Tour Management System – Final Technical Protocol

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

# Application Architecture

## 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

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:

*Hier UML Klassendiagramm einfügen*

# Use Cases

## Use-Case Diagramm:

*Hier Use Case Diagramm einfügen*

## Use Cases:

* Touren erstellen, bearbeiten, löschen
* Tour Route im Leaflet auswählen
* Logs erstellen, bearbeiten löschen
* Einzelnen Report generieren
* Summary Report generieren
* Import/Export von Tourdaten als (JSON)
* Tour liste sehen
* Details und logs pro Tour sehen
* Tour Log Details in einer Tabelle sehen
* Random Tour Vorschlag

# UX – User Experience

## Wireframes:

*Hier neue Wireframes und kurze Beschreibungen einfügen*

Alte WireframesA screenshot of a computer

AI-generated content may be incorrect.

# Verwendete Technologien

* JavaFX, Spring Boot, Hibernate, PostgreSQL
* Leaflet.js + OpenRouteService API für Kartenintegration

Hier weitere Technologien einfügen und erklären warum

# 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

# 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.

# 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)

# Lessons learned

Lessons learned einfügen

# Tracked Time

Tiem Tabel learned einfügen

# GIT Repository

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

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