Cookable

By:
Copyright Team 5 CSC 648-848 Spring 2019
Trevor Sampson (Team Lead, Front & Back-end) tsampson1@mail.sfsu.edu
Thanh Dip (Back-end Lead Developer, Database & Back-end)
Pyae Naing (Back-end Developer, Github Master)
Tony Rodriguez (Front-end Lead Developer)
Shivam Rai (Front-end Developer, UI/UX Developer)

Milestone 4

May 12, 2019	Milestone 4 Version 1
April 19, 2019	Milestone 2 Version 2
April 3, 2019	Milestone 2 Version 1
March 28, 2019	Milestone 1 Version 2
March 14, 2019	Milestone 1 Version 1

1. Product Summary

Product Name: Cookable
Product URL: http://ec2-52-53-159-55.us-west-1.compute.amazonaws.com

1. A casual user can find recipes without registering.
2. A casual user can search ingredients to find recipes.
3. A casual user can search for recipes by name.
4. Recipes will be displayed to user on homepage.
5. New users shall be able to register.
6. A privacy policy shall be provided to new user when registering.
7. A privacy policy shall be viewable in user profile settings.
8. A registered user will be able to manage their own profile information.
9. A registered user will be able to manage their own pantry.
10. A registered user will be able to manage/create their own recipes.
11. A registered user will be able to manage their "favorite" recipes.
12. An administrator will be able to remove recipes.
13. A free text search box shall be used for performing recipe searches.
14. All private information will be stored and encrypted appropriately.
15. All SQL statements shall be prepared by an ORM.

We are cookable and it is our goal to provide you with the best recipe searching experience possible. There are many other websites out there that offer the option to search recipes, but none of them have a way of storing which ingredients you currently own in your home. How many times have you searched for a recipe, only to find out that you don't own all of the ingredients needed to make it? With Cookable, you won't have that issue. Registered users have the option to use what we call the Pantry, which will allow them to store the ingredients they currently own on their profile. Once the ingredients are added to the Pantry, you can then search for recipes for which you currently own all of the ingredients. You no longer have to make a last minute trip to the grocery store, because you can rest assured that you own everything needed to make the recipe. With Cookable, your recipe search just got easier. So, what's for dinner?

2. Usability Test Plan

- Test objectives: For our usability test, the Cookable team would like to test the following:

- Searching Recipes.
- Searching Ingredients.
- Adding a recipe.
- Viewing a recipe.

- We would want to gauge user experience about searching the ingredients by recipe, this includes both unregistered and registered users.
- We would also like user to view and share recipes. For a registered user, we would like to the user be able to Create new recipes and add favorites from existing recipes.
- We feel that these are basic components of Cookable, thus being basic functions that we are providing to users. We feel that it's important that users find these functions easy to use.

- Test description:

- System Setup: Website is deployed on Amazon Web Service running on ubuntu. The database is MySQL server. The user would be using at least latest 5th iteration of Google Chrome from date. Operating systems would be Windows 10, OSX Mojave or earlier(3 iterations) or any flavor of linux system that is running chromium.
- Intended users: These are general public, we feel that cooking is important part of everyone's day to day routine and our site would be used to aid in cooking. The test will be performed with SFSU students who are not part of the Computer Science department.
- URL of the system to be tested: http://ec2-52-53-159-55.us-west-1.compute.amazonaws.com What is to be measured: We will measure the user's satisfaction with Cookable when searching for recipes/ingredients, viewing/adding a recipe.

- Usability Task description:

Describe the task testers do before filling out the Likert questionnaire – a few lines (check class slides) - Questionnaire: Provide 3 Lickert scale questions getting user satisfaction after the above task has been performed, in a proper format as it is to be used by reviewer (check class slides) – 3/4 page

- Usability Task description:

- Task 1: Log on to Cookable and search for recipes.

Task	**Description**
Task	Search for a recipe by name.
Machine state	Home page of *Cookable* http://ec2-52-53-159-55.us-west-1.compute.amazonaws.com**;** user is not registered on the website.
Successful completion criteria	User is displayed a list of recipes with supplied name in search.
Benchmark	<=1 minute.

- Task 2: Register and sign in to Cookable.

Task	**Description**
Task	Create an account on cookable by supplying valid user information.
Machine state	Home page of *Cookable* http://ec2-52-53-159-55.us-west-1.compute.amazonaws.com **;** user logged into an account
Successful completion criteria	Check that user has an account through MyProfile Settings page.
Benchmark	<=5 minutes.

- Task 3: Login into the website and create a recipe.

Task	**Description**
Task	Creating a recipe
Machine state	Home page of *Cookable* http://ec2-52-53-159-55.us-west-1.compute.amazonaws.com , user logged into an account.
Successful completion criteria	Check that the updated information is present and can be seen in my Recipes.
Benchmark	<=5 minutes.

Questionnaire:

The recipe(s) I searched for were easy to identify & displayed efficiently:
Strongly Agree Agree Neutral Disagree Strongly Disagree
Comments:

It is very easy to find a recipe:
Strongly Agree Agree Neutral Disagree Strongly Disagree
Comments:

The process for searching a recipe was very easy and efficient:
Strongly Agree Agree Neutral Disagree Strongly Disagree
Comments:

The process of creating an account was very simple:
Strongly Agree Agree Neutral Disagree Strongly Disagree
Comments:

It was very easy to edit the details of an account:
Strongly Agree Agree Neutral Disagree Strongly Disagree
Comments:

The website was very consistent:
Strongly Agree Agree Neutral Disagree Strongly Disagree
Comments:

3. QA Test Plan

- Test objectives:

- Purpose: Quality assurance tests are carried out to validate that the functionalities of the system conform to the requirements. Multiple tests are carried out to validate that all functionalities work in accordance with the specifications mentioned in the previous phases.

- Hardware and Software Setup (including URL)

- Hardware Setup: Modern PC that has a web browser (Firefox, Chrome).
- Software Setup: Cookable Home Page on http://ec2-52-53-159-55.us-west-1.compute.amazonaws.com (MySQL database is preloaded with 37 recipes to search)

- QA Test Plan:

The user will follow the test cases below in searching for the recipe they wish to cook. Recipe is determined by ingredient type or meal name. User will be unable to add a recipe if they do not have a registered account on Cookable.

- Features To Be Tested:

Recipes can be added and viewed by the user once an account has been made for them on Cookable. The search and adding recipe function is still to be tested.

- Search for a pre-existing recipe using the recipe name in the search bar.
    a. Search for the recipes that has the name “Pork” in the search bar.
    b. Search for the specific recipe “Pork Stew in Green Salsa” in the search bar.
    c. View the recipe “Pork Stew in Green Salsa” and it’s cooking instructions.
- Search for a pre-existing recipe using the ingredient name in the search bar.
    a. Search for the recipe that contains “eggs” in the search bar
    b. View the recipe, food description, and cooking instructions
- Adding a new recipe to the database.
    a. Will require name of recipe, ingredients, cooking time, meal type and instructions
    b. Can only be added by a user with a cookable account

	Description	Test Input	Expected Output	PASS / ON TRACK / FAIL
1	Search for the recipes that has the name “Pork” in the search bar	“pork”	A list of recipes which includes the search term in all of the recipe name.	PASS
2	Search for the specific recipe “Pork Stew in Green Salsa” in the search bar..	“Pork Stew”	A list of recipes which includes the search term in all of the recipe names.	PASS
3	View the recipe “Pork Stew in Green Salsa” and it’s instructions	Under the recipe name, select the button “view”.	A list of instructions on how to cook the “Pork Stew in Green Salsa” will be displayed.	ON TRACK
4	Search for a recipe that contains the ingredient “egg” using the search bar.	“Egg”	A list of recipes which includes the searched ingredient in the ingredient list.	PASS
5	View the first recipe that contains the ingredient “egg” and it’s instructions.	Under the recipe name, select the button “view”.	A list of instructions on how to cook the viewed recipe will be displayed.	ON TRACK

4. Code Review

Coding Style: Our coding style is focused on maintaining both an easy to read and easy to understand structure. When writing code, any team member on the project should be able to reference the code and understand what the purpose of the code is. For both front and back-end camelCase naming is to be used.

Back-end:

Formatting

Braces are always needed even for simple if statements and every control structure starts with a open brace in line and ends with a close brace column line below it.

I.E

```
while (x == y) {
 something();
 somethingElse();
}
```

One-liners are fine as long as it is easier to read and the logic follows like a sentence.

I.E

```
where: { recipeName: { [Op.like]: "%" + req.query.recipeName + "%" } }
```

All indentations and tabs are 1 space. Line ends always have a semi-colon.

Naming

Variable names should be relatively short and descriptive with as little abbreviations as possible. 25 characters is fine but 30 and is too much.
No underscores. All names of variable, functions, etc. are in camel case.

Functions

Functions names for routes start with an action and then the subject of that action. So, to add a user to the database, the function name would be “addUser”. On that note, main tables like Users

and Recipes use the actions “create” and “delete” while secondary tables that derive that use the action “add” and “remove”.
I.E
“createUser”
“addIngredientToPantry”

All individual functions, classes, or models are exported using the “Exports” module of Node.js so things can be accessed by

Comments

Comments are used to format and section off code to make things more readable and easy to scan with the naked eye. Other comments should be used when necessary to explain code that is not regularly used or does not follow normal logic.

The code was in general written to be easily maintained and readable.

Front-end:

For the front-end code the two main things we addressed were how we were going to structure our React components and functions. For component creation there are two main approaches:

Option 1: Javascript function

```
function Welcome(props) {
        return <h1>Hello, {props.name}<h1>;
}
```

Option 2: ES6 class

```
class Welcome extends React.Component {
        render() {
                return <h1>Hello, {props.name}<h1>;
        }
}
```

We decided to go with the ES6 class implementation due to them offering more features that the Javascript function implementation. For our function declaration there are two forms to choose from:

Option 1: Javascript function

```
updateUser(user) {
        this.setState({  “user”: user });
}
```

Option 2: ES6 arrow function

```
updateUser = (user) => {
        this.setState({ “user”: user });
}
```

We decided to use the ES6 arrow function it automatically binds the function to the context in which the function is being called. This results in less bugs when calling the functions. The use of regular javascript functions requires that you manually bind all functions to component they are being created in, which can be tedious.

For our code review we chose our search functionality to be reviewed. We request that the reviewer verify that the code we provide is written to the standards as described above.

Back-end Search Code

```
const Recipe = require("../models/recipe");
const Favorites = require("../models/favorites");
const Likes = require("../models/likes");
const Reviews = require("../models/reviews");
const Sequelize = require("sequelize");
const RecipeImages = require("../models/recipeImages");
const ingredientList = require("../models/ingredientsListFulls");
const instructions = require("../models/instructions");
const defaultImageUrl =
"https://www.creativefabrica.com/wp-content/uploads/2018/09/Crossed-spoon-and-fork-logo-by-yahyaanasatokil
lah-580x387.jpg";
const Op = Sequelize.Op;

exports.searchRecipe = function (req, res) {
 (limit = 20),
   Recipe.findOne({
     where: { recipeName: { [Op.like]: "%" + req.query.recipeName + "%" } }
   })
     .then(recipes => {
       res.json({ recipe: recipes });
     })
     .catch(function (err) {
       res.send("Error");
     });
};


exports.searchRecipe = async function (req, res) {
 try{
 let recipeSearch;
 let ingredientSearch;
 recipeSearch = await searchByRecipe(req.query.recipe);
 ingredientSearch = await searchByIngredient(req.query.recipe);
 let arr3 = recipeSearch.concat(ingredientSearch).unique();
 res.status(200).json(arr3);
 }
 catch(e){
   res.status(200).send('Error: ' + e);
 }
};

Array.prototype.unique = function() {
 var a = this.concat();
 for(var i=0; i<a.length; ++i) {
     for(var j=i+1; j<a.length; ++j) {
         if(a[i].recipeID === a[j].recipeID)
             a.splice(j--, 1);
     }
 }
 return a;
};

async function searchByRecipe(req) {
 let recipe;
 let arr = [];
 let recipeurl;
 try {
   recipe = await getRecipeByName(req);
   arr = await getarray(recipe);
   recipeurl = await getImageUrl(arr);
   recipe = await combinethem(recipe, recipeurl)
   return recipe;
```

```
 } catch (e) {
   return e;
 }
}


async function searchByIngredient(req) {
 let recipe;
 let ingredientused;
 let arr = [];
 let recipeurl;
 try {
   ingredientused = await getRecipeByIngredient(req);

   arr = await getarrayinorder(ingredientused);
   recipe = await getRecipei(arr);
   recipeurl = await getImageUrl(arr);
   recipe = await combinethem(recipe, recipeurl)
   return recipe;
   // res.json(recipe);
 } catch (e) {
   return e
   // res.send('Error');
 }
}


// helper functions
function getRecipeByName(req) {
 return Recipe.findAll({
   where: {
     recipeName: { [Op.like]: '%' + req + '%' }
   }, raw: true
 });
}

function getRecipeByIngredient(req) {
 return ingredientList.findAll({
   where: {
     ingredientsFull: { [Op.like]: '%' + req + '%' },
   }, raw: true
 })
}

function getarrayinorder(recipe) {
 let arr = [];
 let temp = -1;
 for (let i = 0; i < recipe.length; i++) {
   if (temp != recipe[i].recipeID) {
     arr[i] = recipe[i].recipeID;
     temp = arr[i];
     recipe[i].url = defaultImageUrl;
   }
   else {
     recipe.splice(i, 1);
     i--;
   }
 }
 return arr;
}

function getarray(recipe) {
 let arr = [];
 for (let i = 0; i < recipe.length; i++) {
   {
     arr[i] = recipe[i].recipeID;
```

```
    recipe[i].url = defaultImageUrl;
   }
 }
 return arr;
}

function combinethem(recipe, recipeUrl) {
 for (let i = 0; i < recipe.length; i++) {
   for (let j = 0; j < recipeUrl.length; j++) {
     if (recipe[i].recipeID === recipeUrl[j].recipeID) {
       recipe[i].url = recipeUrl[j].recipeImageDir;
       break;
     }
   }
 }
 return recipe;
}

function getRecipe() {
 return Recipe.findAll({
   order: [[Sequelize.literal("RAND()")]],
   limit: 8
 });
}

function getImageUrl(arr) {
 return RecipeImages.findAll({
   where: {
     recipeID: arr
   }
 });
}

function getRecipei(arr) {
 return Recipe.findAll({
   where: {
     recipeID: arr
   }
 });
}
```

Reviewed Back-end Search Code with comments:

```
//Great use of clearly understandable const variable names, and easy to follow references
//to the subdirectories in which they are contained!
//Header is present and clear

const Recipe = require("../models/recipe");
const Favorites = require("../models/favorites");
const Likes = require("../models/likes");
const Reviews = require("../models/reviews");
const Sequelize = require("sequelize");
const RecipeImages = require("../models/recipeImages");
const ingredientList = require("../models/ingredientsListFulls");
const instructions = require("../models/instructions");
const defaultImageUrl =
"https://www.creativefabrica.com/wp-content/uploads/2018/09/Crossed-spoon-and-fork-logo-by-yahyaanasatokil
lah-580x387.jpg";
const Op = Sequelize.Op;


//Very easy to follow the parameter requirements
//of your createRecipe
//Header is present and clear
exports.createRecipe = function (req, res) {
  Recipe.create({

    //Easy to infer based on variable names and understand recipe, description,
   //cuisine type, time to cook, author, caloric values, and ID
        //Header is present and clear
    recipeName: req.body.recipeName,
    description: req.body.description,
    cuisine: req.body.cuisine,
    calorieCount: req.body.calorieCount,
    cookingTime: req.body.cookingTime,
    authorName: req.body.authorName,
    userID: req.body.userID,
    isUserCreated: req.body.isUserCreated
  })
        //Outputting the results of the query
        //using console log and send
    .then(recipe => {
      console.log(recipe.get({ plain: true }));
      res.send(recipe.get({ plain: true }));
    })
        //Error checking, great job
    .catch(err => {
      res.send("Error");
      console.log(err);
    });
};

exports.searchRecipe = function (req, res) {
  (limit = 20),
    Recipe.findOne({
      where: { recipeName: { [Op.like]: "%" + req.query.recipeName + "%" } }
    })
```

```
      .then(recipes => {
        res.json({ recipe: recipes });
      })
      .catch(function (err) {
        res.send("Error");
      });
};

//Header is present and clear
exports.getRecommendation = async function (req, res) {
  let recipe;
  let arr = [];
  let recipeurl;
  try {
    recipe = await getRecipe();
    arr = await getarray(recipe);
    recipeurl = await getImageUrl(arr);
    recipe = await combinethem(recipe, recipeurl)
    res.json(recipe);
  } catch (e) {
    res.send('Error');
  }
};

//Header is present and clear
exports.searchRecipe = async function (req, res) {
  try{
  let recipeSearch;
  let ingredientSearch;
  recipeSearch = await searchByRecipe(req.query.recipe);
  ingredientSearch = await searchByIngredient(req.query.recipe);
  let arr3 = recipeSearch.concat(ingredientSearch).unique();
  res.status(200).json(arr3);
  }
  catch(e){
    res.status(200).send('Error: ' + e);
  }
};

Array.prototype.unique = function() {
  var a = this.concat();
  for(var i=0; i<a.length; ++i) {
      for(var j=i+1; j<a.length; ++j) {
          if(a[i].recipeID === a[j].recipeID)
              a.splice(j--, 1);
      }
  }

  return a;
};

//Header is present and clear
async function searchByRecipe(req) {
  let recipe;
  let arr = [];
  let recipeurl;
  try {
```

```
    recipe = await getRecipeByName(req);
    arr = await getarray(recipe);
    recipeurl = await getImageUrl(arr);
    recipe = await combinethem(recipe, recipeurl)
    return recipe;
//if possible maybe name and define the error that you are catching
//so that I can debug the program easily if I am given the program
  } catch (e) {
    return e;
  }
}

//Header is present and clear
 async function searchByIngredient(req) {
  let recipe;
  let ingredientused;
  let arr = [];
  let recipeurl;
  try {
    ingredientused = await getRecipeByIngredient(req);

    arr = await getarrayinorder(ingredientused);
    recipe = await getRecipei(arr);
    recipeurl = await getImageUrl(arr);
    recipe = await combinethem(recipe, recipeurl)
    return recipe;
//Remember to remove the commented out code though I understand
//that it is leftover and that your final version will not include these comments
//also
    // res.json(recipe);
  } catch (e) {
    return e
    // res.send('Error');
  }
}

//Again, I can clearly identify that in order to use the viewRecipe function,
//we call the hasMany function and pass in a request and response function.
exports.viewRecipe = function (req, res) {

//And Again, it is clear that we are checking the Recipe images, Likes,
//Favorites, Reviews, etc, using the foreign key from the database table
//referencing the recipeID associated with the Recipe object that we are viewing
//Header is present and clear
  Recipe.hasMany(RecipeImages, { foreignKey: 'recipeID' });
  Recipe.hasMany(Likes, {foreignKey: 'recipeID'});
  Recipe.hasMany(Favorites, {foreignKey: 'recipeID'});
  Recipe.hasMany(Reviews, {foreignKey: 'recipeID'});
  Recipe.hasMany(instructions, {foreignKey: 'recipeID'});
  Recipe.hasMany(ingredientList, {foreignKey: 'recipeID'});

//A little confused about belongsTo but given the scope of this review process
//I don't expect to know what the belongsTo function is since it's not
//provided due to the scope of this code review
  RecipeImages.belongsTo(Recipe, { foreignKey: 'recipeID' });
  Likes.belongsTo(Recipe, { foreignKey: 'recipeID' });
  Favorites.belongsTo(Recipe, { foreignKey: 'recipeID' });
```

```
  Reviews.belongsTo(Recipe, { foreignKey: 'recipeID' });
  instructions.belongsTo(Recipe, { foreignKey: 'recipeID' });
  ingredientList.belongsTo(Recipe, { foreignKey: 'recipeID' });

  Recipe.findOne({
    where: { recipeID: req.params.id }, include: [RecipeImages, Likes, Favorites, Reviews, instructions,
ingredientList]
  }).then(recipe => {
    res.send(recipe);
  }).catch(err => res.status(500).send('Error: ' + err));
}

exports.getRecipeInstruction = function (req, res) {

//Great naming convention, we are finding a Recipe
//based on the query constructed from the provided
//recipe name
  Recipe.findOne({
    where: { recipeName: { [Op.like]: "%" + req.query.recipeName + "%" } }
  }).then(recipe => {

//Great follow through, I can see that when we do find the recipe,
//we pull u the instructions by using the recipe ID
    if (recipe != null) {
      instructions.findAll({
        where: {
          recipeID: recipe.recipeID
        }
      }).then(i => {
        res.json(i);
//If possible to name the error, try to name it
      }).catch(e => {
        console.log(e);
        res.status(500).send('Errror: ' + e);
      })
    }
    else {
//Great status being sent, very straightforward, comprehendable
      res.status(404).send('Cannot find Instruction')
    }
  }).catch(e => {
    res.send('Error');
    console.log(e)
  });

};

//Well named helper functions, we get the recipes by name, or ingredients,
//we have an ordered array too! Great!

// helper functions
function getRecipeByName(req) {
  return Recipe.findAll({
    where: {
//Great use of SQL selection parameters here
      recipeName: { [Op.like]: '%' + req + '%' }
    }, raw: true
```

```
  });
}

function getRecipeByIngredient(req) {
  return ingredientList.findAll({
    where: {
      ingredientsFull: { [Op.like]: '%' + req + '%' },
    }, raw: true
  })
}

function getarrayinorder(recipe) {
  let arr = [];
  let temp = -1;

//It looks like we are iterating over an array
//and calling the default image link
  for (let i = 0; i < recipe.length; i++) {
    if (temp != recipe[i].recipeID) {
      arr[i] = recipe[i].recipeID;
      temp = arr[i];
      recipe[i].url = defaultImageUrl;
    }

//A little confused here although it may just be escaping me at the moment
//how to follow what is being spliced, looks like an internal mechanism
//for your array. Good job!
    else {
      recipe.splice(i, 1);
      i--;
    }
  }
  return arr;
}

//It looks like you're building an array
//and defining the default image url parameter
//that will be used to render the recipe image,
//by accessing the url variable stored in each recipe
//object. Very good job, easy to follow.
function getarray(recipe) {
  let arr = [];
  for (let i = 0; i < recipe.length; i++) {
    {
      arr[i] = recipe[i].recipeID;
      recipe[i].url = defaultImageUrl;
    }
  }
  return arr;
}

//Here you are combining the array of recipe IDs
//with the URL to each recipe,
//and assigning the recipe url to the
//directory associated with the recipe itself
//(building the recipe image URL?)
//Looks fancy!
```

```
function combinethem(recipe, recipeUrl) {
  for (let i = 0; i < recipe.length; i++) {
    for (let j = 0; j < recipeUrl.length; j++) {
      if (recipe[i].recipeID === recipeUrl[j].recipeID) {
        recipe[i].url = recipeUrl[j].recipeImageDir;
        break;
      }
    }
  }
  return recipe;
}


function getRecipe() {
  return Recipe.findAll({
    order: [[Sequelize.literal("RAND()")]],
    limit: 8
  });
}

function getImageUrl(arr) {
  return RecipeImages.findAll({
    where: {
      recipeID: arr
    }
  });
}

function getRecipei(arr) {
  return Recipe.findAll({
    where: {
      recipeID: arr
    }
  });
}

//Peer Review Completed by Hunter Graves, Back-End Lead for Team 1
//At approximately 8pm, May 12th, 2019
//
//Trevor, all in all I'd say that you have constructed a very easy to follow
//search function and associated helper functions and algorithms for your arrays.
//One thing to note is that it'd be useful to have some names associated with your
//errors that you are throwing, where they are named "e". Other than that, I'd say
//that you could use some comments mainly around the algorithmic areas below, where
//you are returning inorder arrays of Recipes and also fetching the URL and directory
//paths for your recipes. Headers are very clearly defined and the naming convention
//was phenomenal, it was incredibly straightforward code. Your team's use of
//naming convention and indentations allowed me to develop a high level understanding
//of your code. Great Job!
//
//-Hunter Graves
//Backend Lead
//Team 1
//
```

Email Transcript:

Hunter Laszlo Graves

Sun 5/12/2019 8:13 PM

To:Trevor Michael Sampson <tsampson1@mail.sfsu.edu>; Cc:Anthony Dean Branda <abranda@mail.sfsu.edu>;

1 attachments (10 KB) REVIEWEDrecipe.controller.txt;

Trevor,

Embedded in this email is a copy of the final remarks regarding the code that you submied to me for review. These final remarks are also embedded in commented notaon at the boom of your email-friendly text version of your javascript file that you and your team provided for the review.

Please note: I have renamed the email friendly text version of your javascript file to: "REVIEWEDrecipe.controller.txt", and it is aached in this newly named format.

I hope that you find these remarks easy to view both inside of this email, and inside of your file.

Peer Review Completed by Hunter Graves, Back-End Lead for Team 1 At approximately 8pm, May 12th, 2019

Trevor, all in all I'd say that you have constructed a very easy to follow
search funcon and associated helper funcons and algorithms for your arrays.
One thing to note is that it'd be useful to have some names associated with your errors that you are throwing, where they are named "e". Other than that, I'd say
that you could use some comments mainly around the algorithmic areas below, where you are returning inorder arrays of Recipes and also fetching the URL and directory paths for your recipes. Headers are very clearly defined and the naming convenon was phenomenal, it was incredibly straighorward code. Your team's use of
naming convenon and indentaons allowed me to develop a high level understanding of your code. Great Job!

-Hunter Graves Backend Lead Team 1

From: Trevor Michael Sampson
Sent: Sunday, May 12, 2019 7:39:08 PM **To:** Hunter Laszlo Graves
Cc: Anthony Dean Branda
Subject: Re: Code Review for Milestone 4

Hunter,

Overall I think the code that you sent me looks very good. Even though I am very unfamiliar with php, your team's naming convenons and code structure made it very easy to understand what the funcons are meant to do.

The only thing I would add is some descripons/headers to explain how the class and funcons are meant to be used. This will allow future contributors to quickly and easily incorporate and use your code. It would also be nice if you added some comments explaining the steps to each funcon. For example, on line 29 you could explain that the line is used to establish the SQL connecon. Other than that everything looks really good.

Aached is the revised copy of your code with comments. I hope my comments are helpful and useful to you and your team.

Best, Trevor

From: Trevor Michael Sampson
Sent: Sunday, May 12, 2019 7:26:14 PM **To:** Hunter Laszlo Graves
Cc: Anthony Dean Branda
Subject: Re: Code Review for Milestone 4

Hunter,

Sorry for the trouble with the file. Aached is a copy of the code as a txt file. Let me know if you need anything else.

Best, Trevor

From: Hunter Laszlo Graves
Sent: Sunday, May 12, 2019 7:24:23 PM **To:** Trevor Michael Sampson
Cc: Anthony Dean Branda
Subject: Re: Code Review for Milestone 4

Trevor,

It seems that there is an Outlook feature blocking our team from being able to download the aached JS file that you previously sent. I apologize for the delay, it was blocked on my work computer and I thought that it was blocked due to the worksite's internal security. Can you please resend the file as a .txt file or perhaps copy and paste it directly into the email?

Thank you,
-H

From: Hunter Laszlo Graves
Sent: Sunday, May 12, 2019 7:15:43 PM **To:** Trevor Michael Sampson
Cc: Anthony Dean Branda
Subject: Re: Code Review for Milestone 4

Trevor,

My apologies for the delay. The aached list.php file has been aached for your review process. I will respond shortly with my observaons regarding your code.

Thank you, -H

From: Trevor Michael Sampson
Sent: Sunday, May 12, 2019 5:51:11 PM **To:** Hunter Laszlo Graves
Cc: Anthony Dean Branda
Subject: Re: Code Review for Milestone 4

Hi Hunter,

Thanks for contacng me in regards to your code review. I see the code in the pdf, but I was wondering if it would be possible for you to send me the file for the code so that I can easily mark it with comments.
Aached is a copy of our code style documentaon and code for you to review. Thanks again for taking the me to work with us. It is greatly appreciated.

Please let me know if there is anything else you need from me.

Best,
Trevor Sampson

From: Hunter Laszlo Graves
Sent: Sunday, May 12, 2019 4:19:59 PM **To:** Trevor Michael Sampson
Cc: Anthony Dean Branda
Subject: Code Review for Milestone 4

Hello Trevor,

Please find on pages 9 and 10, in the aached PDF file, the secon of code that our team has requested for you to review.

Thank you for your me and your consideraon,
-H

5. Securities self-check

Below are list of assets that are needed to be protected and threats that might be possible. A more detailed explanation will follow.

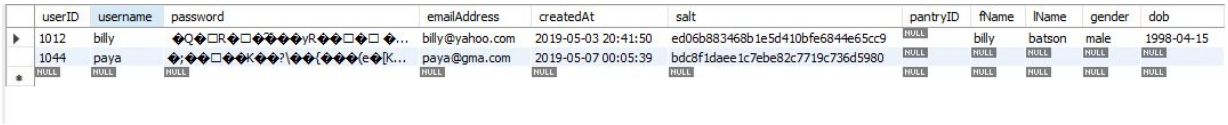
Major assets:

ASSET	VALUE	EXPOSURE
Node.js server	High	High. Would have to restore whole system for app to work.
Database (Includes recipes and ingredient information)	High	High. Main functionality of app is to search through this info. Would need to restore all data.
User's information including password	High	High. Would harm users.
Images	Low	High. Makes site look bad.

Threat	Probability	Control	Feasibility
Unauthorized user gains access to other user's data.	Low	Authenticate with username and password.	Basic implementation with encryption.
Unauthorized user breaks database or server.	Low	User highest security standards for protecting server access and backup offsite..	Cost scales with data held in database. Make sure secure keys are only distributed as needed.

Encryption:

All user passwords are encrypted using **pbkdf2** and stored as a **SHA512** with a **16 byte salt** to prevent rainbow attacks. No password given is handled or stored in plaintext after the processing of the input from the user.



userID	username	password	emailAddress	createdAt	salt	pantryID	fName	lName	gender	dob
1012	billy	◆Q◆□R◆□◆◆◆◆yR◆◆□◆□◆ ◆...	billy@yahoo.com	2019-05-03 20:41:50	ed06b883468b1e5d410bfe6844e65cc9	NULL	billy	batson	male	1998-04-15
1044	paya	◆;◆◆□◆◆K◆◆?\◆◆{◆◆◆(e◆[K...	paya@gma.com	2019-05-07 00:05:39	bdc8f1daee1c7ebe82c7719c736d5980	NULL	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Example of password stored:
◆;◆◆◆◆K◆◆?\◆◆{◆◆◆(e◆[K
◆\◆c◆◆Q(X◆q4q◆N◆Wt◆?◆
◆~裩_c◆◆◆◆`i◆[◆◆◆u3B◆◆(
◆◆◆◆1◆v◆=◆i◆;◆6*◆8◆◆◆◆J◆◆

It shows up strange because it used all kinds of data that this document does not support in displaying.

Route protection:

Any route that returns protected information is validated by passing unique Java Web Tokens. Each JWTs are generated for users and that will give them access to protected information, usually their own information.

Data validation:

Data inputs for search or general database access all go through **Sequelize,** an ORM middleware for MySQL and other RDBMSs. It automatically handles SQL injection attacks by escaping replacements. Any dangerous characters will only be processed as characters and not commands.

The upside and downside to this is the security is left to Sequelize which may have flaws but at the same time they are software made to validate and perform SQL commands.. In the case that there is a compromise, highly vulnerable routes can easily be changed to whitelist only certain characters and then passed through Sequelize for extra security. For now it is unnecessary since it will introduce unneeded complexity.

6. Adherence to Non-functional specs self-check

Security:

1. Login or creating account shall be a requirement for all user who wishes to save the information (DONE)
2. The username will not be available if another user has already created the account with the same name (DONE)
3. There will not be any restrictions and limits for a password. (DONE)
4. The user will be stayed logged in unless he/she refresh the page. (DONE)

Performance:

1. The load time should not take more than 1 min. (DONE)
2. The search shall be executed in the background. (DONE)
3. Query shall be executed in the background. (DONE)

Availability:

1. The server shall be up for as long as admin wish it to be. (DONE)
2. The server will be down for no more than 1 day if there is any maintenance. (DONE)
3. The server shall not crash for any client-side reasons. (ON TRACK)

Recovery:

1. If there is any event where server crash, recover time shall not take longer than 1 day. (DONE)
2. In the case of server failure, there should be through revision for the server log (DONE)

Data and privacy:

1. Data will be backed up every week. (DONE)
2. The administrator can view and look at any account data, including username and password. (ON TRACK)
3. There will not be any image uploads for the profile picture. (DONE)
4. All passwords will be encrypted with PBKDF2. (DONE)

Conformance with Coding Standards:

1. The coding style will be professional and easy to read. (ON TRACK)
2. The only working code will be pushed or submitted. (DONE)
3. All the code will be tested before submitting. (DONE)
4. Any error will be stored in the log and be reviewed. (DONE)

5. Any error shall be handled in a way that does not affect the functionality of the site. (DONE)
6. This site shall be tested and debugged thoroughly before final submission. (DONE)

Compatibility:

1. The site shall be compatible with the Internet Explorer 10+. (DONE)
2. The site shall be compatible with the Firefox 60.0+. (DONE)
3. The site shall be compatible with the Chrome 65+. (DONE)
4. The site shall be compatible with devices running Android 7.0+ with Firefox or Chrome as specified above. (ON TRACK)
5. The site shall be compatible with devices running IOS 11.4.1+ with Firefox or Chrome as specified above. (ON TRACK)

Product's Standard

1. The application will look casual and user-friendly. (DONE)
2. The site will be simple to navigate and easy to be used by anyone. (ON TRACK)
3. The site will provide related or recommended products base on the current product. (DONE)
4. The site will look the same across all different browser. (DONE)