

SE31520
Developing Internet-Based Applications

Enhancing the CS-Alumni Application

Submission Due 4pm Monday, Decemeber 8, 2014

Thomas Mark Rosier (THR2)
110113188

Contents

Application Design	3
CS-Alumni Chrome Browser Extension	3
Rational	3
Module Diagrams	4
Program Operation	5
CS-Alumni Rails Application	10
New Additions	10
Modifications	12
Application Testing	13
Rails Unit Tests	13
QUnit Tests	13
Technologies	16
CS-Alumni Chrome Browser Extension	16
HTML	16
CSS	16
JavaScript	16
Chrome Extensions API	16
Bootstrap	17
jQuery	17
Moment.js	17
Alertify.js	17
Qunit	17
CS-Alumni Rails Application	18
Ruby	18
Ruby on rails	18
Rack CORS Middleware	18
Communication Between Applications	18
RESTful Interfaces	18
Basic Authentication	18
Evaluation	19
Something Extra	20
Attributions	21

Application Design

CS-Alumni Chrome Browser Extension

Rational

My rational behind deciding to develop a browser extension for this project, was that this has been a technology that I have been following for a while and I have not seen it used that regularly for small little applications that interface with a bigger application.

Another reason for my choice to develop a browser extension was that I wanted to learn about implementing some of the more advanced features within the Chrome SDK; for example the use of 'Background Pages' which are used to keep a task running in the background while the extension is closed.

One of the newer features within the Chrome SDK that had also caught my attention in the weeks before starting this assignment was the use of Chrome desktop notifications where you can create a notification within your application that will be displayed to the end user via the Chrome UI's notification draw.

I wanted to see if it was possible to encapsulate some of the key management features of the CS-Alumni application into an attractive and well presented browser extension that also notified the user of any new broadcasts that had been made.

Why did I choose to develop a Chrome extension over a Firefox, Safari or Internet Explorer? The main reason was due to Chris Loftus's preference of Chrome over the other browsers. In addition to that I felt that due to developing for Chrome this also meant that the code will work without modification in Opera. This is a bonus in the real world as it means we can support a greater population of users without investing more money into development.

I also feel that Chrome has a much nicer development environment for developing browser extensions over its competitors, along with having some newer and more unique features that allow the Chrome based extension to stand out.

If this was a real world product it would not take many resources to port the extension to work on Firefox and Safari as well, but this was not within the scope of this assignment.

Module Diagrams

Program Operation

When the user opens the browser extension they will be presented with the home screen. This shows a brief summary about the application along with a disclaimer.

Due to this being an assignment this is essentially a development version of the browser extension. So I have included a link to the Unit tests on the home page to make it easily accessible during the development and testing of the application. If this extension was intended to be released as production code this link would be removed or turned off within the configuration.

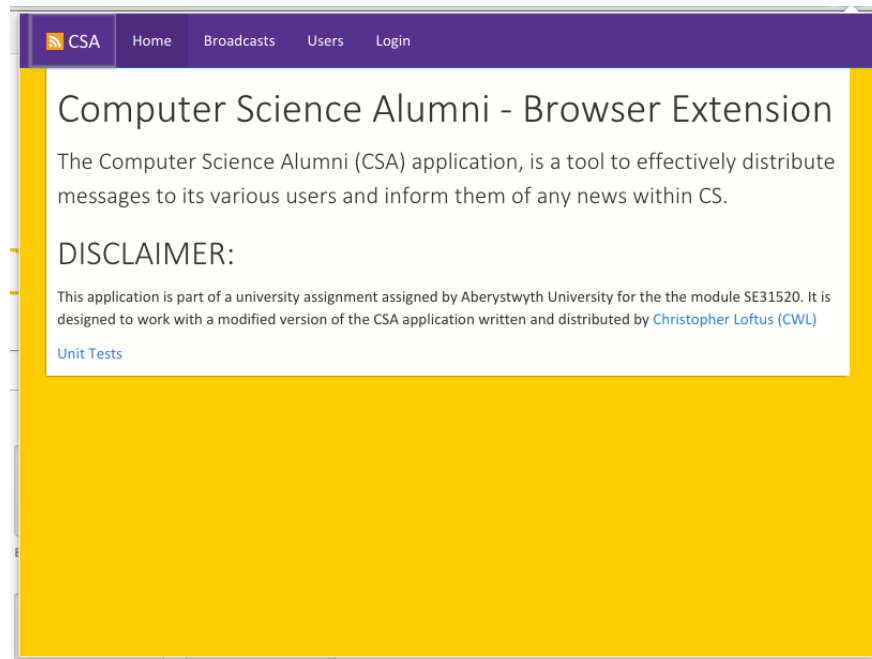


Figure 1: The screenshot above is the landing screen for the browser extension.

From the home page the user can navigate to the 'Broadcasts' tab. In the figure captioned there is no broadcasts available to the user so they are shown a message that says that there are no broadcasts.

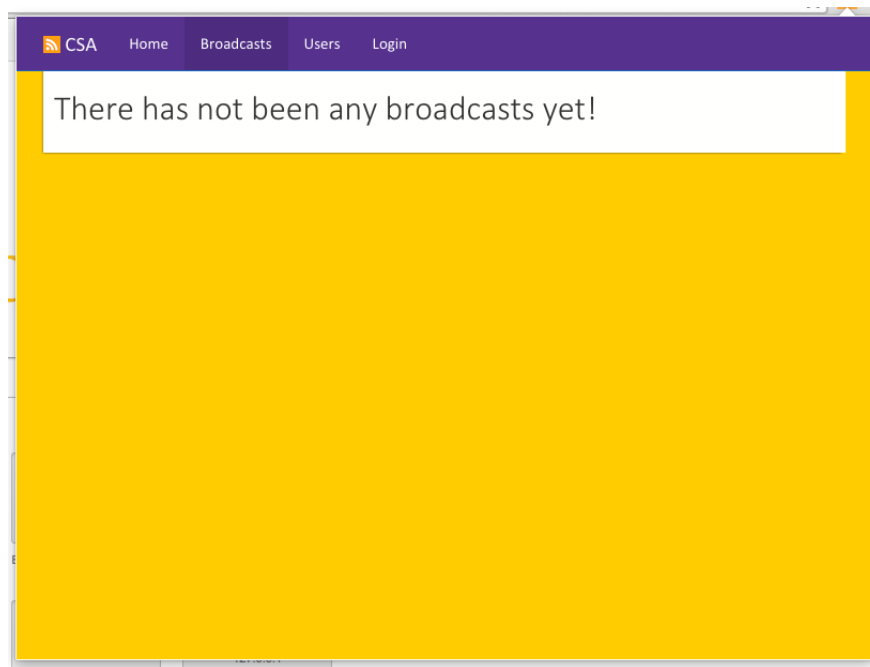


Figure 2: Above we see the broadcasts screen with no broadcasts available.

Again below we see the 'Broadcasts' tab. This time we see how the page appears when there are broadcasts.

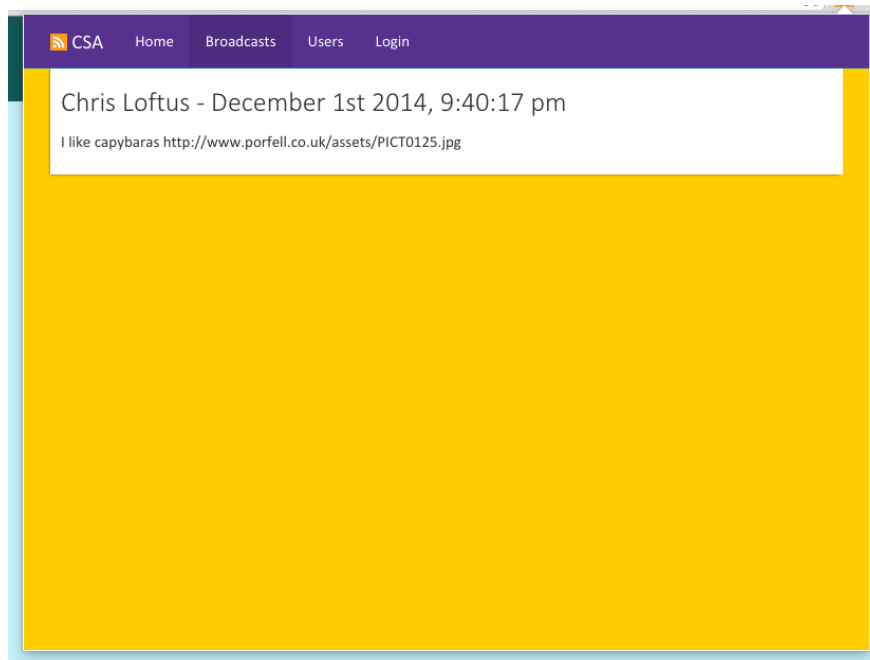


Figure 3: This is what the populated broadcast screen looks like.

This is the 'users' tab of the browser extension where the user can view a specific user's details and edit their information if it is deemed appropriate.

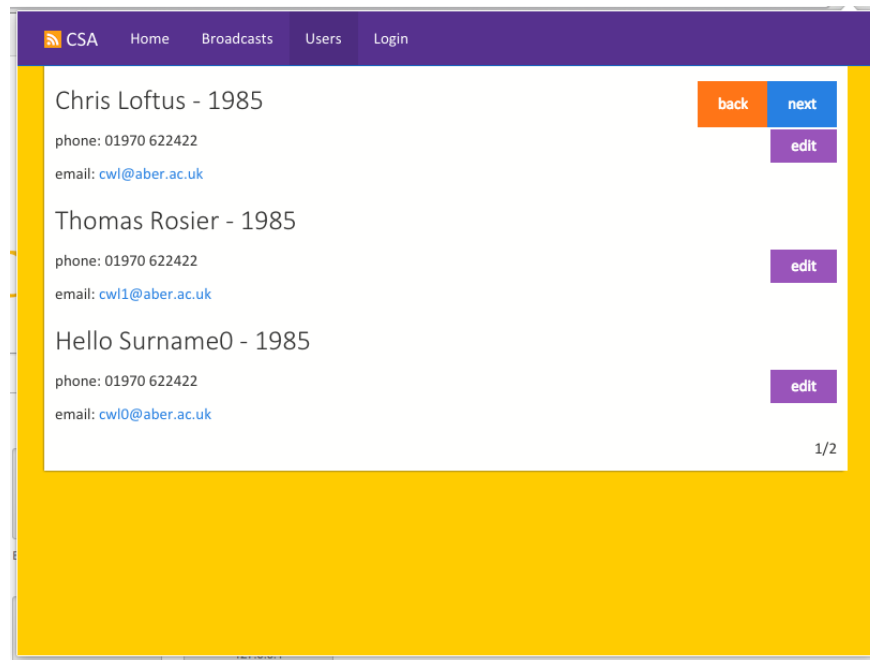


Figure 4: Here we have a screenshot of the user's screen.

When the edit button has been selected for a specific user it shows this modal window that allows the user to edit the details that are kept on the current user they are editing.

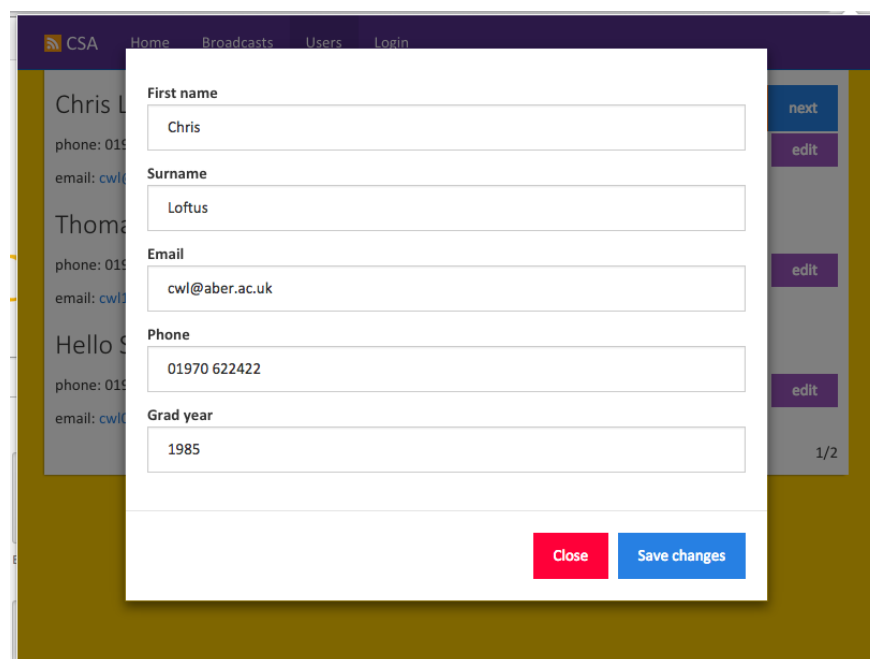


Figure 5: Above we have a screenshot showing the modal window we used to edit the user information.

The 'Login' tab is where the user enters their login credentials that will be used to authenticate the user against the CS-Alumni application.

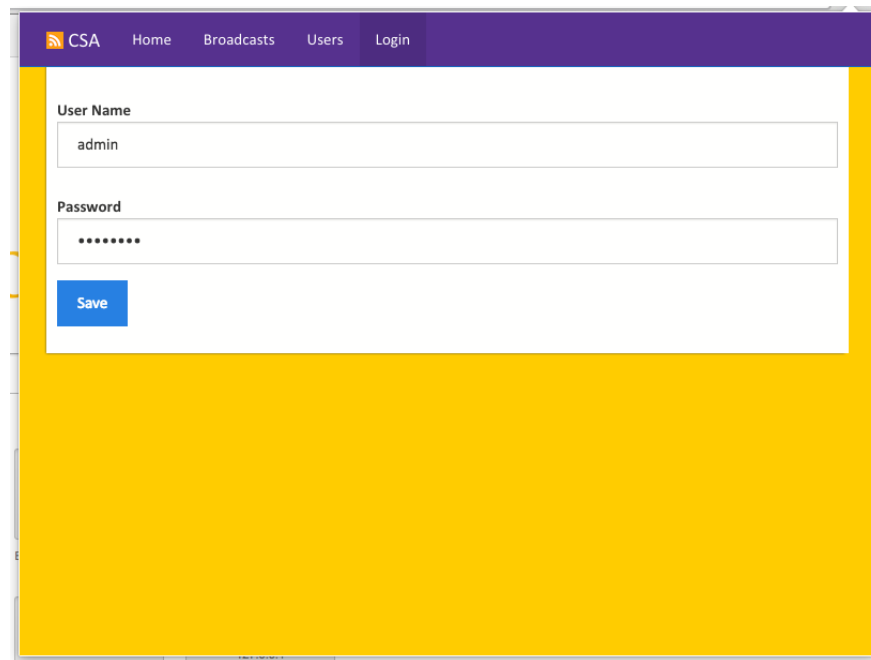
A screenshot of the login screen in the CSA application. The interface has a purple header bar with the CSA logo and navigation links: Home, Broadcasts, Users, and Login. The main content area is white and contains a 'User Name' field with the text 'admin', a 'Password' field with masked characters '*****', and a blue 'Save' button. The background of the page is a solid yellow color.

Figure 6: This is a screenshot of the login screen.

In the screenshot below I have created a new broadcast within the CS-Alumni application to showcase the notification features within the browser extension.

Note: The destination list differs from the standard CSA application. They have been modified to remove the networks that do not work along with adding in the functionality to target the browser extension specifically.

A screenshot of the 'New broadcast' screen in the CSA application. The interface has a dark teal header bar with a language dropdown set to 'English' and navigation links: Home, Jobs, Profile, Users, and Broadcasts. The main content area is light blue and features a large red rectangular text input field containing the text 'I like capybaras http://www.porfejl.co.uk/assets/PICT0125.jpg'. To the right of the input field is a list of notification options: 'General news' (selected with a radio button), 'Jobs news' (unselected with a radio button), 'Email' (checked with a checkbox), 'Extension' (checked with a checkbox), and 'Twitter' (checked with a checkbox). Below the input field is a 'Shortens URL' label and an empty text input field. At the bottom are two buttons: 'Broadcast' and 'Back'.

Figure 7: Creating a broadcast within the CSA application.

Confirmation of the broadcast being created with the parameters that I gave on the previous screenshot is pictured in the screenshot that has been included below.

You can see in the list of feeds that it includes the feed named 'Extension' which signifies that we want to alert users that are using the browser extension.

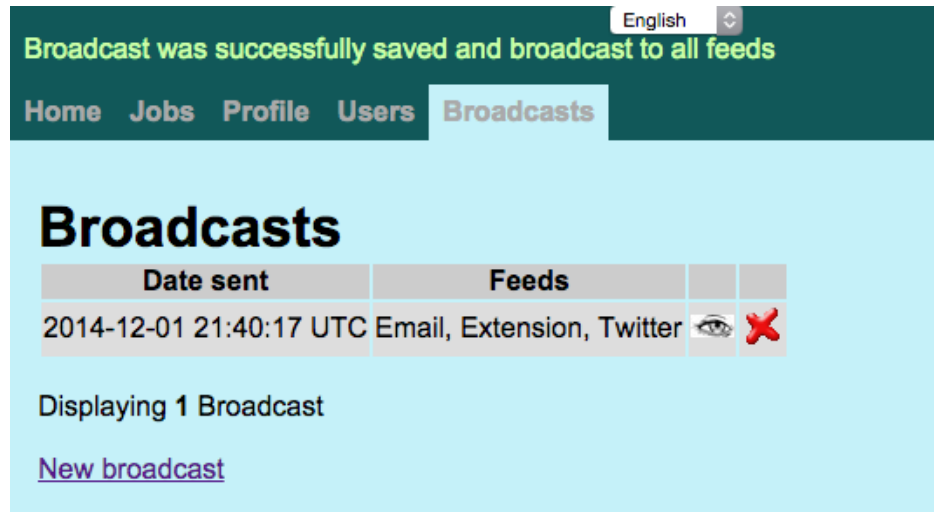


Figure 8: Here is the confirmation that a new broadcast has been created.

Here is shown a Chrome Notification for the broadcast that we just created. This notification will only be shown once when it has been created, then is stored and will not be shown again to the user which will prevent them from being annoyed by repeated notifications telling them the same information over and over.

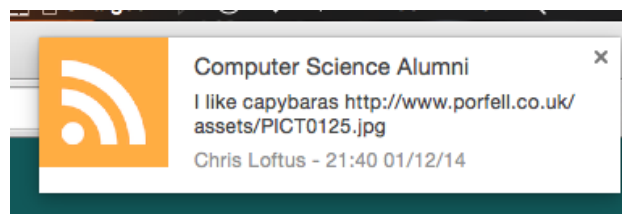


Figure 9: Here is a Chrome notification showing the latest broadcast.

Due to the nature of how the browser Opera is designed it users the same core frameworks as Chrome thus allowing it to use the same browser extensions without any modifications. The screenshot below shows the same broadcast that we created earlier also being displayed within Opera's notification system.

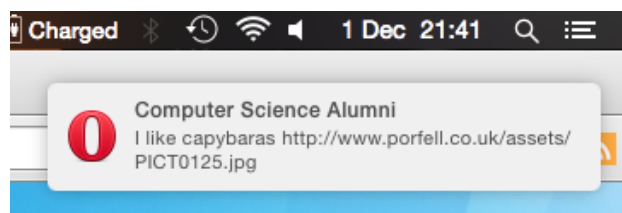


Figure 10: Same again but for Opera this time just to show multi platform support.

Just to confirm that this is a real broadcast and has not broken any of the other functionality within the CS-Alumni application, this is a screenshot of the broadcast on the Twitter website.



Figure 11: Proof that the broadcasts still get sent to Twitter.

CS-Alumni Rails Application

For more details on the changes made to CS-Alumni application please see the 'changelog.txt' file in the root of the project directory.

New Additions

New Database View

For effectively retrieving information from the database to get the latest broadcasts, I decided to create a database view that would join all the relevant information that I required to show a new broadcast.

Using a view allows using the database engine to do all the hard work of getting the data.

```
CREATE OR REPLACE VIEW extension_feeds as

/* Get our data from the database */
select us.firstname || ' ' || us.surname 'Name',
       us.email 'Email',
       bc.content 'Content',
       bc.created_at 'Created',
       bc.updated_at 'Modified'

/* Tables we want to retrieve from */
from   users      us,
       broadcasts bc,
       feeds      fe,
       broadcasts_feeds fb

/* Joins between tables */
where  us.id = bc.user_id
and    bc.id = fb.broadcast_id
and    fe.id = fb.feed_id

/* Query the data */
and    fe.name in ('extension')
```

Figure 12: This shows the database view I created.

New RESTful interface

On top of the new database view that has been created, I have added a RESTful interface to interact with the data that has been extracted from the database view.

This enables effective retrieval of the data that we want to work on, The reason for implementing it this way was that I only wanted to make a single REST call to the CS-Alumni application to get all the currently relevant broadcasts with all the information about the user and the time that the broadcast was published. Please refer to page 18 and the section titled 'RESTful Interfaces' for more information.

```
1  [
2    {
3      "name": "Chris Loftus",
4      "email": "cwl@aber.ac.uk",
5      "content": "Broadcasts are cool things",
6      "created": "2014-12-04T20:10:43.703Z",
7      "modified": "2014-12-04T20:10:43.703Z"
8    },
9    {
10     "name": "Chris Loftus",
11     "email": "cwl@aber.ac.uk",
12     "content": "I like capybaras http://www.porfell.co.uk/assets/PICT0125.jpg",
13     "created": "2014-12-01T21:40:17.979Z",
14     "modified": "2014-12-01T21:40:17.979Z"
15   }
16 ]
```

Figure 13: This shows the JSON object that is returned from the RESTful interface I have added.

Modifications

Changes to Seeds

For 'db/seeds.rb' I have changed the default seeds for the application so that the default password is easier to remember which is now 'password'.

I have also added in the new feed type for publishing exclusively to browser extensions along with the database view for getting the simplified broadcast information covered in the previous section.

Changes to Broadcasts

In the broadcast section of the application I have added in support for posting directly to the browser extension only, along with disabling the feeds that have not been implemented yet.

The other changes consist of adding in support for the new RESTful interface that I have added.

Changes to Security

There has been some changes to the forgery protection within the application, to enable the use of 3rd party extensions to interact via the RESTful interfaces that have already been setup.

To enable the tests to run successfully I have disabled SSL for when the application is within test mode thus enabling the tests to pass correctly without a valid SSL certificate.

I have also added in CORS to enable the ability to do cross domain HTTP requests this ensure that the browser extension can work even when not on the same network as the CS-Alumni application, if this was not enabled then the browser extension would be next to useless.

Changes to REST implementation

Some changes have been made to the RESTful interfaces that server the users information, it now if the user is not an administrator it will return the user own object rather than denying them access to the page which means they can still edit there user account with minimum code changes to the client application.

Changes to Tests

There has had to be some changes made to the test classes due to them not being successful when the CS-Alumni application was distributed to us.

Some new YAML files have had to be created to add in some missing information for the database to authenticate the user successfully with the application along with some of the changes previously mentioned in the security section of this document.

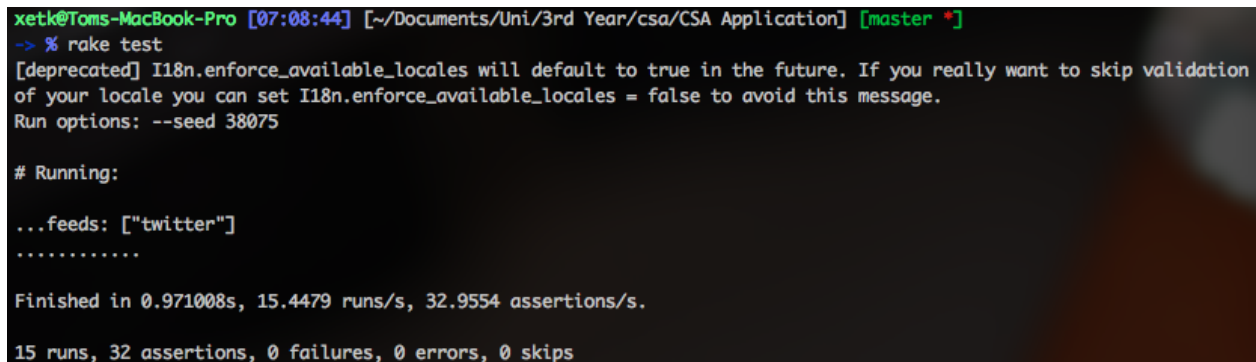
One of the test methods was missing some of the parameters to make the application work as intended.

Application Testing

Rails Unit Tests

For testing the rails application I had to fix the functionality of the included tests to make them work with the rails application. Once these changes had taken place it was possible to complete the tests successfully.

One issue I have run into though is on first run of the rake test one of the tests will fail but on the second execution the tests will work perfectly. My theory behind this is that it is not guaranteed that the tests will run in order and that one of the other tests authenticates with the CS-Alumni application beforehand allowing the transaction to work as intended second time round.

A terminal window screenshot showing the execution of Rails unit tests. The prompt is 'xetk@Toms-MacBook-Pro [07:08:44] [~/Documents/Uni/3rd Year/csa/CSA Application] [master *]'. The command entered is '-> % rake test'. The output shows a deprecation warning about I18n.enforce_available_locales, followed by 'Run options: --seed 38075'. It then says '# Running:' and shows '...feeds: ["twitter"]' followed by a series of dots. The final output is 'Finished in 0.971008s, 15.4479 runs/s, 32.9554 assertions/s.' and '15 runs, 32 assertions, 0 failures, 0 errors, 0 skips'.

```
xetk@Toms-MacBook-Pro [07:08:44] [~/Documents/Uni/3rd Year/csa/CSA Application] [master *]
-> % rake test
[deprecated] I18n.enforce_available_locales will default to true in the future. If you really want to skip validation
of your locale you can set I18n.enforce_available_locales = false to avoid this message.
Run options: --seed 38075

# Running:

...feeds: ["twitter"]
.....

Finished in 0.971008s, 15.4479 runs/s, 32.9554 assertions/s.

15 runs, 32 assertions, 0 failures, 0 errors, 0 skips
```

Figure 14: This shows that the Ruby on Rails tests work successfully.

QUnit Tests

While developing my browser extension for this assignment I decided it was best to take a test driven style methodically to the design and construction of the extension, creating tests and then making the code that works with the test.

I used an external JavaScript unit testing framework to allow the code to be tested efficiently. The framework I used was called QUnit and was developed by the same developers that created the other JavaScript library that I used called jQuery.

For each of the libraries that I wrote to give the core functionality to the application I created their own page of tests. The results of the unit tests at the end of the project are pictured below.

This is the class that handles all of the background interactions with the REST services. This creates notifications for new broadcasts being created.

Background.js – Unit Tests.

CSA Extension - Unit Testing		
<input type="checkbox"/> Hide passed tests <input type="checkbox"/> Check for Globals <input type="checkbox"/> No try-catch		
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.71 Safari/537.36		
Tests completed in 41 milliseconds. 3 assertions of 3 passed, 0 failed.		
1. Background pulls from REST correctly and stores it. (1)	Rerun	19 ms
2. Background Initiates Correctly. (1)	Rerun	2 ms
3. REST generates auth token correctly. (1)	Rerun	1 ms
Back to Testing		

Figure 15: This is the unit tests for the background.js module for the browser extension.

This library is used to get the version of the browser that is in use.

Browser Info – Unit Tests.

CSA Extension - Unit Testing		
<input type="checkbox"/> Hide passed tests <input type="checkbox"/> Check for Globals <input type="checkbox"/> No try-catch		
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.71 Safari/537.36		
Tests completed in 30 milliseconds. 3 assertions of 3 passed, 0 failed.		
1. Check Browser Name (1)	Rerun	2 ms
2. Check Browser Minor Version (1)	Rerun	0 ms
3. Check Browser Major Version (1)	Rerun	0 ms
Back to Testing		

Figure 16: This image shows the browserinfo.js module completing its tests successfully.

Library for handling interaction with local storage.

Local Storage – Unit Tests.

CSA Extension - Unit Testing		
<input type="checkbox"/> Hide passed tests <input type="checkbox"/> Check for Globals <input type="checkbox"/> No try-catch		
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.71 Safari/537.36		
Tests completed in 27 milliseconds. 3 assertions of 3 passed, 0 failed.		
1. Load Nothing (1) Rerun		3 ms
2. Local Storage Save. (1) Rerun		2 ms
3. Local Storage Reload. (1) Rerun		0 ms
Back to Testing		

Figure 17: This shows the LocalStorage.js module passing all its unit tests.

This is a wrapper around some of the AJAX methods within the jQuery library to make interaction with the server a little bit simpler.

Rest Tool Kit– Unit Tests.

CSA Extension - Unit Testing		
<input type="checkbox"/> Hide passed tests <input type="checkbox"/> Check for Globals <input type="checkbox"/> No try-catch		
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/39.0.2171.71 Safari/537.36		
Tests completed in 70 milliseconds. 7 assertions of 7 passed, 0 failed.		
1. Auth Token is built correctly. (1) Rerun		2 ms
2. Rest Get Object (1) Rerun		20 ms
3. Authentication succeeded. (1) Rerun		12 ms
4. Build Auth Token (1) Rerun		1 ms
5. Authentication clear succeeded. (2) Rerun		19 ms
6. Update user record via POST (1) Rerun		2 ms
Back to Testing		

Figure 18: Above we see the RestToolkit.js module successfully completing all its unit tests.

This is a small utility library I have written to share commonly used functions between the various screens within the application.

Utils.js – Unit Tests.

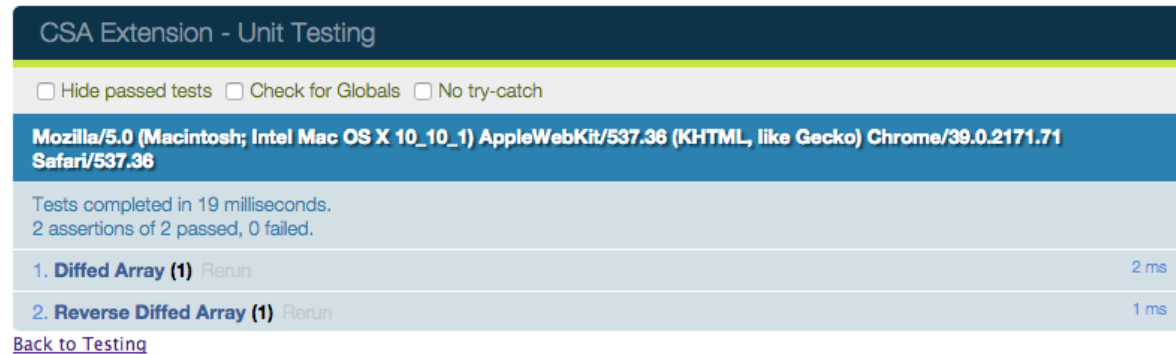


Figure 19: Included above we see the Utils.js module efficiently complete all of its unit tests.

Technologies

CS-Alumni Chrome Browser Extension

HTML

HTML is a markup language that dictates the layout of the screens that are shown within the browser extension it is the skeleton in which the content is placed and displayed to the end user, it has no style to start with it just is the structure of the screen.

CSS

CSS works tangent with HTML and gives the skeleton of a document some style, it is responsible for colour schemes along with the positioning of various elements on the page and how they are displayed.

JavaScript

Is the programming language that the browser extension has been written in, Google's definition for JavaScript is 'an object-oriented computer programming language commonly used to create interactive effects within web browsers.'

JavaScript is an integral part of creating an interactive experience for web based content. It interacts with the various libraries we require to make the application work as intended. It works together with the other web based web technologies within the browser extension to give us total control of the content that is presented on the screen.

Chrome Extensions API

This is the framework that we must adhere to to make the 3 main web based technologies (HTML, CSS and JavaScript) work as a browser extension, the Chrome Extensions API bundles all of the technologies we need to create a viable and fully working browser extension, it takes in the web technologies plus a few extra configuration files to generate a browser extension that will do what the developer intended.

Bootstrap

Bootstrap I see as an essential tool to creating a modern looking website, it unifies the appearance of the application along with giving the ability to make the pages adaptive and work on any screen size. This means that if we were to run the browser extension as a web page it would work on any screen size or any resolution.

I choose to not use the standard Bootstrap theme opting to use a modified one that was supplied by <http://bootswatch.com/> (Theme name is Yeti) I choose to modify this further and make it try and more closely fit the Aberystwyth colour scheme.

jQuery

Along with Bootstrap I also see jQuery as an essential library to be able to create modern and feature full web applications (If I remember correctly all of the other JavaScript libraries I am using rely on jQuery).

Within the application I use jQuery to handle many bits of the functionality within the screens, the key points being aJax requests this enables very simple REST type responses to the server without much hassle, I also use it to edit DOM elements in real time and make them display the information I want them to display in real time.

The jQuery library is so vast that I use a lot of the trivial functions for doing all sorts of other work as the method calls and work that is required is greatly reduced.

Moment.js

This library simplifies interaction with date objects, its is well known that is quite difficult working with date objects within JavaScript, this library takes all the pain out of modifying the dates and putting them into a format that developer actually wants.

I use it for the broadcast times to make them display in the correct way, the objects have been converted into a JSON string and when they are converted back they stay as a string I use moment to convert theses strings into a viable date object that then can be displayed on the screen.

Alertify.js

Alertify is a JavaScript library that makes modern user alerts, this gives well designed and easy to use dialogs to present information to the user.

I have used it within the application to notify when a user has not been logged in and redirect them to the login page, along with showing a message confirming that there settings have been saved successfully.

Qunit

Due to doing a test driving development strategy to developing this browser extension I needed a unit test framework for JavaScript, it is developed by the same developers as jQuery and seems to have strong industry recognition.

I have used this for all my unit testing for the JavaScript libraries that I have written for this project, It has a simple and efficient way of testing the functions along with having a simple and powerful syntax that allows tests to be written without much code.

CS-Alumni Rails Application

Ruby

Ruby is a object orientated program language that is dynamically typed which means there is no strict types for objects, development of Ruby was started in the early 1990's as an alternative to Python and Perl which it is heavily influenced by.

We needed to use Ruby because the application we were supplied was written using the Ruby on Rails which I will cover in the next section

Ruby on rails

Ruby on Rails (RoR) is a web framework that allows the use of the Ruby programming language to create web based applications, it has a strong modularity and is easily extended through the use of Ruby Gems which are small library that can be imported to allow extra functionality within the project. RoR is built on strong model view controller principles, this tries to segregate the code out into the functional modules that they are really responsible for, in theory this means that the code should be more modular and thus more portable work in different situations.

The CS-Alumni application that we were given as a starting point for this project was written in RoR and uses multiple gems to accomplish the functionality that was desired by its original developer.

Rack CORS Middleware

Although the RoR CS-Alumni application used many Ruby Gems already, I had to add in the use an external 3rd party library called 'Rack CORS Middleware' [see reference 4, page 22].

This enabled the use of cross domain REST requests without this being enabled it would be impossible to-do cross domain style interactions.

Communication Between Applications

RESTful Interfaces

For interaction between the CS-Alumni application and my browser extension the most viable method obtaining communication between the two application is to use RESTful web services.

REST uses the standard HTTP protocol and allows the ability to send various data over text formats(usually JSON or XML) the application should be able to request data and be returned back to over a few HTTP requests.

Please follow theses references [see reference 3, page 22], [see reference 2, page 22] for the guides I read to understand how create new RESTful interfaces within the CS-Alumni application.

Basic Authentication

To authenticate the browser extension with the CS-Alumni application it was appropriate to use basic authentication to verify if the user had permission to view the data within the application, this basically consists of encoding the username and password using a base64 encoder, and attaching the encoded password as an extra header on the HTTP requests that are being sent to the server to ensure the user is authenticated. [see reference 1, page 22]

Evaluation

Something Extra



Figure 20: Above is a image of the Pebble smartwatch running the CSA application.



Figure 21: This is a screenshot taken from the Pebble smartwatch.

Attributions

References

- [1] *Basic access authentication*. URL: https://en.wikipedia.org/wiki/Basic_access_authentication.
- [2] Mark McDonnell. *SQLite and ActiveRecord*. Oct. 2013. URL: <http://www.integralist.co.uk/posts/sqlite-and-activerecord/>.
- [3] Daniel Morrison. *Building Awesome Rails APIs: Part 1*. June 2013. URL: <http://collectiveidea.com/blog/archives/2013/06/13/building-awesome-rails-apis-part-1/>.
- [4] Calvin Yu. *Rack CORS Middleware*. Jan. 2014. URL: <https://github.com/cyu/rack-cors>.