TourPlanner Application Protocol

App Architecture

Layer Architecture

The TourPlanner follows a layered MVVM (Model-View-ViewModel) architecture with clear separation of concerns:

Presentation Layer:

- JavaFX FXML views with controllers
- ViewModels managing presentation logic and data binding

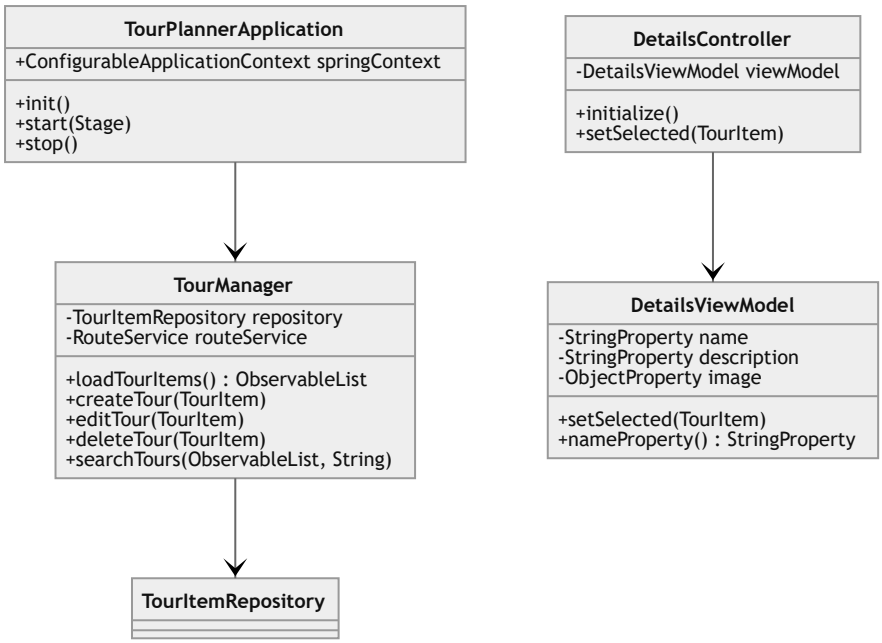
Business Logic Layer:

- Core services implementing business operations
- External integration services

Data Access Layer:

- JPA repositories for data persistence
- DTOs for data transfer

Class Diagram Structure

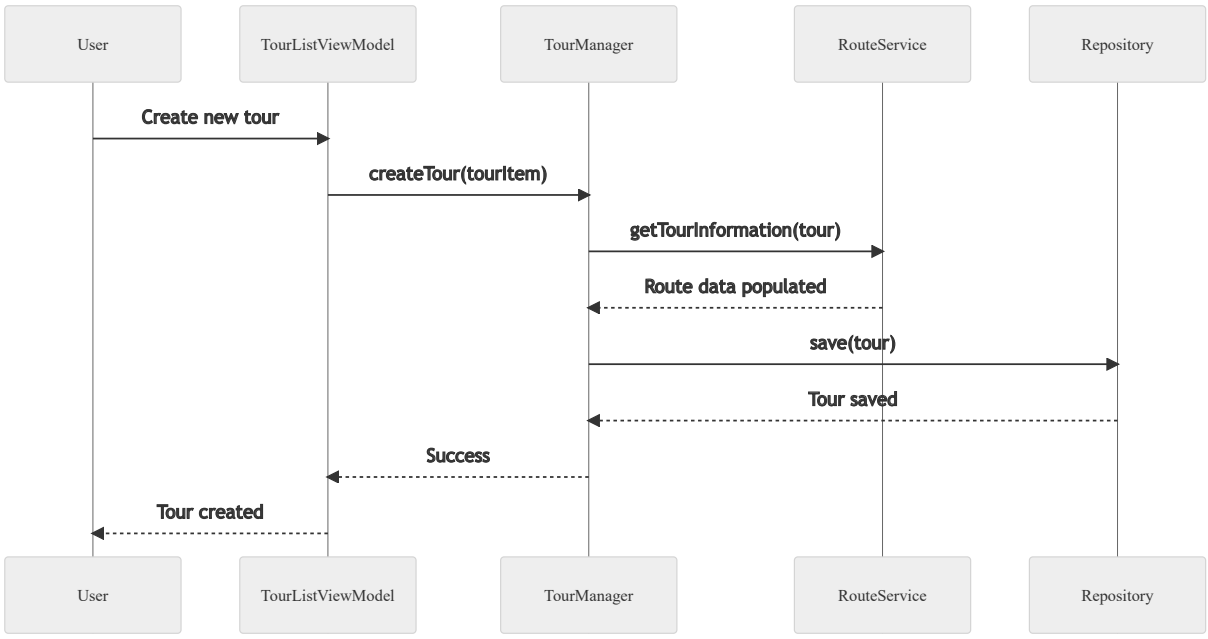


Use Cases

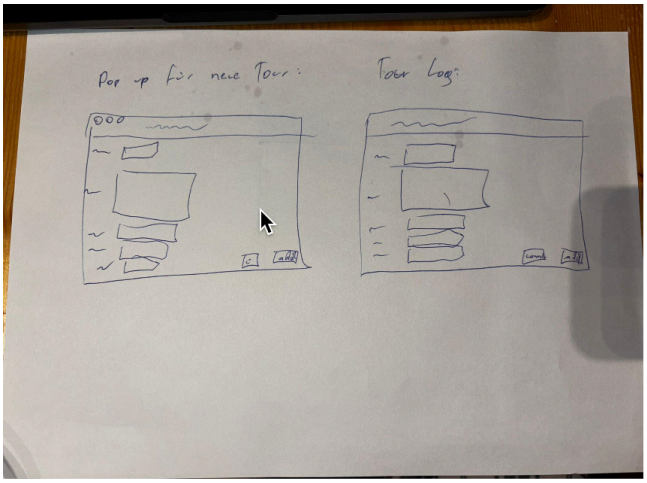
Primary Use Cases

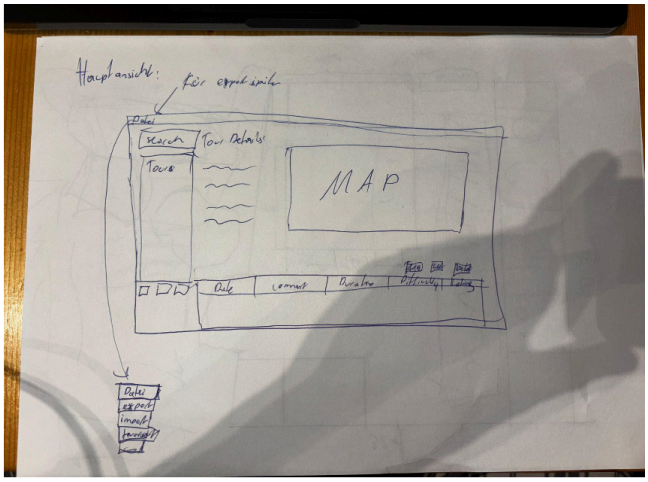
1. **Tour Management**: Create, edit, delete, and search tours
2. **Tour Log Management**: Add, modify, and view tour logs
3. **Route Calculation**: Calculate distances and durations using external APIs
4. **Report Generation**: Generate PDF reports for tours and summaries
5. **Data Import/Export**: Export/import tour data in JSON format

Sequence Diagram - Tour Creation



UX Design





Main Interface Wireframe

The application uses a multi-pane layout with:

- **Left Panel**: Tour list with search functionality
- **Center Panel**: Tour details display with map integration
- **Bottom Panel**: Tour logs table
- **Top Menu**: File operations and report generation

Data Binding Implementation

The UI uses JavaFX property binding for real-time updates

Library Decisions

Core Libraries

- **Spring Boot**: Dependency injection and application configuration
- **JavaFX**: Desktop UI framework with FXML
- **JPA/Hibernate**: Data persistence layer
- **Jackson**: JSON serialization for import/export
- **iTextPDF**: PDF report generation
- **Log4j2**: Logging framework

Lessons Learned

- Spring integration with JavaFX requires careful controller factory setup
- External API integration needs robust error handling

Design Patterns

MVVM Pattern

The application implements MVVM with clear separation:

- **Models**: Domain entities (TourItem, TourLog)
- **Views**: FXML files with minimal logic
- **ViewModels**: Presentation logic and data binding

Service Layer Pattern

Business logic is encapsulated in service classes

Repository Pattern

Data access is abstracted through JPA repositories

Observer Pattern

JavaFX properties enable automatic UI updates when data changes

Unit Testing

The application includes UI testing. Mostly important were the API implementations and other Services, such as the unique feature

Test Coverage Areas

- ViewModel data binding functionality
- Service layer business logic
- Integration with external services

Unique Features

Calories

Calories are calculated based on the distance and the transportation type.

Tracked Time

This implementation took 30hrs per person. In total 60hrs were spend on the project

Git Repository

Repository Link: https://github.com/tomiella/TourPlannner