A Job Portal For Software Developers

Final Year Project
B.Sc.(Hons) in Software Development

BY
NEIL KYNE

MAY 10, 2020

Advised by Dr. John French
DEPARTMENT OF COMPUTER SCIENCE AND APPLIED PHYSICS
GALWAY-MAYO INSTITUTE OF TECHNOLOGY (GMIT)

Contents

List of Figures

Abstract

This final year project was inspired by my desire to develop a practical web application. A project that could give me a sense of what it would be like to work on a production ready application in this industry today. I chose to make it a job portal aimed at developers simply because I felt it would further engross me into this industry further.
The application uses React as its primary technology with an API built using Spring Boot. All of the job data is hosted on MongoDB Atlas. It uses JWT for user authentication and is fully deployed on Heroku.
The application itself contains many features such as the ability to login and logout, perform CRUD functionalities on jobs as well as custom searches. The main goal in developing the front-end was to make it as 'Reactful' as possible, minimising repeat code by using as many components as possible while developing the back-end focused more on ensuring the API was RESTful.
The following document details my development journey throughout my final year, from initial research into technologies I was interested in using to problems faced during development.

Chapter 1

Introduction

The following document is a dissertation for my final year project in which I hope to achieve a B.Sc. (Hons.) in Software Development. This project was undertaken by just myself and all work shown is my own.
From the beginning of the year I knew I wanted to make a practical web application, a website that could actually be useful to people and would also provide me with a suitable challenge and rewarding, worthwhile learning outcomes. Much research went into what kind of project I would do and what technologies I would use to do it which will be discussed further on in the document.
The finished project however is a job site for software developers. I opted to use mainly React, a Javascript library for the front-end along with some accompanying frameworks and used Spring Boot for the back-end as well as MongoDB for the database.

1.1 Early Research

Initially the project was going to take a much different shape in my mind. I had planned on it being more of a general purpose 'odd-job' type app where users could post or accept listings advertising a certain amount of pay for menial jobs such as mowing a lawn or cleaning up around a house etc.
As the idea grew in my head I realised I would prefer it to be specific to my field of study. I decided my project would be a job site specifically for software developers. This way I could incorporate OAuthentification into the website with developers logging in via their LinkedIn or even Github accounts. This would tie in nicely to the general theme and purpose of the website. OAuth doesn't share password data but instead uses authorization tokens to prove an identity between consumers and service providers. OAuth is an authentication protocol that allows you to approve one application interacting with another on your behalf without giving away your password [1].

1.1.1 MEAN Stack

I had initially planned on developing the project using the MEAN stack. The MEAN stack (MongoDB, Express.js, AngularJS and Node.js) is a free and open-source JavaScript software stack for building dynamic web sites and web applications [2]. Because all components of the MEAN stack support programs that are written in JavaScript, MEAN applications can be written in one language for both server-side and client-side execution environments. However, after discussing it with my supervisor, I learned that this might not have appeared challenging enough to some as it had already been covered in the previous year. A big part of the applied project is to showcase self learning and the ability to understand foreign concepts that haven't been explicitly taught in class. Although I joined this year after 2 years out of college I agreed it would be better if I tried to show technologies not already covered on the curriculum and weren't seen before among this group of students.

1.1.2 MERN Stack

After deciding against the MEAN stack, I looked into the MERN stack (MongoDB, Express, React, and Node. js), again however, there wasn't really enough variation in language and I would have been making a glorified JavaScript application.

1.1.3 Dropwizard & Spring Boot

I decided I was trying too hard to conform to a predefined 'stack' and realised I didn't have to choose a set of technologies that work well together just because they form a nice acronym. I wanted to have a back-end built in a language other than JavaScript and for developing RESTful web services you don't need to look much farther than DropWizard or Spring Boot.

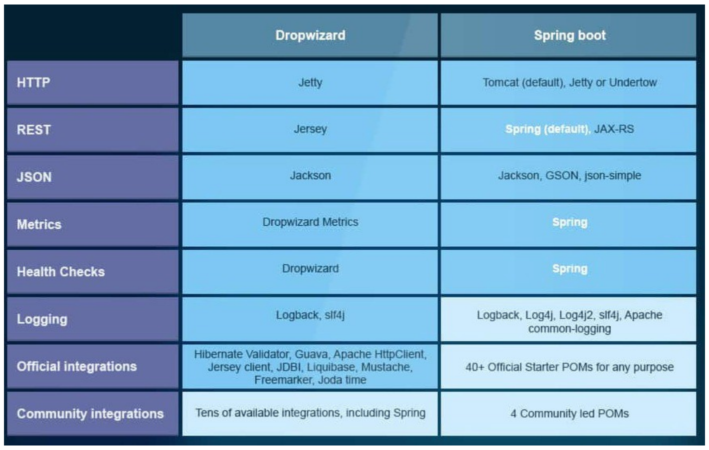
Dropwizard

Dropwizard is a Java framework for developing web services in Java. Dropwizard makes use of multiple different packages such as **Jetty, Jersey and Jackson** that let you deploy a micro-service with relative ease[3]. On top of that, we were actually using Dropwizard in our first semester for the module 'Distributed Systems' so I would have had the added help of knowledge gained from these assignments.

Spring Boot

The other technology I looked into was Spring Boot. Like Dropwizard, Boot makes it all too easy to get a Java web application up and running in mere minutes. Spring Boot is a Spring framework module which provides Rapid Application Development (RAD) feature to the Spring framework. It is highly

dependent on the starter templates feature which is very powerful and works flawlessly [4].
I did a lot of research into these two technologies and found that I was only one of a great many developers unsure about which one to go for. There are countless articles and websites online going into great details comparing the two frameworks and I found it quite easy to get bogged down by an overload of information on the two (Figure 1.1.3). In the end I opted for Spring Boot, I found examples of how well it interacts with MongoDB and decided that would be very useful in my application. I don't think development of the API would have suffered at all had I gone for Dropwizard, they're both clearly two very popular technologies for Java web development.



	Dropwizard	Spring boot
HTTP	Jetty	Tomcat (default), Jetty or Undertow
REST	Jersey	Spring (default), JAX-RS
JSON	Jackson	Jackson, GSON, json-simple
Metrics	Dropwizard Metrics	Spring
Health Checks	Dropwizard	Spring
Logging	Logback, slf4j	Logback, Log4j, Log4j2, slf4j, Apache common-logging
Official integrations	Hibernate Validator, Guava, Apache HttpClient, Jersey client, JDBI, Liquibase, Mustache, Freemarker, Joda time	40+ Official Starter POMs for any purpose
Community integrations	Tens of available integrations, including Spring	4 Community led POMs

Figure 1.1: Comparison of 3rd party libraries

1.2 What the Project Brings

The premise of the project as touched on above, was to bring a small number of technologies together, and have them working efficiently and effectively with a clear reason for their use. As it was just myself in the group, I was generally happy for the scope to be a bit smaller as long as the project itself was an effective program functioning well across different levels in the stack.
The 3 main technologies effectively offer everything a web application would need to be deployed and used by an actual user.

1.3 Scope

As mentioned earlier, I developed the project by myself. Taking this into consideration I felt that a full stack application, hosted online was suitably challenging.

The project's architecture consists of three unique tiers all communicating in a stack to form the web application. The three parts are as follows:

1.3.1 Database/Back-End Tier

I used MongoDB for my database technology. MongoDB is a NoSQL database, it is document oriented and it stores data in collections instead of tables [5]. The main use of the database is to store my job objects. The users of my application can perform basic CRUD functionality on the jobs stored in the database. My MongoDB database is hosted on a cloud service called MongoDB Atlas. This is done so that when they application is deployed online via Heroku, the data is available at all times.

1.3.2 Server Side/Middle Tier

For my 'middle' tier I used Spring Boot. Spring Boot is a fantastic variation of Spring that allows for quick painless development of a web application with almost no configuration. The RESTful API is contained in my Spring server and connects the React application to the Mongo database containing the job objects to be manipulated. The REST API is hosted as its own application on Heroku and communicates with the React application to perform requests.
A deciding factor in choosing Spring boot as my server side technology was it's very powerful MongoDB connector. All you need to do is simply extend the 'MongoRepository' class from a repository interface which contains plenty of generic methods already. Impressively, you do not need to manually implement these queries yourself, instead you can use Spring Boot's repository naming conventions, and the 'MongoRepository' will intelligently construct the queries at run-time [6].

1.3.3 Front-End Tier

For the front end of my web application I decided to use React. React is a Javascript library for building UI's. However, as it is only concerned with rendering data to the DOM. React requires additional libraries for state management, sending HTTP requests and routing such as React Router and Axios [7]. I use multiple Javascript libraries in my application to compliment React such and will examine them later in the document.
I performed testing on the front-end application using two testing frameworks, 'Jest' and 'React-Testing-Library'.

1.4 Objectives

Like the vast majority of students in my year, my primary objective in developing this application was to learn a new set of skills and to become familiar with a new set of technologies. It was also a great opportunity to develop and improve my project management skills as it demands attention over an extended period of time. I believe by choosing technologies I have had no substantial previous experience with I definitely accomplished this objective. I set the following goals for my project from the very beginning of the year:

- I wanted my application to have a number of moving parts and to be a real 'Full Stack Application'. I felt a 3 tier system would go a long way towards showing me how a production ready application might actually look in the working world. When developing across different levels you learn a lot about how each level of the stack works, how they communicate with each other, what works well together and what doesn't.
- As I have a big interest in web development in particular, I specifically wanted to become very familiar with React. According to Stack Overflow's 2019 survey, more developers say they use React.js than Angular, a switch from last year [8]. It ranked second overall behind JQuery.
- Adding to what I said earlier about getting a taste for what a real world application might look, I felt it was very important to have genuine authentication. I initially wanted OAuthentication but opted instead for JSON Web Tokens or failing that, basic authentication.
- I wanted users to be able to perform the basic CRUD functionalities with jobs.
- I wanted the application to be deployed on Heroku.

1.5 Documentation

This document will give a detailed description of my research, planning, reasoning and thought process throughout the development of my application. It is split up into six chapters. Each detailing a different aspect of the work put into the application itself. They are:

Introduction

The current chapter, the introduction, has covered what my aims for the project were early on. What research I did before choosing to develop using the technologies I did and also my objectives for the project before beginning development.

Methodology

This chapter will go over how I went about the development of my project at a more detailed level. It will cover how often I met with my supervisor and how often I dedicated each week to the project. It will essentially cover how I went about developing and planning the entire project, which project methodology was used and if I felt it was at the right level.

Technology Review

This chapter will documents the outcome of my research into the technologies I used and how I used them to achieve my objectives. Every technology used in the development of the application will be described here, covering what they do in general and what they offer to my application specifically.

System Design

This section will describe the architecture of the application. It will cover how each tier of the application communicate together to form the complete project. It will contain diagrams and screenshots to help visualise the structure and architecture of the application.

System Evaluation

In this chapter I will do a full evaluation of the application as a whole. I will compare the final outcome of the application with the objectives I initially outlined in the Introduction chapter. I will also outline any limitations or difficulties I found in the project be it through technologies used or through my own decisions and mistakes. I will also suggest improvements I believe could be made on the application.

Conclusion

The conclusion will be a summary of the entire finished project. It will discuss my initial goals for the project and how well I managed to achieve them. I will cover what I feel I learned throughout the development of the project and the writing of the dissertation. It will also outline any regrets I have with the project be they from using the wrong technologies or other matters. Overall I hope it will provide a final piece of insight on my experience with the project and hopefully wrap up the documentation.

Chapter 2

Methodology

2.1 Initial Planning and Research

When I first set out beginning this project I wanted to make sure I gave the planning phase enough attention. Often times in the past for previous college projects I found myself jumping straight into coding without having properly thought out and planned what I actually needed to do and in what order. This can often times lead to running into obstacles further down the line in the project that become harder and harder to revert or fix the longer they are left.
I knew I wanted to use a JavaScript framework for the front end design. As I had initially planned on using the MEAN stack for the project I brought this up with my supervisor who informed me that this stack had been covered in some degree in the previous year, therefore it might seem as if I am not challenging myself enough with a new technology. Of course, I wasn't in this year and had no prior learning with it but opted to leave it nonetheless.
After meeting my supervisor and swapping ideas I decided to do some further research into popular technologies for designing web applications and landed on React for the front end and MongoDB for the database tier. My supervisor recommended I look into Spring Boot or Dropwizard for the back-end side of things and from there I had a decent amount of material and technologies to familiarise myself with before the actual development of the application could begin.
Although testing wasn't very high on my list, as is the case for many student projects, I knew I would need to at least 'dip my toe' into it. It was important for me to gain exposure to every facet of developing a full stack application. Research into testing React applications very quickly led me to 'Jest' and 'React Testing Library' which I will cover in further detail later.

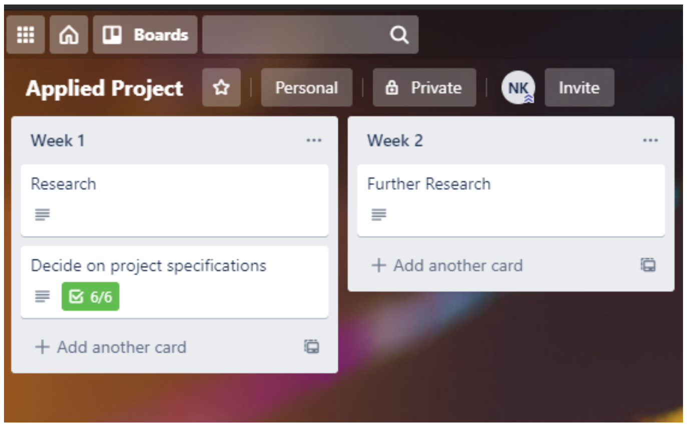


Figure 2.1: My Trello Board

2.2 Project Planning Tools and Project Meetings

As I did not form a group with any other students I simply set aside two hours every week for my own individual project meetings. I usually did this the day before meeting with my supervisor in order for things to be fresh in my mind with which to discuss. Of course as time goes on and depending on other project deadlines, I often went well over two hours a week and occasionally wen under this. I tried to always do something however. As I touched on before, I really didn't want to rush into coding before I had a decent idea of what I wanted to do and in what order I needed to do it. My supervisor was very helpful in this respect and used to set out weekly objectives for me to have completed before our next weekly meeting.
As far as project planning tools, I initially used Trello. Trello is a project managing software that helps to visualise projects as well as organise workloads into different cards [9]. Initially, I used Trello quite a lot for planning and research as well as making sure I set objectives and deadlines for such research (Figure 2.2). As the weeks went on however I found myself using it less and less and instead, just noting what goals I wanted to achieve in a diary instead. I did this because one of Trello's main features is it's team feature which grants multiple people access to a board, as I was the only person in my project group I didn't really need to worry about the planning being online and available to team members.

2.3 Github as a Development Tool

As with all projects I undertake, I used Git for version controlling the application as well as backing up the application. Looking back, I feel I didn't utilise Github

as well as I would have liked. Github offers a lot of functionality such as the issue tracker or the wiki. I largely only used Git for backing up my project and always having the source code accessible from any device.
If I was to start the development of this project again I would definitely have used the issue tracker as often as possible to show the problems I faced with development as well as how I overcame them. I began using the issue tracker very late in the project and I only managed a small handful of entries.Not using it throughout is a big regret I have, since I 've started using it I've found it's a great way to showcase your thinking at each stage of a problem. I also would've used branches to always keep a clean working copy of my application available that I could revert to at any time. Git is a very important tool to software development and I feel I might have used this opportunity better to showcase my understanding of it more.
The wiki is another powerful feature I might have used. The wiki provide a place in your repository to lay out the road-map of your project, show the current status, and document software better. I instead opted to simply keep a Word diary on my local machine, that I made entries to whenever I worked on the project. I believe documenting your thinking during development is very important.

Chapter 3

Technology Review

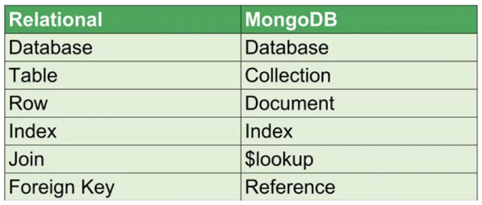
In this section I'll examine the different technologies that were used in the application. I'll explain why I chose to implement each one specifically and whether or not I feel they were the right choice. As well as the major technologies used I'll touch on frameworks or libraries I feel are relevant to mention for the corresponding technology. The core technologies used that I will scrutinize the most in this section are; Spring Boot, React, MongoDB and Heroku.

3.1 MongoDB

MongoDB is a NoSQL document oriented database program. A MongoDB database can contain multiple collections, each record in a collection is document. Documents are a structure composed of file and value pairs, similar to JSON objects or other mapping data types [10]. The table below (Figure 3.1) illustrates some differences between a MongoDB database and a relational database.
Not only Sequential Query Language (NoSQL) is a database type that provides a mechanism for storage and retrieval of data which is modelled by means other than the tabular relations used in relational databases, in the case of MongoDB, it is document based. NoSQL databases are a favored solution for 'Big Data' storage as they incorporate a schema-less data model [11]. 'Schema-less' means the database doesn't have fixed data structure. A NoSQL database provides increased scalability and flexibility compared to relational databases.

3.1.1 MongoDB Atlas

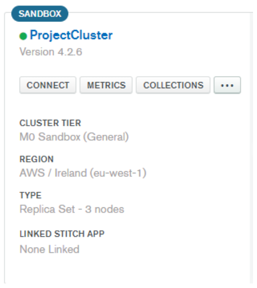
MongoDB Atlas is a cloud based system developed by MongoDB. Atlas provides an easy way to host and manage your data in the cloud with the service provider of your choice, for this application, I used Amazon Web Services (AWS).
Atlas uses clusters to manage the data in a MongoDB project. Clusters are simply Atlas-managed MongoDB deployments and can be either a replica-set



Relational	MongoDB
Database	Database
Table	Collection
Row	Document
Index	Index
Join	$lookup
Foreign Key	Reference

Figure 3.1: MongoDB vs relational DB

or a sharded cluster [12]. (Figure 3.1.1) shows my atlas project is a replica set with three nodes



ProjectCluster
Version 4.2.6

CLUSTER TIER
M0 Sandbox (General)

REGION
AWS / Ireland (eu-west-1)

TYPE
Replica Set - 3 nodes

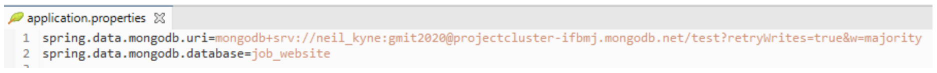
LINKED STITCH APP
None Linked

Figure 3.2: My MongoDB Project Cluster

A replica-set means there are multiple instances (in my case three) of MongoDB which each mirror all the data of the other. A replica-set consists of one primary node (master) and one or more secondary nodes (slaves). Read-operations can be served by any slave, but write-operations always take place on the master of the replica-set and are then propagated to the slaves. Replica-sets offer fault-tolerance as when one of the members of the replica-set goes down, the others take over. When the master goes down, the slaves will elect a new master.

Atlas In My Application

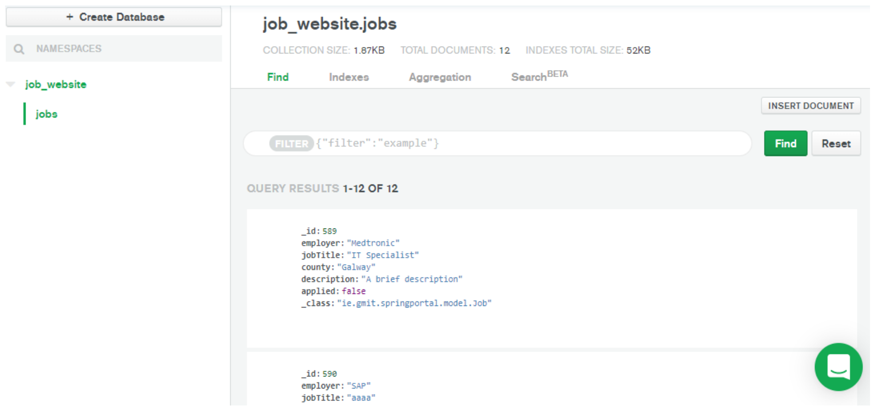
To connect my Spring Boot application to my MongoDB cluster I simply had to configure it's connection address in the 'application.properties' file (Figure 3.1.1).



```
spring.data.mongodb.uri=mongodb+srv://neil_kyne:gmit2020@projectcluster-ifbmj.mongodb.net/test?retryWrites=true&w=majority
spring.data.mongodb.database=job_website
```

Figure 3.3: Connecting to My MongoDB Cluster

After providing the connection details above, our database, collection and all of the documents associated with that collection can be viewed on the MongoDB Atlas project page (Figure 3.1.1).



job_website.jobs

COLLECTION SIZE: 1.87KB TOTAL DOCUMENTS: 12 INDEXES TOTAL SIZE: 52KB

FILTER {"filter":"example"}

QUERY RESULTS 1-12 OF 12

```
_id: 589
employer: "Medtronic"
jobTitle: "IT Specialist"
county: "Galway"
description: "A brief description"
applied: false
_class: "ie.gmit.springportal.model.Job"

_id: 590
employer: "SAP"
jobTitle: "aaaa"
```

Figure 3.4: My MongoDB Database

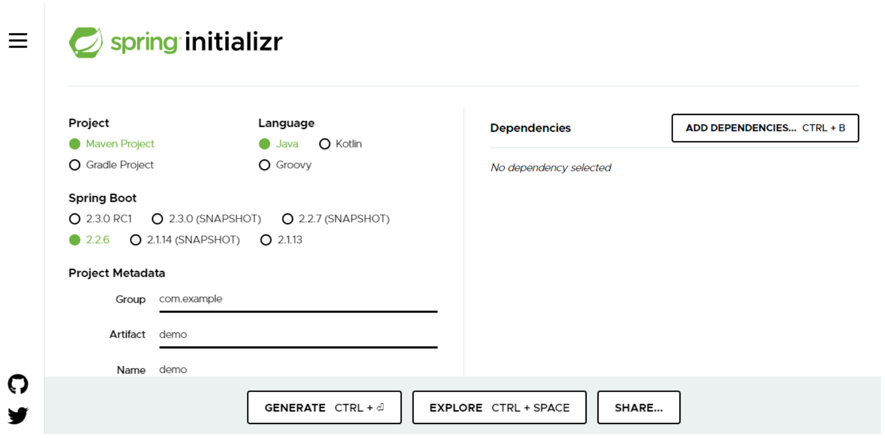
3.2 Spring Boot

To properly understand Spring Boot we must first briefly summarize its foundation framework, Spring. Spring is one of the most popular Java Enterprise Edition (EE) Frameworks for building web services. For the Java platform, the Spring framework provides an elaborate programming and configuration model. It can be used at any kind of deployment platform as unlike other frameworks, Spring focuses on several areas of an application and provides a wide range of features. However, in recent years, due to added functionalities, the Spring framework has become increasingly complex. It requires going through a lengthy procedure in order to start a new Spring project. To avoid starting from scratch and save time, Spring Boot has been introduced. This uses the Spring framework as a foundation [13].

Spring Boot allows developers to create stand-alone, production-grade Spring based Applications that you can "just run". It take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration [14].

3.2.1 Creating Spring Boot Application

I chose to 'Bootstrap' my application with the 'Spring Initializr' provided on the official Spring website (Figure 3.2.1). This is a very handy tool that lets you pre-configure project settings such as dependencies, languages or project type (Maven) and get setup and ready to code within minutes.



Project: [✓ Maven Project] Gradle Project

Language: [✓ Java] Kotlin Groovy

Spring Boot: 2.3.0 RC1 2.3.0 (SNAPSHOT) 2.2.7 (SNAPSHOT) [✓ 2.2.6] 2.1.14 (SNAPSHOT) 2.1.13

Group: com.example

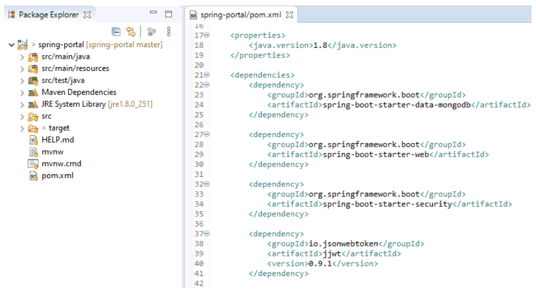
Artifact: demo

Name: demo

No dependency selected

Figure 3.5: Spring Initializr Setup Page

Once I generated the Spring Boot project from the 'Initializr' I simply needed to import that package to Eclipse as a Maven project. As we can see in (Figure 3.2.1) all of the dependencies chosen in the Initializr have been added to the 'pom.xml' file.



```
spring-portal [spring-portal master]
  src/main/java
  src/main/resources
  src/test/java
  Maven Dependencies
  JRE System Library [jre1.8.0_251]
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

```xml
	<properties>
		<java.version>1.8</java.version>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-mongodb</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-security</artifactId>
		</dependency>

		<dependency>
			<groupId>io.jsonwebtoken</groupId>
			<artifactId>jjwt</artifactId>
			<version>0.9.1</version>
		</dependency>

```

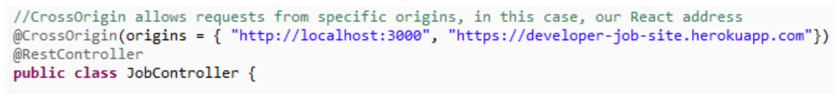
Figure 3.6: Spring Initializr Setup Page

3.3 Heroku

I mentioned earlier in the document I wanted my project to mimic what it would be like to develop a real website. Of course, my application can hardly hold a light to a user ready website we could find on the web at this moment, however, I did want it to be deployed in order to understand what is required to deploy a large application with multiple different technologies and tiers working together. With this in mind, I decided to use Heroku as I had some experience with it from the module 'Software Engineering' already this semester. Heroku is a container-based cloud Platform as a Service (PaaS). PAAS is a category of cloud computing that provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app [15]. Developers use Heroku to deploy, manage, and scale modern apps [16].
To get my application fully hosted on Heroku I had to do three thing;

1. Deploy my Spring Boot API

In order for the front end to make calls to the API and display or manipulate job records on the web, I had to deploy the API as a separate Heroku application. I initially had no real problems with this until i deployed the React app and tried to make requests from the new address. I had forgotten to add the deployed application's front end address to the list of addresses permitted access to the API in the 'JobController' class as well as the 'JWTAuthenticationController' class (Figure 3.3).



```
//CrossOrigin allows requests from specific origins, in this case, our React address
@CrossOrigin(origins = { "http://localhost:3000", "https://developer-job-site.herokuapp.com"})
@RestController
public class JobController {
```

Figure 3.7: Permitting Cross Origin Requests

2. Host my MongoDB Database on Atlas

This was a relatively simple step and has mostly been covered already in the previous section on Atlas. I had to have my database in the cloud so it would be available to my API from anywhere at all times.

3. Deploy my React App

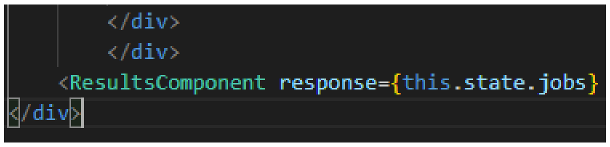
I created another Heroku application for the front end. This is the address that the user uses. As stated above, I had to add the front ends Heroku address to the Spring Boot controller classes in order to allow cross origin requests to the API (Figure 3.3). Once I did this the application was fully deployed.

3.4 React

React (ReactJs) is a Javascript library for building user interfaces. React can be used as a base in the development of single-page or mobile applications. However, React is only concerned with rendering data to the Document Object Model (DOM), and so creating React applications usually requires the use of additional libraries for state management and routing [17]. React Router is one such library I relied heavily upon for this application.
React uses a tool called 'Babel' to convert ES6 into a backwards compatible version of JavaScript that can be run by older JavaScript engines [18]. ES6 is a more modern version of Javascript that brings useful features to a project such as the use of the 'let' and 'const' keywords, the spread operator or even arrow functions all of which are present in my application.

3.4.1 Core Features

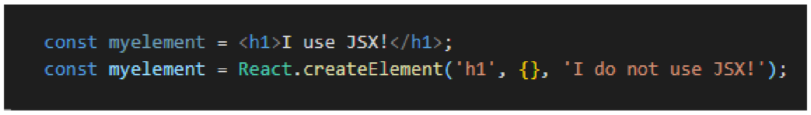
Its most notable core feature, is the use of components and props. Components can be rendered to a particular element in the DOM using the React DOM library. When rendering a component, one can pass in values that are known as "props" [19]. There are two types of components, 'functional components' and 'class components'. They differ in the fact that class components can hold 'state' values and can pass props to children (Figure 3.4.1). Components are extremely useful as you can reuse them as many times as you need. They essentially return a section of the user interface. A well written 'Reactful' application would have lots of components each fulfilling its own purpose and nothing else. This makes an application maintainable.



```
            </div>
            </div>
        <ResultsComponent response={this.state.jobs}
</div>
```

Figure 3.8: Passing a state value to 'ResultsComponent' as a prop called 'response'

Another notable features of React would be its use of 'Javascript-XML' (JSX). React uses JSX in place of HTML, it converts HTML tags into react elements. It is not required to use JSX in React although it is highly encouraged as it makes developing much easier. As demonstrated in (Figure 3.4.1) the first line of code uses JSX, it lets us write HTML directly within a Javascript statement. Whereas in the second line, without the use of JSX, we must call the 'createElement' method and specify the tag. It's much easier to use JSX.



```
const myelement = <h1>I use JSX!</h1>;
const myelement = React.createElement('h1', {}, 'I do not use JSX!');
```

Figure 3.9: The same output, one with JSX and one without

3.4.2 Creating React Application

Like I did for creating my Spring Boot application, I made use of a very popular tool to help me speed up or 'bootstrap' the process of getting a React application off the ground. 'Create-React-App' is an official environment creator provided by the creators of React (Facebook) to help developers get set up to code within minutes. This powerful tool gets an application setup with the right set of dependencies such as babel, react-dom and ES6 without the developer ever having to worry.
To use 'Create-React-App' one must have Node and Node-Package-Manager (NPM) installed. Then they simply run 'npx create-react-app ¡application name¿' and let their environment get setup.
Lastly, to run the react application in development mode we use the command 'npm start'. This runs the application on 'localhost:3000'.

3.5 JSON Web Tokens

As mentioned earlier, my application would need some form of authentication. For this, I used JSON Web Tokens (JWT). JWT is an open standard that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret or a public/private key pair [20], in regards to my application - a secret.

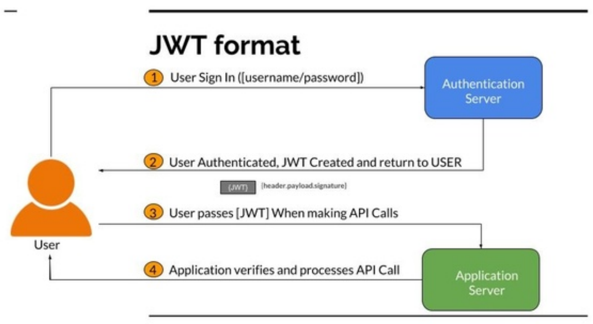

JWT format
1 User Sign In ([username/password])
Authentication Server
2 User Authenticated, JWT Created and return to USER
[JWT] [header.payload.signature]
3 User passes [JWT] When making API Calls
User
4 Application verifies and processes API Call
Application Server

Figure 3.10: JWT process

Specifically, JWT was used for authenticating and authorizing requests to and from my React application to my RESTful API. Below is a broad example of the steps that take place in my application when a user logs in:

- The user sends a POST request to the '/authenticate' endpoint with a username and password.
- The credentials are verified on the back end and a JWT is returned using a secret.
- The JWT and username are committed to the browser's session storage (not the password as it is sensitive data), this allows for verification on future requests.
- All subsequent API calls contain this JWT in their authorization header using 'axios interceptors'.
- Application verifies whether or not the user is logged in (via session storage) and the JWT and then processes the request.

3.6 Secondary Technologies

In this section I'll briefly analyze some of the secondary technologies and programs I used throughout the project that I found useful in development and think would be worthwhile mentioning.

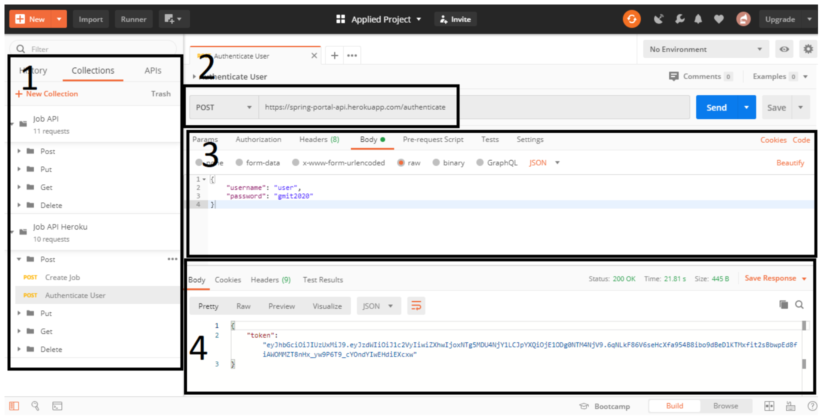
3.6.1 Postman

Postman is a fantastic collaboration platform for API development. It makes it easy for developers to test and document their API as well as save requests in collections and share them with peers. Users can send basic HTTP requests or configure complex requests detailing a range of different authentication types, as well as read their responses. It also lets you send and receive data in a number of different formats such as XML, JSON or even HTML.
Postman is much more pleasant than using a browser and displays data really well. Below (Figure 3.6.1) I have outlined some of the main components of my Postman environment:

1. This area displays the collections I created. Initially I just had one collection - 'Job API'. This grouped requests connecting to my locally hosted API, however after deploying the API to Heroku, I created a second collection for requests to that. Each collection groups requests into the 4 main HTTP request methods.
2. This is where you enter the address and endpoint of the API you wish to connect to as well as the HTTP request method type.

3. This is the request section. As shown below, I am making a POST request to the ’/authenticate’ endpoint, I’m passing it a username and password. If I was making a request to a different endpoint, I would include a JWT token in the ’Authorization’ tab.

4. This is the response section. As we can see, the server sent me back a JWT token. This area also shows the status of the request, in this case, ’200 OK’.



```
POST https://spring-portal-api.herokuapp.com/authenticate
```

```
{
    "username": "user",
    "password": "gmit2020"
}
```

Status: 200 OK Time: 21.81 s Size: 445 B

```
{
    "token":
        "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ1c2VyIiwiZXhwIjoxNTg5MDU4NjY1LCJpYXQiOjE1ODg0NTM4NjV9.6qNLkF86V6seHcXfa954B8ibo9dBeDlKTMxfit2sBbwpEd8f
        iAWOMMZT8nHx_yw9P6T9_cYOndYIwEHdiEXcxw"
}
```

Figure 3.11: Postman home screen

3.6.2 Jest & React Testing Library

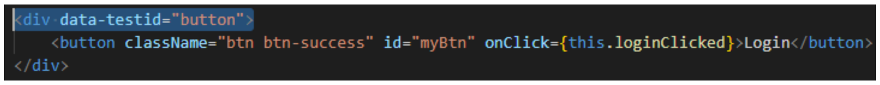
For testing my React application I used two frameworks, **Jest** and **React Testing Library**.

Jest

Jest is a testing library developed by Facebook that can be used for regular Javascript applications as well as React applications. It comes pre-installed with React applications that are built using **Create-React-App**. In my application, to use Jest alone, no pre configuring was required.

React Testing Library

The React Testing Library makes it easy to test React components. **react-testing-library** runs tests that use actual DOM nodes rather than instances of React components. It also exposes a recommended way to find elements by a ’data-testid’ (Figure 3.6.2) as an ”escape hatch” for elements where the text content and label do not make sense or is not practical [21].

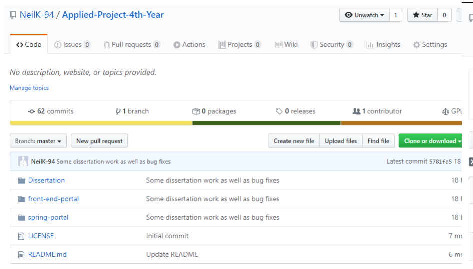


```
<div data-testid="button">
    <button className="btn btn-success" id="myBtn" onClick={this.loginClicked}>Login</button>
</div>
```

Figure 3.12: Identifying a button for testing using 'testid'

3.6.3 GitHub

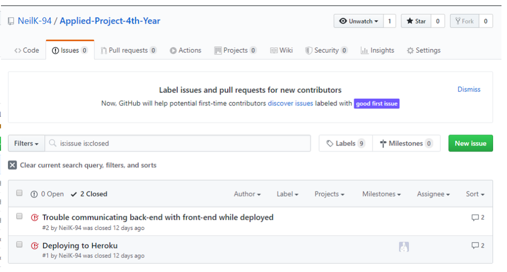
As touched on in the Methodology chapter, GitHub was used throughout development to provide a back-up of my work and to document consistent progress over the seven or eight months I have been planning and developing the application. It's also very useful for version control and makes it easy to revert any changes to a project if a mistake is made via the commit history.
As I had mentioned, I regret not using more of GitHub's features to their fullest such as the Wiki or the Issues tracker. I did make some entries to the 'Issues' tracker late on in the project when I realised it would've been beneficial (Figure 3.6.1). I documented my thoughts and progress mostly in Trello and a simple Word document but would've made better use of these features given a second run at the project.



No description, website, or topics provided.

NeilK-94 Some dissertation work as well as bug fixes — Latest commit 5781fa5 18

Dissertation	Some dissertation work as well as bug fixes	18
front-end-portal	Some dissertation work as well as bug fixes	18
spring-portal	Some dissertation work as well as bug fixes	18
LICENSE	Initial commit	7 m
README.md	Update README	6 m



Label issues and pull requests for new contributors
Now, GitHub will help potential first-time contributors discover issues labeled with good first issue

is:issue is:closed

Clear current search query, filters, and sorts

0 Open ✓ 2 Closed

Trouble communicating back-end with front-end while deployed — 2
#2 by NeilK-94 was closed 12 days ago

Deploying to Heroku — 2
#1 by NeilK-94 was closed 12 days ago

Figure 3.13: GitHub repository

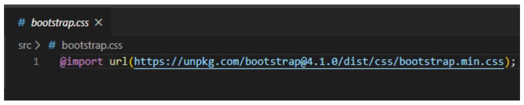
Figure 3.14: GitHub issues tracker

3.6.4 React Frameworks

Many different frameworks were used alongside the React library to develop the front end application, some of the most notable are the following:

1. Bootstrap

Bootstrap is an invaluable CSS framework which helps style web pages. I used it throughout my application and is used to style almost every component. I initially imported the Bootstrap CSS via the site 'UNPKG'. This is a useful content delivery site for everything on NPM. I simply added the URL to a 'bootstrap.css' file and then imported it into my app (Figure 3.6.4). Later on in development I added it via NPM also for easier use of features such as forms.

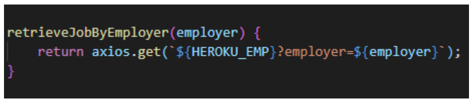


```
@import url(https://unpkg.com/bootstrap@4.1.0/dist/css/bootstrap.min.css);
```

Figure 3.15: Bootstrap CSS file

2. Axios

Axios is a promise based HTTP client used to perform HTTP requests on API's. This framework was used for all requests coming and going from my front-end application.

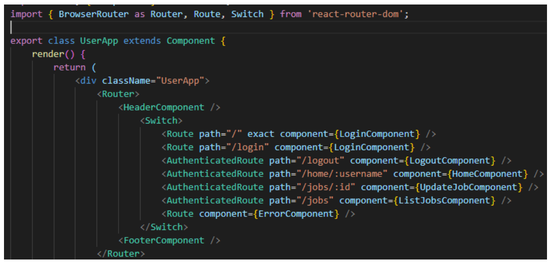


```
retrieveJobByEmployer(employer) {
    return axios.get(`${HEROKU_EMP}?employer=${employer}`);
}
```

Figure 3.16: An Axios request

3. React Router

The React Router framework handles all the routing between components in my application. React Router is a widely used framework for React and can be found in most React applications. I used this framework extensively in my 'UserApp' component which acts as the central hub of my application. From here I control which web-pages are shown to the user (Figure 3.6.4), it is also used for authenticating all routes.



```
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

export class UserApp extends Component {
    render() {
        return (
            <div className="UserApp">
                <Router>
                    <HeaderComponent />
                    <Switch>
                        <Route path="/" exact component={LoginComponent} />
                        <Route path="/login" component={LoginComponent} />
                        <AuthenticatedRoute path="/logout" component={LogoutComponent} />
                        <AuthenticatedRoute path="/home/:username" component={HomeComponent} />
                        <AuthenticatedRoute path="/jobs/:id" component={UpdateJobComponent} />
                        <AuthenticatedRoute path="/jobs" component={ListJobsComponent} />
                        <Route component={ErrorComponent} />
                    </Switch>
                    <FooterComponent />
                </Router>
```

Figure 3.17: Main component using Router

Chapter 4

System Design

4.1 Architecture

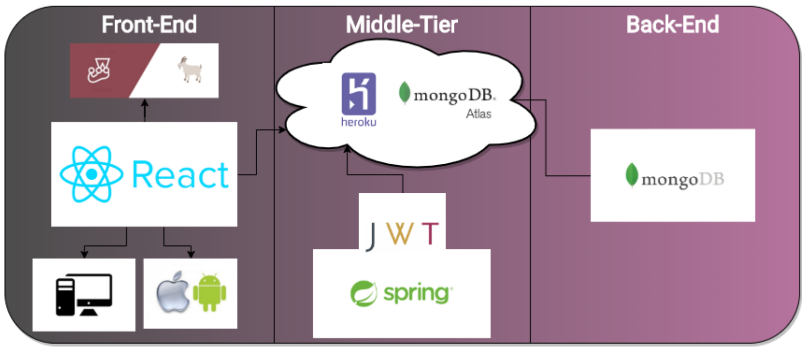


Figure 4.1: Project architecture diagram

The architecture (Figure 4.1) of this project is comprised of three different tiers.

1. The **Front-End** consists of the React application and of course all of the frameworks that go with it.
2. The **Middle-tier** consists of my Spring Boot server which configures the REST API endpoints as well as the JWT API. These API's are of course, also hosted on Heroku and are in use by the front-end application to make requests.
3. Lastly the **Back-End** which acts as a data tier includes a MongoDB database which stores all of the job details. I deployed the database on MongoDB Atlas to ensure the data is always available.

In the following sections I will detail the design for each tier and explain why I made the decisions I did.

4.2 Front End

As this section contains the user interface, the only part of the application that the end user will actually see, I will go into extensive detail on it and provide many screenshots of each page and feature as well as provide some code snippets on how each feature was implemented.

4.2.1 Login Page

The login page (Figure 4.2.1) is the first page the user sees when they use my application They cannot progress to any other pages in my application as all routes are authenticated. It consists of two text-boxes for entering user credentials, as well as a Bootstrap button for beginning the login sequence. If a user enters incorrect credentials they are alerted via a 'Bootstrap alert' Further down the page there is a 'jumbotron' for styling purposes.

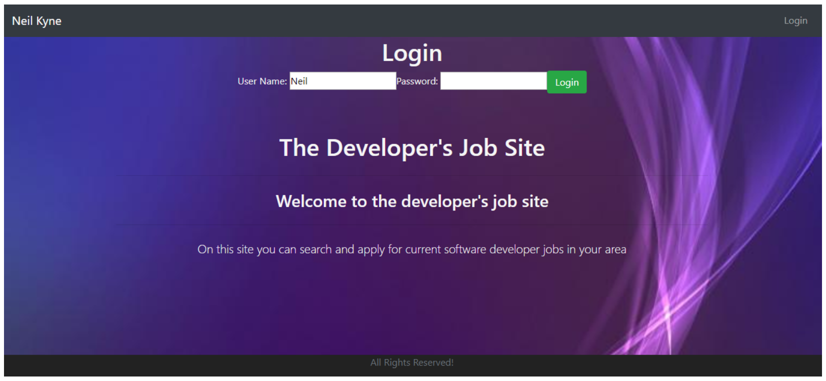

Neil Kyne
Login
Login
User Name:
Neil
Password:
Login
The Developer's Job Site
Welcome to the developer's job site
On this site you can search and apply for current software developer jobs in your area
All Rights Reserved!

Figure 4.2: Login page

4.2.2 Header & Footer

There is a navigation bar (Figure 4.2.2) at the top of every page in my application. This is useful for moving between pages and navigating the site. All routes in my application are authenticated meaning they cannot be accessed unless the user is logged in. Beginning from the left, it is comprised of;

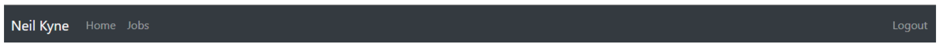


Figure 4.3: My 'navbar' at the top of every page

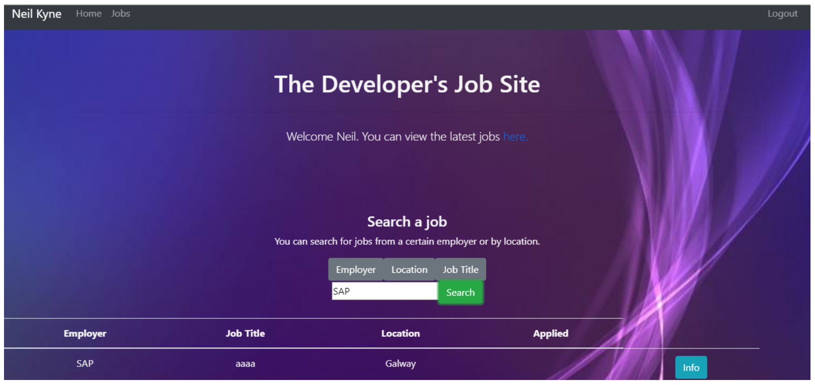
- 'Neil Kyne' - This is a clickable link which simply takes the user to my GitHub profile.

- 'Home' - This brings the user to the home page.
- 'Jobs' - This brings the user to the full list of jobs.
- 'Logout' - This signs a user out and returns them to the login page.

Finally, just like the header, there is a footer at the bottom of every web-page. This just contains a generic 'All Rights Reserved' statement.

4.2.3 Home Page

The home page (Figure 4.2.3) is the core of the front-end application. It consists of four components:



The Developer's Job Site

Welcome Neil. You can view the latest jobs here.

Search a job

You can search for jobs from a certain employer or by location.

Employer	Job Title	Location	Applied
SAP	aaaa	Galway	

Figure 4.4: Home page

1. Home Component

This component simply acts as a greeting to the user. It comprises of a 'jumbotron' containing a welcome message for the user as well as a link to the full list of jobs.

2. Search Component

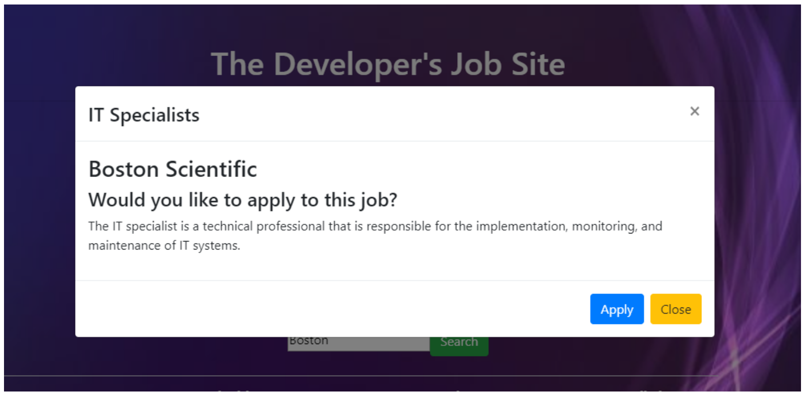
This component handles dynamic searching of jobs by their employer, location or job title.

3. Results Component

Here the results from the 'SearchComponent' are passed in as props to be displayed in a regular HTML table. Each row in the table has a 'View' button which opens up a Bootstrap 'Modal'.

Apply Component

This component (Figure 4.2.3) is a 'modal' that displays more details about the job as well as gives an option to the user to apply for the job. If the user applies for that job they are notified via a Bootstrap alert and a marker is left beside the job in the table.



Boston Scientific

Would you like to apply to this job?

The IT specialist is a technical professional that is responsible for the implementation, monitoring, and maintenance of IT systems.

Figure 4.5: Apply modal

4.2.4 Jobs Page

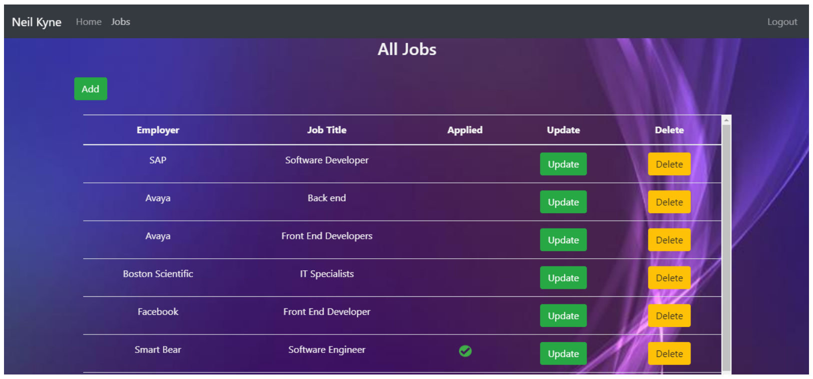
This page (Figure 4.2.4)contains a 'Add' button which brings the user to the 'UpdateJobComponent'. The page also contains a table listing all the jobs in the database as well as two buttons in every row;

- The **Delete button** removes a job listing from the database and refreshes the page.
- The **Update button** brings the user to the update job component.

4.2.5 Update/Create Job Page

This page (Figure 4.2.5) acts as a two in one. If the user clicks on the update button in the 'ListJobsComponent' then the job details are passed in as props to this child component and the user can edit them. However if the user clicks on the add button in the previous component it will display an empty form for the user to fill in job details.
The form containing the job details is a React framework called 'Formik'. This is useful package that simplifies handling user data in React.
Lastly, the page has two buttons;

- The **Save button** validates the form details, in my case it checks minimum character lengths in some of the text boxes. If they do not pass it displays

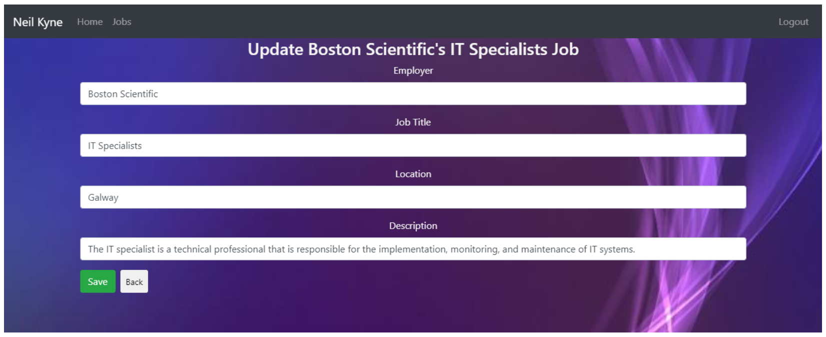


Employer	Job Title	Applied	Update	Delete
SAP	Software Developer		Update	Delete
Avaya	Back end		Update	Delete
Avaya	Front End Developers		Update	Delete
Boston Scientific	IT Specialists		Update	Delete
Facebook	Front End Developer		Update	Delete
Smart Bear	Software Engineer	✓	Update	Delete

Figure 4.6: Jobs page

a Bootstrap alert prompting the user to amend their mistake. If they pass validation it returns the user to the Jobs Page.

- The **Back button** returns the user to the previous page and discards any changes to the job.



Update Boston Scientific's IT Specialists Job

Employer: Boston Scientific

Job Title: IT Specialists

Location: Galway

Description: The IT specialist is a technical professional that is responsible for the implementation, monitoring, and maintenance of IT systems.

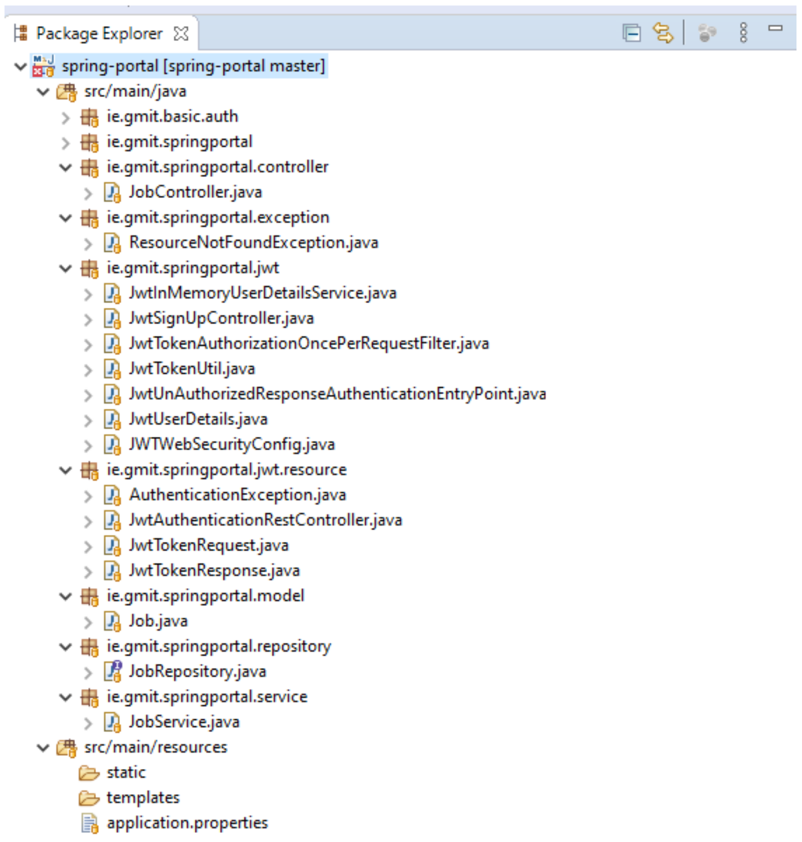
Figure 4.7: Update page

4.3 Middle Tier

This section consists of a lot of the logic and actual computation in my application. The most important part of this tier is of course the REST API. This is what the front-end communicates with to manipulate or remove data. This tier also contains a JWT API which allows users to request new tokens and prove authentication. Both of these API's are hosted on Heroku.

4.3.1 Spring Boot Project Structure

The REST API hosted on Heroku was built using Spring Boot. In this section I will summarize my project setup (Figure 4.3.1) and design for building the API.



```
spring-portal [spring-portal master]
  src/main/java
    ie.gmit.basic.auth
    ie.gmit.springportal
    ie.gmit.springportal.controller
      JobController.java
    ie.gmit.springportal.exception
      ResourceNotFoundException.java
    ie.gmit.springportal.jwt
      JwtInMemoryUserDetailsService.java
      JwtSignUpController.java
      JwtTokenAuthorizationOncePerRequestFilter.java
      JwtTokenUtil.java
      JwtUnAuthorizedResponseAuthenticationEntryPoint.java
      JwtUserDetails.java
      JWTWebSecurityConfig.java
    ie.gmit.springportal.jwt.resource
      AuthenticationException.java
      JwtAuthenticationRestController.java
      JwtTokenRequest.java
      JwtTokenResponse.java
    ie.gmit.springportal.model
      Job.java
    ie.gmit.springportal.repository
      JobRepository.java
    ie.gmit.springportal.service
      JobService.java
  src/main/resources
    static
    templates
    application.properties
```

Figure 4.8: Project structure

My Boot project consists of four main packages;

- The **Model** package describes the model that we want to work with, in this case a job. This is what is persisted to the MongoDB database and what is displayed at our front-end.
- The **Repository** package configures the MongoDB repository. The 'JobRepository' is used for accessing data from the database. It automatically implements all the basic CRUD operations one might want to perform on a database, I also added three custom methods for searching (Figure 4.3.1). Spring Boot tries to auto-configure most of the stuff for you based on the

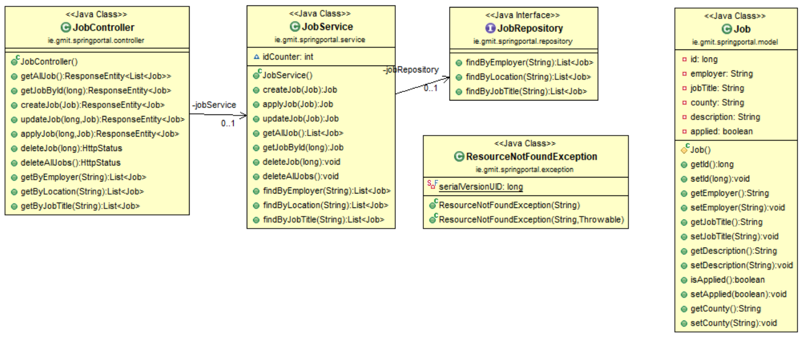
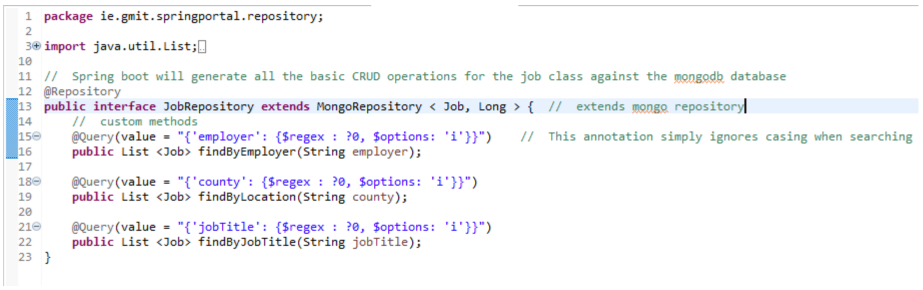

<<Java Class>>
JobController
ie.gmit.springportal.controller
JobController()
getAllJob():ResponseEntity<List<Job>>
getJobById(long):ResponseEntity<Job>
createJob(Job):ResponseEntity<Job>
updateJob(long,Job):ResponseEntity<Job>
applyJob(long,Job):ResponseEntity<Job>
deleteJob(long):HttpStatus
deleteAllJobs():HttpStatus
getByEmployer(String):List<Job>
getByLocation(String):List<Job>
getByJobTitle(String):List<Job>
-jobService
0..1
<<Java Class>>
JobService
ie.gmit.springportal.service
idCounter: int
JobService()
createJob(Job):Job
applyJob(Job):Job
updateJob(Job):Job
getAllJob():List<Job>
getJobById(long):Job
deleteJob(long):void
deleteAllJobs():void
findByEmployer(String):List<Job>
findByLocation(String):List<Job>
findByJobTitle(String):List<Job>
-jobRepository
0..1
<<Java Interface>>
JobRepository
ie.gmit.springportal.repository
findByEmployer(String):List<Job>
findByLocation(String):List<Job>
findByJobTitle(String):List<Job>
<<Java Class>>
ResourceNotFoundException
ie.gmit.springportal.exception
serialVersionUID: long
ResourceNotFoundException(String)
ResourceNotFoundException(String,Throwable)
<<Java Class>>
Job
ie.gmit.springportal.model
id: long
employer: String
jobTitle: String
county: String
description: String
applied: boolean
Job()
getId():long
setId(long):void
getEmployer():String
setEmployer(String):void
getJobTitle():String
setJobTitle(String):void
getDescription():String
setDescription(String):void
isApplied():boolean
setApplied(boolean):void
getCounty():String
setCounty(String):void

Figure 4.9: UML diagram of my API

dependencies that you have added in the pom.xml file. Since I added a 'spring-boot-starter-data-mongodb' dependency, Spring Boot automatically tries to build a connection with MongoDB by reading the database configuration from the 'application.properties' file.



```
package ie.gmit.springportal.repository;

import java.util.List;

//  Spring boot will generate all the basic CRUD operations for the job class against the mongodb database
@Repository
public interface JobRepository extends MongoRepository < Job, Long > {  //  extends mongo repository
    //  custom methods
    @Query(value = "{'employer': {$regex : ?0, $options: 'i'}}")    //  This annotation simply ignores casing when searching
    public List <Job> findByEmployer(String employer);

    @Query(value = "{'county': {$regex : ?0, $options: 'i'}}")
    public List <Job> findByLocation(String county);

    @Query(value = "{'jobTitle': {$regex : ?0, $options: 'i'}}")
    public List <Job> findByJobTitle(String jobTitle);
}
```

Figure 4.10: Mongo repository

- The **Service** package describes the service layer. This package contains a lot of the business logic. The 'JobService' class connects with the 'JobRepository' class to perform CRUD operations on the database.
- Finally the **Controller** package contains the 'JobController' class. This is the API which will be exposed to the front-end users.

Aside from the listed packages there are;

- A **Basic Authentication** package which is no longer in use as I now use JWT authentication.
- An **Exception** package for handling exceptions or errors with requests to the API.

- Two **JWT** packages.
  - The resource package contains the resources related to the REST API and most importantly the 'JWTAuthenticationRestController'. This is the API we send requests to to receive JWT tokens.
  - The JWT package contains all of the actual logic for creating the JWT token and the user details required to do so.

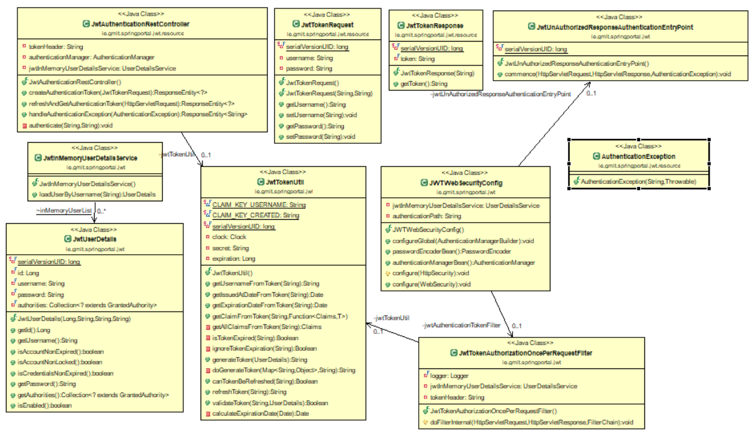

<<Java Class>>
JwtAuthenticationRestController
ie.gmit.springportal.jwt.resource
tokenHeader: String
authenticationManager: AuthenticationManager
jwtInMemoryUserDetailsService: UserDetailsService
JwtAuthenticationRestController()
createAuthenticationToken(JwtTokenRequest):ResponseEntity<?>
refreshAndGetAuthenticationToken(HttpServletRequest):ResponseEntity<?>
handleAuthenticationException(AuthenticationException):ResponseEntity<String>
authenticate(String,String):void
<<Java Class>>
JwtTokenRequest
ie.gmit.springportal.jwt.resource
serialVersionUID: long
username: String
password: String
JwtTokenRequest()
JwtTokenRequest(String,String)
getUsername():String
setUsername(String):void
getPassword():String
setPassword(String):void
<<Java Class>>
JwtTokenResponse
ie.gmit.springportal.jwt.resource
serialVersionUID: long
token: String
JwtTokenResponse(String)
getToken():String
<<Java Class>>
JwtUnAuthorizedResponseAuthenticationEntryPoint
ie.gmit.springportal.jwt
serialVersionUID: long
JwtUnAuthorizedResponseAuthenticationEntryPoint()
commence(HttpServletRequest,HttpServletResponse,AuthenticationException):void
-jwtUnAuthorizedResponseAuthenticationEntryPoint
0..1
<<Java Class>>
AuthenticationException
ie.gmit.springportal.jwt.resource
AuthenticationException(String,Throwable)
<<Java Class>>
JwtInMemoryUserDetailsService
ie.gmit.springportal.jwt
JwtInMemoryUserDetailsService()
loadUserByUsername(String):UserDetails
-jwtTokenUtil
0..1
~inMemoryUserList
0..*
<<Java Class>>
JwtUserDetails
ie.gmit.springportal.jwt
serialVersionUID: long
id: Long
username: String
password: String
authorities: Collection<? extends GrantedAuthority>
JwtUserDetails(Long,String,String,String)
getId():Long
getUsername():String
isAccountNonExpired():boolean
isAccountNonLocked():boolean
isCredentialsNonExpired():boolean
getPassword():String
getAuthorities():Collection<? extends GrantedAuthority>
isEnabled():boolean
<<Java Class>>
JwtTokenUtil
ie.gmit.springportal.jwt
CLAIM_KEY_USERNAME: String
CLAIM_KEY_CREATED: String
serialVersionUID: long
clock: Clock
secret: String
expiration: Long
JwtTokenUtil()
getUsernameFromToken(String):String
getIssuedAtDateFromToken(String):Date
getExpirationDateFromToken(String):Date
getClaimFromToken(String,Function<Claims,T>)
getAllClaimsFromToken(String):Claims
isTokenExpired(String):Boolean
ignoreTokenExpiration(String):Boolean
generateToken(UserDetails):String
doGenerateToken(Map<String,Object>,String):String
canTokenBeRefreshed(String):Boolean
refreshToken(String):String
validateToken(String,UserDetails):Boolean
calculateExpirationDate(Date):Date
<<Java Class>>
JWTWebSecurityConfig
ie.gmit.springportal.jwt
jwtInMemoryUserDetailsService: UserDetailsService
authenticationPath: String
JWTWebSecurityConfig()
configureGlobal(AuthenticationManagerBuilder):void
passwordEncoderBean():PasswordEncoder
authenticationManagerBean():AuthenticationManager
configure(HttpSecurity):void
configure(WebSecurity):void
-jwtTokenUtil
0..1
-jwtAuthenticationTokenFilter
0..1
<<Java Class>>
JwtTokenAuthorizationOncePerRequestFilter
ie.gmit.springportal.jwt
logger: Logger
jwtInMemoryUserDetailsService: UserDetailsService
tokenHeader: String
JwtTokenAuthorizationOncePerRequestFilter()
doFilterInternal(HttpServletRequest,HttpServletResponse,FilterChain):void

Figure 4.11: UML diagram for JWT packages

- The **pom.xml** file contains project information and configuration information for maven to build the project such as dependencies, build directory and source directories. Maven reads the pom. xml file, then executes the goal [22]. Here (Figure 4.3.1) I specified many dependencies to aid in building the API.
- Lastly, the **application.properties** file is used to keep project properties in a single file to run the application in a different environment.

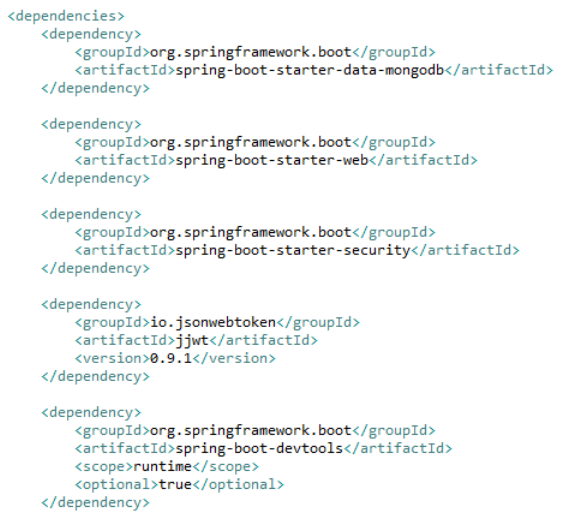
4.3.2 Heroku

In order for the React application to be able to access the API's online they would have to be hosted online. For this I used Heroku. To create an empty Heroku application via the Heroku Command Line Interface (CLI) I ran:

```
heroku create spring-portal-api
```

To deploy the REST API to Heroku I first initialized a Git repository in my 'spring-portal' directory.

```
git init
```



```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-mongodb</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>

    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.9.1</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
```

Figure 4.12: pom.xml file declaring dependencies

I then update the index using the current content found in the working tree, to prepare the content staged for the next commit.

```
git add .
```

This stores the current content of the index in a new commit along with a message from the user describing the changes.

```
git commit -m "Added API to Heroku"
```

Finally, to push to the new Heroku repository you use:

```
git push heroku master
```

4.4 Back-End

The back-end tier stores all of my job data. I deployed a my database online using MongoDB Atlas. This makes it available at all times to the API.

4.4.1 MongoDB Atlas

There were a number of steps required to deploy my database on Atlas.

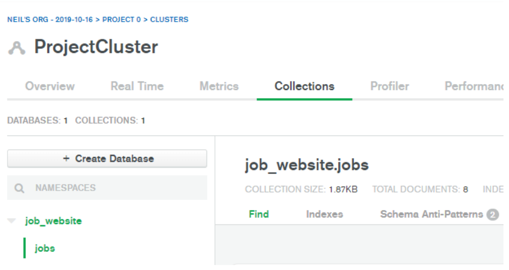
1. Create a MongoDB Atlas account.
2. Create a free tier cluster by clicking ”Build a new cluster”. From here choose a cloud provider and region. Finally Choose the free tier cluster option, selecting the M0 sandbox (free tier) and name your cluster.

- Atlas Free Tier clusters provide a small-scale development environment to host your data. Free Tier clusters never expire, and provide access to a subset of Atlas features and functionality [23].

3. Next, when the cluster is created, click 'Connect'. Here we white-list our IP address. Secondly, add yourself as a database user admin to allow for full access.

4. Now we are ready to connect the cluster to the Spring Boot project. Click connect, configure the project settings and copy the string (Figure 4.4.1).

5. Head over to the 'application.properties' file and paste in the string, replacing the blank username and password with your own.



DATABASES: 1 COLLECTIONS: 1

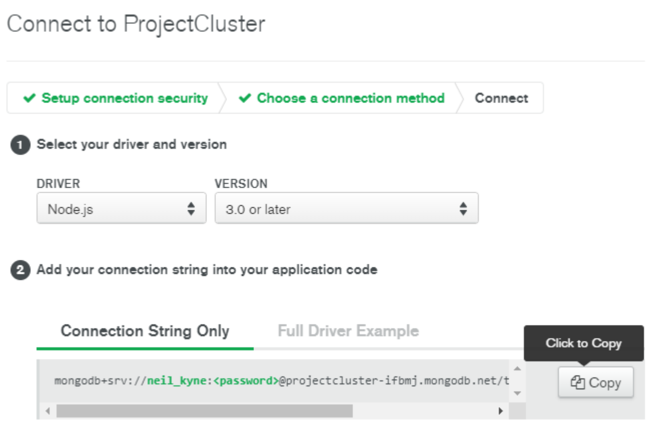
job_website.jobs

COLLECTION SIZE: 1.87KB TOTAL DOCUMENTS: 8 INDE

job_website

jobs

Figure 4.13: Database with its 'jobs' collection


Connect to ProjectCluster
✓ Setup connection security
✓ Choose a connection method
Connect
Connection String Only
Full Driver Example
Click to Copy
Copy

1 Select your driver and version

DRIVER: Node.js

VERSION: 3.0 or later

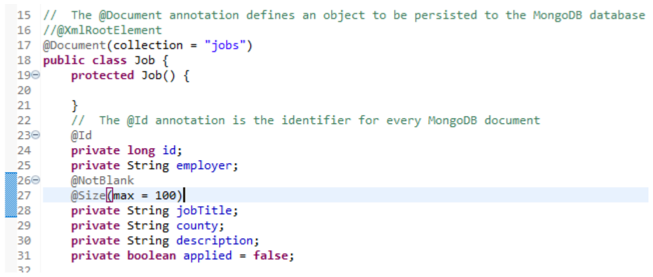
2 Add your connection string into your application code

```
mongodb+srv://neil_kyne:<password>@projectcluster-ifbmj.mongodb.net/t
```

Figure 4.14: Application connection string

It may have been noticed I never specified a name for the database or a name for the collection anywhere on MongoDB Atlas. All of this is configured in my Spring Boot application. I specify these details using the Spring data annotations provided for use with MongoDB. I also specify a few other settings for the

'Job' documents using these annotations (Figure 4.4.1) such as the **@Id** of the document which is the identifier for every mongo document, as well as more basic annotations such as **@NotBlank** which forbids empty values and **@Size** which allows a user to set a max input size.
I define the collection name in the database above the class declaration with **@Document(collection = "jobs")**. Finally the database name is declared in the 'application.properties' file with
spring.data.mongodb.database=job_website.



```
//  The @Document annotation defines an object to be persisted to the MongoDB database
//@XmlRootElement
@Document(collection = "jobs")
public class Job {
    protected Job() {

    }
    //  The @Id annotation is the identifier for every MongoDB document
    @Id
    private long id;
    private String employer;
    @NotBlank
    @Size(max = 100)
    private String jobTitle;
    private String county;
    private String description;
    private boolean applied = false;
```

Figure 4.15: Job model outlining some mongo configurations using Spring-Mongo annotations.

Chapter 5

System Evaluation

In this section I will look to give a critical analysis of my application, I will compare the objectives I set out at the beginning of the project with what I actually managed to achieve. I will also call attention to the shortcomings in the application as well as areas I feel could be improved upon.

5.1 Initial Objectives

In my introduction I set out a number of objectives I wanted to accomplish with this project. Here I will examine each one and determine whether or not I feel I accomplished them sufficiently.

Objective 1

My first objective was to build an application 'with a number of moving parts'. This essentially means I wanted it to be a full stack application. A full stack developer is a person who can develop both client and server software [24]. I really wanted to do this because, as I mentioned earlier on in the document, I felt it would be important to experience what it's like to build a real, genuine web application from top to bottom. I feel I achieved this objective as my application covers every 'stack'.

Objective 2

My second objective was to become 'very familiar' with React. Again I do feel that I have accomplished this objective. I would say technology wise, the project consists of about 60% React/Javascript code while the rest is made up of Java and a few others. I had wanted to become familiar with React as it is an ever increasing language in terms of popularity and I wish to focus on web development once I graduate. I'm far from an expert with React having finished the application but I will say I feel I developed genuine knowledge on the technology that I can use as a base to build on in the future.

Objective 3

My third objective was for the application to have authentication. I decided ideally, the application would have JSON web tokens but failing that would use basic authentication. Thankfully, I managed to implement JSON web tokens. I initially implemented basic authentication and from there moved on to JSON web tokens.

Objective 4

My fourth objective was for the user to be able to perform the basic CRUD functions on the API. This was one of the first things I achieved and added to by implementing custom searches.

Objective 5

Finally, my fifth and final objective was to get the entire application deployed and running on Heroku. This was the last objective I managed and had a reasonable degree of difficulty in achieving. It involved deploying the back-end API and the front-end application as two separate Heroku applications. It was also required for my database to be deployed on MongoDB Atlas to ensure the data was available at all times to the API.

5.2 Limitations & Improvements

Sadly, my project has its limitations. Due to a mix of adverse time management and to a degree, lack of knowledge of the technologies used, my application has many areas I feel could be improved on. Hindsight is a wonderful thing, given what I know know there are plenty of things I would do differently if given the chance to go back and start again.

5.2.1 Sign Up Functionality

One of my biggest regrets with the project is that I never got around to implementing a sign up option. This wouldn't have been too difficult to do but in the end I ran out of time. When I first implemented JWT I hard-coded two user profiles into the user details service. I kept putting off adding the ability to create new user profiles in favor of more pressing work and in the end never implemented it at all.
To do this, I would have needed to create new 'Controller', 'Model' and 'Service' classes for user sign-ups around the 'JWTInMemoryUserDetailsService'. I would then have needed to simply update the 'JWTWebSecurityConfig' class to allow calls to a '/signup' endpoint without authentication.
Once I had that working on the back-end I would have added a sign-up option to the login page which would use Axios to send a request to the '/signup'

endpoint, passing in the new username and password. This would've been a nice addition to the login page as it is a bit bare without it.

5.2.2 More Complex API

With this application I knew I wanted to focus slightly more on the front-end than the back. As a result, I feel my REST API was a little neglected in terms of features. Looking back, I would definitely have liked to add more complex features to my API such as searching within specific constraints (like only searching for jobs in multiple different locations).
I feel the actual job details could've been extended upon and included contact details, salaries and experience requirements just to name a few.

5.2.3 More Front-End Features

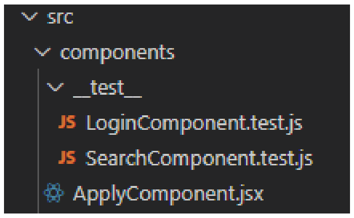
Finally, I would've liked to have added more 'bells & whistles' with the front-end of my application. There are many really interesting frameworks that can make React applications more user friendly and outside of purely functional frameworks such as Axios, I feel I only really used React Bootstrap to make the visual side of the application more appealing. I suppose using the knowledge gained from this project learning React, I will be more capable of adding such features to web applications in the future.

5.2.4 Much More Testing

Throughout development I kept telling myself I would get round to implementing detailed testing when the application was finished. This was rather naive of me as it is well known in modern software development that developers are expected to run their own tests as they go and not rely on testing teams. This was even covered in one of our modules and I majorly regret not heeding this advice.

5.3 Testing

I incorporated a small handful of test in my front-end application using Jest and React-Testing-Library. I mainly tested UI elements of the application and only on two components, the 'LoginComponent' and the 'SearchComponent'.
By default **Jest** expects to find files to test in a '__test__' folder (Figure 5.3). These files must end with '.test.js'. Ideally if there were tests being done on every component in my application, it might be useful to group every component into their own sub folder along with their test file and perhaps their styling file. This would keep the project directory clean and tidy and is something I will look to do in future projects.

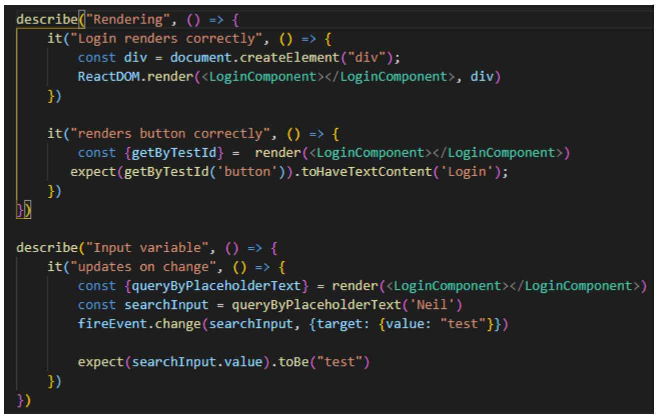


```
src
  components
    __test__
      LoginComponent.test.js
      SearchComponent.test.js
    ApplyComponent.jsx
```

Figure 5.1: Jest required project layout

LoginComponent Tests

In the 'LoginComponent' I run three tests, the first two are contained in the first 'describe'. **Describe** is a function name, along with **it** in a lot of testing framework. It essentially groups together tests into suites.
The first test suite checks to see first, that our component renders correctly, and then to see that out login button renders correctly, with the correct content inside it(Login)
The second tests to ensure that the input box containing the placeholder text, 'Neil' updates on change. It passes the value 'test' into it using a **fireEvent**. fireEvents are useful for performing an action in a test. Finally, we use the **expect** function to tell our test what value to expect the searchInput value to be.



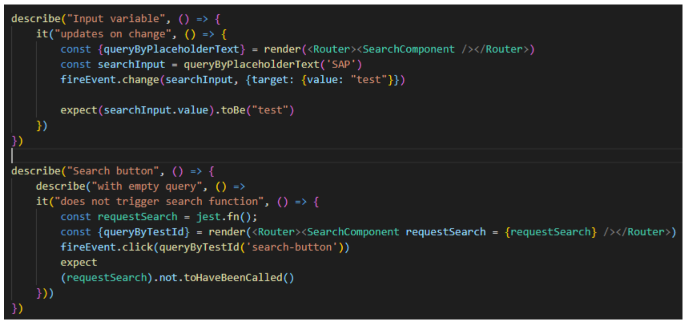
```
describe("Rendering", () => {
    it("Login renders correctly", () => {
        const div = document.createElement("div");
        ReactDOM.render(<LoginComponent></LoginComponent>, div)
    })

    it("renders button correctly", () => {
        const {getByTestId} =  render(<LoginComponent></LoginComponent>)
       expect(getByTestId('button')).toHaveTextContent('Login');
    })
})

describe("Input variable", () => {
    it("updates on change", () => {
        const {queryByPlaceholderText} = render(<LoginComponent></LoginComponent>)
        const searchInput = queryByPlaceholderText('Neil')
        fireEvent.change(searchInput, {target: {value: "test"}})

        expect(searchInput.value).toBe("test")
    })
})
```

Figure 5.2: LoginComponent tests



```
describe("Input variable", () => {
    it("updates on change", () => {
        const {queryByPlaceholderText} = render(<Router><SearchComponent /></Router>)
        const searchInput = queryByPlaceholderText('SAP')
        fireEvent.change(searchInput, {target: {value: "test"}})

        expect(searchInput.value).toBe("test")
    })
})

describe("Search button", () => {
    describe("with empty query", () =>
    it("does not trigger search function", () => {
        const requestSearch = jest.fn();
        const {queryByTestId} = render(<Router><SearchComponent requestSearch = {requestSearch} /></Router>)
        fireEvent.click(queryByTestId('search-button'))
        expect
        (requestSearch).not.toHaveBeenCalled()
    }))
})
```

Figure 5.3: Some SearchComponent tests

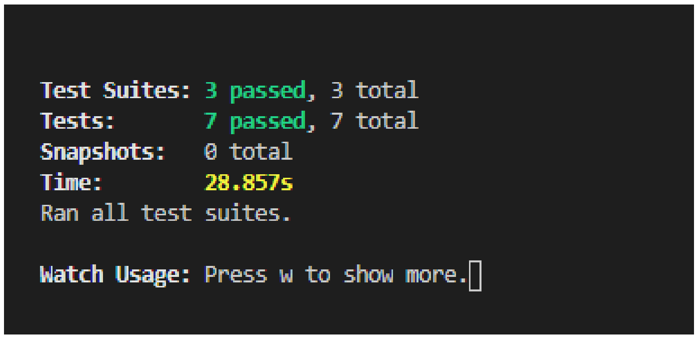
SearchComponent

In the 'SearchComponent' test file I perform three main tests as well as a test to check that the component renders correctly.
The first test, like in the 'LoginComponent' test file, checks to ensure that the 'textBox' value changes when we pass in a test value.
Next I check to ensure that the 'searchButton' when clicked without a query given, does not trigger a search. To do this I use a Jest Mock which is a function that essentially mimics the behaviour of an object. This is then passed in as a prop to the SearchComponent.
Finally, I reverse the previous test to check that when there is data given in a query, and the user clicks the 'searchButton' it would trigger a search.

Running The Tests

Running the tests is simple. As Jest comes pre-configured with 'create-react-app' you need only run:

```
npm test
```



```
Test Suites: 3 passed, 3 total
Tests:       7 passed, 7 total
Snapshots:   0 total
Time:        28.857s
Ran all test suites.

Watch Usage: Press w to show more.
```

Figure 5.4: Test results

Chapter 6

Conclusion

6.1 Project Outcomes

When I began this project I had a reasonably clear idea of what I wanted the project to be;

- A job site aimed specifically at software developers which hosted software development jobs.
- The user should be able to log in securely using a username and password.
- The user should be able to search for jobs by a number of different methods.
- The user should be able to post their own jobs and delete or update them.
- The front end website should be visually appealing with a handful of neat features.
- The API should be RESTful.
- The application should be hosted online and accessible at all times.
- The job data should be deployed on a cloud database.
- There should be tests verifying the integrity and solidity of my front-end application.

I feel I achieved most of the goals I had when I set out with this project. Of course, there is plenty of limitations and room for improvements, however I do feel that I have gained a lot from the project and if I was to go back and start it again, would still choose the technologies that I did.
I created what I believe to be, a decently sized project for a one person group. I believe in covering the different parts of the stack I was exposed to a number of different technologies and thus gained valuable experience in full stack development. It was very important for me from the beginning that it was a full stack application and thankfully I achieved that.

6.1.1 My Findings

I learned a number of things in undertaking this project and will review some below:

- React is a very enjoyable technology to use and I will certainly look to improve upon it further in the future.
- Undertaking a full stack application solidified my desire to work with web applications in the future.
- It is very important to adequately plan out a project of this scope and hold yourself to deadlines and targets.
- Testing should be done alongside development to ensure bugs are caught early and code is clean and reusable.
- Project management is a very important skill to have when working with multiple different projects. It is essential that you dedicate time every week to a project, even if other ones are more pressing. Even spending an hour or two a week on a project that isn't as pressing as another at the time, you ensure you retain key information and continue learning about the technology.

6.2 Summary

In conclusion I must say that overall I have genuinely found the project to be very rewarding. It helped me to gain a decent grasp on a number of new technologies as well as forced me to learn how to manage a larger project than I'm used to doing over a large time-frame. I was forced to comprehensively research multiple technologies and find pros and cons to using each one. This project also demanded I set out a clear plan of action for what I wanted to achieve and within what time-frame.
I think the main goals of this module were to learn how to manage a demanding project over a large period of time and to showcase the ability to set out and learn new technologies without being taught them by a lecturer in class. I do believe I have achieved both of these and had a lot of fun along the way. This project has confirmed for me what I want to do after college and I'm very glad I made the decisions I did. Naturally, not everything went perfectly as I would have liked but the shortcomings I faced have taught me a lot going forward.

Chapter 7

Appendices

7.1 GitHub Repository

https://github.com/NeilK-94/Applied-Project-4th-Year

Bibliography

[1] Varonis. - What Is OAuth: https://www.varonis.com/blog/what-is-oauth/.

[2] Wikipedia. MEAN: https://en.wikipedia.org/wiki/MEAN_(solution_stack).

[3] DropWizard. - Home: https://www.dropwizard.io/en/latest/.

[4] HowToDoInJava. - Spring Boot: https://howtodoinjava.com/spring-boot-tutorials/.

[5] Zachary Parker, Scott Poe, and Susan V Vrbsky. Comparing nosql mongodb to an sql db. In *Proceedings of the 51st ACM Southeast Conference*, pages 1–6, 2013.

[6] Francesco Marchioni. *MongoDB for Java developers*. Packt Publishing Ltd, 2015.

[7] Wikipedia. - React: https://en.wikipedia.org/wiki/React$_{(}web_{f}ramework)$.

[8] Stack overflow developers survey: https://insights.stackoverflow.com/survey/2019.

[9] Wikipedia. - Trello: https://en.wikipedia.org/w/index.php?title=Trellooldid=945177792.

[10] MongoDB. -Document Description: https://docs.mongodb.com/getting-started/cpp/documents/.

[11] Wikipedia. NoSQL: https://en.wikipedia.org/w/index.php?title=NoSQLoldid=952294228.

[12] MongoDB. - Cluster Description: https://docs.atlas.mongodb.com/cluster-configuration/.

[13] DZone. - Understanding basics of Spring vs Spring Boot: https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot.

[14] Spring. -Spring Boot: https://spring.io/projects/spring-boot.

[15] Wikipedia. PAAS: https://en.wikipedia.org/wiki/Platform$_{a}s_{a}{}_{s}ervice$.

[16] MongoDB. Heroku About: https://www.heroku.com/about.

[17] Wikipedia. React: https://en.wikipedia.org/wiki/React$_{(}web_{f}ramework$.

[18] Babel. - Homepage: https://babeljs.io/docs/en/.

[19] ReactJS. - Components and props: https://reactjs.org/docs/components-and-props.htmlprops-are-read-only.

[20] JWT. - What is JWT: https://jwt.io/introduction/.

[21] React-Testing-Library. - Intro: https://testing-library.com/docs/react-testing-library/intro.

[22] JavatPoint. Maven - POM.XML: https://www.javatpoint.com/maven-pom-xml.

[23] MongoDB. - Docs, Deploy a cluster: https://docs.atlas.mongodb.com/tutorial/deploy-free-tier-cluster/.

[24] W3. What is a full stack developer: https://www.w3schools.com/whatis/whatis$_{f}ullstack.asp$.