National College of Ireland

M. Sc. in Web Technologies

2013/2014

|  |  |
| --- | --- |
| **Student Name** | **Student ID** |
| Patrick Dunlevy | 12116921 |
| Paul Harrison | 00111132 |
| Mark Likeman | 12110141 |
| Niall Ryan | 12114812 |

**BottleRocket**

Project in

Advanced Rich Internet Applications



• Motivation

• Project Scope:

BottleRocket is a Rich Internet Application for fans of music and movies. The application allows users to search for information about musicians and artists, stream videos and music by those artists, search for upcoming music events in their area and find information about new movies that are being released in their locality.

• Area of contribution:

BottleRocket is an entertainment mashup, a concept that was foreign to many only a couple of years ago, but is now extremely common on the web. By taking advantage of massive open datasets, provided by various services through their APIs, and the increasing power of client-side web technologies, BottleRocket offers a unique user experience that is easily implemented across many different platforms.

Web applications have evolved significantly in recent years. Whereas 2-3 years ago an application like BottleRocket would have been easy to create using a framework such as Ruby on Rails, the delivery of the application to user’s web browsers would have been slow, and offered a less positive user experience than the same application could offer if it was traditional native software solution.

BottleRocket takes advantage of the emergence of new technologies and frameworks, in conjunction with various open web services to offer an extremely efficient Single Page Application with all processing and business logic contained on the client to provide users with a fast and efficient user experience. By utilising web technologies, BottleRocket can be used on any device that has access to a web browser and has JavaScript enabled, offering the same user experience, regardless of where the user is; all that matters is that they have a good internet connection and that their device is relatively modern.

• State of the Art Review

The explosion in power of mobile devices in recent years, combined with powerful new and emerging web standards and technologies, has drastically altered the landscape of web applications. While early Rich Internet Applications would have been developed using technologies such as Microsoft’s Silverlight and Adobe Flash, the popularity of smartphones and tablets, which will not run those technologies, combined with the continual exponential increase in computing power generally, has led to W3C standards HTML5 and CSS3 becoming the de-facto technologies for structuring and styling Rich Internet Applications, with JavaScript (ECMAScript) providing the behaviour. The general scope of web applications has changed in recent years also. Static sites with a couple of HTML files, a CSS file and one script are fast becoming forgotten as web applications become more and more complex, utilising the modern web standards I have spoken about and frameworks and toolsets to support developers and ease the cost and time-taken to develop an application. Recent years have seen a move towards complex web applications using time-tested software engineering design patterns to structure the code and make it more modular and testable, e.g. Ruby On Rails, Django, both of which make use of the MVC pattern for developing a web application, but these solutions are all server-based, with the HTML, CSS and JavaScript that the user sees being rendered and pushed to the client by the application server. Recent years however have seen the emergence of toolsets and frameworks that allow developers to easily build complex and modular applications that live solely on the client-side, and communicate with a server-based API to retrieve data, what we know as a modern Rich Internet Application. A RIA solution that is built using only client-side technologies is a very beneficial thing for a company as it allows for some of the complex business logic of an application to be pushed completely to the client, reducing the load on the server, and makes it easy to create cross-platform applications that will work in any web browser as the client-side apps are built using standard web technologies. While the performance of client-side RIAs may have been an issue in recent years, the improvements offered by HTML5 and CSS3 and ECMAScript 5 and 6 make Rich Internet Applications built using these technologies as performant as native solutions while also offering an almost identical user experience.

The growth in popularity of HTML5, CSS3 and JavaScript-based Rich Internet Applications has led to some interesting software engineering problems being created for web developers. As mentioned above, the problem of how to structure a traditional website or web application was almost trivial; early sites and applications contained only minimal static content, and Flash or Silverlight-based solutions had Adobe and Microsoft to rely on for best practices and to understand how to architect a system. However, the problem of how to structure a modern client-side web application has only just become to be understood. While it became usual for web developers to keep all of their JavaScript code in one script file, this is obviously bad practice in a modern web application, where the script file could end up being many thousands of lines long. With the emergence of web development as a software engineering discipline, developers have begun to expect that their code be modular and testable, like a traditional desktop application. The best way to ensure modularity and, therefore, testability is to enforce a commonly-used architecture or design pattern on the system being developed, and so a number of client-side frameworks have emerged in recent months and years to try to help web developers solve these problems which were not traditionally associated with web development.

For this project, Team Frankensteer identified three JavaScript frameworks and toolsets which could be used to help structure and develop a Rich Internet Application to fulfil the criteria for this project; BackboneJS, EmberJS and AngularJS. Each of these tools offer similar helper methods to ease the cost and time-taken to develop a Rich Internet Application, and each enforces or allows for easy enforcement of a design pattern to help define the application structure. It is commonly believed that each of the three is an MVC (Model View Controller) framework but that is not necessarily the case.

BackboneJS is a JavaScript library that provides functionality for interacting with a server / API to pull data into a Rich Internet Application, while also giving that application structure by allowing developers to implement the MVC pattern in their application, using Backbone’s Models, Collections and Views. So, Backbone is not necessarily a framework but a collection of library functions to allow developers to implement functionality, similar in ways to jQuery. While jQuery is a library that allows developers to easily traverse and manipulate the DOM, Backbone is a library for interacting with an API and structuring an application.  
Backbone is used in production by 37Signals (www.37signals.com) on their BaseCamp Mobile product and efficiently helped them create a useful mobile app without relying on native solutions. (Fried, 2014)

EmberJS is a client-side MVC framework for creating “ambitious” web applications. Ember features View templating, which implements the Handlebars library, enforces modularity by allowing developers to encapsulate files as Ember Components and to abstract functions into Ember classes/models, helper methods for easily retrieving data from a server and helper methods for easily creating complex Routing. Ember is an opinionated client-side framework that emphasises convention over configuration, allowing developers to create complex applications as long as they do things the “Ember Way”. Ember’s opinionated nature sets it apart from Backbone as a framework instead of a library and is growing in popularity among certain sections of the RIA community.  
Discourse uses Ember for their client-side implementation and lead developer Robin Ward has cited Ember’s simple documentation, excellent support team and it’s use of string-templating instead of DOM templating as major reasons to choose Ember, adding that the “application really benefited from the framework it provides and the excellent support of the community behind it”. (Ward, 2013).

AngularJS is a “toolset for building the framework most suited to your application development”. (AngularJS.org, 2014) This is an important distinction to draw as it sets Angular apart from the other tools outlined above. Backbone can be considered a library as it extends the functionality with a set of wrappers for JavaScript functions and helper methods to extend the functionality of standard JavaScript. Ember is an opinionated MVC framework which offers developers a number of best-practices for developing a complex client-side MVC application and emphasises convention over configuration. Angular, however, is a toolset that allows developers to structure an application in any way they want, an MVW or MV\* toolset that is completely extensible meaning developers can pick and choose what modules they would like to use to help develop their application . (Shan, 2013).

Shan goes on to explain that Angular “doesn’t require any other separate template engine”. He cites other advantages that set Angular apart from the other two, most notably that it has nested template support and auto binding, but is not hampered by dependencies. Shan concludes his article by stating, in support of Angular, that “it reduces the number of lines in your code more than backbone and ember; if your data model is not huge, it provides you fast rendering and execution; manipulates the DOM directly; makes your code testable and lot more”.

Sebastian Porto’s blog, in comparing these frameworks (as well as a fourth, CanJS) makes similar claims in Angular’s favour. It and Ember both receive “yes” ticks for observables, for their use of routing, their two-way bindings, view bindings and use of partials, and for their use of filtering. (Porto, 2013)

Angular is opinionated but not to the extent that it forces developers to use a certain pattern. Theoretically, Angular can be used to create a completely custom application framework for developing specific styles of Rich Internet Applications. A great example of this is the Ionic framework which integrates AngularJS with PhoneGap for creating mobile applications using JavaScript that can be packaged for Android and iOS devices and sold on the respective platform’s App Stores. While Angular can still be considered to be in its infancy, it has achieved great popularity in recent months and is being used to create many complex Rich Internet Applications in similar domains to the proposed BottleRocket application. An example of this is the Seevl music-discovery platform which uses AngularJS as the front-end toolset for its user-facing Rich Internet Application product, seevl.fm. Originally a standard HTML, CSS and plain JavaScript application, seevl.fm was migrated to AngularJS in June 2013, and this migration led to enhanced performance and a smaller (by almost 60%, a huge saving [Passant, 2014]), more modular and stable code-base that was easily testable.

After careful investigation, Team Frankensteer decided to use AngularJS to create the BottleRocket application, due in part to the familiarity certain team members already had with the toolset, but also due to the large community and ecosystem of modules that exist for helping to create Rich Internet Applications using the toolset. The Angular-Seed application structure template was also a big factor in deciding to use Angular as it helps to enforce an MV\* pattern and best-practices which would allow for easily separating the different aspects of the application code into modules which ensures that each layer of the application is testable, independent of other modules. Angular-Seed is extremely helpful as it solves the problem of how to structure a complex JavaScript-based Rich Internet Application, a new problem in this domain.

• User Interface Design

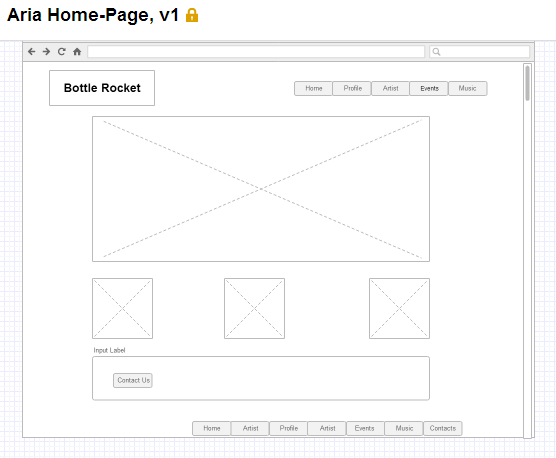
The UI design for Bottle Rocket had to be clear and easily understood by the user, and ultimately provide a successful and enjoyable UI experience. We focused on the user, and how they could achieve their goals using the Bottle Rocket application. The UI had to focus on anticipating what users of our app need to do and making sure that the interface has elements that are easy to access, understood. In choosing the interface elements we wanted them to be consistent in the choice of the layout for our application, and this would help with task completion such as find a music event, searching for music artist, searching for music or bands. Some of the interface elements we wanted to include are as follows:

* Input Controls - such as buttons, textfields, searchboxes
* Navigation Components – nav buttons, search fields

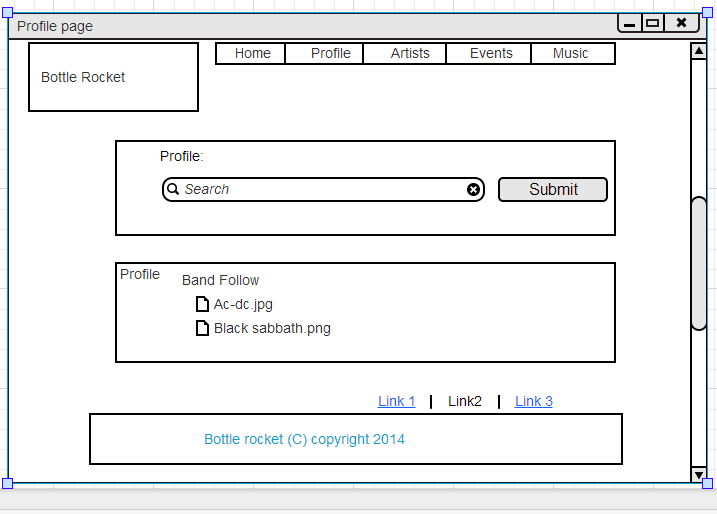
These elements help to display the content to the user, and they don’t need to guess what the elements do on the page. We didn’t include any dropdowns on the main top navigation menu as we didn’t want to force the user to guess what was within the dropdown. We also wanted to keep the interface simple, or that it is almost invisible to the user. The UI elements also had to be consistent and responsive, and help users feel more comfortable and allow them to use the application easily or not get lost. We wanted users to find what they are looking for quickly on the Bottle Rocket app, and then be able to apply this consistency to the rest of the pages. We were purposeful in the layout of the pages, and we were careful with the placement of items so that they would draw attention to features of the app to the user, such as searching for artists, events and bands etc. Colours were carefully used for the layout as they direct attention to items using colour and light. We also used typography to create some clarity and chose font size to help increase the scanability and readability.

We also used some AngularJS animations to change state when the user visits different pages, which communicates to the user that something is happening on the pages. All these helped us arrive at the design of our UI design for our application.

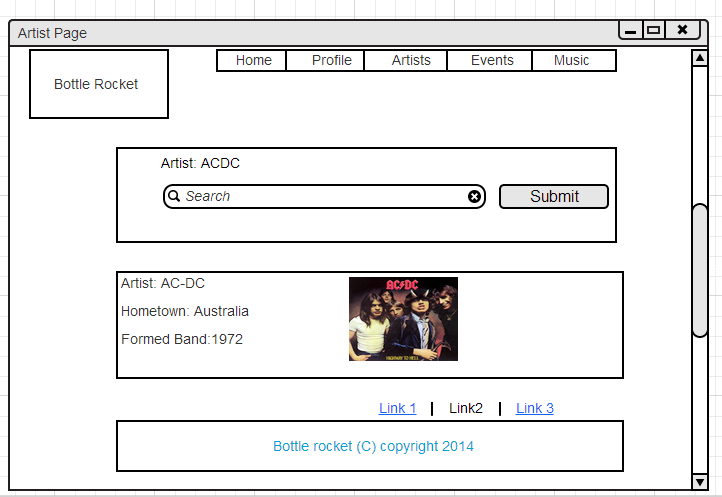
Below are wireframes or mock-ups for the Bottle Rocket application. The wireframes are used as layout page schematic or layout blueprint. The wireframes act as a visual guide that represents the skeletal framework of the website.



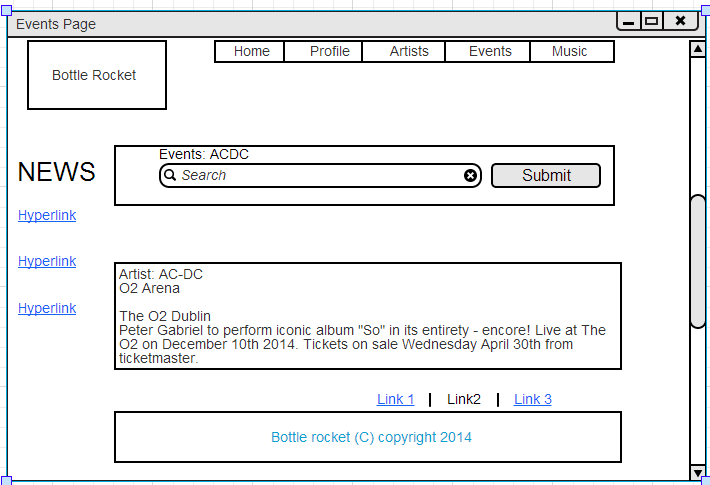
Profile Page

****

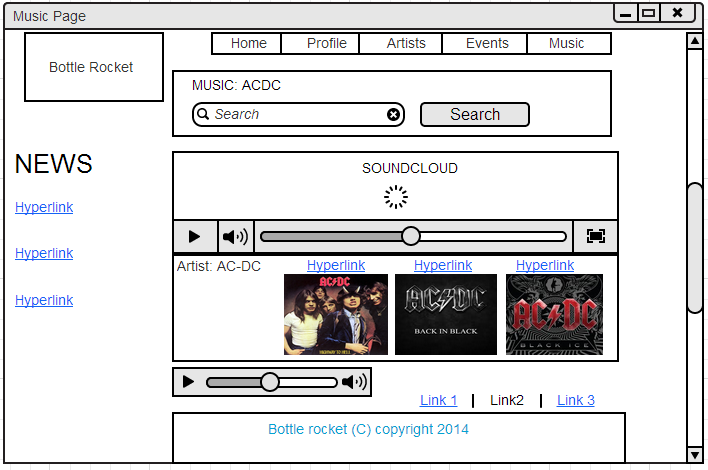
Artist Page

****

Events Page

****

Music Page

****

• Architecture

• Application Architecture:

As we decided to develop the BottleRocket application using the AngularJS toolset, the application architecture was very easy to decide upon. AngularJS allows developers to create an extremely modular application structure and provides a template application for developers to build upon with many best-practices and examples already in place, known as Angular-Seed, made available from the AngularJS website by the Angular Core Development Team. Angular-Seed provides a simple and efficient way to enforce the MVC pattern in a client-side application using AngularJS, allowing for efficient modularity and testability, and solving the problem of how to structure a JavaScript application of huge scale and scope.

Angular-Seed creates a scaffolded application with separate Configuration, Service, Controller, Directive, Filter and Presentation modules. The app configuration file, app.js in BottleRocket, is where each of the modules that make up the application are injected as dependencies in the core application module. The Configuration module also contains routing logic for the application, allowing us to easily add new templates and pages and controlling the AJAX requests which make BottleRocket available as a Single-Page Application.

The Service module is where most of the business logic of the application lives. In the case of BottleRocket, most of the services are AJAX requests using Angular’s $http service. The service module can be seen as similar to the Model folder in a Ruby on Rails application. There are four different types of Service objects that can be created: Constants, Values, Services and Factories.

Directives are unique to Angular and allow encapsulation of HTML and CSS element templates with JavaScript functionality, allowing users to create their own custom elements and call them in the view code e.g. <custom-element></custom-element>. Directives are one of Angular’s “wow” features but are not used extensively in the BottleRocket application due to limited development time and also due to the nature of the application; the benefits of Directives are encapsulation for re-use and easy testability but there aren’t many objects re-used extensively in the application and refactoring the code into directives would have added un-needed complexity.

The Controller module is where the business logic from our services is called into play. As in most MVC applications, our controller module can be seen as the glue that holds the application together, calling the data from our services/models and passing this data to our presentation layer, using Angular helper methods. Some of the controllers also contain business logic due to the implementation of SDKs which fall outside of Angular’s $digest cycle (basically, as the SDKs are not Angular modules, Angular is not aware of them), but careful refactoring of some of the SDK code into services and directives would solve this problem. However, due to time constraints, some of this code still lives in the controllers.

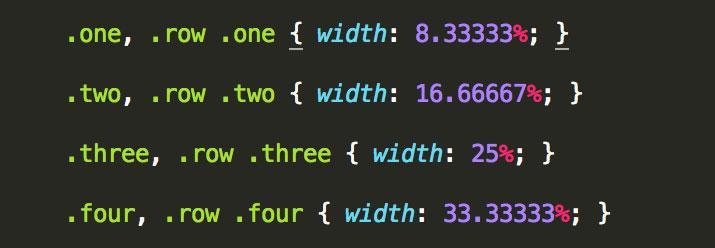
• Security:

One of the problems faced during development of the BottleRocket application was how to pull in data from different sources without falling victim to Same-Origin JavaScript errors. Angular’s $http.jsonp helper methods were used extensively to pull data securely from external web services and this is elaborated on in the Data Strategy section below.   
Another related problem in this area was what to do if our requests to a web service failed so we made use of $http’s “.success().error()” functions to catch an error if the request failed. This is used successfully in the Artist Ctrl as the requests made to the Seevl API have been failing recently due to the platform migrating to different server infrastructure. The controller catches the error and populates the view with dummy data for the purposes of demonstrating the application but in production, ideally this error() function would return a modal that would inform the user that the service is down and offer them a different choice or send them to another part of the application.

• Toolkits and Frameworks:

As mentioned above, BottleRocket was developed using the AngularJS toolset in conjunction with Angular-Seed to create a complex Rich Internet Application that is enjoyable and easy to use for the end user, but easy to test and maintain for the developers. Angular-Seed contains a number of tools to make the development process easier and these are elaborated on in other sections of the application.

We designed the Bottle Rocket app to offer a RIA UI design to the user, and to create a responsive and intuitive experience. To help us design the app this way we had to be clear about how the application structure, screen design, UI controls, and overall interaction design for the app was going to be. The application was built using a very responsive UI or Zurb Foundation around AngularJS. This allowed us to build an advanced client-side application. The Bottle Rocket application was developed using an AngularJS rich Internet application framework. AngularJS which is a JavaScript framework made it easier for us to implement our RIA application.   
We wanted to make our application responsive so we decided to use Zurb Foundation which is a robust and advanced responsive front-end framework. Zurb also would enable us to have an application that would be responsive to mobile devices. It is based on a grid system and that is easy to use, and adapts very well to different viewports. By default the grid is 940px wide when it is at its full width. The grid itself is fluid so as the viewport narrows, the grid adapts accordingly. This means that each column in the grid does not have a static width but is assigned using percentages, and is similar to Skel.js which is another responsive framework and which has a grid that also works off percentages the same way as Zurb does. An example of Zurb Foundation CSS shows this:



In order to use Zurb Foundation with AngularJS in the Bottle Rocket application, we needed to include a module to convert Foundation’s JS into Angular directuves. We used build files from the Angular-Foundation project (https://github.com/madmimi/angular-foundation/)to implement the framework in our application. Once all the files were downloaded we just had to declare a dependency on the mm.foundation module:

angular.module('bottleRocket', ['mm.foundation'])

Once this has been done we simply used the Foundation grid for the pages based on the Wireframes. It was very easy to apply the Foundation grid structure to the application, and most of the heavy CSS layout is done for you.

BottleRocket also makes use of jQuery as Foundation requires it as a dependency but it is not extensively used in the application as a lightweight subset of jQuery called jqLite comes bundled with Angular to allow for DOM-manipulation inside the controllers and directives.

• Data Transfer Strategies:

The BottleRocket application makes extensive use of APIs and web services to get the data that it displays to users. The Seevl API, combined with the YouTube API, provides the functionality found on the Artist Page, allowing users to search for any musical artist and retrieve information about them and see their highest ranking video on YouTube. The Events page makes use of the BandsInTown and Songkick APIs, combined with the HMTL5 geolocation API to provide users with information about upcoming musical events in their areas. The Movies API makes use of the Rotten Tomatoes API to pull in information about upcoming cinema releases. The Music page makes use of the SoundCloud SDK to allow users to search for and play music hosted on SoundCloud. As we developed the application using the MVC pattern, all of these service requests take place in our models (except for the SoundCloud requests which make use of SoundCloud’s SDK which we have not have the time to refactor into services and directives), which, following AngularJS best practices, are contained in our Services module. The requests are made using Angular’s $http helper methods, and make extensive use of JSONP for easily and securely making requests to external APIs without needing to set complicated headers to prevent JavaScript Same-Origin or CORS errors. Keeping these requests in our Services module makes them easily testable, and moving them out to services means the data is easily shared between the controller and directive modules.

• Evaluation and Testing:

Automated Testing

Due to the modular approach that AngularJS allows developers to use, testing applications created with Angular is easy, or at least easier than testing other client-side apps, and the Angular-Seed application template includes a number of helpful tools for creating and running tests. Angular uses the Behaviour Driven Development tool Jasmine for testing the functionality of the application, and employs the Karma test runner for running the Jasmine specs. Testing modules isolation from others is very easy to do with these tools and allows for simpler implementations.

Testing the Services module was made easier by the fact that the AngularJS core modules come with a modules called ngMock which is specifically designed to mock up data to represent the response to HTTP requests from inside the application. This is extremely useful as it means that other tools such as Sinon need not be used, which saves a lot of development time as configuring Sinon, or indeed any other outside tool can be a pain to do in Angular, as to take advantage of the application dependencies the external module needs to be integrated into Angular as its own module (e.g. in a Service or Directive), but integrating this in a test module can be even more complicated as the developer often has to inject and create new scopes and objects manually.

The BottleRocket test suite uses ngMock to test the Services module of the application, and currently has full and passing test coverage. Each specific service call is inside its own spec and there are sub-specs for testing each of the methods a service object has (if any).  
The suite could be improved by implementing tests to cover failing requests as at the moment it only covers server responses of 200.

We have also created many unit tests for our controllers in this application, but many of these tests currently fail. Testing controllers in AngularJS is trivial if the controllers are lightweight but becomes extremely complicated once the controllers begin to depend on more and more modules. As such, our controller tests contain a number of simple specs for some basic functionality but the testing of some of the more complicated controllers would take many more hours of work to complete, but due to the time constraints of this project, we have not been able to complete them to the standard we would like.

Usability Testing

Usability assesses how easy user interfaces are to use. Usability can also be defined by 5 components:

Learnability – We designed the application to be easy to use and whereby users would be able to accomplish basic tasks, such as searching for music events and music artists etc. the first time that they use the application.

Efficiency – Once users have learned to use the application, they should be able to use it more efficiently and know how to search for events etc. or perform these tasks more quickly. The application also needs to enable users to accomplish their goals completely and accurately, and which are measured in time. In testing the application, there were no problems with pages loading etc.

Memorability – When users of the application have not used it for a period of time they should be able return to using the app easily or re-establish their knowledge of using the app or not forget how to perform tasks.

Errors – The application enable users to recover from any errors they they experience while using the application.

Satisfaction – This is how pleasant it is to use the design of the application. It also refers to the perceptions, feelings when users are using the application.

The Bottle Rocket application had to be designed so that it can be easy to use, because if it’s too difficult to use then users will not want to use it, such as getting lost while using the application. Our app had to be clear in what it’s offering and what users can do with it. We developed the interface to be easy to use and where they wouldn’t figure out how to use it. In going about usability testing for our application we collected some feedback as a way to evaluate the Bottle Rocket application, and we observed users interaction with the app. The feedback from these tests helped to improve the design and performance of the application.

The importance of usability testing for our application would allow us to reveal problems with the application design and functionality, and also with navigation issues, or reveal possible issues with how users were able to complete tasks while using the app. During observation users of the application, 5 in total for usability testing purposes, were able to navigate and the all without any difficulties, and users encountered no stumbling blocks when searching for artists, events when using the app. We gained additional feedback using quantitative usability metrics, this included the amount of time or the number of clicks took for the tests users to complete a given task. Our initial participants for our usability testing were 3 average users of websites / web applications and 2 advanced more sophisticated users.

We put together a SUS survey for measuring usability of the app. It consisted of a 10 item questionnaire with five response options for respondents, from Strongly Agree to Strongly Disagree.

The SUS score achieved from testing the users was a score of 80%, which is the top 10% of scores. This was also the point at which all of the users during our usability test would more than likely recommend the Bottle Rocket application to a friend.

• Summary

We are extremely pleased with how the development of this project went. While using AngularJS to full effectiveness can prove to have quite a steep learning curve, we managed to create a complex, responsive application that used many of Angular’s complicated features quite effectively. While the application is not as fully features as we would like, with more development time many of the features that were originally planned but left out could be easily implemented and the testing suite made much more comprehensive.  
Other future plans for the application would be to re-factor some of the controller code into Angular services and directives to allow for full testability, to hook up some service calls to some pages to allow for a more integrated application environment. Finally, we would further explore the Ionic Framework for combining Angular and PhoneGap to allow for placing the application on various mobile stores to take advantage of that market.

• References

Angularjs.org. 2014. *AngularJS — Superheroic JavaScript MVW Framework.* [online] Available at: http://www.angularjs.org [Accessed: 03 May 2014].

Backbonejs.org. 2014. *Backbone.js.* [online] Available at: http://backbonejs.org [Accessed: 03 May 2014].

Emberjs.com. 2014. *Ember.js - A framework for creating ambitious web applications.* [online] Available at: http://emberjs.com [Accessed: 03 May 2014].

Ford, B. (2013). *Huuuuuuge Angular Apps.* [online] Available at: http://briantford.com/blog/huuuuuge-angular-apps [Accessed 3 May. 2014].

Fried, Jason, (2014). *Launch: Basecamp Classic Mobile by Jason Fried of Basecamp.* [online] Available at: http://signalvnoise.com/posts/2761-launch-basecamp-mobile [Accessed 3 May. 2014].

Genev, M. 2013. *Backbone or Angular or Ember? Here is my choice and why - { 100PercentJS }.* [online] Available at: http://www.100percentjs.com/backbone-or-angular-or-ember-here-is-my-choice-and-why/ [Accessed: 03 May 2014].

GitHub, (2014). *madmimi/angular-foundation.* [online] Available at: https://github.com/madmimi/angular-foundation/tree/gh-pages [Accessed 1 May 2014].

Ionicframework.com. 2014. *Ionic: Advanced HTML5 Hybrid Mobile App Framework.* [online] Available at: http://ionicframework.com/ [Accessed: 03 May 2014].

Meyers, C. (2013). *Code Organization in Large AngularJS and JavaScript Applications.* [online] Cliff Meyers. Available at: http://cliffmeyers.com/blog/2013/4/21/code-organization-angularjs-javascript [Accessed 3 May. 2014].

Orsini, L. 2014. *Angular, Ember, And Backbone: Which JavaScript Framework Is Right For You?.* [online] Available at: http://readwrite.com/2014/02/06/angular-backbone-ember-best-javascript-framework-for-you#awesm=~oyNAWtlyhwud92 [Accessed: 03 May 2014].

Passant, A. (2014). *Twitter / terraces: Having fun w/ @CoffeeScript. ....* [online] Twitter.com. Available at: https://twitter.com/terraces/status/447367876681678849 [Accessed 3 May. 2014].

Porto, S. 2013. *A Comparison of Angular, Backbone, CanJS and Ember - Sebastian's Blog*. [online] Available at: http://sporto.github.io/blog/2013/04/12/comparison-angular-backbone-can-ember/ [Accessed: 03 May 2014]

Ward, R. (2013). *Why Discourse uses Ember.js - Evil Trout's Blog*. [online] Eviltrout.com. Available at: http://eviltrout.com/2013/02/10/why-discourse-uses-emberjs.html [Accessed 3 May. 2014].

ZURB, i. (2013). *Foundation | The Most Advanced Responsive Front-end Framework from ZURB.* [online] Foundation.zurb.com. Available at: http://foundation.zurb.com/ [Accessed 1 May 2014].