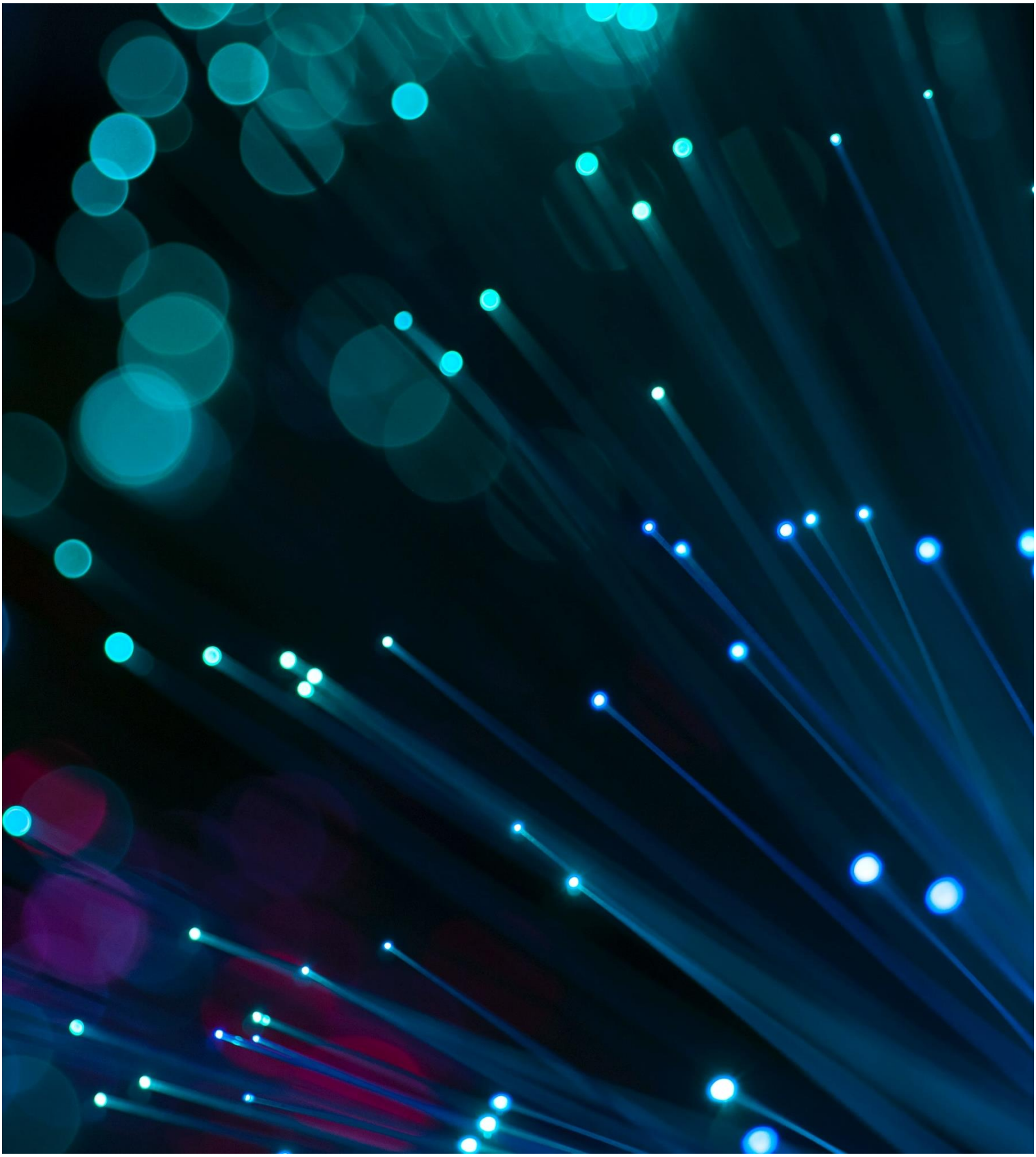


Web Browser Report

F21SC - Industrial Programming - Coursework 1

26/10/2021

H00359236

Contents

Demonstration Video Link: https://youtu.be/moQRrzQgyF4

1. Introduction

The purpose of this report is to highlight the functionality of the web browser application from both a user and developer perspective, provide motivations for particular design decisions, an overview of the testing done and to reflect on the programming language used and its effect on implementation.

The brief was to develop a simple web browser application using C# that allows users to enter a Uniform Resource Locator (URL) which would then generate an HTTP request; the application will then display the HTML webpage returned by the HTTP request on the User Interface (UI). In addition to this, there should be functionality for users to save favourites, view a history of pages they have visited, set the home page and traverse pages using a back and forward button.

2. Requirements Checklist

2.1 Sending HTTP request messages

This application meets this specification fully.

2.2 Receiving HTTP response messages

This application meets this specification fully, including supporting 400, 403 and 404 error messages.

2.3 Display

This application meets this specification mostly. On occasion the title is not parsed properly, and a string of html is returned to the title label instead.

2.4 Home Page

This application mostly meets this specification. The default home page is https://www.hw.ac.uk/ which is loaded and displayed when the browser is loaded. Users can then set a new homepage after searching for it using the browser search.

2.5 Favourites

This application meets this specification fully.

2.6 History

This application meets this specification fully.

2.7 Bulk Download

This application meets this specification fully.

2.8 Graphical User Interface

This application meets this specification fully.

3. Design Considerations

3.1 Class Design

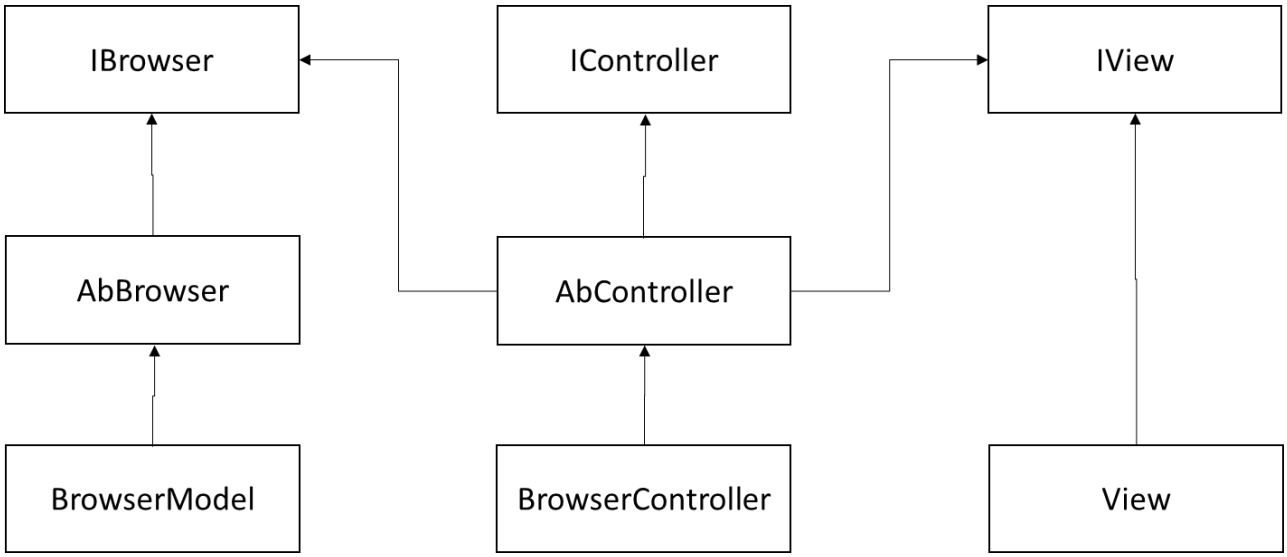
For this application I decided to use the Model View Controller (MVC) and Observer design patterns. I chose MVC in order to separate the representation of the underlying data in the model from the data presented to the user on the GUI; it also allows for the possibility of multiple views based on the same underlying model which would increase extensibility in the future.

The decision to incorporate the Observer pattern alongside this was to promote loose coupling and reduce the amount each class must know about each other's implementation to be functional. The Controller subscribes to events published by the GUI in response to user interaction and thus the View does not have to know about any instances of the Controller or Model which greatly decreases its dependence on these two classes.

To further increase loose coupling the class structure I have used is that each component of the MVC implements an interface for that component; for example, the Model implements the IBrowser interface. The Controller class then references the IBrowser and IView interfaces rather than their concrete implementations.

Finally, the Controller and Model classes inherit from abstract classes that implement their respective interfaces. This allows for specific functionality to be added or changed in different implementations of a Model without effecting its functionality or interaction with other classes.

The diagram below shows the high-level class structure for the MVC classes:



3.2 Data Structures

3.2.1 Favourites

To implement the functionality of the favourites I decided to use a Dictionary implementing the IDictionary interface. Initially I was drawn to using a dictionary because intuitively it made sense seeing as the favourites would be stored by name and have a URL associated with it and so being able to retrieve the URL based on the favourite name seemed a simple to implement in few lines of code. In addition, the lookup speed of a Dictionary is very fast (close to O(1)) and so from a performance perspective it made sense over using a List of objects and iterating through it.

3.2.2 History

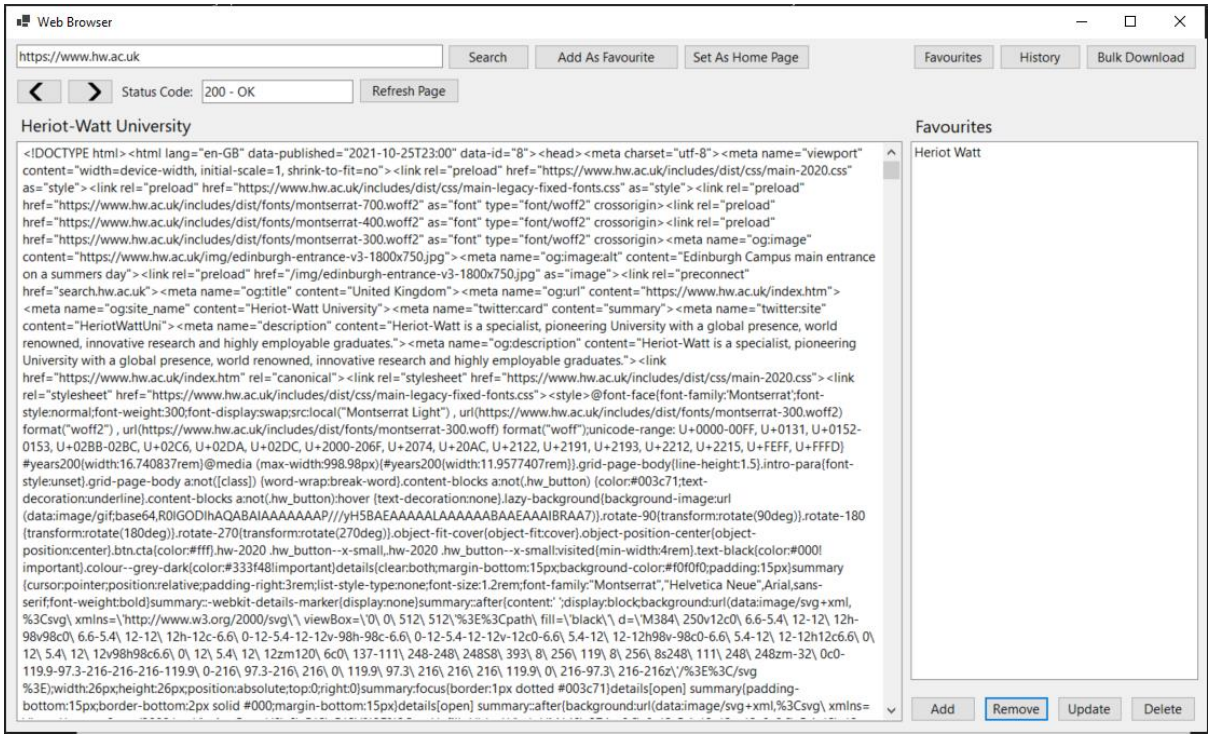
To implement the functionality of the history I decided to use a SortedList. I wanted the history to be displayed in the order in which searches were made and so my first thought was to use a SortedList where the key was the DateTime the HTTP request was sent and the value being the string URL that was searched. This way each new search is automatically added in order and the history is printed to the GUI in order of DateTime.

3.2.3 Custom EventArgs

For the events of requesting a page, loading of a new page, setting of a home page, adding a new favourite and getting a bulk download I implemented custom even arguments because it allowed me to have more control over the data that was passed during these events. For example, the result of an HTTP request could be stored in the PageChangedEventArgs and allowed me to specify fields required by the GUI (e.g. status code and title) without having to process the information closer to the View. In addition, it allowed me to use them as shared objects between the loosely coupled MVC classes in accordance with the Observer pattern.

3.3 GUI Design

I opted for a simple GUI design where the majority of information was already present on the screen with the exception of the favourites and the history. In keeping with most modern browser design, I put the options for favourites and history in the top right corner, where when clicked, would appear on the right-hand side of the GUI after reducing the size of the HTML output window.



3.4 Use of Advanced Language Features

3.4.1 Generics

In the implementation of the history, I decided to implement the class using generics and leave the specification of using DateTime and string as the key value pair until concrete implementation at run time. I did this to increase the flexibility of the history implementation; if in the future a different representation of history was needed instead of using DateTime and string, the class would not have to be altered.

3.4.2 Indexing

I implemented indexing in the custom favourites and history classes so that their values could be retrieved by index quickly, which was especially useful in the history when deleting an item, the selected item to delete is specified by the index of its position in the ListBox. I also implemented IEnumerable in both favourites and history so that I could use foreach loops when adding items to the respective ListBoxes which made coding simpler.

3.4.3 Delegates

I used delegates for the following events:

- Requesting a new page
- Getting a favourite
- Adding a new favourite
- Setting a new home page
- Traversing pages in the history

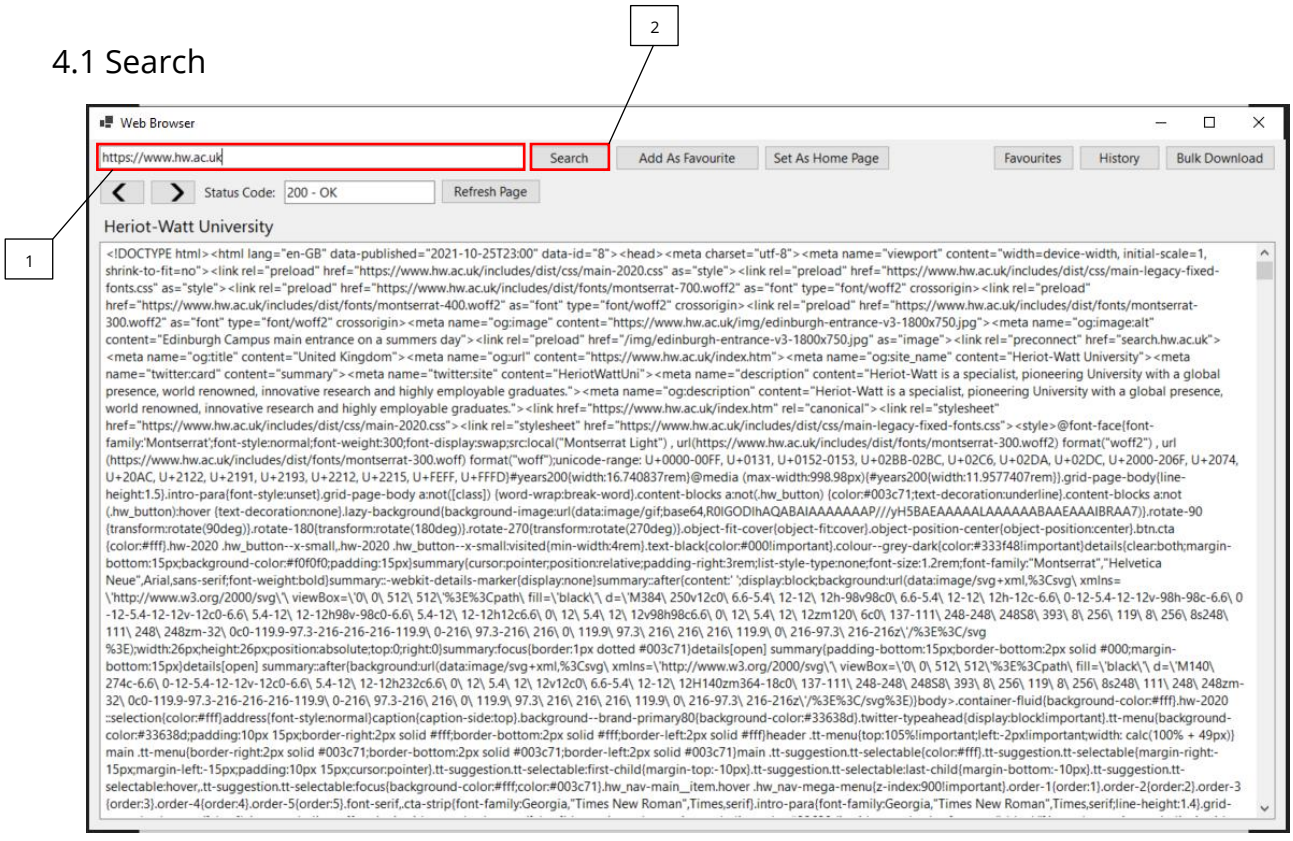
I found this allowed me to encapsulate these methods that required more complex data to be passed when the events were triggered and increase the flexibility of these method implementations.

3.4.4 Reflection

I used reflection in the BulkDownldObj class to serialise the class properties so they could be added to the main text box on the GUI quickly and with minimal coding in the method calling Serialise(). I found this a quick and effective way to retrieve the necessary data.

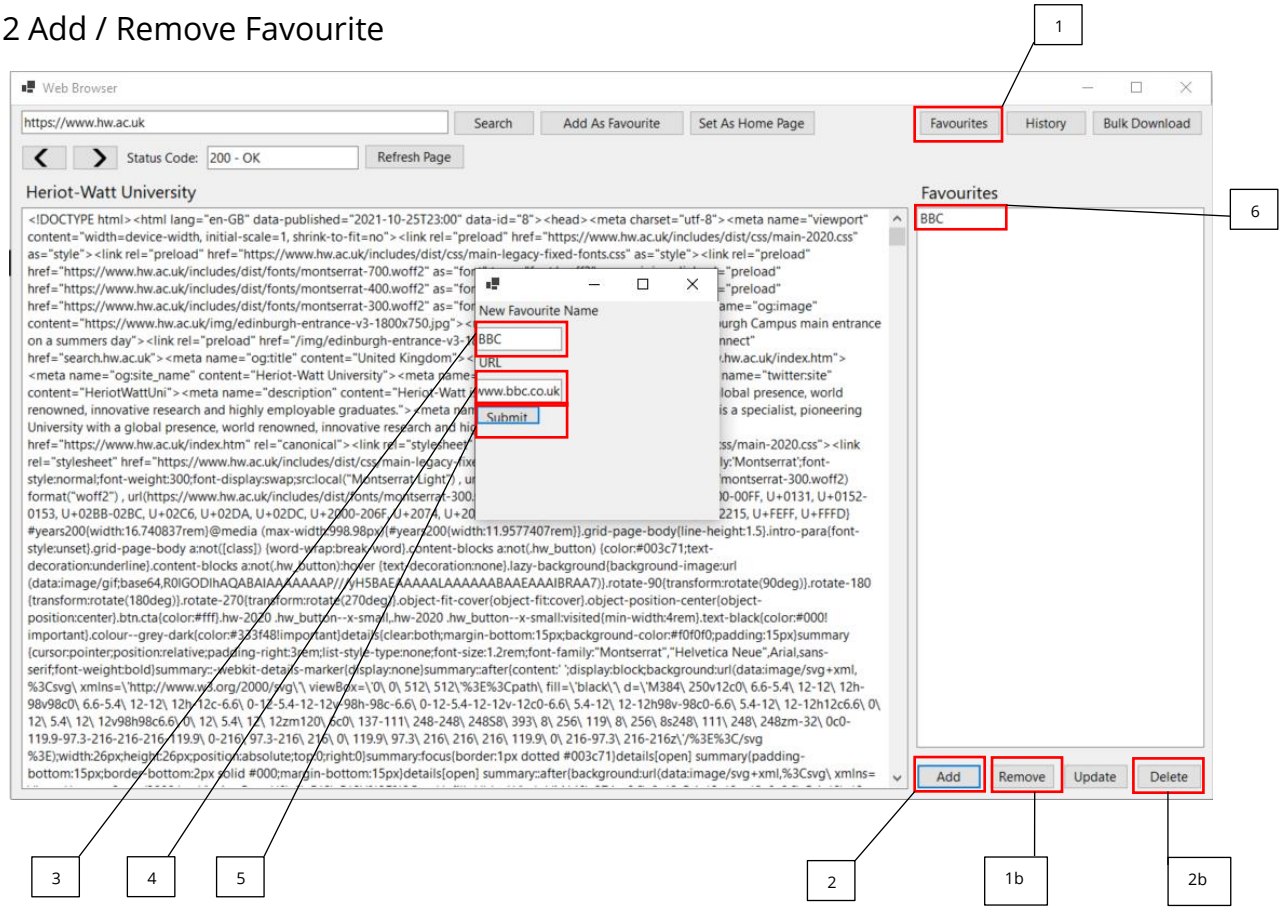
4. User Guide

4.1 Search



1. Enter URL in search bar
2. Click “Submit”

4.2 Add / Remove Favourite

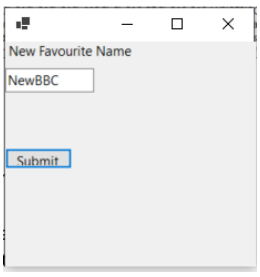


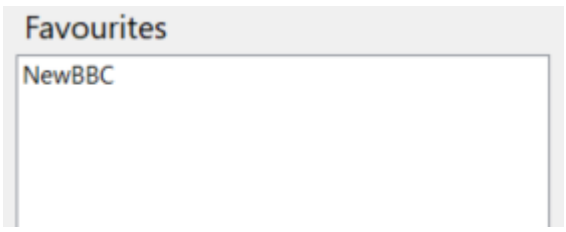
1. Click “Favourites” to open the favourites menu
2. Click “Add”
3. Enter a name for the new favourite
4. Enter a URL to associate with the favourite
5. Click “Submit”
6. You will see the new favourite appear in the favourites menu. Double click the entry to be shown that page.

1b. Click on the favourite you would like to remove and then click “Remove”

2b. To delete all favourites, click “Delete”

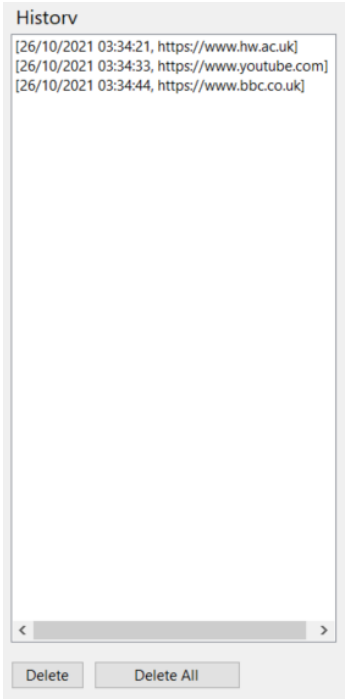
4.3 Update Favourite Name





1. Select a favourite to update and click “Update”
2. Enter a new name in the text box
3. Click “Submit”
4. You will see the name update in the favourites menu

4.4 History



See History Item:

1. Double click history item

Delete History Item:

1. Select an item in the history
2. Click “Delete”

Delete Entire History:

1. Click “Delete All”

5. Developer Guide

My application design is based on the MVC classes interacting through either their interfaces or publishing and consuming events. The View class publishes an event for each user interaction in the GUI and the BrowserController class subscribes to these events:

```
public AbController(IBrowser browser, IView view)
{
    this._browser = browser;
    pageDelegate = new IBrowser.PageChangeDelegate(this.UpdateView);
    _browser.PageChanged += pageDelegate;
    bulkDownldDelegate = new IBrowser.BulkDownloadDelegate(DownloadComplete);
    _browser.BulkDownloadComplete += bulkDownldDelegate;


    this._view              = view;
    urlDelegate             = new IView.URLDelegate(Search);
    _view.RequestURLEvent   += urlDelegate;
    guiLoadedDelegate       = new IView.GetFavouritesDelegate(UpdateFavourites);
    _view.GetFavouritesEvent += guiLoadedDelegate;
    newFavouriteDelegate    = new IView.NewFavouriteDelegate(AddNewFavourite);
    _view.NewFavouriteEvent += newFavouriteDelegate;
    newHomePageDelegate     = new IView.NewHomePageDelegate(SetNewHomePage);
    _view.NewHomePageEvent  += newHomePageDelegate;
    pageTraversal           = new IView.PageTraversalDelegate(HistoryTraversal);
    _view.PageTraversalEvent += pageTraversal;

    _view.RemoveFavouriteEvent      += RemoveFavourite;
    _view.GetFavURLEvent            += FavouriteRequest;
    _view.NewFaveNameEvent          += UpdateFavouriteName;
    _view.GetHistoryEvent           += GetHistory;
    _view.ViewCloseEvent            += SaveFavourites;
    _view.ViewCloseEvent            += SaveHistory;
    _view.ViewLoadedEvent           += InitialisePageIndex;
    _view.ViewLoadedEvent           += LoadHomePage;
    _view.BulkdDownldEvent          += BulkDownload;
    _view.DeleteAllHistoryEvent     += RemoveAllHistory;
    _view.DeleteHistoryItemEvent    += RemoveHistoryItem;
    _view.DeleteAllFavouritesEvent  += RemoveAllFavourites;

}
```

The BrowserController class then processes these events y interacting with the BrowserModel, sending data back to the View if necessary.

The HTTP requests are handled by the BrowserModel class, the results of which are either stored in instance variables ready to be retrieved by the BrowserController, or they are published to a shared object through an event as is the case with the bulk download.

```
// Send an HTTP request using the string URL passed in
// as an argument. Returns a List<string> containing the
// status code, status message, page title and HTML content.
3 references
public async Task<List<string>> GetRequest(string URL)
{
    if(URL == null)
    {
        Debug.WriteLine("Caught null url");
        throw new ArgumentException(URL);
    }
    List<string> arr            = new List<string>();

    HttpResponseMessage request = await client.GetAsync(URL);
    string content              = await request.Content.ReadAsStringAsync();

    int start                   = content.IndexOf("<title>") + 7;
    int stop                    = content.IndexOf("</title");
    int len                     = stop - start;
    string title                = content.Substring(start, len);
    string code                 = Convert.ToString((int)request.StatusCode);

    arr.Add(code);
    arr.Add(request.StatusCode.ToString());
    arr.Add(title.Trim());
    arr.Add(content);

    return arr;
}
```

6. Testing

Test	Steps	Outcome	Comments
Search URL	1. Enter "https://www.youtube.com" in search bar. 2. Click "Search" button.	Pass	HTML, title and status code are displayed as expected.
Search empty string	1. Click "Search" button while search bar is blank.	Pass	Error message displayed correctly.
Search invalid URL	1. Enter "47v782bfy" in search bar. 2. Click "Search" button.	Pass	Error message displayed correctly.
Search valid URL that does not exist	1. Enter "https://www.heriotpink.com" 2. Click "Search" button.	Pass	Error message displayed correctly.
Refresh page	1. Click "Refresh Page".	Pass	Same HTML page reloaded
Add favourite	1. Open Favourites tab. 2. Click "Add". 3. Add name "Stack Overflow" and URL "https://www.stackoverflow.com". 4. Click "Submit"	Pass	Favourite added and diplayed in Favourites window. Correct HTML displayed when new favourite is double clicked.
Add duplicate favourite	1. Open Favourites tab. 2. Click "Add". 3. Add name "Stack Overflow" and URL "https://www.stackoverflow.com". 4. Click "Submit"	Pass	Error message displayed correctly.
Submit favourite with no name or URL	1. Open Favourites tab. 2. Click "Add". 3. Click "Submit"	Pass	Error message displayed correctly.
Remove favourite	1. Open Favourites tab. 2. Select "HW". 3. Click "Remove".	Pass	Favourite successfully removed
Update favourite name	1. Open Favourites tab. 2. Select "HW". 3. Click "Update". 4. Enter "NewName" and click "Submit".	Pass	Favourite name successfully updated and same HTML is displayed as with old favourite name.
Update favourite name to duplicate name	1. Open Favourites tab. 2. Select "HW". 3. Click "Update". 4. Enter "HW" and click "Submit".	Pass	Error message displayed correctly.
Delete all favourites	1. Open Favourites tab. 2. Click "Delete"	Pass	All favourites deleted.
Navigate to history	1. Open History tab. 2. Double click url at the top of the list.	Pass	Correct HTML displayed corresponding to that URL.
Delete history item	1. Open History tab. 2. Select item in history list 3. Click "Delete"	Pass	Correct entry deleted.

Delete all history	1. Open History tab. 2. Click "Delete All"	Pass	All history items deleted.
Bulk download	1. Select "Bulk Download". 2. Enter "bulk.txt". 3. Click "Submit".	Pass	Output displayed as expected.
Bulk download with non-existing file name	1. Select "Bulk Download". 2. Enter "test.txt". 3. Click "Submit".	Partial Pass	Application does not crash, however error message is not displayed.
Bulk download with blank file name	1. Select "Bulk Download". 2. Enter "test.txt". 3. Click "Submit".	Partial Pass	Application does not crash, however error message is not displayed.
"Add As Favourite" from search bar	1. Enter "https://www.hw.ac.uk" 2. Click "Search" 3. Click "Add As Favourite" 4. Enter name "HW". 5. Click "Submit".	Pass	Favourite added and diplayed in Favourites window. Correct HTML displayed when new favourite is double clicked.
"Add As Favourite" from search bar with duplicate name	1. Enter "https://www.hw.ac.uk" 2. Click "Search" 3. Click "Add As Favourite" 4. Enter name "HW". 5. Click "Submit".	Pass	Error message displayed correctly.
"Add As Favourite" from search bar with search bar as blank	1. Clear search bar 2. Click "Add As Favourite" 3. Click "Submit".	Pass	Error message displayed correctly.
Set new home page	1. Enter "https://www.youtube.com" in search bar. 2. Click "Search" button. 3. Click "Set As Home Page".	Pass	On relaunching the application the new home page is set to "https://www.youtube.com".
Set new home page with blank URL	1. Clear search bar. 2. Click "Set As Home Page".	Pass	Error message displayed correctly.

7. Reflections

During the design and implementation of this application I found the C# events to be particularly helpful. They allowed me to keep the Model and View components completely separate from the concrete implementations of each other by having them react to events and publish the results rather than calling methods in the implemented classes. I also found the ability to use generics very helpful and in hindsight I would have used them more to create a more steam lined application and reduce the number of collections I used, particularly for custom EventArgs.

A limit of the application I have made is that because the implementation of the other classes is hidden from View and Model, the method calls to transmit data back and forth can be quite deep which can make debugging challenge as there are multiple places in the call stack that could have caused the error. In the same vein, despite its loosely coupled nature, there is a large burden on the BrowserController class to handle all of the processing and moving of data between the Model and the View which can also hinder debugging.

To overcome these issues, I would fragment the Controller class into smaller classes focussed on smaller aspects of functionality, for example, a Controller class to handle the processing of favourites and another to handle history.

8. Conclusions

Overall, I have found this project challenging yet rewarding as I have learned a lot about C# and its features in addition to deepening my understanding of development and program structure, in particular in a system language.

I am most proud of the level of abstraction I managed to achieve using the MVC and Observer design patterns in combination with using a structure of interfaces and abstract classes and advanced language features.

If I had more time, I would have liked to streamline my code using more generics and reflection and tested it using unit testing.