Protocol for JavaFX Application

1. Application Architecture

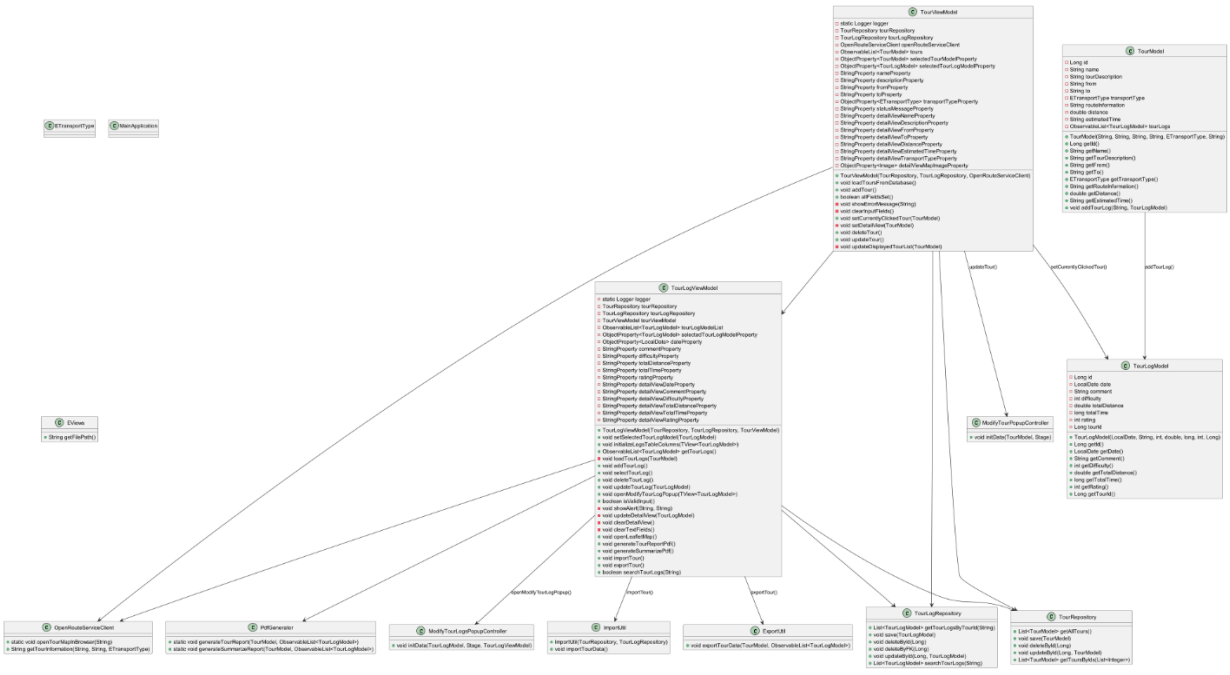
Layers and Their Contents/Functionality:

1. Business Logic Layer (BL):
   - Components: Controllers, iText (for File generation), ViewModels.
   - Functionality: Contains the core business logic and controls the flow of the application. Manages interactions between the UI and the Data Access Layer.

2. Data Access Layer (DAL):
   - Components: Repositories, External API Integration.
   - Functionality: Handles data retrieval, storage, and communication with external

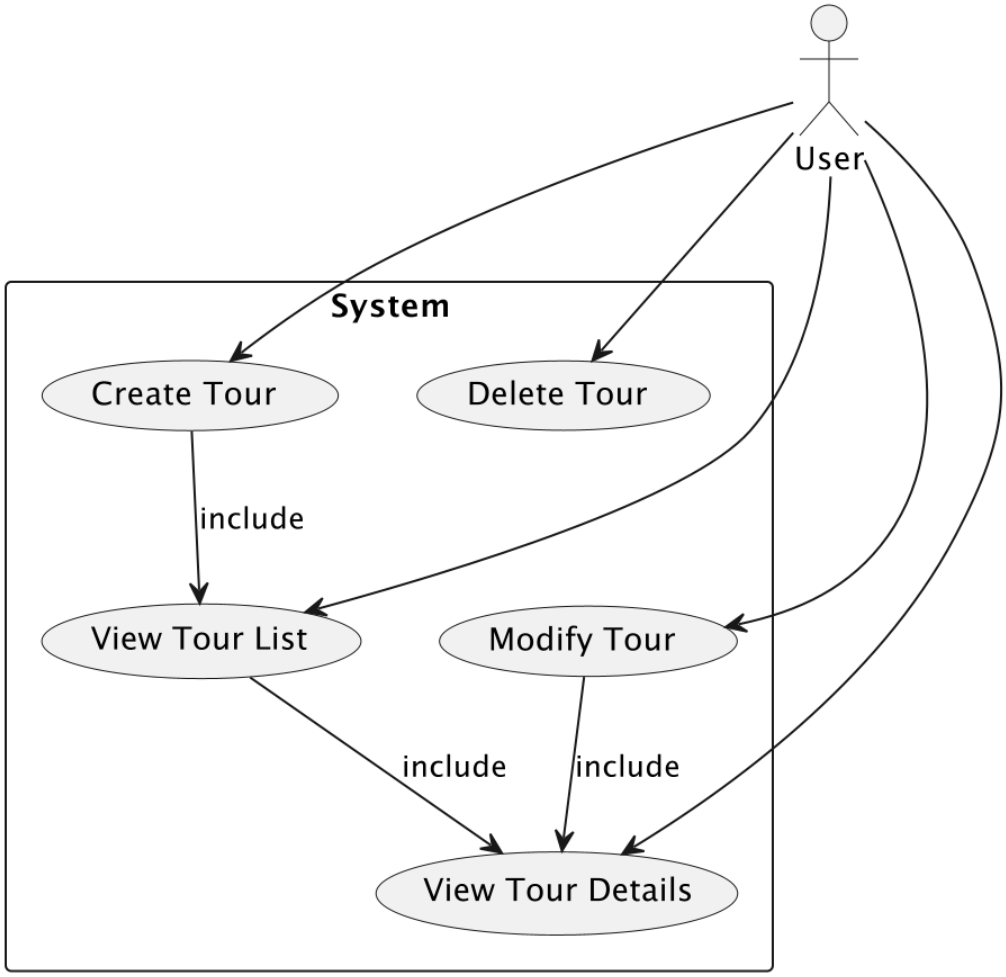
Class Diagram:

A simple Diagram is shown here to understand the relationship between the classes:
(Detailview: https://i.postimg.cc/pVScsT0p/classdiagram.png)



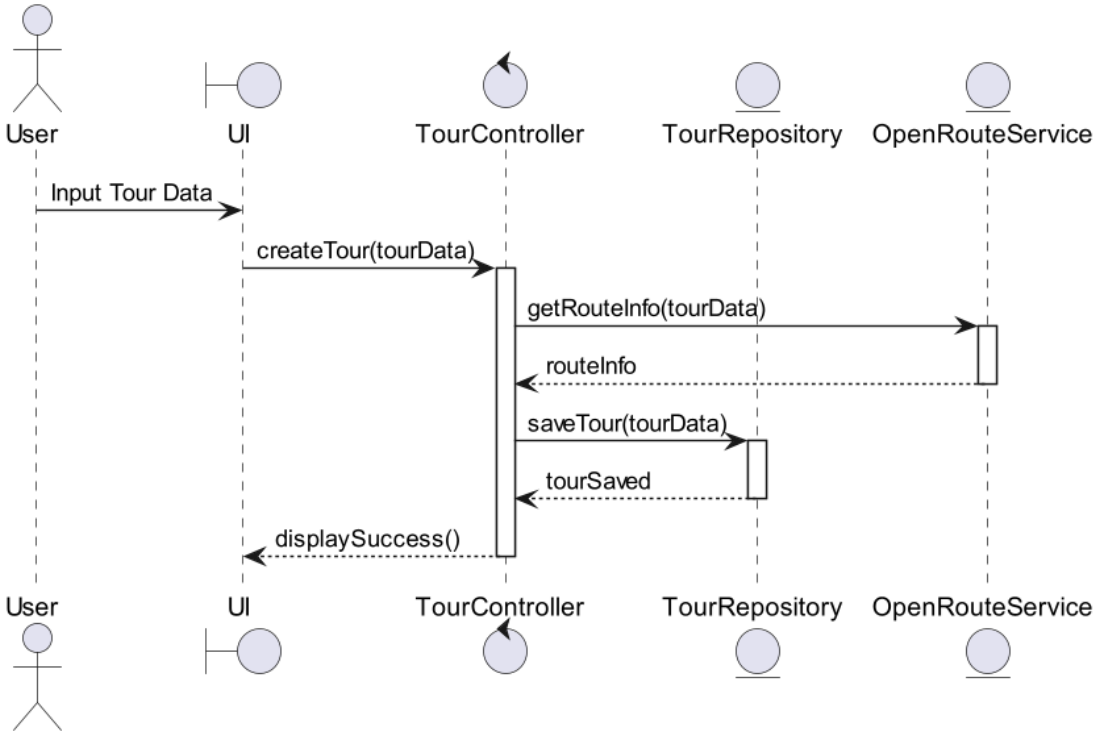
2. Use Cases

Use Case of User (Create Tour):



Use Case Diagram

- **Actors**:
  - **User**: Represents anyone interacting with the system.
- **Use Cases**:
  - **Create Tour**: User inputs details to create a new tour.
  - **View Tour List**: User views a list of all tours.
  - **Modify Tour**: User modifies the details of an existing tour.
  - **Delete Tour**: User deletes an existing tour.
  - **View Tour Details**: User views detailed information about a specific tour.



3. User Experience (UX)

Description:

- Main View:
Displays a list of tours and options to view details or create new tours. It also contains a searchbar with a fulltextsearch function with a refresh button and a darkmode button.

- Create Tour:
Provides a form for users to input new tour details and save them. Also contains a modify and delete button.

- Detail View(Tour):
Provides a detailed view with more information about the currently selected tour.

- Create TourLog:
Provides a form for user to input Logs about a Tour including difficulty popularity ect.

4. Library Decisions

JavaFX:
- Reason: Chosen for its rich UI components and seamless integration with Java.
- Lessons Learne: Mastery of JavaFX layout management and CSS customization was essential for creating a responsive UI.

iText:
- Reason: Used for generating PDF reports of tour details.
- Lessons Learned: Understanding the PDF generation process and handling layout in PDF documents was crucial for producing accurate and readable reports.

OpenRouteService:
- Reason: Integrated for route calculations and map services.
- Lessons Learned: Gained insights into handling external API interactions and managing asynchronous data retrieval in Java.

5. Implemented Design Pattern

Model-View-ViewModel (MVVM):
- Description: Separates the application into the Model (data), View (UI), and ViewModel (intermediary logic that handles data binding and state management).
- Usage: Helps maintain separation of concerns and enhances testability by decoupling the UI from the business logic.

https://refactoring.guru/design-patterns/factory-method
Our repository classes have their constructor set to package visibility and need a RepositoryFactory to be created. This makes the repository classes future proof in case we someday need more dependencies for the repositories. The Factory takes the repository and creates all repository classes with the same dependencies which ensures efficient memory usage.

6. Unit Testing Decisions

Testing Framework:
- JUnit: Utilized for writing and running unit tests.
- Mockito: Employed for mocking dependencies in controller and repository tests.

Approach:
- Controller Testing: Ensure user actions trigger appropriate business logic and state changes.

7. Unique Feature

Dark Theme Mode:
- Description: Provides a dark theme for a visually appealing interface that reduces eye strain and maintains readability.
- Implementation: Custom CSS stylesheets with darker hues, ensuring pleasant aesthetics and usability.

8. Tracked Time

In total it took us about 2 weeks time of 6 hours per day per person. The most time-consuming task was refactoring which took around 40% of the time.

Git Link:

https://github.com/philippchn/SS24_SWEN2_TourPlaner