht leer) getestet und ob bei einem neuen Start- bzw. Zielpunkt ein neues Bild generiert wird.
- Bei der Methode updateLog() wurden die Parameter auf Richtigkeit getestet, z.B. leere Parameter, Distanz keine Zahl oder negativ oder falsches Format beim Datum.
- Außerdem wurden die Methoden save, delete und get (Log oder Tour) getestet.

### LogDataTest

Es wurde bei allen Methoden (getSpeed, getPace, getEnergyConsumption (mit Hiking, Running, Bicycle)) getestet, ob das erwartete Ergebnis der Berechnung zurückgegeben wird.

## Unique Feature

### Kalorienverbrauch

Bei den Logs werden Distanz, verbrauchte Zeit sowie Gewicht eingegeben, und basierend auf den eingegebenen Daten werden dann durchschnittliche Geschwindigkeit, Pace (=min/km) sowie Energieverbrauch berechnet, beim Energieverbrauch wird auch noch die Sportart (Laufen, Radfahren, Wandern) berücksichtigt.

### Diagramme im SummarizeReport

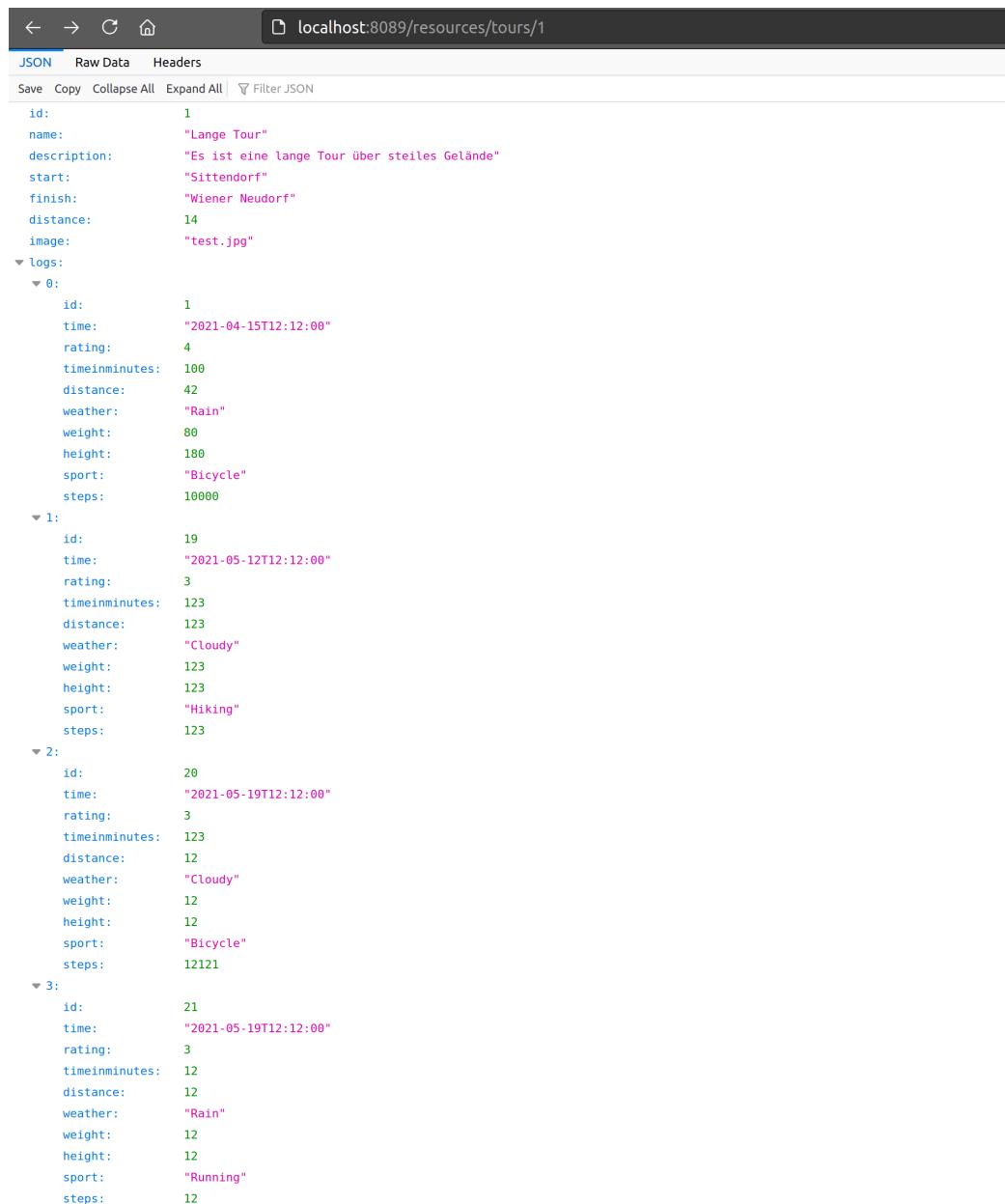
Im Summarize-Report werden noch zusätzlich Balkendiagramme zur besseren Visualisierung hinzugefügt, zu Erstellung von Balkendiagrammen wird die Library JFree verwendet, dann muss eine Hilfsbibliothek zum Hinzufügen in das iText-PDF verwendet werden.

### RESTServer

Mithilfe von Springweb wurde ein kleiner RESTserver aufgesetzt mit folgenden Schnittstellen:

- GET /resources/tours liefert alle Touren im json-Format.
- GET /resources/tours/{id} liefert die Tour mit der ID id.
- POST /resources/tours erstellt eine neue Tour mit den übertragenen Daten.
- PUT /resources/tours/{id} updates die Tour mit der ID id.
- DELETE /resources/tours/{id} löscht die Tour mit der ID id.

- GET /resources/tours/{tourid}/logs liefert eine Liste aller Logs zur Tour mit der ID tourid
- GET /resources/tours/{tourid}/logs/{logid} liefert den Log mit der ID logid zur Tour mit der ID tourid
- POST /resources/tours/{tourid}/logs erstellt für die Tour mit der ID tourid einen neuen Log mit den überlieferten LogDaten
- PUT /resources/tours/{tourid}/logs/{logid} editiert die Logdaten des Logs mit der LogID logid.
- DELETE /resources/tours/{tourid}/logs/{logid} löscht den Log mit der LogID logid
- GET /images/{imagename} liefert das Bild mit dem Filenamen imagename zurück
- GET /resources/tours/{tourid}/report liefert ein BasicTourReport als PDF zurück
- GET /resources/tours/{tourid}/summarizereport liefert ein SummarizeReport als PDF zurück

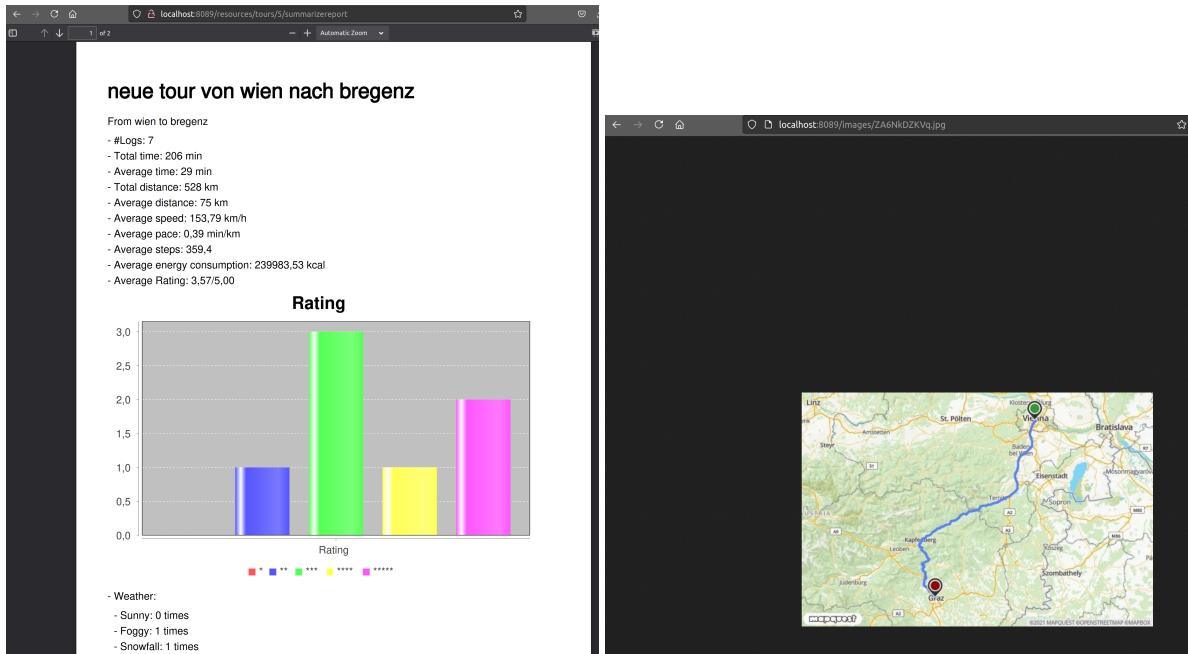


The screenshot shows a JSON viewer interface with the URL `localhost:8089/resources/tours/1` in the address bar. The JSON data represents a tour with the following details:

```

{
  "id": 1,
  "name": "Lange Tour",
  "description": "Es ist eine lange Tour über steiles Gelände",
  "start": "Sittendorf",
  "finish": "Wiener Neudorf",
  "distance": 14,
  "image": "test.jpg",
  "logs": [
    {
      "id": 1,
      "time": "2021-04-15T12:12:00",
      "rating": 4,
      "timeinminutes": 100,
      "distance": 42,
      "weather": "Rain",
      "weight": 80,
      "height": 180,
      "sport": "Bicycle",
      "steps": 10000
    },
    {
      "id": 19,
      "time": "2021-05-12T12:12:00",
      "rating": 3,
      "timeinminutes": 123,
      "distance": 123,
      "weather": "Cloudy",
      "weight": 123,
      "height": 123,
      "sport": "Hiking",
      "steps": 123
    },
    {
      "id": 20,
      "time": "2021-05-19T12:12:00",
      "rating": 3,
      "timeinminutes": 123,
      "distance": 12,
      "weather": "Cloudy",
      "weight": 12,
      "height": 12,
      "sport": "Bicycle",
      "steps": 12121
    },
    {
      "id": 21,
      "time": "2021-05-19T12:12:00",
      "rating": 3,
      "timeinminutes": 12,
      "distance": 12,
      "weather": "Rain",
      "weight": 12,
      "height": 12,
      "sport": "Running",
      "steps": 12
    }
  ]
}

```



## Tracked Time

- Erstellen des FXML Files und Controller mit VM verbinden (nur erster Entwurf): 15h
- Datenbank und Datenbankklassen erstellen: 15h
- AppManager mit VM und DataAccess erstellen: 5h
- MapQuest einbinden (empfangen von Bild hat Schwierigkeiten bereitet): 10h
- Exportieren/Importieren: 2h
- Report erstellen (zuerst Apache PDFBox verwendet, eher unpraktisch, deshalb auf iText umgestiegen, einbinden von JFree (Diagramme) dauerte länger): 10h
- Logging integrieren:
- Feature (Energieverbrauch): 3h
- UnitTests (einige Fehler aufgetreten): 5h
- RESTserver + Integrationstest: 10h
- Ausbessern von GUI, kleine Featureverbesserungen: 15h

Gesamt: 90h