Coursework 1 — Developing a Simple Web Browser

1. Overview

The aim of this coursework is to develop a simple web browser in C#.

This is an **individual project**, and you will have to submit your own, original solution for this coursework specification (please see academic misconduct guidelines), consisting of a report, **the source code, an executable, and an in-person demo** showcasing all implemented features.

The **learning objective** of this coursework is for students to develop proficiency in advanced programming concepts, and to apply these programming skills to a concrete application of moderate size. Design choices regarding tools and libraries chosen for the implementation need to be justified in the accompanying report.

This coursework will develop personal abilities in autonomous problem analysis, the usage of a modern object-oriented language for system programming, and deepen the understanding of integrating components on a Windows system.

2. Lab Environment

For this coursework you should use Visual Studio C# 2022, which will be the reference platform for testing. You can obtain a full version through Azure for Education using your Heriot-Watt student credentials.

For the GUI component, use a library of your choice. Under Visual Studio, the Windows Forms libraries are recommended. Alternatively, you can use Visual Studio Code for developing your application. You need to be aware that GUI development will differ in this environment, and that the libraries available for this platform are likely to be a subset of the libraries for Visual Studio. For the GUI component under Mono, the Gtk# libraries are recommended. Of course, these recommendations are just recommedations. Treat them as a starting point in your exploration of which library to use.

If you develop the software in a different environment, you must make sure that the code is compatible with the above software. Beware of incompatibilities due to different versions of the IDE or compiler that you use. It is recommended that you can run the final application on a Windows lab machine.

For technical HOWTOs about accessing software of relevance for this course, see the technical HOWTOs and the resources available on the course's Canvas page.

The submission of the final software must be in form of both *complete sources* as well as a *standalone executable* that can be installed and executed on any Windows 11 (with .NET Core) system. If external tools are used, they need to be included in this executable. Any dependencies on tools or external software must be clearly marked in the report.

3. Developing a Simple Web Browser

The web browser must provide the following functionality:

- **Sending HTTP request messages** for URLs typed by the user.
- **Receiving HTTP response messages** and display the contents of the messages on the interface. Note that you are only required to display the **raw HTML** code returned to the web browser from the web server (HTML parsing and graphical display are not required). Furthermore, display the return code from the HTTP request. In addition to the `200 OK` HTTP response status code, the

following HTTP response status error codes and the display of their corresponding error messages should be supported:

- `400 Bad Request`
- `403 Forbidden`
- `404 Not Found`

To test these error codes, look up web pages like `https://savanttools.com/test-http-status-codes`.

- **Display** the HTTP response status code and (if applicable) the title of the web page at the 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.

- 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"

- The 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 nookmarks list. On the browser's start up, the bookmarks list should be *loaded* to the browser.

- 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`.

- 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. *Hint:* This can be implemented by parsing the page's DOM after the page loads and extracting the first **five** `<a>` elements and associated `href` attributes, then rendering and displaying them in a sidebar component.

- A **simple GUI** should be provided to perform the operations discussed above. The GUI should be implemented using either the Windows Forms (Windows) or the Gtk# (Linux) libraries. 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 well as buttons to increase accessibility.

- ***[Advanced]*: 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 (covered in the lectures). **Please note, storing homepage, bookmarks, and browsing history is not optional, using a database to store will be counted as an advanced feature. For initial features, the homepage, bookmarks, and history can be stored in text files.**

- ***[Advanced]*: 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.

Library Usage. You should identify the suitable library functions out of the set of libraries installed with the .NET Core platform. The report must clearly motivate the usage of the functions that you have chosen. **You must not use the C# `WebBrowser` class in this implementation, but perform the required HTTP-level communication directly from within your code.** The code must clearly identify the HTTP-level client-server communication, and must explicitly manage home page, bookmarks, and history Lists. *Optionally*, you may add functionality to render a web page, but there must be an option to disable this functionality and to show only the raw HTML that has been retrieved.

4. Report Format

The report should be up to 10 pages long and use the following format (if you need space for additional screenshots, put them into an appendix, not counting against the page limit, but don't rely on the screenshots in your discussion):

1. **Introduction:** state the purpose of the report, your remit and any assumptions you have made during the development process.
2. **Requirements checklist:** clearly state which requirements are implemented and which are not; this should be in the form of a table (as shown in Appendix A) listing all the required features, and stating whether the functionality is fully implemented, partially implemented, or missing.
3. **Design Considerations:** describe the basic design decisions you have made in developing your code, covering class design, choice of data structures, GUI design, usage of advanced language constructs, other performance-relevant choices etc. Please also include the advanced functioanlities implemented in this section.
4. **Developer guide:** describe your application design and main areas of code in order to help another developer understand your work and how they might continue the development. In particular, point out
   - how is your code organized in classes, and which class is responsible for which functionlaity;
   - which libraries and/or technologies you used to execute the HTTP communication;
   - which libraries and/or technologies you used to make the browser settings (home page, history, bookmarks) persistent across sessions;
   - which libraries and/or technologies you used to create the GUI.
5. **Reflections on programming language and implementation (Only MSc Students):** based on your experience in implementing this application, reflect on which language features and technologies have been most helpful, identify limitations of your application and suggest ways to overcome these limitations.
6. **Conclusions:** reflect on what you are most proud of in the application and what you would have liked to have done differently.
7. **References:** a list of references used in this report and in the implementation. (This section does not count toward the report's page limit.)

5. In-person Demo

The in-person demo will be held in the lab sessions after the submission. The demo will be 5-10 minutes per student. The first 5 minutes of the demo will be for you to explain the requirements completed, and the next 5 minutes will be used as Q/A session. Please note that the Q/A may include questions about coding concepts and principles, not just the specific implementation details of your project.

Features not demonstrated in the demo will be considered not implemented. Please try to go through each requirement completed during the demo briefly.

The in-person demo will also be used to assess the completion of system requirements. Please ensure that you demonstrate all advanced features you have implemented during the demo.

6. Submission

The coursework submission consists of

1. a zip archive containing
   - the source code
   - a stand-alone executable
   - the report (in pdf format)
2. an in-person demo of the application

The standard penalty of 30% of the maximum available mark applies to late submissions. No submissions will be accepted after 5 working days beyond the submission deadline. Submission must be through Canvas. This coursework is worth 50% of the final mark.

Note that all submissions will be checked for plagiarism. If you are using code that was not written by one of your group members, including code generated by a large language model such as ChatGPT, then this must clearly be indicated, making clear which parts were not written by you and clearly stating where it came from. See university guidance on academic misconduct.

Feedback will be provided within 15 working days of the submission deadlines. The marking scheme for this project can be seen below.

7. Marking Scheme

Criterion	**Marks**
Meeting system requirements	
Sending HTTP requests	5
Receiving HTTP requests and displaying HTTP response codes	5
Display HTTP response HTML content	5
Display HTTP response HTML title	5
Create and edit homepage	2
Create, edit, view and delete bookmarks	5
Store, view and delete bookmarks	3
Navigate forwards and backwards between webpages visited	10
First 5 urls from the webpage displayed and clickable	5
Overall good GUI experience	5
Total marks for system requirements	**50**
Report Quality (Content and communication, design considerations, reflections, developer guide, feature checklist)	15
The application (code quality, secure and portable coding, clear method interfaces, sufficient comments, sensible variable/object names, good code/class/method design, use of advanced language features)	15
Advanced Implemented database storage using a relational database, including storing history, bookmarks and homepage.	5
Advanced Added multi-user functionality. This feature should also be implemented using a database, and each user should have their own memory of bookmark history and homepage.	5
In-person Demo of running the application and explanation of the requirements completed	10
Total marks	100

Note that 50% of the marks are allocated to *meeting system requirements*! This will be evaluated by testing the stand-alone application you submit and during the in-person demo, so **make sure you do not forget to submit the executable application**. Furthermore, **make sure that the application is portable**, i.e., it does not depend on resources that are present only on your own computer.

Generative AI Usage Policy

The use of generative AI tools, either for coding or for report writing, is **NOT PERMITTED** in this CW.

See: Use of GenAI on this course (AY25–26)

Our policy on the use of generative AI tools in F21SC is as follows:

- Where English is not your first language you may use GenAI to translate the assessment instructions or other relevant content. You may not use GenAI to translate your assessment answers - you must write them in English, as HW requires all work to be completed in English and to be your own work.
- **No GenAI generated content is permitted in your assessment submission.**
- You should ensure that any use of GenAI to support your assessment planning is identified in your notes to ensure that you do not unintentionally use it in the assessment.

This covers your use of tools such as ChatGPT, GitHub Copilot and other “AI search” or “code completion” tools. You must not use these tools to generate text or code for F21SC. We will report suspected use of generative AI as academic misconduct.

Academic Misconduct

Your attention is drawn to the university policy on academic misconduct and to the *Contract Cheating Risk and Avoidance Guide.*

This is an individual project and you will have to submit your own, original solution for this coursework specification, consisting of a report, the source code and an executable.

The code you submit must be your own work. If some text or code in the coursework has been taken from other sources (beyond the material provided for this course), you must give a clear reference to the source in comments in the code or the report. We cannot give any marks for work that is not your own. Failure to clearly reference work that has been obtained from other sources, or copying the words or code of another student, is plagiarism.

Asking a third party to complete coursework on your behalf, and then submitting it as your own work, is contract cheating.You must never give a copy of your coursework writing or code to another student, and you must always refuse any request from another student for a copy of your coursework. Sharing your coursework with another student is collusion.

Suspected plagiarism, contract cheating or collusion will be referred to the School Discipline Committee for investigation. This will delay your feedback, and if you are found guilty of misconduct then it will result in a penalty such as voiding the course or being withdrawn from the university. The use of AI generation and code completion tools is not acceptable.

So please don’t do any of the things above! If you have any concerns about potential academic misconduct, please ask your local course leader for guidance.

For details see: Student Guide to Academic Integrity

A. Feature Checklist Example

Here is an example feature checklist, listing all the features that will be marked and their implementation status. Partially implemented features come with a small note describing the known issues.

FEATURE	**STATUS**
Sending HTTP requests and displaying the contents of the response	**fully implemented**
Displayng the HTTP response status code	**fully implemented**
Displaying the web page's title	**partially implemented** – exception thrown when the `<title>` tag is missing
Reload the current web page	**fully implemented**
Seting the home page	**fully implemented**
Adding pages to bookmarks list	**fully implemented**
Naming bookmarks	**not implemented**
Deleting bookmarks	**fully implemented**
Removing bookmarks	**not implemented**
Navigating to bookmarks	**partially implemented** – browser crashes if an invalid URL is stored as a bookmarks
Maintaining history	**not implemented**
Navigating to previous/next page in history	**not implemented**
Navigating to a selected page in history	**not implemented**
List five urls from the webpage in the side panel	**partially implemented** – sometimes lists URLs in elements other than `<a>`
Settings (home page, bookmarks, history) persist across sessions	**fully implemented**