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](#_Toc527910854)

[1. Introduction 3](#_Toc527910855)

[1.1. Purpose of The Product Design Specification Document 3](#_Toc527910856)

[2. General Overview and Design Guidelines/Approach 3](#_Toc527910857)

[2.1. Assumptions/Constraints/Standards 3](#_Toc527910858)

[3. Architecture Design 4](#_Toc527910859)

[3.1. Logical View 4](#_Toc527910860)

[3.2. Hardware Architecture 6](#_Toc527910861)

[3.3. Software Architecture 6](#_Toc527910862)

[3.3.1. Server-side 6](#_Toc527910863)

[3.3.2. Client-side 7](#_Toc527910864)

[3.4. Security Architecture 7](#_Toc527910865)

[4. System Design 8](#_Toc527910866)

[4.1. Use Cases 8](#_Toc527910867)

[References 9](#_Toc527910868)

# 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).

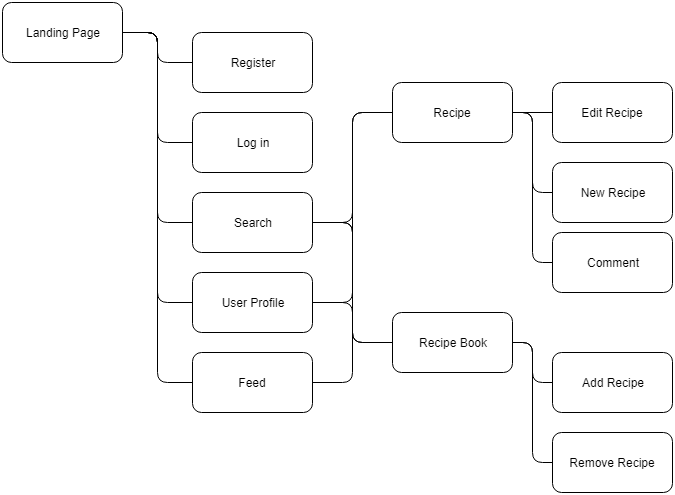


Fig. 3.1.2

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 ORM (object relational 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

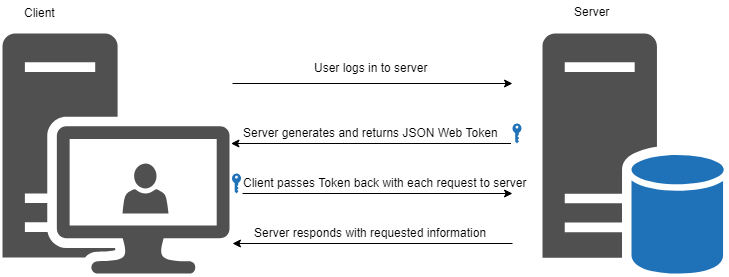
|  |  |
| --- | --- |
| React | The React JavaScript library |
| Axios | Used for making AJAX requests to the server |
| Classnames | Conditional class names in React |
| Redux | Used to manage application-wide state |
| Moment.js | Parse dates to human readable strings |
| Jwt-decode | Decode JSON Web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

# 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 & Data Conversions

# References

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].