INDUSTRIAL PROGRAMING COURSEWORK – SPIDERWEB

By H00333400

INTRODUCTION

This report aims to give the reader an overview into the development of the SpiderWEB software, its functionality, limitations and provide reflection on how it could be improved in the future. The software presented is a solution to a task where, without the aid of conventional libraries, one was to create a simple web browse, capable of retrieving HTML code from a user defined URL. Specifically, the library WebBrowser, was not to be used in the core HTTP communication aspect of the software.

REQUIREMENTS CHECKLIST

Requirement	Requirement met?	Comments
Send HTTP Requests	Y	User can enter and send URLS
Receiving HTTP Response	Y	Response then received, including status codes
Display	Y	HTTP Response code and Status code displayed. User can press a button to refresh display.
Home Page	Y	Home page feature included and loaded on start-up
Favourites	Y	User can set/delete favourites and give them an alias. This list is retained on when program is closed
History	Y	Sites user has visited are stored, searchable and retained after program closure.
GUI Requirements	Y	Windows Form GUI present, handles all the prior requirement and makes use of menus, buttons and shortcuts for user ease

DESIGN CONSIDERATIONS

Model, View Controller

When developing this app, several coding structures were considered. Many rely on triads, for example PAC, RMR or ADR but all have the vague pattern; split your code into rendering, interacting and Data representation(business rules) parts. Whilst there is more to an application than just interaction and presentation. I believed that the MVC or Model, View, Controller was the most useful in the context of the given problem. It allows one to outline structure ideas, prevents bad coding practices and gives greater flexibility to future developers, given its modular structure. In this project, the Model is where the objects are defined and handles the HTTP communication. The view is where all aspects of user interaction are dealt with such as action listeners, layout and data handling. The controller starts the GUI and liaises between the model and view.

HTTPRequest vs WebClient vs HttpClient

There are three methods in which one would conventionally go about HTTP communication from a C# perspective, namely, HTTPRequest, WebClient, HttpClient. HTTPRequest is the original, very flexible method of handling most aspects of the request and response objects without GUI interference. WebClient provides a straightforward but restricted way of handling HTTPs communication as it is a wrapper for HTTP request, thus a little slower. HttpClient is the newest of the methods, giving you the balance of flexibility and simplicity. It only needs to be instantiated once. HttpClient has an obvious advantage - the asynchronous and synchronous

manner in which it can run. As such, unlike HTTPRequest, it does not need event handlers unlike the method used in this application. Whilst HttpClient would have been the ideal way forwards, there was a lot more support for beginners with HTTPRequest, which ultimately governed my decision.

Data Structures

Objects were an obvious choice for storing each http communication and its properties. The 'SpiderWeb' object holds information about the URL request, Date/time of request, HTMLResponse and its corresponding status code from the server. It forms the base unit of storage for server interaction as it is used by the WebClient for the HTTP communication. The Favourites object is used to store only information gained from the user around a URL, that can then form the basis of a SpiderWeb object when wishing to navigate to it.

Lists were used over linkLists to store both SpiderWeb and Favourites objects primarily as items cannot be randomly accessed. For example, to remove a user specified favourite at a given index or navigate to user selected url from the browser history. Lists also have a larger variety of support methods available to them and can easily and cheaply add and remove items.

To retain information between sessions object instances had to be convert to physical storage. In other words, serialisation seemed like the logical was forward as it does just that. Objects were saved to XML as this file extension provides a great deal of flexibility with name tags and given that it can also be used with online databases, provides scope for further development and integration.

GUI Design

The GUI was created using the Designer feature within studio code, in tandem with manual efforts. A page could be assembled in the designer and then, using this as a structure, could then be replicated within the tabs of the browser. TabPages were used as they are an ordered way of displaying numerous scenarios that many users are comfortable with. They also provide a degree of familiarity for the user as they are a common feature in most modern browsers. They do however cause a degree of code bulking but I deemed this to be a necessary evil. By providing buttons, a StripMenu and keyboard shortcut it allows users multiple ways to navigate the browser quickly and simply in a way that is intuitive. Descriptive icons were chosen for buttons rather than text to further aid navigation.

DataGridViews provided a great deal of flexibility when displaying the object lists to the user, whilst not implemented fully in this code they could be bound to specific data sources and used to directly manipulate objects within the lists. Users can interact directly clicking on cells, with relevant action listeners and methods to support effective manipulation.

Rendering HTML

The ability to render the HTML code retrieved from a website was added using WebBrowser classes methods, to provide users a recognisable way to navigate websites.

Advanced Coding Practices

Where possible advances language techniques such as a lambda expressions were used to reduce code. For example :

```
Assert.Throws<WebException>(() => testSpiderWeb.RetrieveHTMLCode(testSpiderWeb));
```

In this case we are throwing an error and we cannot simply call the 'RetrieveHTMLCode' method and put testSpiderWeb object into it. This is because this would cause an exception, whereas we want to catch the

exception. The () passes the action into the lamda expression and the “testSpiderWeb…. (testSpiderWeb)” is the actual call.

USER GUIDE

First Page

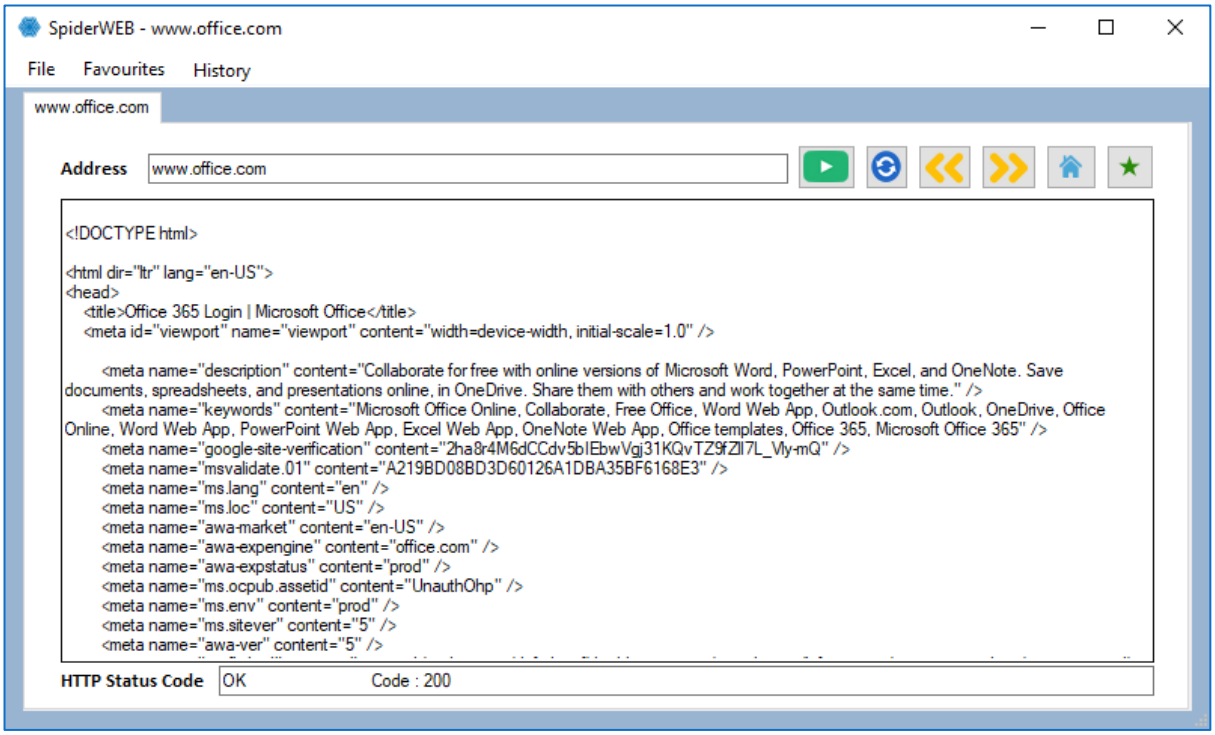
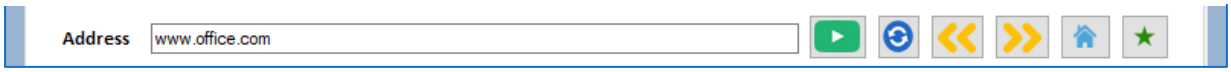


Figure 1 - firstPage on opening

When first entering the application, the user will enter the firstPage screen shown above. This is the main page, hence why it is loaded first. Its main function is to allow the user to navigate to a desired URL. This is achieved by entering a URL in the Address box and pressing the navigate button. On entry if set, the browser will load the homepage and its URL on launch. Now let’s first take a closer look at the navigation tools. Below is a look at the tools with a table denoting each features function.

Figure 2 - firstPage tools



|---------- 1 ----------| 2 3 4 5 6 7

Key	Name	Use
1	Address Bar	Enter the URL of the site you want to navigate to ‘http:’ in links is not required as the browser will add this for you. If a bad URL is entered error boxes will appear
2	Navigation Button	This button sends the URL to the server and requests HTML code back
3	Refresh Page Button	Re-loads the current page from the URL in the address box
4	Backs Button	Takes you 1 website back into the past, unless it can go back no further
5	Forwards Button	Undoes 1 step of the backs button unless it is at the most recent entry
6	Home Page Button	Loads the home page, can be changed in the Favourites tab.
7	Add To Favourites	Allows entry of details of a new favourite to be added to the favourites list

When navigating from one website to another the URL request will then be sent, and depending on the Status code, it will display one of two things. If the Status code displayed at the bottom of window in Figure 1, is OK the HTML will be displayed as above. Otherwise the user will be shown a box informing them of the error and its code as in the 'Error Loading Page' and 'Bad URL' box below.

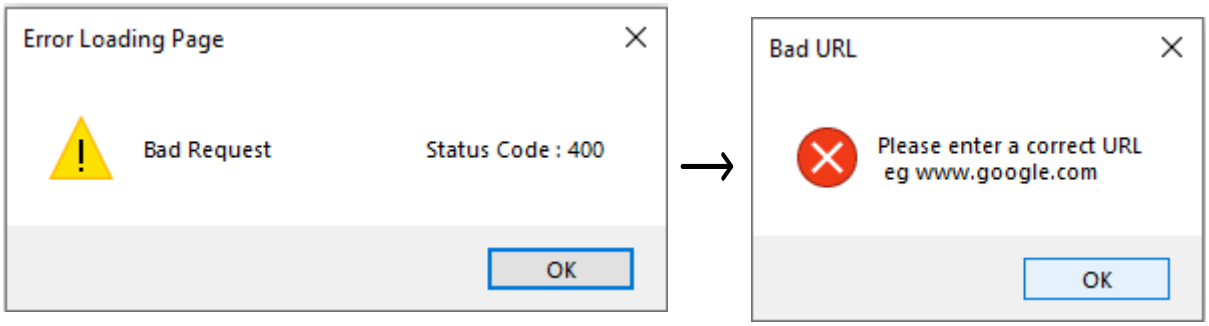


Figure 3 - Error Boxes displayed when an erroneous URL is entered and navigation button is pressed

The error will then be displayed in the HTTP status code box too, as in the Figure 1 and highlighted below in Figure 4.

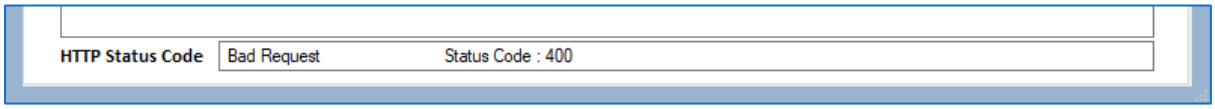
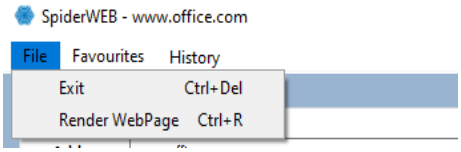


Figure 4 - HTTP Status code box with an error code present

Strip Menu Options

Next let us example the StripMenu at the top of Figure 1 in closer detail. This menu has 3 sections that expand into further actions which are shown below in Figure 5. Each of these options has the name of the option, followed by the keyboard shortcut. The options in these sub menus are as follows :



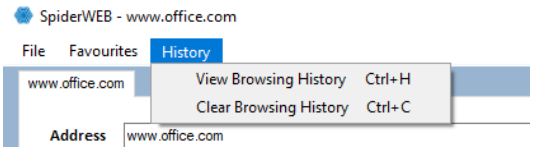


Figure 5 Strip Menu Options Exploded to view further options

File

Exit – Exits the application

Render Web Page – Renders the URL given in the address box in a new window

Favourites

Manage Favourites – Opens the Favourites tab. Here you can add new favourites, remove favourites and change the homepage

History

View Browsing History – Opens the History tab. Here the browsing history can be viewed and searched.

Clear Browsing History – Delete the Browsing history

Render Web Page (Ctrl + R)

This feature allows the user to view a rendered version of the sit currently entered in the address box. Shown below in Figure 6 is difference between the HTML raw code in the background against the rendered page in the foreground.

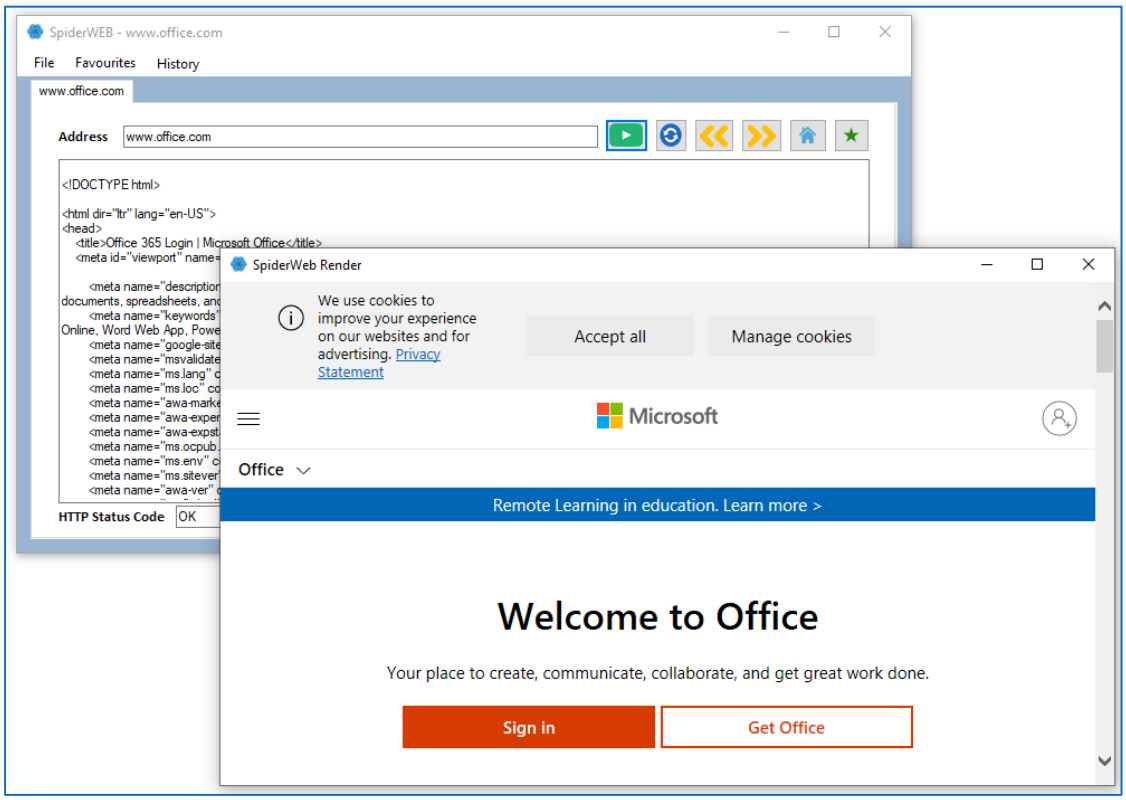


Figure 6 - Displays two views of the same site, the backmost window being the HTML code and the window in the foreground being the rendered site at the same URL.

Manage Favourites / Favourites Tab (Ctrl + F)

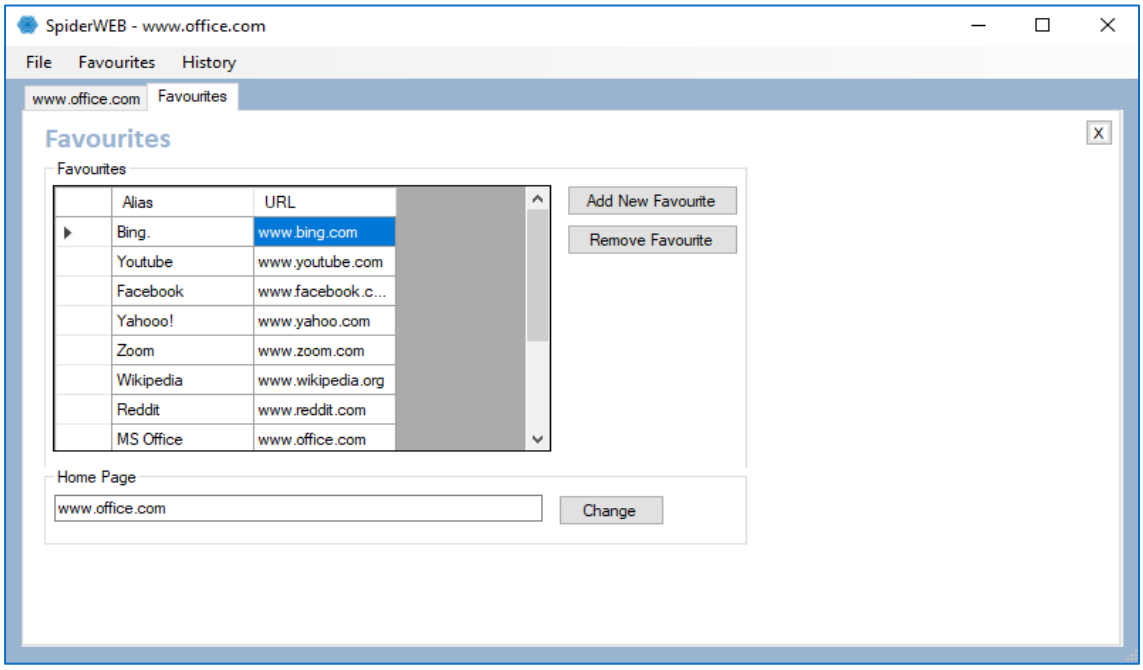


Figure 7 - View of the Favourites Tab - Accessed via the Favourites menu and pressing 'Manage Favourites'.

This page allows users to view their favourites; a list which is saved between sessions to allow users fast and easy navigation to their favourite sites via double clicking on its URL. This tab allows users to add or remove items to the list. After pressing the "add new favourite" button, a new tab will appear asking the user to provide a Nickname, A URL and asked (via a checked box) if they want to also make this their new homepage. Once added it will appear in the data grid view, and the URL can be double clicked to navigate to that site. A user may add multiple instances of the same website. Removing favourites is just as easy, the user simply selects the URL of the favourite they wish to remove and then press the 'remove favourite' button. At the bottom of this tab we have the homepage displayed next to a button labelled 'change'. This button is used to select a new home page by addition of a new one. Pressing the X box on the right will result in the tab closing.

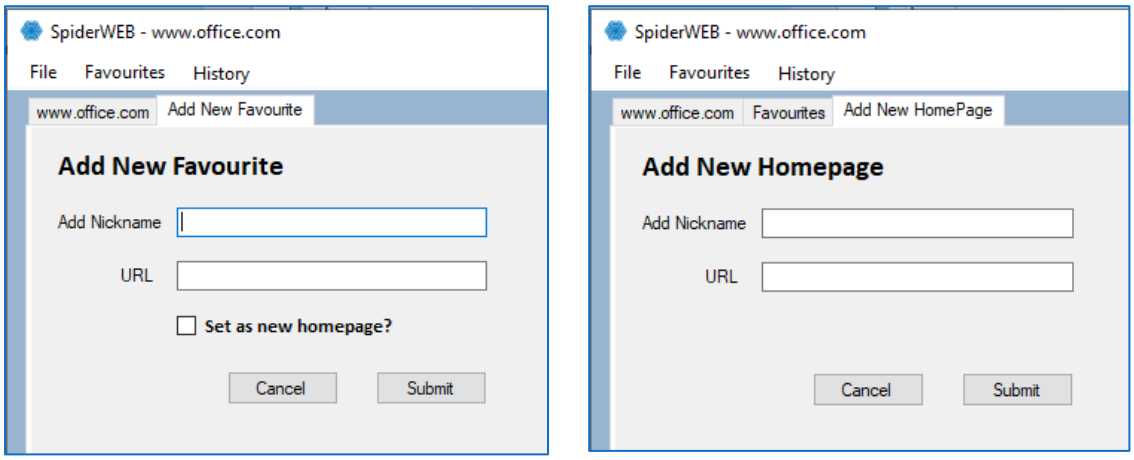


Figure 8 - The two forms that are presented to the user when adding a new Favourite vs when the user adds a homepage. The user can tick the check box to make the favourite the new home page too.

View Browsing History (Ctrl + H)

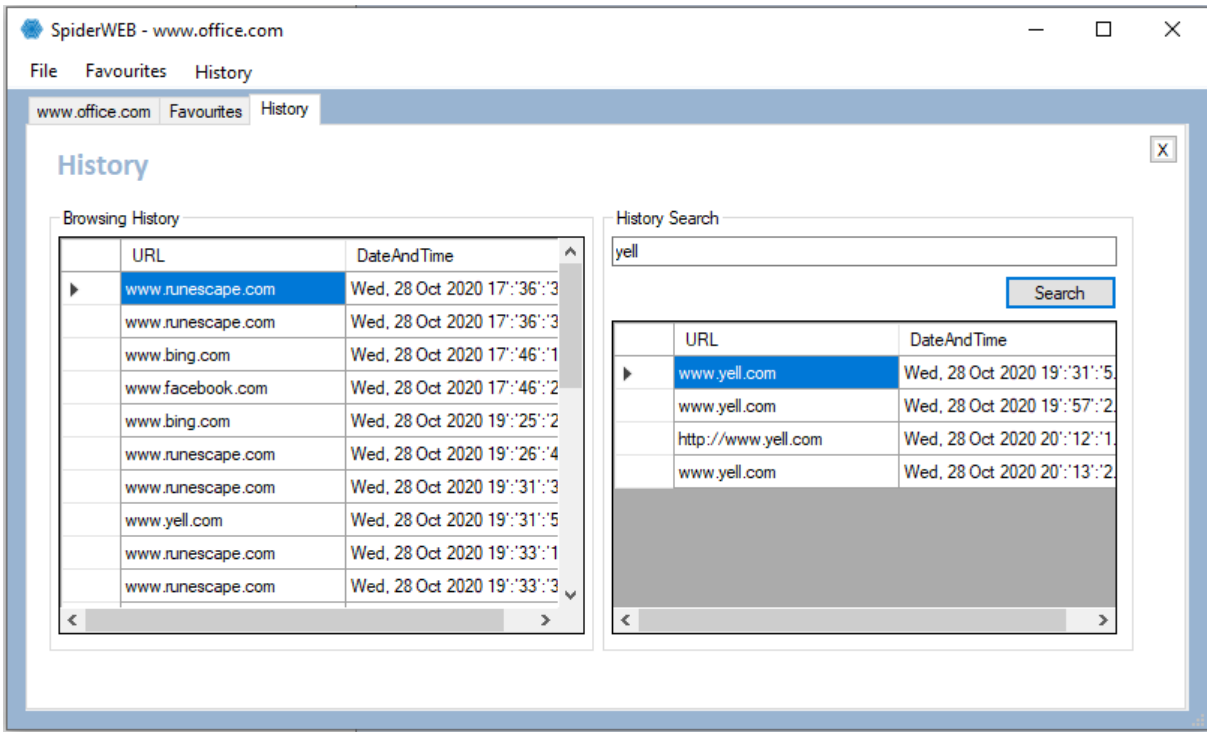


Figure 9 - Displays the History tab where users can view and search sites they have previously visited

This window shows the user a list of sites they have been to in the left hand 'Browsing history' group box. This list is updated every time a user presses the navigation button on the firstPage or double clicks the URL in

favourites or history. As shown about a user can select a cell of interest. Double clicking the URL cell will result in the user being taken to that site. On the right of Figure 9, we see the 'History Search' option. This allows a user to enter a word or URL and search if this item is within the history. For example the user may have forgotten a websites name but know that it contained the word 'yell'. As shown in Figure 9, searching 'yell' brings up 4 different entries to www.yell.com. The user can close this tab by pressing the X box in the top right of the tab. For both History and Favourites, it should be known that one cannot open multiple versions of either tab. In Figure 9 if the user were to do this for either one, the desired tab would move to the active tab.

Clear Browsing History (Ctrl + C)

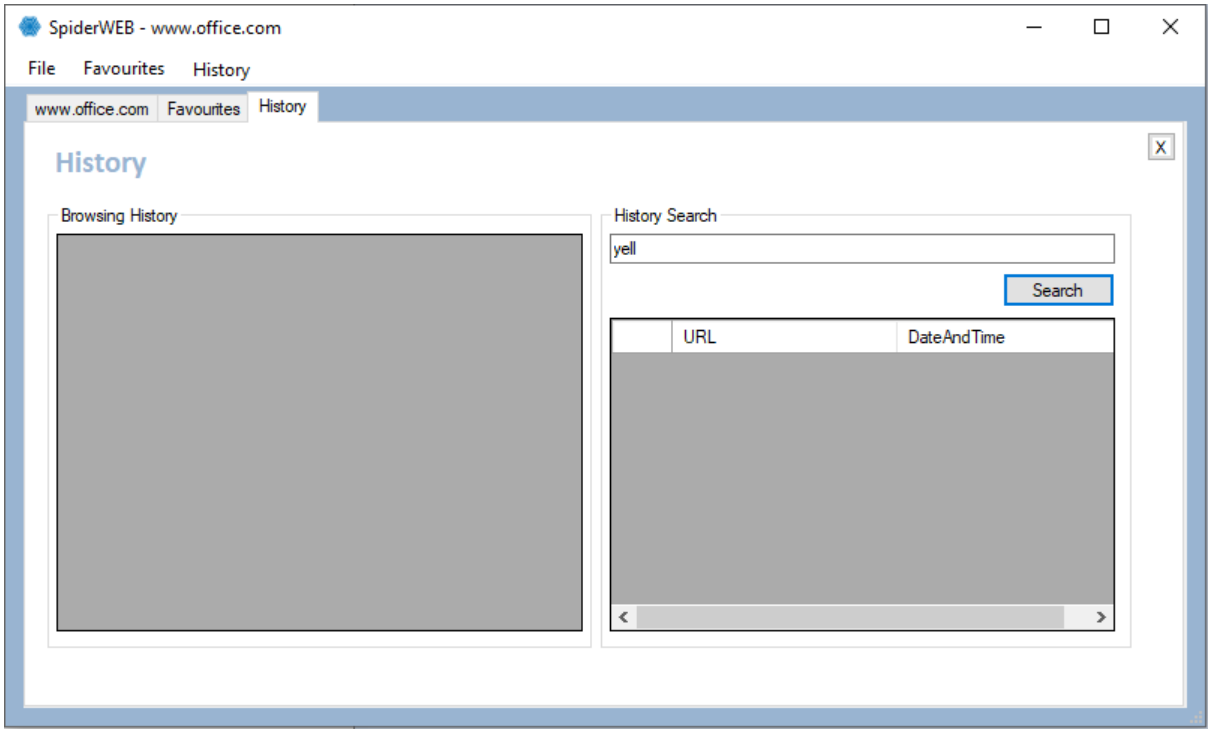


Figure 10 - The same view as Figure 9, however the clear browsing history option has been select thus there are no entries in the Browsing history box or any results for the previous search

The clear browsing history option on the history menu strip removes all websites the user has visited, and the view in Figure 10 is the same as Figure 9, only that the clear browsing history has now been pressed thus there are no results in the browsing history box. Similarly the search of 'yell' now turns up no results.

There is an accompanying video to this report that explains the features of the browser in depth, it can be found at - https://youtu.be/0hoOarNd5cs

Below is a quick reference table for the functionality of each page :

Feature	Feature Summary
FirstPage	Enter a URL and have its HTML response and Status code displayed in the boxes below.
HistoryTab	Display and explore user history
FavouritesTab	Display & edit favourites

DEVELOPER GUIDE

General Overview

The SpiderWeb software is organised into an MVC framework. The Model is where the objects are defined and handles the HTTP communication. The view handles aspects of user interaction are dealt with such as action listeners, layout and data handling. The controller starts the GUI and liaises between the model and view. The firstPage acts as an anchor, to which the other elements are called, added to and removed on demand. This means the bulk of the functional code lies in the firstPage class. The overall architecture is defined in the class diagram which can be found in the appendix.

SpiderWeb - Class

The SpiderWeb class is the beating heart of this application as it contains the methods that handle the HTTP Communication. HTTP communication is invoked by the navigateToURL method, outlined below in the following code fragment. It sets the values of the SpiderWeb Object property URL then calls the HTMLResonseCode() and HTTPSStatus code() method from the SpiderWeb class.

```
//http request using the current given URL in the addressBox by calling methods from spider web class handling request and responses
htmlCodeTextBox.Text = SW.RetrieveHTMLCode(SW);
statusTextBox.Text = SW.HTTPSStatusCode;
```

This is an obvious point for future development, by implementing the previously discussed WebClient or HttpClient classes over HTTPRequest. Despite this, as the application is simple in its requirements HTTPRequest was able to handle the load with relative ease. Within the RetrieveHTMLCode(SpiderWeb SW) method we find the code to create a server request from a given URL as shown below :

```
//URL request to a server
request = (HttpWebRequest)WebRequest.Create(SW.URL);
```

The request variable is of type HttpWebRequest and is set to product webRequest.Create() being passed the SpiderWeb object property URL. This property is set by the user at runtime via a text box in the GUI. There are a few accepted default values that are not mentioned to prevent status codes errors 300-399 : number of redirections by default set to automatic (50) and the default value of AllowAutoRedirect (true);

```
//Response from a server
response = (HttpWebResponse)request.GetResponse();

//will only get here if status == OK
StreamReader responseStream = new StreamReader(response.GetResponseStream());

//reader
SW.HTMLResponseCode = responseStream.ReadToEnd();
SW.HTTPSStatusCode = string.Format("{0} \t\t Code : {1}", response.StatusDescription, (int)response.StatusCode);

//close the stream to release the connection, prevents the application running out of connections
responseStream.Close();


}
```

The response variable shown in the above fragment describes the WebResponse response from the server and is of type HttpWebResponse hence the typecasting. As this is not the webpage itself but the object that describes the response, to get the actual web page text a responseStream method is opened and set to the HTMLResponseCode of the SpiderWeb object. The status code is then interpreted by calling additional class features StatusDescription and StatusCode. StatusDescription gives the text equivalent of the status code but in this case, both are taken and stored to the spider web object property HTTPSStatusCode. The stream is then closed to prevent the program running out of connections. The HTMLResponseCode and HTTPSStatusCode are then used as the content sources for the htmlCodeTextBox elements on the GUI during the NavigateToURL() method, thus displaying the response to the user.

SpiderWeb -Object

This object is used to pass HTTP requests from the GUI interface to the server and back, as it mediates this exchange it can also be used as a discreet data source for the history function of the browser. Every time the user makes a HTTP request the current object is added to a list and contains the time and URL details of said request.

Favourites - Object

The favourites object is used store user entered information about an entered URL, namely the URL, nickname and Boolean value that dictates it role as a homepage or not. If set to true, this is the homepage.

History Traversal Buttons

The forwards and backwards buttons within this program follow a simple logic that allows the user to traverse the URLs of their browsing history both backwards and then forwards. SpiderWeb has a historyCounter value that is used as an index reference to attain the previous or subsequent web page, it is by default set to the same length as the browserHistory -1 to convert the count to an index. As the newest items are added to the back of the list, we must use the size of the list to determine where to begin our counter. For example the user is going to press the back button, currently the counter if called as an in this state will return the item it is currently on thus the first action must be to reduce this counter by 1 when clicked. This counter can now be used an index for the method list<T>ElementAt[x] that allows for random access to the object held at that index. The forward button works by the same logic, but in reverse.

```
private void backButton_Click(object sender, EventArgs e)
{
    // decrement the counter, to give the next item in the list if pressed again - As the newest items are added to the end of the list
    historyCounter--;

    //if the history counter is set to the number of objects in the browserHistory list minus 1 OR the history counter is not less than 0.
    //BrowserHistory count-1 as we shall use this as an index reference, that start at 0 thus prevents a ArrayOutOfBoundsException. Similarly,
    //if the counter is less than 0 it will also cause this error.
    if ((historyCounter <= browserHistory.Count - 1 && historyCounter >= 0))
    {
        //clear the html from current website
        htmlCodeTextBox.Text = "";

        //get the element at the current index value of the history counter
        var lastEntry = browserHistory.ElementAt(historyCounter);

        //set the addressBox text to be the url property of the object at this element
        addressBox.Text = lastEntry.URL;

        //Set the SpiderWeb object to the lastEntry object
        SW = lastEntry;

        //navigate to the site without adding to history as it is already in the history
        htmlCodeTextBox.Text = SW.RetrieveHTMLCode(SW);
        statusTextBox.Text = SW.HTTPSStatusCode;
    }
    else
    {
        //reverse the change to the history counter
        historyCounter++;
    }
}
```

Clickable history / favourites

A feature of this Application is the ability to double click on a DataGridView square containing a URL in both the Favourites and History tab and navigate to it. This is made feasible by the flexibility of the DataGridView, which allows for action listeners on individual cells. As the code fragment below shows, its functionality hinges on a for loop. If the selected cell contains ‘www.’ then it will be set as the URL in the firstPage tab, the navigateToURL() method is then called to load the site.

```
if (favouritesGridDataView.CurrentCell.Value.ToString().Contains("www."))
{
    //set the addressBox text to be this string
    addressBox.Text = favouritesGridDataView.CurrentCell.Value.ToString();
```

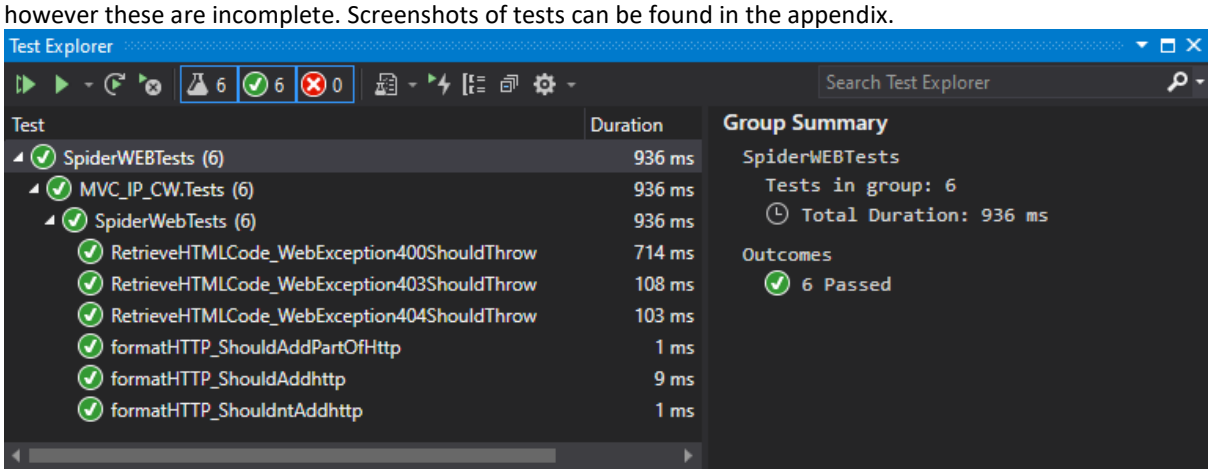
Data storage & XML handling

Storage of objects between sessions is handled by having two separate XML files in the solution folder. On application start-up, these files are deserialised into the appropriate list: browserHistory or userFavourites. Every time the list is changed it is serialised and saved to new XML file of the same name (thus overwriting the old one) as appending was causing issues; a feature which could be improved in future versions. The list must then be deserialised and set as the contents of the list once more. Whilst a little cumbersome it ensures no data is lost if the user quits the app or the app crashes unexpectedly.

TESTING

Testing was approached in two ways, both Xunit tests and User tests were implemented when evaluating the methods and functionality of the GUI. Xunit testing was chosen as Nunit tests everything in the same fixture (or class) using the same instance for each test, Xunit does not and creates a new instance for every test. This ensures that one test result does not accidentally cause another to fail.

Xunit tests should have been run on all the functional methods implemented by the GUI and model Elements, however these are incomplete. Screenshots of tests can be found in the appendix.



Void methods proved difficult or beyond my means to test, they were tested using a logic test. Ie did they do the work that they are supposed to? Below is a set of results for the void methods, excluding action listers that simply call methods.

Void method	Logic Test to pass after running	Passed?
loadHistory()	newHistory file is loaded into to the historyDataGridView, no missing entries (manual check before and after)	Y
addHistory()	New history entry appears at the bottom of the historyDataGridView and is saved in file	Y
clearHistory()	File empty, historyDataGridView empty, search bar no longer shows entries for a previous search	Y
openHistoryTab()	Favourites tab opens and user is directed to it, if open no other favourites pages shall open – the user should be redirected to the open favourites tab	Y
navigateToURL()	Relevant html code appears in htmlCodeTextBox and correct status code appears. For both working and error causing sites.	Y
openfavouritesTab()	Favourites tab opens and user is directed to it, if open no other favourites pages shall open – the user should be redirected to the open favourites tab	Y

loadFavourites()	newFavourites.xml file is loaded into to the favouritesDataGridView, no missing entries (manual check before and after)	Y
addToFavourites()	New history entry appears at the bottom of the favouritesDataGridView and is saved in file.	Y
displayHomePage();	Navigates the user to the set homepage, if not set. Do nothing.	Y
SelectATab();	Changes the users selected tab	Y
TraverseBrowserHistory();	URL loaded on the firstPage moves in the correct direction relevant to button pressed. Checked against the history log.	y

The GUI code involved in user interaction was thoroughly tested manually, via multiple rounds of data entry, removal and general navigation of the application. Where appropriate, message boxes alert the user to erroneous input that have causes potential errors within the code. As rounds of testing progressed, a lot of the user code became surrounded by try/catch blocks, switch and if/else statements to improve stability. Particularly at the server interaction level, not been accepted The GUI was tested robustly however for user caused errors. One known bug with the application is that on publishing, it sometimes displays multiple of the same message box one after the other, the source of which is still to be determined as it does not happen prior to publish. The table below summarises just some of the changes made that only became apparent through testing as a user

Case Caught by User Testing	Solution
XML files deleted between session caused deserialising errors when loading.	If loop that creates a new file if one does not exist within the file path.
Creating multiple favourites or history tabs caused information between tabs to become clear	If loop that tests to see if the relevant tab already exists, if so it then makes that the selected tab
Adding aliases with www. And http in the nickname caused errors as the doubleClick action lister would believe it as a valid site and try to navigate there	By not allowing users to set nicknames containing www. Or http it removed this issue
Pressing the forwards or back buttons multiple times caused an array out of bounds exception	Setting the limits of the method that responds to the history counter to be between >= 0 and list length -1 prevented this issue. And adding a for loop that undoes any incrementing done by the failed attempt at history traversal.
Clearing the browsing history and immediately restarting the program this caused deserialization issues as the content of the file would now be (0,0).	A try catch block was created when deserialising the XML files, this allowed the program to handle any issues when deserialising by simply creating a new file.

REFLECTIONS

The C# language provided an excellent base for making a simple web Browser. The fact there is an entire GUI class already devoted to it already is testament to its applicability in this situation. As the language is object orientated (OOP) it is built with abstraction, encapsulation, inheritance and polymorphism at its core, which when used in conjunction with MVC framework provides excellent modularity within the code. This modularity provides an excellent trouble shooting platform as you can pinpoint exactly which class is causing errors with ease. Inheritance allowed for easy implementation of the ISerializable interface within my model classes, reducing the amount of coding I had to do and allowed for implantation of methods that I could otherwise not code. Whilst I did not use them per se, I also dabbled with IEnumerable for list traversal but could not get it to fit seamlessly in the timeframe. Lastly, as C# complies into a .exe (executable) and contains MSIL code, this is ideal for application development as it makes it easy distribute and run.

CONCLUSIONS

Overall, I am fairly happy with the quality of the code, coding practices used and delivery of requirements, with notable highlights being the simple method I devised for implementing the back button. I would have however, liked to try to implement the HttpClient class for web handling as it is viewed a better way to deal with HTTP requests for those running framework 4.5 onwards. Another limitation of my code lies in that I didn't utilise binding the data to the DataGridView, which would have allowed for greater flexibility with the display and editing of objects within. I had multiple, failed attempts at getting this to work and time was running short thus I had to move on. I would also have liked to better utilise the MVC structure within my code as the bulk of the functional code is within the view classes. I would, looking back, spend greater time separating the view from the functional code.

REFERENCES – SOURCES USED IN CODE DEVELOPMENT

Codecademy. 2020. MVC: Model, View, Controller | Codecademy. [online] Available at: <https://www.codecademy.com/articles/mvc#:~:text=MVC%20gives%20you%20a%20starting,developers%20to%20understand%20your%20code.> [Accessed 29 October 2020].

Corey, T., 2020. [online] www.youtube.com. Available at: <https://www.youtube.com/watch?v=ub3P8c87cwk&ab_channel=IAmTimCorey> [Accessed 29 October 2020].

C-sharpcorner.com. 2020. How To Serialize And Deserialize An XML File Into A C# Object (And Vice-Versa). [online] Available at: <https://www.c-sharpcorner.com/blogs/serialize-and-deserialize-xml-file-into-c-sharp-object-and-vise-versa> [Accessed 29 October 2020].

Docs.microsoft.com. 2020. .NET API Browser. [online] Available at: <https://docs.microsoft.com/en-us/dotnet/api/> [Accessed 20 October 2020].

Savanttools.com. 2020. Test HTTP Status Codes. [online] Available at: <http://savanttools.com/test-http-status-codes> [Accessed 29 October 2020].

Stack Overflow. 2020. Newest Questions. [online] Available at: <https://stackoverflow.com/questions/> [Accessed 29 October 2020].

W3schools.com. 2020. C# Tutorial (C Sharp). [online] Available at: <https://www.w3schools.com/cs/> [Accessed 29 October 2020].

Xunit.net. 2020. Getting Started: .NET Framework With Visual Studio > Xunit.Net. [online] Available at: <h

ttps://xunit.net/docs/getting-started/netfx/visual-studio> [Accessed 29 October]

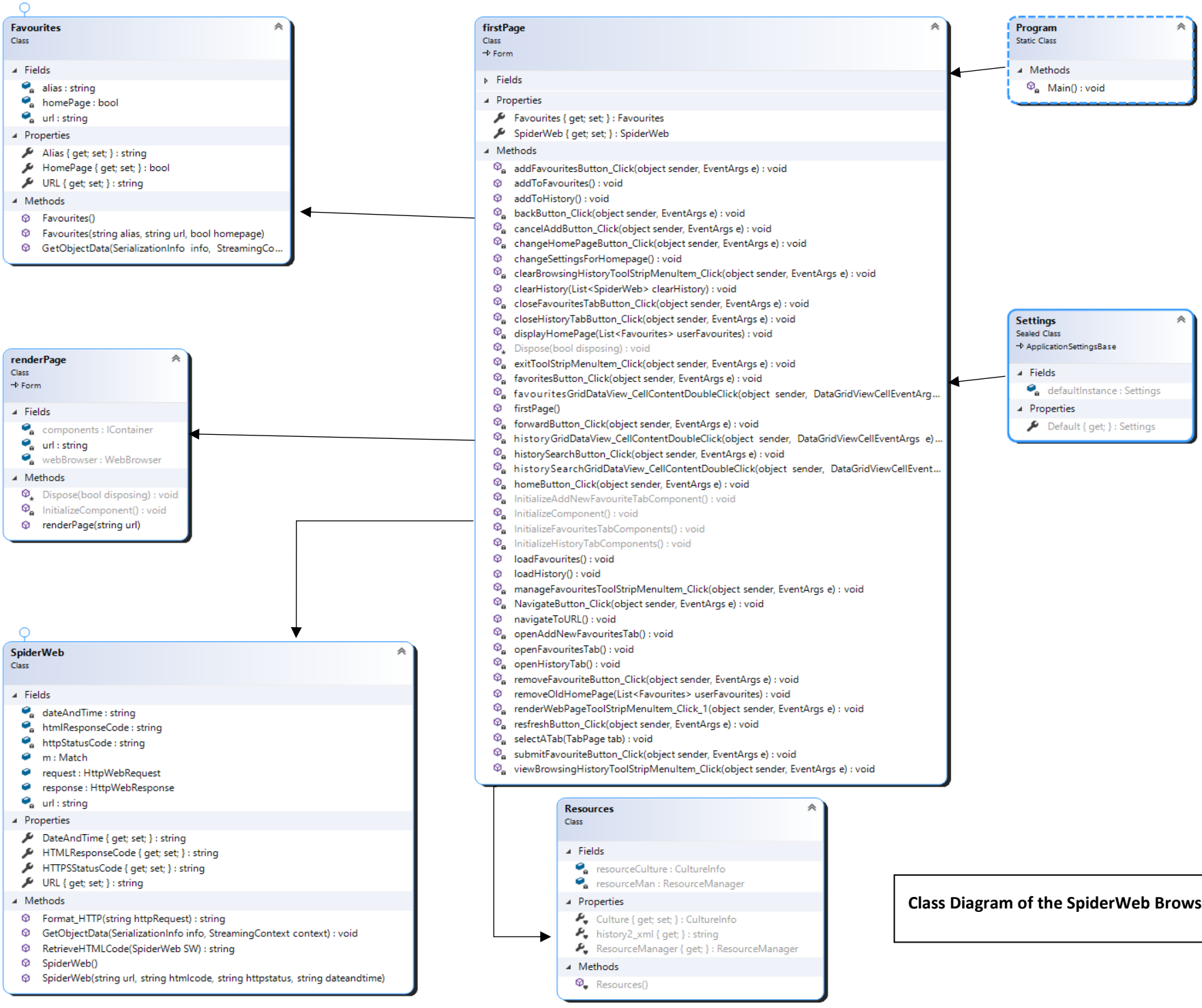
APPENDIX

Test of formatHttp method

```
[Fact]
0 references
public void formatHTTP_ShouldAddhttp()
{
    // Arrange
    string urlWithoutHttps = "www.google.com";
    string expected = "http://www.google.com";

    // Act
    string actual = MVC_IP_CW.SpiderWeb.Format_HTTP(urlWithoutHttps);

    // Assert
    Assert.Equal(expected, actual);
}
[Fact]
0 references
public void formatHTTP_ShouldntAddhttp()
{
    // Arrange
    string urlWithoutHttps = "http://www.google.com";
    string expected = "http://www.google.com";

    // Act
    string actual = MVC_IP_CW.SpiderWeb.Format_HTTP(urlWithoutHttps);

    // Assert
    Assert.Equal(expected, actual);
}

[Fact]
0 references
public void formatHTTP_ShouldAddPartOfHttp()
{
    // Arrange
    string urlWithoutHttps = "ht//www.google.com";
    string expected = "http://www.google.com";

    // Act
    string actual = MVC_IP_CW.SpiderWeb.Format_HTTP(urlWithoutHttps);

    // Assert
    Assert.Equal(expected, actual);
}
```

Testing RetieceHTMLCode method

```
/// <summary> Tests if a link containing a link disgned to throw an error throws ...
[Fact()]
0 references
public void RetrieveHTMLCode_WebException400ShouldThrow()
{
    //Arrange
    SpiderWeb testSpiderWeb = new SpiderWeb();
    testSpiderWeb.URL = "http://status.savanttools.com/?code=400%20Bad%20Request";
    testSpiderWeb.DateAndTime = DateTime.Now.ToString("ddd, dd MMM yyy HH':'mm':'ss 'GMT'");

    // Act & Assert
    Assert.Throws<WebException>(() => testSpiderWeb.RetrieveHTMLCode(testSpiderWeb));
}

/// <summary> Tests if a link containing a link disgned to throw an error throws ...
[Fact()]
0 references
public void RetrieveHTMLCode_WebException403ShouldThrow()
{
    //Arrange
    SpiderWeb testSpiderWeb = new SpiderWeb();
    testSpiderWeb.URL = "http://status.savanttools.com/?code=403%20Forbidden";
    testSpiderWeb.DateAndTime = DateTime.Now.ToString("ddd, dd MMM yyy HH':'mm':'ss 'GMT'");

    Assert.Throws<WebException>(() => testSpiderWeb.RetrieveHTMLCode(testSpiderWeb));


}

/// <summary> Tests if a link containing a link disgned to throw an error throws ...
[Fact()]
0 references
public void RetrieveHTMLCode_WebException404ShouldThrow()
{
    //Arrange
    SpiderWeb testSpiderWeb = new SpiderWeb();
    testSpiderWeb.URL = "http://status.savanttools.com/?code=404%20Not%20Found";
    testSpiderWeb.DateAndTime = DateTime.Now.ToString("ddd, dd MMM yyy HH':'mm':'ss 'GMT'");

    Assert.Throws<WebException>(() => testSpiderWeb.RetrieveHTMLCode(testSpiderWeb));
}
```



Class Diagram of the SpiderWeb Browser Project