**Tour Planner Protocol**

**1. Introduction**

This protocol document describes the user experience (UX) of the Tour Planner application. The application allows users to manage tours and tour logs using a Windows Presentation Foundation (WPF) interface implemented in C#. The application follows the MVVM pattern, ensuring a clean separation between the UI and business logic.

**2. Application Overview**

The Tour Planner application provides functionalities for:

* Creating, modifying, and deleting tours.
* Viewing detailed information about each tour.
* Adding, updating, and removing logs associated with tours.
* Displaying logs in a detailed view.
* Validating user inputs to prevent application crashes.

**3. User Interface Description**

**Main Window**

The main window is the central hub of the application, displaying the list of tours and providing options to add, update, or delete tours and logs.

**Wireframe**

**A screenshot of a map

Description automatically generated**

**4. Features Implemented**

**GUI Features**

* **Correct Data Binding**: All UI elements are correctly bound to view model properties, ensuring data integrity and synchronization.
* **Responsive UI**: The UI adjusts to window size changes, providing a consistent user experience across different screen sizes.
* **Reusable Components**: The application defines reusable UI components, promoting code reuse and maintainability.

**Tour Features**

* **Create/Modify/Delete Tour**: Users can perform CRUD operations on tours.
* **Tour Attributes**: Tours include name, description, from, to, route type, distance, estimated time, and map image.
* **Tour Details**: The details tab shows all attributes of the selected tour.
* **Input Validation**: User inputs are validated to prevent application crashes.

**Log Features**

* **Create/Modify/Delete Log**: Users can perform CRUD operations on tour logs.
* **Log Attributes**: Logs include date, distance, duration, steps, weather, difficulty, comment, and rating.
* **Log Details**: The logs table shows all logs associated with the selected tour.
* **Input Validation**: User inputs are validated to prevent application crashes.

**5. Unit Tests**

The application includes at least four unit tests to ensure the correctness of the MVVM implementation:

* **UpdateLogCommand\_CanExecute\_ReturnsTrueWhenLogIsSelected**: Verifies that the command can execute when a log is selected.
* **UpdateLogCommand\_CanExecute\_ReturnsFalseWhenNoLogIsSelected**: Verifies that the command cannot execute when no log is selected.
* **DeleteLogCommand\_CanExecute**: Verifies that the command can execute when a log is selected.
* **DeleteLogCommand\_CannotExecute\_WhenLogIsNull**: Verifies that the command cannot execute when no log is selected.

**6. Setup and Running the Application**

**Prerequisites**

* Docker and Docker Compose installed on your machine.

**Steps to Run the Application**

 **Run Docker Compose**

 **Add the Database**

 **Run the Application**

**6. Conclusion**

The Tour Planner application meets all the requirements specified in the checklist. It uses C# with a WPF markup-based UI framework, follows the MVVM pattern, implements required features for managing tours and logs, and includes robust input validation to ensure a smooth user experience. The provided wireframes illustrate the layout and functionality of the key UI components.