TourPlanner Protocol

App Architecture

Our Solution contains

1. View and Viewmodels
2. BusinessLayer and Models
3. Data Access Layer

The View and Viewmodels layer contain all xml views and windows and the associated viewmodels. A view knows the viewmodel and the viewmodel knows the businesslayer.

The businessLayer has all services and knows the DAL.

The DAL contains the database and the repositories.

If for example a button is clicked in the view, a command is activated in the viewmodel. This command mostly calls a BusinessLayer function that calls a service or a DAL function, for example to create a Tour and save it in the database.

Use cases

Users can create a tour by entering two addresses, the app sends those to the mapquest API and gives the user a calculated route. Then users can add logs where they give information about the tour they did. Tours and logs can be edited and deleted. Users can also export and import a tour and generate a report of one or all tours.

UX, Libraries, Lessons learned

UX

Users see all tours on the main window, those can be selected and the route map appears in the same window. They can also see the associated logs when clicking on a tour. To create a new tour or log a new window opens that can be closed or a tour or log can be created. If a tour or log is selected they can be deleted or edited when the associated button is clicked.

Libraries

For the report function we use the iText7 libraries to generate a PDF with the tour data.

We also use the Microsoft.Extensions.DependencyInjection library for dependency injection.

Lessons learned

- Create a SelectedItemTracker object to avoid viewModel dependency
- Start with better knowledge of MVVM
- ViewModels shouldn't know each other
- Create more Interfaces for best practice

Design pattern

Dependency Injection

The IoCContainer handles dependency injection to achieve inversion of control which aims to create loosely coupled programs

Unit testing decision

The unit tests test if the delegatecommand works as intended and also tests the edge cases of the userInput functions. We also tried to mock a ViewModel class and test the OnPropertyChanged function.

Unique Feature

We thought about implementing a dark mode but it turned out to be more complicated then we thought and couldn't do it in time.

Tracked Time

See TrackedTime-PDFs

GIT Link

GIT repo Link: https://github.com/unravelet/swen2-TourPlanner.git