

Student Declaration of Authorship

Course code and name:	F21SC Industrial Programming
Type of assessment:	Individual
Coursework Title:	Developing a Simple Web Browser
Student Name:	Salman Ansari
Student ID Number:	H00410360

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.
- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the [University's website](#), and that I am aware of the penalties that I will face should I not adhere to the University Regulations.
- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on [Academic Integrity and Plagiarism](#)

Student Signature: *Salman Ansari*

Date: 24/10/2022

INDUSTRIAL PROGRAMMING

F21SC

Coursework 1

Developing a Simple Web Browser

Name: Salman Ansari

HW ID: H00410360

Index

S. no	Title	Pg. No.
1	Introduction	1
2	Requirement Checklist	1
3	Design Considerations	
	3.1. Classes	3
	3.2. GUI Design	3
	3.3. Data Structures	4
	3.4. Advanced Language Constructs	5
	3.5. Hardware and Software Specifications	5
4	User Guide	6
5	Developer Guide	9
6	Testing	11
7	Conclusion	11
8	Reflections on programming language and implementation	12
9	References	12

1. Introduction

The WebApp project is basically a Windows Form Application made using C# in Visual Studio 2022. The application provides the HTML code of the website whose URL is entered in the URL textbox. If the URL entered is incorrect, then it will return a message notifying there is an error in the URL. It also has multiple functionalities which mimic the working of an actual browser like history, favorites, reload, etc. [Refer Image 1]

I will explain the design principles, logic and the thinking process that I have used in the development of the WebApp application. I will also use it as a guide for others who want to learn how the program works easily. At the end of the report, I will comment on what I learned by working on the project.

2. Requirement Checklist

The following are achieved in the application:

- 1) Loading Home page URL / Edit Home page URL
- 2) Sending HTTP request / Fetching website HTML code
- 3) Add Tab / Remove Tab
- 4) View History List / Close History List / Clear History List
- 5) View Bookmark List / Close Bookmark List / Clear Bookmark List
- 6) Bulk Download (To check status code error)
- 7) [Forward Button / Back Button / Reload Button] to switch between tabs
- 8) Individual functionalities for History and Bookmark List (Delete, Edit, Open new tab)
- 9) Edit Bookmark (Update title and URL)
- 10) Extracting name from the <title> tag and displaying on title bar of application
- 11) Tooltips are added for all buttons for ease of understanding

3. Design Considerations

3.1 Classes

The project consists of 8 different classes, each with their specific functionality:

- 1) **Program Class**: This class contains the Main Method.
- 2) **Address Class**: This class basically deals with 3 properties: title, URL and timestamp.
- 3) **RWFileOperations Class**: This class deals with read / write from a file using StreamWriter and StreamReader.
- 4) **MainEngine Class**: This class deals with the history and bookmark list operations.
- 5) **AllTabs Class (Inside MainEngine.cs)**: This class inherits all the functionalities from TabPage to apply to all the tabs.
- 6) **MainStart Class**: This class consists of a form which includes buttons, textbox, etc. and other GUI functionalities.
- 7) **EditBookmark Class**: This class also consists of a form which has buttons and textbox to edit the bookmark title and URL.
- 8) **EditHomePage Class**: This class has a form which has button and textbox to update the home page URL.

3.2. GUI Design

The WebApp application is made using Windows Form in C# using Visual Studio 2022 IDE. The application GUI is very similar to a Web browser like Google Chrome. It consists of multiple buttons, textbox, menustrip, panels, richtextbox, etc. all on a form. [Refer Image 1]

Icons are used for the basic functionality like Back, Forward, Reload, Home and Search buttons. The GUI is very simple and easy to understand. There are no shortcuts associated with any functionalities. Only the 'Enter' key is used to search with the entered URL. The same can be achieved by pressing the Search button. [Refer Image 1]

The interface of the application consists of a Back button, Forward button, Reload button and a Search Button. It also has a menustrip which consists of various menuitems like Edit home page URL, View history, Close history, Clear history, View bookmark, Close bookmark, Clear bookmark, New tab, Close tab, Bookmark page, Bulk download and Quit application. [Refer Image 1]

Functionalities:

- 1) **Back button:** This button will load the previous URL and display the HTML code.
- 2) **Forward button:** This button will load the URL from which the user has returned from.
- 3) **Reload Button:** This button will reload the current URL.
- 4) **Search Button:** This button will search with the URL entered in the URL textbox. The same works when 'Enter' key is pressed on the keyboard.
- 5) **Edit Homepage URL:** This menuitem will open a form to edit the homepage URL.
- 6) **View History:** This menuitem will open the history listbox. (Disabled when history listbox is already opened)
- 7) **Close History:** This menuitem will close the history listbox. (Disabled when history listbox is already closed)
- 8) **Clear History:** This menuitem will clear the history listbox. (Disabled when history listbox is already cleared)
- 9) **View Bookmark:** This menuitem will open the bookmark listbox. (Disabled when bookmark listbox is already opened)
- 10) **Close Bookmark:** This menuitem will close the bookmark listbox. (Disabled when bookmark listbox is already closed)
- 11) **Clear Bookmark:** This menuitem will clear the bookmark listbox. (Disabled when bookmark listbox is already cleared)
- 12) **New Tab:** This menuitem will open a new tab.
- 13) **Close Tab:** This menuitem will close a tab. (Disabled when only one tab is opened)
- 14) **Bookmark Page:** This menuitem will add the URL in bookmarks list.
- 15) **Bulk Download:** This menuitem will open a File Explorer from where the user has to select a file containing URLs which will open a new tab and load the URLs and display various error codes.
- 16) **Quit Button:** This menuitem will close the application.

Bookmarks and History Listbox will display the URLs fetched from Favorite.txt and History.txt respectively. Upon clicking any link in both the listboxes, a small panel will open with Delete and Open in new tab for History list and Edit, Delete and Open in new tab for Bookmarks List. The edit option for Bookmarks List will open another form where the user can update the existing title and URL of the selected URL from the Bookmarks List.

3.3. Data Structures

The Data structures that I have used in building this application are Dictionary and List.

History

For history, I have chosen a list as history is always stored in order. The order is based on timestamp. Also, a dictionary is not required for history as we need not worry about multiple entries of the same URL in the history list. We maintain a “History.txt” file where all URLs are stored based on the time they were searched. Whenever the user starts the application, a method named “LoadTabWithUserHistory” will read the content from “History.txt file” and load it in the history listbox. Whenever a HTTP request is sent, the URL is pushed into the textfile and inserted in the first position of List.

Bookmarks

For Bookmarks, I have chosen a dictionary to store bookmarks. Dictionary is a better option here as I have decided that only unique URLs are to be stored in bookmarks list. Every time a URL is to be pushed in the list; it always checks whether the URL is already present in the list. If it is already present, then a messagebox pops up informing that the URL already exists. If it is not present, it will be pushed into the list. We can also add, edit or delete a URL. The pros of using a dictionary is that it takes only $O(1)$ [constant time complexity] for adding, deleting or editing a URL in the list. The cons of using this is that every time the app is started, the entire bookmark list is loaded into the listbox and ordered in ascending order by time.

Tabs

For “AllTabs” class, which inherits “TabPage”, I have used a list here as well to store the URLs every time the user sends a HTTP request. This makes it easy for different tabs to move to previous or next page as each tab will have its own list. We have also used an index for each list which will point to the current URL so that the user can easily switch between pages.

3.4 Advanced Language Constructs

Delegate and Generics

I have implemented these concepts to help reduce the duplicity of the code in the “RWFileOperations” class. In the class, I have used different methods to show how to transfer each line to list and dictionary.

Events

I have used events in my “MainEngine” class. Whenever we add a URL to history list, it calls the “AllTabs” class instance. But only the “MainEngine” class knows which URL is present in the listbox of the “Home” class. So, the “MainEngine” class tells the “Home” class to add the URL into the history listbox whenever there is a URL added in the “MainEngine” class. The same also works for the bookmarks list too.

LINQ

I have made use of Language Integrated Query every time when loading bookmarks to sort each URL in order of time. Any change made in the URL and title inside bookmarks list will not reflect in the tab title. It will match the index with the Booksmars.txt file and fetch the title and URL from there.

3.5. Hardware and Software Specifications

Visual Studio Community 2022 (17.3.6)

Windows OS version (10.0.22000)

C# Language Version (10.0)

Net SDK 6.0.402

Net Framework 6.0.10

4. User Guide

This is the user interface of the WebApp.

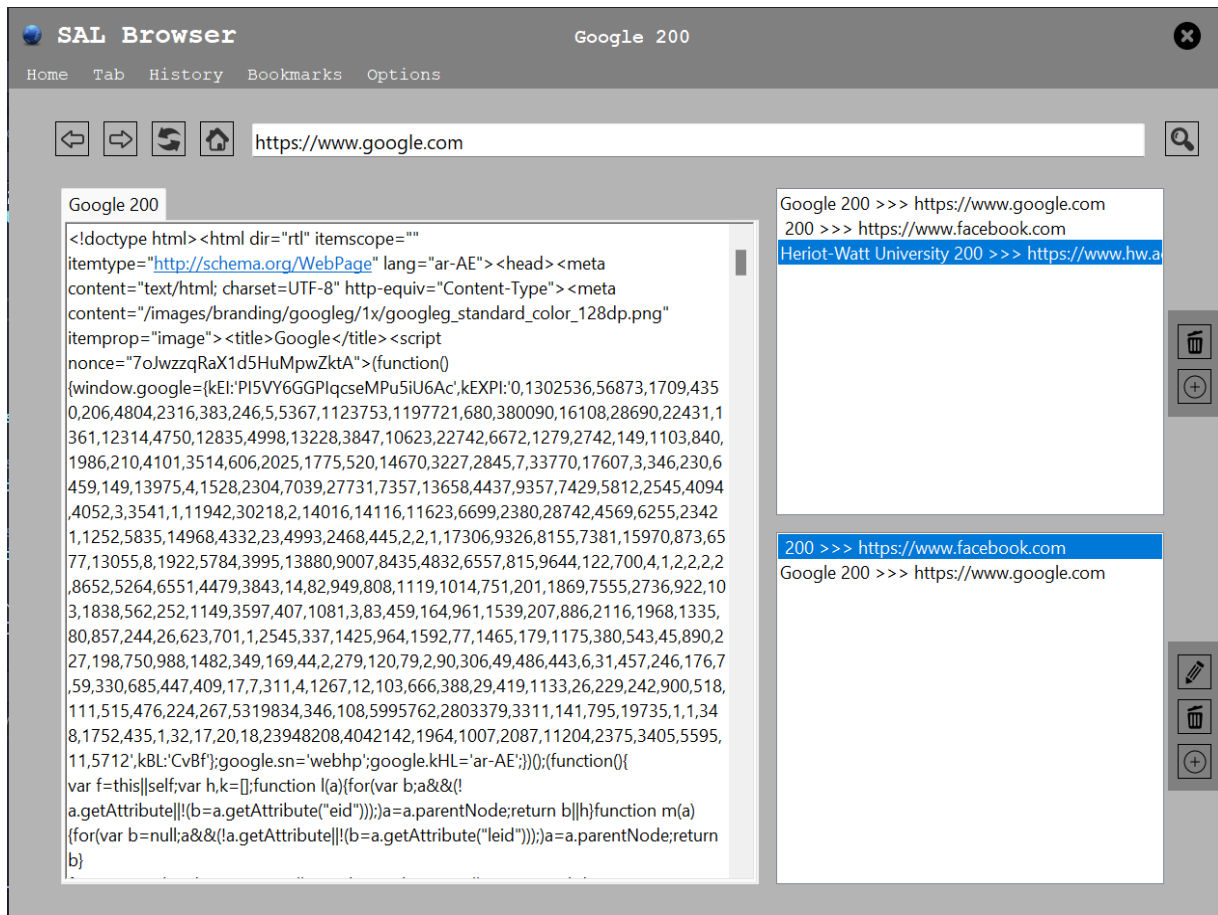


Image 1: Home Screen of the application

Starting from top, there is a Back button, Forward button, Reload button, Home button, Search bar and a Search button. The textbox is where the user will enter the URL and press the “Enter” key on keyboard or click on the Search button, if the URL is correct then the HTML code of the same will display in the display area Panel. If the URL is incorrect, an error message will be displayed. [Refer Image 1]

When we have put in multiple URLs, we can move back to the previous URL or to the next URL by clicking on the forward and back button. Also, if you are on the first page, then the forward and backward button will not be visible.

The Reload button will reload the current URL.

The Homepage button will load the homepage URL which is stored in “Homepage.txt” when the button is clicked.

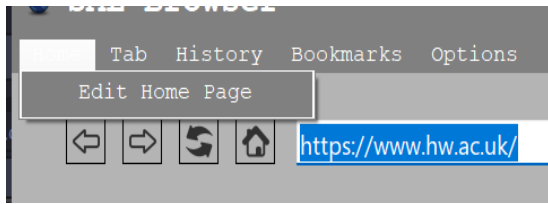


Image 2: HomeMenuItem -> Edit Homepage

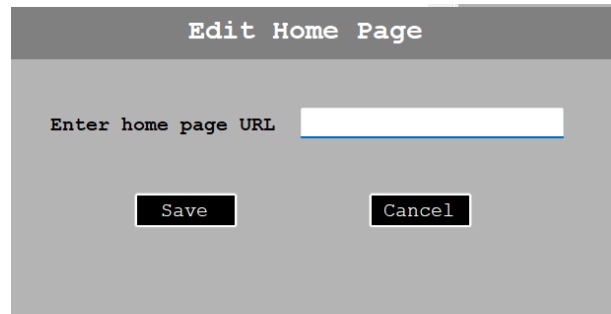


Image 3: Homepage Edit form

Furthermore, we can change the homepage URL, by clicking on the Edit homepage menu item which will open the Edit homepage form. Here we can set the homepage URL. It will override the existing URL in the “Homepage.txt”. [Refer Image 2 & 3]

A new tab can be added by accessing the Tabs menustrip and clicking on the New tab. A tab can be closed by clicking on the Close tab. [Refer Image 4]

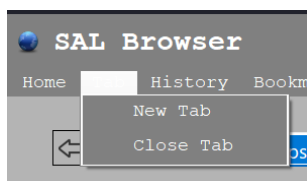


Image 4: Open/Close Tab

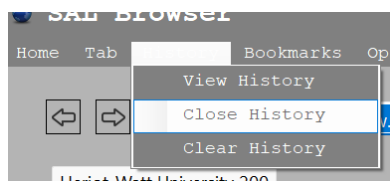


Image 5: History Menuitem

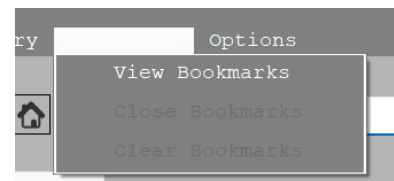


Image 6: Bookmark Menuitem

The history can be accessed by clicking on the History menustrip. The user will be prompted with three options: View history, Close history, and Clear history. The user can view/close the history by clicking on View/Close history menuitem [Refer Image 5]. The user can also clear history if he wishes to do so. The bookmarks can be accessed in the same way by clicking on the Bookmarks menuitem. [Refer Image 7]

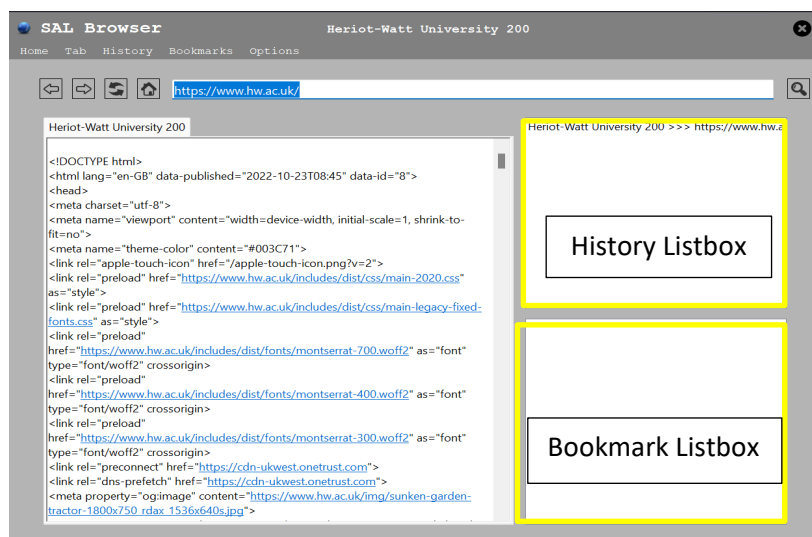
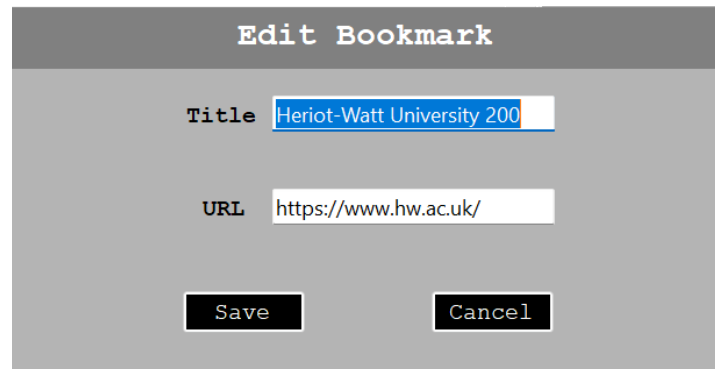


Image 7: History and Bookmark Listbox

A bookmark in the Bookmark list can be edited by clicking on the URL to be edited. A small panel appears on the right of the listbox with 3 options to Edit, Delete or Open the URL in new tab. When the user clicks on the Edit option, a form appears where the user can change the title and URL of the selected index. [Refer Image 8]



The image shows a dialog box titled "Edit Bookmark". It contains two text input fields: "Title" with the value "Heriot-Watt University 200" and "URL" with the value "https://www.hw.ac.uk/". At the bottom, there are two buttons: "Save" and "Cancel".

Image 8: History Panel

A similar panel appears when the user clicks on any URL inside the history listbox with 2 options: Delete and Open the URL. [Refer Image 9, 10]

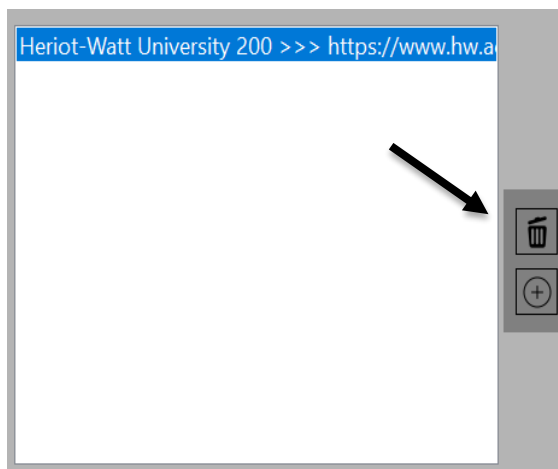


Image 9: History Panel

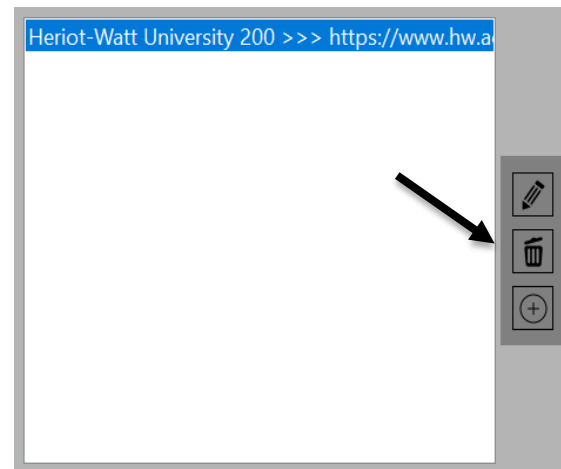


Image 10: Bookmark Panel

Finally, there is a Bulk download menuitem in Options. Clicking on the bulk download will open a file explorer from where the user can select a text file which contains incorrect URLs. A new tab will open displaying the different status code of each URL in the text file.

The title of every tab will be displayed on the title bar of the application and is dynamically updated when switched between tabs. [Refer Image 11]

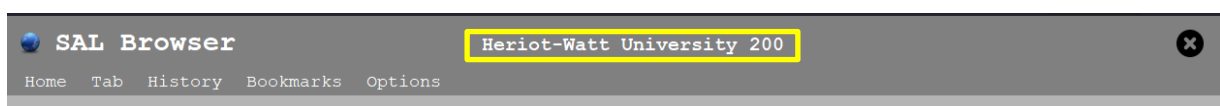


Image 11: Title bar

5. Developer Guide

Address class deals with basically three different properties: [Refer Image 12]

- 1) Title of the webpage
- 2) URL of the webpage
- 3) Timestamp

```
public Address(String title, String url, DateTime time)
{
    this.time = time;
    this.title = title;
    this.url = url;
}
```

Image 12: Address Class

AllTabs class is inherited from the TabPage class. Two methods are overloaded in this class. [Refer Image 13]

```
private MainEngine appHandler;

2 references
public MyTabPage(ref MainEngine appHandler)...

2 references
public MyTabPage(ref MainEngine appHandler, String url)...
```

Image 13: Method Overloading

ForwardAddressRequest method in the AllTabs class is an important class in the application as it deals with forwarding HTTP requests. I have used try and catch inside this method to tackle any exception that may occur. Whenever the user puts a URL link in textbox and presses “Enter” key on the keyboard, The ForwardAddressRequest method is called, and it passes the HTTP request for that URL. If the URL is correct, the server responds and the HTML code for the website is displayed, and title of the website is shown in the title bar of the application. If an error occurs, it will show the error message. [Refer Image 14]

```
public void ForwardAddressRequest(String url)
{
    // try and catch is used to detect any errors that may pop
    try
    {
        HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
        request.KeepAlive = false;
        request.Method = "Get";
        HttpWebResponse response = (HttpWebResponse)request.GetResponse();
        using StreamReader readStream = new StreamReader(response.GetResponseStream());
        String temp = readStream.ReadToEnd();
        this.Text = PullURLHeader(temp).Trim() + " " + Convert.ToInt32(response.StatusCode);
        address_list.Add(url);
        pointer_list = address_list.Count - 1;
        displayArea.Text = temp;
        appHandler.addToHistory(new Address(Text, url, DateTime.Now));
        title_header = this.Text;
    }
}
```

Image 14: Forwarding HTTP Request

RWFileOperations class deals with the pushing and pulling content from text files namely History.txt and Bookmarks.txt. We have used StreamReader and StreamWriter built in classes to read/write data. [Refer Image 15 & 16]

```
public void FileUpdate(String text, String fileName, bool append)
{
    try
    {
        StreamWriter streamwriter = new StreamWriter(fileName, append);
        streamwriter.WriteLine(text);
        streamwriter.Close();
    }
    catch (Exception e)
    {
        Console.WriteLine(e+" Failed to load "+fileName);
        throw;
    }
}
```

Image 15: Writing Content from File

```
public void FetchFromFile<T>(ref T t, String fileName, SpecificStructure<T> structure)
{
    String line;
    try
    {
        using (StreamReader streamreader = new StreamReader(fileName))
        {
            while ((line = streamreader.ReadLine()) != null)
            {
                //because it is reference, so we can change the original data structure
                structure(ref t, temp_operations, line);
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("The file ( "+fileName+" ) could not be read:");
        Console.WriteLine(e.Message);
    }
}
```

Image 16: Reading Content from File

MainStart Class contains the buttons and forms all linked together thereby making it easier for the developer to understand the code.

6. Testing

Bugs:

- 1) The Clear bookmark menuitem is disabled even though the bookmark list not clear.
- 2) The Close tab menuitem is not disabled even though there is only one tab open.
- 3) The Forward and Back button on the form appear on startup even though that is the only tab open

All other functionalities are functioning properly as they are supposed to function.

Address Textbox:

- 1) Whenever the user enters a URL in the textbox, the URL is validated using Regular Expressions.
- 2) If there is an issue in the URL, an error message will show up in the Tab instead of the HTML code of the website.

Bulk Download:

- 1) If the user tries to open any other file format except .txt format, a MessageBox will pop up showing the error.
- 2) A MessageBox will pop up even if the file is empty.

7. Conclusion

The development of this project has helped me learn some concepts of C# that were unknown to me before. Even though C# looks quite like Java, both have their unique distinctiveness. This project helped me grow stronger in my concepts, also at the same time learn new features. If I had more time, I would move forward with this project and add functionality where the user could switch between HTML code layout and displaying the actual webpage GUI in my application.

8. Reflections on programming language and implementation

I've learned a lot while building this application. I found LINQ and its properties very useful. It is easy to setup and handles data more flexibly than SQL.

One major limitation to this application is that the MainStart form has a lot of relationships within components. For e.g.: When the history and bookmark list are open, and the user clicks on the bulk download menuitem, both the history and bookmark list along with their individual panels are closed. Decoupling is one solution, in which first we find the relationships between components. If one component relies on many components, then grouping them is the best possible solution, but it is difficult to achieve.

I personally believe system language like C# is a better option for this application as it deals with low level functions like passing HTTP requests than using scripting languages, even though they are more widely supported on all platforms.

The program runs smoothly on the Windows operating system.

9. References

- <https://learn.microsoft.com/en-us/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022>
- <https://learn.microsoft.com/en-us/dotnet/desktop/winforms/input-mouse/events?view=netdesktop-6.0>
- <https://learn.microsoft.com/en-us/dotnet/api/system.windows.forms.control.keypress?view=windowsdesktop-6.0>
- <https://zetcode.com/csharp/getpostrequest/>
- <https://codingshiksha.com/c-2/how-to-deploy-net-c-windows-forms-or-console-app-on-desktop-with-setup-exe-file-using-clickonce-full-tutorial-for-beginners/>
- <https://learn.microsoft.com/en-us/dotnet/api/system.io.streamreader?view=net-7.0>
- <https://learn.microsoft.com/en-us/dotnet/api/system.io.streamwriter?view=net-7.0>