Tour Planner – Final Protocol

Elaine Fink, Patrick Florea
Technikum Wien – SWEN 2
Date: 6.7.2025

Neue Funktionen seit Intermediate Hand-in:

Das hier ist das Final Protokoll unseres SWEN 2 Semester Projects. Seite dem Intermediat Hand In haben wir folgende Funktionen hinzugefügt bzw. verbessert:

- Verbesserte UI-Feldbindung & Eingabevalidierung
- Datenbank Anbindung an PostgreSQL mit Spring Boot und Hibernate
- Kartenanzeige mit Leaflet & OpenRouteService
- Automatisch berechnete Tourattribute:
    - Popularität (basierend auf Log-Anzahl)
    - Kinderfreundlichkeit (aus Schwierigkeit, Zeit, Distanz)
- JSON-Import/Export
- Zwei Berichtstypen:
    - Tour-Report (alle Tour-Infos + Logs)
    - Summary-Report (Durchschnittswerte aller Touren)
- Logging
- Eine zentrale config-Datei (DB-Zugang, API-Key etc.)
- Unique Feature Zufällige Tourauswahl ("Random Tour")
- Wir haben keine Search Funktion eingebaut

1. Application Architecture

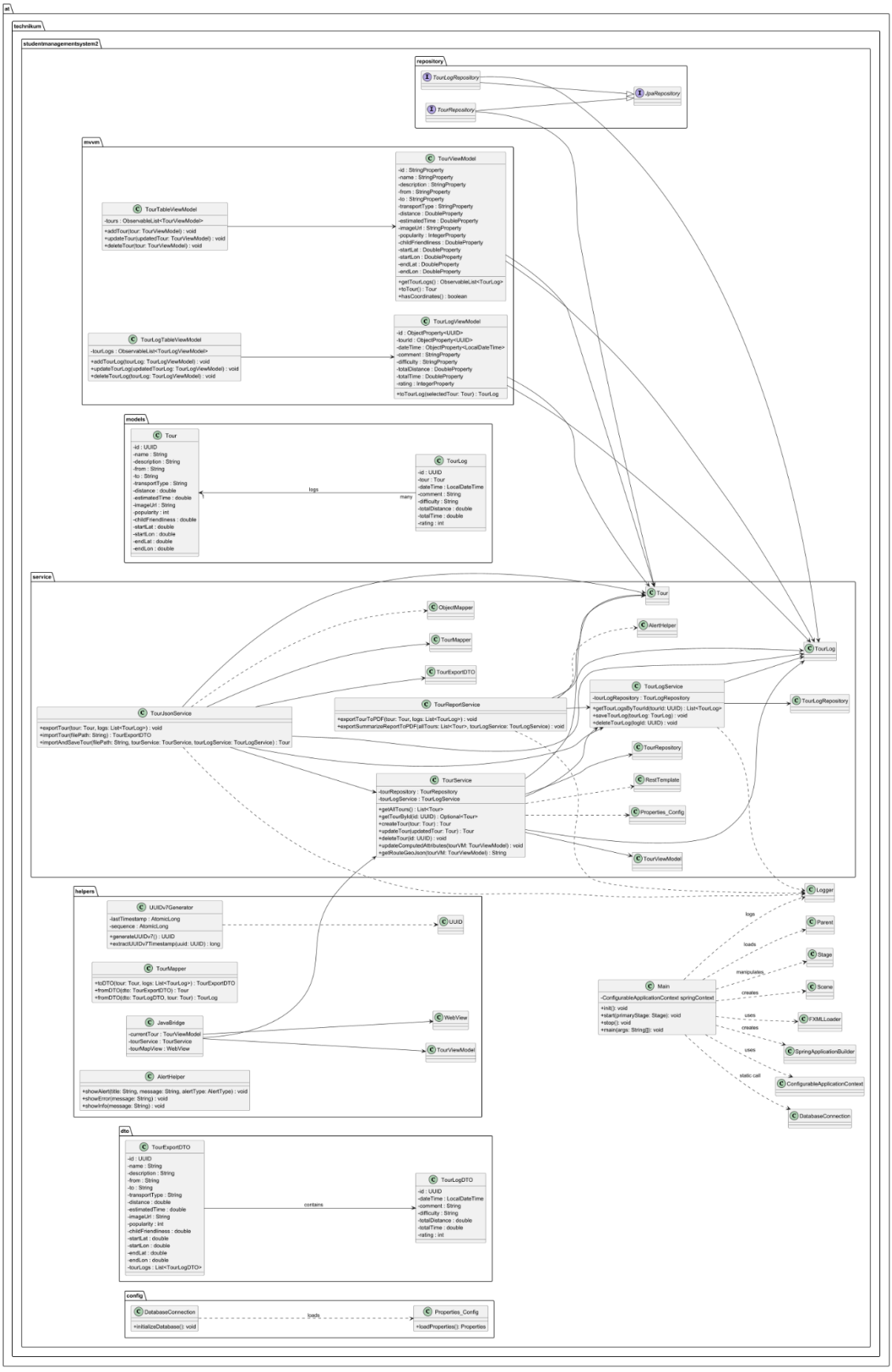
Layer Overview:

Die Anwendung folgt einer Layer-basierten Architektur mit folgenden Schichten:

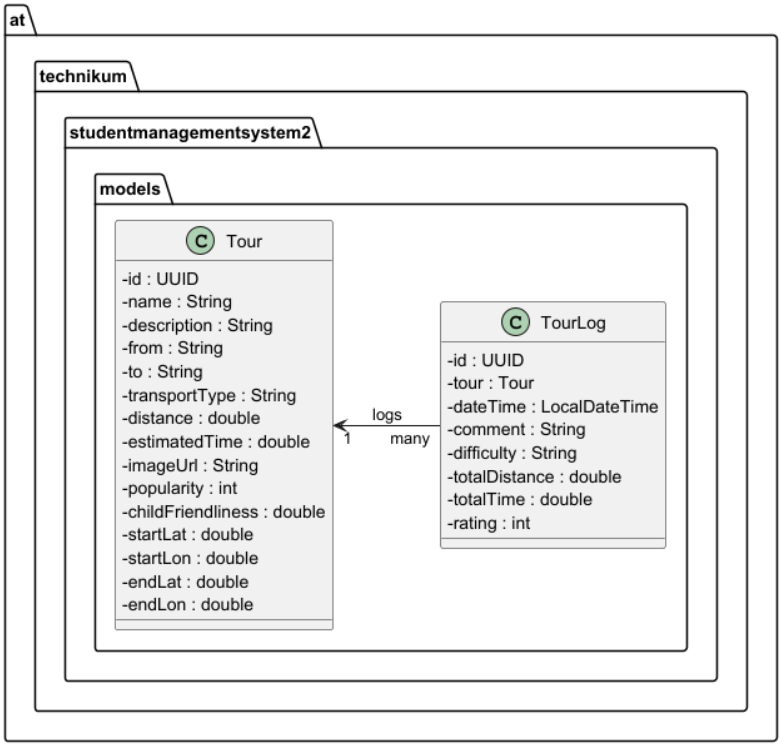
- FXML Views – UI Layouts
    - Main.fxml für das Hauptfenster (Tourliste und Tour Detailansicht mit Log Tabelle)
    - Jeweils ein Pop Up Fenster zum Erstellen/Bearbeiten von Touren oder Tour Logs
- ViewModels – Datenbindung & Logik zwischen UI und Controller
    - Jeweils ein ViewModel für Touren und Tour Logs
    - Ein Viewmodel für die TourTabelView
    - Eine Viewmodel für die TourLogTabelView
- Controllers – Event-Handling
    - Main Controller - leitet alle anafragen aus dem Main.fxml an die Services weiter, startet aufrufe der Pop Up Fenster
    - Jeweils einen Controller für Tour und Tour Log Edit/New Dialog
- Services – Geschäftslogik
    - Tour Service – Verarbeitet die Anfragen zu Tour Daten und holt daten aus dem Tour Repository
    - Tour Log Service – Verarbeitet die Anfragen zu Tour Log Daten und holt Daten aus dem / leitet weiter an das Tour Log Repository
    - Tour Service – Verarbeitet die Anfragen zu Tour Daten und holt Daten aus dem / leitet weiter an das Tour Repository
    - Tour JSON Service – Verarbeitet Anfragen zum Import und Export von Touren als JSON und holt Daten aus dem / leitet weiter an das Tour Service und Tour Log Service
    - Tour Report Service – Verarbeitet Anfragen zum Erstellen von Tour Reports und Summary Reports und holt sich die Daten aus dem Tour Service und Tour Log Service.
- Repositories– Datenbankzugriffe (via Springboot und Hibernate)
    - Tour Repository – liefert Tour Daten aus der DB an das Tour Service
    - Tour Log Repository – liefert Tour Log Daten aus der DB an das Tour Log Service
- PostgreSQL Datenbank – Speichert Tour und Tour Log Daten in Relationalem Datenmodel

Class Diagram:

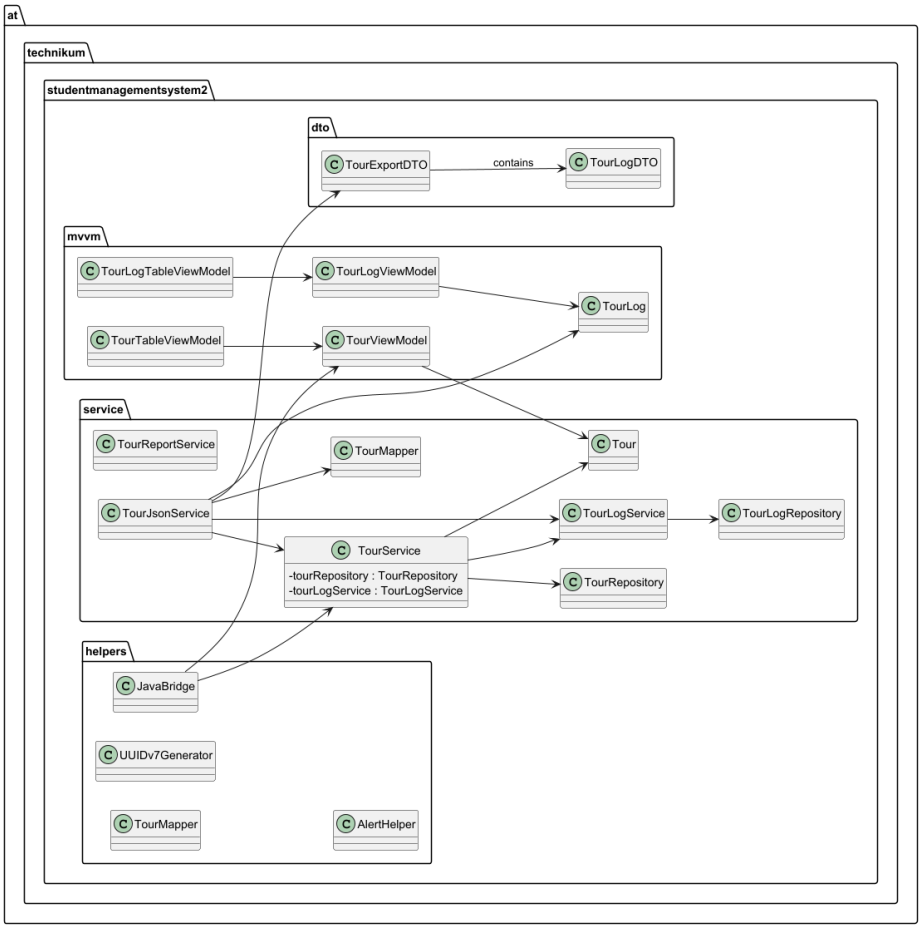
Klassendiagramm Gesammtübersicht



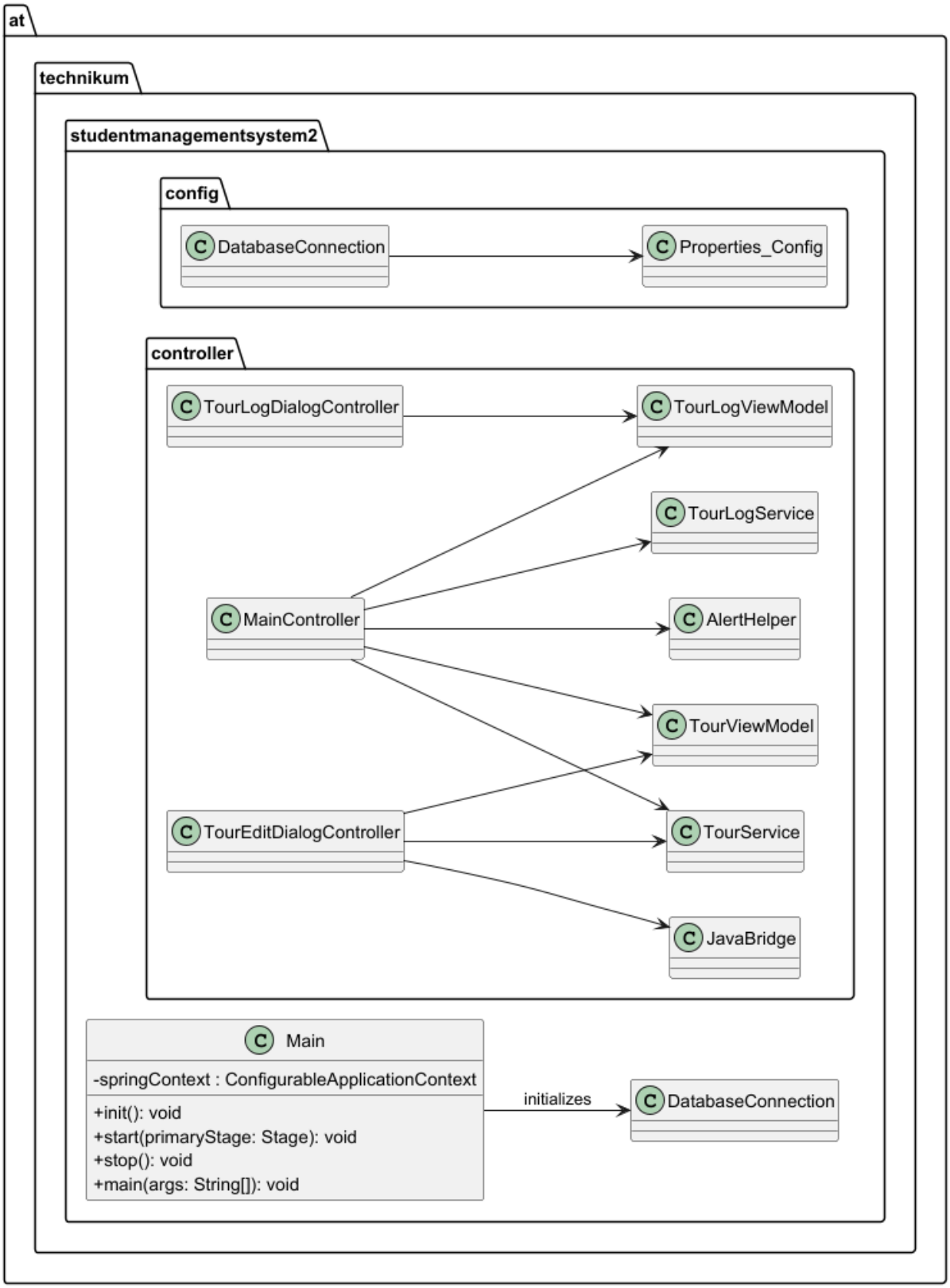
Domain Model Übersicht



Services MVVM Übersicht

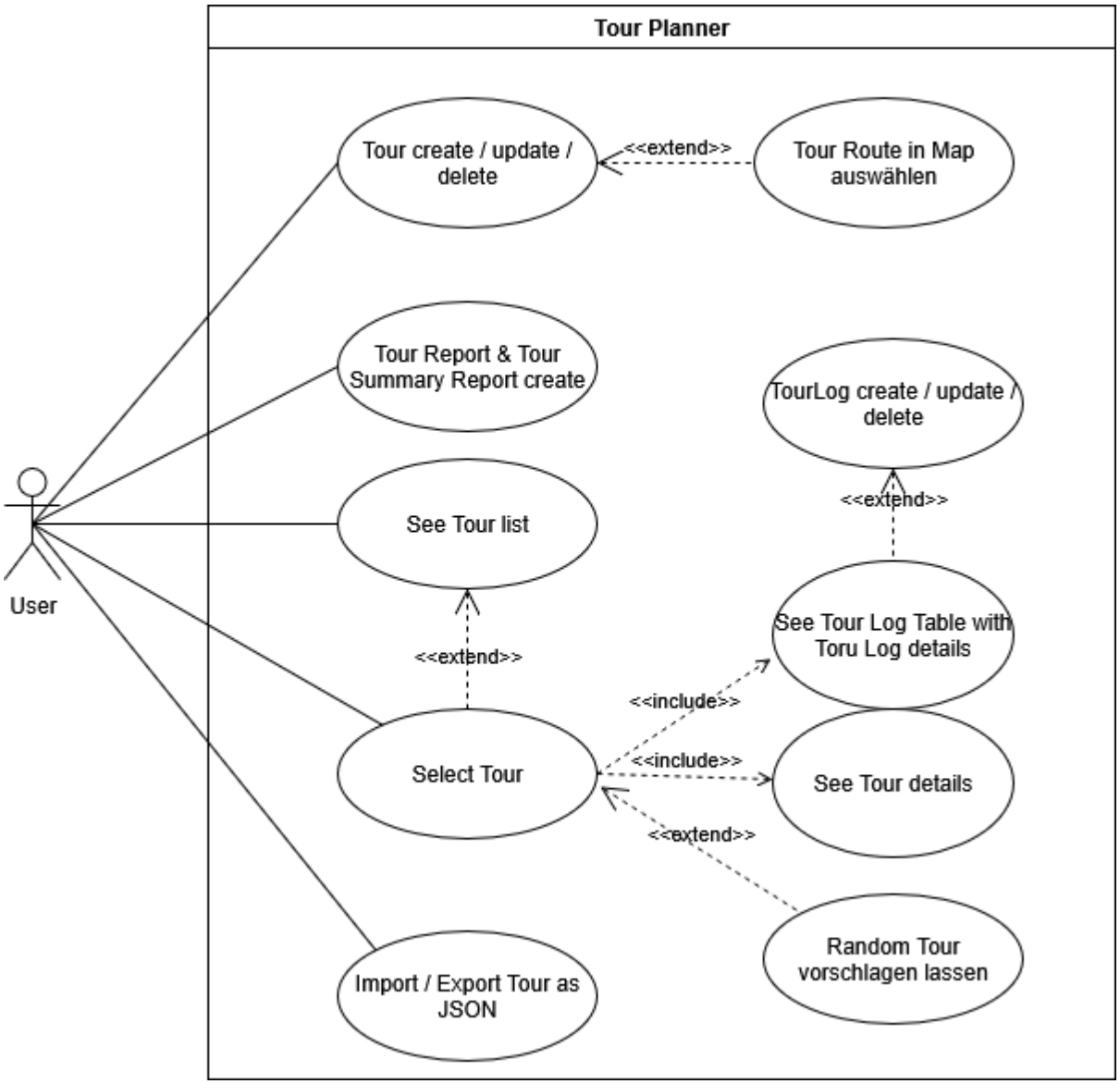


App UI Config Übersicht



2. Use Cases

Use-Case Diagramm:



Use Cases:

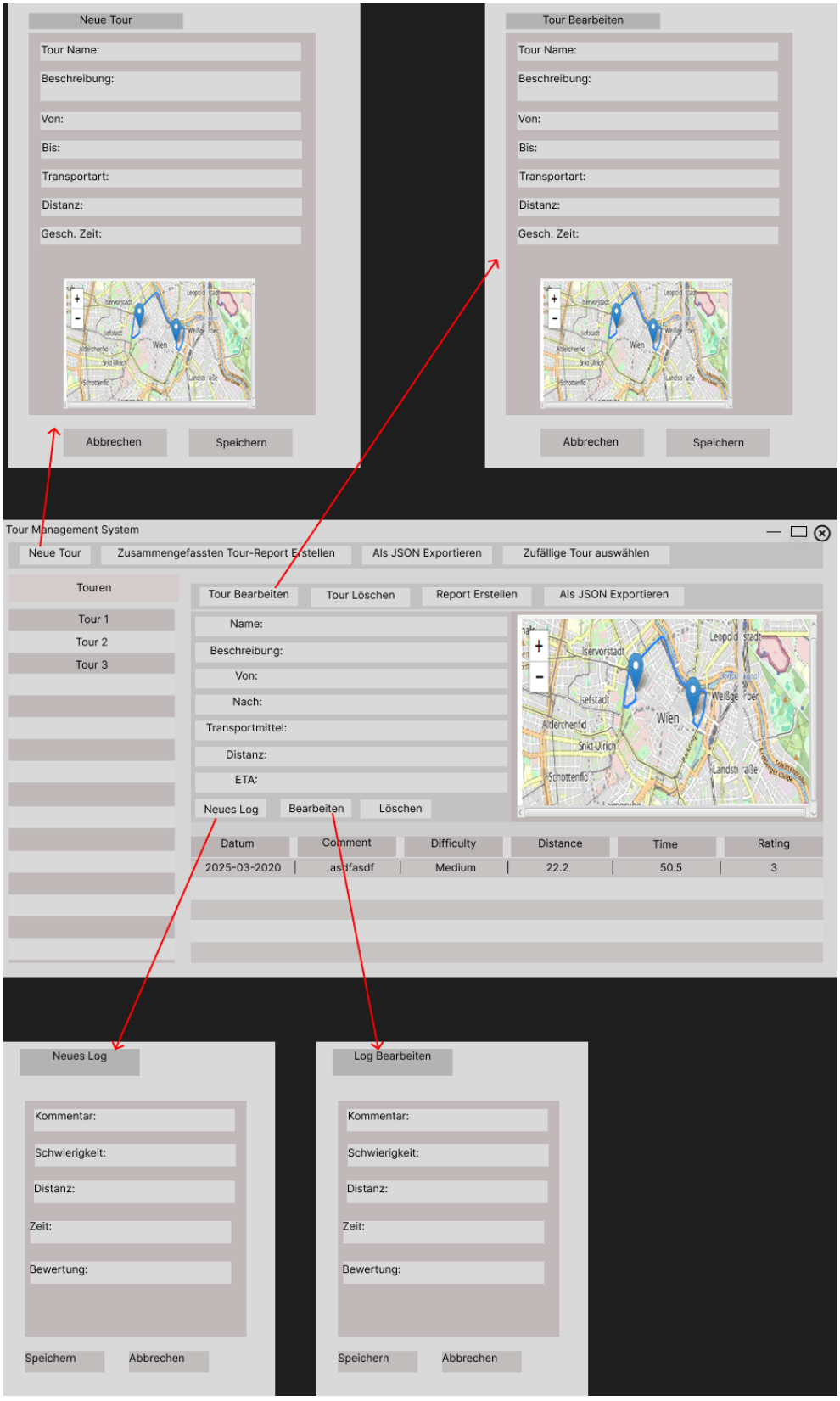
- Tour create / update / delete
  - Benutzer kann Touren anlegen, bearbeiten oder löschen.
  - Erweiterung: Tour Route in Map auswählen (Leaflet-basiertes Routing bei Erstellung/Bearbeitung)
- Select Tour
  - Ermöglicht dem Benutzer, eine Tour aus der Liste auszuwählen.

  - Inkludiert See Tour details & See Tour Log Table with Tour Log details
  - Erweitert Random Tour vorschlagen lassen (unique Feature)
- See Tour list
  - Der User kann eine Übersicht aller Touren sehen.
  - Erweiterung: durch Auswahl einer Tour (Select Tour)
- TourLog create / update / delete
  - Logs zu einer ausgewählten Tour können erstellt, bearbeitet oder gelöscht werden.
  - Abhängig von: ausgewählter Tour (da Logs zur Tour gehören)
- Tour Report & Tour Summary Report create
  - Generierung eines Berichts für eine einzelne Tour oder einer zusammenfassenden Übersicht über alle Touren.
  - Zugriff nach Auswahl einer Tour oder für alle Touren (Summary)
- Import / Export Tour as JSON
  - Ermöglicht das Speichern oder Laden von Tourdaten im JSON-Format.
  - Export einzelner Touren erfolgt nach Auswahl.
- Tour Route in Map auswählen
  - Wird nur im Rahmen von Tour-Erstellung/Bearbeitung aufgerufen
  - Auswahl von zwei Punkten via Leaflet-Karte; Route wird generiert, Screenshot wird gespeichert.
- See Tour details
  - Zeigt alle Tourdaten: Name, Distanz, Beschreibung, Route (als Map-Bild), sowie berechnete Werte wie Childfriendliness und Popularity.
- See Tour Log Table with Tour Log details
  - Detaillierte Tabelle aller Logs zur ausgewählten Tour (inkl. Zeit, Bewertung, Schwierigkeit etc.)
- Random Tour vorschlagen lassen
  - Erweiterung von Select Tour
  - Liefert eine zufällige Tourauswahl und ist unser unique Feature.

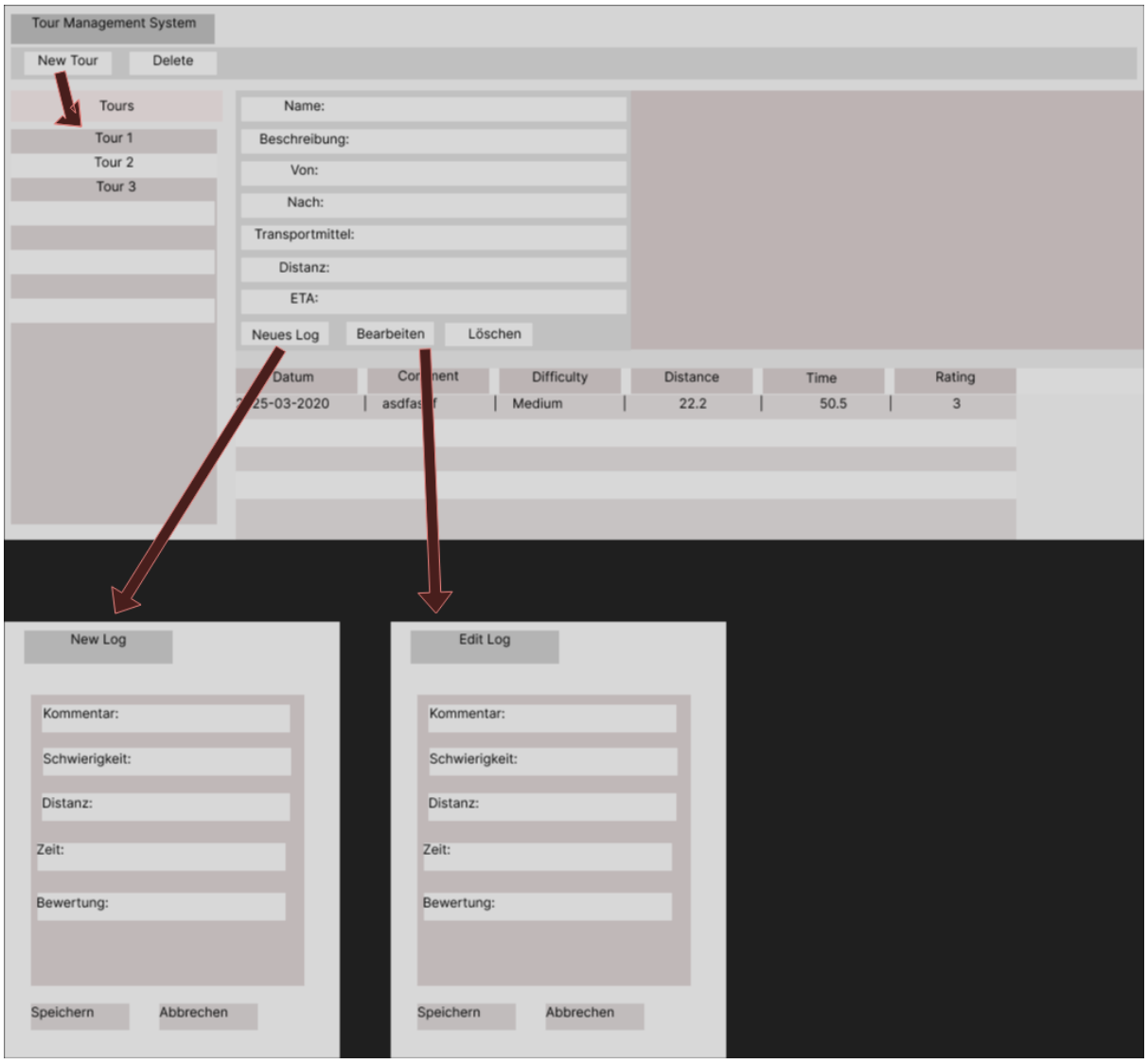
3. UX – User Experience

Wireframes:

Neue Wireframes



Alte Wireframes vom Intermediat Handin



4. Verwendete Technologien

- Java
  Hauptsprache für die Implementierung. Vorgeben von der Lehrveranstaltung.
- MVVM (Model-View-ViewModel)
  Architekturpattern zur Trennung von UI, Logik und Daten. Wird in Kombination mit JavaFX umgesetzt für bessere Testbarkeit und Strukturierung.
- JavaFX (mit FXML)
  Für die Desktop-Benutzeroberfläche. Unterstützt FXML als markup-basierte UI-Beschreibung und ist ideal für MVVM.
- Spring Boot
  Framework zur Erstellung des Backend. Bietet einfache Konfiguration, REST-Support und klare Trennung der Schichten.
- Hibernate (JPA)
  Objekt-relationales Mapping (ORM) zur einfachen Datenbankanbindung. Verhindert SQL-Injection durch automatische Parameterbindung.
- PostgreSQL
  Relationale Datenbank zur Speicherung aller Tourdaten, Logs, Metriken und Screenshots.
- Leaflet.js
  JavaScript-Bibliothek zur Darstellung und Interaktion mit Karten (in WebView eingebunden).
- OpenRouteService API
  Zum Berechnen von Routen auf Basis ausgewählter Start- und Zielpunkte auf der Karte.
- html2canvas
  Zum Erzeugen eines Screenshots der Tour-Route, der als statisches Bild in der Tour gespeichert und angezeigt wird.
- Jackson
  Bibliothek zur Serialisierung und Deserialisierung von JSON-Daten. Ermöglicht Tourdaten-Import und -Export. Wir haben uns für JSON mit Jackson entschieden da es ein leicht einzulesendes Format ist und geschachtelte Strukturen unterstützt (Logs bei Touren).
- iTextPDF
  Bibliothek zur Generierung von PDF-Dateien. Wird verwendet für Tour-Reports und Summary-Reports.
- log4j2
  Logging-Framework zur Protokollierung von Systemereignissen, Fehlern und Benutzeraktionen in verschiedenen Log-Leveln.
- Junit
  Framework für Unit Tests.

5. Unit Testing

Wir haben Unit Tests von den Services und den MVVM-Modellen mit JUnit umgesetzt.

Service-Schicht (Business-Logik)

Hier wird die Geschäftslogik von Touren und Tour Logs getestet – also z. B. das Erstellen, Bearbeiten und Löschen von Touren und Logs. Diese Schicht ist entscheidend, weil sie unabhängig von der UI-Logik ist und die eigentliche Funktionalität der Anwendung trägt.

Getestete Dateien: Tour Service, Tour Log Servcie, Tour Json Service

MVVM-Modelle (UI-Logik)

Hier testen wir die View Models, die die Verbindung zwischen UI und Geschäftslogik herstellen. Die Tests stellen sicher, dass Benutzeraktionen korrekt abgebildet werden, z. B. das Hinzufügen und Entfernen von Touren in Tabellenansichten.

Getestete Dateien: Tour View Model, Tour Log View Model, Tour Table View Model, Tour Log Table View Model

6. Unique Feature - Random Tour Generator

Der Nutzer kann sich per Klick auf den Button „zufällige Tour auswählen“ eine zufällige Tour vorschlagen lassen. Das soll zur Entdeckung neuer Touren motivieren.

7. Design Pattern – Repository Pattern

Verwendetes Muster:

Die Anwendung implementiert das Repository Design Pattern, um die Datenzugriffsschicht klar von der Geschäftslogik zu trennen. Dadurch bleibt die Architektur modular, testbar und erweiterbar. Unsere Services greifen selbst nicht auf die DB zu, sondern arbeiten nur über die Repositories.

Vorteile der Umsetzung:

- Die Geschäftslogik bleibt datenbankunabhängig und testbar
- Einheitlicher Zugriff auf Daten durch definierte Interfaces
- Die Umstellung auf eine andere Datenquelle wäre mit minimalem Aufwand möglich (z. B. Umstieg von PostgreSQL auf MongoDB)

8. Lessons learned

- Hibernate reduziert Aufwand, aber erfordert korrektes Mapping.

  Die automatische Speicherung und das Laden von Objekten funktionierten gut – aber nur mit richtig gesetzten Annotationen

- FXML erleichtert UI-Entwicklung, ist aber leicht fehleranfällig.

  Ein falscher Controller-Name oder Tippfehler im XML führt schnell zu Laufzeitfehlern. Vor allem bei den Namen haben wir teilweise ewig nach dem Fehler gesucht.

- Leaflet.js und OpenRouteService waren fehleranfällig und schwer zu integrieren.

  Es hab viele Bugs in der Ansicht – z. B. falsche Kartendarstellung, Route wird nicht geladen, Route wird nicht angezeigt, die Karte ist nicht ganz sichtbar usw.

- Screenshot-Erstellung mit html2canvas ist verzögert

  Wir mussten lernen, dass der Screenshot verzögert generiert wird und haben anfangs den Dialog zu früh geschlossen, bevor der Screenshot fertig war. Dadurch entstanden unvollständige oder leere Kartenbilder.

- Import/Export über JSON war eine robuste Lösung.

  Anfangs haben wir versucht auch das Importieren und Exportieren mit PDFs zu machen, haben das aber verworfen, weil der Import zu aufwändig war. JSON war einfacher zu handhaben.

9. Tracked Time

- Frontend (UI, Karten, MVVM, Controller): 45 Stunden
- Backend (DB, Repo, Service, Route API, JSON, PDF): 35 Stunden
- Sonstiges (Logging, Helper Klassen, Config, usw.): 10 Stunden

10. GIT Repository

Die GIT History kann über unser öffentliches GIT Repository eingesehen werden.

GIT-URL: https://github.com/SilverPheonix/TourPlanner