Web Browser

Coursework 1

F20SC: Industrial Programming

Sam Fay-Hunt — `sf52@hw.ac.uk`

October 29, 2020

Contents

1 Introduction

2 Requirements

This Section will provide a comprehensive requirements checklist. The requirements have been split into 2 sections: Section. 2.1 (Model Requirements) contains all the requirements related to the "business logic", Section. 2.2 (User Interface Requirements) will describe all the requirements related to the user interface view and control components.

Requirements marked with the Priority **Essential** are requirements explicitly requested in the task brief.

The priority for each requirement is encoded as follows:

- **Essential** - This priority indicates that this requirement must be implemented to satisfy basic functionality of the browser.
- High - A high priority indicates that this requirement is important for providing a good quality user experience.
- Medium - A medium priority requirement is nice to have but if it is missing it is acceptable.
- Low - Low priority indicates that this is unlikely to be fulfilled within the time frame of the project, a stretch goal at best.

2.1 Model Requirements

Requirement	**Description**	**Priority**	**Status**
Send HTTP requests	Send HTTP request messages for URLs typed by the user	**Essential**	Complete
Recieve HTTP responses	recieve and store HTML code & response status codes	**Essential**	Complete
Display HTTP response	Display HTTP statuscode & the title of the web page	**Essential**	Complete
Home page	Create and edit home page URL	**Essential**	Complete
Favourites	Add URL with name to a list of favourite web pages	**Essential**	Complete
Navigate to Favourite	Seleting a favourite from the lsit will navigate to the page	**Essential**	Complete
Modify Favourites	Support editing & removing favourite items	**Essential**	Complete
History	A list of visited URLs should be maintained	**Essential**	Complete
Navigate history	Selecting a history item from the list will navigate to the page	**Essential**	Complete
Persistant home page	Store the home page url locally and load it to the browser on startup	**Essential**	Complete
Persistant favourites list	Serialize the favourites list and load it to the browser on startup	**Essential**	Complete
Persistant history list	Serialize the history list and load it to the browser on startup	**Essential**	Complete

Requirement	**Description**	**Priority**	**Status**
Status code error messages	Support for the corresponding messages for the following status codes: 200, 400, 403, 404	**Essential**	Complete
Modify history	The user should be able to clear all or specific history items	Medium	Complete
History names	The list of URLs should have reference to the associated page title	Medium	Complete
Sort history	Sort the history chronologically (using the access time)	Medium	Complete
Sort favourites	Sort the favourites by their associated names alphabetically	Medium	Complete
Prepend incomplete URL with protocol	Prepend the protocol ”http://” & ”www.” to the URL when missing	Low	Incomplete

2.2 User Interface Requirements

Requirement	**Description**	**Priority**	**Status**
URL input box	Enter a URL to send HTTP requests	**Essential**	Complete
Display HTML code	Main GUI view component for displaying HTML returned from a HTTP response	**Essential**	Complete
Display HTTP status code	A GUI element of a users HTTP status codes & their corresponding messages	**Essential**	Complete

Requirement	**Description**	**Priority**	**Status**
Display web page title	Web page title displayed at top of browser window	**Essential**	Complete
Navigate to home page button	A button that when pressed navigates to the home page	**Essential**	Complete
Set home page	A button to set the current page as the home page	**Essential**	Complete
Set favourites	A button to set the current page as a favourite with the page title as the associated name	**Essential**	Complete
Set custom favourite	A button to set a favourite with a user defined title	**Essential**	Complete
View favourites list	A selectable menu to display all favourites currently represented in the model favourites list	**Essential**	Complete
Edit favourites window	A window that enables the user to delete or update favourites	**Essential**	Complete
View history list	A selectable menu to display all history items currently represented in the models history list	**Essential**	Complete
Edit history window	A window that provides controls enabling the user to delete or edit history items	**Essential**	Complete
URL input box	Enter a URL to send HTTP requests	**Essential**	Complete
Shortcut keybinds	Make use of shortcut keys to improve accessibility	**Essential**	Partially Complete

Requirement	**Description**	**Priority**	**Status**
Display sorted history menu items	Render the history list in sorted order	Medium	Incomplete
Display sorted favourites menu items	Render the favourites list in sorted order	Medium	Incomplete

3 Design Considerations

3.1 Class design

In general the high level design of this browser has loosely followed the MVC pattern.

3.1.1 PageContent

The most important class to the underlaying model of this web browser is the **PageContent** class, this class contains references to the **PageHistory**, **History**, **Favourites**, and **BrowserResponse** classes. It abstracts away alot of the async behaviour by using an Event called **ContextChanged** which is triggered any time a HTTP get request returns a new **BrowserResponse** instance, this means the GUI elements can subscribe to this event and repaint the relevant elements when it gets triggered. This class also exposes all the navigation functionality to the GUI and the information such as the HTML code, web page title and status code returned via get requests.

3.1.2 BrowserResponse & HttpRequests

HTTP GET requests are handled by the *static Get(string url)* method of the **HttpRequests** class, this method returns an instance of the **BrowserResponse** class which is instantiated asynchronously and contains properties including the page title, HTML source code, URL, and status code.

3.1.3 History & Favourites

There is a significant degree of overlap in the behaviour of the **History** and **Favourites** classes so the common functionality has been implemented in a class called **EntryRecord**. The main distinction between them is the behaviour when there is a duplicated title in their respective lists, this is realised by overriding the *KeyExists()* method. **History** handles name collisions for new history entries by appending an integer to the end of the name, **Favourites** on the other hand simply forces the user to enter a new unique custom name if they wish to favourite a page with a name already in its collection.

3.1.4 PageHistory

PageHistory is a session based history navigation class, it only exposes methods for moving back and forwards through the history for a given session, and adding new history items to its own list. It also has properties that provide information about the current node being pointed to in the list. This class is independent of the **History** class in terms of data, but the nodes representing a single page in the history do inherit from the same abstract class called **Entry**. It was a deliberate design descision to seperate the **PageHistory** and **History** classes, this behaviour is reflected in several prominent web browsers, for example Google Chrome and Mozilla Firefox.

3.2 Data structures

3.3 Gui Design

3.4 Advanced language constucts

4 User Guide

5 Developer Guide

6 Testing

7 Conclusions