## Overall System Design

It was decided that to implement the project that Microsoft’s Windows Presentation Foundation (WPF) should be used. WPF provides a set of premade user interface components and a flexible platform for creating and styling applications. To accompany the framework Microsoft also invented a software design pattern called “Model View View Model” (MVVM) which provides a layer of abstraction between business and presentation logic while allowing a test-driven design. It was decided that we should try and implement a MVVM pattern in our application to best utilise the features of WPF.

To implement MVVM we decided that each view should have its own view model that encapsulated all the data required to render the view, with no interop between view models. This choice may result in slightly higher memory usage but lowers the chances of unintended side effects occurring because of actions performed in another part of the system.

While the database library currently creates a local copy of the database it was decided that all view models should frequently refresh, and save, any data they had acquired from the database to help reduce conflict chance if multiple users were using the system at the same time.

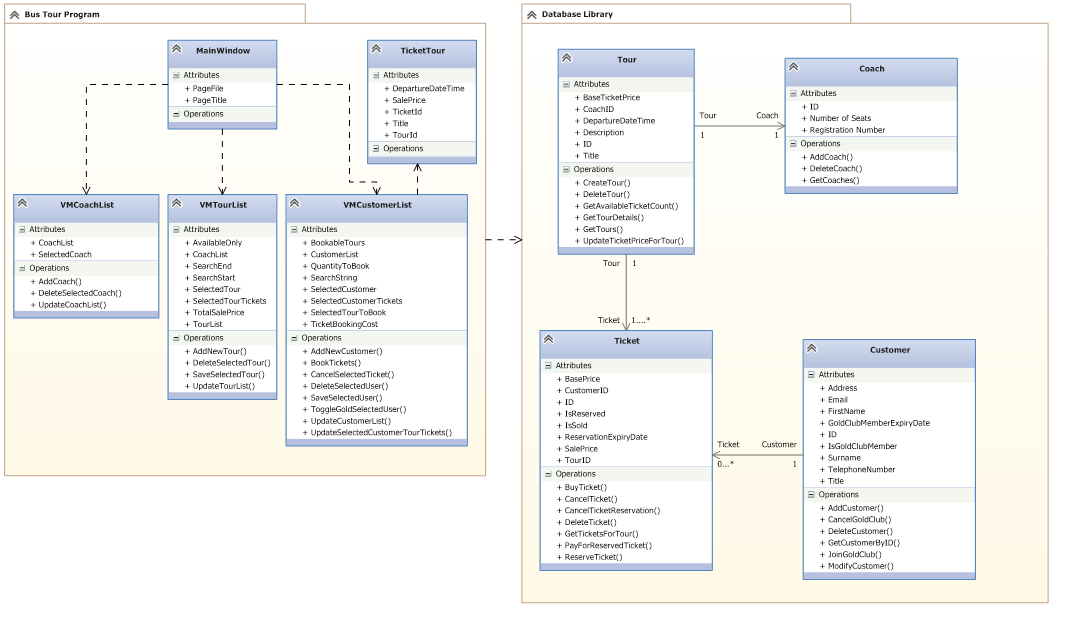
## Class Design

Below is the final abstract class design for the application.

The VM\* classes are viewmodels and contain data that WPF can bind to.

The inclusion of a TicketTour class was required to nicely display a list of tickets that a customer has by performing an INNER JOIN between tours and tickets.

Not included in this diagram is the “Database” singleton class, used for accessing the database without having to reinitialize the library.



## User Flows & Activity Diagrams

### Purchase Tickets



a

### Cancel Ticket



### Gold Upgrade



## Use Case Diagram



## Data Validation

WPF bindings provide a convenient way of performing data validation by automatically giving the user visual cues if their input does not match the type of the bound object. Unfortunately WPF can’t, by default, enforce any type of pattern matching, or ensure that an entire form is valid before allowing it’s submission so these features must be coded into the view models or the underlying code.

## User Interface Design Rationale

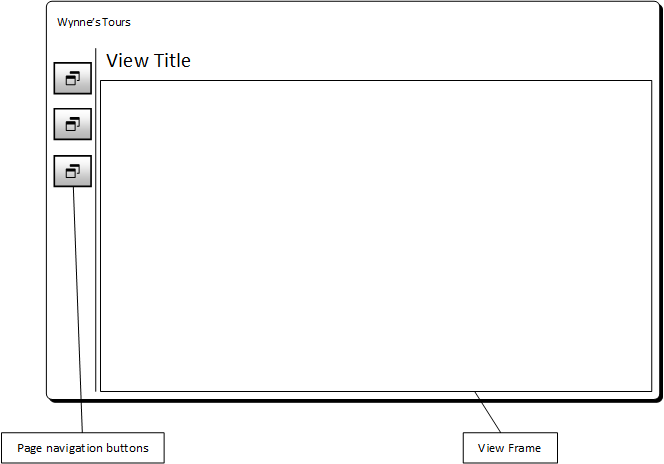
Below are the initial drafts for the various views in the application, along with any assumptions or decisions made during their conception.

Data Validation

### Main Window

The design surrounding all content in an application should be consistent and usable to ensure that the user is not flabbergasted by new controls appearing and disappearing at seemingly random points in the user experience. To this end it was decided that the main user design should be as minimal as possible and serve as a navigation shell, utilising sub-views to display actual content.

This resulted in a simple vertical tab drawer on the left hand side of the window, with a title and a frame occupying the remaining window space.



### Customers View

The customer list is the first part of the application will see, it should visually stand out while not appearing overly dense. To this end we decided to design a simple list of customer names that expands into a full blown edit view when a user is selected. The lack of dialogs prevents jarring transitions between states and helps to ensure a consistent user experience.

Editing customers is also made easy by having all relevant data for that customer in one place, including the tickets they’ve booked. Booking tours is considered to be a subset of

### Tours View

Again, for the sake of maintaining consistency we mimicked the design for the customer list for the tour list, with slight changes. The user is initially presented with a list of tours and their descriptions, when a tour is selected it is changed to edit mode, where all properties of the tour are grouped together and can be changed.

It was decided that in lieu of producing actual reports, the tour listings should also provide some statistical data to help in the management of the company.

### Coaches View

The simplest of all the pages – provides a simple form for creating new coaches (inline editing can’t be done because the Database Library lacks a modify coach method) and a list for viewing or deleting existing coaches.

## File organisation

* Base directory
  + pages/
    - Contains the fragments used in the MainWindow frame and their underlying code
  + viewmodels/
    - Contains the viewmodels used to bind data to the WPF forms.
  + icons/
    - Contains the icon assets

## Overall Test Strategy

The majority of testing will be performed in a “black-box” manner by the development team, ensuring that the program works as intended without actually adding any instrumentation to the code. While this method isn’t entirely fool proof, it does save a significant portion of time in the implementation phase of the project.

Upon delivery we will also undergo user acceptance testing with the client to ensure that it meets the specifications required by them, and find any areas that require improvement or reimplementation to meet the client’s needs.