## 

## **HERIOT-WATT UNIVERSITY DUBAI**

## **SCHOOL OF MATHEMATICAL AND COMPUTER SCIENCES**

## 

## **F21SC - Industrial Programming 2023-24**

## **Coursework 1**

|  |  |  |
| --- | --- | --- |
| **Student ID Number** | **Student Name** | **Programme** |
| H00441124 | Ilia | MS Data Science |

## 

## 1. Introduction

The goal of this paper is to provide a comprehensive overview of the design, development, and implementation of the SimpleWebBrowser project. My task was to design a web browser that would be efficient and easy to use, while also including essential features like browsing history, favorites links, and bulk download feature.

## 2. Requirements Checklist

All the required features have been implemented successfully with all needed functionality.

Here is the list of them:

* Browser navigation: forward, backward and refresh buttons
* History feature with the possibility of having a look at the previously visited URL’s sorted by the time of visiting
* Favorites management: adding, viewing, and removing favorites.
* Home page management.
* Bulk download functionality.
* User interaction dialogs: input, favorites, and history views.

## 3. Design Considerations

When making the SimpleWebBrowser app, several factors were considered to make sure it worked well and was easy to understand.

**Class Design:**

* **Organization:** Different classes were created to manage distinct tasks. For instance, “BulkDownloader” handles downloading tasks, and “FavoritesManager” is responsible for managing favorite websites. This approach simplifies troubleshooting and future feature additions.
* **Separation of UI and Logic:** It was important to keep the user interface separate from from the underlying logic. This makes it easy to update and organize."

**Data Structures:**

* **Data Management:** Lists were used to track the websites visited, while key-value pairs were employed to manage favorite sites. These structures enable efficient data retrieval and manipulation.

**GUI Design:**

* **GTK Toolkit:** The GTK toolkit was chosen for its wide acceptance and robust features, promoting in the design of the user interface.
* **Modal Dialogs:** User feedback and interactions, such as displaying favorite sites or requesting additional input, were handled using modal dialogs.

**Advanced Language Constructs:**

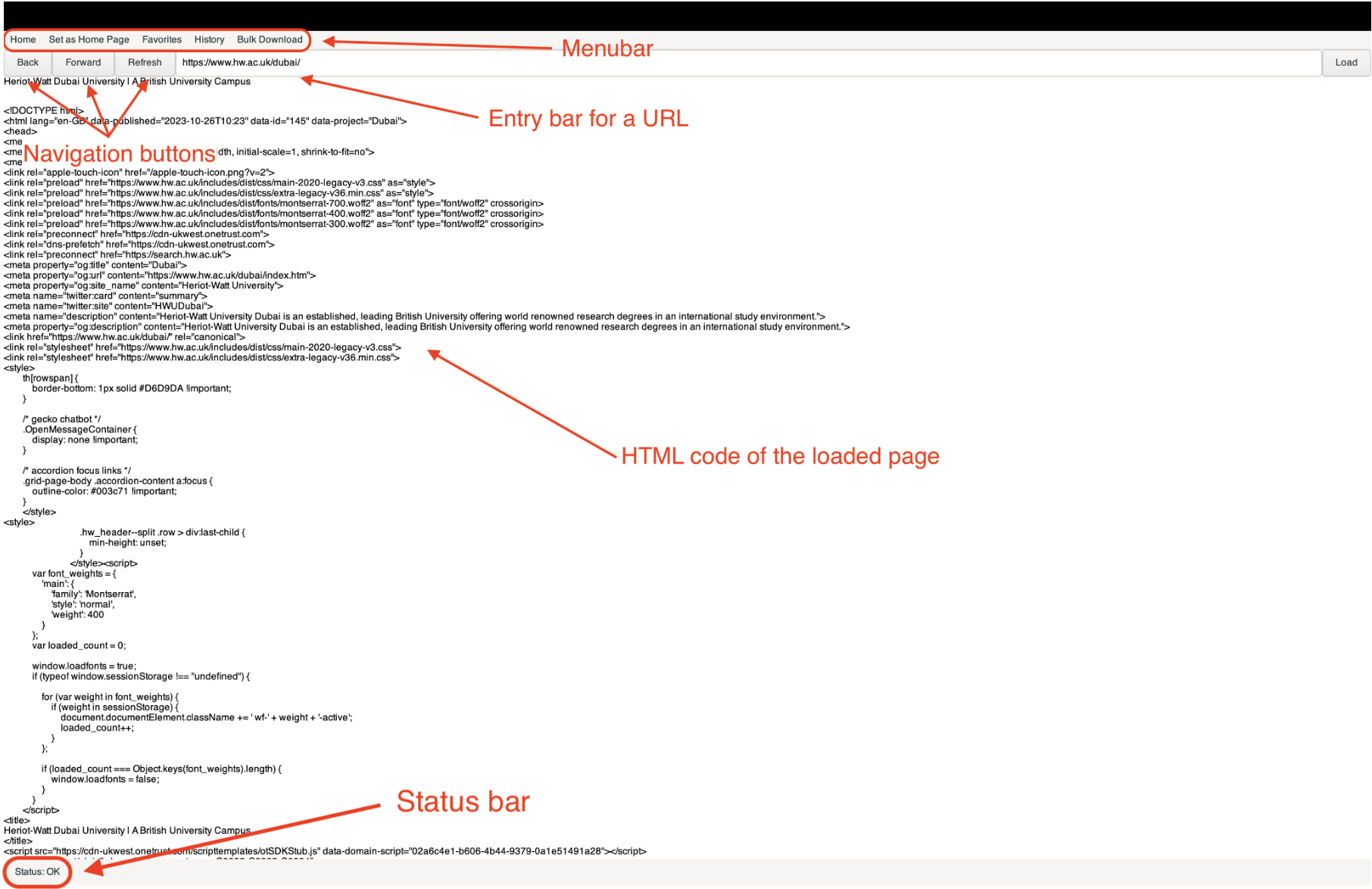
* **Event Handling:** The program responds to user interactions through events. This means when a user clicks a button or inputs data, an event gets triggered, prompting the program to act.

**Performance Choices:**

* **Managing Memory:** To ensure the application runs efficiently, there's a limit to the number of sites saved in the browsing history and download list. This measure helps in optimizing memory use and improving efficiency.

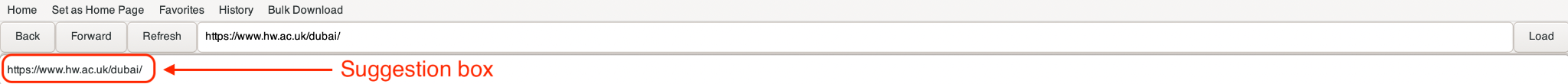
## 4. User Guide

The SimpleWebBrowser application is designed with user-friendliness in mind, ensuring smooth navigation and usage of its features.



*The home screen of the web browser with description of main parts*

**Navigation**: The Entry Bar is located at the top allowing you to input your preferred URL. As you type in the Entry Bar, a suggestion box will appear showing matching URLs from your history, which makes it a lot easier to revisit previously viewed web pages. Once your URL is entered, click the "Load" button to get the HTML code of the downloaded page. "Back", "Forward", and "Reload" buttons were also implemented for navigation.



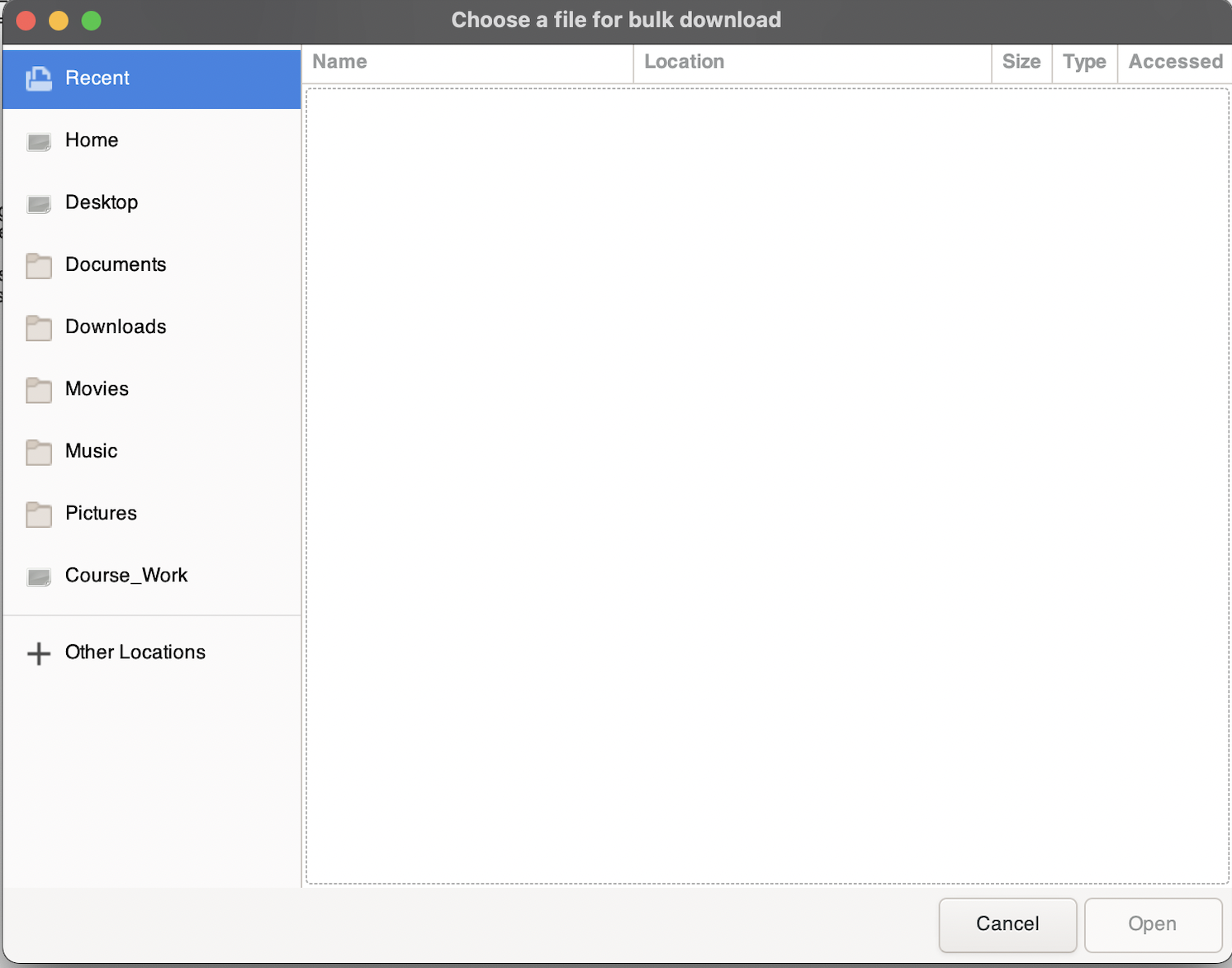
*Suggestion box in the entry bar*

**Favorites**: To save a website and easily find it again, you must first open the URL by clicking the “Load” button. Then you can click on the “Favorites” menu where you need to select the “Add to Favorites” button to add it to your favorites list. When you do this, a new window will appear where you can enter a specific website name. To view all your favorite websites, you can click the "View Favorites" button and a window will appear with website names and links.

|  |  |
| --- | --- |
|  | |
|  |  |

*Submenu of the favorites feature and dialog windows for adding and viewing favorites websites*

**Bulk** **Download**: If you want to use this feature, you need to click the corresponding button in your browser, after which a dialog box will appear where you need to choose a txt file with selected links from your computer. Then you will see a specific structure (<code> <bytes> <URL>) providing information about your request.



*Dialog window of the bulk download feature*

**History**: This feature stores all the web pages you visit in a specific text file, giving you the option to click the View History button and view your search history in a dialog box. For viewing a history was implemented one more advanced feature like timestamps which allows you to see visited websites in the chronological order with precise time.

|  |
| --- |
|  |
|  |

*Submenu of the history feature and dialog window*

**Status Display:** Users will notice a special feature that displays the status of their HTTP request in the bottom-left corner of the browser. This feature expands the status code into a more detailed text message, giving additional information about the type of response the server sent back.

## 5. Developer Guide

**Application Design Overview:**

SimpleWebBrowser is designed with a clear structure that keeps different parts of the application separate. This design makes sure the visual elements (like buttons), the stored data (like favorites), and the main functions (like loading a webpage) are kept in distinct areas. This separation helps in fixing issues, making updates, or adding new features to the browser.

**Architecture:**

The application's architecture is modular. There are two folders: “Main Components”, which contains files responsible for the primary functionality of the browser, and “Controllers” that houses files implementing various features.:

**Main Components:** This is where the core functionalities of the browser lie:

- *Program.cs:* The primary entry point for the application.

- *BrowserWindow.cs:* The visual part of the browser. It contains all the UI elements - from navigation buttons to address bars.

- *HttpRequestManager.cs:* A utility class designed to handle the formation, sending, and reception of HTTP requests.

**Controllers:** This is the logic layer of our application. Classes within this directory handle the various features of the web browser.

- *HomePageManager.cs:* Manages the browser's home page.

- *FavoritesManager.cs:* It's where all the user's favorite sites are stored, retrieved, and managed.

- *HistoryManager.cs:* Keep track of sites you've visited.

- *BrowserDialogs.cs:* A special class for managing all dialog boxes that optimizes user interaction.

- *BulkDownloader.cs:* Handles bulk downloading tasks, allowing users to load multiple URL’s simultaneously.

**Code Fragments:**

**1. Flow of the application (from Program.cs):**

*```*

*namespace SimpleWebBrowser*

*{*

*class Program*

*{*

*// Main entry point of the application.*

*static void Main()*

*{*

*// Initialization procedures.*

*Application.Init();*

*// Instantiate main application window.*

*var mainWindow = new MainAppWindow();*

*// Display the main window.*

*mainWindow.ShowAll();*

*// Start the application loop.*

*Application.Run();*

*}*

*}*

*}*

*```*

This snippet demonstrates a clear high-level overview of the main flow of the application.

**2. Managing the Home Page (from HomePageManager.cs):**

*```*

*namespace SimpleWebBrowser.Controllers*

*{*

*public class HomePageManager*

*{*

*private string homePage = "http://defaultHomePage.com";*

*// Property to get or set the home page URL*

*public string HomePage*

*{*

*get => homePage;*

*private set => homePage = value;*

*}*

*// Set the home page URL from a UI element*

*public void SetHomePageFromUIElement(UIElement element)*

*{*

*homePage = element.TextValue;*

*}*

*// Load the current home page URL into a UI element*

*public void LoadHomePageToUIElement(UIElement element)*

*{*

*element.TextValue = homePage;*

*}*

*// Attach home page related events to UI elements*

*public void AttachHomePageEvents(...)*

*{*

*// Define actions for UI events related to home page functionalities.*

*}*

*}*

*}*

*```*

This snippet of code demonstrates how easy it is to configure and get the browser's home page.

## 6. Testing

**1. Test Case: Launch Application**

- *Description:* Launch the web browser application.

- *Result:* The application opens with a default homepage, and no errors are displayed.

**2. Test Case: Enter URL**

- Description: Enter a valid website URL into the address bar and press Enter.

- *Result:* The website loads correctly.

**3. Test Case: Invalid URL Handling**

- *Description:* Enter an invalid URL format into the address bar.

- *Result:* The browser displays an error message.

**4. Test Case: Navigation Buttons**

- *Description:* Navigate to multiple websites and use the back and forward buttons.

- *Result:* The browser correctly navigates back and forth between visited sites.

**5. Test Case: Refresh Page**

- *Description:* Visit a webpage and click on the refresh button.

- *Result:* The current page reloads without errors.

**6. Test Case: Bulk Download Feature**

- *Description:* Initiate a bulk download for multiple links.

- *Result:* All selected links were successfully downloaded, the progress is displayed by the specific structure of this feature.

**7. Test Case: Favorites Functionality**

- *Description:* Add a website to bookmarks and access it from the bookmarks menu.

- *Result:* The website is added to the bookmarks, and clicking on the bookmarked site from the menu navigates to the correct page.

**8. Test Case: Set as Homepage Feature**

- *Description:* Set a specific website as the browser's home page and navigate away from the home page. Later, use the home page button to navigate back.

- *Result:* When using the home page button, the browser navigates to the chosen new home page for the current session. The browser returns to the university website by default when it is restarted.

**9. Test Case: History Access**

- *Description:* Visit several websites and then access the browser's history.

- *Result:* The list of visited websites is displayed in the history section.

**10. Test Case: Error Handling for Bulk Download**

- *Description:* Try to bulk download from invalid or broken links.

- *Result:* The browser provides an error message or feedback indicating which links failed to download and which were downloaded successfully.

## 7. Reflections on Programming Language and Implementation

I discovered C# to be a very interesting language during the process of creating this web browser, especially in contrast to Python, which is my primary language. As it seemed to me Python is better for task automation, developing less complex application or data science because the language is much easier itself and you don’t need to write so much code to perform simple function while in C# writing code is a bit tricky. However, I understand purpose of it, for instance, if you need faster compiling language or memory usage is crucial for a task, then better use C#.

**Useful language features and technologies:**

**1. Strong Typing of C#:** The static typing system of C# helps to avoid potential errors at compile-time. This is in contrast to Python's dynamic typing, where most type-related errors are discovered at runtime. This helped in building a more robust application.

**2. Built-in GUI support:** With C# and the .NET framework, developing graphical user interfaces was smooth, especially with tool like Gtk#.

**Limitations of the Application:**

**1. Limited Web Rendering:** The most significant limitation is that the browser can only display the raw HTML content of web pages.

**2. System Capacity Limitation:** Due to the simplicity of the browser's design and the certain limitations of the technologies used, it may not perform well under high user loads. This could lead to delays, problems with performance, or even crashes if many users simultaneously access it.

**Ways to Overcome the Limitations:**

**1. Advanced Rendering Engine Integration:** To provide a more complete web browsing experience, integrating a comprehensive rendering engine, like Chromium or WebKit, would be necessary.

**2. Extend Language Features:** It would be possible to delve into C# to use more sophisticated features or optimizations, including asynchronous programming, which could improve browser performance, especially when extracting and rendering web pages.

In the end, moving to C# for this project was a fresh experience with its own ups and downs. Even though my background in Python gave me a solid understanding of programming concepts, C# had its own way of doing things. It played a big role in how the app turned out. Sure, there were some bumps along the way, but I can see how to make things even better in the future.

## 8. Conclusions

I'm happy with how the SimpleWebBrowser project turned out. In addition to proper coding practices, the modular design opens the door for further improvements.

Even the process of creating the application was tough, but it gave an interesting experience of using classical OOP-based coding language.

Future work may focus on improving the user experience, integrating sophisticated rendering engines, and performance optimization to deliver a more smooth browsing experience.

Here is the link to the video of running the application:

<https://drive.google.com/file/d/1mDIVf-k4aNJBcL5iiyZjEKZ3oudT5Hlp/view?usp=drive_link>

## 9. References

1. GTK Documentation: [<https://www.gtk.org/docs/>]
2. Microsoft C# Guide: [<https://learn.microsoft.com/en-us/dotnet/csharp/>]
3. Microsoft .NET Guide: [<https://learn.microsoft.com/en-us/dotnet/fundamentals/>]
4. Model-View-Controller: [<https://medium.com/@charithavenkataraju/model-view-controller-mvc-ba6c07dff565>]
5. C# for .NET Developers [<https://www.coursera.org/learn/c-sharp-for-dot-net>]
6. C# Tutorial Contents: [<https://csharp-station.com/Tutorial/CSharp/SmartConsoleSetup.aspx>]
7. Code Project: Web browser in C# [<https://www.codeproject.com/Articles/60179/Web-Browser-in-C>]
8. ZetCode: GTK# Tutorial [<https://zetcode.com/gui/gtksharp/>]
9. Microsoft: HttpWebRequest Class [<https://learn.microsoft.com/en-us/dotnet/api/system.net.httpwebrequest?view=net-7.0>]