10/12/2025

F21SC coursework 1

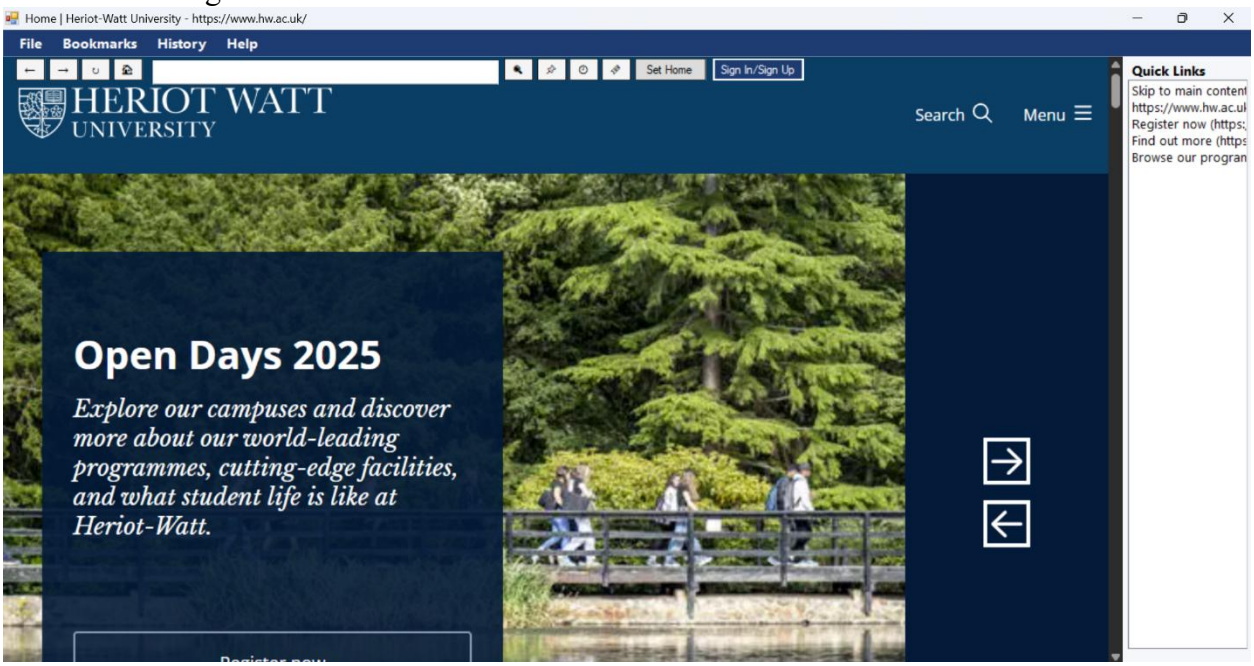
Heriot-Watt University

Chakravarti, Aoishi
H00411515

I. Introduction

This report serves as a guide and checklist for the creation of the application, **F21SC_webbrowser**. This application, as per the requirements, works entirely on C# and .NET Framework. For creating the application, I used the **WinForms app template** found in **Visual Studio**. At the beginning of development, my knowledge on C# was set at the basics. Given my basic knowledge, I assumed this project to be challenging. But by the end, I was equipped with knowledge on most C# commands, which I learnt amidst development by searching the functions that work best for each requirement.

Based on the requirements, the application is merely a simulation that imitates browser behaviour. With that logic in mind, the .zip file contains all the cs files for each of the requirements, with their functions explained in the comments. Although the project required me to build a simple browser, it turned into a difficult yet interesting endeavour that taught me a lot of C# concepts along the way. Accompanying this report are screenshots showcasing the functionality of the web browser, such as the one given below:



This screenshot, for instance, was taken during the process of building the homepage. As given in the image above, the current default it is https://hw.ac.uk. In the later pages, I will show and further explain the requirements that were met alongwith the functions used.

II. Requirements checklist

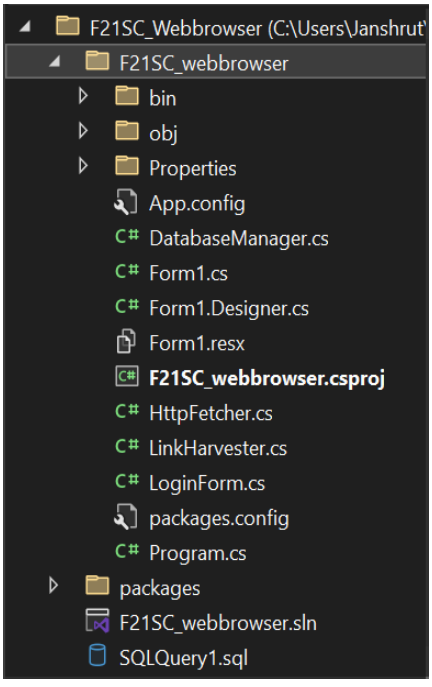
Sno.	Requirements	Were they met? (Yes/No)
1	Sending HTTP request messages for URLs typed by the user	Yes
2	Receiving HTTP response messages and the contents of the messages in the form of 200 OK HTTP response status code, alongwith the following response status error codes and their corresponding error messages: – 400 Bad Request – 403 Forbidden – 404 Not Found	Yes
3	Display HTTP response status code and the title of the web page at top of the browser's main window. Reload the current page by sending another HTTP request for the current web page, and display the contents of the page together with the title and the HTTP response status code as specified above	Yes
4	The user should be able to create and edit a home page URL. The Home page URL should be loaded on the browser's startup, and it should be initialised with "https://hw.ac.uk"	Yes
5	User should be able to add a URL for a web page requested to a list of bookmarks. The user should also be able to associate a name with each bookmark URL. Support for bookmark items modification and deletion is required. The user should be able to request a bookmark by clicking its name on the bookmarks list.	Yes

	On the browser's start up, the bookmarks list should be loaded to the browser	
6	The browser should maintain history, i.e., a list of URLs corresponding to the web pages requested by the user. Users should be able to navigate backwards and forward through their browsing sequence, and jump to a specific page by clicking on its entry in the History list. On startup, the browser should load the most recent history. Additionally, the history should eliminate consecutive duplicate entries to reduce clutter, while preserving duplicates that are separated by other pages. For example, the sequence A → B → B → C → B → B → D would be stored as A → B → C → B → D	Yes
7	The browser should support a feature that, upon loading a web page, automatically harvests the first five URLs found within the page content. These URLs should be displayed as a clickable list in a dedicated panel at the side of the webpage, allowing users to quickly access links available on the webpage.	Yes
8	A simple GUI should be provided to perform the operations discussed above. The GUI for the web browser should support the following: – Using the GUI, the user should be able to perform the operations discussed above. – Make use of menus (with appropriate shortcut keys) as	Yes

	well as buttons to increase accessibility	
9	Implement database storage and store information such as the homepage, bookmarks, and browsing history in a relational database of your choice. All database operations should be performed using the LINQ library.	Yes
10	Add multi-user functionality and extend the system to support multiple users, where each user has their own personalized homepage, bookmarks, and browsing history. This feature should also be implemented through the database	Yes

III. Design Considerations

The application, F21SC_webbrowser, is designed to be modular while maintaining its user-centric functionality. The project is structured to separate the user interface from the underlying logic, ensuring clarity and scalability.



1. Architecture Overview

Since the application follows a modular design, each feature associated with the requirements are coded in separate .cs files. As given in the image, major components include Homepage management, bookmarks, browsing history, link harvesting, etc. These modules interact through well-defined methods and event handlers to initiate processes without affecting others.

2. User Interface Design

The GUI (See Page 1), built using **WinForms** found in **Visual** Studio, was coded into the file, **Form1.cs**, provides an intuitive and accessible interface:

- The main window contains a URL bar, navigation buttons (Back, Forward, Reload, Home), and panels for bookmarks, history, and harvested links.
- Menus with shortcut keys to enhance usability (e.g., Ctrl+H for Home, Ctrl+B Bookmarks).

- A button for customizing the homepage as per the user's wishes.

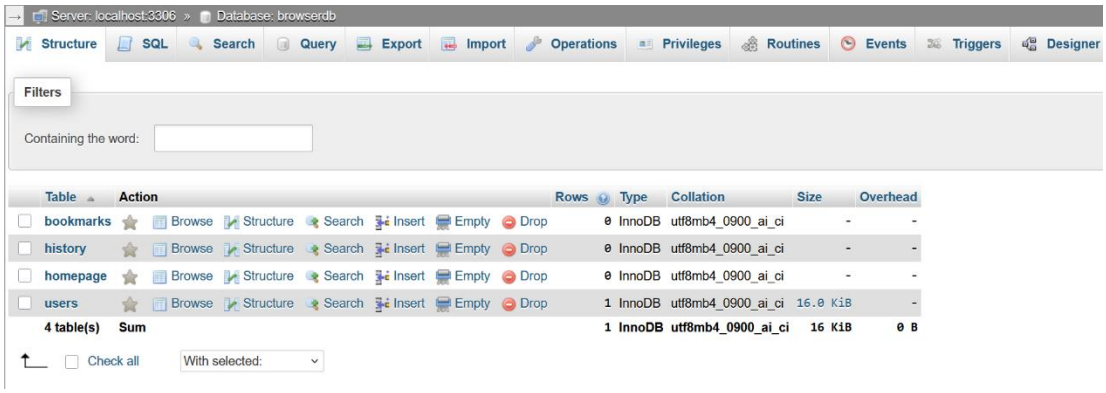
3. HTTP Communication

The browser also handles web requests and errors. Let's say the user enters a non-existent URL, then the application sends an HTTP request and receives the response, extracting the **status code, title, and HTML content**. Standard error codes (404,403,400) are detected and presented to the user. Since this an innate feature of any application, it has been hard coded into Form1.cs instead of a separate .cs file.

```
1 reference
private async void WebView2_NavigationCompleted(object sender, CoreWebView2NavigationCompletedEventArgs e)
{
    string url = webView2.CoreWebView2.Source;

    if (e.IsSuccess)
    {
        // Navigate successful → treat as 200 OK
        if (!string.IsNullOrEmpty(url) && !isNavigating)
        {
            dbm.AddHistoryForCurrentUser(webView2.CoreWebView2.DocumentTitle, url);
        }

        // Update window title
        string title = webView2.CoreWebView2.DocumentTitle ?? "Untitled Page";
        this.Text = $"{title} - {url} (200 OK)";

        // Harvest links
        LinkHarvester harvester = new LinkHarvester();
        var links = await harvester.GetFiveLinksAsync(webView2.CoreWebView2);
        DisplayHarvestedLinks(links);
    }
    else
    {
        // Handle navigation failure
        this.Text = "...";
        string errorMessage = "An error occurred while loading the page.";
        string errorCode = "Navigation Failed";

        if (e.WebErrorStatus == CoreWebView2WebErrorStatus.HostNameNotResolved)
        {
            errorCode = "404 Not Found";
            errorMessage = "The server could not be found. Please check the URL and try again.";
        }
        else if (e.WebErrorStatus == CoreWebView2WebErrorStatus.ConnectionAborted)
        {
            errorCode = "400 Bad Request";
            errorMessage = "Connection was aborted. The server may be down or the URL is incorrect.";
        }
        else if (e.WebErrorStatus == CoreWebView2WebErrorStatus.Unknown)
        {
            errorCode = "404 Not Found";
            errorMessage = "The domain name could not be resolved or the server is unreachable.";
        }

        // Update the window title to show the error code (Not 200 OK)
        this.Text = $"{errorCode} - {url}";
    }
}
```

4. Data Management

At the beginning of development, the application depended on XML to store data. By the end, the application was connected to **phpMyAdmin**, an open-source and downloadable relational DBMS software. As a result, persistent data, including homepage, bookmarks, and history are stored in a relational database 'browserdb', and accessed via LINQ queries. On startup, the browser loads the user's personalized data, ensuring continuity between sessions. As given in the image, four tables were made for bookmarks, history, users, and the users' customized homepage. These tables are



set with a foreign key relationship with the column ‘username’ from table ‘users’. This helps in storing data user-wise.

5. Multi-user Support

   The system supports multiple user profiles, each with personalized homepage settings, bookmarks, and browsing history. Logging in dynamically updates the interface to reflect the active user’s data.

6. History Management

   Browsing history is carefully maintained to avoid clutter: consecutive duplicate entries are collapsed, while non-consecutive entries are preserved. Users can navigate backward and forward through the history or jump directly to a previous page via the history panel. Also, the history panel comes with a button that the user can use to erase their browsing history.

7. Link Harvesting Feature

   Upon loading a webpage, the first five hyperlinks within the content are automatically extracted and displayed on a side panel. Users can click these links to quickly navigate to related pages/sections. HTML content is parsed using C# techniques to ensure accurate link extraction.

8. Error Handling and Robustness

   The application is designed to handle invalid URLS, network errors, and database issues. Users are notified of errors without crashing the browser, providing a reliable and smooth experience.

9. Extensibility

   The modular design allows future enhancements, like tabbed browsing or file downloads, all to be added with minimal changes to existing code. The separation of GUI and backend logic support maintaining and scalability.

IV. Developer Guide

I have named the application as ‘ac2105 Explorer’. The application has been designed with clarity, modularity, and maintenance in mind, making it easier for other developers to understand and extend its functionality.

1. Code Organization and Classes

The code is organized into four .cs files, each implementing a specific or a collection of features. Most of the work has been done in the main file, ‘Form1.cs’, so the methods are commented to display their function:

a. Form1.cs

- Responsible for the primary GUI of the application, including the URL bar, navigation buttons, panels for bookmarks, history, and harvested links.

- Handles user interactions and delegates functionality to other files and classes.

b. 'WebView2_NavigationCompleted' class
   - Handles Web navigation and HTTP communication, including sending requests and receiving responses.
   - Found in Form1.cs, it parses the response for status codes, page titles, and HTML content.
   - Manages error detection and reporting (e.g., 400, 403, 404 errors).

c. 'button8_Click' and 'button10_Click' classes
   - Manages the homepage URL, including creation, editing, and loading at startup.

d. 'button5_Click' and 'button6_Click' classes
   - Handles bookmarks storage, modification, deletion, and retrieval.
   - Link bookmarks to user profiles to support multi-user functionality.

e. 'button7_Click' class
   - Maintains browsing history while eliminating consecutive duplicates.
   - Supports backward and forward navigation and loading pages from history.

f. LinkHarvester.cs
   - Parses loaded HTML content to extract the first five hyperlinks.
   - Displays the harvested links in a dedicated side panel for quick navigation.

g. DatabaseManager.cs
   - Implements all database operations, including saving and retrieving, homepage, bookmarks, and browsing history.
   - Uses LINQ to perform queries on the relational database and is the main backend function.

h. HttpFetcher.cs
   - Contains the HttpClient instance used for sending and receiving HTTP requests that are identified by the URI.
   - It also uses an asynchronous method for sending GET requests and reading the response as a string.

2. HTTP Communication

- HTTP requests and responses are handled by the C#'s 'HttpClient' class.
- Responses are parsed to extract status codes, titles, and HTML content.
- Standard HTTP errors (400,403,404) are identified and displayed to the user.

3. Persistent Browser Settings

- Homepage, bookmarks, and browsing history are persisted using **a relational database** built in phpMyAdmin.

- The **LINQ library** is used for all CRUD operations, ensuring that data is efficiently queried, updated, and maintained across sessions.
- Multi-user support is implemented by linking all stored data to a **unique username**, allowing each user to have personalized settings.

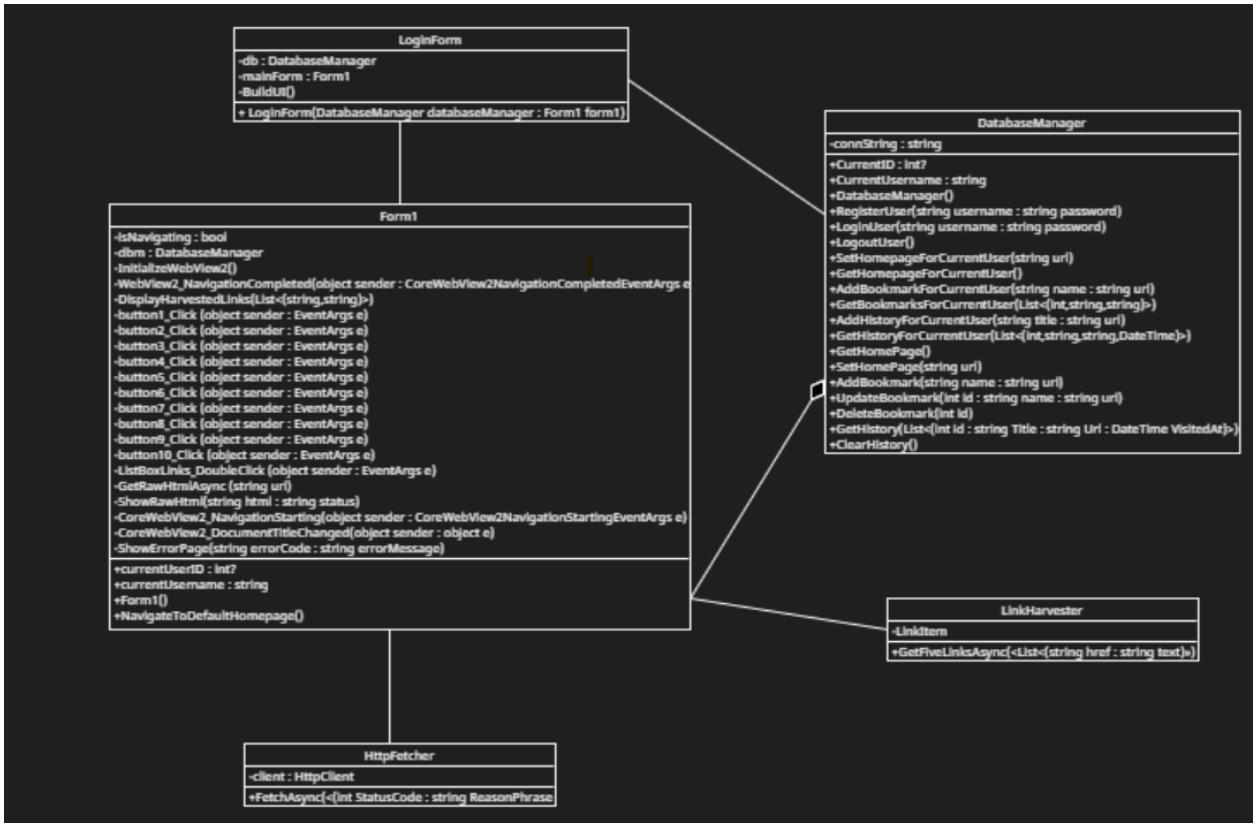
4. Graphical User Interface

- The GUI is created using **Windows Forms (WinForms)**.
- Controls such as menus, buttons, panels, and text boxes are used to provide an interactive interface.
- Event handlers in Form1.cs connect the GUI actions to backend functionality, maintaining separation between interface logic.
- Shortcut keys and resizable panels improve usability and accessibility.

5. Extensibility Notes for Developers

- The modular structure allows new features, such as tabbed browsing, download manager, or advanced link extraction, to be added with minimal impact on existing code.
- GUI and backend logic are separated to make it easy to update one without affecting the other.
- All core functionality is contained is dedicated classes, providing clear points of extension for future development.

6. Class Diagram



7. Libraries / Technologies Used

- System.Net.Http: Use in HttpFetcher.cs for utilizing the **HttpClient and HttpResponseMessage classes**. This library provides methods for sending HTTP requests and receiving HTTP responses from web servers, allowing one to obtain the website's raw HTML code.
- System.Data.SqlClient: Used in DatabaseManger.cs. Provides connectivity to a SQL Server database for storing persistent data.
- System.Windows.Forms: The GUI is built in WinForms.
- Microsoft.Web.WebView2: Provides a modern web browser engine (based on Microsoft Edge/Chromium) to display web content inside the form.

V. Conclusion

Overall, this project has been a fascinating endeavour. The browser successfully demonstrated the development of a functional web browser application built solely using C# and Windows Forms. I am mostly proud of how the GUI turned out, along with how the application retrieves the raw HTML codes. Working with C# might have started out with some difficulty but by the end I truly enjoyed using it. My wish was to make the Form1.cs less crowded but otherwise everything turned out well.

VI. References

- *W3schools.com* (no date) *W3Schools Online Web Tutorials*. Available at: https://www.w3schools.com/ (Accessed: 22 October 2025).
- BillWagner (no date) *C# guide - .net managed language*, *C# Guide - .NET managed language | Microsoft Learn*. Available at: https://learn.microsoft.com/en-us/dotnet/csharp/ (Accessed: 22 October 2025).