10/21/2018

Gavin Hanna

Higher diploma in science in web technologies

Product Design Specification

Recipe Book

# Version History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Version # | Implemented By | Revision Date | Approved By | Approval Date | Reason |
| 1.0 | Gavin Hanna | 21/10/2018 |  |  | Initial Draft |

Table of Contents

[Version History 1](#_Toc528342615)

[1. Introduction 3](#_Toc528342616)

[1.1. Purpose of The Product Design Specification Document 3](#_Toc528342617)

[2. General Overview and Design Guidelines/Approach 3](#_Toc528342618)

[2.1. Assumptions/Constraints/Standards 3](#_Toc528342619)

[3. Architecture Design 4](#_Toc528342620)

[3.1. Logical View 4](#_Toc528342621)

[3.2. Hardware Architecture 6](#_Toc528342622)

[3.3. Software Architecture 6](#_Toc528342623)

[3.3.1. Server-side 6](#_Toc528342624)

[3.3.2. Client-side 7](#_Toc528342625)

[3.4. Security Architecture 7](#_Toc528342626)

[3.5. Communications Architecture 8](#_Toc528342627)

[3.6. Performance 10](#_Toc528342628)

[4. System Design 10](#_Toc528342629)

[4.1. Use Cases 10](#_Toc528342630)

[4.2. Database Design 10](#_Toc528342631)

[4.3. Data Conversions 12](#_Toc528342632)

[4.4. Application Program Interfaces 12](#_Toc528342633)

[4.5. User Interface Design 13](#_Toc528342634)

[4.6. Performance 18](#_Toc528342635)

[Acronyms, Abbreviations and Terms 19](#_Toc528342636)

[References 20](#_Toc528342637)

# 1. Introduction

## 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.

# 2. General Overview and Design Guidelines/Approach

## 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.

# 3. Architecture Design

## 3.1. Logical View

Fig. 3.1.1

The above diagram (fig. 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).

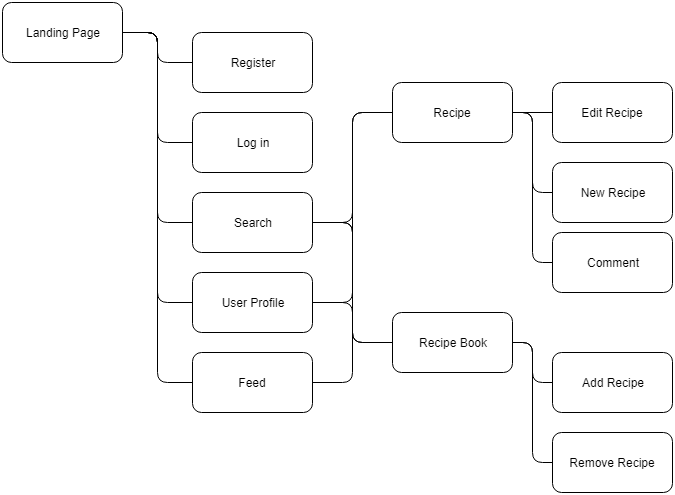


Fig. 3.1.2

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.

# 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.

# 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.

## 3.3.1. 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 |

## 3.3.2. Client-side

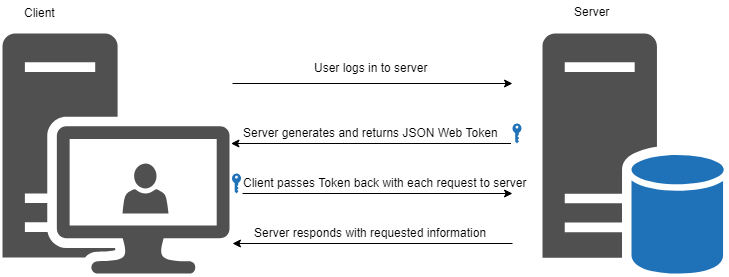
|  |  |
| --- | --- |
| React | The React JavaScript library |
| Axios | Used for making AJAX requests to the server |
| Classnames | Conditional class names in React |
| Redux | Used to manage application-wide state |
| Moment.js | Parse dates to human readable strings |
| Jwt-decode | Decode JSON Web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

# 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.



# 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.

## 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.

# 4. System Design

## 4.1. Use Cases

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

## 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.

## 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.

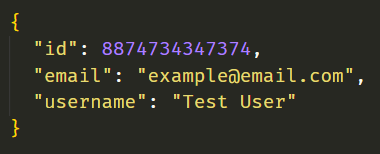


Fig. 4.3.1.

## 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.

## 4.5. User Interface Design

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

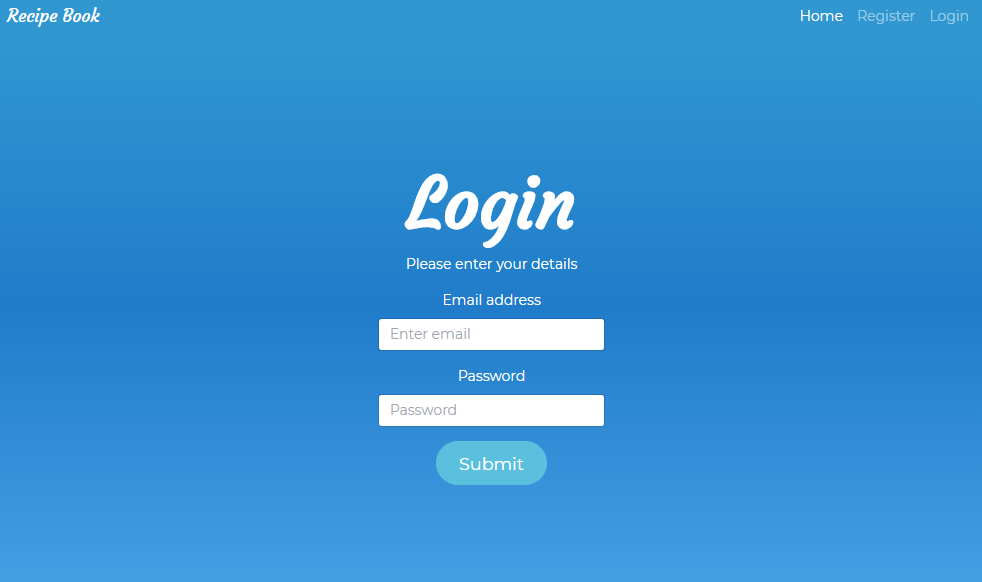
The appearance of the application will be built using Twitter Bootstrap v4. Bootstrap is a very popular CSS grid-based framework for very quickly developing websites, with much of the common use cases already created. Bootstrap also saves a large amount of development time by handling the mobile responsiveness, 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:

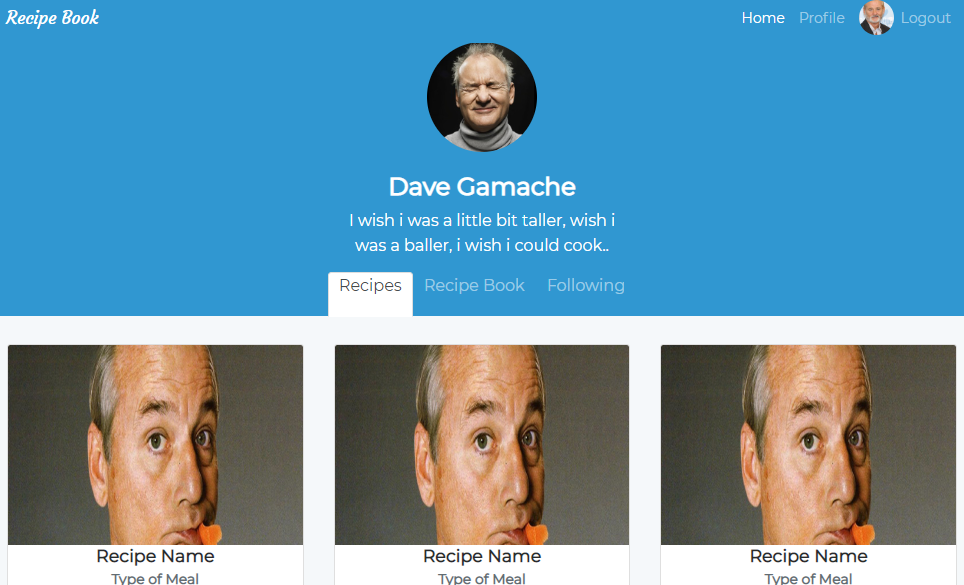
Landing Page



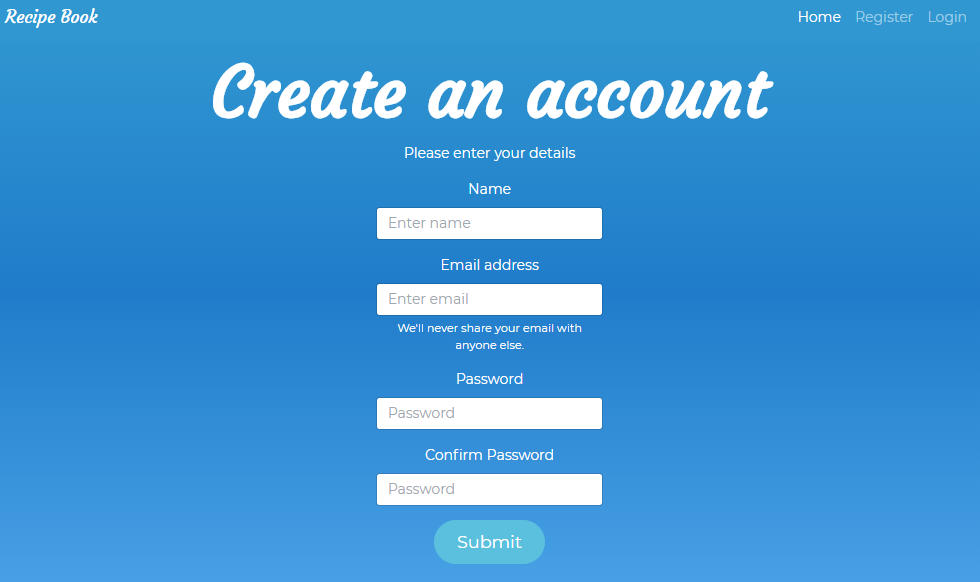
Login Page



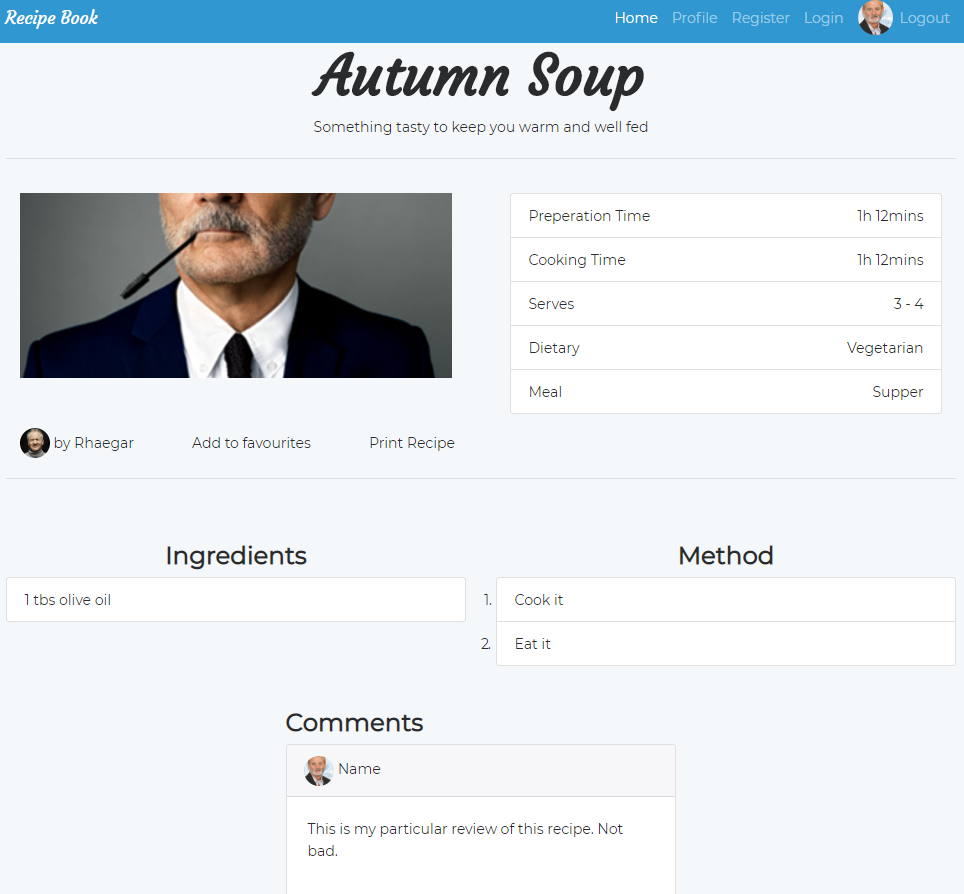
Profile Page



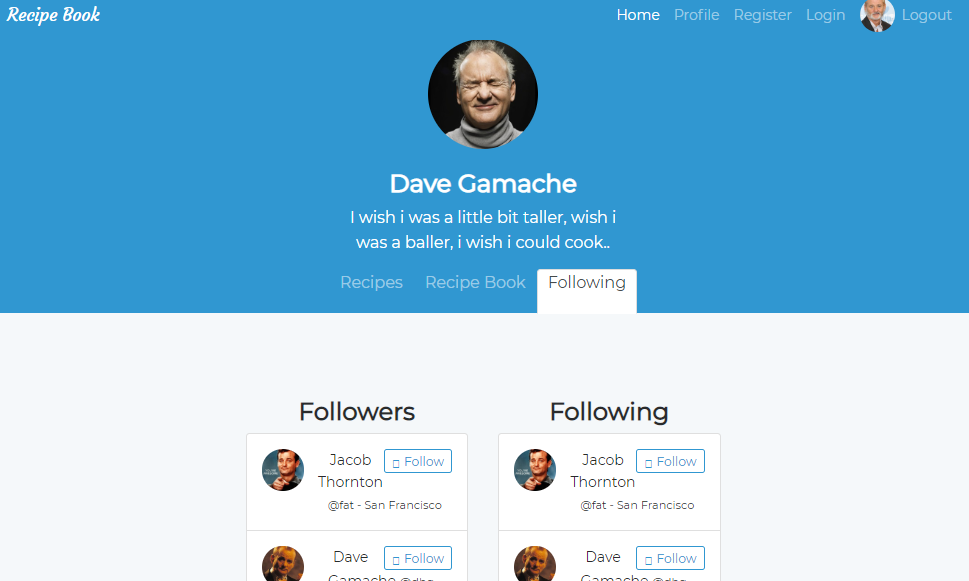
Register Page



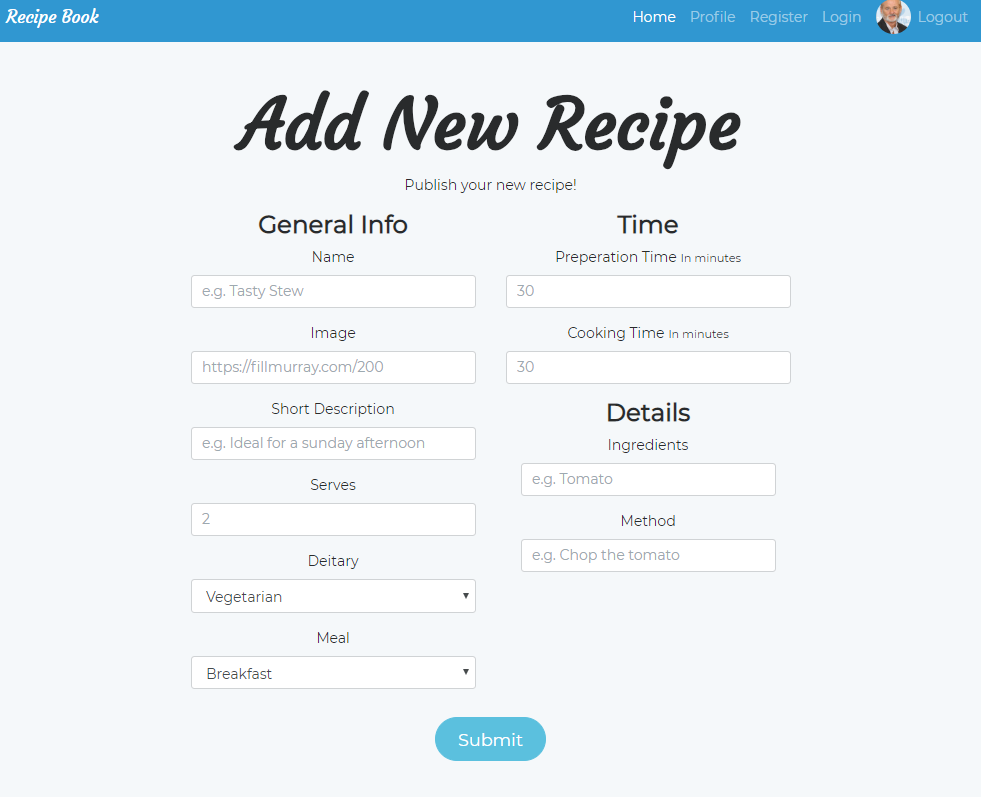
Recipe Page



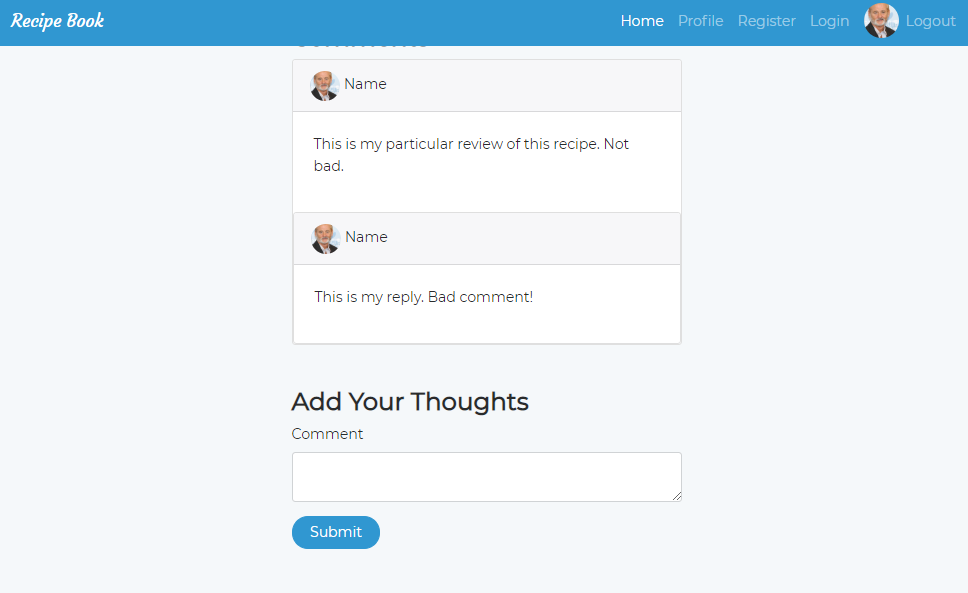
Following Information Page



Create Recipe Page



Comment Detail on Recipe Page



## 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 |
|  |  |

# 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].