UNIVERSITY OF ST. ANDREWS, SCHOOL OF COMPUTER SCIENCE

CS5003 - Practical 1

Time Tracking

Self Report

Student ID: 150022355

February 23, 2016

Introduction

To begin with, there were some difficulties with trying to implement the Google Drive SDK to store and collect application data. Subsequently, some minor complexities occurred and therefore a decision to focus on the core functionality of the program was put first. With this in mind, the program was set out to follow a MVC design pattern. Specifically, the "*Observer and Listener*" pattern which primarily focuses on a key object (known as a subject), which keeps a universal list of objects depending on its observers. As a result, the pattern can automatically notify the observers of any changes to state. Furthermore, the Observer and Listener pattern was perfect to utilise for this particular assignment.

Key Points

1. Assignment can be located here:
   https://ojd2.host.cs.st-andrews.ac.uk/CS5003-Practical-1/
2. You will need a *Gmail* account to sign in and use the application.
3. If you cannot sign in, you can open *'no-login.html'*.

The Observer & Listner MVC

The MVC pattern follows some simple logic, which is ideal for lists and manipulating lists. For example, when a subject needs to notify the observers about a change of state (deleting an object) happening, it signals a notification to the observers. In addition to this, the subject is updated and keeps track of all subsequent changes to the state. This type of programming was difficult to adjust and implement, so from researching various reports and websites, Addy Osmani, presented some simple baby steps towards building the MVC observer pattern. With this in mind, the program had to be separated into parts where necessary.

0. The Stopwatch

This is rather verbose and non MVC, it is a simple timer which we will use to capture time and bind to our user tasks. The only MVC implementation for the stopwatch are some methods which extend the stopwatch by use of some simple controller event handlers. These handlers will store the stopwatch time upon click. For example: If a user hit's start and then hit's 'stop' - the current time displayed will be passed through via the controller. Then the controller will connect and pass on these methods to the view. The view contains some simple data binding methods which append the time, user task input name, and some other metadata to an area in the HTML. Although the stopwatch element contains no real MVC structure, the code is organised similarly.

To start with, the stopwatch begins by initiating some public variables for the stopwatch to implement. We begin by setting up our timer variables for 'seconds', 'minutes', 'hours' and a global 'interval'. We also include some references to our HTML DOM elements for the stopwatch buttons. Our 'timeData' variable will be used for storing the time when a user hits the stop button.

Beginning with line 53, the code performs a method which calls other methods when a user clicks 'start'. When a user clicks 'start' we begin with incrementing our seconds, minutes and hours using the following global time calculations. When the code reaches line 69, we call our append() and stopWatch() functions. These two functions act like 'the view' model would in a MVC. They simply append the formatted increments from above and add to the HTML. This does our magic and makes the counter tick. The stopWatch() function is called at this point because it uses the JavaScript 'setTimeout' routine.

Both the stop() and pause() functions follow the same principles. The stop() function is executed upon click as well. From here we pause the time again using the 'clearTimeout' routine. Our 'timeData' variable stores the current incremented values and then we append the data to our chosen project in the ProjectList subject created below.

1. The Observer MVC Pattern for Projects & Tasks

First, we begin by setting up some default project methods, this will act as our first subject. A subject maintains a list of observers, facilitates adding or removing observers. In this case, this will be keeping track of adding projects and removing projects which get appended to our projects container in the HTML. The code for this starts at line 159. Next, the code uses prototype's to inherit object instances. The First prototype is for adding projects. Projects will be created as objects (obj) and pushed onto our inherited subject. The subject will then contain objects which can be later manipulated.

As a result, our MVC now needs to move onto the controller. This connects our models for ProjectList() and TaskList() by connecting our prototype's together with our view. For example, we inherit the ProjectList() and give the ProjectList() the ability to add, remove or notify observers on the observer list.

Finally, the view model is where we extend and execute our inherited prototypes. We do this by using concrete observer functions. These methods are used to build upon our subject and appended all changes to the applications HTML.

Final thoughts

Some important decisions had to be made and some conflicts became apparant when appending and extending data from the view. For example, initially, the projects were being appended to the HTML as simple <li> tags. However, it made things difficult when it came to both selecting objects and removing objects. Therefore, a decision to append the projects as <select> options was made. This enabled the user to directly select and trigger more precise event handlers on click / change. Although this seemed the smarter option at the time, rendering the inserted HTML inside of the <select> options was proving costly in other web browsers. Adding and appending task values and project values were built using a mixture of jQuery and JavaScript. Currently the application only lets users add a project with one task at a time. With this in mind, the UI was also a difficult task. Trying to build and compose a smooth UI was a vigorous task. From another perspective, this could of been planned more of before beginning the programming and more research into the Google Drive SDK could of been done.

Key readings

Snook, J., 2009. JavaScript MVC, A List Apart. Accessed 06 February 2016.
Available at: http://alistapart.com/article/javascript-mvc.

Osman, A., 2015. Learning JavaScript Design Patterns, Observer Pattern. Online Book. Accessed 08 February 2016.
Available at:.https://addyosmani.com/resources/essentialjsdesignpatterns/book/#observerpatternjavascript

Mozilla, Firefox. 2016. MDN, WindowTimers.setTimeout(). Accessed 08 February 2016. Available at:
https://developer.mozilla.org/en-US/docs/Web/API/WindowTimers/setTimeout

Mozilla, Firefox. 2016. MDN, WindowTimers.clearTimeout(). Accessed 08 February 2016. Available at:
https://developer.mozilla.org/en-US/docs/Web/API/WindowTimers/clearTimeout