2018/2019

Student: Alex McBride

Student ID: S1715224

Course: BSc Computing (4th year)

Lecturer: Jim Devon

Client-Side Web Development

Coursework Report

Table of Contents

[1 Introduction 2](#_Toc532730896)

[2 Structure 3](#_Toc532730897)

[3 Testing 6](#_Toc532730898)

[4 Reflective 32](#_Toc532730899)

[5 References 35](#_Toc532730900)

# Introduction

This report will cover the development of a single page application for the client-side web development coursework, show the testing results for the web site, as well as reflect on what I have learned during its development.

# Structure

This section of the document will look at the structure of the web applications, how the various modules are arranged, the data structure of the initial payload and browser storage, and more.

The application is a Single-Page Application (SPA), so the whole application runs inside of the single index html page, and the content for each page is loaded dynamically. The index page has been kept as minimal as possible, with only a few elements needed by the shell of the application, such as the header, footer, and divs needed to hold various page elements. The app has been split into multiple folders, with scripts containing all the JavaScript files, styles containing the CSS, images containing icons and images, and videos containing video files. This was done to try keep the different file types used by the application separate. Each JavaScript module has its own file inside of the scripts folder, and the name of the file matches the name of the module.

The application required that different page types be handled, such as image, slideshow, video, quiz etc. The architecture of the application was constructed in a modular so that new page types can be added easily. This is managed in the app.js module, which is the first part of the application to be run and contains code to setup and manage everything else. The app.run() method tries to load the initial JSON payload from the server (if needed), and if successful initializes the various UI components for the menu, login, and search features, before setting up the handlers for various types of page content. As many parts of the application need to be able to render and re-render the main content when it dynamically changes, this is handled by the app.render() method, which allows this to be contained in one place. This increases the coupling between modules but means that the content rendering code does not need to be spread throughout the application.

The content-loader.js module watches for changes to the location hash, and then loads the relevant page handler to display it. It also will attempt to load the data for that page (more on that below), if a page ID is passed in the URL hash. URL hashes take for the form of #page-type/id/optional-data. The page type specifies the type of handler to load (e.g. post, image etc), the ID specifies the ID of any data for that page, and the optional part can contain any further data, like a specific slide from the slideshow to display. Using the browser’s hash to load content is a good idea, as it means that the back button and other browser features keep working, which improves usability. However, this is at the expense of extra complexity for the app. The handler for each page type is contained in the page.js module, which either renders the HTML for that page itself, or delegates it to a manager module, for instance the slideshow or quiz managers.

The initial payload for the SPA is contained inside a single file called data.json, found in the web server’s root directory. This contains a list of objects each one representing a page of the web site. Each page has three standard attributes, id, type, and title, as shown in Table 1. The id is a string used to identify the page inside the app, both in the URL hash which tells it which page to display, but also when saving/loading data in local storage. The type attribute identifies the type of the page, either post, image, slideshow, video, or quiz. Finally, the title is the name of the page, as it appears both in the menu and the header. The rest of each JSON object differs depending on the page in question, as outlined in Table 2. The code for the site has been designed to be modular and it’s easy to add new page types without having to rewrite the code.

Table 1 - JSON payload structure shared by all pages

|  |  |  |
| --- | --- | --- |
| **Page-type** | **Attribute** | **Description** |
| All | id | A string identifier for the page (that can be included in a URL) |
| type | The type of the page |
| title | The title of the page, as appears in the menu and the header |

Table 2 – additional JSON payload structure for different page types

|  |  |  |
| --- | --- | --- |
| **Page type** | **Attribute** | **Description** |
| Post | content | A block of text containing the contents of the post |
| Image | src | The source URL for the image |
| Slideshow | images | An array of slideshow images, each one with a title and a src URL |
| Video | src | The URL for the video |
| contentType | The video content type (e.g. video/mp4) |
| Quiz | description | A description of the quiz contents |
| questions | An array of quiz questions, each containing:   * Text - The text of the question * correctIndex - The index of the correct answer in the options array * options - An array of possible answers to the question |
| answers | Associated array of previous answers with the user’s name as key |
| currentAnswers | Array containing list of current answers in the question, so they don’t get lost if the user refreshes the page |
| Heroes | heroes | A list of heroes, converted to JSON from the original CSV file |

Once the initial data.json payload has been downloaded, it is stored in the browser’s local storage (see Figure 1). When the app first loads it checks to see the data is present, and if needed an AJAX call is made to fetch it. A version number is stored in local storage and compared against one in the data-store.js module, and if they do not match then local storage is wiped and the payload redownloaded. This allows the local storage to be refreshed easily during development as the data structure is frequently changing. Each page is converted into a JSON string and stored in local storage with the ID as key, making it easy to retrieve them to be displayed. As local storage is just a key/value store, a way is needed to be able to retrieve a list of all page IDs, for instance for displaying them on the main menu, and so an attribute called ‘ids’ was added that contains a comma delineated list of ids, for easy retrieval. We also store a further attribute called ‘indexes’, which is used in the search feature. As local storage cannot be indexed or searched, we created our own ‘ghetto’ index that contains the title of each page along with its ID stored as JSON strings. This lets us quickly search titles to find their IDs to create the search results. Of course, this is slightly limited in terms of the results you’ll get back but works efficiently enough for a simple site like this. Lastly, in local storage, we store a small object containing the logged in user’s username, which lets the site remember them when the page is refreshed.

This data structure was chosen to keep things simple and modular. By having everything in one big file and having them share a basic format, we can reuse as much code as possible, for instance in the content-loader, menu and data-store. While splitting the JSON payload up into smaller files could have improved performance, it would have added more complexity to the site, and would not have been as flexible requiring the SPA to make further requests to the server after it had initially loaded. In the end the size of the whole site, excluding images and videos, is only around 250kb which is reasonably small. For large amounts of data, a different method of loading the JSON would likely be needed, but for a small site like this, the one chosen proved adequate.

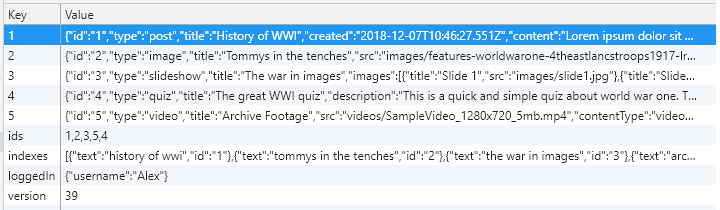


Figure 1 – Example local storage in the chrome browser

Each module was implemented using the revealing module pattern. The module pattern was a requirement of the coursework, and the revealing pattern chosen as it avoided cramming too much code into the object literal at the bottom of the immediate function. Using the revealing module pattern also allowed internal module functions and attributes to be renamed and reconfigured without affecting the outside interface, as well as allowing the access to the function to be modified easily without having to rewrite lots of code. This allowed the code to be written in a more encapsulated fashion, where changes inside of each module were insulated from affecting the code outside of it. There are a set of required modules, which were included, along with several more that were added to aid the development of the code. These are listed in Table 3.

Table 3 - list of modules

|  |  |
| --- | --- |
| **Name** | **Description** |
| app.js | Starts the application and renders the components |
| content-loader.js | Maps the content handlers to different pages, and handles the browser location change |
| menu.js | Renders the main menu |
| page.js | Contains the handler for each page |
| admin-manager.js | Module for managing pages, logged in users can add, edit, and delete pages. |
| admin-forms.js | Helper created to keep admin-manager.js smaller and easier to manager, contains all the page classes |
| ajax.js | Small module to encapsulate a basic AJAX JSON fetch call |
| content.js | Small helper module to encapsulate writing to the main content element of the page, as this needs to be done from multiple places in the app |
| data-store.js | Module implementing the repository pattern that abstracts away browser storage |
| login-manager.js | Module to manage logging the user in and out (simple unsecure hard coded users are password) |
| quiz-manager.js | Module to manage quizzes, each quiz is represented by an internal class and stored in a map, to allow multiple quiz states to be maintained at once. |
| search-manager.js | Module to manage searching the site (currently only page titles), draws the search box on the page, and then displays the search results. |
| slideshow-manager.js | Module to manage displaying a slideshow, a collection of images. |
| url-helper.js | Small helper module to manage pulling data out of URLs. |

During development an effort was made to use no third party of outside libraries or code, with the one exception made for bootstrap.css. Bootstrap is a popular CSS library that provides more aesthetically pleasing default styles and responsive layouts. All JavaScript code was written using ECMAScript 5 (ES5), as explained in the reflective part of this report.

# Testing

Here are the results of testing the application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Name** | **Input** | **Expected** | **Actual** |
| 1 | Test start of app | Start server and navigate to app | Home page is displayed with list of menu items on left | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | Test post page | Select item ‘Why did the war start?’ page from left menu | Post is displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | Test image page | Select ‘Countries of the war’ image menu | Image is displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | Test video page | Select ‘Basic training’ from menu | Video is displayed and can be played | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | Test slideshow | Select ‘War in images’ from menu | Frist slide of slideshow displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6 | Test slideshow next | Press ‘next’ button on slideshow | Slideshow moves to next slide | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | Test slideshow previous | Press ‘previous’ button on slideshow | Slideshows moves to previous slide | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 8 | Test slideshow end | Press ‘end’ button on slideshow | Slideshow goes to last slide | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 9 | Test slideshow start | Press ‘start’ button on slideshow | Slideshow does to first slide | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | Test slideshow go to slide input | Enter ‘2’ in go to slide input and press enter | Slideshow goes to slide 2 | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | Test slideshow go to slide bad input | Enter ‘a’ in go to slide input and press enter | Message ‘a is not a number’ is displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | Test slideshow go to slide out of range | Enter ‘10’ in go to slide input and press enter | Message ’10 is outside of slideshow range’ is displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 13 | Test heroes page | Select ‘Heroes of the great war’ from menu | List of heroes is displayed with first one displayed in details pane | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 14 | Test heroes page select hero | Select ‘Private H. W. Richards’ from the list of heroes | The details for this private are displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 15 | Test quiz displayed | Select ‘The Great War Quiz’ from the menu | The quiz start screen is displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 16 | Test quiz start button | Press start button on quiz | First question is displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 17 | Test quiz correct answers | Enter the following quiz answers: 3, 1, 4, 2, 4 | A list of all correct answers is displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 18 | Test quiz name | Enter the name ‘Alex’ in the score box and press the Complete button | The start page is displayed containing the results | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 19 | Test quiz incorrect answers | Press start button to begin new quiz and then enter answers: 1, 2, 1, 1, 1 | The results page is displayed with all incorrect answers | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 20 | Test quiz name again | Enter the name ‘Steve’ in the score box and press the Complete button | The start page is displayed using the results, with the results in the correct order | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 21 | Test search input | Input ‘war’ in search box and press Go button | List of menu items containing war is displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 22 | Test login page | Select login button in top corner | Page is displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 23 | Test incorrect login | Enter incorrect login information and press Login | The message ‘Username and password incorrect’ is displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 24 | Test correct login | Enter following data:   * Username: Alex * Password: password1 | The message ‘Welcome, Alex!’ is displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 25 | Test manage pages displayed | Select the ‘Manage pages’ button in the top right | The manage pages page is displayed | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 26 | Test post page invalid | Select Post from page type drop down, leave the inputs blank, and press Save | Validation messages are displayed for title and content | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 27 | Test post page with valid input | Select Post from page type drop down, input the following:   * Title: Test Title * Content: Test Content * Press Save | Page saved! Message is displayed above post, and Test Title is now displayed in left menu | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 28 | Test post edit | Select Test Title (post) from manage pages drop down, enter the following input:   * Title: Test Title 2 * Content: Test Content 3 * Press Save | Page Saved! Displayed. Also, title is updated on left menu and in drop down. | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 29 | Test post delete | Select Test Title (post) from manage pages drop down, press Delete button, click OK | Page is removed from menu and from drop down | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 30 | Test add image | Select image from form type drop down:   * Title: Test Image * Description: A description * Image URL: images/slide1.jpg   Press Save | New menu item ‘Test Image’ should appear and selecting it shows the new image | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 | Edit image page | Select Manage Pages and select Test Image (image) from manage pages drop down. Change the form to the following:   * Title: Test Image 2 * Description: A description 2 * Image URL: images/slide2.jpg   Press Save | The message ‘Page saved!’ is displayed, the title in the menu has been changed, and selecting it shows the updated page. | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 32 | Test image delete | Select Manage Pages and select Test Image (image) from manage pages drop down. Select Delete button and click OK. | Form disappears, and menu item is removed from menu. |  |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 33 | Test add video | Select Manage Pages, select Create New Page from first drop down, and Video from the second. Enter the following:   * Title: Video Title * Description: Video description * Video URL: videos/basic\_training.mp4 * Content-type: video/mp4   And press Save. | Message ‘Page saved!’ displayed, new menu item appears on left, selecting it takes you to new video page. | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 34 | Test edit video | Select Manage Pages, select Video Title from first drop down. Change the form to the following:   * Title: Video Title 2 * Description: Video description 2 * Video URL: videos/basic\_training.mp4 * Content-type: video/mp4   And press Save. | Message ‘Page saved!’ displayed, menu has been updated, selecting menu item takes you to updated video page. | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 35 | Test delete video | Select Manage Pages, select Video Title from first drop down. Press Delete button and then click OK. | Video page is removed from menu. | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 36 | Test invalid slideshow | Select Manage Pages, select Create new page, select Slideshow. On new slideshow click Add Slide. Leave all inputs blank and click Save. | Validation messages are displayed for the Title and Slideshow | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 37 | Test add slideshow | Select Manage Pages, select Create new page, select Slideshow. Enter input:   * Title: Slideshow 1   Click Add slide twice, enter input:   * Slide 1 – images/slide1.jpg * Slide 2 – images/slide2.jpg   Click Save | Message ‘Page saved!’ is displayed, new item added to menu, selecting it takes you to page showing slideshow and you can move between them. | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 38 | Test edit slideshow | Select Manage Pages, select Slideshow 1 from drop down. Enter input:   * Title: Slideshow 2   Click Add slide twice, enter input:   * Slide 3 – images/slide3.jpg * Slide 4 – images/slide4.jpg   Click Save | Message ‘Page saved!’ is displayed, menu item is updated, selecting it takes you to page showing updated slideshow | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 39 | Test delete slideshow | Select Manage Pages, select Create new page, select Slideshow, and click Delete and click OK. | Slideshow is deleted | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 40 | Test add quiz | Select Manage Pages, select Quiz, enter quiz details and press Save.  Title: Test Quiz  Description: A description  Add a couple of questions… | Quiz is added. | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 41 | Test edit quiz | Select Manage Pages, select Quiz, update quiz details and press Save.  Title: Test Quiz 2  Description: A description 2  Add a couple of questions… | Quiz is updated. | Not as expected, page needs to be refreshed before changes to quiz show |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 42 | Test delete quiz | Select Manage Pages, select Quiz, press Delete and click OK | Quiz is delete. | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 43 | Test add heroes | Select Manage Pages, select Create, select Heroes, add info:   * Title: Heroes 1 * Add details for a couple of heroes.   Press Save | Heroes is saved, menu item appears on left, clicking on it takes you to new heroes page. | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 44 | Test edit heroes | Select Manage Pages, select Heroes 1 page, update info:   * Title: Heroes 2 * Update details for heroes.   Press Save | Heroes is saved, menu item is updated on left, clicking on it takes you to updated heroes page. | As expected |
|  | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 45 | Test delete heroes | Select Manage Pages, select Heroes 2 page, press Delete and click OK | Heroes is deleted | As expected |
|  | | | | |

# Reflective

This section of the report is a reflective look at what I personally learned from the project, looking at coding the app, usability, software design and design patterns.

Before working on this project, I had only limited experience of working with JavaScript. Most of my previous experience had been working with backend languages, such as Java and C#. Therefore, working on this project allowed me to get some hands-on experience of working with JavaScript that I had previously lacked. Initially I was confused about the differences between ES5 and ES6, and so I decided to implement the code entirely in ES5, without using any of the more modern features. This was successful, as it made it clearer to me the differences between the two versions, and how each can be used. Of course, this slowed down development, as sometimes I would need to figure out if a piece of code was being used correctly. This became easier once I moved to using an IDE with analysis tools, which I will cover in the later part of this report. I am looking forward to using ES6 in a future project, so I can make a more thorough comparison, and explore other technologies such as Babel, and TypeScript.

The app in general was a new learning experience as it was the first time I had worked on an SPA written with JavaScript, having previously only worked with backend languages such as C# and Java. Adapting meant changing how I thought about web development from thinking of discrete requests such as GET and POST, and instead thinking of the site like a traditional desktop app with actions triggered by user events. This also extended to the things such as how the initial payload of data was handled, as well as how that data was stored. Without a backend relational database to store the data, I had to change how I thought about storing and retrieving objects.

As I wanted to make sure I understood the technology as much as possible, I decided to develop the app without the use of any libraries or frameworks, such as jQuery. This meant that all code had to be written from scratch. I found that work was quite slow in the beginning, as every line of code had to be researched online, and often a lot of the vanilla JavaScript function calls and APIs are not very intuitive. Overall, I found this approach to be successful, and feel I have a much better understanding of what is going on under the covers in the browser than I would had I used a library.

While the app being developed was a Single Page Application, I wanted to keep some of the aspects of the web which make it more usable, such as the back button, deep linking, and the ability to bookmark pages. This meant learning to understand and use the browser’s location hash property. To do this the site had to be architected so that different content types were loaded dynamically based on changes to the hash property. This was difficult at first, as it increased the complexity of the content loader, however I was able to implement it as intended. This hopefully will increase the usability of the app, but allowing features expected of the web to function as expected, such as the back button. Having basic features that users rely on, such as bookmarking or the back button not work properly, is a big hindrance to usability that hopefully my SPA avoids.

I decided to implement the application using the revealing module pattern. I experiment first with using the plain module pattern, but I found that this required placing a lot of code in the object initialiser at the bottom of the module, which I thought looked messy. Also, when refactoring code, which I try to do a lot, this meant having to change and rewrite code simply to change access modifiers, which was not ideal. Overall the revealing module pattern seemed to be a better fit in this regard. It seems that the various module patterns exist to shoehorn into JavaScript various object-orientated programming concepts that it lacks, and so I will be interested to see if the new futures in ES6 are able to improve matters with its addition of proper OOP concepts such as classes. Overall, I found that the module pattern was beneficial, as it allowed different parts of the system to be developed in isolation and using the modules as proxy namespaces prevented the global scope from being infected with lots of stuff that shouldn’t be there.

When time came to implement some of the more complex parts of the code, such as the quiz-manager or admin-manager modules, it quickly became apparent that a good way of organising the code in these modules was needed. I looked around at various design patterns, I decided to use a form of the state design pattern, whereby an object changes its behaviour based on its internal state (Gamma, et al., 1995). For instance, the quiz module has three states, start, question, and results. Each time you tell the module to display itself, it does this based on its state. The state is represented by a function variable stored in the module. When the quiz start button is pressed the current state variable is changed to point to the function for outputting a question, and then the module redraws itself. I found this was an effective way of organising the code for the modules without them getting out of hand. Replacing my initial spaghetti code with something a bit more organised allowed more complex pieces of logic to be implemented more smoothly.

One requirement of the coursework was that we had to include several specific modules with specific requirements, which did cause some issues at first. It was difficult to know what to put into each one and exactly how it was intended to be used. In the end I had to bend my code a bit to fit with what was expected, and if left to come up with a module architecture of my own I would probably have been able to create something that suited my purposes a little easier. Regardless, I think that what is there fits with the requirements of the coursework.

Another requirement was the creation of a simple login system for the site, to allow an authenticated person to be able to edit the content of the SPA. I was interested to investigate what options if any there were for client-side authentication, as previously I had one done this sort of thing on the backend. After doing some research I realised that there were no ways of securing the client side, as any login system can be circumvented by anyone with even a basic knowledge of JavaScript. Especially in modern browsers like Chrome that let you modify pages easily after they are loaded by right-clicking on the page. In the end it didn’t seem worth even encrypting the password in the file, and while there are built-in hashing functions in the browser, notably the web crypto API, they not well supported yet by all browsers. It appears that if you need any side of real authentication on a website, this must be done on the server side, otherwise it cannot be secure at all.

Something I had little to no experience of before starting the coursework was debugging a JavaScript application. During coding I found the Chrome Web Development tools to be vital in this area, and by selecting the Sources tab in the tools I was able to set breakpoints and step through the code as it was running the browser. I attempted to attach the VS Code editor’s debugger to Chrome, but for some reason could not get this to work reliably. However, the Chrome tools were good enough for me to debug the code without needing anything else. Being able to step through the code step by step was very useful, especially when dealing with some of the more difficult to see runtimes issues, such as parameters being undefined. One of the most frustrating coding issues I ran into was how JavaScript would not generate any sort of error for function parameters being missing, but by stepping through the code in the browser you can easily see which parameters are undefined in the locals and fix them quickly. This helped massively during development.

As mentioned above, I had decided to use only the ES5 subset of JavaScript. As time went on I wanted a way to check that I was not accidently using language features from other JS versions. After looking at a few different tools I found the WebStorm IDE from JetBrains, which has powerful code analysis features and allows you to select the version of JavaScript you want to enforce. Towards the end of development, I switched from VS Code to WebStorm as an editor, and this brought focus on a huge number of issues and bugs that I would otherwise have missed. This allowed me to ensure both that I was using the features of the right JS version, but also that helped me catch hundreds of tiny errors, such as missing semicolons, or unintentional type coercion. Another great feature of this editor was that it had syntax highlighting and checking of HTML inside of strings, which was is something my SPA has a lot of, and against was very useful. By going through each file and fixing every warning I was able to improve the quality of the code a lot and using a tool such as WebStorm will be a requirement for any future JavaScript work I do.

In conclusion I was generally happy with the coding work done on the coursework and feel that I have achieved all the learning goals I had when starting it. My biggest goal was to become more comfortable with JavaScript and get a better understanding of the different versions of the language and how they fit together, and I feel I have managed that. Additionally, building a whole SPA with pure vanilla JS allowed me to learn things and master skills that I would not have been able to be using a framework or library. Additionally, I feel that what I learned about storage, debugging, and using code analysers, I will be able to directly apply to future endeavours such as my honours project.

# References

Gamma, E., Helm, R., Johnson, R. & Vlissides, J., 1995. State pattern. In: *Elements of Reusable Object-Orientated Software.* Westford, Massachusetts: Addison-Wesley, pp. 305-314.