

# SOFT355 – Distributed Application Development

10555972 | Plymouth University | Computer Science | 8<sup>th</sup> January 2020

## CONTENTS

---

Functionality .....	2
Requirements.....	2
Design.....	3
Testing.....	3
DevOps.....	4
Personal Reflection .....	4
Appendices.....	6
Appendix 1 .....	6
Appendix 2 .....	7
Appendix 3 .....	8
Appendix 4 .....	8

## FUNCTIONALITY

---

The web application I have developed is a basic to do list application which allows users to access a personal to do list and perform functions such as: create, edit and complete to do items. The application allows for users to login and view their own to do list after they have registered for an account.

To interact with the application the users are required to first register their details, this allows them to set up a personal login using their email address and a password. Once an account has been created the user will be able to sign in and will gain access to their personal to do list. From here they will have full accessibility to add, delete and edit items from their to-do list. Using the simple user interface design users can enter a new to do item and click on a button to add it to the non-completed to do items. There is the ability to click on an item which either marks it as completed and moves it to the completed list or vice versa. Users are able to click either the edit or delete button next to each of the items in the list which allows for them to complete the desired action. In addition to this, users also have the ability to log out of their account once they have finished using the application. This will return users to the login screen where another user is able to either login or register. Screen shots of the application can be seen in appendix 1.

To develop and implement this application I have used the MEAN stack. I have used Node.js for the server side of this application as it allows for a JavaScript environment and uses asynchronous and event driven functions. Angular JS has been used to implement AJAX to handle post requests coming from the client to then be dealt with on the server. It has also been used for some of the client-side error handling and data manipulation. I have used Express.js for all of the routing and navigation on my application and for handling incoming AJAX requests. If a user is logged in at multiple places (e.g. on a computer and mobile device), when a change is made the application will alert the user that a change has been made and encouraging them to refresh the page. To do this I am using web sockets which enable a client to make a connection to the server once a user is logged in. I am using embedded JavaScript templates to allow me to easily inject content into HTML pages, sending the data straight from the server.

For my database I have used MongoDB and am I using Mongoose to interact with it at a server level. This has allowed me to easily store data in multiple collections without having to write any SQL or normalise any tables. I am using a package called bcrypt which allows me to salt and hash a password easily to then be stored in my database. It also allows me to verify a password by using the stored hash against a plain text password. This results in a very secure way to store and verify a user's password. In addition to bcrypt I am using a library called passport which allows me to use express sessions across my application and verify a user's authentication across all web pages. Once a user logs in their session is stored and is automatically killed after 1 hour of inactivity or when the user manually logs out using the logout button.

## REQUIREMENTS

---

The plan for this application is to be aimed at and to be usable by everyone. Although, it will be most useful for people who likes to be organised and efficient when it comes to completing tasks in their daily

life. The requirements for the application were quite simple. The user should be able to register, login, and be able to perform all functionality to their own to do item.

The application allows for users to create their own account which means that they are able to log in where ever they are and access their current to do list. The application is also completely scalable which allows users to use the application on mobile device as well as a computer (appendix 2). The users have two lists, a to do list and a completed list. This has been designed so that it is possible to not only remove items from your to do list but keep them stored in a separate list for future reference. Users are able to easily move items between the two lists depending on the status of that item.

As users will be able to use this application across multiple devices, another feature of the application is to show when a change has been made to your to do list. This is useful if you are logged in on multiple devices or if there is more than one person working from the same to do list.

## DESIGN

---

The design on my system can be seen in appendix 3. Throughout my application I have implemented the Model-View-Controller (MVC) pattern. I have used the MVC pattern because it helps keep all of the code and file organised, structured and easily readable. The model contains files that structure all of my data in my application (appendix 4), the view contains the files needed to display the data and the controller contains the files that handles all of the user's interactions.

Requests are sent from the client to my server through an API which is made up of AJAX requests. These requests are then handled by the server and the server makes a query to the database. Whether that be a get, post, put or delete request. The database responds to the server, and the server responds to the client with some data ready to be displayed on the browser.

## TESTING

---

Throughout the development of the application I have used test driven development (TDD), which has allowed me to ensure that my code is working after every change is made. As explained previously every time a code commit is made the build server automatically runs all of my tests and reports if any have failed. Overall this results in much more frequent runs of tests and reduces bugs dramatically. TDD allows for the single responsibility principle to be applied, which forces loose coupling between tests. This is because each test should focus on one functionality of your code and shouldn't interact with anything else. In addition, it creates a more thoughtful development environment as it helps to first think about code and designing a better structure before writing it. For creating and running the unit tests I am using Mocha and Chai as this allows for simple testing. Mocha is a testing framework that allows tests to be run from within a node terminal and display the test results in an easy to read manor. Chai is an assertion library that is used to compare results of the tests with the expected results.

The application was aimed for users who wanted a very simple and easy to use to do list that can be used on any device. To test that the application met all of the requirements I ran 3 one-on-one focus sessions with potential users. During these sessions I had them use the application on a computer and a mobile device and asked them to perform all of the main functions, such as; register, login, create a to

do item and makes changes to a to do item. From completing these sessions, I was able to gain a much better understanding with how a user would interact with the system and it allowed me to make necessary changes to both my front-end and back-end design to further improve the system.

When it comes to data validation most of this is completed client side using HTML5 built in functions, this makes it easy to specify what to expect from a user input and will flag if the input is not correct. If the server responds with an error, then error handling is completed using AJAX functions and then displayed to the user depending on what the error is.

## DEVOPS

---

To develop my application, I used Visual Studio Code as this allowed me to edit all of my files, debug my code and run terminals from within the IDE. In addition to this, it allows me to install extensions from a very large market place for any file type that I might be working on. To debug any code client side, I used Google Chrome the provided debug tool allows me to see all of client-side source files. I am able to set break points, see variables in real time and view all active connections, which was very useful when implementing web sockets. Whilst developing my application I simply created a local Node server anytime I wanted to test it. This allowed me to ensure I was testing the application exactly how it would be used in production.

It's very good practice to implement a continuous integration and continuous delivery (CI/CD) when developing an application. This is because it ensures that code changes are being committed to version control often and that the code compiles without any error whilst passing all unit tests. To implement a CI/CD environment for developing my application, I set up a build server running an open source server called Jenkins and hosted it on Microsoft Azure. I configured a webhook to my GitHub repository so that every time a commit is made to the repository my build server picks it up and clones it. Once the clone has been made, the server then compiles the application ensuring that there are no errors. Then finally it will run all of my unit tests and show a report on the results. If this application was being pushed into development, it would then be very easy to set up a script in Jenkins that simply pushed the application to a live server if all of the tests pass. This would result in the updated application being live from a single commit to source control.

## PERSONAL REFLECTION

---

When I initially started creating the application, I had planned for a lot more features and functions. However, the application had to be scaled down a bit to ensure that it could be completed within the time frame. Due to this, I would ensure that for my next project I will start small and add features once a base product has been created and I will always assume things are going to take longer than I originally think.

Using the MEAN stack worked very well for my application and allowed me to code purely in JavaScript, along with HTML and CSS. Using Passport JS to handle express sessions and my login system also worked well and was simple to implement into my system. However, I could have used web sockets more throughout my project to allow for better two-way communication between my clients and my server. I

also could have included more unit tests to ensure that smaller functions within my application were working as expected.

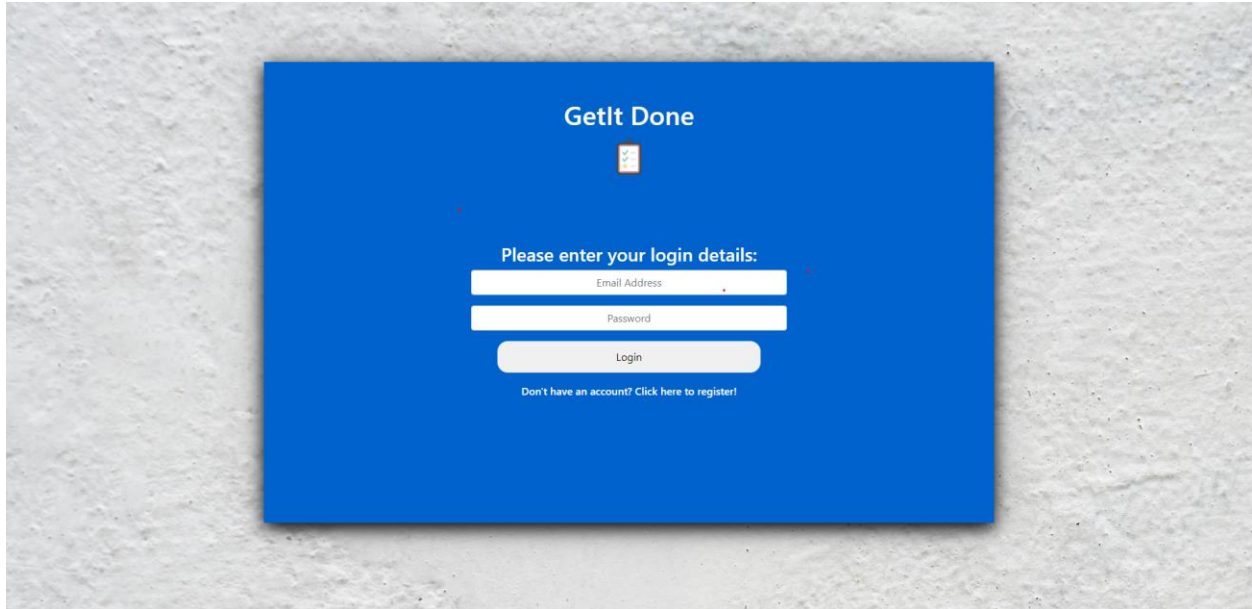
Although most of the validation is completed client side, I would have liked to of included more data validation on the server side. In addition, I think it would have been better if there was more error logging both on the server side and client side. Not only would this have helped me when developing the application and fixing bugs, it would also be useful to the user if there ever was an issue.

Overall, I believe that the development, implementation and testing of the application went well and the overall product meets the requirements. The technologies used throughout were appropriate for the project and allowed for a simple application.

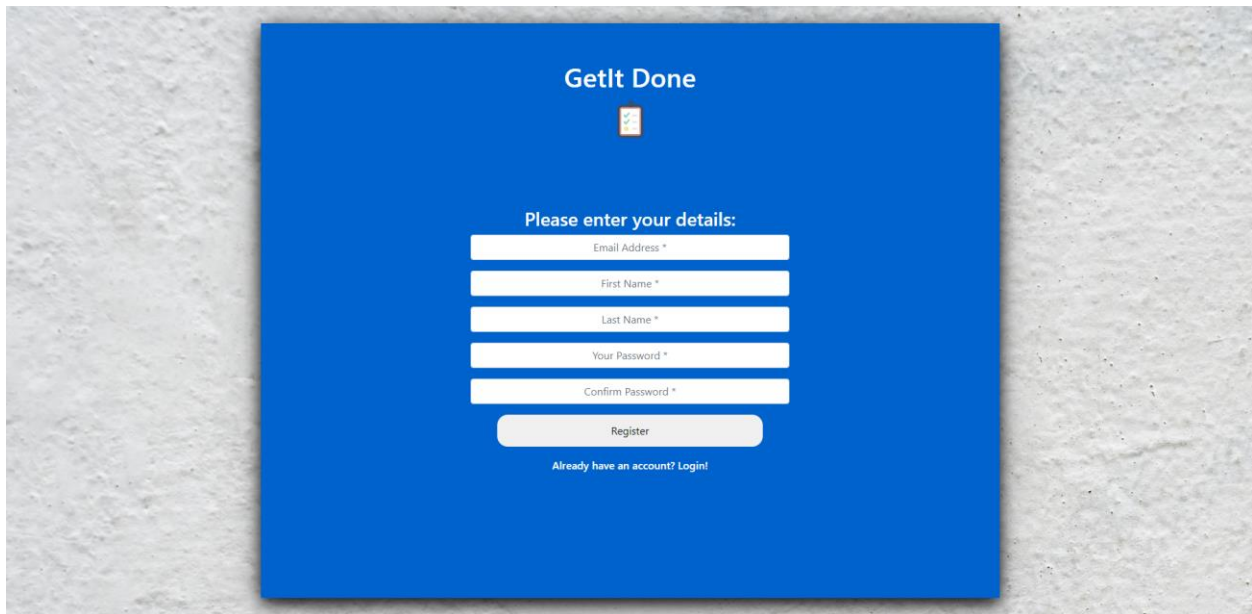
# APPENDICES

---

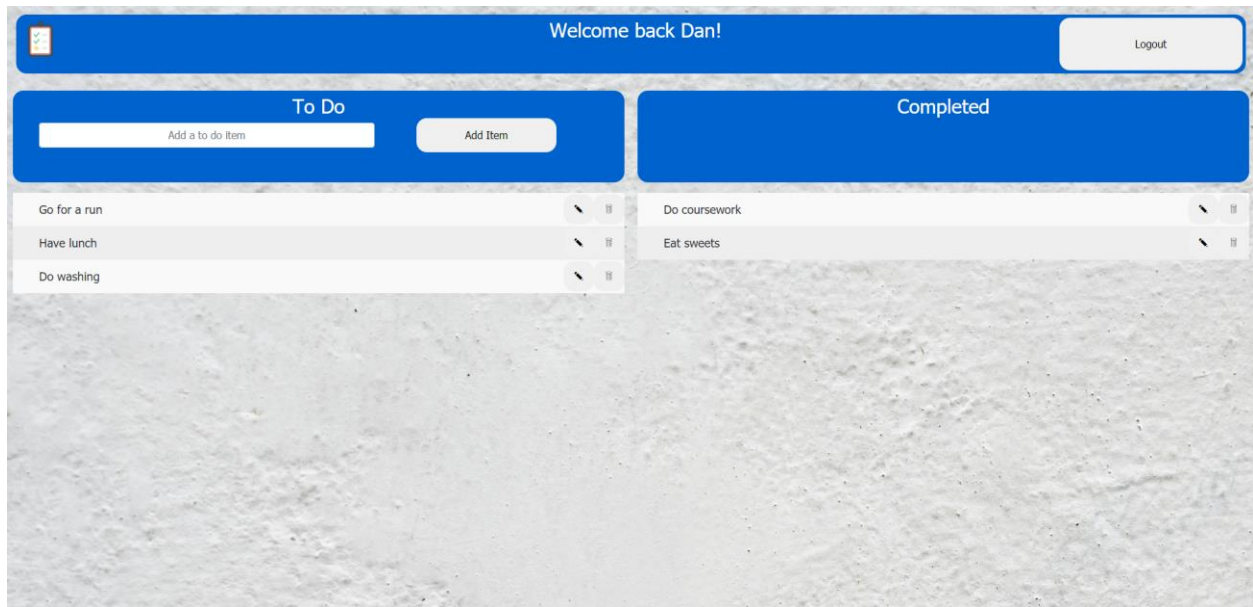
## APPENDIX 1



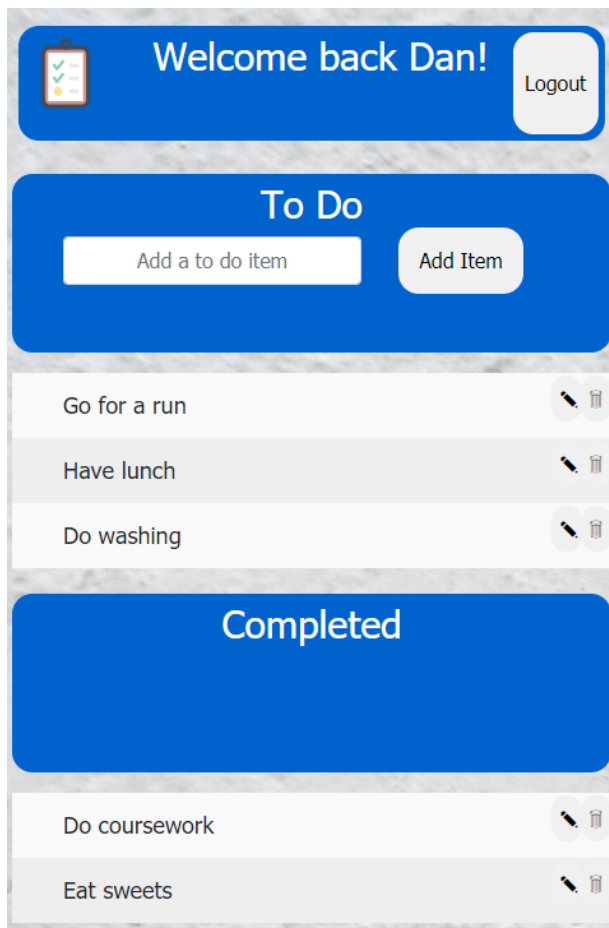
A login form mockup on a blue background. At the top, the text "GetIt Done" is centered, followed by a small icon of a notepad with a pencil. Below this, the text "Please enter your login details:" is centered. There are two input fields: "Email Address" and "Password". Below the "Password" field is a "Login" button. At the bottom, there is a link: "Don't have an account? Click here to register!"



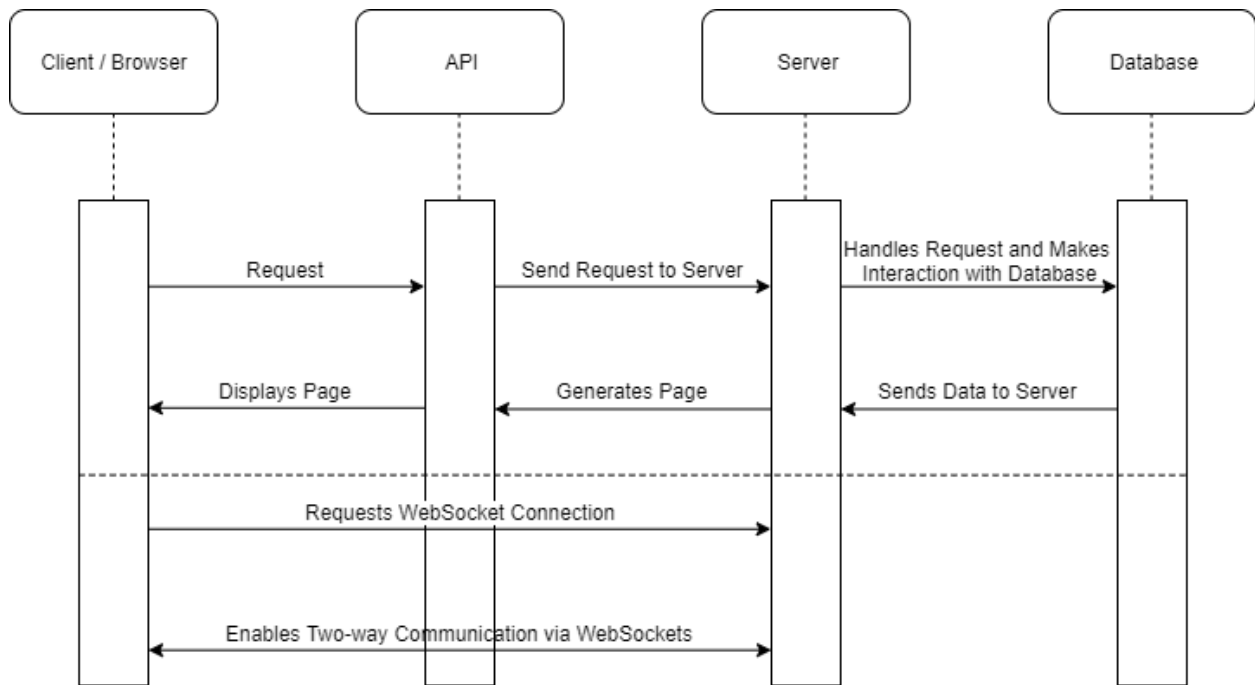
A registration form mockup on a blue background. At the top, the text "GetIt Done" is centered, followed by a small icon of a notepad with a pencil. Below this, the text "Please enter your details:" is centered. There are five input fields: "Email Address \*", "First Name \*", "Last Name \*", "Your Password \*", and "Confirm Password \*". Below the "Confirm Password \*" field is a "Register" button. At the bottom, there is a link: "Already have an account? Login!"



## APPENDIX 2



### APPENDIX 3



### APPENDIX 4

