Tour Planner

**Form a team of two students** to develop an application based on the GUI frameworks C# / WPF or Java / JavaFX. The user creates (bike-, hike-, running- or vacation-) tours in advance and manages the logs and statistical data of accomplished tours.

Requirements

Goals

* implement a **graphical-user-interface** based on WPF or JavaFX
* apply the **MVVM-pattern** in C# / **Presentation-Model** in Java
* implement a **layer-based architecture** with a UI Layer, a business layer (BL), and a data access layer (DAL)
* implement **design-patterns** in your project
* define your own reusable **UI-component**
* store the tour-data and tour-logs via O/R-mapper in a PostgreSQL **database**; images should be stored externally on the filesystem
* use a **logging** framework like log4net or log4j
* generate a **report** by using an appropriate library of your choice
* generate your own **unit-tests** with JUnit or NUnit
* keep your **configuration** (DB connection, base directory) in a separate config-file - not in the compiled source code
* **document** your application architecture and structure as well as the development process and key decisions using UML and wireframes

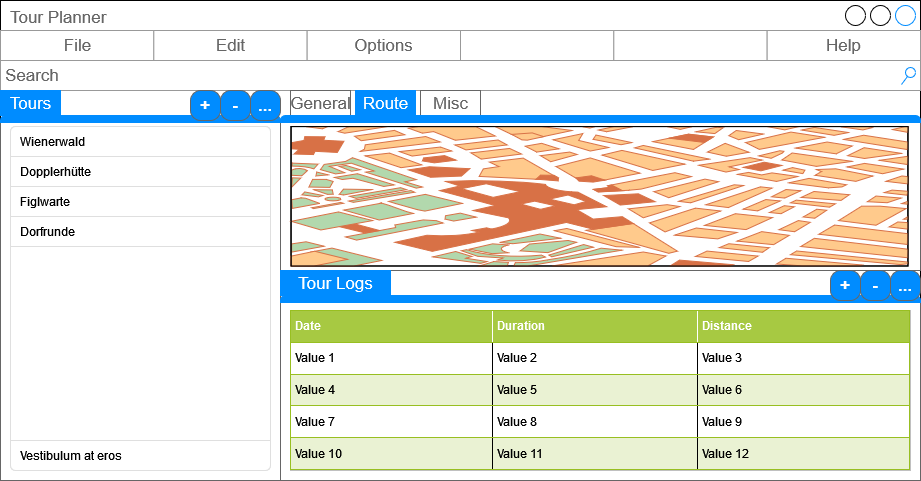
Features

* the user can create new **tours** (no user management, login, registration... everybody sees all tours)
* every tour consists of **name, tour description, from, to, transport type, tour distance, estimated time, route information** (an image with the tour map)
  + the image, the distance, and the time should be retrieved by a REST request using the OpenRouteservice.org APIs and OpenStreetMap Tile Server (<https://openrouteservice.org/dev> , <https://tile.openstreetmap.org/> )
* **tours** are managed in a list, and can be **created, modified, deleted** (CRUD)
* for every tour the user can create new **tour logs** of the accomplished tour statistics
  + multiple tour logs are assigned to one tour
  + a tour-log consists of **date/time, comment, difficulty, total distance, total time, and rating** taken on the tour
* **tour logs** are managed in a list, and can be **created, modified, deleted** (CRUD)
* **validated** user-input
* **full-text search** in tour- and tour-log data
* automatically **computed tour attributes**
  + popularity (derived from number of logs)
  + child-friendliness (derived from recorded difficulty values, total times and distance)
  + full-text-search also considers the computed values
* **import and export** of tour data (file format of your choice)
* the user can generate two types of reports
  + a **tour-report** which contains all information of a single tour and all its associated tour logs
  + a **summarize-report** for statistical analysis, which for each tour provides the average time, -distance and -rating over all associated tour-logs
* add a **unique feature**

Optional Bonus Features (for bonus points)

* create a **REST-server** that is responsible for data management and persistence
  + you can use any helper class like .NET's [HttpListener](https://docs.microsoft.com/en-us/dotnet/api/system.net.httplistener) or Java's [HttpServer](https://docs.oracle.com/en/java/javase/15/docs/api/jdk.httpserver/com/sun/net/httpserver/class-use/HttpServer.html).
  + consider that different UIs can work on tour data, so that data needs to be in sync between different UIs
  + consider that different UIs should not be able to overwrite their work

User-Interface Structure



Hand-In

Create a desktop application in C# (WPF) or Java (JavaFX) which fulfills the requirements stated in this document. Add unit tests (20+) to verify your application code. Upload your final code snapshot.

Add a protocol as pdf with the following content:

* protocol about the technical steps and decisions you made (designs, failures and selected solutions)
* document your application features using an UML use case diagram
* document your UI-flow using wireframes
* document the application architecture using UML:
  + class diagram
  + sequence diagram for full-text search
* explain why these unit tests are chosen and why the tested code is critical
* track the time spent with the project
* consider that the git-history is part of the documentation (no need to copy it into the protocol)

For the final presentation prepare the following:

* present the working solution with all aspects
* execute the unit-tests and explain the results
* present the key items of your protocol (see above)

Mandatory Technologies

* C# / Java as desktop application
* GUI-framework WPF (for C#) or JavaFX (for Java) or another Markup-Language-based UI Framework
* OR-Mapper (like .Net/Entity-Framework for C# or Java/JPA+Hibernate via Spring Boot for Java)
* HTTP for communication
* JSON serialization & deserialization
* Database Engine PostgreSQL used by the OR-Mapper
* Logging with log4j (Java) or log4net (C#) or another .NET Microsoft.Extensions-Solution.
* A report-generation library of your choice
* NUnit / JUnit

Grading

For a detailed point distribution see the accompanying checklist.

Must Haves

In case you don't implement the following required minimum goals, the hand-in is graded with 0 points:

* use a UI technology based on markup language (XAML, FXML)
* implement MVVM for the UI
* implement a layer-based architecture (for Java: implement Business and Data Access Layer using Spring Boot)
* implement at least one design pattern (and mention it in the protocol)
* use an O/R-mapper to store at least some data in the PostgreSQL database (for Java: via Spring Boot with JPA/Hibernate)
* store your application configuration in a config file
* integrate the OpenRouteservices.org and OpenStreetMap
* integrate log4j/log4net for logging
* integrate a PDF generation library
* implement at least 20 unit tests

Points Distribution (60 Points)

* 35: functional requirements
  + GUI in general
    - design and function
    - unique feature
  + tours
    - create/modify/delete a tour
    - view/manage tours in a list
    - input-validation
    - computed attributes
  + tour-logs
    - create/modify/delete tour-logs assigned to a tour
    - view/manage tour-logs as list
  + full-text search, also in computed attributes
  + generate reports
* 15: non-functional requirements
  + persistence (for Java: must be implemented using Spring Boot)
  + configuration
  + unit-tests
* 10: protocol
  + design and architecture
  + lessons learned
  + unit test design
  + time spent
  + link to git
* 5: bonus points (but not more than 60 points overall!)