**Tour Planner**

**Protocol**

**App Architecture**

Diagram

Description automatically generated

* View holds the graphical elements as well as the Bindings to their respective Variables in View Model.
* View-Model holds the input-output variables (potentially redundant) that are necessary for the communication with the view. View-Model initiates CRUD operations on the repository and updates output variables if Repository updates theirs.
* Repository holds the relevant data that is the later used in the presentation layers. It performs the necessary CRUD operations on said data. Models themselves are responsible for conversion to appropriate display Format. Repository gets data either from View-Model or as a method result from DAL
* DAL contains sets of functions that are necessary for the data retrieval from data sources.

**Libraries**

* PDFSharp for Report Generation
* Newtonsoft.Json
* Log4net
* MSTest, NUnit – Lesson learned: not to use MSTest due to slow loading times. There are plenty of other libraries.
* Moq

**Design Patterns**

* Singleton – Class has to be instantiated exactly once and its address can be resolved statically.
  + Used on the View-Models.
    - In this project every view model was instantiated only once .
    - View-Models didn’t have to be mocked.
    - If duplicates of view models would be necessary – having many simultaneous views and their view-models, this would not scale well.
  + Used on repository.
    - In this project only one instance of repository was necessary.
    - Repository didn’t have to be mocked.
    - If Repository contained more context-dependent data, this wouldn’t scale well.
* Command Pattern – Class encapsulates all of the necessary data to perform operations.
  + Repository possesses all the necessary data to perform CRUD, filtering operations. Other classes are only used as data structures (Log, Tour) or Stateless Methods (Statistics, Conversions).
* Chain of responsibility – Commands get passed down through layers. If a layer can resolve the command it does, if not it passes the remaining command down the layer.
  + Results of a method in a particular class depend only on the implementation of that same class and the classes down the chain.
  + The only alteration coming from a lower class is the result of the method from that lower class.

**Unit Testing**

* I decided against dependency injection. Objects can be inserted during the test directly or via reflection (if private).
* I decided to use Interfaces for stateless classes, and their respective instances in the Repository. These classes are mocked, so they don’t modify data during testing.
* I decided against static methods, due to the difficulty during mocking.

**Time & Git**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Tasks** | **Commit** | **Time** |
| 24.10 | View, View Model | Init | 3h |
| 25.10 | Repository | 00:47 26.10 | 12h |
| 26.10 | View Model Commands, Database | 15:43 26.10 | 4h |
| 27.10 | Image Loading, Refactoring |  | 4h |
| 28.10 | Database, Mapquest, Image Loading, Pdf Generation | 16:42 28.10 | 6h |
| 29.10 | Finishing Crud, export Import | 03:03 30.10 | 14h |
| 30.10 | Tests, Refactoring, Protocol | 23:06 30.10 | 8h |
| 31.10 | Refactoring, Exceptions, Logging |  |  |
| 01.11 |  |  |  |
|  |  |  | 51h |

https://github.com/WiczusFH/Tour\_Planner.git