MERN-server

Shelby El-rassi	Adrienne Smith
www.shelby-el-rassi.com	adriennesmith-portfolio.netlify.app/
github.com/Shelby219	github.com/aes89

Deployed App:

Documentation Repository: https://github.com/CA-MERN/MERN-Part-A-Docs

Client Repository: https://github.com/CA-MERN/MERN-client

Server Repository: https://github.com/CA-MERN/MERN-server

Tasks

▶ Click to expand

Date Completed	Tasks Allocated	Completed?	Allocated to?
16/12/2020	Research Spoonacular API and test.	✓	Shelby
07/12/2020	User/Auth/Settings/Pref Back-end Code/Testing.	✓	Shelby
08/12/2020	Ingredient/Fridge/Pantry Back-end Code/Testing.	✓	Shelby
24/12/2020	Browse Recipe Back-end Code/Testing.	✓	Shelby
24/12/2020	Single Recipe Back-end Code/Testing.	✓	Shelby
17/12/2020	Saved Recipe Back-end Code/Testing.	✓	Shelby
24/12/2020	Connecting of Back and Front End.	×	Shelby
24/12/2020	Browse Recipe Components React Front-end .	×	Shelby
24/12/2020	Saved Recipe Components React Front-end .	×	Shelby
24/12/2020	Single Recipe Components React Front-end .	×	Shelby
24/12/2020	Single Recipe Components React Front-end .	×	Shelby

Manual Testing Log - Development

▶ Click to expand

Date	Feature	Test	Notes
16/12/2020	https://api.spoonacular.com/recipes/findByIngredients?ingredients=chicken,+cheese&number=25&apiKey={API KEY HERE}	Correct	Spooancular FIND BY INGREDIENTS API Endpoint test via Postman
16/12/2020	https://api.spoonacular.com/recipes/complexSearch?includeIngredients=lemon,strawberries&fillIngredients=true&intolerances=gluten&number=25&apiKey={API KEY HERE}	Correct	Spooancular COMPLEX SEARCH API Endpoint test via Postman
16/12/2020	https://api.spoonacular.com/recipes/716429/information?includeNutrition=false&apiKey={API KEY HERE}	Correct	Spooancular API Endpoint test via Postman

Date	Feature	Test	Notes
16/12/2020	https://api.spoonacular.com/recipes/informationBulk?ids=715538,716429&apiKey={API KEY HERE}	Correct	Spooancular BULK RECIPE SEARCH VIA ID API Endpoint test via Postman
16/12/2020	https://api.spoonacular.com/recipes/random?number=2&apiKey={API KEY HERE}	test	Spooancular RANDOM RECIPE SEARCH API Endpoint test via Postman
16/12/2020	FUNCTION - ingredientJoiner()	Console Test	First Tested via console with dummy data, before incorporating into Mocha Unit Testing
16/12/2020	FUNCTION - preferenceSeparator()	Console Test	First Tested via console with dummy data, before incorporating into Mocha Unit Testing
16/12/2020	FUNCTION - queryEditor()	Console Test	First Tested via console with dummy data, before incorporating into Mocha Unit Testing
16/12/2020	FUNCTION - userQueryBuilder()	Console Test	First Tested via console with dummy data, before incorporating into Mocha Unit Testing
24/12/2020	FUNCTION - recipeIdGetter()	Console Test	First Tested via console with dummy data, before incorporating into Mocha Unit Testing
24/12/2020	FUNCTION - basedOnPreferenceExtractor()	Console Test	First Tested via console with dummy data, before incorporating into Mocha Unit Testing
Date	Feature	Test	Notes
Date	Feature	Test	Notes
Date	Feature	Test	Notes

Automated Testing Log - Development

▶ Click to expand

Expecting Tests

Date	Feature	Test	Notes
01/12/2020	GET Register User	Passing	
01/12/2020	POST Register User	Passing	
01/12/2020	GET Login User	Passing	
01/12/2020	POST Login User	Passing	
08/12/2020	GET Logout User	Passing	
01/12/2020	Find a User from DB	Passing	
06/12/2020	GET User Settings	Passing	Passes, but being able to test with this middleware (passport.authenticate('jwt', {session: false})) not in place
06/12/2020	PATCH Edit User Settings	Passing	
07/12/2020	GET User Preferences	Passing	Passes, but being able to test with this middleware (passport.authenticate('jwt', {session: false})) not in place
07/12/2020	PATCH Edit User Preferences	Passing	Ensure req.body.preference is updated in codebase
07/12/2020	GET Fridge Ingredients	Passing	Passes, but being able to test with this middleware (passport.authenticate('jwt', {session: false})) not in place
07/12/2020	POST New Fridge Ingredient	Passing	
08/12/2020	DELETE Fridge Ingredient	Passing	
10/12/2020	DELETE ALL Fridge Ingredients	Passing	
08/12/2020	GET Pantry Ingredients	Passing	Passes, but being able to test with this middleware (passport.authenticate('jwt', {session: false})) not in place
08/12/2020	POST New Pantry Ingredient	Passing	
08/12/2020	DELETE Pantry Ingredient	Passing	
10/12/2020	DELETE ALL Pantry Ingredients	Passing	
09/12/2020	POST Upload profile picture to s3	Passing	
16/12/2020	GET - Browse Recipes- via Recipe Utils returnRecipesToBrowse(req)	Passes, but being able to test with this middleware (passport.authenticate('jwt', {session: false})) not in place	This function tests finding a User in Db per params, builds the query info per the data from user, uses that data to axios request Spoonacular API for recipes based off ingredients, then collect those recipes IDs, sanitize the data, then use the IDS for another API call to get the detailed recipe information.
20/12/2020	Recipe Controller displayRecipes(req)	Passing	
21/12/2020	GET All Saved Recipes	Passing	Passes, but being able to test with this middleware (passport.authenticate('jwt', {session: false})) not in place && Line 27 of recipe utils allowed me to test it using test user
21/12/2020	GET Single Saved Recipes if in DB	Passing	Passes, but being able to test with this middleware (passport.authenticate('jwt', {session: false})) not in place
21/12/2020	GET Single Saved Recipes if not in DB- use Spoonacular	Passing	Passes, but being able to test with this middleware (passport.authenticate('jwt', {session: false})) not in place && Double check this- async promise
22/12/2020	POST Add new saved recipe	Passing	Line 74 of recipe controller allowed me to test using test user

Date	Feature	Test	Notes
22/12/2020	DELETE A saved recipe	Passing	

| 11/01/2020 | POST Forgot Password | ✕ | Notes| | 11/01/2020 | GET Reset Password with token | ✕ | Notes| | 11/01/2020 | PUT Reset Password in DB | ✕ | Notes| | 11/01/2020 | Feature | ✕ | Notes| | Date | Feature | ✕ | Notes|

Expect to Fail Tests

Date	Feature	Test	Notes
09/12/2020	POST Login User- Incorrect Password	Passing	
09/12/2020	POST Register User- Incorrect Email and Password Format	Passing	
09/12/2020	GET User Settings- Incorrect Params	Passing	
09/12/2020	PATCH User Settings- Incorrect Email, Password, Name Format	Passing	
22/12/2020	GET Single Saved Recipes- Recipe ID not found	Passing	
11/01/2020	POST Register Create- if Error is thrown when req.login	✕	Notes
11/01/2020	POST Register Create- if Error is thrown when creating user	✕	Notes
11/01/2020	GET User Settings- if Error is thrown 404	✕	Notes
11/01/2020	PATCH User Settings- if Error is thrown 500	✕	Notes
11/01/2020	POST Forgot Password- if Error is thrown 500	✕	Notes
11/01/2020	GET Reset Password with token- if Error is thrown 500	✕	Notes
11/01/2020	PUT Reset Password in DB- if Error is thrown 500	✕	Notes
11/01/2020	GET All Ingredients- if Error is thrown 500	✕	Notes
11/01/2020	POST New Ingredient- if Error is thrown 500	✕	Notes
11/01/2020	DELETE A Ingredient- if Error is thrown 500	✕	Notes
11/01/2020	DELETE All Ingredients- if Error is thrown 500	✕	Notes
11/01/2020	GET Browse Recipes- if Error is thrown 500	✕	Notes
11/01/2020	GET Saved Recipes- if Error is thrown 500	✕	Notes
11/01/2020	GET A Recipe- if Error is thrown 500	✕	Notes
11/01/2020	DELETE A Recipe- if Error is initially thrown	✕	Notes
11/01/2020	DELETE A Recipe- if Error is thrown 500	✕	Notes

Manual Testing Log - Production

▶ Click to expand

Date	Feature	Test
test	test	test

Automated Testing Log - Production

▶ Click to expand

Date	Feature	Test
test	test	test

Sprint Planning

▶ Click to expand

We determined that setting weekly sprints was an ideal format for our project. We created a card in Trello that organised them by date and we were able to form checklists of what we wanted to have completed at the end of each sprint for the front-end and back-end. Whilst working we have a current doing card and then a completed card which we are able to distinguish each feature/component being worked on and what is completed.

In the initial planning stages we planned our Trello for the server/client based off features which would be the names of the branches. Our first feature for server/client was the user and during the first Sprint it was decided Shelby would complete the back-end code and testing and Adrienne would complete the front-end code and testing. Each morning we begin with our own stand up in which we show what we have worked on, explained our code, listed any challenges and also any wins. Since we are working on back-end and front-end seperate, this ensures we are both know what is happening on each feature.

Initially we were going to switch front-end and back-end for each feature, but we decided for the MVP product that Shelby would stick to the back-end and Adrienne on the front-end to ensure we delivered a great MVP product on time. This plan tailored to each of our strengths. This being said, once the MVP is completed all our nice to have features that we want to implement, we will switch roles for the implementation of these features. In the planning stage we decided to pair programme when it comes for connecting the server and client, which we are planning on doing at the end of each feature branch.

Additionally when it comes time to styling we will likely do a mixture of pair programming and allocation of components to style as we both really enjoy styling.

Sprint 1- 30/11- 6/12

▶ Click to expand

USER BRANCH

Shelby:

At this start of this Sprint, Shelby set up the initial back-end server code and all the express/mongo/mongoose connections and tested it was all set up correctly. Then the first component worked on was the implementation of passport, passport-JWT and jsonwebtoken for user account and authorisation. The implementation of this involved using the express session to pass around the JWT. Alongside this was the initial user account routes , the setting up of the testing of these API end points was a steep initial learning curve. This began with researching testing frameworks in which Mocha along with super test was chosen. Shelby decided on constructing the tests with a description of each Http request eg. 'GET /ingredients/:username/fridge'. The get requests were test with expecting a 200 code back along with JSON content, the post/patch requests tested by sending dummy data through the test database and testing the response matching, and the delete requests were tested with a 204 response code. The biggest hurdles during the process were setting up the correct dummy data, the tear down data functions and deciding on the structure of the tests.

Some issues were the concern of updating the user via account settings page and then the whole data being overridden, however this issue was solved for the moment since the whole user model is being sent to the account settings page, so there for can be returned with the new data. However this solution is ok for the level the project is at now, for future scalabilty this would need to be altered.

Sprint 2- 7/12- 13/12

▶ Click to expand

Shelby:

FRIDGE/PANTRY BRANCH

During this sprint the CRUD for ingredients was implemented. Shelby managed to keep the codebase dry by not doing Fridge and Pantry CRUD, rather just implementing an Ingredient CRUD base and using conditionals checking the path name, which then determines which part of the user model gets updated.

USER BRANCH

When implementing s3 and Multer for profile image upload, some blockers were incorrect set up of IAM policy, the use of .single with multer (use .any to ensure the image would upload.)

Shelby also began implementing validation using express-validator starting with validation for the email, password and user information on registering, account settings page and login.

Started writing passing fail tests to test the end points when errors arise. This pair with using validation I was able to test the results of invalid data being input for the user model.

Current blockers are implementation of persisting cookies with mocha/supertest testing so tests can be run even with authenticated routes. eg. with the middleware of "passport.authenticate('jwt', {session: false})". Currently all tests are based with this middleware not being implemented. Code that was tried includes, using superagent, setting headers, setting a beforeAll function of logging in the user and trying to manually set the cookies. The closest to success was using a beforeAll function of logging in the user, however accessing the cookies from that Http request response was not successful. This task will be moved to next sprint.

CLIENT

Completed the intial styling for the home/nav/login/register to start the basis of styling, to enable easier implementation of the react client-side.

Sprint 3- 14/12-27/12

▶ Click to expand

RECIPE BRANCH

Began Work on this feature branch on the server client. Initial routes set up. The biggest challenge was the code required for the process of getting the user data from the DB (being ingredients and preferences), error handling, sanitising the data (functions checking if null, processing booleans into an array then finally a string), then sending the correct data to the Spoonacular API calls. During the code process of the helper functions a lot of manual testing done via the console was done with some dummy data, to ensure that the JS functions were working as intended. Additionally testing Spoonacular API via postman was done to determine with Http request URLs were the right ones to use for this application.

Through Automatic testing coupled with some manual testing the main utility function for return recipe data for the browse page is: finding a User in Db per params, builds the query info per the data from user, uses that data to axios request Spoonacular API for recipes based off ingredients, then collect those recipes IDs, sanitize the data, then use the IDS for another API call to get the detailed recipe information.

In my testing of the main function in which makes all the API calls and data validation, I had some trouble testing with getting the data. I was trying to return it as a variable, then I used await outside the main async function (even though the test function was async). What you was needed was to wrap the await call inside an async function, and then call that async function in the top-level of your script. Immediately outputting the result just returned a promise pendings, then using the given code with another await to return the promise returned undefined. The below is the serious of options:

In my first test call:

```
const recipes = returnRecipesToBrowse(req);
console.log(recipes); // will give you something like Promise {pending}
```

Then this was tried:

```
const recipes = await returnRecipesToBrowse(req);; // will error
console.log(recipes); //undefined
```

What was the final result was:

```
const returnRecipesToBrowse = async (req) => {
   const recipes = await User.findOne({ username: req.user.username })
    .then(recipes =>  userQueryBuilder(recipes))
    .then(queryItems =>  sanitizeDataForIngredientQuery(queryItems))
    .then(recipesObject => recipeIdGetter(recipesObject.data))
    .then(data => detailedRecipeAPISearch(data))
    .then(recipes =>  {return recipes})
    .catch(error => {return error})
  return recipes
};

returnRecipesToBrowse(req); // run the async function at the top-level, since top-level await is not
currently supported in Node
```

I did not need to await on the final returnRecipesToBrowse(req) call, since Node won't exit until its event loop is empty.

When implementing the main code for displaying recipes for browsing, it was discovered that there were certain limitations with using the Spoonacular API. The 'search recipes' which enables a complex search with ingredients and other query parameters like diet and intolerances, proved not useful as it only displays recipes that have all the ingredients in the query not recipes that include one or more of the ingredients. This search was much too specific as we needed to return recipes with one or more of the query ingredients. To supplement the above option, it was decided to use the 'search recipes by ingredients', which will return recipes that include one or more the ingredients in the query, however the returned object is not detailed. Using the object returned above, the recipe ID's were extracted to then use in another query which is ' get recipe information bulk' which returns details recipe information using the recipe ID's as the parameters. The returned object from this query though I believe was limited by the paid tiers of the API. Which meant the preferences list was reduced down to just include vegetarian, vegan, gluten-free, dairy-free, very healthy, cheap, popular, sustainable, and low-fod-map. In future the payment tier may not opted to increase which would enable more preference options.

To overcome the blocker of needing the information from the ingredient search query, but also the information from the get recipe bulk query, the used and missed ingredients were filtered out from the first lot of returned data, then passed onto the next function, so that after the bulk recipe query was returned the two objects could be joined and returned.

CLIENT-SIDE

The initial connecting of the front-end and back-end was started. This started a learning curve with how having the JWT in a cookie works. To begin with registering a user was connected, and logging in a user, this followed some blockers including the register user function on the back-end not

signing a JWT, and on the front-end determining how to keep a user logged in. Local storage was implemented for this issue with the storage housing the username and at the moment the JWT (which is not necessary, but just in place for manual testing). Along with local storage is the state manager being redux.

Sprint 4- 28/12/20 - 10/1/21

▶ Click to expand

STYLING

Foundational styling was done as I went to ensure easy readability of the pages being worked on. This involved implementing a Grid layout from Material UI for styling. The main components used from Material UI include the autocomplete component, Grid, Paper and Buttons. Additional usage will be put in place once the final in depth styling is completed. Initial set up React Toastify for Notifications was put in place for later use. Some refining of a basic footer and the top logo for linking back to the home page was completed. Adjusted the user profile image styling as Adrienne was going to be implementing this code. Not found page was implementing and styled.

Loading screen was implemented with React Loading to enable a loading time frame for browse recipes, fridge, pantry and eventually saved recipes. To ensure the data has loaded correctly. The initial of adding transitions and effects for better user experience was started.

BROWSE RECIPES

Browse Recipe component was created and some test data was put in place via a JSON file to enable to initially styling of the recipe cards. Like the separate ingredient component, using a separate recipe card component means it can be reused when it comes to displaying the saved recipes pages.

Then the start of the coding process for calling the backend to return the recipes for browsing. The services code for calling the DB route was implementing quite quickly, initially with the idea that the DB would be called by clicking the search recipe button. This may be changed back to this. However right now it was changed that that button takes you to the browse recipe page, then a useEffect calls the DB and returns the browse data IF the local storage is empty. If the local storage has browse recipes in it. A major blocker during this project was the object coming up with [object, object]. This was discovered that local storage, in order to pass it around from page to page, requires the data to be stringified then parsed back. Once this was implementing the data displayed from local storage no issue.

The next step is being able to refactor the code to allow for if the user was to do a refresh search, that the browse recipe DB route will be called again and then update local storage. Also to make it obvious to the user that if it was a search with no ingredients, that a random search was made just to show some random recipes.

Issue: check if the server code is working correctly with taking fridge and pantry ingredients for the recipe search.

FRIDGE PANTRY

Adrienne implemented the foundational code for the fridge and pantry components and the functions for connecting to the back end, alongside was some great code for being able to save the state of multiple ingredients for adding to the DB. I just did some refactoring as instead of manually coding the add multiple ingredients, Material UI has this available in the autocomplete component. Additional I was able to incorporate the ingredient services methods.

The connecting of the backend to the frontend was completed for the fridge and pantry for adding, deleting and deleting all ingredients. Some issues along the way were being able to take the array of ingredients from autocomplete and adding them to the array in the DB. Initially it was adding the array to the array in DB, this was able to be resolved by updating on the server side the following:

```
from (newItem)

to (...newItem)
```

Then was the issue of duplicates being able to be added to the list. Adrienne is currently working on a fix for this issue, because as it stands you can add duplicates, but when you delete one, it deletes all as that is the server method of deleting. So to overcome this, a function for stopping duplicates is needed.

USER

A blocker which was causing issues when finding by username and updating for settings and preferences, was this:

```
//ISSUE
User.findOneAndUpdate(req.params.username)

//FIXED
User.findOneAndUpdate({ username:  req.params.username })
```

This above caused issue with for example it would update the wrong users profile image, and I believe it was updating the first user in the DB.

A fix was made for the Regex for registering a user and the username, this was fixed so as to allow for usernames of 5 letters or more.

A major blocker was the user password being rehashed each log in/update. It was thought that this was fixed with a pre mongoose model method, however it was discovered it was just the ingredients adding and saving that was overriding the current users information and rehashing the password. When they was changed to findOneAndUpdate instead of FindByID this issue was fixed.

Another issue was on initial login the users current profile image was not loaded until navigating to the settings page, this was due to the DB not loading the profile on initial home page render. This was fixed by doing a single DB call to get the profile image and save in redux so that it would load on first login.

INGREDIENTS

A conditional was implementing that if no ingredients for fridge or pantry were present it would render a white space filler letting the user know that there are no ingredients present.

DEPLOYMENT

Server code was successfully deployed to Heroku. The client code however was deployed to netlify, but some issues are remaining. Keep working on deployment.

APP

A private route function was set up so that it could be utilised with any routes that require logging in. This then redirects the user to the home page.

Blockers:

Password being hashed on hash. Local storage holding JSON file, needed to pass string and then parse back our to JSON for render

HOME:

A json file with random food jokes was created and utilised on the home page to display random food jokes. This was done as a JSON file to enable new jokes to be added later.

RESET PASSWORD:

For a fuller user experience, I implementing a reset password feature via nodemailer and a built in module in Node.js called crypto which will hash a unique token. This was implemented relatively quickly with an initial post route for clicking reset password, which opens a modal in React with a form which the user enters their email, then on submit it fires a function on the server which will check if the user is in the DB, sending back errors if not, then created a random token via crypto, then inserting this and a token expiring into the users document in MongoDB. Then via Nodemailer using a gmail accounted created for this application, it sends a basic template outlining the instructions for resetting their password with a link that will take the user to a reset password form. The get request for this form involves checking that the token in the link is in the users DB first before allowing the user to see the page. This subsequently the user can enter their new password in the form, which then submits a request to the DB to update the password, using the username, token and date less then expiring to ensure the correct user's password is updated. The main blockers during the process was determining the correct was the insert a field into a document, upsert was not working so after some research and trial and error the below was the correct code:

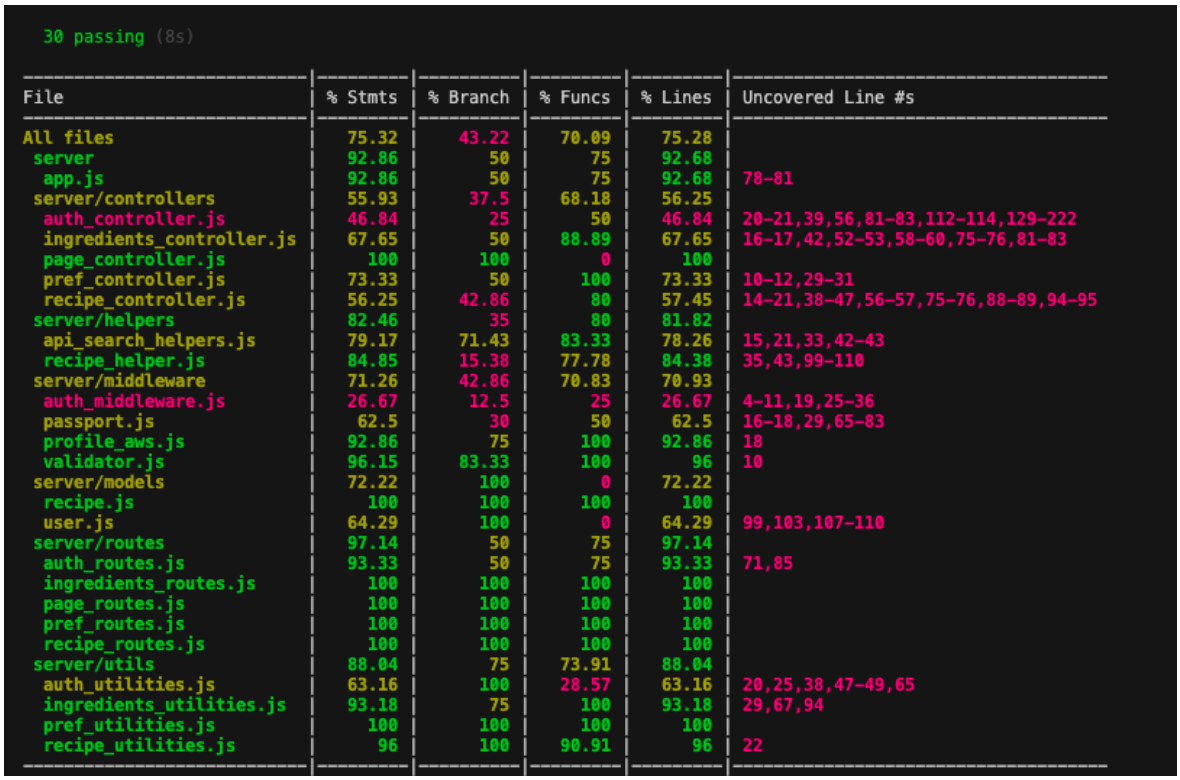
```
{
            returnNewDocument: true,
            new: true,
            strict: false
          }
```

The strict: false allowed me to insert a field that was not in the current schema. Additionally some other blockers, were ensuring there was enough time on the loading screen to make sure the server can fulfill the request, the function which sends the nodemailer can load slow at times, so ensuring there were enough seconds to allow for this was key.

Sprint 5- 10/01- 19/01

▶ Click to expand

REVIEWING TESTING



```
  30 passing (8s)

-------------------------|---------|----------|---------|---------|-------------------------------------
File                     | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-------------------------|---------|----------|---------|---------|-------------------------------------
All files                |   75.32 |    43.22 |   70.09 |   75.28 |
 server                  |   92.86 |       50 |      75 |   92.68 |
  app.js                 |   92.86 |       50 |      75 |   92.68 | 78-81
 server/controllers      |   55.93 |     37.5 |   68.18 |   56.25 |
  auth_controller.js     |   46.84 |       25 |      50 |   46.84 | 20-21,39,56,81-83,112-114,129-222
  ingredients_controller.js |   67.65 |       50 |   88.89 |   67.65 | 16-17,42,52-53,58-60,75-76,81-83
  page_controller.js     |     100 |      100 |       0 |     100 |
  pref_controller.js     |   73.33 |       50 |     100 |   73.33 | 10-12,29-31
  recipe_controller.js   |   56.25 |    42.86 |      80 |   57.45 | 14-21,38-47,56-57,75-76,88-89,94-95
 server/helpers          |   82.46 |       35 |      80 |   81.82 |
  api_search_helpers.js  |   79.17 |    71.43 |   83.33 |   78.26 | 15,21,33,42-43
  recipe_helper.js       |   84.85 |    15.38 |   77.78 |   84.38 | 35,43,99-110
 server/middleware       |   71.26 |    42.86 |   70.83 |   70.93 |
  auth_middleware.js     |   26.67 |     12.5 |      25 |   26.67 | 4-11,19,25-36
  passport.js            |    62.5 |       30 |      50 |    62.5 | 16-18,29,65-83
  profile_aws.js         |   92.86 |       75 |     100 |   92.86 | 18
  validator.js           |   96.15 |    83.33 |     100 |      96 | 10
 server/models           |   72.22 |      100 |       0 |   72.22 |
  recipe.js              |     100 |      100 |     100 |     100 |
  user.js                |   64.29 |      100 |       0 |   64.29 | 99,103,107-110
 server/routes           |   97.14 |       50 |      75 |   97.14 |
  auth_routes.js         |   93.33 |       50 |      75 |   93.33 | 71,85
  ingredients_routes.js  |     100 |      100 |     100 |     100 |
  page_routes.js         |     100 |      100 |     100 |     100 |
  pref_routes.js         |     100 |      100 |     100 |     100 |
  recipe_routes.js       |     100 |      100 |     100 |     100 |
 server/utils            |   88.04 |       75 |   73.91 |   88.04 |
  auth_utilities.js      |   63.16 |      100 |   28.57 |   63.16 | 20,25,38,47-49,65
  ingredients_utilities.js |   93.18 |       75 |     100 |   93.18 | 29,67,94
  pref_utilities.js      |     100 |      100 |     100 |     100 |
  recipe_utilities.js    |      96 |      100 |   90.91 |      96 | 22
-------------------------|---------|----------|---------|---------|-------------------------------------
```