SWEN 2

Tour Planer

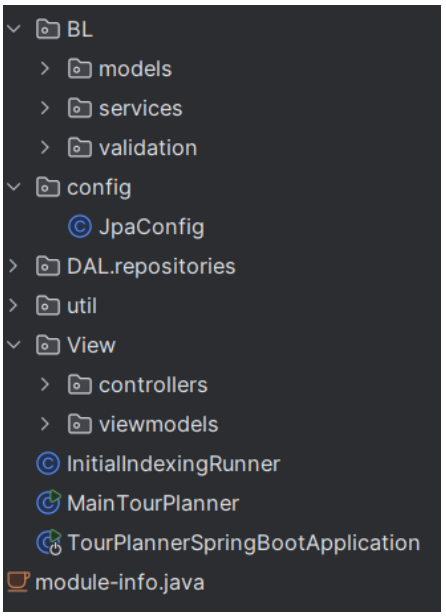
Gruppe 5

Elena Steigberger | Helene Harrer

Inhaltsverzeichnis

1. Technical Steps and Decisions

1.1. File Struktur & Layers



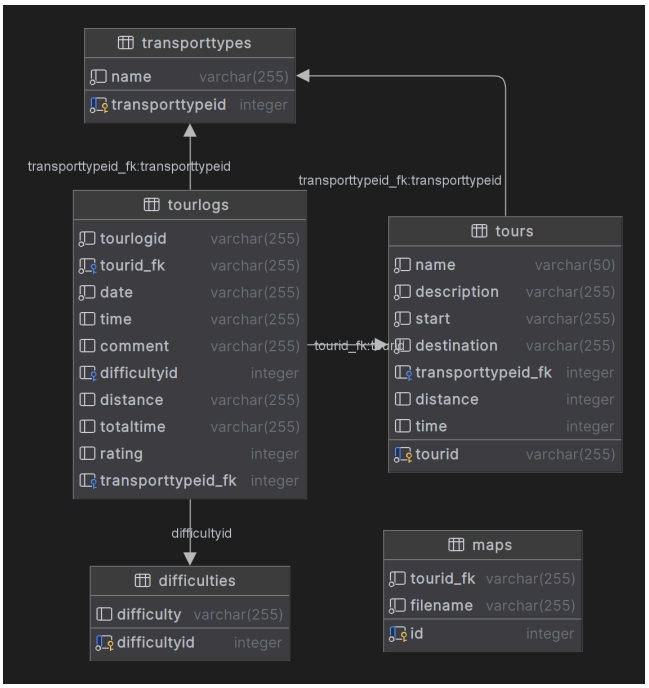
Unsere Anwendung folgt dem Model-View-ViewModel (MVVM) Muster und ist in verschiedene Schichten unterteilt, um die Verantwortlichkeiten klar zu trennen und die Wartbarkeit des Codes zu erhöhen:

1. BL (Business Logic):
   - models: Enthält die Datenmodelle, die die Struktur der Daten definieren, die in der Anwendung verwendet werden.
   - services: Beinhaltet die Geschäftslogik und die Regeln, wie Daten verarbeitet und manipuliert werden.
   - validation: Enthält Klassen zur Validierung der Daten, um sicherzustellen, dass nur korrekte Daten verarbeitet werden.
2. config:
   - JpaConfig: Konfiguriert JPA (Java Persistence API) und andere benötigte Einstellungen für die Datenbankverbindungen.
3. DAL:
   - repositories: Diese Schicht verwaltet die Datenzugriffsschicht (Data Access Layer) und enthält Repositories, die für die Kommunikation mit der Datenbank verantwortlich sind.
4. util:
   - Dieser Ordner enthält Hilfsklassen und Utility-Funktionen, die an mehreren Stellen in der Anwendung wiederverwendet werden. Die Entscheidung, einen separaten util-Ordner zu erstellen, basiert auf der Notwendigkeit, allgemeinen Code zu kapseln und die Wiederverwendbarkeit und Modularität zu erhöhen. Hier befindet sich z.B. der PDF-Generator für die Reports.
5. View:
   - controllers: Enthält die Controller-Klassen, die als Bindeglied zwischen der Benutzeroberfläche (UI) und der Geschäftslogik fungieren.

- viewmodels: Beinhaltet die ViewModel-Klassen, die die Daten und Befehle enthalten, die an die View gebunden sind, und sorgt für die Trennung zwischen UI und Geschäftslogik.

Diese Struktur ermöglicht eine klare Trennung der Verantwortlichkeiten und erleichtert die Wartung und Erweiterung der Anwendung. Das MVVM-Muster sorgt dafür, dass die Benutzeroberfläche unabhängig von der Geschäftslogik entwickelt werden kann, während Utility-Klassen im util-Ordner wiederverwendbare Funktionen kapseln, um redundanten Code zu vermeiden.

1.2. Datenbank-Schema & Docker



Unser Datenbankschema für die TourPlanner-Anwendung ist in PostgreSQL implementiert und umfasst fünf Tabellen: tours, tourlogs, transporttypes, difficulties und maps. Die Beziehungen zwischen diesen Tabellen sind durch Fremdschlüssel definiert, um die Referenzen zwischen Touren, Transporttypen, Schwierigkeitsgraden und zugehörigen Log-Einträgen zu verwalten. Die Verwaltung der PostgreSQL-Datenbank erfolgt mittels Docker, wobei wir Docker-Images und Docker-Compose-Dateien verwenden, um eine konsistente und reproduzierbare Umgebung für die Entwicklung und den Betrieb bereitzustellen.

1.3. OpenRouteService & Mapbox

Unsere Anwendung nutzt “OpenRouteService” und “Mapbox” für die Erstellung und Anzeige von Karten. OpenRouteService wird verwendet, um geografische Daten wie Start- und Zielkoordinaten sowie Routeninformationen abzurufen. Mapbox dient zur Generierung statischer Kartenbilder, die in der Detailansicht der Tour angezeigt werden.

```
String url = String.format(
        "https://api.openrouteservice.org/geocode/search?api_key=%s&text=%s",
        ApplicationContext.getApiKeyOrs(), URLEncoder.encode(location, StandardCharsets.UTF_8));
```

Hier ist ein Beispiel eines Requests, der die Koordinaten einer Location requestet.

Um die Sicherheit zu gewährleisten, werden die API-Schlüssel extern in einer Datei gespeichert und nicht direkt im Code. Beim Klick auf das Kartenbild (bereitgestellt von Mapbox) in der Detailansicht öffnet sich eine interaktive Karte im Standardbrowser des Benutzers mithilfe von Leaflet, wodurch eine benutzerfreundliche Navigation ermöglicht wird. (siehe MapRequester-Klasse in /util)

1.4. Report-Generation

Für die Generierung des TourReports und des StatsReports verwenden wir das Framework iTextPDF. Bei der Erstellung wird dabei ein PDF generiert, in das dann hineingeschrieben wird. Den Inhalt stellen wir selbst aus den jeweiligen Daten der Tour zusammen.

```
private void writeTourReport(TourModel tour, Path filePath) throws IOException {  1 usage  elektrolena
    try {
        PdfWriter writer = new PdfWriter(filePath.toString());
        PdfDocument pdf = new PdfDocument(writer);
        Document document = new Document(pdf);

        addHeader(document, tour);
        addMapImage(document, tour);
        addTourDataSection(document, tour);
        addSectionSeparator(document);
        addTourLogsSection(document, tour);

        document.close();
    } catch (FileNotFoundException e) {
        log.error("Failed to write pdf to tour report", e);
    }
}
```

Anschließend wird das PDF geschlossen und der User kann es an einem von ihm gewählten Ort speichern. Als letztes wird es ihm in der Vorschau angezeigt.

1.5. Export/Import

Beim Exportieren wird die ausgewählte Tour zusammen mit allen zugehörigen Logs in ein JSON-Format umgewandelt und in einer Datei gespeichert. Diese JSON-Datei kann dann auf dem Computer des Benutzers gespeichert werden.
Beim Import ermöglicht die Anwendung dem Benutzer, eine zuvor gespeicherte JSON-Datei hochzuladen. Die Anwendung liest die Datei, mappt die Daten auf die entsprechenden Tour- und TourLog-Modelle und speichert diese wieder in der Datenbank.
(siehe JSONGenerator-Klasse in /util)

1.6. Logging

Das Logging haben wir mit Hilfe des Log4J Frameworks eingebaut. Dafür haben wir die notwendigen Dependencies in die pom.xml eingefügt und die module-info.java erweitert.

```
// Logging Modules
requires org.slf4j;
requires org.apache.logging.log4j;
requires org.apache.logging.log4j.core;
requires org.apache.logging.log4j.slf4j2.impl;
```

So konnten wir die Annotation @Slf4j an die gewünschten Klassen anfügen und den Logger des Frameworks nutzen. Welche logs ausgegeben und in logfiles gespeichert werden, wird in der Konfigurationsdatei log4j.xml festgelegt.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Configuration status="DEBUG">
    <Appenders>
        <Console name="ConsoleAppender" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
        </Console>
        <File name="FileAppender" fileName="src/main/resources/technikum/at/tourplanner_swen2_team5/logs/${date:yyyyMMdd}.l
            <PatternLayout pattern="%d{yyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
        </File>
    </Appenders>
    <Loggers>
        <!-- Logger for everything except own project -->
        <Logger name="org" level="error" additivity="true">
            <AppenderRef ref="ConsoleAppender" />
            <AppenderRef ref="FileAppender" />
        </Logger>
```

1.7. Search mit Spring Boot

Um die Volltextsuche so effizient wie möglich zu gestalten, haben wir uns für **Hibernate Search 7.1.1.Final** mit **Hibernate 6.4.4.Final**, **Spring Boot 3.2.2**, **Jakarta Persistence 3.1.0** und **Apache Lucene 9.9.2** entschieden.

Hibernate Search kann die angebundene Datenbank durchsuchen und gefundene Objekte, die mit der Suche übereinstimmen, in einer Liste zurückgeben.

```
@Transactional  1 usage   elektrolena
public List<TourModel> searchTours(String keyword) {
    SearchSession searchSession = Search.session(entityManager);
    return searchSession.search(TourModel.class) SearchQuerySelectStep<capture of ?, EntityReference, TourModel, SearchLoadingOptionsStep, capture of ?, capture of ?>
            .where(f -> f.match().fields( ...strings: "name", "description", "start", "destination").matching(keyword)) capture of ?
            .fetchHits( integer: 20); // Fetch the top 20 results
}
```

Alle Attribute, die bei der Suche berücksichtigt werden sollen, müssen hierbei im dazugehörigen Model mit @FullTextField annotiert werden.

```
@FullTextField
@Getter
@Column(name = "name", nullable = false)
private String name;
```

1.8. Mandatory Zusatz-Feature (Sortieren der Touren)

Für das Sortieren der Tour-Liste haben wir drei verschiedene Buttons eingefügt, die die Touren nach Neuigkeit, Popularität oder Kinderfreundlichkeit anordnen können. Dabei können sie entweder aufsteigend oder absteigend sortiert werden.

Dies passiert im TourListController selbst und wird mit booleans geregelt.

```
private boolean isDescendingRecent = true;  9 usages
private boolean isAscendingPopularity = true;  7 usages
private boolean isAscendingChildFriendliness = true;  6 usages
```

1.9. Schwierigkeiten & Lösungen

1.9.1. Interaktive Map anzeigen

Während der Entwicklung unserer Anwendung hatten wir ursprünglich vor, eine interaktive Karte direkt in die Anwendung zu integrieren. Dies stieß jedoch auf erhebliche Schwierigkeiten aufgrund von Einschränkungen, die ab Java Version 18 eingeführt wurden. Das Hauptproblem bestand darin, dass die Karte unvollständig angezeigt wurde, wegen fehlender Kacheln, was für die Erstellung interaktiver Karten unerlässlich ist. Daher mussten wir auf eine alternative Lösung umsteigen.
Um diese Einschränkungen zu überwinden, entschieden wir uns, Mapbox zu verwenden, das statische Kartenbilder generieren kann. Diese Bilder werden dann innerhalb unserer Anwendung angezeigt. Wenn ein Benutzer auf das Kartenbild in der Tour-Detailansicht klickt, öffnet sich eine interaktive Karte im Standard-Webbrowser unter Verwendung von

Leaflet. Diese Lösung nutzt die Leistungsfähigkeit externer APIs und sorgt gleichzeitig für ein nahtloses Benutzererlebnis innerhalb der Grenzen der Java-Umgebung.

1.9.2. Responsive-Design

Die Implementierung der Responsiveness in unserer Anwendung war eine anspruchsvolle Aufgabe. Während viele Layout-Anforderungen mithilfe des GridPane in JavaFX relativ gut umgesetzt werden konnten, stellten sich einige andere Aspekte als besonders schwierig heraus. Ein großes Problem war die Detailansicht der Touren, in der die Informationen zur Tour und die Karte der Tour nebeneinander dargestellt werden sollten. Das Ziel war es, dass diese beiden Bereiche bei einer Verkleinerung des Bildschirms untereinander angeordnet werden.
Trotz zahlreicher Versuche und verschiedener Ansätze war es nicht möglich, dieses Verhalten zuverlässig zu implementieren. Wir probierten verschiedene Layouts und Anpassungen, aber keine Lösung erwies sich als robust genug, um unsere Anforderungen vollständig zu erfüllen. Schließlich entschieden wir uns, eine Scrollbar zu verwenden. Diese Lösung ermöglichte es uns, den Inhalt scrollbar zu machen, wenn der Bildschirm kleiner wird, sodass der Benutzer die Informationen weiterhin vollständig einsehen kann, ohne dass Elemente abgeschnitten werden. Diese Lösung stellte sicher, dass die Benutzerfreundlichkeit auch bei kleineren Bildschirmgrößen gewährleistet bleibt.

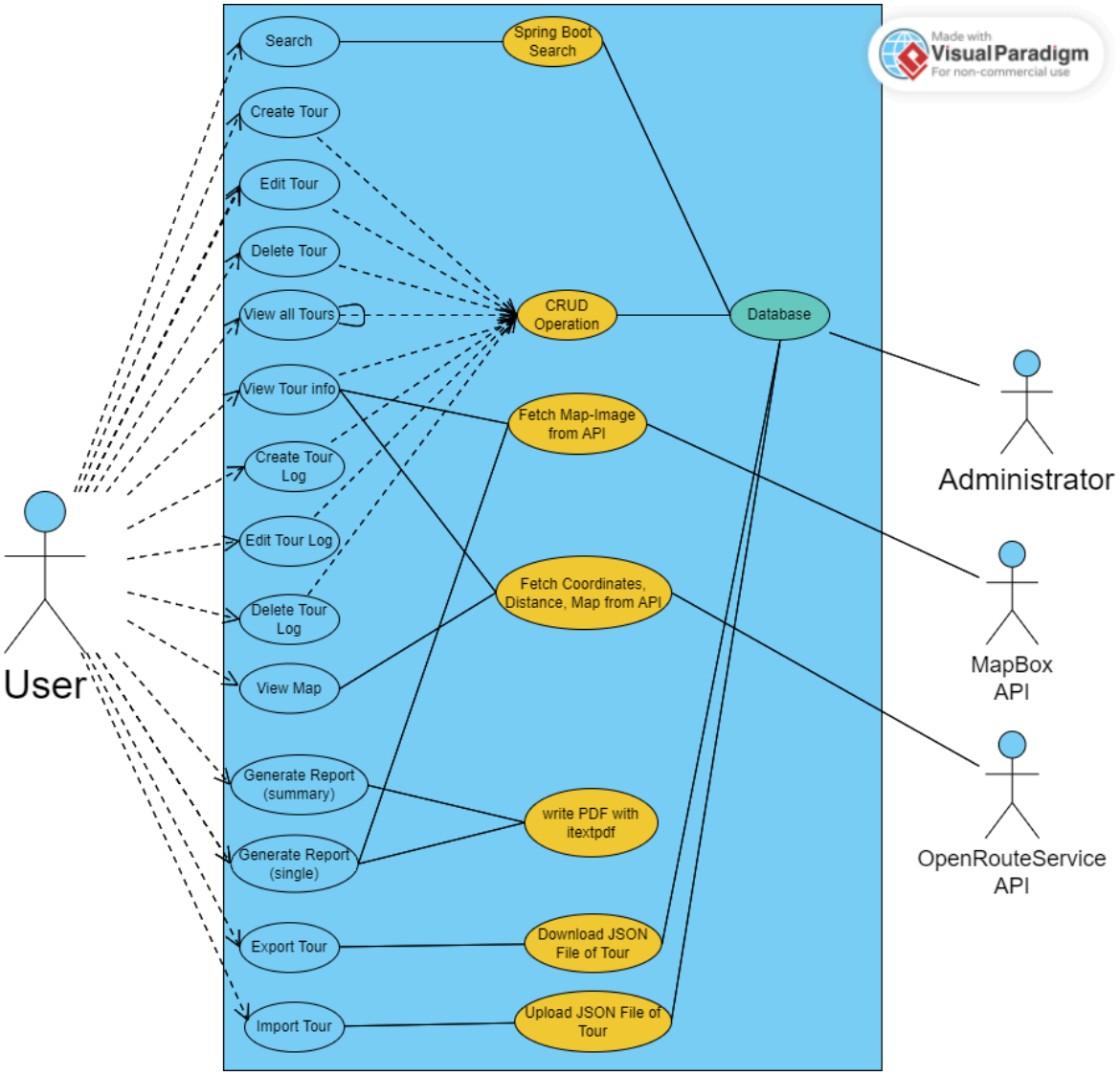
1.9.3. SpringBoot

Das Einbinden von Springboot war eine sehr große Herausforderung. Da wird das Framework erst genutzt haben, als wir die Volltextsuche implementiert haben, war der meiste Code unseres Projektes schon da und musste daher angepasst werden.

Das größere Problem war jedoch die Abstimmung aller Versionen der APIs und Frameworks, die wir schon eingebunden hatten, da nicht alle mit der Springboot Version kompatibel waren. Dies kostete uns einige Stunden.

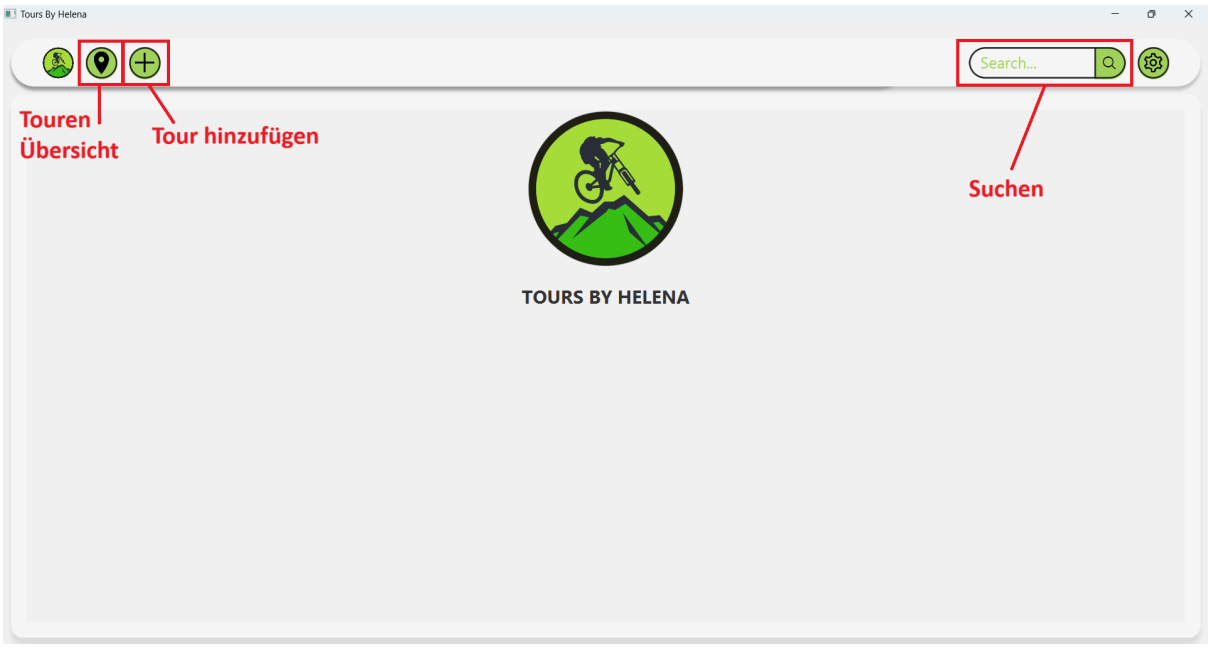
Die Arbeit zahlte sich am Ende jedoch aus, da die Volltextsuche nun einwandfrei und effizient funktioniert.

2. UML Use Case Diagramm



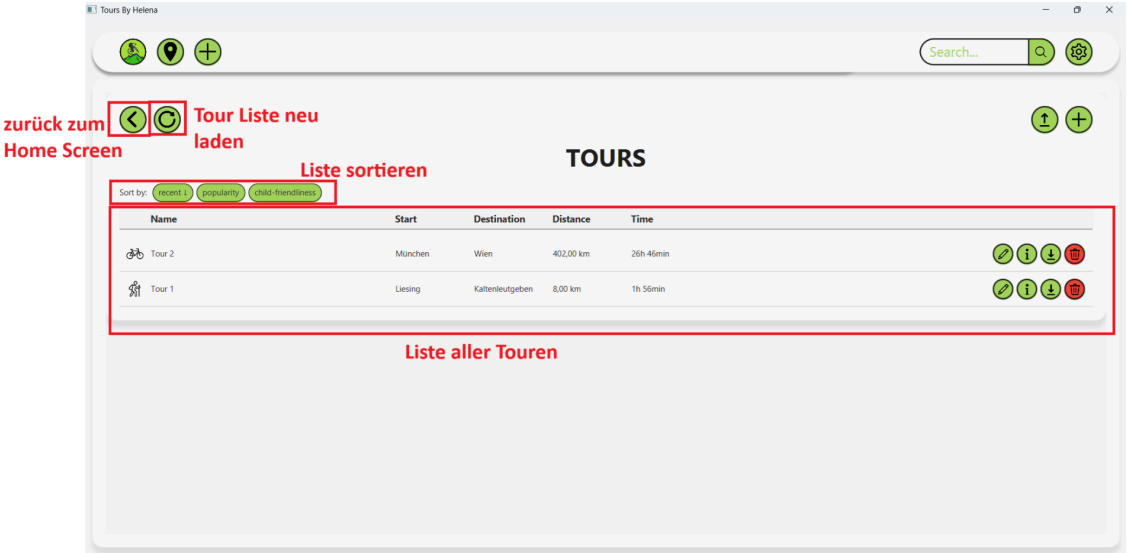
3. UI-Flow & Wireframes

3.1. Home Screen

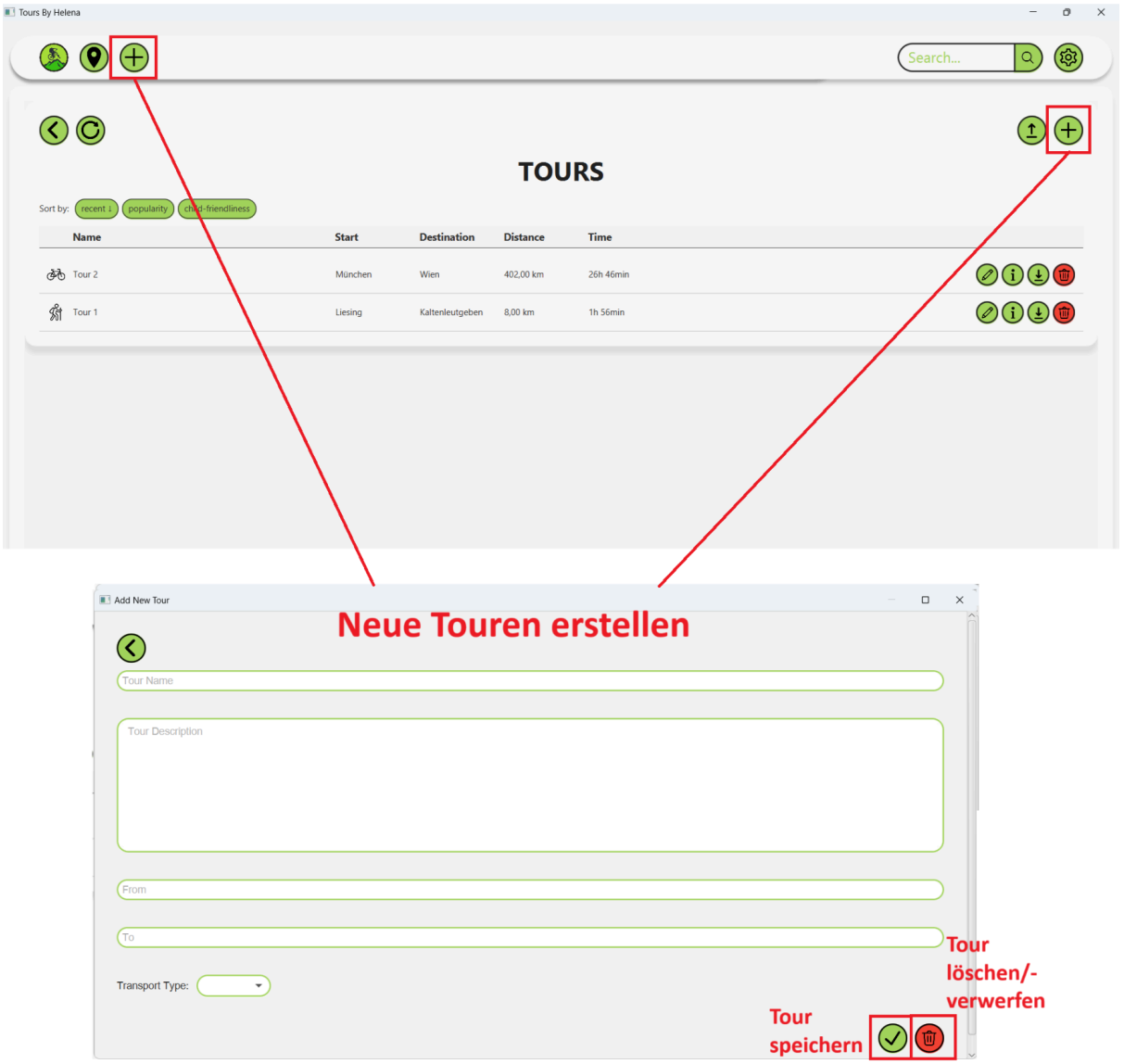


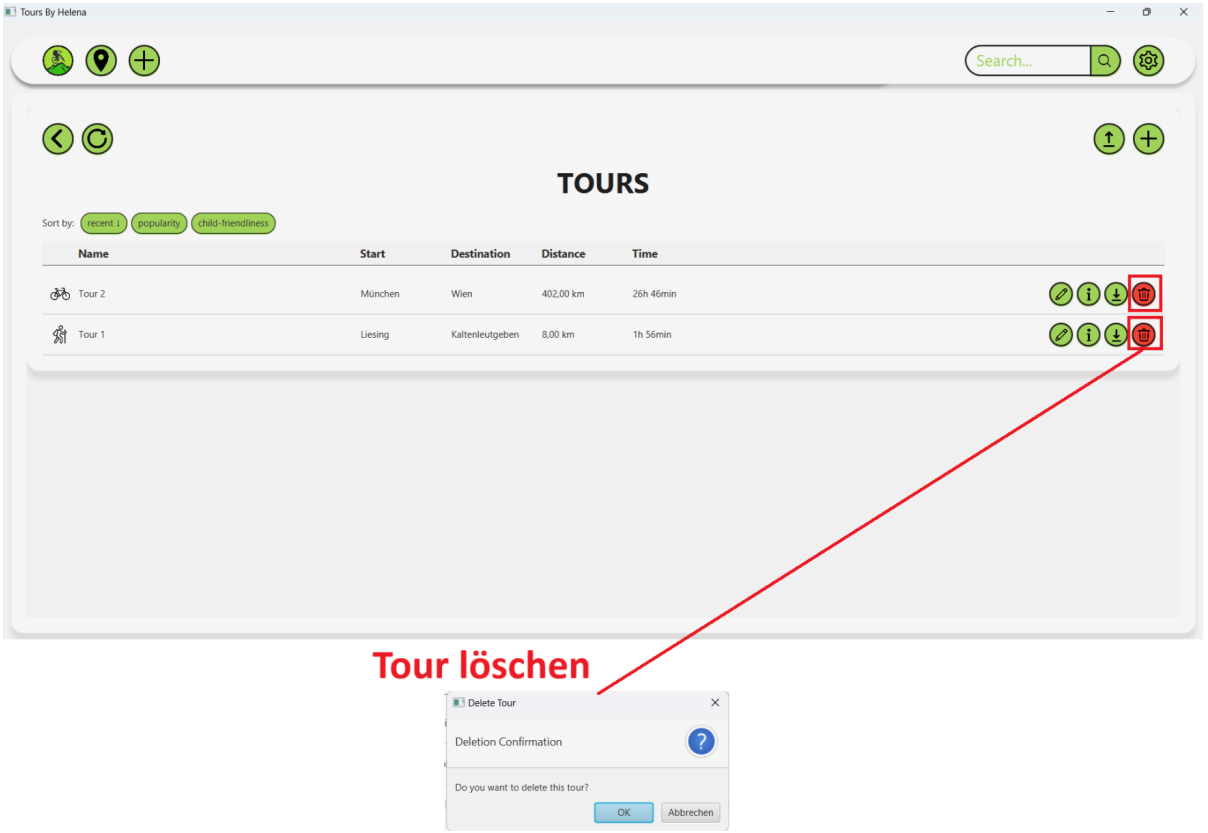
Wenn die Applikation gestartet wird, öffnet sich der Home Screen mit einer Navbar, von der man zur Touren Übersicht kommt, Touren hinzufügen kann und nach Touren, Logs usw. suchen kann. Wir hatten zu beginn vor einige Settings, wie z.B. dark mode einzubauen, daher ist ganz rechts noch der Button für Einstellungen. Dies haben wir jedoch dann aus Zeitdruck nicht mehr machen können.

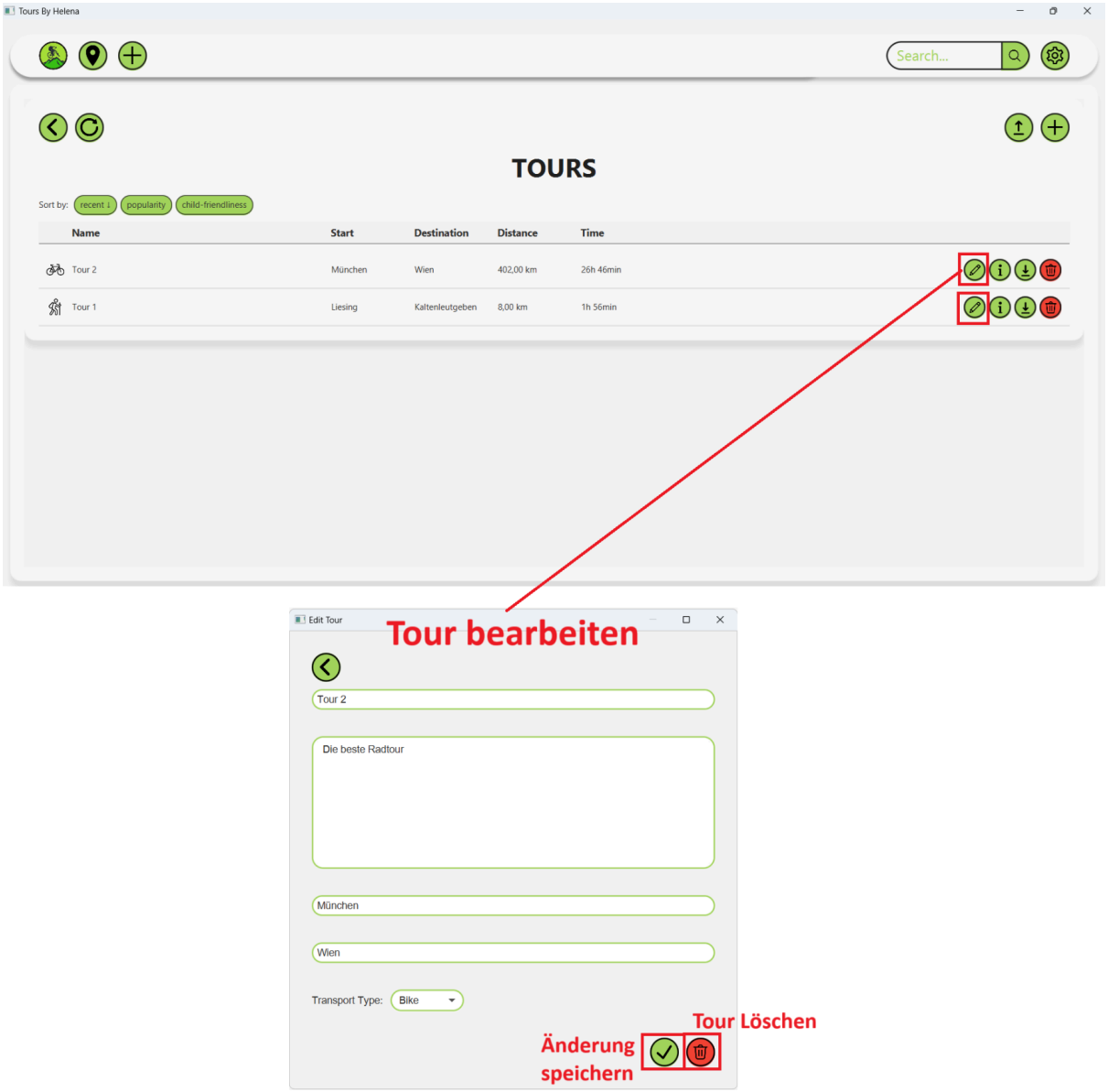
3.2. Touren Übersicht



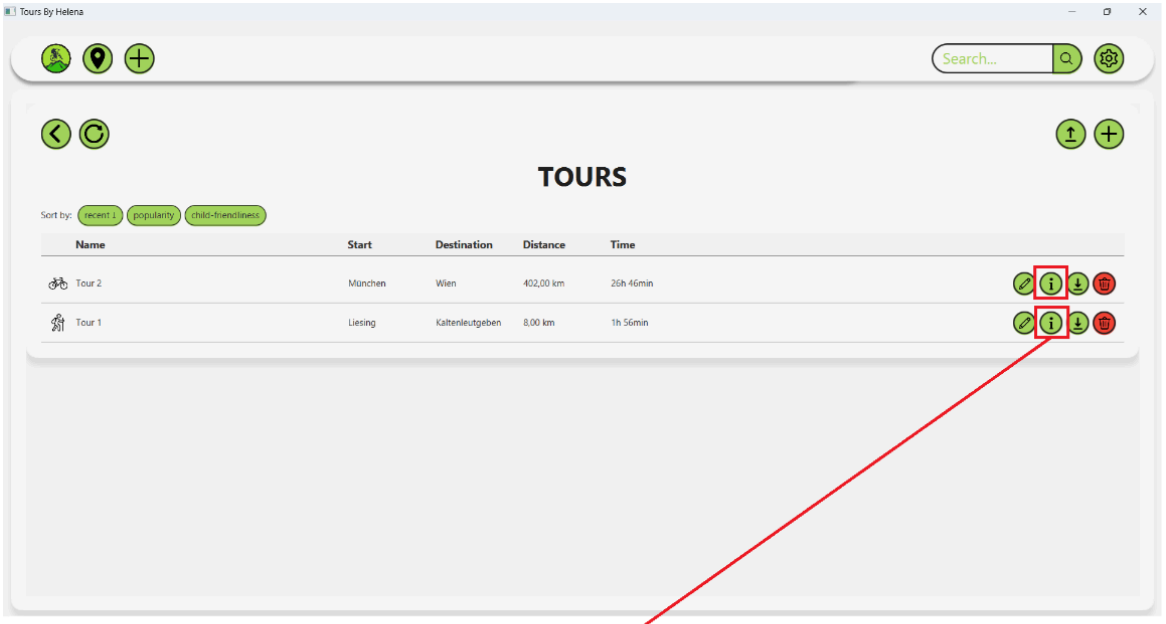
3.3. Touren hinzufügen/bearbeiten/löschen

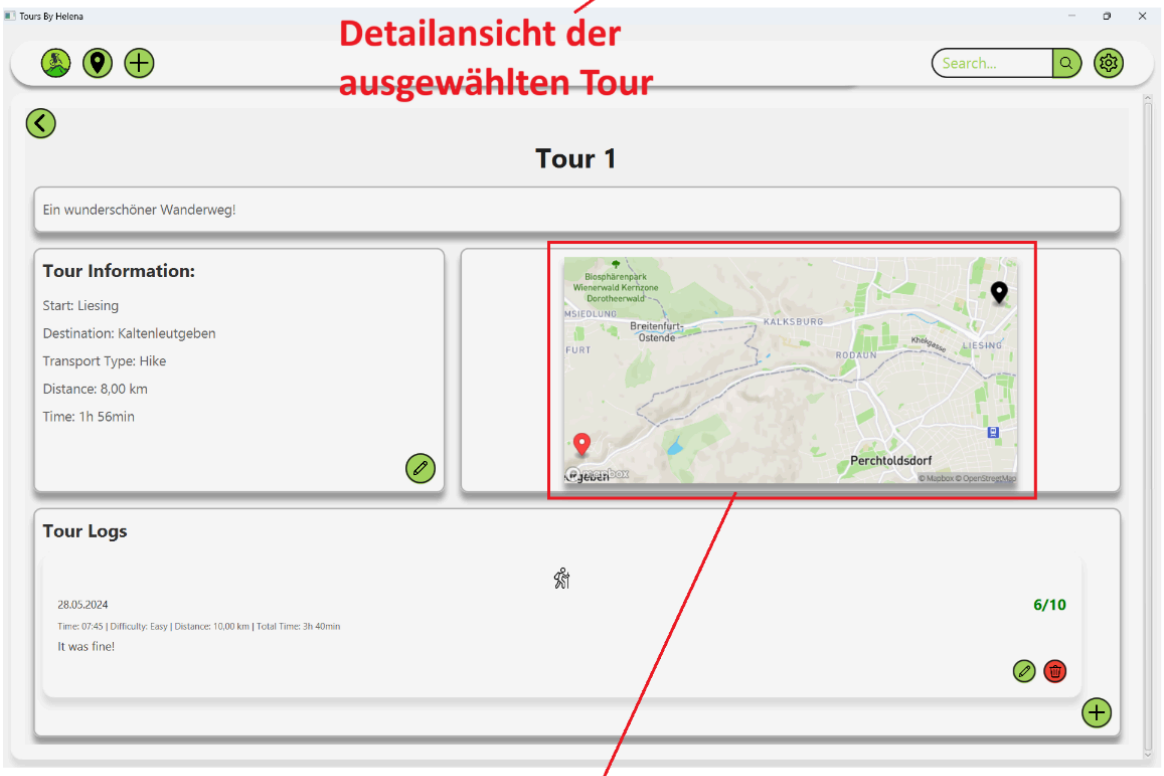


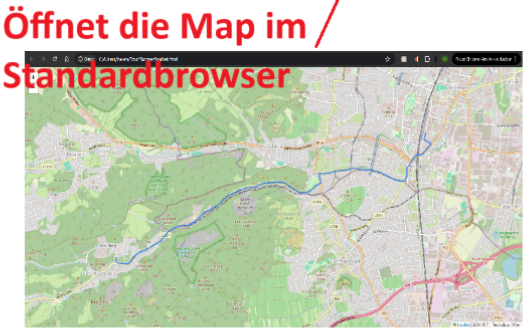

Tours By Helena
Search...
TOURS
Sort by:
recent ↓
popularity
child-friendliness
Name
Start
Destination
Distance
Time
Tour 2
München
Wien
402,00 km
26h 46min
Tour 1
Liesing
Kaltenleutgeben
8,00 km
1h 56min
Tour löschen
Delete Tour
Deletion Confirmation
Do you want to delete this tour?
OK
Abbrechen


Tours By Helena
Search...
TOURS
Sort by:
recent
popularity
child-friendliness
Name
Start
Destination
Distance
Time
Tour 2
München
Wien
402,00 km
26h 46min
Tour 1
Liesing
Kaltenleutgeben
8,00 km
1h 56min
Edit Tour
Tour bearbeiten
Tour 2
Die beste Radtour
München
Wien
Transport Type:
Bike
Tour Löschen
Änderung speichern

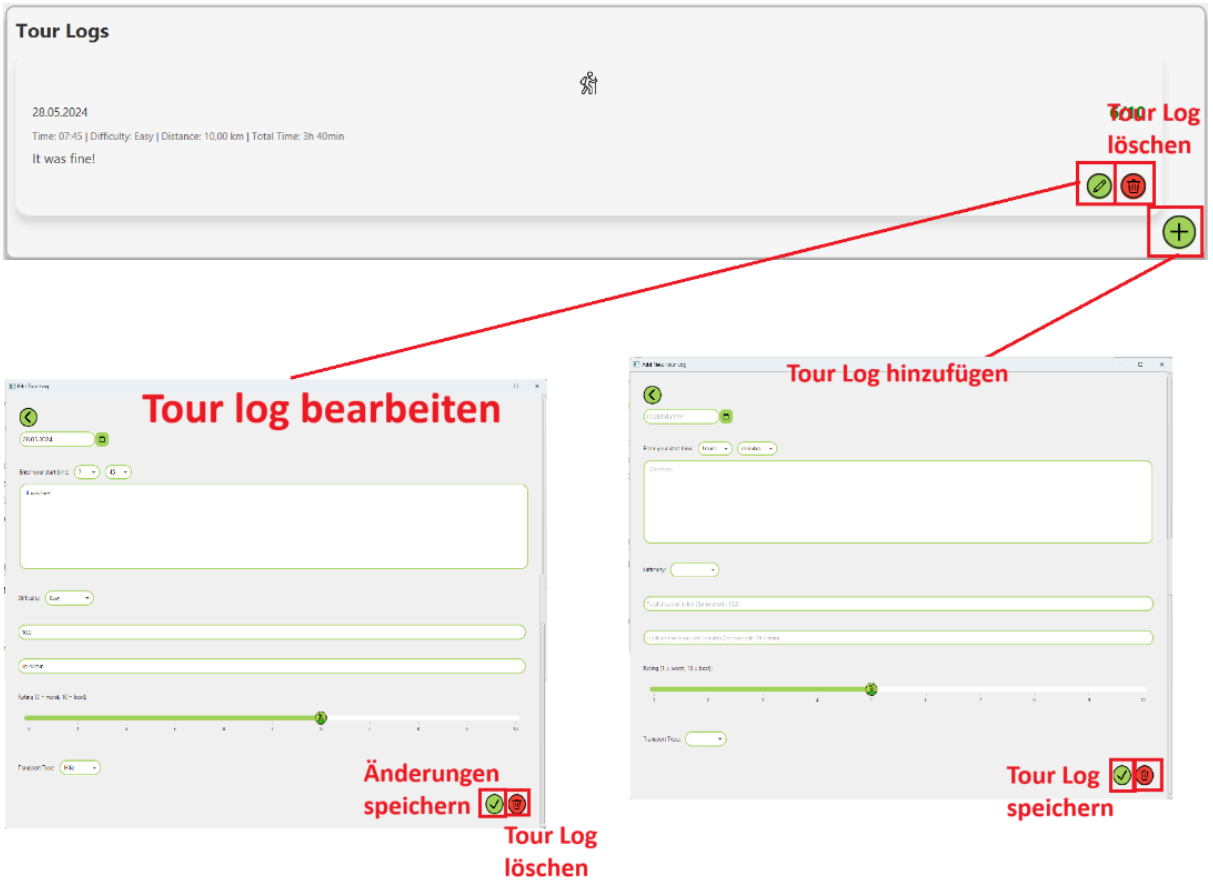
3.4. Tour Detailansicht



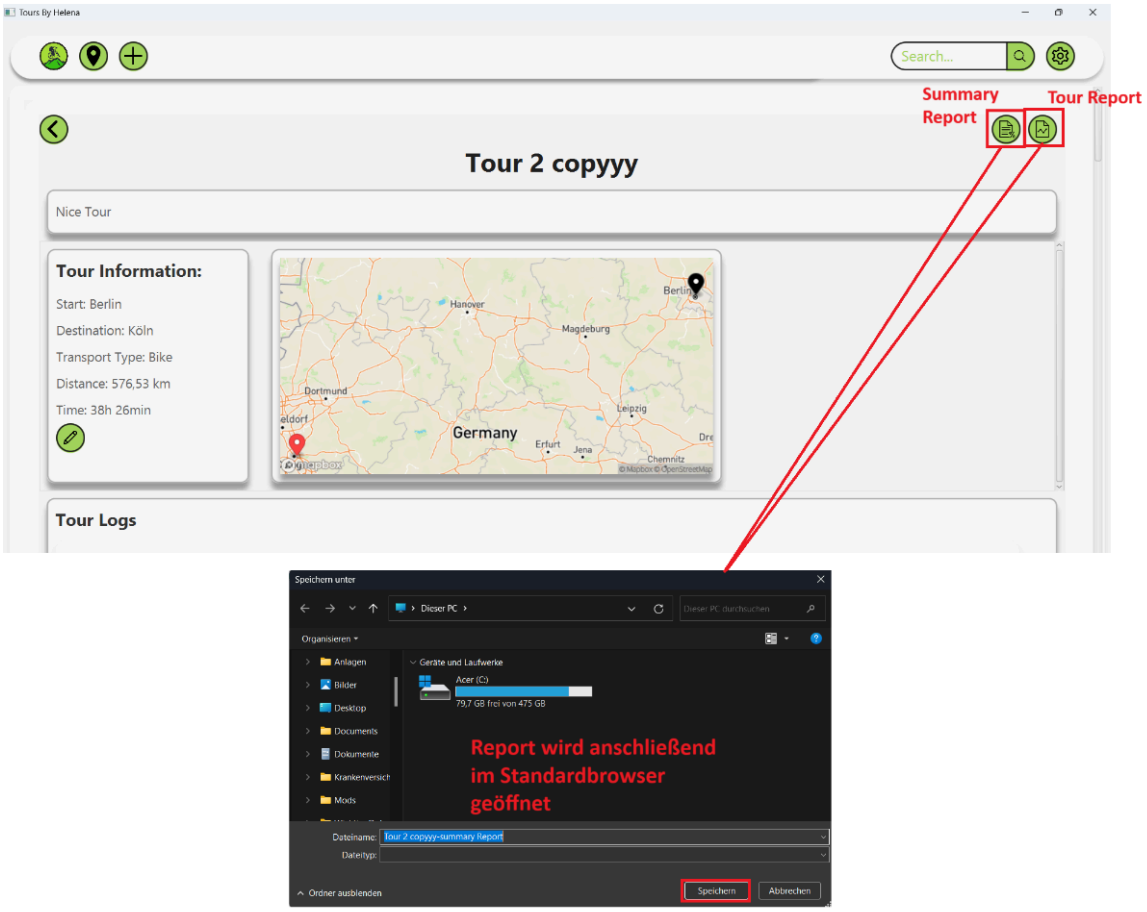




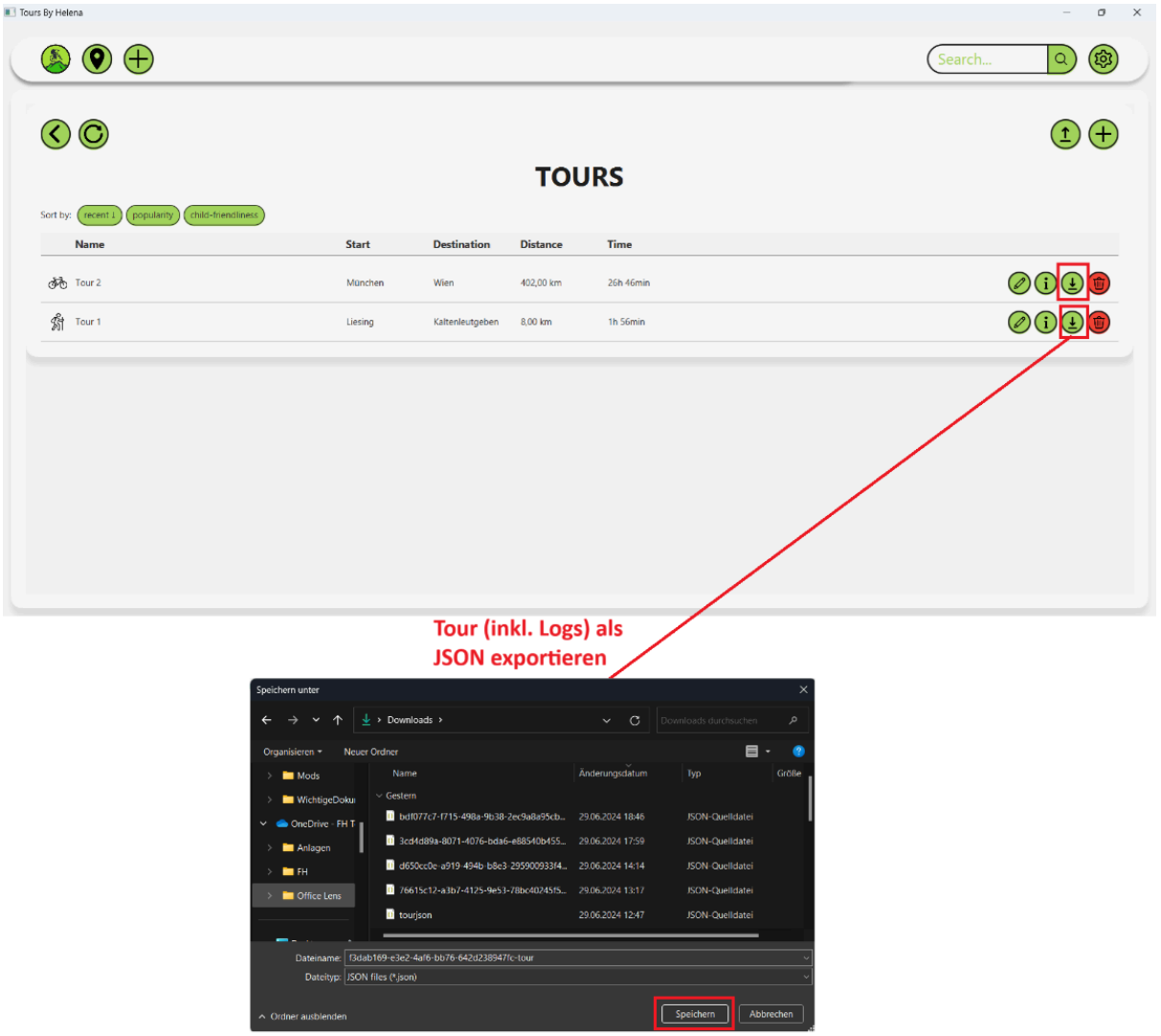
3.5. Tour Logs hinzufügen/bearbeiten/löschen

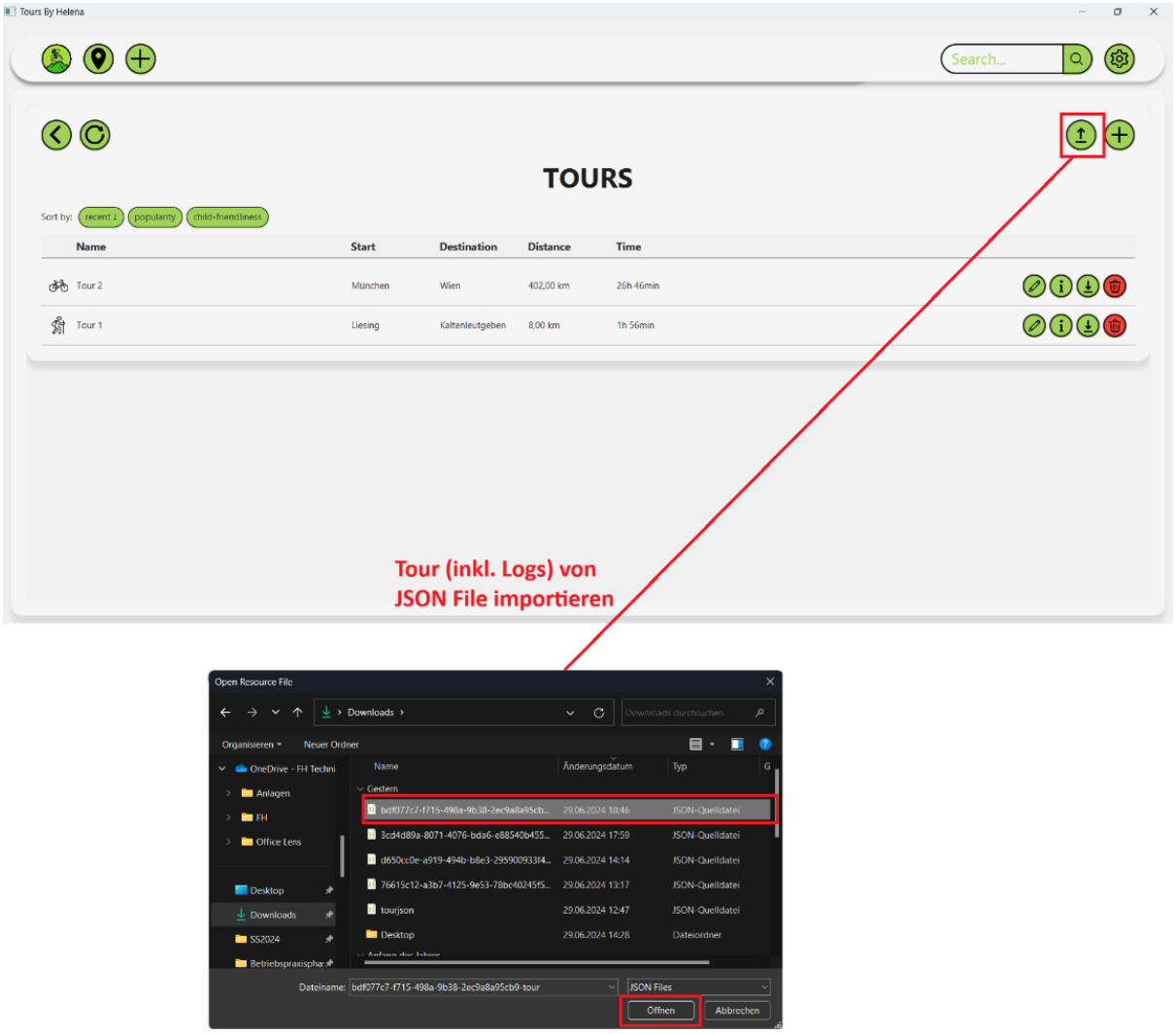


3.6. Reports herunterladen


Tours By Helena
Search...
Summary Report
Tour Report
Tour 2 copyyy
Nice Tour
Tour Information:
Start: Berlin
Destination: Köln
Transport Type: Bike
Distance: 576,53 km
Time: 38h 26min
Tour Logs
Hanover
Magdeburg
Berlin
Dortmund
Leipzig
Germany
Erfurt
Jena
Chemnitz
Speichern unter
Dieser PC
Organisieren
Anlagen
Bilder
Desktop
Documents
Dokumente
Mods
Geräte und Laufwerke
Acer (C:)
79,7 GB frei von 475 GB
Report wird anschließend im Standardbrowser geöffnet
Dateiname:
Tour 2 copyyy-summary Report
Dateityp:
Ordner ausblenden
Speichern
Abbrechen

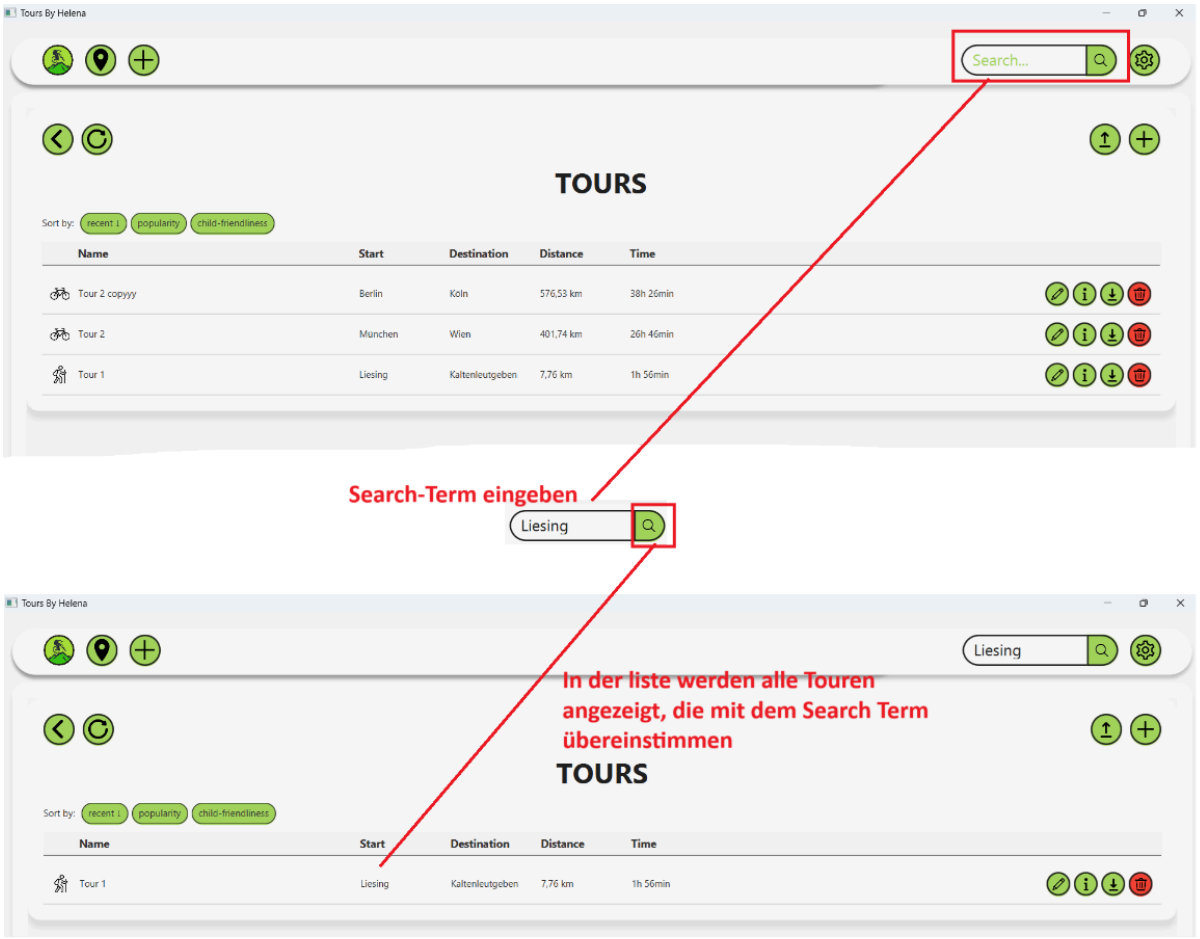
3.7. Import/Export von Touren




Tours By Helena
Search...
TOURS
Sort by:
recent
popularity
child-friendliness
Name
Start
Destination
Distance
Time
Tour 2
München
Wien
402,00 km
26h 46min
Tour 1
Liesing
Kaltenleutgeben
8,00 km
1h 56min
Tour (inkl. Logs) von
JSON File importieren

Open Resource File
Downloads
Organisieren
Neuer Ordner
Name
Änderungsdatum
Typ
Gestern
tourjson
29.06.2024 12:47
JSON-Quelldatei
Desktop
29.06.2024 14:28
Dateiordner
Dateiname:
JSON Files
Öffnen
Abbrechen

3.8. Suchen

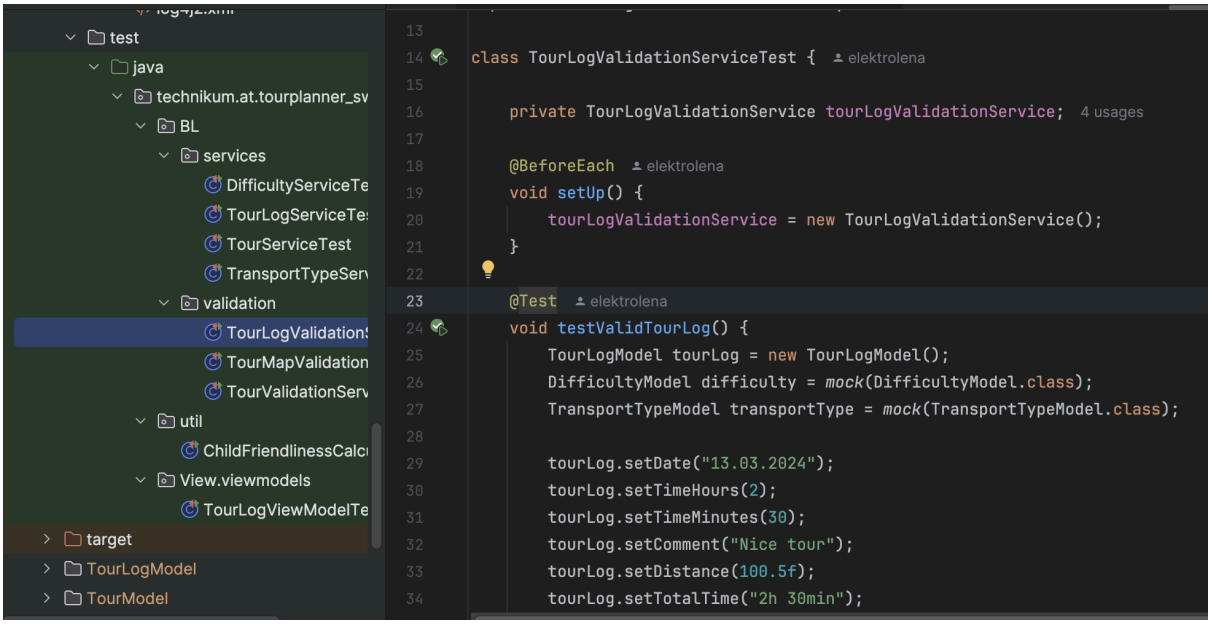


4. Architecture UML Diagram

https://github.com/helhar1234/TourPlanner_SWEN2_Team5/blob/main/src/docs/class%20diagramm.png
(siehe Beschreibung in Technical Steps and Decisions)

5. Unit Testing

Bei den Unit Tests liegt ein klarer Fokus auf der Business Layer des Projektes. Vor allem Validierungsklassen und Serviceklassen wurden hierbei ausführlich getestet.



5.1. Validierungen

Um eine reibungslose Benutzererfahrung zu gewährleisten und Probleme durch fehlerhafte Eingaben zu vermeiden, haben wir die Validierungen mit verschiedenen Eingaben ausführlich getestet. Dies stellt sicher, dass die Anwendung robust gegen ungültige oder unerwartete Benutzereingaben ist.

5.2. Services

Services sind eine wichtige Schnittstelle zwischen Repositories und ViewModels. Daher ist es essentiell, dass sie einwandfrei funktionieren. Um dies sicherzustellen, haben wir die Services gründlich getestet, um deren Zuverlässigkeit und Korrektheit zu gewährleisten.

5.3. ViewModels

ViewModels spielen eine entscheidende Rolle bei der Darstellung der Daten im UI und der Interaktion mit den Services. Deshalb ist es wichtig, auch die ViewModels gründlich zu testen, um sicherzustellen, dass sie korrekt arbeiten und eine nahtlose Benutzererfahrung ermöglichen.

5.4. Child-Friendliness Rechner

Der Child-Friendliness Rechner berechnet die Kinderfreundlichkeit einer Tour. Sein Aufbau ist etwas komplizierter und daher auch fehleranfällig. Um eine sinnvolle Berechnung der Kinderfreundlichkeit zu gewähren, haben wir ihn mit unterschiedlichen “Eingaben” per Unit Tests getestet.

6. Time Tracking

In unserem Projekt hatten wir anfangs keine großen Schwierigkeiten mit der Zeitorganisation, da wir bereits ein eingespieltes Team waren. Ein herausforderndes Problem ergab sich jedoch aus der Taktung unserer Präsenzstunden, die dreimal pro Woche stattfanden. Diese enge Terminierung ließ oft zu wenig Raum, um das Erlernte umzusetzen, bevor bereits das nächste Thema anstand. Wir lösten dieses Problem, indem wir in Intervallen arbeiteten. Manchmal nutzten wir ein ganzes Wochenende, um intensiv am Projekt zu arbeiten, was uns ermöglichte, im Unterricht wieder auf dem aktuellen Stand zu sein. Anfangs bearbeiteten wir die meisten Aufgaben gemeinsam, bis zur Implementierung des MVVM. Danach übernahm Helene die Weiterentwicklung bis zur Implementierung der Karte, woraufhin Elena das Projekt bis zum Abschluss weiterführte. Dieses Vorgehen wählten wir, um uns nicht gegenseitig bei der Arbeit zu behindern, da ein paralleles Arbeiten in diesen Phasen schwierig war. Trotzdem hielten wir uns stets über Fortschritte, Fehler und Bugs auf dem Laufenden. **Im Durchschnitt arbeiteten wir etwa 6 Stunden pro Woche pro Person am Projekt. In den letzten 2 Wochen waren es dann ca. 30 Stunden pro Woche pro Person.**

Zusatzinformation zu Git: In unserem Git-Repository ist ersichtlich, dass Helene deutlich mehr Zeilen Code gepusht hat. Dies liegt jedoch daran, dass sie anfangs versehentlich auch die Target-Files hochgeladen hat, da ein Problem mit der .gitignore-Datei vorlag. Dieses Problem wurde behoben, als uns der Fehler auffiel.

7. Git-Repository

https://github.com/helhar1234/TourPlanner_SWEN2_Team5
README.md lesen für Informationen bzgl. Ausführung der Applikation.