Cover Page

James O’Meara

Contents

[1Table of Figures 5](#_Toc478002304)

[2 Acknowledgments 6](#_Toc478002305)

[3 Introduction 7](#_Toc478002306)

[3.1 Brief 7](#_Toc478002307)

[3.2 Single Page Web App 7](#_Toc478002308)

[3.3 Software engineering Processes’ 8](#_Toc478002309)

[3.3.1 Requirements 8](#_Toc478002310)

[3.3.2 Design 8](#_Toc478002311)

[3.3.3 Implementation 8](#_Toc478002312)

[3.3.4 Acceptance, installation, deployment 8](#_Toc478002313)

[3.3.5 Maintenance 8](#_Toc478002314)

[3.4 Key challenges 8](#_Toc478002315)

[3.5 Timeline 9](#_Toc478002316)

[3.6 Design considerations 10](#_Toc478002317)

[3.7 User requirements 11](#_Toc478002318)

[3.8 System architecture 11](#_Toc478002319)

[4 Technologies 12](#_Toc478002320)

[4.1 Frameworks 12](#_Toc478002321)

[4.1.1 AngularJS 12](#_Toc478002322)

[4.1.2 NodeJS 13](#_Toc478002323)

[4.2 Package Managers 14](#_Toc478002324)

[4.2.1 NPM (Node Package Manager) 14](#_Toc478002325)

[4.2.2 Bower (Client Side Package Manager) 15](#_Toc478002326)

[4.3 Automation Tools 15](#_Toc478002327)

[4.3.1 Grunt (Automation Tool) 15](#_Toc478002328)

[4.3.2 Jenkins 15](#_Toc478002329)

[4.4 Testing Frameworks & Tools 16](#_Toc478002330)

[4.4.1 Jasmine 16](#_Toc478002331)

[4.4.2 Karma 16](#_Toc478002332)

[4.4.3 Jasmine\_node 16](#_Toc478002333)

[4.4.4 Selenium & Protractor 16](#_Toc478002334)

[4.5 Source Control 17](#_Toc478002335)

[4.4.1 GitHub 17](#_Toc478002336)

[4.6 Servers (VMS) 17](#_Toc478002337)

[4.6.1 Amazon EC2 17](#_Toc478002338)

[4.7 Database 18](#_Toc478002339)

[4.7.1 MongoDB 18](#_Toc478002340)

[4.8 Styling 18](#_Toc478002341)

[4.8.1 Css 18](#_Toc478002342)

[4.9 Languages 18](#_Toc478002343)

[4.9.1 Javascript 18](#_Toc478002344)

[4.9.2 Html 18](#_Toc478002345)

[Implementation 19](#_Toc478002346)

[Prerequisites 19](#_Toc478002347)

[Project structure 19](#_Toc478002348)

[Main package structure 19](#_Toc478002349)

[Setting up Angular 19](#_Toc478002350)

[Setting up Node Server 19](#_Toc478002351)

[Setting up testing Frameworks 19](#_Toc478002352)

[Git integration 19](#_Toc478002353)

[Automation (grunt) Building app 19](#_Toc478002354)

[Setting up build & Deployment Server 19](#_Toc478002355)

[Setting up CI with Jenkins 19](#_Toc478002356)

[Tutorials 19](#_Toc478002357)

[Results 19](#_Toc478002358)

[Conclusion 20](#_Toc478002359)

[Evaluation 20](#_Toc478002360)

[Future Work 20](#_Toc478002361)

[Bibliography 20](#_Toc478002362)

[Appendix 20](#_Toc478002363)

# 1Table of Figures

Text

[Figure 1 : High level View Example 10](#_Toc478007685)

# 2 Acknowledgments

I would like to thank IBM, while I was on placement I approached My manager and supervisor regarding Ideas for projects. As this was linked to what I was working on during placement I thought this would be a good idea, since I would have a basis to work on and look more into.

I would also like to thank my supervisor for taking on me and my project and helping me in anything I needed.

# 3 Introduction

## 3.1 Brief

Summarise

Tutorial app / End to End development bringing the user through with tutorials how to set up a single page web app with AngularJS and NodeJS. It will demonstrate end to end development, from hosting the app, building the app, deploying, testing and continuous integration

Create an app that will aid new teams, students or other developers make a start with an AngularJS and NodeJS dynamic single page web app. It should give insight into the full end to end development process from developing the angular components, good practices and full testing frameworks and coverage. It should also have full continuous integration support

## 3.2 Single Page Web App

A Single page web app, is simple that, A single web page which doubles up as an App. Rather than creating a conventional website where the user would be directed & redirected to several pages and have to wait for each individual page to load. A single page web app is an app that can be accessed through the web, and will provide a single access point for simple interaction for the user.

similarly, to an app on a mobile device this web app will provide the same level of functionability, but through a different medium (the Web).

## 3.3 Software engineering Processes’

Here are the general processes in software engineering, whether creating an app for your own use or up to an enterprise app for a large company, you should follow these. The way in which they may be ordered can be somewhat different if you use an Agile, Waterfall or any other approach. But the concept is the same.

### 3.3.1 Requirements

Application requirements are gathered here.

### 3.3.2 Design

Describes desired features and operations in detail, including screen layouts, business rules, process diagrams, pseudocode and other documentation.

### 3.3.3 Implementation

When actual code is written

### 3.3.4 Acceptance, installation, deployment

The final stage of initial development, where the software is put into production and runs actual business.

### 3.3.5 Maintenance

After the product, has launched, this is when additional changes are made, corrections/bugs are fixed and any other modifications that are required to keep it in shape.

## 3.4 Key challenges

Key Challenges for this Project is the level of detail for the tutorials for taking the user from nothing – to having a simple single page web app created, complete CI integration, building and testing. It must appeal to developers that have little to no knowledge of AngularJS and NodeJS but also be specific enough to give the developer an insight into the complete end to end process of a product.

## 3.5 Timeline

( Semester 1 )

Weeks 6-8

* Node & angular set up
* Database connected

Week 9-10

* Automated testing frameworks added
* Portion of instructions done

Week 11-12

* Study for exams…

( Semester 2 )

Week 1-3

* Set up VM for Jenkins and for hosting
* Continue tutorials
* Continuous integration (Jenkins setup)
* Git push requests

Week 4-6

* Set up Jenkins to build on hosting VM
* Research into D3

Week 7-9

* Create data and force graphs for high level views
* Start report?

Week 10-12

* Finish site
* Finish Report

## 3.6 Design considerations

**How would my tutorials look like?**

The tutorials should have clear descriptions, screenshots and examples of code. It should be at a level of detail not to overwhelm who is reading it, but still detailed enough and separated out to make it easy to find and follow certain things.

**Who/What is my audience?**

My audience will be developers / software engineers getting their hands on with AngularJS and NodeJS for the first time. And help give an insight into the whole process from creating the code, to testing it, how it is built/bundled for a website. Then how Continuous integration takes its part for nightly builds and tests and deploys it to the hosting server.

**How will my tutorials be accessed?**

They will be done in word/files or PDF’s and I may combine them to possibly make a small book. But for the fact of having them easily accessible I will make them accessible on my sample app. They will also be included in with the project if the user wants to clone the repository and play around the code themselves, and read the tutorials alongside the source code.

**How am I going to set up Continuous integration?**

I would like to have some VMS accessible at any time, ideally free since money may be an issue to continue with this project. Ideally I would like two, one for a build server and one for a hosting server, and ideally something I can configure/install and start from scratch so I can document this.

**What Testing frameworks would I use?**

For this I would have to look at what there is available, Ideally I would like to cover all aspects of testing, black/white box testing methods and incorporate some kind of code coverage metrics. Testing should include server-side, client-side and End-to-end (automated click through the site)

## 3.7 User requirements

User should be able to look at some sort of poster or graph at first to see the overall scale of the topic, i.e. some sort of high level diagram to get a feel of what keywords/key items that are included and what they are related to. This could be done with word, paint etc., something that could give the user some interactivity would be the best solution to see what is linked to what.

E.g. something along the lines of this very simple graph, that could demonstrate all the links between the different components and processes.

Node

Web App

JavaScript

Angular

Figure 1 : High level View Example

The App should also display some example elements in it, that could be used solely for examples or tutorials, as some parts may start to take on a bit of complexity. The ideal situation would be for the user to clone the repository look at these tutorials, examples on the site and then begin to play about, see how it works, and make their own.

## 3.8 System architecture

This app will have a NodeJS server side and AngularJS client side. This app will not require much support from the server, as Angular JS allows for a responsive dynamic content. It will use a plugin architecture to make use of any modules available by NPM (Node Package Manager) to control all the packages that may be needed.

<https://angular.io/docs/ts/latest/guide/architecture.html>

# 4 Technologies

Below are the technologies that will be used in this project. How they may compare to similar technologies and any they are used.

## 4.1 Frameworks

These will be the main frameworks I will be using in my project, They are Angular and NodeJS which will deal with the front end and the back end of the app.

### 4.1.1 AngularJS

AngularJS is a JavaScript framework to create responsive Dynamic Web Pages. AngularJS lets you use html and JavaScript to create and extend custom elements. AngularJS is also very powerful in a way which makes html dynamic and responsive, without any page reloads since with html alone, element changes or Dom changes must require commutation back and forth between the server and client.

AngularJS is based of JavaScript and s loaded into the browser, allowing for your computer to do the work rather than waiting on a server to respond with what view to load or what type of page. This ensures the page runs faster and more responsive since the user is not waiting for a refresh, or a response from the server. AngularJS does not completely cut out requests and responses to and from the server but removes the unnecessary unneeded requests.

For example, AngularJS 2-way data binding, allows the developer to insert a text box in one position and a display of what is in the text box elsewhere, is something is added to the first, then it will appear in the second. Although this is a very simple example, this means there could be some data loaded into the dom and the use could manipulate it with functions or buttons available dynamically and responsively without having to request the server to do so.

### 4.1.2 NodeJS

NodJS is a JavaScript Framework for the server side. NodeJS is “event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, [Npm](https://www.npmjs.com/), is the largest ecosystem of open source libraries in the world.”

NodeJs is Asynchronous event driven, this means there is a process in the background polling for events, and if certain events fire then certain parts of code or functions would execuete, In JavaScript polling is done by the runtime engine, this allows NodeJS and AngularJS to be more dynamic and allow certain things to happen as events fire. Say you click this button and fire off an event, this means that the underlying engine will pick it up and execute what needs to be done, instead of waiting for a loop. This can prove to be hard to debug as you can not always tell what event may fire off first or respond first.

NodeJS advantage also lies within it being based on JavaScript many developers may work solely on the client side with JavaScript and may not ever touch the backend, with NodeJS and AngularJS working so well together it stands to no reason why anyone working on front cannot wok on the backend and vice-versa. JavaScript and NodeJS is single threaded and since JavaScript is widely known it makes sense to use it as a backend as to not create that new layer of learning on top of the developers currently working on a project.

Synchronous and asynchronous transmissions are two different methods of [transmission](https://en.wikipedia.org/wiki/Data_transmission) [synchronization](https://en.wikipedia.org/wiki/Synchronization#Communication). Synchronous transmissions are synchronized by an external clock, while asynchronous transmissions are synchronized by special signals along the transmission medium.

<https://en.wikipedia.org/wiki/Comparison_of_synchronous_and_asynchronous_signalling>

## 4.2 Package Managers

A Package manager is something that will keep track of the dependencies installed their version and you can also customize what dependencies you want to install and when. Example, install all dependencies for development, since this may require many more dependencies for testing and building, you could also set up it in a way it will only install the dependencies it will need to run on the host server. Making the app take up less room and be more efficient in memory usage.

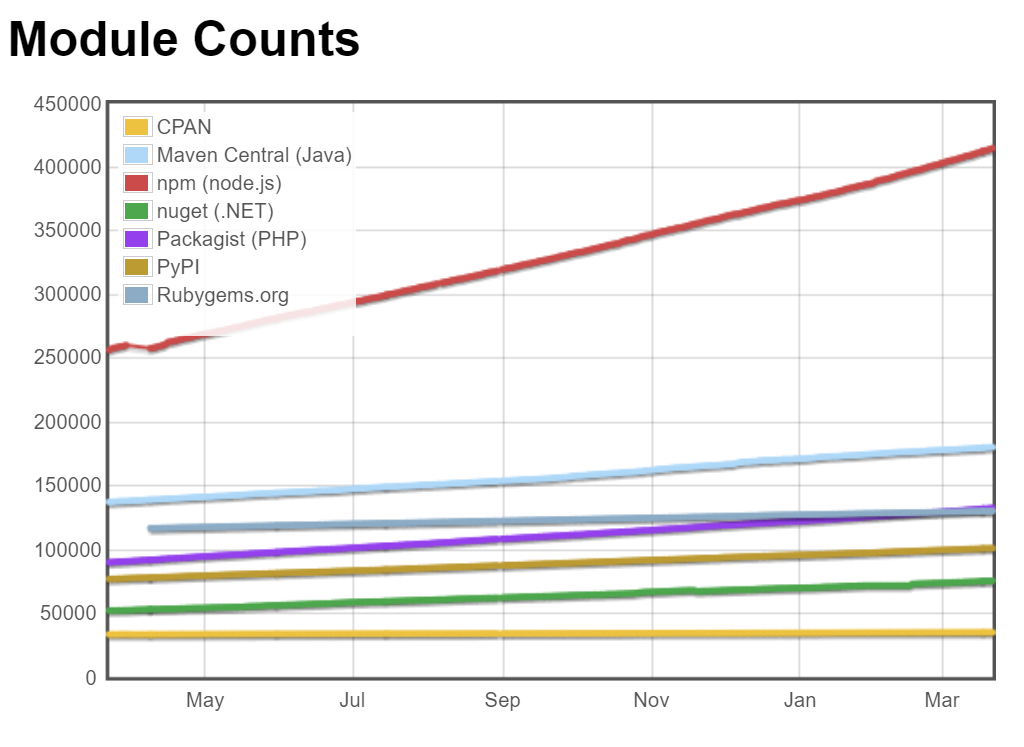
### 4.2.1 NPM (Node Package Manager)

“Npm is the package manager for JavaScript. Find, share, and reuse packages of code from hundreds of thousands of developers — and assemble them in powerful new ways.”

<https://www.npmjs.com/>

Npm is there to install and manage dependencies of your project. You can install dependencies at certain versions and save it in a file called “package.json”, in this file you will find all the dependencies you have installed and the versions In which you have chosen.

Npm is now quite a large package manager that has more than 6 million users a month, and is growing fast compared to many other repository (maven, CPAN, PyPI etc)



<http://www.modulecounts.com/>

### 4.2.2 Bower (Client Side Package Manager)

Compared to NPM, Bower concentrates on client side repository’s and handing client side dependencies’.

“Bower can manage components that contain HTML, CSS, JavaScript, fonts or even image files. Bower doesn’t concatenate or minify code or do anything else - it just installs the right versions of the packages you need and their dependencies.”

<https://bower.io/>

Bower is optimized for the front end of an application compared to NPM which would be an overall package manager, Bower also keeps tracks of the dependencies installed and the versions picked. NPM can be used to install front end packages as well, but bower is more accustomed to installing html, css and other front end packages that can be used in development

## 4.3 Automation Tools

Automation tools are there to take the extraneous tasks away an automate them, they can range from simple tools that will recognize a file being saved and then run a command or it could be all the way to recognizing a git push , re cloning the repo, re installing dependencies re building testing in every way and then finally deploying and restarting the server.

### 4.3.1 Grunt (Automation Tool)

Grunt is a great tool, that can be installed via NPM. Grunt is a task runner and can be used to control any tasks that may need run locally for said app. It can be used to re run builds once a source file has been saved, it can be used to control many other task runner or test runners like karma, jasmine\_node or selenium and protractor, which we’ll find out about later.

<https://gruntjs.com/>

### 4.3.2 Jenkins

Jenkins Is an open source automation server, and is a pretty powerful tool. You can setup a variety of tasks, whether it’s working on the local machine it is running on, set up salve machines to run tasks and processes on or pull/copy/deploy to/from many other places.

Jenkins is a great Continuous integration tool used for automating any project no matter the size. You can set up multiple jobs/tasks to clone your repository, build and install it, run tests and if and only if they all run successfully you can run another job to deploy your app to another server.

## 4.4 Testing Frameworks & Tools

Multiple Frameworks were used for testing, as each method of testing would have tobe carried out and documented. Ranging from client side testing, server side testing and end to end click through testing.

### 4.4.1 Jasmine

Jasmine is the language used for writing the tests, it is a version of JavaScript called behavior driven JavaScript. Jasmine allows the developer to create different suits for each test if needed. Jasmine has a simple way of creating tests “it” function for the actual test and then using “expect” something to be something function.

<https://jasmine.github.io/1.3/introduction>

### 4.4.2 Karma

Karma is a Test runner for client side unit tests. Karma will launch a browser instance of your choice (headless, Firefox, Chrome etc.) and run the tests on that. It will display the output on the command line window.

Karma can also be integrated into Grunt, Jenkins, Travis and many more. Karma can be configured to run continuously, this means as you are editing code, save a file it will pick this up and re-run all the tests and provide the output. Output can also be contained in a html file to show coverage; output can be configured for output type also.

<https://karma-runner.github.io/1.0/index.html>

### 4.4.3 Jasmine\_node

Jasmine Node is the server side equivalent of Karma for server side unit testing, this framework will run the server side tests and output to console. Jasmine Node also provides coverage. Although jasmine was initially made for client side testing, it is quite powerful at mocking, and this can be of great advantage for server side testing.

<https://blog.codeship.com/jasmine-node-js-application-testing-tutorial/>

### 4.4.4 Selenium & Protractor

Protractor is the test runner for this type of test. These tests are used to do click thoughts of the app. For example, go to url, click a button, and then expect text to change colour and something to pop up etc. These are called End to End tests.

Selenium here is used as the webserver, while Protractor is the test runner. Like karma, selenium can create different instances for each test, Chrome, Firefox IE11 etc.

## 4.5 Source Control

Source control is going to be the home of all the code. Where each developer will access the code and update it. Source controls provide great control over who can edit what and when, the history of each edit and structure of the source files.

### 4.5.1 GitHub

GitHub is a great example of a common source control, it has taken off over the last few years and now has over 20million users, and is comparable to subversion, although GitHub is decentralized and each developer, when they make a clone of a repository will have it stored locally. This allows developers to create different branches in their own local branch and then merge it back with the master branch (root branch).

With this local copy of the repository it also comes with all the history, all the previous commits etc. And on top of that since you have it stored locally, you may work on it offline and do not need a constant connection to the centralized version stored in GitHub.

## 4.6 Servers (VMS)

There were many options open to me, in regards to acquiring a virtual machine or server. I had a choice of:

* NUIG to provide me one
* Amazon web services
* Microsoft Azure

Although these are all good services, the one I could obtain by NUIG, would be free, but I would not have root access. This was crucial to me since this project would need at least 1 or 2 servers and access to install any necessary programs and use of ports.

As a student Amazon Web services allows students a year’s free access to VMs, databases etc. although only micro instances, enough for a project so this was perfect. Microsoft Azure was also an option, but this would only provide me with 3 months of free services.

### 4.6.1 Amazon EC2

Amazon EC2 is amazon’s web services platform for controlling virtual machines, it is super easy to spin up and shut down as much machines as you would like. Under the free tier that I got allowed me to have 1 t2 micro instance for a full month for 12 months, this was good but I could use 2 instances, as long as I did not go over the prescribed usage then I would not have to pay for them, although it is very cheap for a t2 micro instance this exactly what I was looking for.

When you create an instance, you are presented with all various operating systems and you may choose which type you want which was also a plus.

## 4.7 Database

Somewhere that all data will be stored, allowing different access to different users is also great.

### 4.7.1 MongoDB

I used MongoDB provided by NUIG as a student I was eligible to get a database and use it for this project. I could have used Amazon Web Services, but I had already this set up before I found out about Amazon EC2.

MongoDB is an open source document oriented database. This allows you to store objects within it, it is termed NoSQL, meaning that this database is not a traditional relational database.

## 4.9 Languages

These are the languages that will be used during this project.

### 4.9.1 JavaScript

JavaScript is a high-level object oriented programming language; it is widely used in web browsers. In this project, it is used in almost everything, from client side to server side since both angular and Node both use JavaScript. Also, all the testing frameworks use jasmine, which is a JavaScript testing language. And finally, all the test runners, task runners, grunt, Jenkins, karma, protractor etc. use JavaScript.

### 4.9.2 Html

Html is hypertext Markup Language which is used for web pages. It uses a standardized tagging system which can take multiple attributes to change things like colour, size, placement etc.

Html will be used to create the web pages.

### 4.9.3 CSS

CSS is a styling language that ties in with JavaScript and Html, to give the web pages a style, colours, placement of elements etc.

# 5 Implementation

## 5.1 Prerequisites

Must have prior knowledge of JavaScript at a beginner’s level, and an understanding of MVC (Modal View Controller)

As JavaScript is widely used throughout this project for creating the back end, front tend of the app, the testing and most of the continuous integration.

Also, some knowledge of Linux and have used command line before. This will be needed to SSH into the various virtual machines and there will be no display.

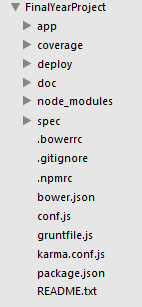
## 5.2 Project structure

Order:

* Setting up Node Server.
* Setting up Angular Web App.
* Create Git Repository, link with app.
* Set up automaiton for development
* Configure building tasks for app
* Start on tutorials, this will be continued as more work is done.
* Set up client side tests
* Set up server side tests
* Boot up 2 linux vm instances
* Set up Jenkins on one VM
* Set up Node and host app on other VM
* Create Jenkins build tasks

## 5.3 Main package structure

This is the file structure of the App, there are many folders, for the app source code, output from coverage, test spec files, and dependacies for the app.



* App

In here all the source code can be found, there will be a public folder for anything that will be publicly available for the app. Along with any publicly available libraries.

* Coverage

This is the output from running the tests, This can be in the form on a .json file, html file or otherwise. Generally, it will be outputted as .html which will give an intractable page, in which the developer can see what is being tested and how often.

* Deploy

This folder will store any files that are for deployment. As there will be a Jenkins server which can handle this, it may be necessary to deploy the app locally.

* Doc

This folder will store all of the tutorials, this will be there for anyone wanting to clone this repository and browse the tutorials

* Node Modules

This Folder will store all the dependencies that NPM will install

* Spec

This folder will house all the test files

* .bowerrc, .npmrc

These are the config files for Npm and bower, these files will detonate where each manager will install its packages and any other config

* .gitIgnore

Config file for Git, useful to outline what files should be ignored for committing, since we do not want to add the dependency files and folders into the git repository and force the user to install them themselves a gitignore can be used.

* Bower.json, package.json

These 2 files are the .json config files in which the dependencies are specified for Npm and bower. These files also keep track of version of each dependency and type.

* Gruntfile.js

This is the config file for grunt, all tasks will be foind inside this file. Grunt will take palce of the main task runner of the app, from grunt we will call our other test runners via grunt plugins. This will streamline how tasks will be run as only one task runner will be used/called.

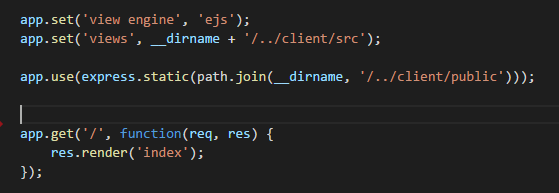
* Conf.js

This is the config file for protractor and selenium, this file will store locaitons of all the test files that are for End to End tests among any other relevant config.

* Karma.conf.js

Karma config for all client side tests. This config will store locations of all client side tests/specs and will handle any config for karma test runner.

## 5.4 Setting up Node Server

First off a new empty folder was created, Node was installed and then a node project was created via nodes “npm init” command. This will create the file “package,json” and ask a few questions on the configuration of the app.

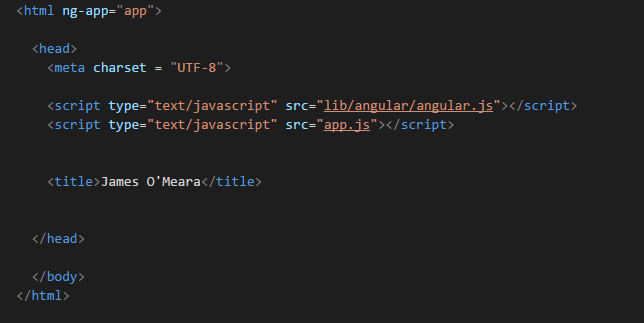
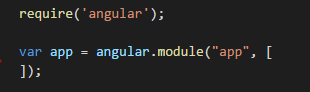
Next express was installed via Npm and a “server.js” file was created and the code to start a node server was added. Once the URL was accesses it would return an index file.

## 5.5 Setting up Angular

Since Node had been set up, there now had to be a file to be rendered, this is where the Angular side kicks in, Bower was installed via Npm, as this could then be used to keep track of client side dependencies and then angular was installed via Bower.

A “ejs” file was made and called “index.ejs” this will be the root file that will be rendered when the node server is requested to render the app.

Since we are using angular, angular must be included into the file so it will be loaded into the browser. Next I created the starting point of an angular module app. This will be the entry point for the angular within the browser.

The app is now ready to start with a minimal node server and angular app.

## 

## 5.6 Setting up testing Frameworks

Karma

Jasmine node

Protractor

Coverage

## 5.7 Git integration

Creating git project

Linking with Jenkins

## Automation (grunt) Building app

Grunt watch task

Npm start task

Karma tasks

Jasmine node

Browserfiy

## Setting up build & Deployment Server

Port forwarding,

installing packages,

using amazon ec2,

booting up instances /

## Setting up CI with Jenkins

Creating tasks and linking git with Jenkins

slave machines

## Tutorials

Using word

Screenshots

Small expolinations

Spitting up to separate files

## Working on the App

## D3 integration

## Simple examples

## Embedding plunker and jsfiddle

## Loading tutorials into the app

# Results

Text

# Conclusion

Text

# Evaluation

Text

# Future Work

Text

# Bibliography

Text

# Appendix