Overview

This project is a full-stack web application that allows users to store and perform CRUD (Create, Read, Update, Delete) actions on recipes with a dynamic user interface. Each recipe has the food item name, ingredients needed for it, instructions to follow as well as an image of the food item. The application is divided into client-side (Front-end) and server-side (Back-end), each within their own directory.

Features

- An interactive GUI that allows the user to look at, create, update and delete recipes which utilizes modals for user forms.
- A search bar that utilizes regex matching to allow the user to search for recipes by their name.
- A "All Recipes" button that allows the user to reset their search results and go back to viewing all their stored recipes.
- An image is associated with each recipe, which gives user an idea of the end-result of the recipe in a visual manner.
- A horizontal scrolling to view recipes within the app using two scroll buttons, which allows the user to view all elements/functionality at all times.

Project Architecture

It uses vanilla JavaScript and HTML/CSS for the client side of the application (front-end). The client side uses PouchDB to store images locally in the client side browser using IndexDB and WebSQL in the browser. The server-side uses Express, a Node.js web application framework to handle routes and provides a RESTful API interface. The application uses MongoDB Atlas for cloud data storage to store data for the recipe while the images are stored locally in the server.

Getting Started and Configuring database

First, make sure you are in the application directory in your terminal and run the following command to install as the dependencies needed for the application. This command installs all modules listed as dependencies in `package.json`.

```
npm install
```

To configure and connect to a database, set up a database in MongoDb Atlas named "recipes" and create a collection named "recipes" inside it. Then, create a `.env` file in the application directory and set `DB_URL` to your MongoDb Atlas cluster connection URI. The connection URI can be found by clicking the connect button in the cluster's "Overview" interface. (You will need a MongoDb Atlas account for this, a tutorial for this step can be found here: MongoDb Atlas tutorial). Optionally, you can also set `PORT` to your port of choice to make the application run in that port. The default port is 3000.

Now, run the following command to start running the application.

```
npm start
```

The application will now be available locally in `http://localhost:3000`. As mentioned above, you can change the port number by setting a different port in your `.env` file.

Client-Side/Front-End

The front-end of the application is implemented using HTML, CSS, and JavaScript. It consists of a single html and css file - `index.html` and `main.css` and has several JavaScript files that handle different aspects of the application.

Data Handling

Requesting the server for data and storing them locally is handled by `data.js` and `image.js`, which both provide functions that send GET requests to the server to retrieve the required data. `imagedb.js` uses PouchDb to store images locally in client side as blob objects, which is used as necessary to get source urls for images.

Views

The application mainly has two views: i. the main view that has all shows the recipe cards as well as the navigation bar and ii. modal views that provide functionality such as information about each recipe, creating a recipe and updating a recipe. The modal view is shown on top of the main view when it is active and provides a button to close the view or can be closed by simply clicked outside the modal's body.

content.js

It defines a ContentComponent class that handles the initialization and rendering of the recipe cards in the body. It imports the getAllData, getFilteredData, getRecipe functions from data.js, and the RecipeView class from recipeView.js. It maintains a state for the current recipes being viewed by the user and provides each recipe with a recipe card to render them in the application.

ModalView.js

The ModalView.js file is responsible for handling the modal views in the application. It defines a ModalView class that handles the opening and closing of modals. This is a superclass that is inherited by creating recipe modal, recipe view modal as well as the update recipe modal.

recipeView.js

The recipeView.js file is responsible for providing each recipe with a modal view to view the recipe's contents. It defines a RecipeView class that handles the modal view for each recipe. It inherits the ModalView class to handle the rendering for the modal and also provides a button to lead to the update recipe modal.

updateRecipe.js

The updateRecipeView.js file is responsible for providing each recipe with a modal view that has a form to change the recipe's contents. It defines an UpdateRecipeView class which inherits the ModalView class to handle the rendering for the modal and also handles requests to the server to update a recipe.

createRecipe.js

The createRecipe.js file is responsible for providing the user with a modal view that has a form to add a new recipe. It defines a CreateRecipeView class which inherits the ModalView class to handle the rendering for the modal and also handles requests to the server to add a new recipe.

Event Listeners

main.js

The main.js file is the entry point of the application. It adds functionality to "All Recipes" button and the search bar by adding event listeners that re-fetch and re-render the recipes as necessary. It contains event listeners for closing modals and scrolling content.

Server-side/Back-End

The back-end of the application is implemented using Express to create and handle the routes for our server. It consists of several JavaScript files that handle different aspects of the application.

Data Structure

Each recipe is stored as a object with the following properties:

- name,
- ingredients,
- instructions,
- imageUrl

The `recipe.js` file defines a recipe class with the above attributes and also a builder class to build a recipe. The imageUrl stores the unique name with which the image file for that recipe is stored in the server.

Routing

Index.js

The index.js file is the entry point of the server. It defines a RecipeServer class that handles the initialization of the database, controller, and routes, and the starting of the server. Saving the image file is handled by `multer`, which provides each image with a unique file name and saves it in the `uploads` directory.

RecipeController.js

The recipeController.js file is responsible for handling the functionality behind the routes (create, read, update, image, delete) for the recipes. It defines a RecipeController class that handles the saving, retrieval, modification, and deletion of recipes by calling the database functions but also makes sure that duplicate or old files saved by multer are deleted when a recipe is deleted.

Database functionality

RecipeDB.js

The recipeDB.js file is responsible for storing and retrieving recipes from the database. It defines a RecipeDB class that connects to a mongoDb Atlas cluster, and stores recipes in the `recipes` collection of the `recipes` database in that cluster.

Integration

The front-end and back-end communicate using the fetch API in JavaScript. Asynchronous requests are sent from the front-end to the back-end to perform CRUD operations. Responses are handled and the UI is updated accordingly. All requests are error handled using try catch and the user is alerted if there is an error in handling any action using an alert.