Tour Planner

Form a team of two students to develop an application based on the backend frameworks C# / ASP.Net or Java / Spring Boot with an Angular frontend. The user creates (bike-, hike-, running- or vacation-) tours in advance and manages the logs and statistical data of accomplished tours.

Requirements

Goals

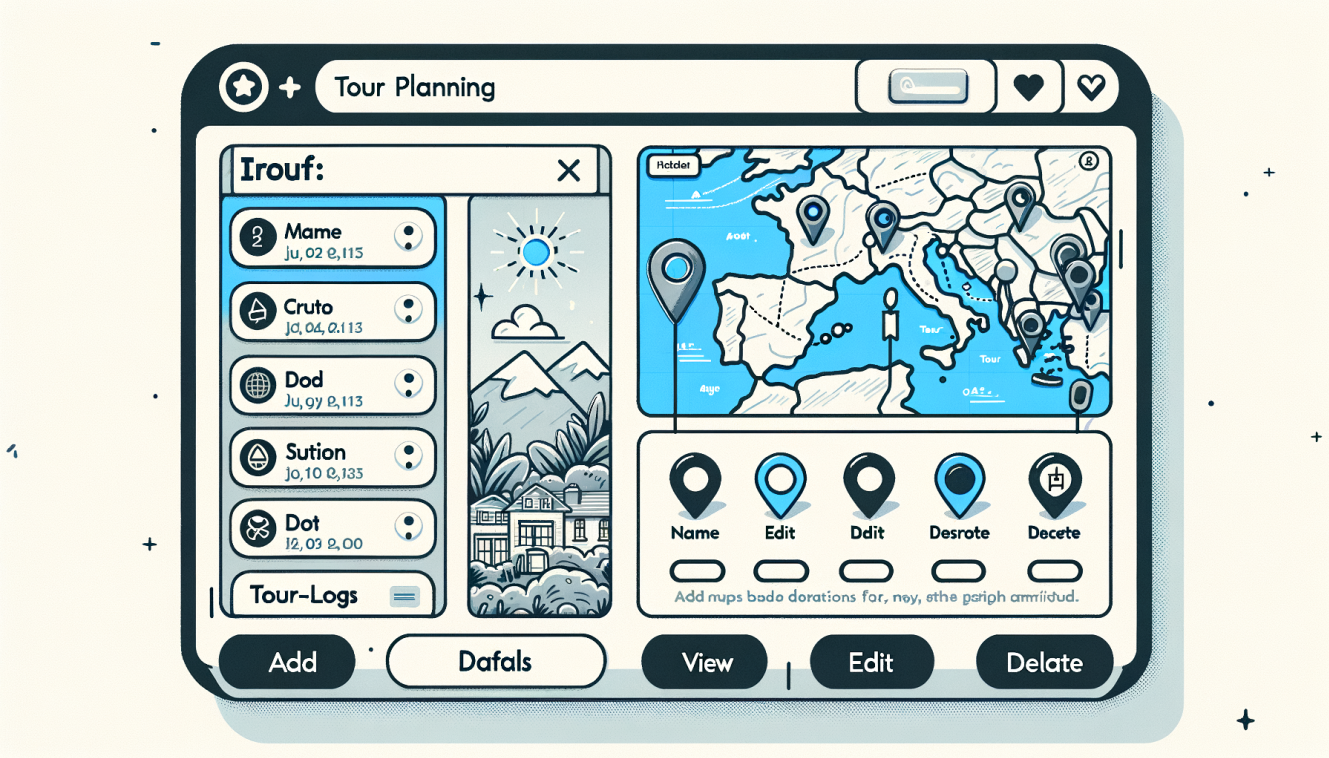
- implement a **web-based user-interface** based on Angular
- apply the **MVVM-pattern** when using Angular
- implement a **client-server** system using a middleware-based backend ASP.Net (C#) or Spring Boot (Java)
- apply a **layer-based architecture** with a presentation layer, a business layer (BL), and a data access layer (DAL)
- implement **design-patterns** in your project
- define your own reusable **web-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 your own **unit-tests** with JUnit or NUnit
- keep your **configuration** (DB connection, base directory) separate - not in the 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 self-**register** and then **login** using credentials
- the user can **create new tours**
- every tour consists of **name, tour description, from, to, transport type, tour distance, estimated time, route information** (the graphical display of the tour map)
  - the **distance**, and the **time** should be retrieved by a REST request using the **OpenRouteservice.org** API and the **graphical map** using leaflet
- **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)**
- **tours** and **tour logs** belong to a single user who has created them. No sharing of anything to other users.
- **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)
- add a unique feature

User-Interface

- **Define your own UI-Design** which is capable of providing all required functionallity.
- Create a **wiremock** of your design and add it to the documentation (mandatory).



(just an AI generated funny picture ;-))

Hand-In

Create a two-tier application in C# (ASP.Net) or Java (Spring Boot) with Angular in the frontend 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 two-tier application
- backend framework ASP.Net (for C#) or Spring Boot (for Java)
- frontend framework Angular
- 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.
- 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 web-framework for the frontend (Angular)
- use a middleware for the backend (ASP.Net or Spring Boot)
- implement a layer-based architecture
- 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
- store your application configuration separate (not in the commited source)
- integrate the OpenRouteservices.org and Leaflet
- integrate log4j/log4net for logging
- implement at least 20 unit tests

Submissions

The submission is done in two steps:

1. **Intermediate-Submission** (class 13): covers the web-frontend and backend integration
2. **Final Submission** (class 24): added business-logic, DAL and all other features.

Hand in the latest version of your source code as a zip in Moodle (legal issue) with a README.txt (or md)-file pointing to your git-repository.

See the corresponding Checklist-Excel sheets for the MUST-HAVES, Grading-Items and Grading-Points. Late submissions are not accepted! Missing hand-ins or missing MUST-HAVEs will automatically grade the submission with 0 points!

Final Presentation

For the final presentation in class 24-25, be prepared with your * working solution already started on your machine * setup your environment so you can show the application directly. * open your design (see: protocol) to show your architecture / approach.

Bonus Features

With optional features implemented you can compensate possible errors in the implementation above. Nevertheless, it is not possible to exceed the maximum number of points (= 100%).