TourPlanner – Project Report

1. App Architecture

Layered Structure

TourPlanner follows a clean **3-layer architecture** with clear separation of concerns:

1. Presentation Layer (TourPlannerFXApp)

- **Controllers**: e.g. TourDetailsController, TourOverviewController
- **ViewModels**: e.g. TourDetailsViewModel
- **Models**: Tour, Log
- **Responsibilities**: UI control, data binding, user interaction

2. Business Layer (TourPlannerFXApp)

- **API Client**: TourApiService
- **Responsibilities**: HTTP communication, business logic, data processing

3. Data Layer (TourPlannerDAL)

- **REST Controllers**: TourController, LogController
- **Services**: e.g. ReportService, OpenRouteService
- **Models**: Tour, Log
- **Responsibilities**: Database access, REST API handling, data validation

Class Diagram

```
classDiagram

  class Tour {

    +int id

    +String name

    +String tourDescription

    +String fromLocation

    +String toLocation

    +String transportType

    +double tourDistance

    +double estimatedTime
```

```
    +String quickNotes
    +List<Log> logs
}

class Log {
    +int id
    +int tourId
    +Date date
    +Time time
    +String comment
    +int difficulty
    +double totalDistance
    +Time totalTime
    +int rating
}

class TourApiService {
    +getAllTours()
    +addTour(Tour)
    +updateTour(Tour)
    +deleteTour(Tour)
    +getLogsForTour(int)
    +addLog(int, Log)
}
```

```
class TourDetailsViewModel {
  +setTourModel(Tour)
  +createNewLog()
  +updateSelectedLog()
  +deleteSelectedLog()
}

class TourDetailsController {
  +setTour(Tour)
  +onAddLogButtonPressed()
  +onUpdateLogButtonPressed()
  +onDeleteLogButtonPressed()
}

Tour ||--o Log : "has many"
TourDetailsController --> TourDetailsViewModel
TourDetailsViewModel --> TourApiService
TourApiService --> Tour
TourApiService --> Log
```

2. Use Cases

Tour Management

- **UC1**: Create tour
- **UC2**: Edit tour
- **UC3**: Delete tour
- **UC4**: Search tours

Log Management

- **UC5**: Add log entry
- **UC6**: Edit log entry
- **UC7**: Delete log entry

Routing & Maps

- **UC8**: Calculate route
- **UC9**: Display map

Import/Export

- **UC10**: Export tours
- **UC11**: Import tours

Use Case Diagram

```
graph TB

  User((User))



  subgraph "Tour Management"

    UC1[Create Tour]

    UC2[Edit Tour]

    UC3[Delete Tour]

    UC4[Search Tours]

  end



  subgraph "Log Management"

    UC5[Create Log]

    UC6[Edit Log]

    UC7[Delete Log]

  end
```

```
subgraph "Routing"
   UC8[Calculate Route]
   UC9[Display Map]
end

subgraph "Import/Export"
   UC10[Export Tours]
   UC11[Import Tours]
end

User --> UC1
User --> UC2
User --> UC3
User --> UC4
User --> UC5
User --> UC6
User --> UC7
User --> UC8
User --> UC9
User --> UC10
User --> UC11
```

Sequence Diagram – Create Log Entry

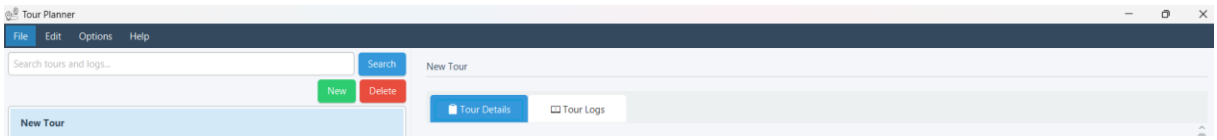
```
sequenceDiagram
    participant U as User
    participant C as TourDetailsController
    participant VM as TourDetailsViewModel
    participant API as TourApiService
    participant DB as Database

    U->>C: Click "Add Log" Button
    C->>VM: createNewLog(date, time, comment, ...)
    VM->>API: addLog(tourId, log)
    API->>DB: POST /api/tours/{id}/logs
    DB-->>API: Log created
    API-->>VM: Log object
    VM-->>C: Log added to ObservableList
    C-->>U: Update UI (ListView)
```

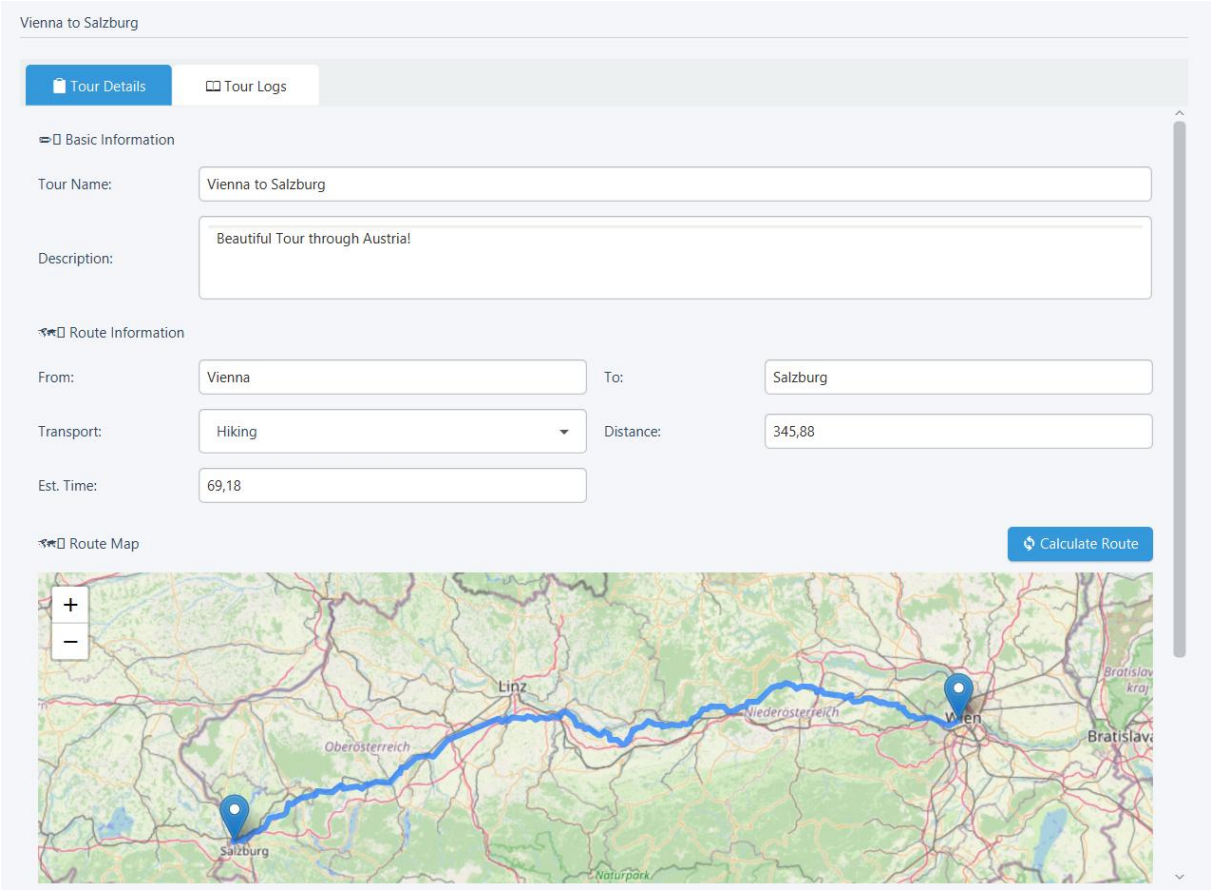
3. User Experience (UX)

Wireframes

- Main window layout



- Tour details view with tab navigation



UX Design Decisions

- **Split-pane layout**: Simultaneous view of tour list and details
- **Tab-based navigation**: Logical separation of details, logs, and maps
- **Inline editing**: Direct editing of log entries in the ListView
- **Responsive design**: Automatic adaptation to various window sizes
- **Color scheme**: Modern blue palette with high contrast ratio

4. Library Decisions

Frontend (JavaFX)

- **JavaFX**: Native Java GUI framework
- **Jackson**: JSON serialization for API communication
- **Apache HttpClient**: HTTP requests to backend API
- **Lombok**: Reduces boilerplate code

Backend (Spring Boot)

- **Spring Boot**: Rapid application development
- **Spring Data JPA**: Database access layer
- **PostgreSQL**: Robust relational database
- **PDFBox**: PDF generation for reports
- **Jackson**: JSON processing

5. Design Patterns

1. **Singleton Pattern**
    - Used for TourApiService to ensure a single instance for API communication
2. **Model-View-ViewModel (MVVM)**
    - **View**: FXML and controllers
    - **ViewModel**: TourDetailsViewModel
    - **Model**: Tour, Log
3. **Observer Pattern**
    - JavaFX properties for automatic UI updates
4. **Factory Pattern**
    - Used in TourApiService for building HTTP requests

6. Unit Testing

Testing Strategy

- **Unit Tests**: Test individual methods and classes
- **Integration Tests**: Validate full API interactions
- **UI Tests**: JavaFX interface testing

Test Classes

Backend

- TourTest: Model validation
- TourControllerTest: REST API tests

Frontend

- TourApiServiceTest: API client testing

- TourDetailsViewModelTest: ViewModel logic tests
- TourPlannerAppTest: UI functionality tests

7. Unique Feature

Quick Notes Feature

1. Inline editing of notes
2. Auto-save functionality
3. Rich text formatting support

8. Time Investment

Phase	Hours	Description
Project Planning	8	Architecture design, database schema
Backend Development	24	REST API, services, database integration
Frontend Development	32	JavaFX UI, controllers, view models
API Integration	16	HTTP client setup, data binding
Testing	12	Unit and integration tests
UI/UX Design	8	Wireframes, styling, responsive layout
Documentation	6	Code comments, README file
Bugfixing/Refactoring	14	Error handling, performance optimization
Total	**120**	

9. Git Repository

- **Repository Link**: https://github.com/JoelK27/TourPlanner