SYSTEMS PROGRAMMING AND SCRIPTING

FLORIAN BERGMANN
PERSON ID: H00020398

Assessment Two: Web Browser

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

AR	Additional-Requirements
FR	Favourite-Requirements
GR	GUI-Requirements
GUI	graphical user interface
HPR	Homepage-Requirements
HR	History-Requirements
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
MVP	model view presenter
PR	Printing-Requirements
URL	Uniform Resource Locator
WR	Web-Requirements
XML	Extensible Markup Language

Part I

DEVELOPMENT OF A BASIC WEB BROWSER

1 INTRODUCTION

This chapter will provide an overview over the document and recapture the requested requirements.

1.1 DOCUMENT OVERVIEW

The document is divided into seven chapters that will describe different aspects of the developed program:

Chapter 1 provides an overview over the document and the specified requirements, alongside certain assumptions that were made during the development.

Chapter 2 will give a short outline of the requirements that were fulfilled, as well as mention added functionality that was not requested.

Chapter 3 will provide a high level overview over the system's architecture and its sub-modules.

Chapter 4 is a short user guide that will describe the usage of the program by leading the reader through a selected choice of use cases to accomplish certain tasks.

Chapter 5 will be based upon Chapter 3 and provide an in-depth explanation implementation-details.

Chapter 6 will outline how testing was performed and which cases have been covered.

Chapter 7 will provide a reflection of the development process and the program and highlight areas of interest from the developer's point-of-view.

1.2 REMIT

The remit will summarize the requirements provided in the document *Systems Programming & Scripting (2010/2011) Assessment Two* and list all assumptions made in respect to a certain requirement.

For later reference throughout the document, the requirements will be divided into seven groups (Web-Requirements (WR), Homepage-Requirements (HPR), Favourite-Requirements (FR), History-Requirements (HR), Printing-Requirements (PR), GUI-Requirements (GR), Additional-Requirements (AR)) and a unique identifier will be assigned to each requirement.

WR01: Send Hypertext Transfer Protocol (HTTP) request messages for URLs typed in by the user.

WR02: Receive HTTP responses for the send requests.

WR03: Display received HyperText Markup Language (HTML)-code.

WR04: Display the HTML-code, when the received message is either a `200`, `400`, `403` or `404` status-code.

HR01: Allow setting and editing of a homepage-URL.

HR02: Load homepage on application start-up.

FR01: Allow adding and deleting of a URL to a list of favourites. Allow editing of favourites present on the list.

FR02: Allow the specification of a name for a favourite.

FR03: Request a favourite's URL when the favourite is activated.

FR04: Load all favourites on application start-up.

HR01: All pages that are requested shall be saved in a history.

HR02: Request a URL when an entry in the history is activated.

HR03: Load history on application start-up.

PR01: Allow printing of the currently displayed web page.

GR01: Provide a GUI for the actions specified in the requirements.

GR02: Use menus, buttons and short cuts to increase *accessibility*[1].

AR01: Utilize multi-threading to keep the application responsive.

AR02: Allow requesting of multiple web pages simultaneously.

1 Assumption: accessibility can be enhanced by using a standard layout common in Windows environments.

2 REQUIREMENT'S CHECKLIST

The following list shall provide an overview over the fulfilled requirements. The numbers correspond to those used in Chapter 1.

WR01: Fulfilled by utilizing the HTTPWebRequest-classes of .NET.

WR02: Fulfilled by utilizing the HTTPWebResponse-classes of .NET.

WR03: Fulfilled in the GUI.

WR04: Fulfilled via error-handling.

HR01: Fulfilled as application setting [1].

HR02: Fulfilled.

FR01: Fulfilled [2]

FR02: Fulfilled.

FR03: Fulfilled. URL will open in current window.

FR04: Fulfilled.

HR01: Fulfilled.

HR02: Fulfilled. URL will open in current window.

HR03: Fulfilled [3].

PR01: Fulfilled. User can print the HTML-code of the current window.

GR01: Fulfilled. See Chapter 4.

GR02: Fulfilled. See Chapter 4.

AR01: Fulfilled. See Chapter 3 and Chapter 5 for further information.

1 A Windows conform standard path based on the user account is chosen to save the homepage and can not be adjusted.
2 A Windows conform standard path based on the user account is chosen to save the favourites and can not be adjusted.
3 A Windows conform standard path based on the user account is chosen to save the history and can not be adjusted.

AR02: Fulfilled.

Apart from these requirements the following features have been added to the application to enhance the user-experience:

HISTORY DELETION: It is possible for the user to `clear` the history. This will delete all entries that are present.

HISTORY GROUPING: The history-items will be grouped by the date on which they were visited to provide easier navigation-possibilities.

ERROR NOTIFICATION: Upon entering invalid information the user will be informed about the mistakes.

3 DESIGN CONSIDERATIONS

This chapter provides a general overview of the application's design without explaining implementation details.

Therefore, it will provide a general overview of the application's architecture and point out areas of interest in the design: these areas will describe the reasons the current approach has been chosen. If appropriate, the design patterns that were employed will be mentioned and described. Concrete implementation details will later be described in Chapter 5.

3.1 ARCHITECTURAL OVERVIEW

The application was built with the MVP pattern in mind as a basis for the architecture[1].

This way the program logic can be separated from the display logic. To achieve this, the view passes all method-calls needed due to user interaction to the presenter. The presenter routes them to the model if necessary.

All return values will then be passed back to the presenter. The presenter will then decide what action the view needs to take according to the received values. These action will be communicated to the view via an interface the view needs to implement.

This leads to a **decoupling** between the view and the model and allows the presenter to be reusable across multiple views, as a new view only has to implement the interface as well to fulfil the *contract* with the presenter.

A visualisation of this pattern looks like this:

1 Further information about the MVP pattern can be found at MVP-Pattern (`http://msdn.microsoft.com/en-us/magazine/cc188690.aspx`)

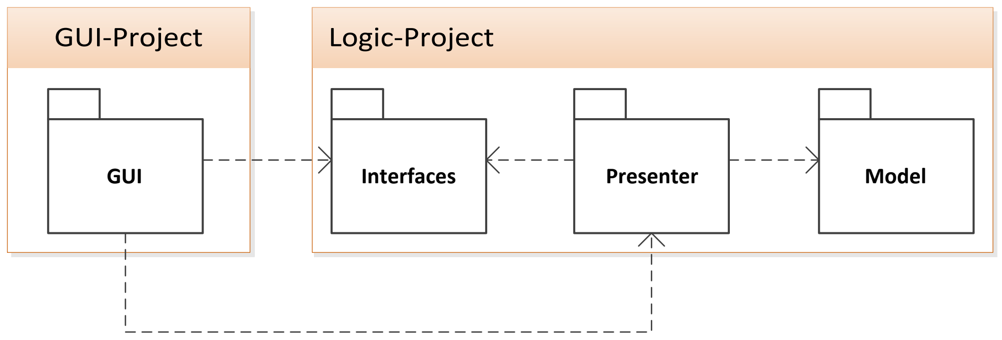


Figure 1: MVP-Pattern

This diagram already shows the possibility of the division into two distinct projects:

GUI: Hosts the GUI implemented in WinForms. This includes all display-related logic: performing actions based on events, displaying and hiding controls, responding to user interaction.

LOGIC: Hosts the *business*-logic, the interfaces and the presenters.

As can already be inferred from the short description, most of the application's logic is based in the logic project, which (excluding the view interfaces) encompasses the following classes:

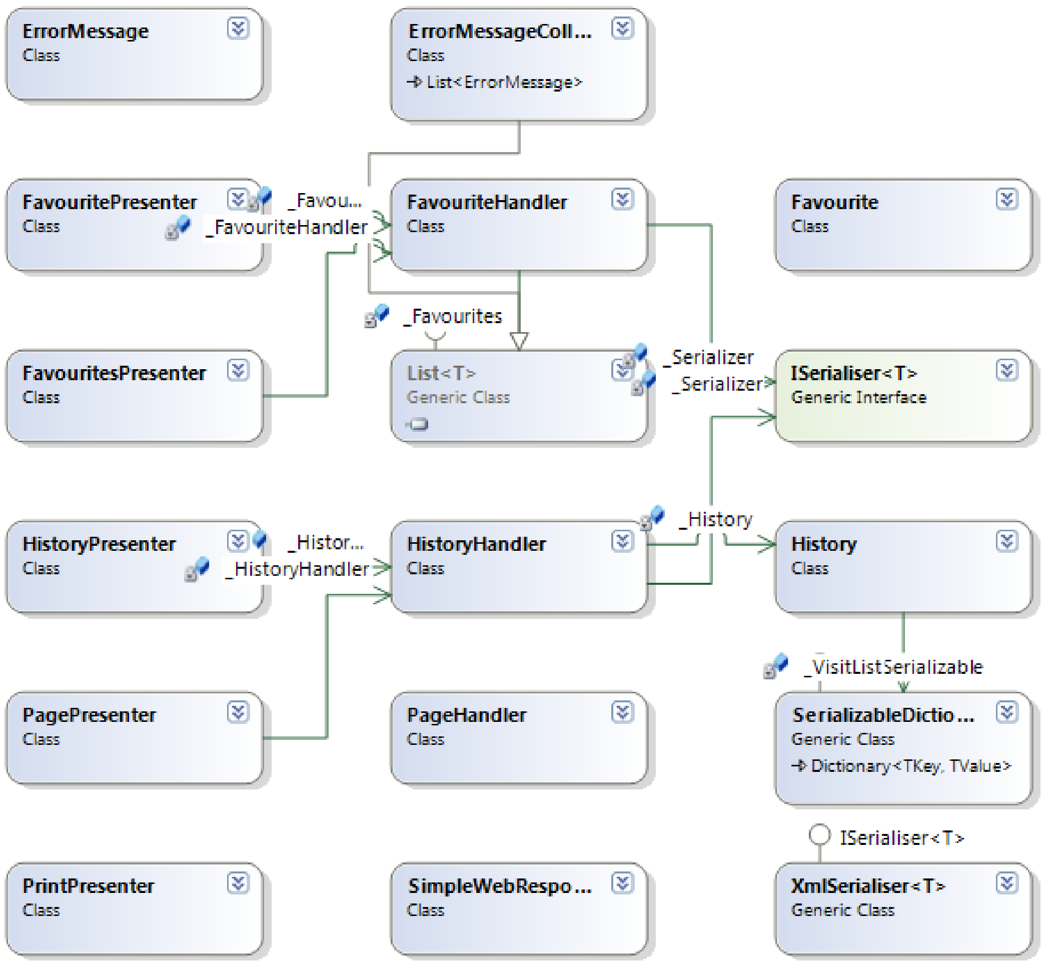


Figure 2: Classes of the logic project.

While implementing the application certain high-level design decisions were made that shall be described in the following two paragraphs.

3.2 SEPARATE CHANGE

The first decision made was to separate changes in distinct classes as much as possible: to achieve this layers of abstraction have been implemented between the presenter and the concrete data classes.

These layers of abstraction are implemented in the classes `FavouriteHandler` and `HistoryHandler`. From a design point-of-view, they provide the presenters that are utilizing them with a facade to the data-classes `History` and `Favourite`.

To centralize changes and prevent data loss due to multiple handlers (e.g. multiple presenter hold a reference to a different HistoryHandler - thus modifying different histories), both objects were implemented via the `Singleton` pattern, ensuring only one object is present at any given time. This way all presenters will modify the same data.

Moreover the handler classes implement an `Observer` pattern to notify the relevant presenters about any changes that occur in the data. This way the presenter can ensure that the view is always displaying accurate data. This approach was necessary, as the `PagePresenter` and the `HistoryPresenter` both operate on the history-data: the `PagePresenter` adds every visited page to the history, whereas the `HistoryPresenter` ensures that the History is displayed in the view.

The same principle applies in the case of the `FavouritePresenter` and the `FavouritesPresenter`. The first one adds and edits one single `Favourite`, whereas the second presenter handles the display of multiple `Favourites` and the provides the necessary logic for deleting `Favourites`.

3.3 LOOSE COUPLING

To provide a loosely coupled application that may be enhanced with little effort at a later time, the communication between distinct logical groups in the application was implemented via interfaces.

The logical groups (that are not denoted separately in the projects) are:

- The View: Realized in a separate project.
- The Business Logic: Realized in the logic project.
- The Persistence Logic: Realized in the logic project in the classes `ISerialiser` and the concrete implementations of the interface.

As already shown in the MVP pattern the presenter $\longleftrightarrow$ view communication is decoupled via interfaces. Moreover the logic $\longleftrightarrow$ persistence communication is decoupled via the `ISerialiser` interface as well.

This will allow a change in the persistence logic without affecting the business logic (for further details see Section 5.2.4).

3.4 THREADING

To fulfil the requirements `AR01` and `AR02` multi threading was introduced in the application when a web-page is requested.

Therefore the request will be carried out in an `asynchronous` thread.

As the view should not be concerned about concrete implementation choices, the decision was made, to introduce multi-threading in the **logic!** (**logic!**) project. Inside this project, it is implemented in the `TextPage`-presenter. This way the application will not loose one of its core-functions should the customer decide to exchange the GUI. Moreover the presenter was chosen to implement the multithreading logic, as all the logical classes did not seem viable to provide the threading functionality. By applying the principle that each object should *do one thing well*, the `PageHandler`-class seemed an ill-fit, as it already provided the functionality to request web-pages.

As a consequence, the presenter - that already manages the flow of information through the application - seemed a better and more fitting candidate to handle the flow of multi-threaded information as well.

Apart from changes to the presenter, certain alterations had to be implemented in the GUI to support multi-threading in a WinForms application (see Section 5.2.1 for the details): should the GUI be exchanged these problems might need to be taken into account again.

4 USER GUIDE

This chapter will describe how to access the five functionalities provided by the application via its GUI:

- Requesting and displaying web-pages.
- Managing the history of requested web-pages.
- Managing user-defined favourites.
- Managing a user-defined home-page.
- Allow printing of the currently displayed web-page.

4.1 REQUESTING A WEB-PAGE

Upon starting the application the user will see the following window:

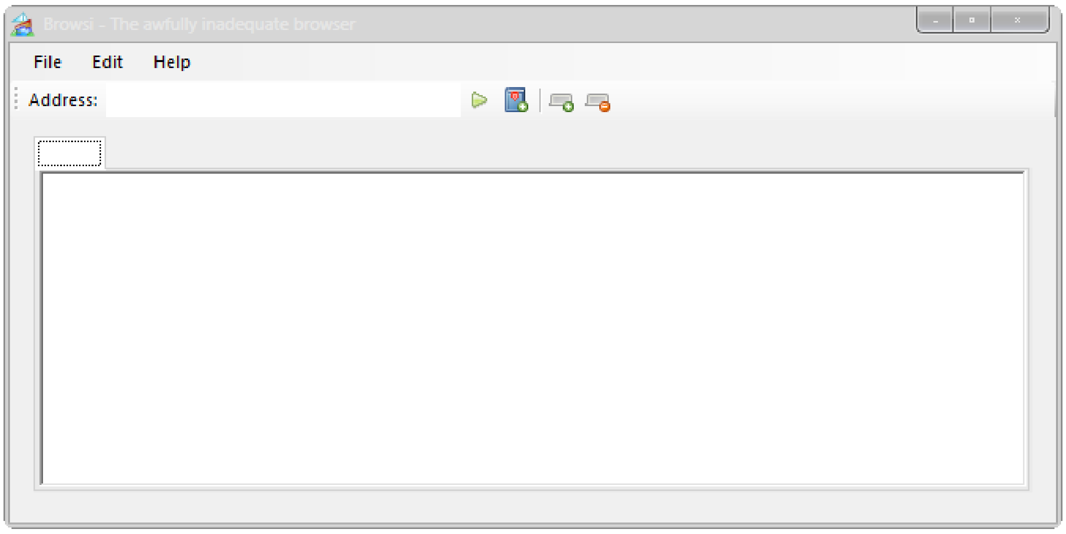


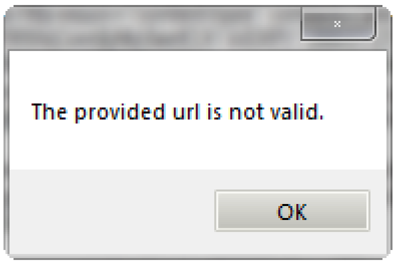
Figure 3: The application's main window.

To request a web-page the user must enter the desired address in the `Address` text-box and then press the `Go`-button (▷).

The format of the entered address will be verified. It must match the following pattern[1]:

```
http://[www.]address.domain[/url-path]
```

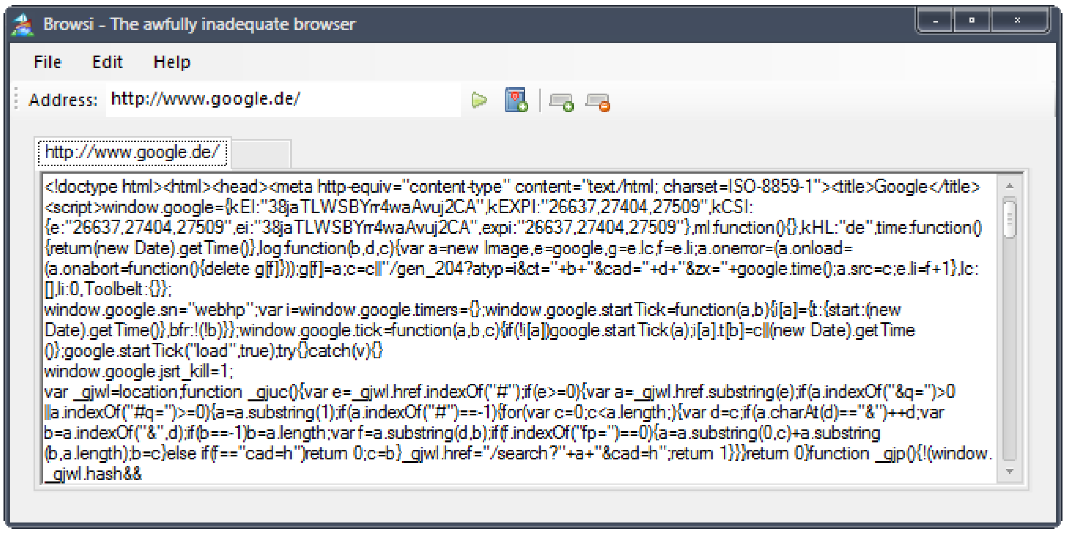
Should the entered address not match the format of a valid web-address, the program will notify the user about the error by displaying a message-box:



The provided url is not valid.

Figure 4: Error message due to invalid url.

When the url is valid the browser will display the HTML-code in the current tab:



```
<!doctype html><html><head><meta http-equiv="content-type" content="text/html; charset=ISO-8859-1"><title>Google</title>
<script>window.google={kEI:"38jaTLWSBYrr4waAvuj2CA",kEXPI:"26637,27404,27509",kCSI:
{e:"26637,27404,27509",ei:"38jaTLWSBYrr4waAvuj2CA",expi:"26637,27404,27509"},ml:function(){},kHL:"de",time:function()
{return(new Date).getTime()},log:function(b,d,c){var a=new Image,e=google,g=e.lc,f=e.li;a.onerror=(a.onload=
(a.onabort=function(){delete g[f]}));g[f]=a;c=c||"/gen_204?atyp=i&ct="+b+"&cad="+d+"&zx="+google.time();a.src=c;e.li=f+1},lc:
[],li:0,Toolbelt:{}};
window.google.sn="webhp";var i=window.google.timers={};window.google.startTick=function(a,b){i[a]={t:{start:(new
Date).getTime()},bfr:!(!b)}};window.google.tick=function(a,b,c){if(!i[a])google.startTick(a);i[a].t[b]=c||(new Date).getTime
()};google.startTick("load",true);try{}catch(v){}
window.google.jsrt_kill=1;
var _gjwl=location;function _gjuc(){var e=_gjwl.href.indexOf("#");if(e>=0){var a=_gjwl.href.substring(e);if(a.indexOf("&q=")>0
||a.indexOf("#q=")>=0){a=a.substring(1);if(a.indexOf("#")==-1){for(var c=0;c<a.length;){var d=c;if(a.charAt(d)=="&")++d;var
b=a.indexOf("&",d);if(b==-1)b=a.length;var f=a.substring(d,b);if(f.indexOf("fp=")==0){a=a.substring(0,c)+a.substring
(b,a.length);b=c}else if(f=="cad=h")return 0;c=b}_gjwl.href="/search?"+a+"&cad=h";return 1}}}return 0}function _gjp(){!(window.
_gjwl.hash&&
```

Figure 5: Browser displaying the current web-page.

To request multiple pages simultaneously it is possible to utilize tabs:

To create a new tab, the user simply presses the `AddTab`-button (). This will create new (and empty) tab-page.

1 Parts in square-brackets may not be required for all web-pages.

Should the user want to close a tab it is necessary to select the tab that should be closed and press the `RemoveTab`-button ().

4.2 MANAGING THE HISTORY

Every web-page requested by the user will be saved in the `history`.

The history can be accessed by pressing the `History`-button () in the `MenuBar` under `Edit` → `History`.

This will display a panel in the main window of the application that shows the `History` in one and the `Favourites` in another tab:

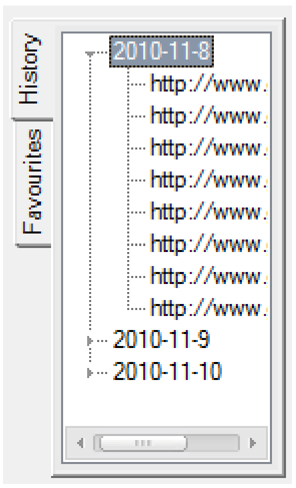


Figure 6: The history tab.

As can be seen the history is ordered in a tree-structure that groups together addresses that were visited on the same day.

To visit one of the pages the user just double-clicks one of the links in the tree and the application will open the requested page in the current tab.

As the history is prone to become quite huge it is possible to clear it. To perform this action the user selects the `ClearHistory`-option () from the `MenuBar`. This operation can be found under `Edit` → `History` → `Clear`.

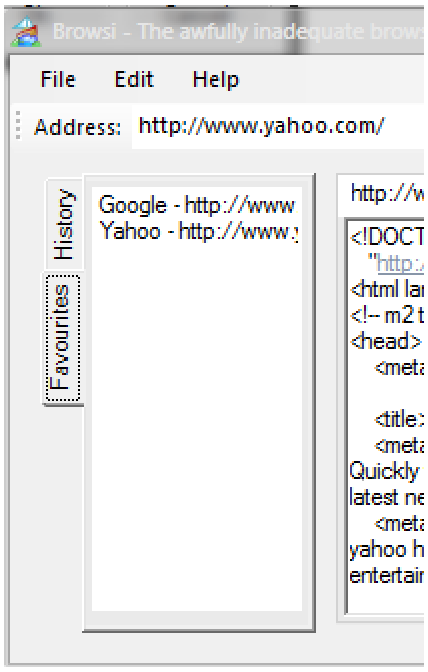
4.3 MANAGING FAVOURITES

The application allows a user to add, edit and delete favourites.

There are two ways to add a favourite: either the user clicks the `AddFavourite`-button () in the `AddressBar` or in the `ContextMenu` of the Favourites-List.

To open the `Favourites-List` the user clicks the `Favourite`-button in the `MenuBar`: `Edit` → `Favourites`.

This will open the Favourites-List:



Google - http://www
Yahoo - http://www.

http://w

<!DOCT
"http:
<html la
<!-- m2 t
<head>
<meta
<title>
<meta
Quickly
latest ne
<meta
yahoo h
entertai

Figure 7: The favourites tab.

When the user right-clicks in this list a context-menu will open that will allow the user to either `Add`, `Edit` or `Delete` a favourite.

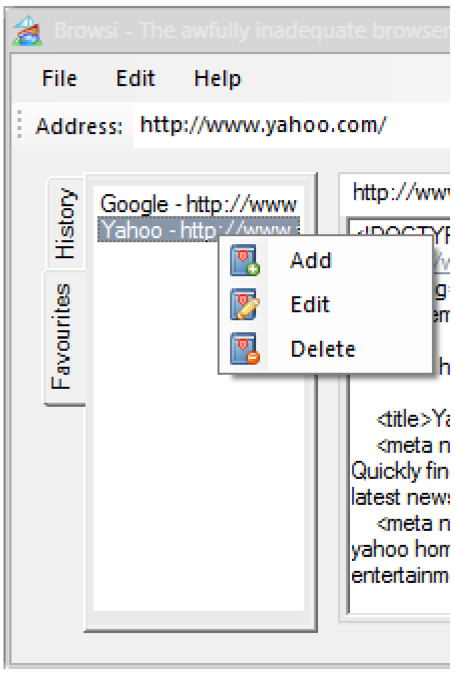


Figure 8: The context menu.

When the user decides to add a favourite, a new window will pop-up that asks him to enter a name and a URL for the favourite. The URL will, by default, be set to the address of the currently activated tab.

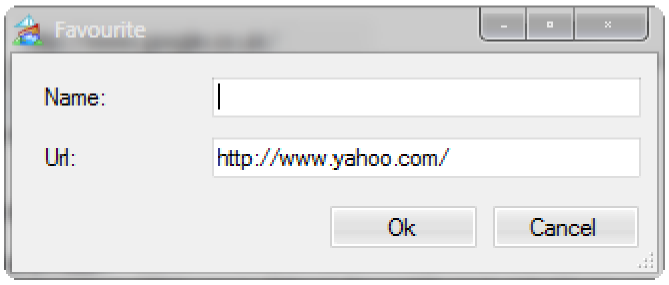


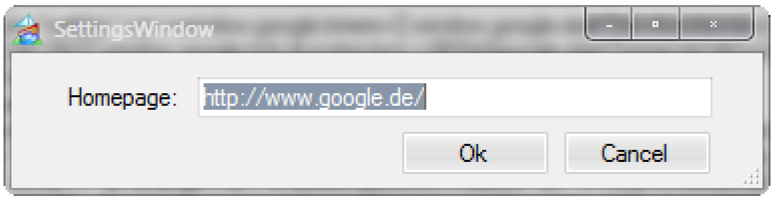
Figure 9: Add-favourite window.

4.4 SETTING A HOME-PAGE

It is also possible to set a home-page that will be opened as soon as the application is started.

Therefore the user opens the `Settings`-Window by clicking on the `Settings`-button ( ) found under `File` → `Settings`.

This will open the `Settings`-Window that allows the user to enter a home-page:



SettingsWindow

Homepage: http://www.google.de/

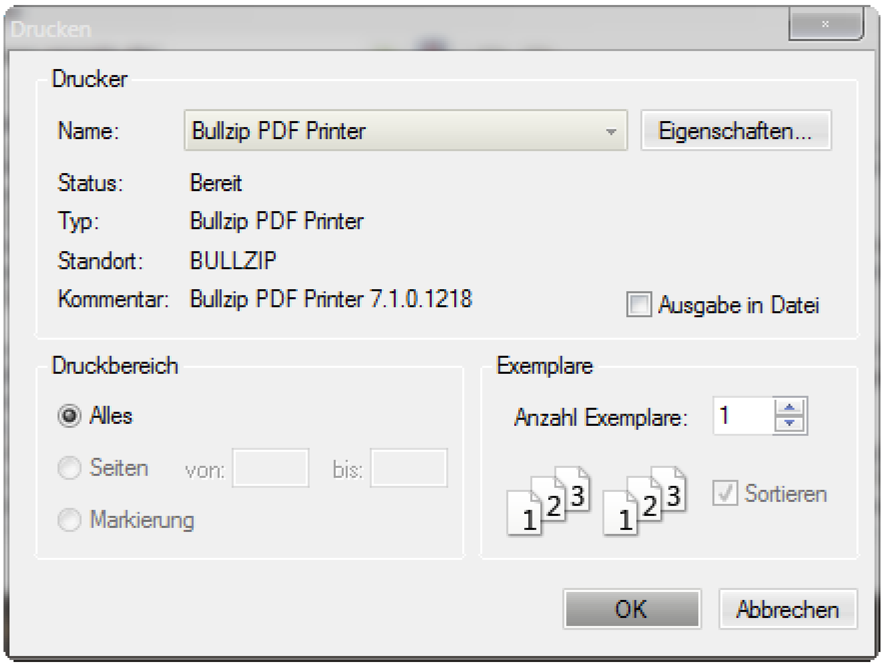
Figure 10: Settings window.

4.5 PRINTING A WEB-PAGE

The last function the user can perform is to print a web-page.

Therefore the tab that should be printed needs to be selected and then the `Print`-button () needs to be clicked (`File` → `Print`).

This will display a print-dialog that allows the user to tweak desired settings:



Drucken

Drucker

Name:	Bullzip PDF Printer
Status:	Bereit
Typ:	Bullzip PDF Printer
Standort:	BULLZIP
Kommentar:	Bullzip PDF Printer 7.1.0.1218

☐ Ausgabe in Datei

Druckbereich

[✓ Alles] Seiten von: bis: Markierung

Exemplare

Anzahl Exemplare: 1

☑ Sortieren

Figure 11: Common windows print-dialog.

Clicking the `OK`-button will the print the HTML of the current tab.

5 DEVELOPER GUIDE

This chapter will guide the reader through the different requirements and their implementation in the application.

Having finished with the requirements a section highlighting certain problems or special cases that were encountered during the development will be described.

5.1 IMPLEMENTATION OF THE REQUIREMENTS

5.1.1 *Web-requirements*

Requirements `WR01`, `WR02` and `WR03` are implemented in the `PageHandler`-class:

It requests a provided URL via its `FetchUrl()`-method that will return a `SimpleWebResponse`-object. The `SimpleWebResponse`-object encapsulates just a title, body and a url - all elements needed to display a web-page.

To fulfil requirement `WR04` it is necessary to assign a `WebResponse` after catching an exception that is thrown by the `.NET`-framework if a response does not contain a 200 (OK)-message:

```
try
{
        this.Request = WebRequest.Create(this.RequestUrl);
        this.Response = this.Request.GetResponse();
}
catch (WebException e)
{
        logger.Error("WebException ({0}) occured when fetching the url: {1}", e.Message
            , this.RequestUrl);
        this.Response = e.Response;
}
```

Listing 1: Fetching URLs with error-codes

`WR04` is implemented in the `PagePresenter`: the page-presenter starts a `FetchUrl()`-method-call via an asynchronous delegate and provides a callback-mechanism to its `Done()`-method.

```
Func<SimpleWebResponse> method = pageHandler.FetchUrl;
method.BeginInvoke(Done, method);
```

Listing 2: Fetching URLs with error-codes

When the `Done()`-method is invoked, the presenter calls the view's `DisplayWebpage()`-method that prints the received HTML-code[1].

5.1.2 *Homepage-Requirements*

The homepage is saved as a simple string. This leads to the possibility to save it via the `ApplicationSettings`[2] - a facility provided by the `.NET` framework.

The only problem in this implementation is the fact that only the `WinForms` project is allowed to access the application-settings[3].

Due to this reason the code to write the string is located in the `SettingsWindow`:

```
private void SaveSettings(String homePage)
        {
            Settings settings = Settings.Default;
            settings.Homepage = homePage;
            settings.Save();
        }
```

Listing 3: Saving the homepage-settings.

5.1.3 *Favourite-requirements*

The requirements `FR01` and `FR02` were implemented in the `Favourite` and `FavouriteHandler`-classes.

Apart from handling the adding (`AddEntry()`), deleting (`DeleteFavourite()`) and editing (`EditFavourite()`) of favourites, the `FavouriteHandler` also handles the saving (`SaveFavourite()`) and loading (`LoadFavourites()`) of favourites.

Requirement `FR04` is fulfilled by the `FavouritePresenter`: upon creation, the presenter determines the file-path for the favourite file and sets it in the `FavouriteHandler`:

```
private void SetUpHandler()
        {
            String appFolder = Environment.GetFolderPath(Environment.SpecialFolder.
                ApplicationData);
            String history = "Favourites.xml";
            this._FavouriteHandler.SetFilePath(Path.Combine(appFolder, history));
            this._FavouriteHandler.LoadFavourite();
        }
```

1 See Section 5.2.1 to see details of the necessary view-implementation.
2 See http://msdn.microsoft.com/en-us/library/k4s6c3a0.aspx (`http://msdn.microsoft.com/en-us/library/k4s6c3a0.aspx`) for more information
3 It would be possible to provide a reference to the GUI-project in the logic-project. However this would lead to a dependency between the logic-project and the GUI - a circumstance that should be prevented by utilizing the MVP-pattern. Due to this fact the GUI writes the application settings and the logic stays independent from the GUI.

Listing 4: Setting up the favourite handler's filepath.

This approach was chosen to keep the `FavouriteHandler` reusable, whereas the presenter is already closely tied to a concrete implementation, being responsible for handling the information flow.

Requirement `FR03` was implemented in the GUI: when the user clicks an element in the favourite-list, the element is determined and the url of the favourite is written into the URL-text-field. After this has happened, a standard URL-request is issued and the URL will be displayed in the active tab.

5.1.4 *History-requirements*

The implementation of the history-requirements is similar to the those of the favourites.

The main-classes are `History` and `HistoryHandler`, whereas the loading on start-up is provided by the `HistoryPresenter`.

```
private void SetUpHandler()
        {
            String appFolder = Environment.GetFolderPath(Environment.SpecialFolder.
                ApplicationData);
            String history = "History.xml";
            this._HistoryHandler.SetFilePath(Path.Combine(appFolder, history));
            this._HistoryHandler.LoadHistory();
        }
```

Listing 5: Setting up the history handler's filepath.

The loading of a URL on user-interaction is implemented in the GUI via a `NodeMouseClick-Event` on the `History-TreeView`.

5.1.5 *Printing-requirements*

Requirement `PR01` was implemented in the `MainWindow` and the `PrintPresenter`-classes.

The `PrintPresenter`'s `Print()`-method will be called for every page that needs to be printed.

The method will determine the size of the String that shall be printed and set the `HasMorePages`-property to true if the String would not fit onto one page:

```
public void Print(System.Drawing.Printing.PrintPageEventArgs e)
        {

            Font font = _PrintView.CurrentFont;
            int charactersOnPage = 0;
            int linesPerPage = 0;

            // Sets the value of charactersOnPage to the number of characters
            // of PrintString that will fit within the bounds of the page.
            e.Graphics.MeasureString(PrintString, font,
                e.MarginBounds.Size, StringFormat.GenericTypographic,
                out charactersOnPage, out linesPerPage);

            // Draws the string within the bounds of the page
            e.Graphics.DrawString(PrintString, font, Brushes.Black,
                e.MarginBounds, StringFormat.GenericTypographic);

            // Remove the portion of the string that has been printed.
            PrintString = PrintString.Substring(charactersOnPage);

            // Check to see if more pages are to be printed.
            e.HasMorePages = (PrintString.Length > 0);
        }
```

Listing 6: Printing.

5.1.6 *User-interface requirements*

All actions can be performed via a GUI: see Chapter 4 for an introduction of how to use the provided GUI.

As the GUI was implemented via the `WinForms` designer and all actions that are not just altering the GUI are passed to the presenters, no significant logic that has not already been mentioned is implemented in the GUI.

Due to this fact no more implementation-details about the requirements `GR01` and `GR02` will be described.

5.2 DETAILS OF THE IMPLEMENTATION

Apart from just providing an overview over the requirements and their corresponding implementation, this section will provide an overview over certain areas of code that might be hard to understand without further explanation, but are not directly linked to a certain requirement.

5.2.1 *View*

All GUI classes have a common ancestor: the class `ThreadingView`:

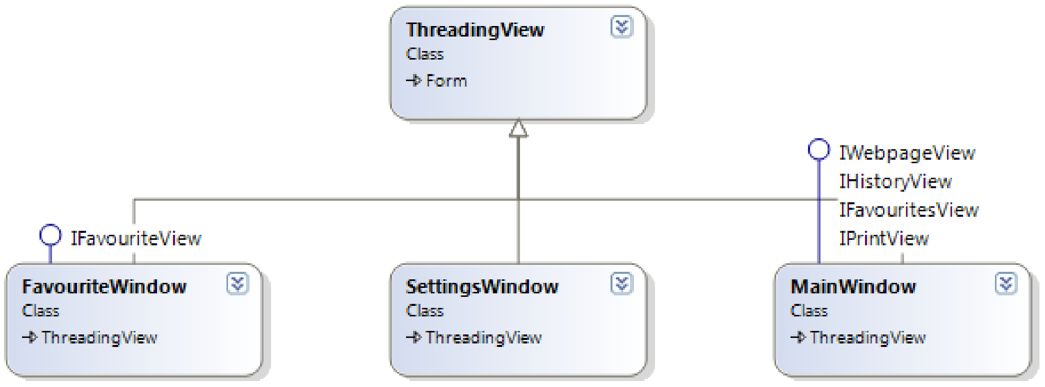


Figure 12: GUI class hierarchy

Due to this approach all classes are able to use the `UpdateUI()`-method of this class when another thread wants to alter a GUI-control:

```
protected void UpdateUI(MethodInvoker uiDelegate)
        {
            if (InvokeRequired)
            {
                this.Invoke(uiDelegate);
            }
            else
            {
                uiDelegate();
            }
        }
```

Listing 7: Updating the GUI from the GUI-thread.

This is necessary as only the thread that created a GUI-component is allowed to update it. Via `InvokeRequired` it is possible to check if another thread tries to modify the component (returns `true` if another thread wants to modify it, `false` otherwise).

In the current application this behaviour might occur when the page-request-thread calls the `PagePresenter's done()` method and the presenter tries to force the view to display the web-page. To prevent an exception from ocurring the code to update the GUI-component is called like this:

```
public void DisplayWebPage(SimpleWebResponse response)
        {
            foreach (TabPage page in this.webSitesTabControl.TabPages)
            {
                if (page.Name.Equals(response.Url))
                {
                    MethodInvoker uiDelegate = delegate
                    {
                        page.Controls[0].Text = response.Html;
                        page.Text = response.Title;
                    };
                    UpdateUI(uiDelegate);
                }
            }
        }
```

Listing 8: Displaying the web-page from the GUI-thread.

This way the GUI-thread will alter the component.

5.2.2 *Observer-pattern*

As already mentioned in Section 3.2 the observer pattern is used to keep the different presenters up-to-date. Namely this approach concerns the `Favourite`- and `HistoryHandlers` and their respective presenters.

However the implementation was not performed utilizing multiple classes as described in Gamma et al. (1994), but by utilizing `.NET` specific concepts like delegates.

Therefore the handlers declare a delegate that the observers can use. Moreover an event needs to be declared that will chain the multiple methods of the observers, so that every observer will be notified of occuring changes.

The concrete implementation-stubs look like this:

```
public delegate void ChangeHandler(object subject);
public event ChangeHandler ChangeEvent;
```

Listing 9: Declaration of delegate and event.

When a change occurs the `Notify()`-method will be called that notifies all observers of the change:

```
private void Notify()
{
      if (ChangeEvent != null)
      {
           ChangeEvent(this);
      }
}
```

Listing 10: Notifying observers.

The observers can then handle the change in their subscribed method.

To subscribe, the following chaining to the event is sufficient:

```
this._HistoryHandler.ChangeEvent += new HistoryHandler.ChangeHandler(this.Update);
```

Listing 11: Registering as an observer.

After that the `Update()`-method can handle all changes:

```
public void Update(object subject)
{
      if (subject is HistoryHandler)
      {
          HistoryHandler histhandler = subject as HistoryHandler;
          this._HistoryView.DisplayHistory(histhandler.History);
      }
}
```

Listing 12: Handling changes in the observer.

5.2.3 *Singleton-pattern*

Another pattern that was used in the `Handler`-classes was the `Singleton`-pattern.

The pattern was employed due to the fact that two different presenters access each handler and both presenter need to modify the same data.

However, the implementation of the singleton is the standard one found in many object-oriented languages:

```
private static HistoryHandler _Instance;

/// <summary>
/// Gets the instance.
/// </summary>
/// <value>The instance.</value>
public static HistoryHandler Instance
{
            get
            {
                if (_Instance == null)
                {
                    lock (lockObject)
                    {
                        if (_Instance == null)
                        {
                            _Instance = new HistoryHandler();
                        }
                    }
                }
                return _Instance;
            }
        }
```

Listing 13: Thread-safe lazy-initialization.

5.2.4 *Serialisation*

The last thing to be pointed out is the implementation of the persistence-functionality.

The two classes responsible to persist the data (`FavouriteHandler` and `HistoryHandler`) refer to the persisting class only via its interface (`ISerialiser<T>`) making use of polymorphism. This way the implementation can easily be swapped out to support a different kind of serialisation.

In the delivered application the persisting of the history and the favourites is performed via the Extensible Markup Language (XML)-format. Should another format be used it suffices to create a new class that implements the `ISerialiser<T>`-interface and provide the new mechanism for serialisation.

Then it is enough to instantiate an instance of this new class in the two handlers.

Due to the fact that the current implementation utilises `Generics` one class can be used for multiple classes.

However, while implementing the application certain limitations of the `.NET`-framework made it necessary to "implement"[4] a new `SerializableDictionary`-class, as the dictionaries provided by the .NET framework are not serialisable to an XML-representation.

4 The used implementation was taken from `http://weblogs.asp.net/pwelter34/archive/2006/05/03/444961.aspx` - as stated in the comments this implementation was tested against 30 test-cases and due to this fact preferred to a self-implemented one.

6 TESTING

The testing of the application was performed in two stages:

- In the early development stages unit-tests were written for the base classes.
- After chaining the application parts together, the applications correct behaviour was mainly tested by using the application. This was necessary as unit-testing GUI and multi-threaded code is extremely difficult.

6.1 UNIT-TESTS

The source-code of the written unit-tests can be seen in Section A.3.

To run the test-cases the NUnit-framework (`http://www.nunit.org/`) is needed.

The test-cases cover the passing of different types of parameters.

A short list of tests shall be provided to give a short overview:

Class Under Test	Method	Input	Output
FavouriteHandler	EditFavourite	null	ArgumentNull-Exception
FavouriteHandler	EditFavourite	Favourite that is not present	Argument-Exception
FavouriteHandler	DeleteFavourite	null	ArgumentNull-Exception
FavouriteHandler	DeleteFavourite	Favourite that is not present	Argument-Exception
FavouriteHandler	AddFavourite	-	List contains one more element
FavouriteHandler	AddFavourite - Delete Favourite	Add Favourite and delete same	List size is unchanged
FavouriteHandler	EditFavourite	Add Favourite and edit same	Favourite is changed
Favourite	Create	Valid arguments	Favourite is created
Favourite	Create	Illegal-URL	Argument-Exception

Favourite	Create	Null-URL	Argument-Exception
Favourite	Create	Empty-String-Name	Argument-Exception
Favourite	Create	Null-Name	Argument-Exception
History	Create	Valid arguments	History is created
History	Create	Invalid-URL	Argument-Exception
XMLSerialiser	Write	No Filepath	NoFilePathSet-Exception
XMLSerialiser	Read	No Filepath	NoFilePathSet-Exception
PageHandler	FetchURL	Valid-URL	Html-Code. Status Code 200
PageHandler	FetchURL	Valid-URL but not found	Html-Code. Status Code 404[1]
PageHandler	IsValidURL	Valid-URL	True
PageHandler	IsValidURL	Invalid-URL	False

Table 1: Performed tests.

6.2 ACCEPTANCE-TESTS

After testing the base classes the main testing was performed by running the application and entering different kinds of input:

Tests included the correct display of error-messages if a field containing a URL was filled in incorrectly or if values were not provided that were needed (e.g. omitting a name for a favourite).

A list of performed tests:

Under Test	Input	Output
MainWindow	Invalid-URL	ErrorMessage
MainWindow	Valid-URL	Display HTML

1 As no certain way exists to generate messages that provoke 400 or 403 status codes, these could not be tested in a unit test.

MainWindow	Valid-address but resource not found (404)	Display HTML
MainWindow	Add-Bookmark	Display add bookmark window
MainWindow	Show history	Display history window
MainWindow	Show favourites	Display favourites window
MainWindow	Show settings	Display settings window
MainWindow	Click favourite-URL	Request favourite-URL
MainWindow	Double-click history-URL	Request history-URL
MainWindow	Click clear history	Delete history
MainWindow	Request multiple pages	Display HTML
MainWindow	Request multiple pages with some wrong	Display HTML for valid, error message for invalid URLs
MainWindow	Click print button	Display print-dialog
AddFavouriteWindow	Click cancel	Close window without change
AddFavouriteWindow	Add bookmark with valid input	Add bookmark close window
AddFavouriteWindow	Add bookmark invalid values	Display error message, dispose without change
SettingsWindow	Click cancel	Close window without change
SettingsWindow	Add homepage with valid URL	Change homepage and close window
SettingsWindow	Add homepage with invalid URL	Display error message and close window without change
Printing	Click cancel	Close window without printing
Printing	Click ok	Print current tab's HTML

Table 2: Performed tests.

Even though this can not guarantee that the application is error free, it provides a good measure that it *should* work reliably in most cases.

7 CONCLUSIONS

To conclude this report a short summary of the achieved goals (apart from fulfilling the requirements) should be provided:

The application's design tries to be robust, yet adaptable if changes need to be made. To ensure the achieving of these goals, patterns were used where it seemed appropriate.

Moreover the loose coupling should help in the achieving of these goals as well.

The approach to provide a well-designed application from a software engineering point-of-view may have prevented the addition of many convenience features. However, it seemed more important to deliver an easily adaptable program: exchanging the GUI or the persistence mechanism should be fairly easy. The presenters are already in place thus that only the GUI classes need to be modified If the persistence mechanism should be changed the only thing that needs to be provided is a new implementation that adheres to the `ISerialiser<T>` interface.

However concerning the persistence, there is one major flaw in the application: the fact that the `ApplicationSettings` need to be persisted in the WinForms-project. The development of a small framework to allow a similar mechanism in other projects might prevent this circumstance in future projects.

Overall the application should fulfil the requirements, which seems to be the most important goal to achieve in the first place.

Part II

APPENDIX

A APPENDIX: SOURCE CODE

A.1 GUI

Project: GUI

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Assessment_Two
{
    /// <summary>
    /// Class provides all inheriting views with delegate mechanism to allow threads
        to redraw elements on the main thread.
    /// </summary>
    public class ThreadingView : Form
    {
        protected void UpdateUI(MethodInvoker uiDelegate)
        {
            if (InvokeRequired)
            {
                this.Invoke(uiDelegate);
            }
            else
            {
                uiDelegate();
            }
        }
    }
}
```

Listing 14: ThreadingView.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Assessment_Two_Logic.Interfaces;
using Assessment_Two_Logic.Model;
using Assessment_Two_Logic.Presenter;
using Assessment_Two.Properties;

namespace Assessment_Two
{
    public partial class MainWindow : ThreadingView, IWebpageView, IHistoryView,
        IFavouritesView, IPrintView
    {
        String _StringToPrint;

        private int _NumberOfTabs;
        private int _ReuestedPages;

        private PagePresenter _PagePresenter;
        private HistoryPresenter _HistoryPresenter;
        private FavouritesPresenter _FavouritesPresenter;
        private PrintPresenter _PrintPresenter;

        public MainWindow()
        {
            InitializeComponent();
            this._NumberOfTabs = 0;
            this.CreateTab();

            this.splitContainer1.Panel1Collapsed = true;
            this.splitContainer1.Panel1.Hide();

            this._PagePresenter = new PagePresenter(this);
            this._HistoryPresenter = new HistoryPresenter(this);
            this._FavouritesPresenter = new FavouritesPresenter(this);
            this._PrintPresenter = new PrintPresenter(this);

            LoadHomePage();
        }

        private void LoadHomePage()
        {
            Settings settings = Settings.Default;
            String homePage = settings.Homepage;

            this.urlTextBox.Text = homePage;
            if (!String.IsNullOrEmpty(homePage))
            {
                TabPage tp = webSitesTabControl.SelectedTab;
                tp.Name = homePage;
                this._PagePresenter.RequestWebpage();
            }
            else
            {
                this.urlTextBox.Text = "http://";
            }
        }

        #region Interfaces

        public string Url
        {
            get
            {
                return this.urlTextBox.Text;
            }
            set
            {
                MethodInvoker uiDelegate = delegate
                {
                    this.urlTextBox.Text = value;
                };
                UpdateUI(uiDelegate);
            }
        }

        public string SiteText
        {
            get
            {
                TabPage page = this.webSitesTabControl.SelectedTab;
                return page.Controls[0].Text;
            }
        }

        public void DisplayErrors(ErrorMessageCollection errors)
        {
            MessageBox.Show(errors.ToString());
        }

        public Favourite Favourite
        {
            get
            {
                return (Favourite)this.favouriteListBox.SelectedItem;
            }
        }

        public void DisplayHistory(History history)
        {
            this.historyTreeView.Nodes.Clear();
            HashSet<String> coll = new HashSet<String>();
            foreach (DateTime t in history.VisitList.Keys)
            {
                String item = String.Format("{0}-{1}-{2}", t.Year.ToString(), t.Month.ToString(), t.Day.ToString());
                coll.Add(item);
            }
            foreach (String str in coll)
            {
                this.historyTreeView.Nodes.Add(str);
            }
            foreach (KeyValuePair<DateTime, String> t in history.VisitList)
            {
                String item = String.Format("{0}-{1}-{2}", t.Key.Year.ToString(), t.Key.Month.ToString(), t.Key.Day.ToString());
                foreach (TreeNode tn in historyTreeView.Nodes)
                {
                    if (tn.Text.Equals(item))
                    {
                        tn.Nodes.Add(t.Value);
                    }
                }
            }
        }

        public void DisplayFavourites(ICollection<Assessment_Two_Logic.Model.Favourite> favourites)
        {
            this.favouriteListBox.Items.Clear();
            foreach (Favourite fav in favourites)
            {
                this.favouriteListBox.Items.Add(fav);
            }
        }

        public string Print
        {
            get { return this.SiteText; }
        }

        public Font CurrentFont
        {
            get
            {
                Font font = this.Font;
                return font;
            }

        }

        public void DisplayWebPage(SimpleWebResponse response)
        {
            foreach (TabPage page in this.webSitesTabControl.TabPages)
            {
                if (page.Name.Equals(response.Url))
                {
                    MethodInvoker uiDelegate = delegate
                    {
                        page.Controls[0].Text = response.Html;
                        page.Text = response.Title;
                    };
                    UpdateUI(uiDelegate);
                }
            }
        }
        #endregion

        #region Eventhandler

        private void addTabToolStripButton_Click(object sender, EventArgs e)
        {
            this.CreateTab();
        }

        private void deleteTabToolStripButton_Click(object sender, EventArgs e)
        {
            this.DeleteTab();
        }

        private void goToolStripButton_Click(object sender, EventArgs e)
        {
            String url = this.urlTextBox.Text;
            this.webSitesTabControl.SelectedTab.Name = url;
            this._PagePresenter.RequestWebpage();
        }

        private void historyToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (!IsPanelVisible())
            {
                ChangePanelVisibility();
                DisplayHistoryPage();
            }
            else
            {
                if (IsHistoryTabVisible())
                {
                    ChangePanelVisibility();
                }
                else
                {
                    DisplayHistoryPage();
                }
            }
        }

        private void addFavouriteToolStripButton_Click(object sender, EventArgs e)
        {
            AddFavourite();
        }

        private void treeView_NodeMouseClick(object sender,
            TreeNodeMouseClickEventArgs e)
        {
            TreeNode tn = this.historyTreeView.SelectedNode;
            if (tn != null)
            {
                if (tn.Level == 1)
                {
                    RequestPage(tn.Text);
                }
            }
        }

        private void addToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.AddFavourite();
        }

        private void editToolStripMenuItem1_Click(object sender, EventArgs e)
        {
            FavouriteWindow fw = new FavouriteWindow();

            fw.IsEdit = true;
            Favourite fav = (Favourite)this.favouriteListBox.SelectedItem;
            fw.Favourite = fav;
            fw.Url = fav.Url;
            fw.FavName = fav.Name;
            fw.ShowDialog();
        }

        private void favouritesToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (!IsPanelVisible())
            {
                ChangePanelVisibility();
                this.DisplayFavouritesPage();
            }
            else
            {
                if (!IsHistoryTabVisible())
                {
                    ChangePanelVisibility();
                }
                else
                {
                    DisplayFavouritesPage();
                }
            }
        }

        private void favouriteListBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            var item = this.favouriteListBox.SelectedItem;
            if (item != null)
            {
                Favourite fav = item as Favourite;
                this.RequestPage(fav.Url);
            }
        }

        private void deleteToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this._FavouritesPresenter.DeleteFavourite();
        }

        private void printToolStripMenuItem_Click(object sender, EventArgs e)
        {
            printDialog = new PrintDialog();
            printDialog.Document = printDocument;
            DialogResult result = printDialog.ShowDialog();
            this._PrintPresenter.SetPrintString();
            if (result == DialogResult.OK)
            {
                // ToDo: Make this call Async?
                printDocument.Print();
            }
        }

        private void settingsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            SettingsWindow sw = new SettingsWindow();
            sw.ShowDialog();
        }

        private void MainWindow_FormClosing(object sender, FormClosingEventArgs e)
        {
            this.Save();
        }

        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.Save();
            this.Dispose();
        }

        private void printDocument_PrintPage(object sender, System.Drawing.Printing.
            PrintPageEventArgs e)
        {
            this._PrintPresenter.Print(e);
        }

        private void webSitesTabControl_SelectedIndexChanged(object sender, EventArgs
            e)
        {
            if (webSitesTabControl.SelectedTab != null)
            {
                String newUrl = webSitesTabControl.SelectedTab.Name;
                if (!newUrl.StartsWith("http://"))
                {
                    this.urlTextBox.Text = "http://";
                }
                else
                {
                    this.urlTextBox.Text = newUrl;
                }

            }
        }
        #endregion

        private void CreateTab()
        {
            this._NumberOfTabs++;
            String nameOfNewTab = "tabPage" + this._NumberOfTabs;
            String nameOfTextBox = "webPage" + this._NumberOfTabs + "RichTextBox";
            this.webSitesTabControl.SuspendLayout();
            TabPage tb = new TabPage();
            this.webSitesTabControl.TabPages.Add(tb);
            tb.Name = nameOfNewTab;

            RichTextBox rtb = new RichTextBox();
            rtb.Location = new Point(3, 3);
            rtb.Name = nameOfTextBox;
            rtb.Dock = DockStyle.Fill;

            tb.Controls.Add(rtb);
            this.webSitesTabControl.ResumeLayout();
        }

        private void DeleteTab()
        {
            TabPage currentTab = this.webSitesTabControl.SelectedTab;
            if (currentTab != null)
            {
                this.webSitesTabControl.TabPages.Remove(currentTab);
                this._NumberOfTabs--;
            }
            if (this._NumberOfTabs.Equals(0))
            {
                this.CreateTab();
            }
        }

        private bool IsHistoryTabVisible()
        {
            bool visible = false;
            if (this.sideTabControl.SelectedTab == this.sideTabControl.TabPages[0])
            {
                visible = true;
            }
            return visible;
        }

        private bool IsPanelVisible()
        {
            return !this.splitContainer1.Panel1Collapsed;
        }

        private void DisplayHistoryPage()
        {
            this.sideTabControl.SelectedTab = this.sideTabControl.TabPages[0];
        }

        private void ChangePanelVisibility()
        {
            Boolean isInvisible = this.splitContainer1.Panel1Collapsed;
            if (isInvisible)
            {
                this.splitContainer1.Panel1Collapsed = false;
                this.splitContainer1.Panel1.Show();
            }
            else
            {
                this.splitContainer1.Panel1Collapsed = true;
                this.splitContainer1.Panel1.Hide();
            }
        }

        private void AddFavourite()
        {
            FavouriteWindow fw = new FavouriteWindow();
            fw.Url = this.urlTextBox.Text;
            fw.ShowDialog();
        }

        private void DisplayFavouritesPage()
        {
            this.sideTabControl.SelectedTab = this.sideTabControl.TabPages[1];
        }

        private void Save()
        {
            this._FavouritesPresenter.SaveFavourites();
            this._HistoryPresenter.SaveHistory();
        }

        private void clearToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this._HistoryPresenter.ClearHistory();
        }

        private void favouriteContextMenu_Opening(object sender, CancelEventArgs e)
        {
            Object favourite = favouriteListBox.SelectedItem;
            if (favourite != null)
            {
                this.EnableContextButtons(true);
            }
            else
            {
                this.EnableContextButtons(false);
            }
        }

        private void EnableContextButtons(bool p)
        {
            this.editToolStripMenuItem1.Enabled = p;
            this.deleteToolStripMenuItem.Enabled = p;
        }

        private void RequestPage(String text)
        {
            this.urlTextBox.Text = text;
            this.webSitesTabControl.SelectedTab.Name = text;
            this._PagePresenter.RequestWebpage();
        }

        private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
        {
            AboutBox ab = new AboutBox();
            ab.ShowDialog();
        }
    }
}
```

Listing 15: MainWindow.cs

```
ï»¿using System;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Assessment_Two_Logic.Interfaces;
using Assessment_Two_Logic.Presenter;
using Assessment_Two_Logic.Model;

namespace Assessment_Two
{
    public partial class FavouriteWindow : ThreadingView, IFavouriteView
    {
        private FavouritePresenter _FavouritePresenter;

        private Favourite _Favourite;

        public bool IsEdit { get; set; }

        public FavouriteWindow()
        {
            InitializeComponent();
            this.IsEdit = false;
            this._FavouritePresenter = new FavouritePresenter(this);
        }

        public string Url
        {
            get
            {
                return this.urlTextBox.Text;
            }
            set
            {
                MethodInvoker uiDelegate = delegate
                {
                    urlTextBox.Text = value;
                };
                UpdateUI(uiDelegate);
            }
        }

        public string FavName
        {
            get
            {
                return this.nameTextBox.Text;
            }
            set
            {
                MethodInvoker uiDelegate = delegate
                {
                    nameTextBox.Text = value;
                };
                UpdateUI(uiDelegate);
            }
        }

        public Assessment_Two_Logic.Model.Favourite Favourite
        {
            get
            {
                return this._Favourite;
            }
            set
            {
                this._Favourite = value;
            }
        }

        public void DisplayErrors(Assessment_Two_Logic.Model.ErrorMessageCollection
            errors)
        {
            MessageBox.Show(errors.ToString());
        }

        private void okButton_Click(object sender, EventArgs e)
        {
            if (!IsEdit)
            {
                this._FavouritePresenter.AddFavourite();
            }
            else
            {
                this._FavouritePresenter.EditFavourite();
            }
            // ToDo: Only dispose if changes work.
            this.Dispose();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.Dispose();
        }


    }
}
```

Listing 16: FavouriteWindow.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Assessment_Two.Properties;
using Assessment_Two_Logic.Model;

namespace Assessment_Two
{
    public partial class SettingsWindow : ThreadingView
    {
        public SettingsWindow()
        {
            InitializeComponent();
            LoadSettings();
        }

        private void LoadSettings()
        {
            Settings settings = Settings.Default;
            this.homepageTextBox.Text = settings.Homepage;
        }

        private void okButton_Click(object sender, EventArgs e)
        {
            String homepage = homepageTextBox.Text;
            this.SaveSettings(homepage);
            this.Dispose();
        }

        private void SaveSettings(String homePage)
        {
            if (!PageHandler.IsValidUrl(homePage))
            {
                MessageBox.Show("The provided url is not valid.");
            }
            else
            {
                Settings settings = Settings.Default;
                settings.Homepage = homePage;
                settings.Save();
            }
        }

        private void cancelButton_Click(object sender, EventArgs e)
        {
            this.Dispose();
        }
    }
}
```

Listing 17: SettingsWindow.cs

A.2 LOGIC

A.2.1 *Interfaces*

Project: Interfaces

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Assessment_Two_Logic.Model;

namespace Assessment_Two_Logic.Interfaces
{
    /// <summary>
    /// Interface to allow a favourites-presenter to communicate with the view.
    /// </summary>
    public interface IFavouritesView : IView
    {

        /// <summary>
        /// Gets the favourite.
        /// </summary>
        /// <value>The favourite.</value>
        Favourite Favourite { get; }

        /// <summary>
        /// Displays the favourites.
        /// </summary>
        /// <param name="favourites">The favourites.</param>
        void DisplayFavourites(ICollection<Favourite> favourites);
    }
}
```

Listing 18: IFavouritesView.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Assessment_Two_Logic.Model;
```

```

namespace Assessment_Two_Logic.Interfaces
{
    /// <summary>
    /// Interface to allow a favourite-presenter to communicate with the view.
    /// </summary>
    public interface IFavouriteView : IView
    {
        /// <summary>
        /// Gets or sets the favourite.
        /// </summary>
        /// <value>The favourite.</value>
        Favourite Favourite { get; set; }
        /// <summary>
        /// Gets or sets the URL.
        /// </summary>
        /// <value>The URL.</value>
        String Url { get; set; }
        /// <summary>
        /// Gets or sets the name of the fav.
        /// </summary>
        /// <value>The name of the fav.</value>
        String FavName { get; set; }
    }
}
```

Listing 19: IFavouriteView.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Assessment_Two_Logic.Model;

namespace Assessment_Two_Logic.Interfaces
{
    /// <summary>
    /// Interface to allow a history-presenter to communicate with the view.
    /// </summary>
    public interface IHistoryView : IView
    {
        /// <summary>
        /// Displays the history.
        /// </summary>
        /// <param name="history">The history.</param>
        void DisplayHistory(History history);
    }
}
```

Listing 20: IHistoryView.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing.Printing;
using System.Drawing;

namespace Assessment_Two_Logic.Interfaces
{
    /// <summary>
    /// Interface to allow a print-presenter to communicate with the view.
    /// </summary>
    public interface IPrintView
    {
        /// <summary>
        /// Gets the current font.
        /// </summary>
        /// <value>The current font.</value>
        Font CurrentFont { get; }

        /// <summary>
        /// Gets the String to be printed.
        /// </summary>
        /// <value>The string.</value>
        String Print { get; }
    }
}
```

Listing 21: IPrintView.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections.Generic;

namespace Assessment_Two_Logic.Interfaces
{
    /// <summary>
    /// Interface to allow different types of serializers to be used if necessary.
    /// </summary>
    /// <typeparam name="T"></typeparam>
    public interface ISerialiser<T>
    {
        /// <summary>
        /// Gets or sets the file path.
        /// </summary>
        /// <value>The file path.</value>
        String FilePath { get; set; }

        /// <summary>
        /// Writes the specified t.
        /// </summary>
        /// <param name="t">The t.</param>
        void Write(T t);

        /// <summary>
        /// Reads this instance.
        /// </summary>
        /// <returns></returns>
        T Read();
    }
}
```

Listing 22: ISerialiser.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Assessment_Two_Logic.Model;

namespace Assessment_Two_Logic.Interfaces
{
    /// <summary>
    /// Generic interface to enforce certain methods in all views.
    /// </summary>
    public interface IView
    {
        /// <summary>
        /// Displays the errors.
        /// </summary>
        /// <param name="errors">The errors.</param>
        void DisplayErrors(ErrorMessageCollection errors);
    }
}
```

Listing 23: IView.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Assessment_Two_Logic.Model;
using System.Net;

namespace Assessment_Two_Logic.Interfaces
{
    /// <summary>
    /// Interface to allow a page-presenter to communicate with the view.
    /// </summary>
    public interface IWebpageView : IView
    {
        /// <summary>
        /// Gets or sets the URL.
        /// </summary>
        /// <value>The URL.</value>
        String Url { get; set; }

        /// <summary>
        /// Gets or sets the site text.
        /// </summary>
        /// <value>The site text.</value>
        String SiteText { get; }

        /// <summary>
        /// Displays the web page.
        /// </summary>
        /// <param name="webpage">The webpage.</param>
        void DisplayWebPage(SimpleWebResponse webpage);
    }
}
```

Listing 24: IWebPageView.cs

A.2.2 *Model*

Project: Model

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Assessment_Two_Logic.Model
{
    /// <summary>
    ///
    /// </summary>
    public class ErrorMessage
    {
        private string _Message;
        private string _Source;

        /// <summary>
        /// Initializes a new instance of the <see cref="ErrorMessage"/> class.
        /// </summary>
        /// <param name="message">The message.</param>
        /// <param name="source">The source.</param>
        public ErrorMessage(string message,
            string source)
        {
            _Message = message;
            _Source = source;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="ErrorMessage"/> class.
        /// </summary>
        /// <param name="message">The message.</param>
        public ErrorMessage(string message)
        {
            _Message = message;
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="ErrorMessage"/> class.
        /// </summary>
        public ErrorMessage()
        {
        }

        /// <summary>
        /// Gets or sets the message.
        /// </summary>
        /// <value>The message.</value>
        public string Message
        {
            get
            {
                return _Message;
            }

            set
            {
                _Message = value;
            }
        }

        /// <summary>
        /// Gets or sets the source.
        /// </summary>
        /// <value>The source.</value>
        public string Source
        {
            get
            {
                return _Source;
            }

            set
            {
                _Source = value;
            }
        }

        /// <summary>
        /// Returns a <see cref="System.String"/> that represents this instance.
        /// </summary>
        /// <returns>
        /// A <see cref="System.String"/> that represents this instance.
        /// </returns>
        public override string ToString()
        {
            return _Message;
        }
    }
}
```

Listing 25: ErrorMessage.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Assessment_Two_Logic.Model
{
    public class ErrorMessageCollection : List<ErrorMessage>
    {
        /// <summary>
        /// Returns a <see cref="System.String"/> that represents this instance.
        /// </summary>
        /// <returns>
        /// A <see cref="System.String"/> that represents this instance.
        /// </returns>
        public override string ToString()
        {
            StringBuilder sb = new StringBuilder();

            foreach (ErrorMessage item in this)
            {
                if (sb.Length > 0)
                {
                    sb.Append(Environment.NewLine);
                }

                sb.Append(item.ToString());
            }

            return sb.ToString();
        }
    }
}
```

Listing 26: ErrorMessageCollection.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Assessment_Two_Logic.Model
{
    /// <summary>
    /// Stores a favourite with a display name.
    /// </summary>
    public class Favourite
    {
        private String _Url;

        /// <summary>
        /// Gets or sets the URL.
        /// </summary>
        /// <value>The URL.</value>
        public String Url
        {
            get
            {
                return _Url;
            }
            set
            {
                if (PageHandler.IsValidUrl(value))
                {
                    _Url = value;
                }
                else
                {
                    throw new ArgumentException("The favourite-url does not match a
                         valid format.");
                }
            }
        }

        private String _Name;
        /// <summary>
        /// Gets or sets the name.
        /// </summary>
        /// <value>The name.</value>
        public String Name
        {
            get
            {
                return _Name;
            }
            set
            {
                if (!String.IsNullOrEmpty(value))
                {
                    _Name = value;
                }
                else
                {
                    throw new ArgumentException("The favourite-name is not in a valid
                        format");
                }
            }
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Favourite"/> class.
        /// </summary>
        public Favourite()
        { }

        /// <summary>
        /// Initializes a new instance of the <see cref="Favourite"/> class.
        /// </summary>
        /// <param name="url">The URL.</param>
        /// <param name="name">The name.</param>
        public Favourite(String url, String name)
        {
            this.Name = name;
            this.Url = url;
        }

        /// <summary>
        /// Edits the favourite.
        /// </summary>
        /// <param name="newUrl">The new URL.</param>
        /// <param name="newName">The new name.</param>
        public void EditFavourite(String newUrl, String newName)
        {
            this.Name = newName;
            this.Url = newUrl;
        }

        /// <summary>
        /// Returns a <see cref="System.String"/> that represents this instance.
        /// </summary>
        /// <returns>
        /// A <see cref="System.String"/> that represents this instance.
        /// </returns>
        public override string ToString()
        {
            return this.Name + " — " + this.Url;
        }
    }
}
```

Listing 27: Favourite.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NLog;
using Assessment_Two_Logic.Interfaces;

namespace Assessment_Two_Logic.Model
{
    public class FavouriteHandler
    {
        private static Logger logger = LogManager.GetCurrentClassLogger();

        public delegate void ChangeHandler(object subject);

        public event ChangeHandler ChangeEvent;

        /// <summary>
        /// Object used to locking to prevent deadlocks.
        /// </summary>
        private static Object lockObject = new Object();

        private static FavouriteHandler _Instance;

        /// <summary>
        /// Gets the instance.
        /// </summary>
        /// <value>The instance.</value>
        public static FavouriteHandler Instance
        {
            get
            {
                if (_Instance == null)
                {
                    lock (lockObject)
                    {
                        if (_Instance == null)
                        {
                            _Instance = new FavouriteHandler();
                        }
                    }
                }
                return _Instance;
            }
        }

        private ISerialiser<List<Favourite>> _Serializer;
        private List<Favourite> _Favourites;

        /// <summary>
        /// Gets the history.
        /// </summary>
        /// <value>The history.</value>
        public List<Favourite> Favourites
        {
            get
            {
                return _Favourites;
            }
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="FavouriteHandler"/> class.
        /// </summary>
        private FavouriteHandler()
        {
            this._Serializer = new XmlSerialiser<List<Favourite>>();
            this._Favourites = new List<Favourite>();
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="FavouriteHandler"/> class.
        /// </summary>
        /// <param name="filePath">The file path.</param>
        private FavouriteHandler(String filePath)
        {
            this._Serializer = new XmlSerialiser<List<Favourite>>(filePath);
            this._Favourites = new List<Favourite>();
        }

        /// <summary>
        /// Adds the entry.
        /// </summary>
        /// <param name="url">The URL.</param>
        public void AddEntry(String name, String url)
        {
            Favourite favourite = new Favourite(url, name);
            this.Favourites.Add(favourite);
            this.Notify();
        }

```

```
        /// <summary>
        /// Edits the favourite.
        /// </summary>
        /// <param name="fav">The fav.</param>
        /// <param name="newName">The new name.</param>
        /// <param name="newUrl">The new URL.</param>
        public void EditFavourite(Favourite fav, String newName, String newUrl)
        {
            if (fav != null)
            {
                Boolean favouritePresent = false;
                foreach (Favourite favourite in this.Favourites)
                {
                    if (favourite.Equals(fav))
                    {
                        favourite.Name = newName;
                        favourite.Url = newUrl;
                        favouritePresent = true;
                    }
                }
                if (!favouritePresent)
                {
                    throw new ArgumentException("The favourite that should be changed
                        was not found.");
                }
                this.Notify();
            }
            else
            {
                throw new ArgumentNullException("The provided favourite-reference was
                    null");
            }
        }

        /// <summary>
        /// Deletes the favourite.
        /// </summary>
        /// <param name="fav">The fav.</param>
        public void DeleteFavourite(Favourite fav)
        {
            if (fav != null)
            {
                this.Favourites.Remove(fav);
                this.Notify();
            }
            else
            {
                throw new ArgumentNullException("The provided favourite-reference was
                    null");
            }
        }

        /// <summary>
        /// Sets the file path.
        /// </summary>
        /// <param name="path">The path.</param>
        public void SetFilePath(String path)
        {
            this._Serializer.FilePath = path;
        }

        /// <summary>
        /// Loads the history.
        /// </summary>
        public void LoadFavourite()
        {
            try
            {
                List<Favourite> favourites = this._Serializer.Read();
                this._Favourites = favourites;
                this.Notify();
            }
            catch (Exception e)
            {
                logger.Error(e);
            }
        }

        /// <summary>
        /// Loads the history.
        /// </summary>
        /// <param name="filePath">The file path.</param>
        public void LoadFavourite(String filePath)
        {
            String oldPath = this._Serializer.FilePath;
            this._Serializer.FilePath = filePath;
            List<Favourite> favourites = this._Serializer.Read();
            this._Favourites = favourites;
            this._Serializer.FilePath = oldPath;
            this.Notify();
        }

        /// <summary>
        /// Saves the history.
        /// </summary>
        public void SaveFavourite()
        {
            this._Serializer.Write(this.Favourites);
        }

        /// <summary>
        /// Saves the history.
        /// </summary>
        /// <param name="path">The path.</param>
        public void SaveFavourite(String path)
        {
            String oldPath = this._Serializer.FilePath;
            this._Serializer.FilePath = path;
            this.SaveFavourite();
            this._Serializer.FilePath = oldPath;
        }

        private void Notify()
        {
            if (ChangeEvent != null)
            {
                ChangeEvent(this);
            }
        }
    }
}
```

Listing 28: FavouriteHandler.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Assessment_Two_Logic.Model
{

    /// <summary>
    /// Stores the history of visited webpages with their associated date.
    /// </summary>
    public class History
    {
        private SerializableDictionary<DateTime, String> _VisitList;

        /// <summary>
        /// Gets the visit list.
        /// </summary>
        /// <value>The visit list.</value>
        public SerializableDictionary<DateTime, String> VisitList
        {
            get
            {
                return _VisitList;
            }
            set
            {
                this._VisitList = value;
            }
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="History"/> class.
        /// </summary>
        public History()
        {
            this._VisitList = new SerializableDictionary<DateTime, string>();
        }

        /// <summary>
        /// Adds the item.
        /// The associated time will be the current time on the executing machine.
        /// </summary>
        /// <param name="url">The URL.</param>
        public void AddItem(String url)
        {
            DateTime time = DateTime.UtcNow;
            this.AddItem(time, url);
        }

        /// <summary>
        /// Adds the item to the history.
        /// </summary>
        /// <param name="time">The time of the visit.</param>
        /// <param name="url">The URL.</param>
        public void AddItem(DateTime time, String url)
        {
            if (PageHandler.IsValidUrl(url))
            {
                this._VisitList.Add(time, url);
            }
            else
            {
                throw new ArgumentException("The provided url does not resemble a
                    valid format.");
            }
        }

        /// <summary>
        /// Clears the history.
        /// </summary>
        public void ClearHistory()
        {
            this._VisitList.Clear();
        }

        /// <summary>
        /// Determines whether this instance is empty.
        /// </summary>
        /// <returns>
        ///     <c>true</c> if this instance is empty; otherwise, <c>false</c>.
        /// </returns>
        internal bool IsEmpty()
        {
            return this._VisitList.Count == 0;
        }

        /// <summary>
        /// Adds the item.
        /// </summary>
        /// <param name="item">The item.</param>
        internal void AddItem(KeyValuePair<DateTime, string> item)
        {
            this.VisitList.Add(item.Key, item.Value);
        }
    }
}
```

Listing 29: History.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NLog;
using Assessment_Two_Logic.Interfaces;

namespace Assessment_Two_Logic.Model
{
    public class HistoryHandler
    {
        private static Logger logger = LogManager.GetCurrentClassLogger();

        public delegate void ChangeHandler(object subject);

        public event ChangeHandler ChangeEvent;

        /// <summary>
        /// Object used to locking to prevent deadlocks.
        /// </summary>
        private static Object lockObject = new Object();

        private static HistoryHandler _Instance;

        /// <summary>
        /// Gets the instance.
        /// </summary>
        /// <value>The instance.</value>
        public static HistoryHandler Instance
        {
            get
            {
                if (_Instance == null)
                {
                    lock (lockObject)
                    {
                        if (_Instance == null)
                        {
                            _Instance = new HistoryHandler();
                        }
                    }
                }
                return _Instance;
            }
        }

        private ISerialiser<History> _Serializer;
        private History _History;

        /// <summary>
        /// Gets the history.
        /// </summary>
        /// <value>The history.</value>
        public History History
        {
            get
            {
                return _History;
            }
        }

        private HistoryHandler()
        {
            this._Serializer = new XmlSerialiser<History>();
            this._History = new History();
        }

        private HistoryHandler(String filePath)
        {
            this._Serializer = new XmlSerialiser<History>(filePath);
            this._History = new History();
        }

        /// <summary>
        /// Adds the entry.
        /// </summary>
        /// <param name="url">The URL.</param>
        public void AddEntry(String url)
        {
            this.History.AddItem(url);
            this.Notify();
        }

        /// <summary>
        /// Adds the entry.
        /// </summary>
        /// <param name="time">The time.</param>
        /// <param name="url">The URL.</param>
        public void AddEntry(DateTime time, String url)
        {
            this.History.AddItem(time, url);
            this.Notify();
        }

        /// <summary>
        /// Sets the file path.
        /// </summary>
        /// <param name="path">The path.</param>
        public void SetFilePath(String path)
        {
            this._Serializer.FilePath = path;
        }

        /// <summary>
        /// Loads the history.
        /// </summary>
        public void LoadHistory()
        {
            try
            {
                History history = this._Serializer.Read();
                this._History.ClearHistory();
                foreach (var item in history.VisitList)
                {
                    this._History.AddItem(item);
                }
                this.Notify();
            }
            catch (Exception e)
            {
                logger.Error(e);
            }
        }

        /// <summary>
        /// Loads the history.
        /// </summary>
        /// <param name="filePath">The file path.</param>
        public void LoadHistory(String filePath)
        {
            String oldPath = this._Serializer.FilePath;
            this._Serializer.FilePath = filePath;
            History history = this._Serializer.Read();
            this._History.ClearHistory();
            foreach (var item in history.VisitList)
            {
                this._History.AddItem(item);
            }
            this._Serializer.FilePath = oldPath;
            this.Notify();
        }

        /// <summary>
        /// Saves the history.
        /// </summary>
        public void SaveHistory()
        {
            this._Serializer.Write(this.History);
        }

        /// <summary>
        /// Saves the history.
        /// </summary>
        /// <param name="path">The path.</param>
        public void SaveHistory(String path)
        {
            String oldPath = this._Serializer.FilePath;
            this._Serializer.FilePath = path;
            this.SaveHistory();
            this._Serializer.FilePath = oldPath;
        }

        /// <summary>
        /// Notifies the observers.
        /// </summary>
        private void Notify()
        {
            if (ChangeEvent != null)
            {
                ChangeEvent(this);
            }
        }

        /// <summary>
        /// Clears the history.
        /// </summary>
        internal void ClearHistory()
        {
            this.History.ClearHistory();
            this.Notify();
        }
    }
}
```

Listing 30: HistoryHandler.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Assessment_Two_Logic.Model
{
    /// <summary>
    /// Exception to notify the caller that no filepath was set.
    /// </summary>
    public class NoFilePathSetException : Exception
    {
        /// <summary>
        /// Initializes a new instance of the <see cref="NoFilePathSetException"/>
            class.
        /// </summary>
        /// <param name="message">The message.</param>
        public NoFilePathSetException(String message) : base(message)
        { }
    }
}
```

Listing 31: NoFilePathSetException.cs

```
using System;
using System.Net;
using System.Threading;
using Assessment_Two_Logic.Interfaces;
using System.IO;
using NLog;
using System.Text.RegularExpressions;

namespace Assessment_Two_Logic.Model
{
    /// <summary>
    /// Allows the fetching of urls in a thread.
    /// Will notify the caller via "ThreadFinished" callback method.
    /// </summary>
    public class PageHandler
    {
        private const string HTTP_REGEXP = @"^http\:\/\/[\w\-\.]+\.[a-zA-Z]{2,3}(\/\S
            *)?$";
        private static Logger logger = LogManager.GetCurrentClassLogger();

        /// <summary>
        /// Stores the url to be aquired by this handler.
        /// </summary>
        private String requestUrl;

        /// <summary>
        /// Gets or sets the request URL.
        /// </summary>
        /// <value>The request URL.</value>
        public String RequestUrl
        {
            get { return requestUrl; }
            set { requestUrl = value; }
        }

        /// <summary>
        /// Stores the WebRequest.
        /// </summary>
        private WebRequest request;

        /// <summary>
        /// Gets or sets the request.
        /// </summary>
        /// <value>The request.</value>
        public WebRequest Request
        {
            get { return request; }
            set { request = value; }
        }

        private WebResponse response;

        /// <summary>
        /// Gets or sets the response.
        /// </summary>
        /// <value>The response.</value>
        public WebResponse Response
        {
            get { return response; }
            set { response = value; }
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="PageHandler"/> class.
        /// </summary>
        public PageHandler()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="PageHandler"/> class.
        /// </summary>
        /// <param name="url">The URL.</param>
        public PageHandler(String url)
        {
            this.RequestUrl = url;
        }

        /// <summary>
        /// Fetches the URL.
        /// </summary>
        /// <returns>The webresponse object.</returns>
        public SimpleWebResponse FetchUrl()
        {
            if (IsValidUrl(this.RequestUrl))
            {
                try
                {
                    String htmlCode = "";
                    try
                    {
                        this.Request = WebRequest.Create(this.RequestUrl);
                        this.Response = this.Request.GetResponse();
                    }
                    catch (WebException e)
                    {
                        logger.Error("WebException ({0}) occured when fetching the url
                            : {1}", e.Message, this.RequestUrl);
                        this.Response = e.Response;
                    }
                    StreamReader sr = new StreamReader(this.Response.GetResponseStream
                        ());
                    htmlCode = sr.ReadToEnd();
                    SimpleWebResponse swr = new SimpleWebResponse(this.RequestUrl,
                        this.RequestUrl, htmlCode);

                    return swr;
                }
                catch (Exception e)
                {
                    logger.Error("Exception ({1}) occured, when creating request for
                        url: {0}", this.RequestUrl, e.Message);
                    throw new ArgumentException(String.Format("The Url: {0} could not
                        be fetched.", this.RequestUrl));
                }
            }
            else
            {
                throw new ArgumentException(String.Format("The provided url did not
                    match the specified format for html-urls: {0}", this.RequestUrl));
            }
        }

        /// <summary>
        /// Determines whether [the specified URL] is in a valid format.
        /// </summary>
        /// <param name="url">The URL.</param>
        /// <returns>
        ///     <c>true</c> if [is valid URL] [the specified URL]; otherwise, <c>false
            </c>.
        /// </returns>
        public static bool IsValidUrl(String url)
        {
            bool isValidUrl = false;
            Regex regexp = new Regex(HTTP_REGEXP);
            isValidUrl = regexp.IsMatch(url);
            return isValidUrl;
        }
    }
}
```

Listing 32: PageHandler.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Text;
using System.Xml.Serialization;

namespace Assessment_Two_Logic.Model
{
    /// <summary>
    /// Class to represent a serializable dictionary.
    /// </summary>
    /// <typeparam name="TKey">The type of the key.</typeparam>
    /// <typeparam name="TValue">The type of the value.</typeparam>
    [XmlRoot("dictionary")]
    public class SerializableDictionary<TKey, TValue>
        : Dictionary<TKey, TValue>, IXmlSerializable
    {
        #region IXmlSerializable Members
        public System.Xml.Schema.XmlSchema GetSchema()
        {
            return null;
        }

        public void ReadXml(System.Xml.XmlReader reader)
        {
            XmlSerializer keySerializer = new XmlSerializer(typeof(TKey));
            XmlSerializer valueSerializer = new XmlSerializer(typeof(TValue));

            bool wasEmpty = reader.IsEmptyElement;
            reader.Read();

            if (wasEmpty)
                return;

            while (reader.NodeType != System.Xml.XmlNodeType.EndElement)
            {
                reader.ReadStartElement("item");

                reader.ReadStartElement("key");
                TKey key = (TKey)keySerializer.Deserialize(reader);
                reader.ReadEndElement();

                reader.ReadStartElement("value");
                TValue value = (TValue)valueSerializer.Deserialize(reader);
                reader.ReadEndElement();

                this.Add(key, value);

                reader.ReadEndElement();
                reader.MoveToContent();
            }
            reader.ReadEndElement();
        }

        public void WriteXml(System.Xml.XmlWriter writer)
        {
            XmlSerializer keySerializer = new XmlSerializer(typeof(TKey));
            XmlSerializer valueSerializer = new XmlSerializer(typeof(TValue));

            foreach (TKey key in this.Keys)
            {
                writer.WriteStartElement("item");

                writer.WriteStartElement("key");
                keySerializer.Serialize(writer, key);
                writer.WriteEndElement();

                writer.WriteStartElement("value");
                TValue value = this[key];
                valueSerializer.Serialize(writer, value);
                writer.WriteEndElement();

                writer.WriteEndElement();
            }
        }
        #endregion
    }
}
```

Listing 33: SerializableDictionary.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Assessment_Two_Logic.Model
{
    /// <summary>
    /// A simplified webresponse reduced to the attributes needed to display a webpage
         .
    /// </summary>
    public class SimpleWebResponse
    {
        public String Url { get; set; }
        public String Title { get; set; }
        public String Html { get; set; }

        /// <summary>
        /// Initializes a new instance of the <see cref="SimpleWebResponse"/> class.
        /// </summary>
        /// <param name="url">The URL.</param>
        /// <param name="title">The title.</param>
        /// <param name="htmlCode">The HTML code.</param>
        public SimpleWebResponse(string url, string title, string htmlCode)
        {
            this.Url = url;
            this.Title = title;
            this.Html = htmlCode;
        }
    }
}
```

Listing 34: SimpleWebResponse.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Assessment_Two_Logic.Interfaces;
using System.IO;
using System.Xml.Serialization;

namespace Assessment_Two_Logic.Model
{
    /// <summary>
    /// An XmlSerialiser.
    /// </summary>
    /// <typeparam name="T">The type that can be serialised via this Serialiser.</
         typeparam>
    public class XmlSerialiser<T> : ISerialiser<T>
    {
        private String _FilePath;

        public string FilePath
        {
            get
            {
                return this._FilePath;
            }
            set
            {
                this._FilePath = value;
            }
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="XmlSerialiser&lt;T&gt;"/> class.
        /// </summary>
        public XmlSerialiser()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="XmlSerialiser&lt;T&gt;"/> class.
        /// </summary>
        /// <param name="filePath">The file path.</param>
        public XmlSerialiser(string filePath)
        {
            this.FilePath = filePath;
        }

        /// <summary>
        /// Writes the specified t to the filepath.
        /// </summary>
        /// <param name="t">The t.</param>
        public void Write(T t)
        {
            if (!String.IsNullOrEmpty(this.FilePath))
            {
                XmlSerializer serializer = new XmlSerializer(typeof(T));
                TextWriter writer = new StreamWriter(this._FilePath);
                serializer.Serialize(writer, t);
                writer.Close();
            }
            else
            {
                throw new NoFilePathSetException("No file path set to save the elements to.");
            }
        }

        /// <summary>
        /// Reads a specified t from the filepath.
        /// </summary>
        /// <returns>The t read.</returns>
        public T Read()
        {
            if (!String.IsNullOrEmpty(this.FilePath))
            {
                T t;
                XmlSerializer serializer = new XmlSerializer(typeof(T));
                TextReader reader = new StreamReader(this._FilePath);
                t = (T)serializer.Deserialize(reader);
                reader.Close();
                return t;
            }
            else
            {
                throw new NoFilePathSetException("No file path set to read the
                    elements from.");
            }
        }

    }
}
```

Listing 35: XmlSerialiser.cs

A.2.3 *Presenter*

Project: Presenter

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Assessment_Two_Logic.Interfaces;
using Assessment_Two_Logic.Model;

namespace Assessment_Two_Logic.Presenter
{
    /// <summary>
    /// Presenter to show, add and edit a single favourite object.
    /// </summary>
    public class FavouritePresenter
    {
        /// <summary>
        /// Reference to the view.
        /// </summary>
        private IFavouriteView _FavouriteView;

        /// <summary>
        /// Reference to the favourite handler.
        /// </summary>
        private FavouriteHandler _FavouriteHandler;

        /// <summary>
        /// Initializes a new instance of the <see cref="FavouritePresenter"/> class.
        /// </summary>
        /// <param name="view">The view.</param>
        public FavouritePresenter(IFavouriteView view)
        {
            this._FavouriteView = view;
            this._FavouriteHandler = FavouriteHandler.Instance;

        }

        /// <summary>
        /// Adds the favourite.
        /// </summary>
        public void AddFavourite()
        {
            try
            {
                String favName = this._FavouriteView.FavName;
                String favUrl = this._FavouriteView.Url;
                this._FavouriteHandler.AddEntry(favName, favUrl);
            }
            catch (ArgumentException e)
            {
                this.DisplayError(e.Message);
            }
        }

        /// <summary>
        /// Edits the favourite.
        /// </summary>
        public void EditFavourite()
        {
            try
            {
                Favourite fav = this._FavouriteView.Favourite;
                String favName = this._FavouriteView.FavName;
                String favUrl = this._FavouriteView.Url;
                this._FavouriteHandler.EditFavourite(fav, favName, favUrl);
            }
            catch (ArgumentException e)
            {
                this.DisplayError(e.Message);
            }
        }

        /// <summary>
        /// Displays the error.
        /// </summary>
        /// <param name="p">The String to create an error from.</param>
        private void DisplayError(string p)
        {
            ErrorMessage em = new ErrorMessage(p);
            ErrorMessageCollection emc = new ErrorMessageCollection();
            emc.Add(em);
            this._FavouriteView.DisplayErrors(emc);
        }
    }
}
```

Listing 36: FavouritePresenter.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using Assessment_Two_Logic.Interfaces;
using Assessment_Two_Logic.Model;

namespace Assessment_Two_Logic.Presenter
{
    /// <summary>
    /// Presenter to show and delete a list of favourites.
    /// </summary>
    public class FavouritesPresenter
    {
        private FavouriteHandler _FavouriteHandler;

        private IFavouritesView _FavouritesView;

        /// <summary>
        /// Initializes a new instance of the <see cref="FavouritesPresenter"/> class.
        /// </summary>
        /// <param name="view">The view.</param>
        public FavouritesPresenter(IFavouritesView view)
        {
            this._FavouritesView = view;
            this._FavouriteHandler = FavouriteHandler.Instance;
            this._FavouriteHandler.ChangeEvent += new FavouriteHandler.ChangeHandler(
                this.Update);
            SetUpHandler();
        }

        private void SetUpHandler()
        {
            String appFolder = Environment.GetFolderPath(Environment.SpecialFolder.
                ApplicationData);
            String history = "Favourites.xml";
            this._FavouriteHandler.SetFilePath(Path.Combine(appFolder, history));
            this._FavouriteHandler.LoadFavourite();
        }

        /// <summary>
        /// Updates the specified subject.
        /// Realizes the observer pattern.
        /// </summary>
        /// <param name="subject">The subject.</param>
        public void Update(object subject)
        {
            if (subject is FavouriteHandler)
            {
                FavouriteHandler favHandler = subject as FavouriteHandler;
                this._FavouritesView.DisplayFavourites(favHandler.Favourites);
            }
        }

        /// <summary>
        /// Deletes the favourite.
        /// </summary>
        public void DeleteFavourite()
        {
            try
            {
                Favourite fav = this._FavouritesView.Favourite;
                this._FavouriteHandler.DeleteFavourite(fav);
            }
            catch (ArgumentException e)
            {
                this.DisplayError(e.Message);
            }
        }

        /// <summary>
        /// Sets the favourites path.
        /// </summary>
        /// <param name="path">The path.</param>
        public void SetFavouritesPath(String path)
        {
            this._FavouriteHandler.SetFilePath(path);
        }

        /// <summary>
        /// Saves the favourites.
        /// </summary>
        public void SaveFavourites()
        {
            try
            {
                this._FavouriteHandler.SaveFavourite();
            }
            catch (NoFilePathSetException e)
            {
                this.DisplayError(e.Message);
            }
        }

        /// <summary>
        /// Displays the error.
        /// </summary>
        /// <param name="p">The String to create an error from.</param>
        private void DisplayError(string p)
        {
            ErrorMessage em = new ErrorMessage(p);
            ErrorMessageCollection emc = new ErrorMessageCollection();
            emc.Add(em);
            this._FavouritesView.DisplayErrors(emc);
        }
    }
}
```

Listing 37: FavouritesPresenter.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using Assessment_Two_Logic.Model;
using Assessment_Two_Logic.Interfaces;

namespace Assessment_Two_Logic.Presenter
{
    /// <summary>
    /// Presenter to show the history.
    /// </summary>
    public class HistoryPresenter
    {
        /// <summary>
        /// Reference to the history handler.
        /// </summary>
        public HistoryHandler _HistoryHandler;

        /// <summary>
        /// Reference to the accompanying view.
        /// </summary>
        public IHistoryView _HistoryView;

        /// <summary>
        /// Initializes a new instance of the <see cref="HistoryPresenter"/> class.
        /// </summary>
        /// <param name="view">The view.</param>
        public HistoryPresenter(IHistoryView view)
        {
            this._HistoryView = view;
            this._HistoryHandler = HistoryHandler.Instance;
            this._HistoryHandler.ChangeEvent += new HistoryHandler.ChangeHandler(this.
                Update);

            SetUpHandler();
        }

        private void SetUpHandler()
        {
            String appFolder = Environment.GetFolderPath(Environment.SpecialFolder.
                ApplicationData);
            String history = "History.xml";
            this._HistoryHandler.SetFilePath(Path.Combine(appFolder, history));
            this._HistoryHandler.LoadHistory();
        }

        /// <summary>
        /// Updates the specified subject.
        /// Realizes the observer pattern.
        /// </summary>
        /// <param name="subject">The subject.</param>
        public void Update(object subject)
        {
            if (subject is HistoryHandler)
            {
                HistoryHandler histhandler = subject as HistoryHandler;
                this._HistoryView.DisplayHistory(histhandler.History);
            }
        }

        /// <summary>
        /// Saves the history.
        /// </summary>
        public void SaveHistory()
        {
            try
            {
                this._HistoryHandler.SaveHistory();
            }
            catch (NoFilePathSetException e)
            {
                this.DisplayError(e.Message);
            }
        }

        /// <summary>
        /// Clears the history.
        /// </summary>
        public void ClearHistory()
        {
            this._HistoryHandler.ClearHistory();
        }

        /// <summary>
        /// Displays the error.
        /// </summary>
        /// <param name="p">The String to create an error from.</param>
        private void DisplayError(string p)
        {
            ErrorMessage em = new ErrorMessage(p);
            ErrorMessageCollection emc = new ErrorMessageCollection();
            emc.Add(em);
            this._HistoryView.DisplayErrors(emc);
        }
    }
}
```

Listing 38: HistoryPresenter.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Assessment_Two_Logic.Interfaces;
using Assessment_Two_Logic.Model;
using System.Net;
using NLog;

namespace Assessment_Two_Logic.Presenter
{
    /// <summary>
    /// Used to request webpages in an asynchronous fashion.
    /// </summary>
    public class PagePresenter
    {
        private static Logger logger = LogManager.GetCurrentClassLogger();

        /// <summary>
        /// References the associated view.
        /// </summary>
        private IWebpageView _WebPageView;
        /// <summary>
        /// References the history handler.
        /// </summary>
        private HistoryHandler _HistoryHandler;

        /// <summary>
        /// Initializes a new instance of the <see cref="PagePresenter"/> class.
        /// </summary>
        /// <param name="view">The view.</param>
        public PagePresenter(IWebpageView view)
        {
            this._WebPageView = view;
            this._HistoryHandler = HistoryHandler.Instance;
        }

        /// <summary>
        /// Requests the webpage.
        /// To keep the UI-Thread responsive the request will be started in a thread.
        /// </summary>
        public void RequestWebpage()
        {
            String requestUrl = this._WebPageView.Url;
            bool validUrl = PageHandler.IsValidUrl(requestUrl);
            if (validUrl)
            {
                PageHandler pageHandler = new PageHandler(requestUrl);
                this._HistoryHandler.AddEntry(requestUrl);
                Func<SimpleWebResponse> method = pageHandler.FetchUrl;
                method.BeginInvoke(Done, method);
            }
            else
            {
                this.DisplayError("The provided url is not valid.");
            }
        }

        /// <summary>
        /// Callback when a thread is finished with processing.
        /// </summary>
        /// <param name="result">The result of the threaded call.</param>
        public void Done(IAsyncResult result)
        {
            lock (this)
            {
                try
                {
                    var target = (Func<SimpleWebResponse>)result.AsyncState;
                    SimpleWebResponse response = target.EndInvoke(result);
                    this._WebPageView.DisplayWebPage(response);
                }
                catch (Exception e)
                {
                    logger.Error(String.Format("An error occured when fetching a url.
                        {0}", e.Message));
                    this.DisplayError(e.Message);
                }

            }
        }
        /// <summary>
        /// Displays the error.
        /// </summary>
        /// <param name="p">The String to create an error from.</param>
        private void DisplayError(string p)
        {
            ErrorMessage em = new ErrorMessage(p);
            ErrorMessageCollection emc = new ErrorMessageCollection();
            emc.Add(em);
            this._WebPageView.DisplayErrors(emc);
        }
    }
}
```

Listing 39: PagePresenter.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Assessment_Two_Logic.Interfaces;
using System.Drawing.Printing;
using System.Drawing;


namespace Assessment_Two_Logic.Presenter
{
    /// <summary>
    ///
    /// </summary>
    public class PrintPresenter
    {
        private IPrintView _PrintView;

        /// <summary>
        /// Gets or sets the string to be printed.
        /// </summary>
        /// <value>The string to be printed.</value>
        private String PrintString { get; set; }

        /// <summary>
        /// Initializes a new instance of the <see cref="PrintPresenter"/> class.
        /// </summary>
        /// <param name="view">The view.</param>
        public PrintPresenter(IPrintView view)
        {
            this._PrintView = view;
        }

        /// <summary>
        /// Prints the printstring.
        /// Method from Microsoft.
        /// </summary>
        /// <param name="e">The <see cref="System.Drawing.Printing.PrintPageEventArgs
            "/> instance containing the event data.</param>
        public void Print(System.Drawing.Printing.PrintPageEventArgs e)
        {

            Font font = _PrintView.CurrentFont;
            int charactersOnPage = 0;
            int linesPerPage = 0;

            // Sets the value of charactersOnPage to the number of characters
            // of PrintString that will fit within the bounds of the page.
            e.Graphics.MeasureString(PrintString, font,
                e.MarginBounds.Size, StringFormat.GenericTypographic,
                out charactersOnPage, out linesPerPage);

            // Draws the string within the bounds of the page
            e.Graphics.DrawString(PrintString, font, Brushes.Black,
                e.MarginBounds, StringFormat.GenericTypographic);

            // Remove the portion of the string that has been printed.
            PrintString = PrintString.Substring(charactersOnPage);

            // Check to see if more pages are to be printed.
            e.HasMorePages = (PrintString.Length > 0);
        }

        public void SetPrintString()
        {
            this.PrintString = this._PrintView.Print;
        }
    }
}
```

Listing 40: PrintPresenter.cs

A.3 TESTS

Project: Tests

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Assessment_Two_Logic.Model;
using NUnit.Framework;

```

```
namespace Assessment_Two_Specs
{
    class FavouriteHandlerTest
    {
        private FavouriteHandler handler;

        [SetUp]
        public void SetUp()
        {
            this.handler = FavouriteHandler.Instance;
        }

        [Test]
        [ExpectedException(typeof(ArgumentNullException))]
        public void TestEditFavouriteNull()
        {
            this.handler.EditFavourite(null, "12", "12");
        }

        [Test]
        [ExpectedException(typeof(ArgumentException))]
        public void TestEditNonPresentFavourite()
        {
            Favourite fav = new Favourite();
            this.handler.EditFavourite(fav, "12", "12");
        }

        [Test]
        public void TestDeleteNonPresentFavourite()
        {
            Favourite fav = new Favourite();
            int length = this.handler.Favourites.Count;
            this.handler.DeleteFavourite(fav);
            int lengthAfterRemoval = this.handler.Favourites.Count;
            Assert.AreEqual(length, lengthAfterRemoval);
        }

        [Test]
        [ExpectedException(typeof(ArgumentNullException))]
        public void TestDeleteNull()
        {
            this.handler.DeleteFavourite(null);
        }

        [Test]
        public void TestAddAndRemoveFavourite()
        {
            this.handler.AddEntry("test", "http://www.test.de/");
            Favourite fav = this.handler.Favourites.ElementAt(0);
            this.handler.DeleteFavourite(fav);
            Assert.AreEqual(0, this.handler.Favourites.Count);
        }

        [Test]
        public void TestAddFavourite()
        {
            this.handler.AddEntry("test", "http://www.test.de/");
            Assert.AreEqual(1, this.handler.Favourites.Count);
        }

        [Test]
        public void TestEditFavourite()
        {
            this.handler.AddEntry("test", "http://www.test.de");
            Favourite fav = this.handler.Favourites.ElementAt(0);
            this.handler.EditFavourite(fav, "New", "http://www.test2.de/");
            Assert.AreEqual(fav.Url, "http://www.test2.de/");
            Assert.AreEqual(fav.Name, "New");
        }
    }
}
```

Listing 41: FavouriteHandlerTest.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Assessment_Two_Logic.Model;
using NUnit.Framework;

namespace Assessment_Two_Specs
{
    [TestFixture]
    public class FavouriteTest
    {
        [Test]
        public void TestFavouriteValidUrl()
        {
            Favourite fav = new Favourite("http://www.google.de/", "Test");
            Assert.AreEqual("Test", fav.Name);
            Assert.AreEqual("http://www.google.de/", fav.Url);
        }

        [Test]
        [ExpectedException(typeof(ArgumentException))]
        public void TestFavouriteInvalidUrl()
        {
            Favourite fav = new Favourite("http://www.google.d/", "Test");
        }

        [Test]
        [ExpectedException(typeof(ArgumentException))]
        public void TestFavouriteNoName()
        {
            Favourite fav = new Favourite("http://www.google.de/", null);
        }

        [Test]
        [ExpectedException(typeof(ArgumentException))]
        public void TestFavouriteEmtpyName()
        {
            Favourite fav = new Favourite("http://www.google.de/", "");
        }
    }
}
```

Listing 42: FavouriteTest.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NUnit.Framework;
using Assessment_Two_Logic.Model;

namespace Assessment_Two_Specs
{
    [TestFixture]
    class HistoryTest
    {
        [Test]
        public void TestHistoryValidValues()
        {
            History hist = new History();
            hist.AddItem("http://www.google.de/");
            Assert.AreEqual(1, hist.VisitList.Count);
            Assert.AreEqual("http://www.google.de/", hist.VisitList.ElementAt(0).Value
                );
        }

        [Test]
        [ExpectedException(typeof(ArgumentException))]
        public void TestHistoryInvalidUrl()
        {
            History hist = new History();
            hist.AddItem("http://www.google.d/");
        }
    }
}
```

Listing 43: HistoryTest.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Assessment_Two_Logic.Model;
using NUnit.Framework;
using System.Net;

namespace Assessment_Two_Specs
{
    [TestFixture]
    class PageHandlerTest
    {
        private PageHandler handler;

        [SetUp]
        public void SetUp()
        {
            this.handler = new PageHandler();
        }

        [Test]
        public void Test404ErrorPage()
        {
            this.handler.RequestUrl = "http://www.google.de/err";
            this.handler.FetchUrl();
            HttpWebResponse resp = this.handler.Response as HttpWebResponse;
            Assert.AreEqual(resp.StatusCode, HttpStatusCode.NotFound);
        }

        [Test]
        public void Test200Page()
        {
            this.handler.RequestUrl = "http://www.google.de/";
            this.handler.FetchUrl();
            HttpWebResponse resp = this.handler.Response as HttpWebResponse;
            Assert.AreEqual(resp.StatusCode, HttpStatusCode.OK);
        }
    }
}
```

Listing 44: PageHandlerTest.cs

```
ï»¿using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NUnit.Framework;
using Assessment_Two_Logic.Model;
using Assessment_Two_Logic.Interfaces;
```

```

namespace Assessment_Two_Specs
{
    [TestFixture]
    class SerialiserTest
    {
        private ISerialiser<String> serialiser;

        [SetUp]
        public void SetUp()
        {
            this.serialiser = new XmlSerialiser<String>();
        }

        [Test]
        [ExpectedException(typeof(NoFilePathSetException))]
        public void TestWriteWithOutFilePath()
        {
            serialiser.Write("Test");
        }

        [Test]
        [ExpectedException(typeof(NoFilePathSetException))]
        public void TestReadWithOutFilePath()
        {
            serialiser.Read();
        }
    }
}
```

Listing 45: SerialiserTest.cs

BIBLIOGRAPHY

Balzert, Helmut (2009). *Lehrbuch der Software-Technik: Basiskonzepte und Requirements Engineering*. 3. Heidelberg: Spektrum. ISBN: 9783827417053.

Boodhoo, Jean-Paul (2006). *Design Patterns: Model View Presenter*. English. Microsoft. URL: http://msdn.microsoft.com/en-us/magazine/cc188690.aspx (visited on 19/10/2010).

Dorman, Scott (2010). *Sams Teach Yourself Visual CSharpï¿½2010 in 24 Hours: Complete Starter Kit*. Sams Publishing. ISBN: 978-0-672-33101-5.

Freeman, Eric and Elisabeth Freeman (2004). *Head First - Design Pattern*. Ed. by Mike Loukides. O'Reilly. ISBN: 0-596-00712-4.

Gamma, Erich et al. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed. Addison-Wesley Professional. ISBN: 0201633612. URL: http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1.

Noyes, Brian (2006). *Data Binding With Windows Forms 2.0 - Programming Smart Client Data Applications With .NET*. Addison Wesley Professional. ISBN: 978-0-321-26892-1.

Sommerville, Ian (2006). *Software Engineering*. 8th ed. Addison Wesley. ISBN: 9780321210265.

Stellman, Andrew and Jennifer Greene (2010). *Head First CSharp*. O'Reilly. ISBN: 978-1-449-38034-2.