TourPlanner Project Protocol

Made by: Tedi Dobi, Luca Bonavetti

Link to GitHub: https://github.com/TD717/tourplanner.git

The TourPlanner application is an application built using JavaFX for Frontend and Spring Boot for Backend. It allows a user to add/delete/edit tours, generate and view tour logs, import/export CSV data and generate PDF reports. Additionally, it provides a statistics dashboard, which shows analytics and insights for tours.

Highlights

- Unique Feature: Statistics Dashboard
- Architecture: MVVM pattern with 3-tier architecture
- Technology: JavaFX + Spring Boot + PostgreSQL Database
- Development Time: 68 hours total

Technical Steps

- Architecture Pattern: MVVM (Model-View-ViewModel)
- Database: PostgreSQL Database
    - Easy setup and fast development
    - Portable application deployment

- Framework Stack: Spring Boot + JavaFX
    - Spring Boot provides excellent dependency injection and data access
    - JavaFX offers desktop UI features
    - Good integration capabilities

- External API Integration: OpenRouteService for route calculation and geocoding.

Application Architecture

3-Tier Architecture Overview**:** The application follows a clear 3-tier architecture with separation of concerns:

- Presentation:
    - Includes JavaFX Views, ViewModels and ViewFactory
    - Provides UI logic data binding and user interaction
- Business:
    - Includes services, DTOs and external API
    - Provides business logic, data transformation and external integration
- Data
    - Includes repositories, entities and database
    - Provides data persistence, data access and storage

Layer Details

- Presentation Layer
    - Views: JavaFX FXML files with controllers
    - ViewModel: Business logic, data binding and state management

    - ViewFactory: Dependency injection and view creation
- Business Layer
    - Services: Business logic, data transformation and external API integration
    - DTOs: Data transfer objects for layer communication
    - External APIs: Route calculation, geocoding and PDF generation

- Data Layer
    - Repositories: Data access using Spring Data JPA
    - Entities: JPA entities representing database tables
    - Database: Database with JPA/Hibernate ORM

Design Patterns Implemented

- MVVM Pattern
    - View: FXML files with controllers
    - ViewModel: Business logic and data binding
    - Model: JPA entities and DTOs

- Repository Pattern
    - Spring Data JPA repositories

- Factory Pattern
    - Implementation of a ViewFactory
    - Dependency injection

- DTOs
    - TourDTO and TourLogDTO
    - Decoupling from entities

Technology Stack

- Backend Framework: Spring Boot
- Frontend Framework: JavaFX
- Database: PostgreSQL
- ORM: JPA/Hibernate
- PDF Generation: iText
- JSON Processing: Jackson
- External API: OpenRouteService

Unique Feature: Statistics Dashboard

The Statistics Dashboard is the unique feature that we developed for our project. It provides analytics and insights for tour planning and performance analysis.

- Comprehensive data analytics
- Total tours and logs: Overview of all tour data
- Most popular tour: Automatic identification based on log count
- Real-Time dashboard
    - Automatic updates: Statistics update when data changes
    - Professional layout: Clean dashboard design

    - Quick insights: Immediate access to metrics

Time Tracking

- Planning & Design
    - 13 hours
    - Requirements analysis, architecture design and technology selection
- Backend Development
    - 23 hours
    - Entity design, repository implementation and service layer development
- Frontend Development
    - 26 hours
    - JavaFX UI development, FXML design, ViewModel implementation
- External API Integration
    - 10 hours
    - OpenRouteService integration, geocoding, route calculation |
- Testing
    - 10 hours
    - Unit tests, integration tests, test configuration
- Documentation
    - 5 hours
    - Code documentation, protocol writing, UML diagram |
- Bug fixing
    - 8 hours
    - Performance optimization, UI improvements |
- Total
    - 95 hours
    - Complete application development

Lessons from this project

- External API integration: Took longer than expected due to lack of experience
- UI Development: JavaFX had a steep learning curve, but was manageable
- Spring Boot: Spring Boot was initially hard to implement, but there are a lot of resources available