Department of Informatics,
University of Leicester

CO3015 Computer Science Project

An Interactive Web Application to enhance learning conditions in lectures

Dissertation

Author
Benjamin Thomas
Meysner

April 29, 2018

Declaration

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other peoples work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s).

Any part of my own written work, or software coding, which is substantially based upon other peoples work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s).

I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Benjamin Thomas Meysner

Signed: [illegible]
Date:26/04/2018

Contents

List of Figures

1 Abstract

My personal experience as a student has often led me to question the reasons for absence of motivation in students to participate in presentational lectures. Lectures often account for a large proportion of a student's time-tabled events and are integral to a student's progression through university. They provide an opportunity to listen to experts and most importantly provide a platform for communication between peers.

I do not consider the lack of enthusiasm to participate or even attend lectures to be exclusive to a student's lack of interest in their studies. I do, however, believe that the issue is more complex and concepts such as learning styles and feelings of anxiety have a larger impact on a students engagement.

In an endeavour to address this, I have developed an interactive Web Application which is designed to enhance live presentations, primarily for use within educational institutes but is flexible enough adapt to changing requirements. Core features of this application allow for users to create interactive events on the "fly" with the ability for other users to join them and post messages, in real-time with confidentiality. Incorporating various high quality interface components provides users with the correct balance of good usability and visual stimulation.

This dissertation explains the project life-cycle in its entirety, from design to deployment and the reasons for choices made during its course.

2 Introduction

As with many educational courses, needs and training, the requirement of adopting effective methods of transferring information from a singular source to a large quantity of people is an extremely important one. Whilst there are different ways and techniques of accomplishing this, the natural solution to such a problem in many cases is using a presentational format. This however, is not suggestive that it is the most effective, rather that it may be the most time efficient. Spending large amounts of time as an audience member of such events has credited me with the ability to recognise the possible ineffectiveness and associated problems that arise. One such problem originating from this is what is known as Glossophobia. Glossophobia, or public speaking anxiety, is one of the most prevalent world fears, affecting approximately 75% of the population. Statistically, more people claim a fear of public speaking than a fear of death [1]. With many lectures containing in excess of 100 individuals it is difficult to find fault in this. Public speaking often has adverse effects on a person. Such anxieties may stem from wrongly assuming that their participation is irrelevant, unnecessary and distractive, in which case they refuse to communicate with the presenter. Participation must be encouraged in whatever form. The issue in question often has individuals exiting presentations lacking confidence in themselves, the presented material and the presenter, which in turn can have a negative impact on their performance, especially if in an educational capacity. The notion of differing learning styles also arises, being stationary and listening to an expert discuss complex ideas for long periods without any interaction could have the ability to detach even the most engaged students. Also, if people in the audience aren't communicating with the presenter then how does he or she know where there are potential learning gaps? It is quite possible that there may be many people equally struggling to grasp the same particular concept. Analysing collective data from an audience is vital for presenters to improve future presentations.

2.1 Aims of the Project

The intention of this project is to develop a robust and secure Web Application which serves as an additional tool for improving communication between the audience and presenter during live presentations. Users of this application will need to have the ability to easily create events through an interface which are associated to their live presentations. Audience members will then be able to join these created events and communicate with real-time messaging through their devices in a safe and friendly environment. The application will provide graphical representations of data for each event which can assist the improvement of live presentations. The interface of the application will be streamlined and lightweight to ensure swift access to its features. Additionally, it should be visually stimulating for a enjoyable user experience.

2.2 Objectives

In order to achieve my aims, I will need to break them down into the following objectives :-

1. Develop a Web Application using a *MVC* framework
    (a) Setup a database for persisting user and event data
        i. Independent research into deciding the right database for my application
    (b) Apply Agile Project Management
        i. Exploration into unfamiliar testing frameworks
        ii. Vigorous test suite consisting of unit, integration and system tests
        iii. Continuous Integration
    (c) Establish real-time messaging functionality for user created events
        i. Independent research into current technologies, selecting a technology which is best suited to my project and its requirements
    (d) Deploy Web Application online
        i. Requiring research into *Amazon Web Service AWS* and Java Web Application Hosting
2. Develop a graphical user interface (GUI) which is streamlined, functional and intuitive
    (a) The user interface must be responsive and remain correct on various popular devices
    (b) Features of the application must be readily accessible without traversing through extraneous pages
3. Develop graphical representations of historical event data
    (a) Implementing front-end JavaScript libraries
4. Full evaluation of the finished product

3 Requirements & Constraints

3.1 Functional Requirements

Functional requirements specifying the intended behaviour of the application.

3.1.1 End-User Requirements

EUR1 Users can register and will be able to log in with either a registered username or a registered email address

EUR2 Users can log out of their account

EUR3 Users can create events and be provided with a unique event identifier

EUR4 Using an event identifier, a user can join an event and begin posting messages

EUR5 Users can post a message to an event, this message is then instantly visible to other users connected without page refreshes

EUR6 Users can end any event they are the creator of, disconnecting any users instantly and redirecting them appropriately

EUR7 Users can leave any events they have joined

EUR8 A User cannot join any event which has been ended (by the application or by the creator)

3.1.2 System Requirements

SYSR1 Disconnect all connected end User's once an event has reached a maximum inactivity limit, the event is then no longer connectable from a user perspective

SYSR2 Issue warnings to a User for any message sent that are deemed offensive

SYSR3 Offensive messages are cloaked before they are published to an event

SYSR4 Display statistical information which is personal to a logged in User, including but not limited to, *Total Events Joined, Total Events Created, Total Posts Made*

SYSR5 A historical log of all User activity is made available to users, including graphical representations of event data for any user created events

3.2 Non-Functional Requirements

Non-functional requirements specifying the quality and resource attributes of the application.

3.2.1 Security Requirements

SECR1 For security, User's must log in to access any feature of the application

SECR2 Secure hashing function to be applied to store passwords

SECR3 Implementation of back-end form validation to reduce exploits

SECR4 Secure communication through implementation of HTTPS/SSL Certificate

SECR5 Correct handling of JavaScript-disabled browsers to protect the integrity of the application

3.2.2 Database Functionality

DR1 Must have the ability to be scalable and accommodate daily growth in User registrations, event creations and message posts (1000s)

DR2 Operations on large data sets need to be immediate, time taken to insert a single entry needs to be <1S

3.2.3 Performance Requirements and Interface Constraints

PERF1 Event creation process must be made accessible from the home page

PERF2 Joining an event must be made accessible from the home page

PERF3 Site remains responsive across phone, tablet and desktop screen sizes

PERF4 Mirrored front-end form validation to improve efficiency and reduce server traffic, indication of any erroneous input is displayed to the user

PERF5 A User should be able to easily identify their location on the site through good design techniques

PERF6 All pages must fully load in a time <3S

3.2.4 Deployment Requirements

DEPL1 Web Application to be hosted by an online service, this service must allow to application to be accessible online without any limitations on user traffic

Use Case Diagram
The following diagram may better help visualise the system's requirements and overall expectations.

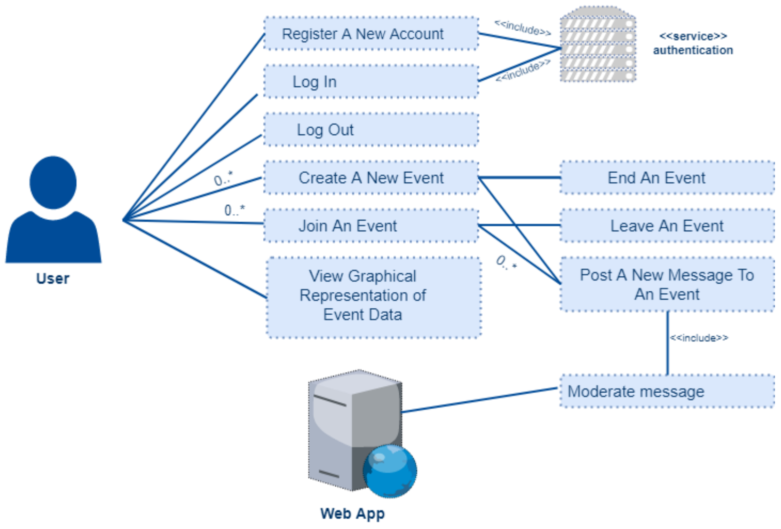


Figure 1: Use Case Diagram

3.3 Optional & Out of Scope Requirements

This is a project which has a large scope for improving existing features and implementing additional ones. Throughout the design and planning process I identified areas of the project where the requirements could be improved and also extended with additional functionality. However, bound by resource constraints and the eagerness for quality over quantity some of these where, in reality, not possible to develop. A comprehensive summary of these may be found in the section ***Extending the system further...*** under ***On reflection***.

4 Architecture & Design of the System

The purpose of this chapter is to formalise the project requirements into concrete design proposals and further discuss any technical aspects introduced. Diagrams have been included to better conceptualise any abstruse information.

4.1 Storage Tier - Persistence Strategy & Database Design

As per the requirements of this project, the application's functionality requires various operations on persisted data (Non-volatile data) and is integral to the initial objectives. For this project, I have chosen to use MongoDB, a NoSQL variant rather than MySQL and other SQL-based alternatives (Relational Databases). MongoDB offers favourable solutions to scalability and performance, two crucial factors I was looking for in a Database System. The trade-off with this is having the less depth in functionality and reduced complexity of transactions.

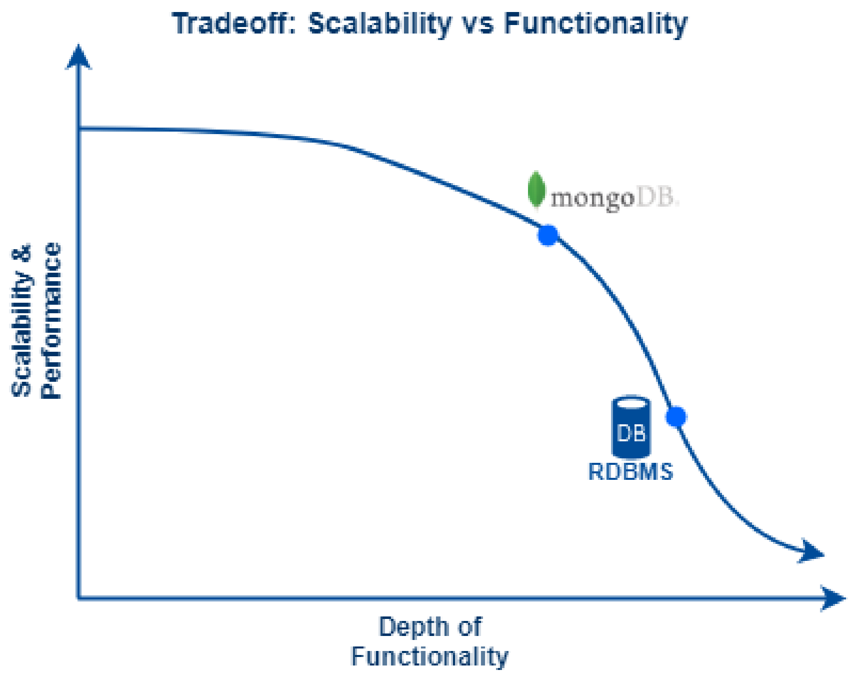


Figure 2: Trade-off between Scalabilty/Performance vs Depth of Functionality. Graph data taken from 10Gen (The MongoDB Company) Website and reconstructed

To further complement the benefits of using MongoDB, I have chosen to use

Cloud Database Storage over a Local Server. The benefits of using a Cloud Database is that it again improves scalability, accessibility and reduced administrative burdens (less features to manage, and therefore no need to adopt one's own security measures to protect data).

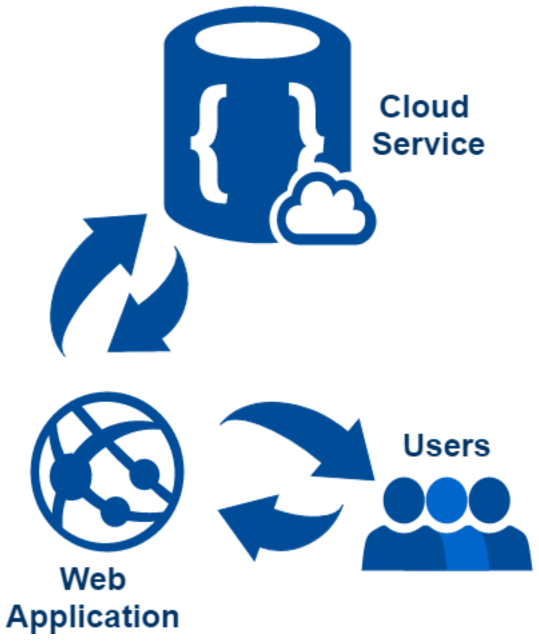


Figure 3: Direction of data exchange between various elements of the system.

4.1.1 Data Modelling & Relationships

MongoDB stores data records in BSON format, which is a binary representation of JSON format and refers to them as *Documents*. Multiple documents are stored in *Collections*. In MongoDB we do not map relationships with multiple tables and keys as you would typically find in relational databases. In this case the data is what is known as denormalised and documents associated to a particular entity are embedded within. This reduces the number of collections and accesses to the database, thus improving performance and efficiency.

For this project, there will be one account type of "User" and therefore 3 essential entities, **User**, **Event** and **Message** which need to be persisted together with their correct relationships. The relationships between these entities must satisfy the following statements summarised from the requirements...

1. A User can join zero to many Event's
2. A User can post zero to many Messages
3. Events can accept zero to many User's

4. Events can contain zero to many Message's

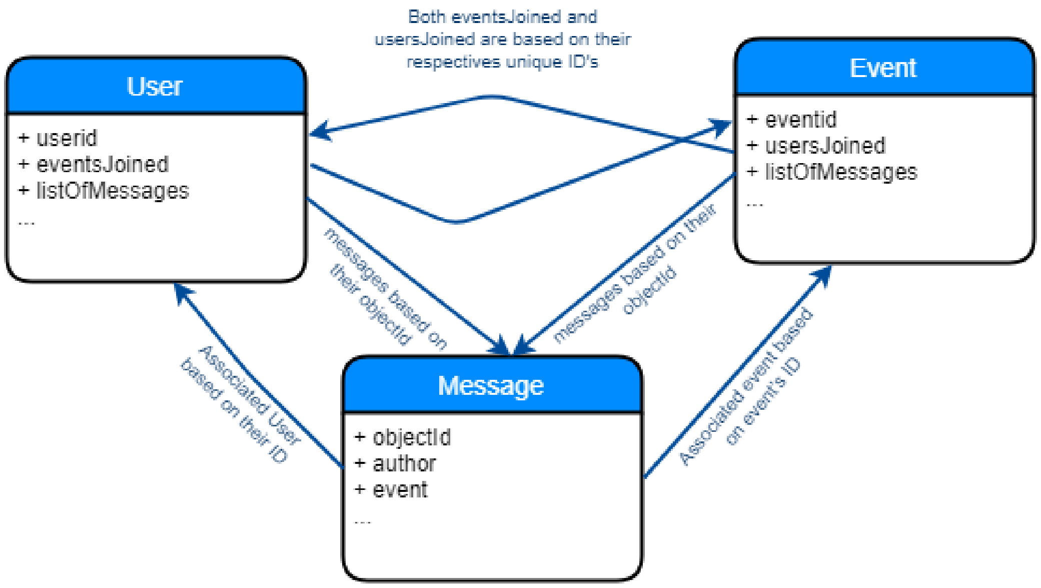

Both eventsJoined and usersJoined are based on their respectives unique ID's
User
+ userid
+ eventsJoined
+ listOfMessages
...
Event
+ eventid
+ usersJoined
+ listOfMessages
...
messages based on their objectId
messages based on their objectId
Message
+ objectId
+ author
+ event
...
Associated User based on their ID
Associated event based on event's ID

Figure 4: Relationships between entities of the system.

4.2 Application & Business Tier

4.2.1 MVC Architecture & Spring Framework

The architectural style chosen for my system is the Model-View-Controller (MVC) pattern. This style is particularly popular for Web Applications as it allows the developer/s to separate the areas of concern, creating layers of abstraction and thus improving the maintainability and stability of the system. The Model layer is the central component of the architecture and is responsible for managing the data, the business logic and rules of the application. The Controller provides the link between the user interface (views) and the business logic (models) of the application. Finally, the Views are what the user sees, in this case it will be the user interface. This pattern will allow me to work on separate areas of the system without affecting other code-bases. *Spring Framework* is a natural pre-packaged solution to a Java-based Web Application using the MVC architecture. Importantly, It also provides a security module for which I can handle the authorisation and authentication of users throughout the system.

4.2.2 Spring Boot

Spring is a large framework and provides a wide variety of features to address modern and business scale applications. The result of this is that there are large quantities of complex configurations and boiler plate code. For smaller projects, the full range of modules which the framework provides may not be relevant. To address this, we have *SpringBoot*. SpringBoot provides an easier and convenient way of setting up standalone Web Applications quickly, reducing the configuration time and code needed to do so. The project will be making use of SpringBoot to improve the start up efficiency, there will also involve less maintenance of configuration files due to it's Auto-Configuration features. It additionally provides smooth integration with a Tomcat server thus providing a simpler deployment of the Web Application.

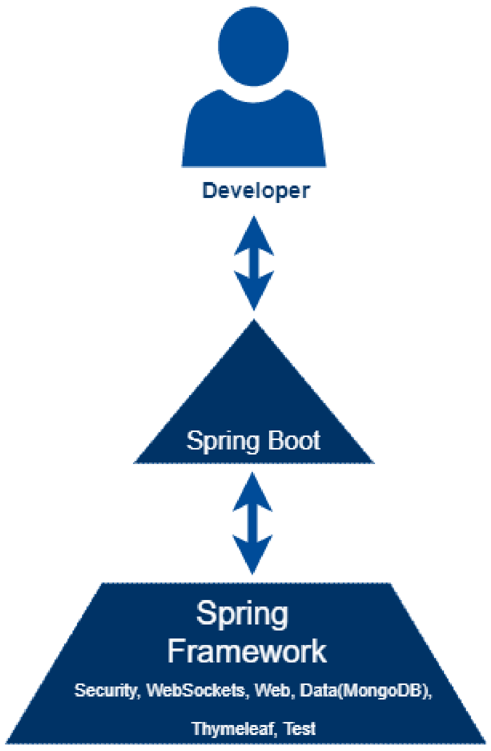


Figure 5: Spring Framework.

4.2.3 Apache Maven

As the project grows in complexity, the number of manual steps and commands needed to build the project increases. To anticipate this, it is vital that I use some form of automation to assist with the build. I will be using *Apache Maven* to achieve this. Maven will take care of managing the dependencies of the project, running automated tests and producing any resulting documentation.

4.2.4 WebSockets for Real-Time Communication

To achieve the requirement of real-time messaging within events, I will use an existing technology - WebSockets. For this project, we require continuous data exchange between the Server and the Client without ever having to reload or reconnect. WebSockets provide a solution to this requirement through establishing a initial single TCP connection, which remains active throughout the life-cycle of the WebSocket connection. This means that the Client, in this case an Audience Member, can open a connection to the Server and this connection will remain open until either the Audience Member chooses to end the connection or the Server does.

The project also requires non-conformity in Client/Server communication;

we do not wish for communication to be restricted to pre-defined message protocols of request/response. WebSockets address this by allowing full-duplex bi-directional communication over the single TCP connection. The result is that Audience Members can communicate with the Server *at any time*, in this case, post messages to an Event, without requiring any response first. This also allows the Server to communicate with the Audience Member *at any time*, perhaps send any suitable warnings or to close a connection.

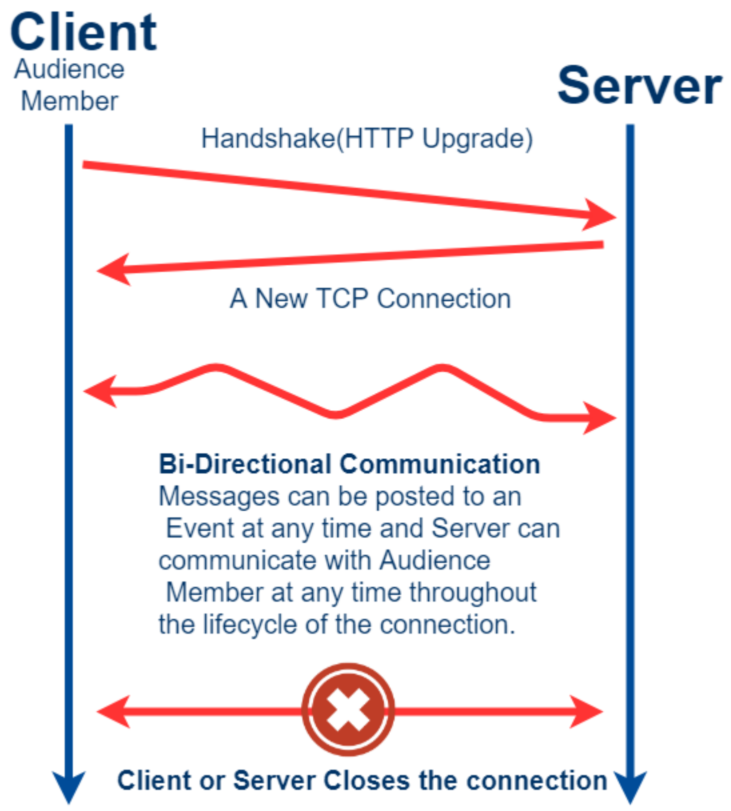

Client
Audience Member
Server
Handshake(HTTP Upgrade)
A New TCP Connection
Bi-Directional Communication
Messages can be posted to an Event at any time and Server can communicate with Audience Member at any time throughout the lifecycle of the connection.
Client or Server Closes the connection

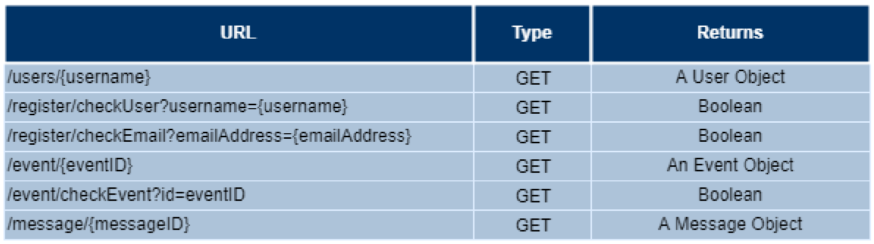
Figure 6: Visual Representation of a WebSocket life-cycle.

Spring Framework has support for WebSocket-style messaging, and integrates nicely with another protocol - STOMP (Simple Text Orientated Messaging Protocol). STOMP interacts with WebSocket at a higher level abstraction and offers text-based communication, ideal for this project as communication will be within written languages. For client-side connectivity to the WebSocket I will use a JavaScript library known as sockJS. sockJS will provide a WebSocket-like object that gives a full-duplex, bidirectional communication through a JavaScript API.

There were other alternatives to achieve real-time communication such as an AJAX with long-polling technique, however due to performance constraints, these would not satisfy the project requirements and were dismissed.

4.2.5 RESTful Web Services

Scalability through abstraction is one of the key design approaches for this project. The idea of decoupling the system components wherever possible, for example, Client and Server, would allow the project to grow in size and complexity without applying maintenance costs throughout the application. I will be using RESTful Web Services for aspects of this project where data is required to be exposed to the Client. RESTful Web Services allow room to develop any data-access functionality on the Server-side freely without Client-Side Development and also allows the Client to access the exposed data through URLs in an Ad-Hoc manner. These principles will be used in fetching Usernames, Event Information and Message Information from the Database. Examples of possible Data Access URLs have been given.



URL	Type	Returns
/users/{username}	GET	A User Object
/register/checkUser?username={username}	GET	Boolean
/register/checkEmail?emailAddress={emailAddress}	GET	Boolean
/event/{eventID}	GET	An Event Object
/event/checkEvent?id=eventID	GET	Boolean
/message/{messageID}	GET	A Message Object

Figure 7: Examples of RESTful URLs.

4.3 Presentation Tier

4.3.1 User Interface

For this product, the User Interface will be implemented in the form of an interactive website. The Spring MVC Framework refers to specific visual output of the Web Application as a "View" and is, in this case, responsible for configuring the routing of "View" names to internal pages of the web application. The framework is heavily customisable and there are many "View" technologies available. I have chosen to use Thymeleaf as the View technology for this project as it sports many attractive features which will aid the development of a good and robust interface. Thymeleaf uses plain HTML as its templating language and therefore offers well documented and common mechanisms for constructing the page layouts.
As identified through the Project Requirements *(PERF3)*, it is essential that the pages incorporate responsive attributes and remain structurally appealing across phone, tablet AND desktop screen sizes. A Client-side HTML/JavaScript framework known as Bootstrap has been chosen to address this issue. The

Bootstrap framework manages the responsiveness of a layout via a fluid grid system which can be configured to dynamically adjust to different screen resolutions. It also offers huge amounts of resources through an active community and pre-styled components to save time during development. A simplified diagram depicting a proposed layout and its responsive nature has been provided. However, during the planning and prototyping phase, I did not decide to create a full array of wire frame designs for the layout as I did not believe that they would be of any benefit. Wire frames can take large amounts of time to create and, from experience in other large projects, the final release often bares little resemblance to them.

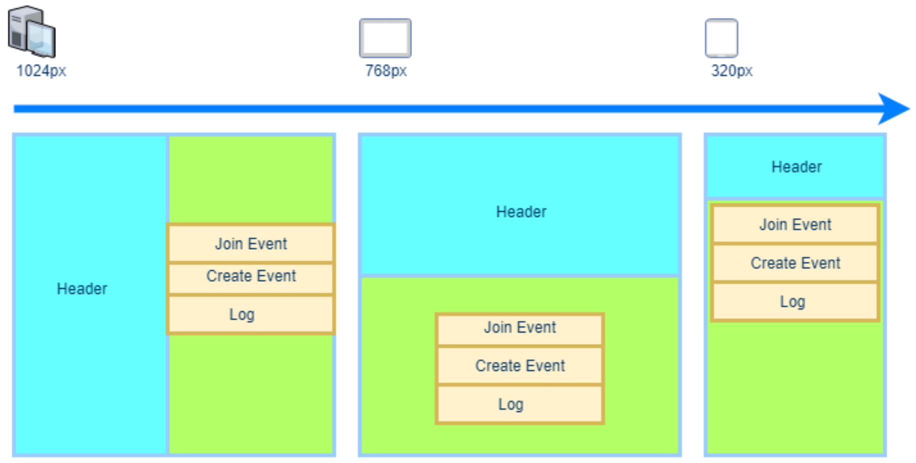


Figure 8: Responsive Web Design proposals.

NOTE This satisfies Requirements *PERF1* and *PERF2* as they provide quick access to the application's features. Joining an Event and Creating an Event can be performed from the Home Page, with the inclusion of a separate link for Event Logs.

4.3.2 Displaying Event Data

This project is designed to not only be useful for members of the audience but to be an effective tool for the event creators at the same time. Implementation of a word occurrence count algorithm can be used to produce meaningful data to display. This will indicate to the Event Creator which words commonly appear and identify any trends or areas of concern among a live audience. The business logic will split the message up into a list of words and then remove any words which are deemed to have a little importance. These idle words are referred to as "Stop Words" and will be taken from a predefined file. Iterating and counting word occurrences through the length of the remaining list of words will result in a data set in the form of (*Key*, *Value)*, with the *Key* being the word and *Value* being the occurrence count. This format is inter-operable and can be used as input for Data Visualisation libraries.

In this system, we will be displaying the data in the form of a Word Cloud. A Word Cloud is an image composed of words in which the size of the word indicates the word frequency or importance. It is a simple idea but a complex algorithm to build (probably similar in size to this project itself) and so by using an existing library dedicated to this function we can quickly and efficiently implement this feature to a high standard. This may be better demonstrated with the following pseudo-code.

```
// On Application Boot
StopWordList ← from file

// At every time you load Event Statistics
// Map of words representing (K, V)
WordCount ← new Map
listOfWords ← new list of String

for each message in eventMessages
  for each word i in message
    if StopWordList does not contain i
        update WordCount(i, value+=1)
return WordCount
```

Integration with a Word Cloud library will be done on the Client-Side with a JavaScript library. Using Jason Davies *d3-Cloud Library* the product will be something similar to this.

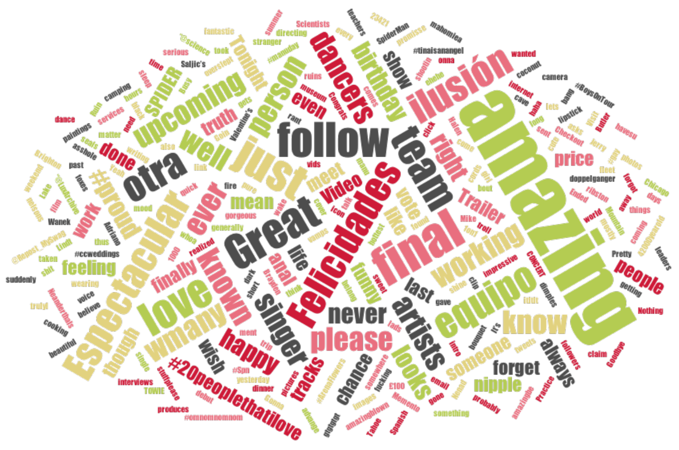


Figure 9: Example of A Word Cloud diagram.

5 Implementation & Testing

In this chapter, I will thoroughly visit each feature implemented, covering detailed implementation steps, aesthetic aspects and then finally discuss how I built a comprehensive test suite forming the final package.

5.1 Implementing the System

5.1.1 Spring Boot Overview

As mentioned before, Spring is a large Framework and as such, can be quite complex to implement. To aid with this, we are using Apache Maven to manage the complex tree of dependencies needed for building the project. Maven uses an XML-based file, commonly titled *pom.xml* to manage this for us. In the pom file for this project we add the following code to automatically download the dependencies required from the Spring Framework.

```
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>1.4.7.RELEASE</version>
   <relativePath />
   <!-- lookup parent from repository -->
</parent>
...
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-websocket</artifactId>
</dependency>
```

There are other configuration instances required in this file, however I will not cover them here as it would result in diverging from the purpose of this report.

5.1.2 Domain Models

Domain Models are Java Classes (in this case) that model a problem domain. These normally do not implement any technical functionality but contain fields to describe an instance of something. As specified previously in the system Design chapter, this project should include 3 essential domain models, `User.java` to model a User, `Event.java` to model an Event and `Message.java` to model a Message sent from the Client to the Server. There is the inclusion of two extra java classes, `OutputMessage.java` which models a message sent from the Server back to the Client and `WordCount.java`, which is the model of the frequency a Word appears in a Message. These are placed inside the package `com.talkable.domain`.

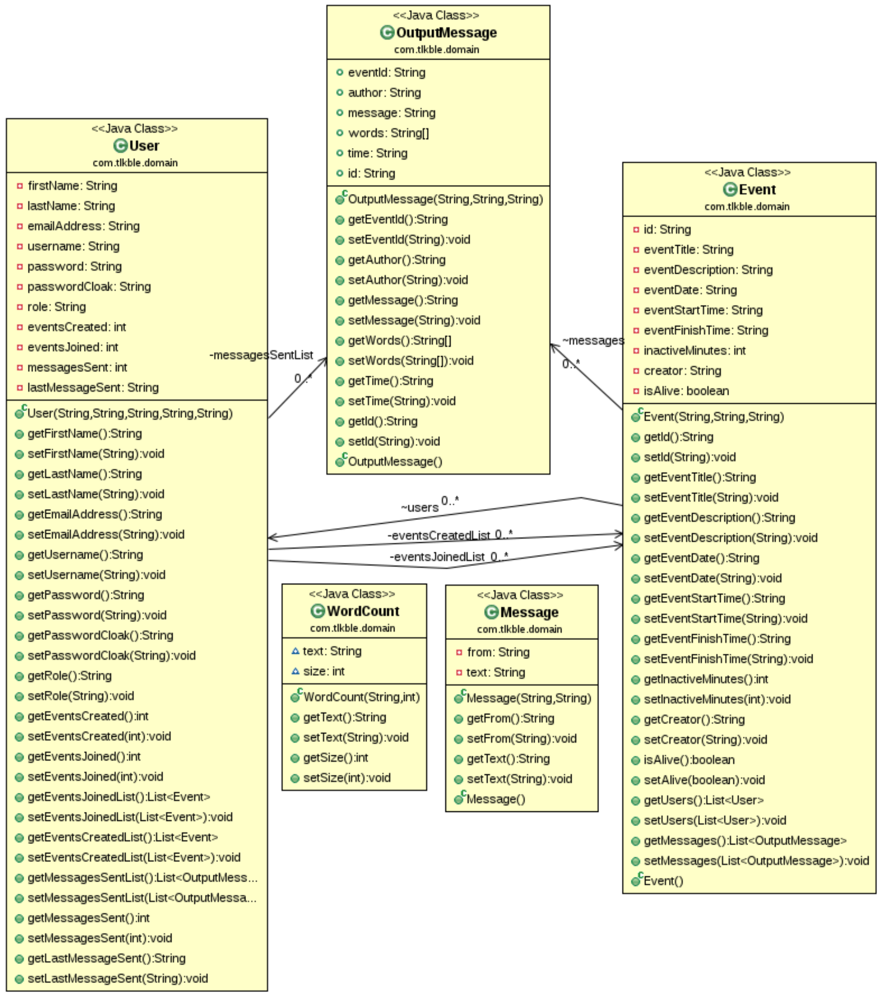


Figure 10: Class Diagram of Domain Models

We can see that this Class Diagram follows the Entity Relationship Design to some extent but with additional fields and methods where they were required. The Generated Diagrams may look fairly extensive with many *Getter* and *Setter* methods but the Java Files are less-so with the help of the Lombok library. Lombok reduces the amount of boilerplate code needed by incorporating annotations to replace it. For example, when defining the member field `username` inside the class `User.java`, we insert the annotations `@Getter` and `@Setter` alongside it to generate the respective methods at compile-time.

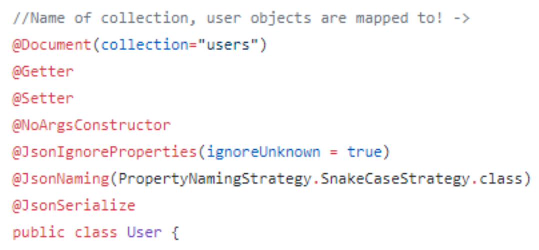
```
@Getter @Setter private String username;
```

produces the following at compile-time;

```
private String username;
public String getUsername() { return username; }
public void setUsername(String username) { this.username = username; }
```

5.1.3 Data Access Objects & Connecting to the Cloud

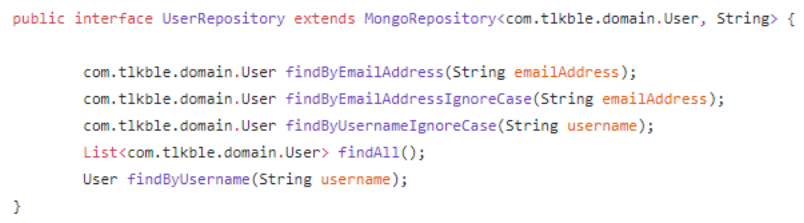
The Domain Objects `User`, `Event`, `OutputMessage` are given an annotation of `@Document(collection="respectivename")` at class level, this tells Mongo that these are entities which need to be persisted and which collection the object belongs to.



```
//Name of collection, user objects are mapped to! ->
@Document(collection="users")
@Getter
@Setter
@NoArgsConstructor
@JsonIgnoreProperties(ignoreUnknown = true)
@JsonNaming(PropertyNamingStrategy.SnakeCaseStrategy.class)
@JsonSerialize
public class User {
```

Figure 11: @Document Annotation

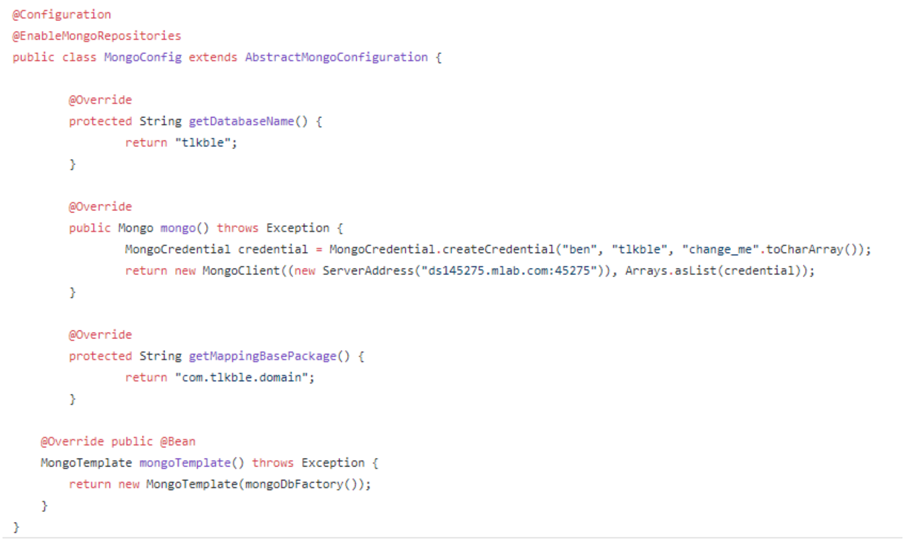
In order to establish the functionality of saving, updating and performing various queries on collections, I have set up the DAO in the form of Repositories for each Domain Model. These interfaces extend the `MongoRepository Class` and are located in the `com.talkable.repository` package.



```
public interface UserRepository extends MongoRepository<com.tlkble.domain.User, String> {

        com.tlkble.domain.User findByEmailAddress(String emailAddress);
        com.tlkble.domain.User findByEmailAddressIgnoreCase(String emailAddress);
        com.tlkble.domain.User findByUsernameIgnoreCase(String username);
        List<com.tlkble.domain.User> findAll();
        User findByUsername(String username);
}
```

Figure 12: Data Access Object

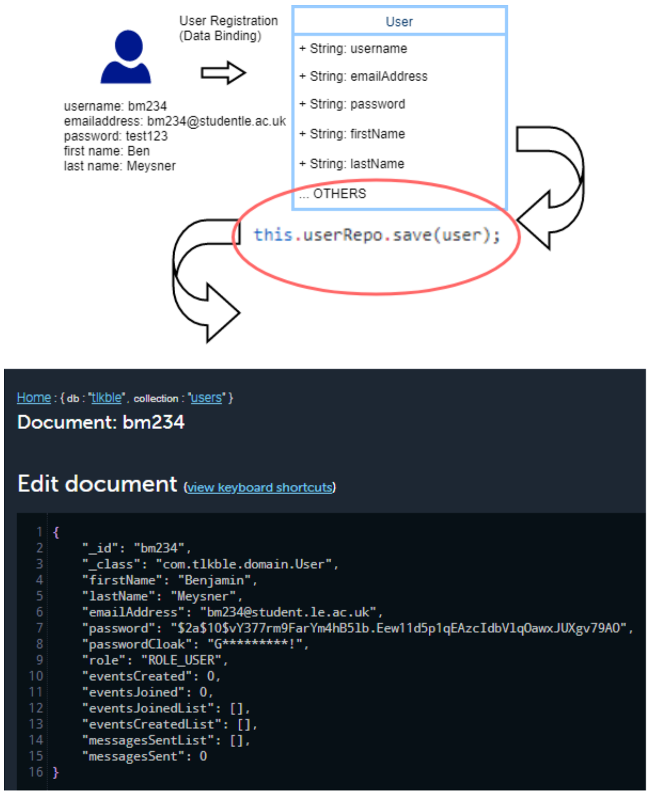
For the final part of Database connectivity, I use Spring Data's MongoTemplate rather than MongoRepository as it is quick and easy to configure setup and caters for all the project's persistence operations. I made a configuration file `MongoConfig.java` which extends `AbstractMongoConfiguration`. Inside this file is where I declare the base packages for `@Document` annotation Scanning (`com.talkable.domain`), database name and importantly provide database credentials within a `new MongoCredentials instance`. This configuration class is given the annotation `@EnableMongoRepositories` which activates the MongoDB repository infrastructure discussed previously.



```java
@Configuration
@EnableMongoRepositories
public class MongoConfig extends AbstractMongoConfiguration {

        @Override
        protected String getDatabaseName() {
                return "tlkble";
        }

        @Override
        public Mongo mongo() throws Exception {
                MongoCredential credential = MongoCredential.createCredential("ben", "tlkble", "change_me".toCharArray());
                return new MongoClient((new ServerAddress("ds145275.mlab.com:45275")), Arrays.asList(credential));
        }

        @Override
        protected String getMappingBasePackage() {
                return "com.tlkble.domain";
        }

    @Override public @Bean
    MongoTemplate mongoTemplate() throws Exception {
        return new MongoTemplate(mongoDbFactory());
    }
}
```

Figure 13: MongoDB configuration file

To provide an example of how this works in practice, a diagram showing the rough process of persisting a new User to the mLab.com Cloud Database using the repository interface is given below.

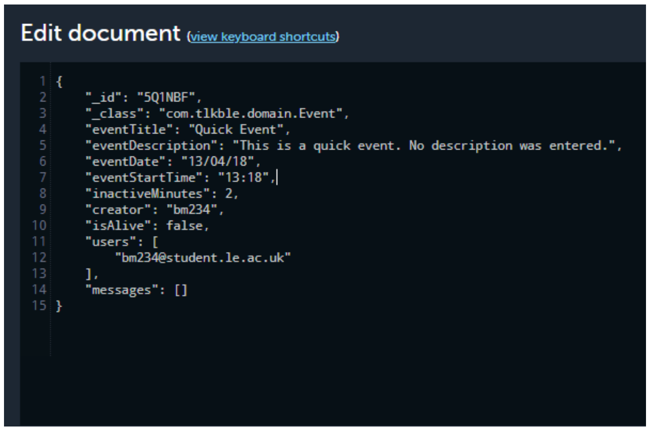


```
this.userRepo.save(user);
```

```
{
    "_id": "bm234",
    "_class": "com.tlkble.domain.User",
    "firstName": "Benjamin",
    "lastName": "Meysner",
    "emailAddress": "bm234@student.le.ac.uk",
    "password": "$2a$10$vY377rm9FarYm4hB51b.Eew11d5p1qEAzcIdbV1qOawxJUXgv79AO",
    "passwordCloak": "G*********!",
    "role": "ROLE_USER",
    "eventsCreated": 0,
    "eventsJoined": 0,
    "eventsJoinedList": [],
    "eventsCreatedList": [],
    "messagesSentList": [],
    "messagesSent": 0
}
```

Figure 14: Persisting a new User to mLab

In the same respect, A new `Event` Instance can be persisted to mLab and will result in the following figure below, the same applies to `OutputMessage`. I persist `OutputMessage`'s instead of Message's as it is the product of Message Moderation and applied attributes instead of the raw `Message` sent from the Client.



Edit document (view keyboard shortcuts)

```
{
    "_id": "5Q1NBF",
    "_class": "com.tlkble.domain.Event",
    "eventTitle": "Quick Event",
    "eventDescription": "This is a quick event. No description was entered.",
    "eventDate": "13/04/18",
    "eventStartTime": "13:18",
    "inactiveMinutes": 2,
    "creator": "bm234",
    "isAlive": false,
    "users": [
        "bm234@student.le.ac.uk"
    ],
    "messages": []
}
```

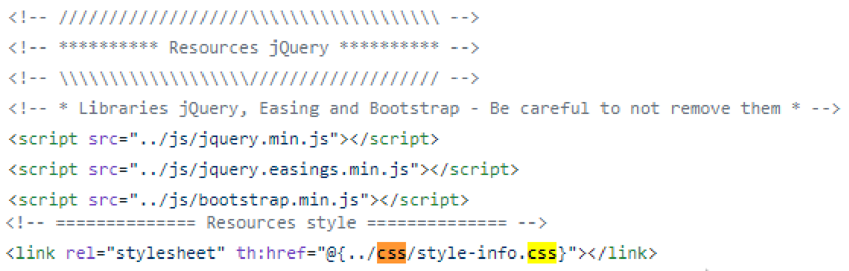
Figure 15: An Event saved to mLab

5.1.4 Building the User Interface

With the aid of the Thymeleaf Template Engine and Twitter Bootstrap Framework, the Web Application has been developed to incorporate views which have both dynamic content and responsive elements. This has ensured that each view is functional and aestically pleasing, but also adapts automatically to devices of different sizes as per the requirements.

5.1.4.1 Bootstrap

Regarding the implementation of Bootstrap, it is installed by downloading the correct CSS and JavaScript files into the project resources folder. All HTML pages of the Web Application requiring Bootstrap must then apply `Link` tags to apply the CSS properties and a `Script` tag to point to the external JavaScript files.



```
<!-- ///////////////////\\\\\\\\\\\\\\\\\\ -->
<!-- ********** Resources jQuery ********** -->
<!-- \\\\\\\\\\\\\\\\\\\////////////////// -->
<!-- * Libraries jQuery, Easing and Bootstrap - Be careful to not remove them * -->
<script src="../js/jquery.min.js"></script>
<script src="../js/jquery.easings.min.js"></script>
<script src="../js/bootstrap.min.js"></script>
<!-- ============== Resources style ============== -->
<link rel="stylesheet" th:href="@{../css/style-info.css}"></link>
```

Figure 16: Bootstrap resources

Bootstrap ensures compatibility for different devices via a fluid grid system. Bootstrap allows up to 12 columns to be spread across the view-port and are rearranged depending on screen size. However this is achieved, is totally up to the designer. Simple mathematics suggests that we can have columns in the following sizes (12 X Span 1), (6 X Span 2), (4 X Span 3), (3 X Span 4), (2 X Span 6) and (1 X Span 12). In the case of this project we use 2 columns for the layout of larger screened-devices.

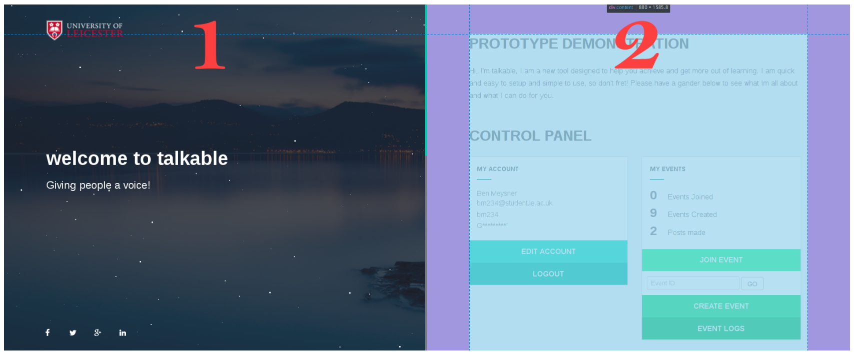
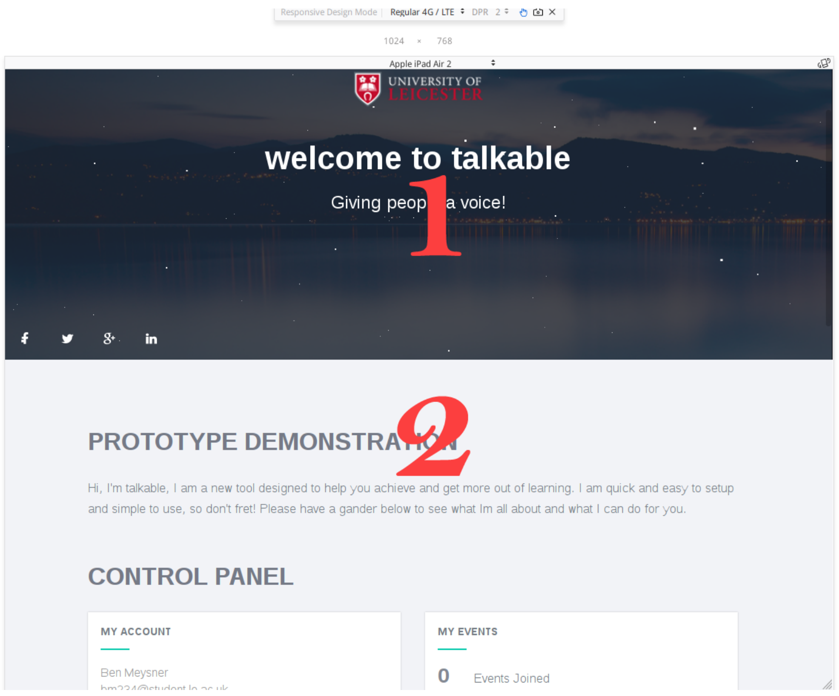


Figure 17: Desktop column sizes

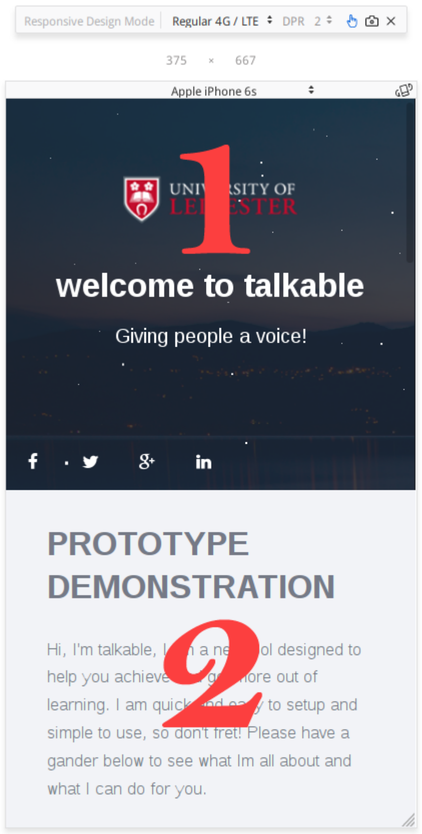
This is then scaled accordingly for a device of tablet size. The 2 columns are collapsed and placed one under because of the reduced view-port width.



Hi, I'm talkable, I am a new tool designed to help you achieve and get more out of learning. I am quick and easy to setup and simple to use, so don't fret! Please have a gander below to see what Im all about and what I can do for you.

Figure 18: Tablet column sizes

Finally, to account for Users with smaller devices such as phones, we again, scale down accordingly. However, we keep the same main structure as the tablet layout, but containing elements are reduced in size to make it fit on one screen without any horizontal scrolling.



welcome to talkable

Giving people a voice!

PROTOTYPE DEMONSTRATION

Hi, I'm talkable, I [illegible] a ne[illegible]ol designed to help you achieve[illegible]g[illegible]ore out of learning. I am quick [illegible]d ea[illegible]y to setup and simple to use, so don't fret! Please have a gander below to see what Im all about and what I can do for you.

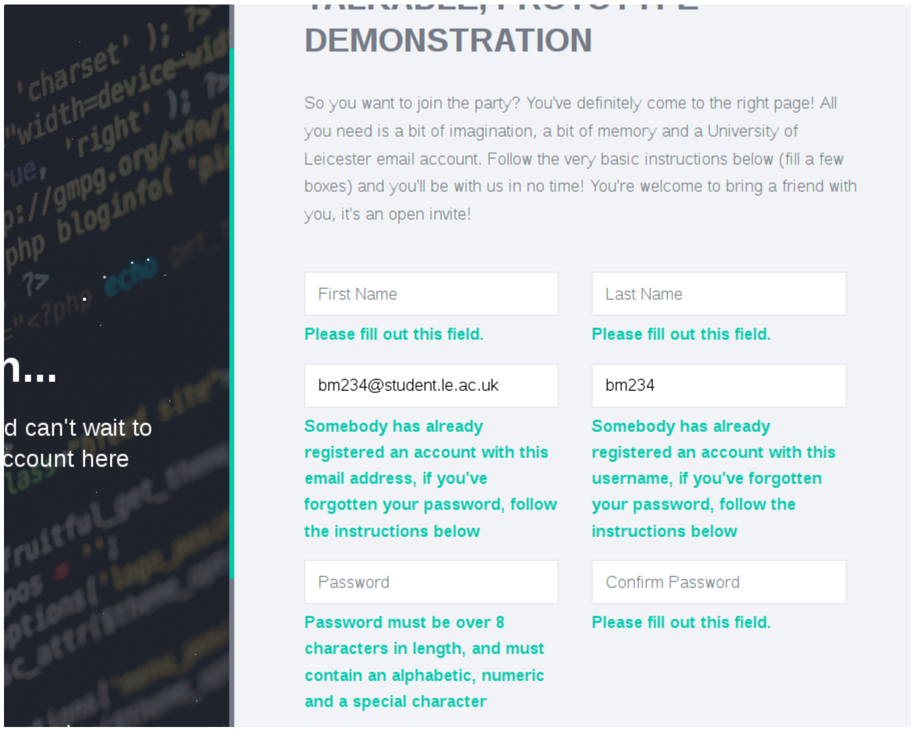
Figure 19: Phone column sizes

The scale properties of the layout remain consistent across all views of the application. Bootstrap makes this customisable by use of `@Media` tags above a selection of CSS properties within the Bootstrap CSS File. These tags normally take an extra parameter such as `min-width:` or `max-width:` which apply these properties while the parameter is true.
Not only does Bootstrap ensure compatibility for various devices, it also ensures use-ability. As the layout is scaled down gradually for smaller devices, the HTML elements maintain their size. This is because the usage of smaller devices can be fiddly, and navigating through Web Sites with small links frustrating.

This can be seen in above, the `sosicons` - or Social Media Icons maintain their size even when the view-port is reduced. This is consistent throughout the application with all buttons and links.

5.1.5 User Registration

The first feature for a new User of the application is the ability to register a new account. To register, the user is required to present a `first name`, `last name`, `email address`, `username` and a `password`. Client-side and Server-side validation is performed on the user inputs to verify the integrity of the data. All Client-side form validation is handled by using `1000hz Bootstrap Validator` - A JavaScript library for validating form fields dynamically. Basic form validation includes making sure no fields are empty and all information required is present. More advanced validation features include regular expressions for validating the email address and the required password format, appropriate messages are presented to the user.



DEMONSTRATION

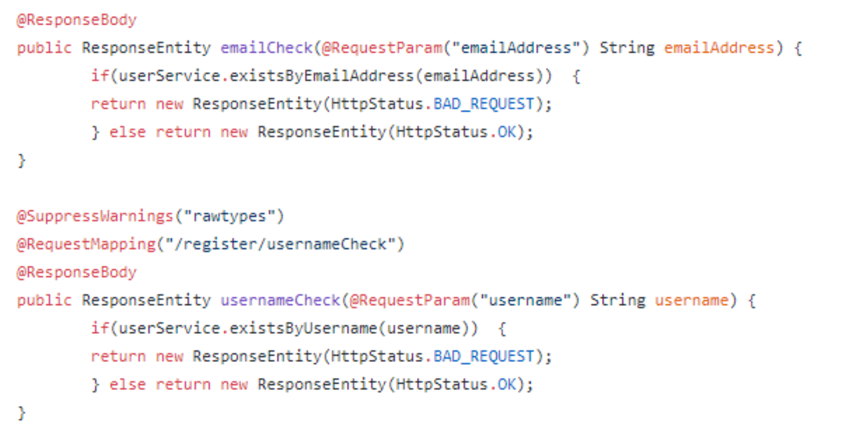
So you want to join the party? You've definitely come to the right page! All you need is a bit of imagination, a bit of memory and a University of Leicester email account. Follow the very basic instructions below (fill a few boxes) and you'll be with us in no time! You're welcome to bring a friend with you, it's an open invite!

First Name	Last Name
Please fill out this field.	Please fill out this field.
bm234@student.le.ac.uk	bm234
Somebody has already registered an account with this email address, if you've forgotten your password, follow the instructions below	Somebody has already registered an account with this username, if you've forgotten your password, follow the instructions below
Password	Confirm Password
Password must be over 8 characters in length, and must contain an alphabetic, numeric and a special character	Please fill out this field.

d can't wait to
ccount here

Figure 20: Registration Form Validation

To validate if a username or email address has been already registered, `1000hz Validator` library provides a `data-remote` attribute on the form field, which performs an AJAX request to the Server for the given URL. We handle this on the Server-side Controller, in a method to check a username or email address exists in the Database and returning a `ResponseEntity` of either `HTTPStatus.OK` or `HTTPStatus.BAD_REQUEST` depending on the result.



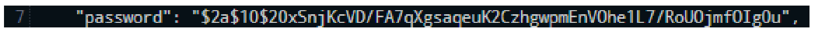
```
@ResponseBody
public ResponseEntity emailCheck(@RequestParam("emailAddress") String emailAddress) {
        if(userService.existsByEmailAddress(emailAddress))  {
        return new ResponseEntity(HttpStatus.BAD_REQUEST);
        } else return new ResponseEntity(HttpStatus.OK);
}

@SuppressWarnings("rawtypes")
@RequestMapping("/register/usernameCheck")
@ResponseBody
public ResponseEntity usernameCheck(@RequestParam("username") String username) {
        if(userService.existsByUsername(username))  {
        return new ResponseEntity(HttpStatus.BAD_REQUEST);
        } else return new ResponseEntity(HttpStatus.OK);
}
```

Figure 21: Registration Form Validation 2 - Controller Code

Assuming all Front-End validation checks are successful, the information is then processed on the Server-Side for validation checks which mirror those of the front-end and password security.

5.1.5.1 Password Storage

The User's password is considered sensitive information and thus cannot be stored as plaintext inside the database for obvious reasons, notably data breaches and man-in-the-middle attacks. Using `BCrypt`, a Java library to hash passwords, I apply its hashing function to the User supplied password before the the User object is persisted to the database. The result is an illegible sequence of characters which have little meaning to a would-be attacker.



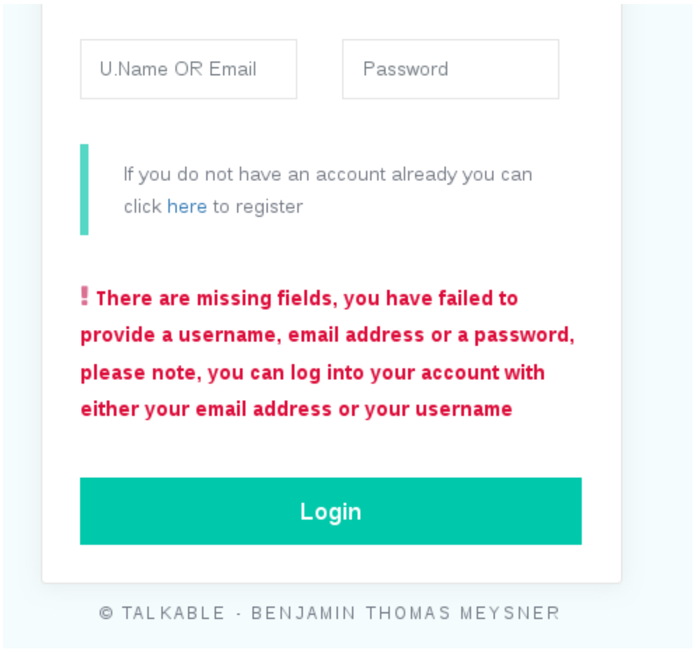
```
    "password": "$2a$10$20xSnjKcVD/FA7qXgsaqeuK2CzhgwpmEnVOhe1L7/RoUOjmfOIg0u",
```

Figure 22: Password encryption with BCrypt

After successful registration, A User is logged in automatically and redirected to the home page `/user/home`.

5.1.6 User Login

Once a User has an account registered, they are then able to login to the system. The User is required to enter either their username **OR** email address **AND** their password. Any erroneous input such as missing fields, username or email not existing and bad credentials combination is this time handled only on the Server-Side by including a `FlashAttribute` with a corresponding error message.


U.Name OR Email
Password
If you do not have an account already you can click here to register

! There are missing fields, you have failed to provide a username, email address or a password, please note, you can log into your account with either your email address or your username

Login
© TALKABLE - BENJAMIN THOMAS MEYSNER

Figure 23: User Login Box with Error

To check the combination of username/email address and password, I use a method `checkPassword` which takes both the password from the login request, which is in plaintext, and the database stored password taken from the user matching the given username/email address. `BCrypt` has a built-in method to check the equality of both passwords by applying the same hash function to the plaintext and checking it is equal to one taken from the database.

Given the credentials are correct, Spring Security will authenticate the user and add the user object as a session variable. The authenticated User is then redirected to the Application home page `/user/home`.

5.1.7 Creating & Connecting to Events

One of the key features of this project is the ability for Users to create Events. As this process happens during a live presentation, it is important that a User can create an event quickly without hassle. To implement this, each Event is given a unique identifier which is secure enough to maintain an extremely low risk of collision, but terse enough that it can be used as a literal key for Event entry. On the Client-side, a 6 character identier is created by generating two random 3 character strings which are concatenated. Using `Math.random()` of the JavaScript library, multiplied by a constant of `46656`, it converts a randomly generated floating point <1 to a floating point >1. A binary `OR` operation is applied to this number to remove the fractional component, leaving an integer which is >1297 and <46656.

```
 ((Math.random() * 46656) | 0);
```

This integer is represented in base 36, resulting in a 3 character alphanumeric. The concatenation of both string make up a 6 character `eventId`.

```
firstPart.toString(36)); secondPart.toString(36));
eventId = (firstPart + secondPart).toUpperCase();
```

The User is prompted to enter a title for the Event and a Description, both of which are optional. If the User chooses to not enter any of these, the default of `‘‘Quick Event’’` for the title and `‘‘This is a quick event. No Description was entered.’’` for the description is set.

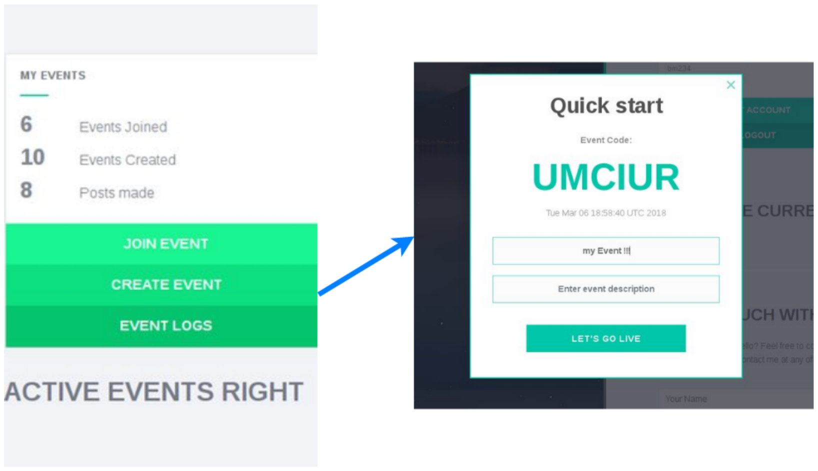
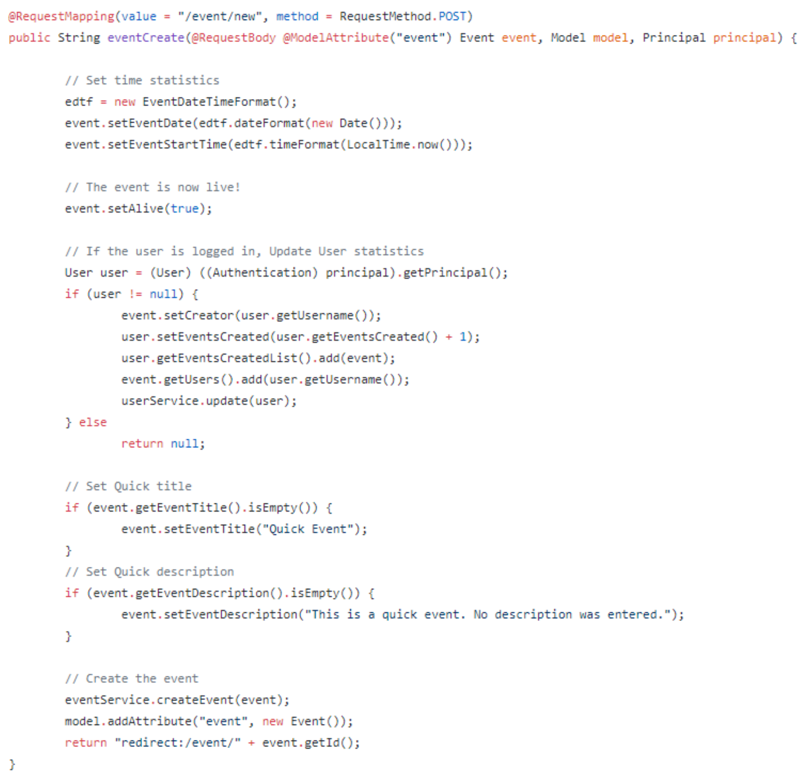


Figure 24: Creating an Event - User Prompt

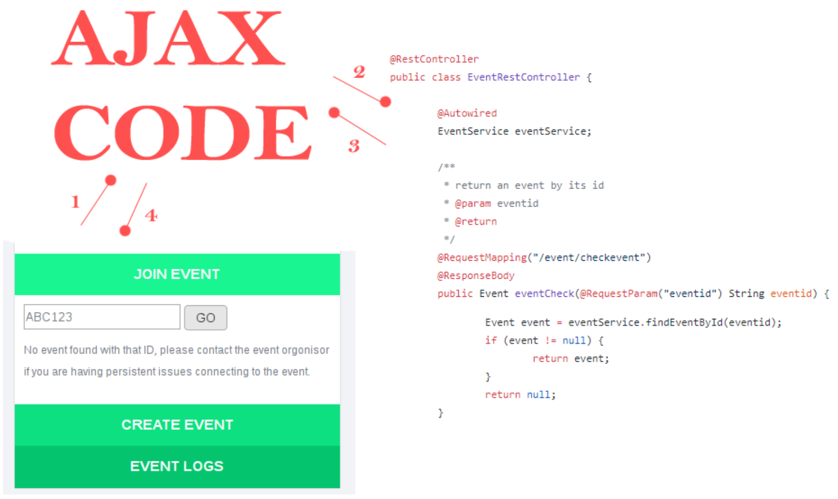
On the Server-side, in `EventController.java`, this POST action is mapped to `/event/new`. Here we set additional attributes such as the event start time, event creation date and the current status of the event (Alive or Not Alive). The Event is considered Alive if it has been started and has not been ended, either by a User or by the System. The new Event is then persisted to the Database with all the associated information. Upon success, the User is redirected to the new Event by using `return "redirect:/event/" + event.getId();`.



```java
@RequestMapping(value = "/event/new", method = RequestMethod.POST)
public String eventCreate(@RequestBody @ModelAttribute("event") Event event, Model model, Principal principal) {

        // Set time statistics
        edtf = new EventDateTimeFormat();
        event.setEventDate(edtf.dateFormat(new Date()));
        event.setEventStartTime(edtf.timeFormat(LocalTime.now()));

        // The event is now live!
        event.setAlive(true);

        // If the user is logged in, Update User statistics
        User user = (User) ((Authentication) principal).getPrincipal();
        if (user != null) {
                event.setCreator(user.getUsername());
                user.setEventsCreated(user.getEventsCreated() + 1);
                user.getEventsCreatedList().add(event);
                event.getUsers().add(user.getUsername());
                userService.update(user);
        } else
                return null;

        // Set Quick title
        if (event.getEventTitle().isEmpty()) {
                event.setEventTitle("Quick Event");
        }
        // Set Quick description
        if (event.getEventDescription().isEmpty()) {
                event.setEventDescription("This is a quick event. No description was entered.");
        }

        // Create the event
        eventService.createEvent(event);
        model.addAttribute("event", new Event());
        return "redirect:/event/" + event.getId();
}
```

Figure 25: Creating an Event - Controller Code

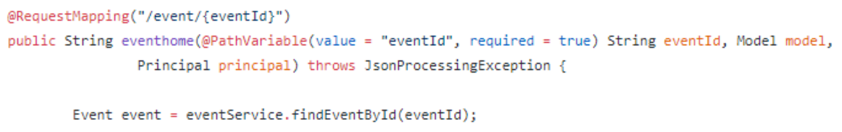
Once an Event has been created it is then possible for Users to join it using the `eventId` created. When a User clicks the `JOIN EVENT` button, a drop down box (toggle-able) appears prompting the User to enter an Event ID. This is an AJAX form which performs a `GET` request on the URL `/event/checkevent` with a parameter of `eventID`. On the Controller side, the method handling this request extracts the eventID and searches the collection for a matching Event. The method returns an Event if one exists and null otherwise. The AJAX function handles the data returned appropriately by either redirecting the user to the requested event page or by displaying a message of `‘‘No Event with that ID exists’’`. Accessing the correct Event page is achieved by a URI in the form `/event/eventId` where the `eventId` is a path variable. The `eventId` is extracted from the URI in the controller method and used to find a matching event from the collection. If any event has been found, it’s attributes are added to a model and used for displaying purposes in the event view.



No event found with that ID, please contact the event orgonisor if you are having persistent issues connecting to the event.

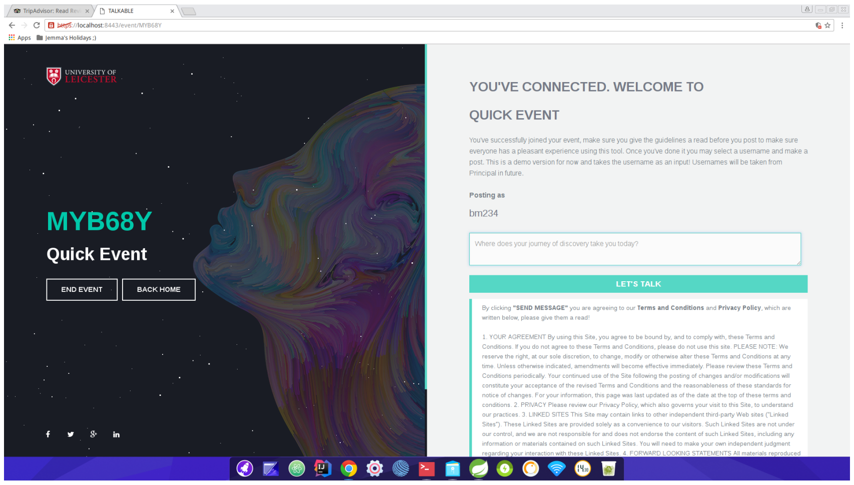
```
@RestController
public class EventRestController {

    @Autowired
    EventService eventService;

    /**
     * return an event by its id
     * @param eventid
     * @return
     */
    @RequestMapping("/event/checkevent")
    @ResponseBody
    public Event eventCheck(@RequestParam("eventid") String eventid) {

        Event event = eventService.findEventById(eventid);
        if (event != null) {
            return event;
        }
        return null;
    }
```

Figure 26: Joining an Event - The Process



```
@RequestMapping("/event/{eventId}")
public String eventhome(@PathVariable(value = "eventId", required = true) String eventId, Model model,
            Principal principal) throws JsonProcessingException {

        Event event = eventService.findEventById(eventId);
```

Figure 27: URI template for an Event



YOU'VE CONNECTED. WELCOME TO

QUICK EVENT

You've successfully joined your event, make sure you give the guidelines a read before you post to make sure everyone has a pleasant experience using this tool. Once you've done it you may select a username and make a post. This is a demo version for now and takes the username as an input! Usernames will be taken from Principal in future.

Posting as

bm234

Where does your journey of discovery take you today?

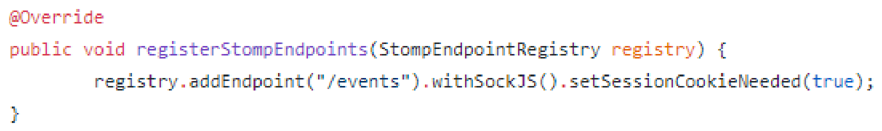
By clicking **"SEND MESSAGE"** you are agreeing to our **Terms and Conditions** and **Privacy Policy**, which are written below, please give them a read!

1. YOUR AGREEMENT By using this Site, you agree to be bound by, and to comply with, these Terms and Conditions. If you do not agree to these Terms and Conditions, please do not use this site. PLEASE NOTE: We reserve the right, at our sole discretion, to change, modify or otherwise alter these Terms and Conditions at any time. Unless otherwise indicated, amendments will become effective immediately. Please review these Terms and Conditions periodically. Your continued use of the Site following the posting of changes and/or modifications will constitute your acceptance of the revised Terms and Conditions and the reasonableness of these standards for notice of changes. For your information, this page was last updated as of the date at the top of these terms and conditions. 2. PRIVACY Please review our Privacy Policy, which also governs your visit to this Site, to understand our practices. 3. LINKED SITES This Site may contain links to other independent third-party Web sites ("Linked Sites"). These Linked Sites are provided solely as a convenience to our visitors. Such Linked Sites are not under our control, and we are not responsible for and does not endorse the content of such Linked Sites, including any information or materials contained on such Linked Sites. You will need to make your own independent judgment regarding your interaction with these Linked Sites. 4. FORWARD LOOKING STATEMENTS All materials reproduced

Figure 28: The Event page from a User Perspective

Once a User has landed on an active Event page, the application attempts to establish a TCP connection between the User and the Server. The Web Socket configuration in the server defines only one endpoint which `SockJS` will make an attempt to connect to - `/events`.

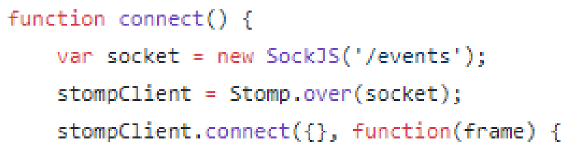
`WebSocketConfig.java`



```
@Override
public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/events").withSockJS().setSessionCookieNeeded(true);
}
```

Figure 29: Configurating of the STOMP endpoint

`event.html`

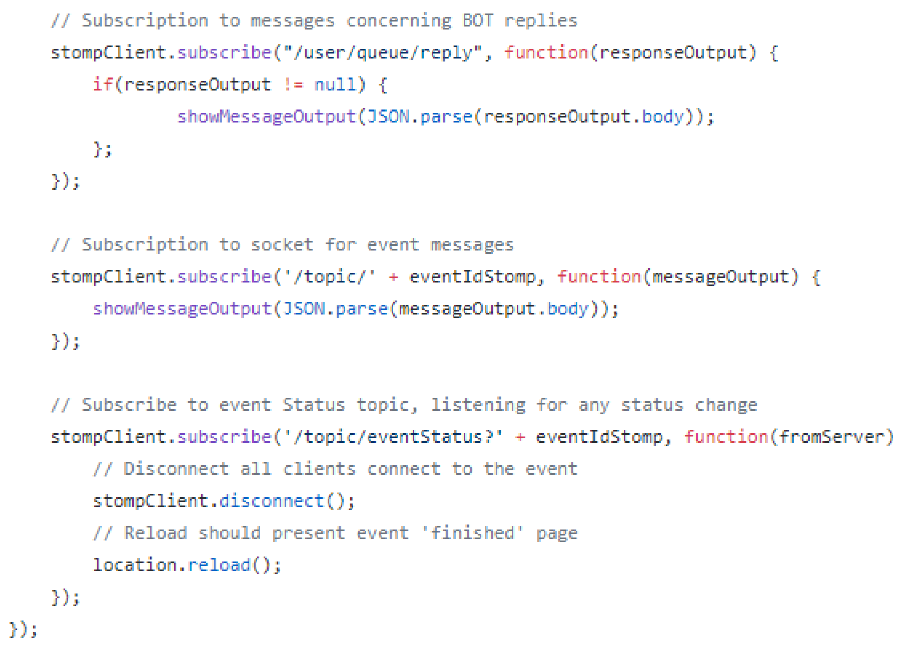


```
function connect() {
    var socket = new SockJS('/events');
    stompClient = Stomp.over(socket);
    stompClient.connect({}, function(frame) {
```

Figure 30: SockJS - establishing a TCP Connection

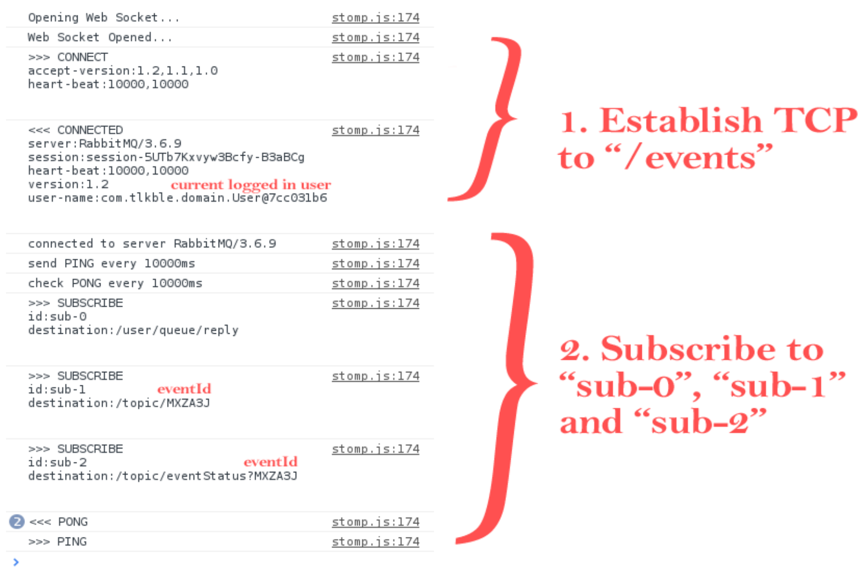
A successful connection to the STOMP endpoint will trigger `SockJS` to subscribe to various STOMP destinations which listens for any messages incoming through `clientOutBoundChannel` of the connection. For this project, the User makes 3 subscriptions, firstly the user subscribes to `/user/queue/reply` which listens for messages returned from the Message moderation method (the implementation for this is reviewed in the next section). Secondly, the user subscribes to `/topic/eventIdStomp` where eventIdStomp is the ID for the current event. This listens for messages posted to the event by other users. Finally the user subscribes to `/topic/eventStatus?eventIdStomp` which listens for any messages from the Server regarding the status of the event, this forms the basis for automatically redirecting a User once the event has ended.

NOTE `/topic/` prefix is added to the subscription requests as these are preconfigured "typical" STOMP destinations and any messages flowing through the `clientInboundChannel` are forwarded to the STOMP broker - in this case `RabbitMQ`.
`event.html`



```
    // Subscription to messages concerning BOT replies
    stompClient.subscribe("/user/queue/reply", function(responseOutput) {
        if(responseOutput != null) {
                showMessageOutput(JSON.parse(responseOutput.body));
        };
    });

    // Subscription to socket for event messages
    stompClient.subscribe('/topic/' + eventIdStomp, function(messageOutput) {
        showMessageOutput(JSON.parse(messageOutput.body));
    });

    // Subscribe to event Status topic, listening for any status change
    stompClient.subscribe('/topic/eventStatus?' + eventIdStomp, function(fromServer)
        // Disconnect all clients connect to the event
        stompClient.disconnect();
        // Reload should present event 'finished' page
        location.reload();
    });
});
```

Figure 31: SockJS - subscribing to destinations



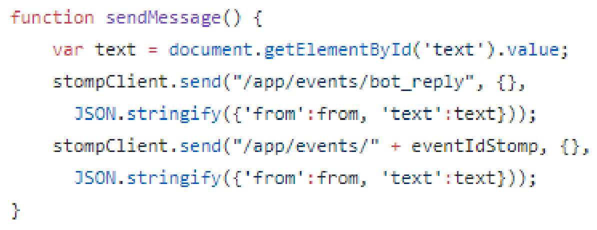
```
Opening Web Socket...                             stomp.js:174
Web Socket Opened...                              stomp.js:174
>>> CONNECT                                       stomp.js:174
accept-version:1.2,1.1,1.0
heart-beat:10000,10000


<<< CONNECTED                                     stomp.js:174
server:RabbitMQ/3.6.9
session:session-5UTb7Kxvyw3Bcfy-B3aBCg
heart-beat:10000,10000
version:1.2
user-name:com.tlkble.domain.User@7cc031b6


connected to server RabbitMQ/3.6.9                stomp.js:174
send PING every 10000ms                           stomp.js:174
check PONG every 10000ms                          stomp.js:174
>>> SUBSCRIBE                                     stomp.js:174
id:sub-0
destination:/user/queue/reply


>>> SUBSCRIBE                                     stomp.js:174
id:sub-1
destination:/topic/MXZA3J


>>> SUBSCRIBE                                     stomp.js:174
id:sub-2
destination:/topic/eventStatus?MXZA3J


2 <<< PONG                                        stomp.js:174
>>> PING                                          stomp.js:174
>
```

current logged in user
eventId
eventId
1. Establish TCP to "/events"
2. Subscribe to "sub-0", "sub-1" and "sub-2"

Figure 32: event.html - console output from SockJS

5.1.8 Posting Messages to Events

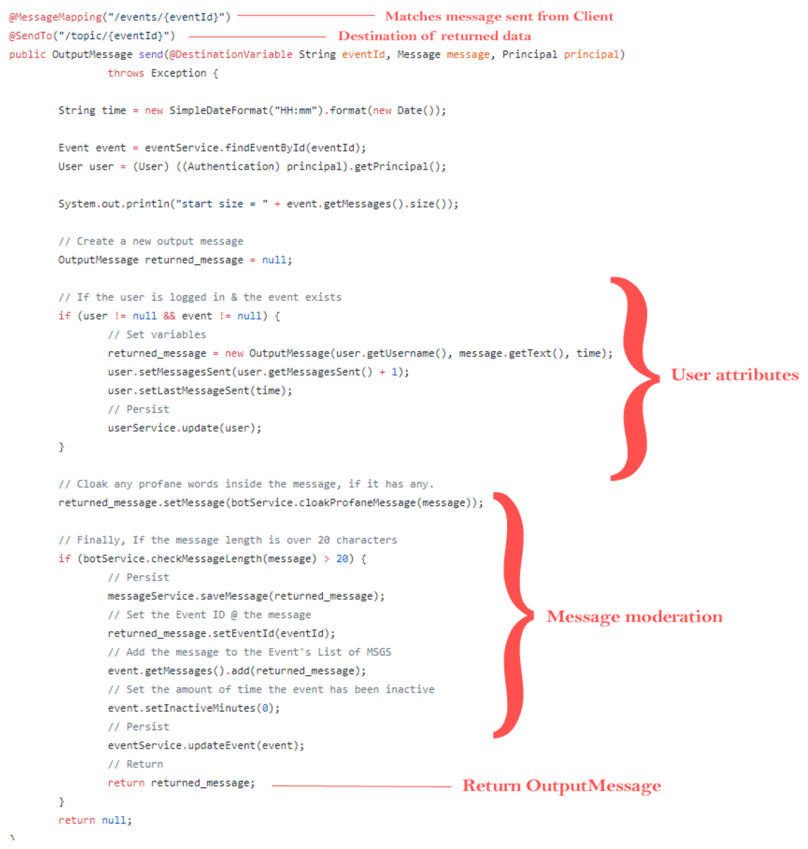
One of the core aims of the project is the ability of the application to support communication between the audience and presenter. After successfully establishing the Web Socket connection and subscribing to the correct destinations, the User is available to post and receive messages from the server and remains idle until either of these transpire.
A User can post a message to an event by entering a sequence of characters in the supplied text area and clicking the button "LET'S TALK". Using the text area's `id`, sockJS extracts the data and sends it to the destination `/app/event/eventId` in JSON format. This is subsequently handled in `MessageController.java`.



```
function sendMessage() {
    var text = document.getElementById('text').value;
    stompClient.send("/app/events/bot_reply", {},
      JSON.stringify({'from':from, 'text':text}));
    stompClient.send("/app/events/" + eventIdStomp, {},
      JSON.stringify({'from':from, 'text':text}));
}
```

Figure 33: SockJS - sending a Message

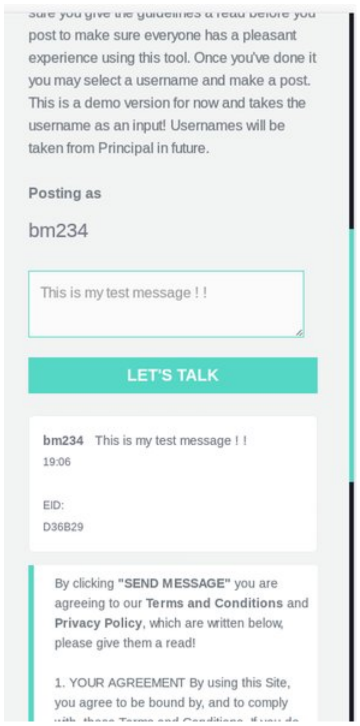
An annotation of `@MessageMapping` is added to the Message Handling method for Event Messages which maps correctly to the destination sent from the Client. This Message Handling method creates a new instance of `OutputMessage` which applies some message moderation and a number of additional attributes and is returned from the method. Another annotation `@SendTo` is added above the method which takes a String parameter - the destination of the returned data. Using the destination of `/topic/eventId` the message is then returned to the Client as this is one of the 3 destination they originally subscribed to. JavaScript is used to create and display HTML DIV elements dynamically containing the `user`, `message`, `eventId` and `time`. The most recent messages are displayed at the top.



```java
@MessageMapping("/events/{eventId}")
@SendTo("/topic/{eventId}")
public OutputMessage send(@DestinationVariable String eventId, Message message, Principal principal)
            throws Exception {

        String time = new SimpleDateFormat("HH:mm").format(new Date());

        Event event = eventService.findEventById(eventId);
        User user = (User) ((Authentication) principal).getPrincipal();

        System.out.println("start size = " + event.getMessages().size());

        // Create a new output message
        OutputMessage returned_message = null;

        // If the user is logged in & the event exists
        if (user != null && event != null) {
                // Set variables
                returned_message = new OutputMessage(user.getUsername(), message.getText(), time);
                user.setMessagesSent(user.getMessagesSent() + 1);
                user.setLastMessageSent(time);
                // Persist
                userService.update(user);
        }

        // Cloak any profane words inside the message, if it has any.
        returned_message.setMessage(botService.cloakProfaneMessage(message));

        // Finally, If the message length is over 20 characters
        if (botService.checkMessageLength(message) > 20) {
                // Persist
                messageService.saveMessage(returned_message);
                // Set the Event ID @ the message
                returned_message.setEventId(eventId);
                // Add the message to the Event's List of MSGS
                event.getMessages().add(returned_message);
                // Set the amount of time the event has been inactive
                event.setInactiveMinutes(0);
                // Persist
                eventService.updateEvent(event);
                // Return
                return returned_message;
        }
        return null;
}
```

Matches message sent from Client
Destination of returned data
User attributes
Message moderation
Return OutputMessage

Figure 34: MessageController.java - Message Handling


sure you give the guidelines a read before you post to make sure everyone has a pleasant experience using this tool. Once you've done it you may select a username and make a post. This is a demo version for now and takes the username as an input! Usernames will be taken from Principal in future.

Posting as

bm234

This is my test message ! !

LET'S TALK

bm234 This is my test message ! !
19:06

EID:
D36B29

By clicking **"SEND MESSAGE"** you are agreeing to our **Terms and Conditions** and **Privacy Policy**, which are written below, please give them a read!

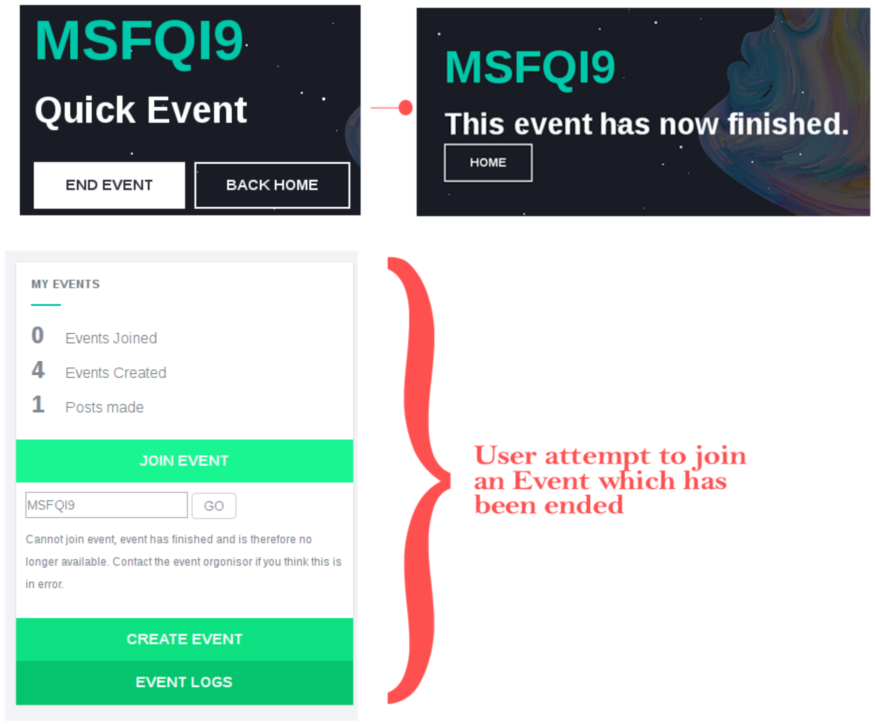
1. YOUR AGREEMENT By using this Site, you agree to be bound by, and to comply

Figure 35: A Message posted to an Event

Once a Message has successfully finished its journey through the application, all Users subscribed to the destination from which the message was sent via `@SendTo` will be able to observe the new Message. Users Message lists update in real-time without any need for page refreshes.

5.1.9 Ending an Event

Users may end an Event they've created by clicking the `‘‘END’’` button. This sets the Events `isAlive` attribute to false signalling the Event has ended and all Users currently on the Event page to redirect instantly to an "EVENT ENDED" page. Once Event's are ended, it is now impossible for Users to join/re-join them. This is also indicated through the quick join form which displays an appropriate message for Event's that are inactive. Regarding the implementation of automatic User redirection, we use a class `SimpMessagingTemplate` which is part of the Spring Messaging Framework. This has a method `convertAndSend(..,...);` which takes 2 parameters. Firstly, the destination of the Message is added, in this case we use the destination `/topic/eventStatus?" + eventId` as this is a destination a Client connected to the Event is subscribed to. Secondly, it takes some data as its second parameter. We just use a Boolean value of `true` to indicate that for the specified destination, there is indeed communication in the `clientOutboundChannel`. On the Client side, this new Message triggers the browsers window to reload which subsequently loads the “EVENT ENDED” page.



Cannot join event, event has finished and is therefore no longer available. Contact the event orgonisor if you think this is in error.

Figure 36: Ending an Event

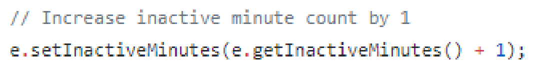
The following snippet of code is responsible for sending the data to the Client.

```
// Send signal to all clients connected that the event has finished
template.convertAndSend("/topic/eventStatus?" + eventId, true);
```

Figure 37: MessageController.java - convertAndSend

5.1.10 Moderation of Events & Messages

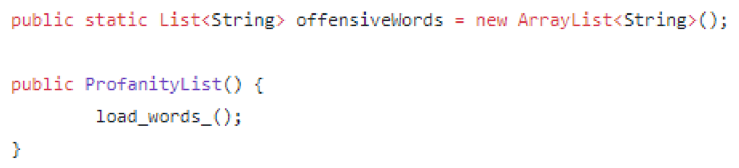
The system also incorporates a feature to end Events which have sustained a period of inactivity and to redirect all Users as this occurs. Event inactivity is defined as an Event which has not received any Messages for a predefined period of 180 minutes. The functionality for this is implemented in `EventController.java`. A method headed with a `@Scheduled` annotation and a `cron` expression of `0 * * ? * *` routinely calls another method every `60` seconds. This other method `normalise()` cycles through all active events in the database and firstly checks if they have inactivity count of >179 minutes, if the Event satisfies this condition then the Event is set as inactive and all Users are signalled to move to an “EVENT ENDED” page using the same code snippet above in figure 53. If the Event does not satisfy this condition meaning it has an inactivity count<180 minutes, the inactiveMinutesCount variable is increased by 1.



```java
// Increase inactive minute count by 1
e.setInactiveMinutes(e.getInactiveMinutes() + 1);
```

Figure 38: EventController.java - increased inactive minute count

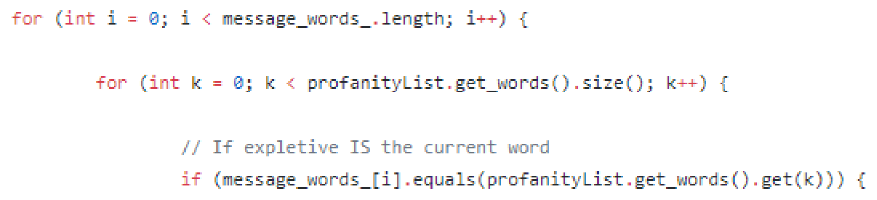
To further the moderation features, User Messages are cloaked if they contain any words which are deemed offensive or profane. A cloaked Message is a Message whose characters are replaced with a ‘*’ apart from the first and the last character. This is implemented by scanning the Messages sent from the Client to `/topic/eventId` in the Message Handling method in `MessageController.java`. A list of offensive words located remotely are loaded into an ArrayList on application boot and each word in the Message is checked to see if it appears in this list.



```java
public static List<String> offensiveWords = new ArrayList<String>();

public ProfanityList() {
        load_words_();
}
```

Figure 39: ProfanityList.java - loading the words

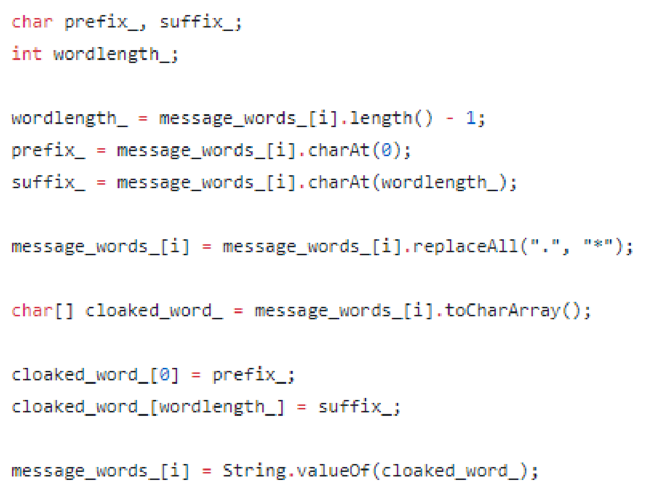
Each word of the Message is split on a space character and each resulting String is checked to see if it exists inside the list of offensive words.



```java
for (int i = 0; i < message_words_.length; i++) {

        for (int k = 0; k < profanityList.get_words().size(); k++) {

                // If expletive IS the current word
                if (message_words_[i].equals(profanityList.get_words().get(k))) {
```

Figure 40: BotService.java - Checking for profanity

If an instance of an offensive word is discovered, it is cloaked by firstly saving a copy of the first character and the last character and then converting all characters of the word to '*'. A new `char` array is declared with the contents of the String of '*' characters. Lastly, the first character and last are returned to their original contents and the original word is replaced with the String conversion of this char array.

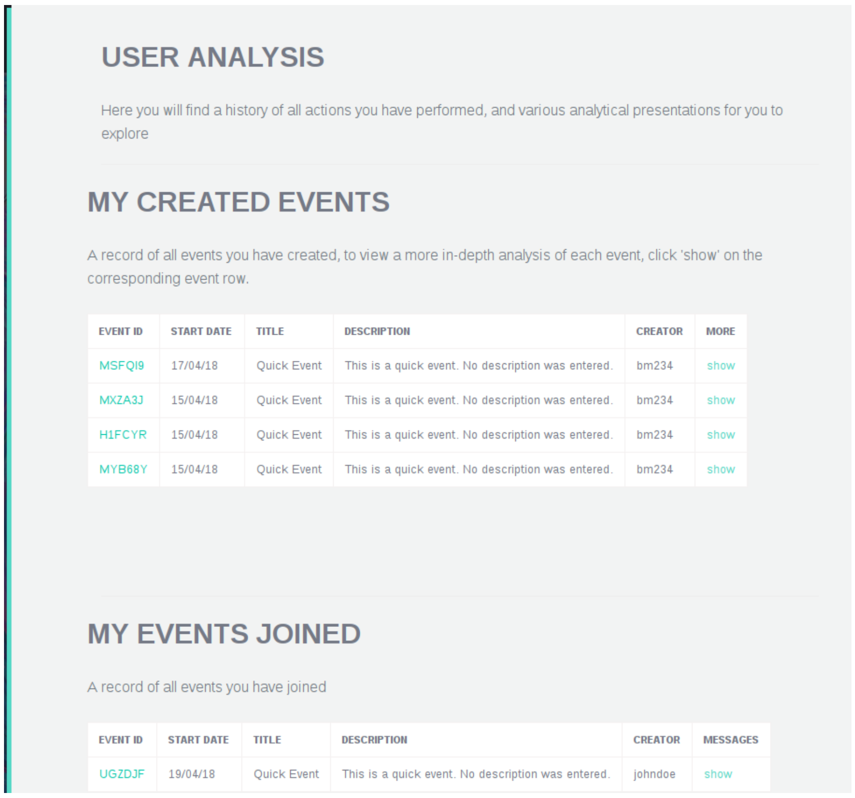


```
char prefix_, suffix_;
int wordlength_;

wordlength_ = message_words_[i].length() - 1;
prefix_ = message_words_[i].charAt(0);
suffix_ = message_words_[i].charAt(wordlength_);

message_words_[i] = message_words_[i].replaceAll(".", "*");

char[] cloaked_word_ = message_words_[i].toCharArray();

cloaked_word_[0] = prefix_;
cloaked_word_[wordlength_] = suffix_;

message_words_[i] = String.valueOf(cloaked_word_);
```

Figure 41: BotService.java - Cloaking an offensive word

5.1.11 Event Logs & Analysis

Users are able to view a complete history of all events they've joined and created by clicking the "EVENT LOGS" button on the home page. There are two tables presented, one for the Events a User has joined and the other for the Events they've created. Other information is held in this table such as Event Id, title, description, join date and creator.



USER ANALYSIS

Here you will find a history of all actions you have performed, and various analytical presentations for you to explore

MY CREATED EVENTS

A record of all events you have created, to view a more in-depth analysis of each event, click 'show' on the corresponding event row.

EVENT ID	START DATE	TITLE	DESCRIPTION	CREATOR	MORE
MSFQI9	17/04/18	Quick Event	This is a quick event. No description was entered.	bm234	show
MXZA3J	15/04/18	Quick Event	This is a quick event. No description was entered.	bm234	show
H1FCYR	15/04/18	Quick Event	This is a quick event. No description was entered.	bm234	show
MYB68Y	15/04/18	Quick Event	This is a quick event. No description was entered.	bm234	show

MY EVENTS JOINED

A record of all events you have joined

EVENT ID	START DATE	TITLE	DESCRIPTION	CREATOR	MESSAGES
UGZDJF	19/04/18	Quick Event	This is a quick event. No description was entered.	johndoe	show

Figure 42: Event Log Page

A `User` class contains attributes `eventsCreated` and `eventsJoined` which is where the data for the table derives from. The list of Events are displayed in reverse order using `Colections.reverse()` so the Events a User has created or joined most recently are placed at the top of the table. In addition to this, a User has the ability to dive into individual analysis on each Event by clicking “show” on the respective Event row. This takes the User to a separate page which homes the WordCloud and any other additional analytics if the system is extended further. However, the functionality of the WordCloud was not entirely implemented. Using a REST service, I was able to get the word frequency for a particular event which forms the input of the WordCloud library, however, due to the complexity of the JavaScript library and time constraints I was not able to fully configure the JavaScript code with the data from the Server.

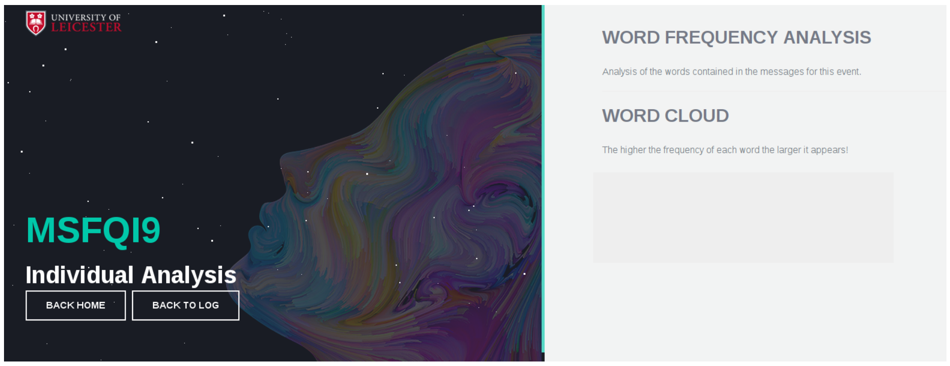
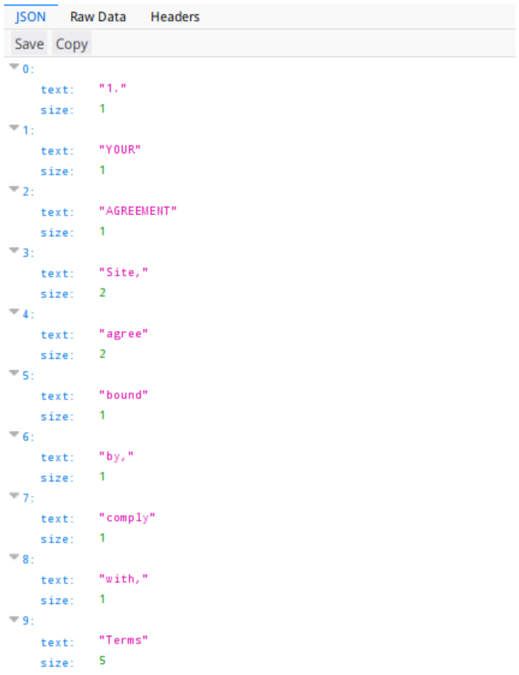

UNIVERSITY OF LEICESTER
MSFQI9
Individual Analysis
BACK HOME
BACK TO LOG
WORD FREQUENCY ANALYSIS
Analysis of the words contained in the messages for this event.
WORD CLOUD
The higher the frequency of each word the larger it appears!

Figure 43: Event Analysis Page

The URL to retrieve an Event's Word Count is `/event/getWordCount?eventid=...`. The result is a list of `WordCount` Objects in JSON which displays like this.



```
▼0:
    text:   "1."
    size:   1
▼1:
    text:   "YOUR"
    size:   1
▼2:
    text:   "AGREEMENT"
    size:   1
▼3:
    text:   "Site,"
    size:   2
▼4:
    text:   "agree"
    size:   2
▼5:
    text:   "bound"
    size:   1
▼6:
    text:   "by,"
    size:   1
▼7:
    text:   "comply"
    size:   1
▼8:
    text:   "with,"
    size:   1
▼9:
    text:   "Terms"
    size:   5
```

Figure 44: REST Service for an Event's Word Counts

5.1.12 Handling JavaScript-disabled Browsers

Important elements of this application require portions of JavaScript code to be executed on Users machine, which of course is not something which can be made compulsory. Maintaining consistency throughout the application is essential to the initial objectives, and in which case the system needs to be able to appropriately handle this scenario without loosing any aesthetic issues or other run-time errors. To implement this, `HTML` offers the <`noscript`> tag which defines an alternate content for Users that have disabled scripts in their browser or have a browser that doesn't support script. By removing individual DOM elements from view using their CSS properties within the `noscript` tag and insert a warning message, I was able to provide a ubiquitous gesture to Users that the application requires scripts to be enabled in order to function correctly.

5.1.13 Securing Communication with SSL

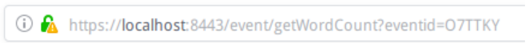
Once all the core features of the system were implemented, it was necessary to secure the application's communication prior to deploying it online. For this project, all communication is secured over HTTPS / SSL (Secure Socket Layer), this ensures all messages sent between Server and Client is encrypted thus protecting sensitive information from intruders such as log in credentials. Firstly, Spring requires an SSL certificate to enable HTTPS, this is achieved by generating a new Key Store via Terminal with the following command.

```
-keyalg RSA -keysize 2048 -keystore keystore.jks -validity 3650
```

After following on-screen instructions, a new Key Store file is created which is placed in the project resources folder to make it accessible to the configuration file. The `application.properties` of the project file is modified with the following entries.

```
# Define a custom port instead of the default 8080
server.port=8443

# Tell Spring Security (if used) to require requests over HTTPS
security.require-ssl=true

# The format used for the keystore
server.ssl.key-store-type=PKCS12
# The path to the keystore containing the certificate
server.ssl.key-store=classpath:keystore.p12
# The password used to generate the certificate
server.ssl.key-store-password=password
# The alias mapped to the certificate
server.ssl.key-alias=tomcat
```

Finally, when the project is rebuilt and Spring Boot is invoked, we can access the application using the HTTPS definition and also with the port number declared in `application.properties`.

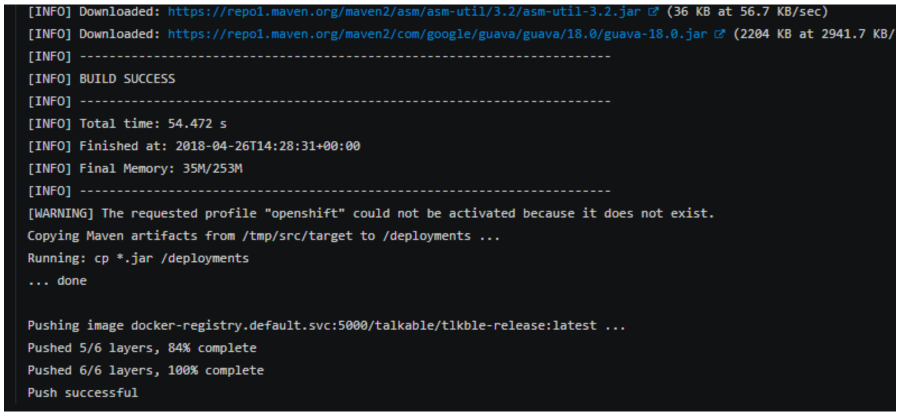


```
https://localhost:8443/event/getWordCount?eventid=O7TTKY
```

Figure 45: Accessing the application through HTTPS

5.1.14 Deployment

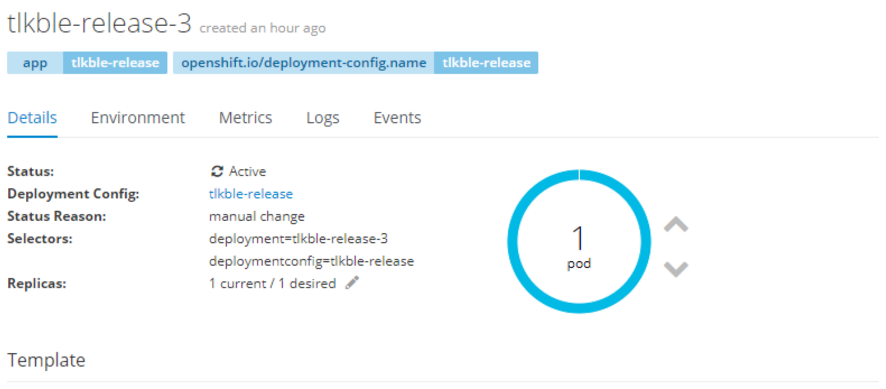
After implementing all features of the system, I then turned my attention towards deploying the application online. There were various services available including Amazon Web Services, however I chose to use `Openshift` as AWS requires financial details to be provided for even using products of their free-tier. Openshift provides automated deployment in a container by linking up with a Git repository. After registering an account, I then created a new `RedHat OpenJDK project` and provided the URL of the project repository. Openshift will then automatically build the project from the files on a remote system.



```
[INFO] Downloaded: https://repo1.maven.org/maven2/asm/asm-util/3.2/asm-util-3.2.jar (36 KB at 56.7 KB/sec)
[INFO] Downloaded: https://repo1.maven.org/maven2/com/google/guava/guava/18.0/guava-18.0.jar (2204 KB at 2941.7 KB/
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 54.472 s
[INFO] Finished at: 2018-04-26T14:28:31+00:00
[INFO] Final Memory: 35M/253M
[INFO] ------------------------------------------------------------------------
[WARNING] The requested profile "openshift" could not be activated because it does not exist.
Copying Maven artifacts from /tmp/src/target to /deployments ...
Running: cp *.jar /deployments
... done

Pushing image docker-registry.default.svc:5000/talkable/tlkble-release:latest ...
Pushed 5/6 layers, 84% complete
Pushed 6/6 layers, 100% complete
Push successful
```

Figure 46: Openshift Web Console 1

After a successful build, the project is to be deployed which through the definition in the pom file invokes Spring Boot. These 2 phases should produce a non-terminating program which is supported by Openshift Pods (system and network resources). At this point, a URL is available to access the Web Application and settings such as routing are available to configure the container to provide secure communication (HTTPS over HTTP). The actual URL for the application is `tlkble-release-talkable.7e14.starter-us-west-2.openshiftapps.com` but has been shortened to `www.talkable.cf` for convenience.



tlkble-release-3 created an hour ago

app tlkble-release | openshift.io/deployment-config.name tlkble-release

Details | Environment | Metrics | Logs | Events

Status:	Active
Deployment Config:	tlkble-release
Status Reason:	manual change
Selectors:	deployment=tlkble-release-3
	deploymentconfig=tlkble-release
Replicas:	1 current / 1 desired

1 pod

Template

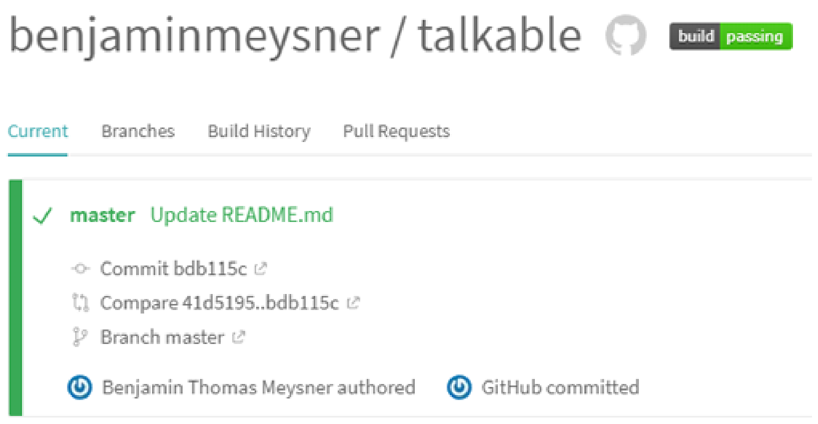
Figure 47: Openshift Web Console 2

5.2 Testing the System

The project's test suite consists of unit testing, integration testing and manual testing, all of which were carried out in an agile manner. However, there were a few issues encountered when implementing unfamiliar frameworks for automated testing, all of which are discussed further in this section.

5.2.1 Test Strategy & Plan

Executing this project in an agile environment meant that testing was to be carried out upon completion of every feature. Regular updates were pushed to Github with their respective Tests which were automatically executed and reported back using `Travis`, a Continuous Integration (CI) server.



benjaminmeysner / talkable build passing

Current | Branches | Build History | Pull Requests

✓ master Update README.md

Commit bdb115c

Compare 41d5195..bdb115c

Branch master

Benjamin Thomas Meysner authored | GitHub committed

Figure 48: Travis CI Report

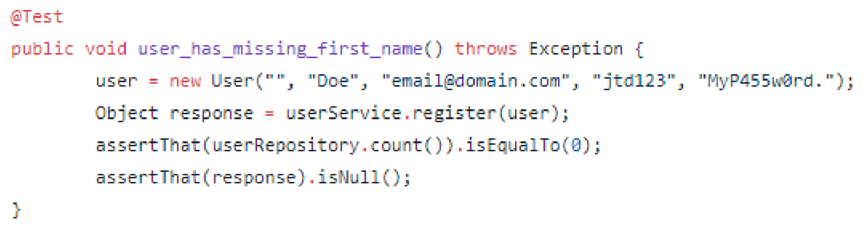
Unit tests would be used to probe the integrity of the feature's components in a stand-alone climate, followed by integration tests to examine how multiple components of the application behave together upon implementation. Initial test plans were established highlighting particular testing frameworks to facilitate automated testing and to provide better coverage. The initial inclusion of `Selenium`, an automation testing framework, was made in the plan to test User Interface (UI) interaction to increase time efficiency and support less manual testing in this area. However, due to many complications surrounding configuring `Selenium` WebDriver versions with the respective versions of Web Browsers it was later dropped from the plan. The reason for this is because I believed the time it would take to set up correctly and write the Java code for running the automation would far extend the time I had available to deploy that feature, therefore the project would fail to fall inline with the plan. I decided to replace automation testing of the User Interface with thorough manual testing.

Server-Side test cases were to be completed with `JUnit` and `Mockito` frameworks whilst Client-side tests were to be either completed with a JavaScript library known as `MochaJS` or through thorough Manual Testing. Automated tests are executed during the `test` phase of the build life-cycle using Maven's `Surefire` plugin [2]. `Surefire` generates a report in `.html` format where results of all test executions are displayed.

5.2.2 Test Data

5.2.2.1 Automated Test Cases

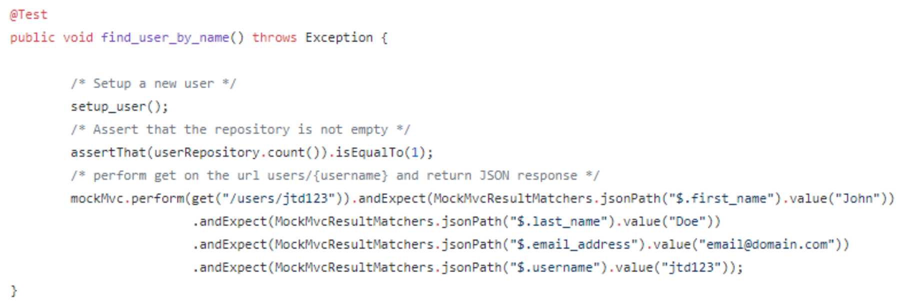
All automated test cases reside in `src/test/java/com/tlkble/suite` and were separated into different classes titled by the controller they are associated with. Unit Tests were written to check multiple variations of the input domain for Server-Side form validation for User log in and registration. Extra unit test were also written to inspect the functionality of Message moderation methods, this includes tests for discovering profane words and also for cloaking mechanisms.



```
@Test
public void user_has_missing_first_name() throws Exception {
        user = new User("", "Doe", "email@domain.com", "jtd123", "MyP455w0rd.");
        Object response = userService.register(user);
        assertThat(userRepository.count()).isEqualTo(0);
        assertThat(response).isNull();
}
```

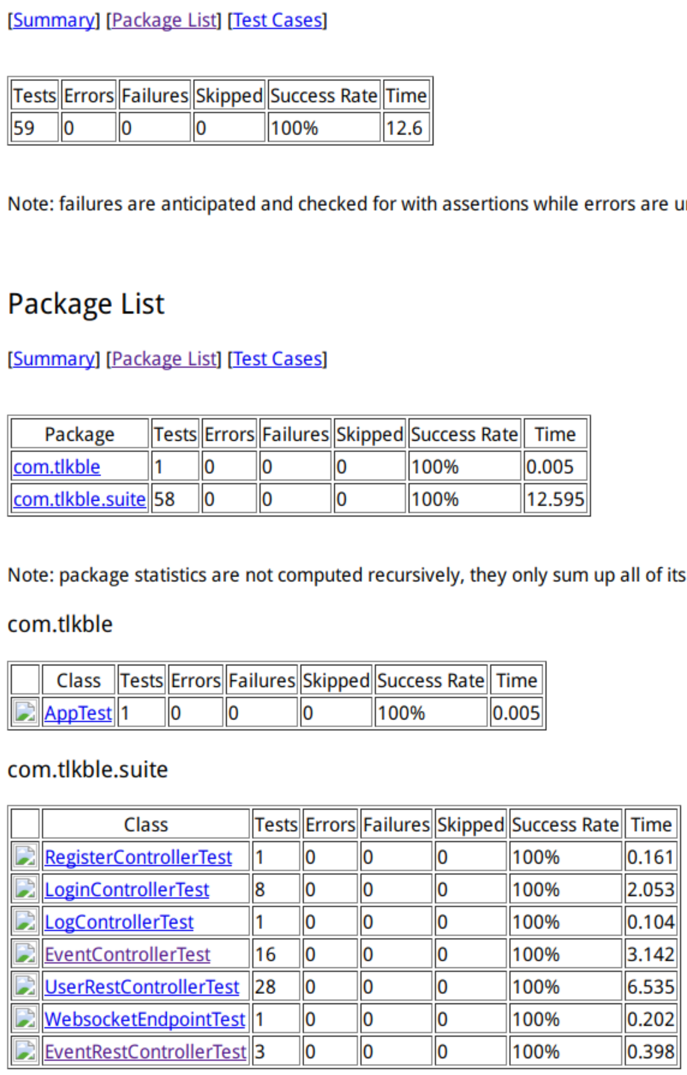
Figure 49: Example of a Unit Test

Integration tests included ensuring data is persisted correctly and also responses from REST services. Furthermore, testing of both authentication and authorisation for page access using Spring Security is included. Below is a summary of the Surefire report generated at compile time. The full report of executed tests can be found in the package Test_Report/sure-fire.html.



```
@Test
public void find_user_by_name() throws Exception {

        /* Setup a new user */
        setup_user();
        /* Assert that the repository is not empty */
        assertThat(userRepository.count()).isEqualTo(1);
        /* perform get on the url users/{username} and return JSON response */
        mockMvc.perform(get("/users/jtd123")).andExpect(MockMvcResultMatchers.jsonPath("$.first_name").value("John"))
                        .andExpect(MockMvcResultMatchers.jsonPath("$.last_name").value("Doe"))
                        .andExpect(MockMvcResultMatchers.jsonPath("$.email_address").value("email@domain.com"))
                        .andExpect(MockMvcResultMatchers.jsonPath("$.username").value("jtd123"));
}
```

Figure 50: Example of an Integration Test


[Summary] [Package List] [Test Cases]

Tests	Errors	Failures	Skipped	Success Rate	Time
59	0	0	0	100%	12.6

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[Summary] [Package List] [Test Cases]

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
com.tlkble	1	0	0	0	100%	0.005
com.tlkble.suite	58	0	0	0	100%	12.595

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

com.tlkble

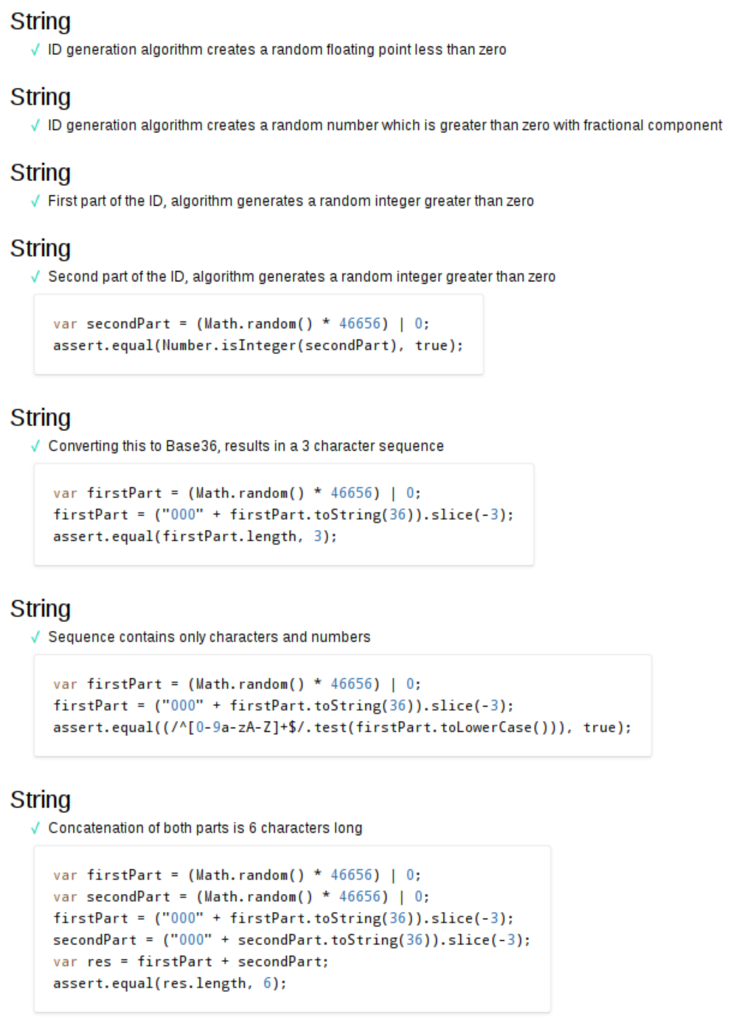
	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	AppTest	1	0	0	0	100%	0.005

com.tlkble.suite

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	RegisterControllerTest	1	0	0	0	100%	0.161
	LoginControllerTest	8	0	0	0	100%	2.053
	LogControllerTest	1	0	0	0	100%	0.104
	EventControllerTest	16	0	0	0	100%	3.142
	UserRestControllerTest	28	0	0	0	100%	6.535
	WebsocketEndpointTest	1	0	0	0	100%	0.202
	EventRestControllerTest	3	0	0	0	100%	0.398

Figure 51: Surefire Test Report

`MochaJS` was used in order to thoroughly test the algorithm step-by-step which generates the ID of an Event. MochaJS also generates a report based on the results of the assertions, this is summarised below.



String

√ ID generation algorithm creates a random floating point less than zero

String

√ ID generation algorithm creates a random number which is greater than zero with fractional component

String

√ First part of the ID, algorithm generates a random integer greater than zero

String

√ Second part of the ID, algorithm generates a random integer greater than zero

```
var secondPart = (Math.random() * 46656) | 0;
assert.equal(Number.isInteger(secondPart), true);
```

String

√ Converting this to Base36, results in a 3 character sequence

```
var firstPart = (Math.random() * 46656) | 0;
firstPart = ("000" + firstPart.toString(36)).slice(-3);
assert.equal(firstPart.length, 3);
```

String

√ Sequence contains only characters and numbers

```
var firstPart = (Math.random() * 46656) | 0;
firstPart = ("000" + firstPart.toString(36)).slice(-3);
assert.equal((/^[0-9a-zA-Z]+$/.test(firstPart.toLowerCase())), true);
```

String

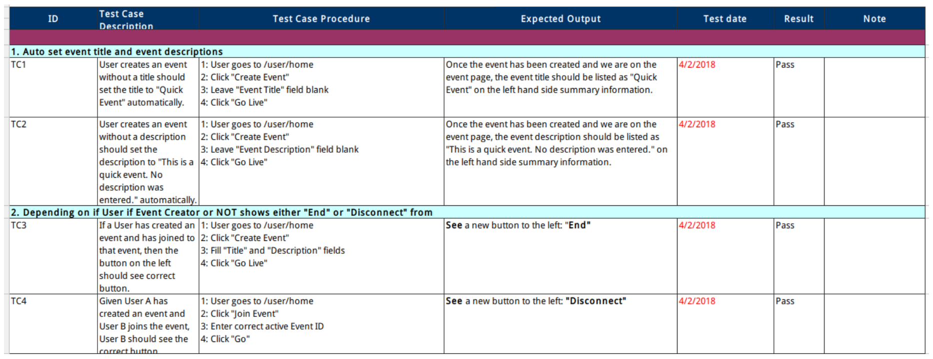
√ Concatenation of both parts is 6 characters long

```
var firstPart = (Math.random() * 46656) | 0;
var secondPart = (Math.random() * 46656) | 0;
firstPart = ("000" + firstPart.toString(36)).slice(-3);
secondPart = ("000" + secondPart.toString(36)).slice(-3);
var res = firstPart + secondPart;
assert.equal(res.length, 6);
```

Figure 52: MochaJS Test Report

5.2.2.2 Manual Test Cases

Manual Testing was not originally part of the test plan but later replaced selenium's automated testing due to complications that arose during the development process. These tests were written to specifically assess how the interface components react to different client scenarios. Tests include the visibility of a `‘‘diconnect’’` or `‘‘end’’` button on an Event page depending on whether a User is a creator or not, automated configuring of Event title and descriptions, further tests around joining and creating events, statistical counters, further WebSocket functionality and scheduled tasks forcing user redirection. The full document of manual test cases can be found in `src/test/java/talkable/suite/manual_tests/`.



ID	Test Case Description	Test Case Procedure	Expected Output	Test date	Result	Note
1. Auto set event title and event descriptions						
TC1	User creates an event without a title should set the title to "Quick Event" automatically.	1: User goes to /user/home 2: Click "Create Event" 3: Leave "Event Title" field blank 4: Click "Go Live"	Once the event has been created and we are on the event page, the event title should be listed as "Quick Event" on the left hand side summary information.	4/2/2018	Pass	
TC2	User creates an event without a description should set the description to "This is a quick event. No description was entered." automatically.	1: User goes to /user/home 2: Click "Create Event" 3: Leave "Event Description" field blank 4: Click "Go Live"	Once the event has been created and we are on the event page, the event description should be listed as "This is a quick event. No description was entered." on the left hand side summary information.	4/2/2018	Pass	
2. Depending on if User if Event Creator or NOT shows either "End" or "Disconnect" from						
TC3	If a User has created an event and has joined to that event, then the button on the left should see correct button.	1: User goes to /user/home 2: Click "Create Event" 3: Fill "Title" and "Description" fields 4: Click "Go Live"	**See** a new button to the left: **"End"**	4/2/2018	Pass	
TC4	Given User A has created an event and User B joins the event, User B should see the correct button	1: User goes to /user/home 2: Click "Join Event" 3: Enter correct active Event ID 4: Click "Go"	**See** a new button to the left: **"Disconnect"**	4/2/2018	Pass	

Figure 53: Snippet from manual_tests.xsl

5.2.3 Evaluation of Test Data & Results

Test Cases were written in an attempt to provide full coverage including boundary testing where test cases were written using the extremes of the input domain. A total of 100+ tests were created with all of them passing. However, I am not totally satisfied with the testing of this software as I believe in some areas, some were written for the sake of a passing test in order to carry on with development. In hindsight, there should of been more thought applied to writing the correct tests to capture the full dexterity of the application with much more emphasis placed on stress testing, especially as this is a product which is designed to be used by many users. This criticism is applicable mainly to automated tests regarding event messaging where they exist few and far between. To improve on this approach for future projects, it may be of use to implement a test coverage tool to measure how much of the software is actually being tested and how much could be considered unreliable. Looking back at the original aims and requirements, it was imperative that the application would satisfy certain performance criteria, this is something that was not considered at any point during development and is something that I would include in future projects.

Looking positively, with the tests that were written, I was able to locate errors in the project I may not have noticed previously and for that, I have increased confidence that important features of the system will behave as expected. I carried out the testing components of this project in an agile manner to the best of my ability to enhance my experience of a technique used commonly for modern day project management.

6 On Reflection

6.1 Critical Appraisal

In order to evaluate the project in its entirety, I need to refer back to the original aims of the project. One extract from the aim,

"Users of this application will need to have the ability to easily create events through an interface which are associated to their live presentations. Audience members will then be able to join these created events and communicate with real-time messaging through their devices in a safe and friendly environment".

Regarding this statement, I can establish that this has been a success, I have built a piece of software that can handle the creation of events through generation of unique Event Ids. I have developed features which facilitate live communication in Events between audience members as registered User's of the system. This component of the project aim remained static through the whole project life-cycle and remained clear throughout the development phase. Secondly,

"The interface of the application will be streamlined and lightweight to ensure swift access to its features, and also be be visually stimulating for a enjoyable user experience.".

Using technologies such as Thymeleaf, Bootstrap, JavaScript and advanced CSS properties I was able to make an application that was visually engaging and most importantly remained responsive across mobile phone, tablet and desktop screen sizes. I dedicated large amounts of time to the front-end aspects. I incorporated CSS animations for DIV elements to make the interface seem more fluid and less rigid. I appreciated the role of good web design techniques, including colour palettes and layout consistency. Although I am satisfied with the user interface and UX components, the experienced gained has driven my standards even higher, and some small inconsistencies with formatting and untidy code would prevent me from publishing my work unless remedied. In future projects, Angular JS or other more modern technologies would be preferred to Thymeleaf, this would express UI development and introduce more interesting features such as two-way data binding which provides a platform to create an even more enriching experience. Despite the fundamental success of the project, the final component making up the aims,

"The application will provide graphical representations of data for each event which can assist the improvement of live presentations".

This is something that I do not consider a success. This was a feature that was to be implemented after completing all other functionality, and having left the development of it late I was unable to fully implement it to a high standard.

The lack of time for development was a product of poor understanding of the cost of development during the planning phase of the project. Early on, I introduced many requirements to the plan which were given too much priority and evidently skewed the allocated development times for integral requirements. For future projects, I would place more importance on creating a better defined list of objectives which in turn would create a well structured list of requirements, this should exclude spending valuable time on features which are not integral to the original aims.

In addition to this, although a considerable amount of testing has been conducted, test coverage was much lower than expected and I do not believe that, in its current state, the product is ready for commercial release. I can attribute this to little experience of writing good test cases and the importance they have to ensure the stability of an application. Looking back, I would of shared more of my attention towards testing all aspects of the requirements, formulating a test plan with the appropriate tools and frameworks required.

Having said that, I can say that the project on a whole was more successful than not. I can attribute most accomplishments on this journey to having well defined aims and objectives. Applying agile practices and breaking the product up into individual features or deliverables was crucial to remaining on track without falling dangerously behind or losing sight of the goal.

6.2 Social, Economic and Commercial Context

The original idea for this project was to develop a free product which can adapt to the requirements of the User and be used not only for academic purposes as suggested in both the abstract and introduction but for business also. I regard the finished product to be able to manage this and increase audience participation in presentations of different industries. The technologies within mean that it could also be highly scalable and has the capability to handle increasingly large quantities of data. The risks attached to this product is mainly due to a single account type, this means that there is currently no way of moderating user activity through the interface, through means of an administrator account or user accounts with added privileges. For this reason, users could potentially post harmful content without running the risk of having their messages or accounts removed. This could pose issues if used in highly professional capacities.

6.3 Extending the system further

If I was to consider extending the software further, I would first include both an account of type “ADMIN” which would have the privileges to remove users, posts and events from the database. Secondly I would include increased privileges for event creators so they can moderate connected users and messages as they see fit. Another feature would include implementing a voting system for posts so that users can “up vote” or “down vote” messages. This would provide greater insight for the event creators as they would be able to view in real-time, issues that need addressing. In this project I have implemented some basic message moderation with scanning for profane words and cloaking mechanisms. However, to take this functionality a step further, I would like to incorporate a minimum measure of dissimilarity between message words and offensive words from a structure by counting the number of character edits taken to achieve it. This measure is known as the Levenshtein distance and the algorithm can be translated to Java fairly easily.

6.4 Personal Development

Upon completing University, my desired profession was to be involved in Web Development or within Software Engineering. This was reflected in the kind of project I undertook for this module. Having no professional experience in either of these fields, It was necessary for me to build up as many skills and experience as I possibly could on this journey. One of these challenges was to develop the Web Application using the Spring framework having only used it in a limited capacity beforehand. Since then, I have become confident with this framework and in turn vastly more efficient with my development. I chose a project which incorporates Web Sockets to expose myself to an unfamiliar technology and add a new weapon to my arsenal. Again, this was a challenge for me as there were many new concepts, protocols and libraries that I had not worked previously such as STOMP and SockJS.

Finally, this module has enhanced my experience in project development. It has allowed me to understand the differences between project aims, objectives and requirements and the importance each one plays in the role of a project life-cycle. Combining the development of the product with agile practices has hugely benefited my knowledge of delivering a project to specification and will hopefully aid me in my future endeavours.

6.5 Conclusion

In summary this project has been largely successful. I have planned, designed and developed a web application which mostly achieves the original aims and objectives which were set. Critical aims such as the creation of events and real-time messaging have been implemented to a good standard. The application boasts a slick interface which is engaging, interactive, and imperatively is responsive across all common devices. There were features which should have been implemented but where left out due to time constraints, I do not see this as failure but as a learning experience for improving my management skills within future projects. Technology is constantly evolving as should the education industry, if this product can be viewed as a step in the right direction and deemed a useful idea, then to me it has been a success.

References

[1] Glossophobia (2001), Do you suffer from glossophobia? http://www.glossophobia.com/ . Cited 4 April 2018

[2] Apache (2004-2008), Maven Surefire Plugin http://maven.apache.org/surefire/maven-surefire-plugin/ . Cited 23 April 2018