ReviewIt - Report

**David Brougham - 1156964**

**James Lord- 1136624**

COMP333 Assignment 3+4

Contents

[Framework – Twitter Bootstrap 2](#_Toc358056142)

[HTML5 Components 3](#_Toc358056143)

[Video Embedding 3](#_Toc358056144)

[Geolocation 3](#_Toc358056145)

[How it works 3](#_Toc358056146)

[CSS3 Components 5](#_Toc358056147)

# Framework – Twitter Bootstrap

For this project we decided to use twitter-bootstrap. Twitter-bootstrap offers many beneficial enhancements with the benefits of the LESS stylesheet language. It also comes with a responsive package when if chosen will make the website responsive to typical resolutions. This allows the web-page to be displayed on multiple devices from traditional desktops to mobile phones while maintaining a layout that is clean and user-friendly. As a requirement was to produce a responsive website, this seemed like a very efficient and easy way of producing a website that meets these requirements. Additionally it is easy to implement with a rails application. With the use of a few different gems (therubyracer, less-rails, twitter-bootstrap-rails) the application can be quickly modified to be ready for the bootstrap implementation. After this, it was just a case of picking and choosing from the vast range of styling options which met the needs of our application.

Figure : Homepage - http://twitter.github.io/bootstrap/

All of these aspects appealed to us, and the simple clean modern look of twitter-bootstrap was attractive and would meet the needs of our application. As our application is very text heavy the need for a clean user-friendly design is important. As bootstrap has a very sleek look with an easy way to override styling options, it is one of the best options for a user-friendly styling option. It’s also feature driven with dropdowns and responsive layout changes that have benefited our application greatly. Additionally, it uses many CSS3 properties. These, when used with the correct browser, offer more styling options and enhancements that would add to the clean sleek look we are aiming to achieve.

Essentially when we were looking for a framework to use, we wanted a sleek professional framework that would style the application in a way that would enhance the application and make it a more user-friendly service. It also had to be easy to implement and not cause too many problems. Bootstrap was able to provide those requirements extremely well. It was therefore the optimal framework to implement.

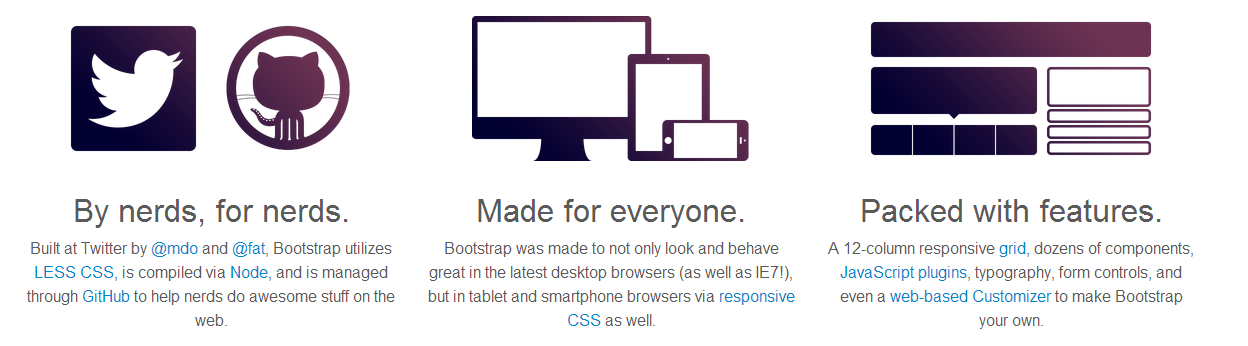


Figure : Homepage - http://twitter.github.io/bootstrap/

# HTML5 Components

## Video Embedding

The video embedding in the ‘About’ page of our app was surprisingly easy thanks to the new video tags in HTML5. The <video> tags mean we no longer have to rely on external plugins to add video content to our site. The HTML5 native tags allowed us to add video easily and since we don’t have to rely on external plugins, the video playback is reliable. Implementation took only a few minutes and would have taken less time still if we hadn’t of encoded the video in a second format.

One of the requirements was that our site should be responsive so we added some css to allow the video to automatically shrink and grow as the page is resized. We wanted our site to run on as many browsers as possible so we also added our video in multiple formats (MP4 and WEBM), which was as easy as adding an extra <source> tag to the HTML. This is an important feature since not all browsers natively support all video formats and the HTML5 video tag allowed us to specify <source> tags for the video in multiple video formats without hassle. Since the video is in multiple supported formats and the video grows and shrinks to fit different screen sizes, this helped us toward the same goal that twitter-bootstrap did: to allow our site to run smoothly and cleanly on as many devices and screen sizes as possible.

The below table shows support for three video formats supported by HTML5. On the left are desktop browsers and to the right are mobile browsers.  
The video formats we used in our app cover every common browser as can be seen by the blue highlights.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Chrome | Firefox | IE9+ | Safari | Opera | Blackberry | Safari (iOS) | Opera Mobile | Firefox (Android) | Chrome (Android) |
| WEBM (VP8) | O | O | O | X | O | X | X | O | O | O |
| OGV (OGG) | O | O | X | X | O | X | X | X | O | X |
| MP4 (H.264) | O | O | O | O | X | O | O | O | O | O |

Figure 2.5: Video codec support in popular browsers. Data sourced from *www.caniuse.com*

## Geolocation

We decided to use geolocation as the main HTML5 component in our application.

When a user logs into the website, they are asked if they wish to share their location. If they respond affirmatively, they will receive a message indicating where they have logged in from.

**Location tracking** action

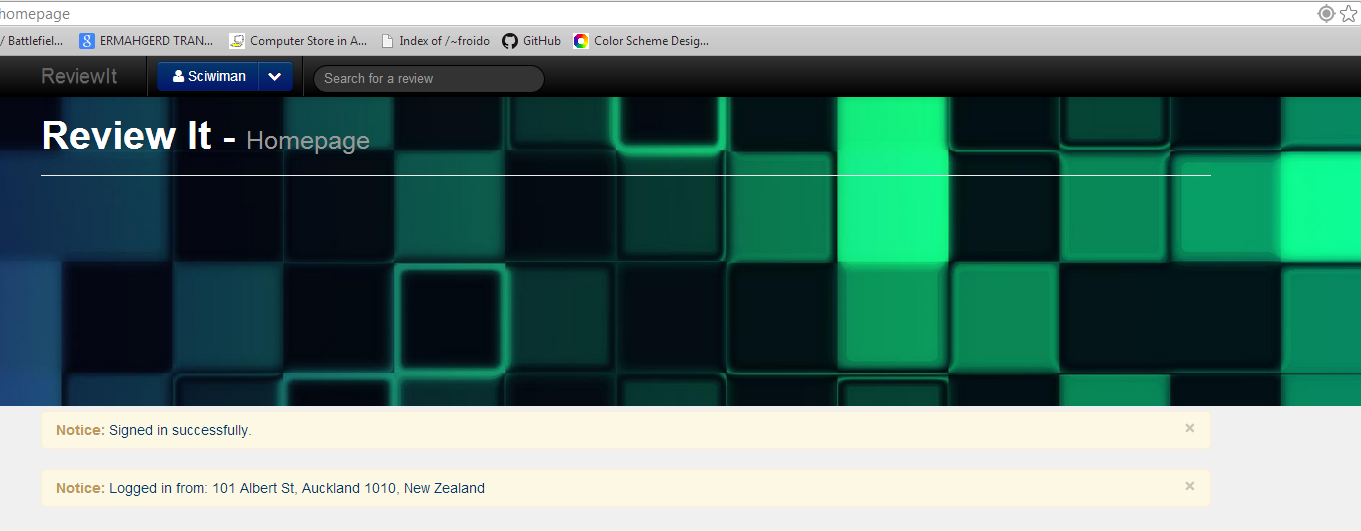


Figure : Geolocation in action

**Message containing current location**

### How it works

Before any of this can work, the Google Maps api needs to be included in the project. This is done by simply including the line:

‘<script src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=false"></script>’

This will retrieve the object so that we can get the location of the user and then reverse lookup to find an appropriate address.

Figure : If there is a notice display the geolocation

Stored within the users subdirectory is a file ‘show.html.erb’. This contains a javascript segment which is the code for the geolocation. It starts by checking to see if there is a notice. A notice will occur on successful login indicating such. This only occurs once, and no other notice will be displayed on this page, therefore on a successful login the location will be displayed. This allows the user to visit their homepage many times in the same session without being hassled with a message indicating current location every time.

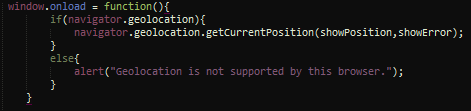
After this, the geolocation can run. Firstly a latlng variable is declared. This is necessary as we will describe later on. Next the window.onload JavaScript function is called to alert the browser to execute the containing code when the page is loaded. In this case it checks to see if navigator.geolocation is supported by the browser and if so, it will retrieve the current position, given parameters ‘showPosition’ and ‘showError’. These parameters are functions. If navigator.geolocation is not supported by the browser, a message is displayed to the user via an alert box.

Figure : When the window loads get the current position

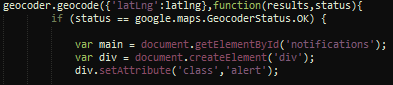
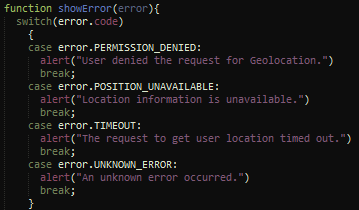
Next, we create a geocoder variable that stores a new instance of the Google Maps geocoder. This is a Google function that will be included by the Google Maps api, a library we included in the application as described above. Now the function ‘showPosition’ is created that takes pos (position) as a parameter. We can now find the position of the user by storing the latitude and longitude of the user in the latlng variable declared before. This is a Google Maps function and gets the co-ordinates from pos, a variable passed into the function. Next, the geocoder encodes the latitude and longitude in the appropriate way for the Google library to understand. It returns a status and the results and these can then be processed in a function. This function firstly checks to make sure the status returned is OK. If so, it then processes the results and displays them to the user via a notice box styled the correct way. If not, it will display an error message to the user and indicate the reason for the error.

Figure 8: Error handling

Figure 7: Displays the data to the user in the correct format

Figure 6: Gets the latitude and longitude of the user

In the last segment, the ‘showError’ function gets passed the error and processes what the error is so as to display the reason for the error to the user in an alert box. The script then ends.

It must be noted that geolocation works best on a wireless connection. This is so we can better triangulate it through other peer wireless networks. We found that the geolocation would retrieve the location of the ISP if there was a wired location from the browser to the internet. This may be resolved if the website were to be published online.

# CSS3 Components

As twitter-bootstrap was a major component of the css styling, a lot of the css3 features were implemented by bootstrap itself. However, to demonstrate that we could implement our own css3 features, a separate stylesheet ‘additional.css’ is included. This contains some css3 features that we implemented ourselves.

Firstly we used ‘background-size:cover’; in the .jumbotron:after element. This ensures the background covers the background irrelevant of its size. This was to keep the banner responsive so it was suitable for any device.

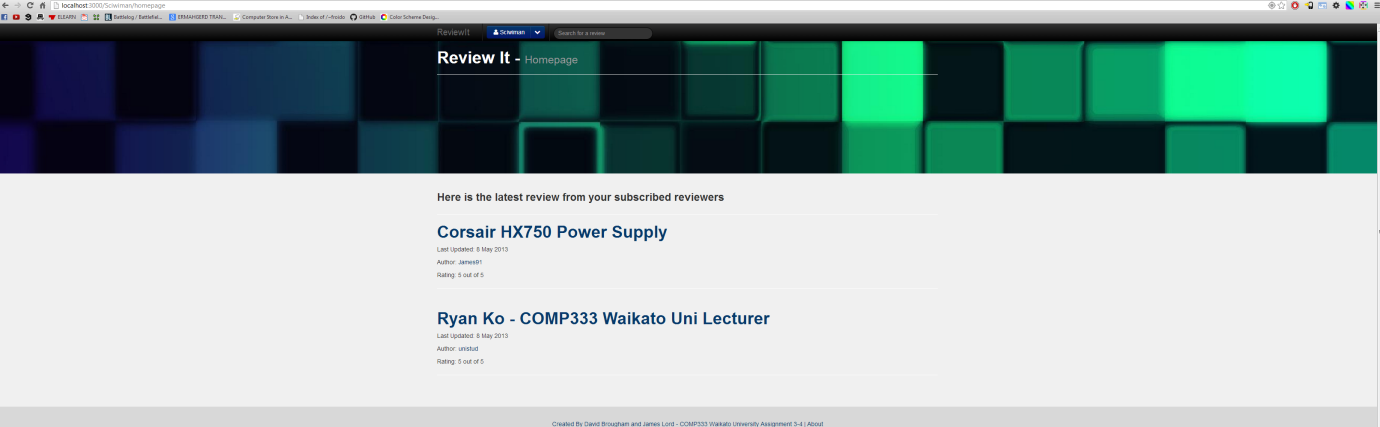


Figure 9: Before CSS3 background-size:cover attribute

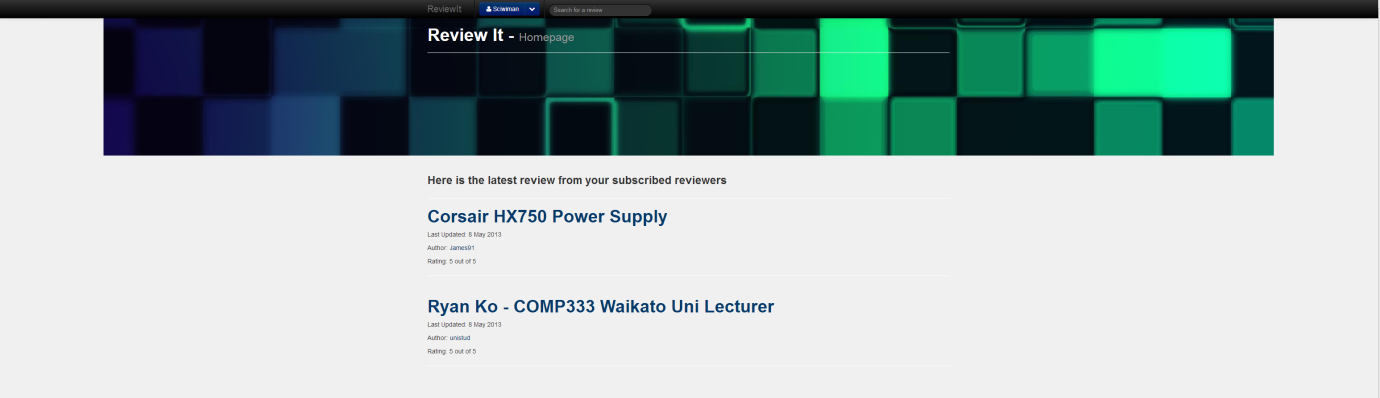


Figure 10: After CSS3 background-size:cover attribute

Another feature that was implemented was the use of ‘transition’. This was to create a smooth hover effect on the title and other elements in the ‘wrap’ div. The code for this is:

‘.wrap, a.brand:hover {-webkit-transition: all 400ms ease-in;}’

This code will transition whatever effect is placed on it over 500ms, and will ease in. That is it will gradually increase in the speed at which the effect takes place. As this cannot be screen captured, its full effect cannot be examined in this report. The a.brand element when hovered on by the mouse will gradually get lighter. This is done by a LESS styling command in the bootstrap\_and\_override.css.less file. It is done with the command:

a.brand:hover{color:lighten(@navbarBackground, 100%);}

This will lighten whatever color the element is by 100% (white). Additionally the a.brand element has a styling option telling it to lighten to 20%. This will create a transition too, from its original color to white whether or not the mouse is hovering on the element.





Figure 11: Transition effect of hover combined with -webkit-transition