**FNDev1 Coursework**

THLibrary: Library Search Facility

**Design**

**Table of Contents**

[Project Setup 3](#_Toc345170952)

[Solution 3](#_Toc345170953)

[Project Structure 3](#_Toc345170954)

[Installation 4](#_Toc345170955)

[TH Library – a Windows 8 Store App 5](#_Toc345170956)

[Overview 5](#_Toc345170957)

[Sample screen layout 5](#_Toc345170958)

[Design Considerations 7](#_Toc345170959)

[Patterns 7](#_Toc345170960)

[Data Storage 7](#_Toc345170961)

[Standard Contracts and Collections 7](#_Toc345170962)

[Design Instance of Data 7](#_Toc345170963)

[Development Issues 8](#_Toc345170964)

[IoC Container 8](#_Toc345170965)

[Asynchronous IO 8](#_Toc345170966)

[Unit Testing 8](#_Toc345170967)

[Evaluation 9](#_Toc345170968)

[Future Development 9](#_Toc345170969)

# Project Setup

## Solution

The project solution is a VS2012 Windows Store Application. It uses the Microsoft Unity dependency injection framework.

The data is held in a text file containing the CSV records of the books. These are loaded at the application start-up to the Business model and all access within the application; searches etc. are performed against this in memory Business Model.

Saved searches are also loaded at application start-up: these are held as an XML serialised data file containing the complete list of searches, in the users’ local storage folder. There will be no saved searches when the application first starts up, or until the user specifically saves a search.

## Project Structure

The project structure follows the [Onion Architecture discussed by JeffreyPalermo](http://jeffreypalermo.com/blog/the-onion-architecture-part-1/), specifically there are separate projects for each component layer.

Core: this is the application core, which includes the domain model

Infrastructure: contains the application cross cutting concerns

IoC: dependency injection layer, used to inject the infrastructure where necessary

THLibrary: this is the UI presentation layer and contains only UI logic.

### THLibrary - UI Layer

The UI layer is a **Windows Store App** that shows a single page only. It shows the Search criteria and the results in a list, along with the details of each book as it is selected from the results list.

The UI layer attempts to follow the MVVM pattern.

* The **View** is represented by the ***MainPage.xaml*** and the code behind code module.
* The **ViewModel** role is taken on by the ***LibraryDataSource*** class. It holds the data model required by the UI and performs most of the processing to support the UI. It contains ObservableCollection<T> for each collection that can make use of the automatic binding facilities offered by the Windows Store App xaml code.
* The **Model** is represented by the rest of the application, namely the Core project which is accessed through the repositories exposed by the core through interfaces..

The ***LibraryDataSource*** follows the structure of the ***SampleDataSource*** provided by the Windows Store App template, which provides the facility of design time data.

### IoC (Inversion of Control)

Microsoft’s **Unity** is used as the IoC container. At present, it is the only such framework the installs on the Windows 8 Store app platform. The specific version is 3.0.1208-Preview, which is a pre-release version and the only version that works with Windows Store App, the target framework is "winrt45".

### Core

This project contains the Data model for the Library books and the associated Authors, Titles, Keywords and ISBN references. It also contains the definitions of the searches, as search criteria. No search results are stored: they are obtained by executing the search again.

It exposes the interfaces required by the UI layer and the necessary factories for creating the instances of the data model classes.

### Infrastructure

This project contains the implementations of the repositories exposed by the Core project. It handles the loading of the data into the application, keeping the instances within the ***Library*** class and the ***SearchData*** classes.

### Documentation

The documentation is held in a separate project within the solution: Documentation. It contains the following folders;

**Analysis** contains analysis and development documents, which are only working papers.

**SandCastle** contains the compiled help file, generated from the XML comments within the code.

**XML** contains copies of the compiler generated xml files for the documentation.

**Design** contains the design document, Design.docx.

## Installation

The contents of the zip file or from the GitHub repository should be copied into the projects folder of the VS2012 installation, which must be running under Windows 8.

### Compilation Errors

The project must be built, preferably by **Rebuild THLibrary** instead of rebuild solution as this will generate errors for the Documentation project.

However, there will be errors generated from the **THLibrary** project which relate to the lack of some XML comments for documentation. This is because the template modules provided by the **VS** template do not all contain xml comments. There should be 37 warning errors). I have not endeavoured to complete documentation for these components.

### Running the Application

It is best to run the application from within the VS installation.

# TH Library – a Windows 8 Store App

## Overview

Application is a single page application, not conforming to exactly the style of program required for a Windows Store app. However, it is intended to provide search facilities and display results of the search immediately.

## Sample screen layout

### Layout

This sample screen layout shows the library contents, before any search has been performed. It shows a defined search in the drop down list on the left, with the definition of the search on the right.

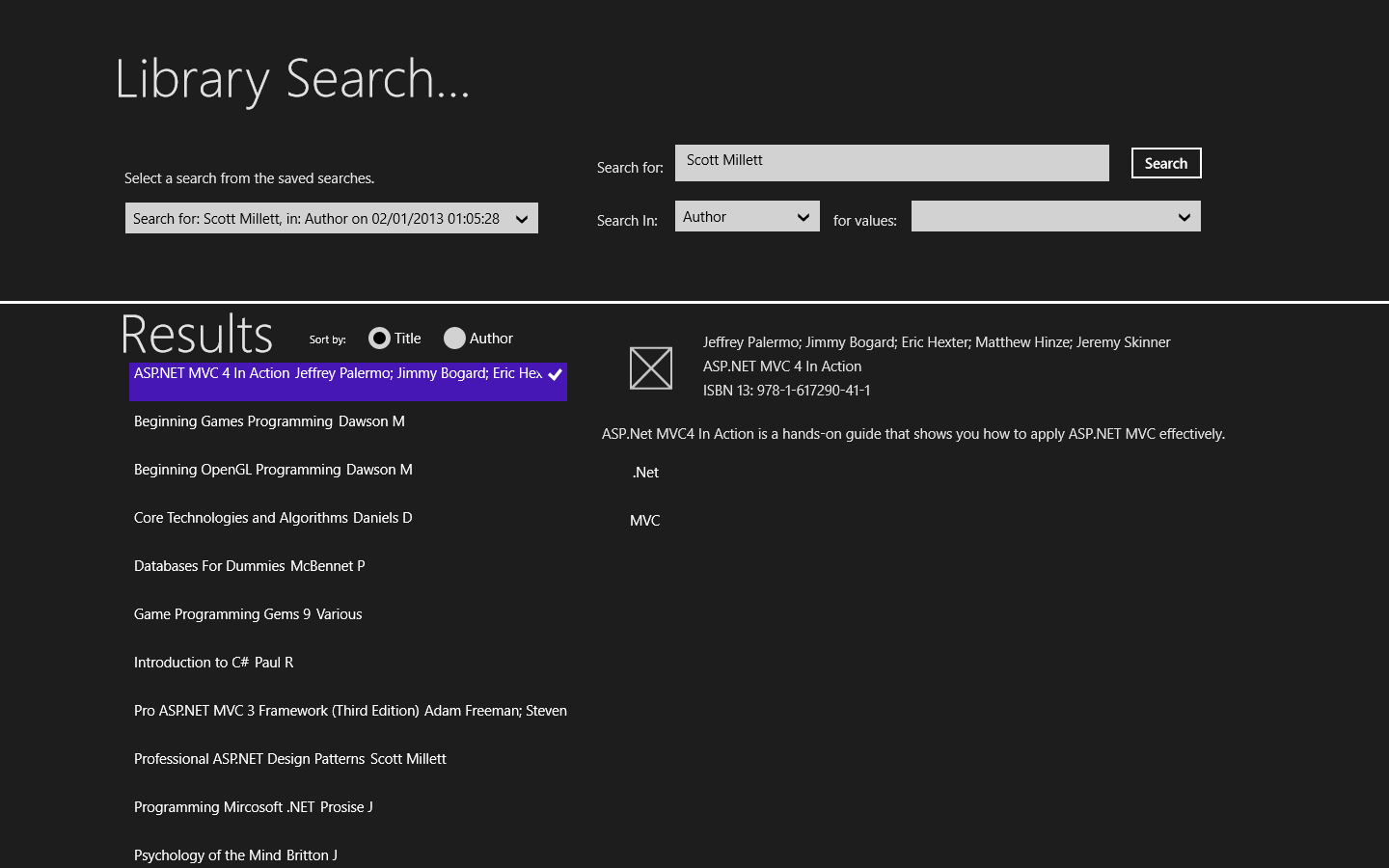


Figure 1 - Sample application page.

The page is divided into two sections, the top being where the search criteria are specified and the bottom section which shows the results of the search.

Within the top section, the left side shows a list of “saved” searches, ***Saved Searches List***, that can be selected to populate the right side, which shows the ***Search Criteria***. The search button will perform the search updating the results in the lower section.

The left hand side of the lower section displays a list of all books that match the search criteria, while the right side displays the details of whichever book in the list is selected. The two radio style buttons on the left side allow the list of books to be sorted in ascending Title or Author order sequence.

### Functionality

On application start, all books are loaded to the list and the details are shown for the first one. Any searches that have been saved are shown in the ***Saved Searches List***.

###### Running a Search

Running a search requires that the ***Search Criteria*** are completed which is done by either:

* Selecting a saved search from the list, which populates the ***search criteria***
* Selecting a Search Type from the dropdown list and adding values to the ***search string*** by selecting values from the ***Values*** dropdown list, or by typing the values directly into the search string

After the criteria have been specified, clicking the **Search** button wil perform the search, displaying the results in the bottom section of the page.

It is possible to enter multiple search values in the search string, but they must be separated by a ‘|’ character.

The following search Types are available.

**Search String** this will match books whose Title or Synopsis contains the values in the search string. The values must be typed into the search string, as they are not selectable from the Values dropdown list.

**Author** This will match books whose Author contains any of the values specified in the search string

**Title** this will match books whose Title contains any of the values specified in the search string

**Keywords** this will match books whose Keywords contain any of the values specified in the search string

**ISBN** this will match books whose ISBN contains any of the values specified in the search string.

###### Saving a Search (Via the AppBar)

Clicking the right mouse button on the page, as per the Windows Store App guidelines, will display the AppBar at the top of the page. This gives access to the **Save Search** button, as well as the **New Search** and **Reset** buttons. Clicking the **Save** button will save the current search criteria and update the Saved searches dropdown list.

###### Creating a New Search

Click the right mouse button on the page and then click the **New Search** button. This will clear the contents of the search string, and reset the SearchType to **SearchString**.

###### Reset the List to show all books.

Click the right mouse button on the page and then the **Reset** button. This will clear the search results and display all books in the library as happens when the application starts.

###### Change the sort of the results

Select either of the radio button at the top of the results list, to change the sequence of the results displayed.

# Design Considerations

## Patterns

To improve separation of concerns, the ‘onion model’ as previous mentioned, was adopted. This relies on the use of Dependency injection to inject the infrastructure components as required.

Within the UI layer the MVVM architectural pattern is employed.

The Abstract Factory pattern is used for creation of class instances, within the core project.

However, there are opportunities to employ other design patterns to further improve the separation of concerns: such as the Command pattern when determining which type of search to execute.

## Data Storage

Provided data is as a text file, specifically CSV format, therefore file access is using a TextReader. The XML serialiser is used for reading and writing the searches file.

## Standard Contracts and Collections

Most collections are of List<T>, so that sorting is possible, and indexing of elements is possible.

The collections of Authors, Keywords, Titles and ISBN references derive from the generic List<string> collection so they have this functionality but are referenced as collections by name.

Use is made of the IEquatable<T> interface to determine how classes are deemed equal. This is used by the UI dropdown lists when checking which item has been selected.

The INotifyPropertyChanged<T>, interface, used within the BindableBase class, combined with the ObservableCollection<T> are used within UI view Models to automate the two binding of UI components and the data model.

## Design Instance of Data

To allow design time instances of data to be available so that the UI looks realistic at design time is possible, but requires that the data model be defined with static methods and be created as a Singleton. Use of the **d** namespace and a namespace to ignore the **d** namespace at runtime allows this to work.

However, it ties the application into this type of data model which is awkward to work with when using the two way data binding between the UI components and the data model itself.

This was the reason for basing the ViewModel on this structure.

# Development Issues

## IoC Container

Using Unity as the IoC container is simple enough: configuration is similar to Ninject, but Ninject does not have a release compatible with Windows Store Apps just yet.

However, it is not possible to use constructer injection with xaml. The **LibraryDataSource** requires a reference to the repositories, but xaml MUST have a parameterless constructor.

Therefore, the repository is injected in the constructor by using the Unity.Container.Resolve<>() method. However, this does not work when referenced directly within the VM constructor. Instead we have to create a Readonly property on the App.xaml file and expose the repositories from these properties. This is then referenced directly from within the ViewModel and this works OK. It is, however, considered a bit of a hack but the code below works.

(App.Current as App).GetRepository()), where GetRepository is the exposed property.

## Asynchronous IO

Problems with the asynchronous calls to file and directory methods were experienced, similar to the lab exercises in that the application simply stops, without throwing any exceptions. This was despite all the calls being initiated from code within the App.xaml.cs file and application start up and suspend/shutdown.

This was unexpected, however, the problem only occurred at application suspend. Therefore, the solution is to keep a reference to the file Handle, obtained at application start and use the handle directly to write the file out.

## Unit Testing

There are no unit tests within this project. This is not deliberate, just that it hasn’t happened that way. I spent a significant part of the project trying to get the xaml based UI layer to work without enough time to implement unit tests for the ViewModel.

# Evaluation

Using the Windows Store App technology is state of the art development. However, this makes the available of 3rd party tools that are compatible with the environment quite restricted. This can be quite restrictive for development.

With the technologies being that new, there is a tendency for things not always to work, the asynchronous IO is an example. This makes it difficult sometimes to find an elegant solution. In fact this isn’t always possible, such as the way the IoC is implemented.

However, a benefit of this coursework is the introduction to the Windows Store App framework, and the possibilities it presents for future application development are quite interesting.

# Future Development

### Code Refactoring

Most of code refactoring is required within the UI layer, specifically:

* where the search is performed at the LibraryRepository.SearchBooksmethod, use should be made of the Command pattern so that the searchCriteria are treated as a command, the search is simply executed thus avoiding the use of the switch statement to determine which search to perform.
* Where instances of the view models are created, currently with the “new” keyword, this code should be refactored to make use of Factories. Although they are only for specific use within this UI layer, it makes for cleaner code.
* The Code behind for the Main Page, needs to be refactored to remove some logic that should be better placed in the LibraryDataSource, viewmodel. There are Todo comments indicating this.

### User Feedback

* When entering the search criteria, specifically the values in the search string, for specific types an auto suggestion should be coded.
* When the search is saved, a message should be displayed to show that the search criteria was successfully saved.