F20SC: Industrial Programming

Coursework 1

Web Browser

Tomasz Mosak (tpm4@hw.ac.uk)
H00245051

Introduction

The purpose of this report is to show the thought process and struggles involved in the implementation of the simple web browser in the language c#. It will go over the met requirements but also touch of any requirements that were not implemented alongside the Design Considerations which will give an in depth insight into the reasoning behind class design, data structures but also GUI design and others. The report will also provide full User/Developer guides. Finally, the report will show a test table showing the expected/output results of extensive testing done on the browser.

At the beginning of the development process, it was thought to be untroublesome and quite straightforward due to the first few requirements getting accomplished within an hour of starting the project. Nevertheless, this quickly became the downfall of the development process as the rest of the implementation was left to a later date which was too close to the deadline without the insight of the difficulty and time consumption the other requirements had hidden behind them.

Furthermore, near the start of the implementation, an assumption was made that the URL would magically always have http://www. at the start of the URL due to the comfort of google chrome browser doing that automatically. For testing, that assumption quickly was debunked when the URI format exception kept showing up and with that the need for checking the URL string was born. Another assumption that was made was that transferring the project from a console application to a windows form application would've been a simple and easy task which as later discussed in the Design Considerations, was a really bad idea.

Web Browser Requirements CheckList

1.1	Sending HTTP Request Messages for URLs Typed by the user
2.1	Receiving HTTP Response Messages
2.2	Display The Contents of the Messages on the interface
2.3	Display HTTP Response Status Error Codes and corresponding Error Messages
2.3.1	Display the 400 Bad Request Error Code + Message
2.3.2	Display the 403 Forbidden Error Code + Message
2.3.3	Display the 404 Not Found Error Code + Message
3.1	Display the HTTP response status code and title at the top of the browsers main page
3.2	Reload Functionality which sends another HTTP request to the current web page
3.2.1	Show the Status Code + Title in the Display Textbox alongside the content
4.1	Home page functionality implemented
4.1.1	The user is able to create/edit the home page URL for the homepage
4.1.2	The Homepage URL is loaded on browser startup
5.1	Favourites functionality implemented
5.1.1	The user is able to add a URL from a web page requested to a list of favourites
5.1.2	The user should be able to associate a name with each favourite URL
5.1.2.1	The SYSTEM is able to associate a name with each favourite URL
5.1.3	The user is able to request a favourite web page by clicking its name on the list
5.1.4	The favourites list is loaded on browser startup
6.1	History functionality implemented
6.1.1	The browser maintains a list of URLs corresponding to web pages requested by user
6.1.2	The user is able to navigate to the previous and next pages
6.1.3	The user is able to jump to a page by clicking the link in the History list
6.1.4	The history list is loaded on browser startup

Graphical User Interface Requirements

1.1	A simple GUI is provided
1.1.1	GUI implemented using Windows Forms libraries
1.2	Using the GUI the user is able to perform all operations from previous section
1.3	GUI makes use of menus, shortcut keys as well as buttons to increase accessibility

Design Considerations

Introduction

As mentioned above in the introduction section, the implementation started as a console application with the purpose of quickly getting the functionality of the WebHandler/History/Favourites classes as the Console.Writeline was a very familiar command at the time instead of display debug information in a Textbox which should’ve been the correct way to start this coursework project. Once the main functionality of handling history/favourites dictionaries was successfully implemented, the struggle of porting the project was a big learning lesson.

Class Design

Overall, each of the classes represented an ‘object’ which could interact with the Web Browser, each class was split into their own file to allow easier readability but also extendability of the implementation fairly simple.

To start with the main class, WebHandle, this as the name suggests handles anything to do with accessing the WorldWideWeb. It has smaller helper functions such as InvalidURL & DecodeString which makes sure that the passed URL string has the http://www. & takes the datastream and throws it into a string by reading each line of the datastream respectively. However, the main function that the entire implementation surrounds is the HTTPRequest function which handles the sending of the request messages from URLs, this class returns the htmlstring which has already been through the DecodeString function or if the Response StatusCode is not 200 then throws the appropriate errors. This class also makes sure that all the necessary files are created if not present already.

History/Favourites/Home classes are very similar to each other with the main difference that History deals with DateTime, String tuples, Home class only has to worry about 1 URL string and Favourites class handles 2 string tuples. The main 2 functions that they all have is Add & Reload which as the names suggest either add a new entry to the files and session storage or Reloads them into the session storage from the file.

The DoublyLinkedList file has 2 classes, 1 for the Nodes and 1 for the actual DoublyLinkedList, creating this reminded me of the Binary Search Tree implementation that we completed during lab time as it follows the same principles. The only challenge here was to extend the implementation of a standard DoublyLinkedList to generics with Key, Value types. The DLL class itself isn’t complete with only the necessary functions required for the implementation of the simple web browser being present.

Lastly the WebBrowser class itself, handles the calls to each of the functions named above depending on which button/menu item/key is pressed. It could be considered the heart of the program as without this class, the entire project would be useless.

Data Structure

The data structure used for this coursework has changed near the end of the implementation to make the forward and backward buttons work without the introduction of stacks. The data structure chosen at the start was a Dictionary appeared to be the best choice at the time due to the <K, V> structure of each entry allowing duplicate control over the favourites list but also means that I can have the history list display the DateTime at which the URL was accessed.

This was seamless until the need to index over the dictionary for the forward/backward page access ability was required to be implemented. However, this wasn't the only issue with the preemptively chosen data structure, another issue that appeared was with trying to delete single elements of the Favourites or History lists. It was at that point where the idea of creating a custom DoublyLinkedList was born.

The current Data Structure in the web browser submitted is a generic DoublyLinkedList which takes <K, V> values just like the dictionary which allowed me to maintain all the code just as it was with the dictionary, with the addition of an indexer and pointers to neighbouring nodes. This change allowed the display of History with most recent entries at the top as well as fixing the 2 issues by introducing an indexer over the DoublyLinkedList giving each node a number which we can point to and delete the Selected webpage entry from History or Favourites.

GUI Design

The design of the Graphical User Interface for the web browser if fairly simplistic with multiple buttons along the top and top left of the actual display box which shows the web pages. They all have text on them to allow quick identification of controls available to the user but also fairly large so that the user's don't have to be pixel perfect with their clicks to be able to use the browser.

Advanced Features

The finished implementation makes use of advanced features such as indexers and generics mentioned above but also exceptions. 2 of the 3 mentioned features are all created for the purpose of using the DoublyLinkedList with 2 different tuple data types. The exceptions are used to capture the error status codes in the HTTPRequest function from the WebHandle class and display custom messages to the user showing the Error code as well as a message.

User Guide

Intended Readership

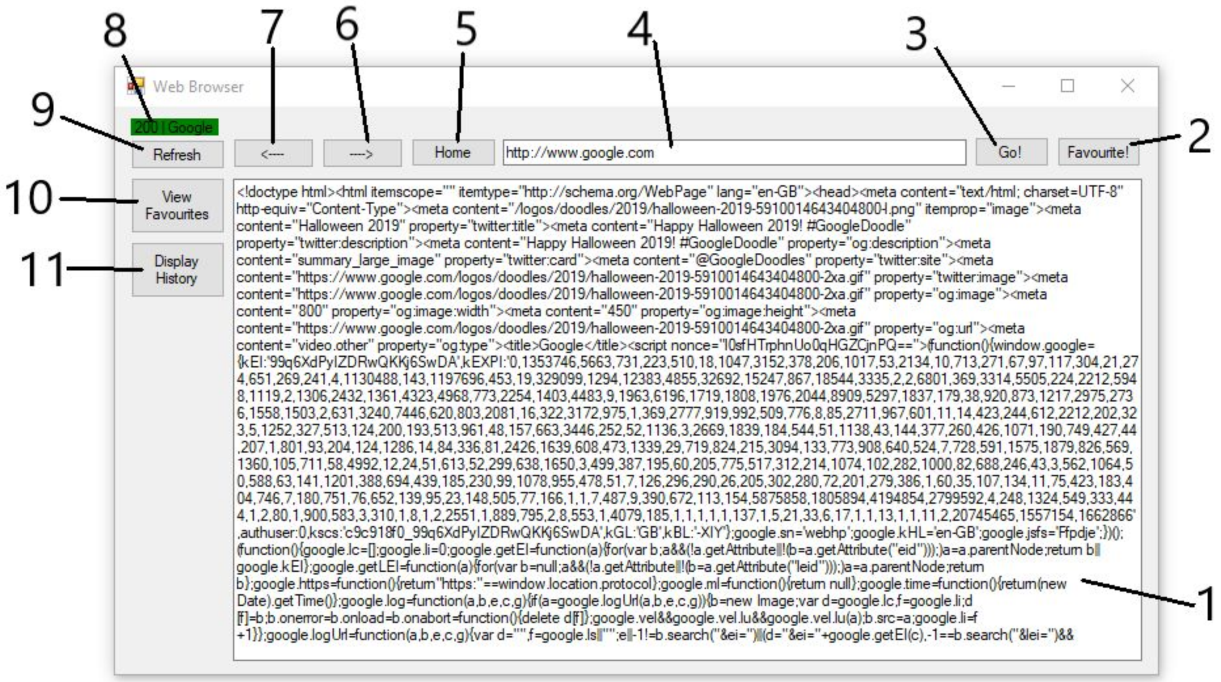
The category of user to whom this User Guide is addressed to are the End Users that are planning on making occasional use of the system but wish to find out about all the features of the system.

Purpose

The purpose of the system is to allow the user to retrieve the html code from any website by specifying its URL. Furthermore, it allows the user to navigate between previous and next web pages that they already visited as well as seeing their entire history which can be cleared. It also allows the user to favourite any URL as well as setting any URL as the homepage which will welcome them when they reload the web browser.

The purpose of this document is to allow the reader an extensive guide on how to access and use every feature of the web browser.

Instructions



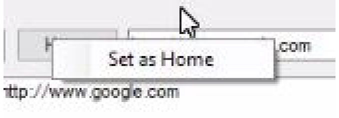
```
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-GB"><head><meta content="text/html; charset=UTF-8"
http-equiv="Content-Type"><meta content="/logos/doodles/2019/halloween-2019-5910014643404800-l.png" itemprop="image"><meta
content="Halloween 2019" property="twitter:title"><meta content="Happy Halloween 2019! #GoogleDoodle"
property="twitter:description"><meta content="Happy Halloween 2019! #GoogleDoodle" property="og:description"><meta
content="summary_large_image" property="twitter:card"><meta content="@GoogleDoodles" property="twitter:site"><meta
content="https://www.google.com/logos/doodles/2019/halloween-2019-5910014643404800-2xa.gif" property="twitter:image"><meta
content="https://www.google.com/logos/doodles/2019/halloween-2019-5910014643404800-2xa.gif" property="og:image"><meta
content="800" property="og:image:width"><meta content="450" property="og:image:height"><meta
content="https://www.google.com/logos/doodles/2019/halloween-2019-5910014643404800-2xa.gif" property="og:url"><meta
content="video.other" property="og:type"><title>Google</title><script nonce="l0sfHTrphnUo0qHGZCjnPQ==">(function(){window.google=
{kEI:'99q6XdPyIZDRwQKKj6SwDA',kEXPI:'0,1353746,5663,731,223,510,18,1047,3152,378,206,1017,53,2134,10,713,271,67,97,117,304,21,27
4,651,269,241,4,1130488,143,1197696,453,19,329099,1294,12383,4855,32692,15247,867,18544,3335,2,2,6801,369,3314,5505,224,2212,594
8,1119,2,1306,2432,1361,4323,4968,773,2254,1403,4483,9,1963,6196,1719,1808,1976,2044,8909,5297,1837,179,38,920,873,1217,2975,273
6,1558,1503,2,631,3240,7446,620,803,2081,16,322,3172,975,1,369,2777,919,992,509,776,8,85,2711,967,601,11,14,423,244,612,2212,202,32
3,5,1252,327,513,124,200,193,513,961,48,157,663,3446,252,52,1136,3,2669,1839,184,544,51,1138,43,144,377,260,426,1071,190,749,427,44
,207,1,801,93,204,124,1286,14,84,336,81,2426,1639,608,473,1339,29,719,824,215,3094,133,773,908,640,524,7,728,591,1575,1879,826,569,
1360,105,711,58,4992,12,24,51,613,52,299,638,1650,3,499,387,195,60,205,775,517,312,214,1074,102,282,1000,82,688,246,43,3,562,1064,5
0,588,63,141,1201,388,694,439,185,230,99,1078,955,478,51,7,126,296,290,26,205,302,280,72,201,279,386,1,60,35,107,134,11,75,423,183,4
04,746,7,180,751,76,652,139,95,23,148,505,77,166,1,1,7,487,9,390,672,113,154,5875858,1805894,4194854,2799592,4,248,1324,549,333,44
4,1,2,80,1,900,583,3,310,1,8,1,2,2551,1,889,795,2,8,553,1,4079,185,1,1,1,1,1,137,1,5,21,33,6,17,1,1,13,1,1,11,2,20745465,1557154,1662866'
,authuser:0,kscs:'c9c918f0_99q6XdPyIZDRwQKKj6SwDA',kGL:'GB',kBL:'-XIY'};google.sn='webhp';google.kHL='en-GB';google.jsfs='Ffpdje';})();
(function(){google.lc=[];google.li=0;google.getEI=function(a){for(var b;a&&(!a.getAttribute||!(b=a.getAttribute("eid")));)a=a.parentNode;return b||
google.kEI};google.getLEI=function(a){for(var b=null;a&&(!a.getAttribute||!(b=a.getAttribute("leid")));)a=a.parentNode;return
b};google.https=function(){return"https:"==window.location.protocol};google.ml=function(){return null};google.time=function(){return(new
Date).getTime()};google.log=function(a,b,e,c,g){if(a=google.logUrl(a,b,e,c,g)){b=new Image;var d=google.lc,f=google.li;d
[f]=b;b.onerror=b.onload=b.onabort=function(){delete d[f]};google.vel&&google.vel.lu&&google.vel.lu(a);b.src=a;google.li=f
+1}};google.logUrl=function(a,b,e,c,g){var d="",f=google.ls||"";e||-1!=b.search("&ei=")||(d="&ei="+google.getEI(c),-1==b.search("&lei=")&&
```

1. This is the Display Textbox which will show the html content of each website visited.
2. This button allows the user to add the current URL seen in (4) to the list of favourites Successfully added Google to Favourites << This is the confirmation text.
3. This button allows the user to display the URL written in (4) in (1) Display Textbox
4. This is the textbox in which the user writes the URL just as seen in the snippet above.
5. Takes the user to their homepage if they defined one. See: (???)
6. Allows the user to move to the next page if there is one. This button is basically useless unless the user first presses (7)
7. Allows the user to move to the previously visited page if they visited more than just the homepage.
8. This displays the Status Code and Colour of the Code(Green means the URL is accessible, Red means Error)& title of the currently visiting page.
9. This button refreshes the current page and gives the status code at the top of (1) as well as the html content.
10. Pressing this button shows the user their favourites list, from there more functionality available. See: (???)

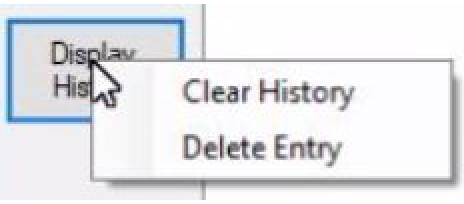
11. Pressing this button shows the user their history list, from there more functionality available See: (???)

Hidden Features

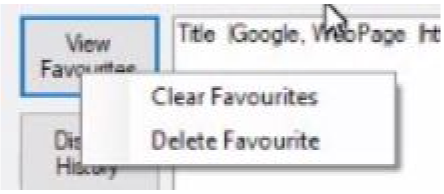
12. Upon right clicking the home button, the “Set As Home” button appears. This allows the user to take the current page and make it their homepage



13. Upon right clicking the Display History(11)/Display Text(1), the user can either clear the entire history or just delete any entry of the history they clicked on.



14. Upon right clicking the View Favourites(10)/Favourite!(2)/Display Text(1), the user can either clear all their favourites or just delete 1 entry from the favourites list



15. Last feature of the program allows the user to press the “ENTER” key on their keyboard to also display the typed URL in (4). This allows the user to skip pressing Go(3) button.

Conclusion

This concludes the User Guide and all the features available in the Web Browser application.

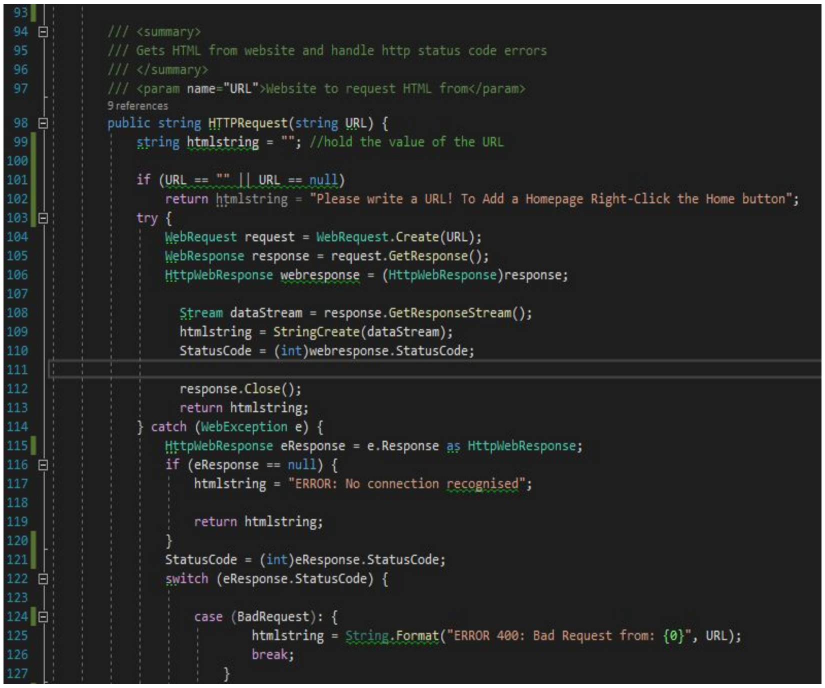
Developer Guide

Overview

The main purpose of this software is to display html code from various URLs but also give the end user functionality to store various URLs in different files such as Favourites or Homepage. It also handles the relationship between files and the Doubly Linked List data structure.

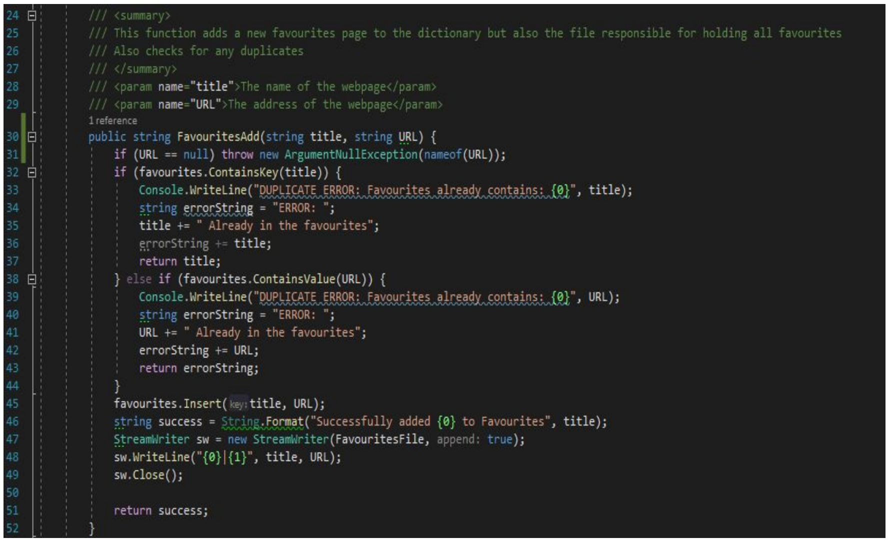
Features

The main chunk of code that is used the most in the entire project is the HTTPRequest(URL) as this handles every request done by the user which nearly all the buttons available to the users provide.



```
 93
 94         /// <summary>
 95         /// Gets HTML from website and handle http status code errors
 96         /// </summary>
 97         /// <param name="URL">Website to request HTML from</param>
            9 references
 98         public string HTTPRequest(string URL) {
 99             string htmlstring = ""; //hold the value of the URL
100
101             if (URL == "" || URL == null)
102                 return htmlstring = "Please write a URL! To Add a Homepage Right-Click the Home button";
103             try {
104                 WebRequest request = WebRequest.Create(URL);
105                 WebResponse response = request.GetResponse();
106                 HttpWebResponse webresponse = (HttpWebResponse)response;
107
108                   Stream dataStream = response.GetResponseStream();
109                   htmlstring = StringCreate(dataStream);
110                   StatusCode = (int)webresponse.StatusCode;
111
112                   response.Close();
113                   return htmlstring;
114             } catch (WebException e) {
115                 HttpWebResponse eResponse = e.Response as HttpWebResponse;
116                 if (eResponse == null) {
117                     htmlstring = "ERROR: No connection recognised";
118
119                     return htmlstring;
120                 }
121                 StatusCode = (int)eResponse.StatusCode;
122                 switch (eResponse.StatusCode) {
123
124                     case (BadRequest): {
125                             htmlstring = String.Format("ERROR 400: Bad Request from: {0}", URL);
126                             break;
127                         }
```

This creates a local output string (Line 99), then using the WebRequest/WebResponse methods we're able to retrieve what we want which is the datastream. The datastream gets passed to a different function called StringCreate(dataStream) which will populate the htmlstring with the actual html from the website. We then close the connection and return the string back to the WebBrowser DisplayTxt. Furthermore, this catches any errors and catagorizes them in a switch statement over the actual statuscode. What's quite nice is that this is built into the WebResponse.

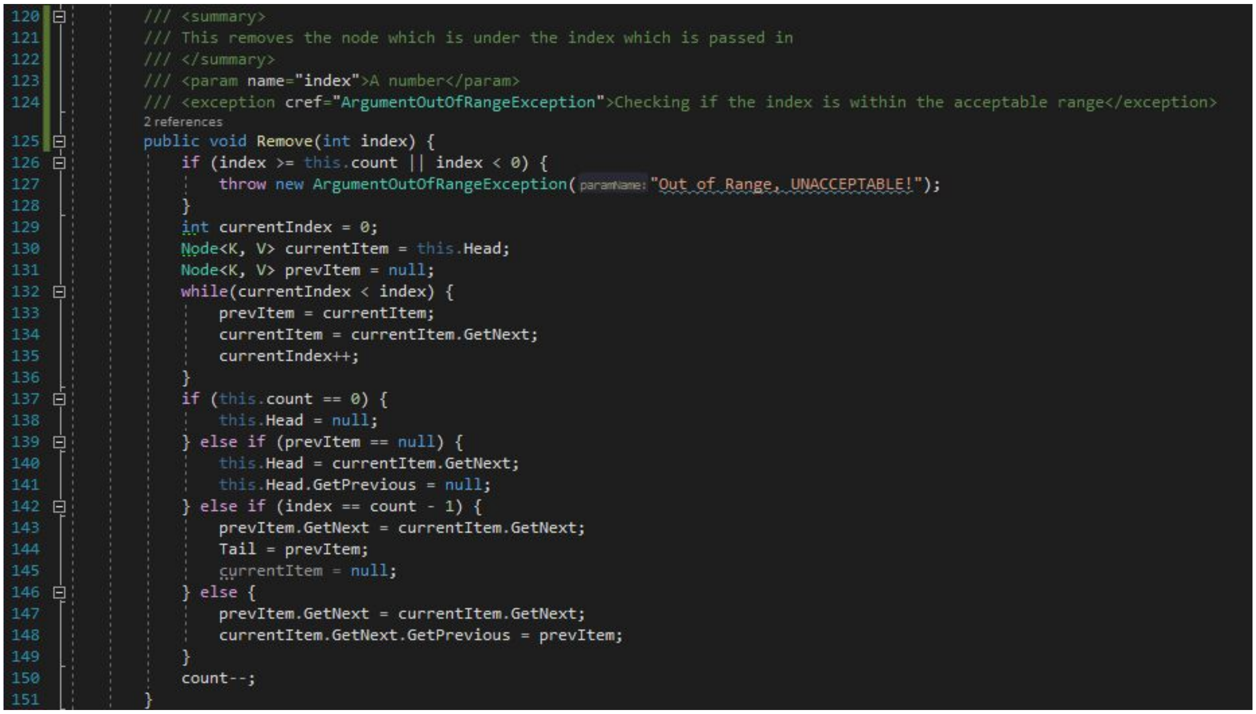


```
        /// <summary>
        /// This function adds a new favourites page to the dictionary but also the file responsible for holding all favourites
        /// Also checks for any duplicates
        /// </summary>
        /// <param name="title">The name of the webpage</param>
        /// <param name="URL">The address of the webpage</param>
        1 reference
        public string FavouritesAdd(string title, string URL) {
            if (URL == null) throw new ArgumentNullException(nameof(URL));
            if (favourites.ContainsKey(title)) {
                Console.WriteLine("DUPLICATE ERROR: Favourites already contains: {0}", title);
                string errorString = "ERROR: ";
                title += " Already in the favourites";
                errorString += title;
                return title;
            } else if (favourites.ContainsValue(URL)) {
                Console.WriteLine("DUPLICATE ERROR: Favourites already contains: {0}", URL);
                string errorString = "ERROR: ";
                URL += " Already in the favourites";
                errorString += URL;
                return errorString;
            }
            favourites.Insert(key: title, URL);
            string success = String.Format("Successfully added {0} to Favourites", title);
            StreamWriter sw = new StreamWriter(FavouritesFile, append: true);
            sw.WriteLine("{0}|{1}", title, URL);
            sw.Close();

            return success;
        }
```

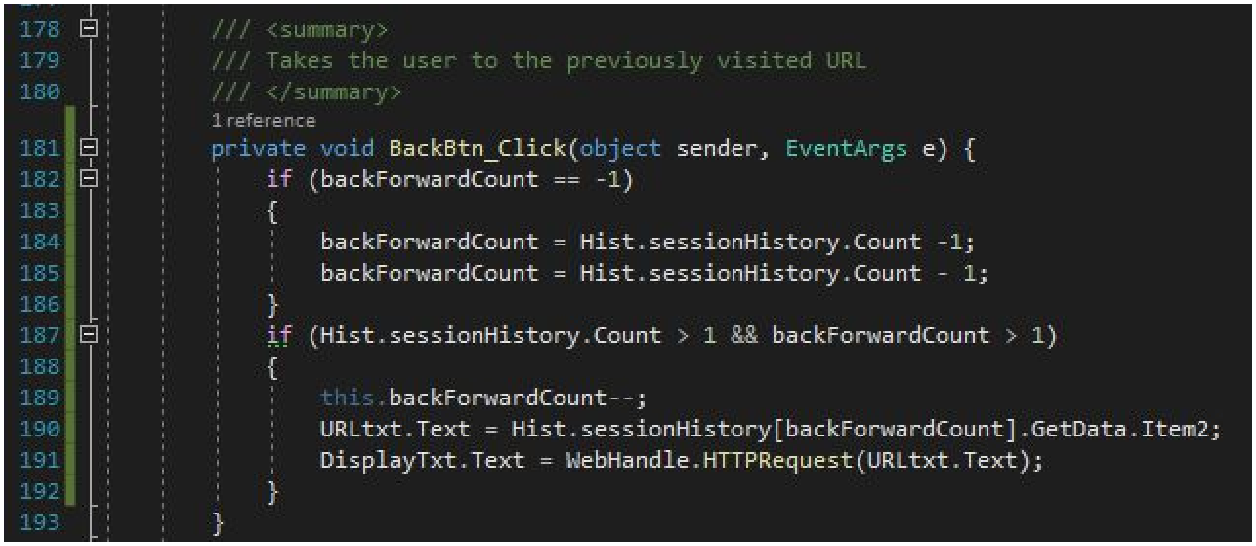
Another feature of the code is the add function of both Favourite/History. They're very similar in implementation with the main difference being the Key under which they operate in the Data Structure. The FavouritesAdd function takes 2 strings to populate the data Tuple in the Node class of the Doubly Linked List. It first checks if the favourites list already has the favourite that the user is trying to add to and if it already exists it throws and error.

However if the favourite is a new favourite it adds it to the Double Linked List and writes to the favourites file incase the user was to shutdown the application after triggering this code block but also to allow the user to maintain their favourite next time they come back.



```
120         /// <summary>
121         /// This removes the node which is under the index which is passed in
122         /// </summary>
123         /// <param name="index">A number</param>
124         /// <exception cref="ArgumentOutOfRangeException">Checking if the index is within the acceptable range</exception>
            2 references
125         public void Remove(int index) {
126             if (index >= this.count || index < 0) {
127                 throw new ArgumentOutOfRangeException(paramName: "Out of Range, UNACCEPTABLE!");
128             }
129             int currentIndex = 0;
130             Node<K, V> currentItem = this.Head;
131             Node<K, V> prevItem = null;
132             while(currentIndex < index) {
133                 prevItem = currentItem;
134                 currentItem = currentItem.GetNext;
135                 currentIndex++;
136             }
137             if (this.count == 0) {
138                 this.Head = null;
139             } else if (prevItem == null) {
140                 this.Head = currentItem.GetNext;
141                 this.Head.GetPrevious = null;
142             } else if (index == count - 1) {
143                 prevItem.GetNext = currentItem.GetNext;
144                 Tail = prevItem;
145                 currentItem = null;
146             } else {
147                 prevItem.GetNext = currentItem.GetNext;
148                 currentItem.GetNext.GetPrevious = prevItem;
149             }
150             count--;
151         }
```

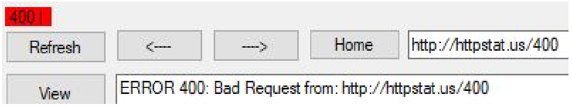
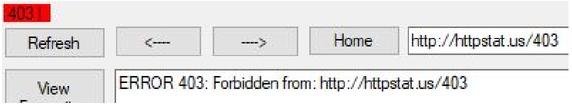
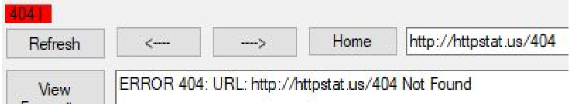
An interesting code block is the Remove function in the Double Linked List as it’s the only one implemented and its to remove the index. Basically what this does is it steps through the DLL with 2 nodes currentNode and prevNode (this allows us to handle finding the node we need to delete correctly with pointers). Then the main chunk of this function starts on line 132 which is the while loop. Basically this steps through the DLL 1 index at a time until the index was found and we just remove the pointers to the node we wish to delete. By doing this, the garbage collector is smart enough to realize that there is no access to that node anymore and it disposes of it correctly for us!

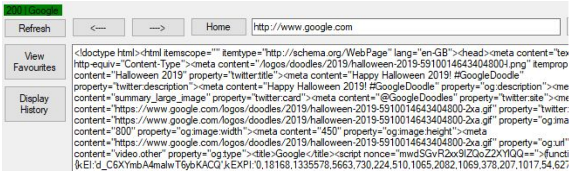


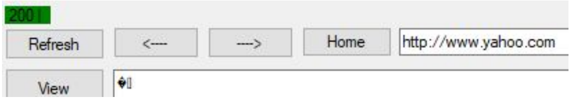
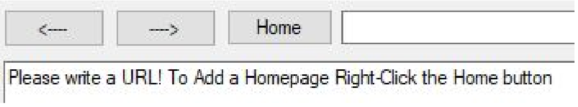
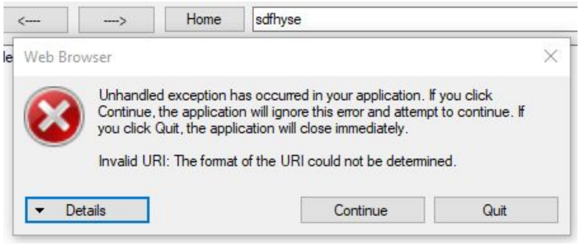
```
/// <summary>
/// Takes the user to the previously visited URL
/// </summary>
1 reference
private void BackBtn_Click(object sender, EventArgs e) {
    if (backForwardCount == -1)
    {
        backForwardCount = Hist.sessionHistory.Count -1;
        backForwardCount = Hist.sessionHistory.Count - 1;
    }
    if (Hist.sessionHistory.Count > 1 && backForwardCount > 1)
    {
        this.backForwardCount--;
        URLtxt.Text = Hist.sessionHistory[backForwardCount].GetData.Item2;
        DisplayTxt.Text = WebHandle.HTTPRequest(URLtxt.Text);
    }
}
```

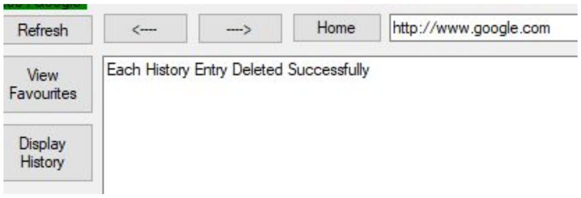
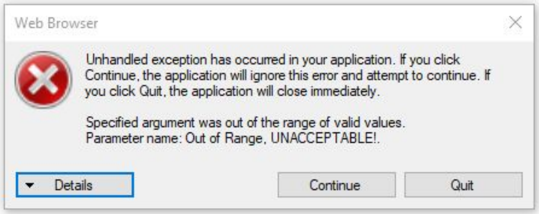
Finally the last snippet of code shows the backBtn_Click function which gets triggered on user click of the ← arrow. The backForwardCount is initially set to -1 so when the user triggers this function for the first time, that Count becomes useful to us as its now keeping track of how many times we can go back in terms of previous URLs visited. Using the indexer we retrieve the previous history page and display it for the user using URLtxt and DisplayTxt.

Testing

URL	Expected	Result	Notes
"http://httpstat.us/400"	ERROR 400: Bad Request	 http://httpstat.us/400 ERROR 400: Bad Request from: http://httpstat.us/400	Same as Expected
"http://httpstat.us/403"	ERROR 403: Forbidden	 http://httpstat.us/403 ERROR 403: Forbidden from: http://httpstat.us/403	Same as Expected
"http://httpstat.us/404"	ERROR 404: Not Found	 http://httpstat.us/404 ERROR 404: URL: http://httpstat.us/404 Not Found	Same as Expected

"google.com"	200 | Google + Display showing code correctly	 200 | Google
		Refresh
		<---
		--->
		Home
		View Favourites
		Display Historyhttp://www.google.com
		<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-GB"><head><meta content="tex
		http-equiv="Content-Type"><meta content="/logos/doodles/2019/halloween-2019-5910014643404800-l.png" itemprop
		content="Halloween 2019" property="twitter:title"><meta content="Happy Halloween 2019! #GoogleDoodle"
		property="twitter:description"><meta content="Happy Halloween 2019! #GoogleDoodle" property="og:description"><me
		content="summary_large_image" property="twitter:card"><meta content="@GoogleDoodles" property="twitter:site"><me
		content="https://www.google.com/logos/doodles/2019/halloween-2019-5910014643404800-2xa.gif" property="twitter:
		content="https://www.google.com/logos/doodles/2019/halloween-2019-5910014643404800-2xa.gif" property="og:ima
		content="800" property="og:image:width"><meta content="450" property="og:image:height"><meta
		content="https://www.google.com/logos/doodles/2019/halloween-2019-5910014643404800-2xa.gif" property="og:url"
		content="video.other" property="og:type"><title>Google</title><script nonce="mwdSGvR2xx9IZQoZ2XYlQQ==">(functi
		{kEI:'d_C6XYmbA4malwT6ybKACQ',kEXPI:'0,18168,1335578,5663,730,224,510,1065,2082,1069,378,207,1017,54,627
			Same as Expected

"yahoo.com"	200 | Yahoo + Display showing code correctly	 http://www.yahoo.com �▯	This could be a result of a different security protocol?
""	"Please write a URL! To Add a Homepage Right-Click the Home button"	 Please write a URL! To Add a Homepage Right-Click the Home button	Same as Expected. Status Code not reset tho.
"ahgtghae"	"Please write a URL! To Add a Homepage Right-Click the Home button"	 <--- ---> Home le Web Browser Details Continue Quitsdfhyse Unhandled exception has occurred in your application. If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will close immediately. Invalid URI: The format of the URI could not be determined.	Implementation breaks if a user doesn't type a domain at the end of string.
Clearing a single entry from History with more than 1 entry	Selected Entry disappears from listbox but also the DLL	Same as Expected.	The indexer in the DLL makes this very simple

Clearing a single entry from History with only 1 entry available	“Each History Entry Deleted Successfully”	 Refresh <---- ----> Home View Favourites Display History http://www.google.com Each History Entry Deleted Successfully	Calls the clear function. This is the same for favourites.
Clearing a single entry without selecting a item	“Each History Entry Deleted Successfully”	 Unhandled exception has occurred in your application. If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will close immediately. Specified argument was out of the range of valid values. Parameter name: Out of Range, UNACCEPTABLE!.	Missed implementation of Out Of Range exception. Writes a custom message but doesn’t handle exception within the code.

Conclusion

Overall, this coursework proved to be more difficult than expected at first, nevertheless, it also proved to be a really fun and interesting learning experience being able to work on examples that used the theory we learned in lectures to give a better understanding of the material.

The doubly linked list data structure was probably the hardest and most rewarding part of the development process as it managed to give me a deeper understanding of generics and indexers but also showed me that planning before implementing is crucial as if this were to be the data structure from the beginning of the development process, this coursework would’ve been slightly easier.

Another feature I am really proud of is the clear single entry from history/favourite DLL/files as this appeared to be a really hard task to do with the Dictionary but once I changed over to a generic Doubly Linked List, doing this was a breeze after a colleague pointed out that the ListBox displaying all the entries of the DLL have indexes, hence why the only remove function of the DLL is using an index as this is the only time I will have to remove 1 node from the DLL.