SEMINARARBEIT

im Studiengang Informatik/Computer Science

Lehrveranstaltung Software Engineering 2 Labor

TourPlanner

Ausgeführt von: Kai Lukasser, Paul Steindl

Personenkennzeichen: 2010257232, **xxx**

BegutachterIn: DI Wolfgang Berger

Wien, 10.07.202

Inhaltsverzeichnis

[1.1 Description 3](#_Toc108306640)

[1.2 Tracked time 3](#_Toc108306641)

[1.3 GitHub Link 3](#_Toc108306642)

[2.1 Design 3](#_Toc108306643)

[2.2 Failures 3](#_Toc108306644)

[2.3 Selected solutions 3](#_Toc108306645)

[3.1 Tourplanner 3](#_Toc108306646)

[3.2 Tourplanner.BusinessLayer 3](#_Toc108306647)

[3.3 Tourplanner.DataAccessLayer 3](#_Toc108306648)

[3.4 Tourplanner.Models 3](#_Toc108306649)

[3.5 Tourplanner.Shared 3](#_Toc108306650)

[3.6 Tourplanner.Shared.Log4Net 3](#_Toc108306651)

[3.7 Tourplanner.UnitTests 3](#_Toc108306652)

[4.1 Via UML use case diagram 4](#_Toc108306653)

[4.2 Via Wireframes 4](#_Toc108306654)

[4.3 Via UML Class Diagram 4](#_Toc108306655)

[4.4 Via sequence diagram for full-text search 4](#_Toc108306656)

# Overview

## Description

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.

## Tracked time

About 70 hours

## GitHub Link

https://github.com/PaulSteindl/Tourplanner

# Technical steps and desicions

## Design patterns

### MVVM

We implemented our project based on the MVVM-pattern. So, we have a decoupling of Model and View through the different ViewModels. Therefore, it makes the ViewModel reusable. It exposes properties and commands for binding in view.

### SOLID

We followed the SOLID principle. Therefore, we have:

* A separation of concerns
* No repetition of code
* We kept everything super simple
* No code what we don’t need or use
* Each unit has limited knowledge about other units

### Layered Architecture

Using logical layers in which components are grouped logically and communicate with each other and with other applications and clients, we have an overall structure for our applications. The structure is described in the following sites.

## Failures

We had one big failure in the frontend. Namely, we implemented the code not based on the MVVM-pattern. At first, we created all the different windows in the MainWindowViewModel. Furthermore, did we implement all buttons in there. In conclusion, we had to change the whole frontend code based on the MVVM-pattern in the last few days.

## Selected solutions

??

# App architecture

## Tourplanner

The Tourplanner project is our main project. It is responsible for everything and starts the application. Furthermore, it represents the frontend based on the MVVM-pattern for decoupling Model and view through ViewModel.

## Tourplanner.BusinessLayer

Our Tourplanner.BusinessLayer project implements the core functionality of the system and also encapsulates the relevant business logic. It consists of different interfaces that other callers can use.

### Calculate Attributes

For calculating some different things like the popularity of a tour or even the average rating of tour logs.

### Check Input

For checking the user input for instance by adding a tour.

### Import Export

For importing and exporting a tour.

### Map Quest API

For fetching all the data for the tours through the map quest API.

### Report

For creating a report or even a summarize report.

### Text Search

For searching through tours and tour logs.

### Tour

For creating, updating, deleting a tour.

### Tour Log

For creating, updating, deleting a tour log.

## Tourplanner.DataAccessLayer

Our Tourplanner.DataAccessLayer project provides access to data. For example, it reads files, creates dictionaries and also has some database commands.

## Tourplanner.DataAccessLayerInMemory

This project represents an InMemoryDataAccessLayer.

## Tourplanner.Models

Keeps the data of the business objects / entities.

## Tourplanner.Shared

We have several log levels between we can choose. It is used for different treatment of messages based on its level of severity.

## Tourplanner.Shared.Log4Net

??

## Tourplanner.UnitTests

We implemented over twenty unit tests, but unfortunately we wrote no tests for the ViewModels although it would be of importance.

# Documentation

## Via UML use case diagram

## Via Wireframes

At the beginning of our project, we created this Wireframing. It was just a simple one for getting an idea how it could look like. However, we overestimated ourselves very much with the graphical overview. In the end, it doesn't look anything like that.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated

Graphical user interface, application

Description automatically generated

## Via UML Class Diagram

## Via sequence diagram for full-text search

# Unit Tests

# Unique feature

Unfortunately, we don’t have a unique feature.

# Lessons learned

In the end, we learned a lot within this project.

* Working in pairs
* Handling merge conflicts
* Implementing a layered architecture
* MVVM-pattern
* How to handle different failures
* How to split up such a project
* And many more