11/28/2018

Gavin Hanna

HIGHER DIPLOMA IN SCIENCE IN WEB TECHNOLOGIES  
national College of Ireland

Technical Report

My Recipe Book

Table of Contents

[Executive Summary 3](#_Toc531792561)

[1. Introduction 4](#_Toc531792562)

[1.1. Background 4](#_Toc531792563)

[1.2. Aims 4](#_Toc531792564)

[1.3. Technologies 5](#_Toc531792565)

[1.4. Structure 6](#_Toc531792566)

[2. System 8](#_Toc531792567)

[2.1. Requirements 8](#_Toc531792568)

[2.1.1 Functional Requirements 8](#_Toc531792569)

[2.1.2. Data Requirements 9](#_Toc531792570)

[2.1.3. User Requirements 11](#_Toc531792571)

[2.1.4. Environmental Requirements 18](#_Toc531792572)

[2.1.5. Usability Requirements 18](#_Toc531792573)

[2.2. Design and Architecture 19](#_Toc531792574)

[2.2.1. Assumptions/Constraints/Standards 19](#_Toc531792575)

[2.2.3. Hardware Architecture 21](#_Toc531792576)

[2.2.4. Software Architecture 21](#_Toc531792577)

[2.2.5. Security Architecture 23](#_Toc531792578)

[2.2.6. Communications Architecture 23](#_Toc531792579)

[2.2.7. Performance 26](#_Toc531792580)

[2.2.8. System Design 26](#_Toc531792581)

[2.2.9. Data Conversions 28](#_Toc531792582)

[2.2.10. Application Program Interfaces 28](#_Toc531792583)

[2.3 Implementation 29](#_Toc531792584)

[2.4. Testing 33](#_Toc531792585)

[2.4.1. Server 33](#_Toc531792586)

[2.4.2. Client 35](#_Toc531792587)

[2.5. Graphical User Interface (GUI) Layout 38](#_Toc531792588)

[2.6. Customer Testing 47](#_Toc531792589)

[2.7. Evaluation 48](#_Toc531792590)

[3. Conclusions 51](#_Toc531792591)

[4. Further Development or Research 52](#_Toc531792592)

[5. References 53](#_Toc531792593)

[6. Appendix 54](#_Toc531792594)

[6.1 Project Proposal 54](#_Toc531792595)

[6.1.1. Objective 54](#_Toc531792596)

[6.1.2. Background 55](#_Toc531792597)

[6.1.3. Technical Approach 56](#_Toc531792598)

[6.1.4. Project Plan 57](#_Toc531792599)

[6.1.5. Evaluation 58](#_Toc531792600)

[6.1.6 References 59](#_Toc531792601)

[6.2 Project Plan 60](#_Toc531792602)

[6.3. Requirement Specification 61](#_Toc531792603)

[6.3.1. Introduction 61](#_Toc531792604)

[6.3.2. User Requirements Definition 62](#_Toc531792605)

[6.3.3. Requirements Specification 63](#_Toc531792606)

[6.3.4. GUI 72](#_Toc531792607)

[6.3.5 System Architecture 73](#_Toc531792608)

[6.3.6. System Evolution 74](#_Toc531792609)

[6.3.7. References 75](#_Toc531792610)

[6.4. Product Design Specification 76](#_Toc531792611)

[6.4.1. Introduction 76](#_Toc531792612)

[6.4.2. General Overview and Design Guidelines/Approach 76](#_Toc531792613)

[6.4.3. Architecture Design 77](#_Toc531792614)

[6.4.4. System Design 83](#_Toc531792615)

[6.4.5. User Interface Design 86](#_Toc531792616)

[6.4.6.References 93](#_Toc531792617)

# Executive Summary

The “My Recipe Book” Progressive Web Application is being developed in order to provide like-minded people with an online space for creating, sharing and organising their personal recipes and/or favourite recipes, and curating their own recipe book, which can be made up of their own, and other user’s recipes.

The developer has elected to create a Progressive Web App, as it can be used in a similar way to a native mobile application, thus saving development time and giving more choice to the user as to how they want to use the application. This may be on a desktop browser, a mobile browser, or saved to the home screen of their mobile phone.

The developer has decided upon the MERN stack for the creation of the application. This stack incorporates a combination of modern JavaScript technologies. A RESTful API is created using Node, with the express web framework, and MongoDB, a JavaScript based NoSQL database. The “view”, or user interface, of the application will be handles by the React library, which handles all of the elements of the application by which a user will directly interact. This client-side application will run on the user’s machine and communicate with the Node backend.

Ultimately the application should provide like-minded users with an all-in-one space for creating, sharing and searching/reading recipes and interacting with other users. The application should have an intuitive and easy to user interface which makes using the application a pleasure rather than a chore.

# 1. Introduction

## 1.1. Background

Recipe based websites are an extremely popular corner of the internet, with the top 15 recipe based websites ranging from 2.25 million unique monthly visitors for KraftRecipes, to 25 million visitors to AllRecipes (Ebizmba, 2018).

The word “food” has an average monthly search count of 823,000, with “recipes” having 450,000 and “recipe” with 90,500 (Searchvolume.io, 2018). In Google Trends, the search term “recipe” never dips below 50% “Interest over time” (Google Trends, 2018).

The popularity of recipe based searches on the internet, and the overwhelming popularity of social networks (Facebook alone has over 2,196,000,000 users (Statista.com, 2018)), lead naturally to the combination of the two, of which there are many such websites currently available online. Two of the most popular are bakespace.com and cucumbertown.com. While these are technically social networks, they either focus more on simply being a recipe search engine, or acting as a blogging platform focused on recipes.

I believe there is a space for a fully realised recipe based social media web application in which people can follow their friends and share their particular cooking styles while interacting with and sampling/critiquing styles of their loved ones.

## 1.2. Aims

The aim of this project is to design and implement a fully functional Progressive Web Application. Capable of offering users a useful and easy to use interface in order to comfortably navigate user submitted recipes and create and curate their own recipes.

Through the development of the application, the developer aims to learn as much as possible about the required technologies. From developing a RESTful API backend, to constructing a functional user interface and connecting each aspect of the application in a concise and efficient way.

## 1.3. Technologies

The application itself will be developed with the following technologies.

Server-side

|  |  |
| --- | --- |
| Node.js | A server-side version of the JavaScript language. |
| Express.js | A web application framework commonly used to build RESTful APIs. |
| MongoDB | A NoSQL database, hosted on mlab.com |
| Mongoose | An ODM (object document mapper), used for managing the interaction between the server and the database. |

NPM Packages used:

|  |  |
| --- | --- |
| Bcryptjs | A JavaScript implementation of the bcrypt library, used for hashing user password. |
| Body-parser | Used for getting information from forms submitted via post requests. |
| Concurrently | An npm package used to run the server and the client applications at the same time |
| Jsonwebtoken | Used for handling the use of JSON Web Tokens in Express |
| Passport | Handles user authentication and authorisation |
| Passport-jwt | A JSON Web Token authentication strategy for use with Passport |
| Validator | Used to handle different types of form/input validation |
| Firebase | JavaScript wrapper for interacting with a firebase instance |

Client-side

|  |  |
| --- | --- |
| React | The React JavaScript library |
| Axios | Used for making AJAX requests to the server |
| Classnames | Conditional class names in React |
| Redux | Used to manage application-wide state |
| Moment.js | Parse dates to human readable strings |
| Jwt-decode | Decode JSON Web Tokens into usable JavaScript Objects |
| React-detect-offline | A react library for detecting and reacting to internet conenctivity |

## 1.4. Structure

Fig. 1.4.1

The above diagram (fig. 1.4.1) outlines a high level view of the overall system architecture.

In this representation, the client represents a user’s browser, on whichever device they are using e.g. tablet, phone, desktop etc. When the client navigates to the web application’s URL, the React application loads in their browser and displays the application to the user. The user can navigate to different “pages” in the application, which will be handles by React using the React Router. The application itself will be a single page application, but will function as a multi-page website. The React Router will handle all routing in the application. Page change renders will happen extremely quickly as the application will not have to request an entirely new document on each page change, but will simply load the specific component which is related to the current URL (e.g. when the user navigates to /login, the React application will load the Login Component).

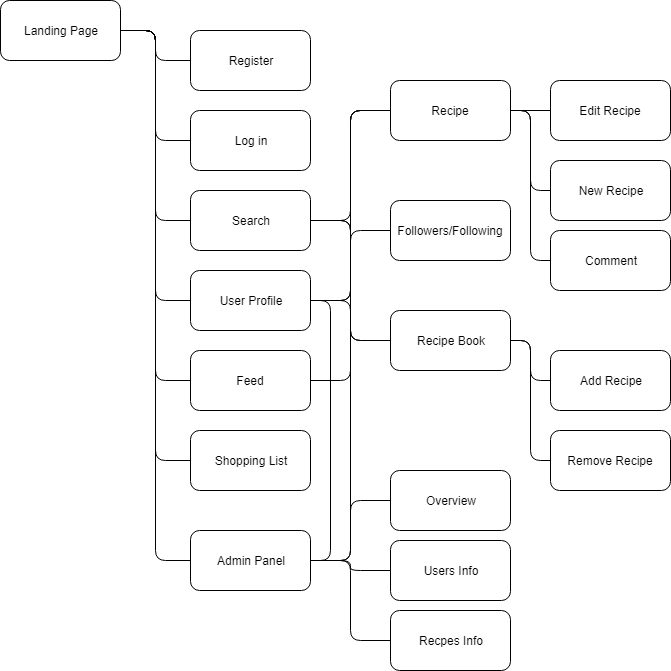


Fig. 1.4.2

The above diagram represents a simple site map. Most navigation revolves around going to and from Recipes and their respective creators, or site users. The search functionality will provide the user with a list of recipes related to their search parameters, which in turn will bring the user to that specific recipe page. From here, if they are logged in they can comment. If it is their own recipe they can edit the recipe.

Similarly from a user’s profile page, where if they are logged in they can choose to “follow” that specific user. Here a recipe can be selected and the user will be taken to that specific recipe page, where again they will be able to comment if they are logged in.

From a user’s profile page, that users specific Recipe Book can be viewed, which is a personally curated list of that users favourite recipes.

The “feed” page, which is available to logged in users, will show the most recently created recipes by the users which they follow. If no users are being followed, the most recent posted recipes will be shown.

# 2. System

## 2.1. Requirements

### 2.1.1 Functional Requirements

The following functional requirements represent the minimum viable functionality by which the application could be considered usable.

* Upon reaching the home page, the content of the page should render correctly. Displaying information about the application and links to Register and Login functionality.
* Register and Login forms should be fully validated.
* Users should be able to view a selection of recipes without having to be logged in or registered.
* Users should be able to register for an account using a valid email address and password. The username should be anything they choose between 3 and 30 characters long.
* Logged in users should be able to view their personal “user” page, containing any previously submitted recipes.
* Logged in users should be able to comment on their and other users recipes.
* Logged in users should be able to create a new recipe, and edit any previously created recipe.
* Creating or updating recipes should have fully functional validation.
* Logged in users should be able to “follow” other users and see their recipes on the initial users homepage, or “feed”.
* Logged in users can add any viewable recipe to their own “Recipe Book”.
* Logged in users can remove recipes from their own “Recipe Book”.
* Logged in users can view other users Recipe Books.
* Once a user logs out, any information relating to the previously logged in user should be removed from the specific client side rendering of the application.

Updated and newly added requirements follow:

* Logged in users should have access to their own personal shopping list.
* Logged in users should be able to add the ingredients of a specific recipe entirely to their recipe list.
* Logged in users should be able to set items in a shopping list to picked up or not picked up.
* Logged in users should be able to clear their shopping list when all items are picked up.
* Admin users should be able to edit any users comments and recipes.
* Admin users should be able to set any other users status to admin.

The most recent requirements were added late in the development cycle, after the main objectives had been completed and basic functionality achieved, it was decided to go ahead with the Shopping List component of the app. Even later it was decided to add some admin functionality to the app, in order to create a space for administrative functionality and application user content moderation.

### 2.1.2. Data Requirements

Data will be persisted in a MongoDb database, the actual data stored is illustrated in the following Schema:

Recipe

|  |  |  |
| --- | --- | --- |
| Data | Type | Is Required |
| user\_id | ObjectId (ref. ‘User’) | True |
| title | String | True |
| description | String | True |
| cookingTime | Number | True |
| prepTime | Number | True |
| serves | Number | True |
| ingredients | Array of Strings | True |
| method | Array of Strings | True |
| Img\_url | String | True |
| meal | String | True |
| dietary | String | True |
| likes | Array of ObjectIds (ref. ‘User’) | False |
| comments | Array of Objects containing User and Text | False |
| created\_at | Date | True |
| edited\_at | Date | False |

User

|  |  |  |
| --- | --- | --- |
| Data | Type | Is Required |
| Name | String | True |
| Username | String | True |
| Email | String | True |
| Password | String | True |
| Img\_url | String | True |
| Blurb | String | False |
| Following | Array of ObjectIds (ref. ‘User’) | False |
| Followers | Array of ObjectIds (ref. ‘User’) | False |
| Created\_at | Date | True |
| Admin | Boolean | True |

RecipeBook

|  |  |  |
| --- | --- | --- |
| Data | Type | Is Required |
| user\_id | ObjectId (ref. ‘User’) | True |
| recipes | Array of ObjectIds (ref. ‘Recipes’) | False |

ShoppingList

|  |  |  |
| --- | --- | --- |
| Data | Type | Is Required |
| User | ObjectId (ref. ‘User’) | True |
| List | Array of Objects {item: String, pickedUp: Boolean} | False |

### 2.1.3. User Requirements

The system must allow unregistered users to browse recent recipes, search recipes by specific keywords and view recipe creator profile pages.

The system must offer an ever present link to the Register and Login pages in the navigation bar and on the landing page of the application.

Once a user has register they will be offered the chance to submit their own recipes, to be able to view their own personal profile page. Users will be able to find and follow other users and to comment on and like other users recipes. Each user can then add recipes that they enjoy to their very own “Recipe Book”, which can be shared with and viewed by other users.

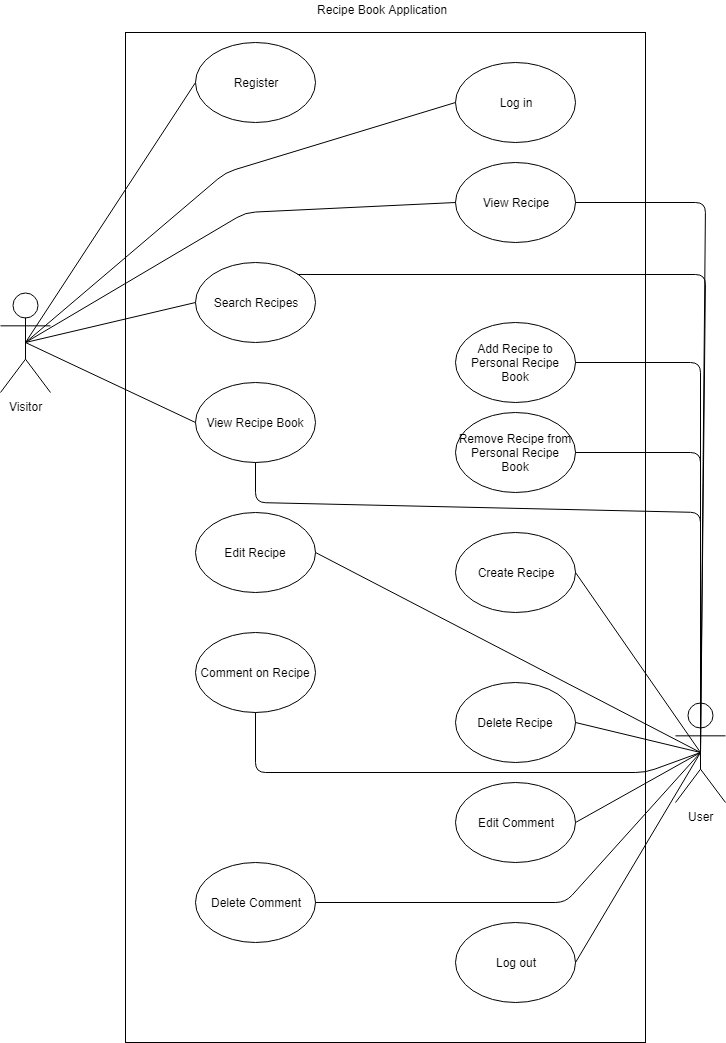
Users should be able to edit their previously submitted comments or previously published recipes.

A mobile site visitor should be offered the option to add the application to their home screen, regardless of their devices operating system, which will act as if it was a native application on that device. If a user opens the application without an internet connection, they should be notified and offered a chance to reload in an attempt to reconnect.

The application will have a responsive navigation bar at the top of the screen to permit easy navigation of the application.

The application must offer each user a personal “Shopping List” component, of which each action (adding items, picking up or deleting items) must be attached to the database.

#### 2.1.3.1 Use Case Diagram

The following Use Case Diagram represents functionality offered to both a non-logged in user (represented as Visitor) and a logged-in user (represented here as User).

#### 2.1.3.2 Requirement 1: User Registration

##### Description & Priority

A user can remain a “visitor” on the website, meaning they can peruse and search for recipes, but cannot create a profile, follow creators or comment on recipes until they register for an account. To unlock full functionality, a visitor must register and become a user.

#### Use Case: Register

**Scope**

The scope of this use case is to allow a user to register an account with the application.

**Description**

This use case describes the steps to be taken in order to successfully create an account on the application.

**Flow Description**

**Precondition**

The system is idle

**Activation**

This use case starts when a user clicks on the “Register” link.

**Main Flow**

1. The user navigates to the “Register” page on the web application.
2. The user enters a username of between 3 and 30 characters.
3. The user enters a valid email address.
4. The user enters a valid password of between 6 and 30 characters.
5. The user confirms the original password.
6. The users clicks on the “Submit” button.
7. The system sends information to the backend server.
8. The server-side application validates the inputs. <E1>
9. The server-side application stores the users details in the database.
10. The server-side application responds with the registered users details (minus the password).
11. The system allows the user to log in to the website.

**Exceptional Flow**

E1: Inputs are invalid

1. The user inputs which are invalid are displayed in red, with a message indicating the reason for the error under each invalid input.
2. The user inputs correct and valid inputs in the form.
3. Return to step 7 of Main Flow.

**Termination**

The application reports a successful registration.

**Post Condition**

The system returns to a wait state.

#### 2.1.3.3 Requirement 2: User Login

#### Description & Priority

In order for a user to access their account and use it’s full functionality, its necessary to that user to login to their account using details used during registration.

#### Use Case: Sign In

**Scope**

The scope of this use case is to allow a user to enter into their personal account on the application.

**Description**

This use case describes the steps to be taken in order to successfully log in to their personal account on the application.

**Flow Description**

#### **Precondition**

The system is idle

**Activation**

This use case starts when a user clicks on the “Login” link.

**Main Flow**

1. The user navigates to the “Login” page on the web application.
2. The user enters their username of between 3 and 30 characters.
3. The user enters their specific email address.
4. The user enters their specific password of between 6 and 30 characters.
5. The users clicks on the “Submit” button.
6. The system sends information to the backend server.
7. The server-side application validates the inputs. <E1>
8. The server-side application responds with the registered users details (minus the password).
9. The system logs the user in to the web application.
10. The system displays the users profile page.

**Exceptional Flow**

E1: Inputs are invalid

1. The user inputs which are invalid are displayed in red, with a message indicating the reason for the error under each invalid input.
2. The user inputs correct and valid inputs in the form.
3. Return to step 7 of Main Flow.

**Termination**

The application reports a successful login and shows the user profile page.

**Post Condition**

The system returns to a wait state.

#### 2.1.3.3 Requirement 3: Create Recipe

#### Description & Priority

In order for a user to publish her/his own recipe, they must first go through the process of submitting validated information via the Create Recipe functionality of the web application.

#### Use Case: Create Recipe

**Scope**

The scope of this use case is to allow a user to submit a new recipe in the form of form data in the web application.

**Description**

This use case describes the steps to be taken in order to submit a new recipe to the web application.

**Flow Description**

**Precondition**

The system is idle

**Activation**

This use case starts when the user clicks on the “Create Recipe” link.

**Main Flow**

1. The user navigates to the “Create Recipe” page on the web application.
2. The user enters the specific recipe information into the input form.
3. The user clicks on the “Submit” button.
4. The application sends the form data to the server-side application.
5. The server-side application validates the inputs. <E1>
6. The server-side application saves the new recipe to the database.
7. The server-side application responds with the newly saved recipe data.
8. The new recipe data is added to the front end application state.
9. The user is taken to the new recipe’s URL.

**Exceptional Flow**

E1: Inputs are invalid

1. The user inputs which are invalid are displayed in red, with a message indicating the reason for the error under each invalid input.
2. The user inputs correct and valid inputs in the form.
3. Return to step 6 of Main Flow.

**Termination**

The application reports the successful saving of the new recipe and shows the user the specific recipe page.

**Post Condition**

The system returns to a wait state.

#### 2.1.3.4 Requirement 4: Update Recipe

#### Description & Priority

**Scope**

The scope of this use case is to allow a user to edit a previously submitted recipe in the form of form data in the web application.

**Description**

This use case describes the steps to be taken in order to update a previously submitted recipe in the web application.

**Flow Description**

**Precondition**

The system is idle

**Activation**

This use case starts when the user clicks on the “Edit Recipe” link on a specific user created recipe’s page.

**Main Flow**

1. The user selects “Edit Recipe” on a specific recipes page.
2. The user is shown a similar form with inputs as in the “Create Recipe” page, however the inputs have been pre-populated with the current recipe’s data.
3. The user edits or changes the pre-populated input fields with new and updated data.
4. The user clicks on the “Submit” button.
5. The application sends the updated form data to the server-side application.
6. The server-side application validates the inputs. <E1>
7. The server-side application updates the recipe in database.
8. The server-side application responds with the newly updated recipe data.
9. The updated recipe data is added to the front end application state.
10. The user is taken to the updated recipe’s URL.

**Exceptional Flow**

E1: Inputs are invalid

1. The user inputs which are invalid are displayed in red, with a message indicating the reason for the error under each invalid input.
2. The user inputs correct and valid inputs in the form.
3. Return to step 7 of Main Flow.

**Termination**

The application reports the successful saving of the updated recipe and shows the user the specific recipe page.

**Post Condition**

The system returns to a wait state.

### 2.1.4. Environmental Requirements

For the application to function correctly, the environment in which is deployed must have a certain minimum set of installed technologies and minimum versions.

#### Server Environmental Requirements

The server must have a recent stable version of node.js installed on the machine. The version of node must be at least 6.15, as the application employs modern JavaScript syntax, which was essentially finalised in version 6.15 (node.green, 2018).

The server must have access to the internet.

#### Client Environmental Requirements

A modern browser is required in order to fully utilise the React application on the client side. All browsers including Internet Explorer 9 and above are supported (ReactJS, 2018).

### 2.1.5. Usability Requirements

The application should have a clear and obvious user interface. The user experience should be smooth and the components of the application should act as the user expects when they are interacting with any aspect of the app.

## 2.2. Design and Architecture

### 2.2.1. Assumptions/Constraints/Standards

For full functionality to be available to the user, it is a assumed a strong internet connection is present. However when connection fails, or in the event of a weak connection, the user will still be presented with some minimal level of information, this may be simply an offline page that will inform the user of the lack of connection and prompt them to reconnect, or some cached information being displayed which was saved during the last page load.

The application itself was developed specifically using the latest version of the Chrome browser on Windows 10. However the application was tested on the Firefox browser on Windows 10, Firefox and Chrome browsers on Ubuntu Linux, Firefox and Chrome on Android 7+ and Safari browser on iPhone X.

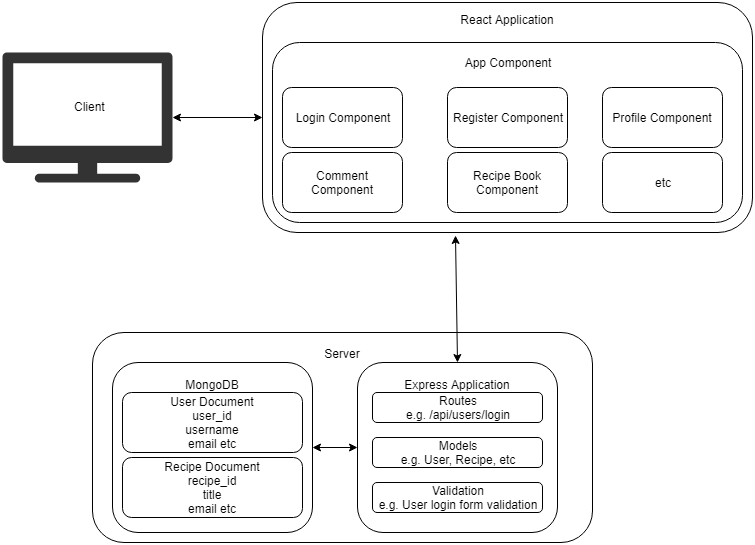
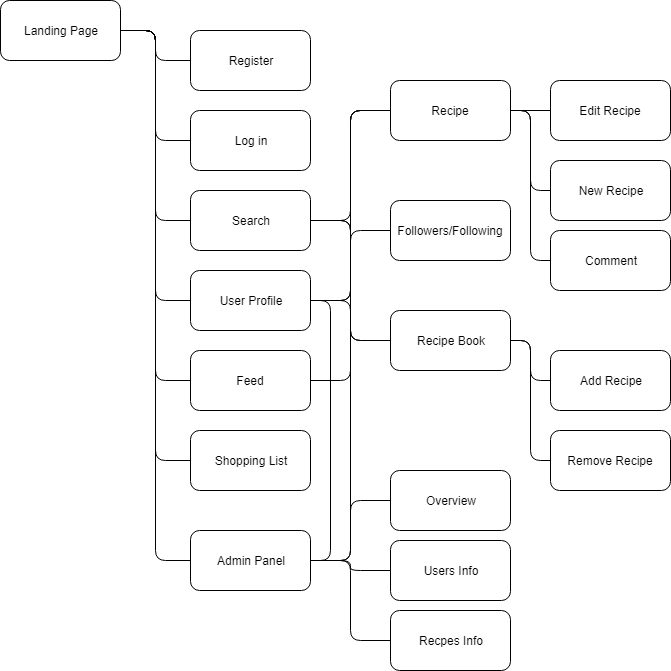
2.2.2. Logical View

Fig. 2.2.2.1

The above diagram (fig. 2.2.2.1) outlines a high level view of the overall system architecture.

In this representation, the client represents a user’s browser, on whichever device they are using e.g. tablet, phone, desktop etc. When the client navigates to the web application’s URL, the React application loads in their browser and displays the application to the user. The user can navigate to different “pages” in the application, which will be handles by React using the React Router. The application itself will be a single page application, but will function as a multi-page website. The React Router will handle all routing in the application. Page change renders will happen extremely quickly as the application will not have to request an entirely new document on each page change, but will simply load the specific component which is related to the current URL (e.g. when the user navigates to /login, the React application will load the Login Component).

Fig. 2.2.2.2

The above diagram represents a simple site map. Most navigation revolves around going to and from Recipes and their respective creators, or site users. The search functionality will provide the user with a list of recipes related to their search parameters, which in turn will bring the user to that specific recipe page. From here, if they are logged in they can comment. If it is their own recipe they can edit the recipe.

Similarly from a user’s profile page, where if they are logged in they can choose to “follow” that specific user. Here a recipe can be selected and the user will be taken to that specific recipe page, where again they will be able to comment if they are logged in.

From a user’s profile page, that users specific Recipe Book can be viewed, which is a personally curated list of that users favourite recipes.

The “feed” page, which is available to logged in users, will show the most recently created recipes by the users which they follow. If no users are being followed, the most recent posted recipes will be shown.

The Admin Panel is available to users who have the special admin flag enabled, which can only be enabled by other admin. The admin sections provide some information regarding Users and Recipes available across the web app.

### 2.2.3. Hardware Architecture

**“Heroku is** the quickest way for a company to become an apps company. Heroku is a service that enables companies to spend their time developing and deploying apps that immediately start producing value.” (Heroku, 2018)

The entire application is hosted on the Heroku platform. The Heroku platform has a free tier with a very easy to use CLI (Command Line Interface) which enables quick and painless iterative deployment.

The client and server was deployed to the same Heroku instance.

When a user navigates to the application’s URL, the client-side application will be downloaded and run on the users machine. Requests for information will be sent to the server from the client, and the server will respond with JSON data which will be used to update the users view.

The client itself simply refers to the users browser, which can be on any machine with a modern web browser. The web application can scale to any viewport size, therefore the application will work on tablets, desktops and mobile phones.

### 2.2.4. Software Architecture

A visual overview of the system architecture can be seen in figure 3.1.1.

The application itself was developed with the following technologies.

#### Server-side

|  |  |
| --- | --- |
| Node.js | A server-side version of the JavaScript language. |
| Express.js | A web application framework commonly used to build RESTful APIs. |
| MongoDB | A NoSQL database, hosted on mlab.com |
| Mongoose | An ODM (object document mapper), used for managing the interaction between the server and the database. |

NPM Packages used:

|  |  |
| --- | --- |
| Bcryptjs | A JavaScript implementation of the bcrypt library, used for hashing user password. |
| Body-parser | Used for getting information from forms submitted via post requests. |
| Concurrently | An npm package used to run the server and the client applications at the same time |
| Jsonwebtoken | Used for handling the use of JSON Web Tokens in Express |
| Passport | Handles user authentication and authorisation |
| Passport-jwt | A JSON Web Token authentication strategy for use with Passport |
| Validator | Used to handle different types of form/input validation |
| Firebase | Used to upload an image to a firebase store and retrieve the URL of that image for saving in the database |

#### Client-side

|  |  |
| --- | --- |
| React | The React JavaScript library |
| Axios | Used for making AJAX requests to the server |
| Classnames | Conditional class names in React |
| Redux | Used to manage application-wide state |
| Moment.js | Parse dates to human readable strings |
| Jwt-decode | Decode JSON Web Tokens into usable JavaScript Objects |

### 2.2.5. Security Architecture

User Authentication and Authorisation

“**Authorization**: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.” (jwt.io, 2018)

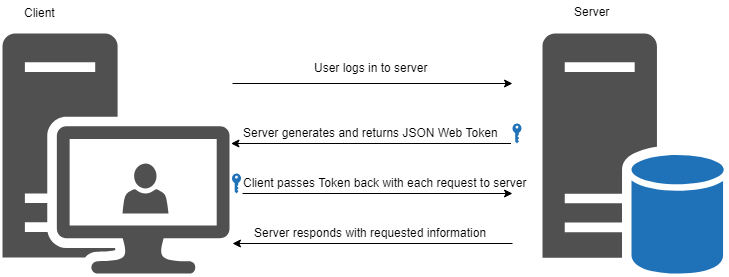
When a user logs in to the website, the server-side application will create a JSON Web Token which will act as an identifier which the client-side application will attach to each request sent to the back-end, to let the server know that the current user is logged in. This will let the application know which routes the user can access and which functionality they have access to.

Fig. 2.2.5.1

### 2.2.6. Communications Architecture

There are several different types of communications that will be performed in the overall application. The front-end application will communicate with the back-end application via AJAX requests. The front-end will send form data and user credentials, for logged in users extra headers will be sent in the form of an “Authorization” header containing the users JSON Web Token for authorisation.

The server sends two types of information to the client, the HTML, CSS and JavaScript files needed to load the application itself, and for further requests, JSON data which will be in response to any requests the client asks of the server.

Furthermore, more communication take place inside the front and the back-end applications.

In the front-end, information will be communicated in the form of “props” and “state”. Props are pieces of information passed from one component to another in React. State will be handled by Redux, and will receive information from the React app in the form of actions and reducers, which are used to update the current application state. The state can then be accessed and used by any component application wide, with the information from the state being accessed via props.

In the back-end, the express application will communicate with the MongoDB instance via the Mongoose Object Document Mapper.   
The application will employ database models, which are created using Mongoose Schema objects. As an example, the User model is represented as the following Mongoose Schema shown in Figure 2.2.6.1.

The contents of the image are contained in a single file, which is exported a the end of the file. This means that the User model can be imported to any other file in the server-side application and the full Mongoose functionality can be used to connect to the User document in the MongoDB database.



Fig. 2.2.6.1

### 2.2.7. Performance

Node.js is good at multitasking. It is single-threaded, non-blocking, and asynchronous. Therefore it can process multiple tasks concurrently in one thread, instead of queueing them. (Ciszewski, 2018)

On the client-side, the performance of the React application will be extremely fast. As a user navigates to a new web page, the page is simply loaded instantly with React using JavaScript, without having to make new requests to the server for HTML and any associated files inside the HTML document. The client may simply make a request to the server for some JSON data to update the already loaded UI.

### 2.2.8. System Design

#### 2.2.8.1 Use Cases

Use cases for the Recipe Book application can be found in the Requirements Specification documentation in section 3.

#### 2.2.8.2 Database Design

The application employs a MongoDB instance, which is hosted on mlab.com, as the database used to persist out data. MongoDB is a NoSQL database, which differs from a SQL database in that it stores data in “documents”, which resemble JSON data in how they are structured. This is an excellent choice for a full-stack JavaScript application as the persistent data will be stored in a format that is almost identical to that consumed by the application.

The application will use Mongoose, a JavaScript based ODM (Object Document Mapper), which is used to create database schemas in order to easily query and manipulate data in the applications database. An example of the mongoose schema can be seen in figure 2.2.6.1.



Fig. 2.3.2.1

An example usage of Mongoose being employed to register a new user in the application is shown above in figure 2.3.2.1.

The “User” model is imported at the top of the file, and is used at the beginning of the figure. A simplified version of the sequence of events represented here in JavaScript code is as follows:

Check in the “User” document for a user that has the email address which has been submitted via the form.

If such a user exists already, respond with an error informing the user.

If such a user does not exist, construct a new instantiation of the User model, using the data received from the front end via the input form.

Bcrypt then goes on the generate an encrypted version of the password received. Once the password has been encrypted, the user is then persisted in the database.

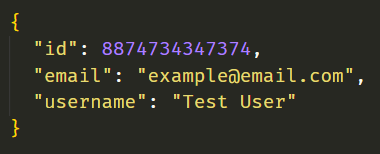
A similar sequence of events will take place, minus the encryption, for the recipes and comments stored in the application database. Once a user, recipe or comment etc is found in the database, the developer can make any changes he/she choose to make to the user object, and then save that object back to the database.

### 2.2.9. Data Conversions

Data transferred between parts of the application will take the form of JavaScript Object Notation. This is a data display format, styled almost identical to how an object would appear in JavaScript. Data requested from the database will be sent as JSON to the client side, where it will be parsed into a usable JavaScript object, or array of objects.

Similarly, when form input data is sent to the server from the client side, it is converted from raw form data into JSON and sent to the server, which parses the information into usable object to be manipulated and eventually stored in the database.

Below is a simple example of JSON information.



### 2.2.10. Application Program Interfaces

An API is a type of interface for communicating with an application or service from the outside. An example would be a weather API. An outside application or user would send a HTTP request to the API, perhaps requesting local weather data, and the weather API would receive the request, gather the needed data and respond with the requested data. Usually in JSON or XML format.

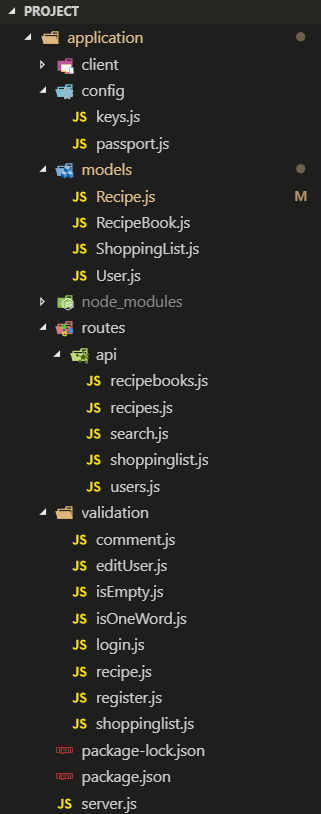
The Recipe Book application will use a database hosted at mLab to persist data, and therefore will communicate with mLabs API via the URL to the application’s database, including a database user credentials for authorisation.

## 2.3 Implementation

As the application is split into client and server side applications, the two will be described separately before describing how they connect.

Fig. 2.3.1

#### Server

The server is a RESTful API. Which is to say it offers an interface for interacting with persistent data, which utilises the HTTP verbs (GET, PUT, POST, DELETE) in order to affect those changes.

The server side application itself is split into modularised folders in order to increase efficiency during development and to make the code easier to read and understand.

As per the image to the left (Fig. 2.3.1), each folder contains the main modules of the application.

From the top, the “client” folder holds the entire client-side application which will interact with the server.

The “config” folder contains api keys or secret keys for use throughout the app.

The “node\_modules” holds the application’s dependencies.

The “routes” folder contains a single “api” folder which handles the routes. For example, the users.js routes will handle any request to /api/users. For example /api/users/register, /api/users/login etc.

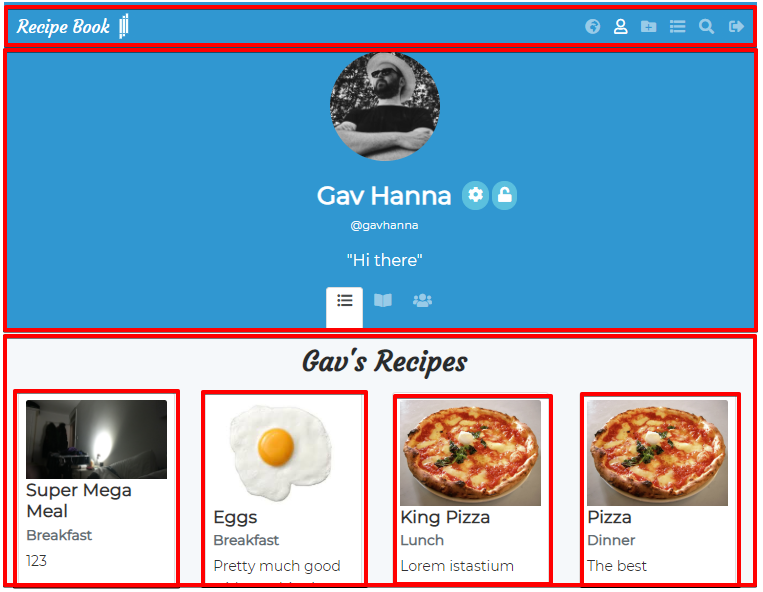
The “validation” folder contains files which handle the validation of all inputs on the client-side. For example the register.js file would take the form input from the /api/users/register body and validate whether the input exists, whether the email address is valid, passwords match etc.

If validation fails it returns responds to the request with errors in JSON format, if it passes it creates the user and responds with the user data.

#### Client

The client application is created using React, a JavaScript library for creating dynamic user interfaces. The state of the application is handled by Redux, an application wide state management service. Thus the description of the client app will be broken down further into these two separate major parts.

##### React

React is a component based library. This is extremely useful as any section of the UI of a web application can be broken down and separated into its own component and can be used and reused wherever needed. Components can either be created using classes, a data structure used for holding all required information about a particular component, it’s current state as well as its functionality, or functions, for “dumb” components that simply display information without storing its own state or functionality.   
Fig 2.3.2  
  
The above image outlines the components visible on this particular page in red. As can be seen in the lower half of the page, the individual recipe components are reused as many times as needed using just the one recipe component. These components are also children of their parent container component. All visible components are in turn children of the overall “App” container component which holds the entire application.

#### Redux

At the time of the creation of this application, the current stable version of React (16~) does not ship with its own application-wide state management system. Therefore all state is passed between components and can often be passed down many generations of children in order to affect change. This can quickly grow to be very difficult to manage in larger applications, this is where Redux comes in. Redux allows any component to “connect” to the single source of truth data store, which means any single change to the data is immediately passed to any component which has connected to that store. While the benefits are clear, the implementation of Redux is quite complicated and may be best explained by describing a typical life cycle of a change to the Redux store.

A change begins with an “action”, these are typically calls to external APIs for data. These actions are associated with a “type”. The types are created by the developer and associated with each action as a form of type-checking. The response from that actions is then dispatched to a “reducer”, this reducer then affects changes to the current state based on data received from the action.

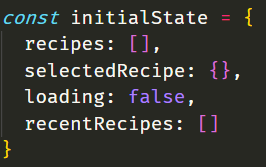
The following is an example of what happens when a user visits another users profile page and the application requests that particular users list of created recipes.

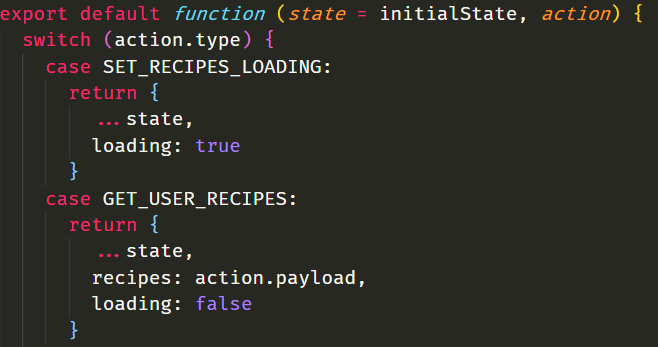
#### Step 1

Fig. 2.3.3

The user visits another user’s profile page. The component calls the getRecipesByUsername action, which is a function. The function takes in the username of the target. A get request is then sent using the axios npm package to the endpoint /api/recipes/user/:username. When the response is received, the data is then dispatched to the reducer along with the type of the action, in this case “GET\_USER\_RECIPES”.

#### Step 2

Fig. 2.3.4

Fig. 2.3.5

The above screenshots are from the recipesReducer file. Figure 2.3.4 shows the “initial state”. This will be populated by data received from the actions. Figure 2.3.5 shows the reducer function, it takes in the current state and the received action. This function uses a JavaScript switch to check the type of the action received. In this case it is the GET\_USER\_RECIPES type, which we sent with the action. The function then returns the state, setting the “recipes” array in the state, to the data received from the server after requesting the recipes of the user of the profile we are viewing.

#### Step 3

The user profile component then receives the newly updated state containing the users recipes and displays those recipes on the page.

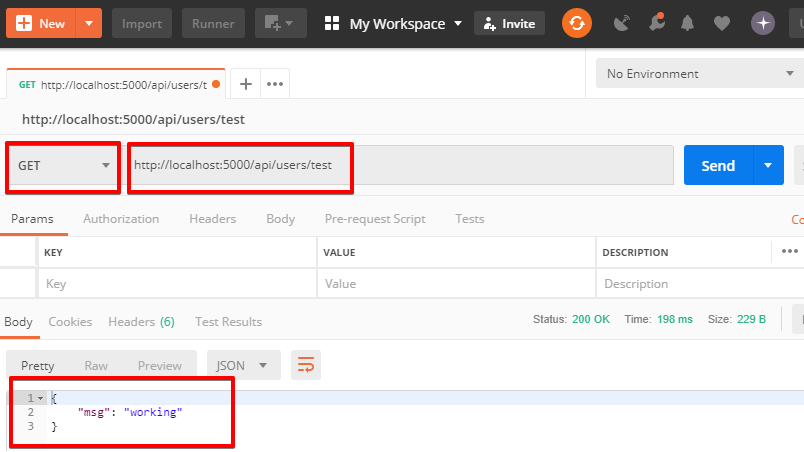
## 2.4. Testing

As with previous sections, the testing of the application will be broken down into two major components: server and client.

### 2.4.1. Server

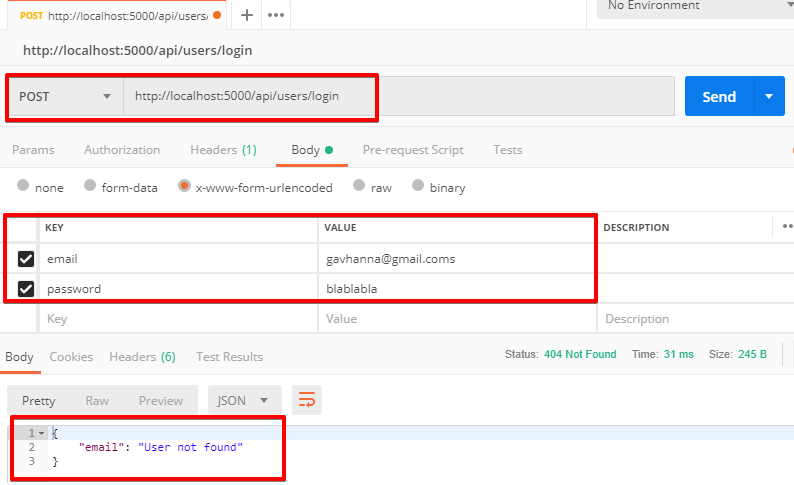
Development of the application began with the server, and as their was no client, or “visual” aspect, of the application, testing of the API took place using the Postman tool.

Postman is used to test API endpoints by making HTTP requests and displaying the returned data. It is by this method that each of the server’s endpoints was tested.

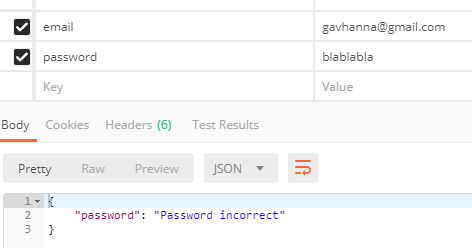
Fig. 2.4.1.1.

The above figure shows an example of a GET request sent to the /api/user/test route in the application, along with it’s JSON response. Key parts of the image are highlighted in red.

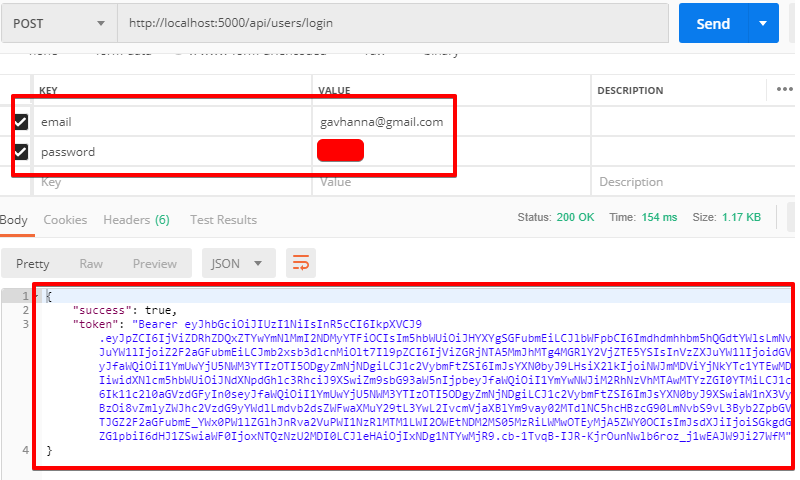
The HTTP verb can be changed, the routes can be edited like a browser’s navigation bar and authorisation, headers and body information can be added to the request.

Fig. 2.4.1.2.

The above figure shows an example of testing the login POST route. The email and password fields have been added, with an intentionally incorrect email address. The response is shown at the bottom of the image as “email: User not found”. Thus the application correctly detects that the email does not connect to a known user in the database.

Fig. 2.4.1.3.

The above figure has amended the email address, however now the incorrect password has been checked and an error returned by the server.

Fig. 2.4.1.4.

Finally, the correct email and password have been submitted, from which the server returns a success message, along with the JSON Web Token which will be used by the client to confirm a logged in user when connecting to the server side application.

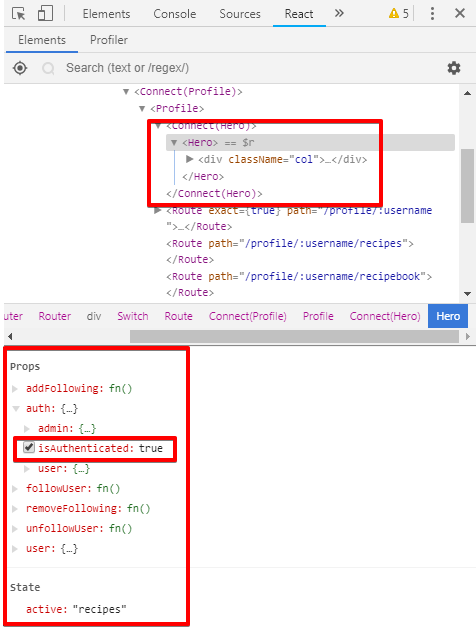
All other routes are tested in exactly this fashion, testing the bearer token, all form input data and requests for information.

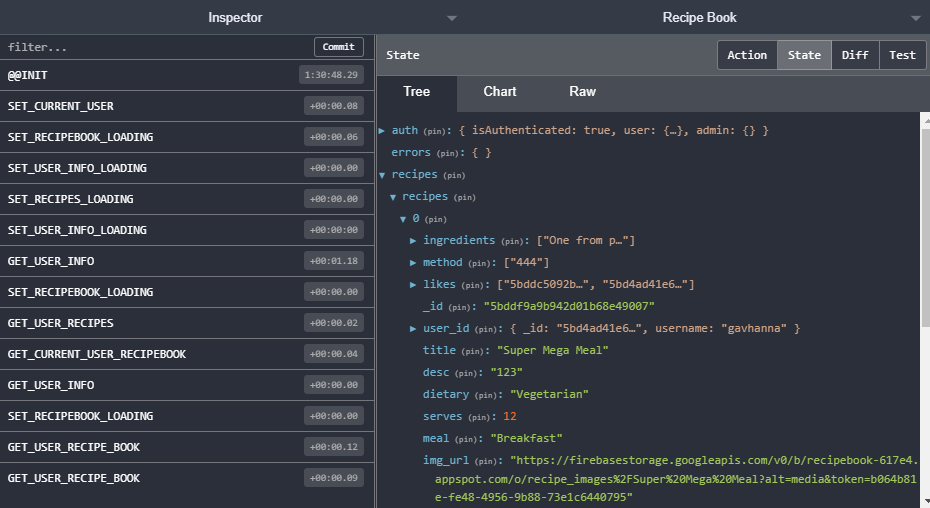
### 2.4.2. Client

Testing of the client-side React/Redux application takes place using three key technologies. The developer console, the React Devtools and the Redux Devtools.

The developer console is often used to console.log any information that might be causing breaking issues or simply to see what data is being received from a request to the server and decide how best to implement the receiving of this data.

The following image (fig. 2.4.2.1) shows the React dev tools in action. The “Hero” component is selected in the top half of the screenshot, and both the components props and state are shown in the bottom half. The props and state and even be modified in real time using these tools to test the effects on the application directly. As can be seen by the highlighted “isAuthenticated” prop. This is a Boolean value which can be set to true or false simply by clicking the checkbox in the devtools. This is extremely useful for testing the effects of data on your application.

Fig. 2.4.2.1

Fig. 2.4.2.2

The above figure shows the Redux dev tools in action. The left section of the image shows the actions that are being sent to the reducers along with some timing information related to each action. The right side shows the applications current application-wide state.

The recipes object has been expanded to give an example of the nested nature of the state. The recipes object contains recipe array, of which the 0th element is expanded to show the information contained in that recipe.

## 2.5. Graphical User Interface (GUI) Layout

Image and information about the GUI can be found in the Product Design Specification document at section ***6.4.5 User Interface Design***. However, as changes have been made during development, updated or entirely new pages/GUI elements will be described in this section.

#### Feed Page

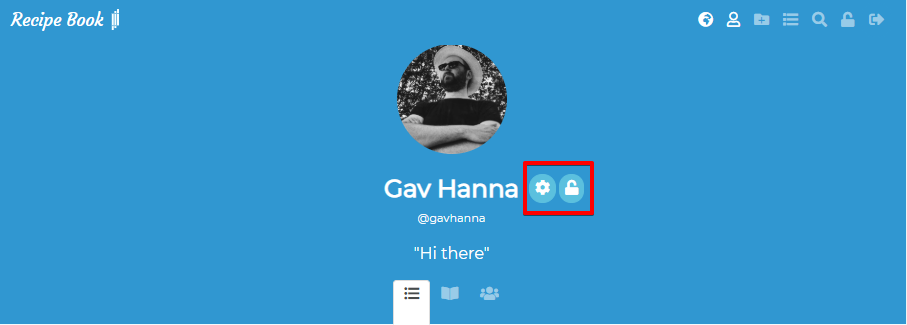
Fig 2.5.1

Firstly, the above image illustrates some additions to the navigation menu on the top right corner of the window. The “earth” icon now brings the user to the Feed page. The list icon now brings the user to their personal shopping list. The magnifying glass brings the user to the search page. The lock icon is only available to admin level users and takes the user to the admin panel.

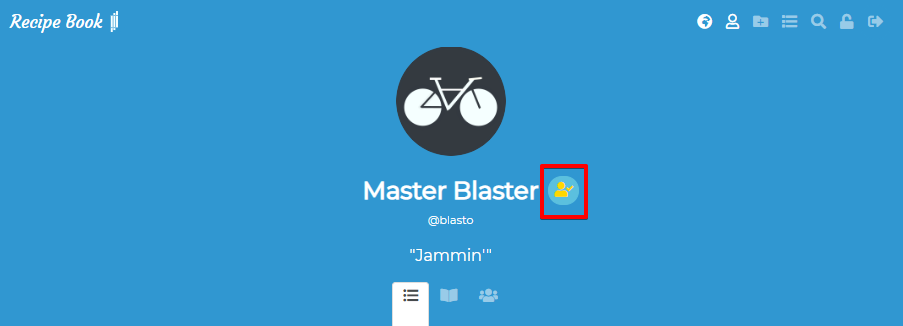
Now the feed page itself, the left side of the screen (on desktop computers, this element is hidden on mobile) now holds a user profile card, containing information about the currently logged in user. Clicking on this takes the user to their personal profile page.

Another addition is the small heart icon with a number beside it. This represents the amount of likes the related recipe has. Clicking on this icon will pop up a modal box with the usernames of the users who liked the recipe, which act as links to their respective profiles.

#### Profile Page

Fig. 2.5.2

The profile page has remained largely unchanged through development, however now the user’s personal profile page now has a button to edit their personal information (name, username, blurb, profile pic). The padlock icon only appears for admin level users and takes the user to the admin panel.

Fig. 2.5.3

The above image shows the available icons when visiting another users profile. The button indicates that the logged in user currently “follows” this user. Clicking on this button will unfollow the user.

#### Recipe Page / Shopping List Page

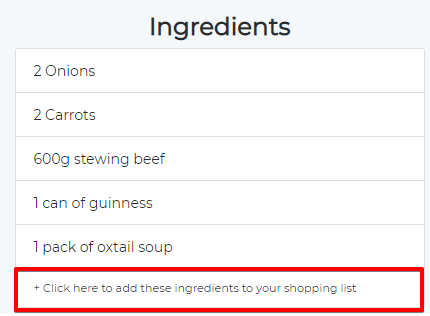
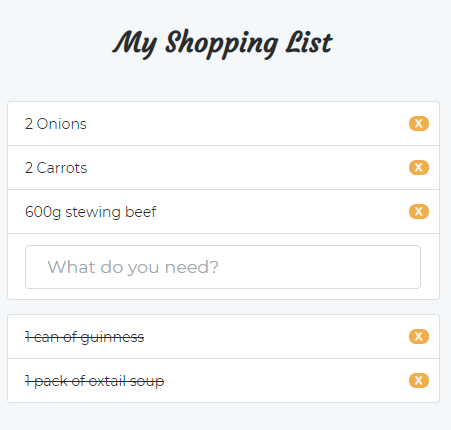
The recipe page has remained largely unchanged, however with the addition of the shopping list component, now the ingredients section offers the user the option to add the current recipes ingredients to their shopping list.

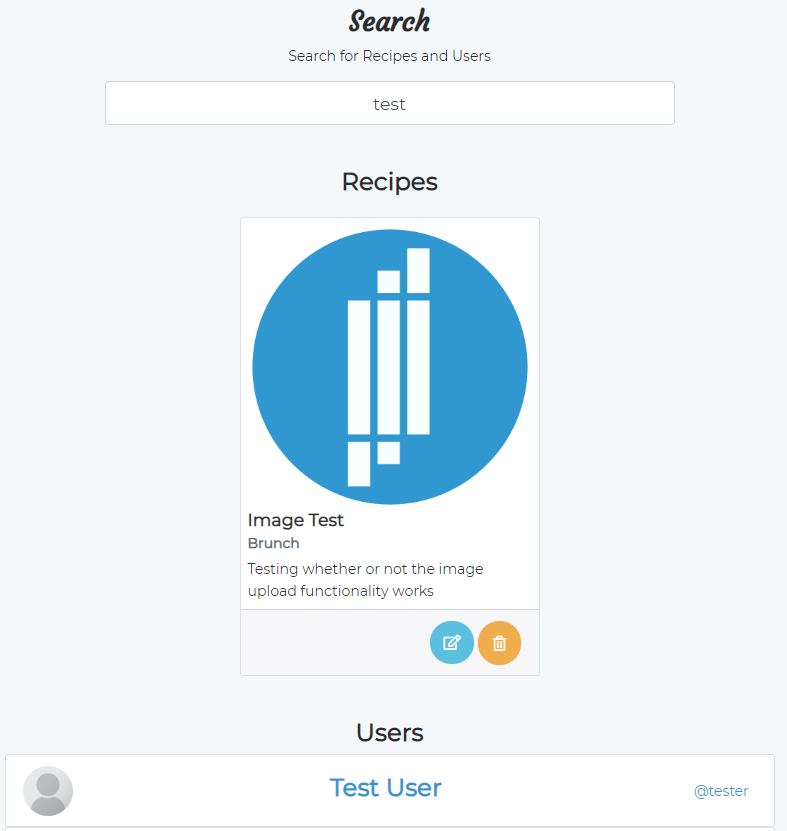
Fig. 2.5.4

Fig. 2.5.5

The image at figure 2.5.4 shows the “add to shopping list” button at the end of the list of recipe ingredients. Upon clicking on this button, the user is taken to their shopping list which has been populated with the added ingredients.

From here the user can click on each item to move it down to the “already picked up” list. If the user clicks on one of the “strike through” list items it will move back to the upper list. The small x buttons will entirely remove the item from the shopping list. When all items are struck through, a button appears to clear the shopping list entirely.

#### Search Page

Fig. 2.5.6

The search page has one input by which the Users and Recipes are searched. The results are displayed recipes first, users second.

#### Admin Panel

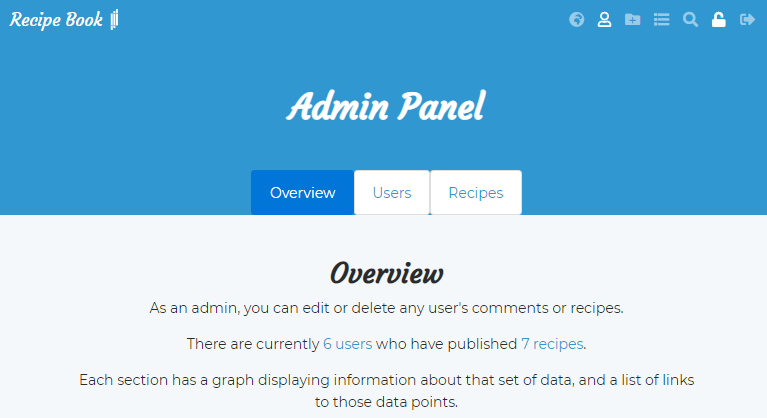
The admin panel was added late in development to offer certain users the ability to moderate site content. The admin panel also displays some basic site information in graph form.

Fig. 2.5.7

The above image displays the overview of the admin page with a brief explanation to the user.

The following image (fig. 2.5.8) shows the “Users” section of the admin page, complete bar chart displaying some information about site data. A list of users and their admin rights is also visible. Admin rights can be changed by clicking on the “False” button which will change the right to “True”, and vice versa.

The following image (fig. 2.5.9) shows the “Recipes” section of the admin page, which displays a pie chart of the percentage breakdown of recipes by meal type posted on the site, followed by a list of actual recipes for quick navigation.

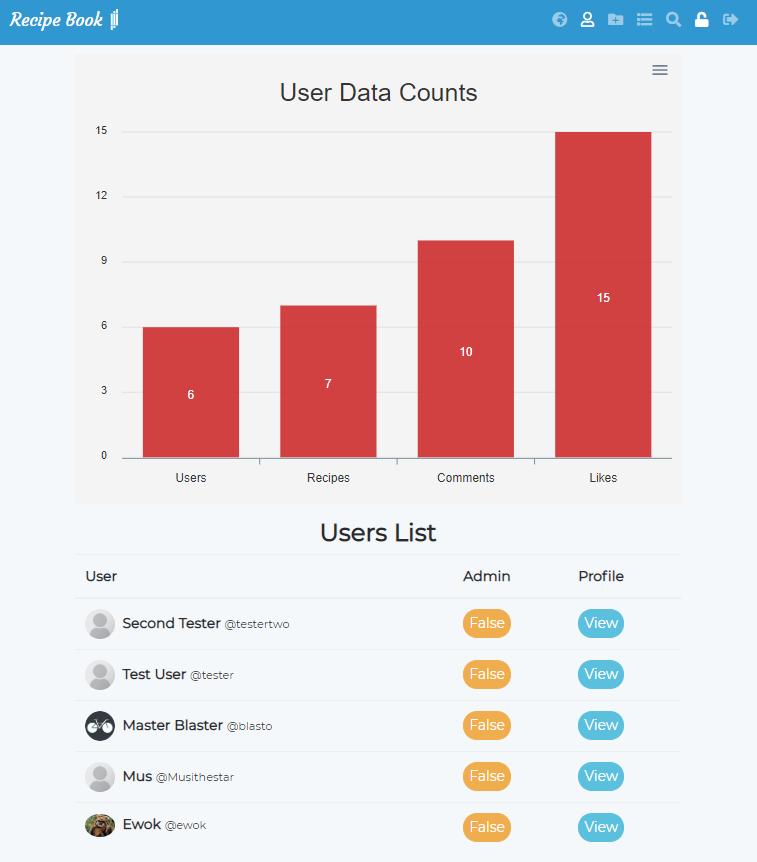


Fig. 2.5.8

Fig. 2.5.9

As the application attempts to mimic the behaviour of a native application, the application can be opened even without an internet connection. This application will show an “No Connection” scree to inform the user of the lack of connectivity, and will resume normal processes once a connection is made available.

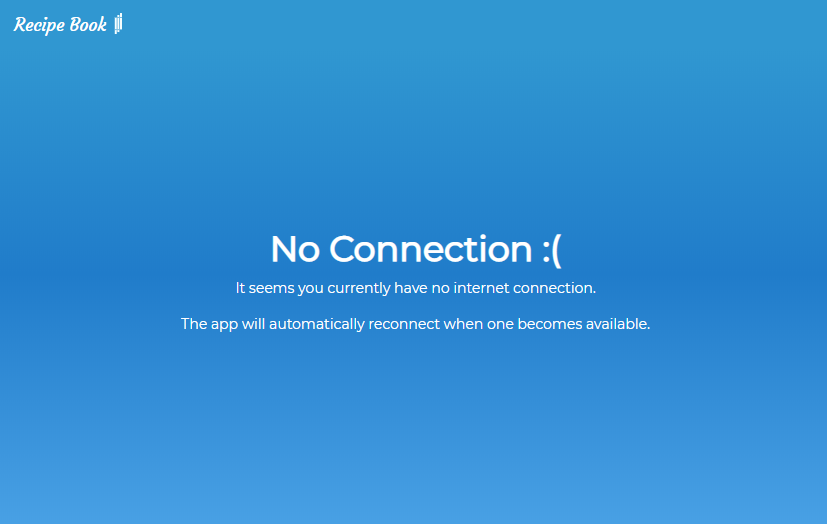
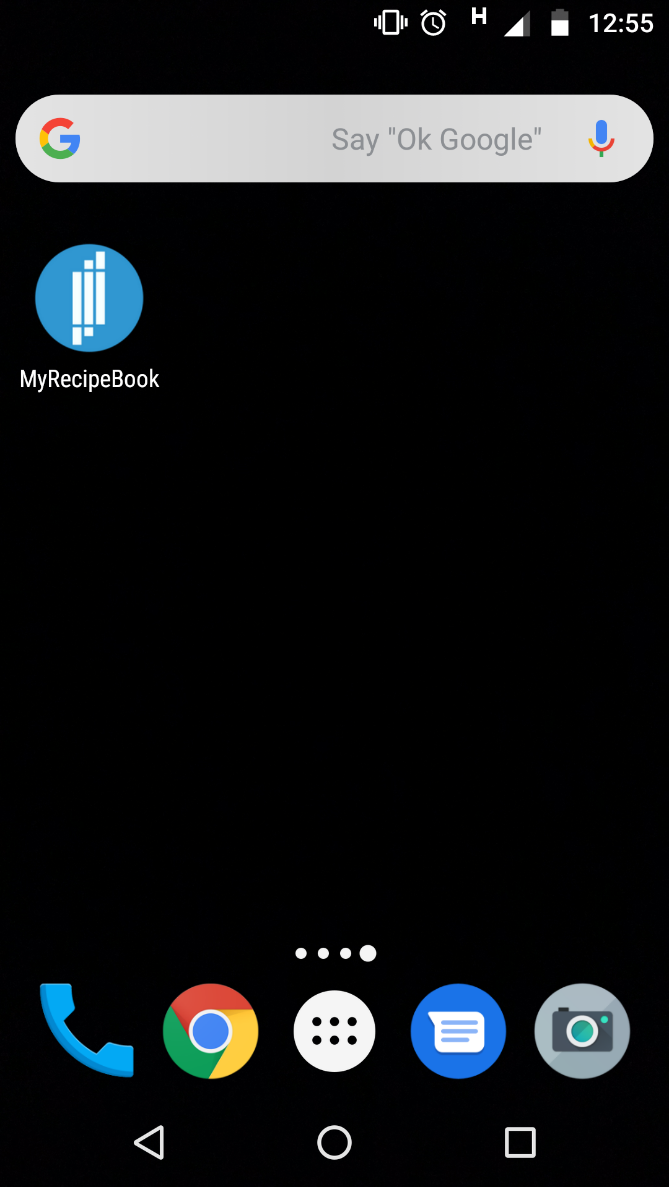
This is achieved through the npm package “react-detect-offline” which provided Online and Offline components. Anything nested inside an Online component will only be shown online, and vice versa. In this application the package is implemented by wrapping the entire application in an Online component, while wrapping a specially created OfflineComponent in the Offline react-detect-offline component. This offline screen is displayed below in figure 2.5.10

Fig. 2.5.10



|  |  |
| --- | --- |
| Fig. 2.5.11 | Fig. 2.5.12 |

Finally, when the app is saved to the home screen of a mobile device, it can be launched by clicking on it’s icon (fig. 2.5.11). When opened, there is a short loading screen (fig. 2.5.12) with the applications name and icon before the app is finally functional.

## 2.6. Customer Testing

During customer testing, several UI/UX issues were identified and resolved. Some examples follow:

In figure 2.5.1, the small “heart” symbol was originally unclickable and simply displayed the number of likes. However, customers often presumed the icon was clickable and were surprised to find no response to clicking. Therefore a popup modal was added to display the usernames of the users that liked the particular recipe and link to their individual profiles.

On the user profile page, under the recipe section, each recipe originally had a “View” button which had to be clicked in order to navigate to that particular recipe page. However users often simply clicked on the image, or anywhere on the card itself with the intention of navigating to that particular recipe. Therefore the upper portion of the card itself now acts as an anchor element to bring the user to that recipe page.

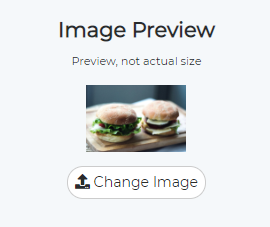
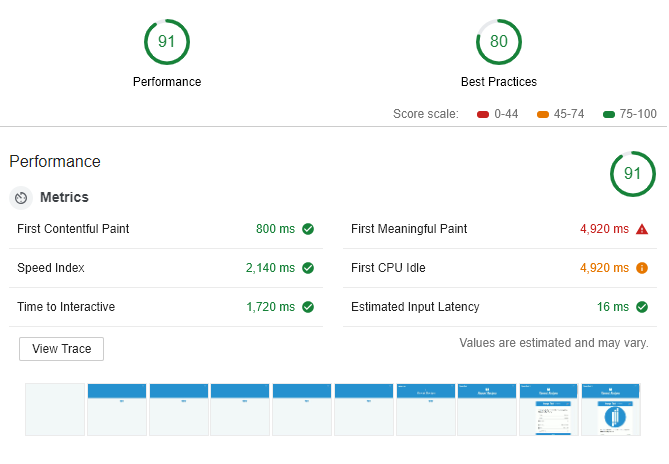
On the user registration and recipe creation pages, when an image was selected for upload, originally there was no preview image displayed to let the user know which image was being used in the recipe/user creation. A way was therefore devised to show a small preview of the image before the actual upload takes place, as can be seen in figure 2.6.1.

Fig. 2.6.1

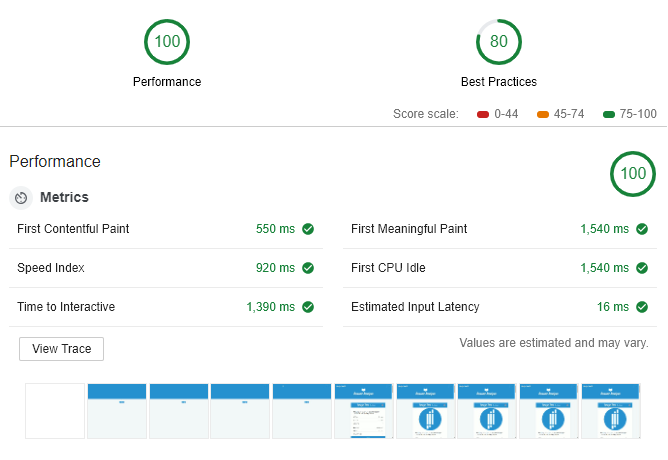
In early versions of the application, delete buttons provided no final warning when deleting items from the database. For e.g. when deleting a recipe, the deletion would be finalised on one click of the delete button. Users requested a confirmation box in order to prevent accidental deletions of data.

## 2.7. Evaluation

The Google tool “Lighthouse” was used in the evaluation of the application in a variety of ways. The initial test was done to test the performance and best practices of the application in an unthrottled and un-cached desktop environment.

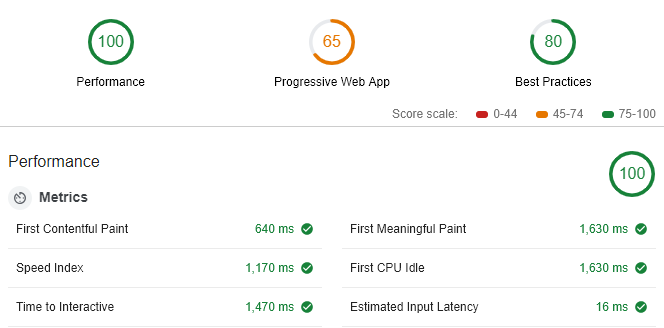
Fig. 2.7.1

Initial test gives generally positive results, with a clearly slow first meaningful paint. This can be very useful information when deciding what further action to take on the application, and what next steps are a high priority.

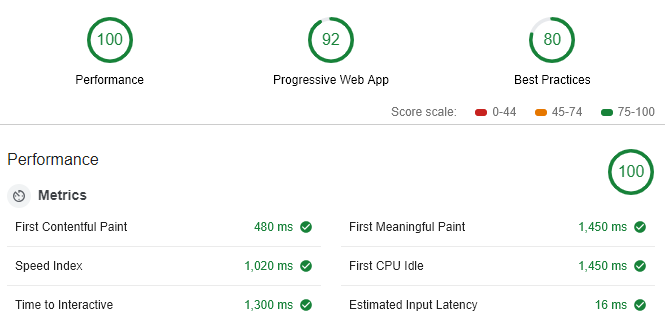
Fig. 2.7.2

The above image shows the results from an unthrottled loading of the page with the cache enabled. The clear improvement over the un-cached page load tells us that the first time the user loads the page will be quite a bit longer than successive page loads.

#### Progressive Web App Test

Fig. 2.7.3

The above image gives the PWA score when serving from an un-cached resource. The low score is due to a 16s loading time on a 3g network. This is very slow and should be a high priority when improving the application.

Fig 2.7.4

The above image shows the PWA being served from cached assets. Again we see that the initial page load takes a considerably longer time to reach a usable state when not initially being served from cached resources.

The developers final evaluation of the application in it’s current state is that while the application functions as intended, and works very well after the first download, there is room for improvement in the performance of the application. The major stumbling block in holding back page load times are the large image sizes of the user submitted imagery. A way of reducing image size on upload should be implemented in order to speed up initial page loads.

# 3. Conclusions

In developing a Progressive Web App using modern JavaScript technologies, several key advantages and disadvantages have become apparent.

Advantages

* PWAs offer the developer far more flexibility in terms of connection availability, by being able to directly control every request sent to the server, and by being able to store data in cache and a browser based database, once a web site is downloaded, subsequent visits to the page can be lightning fast as cached data can be served before going to the network.
* PWAs offer the ability to provide push and desktop notifications to the client which has the possibility of improving the user experience.
* A PWA cab be saved to the home screen of mobile devices and, by opening the app this way, can take on the appearance and behavior of a native application.
* Developing a PWA can save a lot of money and time as the web and mobile versions of the app are actually the same code base and do not require the time and money needed to develop a native iOS and Android application.
* PWAs are very quick and easy to install, as by simply visiting the URL, the user is prompted to add the application to the home screen, removing the necessity of visiting the app store.

Disadvantages

* Push notifications can become annoying to general users once many websites begin to implement the technology and request the users permission to send notifications.
* The notification to alert the user to the fact that they can save the application to their home screen might also annoy some users that may simply want to browse the mobile version of the site.
* PWAs cannot access all functionality of a device that a native application might be able to access, e.g. PWAs cannot access the devices Bluetooth, ambient lighting settings, advanced camera controls, contact, proximity sensors among other features (Khan, 2018).

#### Opportunities

Due to the modular nature of the development of the application, editing existing, or adding new functionality to the application becomes far easier. Adding an entirely new aspect to the application might require adding new routes to the API, creating new React components and new Redux actions/reducers.

# 4. Further Development or Research

As mentioned in 3. Conclusions, the applications is designed in such a way as to be easy for a developer to implement new features or change existing features.

The application itself could be expanded to include data from external APIs, for example a food information API could be brought it to give information about different ingredients for users that are unfamiliar with certain foods.

A private messaging service could be implemented to allow users to privately communicate with one another.

A system of alerting the user to new or updated site content could be implemented, perhaps with email notifications.

The application currently uses Bootstrap to provide the CSS and basic layout functionality (dropdowns, modals etc), however given more time- a fully application specific design could be implemented which could improve the overall UI/UX, visual appearance and reduce the “generic” feeling associated with the Bootstrap framework.

# 5. References

Ciszewski, B., 2018. *Nodejs Performance.* [Online]   
Available at: https://www.netguru.co/blog/nodejs-performance-web-application-benefit  
[Accessed 26 October 2018].

Ebizmba, 2018. *Top 15 Most Popular Recipe Websites | May 2018.* [Online]   
Available at: http://www.ebizmba.com/articles/recipe-websites  
[Accessed 29 November 2018].

Google Trends, 2018. *Google Trends.* [Online]   
Available at: https://trends.google.com/trends/explore?q-recipe  
[Accessed 30 November 2018].

Heroku, 2018. *What is Heroku.* [Online]   
Available at: https://www.heroku.com/what  
[Accessed 21 October 2018].

jwt.io, 2018. *JSON Web Token Introduction.* [Online]   
Available at: https://jwt.io/introduction/  
[Accessed October 21 2018].

Khan, U., 2018. *The Pros and Cons of Progressive Web Apps.* [Online]   
Available at: https://clutch.co/app-developers/resources/pros-cons-progressive-web-apps  
[Accessed 2 December 2018].

node.green, 2018. *Node ES2015 Support.* [Online]   
Available at: https://node.green/  
[Accessed 1 December 2018].

ReactJS, 2018. *React Browser Support.* [Online]   
Available at: https://reactjs.org/docs/react-dom.html#browser-support  
[Accessed 1 December 2018].

Searchvolume.io, 2018. *https://searchvolume.io/.* [Online]   
Available at: https://searchvolume.io/  
[Accessed 30 November 2018].

Statista.com, 2018. *Facebook - Statistics & Facts.* [Online]   
Available at: https://www.statista.com/topics/751/facebook/  
[Accessed 30 November 2018].

# 6. Appendix

## 6.1 Project Proposal

### 6.1.1. Objective

To create a social network web application based around creating and sharing cooking recipes. The application should be capable of being saved to a mobile phone as a PWA (Progressive Web App, a web application which can act as a native mobile app), with some offline capabilities: achieved by using a service worker with either an offline.html page being served when a connection is unavailable, or information being served from the browsers indexedDB API in the absence of a connection.

In order to create a smooth, fast and responsive user experience, the application will be created using the MERN stack. The MERN stack incorporates the following technologies:

* MongoDB – A NoSQL database based on JavaScript
* Express – A Node library designed for creating web applications and APIs
* React – A client side library commonly used for creating Single Page Applications
* Node – The JavaScript V8 engine taken from the browser and placed on the server

The application will be separated into two major components:

* The server – A RESTful API
* The client – A React application which will communicate with the server

The application should allow:

Users to be able to register and log in the application, create edit and delete their own recipes.

Users to be able to follow their favourite recipe creators and comment on their own and other users recipes.

Users to be able to save their favourite recipes to easily review.

Users to “like” other users’ recipes.

### 6.1.2. Background

#### Research

Recipe based websites are an extremely popular corner of the internet, with the top 15 recipe based websites ranging from 2.25 million unique monthly visitors for KraftRecipes, to 25 million visitors to AllRecipes (**ebizimba.com, 2018**).

The word “food” has an average monthly search count of 823,000, with “recipes” having 450,000 and “recipe” with 90,500 (**searchvolume.io, 2018**). In Google Trends, the search term “recipe” never dips below 50% “Interest over time” (**Google Trends, 2018**).

The popularity of recipe based searches on the internet, and the overwhelming popularity of social networks (Facebook alone has over 2,196,000,000 users (**statista.com, 2018**)), lead naturally to the combination of the two, of which there are many such websites currently available online. Two of the most popular are bakespace.com and cucumbertown.com. While these are technically social networks, they either focus more on simply being a recipe search engine, or acting as a blogging platform focused on recipes.

I believe there is a space for a fully realised recipe based social media web application in which people can follow their friends and share there particular cooking styles while interacting with and sampling/critiquing styles of their loved ones.

#### Conclusion

This knowledge, coupled with the statistics that suggest that mobile browsing has now taken over desktop browsing (**Eric Enge at stonetemple.com, 2018**), suggests there is a possibility for a successful recipe based social media application. I believe by building the application with it also being a Progressive Web App so that users can easily use the sites functionality on their phones in the same way as a native app, and also on the browser, a large userbase could be obtained.

### 6.1.3. Technical Approach

#### Research

As in Background, research was conducted on internet search query amounts to determine interest in searching and viewing recipes online, along with research into current social networks and/or blog style websites focused on recipes and the sharing thereof.

Research was also conducted on websites that focus entirely on recipes and providing recipe search functionality, for example **bbcgoodfood.com/recipes**, **easyfood.ie/recipes** and **goodfoodireland.ie/recipes** among others.

#### Requirements Capture

From this research the required recipe related data which would need to be stored in the database was formulated, along with potential recipe page layout design and structure. The Recipe database model would need a minimum of the following rows:

* Id (primary key)
* User\_id (foreign key)
* Name
* Cooking time
* Preperation time
* Serves (integer representing amount of people meal should serve)
* Ingredients (stored as an array of strings in MongoDB)
* Method (stored as an array of strings in MongoDB)
* Likes (array of user IDs)

#### Implementation

As mentioned in the Objective, the application will be split into two major components: the server and the client.

##### The Server

Built using node.js using the express framework to create a RESTful API. The server will store data in a MongoDB instance hosted at **mlab.com**. The server will interact with the database using the Mongoose ORM (Object Relational Mapper). Authentication will be handled with JSON Web Tokens using the jwtwebtoken npm package.

##### The Client

Built using React, starting with boilerplate created with the create-react-app npm package. The application will implement Redux to handle app-wide state management.

### 6.1.4. Project Plan

Please see attached MS Project Gantt chart for detailed view

#### Technical Details

As mentioned in previous sections, the application will be split up in two separate yet communicating applications, the frontend client application and backend server application. The “stack” that was chosen was this application was the MEAN stack.

#### Server

The backend of the application will be built using the Express.js web application framework for node. Express will be used to develop a RESTful API with endpoints for create, read, update and delete methods on objects stored in the database.

The database will be a MongoDB instance which will be hosted at mlabs.com. The Object Relational Mapper Mongoose will be used by the app to communicate with the database.

The following is a list of npm modules used by the server, and their purpose:

|  |  |
| --- | --- |
|  |  |
| body-parser | For parsing form data submitted from the client |
| express | The framework upon which the server application is built |
| jsonwebtoken | Used for handling JSON Web Tokens which will be used for authentication |
| bcryptjs | Used to encrypt and decrypt user passwords using hashes and salts |
| mongoose | An ORM to handle communication with the MongoDB instance |
| passport | Handles authentication and authorisation |
| passport-jwt | A passport authentication strategy using JSON Web Tokens |
| validator | A library of string validators and sanitizers, used for form validation |

#### Client

The frontend will be implemented using React, with redux to manage application state.

|  |  |
| --- | --- |
|  |  |
| axios | Used for making requests to the backend |
| classnames | Create conditional CSS class names inside React |
| Jwt-decode | Used to extract logged in user info from JSON Web Token |
| moment | Format dates to be human readable |
| react | Library for building interactive client applications |
| react-router-dom | Routing on the client side |
| redux | Application wide state management |

### 6.1.5. Evaluation

During the production of the backend of the website, continuous testing will take place using **Postman**. Postman allows developers to test APIs by sending and receiving requests using HTTP verbs and any header or other data they would like to send. This software will therefore be used during the entire development of the server.

During each endpoints creation, requests will be sent to test the endpoint. For example:

If we are creating the user register endpoint, POST data will be sent to the /api/users/register URL for testing.

Once the entire backend has been completed a further suite of tests will confirm whether each endpoint is functioning.

During the production of the frontend of the website, initially data will be requested from the server inside a React component. This will be used for testing the communication between frontend and backend of the application. Once testing has been completed, requests will then be made from inside an action, which will be dispatched to a reducer and added to the application state.

Once this initial testing phase has been completed the system can then be deployed to a server for further testing before release.

### 6.1.6 References

Ebizmba 2018, accessed 04/10/2018,   
<http://www.ebizmba.com/articles/recipe-websites>

Searchvolume.io, accessed 04/10/2018,  
<https://searchvolume.io/>

Google Trends 2018, accessed 04/10/2018,  
<<https://trends.google.com/trends/explore?q-recipe>>

BBC Good Food Recipes, accessed 04/10/2018,  
<<https://www.bbcgoodfood.com/recipes>>

Eric Enge, April 27th 2018, accessed 04/10/2018,  
<<https://www.stonetemple.com/mobile-vs-desktop-usage-study/>>

Easy Food, accessed 04/10/2018,  
<<http://easyfood.ie/home/recipe/>>

Good Food Ireland, accessed 04/10/2018,  
<<https://www.goodfoodireland.ie/recipes>>

mLab, accessed 04/10/2018,  
<<https://mlab.com>>

Postman Application, accessed 04/10/2018,  
<<https://www.getpostman.com/>>

## 6.2 Project Plan

For the project plan, please see attached Gantt chart.

## 6.3. Requirement Specification

### 6.3.1. Introduction

#### Purpose

The purpose of this document is to document the requirements for the development of a recipe based social network web application, specifically designed to offer a space on the web for publishing self-made recipes to share and also to follow your favourite recipe creators and discuss the recipes themselves.

The application is intended to be used by people with an interest in recipes, either simply by browsing some interesting user created content or by sharing and discussing their own and other’s content.

Each user has a specific “Recipe Book” attached to that user, whereby they can save a collection of recipes that they enjoy. Users can add or remove recipes from this book at any time. Recipe Books can then be shared with and viewed by other users.

#### Project Scope

The scope of this project is to design and develop a responsive progressive web application that allows users view user created recipe content and search by key words to view specific types of recipes. Users can also choose to register and create an account on the application which will allow them to create their own recipes, comment on their and others’ recipes and follow their favourite content creators.

Users will be able to directly link to a personal public profile which will display their personal recipes and link to the profiles of the creators they follow.

An unregistered user will be restricted from commenting on recipes, from following creators content and from saving any favourite recipes. However they will be able to view any recipe by searching or direct linking.

The application will be a PWA, a progressive web app, meaning that it will be able to be saved to the home screen of a mobile device and act in a similar fashion to a native application. It will have some level of offline capability, either by serving content from the in-browser database indexedDB or by rendering an “offline” page to let the user know there is no available connection.

The application will also be an SPA, a single page application, having the front end built with React, which will enable very responsive application navigation via the router. Data will be sent and received to a node/express back end application and updated in the client’s view via a Redux store utilized by React.

#### Motivation

The main motivation for developing this application is to attempt to provide a social hub for people that enjoy sharing recipes, both their own and sampling other people’s recipes and meal ideas. A secondary motivation is to get a deeper understanding of the development of a full stack application with separate but communication front and back end applications, including adding the functionality associated with a PWA.

#### Project Objectives

To create a functional and easy to use social network application for users to store and share their favourite meals, try other users meals and add their opinions. When the application has been completed it will be uploaded to Heroku to be used by the public. A feature could be added by which users could suggest improvements or submit bugs that may have slipped through the testing phase.

#### Project Expectations

The project could be monetized should the application become popular enough to warrant upgrades to the hosted database plan.

#### Definitions, Acronyms and Abbreviations

|  |  |
| --- | --- |
| Acronym | Definition |
| PWA | Progressive Web App (Kapoor, 2018), web application that can mimic the behaviours of native applications (offline capability, start app extremely fast from cached data). |
| SPA | Single Page Application is a website that re-renders its content in response to navigation actions (e.g. clicking a link) without making a request to the server to fetch new HTML (Sherman, 2018) |
| JS | The JavaScript programming language |
| CSS | Cascading Style Sheets |
| HTML | Hyper-Text Mark-up Language |
| URL | Uniform Resource Locater, essentially a web address |

### 6.3.2. User Requirements Definition

The system must allow unregistered users to browse recent recipes, search recipes by specific keywords and view recipe creator profile pages.

The system must offer an ever present link to the Register and Login pages in the navigation bar and on the landing page of the application.

Once a user has register they will be offered the chance to submit their own recipes, to be able to view their own personal profile page. Users will be able to find and follow other users and to comment on and like other users recipes. Each user can then add recipes that they enjoy to their very own “Recipe Book”, which can be shared with and viewed by other users.

Users should be able to edit their previously submitted comments or previously published recipes.

A mobile site visitor should be offered the option to add the application to their home screen, regardless of their devices operating system, which will act as if it was a native application on that device. If a user opens the application without an internet connection, they should be notified and offered a chance to reload in an attempt to reconnect.

The application will have a responsive navigation bar at the top of the screen to permit easy navigation of the application.

### 6.3.3. Requirements Specification

A simple, coherent and friendly user interface is the highest priority for the application. The applications interface should be self-explanatory and do exactly what the user would expect it to do, having used other social networks previously, or never having used them at all. Buttons and navigation should be clear and their destinations obvious.

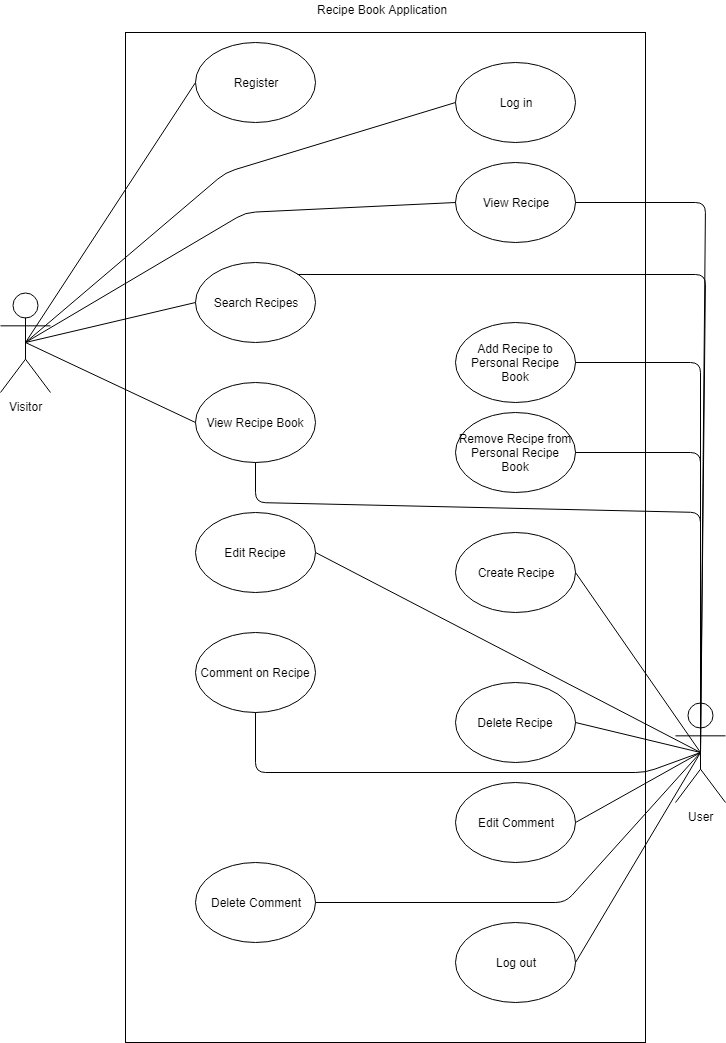
When a user sends or received data from the server side of the application, they should be updated as to the changes made in an understandable and non-invasive way.

#### Functional Requirements

The following functional requirements represent the minimum viable functionality by which the application could be considered usable.

* Upon reaching the home page, the content of the page should render correctly. Displaying information about the application and links to Register and Login functionality.
* Register and Login forms should be fully validated.
* Users should be able to view a selection of recipes without having to be logged in or registered.
* Users should be able to register for an account using a valid email address and password. The username should be anything they choose between 3 and 30 characters long.
* Logged in users should be able to view their personal “user” page, containing any previously submitted recipes.
* Logged in users should be able to comment on their and other users recipes.
* Logged in users should be able to create a new recipe, and edit any previously created recipe.
* Creating or updating recipes should have fully functional validation.
* Logged in users should be able to “follow” other users and see their recipes on the initial users homepage, or “feed”.
* Logged in users can add any viewable recipe to their own “Recipe Book”.
* Logged in users can remove recipes from their own “Recipe Book”.
* Logged in users can view other users Recipe Books.
* Once a user logs out, any information relating to the previously logged in user should be removed from the specific client side rendering of the application.

#### Use Case Diagram

The following Use Case Diagram represents functionality offered to both a non-logged in user (represented as Visitor) and a logged-in user (represented here as User).

#### Requirement 1: User Registration

#### Description & Priority

A user can remain a “visitor” on the website, meaning they can peruse and search for recipes, but cannot create a profile, follow creators or comment on recipes until they register for an account. To unlock full functionality, a visitor must register and become a user.

#### Use Case: Register

**Scope**

The scope of this use case is to allow a user to register an account with the application.

**Description**

This use case describes the steps to be taken in order to successfully create an account on the application.

**Flow Description**

**Precondition**

The system is idle

**Activation**

This use case starts when a user clicks on the “Register” link.

**Main Flow**

1. The user navigates to the “Register” page on the web application.
2. The user enters a username of between 3 and 30 characters.
3. The user enters a valid email address.
4. The user enters a valid password of between 6 and 30 characters.
5. The user confirms the original password.
6. The users clicks on the “Submit” button.
7. The system sends information to the backend server.
8. The server-side application validates the inputs. <E1>
9. The server-side application stores the users details in the database.
10. The server-side application responds with the registered users details (minus the password).
11. The system allows the user to log in to the website.

**Exceptional Flow**

E1: Inputs are invalid

1. The user inputs which are invalid are displayed in red, with a message indicating the reason for the error under each invalid input.
2. The user inputs correct and valid inputs in the form.
3. Return to step 7 of Main Flow.

**Termination**

The application reports a successful registration.

**Post Condition**

The system returns to a wait state.

#### Requirement 2: User Login

#### Description & Priority

In order for a user to access their account and use it’s full functionality, its necessary to that user to login to their account using details used during registration.

#### Use Case: Sign In

**Scope**

The scope of this use case is to allow a user to enter into their personal account on the application.

**Description**

This use case describes the steps to be taken in order to successfully log in to their personal account on the application.

**Flow Description**

#### **Precondition**

The system is idle

**Activation**

This use case starts when a user clicks on the “Login” link.

**Main Flow**

1. The user navigates to the “Login” page on the web application.
2. The user enters their username of between 3 and 30 characters.
3. The user enters their specific email address.
4. The user enters their specific password of between 6 and 30 characters.
5. The users clicks on the “Submit” button.
6. The system sends information to the backend server.
7. The server-side application validates the inputs. <E1>
8. The server-side application responds with the registered users details (minus the password).
9. The system logs the user in to the web application.
10. The system displays the users profile page.

**Exceptional Flow**

E1: Inputs are invalid

1. The user inputs which are invalid are displayed in red, with a message indicating the reason for the error under each invalid input.
2. The user inputs correct and valid inputs in the form.
3. Return to step 7 of Main Flow.

**Termination**

The application reports a successful login and shows the user profile page.

**Post Condition**

The system returns to a wait state.

#### Requirement 3: Create Recipe

#### Description & Priority

In order for a user to publish her/his own recipe, they must first go through the process of submitting validated information via the Create Recipe functionality of the web application.

#### Use Case: Create Recipe

**Scope**

The scope of this use case is to allow a user to submit a new recipe in the form of form data in the web application.

**Description**

This use case describes the steps to be taken in order to submit a new recipe to the web application.

**Flow Description**

**Precondition**

The system is idle

**Activation**

This use case starts when the user clicks on the “Create Recipe” link.

**Main Flow**

1. The user navigates to the “Create Recipe” page on the web application.
2. The user enters the specific recipe information into the input form.
3. The user clicks on the “Submit” button.
4. The application sends the form data to the server-side application.
5. The server-side application validates the inputs. <E1>
6. The server-side application saves the new recipe to the database.
7. The server-side application responds with the newly saved recipe data.
8. The new recipe data is added to the front end application state.
9. The user is taken to the new recipe’s URL.

**Exceptional Flow**

E1: Inputs are invalid

1. The user inputs which are invalid are displayed in red, with a message indicating the reason for the error under each invalid input.
2. The user inputs correct and valid inputs in the form.
3. Return to step 6 of Main Flow.

**Termination**

The application reports the successful saving of the new recipe and shows the user the specific recipe page.

**Post Condition**

The system returns to a wait state.

#### Requirement 4: Update Recipe

#### Description & Priority

**Scope**

The scope of this use case is to allow a user to edit a previously submitted recipe in the form of form data in the web application.

**Description**

This use case describes the steps to be taken in order to update a previously submitted recipe in the web application.

**Flow Description**

**Precondition**

The system is idle

**Activation**

This use case starts when the user clicks on the “Edit Recipe” link on a specific user created recipe’s page.

**Main Flow**

1. The user selects “Edit Recipe” on a specific recipes page.
2. The user is shown a similar form with inputs as in the “Create Recipe” page, however the inputs have been pre-populated with the current recipe’s data.
3. The user edits or changes the pre-populated input fields with new and updated data.
4. The user clicks on the “Submit” button.
5. The application sends the updated form data to the server-side application.
6. The server-side application validates the inputs. <E1>
7. The server-side application updates the recipe in database.
8. The server-side application responds with the newly updated recipe data.
9. The updated recipe data is added to the front end application state.
10. The user is taken to the updated recipe’s URL.

**Exceptional Flow**

E1: Inputs are invalid

1. The user inputs which are invalid are displayed in red, with a message indicating the reason for the error under each invalid input.
2. The user inputs correct and valid inputs in the form.
3. Return to step 7 of Main Flow.

**Termination**

The application reports the successful saving of the updated recipe and shows the user the specific recipe page.

**Post Condition**

The system returns to a wait state.

#### Non-Functional Requirements

The following requirements add extra interactivity and functionality to the application and are to be added to the application after the minimum viable product has been achieved. These requirements apply only to logged-in users.

* A user can “like” another user’s recipe. Clicking the like button again to un-like.
* A user can add the ingredients of the currently viewed recipe to a “shopping list”, which can then be viewed via the navigation bar.
* A user can cross out already purchased items from the shopping list.
* A user can comment on any recipe.
* A user can offer a “rating” for a recipe.
* A user can view a dashboard with information about their own recipes, their recipes ratings, comment count, like count etc.

#### Performance/Response time requirement

The applications response time will depend on the end users internet connectivity. However, regardless of connection, some information should be displayed to the user very quickly. In the event of zero connection, an offline screen should appear and suggest that the user attempt to reconnect. In the event of a slow connection, the basic structure of the application should quickly load and a “loading spinner” should inform the user that information is being requested through the network.

#### Availability requirement

The application will be hosted online via the Heroku platform and will be available to users via the URL. On mobile devices (Android, iOS and Windows Mobile), users will be offered the choice to add the application to their phones home screen for easy access. This PWA approach will offer a near native experience to mobile users.

#### Reliability requirement

The application will function reliably with or without an internet connection, and with or without strong connectivity. While simply being available via the website URL, the application can also be made available on the Android Play Store, the iOS App Store and the Windows App Store.

#### Maintainability requirement

The application will be developed using a modular design, allowing for easy code maintainability and upgradability. The front-end application uses React, which is a component based UI library based on reusable class based components which can be easily reused and modified. The server-side application will modularized and organized into specific folders and associated files for easy readability and updatability.

### 6.3.4. GUI

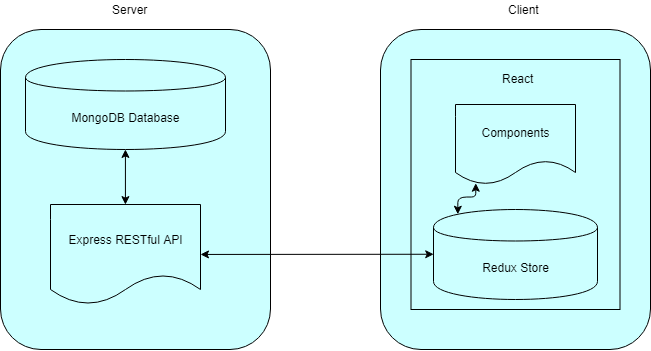
The application will implement the GUI using the Twitter Bootstrap CSS library. The library handles mobile responsiveness and some common functionality like alerts and information boxes, dropdown menus, invalid input warning messages etc. Through appropriate use of this library we can guarantee that our application will work comfortably on any sized screen, on any device with a relatively modern browser.

The wireframes for this project were produced directly in Twitter Bootstrap. The main benefit of producing the initial mock-up in Bootstrap, is that snippets of this code will be taken directly from the HTML of these files and placed in the React component files. Therefore by producing the mock-up in Bootstrap, the developer has already completed part of the work of producing the React client-side application.

Note that the Add Recipe Page at fig 4.7 will be essentially identical to the Edit Recipe page, and is therefore not shown here.

See sections ***6.4.5. User Interface Design*** and ***2.5. Graphical User Interface (GUI) Layout*** for images of the application and mock-up.

### 6.3.5 System Architecture



### 6.3.6. System Evolution

Because of the modular nature of the structure of the application, adding new or editing or expanding current functionality would be relatively easy to do.

Thanks to the full-stack approach of constructing the application with a separate server-side and client-side application, native Android or iOS applications could be built and use the same server-side application as it stands now.

If the user count rose to a significant number, a “premium” account could be developed whereby a premium member would have access to extra features that were unavailable to regular users. Examples of this could be having more than one associated Recipe Book, an information dashboard which could graphically display data about the users recipes and comments, and perhaps simply to remove advertisements that could be implemented on the regular web application.

### 6.3.7. References

Kapoor, S., 2018. *freecodecamp.org.* [Online]   
Available at: https://medium.freecodecamp.org/progressive-web-apps-101-the-what-why-and-how-4aa5e9065ac2  
[Accessed 10 October 2018].

Sherman, P., 2018. *Medium.* [Online]   
Available at: https://medium.com/@pshrmn/demystifying-single-page-applications-3068d0555d46  
[Accessed 10 October 2018].

## 6.4. Product Design Specification

### 6.4.1. Introduction

#### 6.4.1.1. Purpose of The Product Design Specification Document

The Product Design Specification document describes the system architecture and overall system design of the Recipe Book web application. The document is produced during the planning phase of the application development and the purpose is to provide guidance of the construction of the system architecture to the development team, the project manager and project team.

The user interface sections of the document may be shared with the client and/or other stakeholders whose approval or input may be required on the final application UI design.

### 6.4.2. General Overview and Design Guidelines/Approach

#### 6.4.2.1. Assumptions/Constraints/Standards

For full functionality to be available to the user, it is a assumed a strong internet connection is present. However when connection fails, or in the event of a weak connection, the user will still be presented with some minimal level of information, this may be simply an offline page that will inform the user of the lack of connection and prompt them to reconnect, or some cached information being displayed which was saved during the last page load. The level of functionality will be constrained by the time available during development.

The application itself will be developed specifically using the latest version of the Chrome browser on Windows 10. However the application will be tested on the Firefox browser on Windows 10, Firefox and Chrome browsers on Ubuntu Linux, Firefox and Chrome on Android 7+ and Safari browser on iPhone X.

The application will be fully mobile responsive and will be user friendly and functional at any screen size.

### 6.4.3. Architecture Design

#### 6.4.3.1. Logical View

Fig. 6.4.3.1.1

The above diagram (fig. 6.4.3.1.1) outlines a high level view of the overall system architecture.

In this representation, the client represents a user’s browser, on whichever device they are using e.g. tablet, phone, desktop etc. When the client navigates to the web application’s URL, the React application loads in their browser and displays the application to the user. The user can navigate to different “pages” in the application, which will be handles by React using the React Router. The application itself will be a single page application, but will function as a multi-page website. The React Router will handle all routing in the application. Page change renders will happen extremely quickly as the application will not have to request an entirely new document on each page change, but will simply load the specific component which is related to the current URL (e.g. when the user navigates to /login, the React application will load the Login Component).

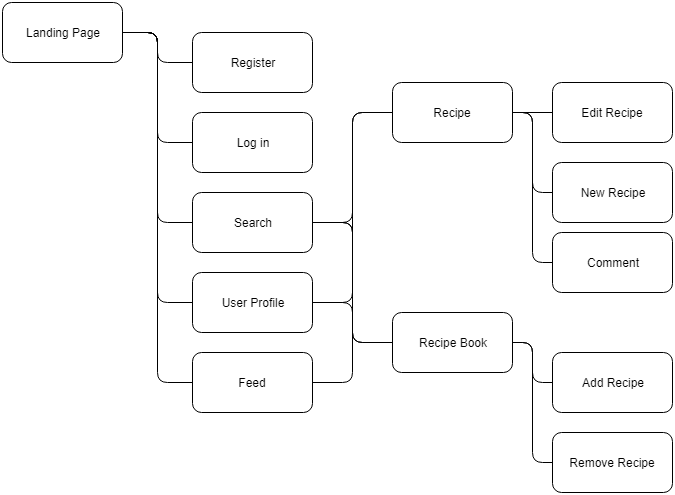


Fig. 6.4.3.1.2

The above diagram represents a simple site map. Most navigation revolves around going to and from Recipes and their respective creators, or site users. The search functionality will provide the user with a list of recipes related to their search parameters, which in turn will bring the user to that specific recipe page. From here, if they are logged in they can comment. If it is there own recipe they can edit the recipe.

Similarly from a user’s profile page, where if they are logged in they can choose to “follow” that specific user. Here a recipe can be selected and the user will be taken to that specific recipe page, where again they will be able to comment if they are logged in.

From a user’s profile page, that users specific Recipe Book can be viewed, which is a personally curated list of that users favourite recipes.

The “feed” page, which is available to logged in users, will show the most recently created recipes by the users which they follow. If no users are being followed, the most recent posted recipes will be shown.

#### 6.4.3.2. Hardware Architecture

**“Heroku is** the quickest way for a company to become an apps company. Heroku is a service that enables companies to spend their time developing and deploying apps that immediately start producing value.” (Heroku, 2018)

The entire application, when completed, will be hosted on the Heroku platform. The Heroku platform has a free tier with a very easy to use CLI (Command Line Interface) which enables quick and painless iterative deployment.

The client and server will be deployed to the same Heroku instance.

When a user navigates to the application’s URL, the client-side application will be downloaded and run on the users machine. Requests for information will be sent to the server from the client, and the server will respond with JSON data which will be used to update the users view.

The client itself simply refers to the users browser, which can be on any machine with a modern web browser. The web application can scale to any viewport size, therefore the application will work on tablets, desktops and mobile phones.

#### 6.4.3.3. Software Architecture

A visual overview of the system architecture can be seen in figure 3.1.1.

The application itself will be developed with the following technologies.

##### Server-side

|  |  |
| --- | --- |
| Node.js | A server-side version of the JavaScript language. |
| Express.js | A web application framework commonly used to build RESTful APIs. |
| MongoDB | A NoSQL database, hosted on mlab.com |
| Mongoose | An ODM (object document mapper), used for managing the interaction between the server and the database. |

NPM Packages used:

|  |  |
| --- | --- |
| Bcryptjs | A JavaScript implementation of the bcrypt library, used for hashing user password. |
| Body-parser | Used for getting information from forms submitted via post requests. |
| Concurrently | An npm package used to run the server and the client applications at the same time |
| Jsonwebtoken | Used for handling the use of JSON Web Tokens in Express |
| Passport | Handles user authentication and authorisation |
| Passport-jwt | A JSON Web Token authentication strategy for use with Passport |
| Validator | Used to handle different types of form/input validation |

##### Client-side

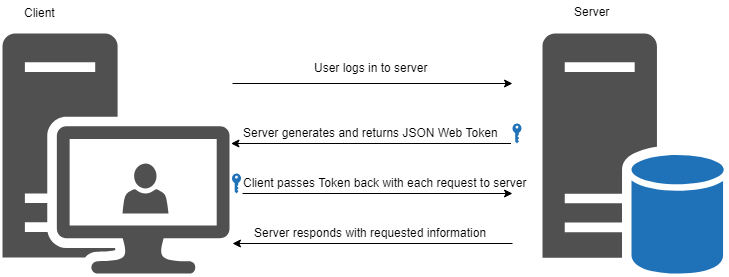
|  |  |
| --- | --- |
| React | The React JavaScript library |
| Axios | Used for making AJAX requests to the server |
| Classnames | Conditional class names in React |
| Redux | Used to manage application-wide state |
| Moment.js | Parse dates to human readable strings |
| Jwt-decode | Decode JSON Web Tokens into usable JavaScript Objects |

#### 6.4.3.4. Security Architecture

User Authentication and Authorisation

“**Authorization**: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.” (jwt.io, 2018)

When a user logs in to the website, the server-side application will create a JSON Web Token which will act as an identifier which the client-side application will attach to each request sent to the back-end, to let the server know that the current user is logged in. This will let the application know which routes the user can access and which functionality they have access to.



#### 6.4.3.5. Communications Architecture

There are several different types of communications that will be performed in the overall application. The front-end application will communicate with the back-end application via AJAX requests. The front-end will send form data and user credentials, for logged in users extra headers will be sent in the form of an “Authorization” header containing the users JSON Web Token for authorisation.

The server sends two types of information to the client, the HTML, CSS and JavaScript files needed to load the application itself, and for further requests, JSON data which will be in response to any requests the client asks of the server.

Furthermore, more communication take place inside the front and the back-end applications.

In the front-end, information will be communicated in the form of “props” and “state”. Props are pieces of information passed from one component to another in React. State will be handled by Redux, and will receive information from the React app in the form of actions and reducers, which are used to update the current application state. The state can then be accessed and used by any component application wide, with the information from the state being accessed via props.

In the back-end, the express application will communicate with the MongoDB instance via the Mongoose Object Document Mapper.   
The application will employ database models, which are created using Mongoose Schema objects. As an example, the User model is represented as the following Mongoose Schema shown in Figure 3.5.1.

The contents of the image are contained in a single file, which is exported a the end of the file. This means that the User model can be imported to any other file in the server-side application and the full Mongoose functionality can be used to connect to the User document in the MongoDB database.



Fig. 3.5.1.

#### 6.4.3.6. Performance

Node.js is good at multitasking. It is single-threaded, non-blocking, and asynchronous. Therefore it can process multiple tasks concurrently in one thread, instead of queueing them. (Ciszewski, 2018)

On the client-side, the performance of the React application will be extremely fast. As a user navigates to a new web page, the page is simply loaded instantly with React using JavaScript, without having to make new requests to the server for HTML and any associated files inside the HTML document. The client may simply make a request to the server for some JSON data to update the already loaded UI.

### 6.4.4. System Design

#### 6.4.4.1. Use Cases

Use cases for the Recipe Book application can be found in the Requirements Specification documentation in section 3.

#### 6.4.4.2. Database Design

The application employs a MongoDB instance, which is hosted on mlab.com, as the database used to persist out data. MongoDB is a NoSQL database, which differs from a SQL database in that it stores data in “documents”, which resemble JSON data in how they are structured. This is an excellent choice for a full-stack JavaScript application as the persistent data will be stored in a format that is almost identical to that consumed by the application.

The application will use Mongoose, a JavaScript based ODM (Object Document Mapper), which is used to create database schemas in order to easily query and manipulate data in the applications database. An example of the mongoose schema can be seen in figure 3.5.1.



Fig. 4.2.1.

An example usage of Mongoose being employed to register a new user in the application is shown above in figure 4.2.1.

The “User” model is imported at the top of the file, and is used at the beginning of the figure. A simplified version of the sequence of events represented here in JavaScript code is as follows:

Check in the “User” document for a user that has the email address which has been submitted via the form.

If such a user exists already, respond with an error informing the user.

If such a user does not exist, construct a new instantiation of the User model, using the data received from the front end via the input form.

Bcrypt then goes on the generate an encrypted version of the password received. Once the password has been encrypted, the user is then persisted in the database.

A similar sequence of events will take place, minus the encryption, for the recipes and comments stored in the application database. Once a user, recipe or comment etc is found in the database, the developer can make any changes he/she choose to make to the user object, and then save that object back to the database.

#### 6.4.4.3. Data Conversions

Data transferred between parts of the application will take the form of JavaScript Object Notation. This is a data display format, styled almost identical to how an object would appear in JavaScript. Data requested from the database will be sent as JSON to the client side, where it will be parsed into a usable JavaScript object, or array of objects.

Similarly, when form input data is sent to the server from the client side, it is converted from raw form data into JSON and sent to the server, which parses the information into usable object to be manipulated and eventually stored in the database.

Below is a simple example of JSON information.

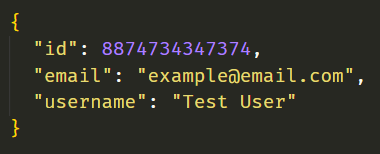


Fig. 4.3.1.

#### 6.4.4.4. Application Program Interfaces

An API is a type of interface for communicating with an application or service from the outside. An example would be a weather API. An outside application or user would send a HTTP request to the API, perhaps requesting local weather data, and the weather API would receive the request, gather the needed data and respond with the requested data. Usually in JSON or XML format.

The Recipe Book application will use a database hosted at mLab to persist data, and therefore will communicate with mLabs API via the URL to the application’s database, including a database user credentials for authorisation.

## 

### 6.4.5. User Interface Design

The user interface of the application will be developed with two main technologies.

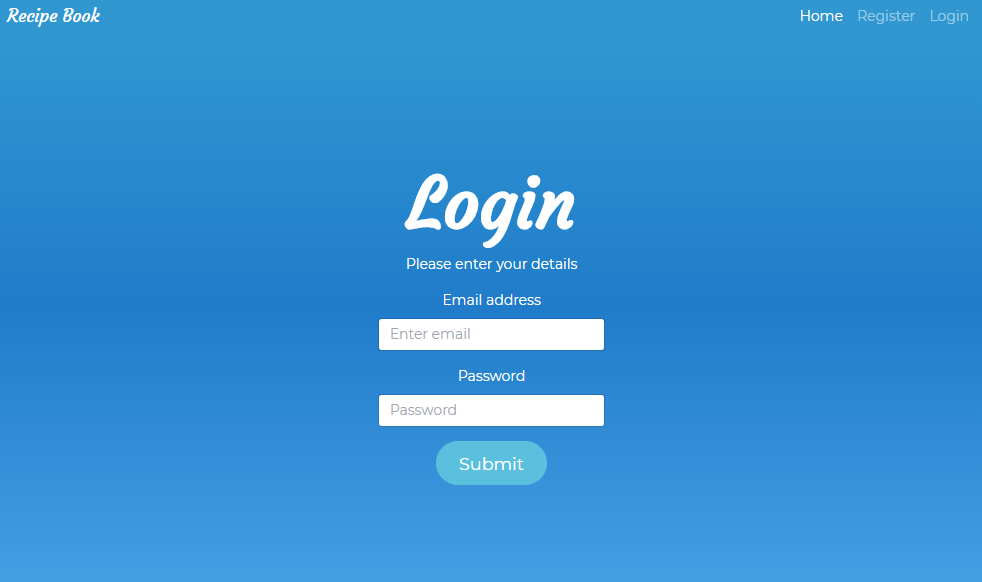
The appearance of the application will be built using Twitter Bootstrap v4. Bootstrap is a very popular CSS grid-based framework for very quickly developing websites, with much of the common use cases already created. Bootstrap also saves a large amount of development time by handling the mobile responsiveness, when implemented properly. The functionality of the user interface will be handles using React. React is a JavaScript library on the client side, designed for handling interactive functionality in a web application.

Some examples of the user interface as it currently stands in this stage of development are as follows:

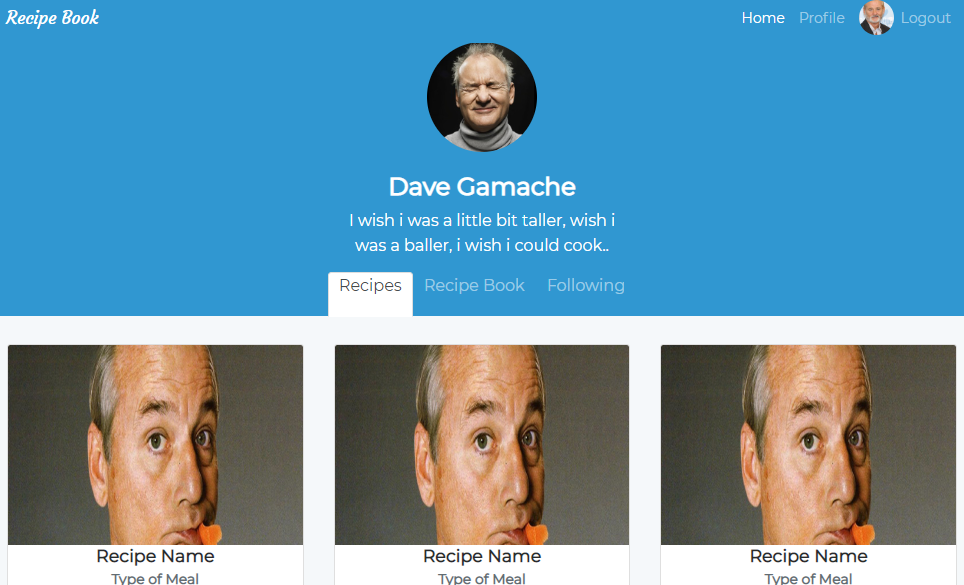
Landing Page



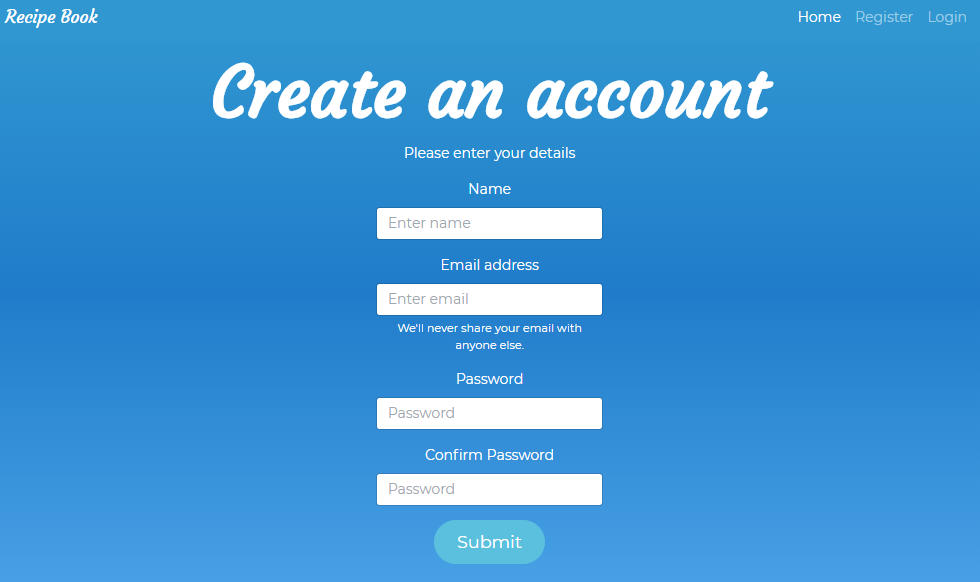
Login Page



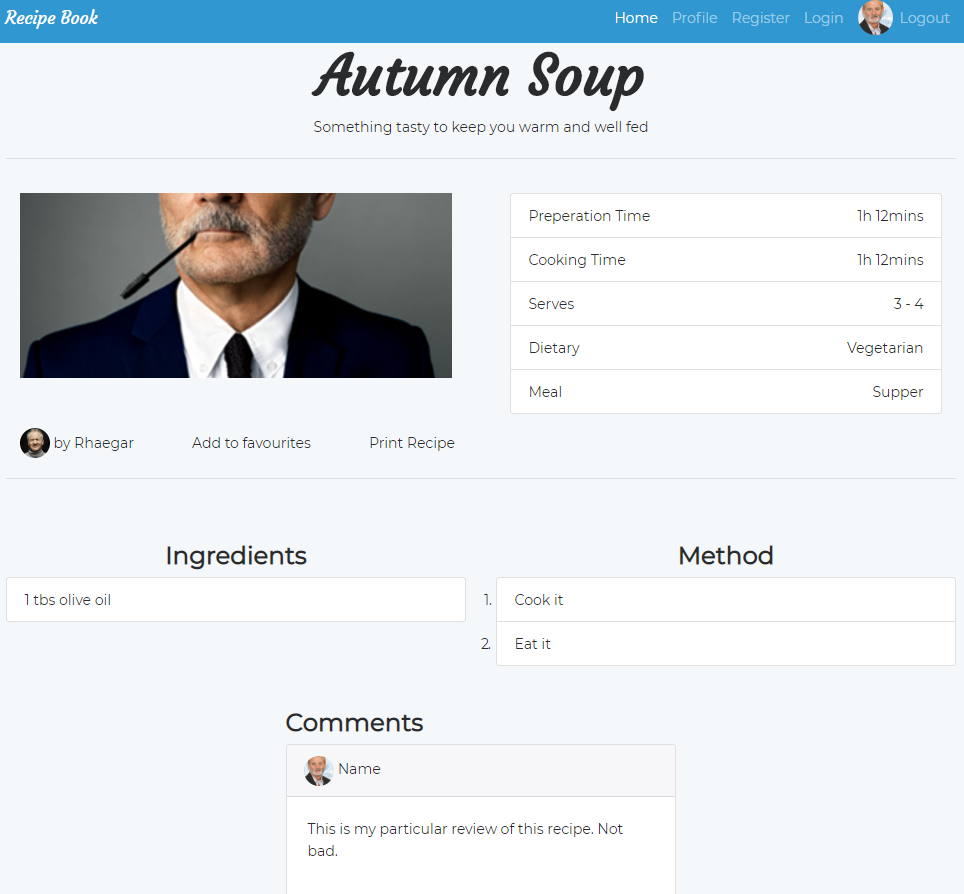
Profile Page



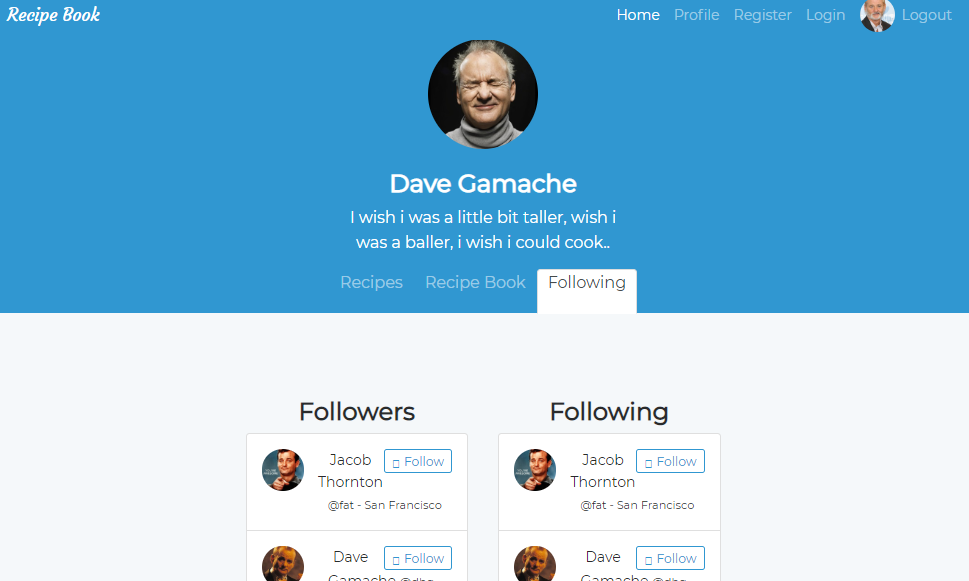
Register Page



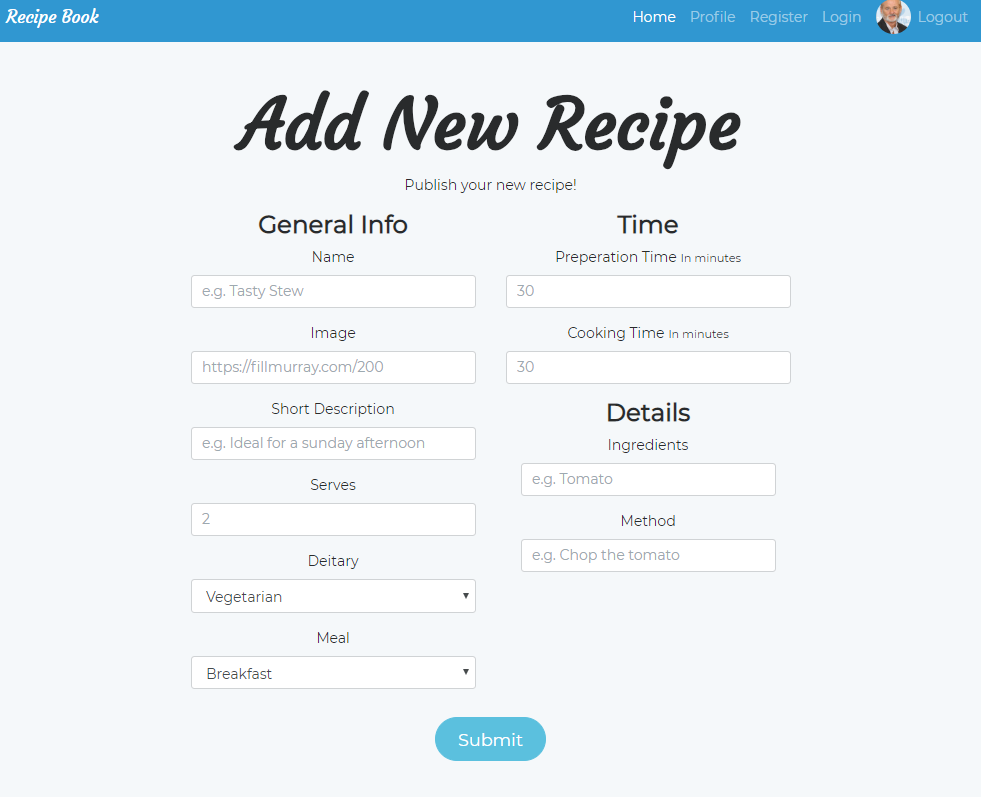
Recipe Page



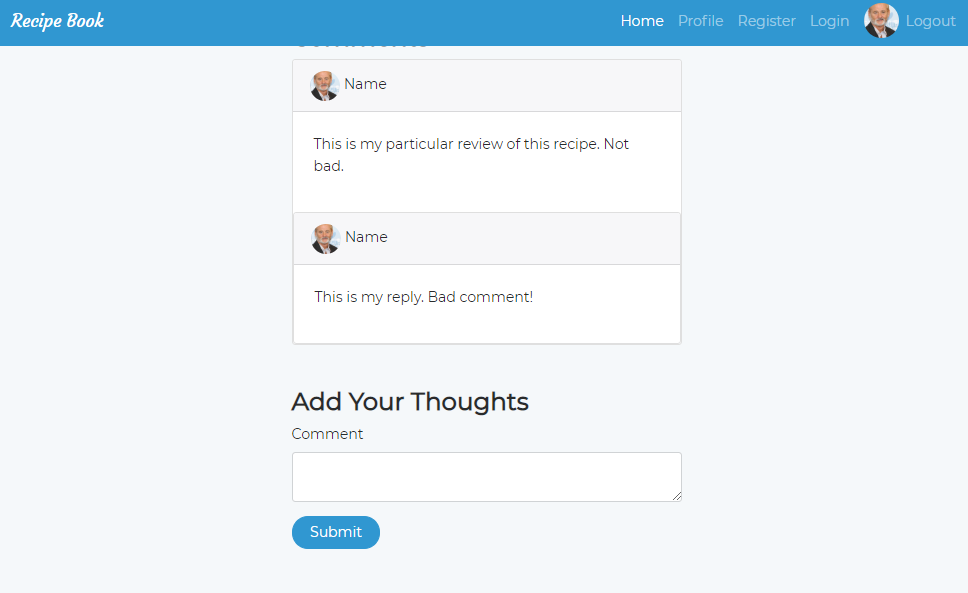
Following Information Page



Create Recipe Page



Comment Detail on Recipe Page



#### 6.4.4.6. Performance

Once a user visits the web application for the first time, the HTML JavaScript and CSS files will be download and cached on the users device, making future page loads very fast. The JavaScript and CSS files will be minified (essentially made smaller by removing white space in the text files and renaming variables to single characters) which will provide a minor speed boost to the application.

In low connection circumstances, the applications basic structure will load, and a loading spinner will notify the user of the state of the application during data transfers. In a zero connection environment, an offline page will appear, notifying the user of the lack of connection and prompting them to refresh and attempt to reconnect. Therefore, the user will always have some visual cue as to the current state of the application.

#### Acronyms, Abbreviations and Terms

|  |  |
| --- | --- |
| HTML | Hyper-text Mark-up Language, the language that provides structure to web pages. |
| CSS | Cascading Style Sheets, adds styling to web pages. |
| UI/UX | User Interface, User Experience |
| CLI | Command Line Interface, text based interface using the terminal or command-line |
| JSON | JavaScript Object Notation, a data inter-change format which is easy for humans to read and write. |
| JWT | JSON Web Token, access tokens used for authorisation and authentication |
| HTTP | Hyper-text Transfer Protocol, underlying protocol of the web |
|  |  |

### 6.4.6.References

Ciszewski, B., 2018. *Nodejs Performance.* [Online]   
Available at: https://www.netguru.co/blog/nodejs-performance-web-application-benefit  
[Accessed 26 October 2018].

Heroku, 2018. *What is Heroku.* [Online]   
Available at: https://www.heroku.com/what  
[Accessed 21 October 2018].

jwt.io, 2018. *JSON Web Token Introduction.* [Online]   
Available at: https://jwt.io/introduction/  
[Accessed October 21 2018].