Final Year Project 2017: Retail Management System.

**Languages:** JavaScript,TypeScript,HTML,CSS.

**Libraries:** Bootstrap, Mongoose.

**Frameworks:** Angular 2.0, ExpressJs.

**Databases:** MongoDB.

**Environments:** Node.Js.

**NPMS:** Mongoose, mongoose-unique-validator.

**Package Managers:** Bower.

**Project Overview**

I have decided that I will create a management system for regional managers of retail stores. The first time I thought of creating a program like this was during summer of 2016 before the start of the semester, I was working in a computer store part time and the regional manager at the time was working beside me as I was covering the front of the store, he started showing me the system the company used to track how each store was doing that day. The system was extremely old and dull to look at, plus there was some problems with the functionality of the GUI.

This inspired me to make a system for managing stores and staff that was not only nicer to look at but took full advantage of the modern technology that I felt the system was missing out on and would gain so much from. After talking to my project supervisor, he informed be about the MEAN stack and how it is currently a combination of technologies that are on the rise. I had used Angular before but after researching Angular 2 I thought it would be worth learning from scratch and decided to challenge myself by attempting to build a front and back end for my project out of 4 new technologies.

After choosing what I will use to create the system I began to set some goals that I wanted to achieve, I had to make sure these where realistic as the first task of this project would be time management and planning since I wasn’t in a group and had a scope of 7-8 weeks. After some careful planning, I decide I wanted to implement the follow features.

* Aesthetic GUI
* CRUD and Restful API
* Use an API from an online source
* Efficient database with no duplicates.
* Support page for users.

**Understanding the technologies**

The design pattern I decided to use was the MEAN stack, which stands for mongodb, expressjs, angular and nodejs. To create my project, I had to first research what each of these are and why they are used so frequently together.

**MEAN Stack:**

**Mongodb:** Mongo is a NoSQL database which can be installed rather simply onto your local machine. Instead of using tables this database used collections in a BSON format, which is a binary version of JSON.

**ExpressJs:** Express is a nodejs framework which allows Javascript to behave like a server side language and interact with the mongo database. Expressjs manages the routes and requests of the MEAN. You can use a variety of templates for the express engine but the one used in this project is known as EJS.

**Angular2:** Normally Angular 1 is what is used in the MEAN stack but you can also use Googles new Angular 2 which is what I decided to do. Angular 2 is front end web app platform which is written is Typescript, it has all the benefits of Aangular1 but now everything is divided into components and services, this makes the code mush easier to manage and it increases in size and makes reusability easy.

**NodeJs:** Node.js is a server-side platform built on Google Chrome's JavaScript Engine V8. It is scalable and fast allowing users to make complex web application in a short amount of time. Nodejs provides an amazing selection of open source libraries which can be set up using the command line package manager npm.

**Other technologies:**

**Bootstrap:** This is a framework that I will be using to create a lot of the front end, I have used bootstrap in previous projects and think it’s an extremely useful tool when creating an aesthetic GUI, which is one of my goals for this project. Bootstrap also is really reliable In making the GUI functional and fast allowing me to focus more time on the backend and the new technologies I haven’t used yet in this project.

**Mongoose:** This is a database modelling tool which is designed to work in an asynchronous environment. Mongoose is installed using npm and then connected to mongodb. Mongoose makes the designing of the database much easier to keep track of as you can keep models of mongoose collection data inside your application to work with.

**Git:** This is a version control system that lets you keep track of your work and save each version as checkpoints that can be pulled from Github.

**Setting up the project**

Before I started designing my pages and the functionality of the app I made the design decision to set up the basics of a MEAN stack app. The reasoning behind this was since this was my first time using these 4 technologies I wanted to ensure everything was set up correctly and that I could understand the flow of the overall project before I started making the features and GUI. I didn’t want to make any assumptions on what was and wasn’t possible with the MEAN stack so I set up the bare bones before I started planning. I used the following tutorial to get understanding of how to properly set up a MEAN stack: <https://www.youtube.com/watch?v=XxNZMI5V3t4>

.

**NodeJs:** Since NodeJs is the core of the mean stack and what I will use to not only host the project but also install other parts as packages I installed that first. NodeJs is installed by going to <https://nodejs.org/en/> and choosing a version. The version I decided to use was 6.10.0 as that is the most stable version currently available and I didn’t think the risk of issues that can arise from new versions was worth nodes newest features.

**Mongodb:** Mongo was surprisingly easy to install locally, I watched thenewbostons tutorial on YouTube to make sure I installed the database correctly then navigated to <https://www.mongodb.com/> to download the installer for my version of windows. Once installed I added the database and shell to my environment variables to make it easier to run the database and open the shell for testing.

**ExpressJs:** ExpressJs was very straight forward to install as all I needed to do was set it up using nodejs. This can be done by simply typing the following into the cmd in the project directory:

npm install express –save

npm install body-parser –save

This line will install expressjs and add it to the project dependencies which can be found by navigating to the package.json file in my project. The body parser is required to handle incoming HTTP requests. I then followed a tutorial on the mindspace YouTube page on connecting express to mongodb in the server.js file and created routes/index.js for managing my expressjs routes later.

**Angular2:** This can simply be set up by typing the following into the cmd to install as add to our dependencies:

npm install angular2 --save

Now we must set up a client folder to contain all our typescript files and angular2 related folders. In this folder I will store our typings.js and tsconfig.ts file for typescript along with our styles.css file for styling our app. I then created a app folder for all angular2 components, services and models. This where all the compiled typescript files go necessary for running the angular2 app. In this app folder we create a main.ts for launching our app in the browser and a module.ts for managing out angular modules. We will have a img folder for managing our JPGS and PNGS that will be used by angular.

**Mongoose:** I set up mongoose using npm install as follows:

npm install mongoose --save

I then navigated to server.js and imported the blurbird library for the mongoose schema to make sure it works smoothly with ES6, connected mongoose to the desired database and set the batch size to take in from the database to 300. We then create a models folder within the project folder to hold our mongoose schemas.

**Bower:** Now with the basics of a MEAN stack set up I decided to use a package manager like node NPM to install bootstrap and jQuery. This was as simple as creating a bower\_componets folder and running the three following commands.

npm install -g bower-installer

Bower install bootstrap

Bower install jQuery

**Views:** Finally we create a views folder and place a index.html file inside, we mark this as the base for our angular2 app and we set up all imports for nodejs,bootstrap and jQuery along with any other tools at this root so they can be accessed throughout this whole app. While the script is importing our app the html of this page will let the user know the program is loading.

**Server-side:** As I mentioned earlier we have a server.js file, I followed the same mindspace tutorial I mentioned before to set this up and then used that as a template to build into my own version that works with mongoose. This and the index.js file found the Routes folder(not to be confused with the one found in Views) are the heart of the server side and where the database engine is set up and requests/responses are handled.

**Running the app:** For this project, I installed nodemon for launching the server using NPM. This is like starting a server as usual but nodemon will update the code in real time so you can see you changes live in development. Make sure the angular2 client has been ran from the app folder using npm start. This will compile the angular2 code making sure it up to date for when the server uses it. To start the server type npm start or nodemon in the cmd of the project directory and then open your browser and go to `127.0.0.1:3000’.

**Git repository:** Once the basics of the Mean stack app have been applied and its is running on <http://127.0.0.1:3000/> I created a git repository here <https://github.com/ConorTighe1995/Final-Project--Retail-Management-System> I then created a README.md file and used the following commands to make my first commit:

git init

git add .

git commit -m “first commit”

git remote add origin <repo address>

git push -u origin master

**Architecture and Design**

In this section I will cover the main parts of the applications and break down the looks, features and design decisions I made while working on that part and its files. It is important to note that a lot of the files I don’t cover here are generated by angular2 and nodejs and where not created by me but are essential parts of the technologies that where created when compiling the code I wrote or during the installation of angular2 or nodejs.

**app.module.ts:** This is an essential file ties all the components together, can be thought of what puts all the components into scope. This is what allows all the sources like google maps or bootstrap to be integrated into each component while only being imported in Views/index.



**app.main.ts:** This is what causes the app to launch and what to be called and set up while it loads in the browser.

**Views/index.html:** this page handles the loading screen and is where we put all our links to the source files.



**app.routing.ts** This is where Angular manages all the possible URL’s in the app. The router outlet can be thought of as a frame and this file decides what pages/view to insert into the frame depending on the URL navigated to.

**How components interact with the server:**

Angular components user’s services to send and receive data from API’s. Services are just component design to manage all communication a database and encourage reusability of these communications.

Service

ExpressJs

Mongo

Component

Views/

index.js

Server.js

app.component:

router-outlet requests a

component for a path

/employees:

employees.component

/support:

support.component

/:

home.component

**Home page:**

In earlier commits of the project I wanted to have a 4-panel menu with a about section that would be revealed when you scroll down. I made the decision to cut off 2 of the panels and merge a lot of features with the other two. So instead of having 4 separate pages with different features there is now 2 panels, one for employees and one for stores. I made this design decision not only to save time but to make the code easier to manage, and make navigation of the app easier for users so they wouldn’t have to keep opening new tabs. Now everything relevant to employees is found under employees and the same for stores, the about section is below both panels explaining the apps purpose. I put in a getting started section to explain what each panel does too.

**app.component.ts:** This is the first component created in the app file, its common to give the first component of a Angular2 app this name as this is the one we will put the router-outlet in to display our views. The component contains the basics for a Angular2 component but links to a html template.



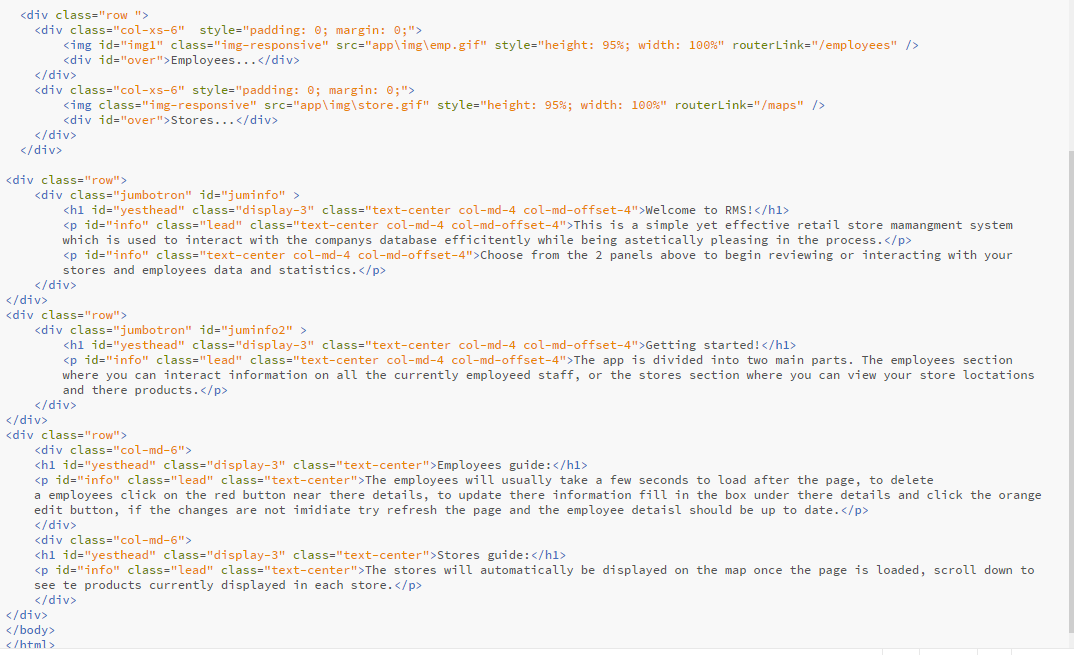
**app.component.html:** The following file contains a navigation menu that will be displayed at the top of the screen. The file contains a watermark and tabs linking to the application Home, About the app and Support for the app. It also contains links that would be relevant to a regional manager. At the bottom of the file we can see the router outlet which will act as a frame where we insert our routes. This combined with the bootstrap navigation bar means we can move from page to page and have the menu stay with the user.



**home.component**: The ts and html file of the home component is used to display the two panels that contain a .gif each and text overlaying the gifs. When clicked they redirect you to the appropriate route. At the bottom of the file is the code for the about section contained in a bootstrap jumbotron and uses google fonts API paired with the boostrap columns and a style.css file to give the About section its look. Below that Is the code for the getting started section which briefly explains what each panel is for.

****

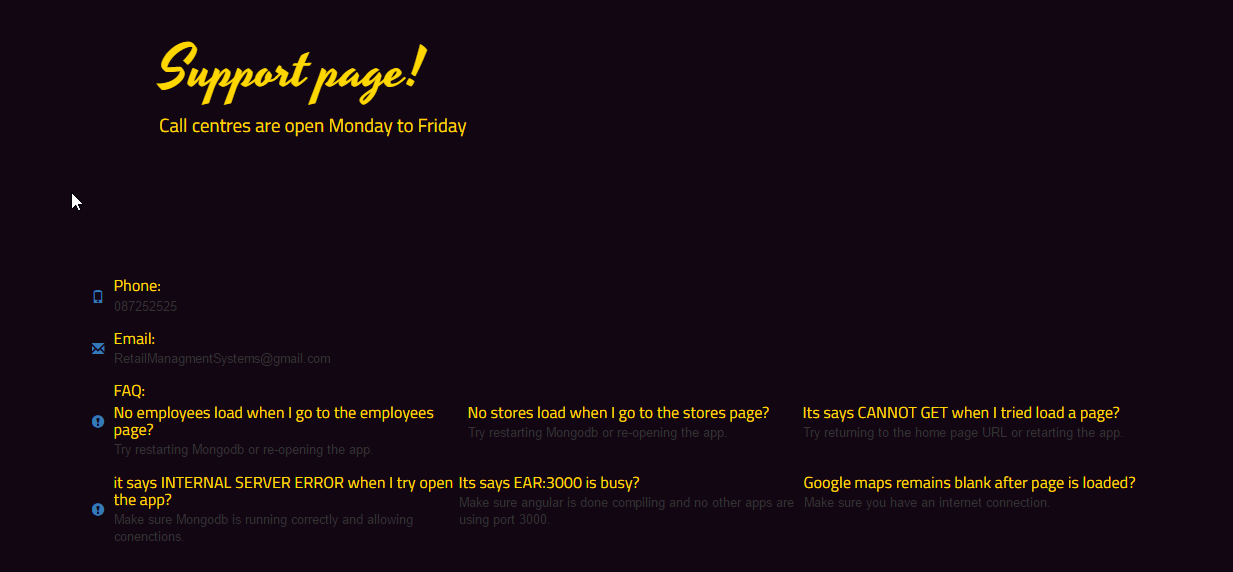
****



**Support page:**

The support page cane be navigated to anywhere in the app by clicking the support tab in the menu at the top of the screen. This page contains all necessary user contact information you’d expect to find on a support page along with a FAQ.

**support.component:** This component mainly uses bootstrap to display the contact details in a easy to view format, making the most of bootstraps glyph-icons, media objects and jumbotron. The email and phone number is dummy information.

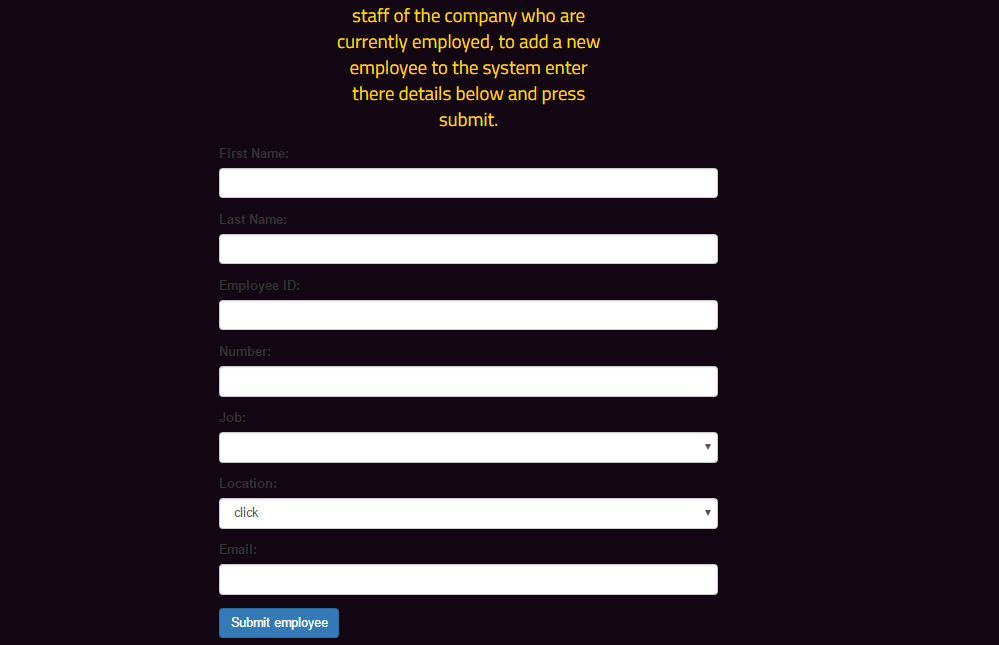




**Employee Page:**

This page uses all CRUD capabilities which was one of my original goals when I started planning the project. This page allows users to submit a new employee at the top and then displays the current employees at the bottom. When you click the employees ID the selected element in the list will expand showing all the information on that employee. This page made up of two main parts, the employees component and the details component.

**employees.component.ts:** this file contain the typescript that takes care of the GET request that makes sure all the stores in the create employees section are up to date and in sync. It also handles the POST request when the submit button is clicked by using Angular’s ngModel to fill out the new message to send to the server that will represent our new employee.

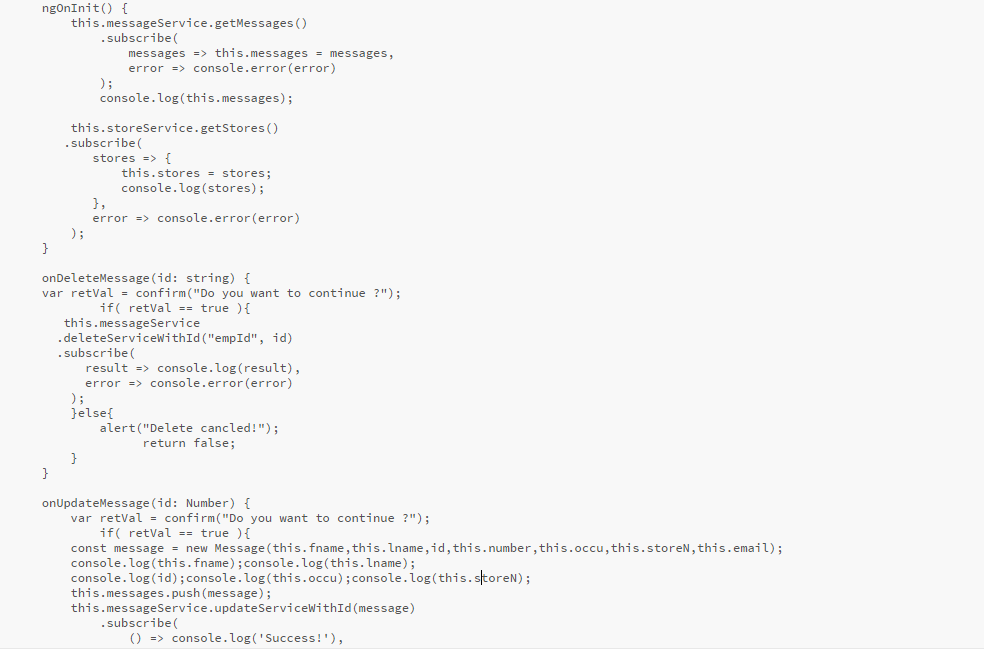




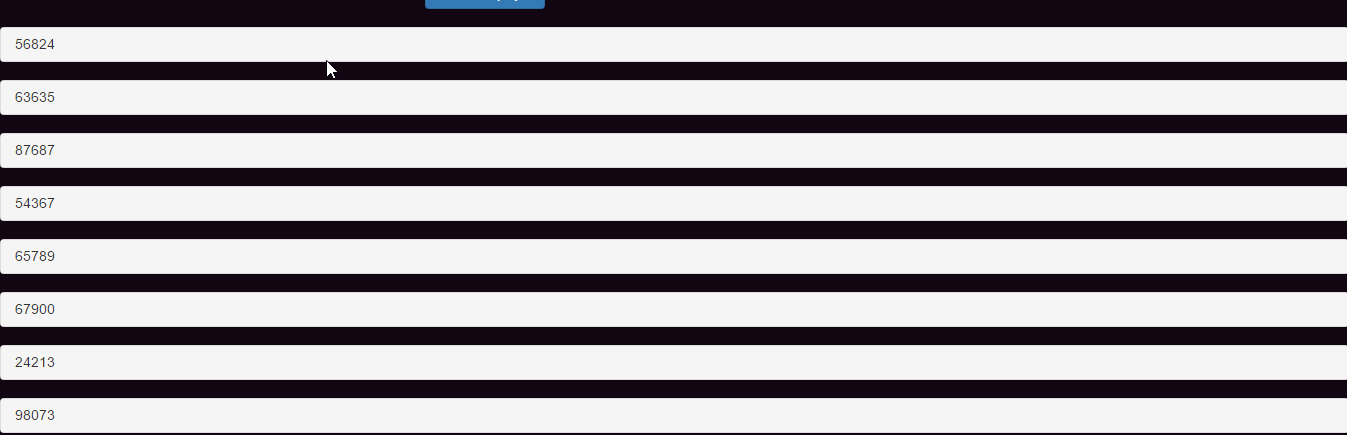
**employee.compontent.html:** this where ngModel binds the text entered in the input bars to the request to be sent to the server. This is also where the title of the employees page and its description is displayed.

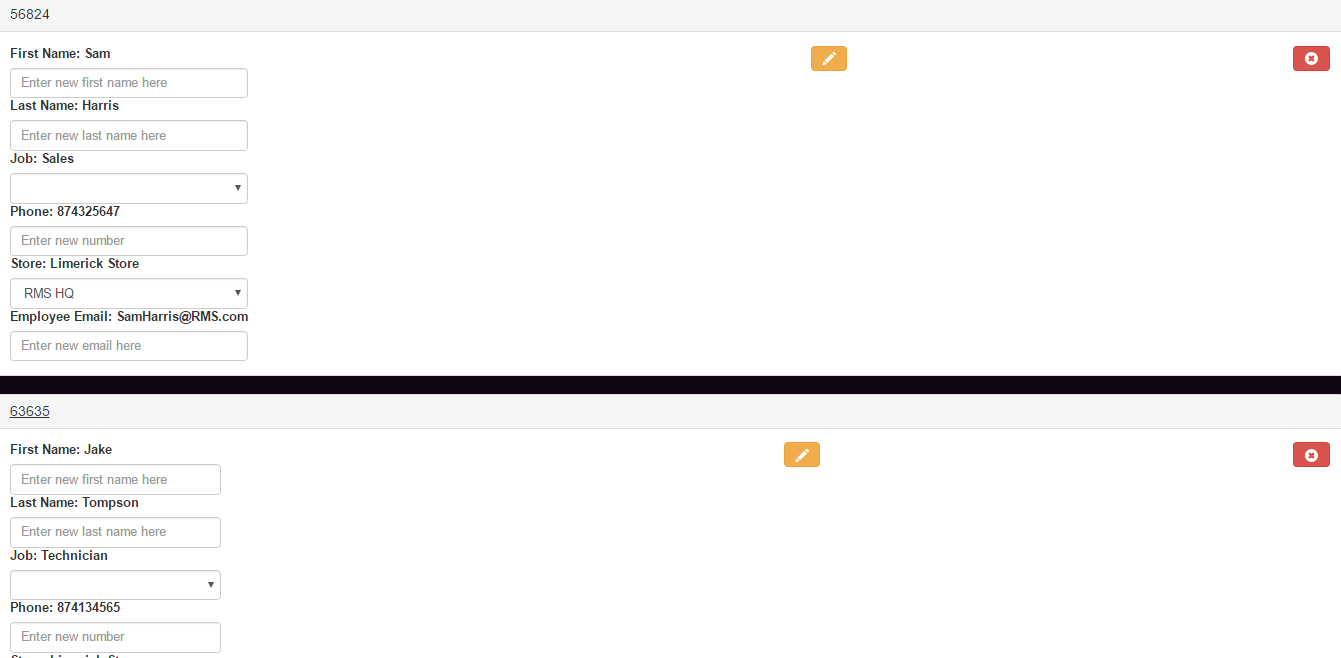
**details.component.ts:** This is where all the incoming employees from the GET request are prepared for the bootrap accordian as soon as the page loads using ngInit(). This file also handles any updates or deletes of employees by contacting the employee service.

**details.component.html:** here we use ngFor to populate the bootstrap accordion to take the response of the GET request that is stored in an array and fill the page with the employees. This also is where the update and delete methods in the component are activated, the input boxes below the data are what will replace the corresponding data above.

****

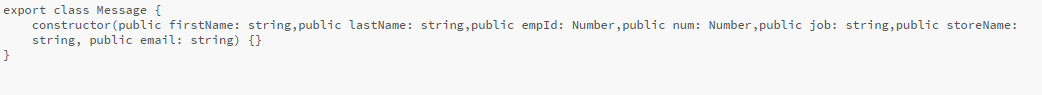
****

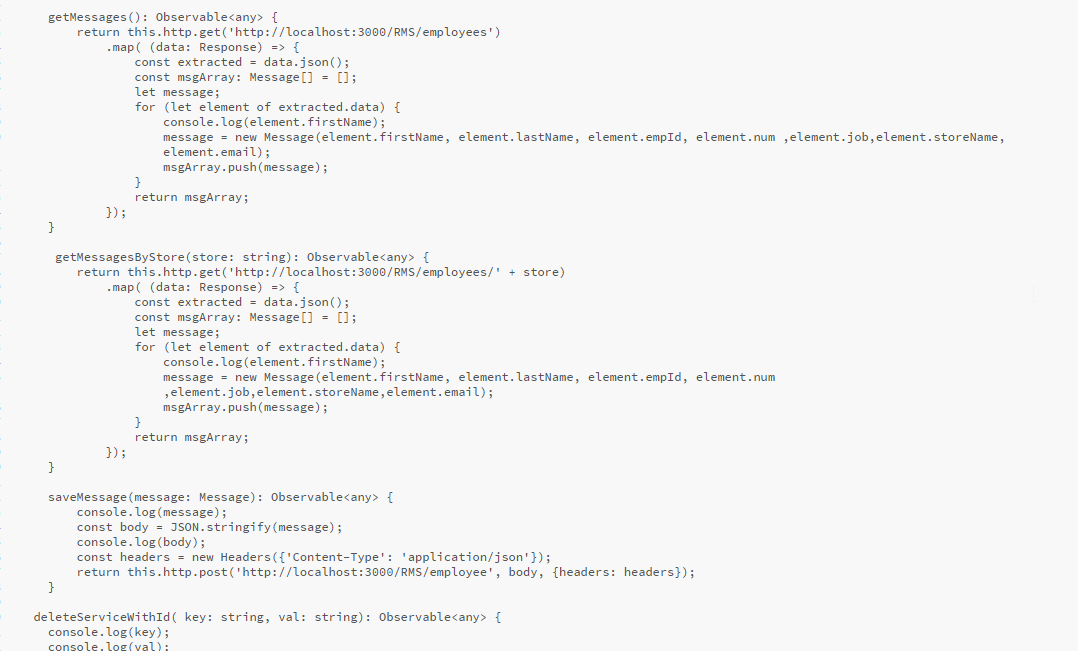
****

****

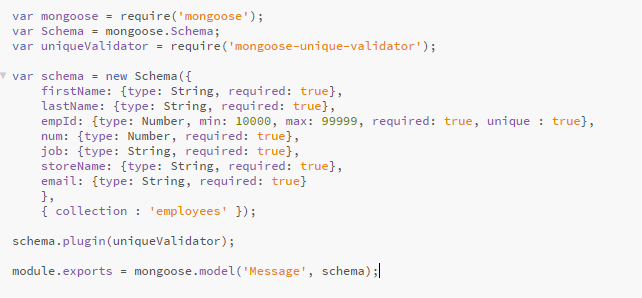
**employee.service.ts:** here is where we handle all api interaction involving the employees collection across the application. All though the bulk of use is on the employees page we also see this service used in the stores page later.

**employee.model.ts:** This is the skeleton for all employee objects. It contains a constructor for an employee that takes their details.





**Models/employee.js:** This is the actually model for the field in out employees collection. All details are required for the database to accept a POST and the empId must be 5 integers long and has to be unique.

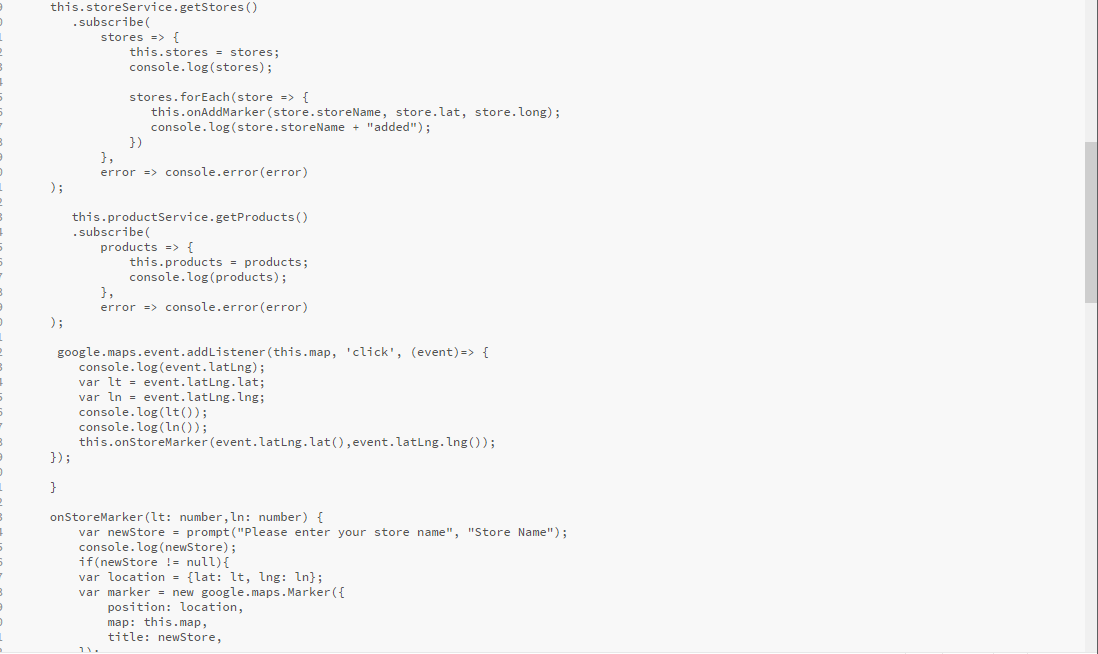


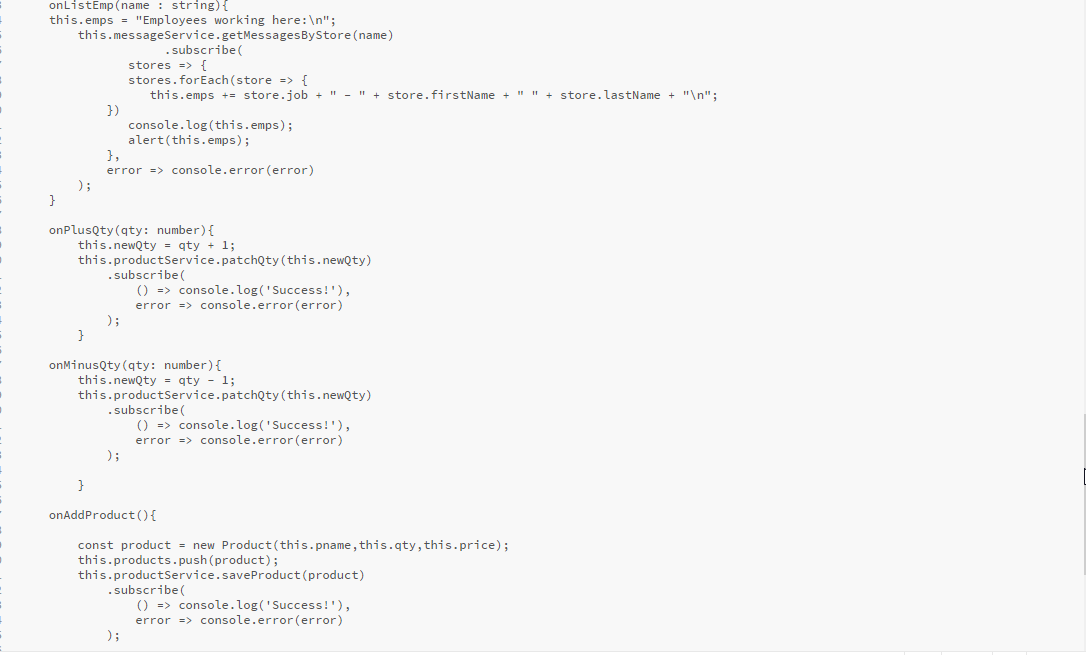
**Stores page:**

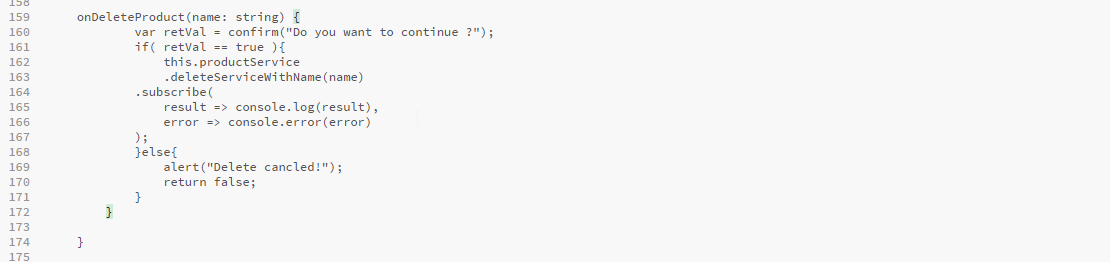
This page is used to show all the stores across the region or country. It used Google maps to display the map and markers and stores the co-ordinates in mongo. Clicking a marker will do a GET request to the server and return an alert with all the employees who work there. You can click on the map to add a new store, it will ask to name the store and once a new name is entered and OK is clicked the details are posted to the database. Below the map is a section to add a product along with a list of all the products within the stores and buttons to edit the quantity.

**map.component.ts:** this one component handles all the methods for the stores page.This is the largest component in the project as it makes us of all the service files I made and combines them with the google maps API. First thing that happens is the google maps is told to focus in one Ireland by giving the map object a lat,long and zoom. Then the call back received from the GET request feeds the latitude and longitude into google maps and creates all the store markers. The products are loaded with a GET request and a lister is added to the map for any clicks for when a user would want to add a store. The rest of the file contains the methods that make the processes above possible and the methods to manipulate the products in the database. The ListEmp method does a GET request using the employees service when a marker is clicked. The add and min methods patch the product quantity by doing +1 or -1 depends on what button is clicked.

**map.component.html:** at the top of the html the id for the map is set, this is where the map will show on screen. Below everything related to the products is stored inside a headered table, the header contains the input for new products and the table rows off all the current ones are listed off using angular2’s ngFor.









**product.service.ts:** this file controls all contact with expressjs to contact the products collection in mongo.

**stores.service.ts:** this file controls all contact with expressjs to contact the stores collection in mongo.

**stores.model.ts:** contractor for creating and passing around store objects between files.

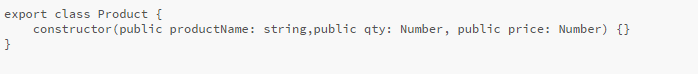
**products.model.ts:** contractor for creating and passing around product objects between files.

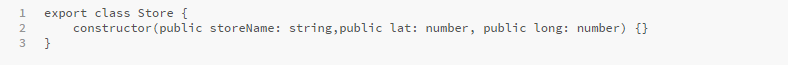
**Models/product.js:** blueprint for the product field that will go into the collections.

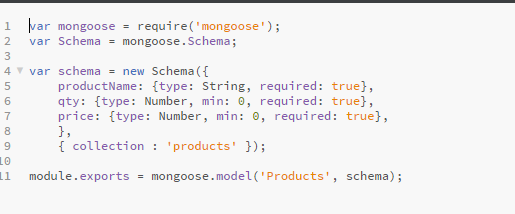
**Models/store.js:** blueprint for the store field that will go into the collections.

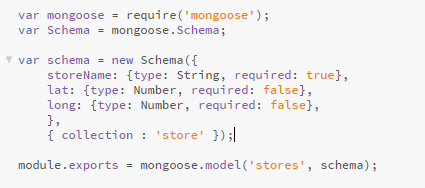




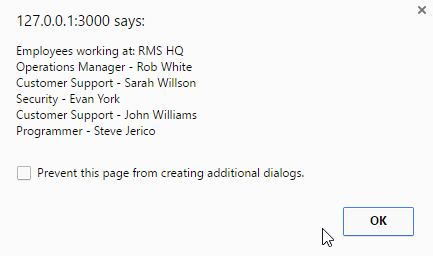
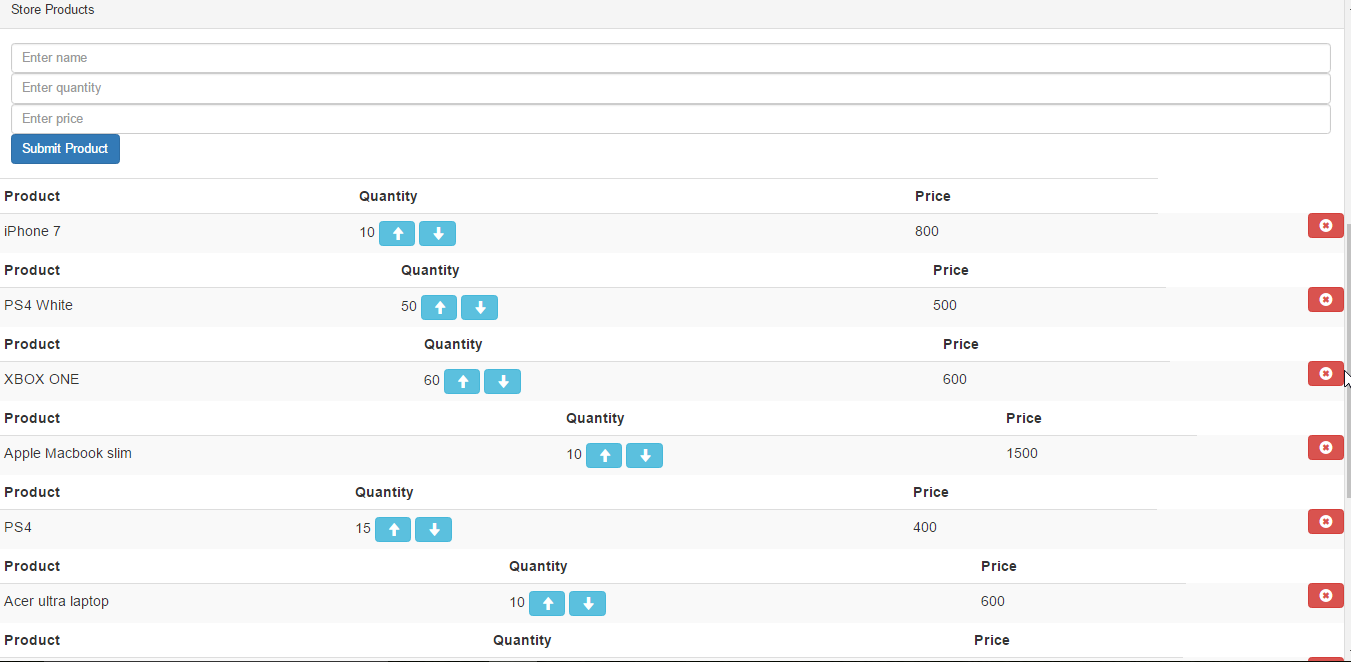








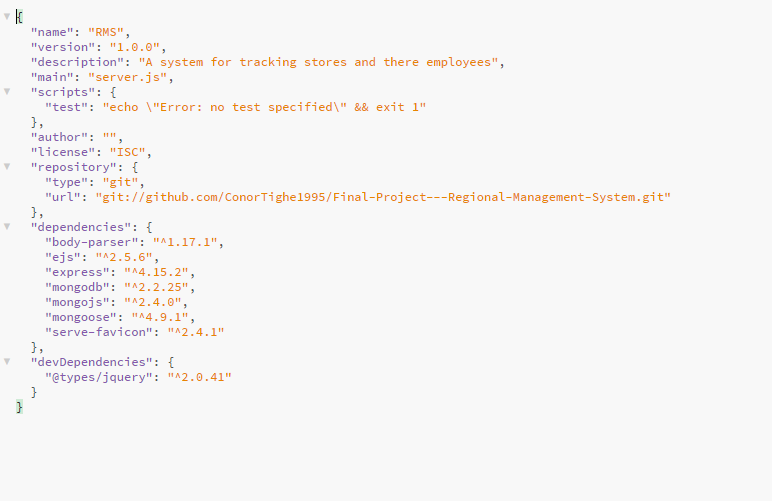




**Server side:**

All these files are found outside the client and are responsible for managing and running the entire app. This is where node will start the project using npm start server.js or nodemon. It’s also where node will check the package.json for everything required. The expressjs routes are handles in the Routes/index not to be confused with angulars View/index. In the server.js file import everything required for express, set the index and port, connect to the database, set the HTTP permissions etc.

















**Known Bugs and design flaws:**

**When the page is refreshed CANNOT GET is shown if you are not on the home page.**

-Fixed: this was fixed since by adding the providers HashLocationStrategy, LocationStrategy to the app.modue.ts file.

**The products don’t align vertically when loaded in from the service.**

-Not Fixed: Tried to use bootstrap columns but these don’t work correctly when binding from a asyn callback.

**You must select a location in the employee’s screen and then reselect the default or registers this as a null value.**

-Not Fixed: This is another problem involving the async call back that loads from the database.

**Request submission to databases can be confusing for users.**

Fixed: When testing m app Id regularly submit this by mistake, I resolved this issue by adding confirmation boxes to all the submit boxes and buttons.

**Conclusion**

I am happy with the results of this project as I completed all the goals I set for myself in the beginning. I managed to not only pick up 4 new technologies in the process of making this project but also enjoyed using them and would use them again in future projects. There are some changes I would make building the project like giving the products there own separate component. There were some time-consuming setbacks when trying to pick up the MEAN stack that I wouldn’t have to have dealt with if I took a more typical approach to a web application that we are taught in GMIT. I feel this was one of the more productive projects I’ve set out to do and the one I’ve learned the most in.