

2015

PROJECT

HIGHER DIPLOMA IN SCIENCE
IN COMPUTER SCIENCE (NFQ - LEVEL 8)
2014 -2015

FINAL REPORT

Michael Bolger
20011937
5/5/2015



Abstract.....	4
Introduction	5
Aim	5
Scope/Objectives	6
Approach.....	7
Project Analysis and Specification.....	8
Feasibility Study	8
Site characterisation.....	8
Website	10
User Requirements	11
System Requirements	11
Functional Requirements	11
Proposed Technologies	12
Hardware.....	12
Software	12
The MEAN stack	12
CSS.....	15
Git.....	16
Testing framework	17
JavaScript Testing.....	17
Testing framework	20
Methodology.....	21
Initial Starting Point and Research	21
Waterfall	21
Agile.....	22
Scrum	22
Development Approach	22
Developers Tools.....	23
Prototypes.....	24
Initial Java Approach	24
Alternative JavaScript Approach	24
Final Approach	24
Risk Management	25
Requirements.....	25

Skills.....	25
Risks.....	25
Management of the Project	26
NodeJS.....	26
NPM.....	26
Express	27
MongoDB	27
AngularJS.....	28
Initial Starting Point.....	28
Building an API	29
Integrating Angular	30
Representing Post Data.....	30
Automating Node.....	31
Build Automation	31
Grunt	31
Gulp.....	31
Adding Security to the Site.....	32
Cookie Based Authentication	33
Token-Based Authentication	34
Bcrypt	36
Authentication	37
Routing	38
webRTC	39
Building the UI.....	41
Screen size testing.....	48
Requests and Response Codes.....	49
Single Page Application	50
End-to-End Testing.....	52
Further Testing.....	52
Recommended Reading	53
Conclusions	54
References	56

Abstract

This project revolves around leveraging the technologies found in the MEAN stack to create a fully functioning web application build around JavaScript. This report provides an outline of which technologies were chosen when building the web app, many of these technologies were incorporated via node modules with the intention of adding functionality to the site. Incorporating the MEAN stack into the design of the web app has proven to be a great learning experience, utilising relatively new technologies in new and interesting ways as a means of web development.

Introduction

On building projects where a percolation area is to be constructed it is important to assess site conditions such as drainage times, water levels and soil types in advance. In these cases a site characterisation is conducted that will assess and document these site-specific conditions. Traditionally, site characterisation is physically documented by the assessor, who walks the site noting onsite-conditions and performing percolation testing.

Percolation testing is the main aspect in determining site suitability, to achieve this six trial holes are dug in the area where the proposed percolation apparatus is to be constructed, with three surface holes and three sub surface holes at a depth of 600mm.

All these tests are traditionally documented manually via a notepad that then has to be transcribed onto digital format for analysis, submission and storage. The initial starting point of this project was to develop a traditional java based website using the eclipse api and play framework which would talk to a dedicated android app via a restful service. Upon further research of available frameworks a decision was made to use the MEAN stack consisting of Mongo, Express, Android and Node to develop the project. The website will be styled in such a way that will allow for access on devices with various screen resolutions, this will allow users to input data in the field away from the traditional office environment.

Aim

The aim of this project will be to develop a site investigation website using the mean stack framework consisting of Mongo, Express, Angular and Node, that will allow for the storage and posting of data.

Scope/Objectives

There are a number of objective key to the development of this project. The core features of communication between the website and database is the major principle of the project, to this end it was proposed to develop the following:

- A website capable of collecting data regarding site characterisation, specifically percolation testing.
- Incorporation of the MEAN stack-programming framework onto the project.
- The implementation of security features to the website to prevent against unauthorised access by a third party.

In addition to the core features outlined above it is proposed to time permitting to develop the following additional features which will complement the core development framework.

- Communication from the mobile app to the website via a real time communication framework such as webRTC.
- The hosting of the web application on a cloud service such as Heroku, or the development of a cloud service within a docker container hosted on a server.

Approach

The current approach of recording site investigation information is via a pdf document, which allows users to input data into various fields, this data would then be printed, distributed and finally a copy stored for reference purposes. To ease the reliance of the pen and paper and physical storage method a website will instead be created to store site characterisation data not only for individual company use, but also to develop a database of projects which could be easily referenced using one website without the need to visit the local authority to gather physical data on stored projects. In essence the project intends to cut down on the overheads of physical documentation control and act as an open source geographical reference for these forms.

To determine which framework would be used in the development of this website, a number of programming frameworks were assessed for use. From a traditional standpoint a java-based approach could have been implemented using a relational database such as MySQL and a framework such as play using a model view controller approach.

An alternative to this traditional method is the MEAN stack, which utilises various JavaScript languages to build web applications and store them in a non-relational database. MEAN stands for the technologies used in developing projects, these are Mongo db, Express, Angular.js and Node.js.

A means of hosting the web service is important, as it will create an online presence for the project; a number of options are open in this regard. The site may be hosted on a dedicated cloud service such as cloudbees or Heroku; alternatively the service could be hosted within a docker container on an AWS instance.

Project Analysis and Specification

Site characterisation is used to determine characteristics such as permeability, soil type, drainage times and other site-specific information. This data is used in conjunction with the Ordnance Survey of Ireland (osi) website to determine if a specific site is appropriate for construction of an onsite drainage scheme such as a percolation areas receiving discharge from primary treatment systems such as septic tanks (osi, 2015). This project proposes to develop a website that will be capable of recording the results of on-site characterisation such as percolation testing and site-specific comments noted during site visitation.

This chapter will outline the proposed analysis and specification for the project outlining the requirements from a users point of view this will include both functional and non-functional requirements.

Feasibility Study

Site characterisation

The typical site investigation procedure for sites without adjacent drainage schemes involves percolation testing to determine the feasibility of onsite drainage schemes that require discharge of treated water via a percolation area. Data is usually gathered by means of taking notes on a notepad noting site-specific conditions, which may be relevant, and would then be transcribed into a pdf document.

The aim of this project is to emulate the site characterisation form from the development of the website. The form itself is broken down into seven sections into which the assigned engineer will input relevant data, these sections are:

Section 1 - General Details

In this section general details are input such as site owners details along with contact information. Also present in this section are development specific information such as the number of Residents, types and designations of bedrooms and the proposed source of the water supply from a public main or private well.

Section 2 - General Details Cont

This section of the documentation continues with gathering general site specific details, in this case soil and aquifer type. This section also ascertains points pertinent to the surrounding features such as additional water schemes within 1km and any protection schemes in the area be it from ground water protection to the prescience of site of Archaeological significance.

Section 3 - Site Assessment

This section covers the visual inspection of the proposed site, this can be conducted from a site reconnaissance or from a desk study garnering details from the ordinance survey website and relevant maps and planning permissions. In this section features within 250m of the proposed site are measured, these features include nearby houses, roads and geographical features.

This section also covers on-site testing such as trial holes, T-tests, P-tests. These tests are the essential aspects of site investigations and will determine if the site is suitable for an on-site drainage scheme and will greatly influence its design.

Section 4 - Conclusions of Site Characterisation

In this section the site is determined to be suitable or not suitable for development. When a site has been designated suitable for development the type of treatment plant is selected along with secondary and tertiary treatment if needed. Also designated in this section is the discharge route for treated water.

Section 5 - Recommendation

This section covers the engineers' recommendations as to the type of treatment system for the site along with the discharge route and the depth of the system. Also in this section it is required to enter site-specific conditions, which cover special works such as site improvement and testing.

Section 6 - Treatment System Details

Proposed primary, secondary and tertiary treatment systems are outlined in this section. Primary treatment requires input of the proposed tank capacity, percolations type i.e. sub-surface or mounded, and the size and number of percolation trenches.

Secondary treatment systems require details of proposed filter systems such as proposed area, depth, invert level and site-specific features such as soil type, constructed wetlands etc. The tertiary

treatment system is used to input data regarding the design of a polishing filter or gravity fed system. These tertiary treatment systems are often optional in the design of treatment systems.

Also included in this section is the discharge route for drainage water and includes details as to the quality of the discharged water. Quality assurance is outlined with installation and commissioning details being noted along with recommendations as to on-going maintenance schedule.

Section 7 - Assessor details

Engineers' details are input in this section such as company, name and address. Additional details such as engineers' qualifications, report date, and insurance details are also required in this section of the site characterisation form.

Website

The initial approach for creating the website was to use traditional web development practices known to the author, these being java, MySQL, HTML and the play framework. The eclipse IDE was to be used as the development environment for this approach. To determine the feasibility of this method a sample mock-up of the project was created incorporating user signup, project creation and storage. This working mock-up of the site was created in the initial months of the project.

As an alternative to the java based approach to web development a JavaScript approach was also researched. During this research the MEAN method of web development stood out as an interesting method in which to create the website. A mock-up site was also created using this method. While comparing these two methods it was decided to use the MEAN stack method to give the author grounding in JavaScript languages rather than rely on the existing knowledge of java and the play framework.

When creating the website the overall goal was to have WebPages viewable on a range of screen sizes without the loss of fidelity or compromising user interactions on a smaller screen such a mobile phone. To achieve this it was proposed to use a Bootstrap model that would be compatible with a multitude of screen sizes while giving the user a clean sleek user interface.

This website was to be active with a Mongo database running in the background serving as a means of storage for not only test results but also users details, there would also be a number of models within the project.

To development these services Sublime text was used for development of the website and twitter bootstrap for styling. To allow for testing of the ui on various screen sizes it was proposed to use Genymotion to emulate smaller screen sizes found on mobile devices.

User Requirements

The website will be accessible to the user through a graphical user interface (GUI) which will allow for the input of data on the website.

The user upon accessing the website will be required to sign in to create a project

The user must be able to retrieve data relating to their project, as the website is an open resource the user must also be capable of viewing other site characterisation tests on the site as they see fit.

System Requirements

Functional Requirements

To enable the various aspects of the project to function in unison there are a number of essential functional requirements that will need to be filled.

These requirements are:

The web-code and database must be of a standard to be deployable to the web; this includes GUI, database and various security features to protect user credentials.

The various services that go into making up the project must be able to communicate with each other, where communication is not possible an alternative means of storing data for later retrieval will need to be implemented.

A testing method is required to test the various aspects of the website for functionality

Proposed Technologies

Hardware

To develop the project access to a number of hardware tools were required for development and testing. Hardware requirements for the project include the following.

Access to a computer capable of running windows or a mac osx environment.

A mobile device or emulator to test functionality of the website on various screen sizes.

The project relies on the use of various standard web technologies, due to this testing can be conducted for much of the project running on localhost, before hosting on a web server. It is proposed to use a machine running on a Mac osx environment for development of the various aspects of the project.

Software

The software needs for the project are minimal, a text editor such as test edit, sublime or NotePad++ will be used for the majority if not all coding for the website on both the front and back end.

Standard web technologies are proposed in the development of the website these include HTML, CSS and JavaScript.

The MEAN stack

The term MEAN stack is used to describe a collection of JavaScript based technologies that can be used in the development of web applications. MEAN is used as a term to describe the following technologies: Mongo DB, Express, Angular.js and Node.js, (An Introduction to the MEAN Stack. 2015). These technologies have been chosen due to their popularity in JavaScript development, the technologies mentioned are by no means set in stone and can be supplemented or replaced with comparable technologies as needed depending on the users experience and preference.

Mongo DB



“Mongo is an open source schema less database system which is very different from the more popular MySQL. The most considerable differences are that MySQL is written using SQL queries, while Mongo DB is focused on BSON (Binary JSON) (Hongkiat, 2015).”

Express

Express is a minimal and flexible node.js web application framework, providing a robust set of features for building single and multi-page, and hybrid web applications.



Express is a minimal framework using the “less is more” approach to programming providing a minimal but robust layer. The idea behind express is to make it as flexible as possible, the default express application give a very minimal framework which can be added to as needed.

This project is designed as a web application, the distinction the author had to make when designing the project was whether the project would take the form of a single-page web application or a multi-page or hybrid web application.

Single-page web Application

A Single-page application is a relatively new idea, instead of a website initiating a network request every time a new page is navigated to a single page application downloads the entire site to the users browser with the intention of speeding up the navigation of the website as pages are pre-loaded. These single page applications are facilitated when using angular or ember, which is compatible with express. This idea of loading data in advance may work well on smaller sites or dedicated applications using a specific set of parameters, however on larger site it would not be productive to load all or the majority of the data in advance due to size (Brown. E, 2014).

Multi-page web Applications

Multi-page web applications follow a more traditional method in serving up data; each page is provided using a separate request to the server. In this method the content delivery can be delivered individually per page and can provide more flexibility when designing the web application.

Approach

The decision was made to develop the project using a Single-page web application approach, this was in part due to the speed increase that would arise from pre loading data and also the fact that the web application is relatively small and the pre-loading times should be minimal giving the end user a more streamlined and faster navigation experience.

Angular.js

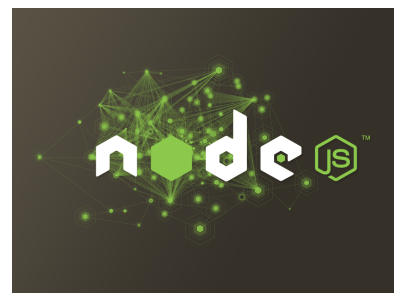


Angular is a structural framework that can be used in the development of web applications, this framework allows for the extension of HTML's syntax. Angular uses data binding and dependency injection with the intention of minimizing the amount of coding needed, this happens in the users browser.

Node.js

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices (Node.js. 2015).

The idea behind node is to enable the rapid building of apps that are lightweight, fast and highly concurrent. It allows developers to work in the same language on the back and front end. Node is different from many web application runtimes in the manner in which it handles concurrency by using an event-driven loop running a single process instead of the traditional threading model (Mobile App Development with Full-Stack JavaScript, 2015).



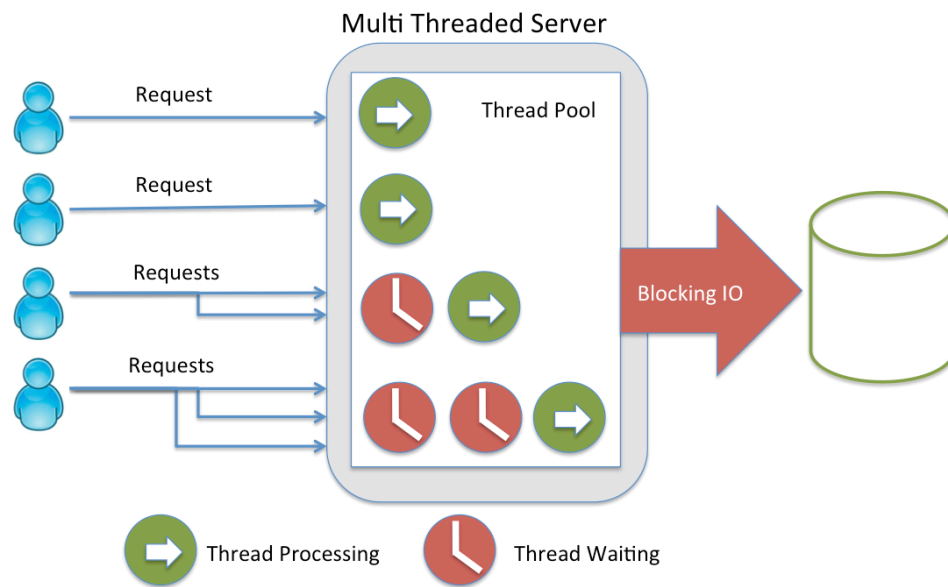


Fig 1. Traditional Multi-Threaded Server

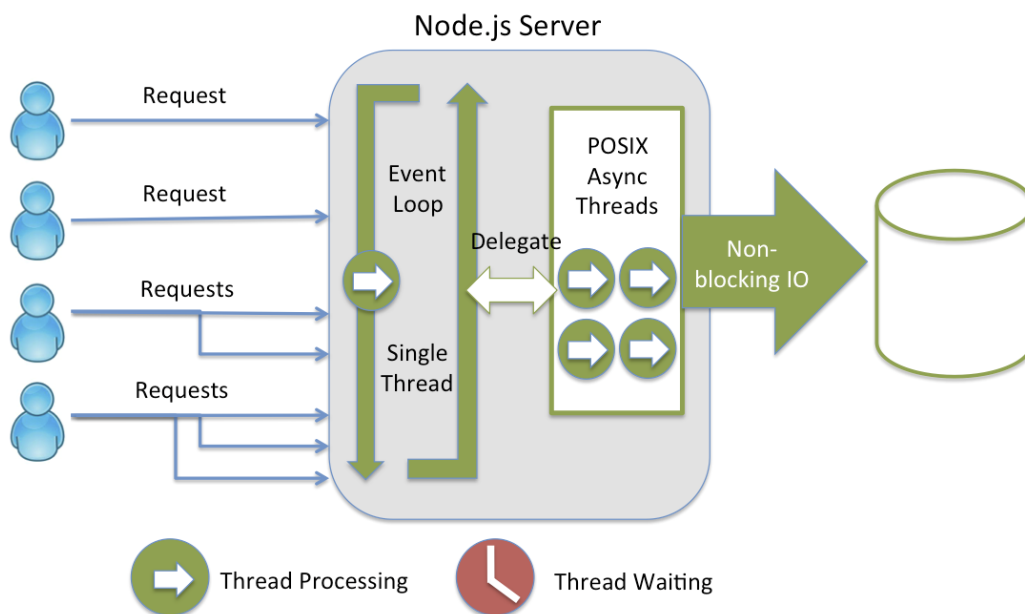


Fig 2. Node.js Server Event Loop

CSS

In regard to CSS, a framework is proposed for use that could increase efficiency when developing the website, there are a number of these frameworks available for use each with their own benefits two of the most popular of these are; twitter Bootstrap and Semantic ui.

Bootstrap



"Twitter Bootstrap is a CSS framework that helps in designing web applications. It is one of the easiest CSS frameworks to use in the Industry today. It assumes you have no designing knowledge and all you want to do is write some HTML as per the Bootstrap specifications. In short, Twitter Bootstrap has already written a CSS style

sheet for you, has inbuilt jQuery support and also has some popular JavaScript tools" (Twitter Bootstrap Tutorial. 2015).

Semantic UI

"Semantic empowers designers and developers by creating a language for sharing UI. Lose the Hieroglyphics: Semantic is structured around natural language conventions to make development more intuitive. Have a conversation with your components: Semantic gives you a variety of UI components with real-time debug output, letting your code tell you what it's doing (Semantic UI - CSS Framework. 2015)."



Bootstrap CSS Framework

The choice was made to use Twitter Bootstrap for a number of reasons the most important of these being the designers familiarity with the framework and the versatility it provides when developing the look of the website. Ideally the author would like to develop a personalised ui for the website, however as there are time constraints on the project a bootstrap template may be used and personalised when creating the ui.

Git

The entire project along with this document will be hosted on a git repo located on github.com. git repos will be used as a means of version control in addition to an alternative means of backing up data in case of data loss from the primary development environment.

Testing framework

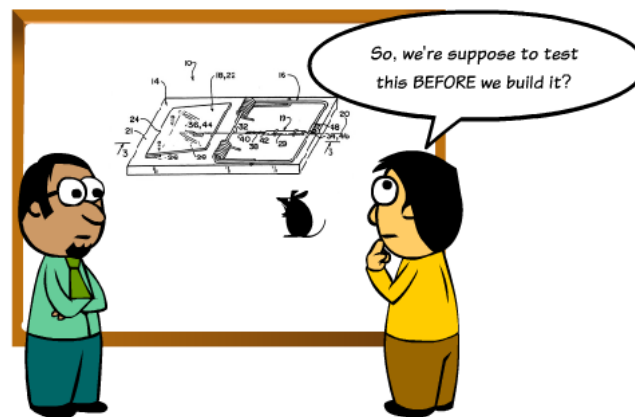


Fig. Testing

The initial starting point of the project was to use a java based environment for development therefore a java based testing methodology would have been implemented, while not being utilised in this project it was important to get a grounding of how testing is performed. Two of the most popular methods when testing java are:

junit

"Read almost any software developer journal or website and we're told that responsible developers write test cases. If those developers are Java developers then, most likely, they use JUnit for those test cases. JUnit is probably the oldest and certainly the most popular Java-based testing framework around. Created by Erich Gamma and Kent Beck, JUnit has become the de facto standard for unit testing. In the interim, however, other frameworks have been built to address various faults and deficiencies with JUnit .(Test Framework Comparison. 2015)"

testNG

"TestNG is a testing framework written by Java and inspired from JUnit and NUnit, it is not only inherited existing functionality from Junit as well as introducing some new innovative functionality that make it powerful, easy to use, reliable, maintainable and testable codes. TestNG is designed to cover all categories of tests: unit, functional, end-to-end, integration, etc. we can take full advantage of TestNG from engineering to quality insurance (TestNG Tutorials. 2015).

JavaScript Testing

The decision was made to develop the project using JavaScript instead of the traditional java based approach, due to this change the available testing methodologies would also have to change.

Protractor

Protractor is an end-to-end testing framework originally created for user with angular.js, protractor is a Node.js application that supports a number of testing libraries such as Jasmine, Mocha, Cucumber and Qunit. Protractor acts as a wrapper numerous testing methodologies utilising the WebDriverJS-API, used in conjunction with a Selenium Server in order to translate user interactions into browser understandable instructions (End-to-End Testing with Aurelia and Protractor, 2015).

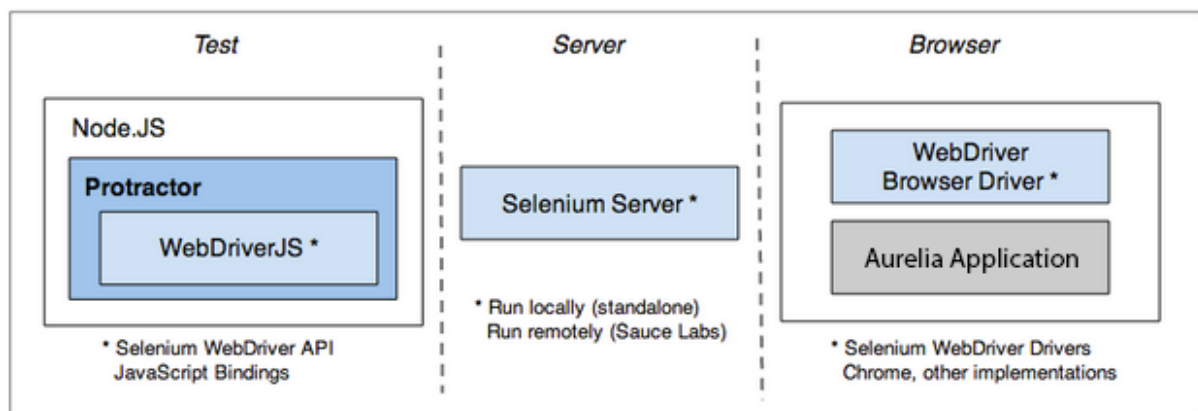


Fig 4. Protractor

Jasmine

Jasmine is an opinionated Behaviour Driven Development (BDD) framework, it utilises the same technology as Mocha along with much of the same methods. Some of the methods used by Jasmine include: describe(), it(), beforeEach() and afterEach(). These methods are shared with Mocha however there is an additional method utilised in jasmine called expect (). These ascertain methods are called matchers, which implement a Boolean comparison function between the object being tested and the expected outcome. *"The Protractor framework integrated with Jasmine can be used to create and organize tests and user expectations. Jasmine is compatible with Protractor due to which all resources that are extracted from browsers can be used to make tests as promises. Those promises are resolved internally by using the "expect" command from Jasmine. That way the promises work smoothly while creating tests"* (Testing AngularJS apps with Protractor I ThoughtWorks. 2015).

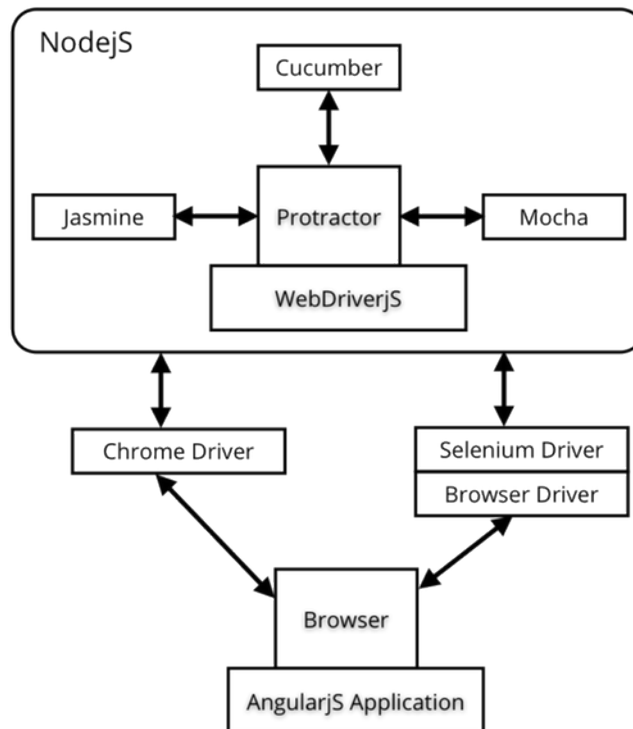


Fig 5. Protractor Runner and Testing Frameworks

Qunit

Qunit is a JavaScript testing framework written by members of the jQuery team. Qunit can be used when testing regular JavaScript code locally or server side. Qunit was released in 2008 and is the oldest JavaScript testing framework on this list.

Mocha

Mocha is the most commonly used testing framework on most Node applications. This testing framework is compatible with angular tools. The framework supports both Behaviour Driven Development (BDD) and Test Driven Development (TDD) unit tests and uses node to run both synchronous and asynchronous tests. "Mocha comes packed with a set of different reporters to present the test results and includes many features, such as pending tests, excluding tests, and skipping tests. The main interaction with Mocha is done using the command-line tool provided, which lets you configure the way tests are executed and reported".

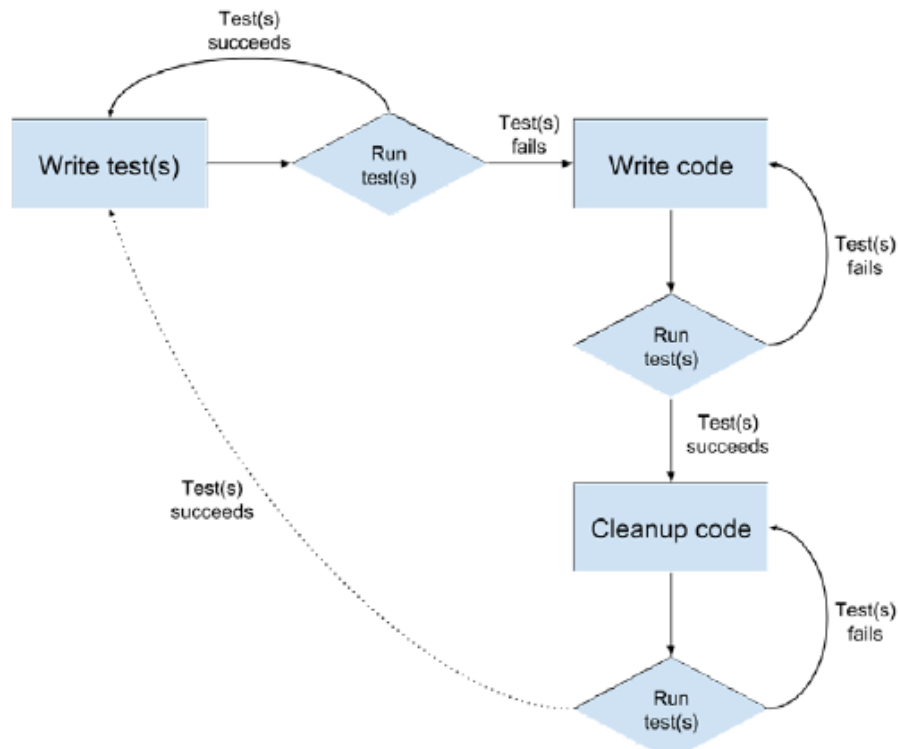


Fig 6. Test Driven Development (TDD)

Testing framework

The options above are only some of the available end-to-end testing methods available for use in this project, each with their very own unique advantages over the other. In addition to end-to-end testing the Node and Angular components will be tested for functionality also.

The decision was made to perform end-to-end testing using Protractor and Mocha; this decision was largely due to the popularity of the programs and the availability of helpful web resources, which have aided in the development of a testing apparatus for the project. Additionally Mocha has been used to test the Node server on a standalone basis independent of protractor.

Karma has been used when testing the angular.js component of the project. Karma is a test runner designed for unit testing that is capable of being run on multiple browsers. This testing methodology shares similar configurations and testing options as Protractor, essentially because the same team who developed Protractor developed it.

Methodology

There are a number of methodologies used in development, these methodologies can be transplanted from a number of fields and vary in complexity depending on the project and the size of the development team. In the inception phase of the project a number design methodologies were assessed to identify the one of most use when developing this project.

Initial Starting Point and Research

Waterfall

The Waterfall design methodology is the classical linear approach used in a number of fields. In this method a stage-by-stage approach is used where each major stage of the project is mapped out in advance typically as illustrated in the figure below (Agile Introduction, 2015).

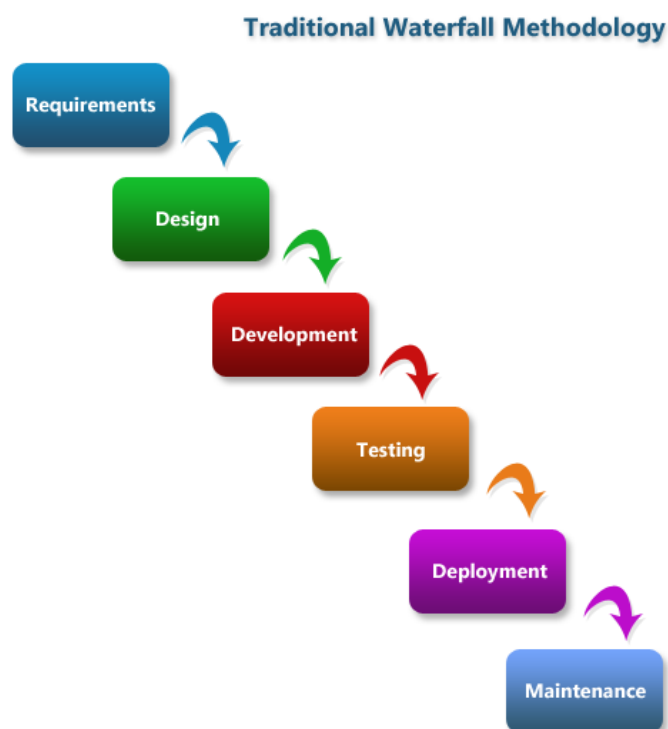


Fig 7. Waterfall Approach

Agile

Agile methods are used to develop software in short timeframes. In this method each iteration of the overall project is treated as a mini project in its own right. "Agile methods emphasize real-time communication, preferably face-to-face, over written documents (Software Development Methodologies. 2015)". The agile method prioritises working software as the major key to progressing each project with deliverables tending to be working code rather than documentation methods.

Scrum

"Scrum is an agile method for project management developed by Ken Schwaber. Its goal is to dramatically improve productivity in teams previously paralyzed by heavier, process-laden methodologies. Scrum enables the creation of self-organizing teams by encouraging verbal communication across all team members and across all disciplines that are involved in the project. A key principle of scrum is its recognition that fundamentally empirical challenges cannot be addressed successfully in a traditional "process control" manner. As such, scrum adopts an empirical approach - accepting that the problem cannot be fully understood or defined, focusing instead on maximizing the team's ability to respond in an agile manner to emerging challenges (Software Development Methodologies. 2015)".

Development Approach

A single developer rather than a development team will develop this project; it will be developed from a unique idea with its core functionality independent from the work environment. Additional development of communication features may be created stemming from experience from work placement.

Due to the nature of the project being developed primarily by a single individual, the developer does not need to be limited by any single design methodology but can choose aspects of each which will further the project as a whole.

At the outset of the project the core functionality was outlined, these consist of the main deliverables for the project i.e. a user login system, posting of site data, security features, etc. These are the main features, which the author believed could be implemented with certainty within the

given timeframe. A second set of features in addition to the core functionality were also outlined, these were the like to have features that would add to the project but were not essential to development, these features included: a web chat function, using docker with the project and hosting the project online.

Developers Tools

The proposed development frameworks have been chosen during the inception phase of the project placing emphasis on the developers' familiarity with each framework and the suitability of the framework for the project.

As mentioned previously the app will utilise the MEAN stack technologies when developing the project along with additional features added through the node project management console (npm)

Prototypes

Initial Java Approach

A prototype of the website was first developed along with a MySQL database to allow for storage of data. The website was built using the Play 1.2.7 framework with HTML 5 and bootstrap being used for the user interface. To test the website a junit testing app was developed to test for functionality and ascertain if data is being correctly saved in the database.

If the project were to continue down this path the next steps would have been to develop an android application for use in the field to record and transmit data to the database via a RESTful service.

Alternative JavaScript Approach

After completion of the prototype in java a second prototype was developed using JavaScript specifically Node, Angular, mongo as a non-relational database and the express framework. This prototype was constructed using only sublime text and a command line interface.

Final Approach

The decision was made to use JavaScript to design the web app. Factoring into this decision was opportunity for the author to gain insight into a new design approach and to learn information regarding the JavaScript development environment. The core functionality and deliverables in the project remained the same, but have had to be completed in a smaller timeframe.

Risk Management

Requirements

As this is a personal project independent from the working environment it will be up to the designer to dictate the overall look and functionality of the project in the given timeframe. The core deliverables can be met from the developers previous experience with the technologies and tools chosen to develop the project, the deciding factor will be time related, i.e. if the project can be completed by the May deadline.

Skills

The skill required to develop this project have been outlined previously; core functionality will be based around the MEAN stack, a new technology used by the developer. Due to the fact that this is a new technology and before committing completely to this method of development a demo application was created as a proof on concept, that would go on to become the basis for the project.

Risks

The project itself is quite database intensive with over one hundred individual fields being used to convey data for site characterisation. Implementing these fields into a web application can be quite time consuming given the limited timeframe of the project.

The MEAN stack method of development is new to the designer who does not have grounding in JavaScript languages, this may lead to time delays as there will need to be a transition time to the new programming languages

There also may need to be a refactoring of the project during the development timeline if unforeseen circumstances arise that could require the learning of new technologies instead of using those initially chosen during the inception of the project.

Management of the Projects Key Technologies

The creation of the web app project using the MEAN stack was a new experience to the developer. To learn these new technologies research had to be undertaken using dedicated textbooks and websites. At the outset of the project the initial approach was to use a java based environment incorporating a MySQL database for the storage of data and a HTML templating engine with styling using css.

Nearly one third of the way through, after a demo was created using java it was decided to change from this java based environment to a JavaScript based one. “Thanks to the available books and online resources I decided to use the MEAN stack to develop the project”.

The MEAN stack stands for MongoDB, Express, AngularJS and nodeJS, the breakdown of these technologies and how they were implemented are as follows:

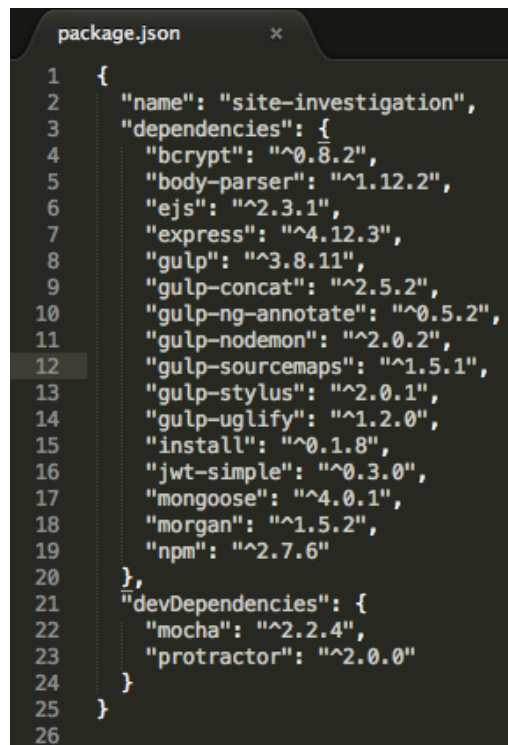
NodeJS

NodeJS is a JavaScript routine run outside of the browser. In this project node has been used as the projects webserver. The technology and terminology used in the node framework has been documented in the technology chapter.

This project was coded in an osx environment. In this environment, node was installed via the brew command: `brew install npm`. Brew itself is an open source environment installed on osx to allow for easy installation of software via the brew install command.

NPM

Node has the capability to install external dependencies via the NPM command. By using this command external modules required for this project can be downloaded and installed to a `node_modules` folder within the project.

A screenshot of a code editor showing a file named 'package.json'. The file contains a JSON object with a 'name' property set to 'site-investigation', a 'dependencies' object listing various npm packages with their version constraints, and a 'devDependencies' object listing development tools. The packages listed include bcrypt, body-parser, ejs, express, gulp, gulp-concat, gulp-ng-annotate, gulp-nodemon, gulp-sourcemaps, gulp-stylus, gulp-uglify, install, jwt-simple, mongoose, morgan, npm, mocha, and protractor. The code is syntax-highlighted, and line numbers 1 through 26 are visible on the left side of the editor.

```
1 {
2   "name": "site-investigation",
3   "dependencies": {
4     "bcrypt": "^0.8.2",
5     "body-parser": "^1.12.2",
6     "ejs": "^2.3.1",
7     "express": "^4.12.3",
8     "gulp": "^3.8.11",
9     "gulp-concat": "^2.5.2",
10    "gulp-ng-annotate": "^0.5.2",
11    "gulp-nodemon": "^2.0.2",
12    "gulp-sourcemaps": "^1.5.1",
13    "gulp-stylus": "^2.0.1",
14    "gulp-uglify": "^1.2.0",
15    "install": "^0.1.8",
16    "jwt-simple": "^0.3.0",
17    "mongoose": "^4.0.1",
18    "morgan": "^1.5.2",
19    "npm": "^2.7.6"
20  },
21  "devDependencies": {
22    "mocha": "^2.2.4",
23    "protractor": "^2.0.0"
24  }
25 }
26
```

Fig 8. package.json file

These dependencies are kept track of in a package.json file. This file keeps a list of the dependencies and their version number; this version number will limit the dependency to that version which is confirmed working. Without this compatibility errors may arise from newer versions of the software.

Express

Express was installed via the npm package manager. Its use in this project was to build the web application with node

MongoDB

MongoDB is a common noSQL database used when developing JavaScript based projects that require a database. MongoDB is a good fit with nodeJS as they both support JSON as a language. During research of the mean stack MongoDB was found to be the most common database used, but not the only database available. In addition to MongoDB other noSQL databases compatible with JSON includes Cassandra and RethinkDB. Mongo was chosen due to its popularity and the availability of helpful documentation online.

MongoDB is a noSQL database, this was the authors' first project using this type of database, and as such the data saved to the database has been limited to user and post data.

A screenshot of a code editor showing the 'user.js' file. The code defines a Mongoose schema for a user with fields 'username' and 'password'.

```
1 | var db = require('../db')
2 | var user = db.Schema({
3 |   username: { type: String, required: true },
4 |   password: { type: String, required: true, select: false }
5 | })
6 | module.exports = db.model('User', user)
```

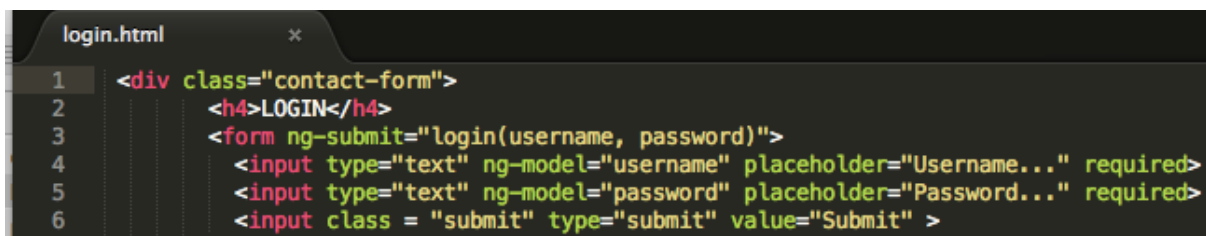
Fig 9. User Model

Mongo stores data in a BSON format, which is a form of JSON in binary form. Mongo was installed onto the system using the brew command: `brew install mongodb` and is run in the command line with: `sudo mongod`.

NOTE: If the node server is run without the mongo database running then no data will be saved or presented on the web app. This occurs when running under localhost.

AngularJS

Angular has been used in this project to represent data from the database to elements within the HTML page. Angular data is declared via the "ng" element within a HTML tag.

A screenshot of a code editor showing the 'login.html' file. The code defines an AngularJS form with two text inputs for 'username' and 'password', and a submit button.

```
1 | <div class="contact-form">
2 |   <h4>LOGIN</h4>
3 |   <form ng-submit="login(username, password)">
4 |     <input type="text" ng-model="username" placeholder="Username..." required>
5 |     <input type="text" ng-model="password" placeholder="Password..." required>
6 |     <input class = "submit" type="submit" value="Submit" >
7 |   </form>
```

Fig 10. Angular ng-submit

In the case of logging in as the image above for data is an angular element as are the username and password

Initial Starting Point

The initial starting point of the project was to create a webpage with data hardcoded and viewed independently of a database using HTML and Angular. Inputs fields were created to represent data entry, this data was finite and would be lost when the webpage was closed.

Building an API

An API was required to get a list of posts and make new posts. To achieve this a body-parser was required to help express in reading JSON on post requests. This body parser was installed as an npm package.

To allow the web app to interact with the database a package called mongoose was required. This dependency was again installed via the npm install command. Mongoose was used to translate data in the database to JavaScript objects for use in the web app. With a connection now in place a mongoose model was created representing user data and post data within a models folder.

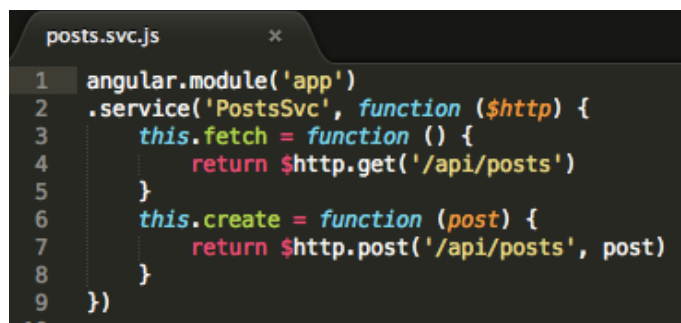


```
server.js
1  var express = require('express')
2  var bodyParser = require('body-parser')
3  var logger = require('morgan')
4
5  var static = require('node-static');
6  //var http = require('http');
7  var file = new(static.Server)();
8  //var app = http.createServer(function (req, res) {
9  |   //file.serve(req, res);
10 | }).listen(2013);
11
12  var app = express()
13  app.use(bodyParser.json())
14  app.use(logger('dev'))
15  app.use(require('./controllers'))
16  //app.use('/api/posts', require('./controllers/api/posts'))
17  //app.use(require('./controllers/static'))
18
19
20  var server = app.listen(3000, function () {
21  |   console.log('server listening on %d', server.address().port)
22  | })
23
24  /*var port = process.env.PORT || 3000
25  var server = app.listen(port, function () {
26  |   console.log('Server', process.pid, 'listening on', port)
27  | })*/*
```

Fig 11. Server.js

Integrating Angular

Angular uses a built in HTTP client \$HTTP to perform HTTP calls, this is the standard http client performing both get and post requests as \$http.get() and \$http.post(). Angular elements within the code can be identified with a “\$” prefix.



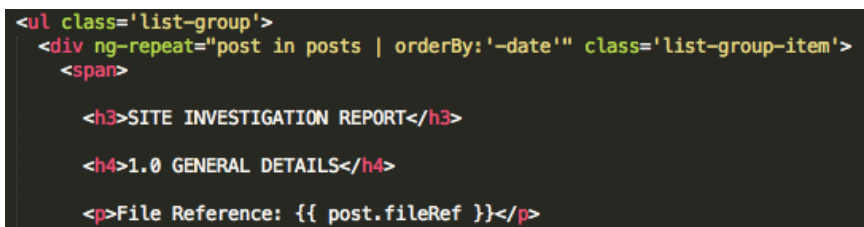
```
posts.svc.js
1 angular.module('app')
2 .service('PostsSvc', function ($http) {
3   this.fetch = function () {
4     return $http.get('/api/posts')
5   }
6   this.create = function (post) {
7     return $http.post('/api/posts', post)
8   }
9 })
```

Fig 12. posts.svc.js

Representing Post Data

At this stage of the project the majority of the coding base had been implemented. The model, view, controller schema had been implemented. How data saves to the database was represented on the webpage could now be examined.

Through angulars ng-repeat feature, post content from the database was represented on the webpage in a list. The issue when implementing data in this way is that the oldest post remains at the top of the webpage, with any new data entered represented at the bottom of the webpage, this is not ideal as the user would have to scroll down to the bottom of the page to view new data.



```
<ul class='list-group'>
  <div ng-repeat='post in posts | orderBy: '-date'' class='list-group-item'>
    <span>

    <h3>SITE INVESTIGATION REPORT</h3>

    <h4>1.0 GENERAL DETAILS</h4>

    <p>File Reference: {{ post.fileRef }}</p>
  </div>
</ul>
```

Fig 13. ng-repeat with orderBy

Two options were available to sort data from newest to oldest. The first was to use angulars orderBy feature within the ng-repeat line of the HTML code. Alternatively this can be achieved by

adding the line `.sort('-date')` into the `post.find()` command in the `api/posts.js` file. The decision was made to use the angular version to sort data, this also raises the option of representing data using various filters.

Automating Node

To enable changing of code within the project on the fly without constantly needing to restart the node server an npm module called nodemon was installed. This module monitors the project for saved changes and automatically restarts the server with these new changes in effect.

Build Automation

It is a common practice for modern JavaScript applications to incorporate some form of build automation into the build process. This build automation would process the various facets of the project and place them within an assets folder. The two most common build automation tools are Grunt and Gulp.

Grunt

Grunt is a task based command line build tool for use in JavaScript projects, in this build tool a JSON file used to specify which tasks are to be run which minimises the JavaScript file and compiles css/sass and places the output file into a folder.

Gulp

Gulp is a stream based automation tool built on nodeJS with tasks written in JavaScript. Gulp contains both a Gulp source and a Gulp file where tasks are defined. Gulp is again installed through the npm command. Gulp generates an `app.js` build file within the assets folder along with css and JavaScript styles. In the development environment Gulp was run with the command: `gulp dev`. This command also acts as a means to monitor changes in the code and restart the node server with these changes in effect.

```

scripts.js
1 |var gulp = require('gulp')
2 |var concat = require('gulp-concat')
3 |var uglify = require('gulp-uglify')
4 |var ngAnnotate = require('gulp-ng-annotate')
5 |var sourcemaps = require('gulp-sourcemaps')
6
7 |gulp.task('js', function () {
8 |    return gulp.src(['ng/module.js', 'ng/**/*.js'])
9 |        .pipe(sourcemaps.init())
10 |        .pipe(ngAnnotate())
11 |        .pipe(concat('app.js'))
12 |        .pipe(uglify())
13 |        .pipe(sourcemaps.write())
14 |        .pipe(gulp.dest('assets'))
15 |})
16
17 |gulp.task('watch:js', ['js'], function () {
18 |    gulp.watch('ng/**/*.js', ['js'])
19 |})

```

Fig 14. scripts.js

The above image is the server.js build file. Folder and files to be monitored for the build are defined as a string in gulp.src. The destination folder where the build will take place has been defined under gulp.dest as the assets folder. The gulp.watch element monitors the selected folders for changes, if changes have been made then gulp will rebuild the assets folder automatically.

Adding Security to the Site

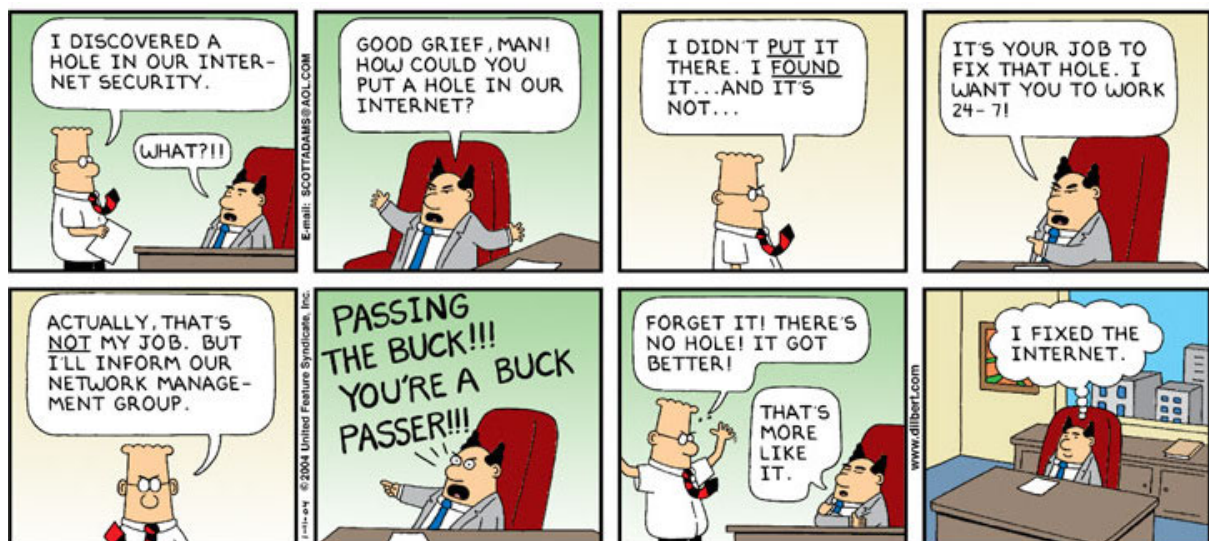


Fig 15. Fixing the Internet

As with any web app that requires input of user data, the security of this data is important. Two methods of authentication were investigated these were:

Cookie Based Authentication

This is the traditional means of authentication on web applications. The session cookie is created and set on the login page and used when navigating the site. The process of cookie authentication involves the following:

- The user provides a username and password on the login page
- When the request is made, the user is validated on the backend by querying the database. If the request is valid then a session is created.
- This session information is provided when accessing restricted portions of the web app

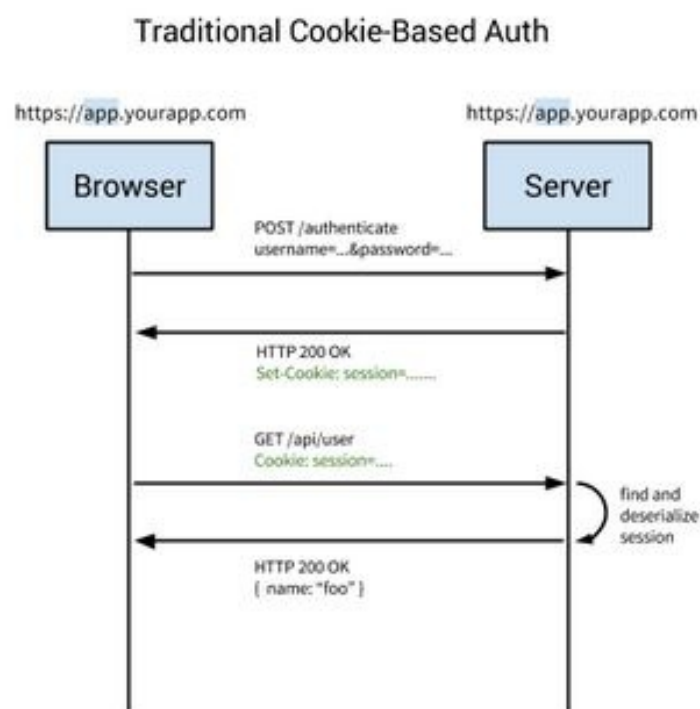


Fig 16. Traditional Cookie-Based Authentication

Token-Based Authentication

Token-based authentication does not use cookies or sessions; instead tokens are used for authenticating the user from each request on the server. Token authentication works as follows:

- The user provides login details.
- The user is validated on the backend by querying the database; if valid a token is created and returned to the user in the response header, this token is stored in local storage.
- Token information is provided upon every request header when accessing restricted areas of the site.

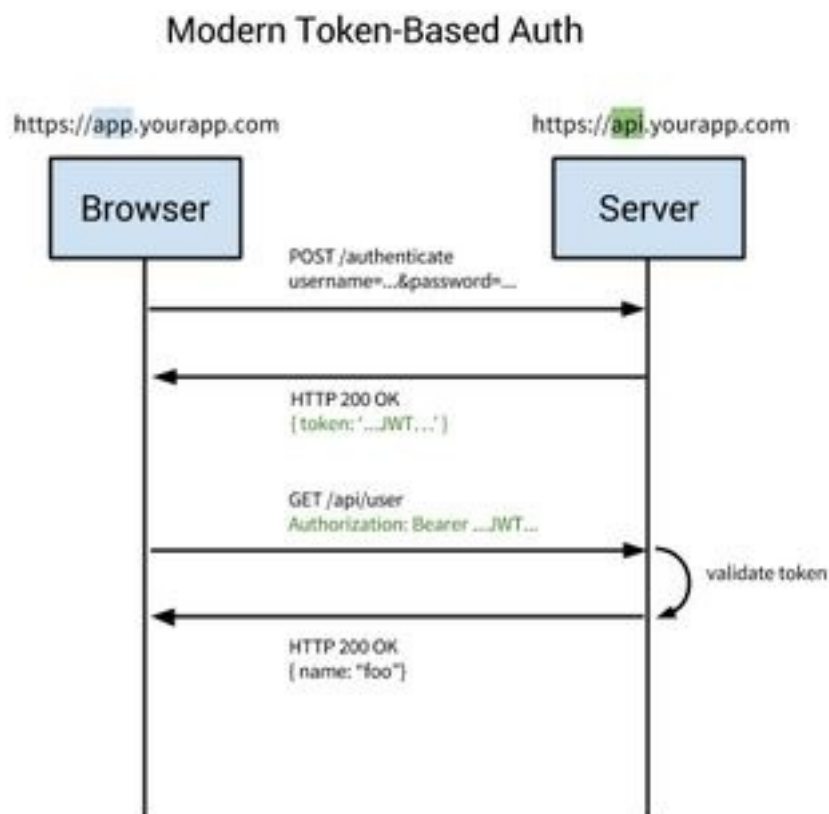


Fig 17. Token Based Authentication

Tokens were chosen for use in this project in the form of Jason Web Tokens (JWT). JWT is a “bearer token which uses a token consisting of three parts: a header, payload and signature.

- The header keeps the token type and encryption method and is encrypted with base64 encryption.

- The payload contains data, in this case user information and is again with base 64 encryption.
- The signature consists of a combination of the header payload and secret key. This secret key can be a passphrase and is kept secretly on the server side.

JWT was installed in the project under the npm command.

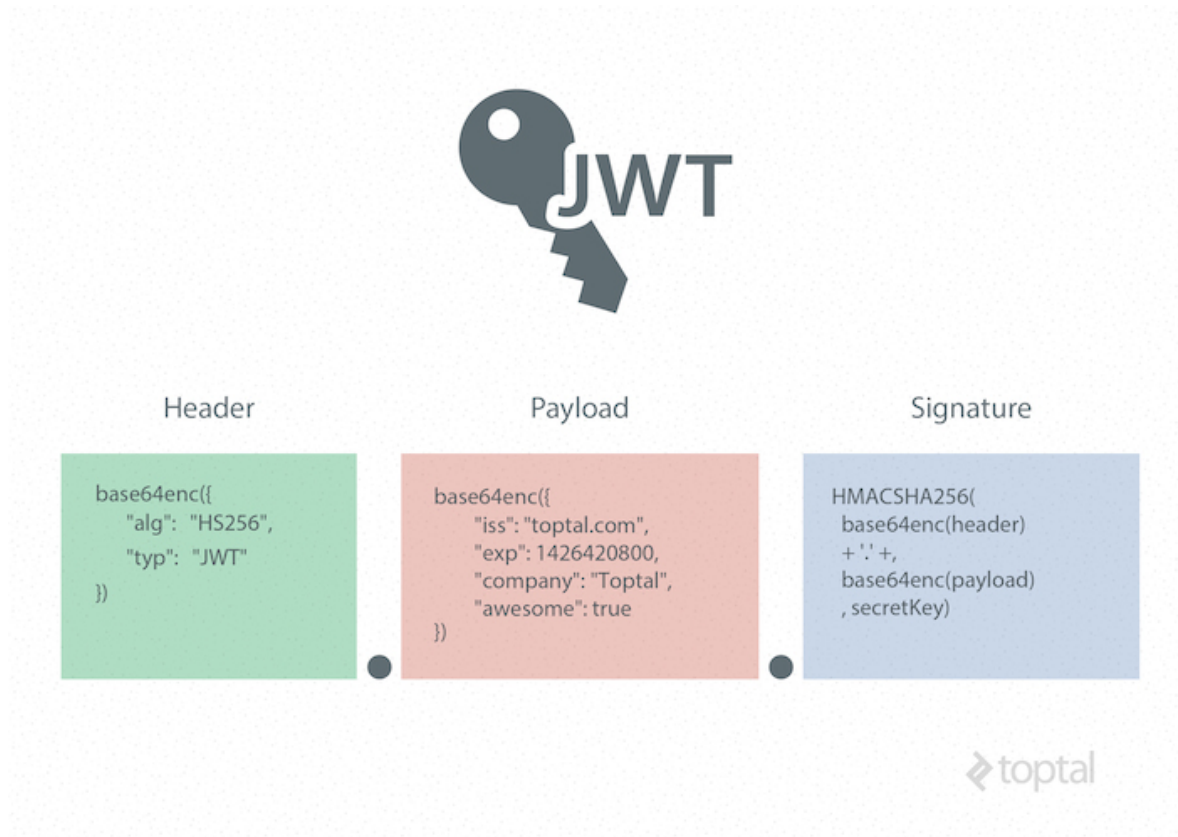


Fig 18. JSON Web Token

When incorporated together angular will request JWT through POST api/sessions. In order to create a JWT private key a new JavaScript document was created named config.js, as per the image below.

```
config.js
1 module.exports = {
2   secret: 'supersecretkey'
3 }
4
```

Fig 19. Config.js file containing the secret key

The config.js file containing the secret key is what makes JWT secure, without this secret key anyone who has a web token can read the data without the need to make a HTTP call.

Bcrypt

Bcrypt was used to encrypt sensitive user password data; it is a hashing algorithm that stores data within the database in a hashed form. When implementing Bcrypt into the project the npm command was used to install the code into the node_modules folder. When implementing Bcrypt into the project a number of passes was set when creating a hashed password. These passes dictate the amount of time the algorithm is used when hashing the password. The data saved to the database does not match input for input i.e. two users with the same password will not have the same hash stored in the database.

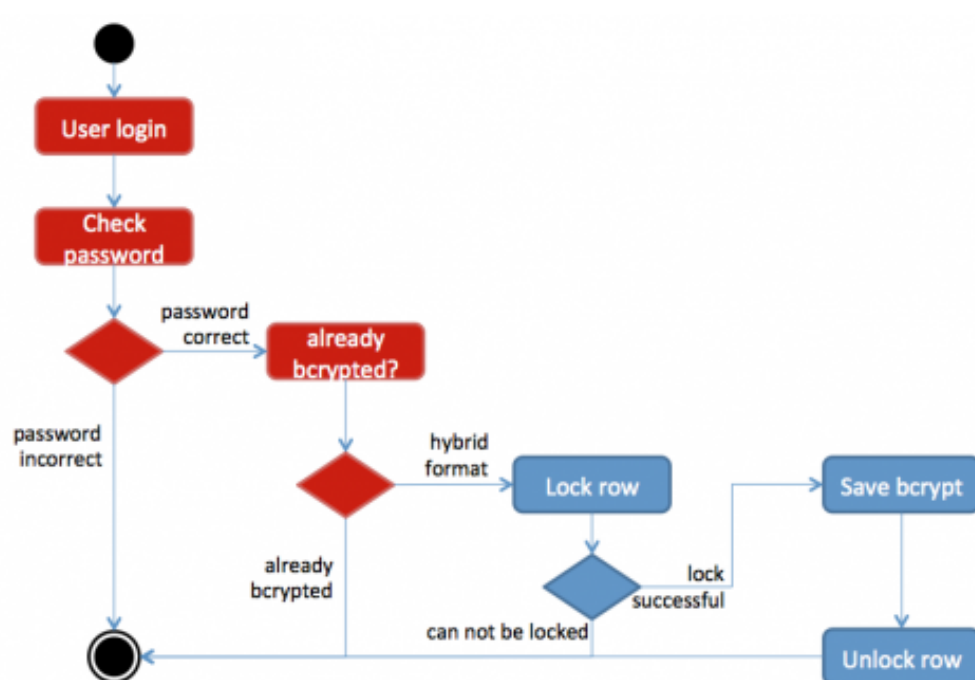
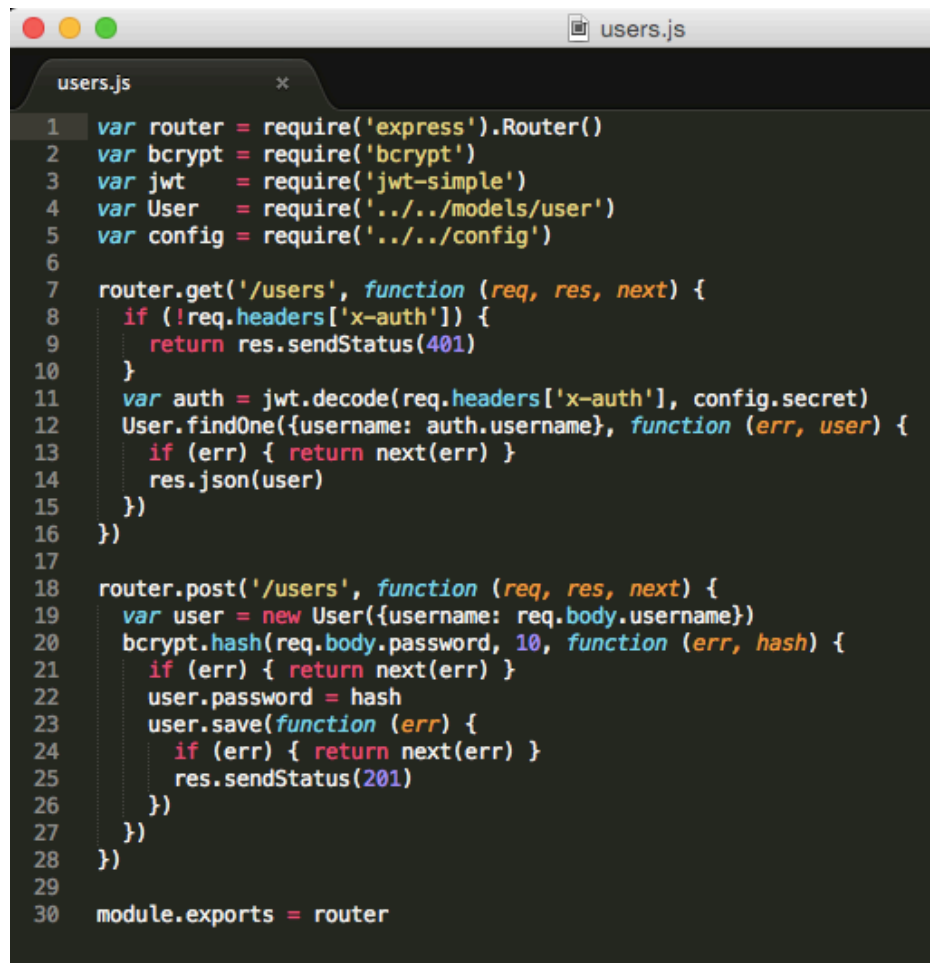


Fig 20. Bcrypt

The implementation of Bcrypt into the project was a relatively simple process, again the technology was installed via npm and referenced in the code.



```

1 var router = require('express').Router()
2 var bcrypt = require('bcrypt')
3 var jwt = require('jwt-simple')
4 var User = require('../models/user')
5 var config = require('../config')
6
7 router.get('/users', function (req, res, next) {
8   if (!req.headers['x-auth']) {
9     return res.sendStatus(401)
10  }
11  var auth = jwt.decode(req.headers['x-auth'], config.secret)
12  User.findOne({username: auth.username}, function (err, user) {
13    if (err) { return next(err) }
14    res.json(user)
15  })
16 })
17
18 router.post('/users', function (req, res, next) {
19   var user = new User({username: req.body.username})
20   bcrypt.hash(req.body.password, 10, function (err, hash) {
21     if (err) { return next(err) }
22     user.password = hash
23     user.save(function (err) {
24       if (err) { return next(err) }
25       res.sendStatus(201)
26     })
27   })
28 })
29
30 module.exports = router

```

Fig 21. user.js

The image above illustrates the user.js code located in the projects controllers' directory. Bcrypt is used in this situation to hash the users password, this is implemented through line 20 of Fig. 21 where the password is hashed, the 10 represented in this line of code represents the amount of times the password is hashed before saving. If the password has been successfully saved the node server will return a HTTP response 201 data created.

Authentication

At this stage of the project, anyone could post data to the project without the need for being logged in, while this is an open repository for data it would not be proper to let anyone posts random data without being logged in first. To solve this, posts authentication was added to the posts.js model, now if a user is not logged in then data will not be saved to the database.

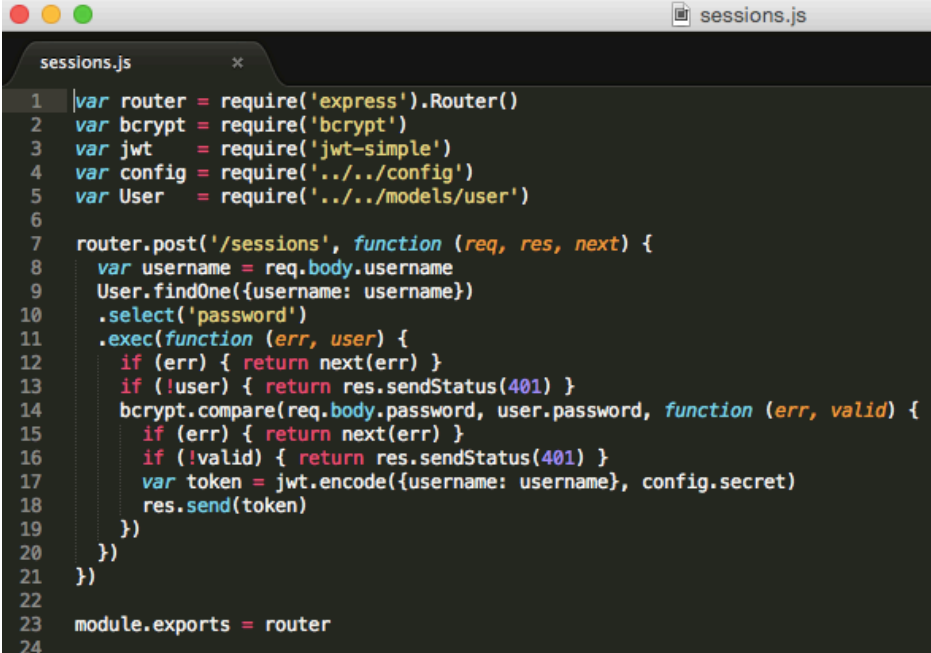
```

158     post.username = req.auth.username
159     post.save(function (err, post) {
160         if (err) { return next(err) }
161         res.status(201).json(post)
162     })
163 })
164
165 module.exports = router
166

```

Fig 22. Authorising Posts by User

Sessions are created to allow users to navigate the site without constantly needing to login to submit data. These sessions were created in api/sessions.js, where if user creation or login data is correct then a JWT token is sent to the server to create a session.



```

sessions.js
1  var router = require('express').Router()
2  var bcrypt = require('bcrypt')
3  var jwt = require('jwt-simple')
4  var config = require('../config')
5  var User = require('../models/user')
6
7  router.post('/sessions', function (req, res, next) {
8      var username = req.body.username
9      User.findOne({username: username})
10     .select('password')
11     .exec(function (err, user) {
12         if (err) { return next(err) }
13         if (!user) { return res.sendStatus(401) }
14         bcrypt.compare(req.body.password, user.password, function (err, valid) {
15             if (err) { return next(err) }
16             if (!valid) { return res.sendStatus(401) }
17             var token = jwt.encode({username: username}, config.secret)
18             res.send(token)
19         })
20     })
21 })
22
23 module.exports = router
24

```

Fig 23. session.js

Routing

To turn the project into a single page application AngularJS router module was required. By adding this router it has allowed angular to use the ng-route statement within the code. In the views/app.html.ejs routing has been enabled with the following links.

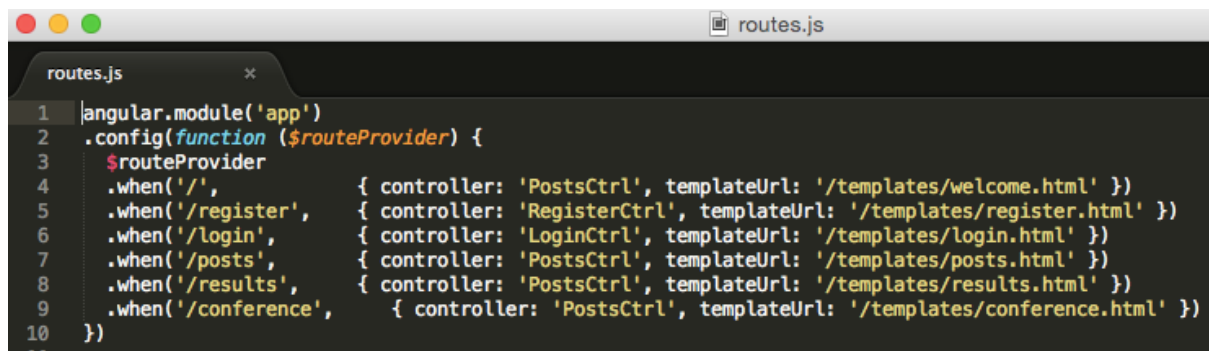
```

150 <div class='container' ng-view></div>
151 <script src='http://ajax.googleapis.com/ajax/libs/angularjs/1.3.0-rc.
152 <script src='http://ajax.googleapis.com/ajax/libs/angularjs/1.3.0-rc.
153 <script src='/app.js'></script>
154 </body>
155 </html>
156

```

Fig 24. ng-route in HTML

To enable routing to work the ngRoute had to be declared as a dependency in ng/module.js. The routes could now be defined within ng/routes.js. Each line represents a route to a page within the web app, with all routes associated with a controller and a template as per figure xxx, when navigating the site using this method a # precedes each link.



```

routes.js
1 angular.module('app')
2 .config(function ($routeProvider) {
3   $routeProvider
4     .when('/', { controller: 'PostsCtrl', templateUrl: '/templates/welcome.html' })
5     .when('/register', { controller: 'RegisterCtrl', templateUrl: '/templates/register.html' })
6     .when('/login', { controller: 'LoginCtrl', templateUrl: '/templates/login.html' })
7     .when('/posts', { controller: 'PostsCtrl', templateUrl: '/templates/posts.html' })
8     .when('/results', { controller: 'PostsCtrl', templateUrl: '/templates/results.html' })
9     .when('/conference', { controller: 'PostsCtrl', templateUrl: '/templates/conference.html' })
10  })

```

Fig 25. routes.js

webRTC

The project features a webRTC chat room featuring video chat for two or more users to interact with each other. “WebRTC is a free, open project that provides browsers and mobile applications with Real-Time Communications (RTC) capabilities via simple APIs. The WebRTC components have been optimized to best serve this purpose” (WebRTC. 2015). The main intention of incorporating webRTC into this project was to facilitate video calls from a user in the field to a user in the office, in this way on site conditions can be communicated from the field.

STUN is a Simple Transversal of User Datagram Protocol (UDP) through Network Address Translators (NATs). The technology helps map a local computers ip port to the public ip port. The user sends a UDP packet out to the STUN server and the server responds with the ip port connected (STUN Client – CodeProject, 2015).

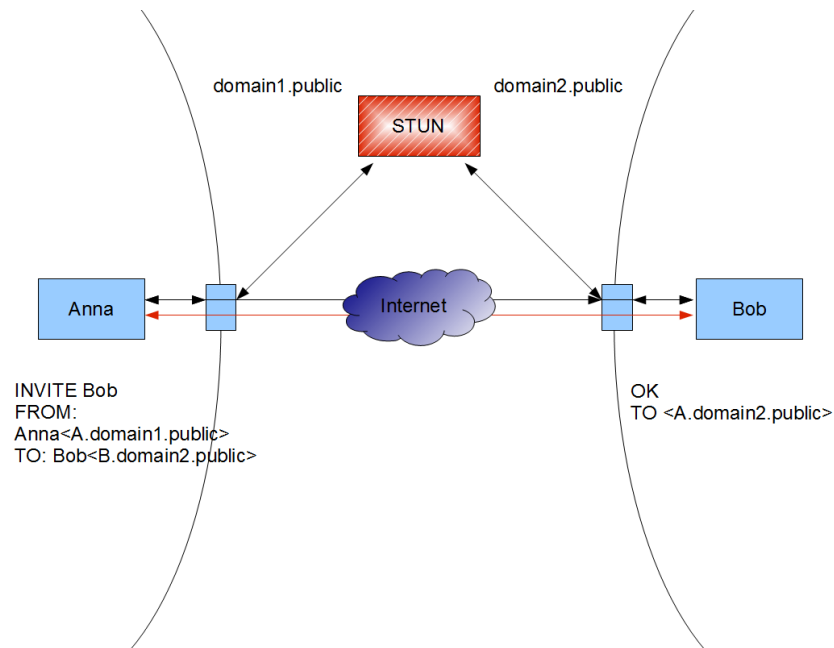


Fig 26. Stun Server Example

webRTC has been incorporated into this project using Google's stun server (stun.1.google.com:19302) which is intended for public use. Coding on incorporating webRTC into the project was achieved through HTML and minimal scripting.

```

5 <body>
6 <br>
7 <div "NOTE: the WebRTC video chat function will only work on
  chrome and firefox browsers"></div>
8
9 <div class="col-md-5 about-info-left">
10 <h4>Remote Video</h4>
11 <div id="remoteVideo"></div>
12 </div>
13 <div class="col-md-7 about-info-right">
14 <h4>Local Video</h4>
15 <div id="localVideo"></div>
16 </div>
17 <script>
18 var webrtc = new SimpleWebRTC({
19   localVideoEl: 'localVideo',
20   remoteVideosEl: 'remoteVideo',
21   autoRequestMedia: true
22 });
23 webrtc.on('readyToCall', function () {
24   webrtc.joinRoom('site-investigation-test123');
25 });
26 </script>

```

Fig 27. webRTC Implementation

Building the UI

During the development of the app, time constraints made it apparent that completely designing the webapps ui would not be possible. A bootstrap template was used in the design of the ui under a creative commons licence from w3c layouts. This layout was chosen due to its visual styling and its compatibility with various screen sizes. This template was heavily modified to serve for use in the webapps.

When accessing the site the user is taken to the welcome page, this page contains links to various sections of the site. The user has multiple means to navigate the site; the animated tab provides links to all pages in the web app as does the quick navigation buttons located below under the quick links section. Unique to the welcome page are descriptions to the main features of the web app, these are presented in text boxes with images and are again clickable to navigate to that section of the site. The navigation bar and quick links at the top of the page are inserted as sections and will appear this way on every page of the site, this was intended to reduce working time and eliminate duplication of data.

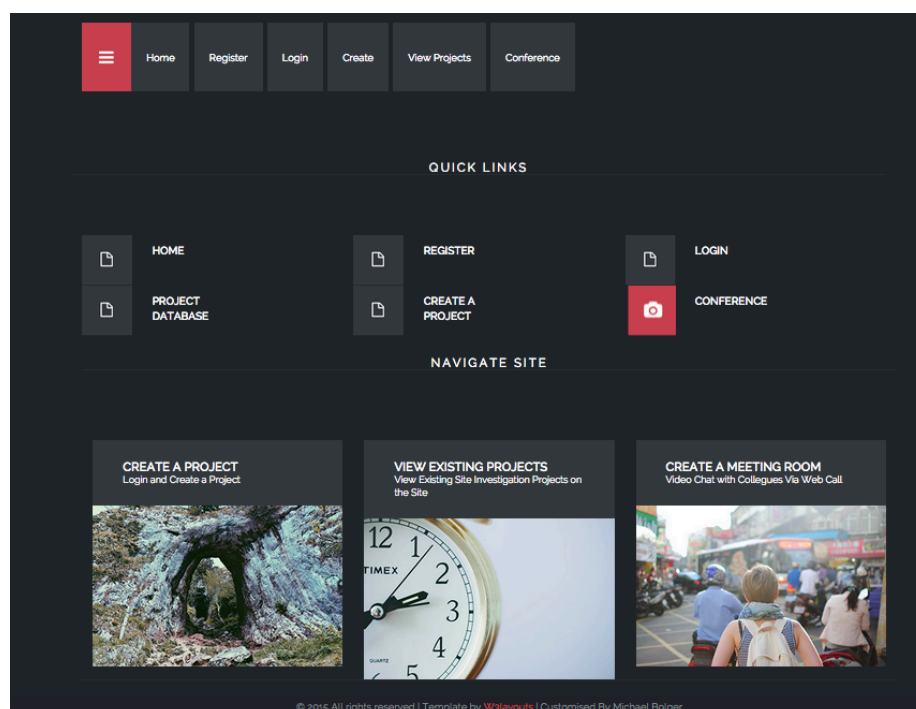
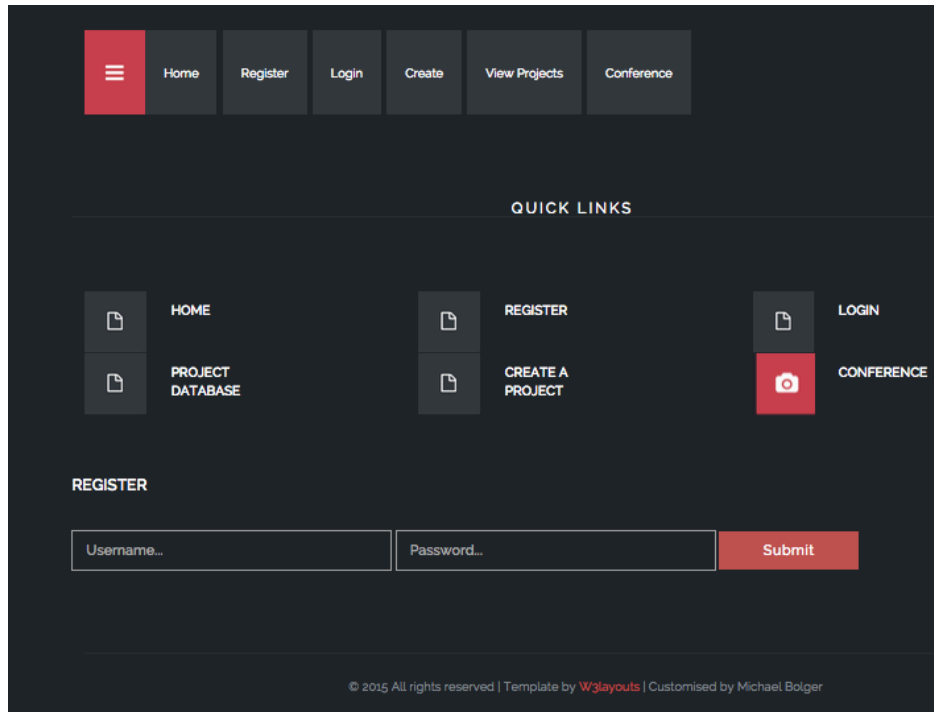


Fig 28. Home Page

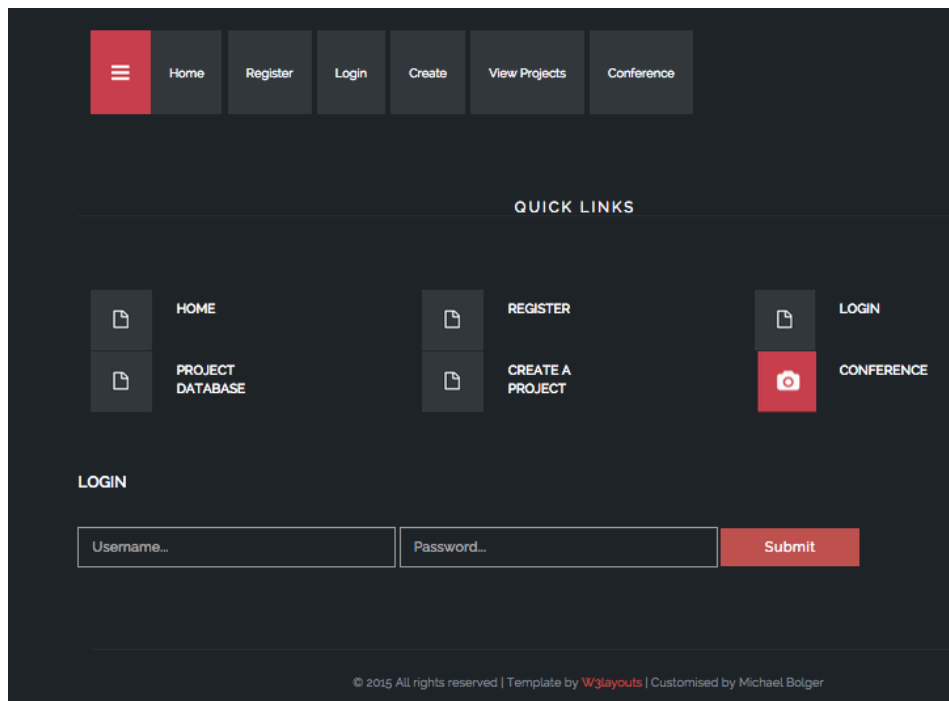
The Registration page is a very simple registration where the potential user enters a username and password; the password is hidden during input.



The registration page features a dark theme with a top navigation bar containing a red menu icon and links for Home, Register, Login, Create, View Projects, and Conference. Below this is a 'QUICK LINKS' section with a grid of buttons: HOME, PROJECT DATABASE, REGISTER, CREATE A PROJECT, LOGIN, and CONFERENCE (which has a red camera icon). The main content area is titled 'REGISTER' and contains a form with 'Username...' and 'Password...' input fields and a red 'Submit' button. The footer includes copyright information: '© 2015 All rights reserved | Template by W3layouts | Customised by Michael Bolger'.

Fig 29. Registration Page

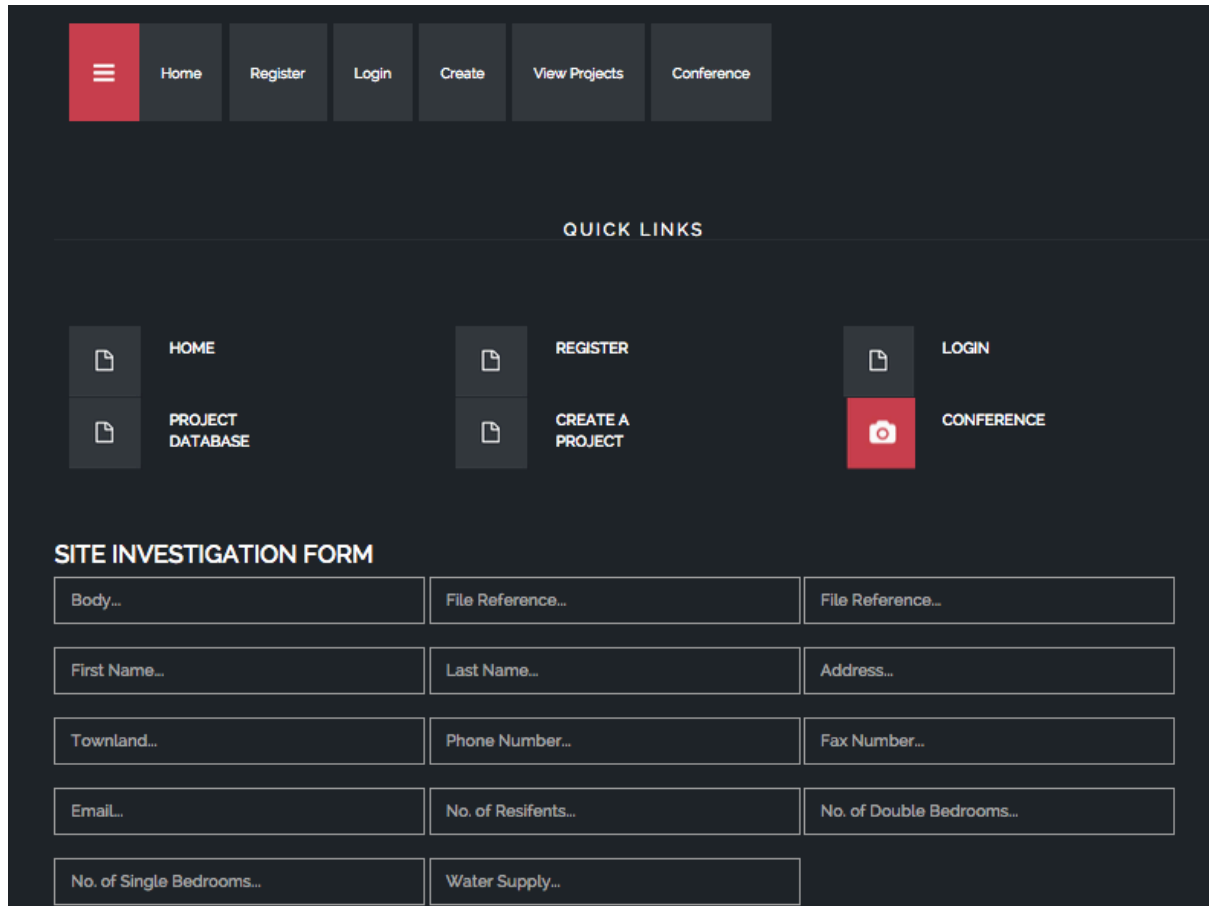
The login page is very similar to the registration page, registered users login via this method, again the password is hidden during input to add an element of security to the process.



The login page has the same layout as the registration page but is titled 'LOGIN'. The 'QUICK LINKS' section is identical. The main form contains 'Username...' and 'Password...' input fields, with the password field having a hidden input type. A red 'Submit' button is positioned to the right of the password field. The footer text is the same as the registration page: '© 2015 All rights reserved | Template by W3layouts | Customised by Michael Bolger'.

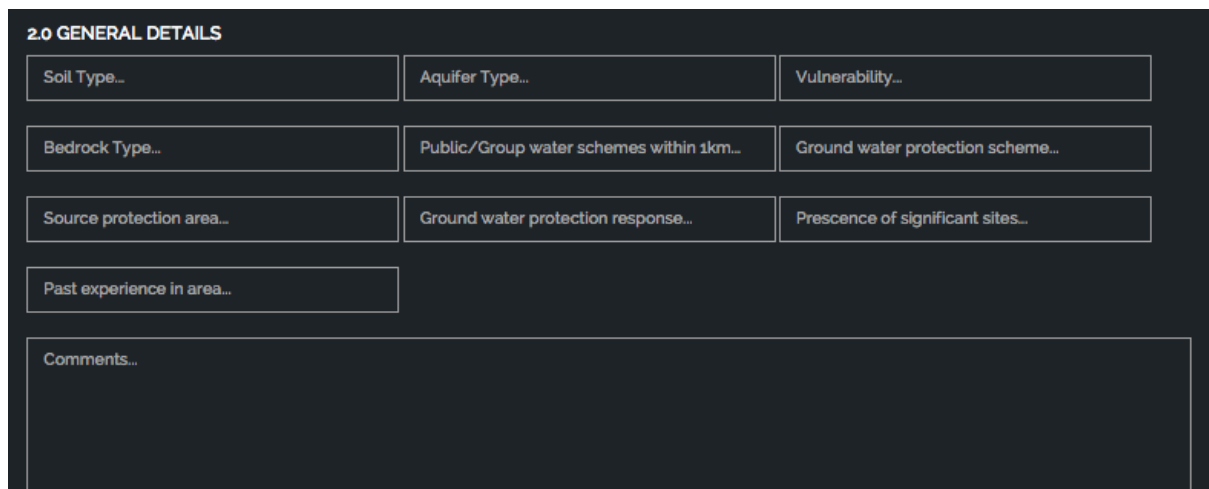
Fig 30. Login Page

The create a project page take the user through the process of completing a site characterisation form; this page is quite expansive with a large number of user inputs. Due to time constraints this was implemented on a single page, in an ideal situation the model would have been segmented to allow for multiple pages specific to each section of the site characterisation form.



The screenshot shows a dark-themed website interface. At the top is a navigation bar with a red hamburger menu icon on the left, followed by links: Home, Register, Login, Create, View Projects, and Conference. Below this is a 'QUICK LINKS' section with a grid of icons and labels: a document icon for HOME, a document icon for PROJECT DATABASE, a document icon for REGISTER, a document icon for CREATE A PROJECT, a document icon for LOGIN, and a camera icon for CONFERENCE. The main section is titled 'SITE INVESTIGATION FORM' and contains a grid of input fields: Body..., File Reference..., File Reference..., First Name..., Last Name..., Address..., Townland..., Phone Number..., Fax Number..., Email..., No. of Resifents..., No. of Double Bedrooms..., No. of Single Bedrooms..., and Water Supply...

Fig 31. Tabs and Section 1



The screenshot shows the '2.0 GENERAL DETAILS' section of the form. It contains a grid of input fields: Soil Type..., Aquifer Type..., Vulnerability..., Bedrock Type..., Public/Group water schemes within 1km..., Ground water protection scheme..., Source protection area..., Ground water protection response..., Prescence of significant sites..., Past experience in area..., and a large text area for Comments...

Fig 32. Section 2 General Details

3.0 ON-SITE ASSESSMENT

3.1 VISUAL ASSESSMENT

Landscape Position...

Slope...

Surface features within a minimum of 250m (Distance to features should be noted in metres)

Houses...

Existing land use...

Vegetation indicators...

Ground water flow direction...

Ground condition...

Sites Boundaries...

Roads...

Outcrops (Bedrock And/Or Subsoil)...

Surface water ponding...

Lakes...

Beaches/Shellfish...

Areas/Wetlands...

Karst features...

Watercourse/Stream...

Springs/Wells...

Integrate the information above in order to comment on: the potential suitability of the site, potential targets at risk, the suitability of the site to treat the wastewater and the location of the proposed system within the site

Comments...

3.2 TRIAL HOLE EVALUATION

Comments...

Fig 33. Section 3 On-site Assessment

T-TEST 2.3

Start time (at 300 mm)...

Finish time (at 200mm)...

Average t(min)...

T-TEST 3.1

Start time (at 300 mm)...

Finish time (at 200mm)...

Average t(min)...

T-TEST 3.2

Start time (at 300 mm)...

Finish time (at 200mm)...

Average t(min)...

T-TEST 3.3

Start time (at 300 mm)...

Finish time (at 200mm)...

Average t(min)...

Comments...

Fig 34. T-Test

3.5 PRE-SOAK P-TEST

Date and time pre-soak started

P-Test Hole 1	Test hole 1 date...	Test hole 1 time...
P-Test Hole 2	Test hole 2 date...	Test hole 2 time...
P-Test Hole 3	Test hole 3 date...	Test hole 3 time...

Fig 35. P-Test pre-soak

3.6 PERCOLATION P-TEST

P-TEST 1.1

Start time (at 300 mm)...	Finish time (at 200mm)...	Average t(min)...
---------------------------	---------------------------	-------------------

P-TEST 1.2

Start time (at 300 mm)...	Finish time (at 200mm)...	Average t(min)...
---------------------------	---------------------------	-------------------

P-TEST 1.3

Start time (at 300 mm)...	Finish time (at 200mm)...	Average t(min)...
---------------------------	---------------------------	-------------------

P-TEST 2.1

Start time (at 300 mm)...	Finish time (at 200mm)...	Average t(min)...
---------------------------	---------------------------	-------------------

P-TEST 2.2

Start time (at 300 mm)...	Finish time (at 200mm)...	Average t(min)...
---------------------------	---------------------------	-------------------

P-TEST 2.3

Start time (at 300 mm)...	Finish time (at 200mm)...	Average t(min)...
---------------------------	---------------------------	-------------------

Fig 36. P-Test pre-soak (cont)

P-TEST 3.1

Start time (at 300 mm)...	Finish time (at 200mm)...	Average t(min)...
---------------------------	---------------------------	-------------------

P-TEST 3.2

Start time (at 300 mm)...	Finish time (at 200mm)...	Average t(min)...
---------------------------	---------------------------	-------------------

P-TEST 3.3

Start time (at 300 mm)...	Finish time (at 200mm)...	Average t(min)...
---------------------------	---------------------------	-------------------

Comments...

Fig 37. Percolation P-Test

4.0 CONCLUSIONS of SITE CHARACTERISATION

Integrating the information from the desk study and on-site assessment (i.e. visual assessment, trial hole and percolation tests) above and conclude the type of system(s) that is (are) appropriate. This information is also used to choose the optimum final disposal route of the treated wastewater.

Suitable for septic tank...

Secondary treatment type...

5.0 RECCOMENDATION

Proposed to install...

...And discharge to...

Trench invert level...

Site specific conditions (e.g. special works, site improvement works testing etc)

6.0 TREATMENT SYSTEM DETAILS

SYSTEM TYPE: Septic Tank System

Tank Capacity(m3)

Proposed to install...

Tank Capacity(m3)

...And discharge to...

Trench invert level...

Percolation Area

Number of Trenches...

Length of trenches...

Invert Level (m)...

Mounded Percolation Area

Number of Trenches...

Length of trenches...

Invert Level (m)...

7.0 SITE ASSESSOR DETAILS

Company Name...

Qualification of assessor...

Indemnity insurance number...

Submit

Fig 38. Conclusions and Submission

When data has been submitted and saved to the database it can be viewed on the view projects page. This page represents all the site-characterisation forms submitted on the site on one page. In an ideal situation the user would be able to limit the number of results presented through a search function, or by a specific input within the report. Unfortunately due to time constraints the data can only be viewed in a date format from newest project submitted to oldest, as a proof of concept for the technologies use this is adequate but for a fully functioning website hosted professionally online these features would have to be implemented.

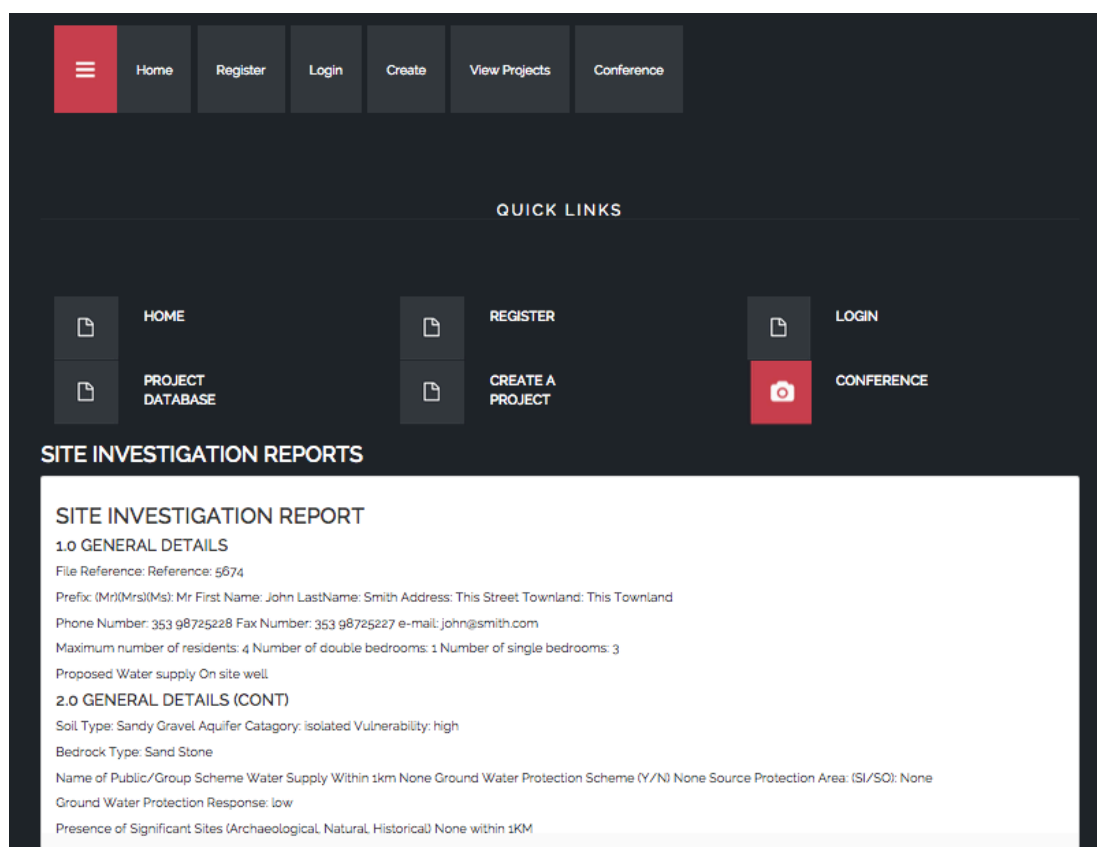


Fig 39. Project Reports Page

Screen size testing

The open source ui was developed with mobile platforms in mind and will adjust the layout of the app to match the screen size and resolution. Initially when starting the project this testing was to be done through emulating various devices via Genymotion, this was not implemented however due to various reasons. Testing of the app on various screen sizes was undertaken using a Google chrome extension called Window Resizer. While useful this extension only adjusts the size of the screen but does not mimic the conditions of a mobile device or the actual mobile device screen, it was however useful to test various screen sizes in this manner especially for desktop screen sizes and resolutions.

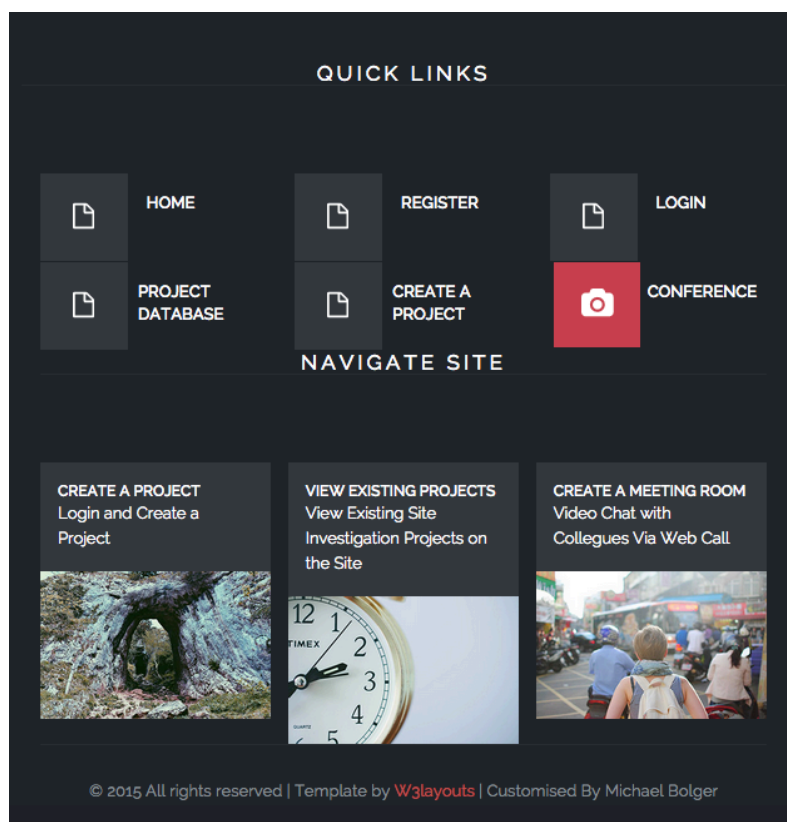


Fig 40. 640x960 iPhone resolution

Requests and Response Codes

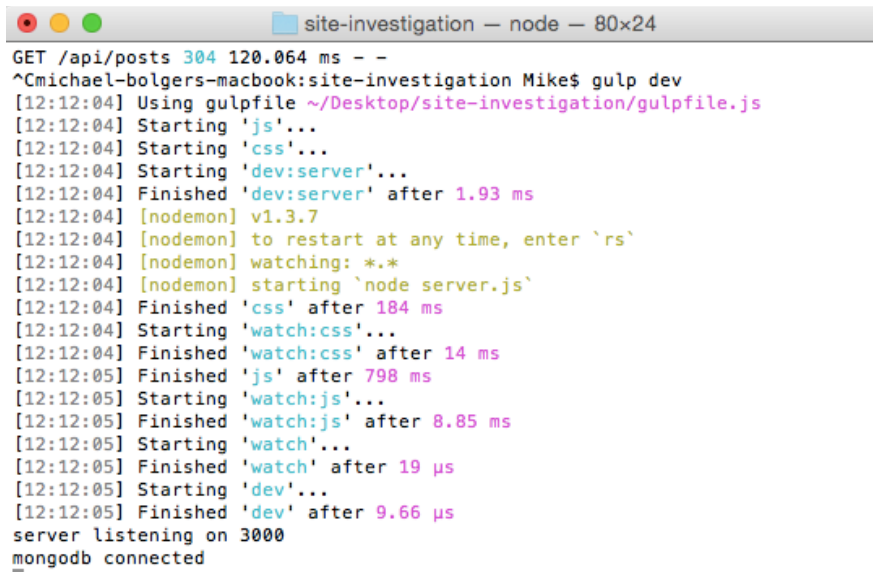
Range	Category
100–199	Informational 100 Continue
200–299	Successful 200 OK 201 Created 204 No Content
300–399	Redirection 301 Moved Permanently 304 Not Modified
400–499	Client Error 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found
500–599	Server Error 500 Internal Server Error 503 Service Unavailable

Fig 41. Request and Response Codes

When requesting or posting data the user will encounter a number of HTTP code, the majority of these will be unbeknown to the user until a problem has been encountered. The image above give an outline of the status codes.

Single Page Application

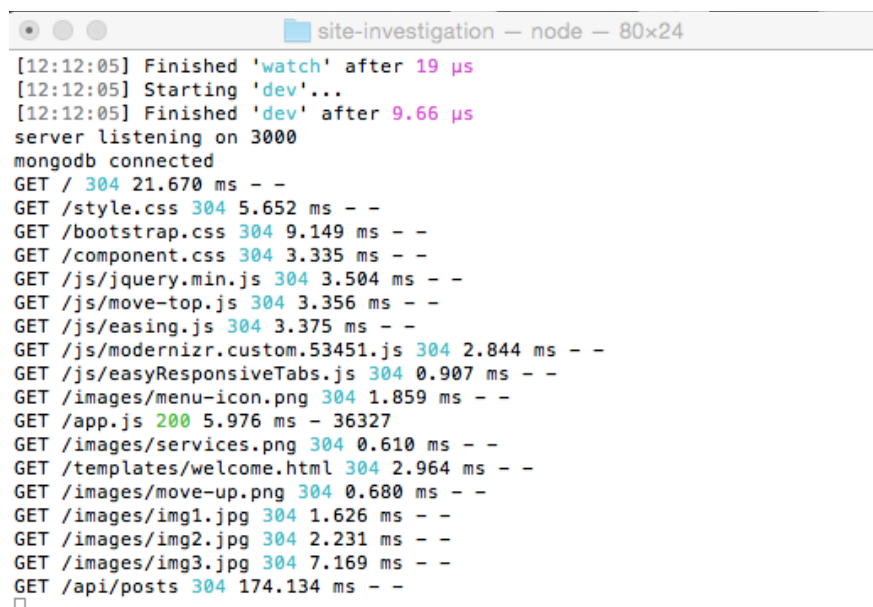
The project has been designed as a single page application where the various pages of the website are loaded in advance, on smaller websites such as this the method is fine as upon initially loading content no further querying takes place which makes navigating the site quicker.



```
GET /api/posts 304 120.064 ms - -
^Cmichael-bolgers-macbook:site-investigation Mike$ gulp dev
[12:12:04] Using gulpfile ~/Desktop/site-investigation/gulpfile.js
[12:12:04] Starting 'js'...
[12:12:04] Starting 'css'...
[12:12:04] Starting 'dev:server'...
[12:12:04] Finished 'dev:server' after 1.93 ms
[12:12:04] [nodemon] v1.3.7
[12:12:04] [nodemon] to restart at any time, enter `rs`
[12:12:04] [nodemon] watching: *.*
[12:12:04] [nodemon] starting `node server.js`
[12:12:04] Finished 'css' after 184 ms
[12:12:04] Starting 'watch:css'...
[12:12:04] Finished 'watch:css' after 14 ms
[12:12:05] Finished 'js' after 798 ms
[12:12:05] Starting 'watch:js'...
[12:12:05] Finished 'watch:js' after 8.85 ms
[12:12:05] Starting 'watch'...
[12:12:05] Finished 'watch' after 19 µs
[12:12:05] Starting 'dev'...
[12:12:05] Finished 'dev' after 9.66 µs
server listening on 3000
mongodb connected
```

Fig 42. Initialising the site with Gulp Dev

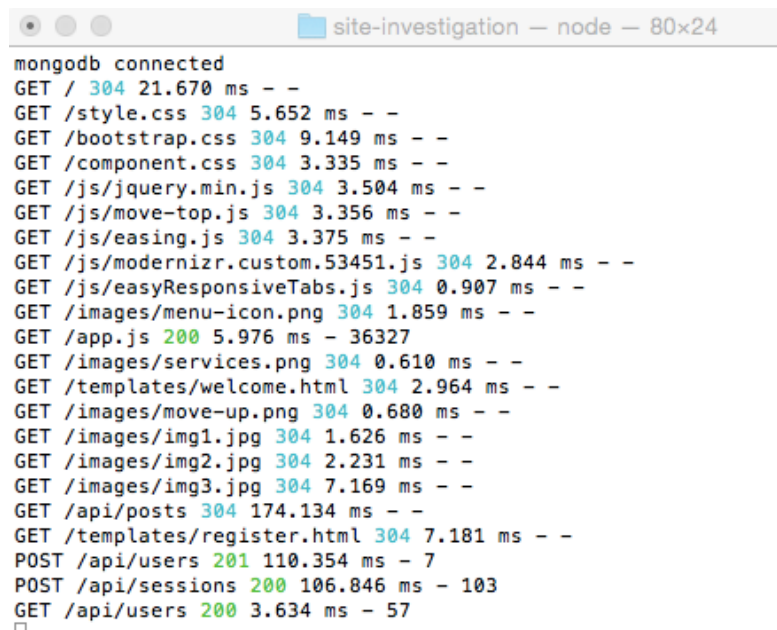
The image above illustrates using gulp how the website is initialised and the data loaded.



```
[12:12:05] Finished 'watch' after 19 µs
[12:12:05] Starting 'dev'...
[12:12:05] Finished 'dev' after 9.66 µs
server listening on 3000
mongodb connected
GET / 304 21.670 ms - -
GET /style.css 304 5.652 ms - -
GET /bootstrap.css 304 9.149 ms - -
GET /component.css 304 3.335 ms - -
GET /js/jquery.min.js 304 3.504 ms - -
GET /js/move-top.js 304 3.356 ms - -
GET /js/easing.js 304 3.375 ms - -
GET /js/modernizr.custom.53451.js 304 2.844 ms - -
GET /js/easyResponsiveTabs.js 304 0.907 ms - -
GET /images/menu-icon.png 304 1.859 ms - -
GET /app.js 200 5.976 ms - 36327
GET /images/services.png 304 0.610 ms - -
GET /templates/welcome.html 304 2.964 ms - -
GET /images/move-up.png 304 0.680 ms - -
GET /images/img1.jpg 304 1.626 ms - -
GET /images/img2.jpg 304 2.231 ms - -
GET /images/img3.jpg 304 7.169 ms - -
GET /api/posts 304 174.134 ms - -
```

Fig 43. Loading Pages

Once data has been loaded on the site once, navigating through pages will return a 304 message, this is not an error but represents the fact that data has already been loaded and does not need to be requested again.



```
mongodb connected
GET / 304 21.670 ms - -
GET /style.css 304 5.652 ms - -
GET /bootstrap.css 304 9.149 ms - -
GET /component.css 304 3.335 ms - -
GET /js/jquery.min.js 304 3.504 ms - -
GET /js/move-top.js 304 3.356 ms - -
GET /js/easing.js 304 3.375 ms - -
GET /js/modernizr.custom.53451.js 304 2.844 ms - -
GET /js/easyResponsiveTabs.js 304 0.907 ms - -
GET /images/menu-icon.png 304 1.859 ms - -
GET /app.js 200 5.976 ms - 36327
GET /images/services.png 304 0.610 ms - -
GET /templates/welcome.html 304 2.964 ms - -
GET /images/move-up.png 304 0.680 ms - -
GET /images/img1.jpg 304 1.626 ms - -
GET /images/img2.jpg 304 2.231 ms - -
GET /images/img3.jpg 304 7.169 ms - -
GET /api/posts 304 174.134 ms - -
GET /templates/register.html 304 7.181 ms - -
POST /api/users 201 110.354 ms - 7
POST /api/sessions 200 106.846 ms - 103
GET /api/users 200 3.634 ms - 57
```

Fig 44. Posting Data

Responses still occur when submitting or retrieving data from the database, in this case a sample user attempts to register on the site represented by POST 201 request fulfilled, and is followed by standard request successful 200 responses both as POST and GET.

End-to-End Testing

To test functionality of the web application the development of an end-to-end testing schema has been designed. These tests are intended to test functionality of the web app by simulating user inputs as if they were navigating the site manually. Protractor was used to run end-to-end testing; protractor is a nodeJS application that uses WebDriver and Selenium to run tests within a firefox or chrome browser.

Protractor required dependencies to run correctly, these dependencies were developed as -dev dependencies and are not needed on a final hosted web app. These dependencies are installed via the npm command with the marker `-dev`. Protractor is only the test runner the actual testing framework used was Mocha. The decision to use Mocha was largely down to the amount of resources available online to aid in implementation.

Protractor when run in the test environment mimics user navigation of the web app via a set of commands created in the test folder under e2e testing. The test implemented runs through the navigating the website to the homepage, through the login process and the submission on viewing of project data.

The test is created and run using two files; the protractor.js file specifies the framework to be used. In this JavaScript file the framework to be used in testing is specified, in this case Mocha and is linked to a file which contains a set of tester implemented commands that specifies how the site should be navigated during the test and mimic user inputs. The file, which specifies how the test is run, is located under the test folder and named `making_a_post.js`. These tests can be expansive or limited as required by the tester; tests can also be linked and navigated through via the protractor console.

Further Testing

End-to-end testing is by no means the only way to test the project; various technologies making up the project could also have been tested individually. The intention was to test the nodeJS server using Mocha and using Karma to test AngularJS, unfortunately the timescale of the project meant that these testing methodologies could not be implemented in time.

Recommended Reading

Creating a web app using the MEAN stack was completely new to me at first, without some very good resources the project could not have been completed in the timeframe. For anyone attempting to tackle node and the MEAN stack the following resources come highly recommended.

Write Modern Web Apps with the MEAN Stack by Jeff Dickey was a good introduction to all the technologies used in developing a web app using JavaScript technologies, in addition to various peripheral features such as testing and hosting of the application.

Web Development with Node and Express By Ethan Brown and O'Reilly media was a good source of general information regarding node and Express outlining an alternative means of implementing these technologies into web apps.

MEAN Web development by Amos Q. Haviv was another good general resource for building a MEAN stack project from beginning to end introducing alternative means to develop, test and implement the project.

Conclusions

The project has been an eye opener into the development of a web application using a JavaScript based approach rather than a Java one. The initial approach for this project was to use the standard and well-trodden road of using Java in association with an SQL database and HTML with Play 1.2.7 acting as the framework to glue all these technologies together. Initially this approach worked well, a prototype was completed in the opening months of the project, however alternative technologies peaked the authors' interest, specifically the MEAN stack. The MEAN stack was a JavaScript based approach for developing web applications without the need to use the Eclipse IDE or Play framework but instead use Node and its dependency management system npm. All dependencies required in the project were installed as modules in the project folder and could be directly referenced within the code. The MEAN stack stands for Mongo, Express, Angular and Node, all JavaScript based technologies and all compatible with JSON to allow for an easy flow of data within the project without needing to convert code from one compatible technology to another. The standout of these technologies has to be Angular, this technology basically acted as a go-between from the back-end database to the front-end HTML page, data was placed in angularised forms could easily be represented on the webpage or saved to the database with minimal actual coding needed.

As a site investigation project it is a good first step and proof of concept that can be taken further, user registration and login features were implemented allowing users to complete site characterisation forms and post data on the web app. To test the login and submission of data Protractor has been implemented as a test runner with Mocha as the testing framework. This end-to-end testing method simulates a user through the login process and the submission of data, by opening a webpage in Google Chrome or Firefox and simulating user input in a visual manner. In addition to these features a video chat feature has been incorporated allowing a one-to-one video conference call to take place within the browser. The web chat utilises WebRTC and Google Hangouts to create a video chat room within the web app allowing for real-time one-to-one communication between users.

The project overall can be considered as a proof of concept for the use of the MEAN stack development approach and WebRTC. Many of these technologies are still in their infancy and may need more time to mature, updates to certain features in WebRTC during the development of the project rendered the conference room non-functioning until the code was updated. While there is a core of functionality within the project, it is by no means the final product, additional features that the author would like to add to the project include: a means to edit existing posts, a user section

where drafts of posts could be saved at various stages of completion, a one to many chat room that could be named and created by the user for others to join, etc.

The project has been a good learning experience overall many new technologies have been researched and implemented into the project not just from the MEAN stack but all the dependencies surrounding the project. The technology which stood out to the author was end-to-end testing using protractor as a test runner and Mocha as the testing framework, visually seeing how the test is run and how the users code determines what happens visually within a web browser is most appealing. In an ideal situation the author would have liked to host the project online via Heroku or cloudbees, however due to time constraints this was not possible. The use of these technologies are a viable alternative to the traditional java based approach that the author has used before, these technologies were easy to use with the amount of helpful data online and a few very good textbooks aiding in development.

References

Ordnance Survey Ireland | Services . 2015. *Ordnance Survey Ireland | Services* . [ONLINE] Available at: <http://www.osi.ie/Services.aspx>. [Accessed 27 January 2015].

Latest Android version breakdown: KitKat and Jelly bean rule. 2015. *Latest Android version breakdown: KitKat and Jelly bean rule*. [ONLINE] Available at: <http://www.androidauthority.com/kitkat-and-jellybean-80-users-563577/>. [Accessed 27 January 2015].

What Are RESTful Web Services? - The Java EE 6 Tutorial. 2015. *What Are RESTful Web Services? - The Java EE 6 Tutorial*. [ONLINE] Available at: <http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>. [Accessed 27 January 2015].

An Introduction to the MEAN Stack. 2015. An Introduction to the MEAN Stack. [ONLINE] Available at: <http://www.sitepoint.com/introduction-mean-stack/>. [Accessed 04 May 2015].

MongoDB For Beginners: Introduction and Installation - Hongkiat. 2015. MongoDB For Beginners: Introduction and Installation - Hongkiat. [ONLINE] Available at: <http://www.hongkiat.com/blog/webdev-with-mongodb-part1/>. [Accessed 04 May 2015].

Web Development with Node and Express – Brown. E(2014). Leveraging the JavaScript Stack. Publisher: O'Reilly Media

Node.js. 2015. Node.js. [ONLINE] Available at: <https://nodejs.org>. [Accessed 04 May 2015].

StrongLoop | Mobile App Development with Full-Stack Javascript (part 1 of 4). 2015. StrongLoop | Mobile App Development with Full-Stack Javascript (part 1 of 4). [ONLINE] Available at: <https://strongloop.com/strongblog/mobile-app-development-with-full-stack-javascript-part-1-of-4-loopback/>. [Accessed 05 May 2015].

Twitter Bootstrap Tutorial. 2015. *Twitter Bootstrap Tutorial*. [ONLINE] Available at: <http://www.sitepoint.com/twitter-bootstrap-tutorial-handling-complex-designs/>. [Accessed 12 March 2015].

Blog. 2015. *Blog*. [ONLINE] Available at: <http://topcoat.io/blog/>. [Accessed 12 March 2015].

Semantic UI - CSS Framework. 2015. *Semantic UI - CSS Framework*. [ONLINE] Available at: <http://cssframeworks.org/details/semantic-ui>. [Accessed 12 March 2015].

Test Framework Comparison. 2015. *Test Framework Comparison*. [ONLINE] Available at: <http://www.theserverside.com/news/1365218/Test-Framework-Comparison>. [Accessed 12 March 2015].

JTiger: unit-testing framework for Java . 2015. *JTiger: unit-testing framework for Java* . [ONLINE] Available at: http://www.automatedtestinginstitute.com/home/index.php?option=com_content&id=713:jtiger-unit-testing-framework-for-java-&directory=77. [Accessed 12 March 2015].

TestNG Tutorials. 2015. *TestNG Tutorials*. [ONLINE] Available at: <http://www.asjava.com/testng/testng-tutorials/>. [Accessed 12 March 2015].

End-to-End Testing with Aurelia and Protractor. 2015. *End-to-End Testing with Aurelia and Protractor*. [ONLINE] Available at: <http://blog.durandal.io/2015/02/16/end-to-end-testing-with-aurelia-and-protractor/>. [Accessed 20 April 2015].

Testing AngularJS apps with Protractor I ThoughtWorks. 2015. *Testing AngularJS apps with Protractor I ThoughtWorks*. [ONLINE] Available at: <http://www.thoughtworks.com/insights/blog/testing-angularjs-apps-protractor>. [Accessed 20 April 2015].

WATERFALL vs. AGILE METHODOLOGY I Agile Introduction For Dummies. 2015. *WATERFALL vs. AGILE METHODOLOGY I Agile Introduction For Dummies*. [ONLINE] Available at: <https://agileintro.wordpress.com/2008/01/04/waterfall-vs-agile-methodology/>. [Accessed 14 March 2015].

Software Development Methodologies. 2015. *Software Development Methodologies*. [ONLINE] Available at: <http://www.itinfo.am/eng/software-development-methodologies/>. [Accessed 14 March 2015].

Play Framework - Build Modern & Scalable Web Apps with Java and Scala. 2015. *Play Framework - Build Modern & Scalable Web Apps with Java and Scala*. [ONLINE] Available at: <https://playframework.com/>. [Accessed 14 March 2015].

WebRTC. 2015. WebRTC. [ONLINE] Available at: <http://www.webrtc.org>. [Accessed 20 May 2015].

STUN Client - CodeProject. 2015. *STUN Client - CodeProject*. [ONLINE] Available at: <http://www.codeproject.com/Articles/18492/STUN-Client>. [Accessed 20 May 2015].