Swen2 First Retake, TourPlannerProject, Melvin Douglas Böning, Bettina Kovac, BIF4, 2025

Protocol

Table of Contents

Introduction and git link

This protocol is for the TourPlanner project found under the following git repository:
https://github.com/BSCDulak/First-Retake-SWEN2-TourPlannerGroupProject

Design and Architecture

Frameworks/Libraries

WPF framework

Nunit framework

Microsoft.EntityFrameworkCore

(for migrations)

Npgsql.EntityFrameworkCore.PostgreSQL

(for postgres database things)

Swen2 First Retake, TourPlannerProject, Melvin Douglas Böning, Bettina Kovac, BIF4, 2025

microsoft.extention

(so we can use appsettings.json)

log4net

(so we can use logging without having to write our own logging code for e.g logging to file)

Layers

UI layer

Consists of the .xaml files where our frontend is situated. We have reusable elements in the UserControls folder like the subtab buttons, which makes it possible to change the frontend consistently on the element inself which can be injected multiple times. With this we avoid having to change e.g. multiple buttons in different parts of the UI. We also use a Visibility bool so we can have some buttons hidden where we don't need then without having to duplicate the buttons xaml. Our UI entry point is the MainWindow.xaml, which gets Tab1 and Tab2 injected which have their own injections from UserControls.

Backend/ViewModel

We use the MVVM pattern, which means all our ViewModels are connected throught the MainWindowViewModel. Our buttons are reusable via the AsyncRelayComand.cs and an inject to the appropriate ViewModel in the MainWindowViewModel.

Database layer

Our database consists of the models found in the Models folder and a variety of logic found in the Data folder. We connect to our database via an appsettings.json. Any changes to our model is applied to the database using the migration feature from Microsoft.EntityFrameworkCore.

Infrastructure layer

Our infrastructure layer consists of the logging folder, where the logic for the log4net library implementation is and our appsettings.json which holds our Database connection and our openroutes service api key.

Design Patterns

Factories

We implement a variety of Factories e.g. our LoggerFactory which is responsible for encapsulate object creation so clients can just `private static readonly ILoggerWrapper log = LoggerFactory.GetLogger();` and don´t need to implement the logger themselves, which means any changes can be concentrated in the factory and everything uses the same config path defined by the factory.

Swen2 First Retake, TourPlannerProject, Melvin Douglas Böning, Bettina Kovac, BIF4, 2025

Command Pattern

We have a command pattern implemented in AsyncRelayCommand.cs where we have a `public AsyncRelayCommand(Func<object?, Task> executeAsync, Predicate<object?>? canExecute = null)` to be used for all our commands (Add, Delete, Update, Report, Export, Import) that get triggered by the button presses on our UI. Unchecked this can cause issues e.g. if a user clicks a button so fast that it triggers a new Add while the old one is still running but there are ways around this like disabling the button for the duration.

Unit Tests

ToursListViewModelTests

This line of tests encompass all the functions of the ToursListViewModel. Add, Delete, Update, Load (from Database on Program start) and MultiDelete. All these functions are essential for the proper functioning of the program and thus need to be tested. We used an InMemoryDb to avoid issues with Db leeching into the ToursListViewModelTests.

TourLogsViewModelTests

Similarly to ToursListViewModelTests in TourLogsViewModelTests we check that the functions are working as intended.

SubtabbuttonTests

Here we test that the SubTabButtons class is properly implemented and can relay button presses. Without button presses our program doesn´t do anything.

ConnectToDatabaseTests

Our Database is vital for persistance so we test if it is actually online and reachable here.

MapViewModelTests

Here we test the logic of the leaflet implementation without actually calling the API to avoid failed tests when the API is unavailable.

ImportExportTests

Here we test the Import/Export functionality of the program, in particular file generation and the blocking of importing tours that are already existing (via Guid BackupId). Using TourId itself caused issues because we have to reset it on import or else it might conflict with a newer database incrementing TourId from 1 on Add Tour and crashing because imported Tour already has the Id needed for the new tour.

Swen2 First Retake, TourPlannerProject, Melvin Douglas Böning, Bettina Kovac, BIF4, 2025

LoggingTests

Here we test if we implemented our logging framework successfully. Considering how helpful the logging was I´d say this is pretty important too. Probably the first thing you should do.

Time Spent

We don´t have a presice time spent value but it is very visible from the git history as the sessions were very concentrated.

Lessons Learned Melvin Douglas Böning

Logging is vital! It helped me catch bugs I wouldn´t have been able to figure so expedient otherwise.

Many small commits are much easiert to review and revert than few big commits.

Writing a readme.md with the basic commands required to do certain things and installation needs per computer is incredibly useful if you switch computers or take a break for a bit. Especially not having to remember/recheck the migration commands was nice.

Helper classes are very nice to keep the viewmodels at a reasonable size.

Lessons Learned - Bettina Kovac

Während der Projektarbeit habe ich wertvolle Erkenntnisse über Teamarbeit und Projektorganisation

gewonnen. Ein zentraler Punkt war die Bedeutung gegenseitiger Rücksichtnahme – insbesondere im

Hinblick auf Zeitpläne und Verpflichtungen. In unserem Fall wurde die Datenbank relativ spät

fertiggestellt, wodurch ich längere Zeit nicht effizient an meinem Teil arbeiten konnte. Eine

funktionierende Mock-Datenbank stand nicht rechtzeitig zur Verfügung, was bei mir zu Unsicherheiten

hinsichtlich der Kompatibilität meines Codes führte. Obwohl ich frühzeitig kommuniziert hatte, dass ich

nur bis zu einem bestimmten Zeitpunkt verfügbar bin, kam es letztlich zu Verzögerungen.

Ein weiterer wichtiger Aspekt ist die klare und verbindliche Arbeitsverteilung. Es sollte nicht

vorkommen, dass Aufgaben nachträglich einseitig auf andere Teammitglieder übertragen werden –

besonders dann nicht, wenn diese bereits ausgelastet sind. Jede:r sollte Verantwortung für den

eigenen Aufgabenbereich übernehmen und diesen zuverlässig bearbeiten.

Ebenso ist die Anerkennung der geleisteten Arbeit essenziell für ein funktionierendes Team. Beiträge sollten weder relativiert noch in Prozentwerten abgewertet werden. Jeder Beitrag – unabhängig vom Umfang – trägt zum Gesamterfolg des Projekts bei, und das sollte auch im Umgang miteinander sichtbar werden.

Darüber hinaus habe ich erkannt, wie wichtig es ist, bei der Planung die unterschiedlichen zeitlichen Ressourcen der Teammitglieder zu berücksichtigen – insbesondere in der Endphase eines Projekts. Eine frühzeitige Koordination sowie Verständnis für die individuellen Zeitpläne sind deshalb unerlässlich.

Abschließend möchte ich noch betonen, dass ich zu Beginn klar gesagt habe, dass für mich eine Note im Bereich 4 völlig ausreichend ist. Das bedeutet nicht, dass ich meine Arbeit schlecht machen will – im Gegenteil. Aber es sollte auch nicht dazu führen, dass einzelne Teammitglieder deutlich mehr leisten müssen, nur weil sie persönlich höhere Ansprüche an die Note haben. Ein gemeinsames Projekt sollte auf Fairness basieren – nicht auf der Erwartung, dass alle dieselben Ziele verfolgen.

Unique Feature – Extension of Computed Attributes:

Im Rahmen unseres Projekts haben wir den bestehenden Punkt "Computed Attributes" aus der Projektcheckliste weiterentwickelt. Eine besondere Funktion, die wir implementiert haben, berechnet automatisch die Entfernung vom Start- zum Zielpunkt sowohl in Kilometern als auch in Minuten. Basierend auf der berechneten Dauer entscheidet das System intelligent, ob die Route zu Fuß zurückgelegt wird oder ob öffentliche Verkehrsmittel empfohlen werden.

Zusätzlich wird die Darstellung auf der Karte dynamisch angepasst: Die Farbe der Route verändert sich je nach gewähltem Fortbewegungsmittel. Diese visuelle Unterscheidung sorgt für eine verbesserte Nutzererfahrung und eine intuitive Übersicht über die vorgeschlagenen Routenoptionen.