1.Introduction

This document presents a comprehensive overview of the design and architectural framework of the project. It is structured to include both frontend and backend components, each separately designed to operate independently. This enhances the overall system structure, facilitating better management, scalability, and maintenance. The frontend focuses on user interaction and visual presentation, while the backend is responsible for data processing and logic, ensuring a robust and efficient project infrastructure.

1.1 App Architecture

In our app architecture, we implement the Model-View-ViewModel (MVVM) pattern to structure our code effectively. This pattern is crucial in separating the roles of data handling, user interface presentation (View), and the intermediary operations linking the Model and View (ViewModel).

- View Layer: The View layer is primarily responsible for the user interface presentation. It manages the rendering of data to the user, captures user inputs, and handles various user events. This layer is directly managed by FXML for defining the user interface structure and JavaFX, which provides a robust framework for building and managing the application's GUI components.

- ViewModel Layer: Serving as the intermediary between the View and the backend, the ViewModel facilitates data bindings and event handling that influence the View. It abstracts the backend logic for the frontend, ensuring that the UI logic is decoupled from the backend processes. Key components like MainViewModel, OverviewViewModel, and SearchBarViewModel play a crucial role in managing the state and behaviors of the UI components.

2.Backend Architecture

The backend is designed using Spring Boot, an influential framework that streamlines the development, testing, and deployment of robust Java-based web applications. Here is an in-depth overview of the backend components, libraries, and configurations employed in the project.

2.1 Spring Boot Configuration

The backend utilizes Spring Boot to facilitate rapid development and efficient management of web services and data access, enhancing the application's robustness and scalability.

2.2 Plugins

Essential Gradle plugins are incorporated to enhance dependency management and integrate smoothly with Spring Boot.

2.3 Libraries and Frameworks

Spring Boot Starters

- Web & Data Access: Equipped with starters for web and JPA, the backend efficiently handles web requests and manages ORM-based data interactions, supporting RESTful services and database operations.
- Data Binding & Serialization: Implements data binding and serialization for efficient data exchange between the frontend and backend.

2.3 PDF and Template Rendering

- Dynamic HTML Views: Integrated with a template engine for server-side HTML rendering, facilitating dynamic content generation.

- PDF Generation: Supports generating PDF documents, useful for report and printable output generation.

2.4 Database Configuration

- PostgreSQL Configured as the runtime database, it offers robust, scalable, and efficient data storage solutions.

- Lombok which simplifies code with annotations that automatically generate boilerplate code such as getters, setters, and other common methods, enhancing readability and reducing potential errors.

2.5 Build and Dependency Management

The backend uses Gradle for building and managing dependencies.

2.6 Testing Framework

The backend's testing strategy is supported by JUnit 5, which provides a comprehensive suite for unit and integration tests to ensure the backend functions as expected.

3.Frontend Architecture

The frontend is built using JavaFX, which structures the application using the MVVM design pattern. This architecture supports a clean separation of concerns, making the application easier to manage and scale.

3.1 User Interface

- Interactive User Interface, the UI is designed to be user-friendly, supporting interactive tasks such as creating, updating, and deleting tours. It provides intuitive navigation and clear, accessible forms for data entry, ensuring a smooth user experience.

- Interactive maps that allow users to visually plan and modify tours. This feature enhances the application by providing a dynamic component where users can engage more deeply with the tool.

3.2 Technology

- We integrated mapping functionalities using Leaflet, a leading open-source JavaScript library for mobile-friendly interactive maps.

3.3 UX

1.Toolbar

- Users can add new tours by clicking the plus icon. This opens a form where they can enter details for a new tour.
- Users can remove existing tours from the system using the minus icon.
- Users can edit the details of an existing tour.

2. Search Functionality

- At the top right of the window, there's a search bar that allows users to quickly find specific tours based on keywords like tour names or destinations.

3. Tour Details

- Tourname: for entering the name of the tour.
- Beschreibung: for a detailed description of the tour.
- Startpunkt and Endpunkt: to specify the starting and ending points of the tour
- Transporttyp: Dropdown for choosing the mode of transport for the tour, such as biking, walking, driving and so on.
- Distanz and Dauer: Fields to enter the calculated distance and expected duration of the tour.

4.Logs

- section for logs related to the tour where each log entry includes:
    - Date: The date of the log entry
    - Duration and Distance: These fields record specific metrics from a completed tour, such as the actual time taken, and the distance covered.
- Users can add, remove, or edit log entries, which helps in tracking the performance or experiences of the tours over time.

4.Use-Cases

Creating Tour:

- User opens the app and clicks on the plus icon.
- User types in a Tourname, description, start location, end location and chooses the transport type for the tour.
- User then submits the details, and the app saves the information.

Viewing Tour Details:

- When the user launches the app, they choose an already established tour from the provided list.
- The user clicks on the desired tour to access its specific details.
- The application then presents the tour's name, destination, detailed description, and the method of transportation used.

Adding Tour Logs:

- User selects a tour from the list to view more details.
- The app displays the selected tour's details and a list of any existing logs.
- User clicks the "Add Tour Log" button to start a new log entry.
- User enters information about the tour including date, duration and distance.
- User submits the completed log form, and the application saves the new log.

Searching for Tours:

- User enters keywords in the search bar and initiates the search.
- The application filters and displays tours that match the search criteria in the tour list.
- User can select any tour from the results to view details or edit as needed.

5. Design Patterns

- MVVM: facilitates easier maintenance and testing by decoupling the UI from the Model.

- Dependency Injection: used to increase the modularity and flexibility of the application. It allows for better management of dependencies between classes, making the system easier to extend and modify. Dependency Injection enhances testability as dependencies can be easily swapped or mocked during testing.

- Factory pattern: simplifies the creation of objects, especially when the creation process is complex or when multiple types of objects need to be created following a common interface. By using a factory, the application's classes are not bound directly to the construction of their dependencies.

- Observer pattern: utilized to establish a subscription mechanism to notify multiple objects about any events that happen to the object they are observing. This is useful in our case where changes in the model need to be reflected immediately in the UI or in other dependent modules without tight coupling between them.

6.Unit Testing Decisions

- Frontend Tests ensuring the ViewModel functionalities correctly manipulate and respond to changes in the Model, affecting the View accurately. Tests simulate user interactions and check the consistency of ViewModel outputs.

- Backend Tests that aim to verify the business logic, data handling, and integration points like API connectivity and database operations. Ensuring stability and reliability through tests that mock external dependencies.

6.1 Frameworks and Tools

- Frontend: Utilizes JUnit for structuring tests along with Mockito for mocking dependencies.

- Backend: Employs Spring Boot Test framework to manage application context and dependencies, facilitating integration and unit testing with ease. Also uses Mockito for mocking and JUnit for assertions.

6.2 Specific testing

- ViewModel Testing: Mocking dependencies such as services and observers to test business logic and data handling isolated from the UI.

- Service Testing: Mocking data repositories and external APIs to test services without actual data operations or network calls.

- API Testing: Ensuring that API endpoints handle requests correctly and respond appropriately using mocked services.

7. Lessons learned

- Integrating code frequently and managing version control effectively are essential to avoid last-minute complexities and conflicts.

- Use version control systems like Git effectively. Regular commits and merges minimize integration problems.

- Consistent testing throughout the development cycle prevents bugs and reduce the time spent on debugging.

- Implement unit tests early and often. Use continuous integration tools to run tests automatically, ensuring that every integration meets quality standards.

- Writing clean, maintainable code is just as important as meeting the functional requirements.

- Stay flexible in your project planning. Allow room for adjustments in your project timeline and scope to accommodate new insights or changes in requirements.

8. Unique feature

Our unique feature makes it possible for the user to sort the tours in their list. This functionality enhances the management and enables users to customize and prioritize their tours.

9. Time spent

46 hours in total approximately

- Backend 6h
- Frontend 40h

10. GitHub Link

https://github.com/Pijuten/Routplanner.git