

## Tour Planner

This desktop application is developed 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.

### Goals

- implement a **graphical-user-interface** based on WPF or JavaFX (supported) or another markup based UI framework
- define your own reusable **UI-component**
- apply the **MVVM-pattern** in C# / **Presentation-Model** in Java
- structure your application in separate **layers** e.g.: business-layer (BL), data-access-layer (DAL), view-model (VM), user-interface (UI)
- store the tour-data and tour-logs in a postgreSQL **database**; images should be stored externally on the file-system
- implement **design-patterns** in your project
- use a **logging** framework like log4net or log4j (or Serilog or using Microsoft.Extensions-Solution)
- generate a **report** by using an appropriate library of your choice
- generate your own **unit-tests** with JUnit or NUnit
- use **documentation-annotations** in the source-code; to be used in a document-generator like Doxygen or Sandcastle.
- **configuration** (db-connection, base-directory) in separate config-file - not in the sourcecode

### MUST HAVES

In case you don't implement the following minimal required goals, the hand-in is automatically evaluated as 0 points (grade: 5, F, de: "Nicht Genügend").

- use a UI technology based on Markup Language (Avalonia, UNO, ...)
- use the mentioned technologies / frameworks (see also mandatory technologies below)
- implement the defined GUI-pattern
- use at least one design pattern (and mention it in the protocol)
- store at least some data in the database
- store all your config in a config file
- implement at least 20 unit tests

## Features

- the user can create new **tours** (no user management, login, registration... everybody sees all tours)
- every tour consists of **name**, **tour description**, **route information** (an image with the tour map) and **tour distance**
  - the image should be retrieved by a REST request using (<https://developer.mapquest.com/documentation/directions-api/> )
- tours are managed in a list, can be **created, modified, deleted, copied** (CRUD)
- **import and export** of tour data (file format of your choice).
- for every tour the user can create new **logs** of the accomplished tour statistics
- multiple **logs** are assigned to one **tour**
- a tour-log consists of **date/time**, **report**, **distance**, **total time**, and **rating** taken on the tour
- add **five more properties** for the tour-log of your choice (f.e. average speed, joule for bicycle-tours)
- **full-text search** in tour- and tour-log data
- the user can print a **tour-report** of one tour with all its logs
- a second **summarize-report** for statistical analyses should also be generated, summarize total-time and -distance over all tour-logs
- add a **unique feature**

## Optional Bonus Features (bonus points)

- 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
- create a **REST-server** that is responsible for data management and persistence
  - you can use any helper class like .NET's [HTTPListener](#) or Java's [HttpServer](#).

## User-Interface

The screenshot shows a desktop application window titled "Tour-Planner". It features a menu bar with "File", "Edit", "Options", and "Help". Below the menu is a search bar with a magnifying glass icon. The main area is divided into two panes. The left pane, titled "Tours", contains a list of tours: "- Tour 1", "- Tour 2", "- Tour 3", and "...". The right pane has a "Title:" label and a large text area for "Route" and "Description". Below this is a "Logs:" section with a table. The table has columns for "Date", "Duration", "Distance", and "...". The first row shows "01.01.2021", "04:05:06", "120km", and "...". The second row shows "02.01.2021", "...", and empty cells for "Distance" and "...". There are also empty rows for "Date", "Duration", and "Distance".

Date	Duration	Distance	...
01.01.2021	04:05:06	120km	...
02.01.2021	...		
...			

## Hand-Ins

Create an desktop-application in C# or Java based on the mentioned GUI-frontends which fulfill the requirements statet in this document. You are not allowed to use object-relational-mappers (ORM), instead the data-layer has to be implemented on your own. Add unit tests (20+) to cerify your application code.

Add five or more properties (mandatory) as extensions for the tour-log and use them at least in the statistical report.

Add a protocol as pdf with the following content:

- protocol about the technical steps you made (designs, failures and selected solutions)
- 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)

The final presentation is done in a 10min presentation at the end of the semester (date will be provided soon)

- 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
- SQL (no OR-mapper)
- HTTP for communication to MapQuest
- JSON.NET / Jackson for JSON serialization & deserialization
- Database Engine postgresSQL with the ADO.Net- (for C#) or JDBC- (for Java) API
- Logginglog4j (Java) or log4net (C#) or another .NET Microsoft.Extensions-Solution.
- a report-generation library of your choice
- NUnit / JUnit

### Grading (50 points)

- 35: functional requirements
  - GUI in general
    - (graphical and software) design
    - function
    - unique feature
  - tours
    - create/modify/delete a tour
    - view/manage tours in a list
    - input-validation
  - tour-logs
    - add/modify tour-logs assigned to a tour
    - view/manage tour-logs as list
  - full-text search
  - generate reports
- 10: non-functional requirements
  - persistence
  - configuration
  - unit-tests
  - source-documentation generated
- 05: protocol
  - design
  - lessons learned
  - unit test design
  - time spent
  - link to git
- 05: bonus points