1. Introduction

1.1 Scope of the Project

The goal of this software is to provide the user with a way to easily search for new recipes based on their current ingredients. The user is able to add ingredients to their pantry and then conduct a search which produces recipes containing those selected ingredients. The software provides the user with community feedback on the recipes in the form of ‘Likes’ and ‘Dislikes’ in order to establish a social aspect. The application provides users with a profile feature which allows them to save their ingredient list and also save their favorite recipes for return usage efficiency.

2. User Interface

2.1 Site Map

Describe how do the pages in the site link to each other.

- Main (search)
    - Header
        - HeaderLogo
        - HeaderMenu
        - Login+Signup/Logout
    - SearchBar
    - TopRecipes
    - ContactInformation
    - Reviews
- Recipes
    - Header
    - IngredientMenu
    - ResultList
- Profile
    - IngredientList
    - FavouriteRecipeList
- Recipe
- Admin
- Submit
- Login
- Signup

2.2 Page Layout and Design

Describe the layout(s) of the page design(s) used in your site.

Home Page:

The main page has a navigation bar on the top of the screen which has several buttons which allow the user to smoothly scroll through the page. The navigation bar also contains the buttons for the Login and Sign-up forms. The Login and Sign-up forms are pop-up modals that can be filled out and submitted by the user.

The page is broken into sections which fill the user's browser window. The first section contains the search bar which allows the user to add ingredients and conduct a search for recipes. The second section contains the top featured recipes on the website. The third section contains user reviews and the last section contains contact information and two forms. The first form is used to send a message to the admins and the second form is to submit a recipe to be added to the database.

Search Results Page:

After the user searches on the home page they are routed to the results page which has a top navigation bar, a fixed side navigation bar, and the recipes in the form of cards in the main section. The side navigation bar allows the user to add new ingredients to the search and to reroute back to the homepage or to their user page.

User Page:

The user page has a top and side navigation bar. The side navigation bar allows the user to add new ingredients to their "pantry", reroute to the home page, or conduct a new search which reroutes them to the results page. The main section shows all of the user's "favorite" recipes.

Administration Page:

The administration page has a side navigation bar that allows the admin to move between sections. The admin is able to view the list of users, add new recipes, add new ingredients, add categories, check the recipe request from the user. Whichever category the admin selects will be loaded in the main section of the page.

3. Database

The App will contain data that is stored MongoDB database.

Properties of the recipe include (data type):

- id (unique string), required
- Title (string)
- Photo of the recipe, optional (string, url, image stored in CDN)
- List of ingredients, required (array of strings, no numbers or special characters)
    - id (unique string), required
    - quantity (number)
    - name (string)
- Cooking time in minutes, required (number)
- Number of meals, required (number)
- Level of the recipe, required (selected from list)
- Instructions, required (string with markdown)
- number of likes (number)
- number of dislikes (number)

Properties of the ingredient include (data type):

- id (unique string)
- name (string), required
- categories (string)

Properties of the user include (data type):

- id (unique string), required
- email (string), required
- password (string), required (6 chars min length)
- favorite_recipes, optional (array of recipes id)
- ingredients, optional (array of string)
- userType (string), required

4. Client Side SW Design

Technologies

Materialize CSS (CSS framework)

The app will be responsive, work great on every device. The UI for mobile and desktop must be a little bit different in term of element management

React JS (Front-end Framework)

Manage two way data binding, state change, request to our server when something changes, not necessary to reload the whole page. The app will have dynamic data.

Redux (State handling for React)

The app uses Redux for handling application state, for example in following cases:

- Async operations with database

- Handling navigation (mobile)
- Handling forms

React Native (Mobile Framework)

Native version of the app will be created for Android and perhaps IOS.

Here you can list and name the classes that you are going to have. For each class you can list their attributes (data fields) and member functions (methods).

If you are not going to have any classes, you can list the (JavaScript) functions that you know that you will have. You can describe the purpose of each function with a few lines of text.

Functions could be described with the following points, but it is not absolutely necessary to describe functions so accurately in the design phase.

- General description of the function and what it is used for.
- The name of the function.
- The return type.
- Ranges of return values and their meanings.
- Parameter names, types, whether the parameter is input, output or both and under what circumstances it is read or written.
- Assumptions on the parameter values.
- Assumptions on other conditions, such as global data or system state.
- Input validations that the function performs.
- Side effects of the function.
- Exceptions the function might throw and under what conditions.
- Non-trivial algorithms used.
- Non-trivial data structures used and for what purpose.
- Other non-trivial functions that the function calls.
- If the software has a layer structure, or some other inner partitioning, then to which part or layer this function belongs (this information should be evident from the naming convention).

Describe here also which global variables, arrays, or other data items you will need.

5. Server Side SW Design

The main purpose of Back-end is for REST API. Client side (React in this case) will call REST API to the back-end service in order to retrieve data and perform other operations from the database.

Express (Back-end Framework)

Manage different routes for different pages. Authentication system (login system). Users can login through registration or Facebook. Confirming registration by sending link to email

Some example routes:

Recipe:
GET /api/recipes: Get all the recipes in the database
GET /api/recipe/:id: Get the recipe according to the id
GET /api/recipe/:id/increaseLike: Increase the number of likes for the recipe specified by its id
GET /api/recipe/:id/decreaseLike: Decrease the number of likes for the recipe specified by its id
GET /api/recipe?ingredients=a, b: Get the recipe that contains the ingredients ‘a’, or ‘b’, or both
POST /api/recipe: Upload the recipe to the database
POST /api/images/upload: Upload images from local to CDN in order to display to the webpage

Ingredient:
GET /api/ingredients: Get all the ingredients in the database
GET /api/ingredient/:id: Get the ingredient according to its id
POST /api/ingredient: Upload an ingredient to the database

Category:
GET /api/categories: Get all the categories in the database
GET /api/category/:id: Get the category specified by its id
GET /api/category/:id/ingredients: Get all the ingredients belong to the category specified by its id
POST /category: Upload a category to the database

Nginx (Web Server)

Acts as a reverse proxy which can redirect the request coming from Port 80 (HTTP) or Port 443 (HTTPS) to a local port (ex: 8080) in which the Express is running.

The advantage of using Nginx instead of only Node.js server is that when the app need to scale up, we can run more servers in local using Express and implement Load Balancer at Nginx to distribute the requests equally to our servers.