OP\_RECIPES QUICK PLANNING

**Models and Queries:**

User : holds user data, links to login

* Get findById() – takes in an id, and returns the requested user data
* Get findByName() – takes in a name, and returns the requested user data
* Get findAll() – returns all users in the db; may make versions to accept parameters
* Post addUser() – takes data for a new user, and adds a new user to the table
* Login – uses login table and Security to validate user and provide auth token/session info
* Logout – removes any provided auth tokens/ session info

Login : user validation info; has a fk for the user it is linked to

* Get findUsernameById() – gets the username for the requesting user
* Get findPasswordByUsername() – gets the username’s associated password
* Get findPasswordById() – gets the password associated with the user-id
* Post addUsername() – allows a user to create or change a username
* Post addPassword() – allows a user to create or change a password

Recipe : holds data for individual recipes; has fk’s for ingredients, steps, and user

* Get findById() – takes in an id, and returns the requested recipe data
* Get findByName() – takes in a name, and returns the requested recipe data
* Get findAllByUser() – takes a userId and returns all associated recipes
* Get findAll() – returns all recipes in the db; may make versions to accept parameters
* Post addRecipe() – takes in recipe data and makes a new table entry

Ingredient : holds basic data on ingredients; primarily has name and category, has fk to unit

* Get findById() – takes in an id, and returns the requested ingredient data
* Get findByName() – takes in a name, and returns the requested ingredient data
* Get findAll() – returns all ingredients in the db; may make versions to accept parameters
* Post addIngredient() – takes in ingredient data and makes a new table entry

Unit : holds data on units of measurement; this data will be instantiated at startup and remain static

* Get findAll() – returns all units in the db

Recipe\_Ingredient : join table for recipes and their ingredients; has additional field for quantity

**FE Views and Functions**

Home :

* Login/Sign Up
* Search recipes
* If logged in, update nav bar
* If logged in, view search page (not implemented)

Recipe\_Search :

* Enter search criteria and receive a display with recipes matching the parameters
* Recipes organized in a grid according to sorting choice
* Could include searchBy:
  + Name
  + Total Time
  + Ingredients
  + Nationality?

Recipe\_Page :

* Created By: username
* Name
* Cook time (not implemented)
* Prep time (not implemented)
* Total time (cook + prep times, not implemented)
* Description (not implemented)
* Photo (not implemented)
  + When implemented, photos will either be stored in an S3 bucket, and referenced in the db, or they will be stored locally, with the path being referenced in the db.
* Ingredients
* Steps

User\_Page :

* Update username/password (not implemented)
* Update name/other data (not implemented)

AddorModify\_Recipe :

* View user’s own recipes
* Modify existing recipe (not implemented)
* Create new recipe
* Delete recipe

Technical Specification

**Frontend:**

ReactJS

* Using primarily functional components. Chosen for maintainability as this is the direction that the React devs have stated they are taking the toolset.
* ReactHooks for state management; using only useState, useEffect
* ReactRouter for navigation. Routes defined in App.js, and links used in navbar.
* JSX: used for writing HTML with embedded JS functionality and conditional rendering

Tailwind

* Extensive use of utility classes provided by tailwind to apply styling to components.
* (Not yet implemented) Themes defined in config file to allow users to choose a theme to apply to the site.

**Backend**

Java with Spring Boot

* Spring Data JPA for db management using the JPA model
* Spring Security with JWT for user authentication
* JUNIT 5 for testing
* Frontend Maven Plugin & Maven Antrun for serving ReactJS frontend from backend

Database

* MariaDb as dictated by project requirements
* HeidiSQL for dB viewing and writing test queries

ToDo’s

* Convert http responses to JSON using ResponseEntity as needed
* ~~Learn more about the use of ResponseEntity~~
* Add error handling to each request.
* ~~Solve currIndex state not updating – last issue to solve to build a full recipe object~~
* Implement getRecipeByUser request
* Display user’s recipes on their recipe page
* Enable the search page
* Implement the masonry layout scheme on the search page
* Implement deleteRecipe request
* Implement view recipe on f/e
* Implement themes in Tailwind; specifically want to implement a dark mode.
* Address excessive re-renders. I’m not sure what exactly is causing these, so research will be needed to prevent excessive renders and API calls that execute on render.
* Write unit tests for each method.

Challenges Encountered