



Dr. Tu COP 4710

RECIPE TRACKER

**Wade Smith
Cedric Nicolas
Abdulrahman Aldhubaiban**

i. Introduction

For this project, we decided to build a simple Web Interface that users can interact with in order to find recipes easily. The motivation behind the idea is that as we all need to eat but may not have the time or money to go get groceries or go out to eat, you may need to rely on what is already in your fridge. As such, in our tool, there is a feature to search for recipes by ingredient, allowing one to choose an ingredient they already have and see where they can go with it.

ii. Client Functionality

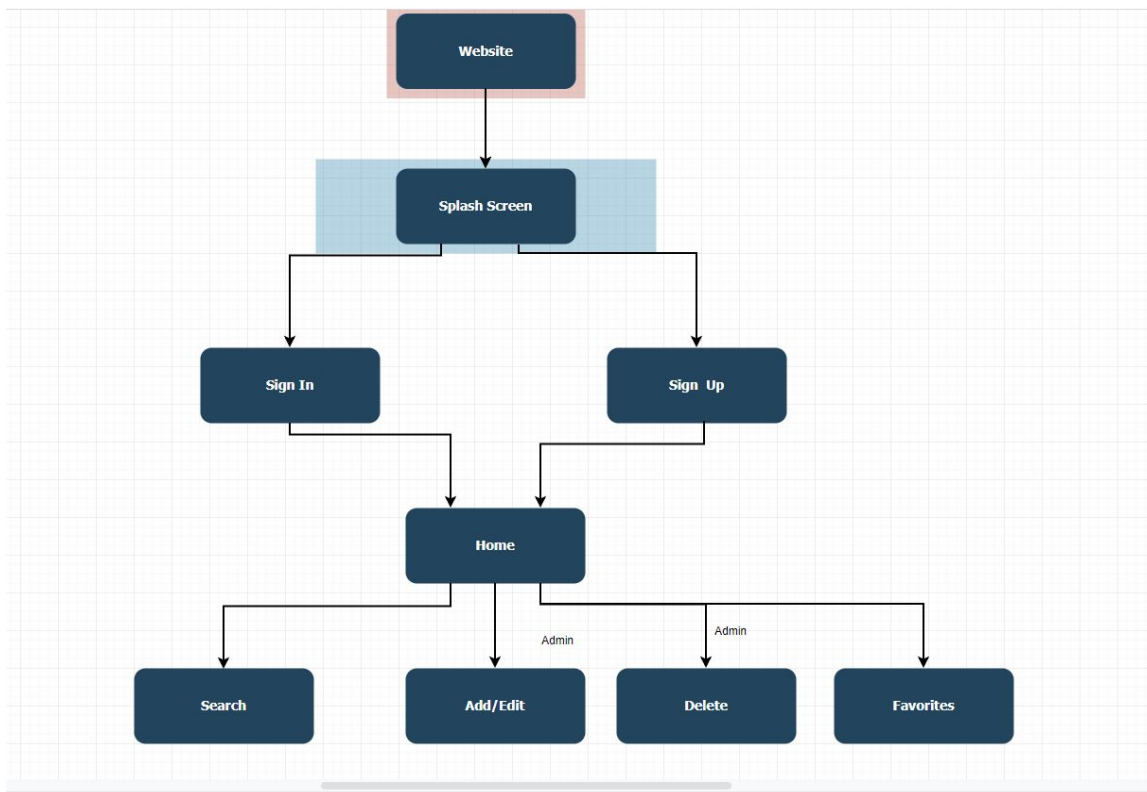


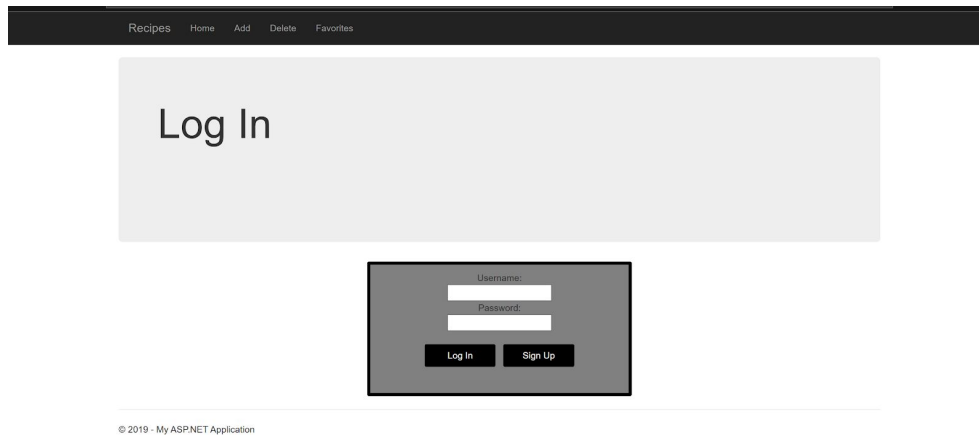
Figure 1: Website Flowchart

As seen in Figure 1, the user needs to sign in/up before interacting with the website. Once you are signed in as user you are then able to search either by ingredient or recipe name, view the ingredients required for the recipe, and add them to your favorites if you wish to do so. However, in order to create, update, or destroy a recipe, the user must have admin privileges.

Interface: ☐

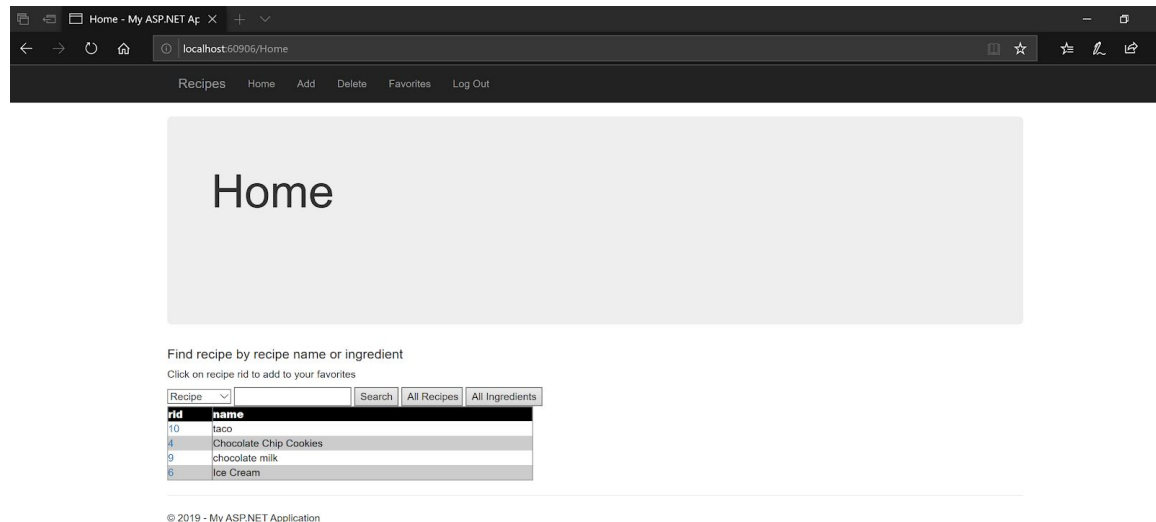
We built our frontend (interface) using ASP.NET, which is an open-source framework for building web applications. Below are screenshots of all of the pages in our frontend.

Figure 2: Login/Sign Up



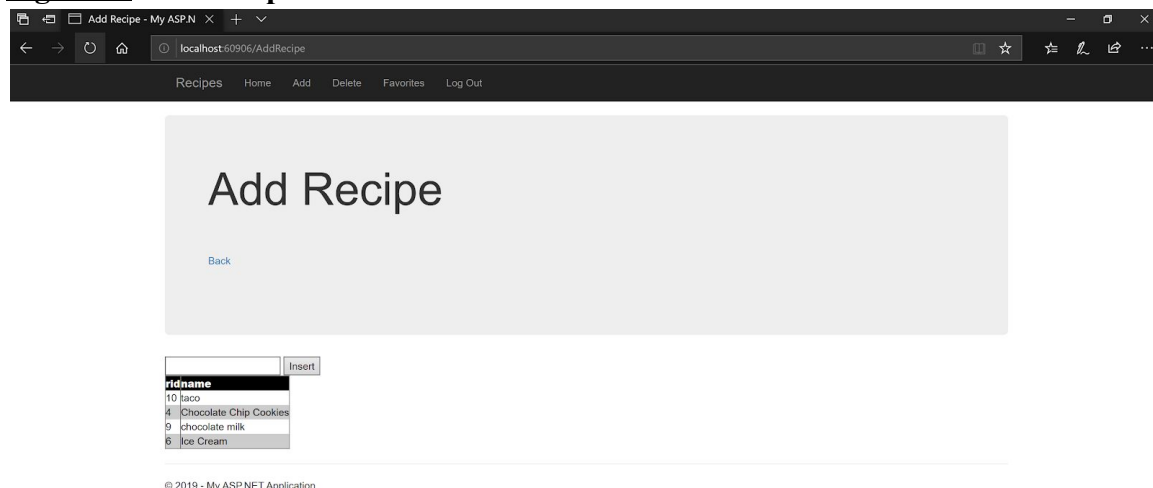
Upon accessing the system, the user is presented with the Login/Signup Screen. On this page, if already registered, the user must enter their Username and Password, then press Log In to log in. If the credentials are incorrect, or if they are nonexistent, the user is alerted with the error. If not registered, the user must enter their desired username and password, then press Sign Up in order to register. If the username is already taken, the user is alerted with the error.

Figure 3: Home



Upon logging in, the user is presented with the Home screen. On this page, the user is able to search our system for recipes. Utilizing the dropdown, the user is able to indicate whether or not they are searching for recipes by ingredient (which recipes contain the indicated ingredient) or by recipe name. They are also able to look at all recipes or ingredients in the database via clicking the respective button.

Figure 4: Add Recipe



Via this page the user can add a recipe to our database, by simply entering the name of the recipe they wish to add and pressing Insert.

Figure 5: Add Ingredient

localhost:60906/AddIngredient

Recipes Home Add Delete Favorites Log Out

Add Ingredient

[Back](#)

4 sugar
 27 flour
 28 spice
 30 milk
 31 salt
 32 chocolate
 33 chocolate

© 2019 - My ASP.NET Application

This page allows the user to add an ingredient to the database by typing the name of the ingredient and pressing Add Ingredient.

Figure 6: Add Ingredient to Recipe

localhost:60906/Add

Recipes Home Add Delete Favorites Log Out

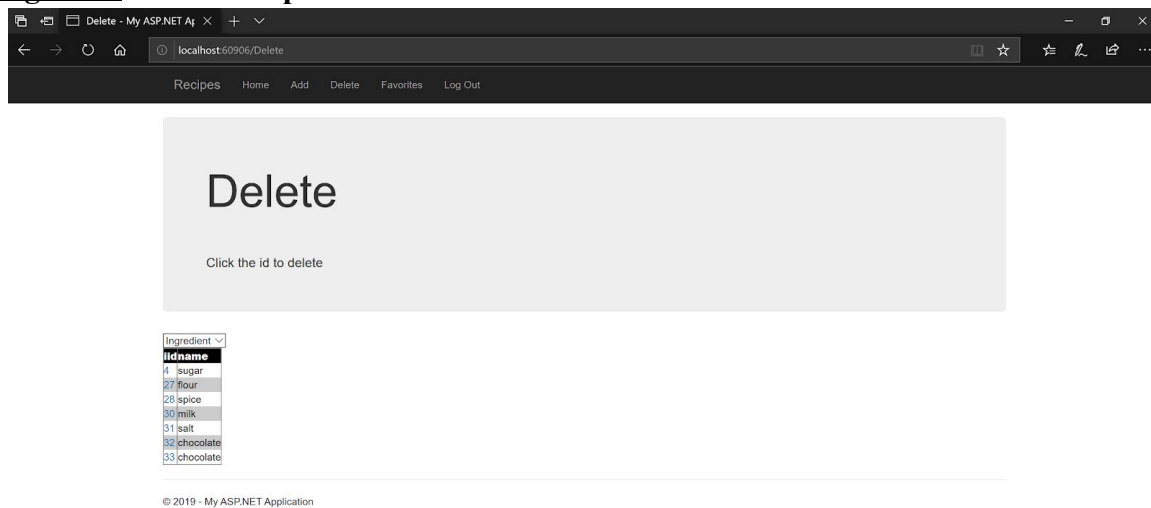
Add

Click rid to add ingredients to recipe

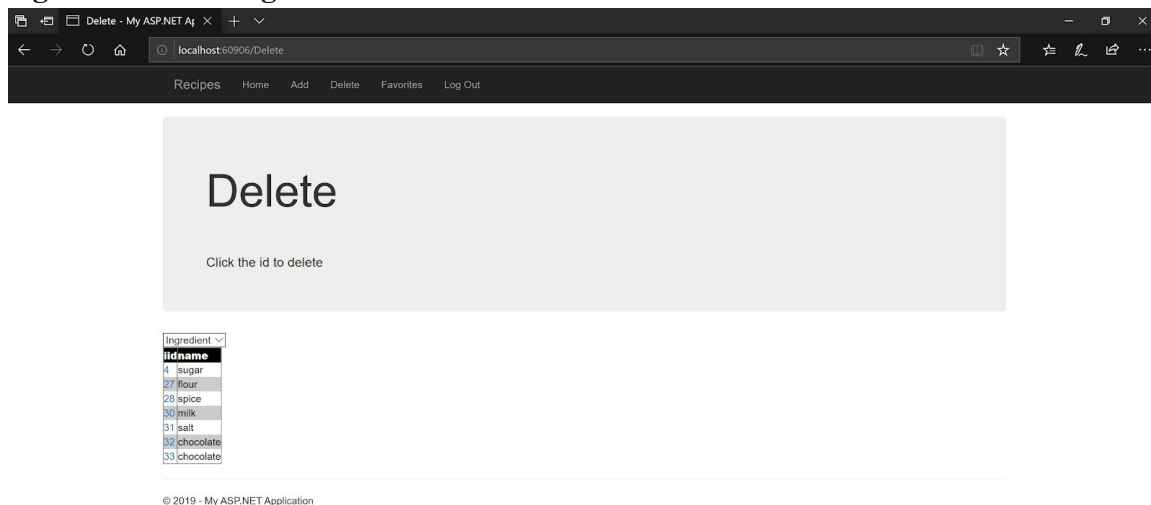
10 taco
 8 chocolate chip Cookies
 9 chocolate milk
 6 ice Cream

© 2019 - My ASP.NET Application

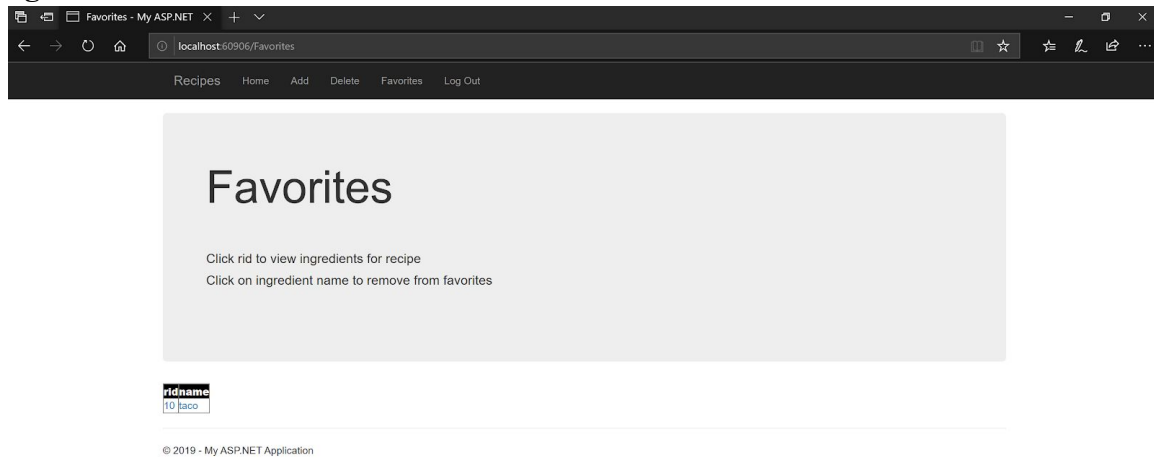
This page allows the admin to add an existing ingredient to an existing recipe on our service.

Figure 7: Delete Recipe

This page allows the admin to delete an ingredient by clicking on the iid of the ingredient.

Figure 8: Delete Ingredient

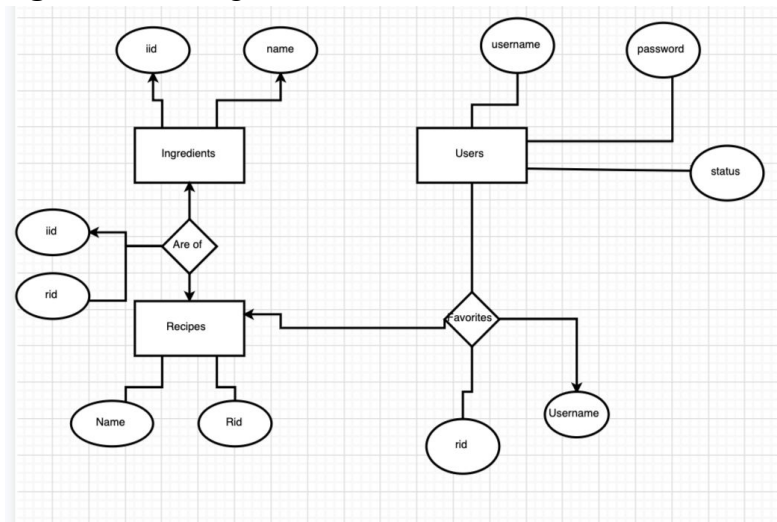
This page allows the admin to delete a recipe by clicking on the rid of the recipe they wish to delete.

Figure 9: Favorites

This page allows a user to view all of their favorites. By clicking on the rid of the recipe, they are able to remove the recipe from their favorites.

iii. Backend Design

Within this section we detail how the backend of the application was designed. A Restful API was built in Node.js in order for the frontend to make changes to the database, which was hosted on Heroku. The Database itself was built using Postgres. An ER Diagram of the Entity Relationships can be found below.

Figure 5: ER Diagram

Users

```
dbdesignfinal::DATABASE=> \d users;
      Table "public.users"
  Column |          Type          | Collation | Nullable | Default
-----+-----+-----+-----+-----
username | character varying(50) |           | not null |
password | character varying(50) |           |          |
status   | boolean                |           |          |
Indexes:
    "users_pkey" PRIMARY KEY, btree (username)
```

Users (password: Char (50), status: Boolean, Username: Char (50))

foreign keys: none

candidate key: username

primary key: username

not null: password, status, Username

This table holds the login information needed from the user. Usernames must be unique as they are primary keys.

Favorites

```
dbdesignfinal::DATABASE=> \d favorites;
      Table "public.favorites"
  Column |          Type          | Collation | Nullable | Default
-----+-----+-----+-----+-----
rid      | integer                |           |          |
username | character varying(50) |           |          |
```

Favorites (rid: Integer, username: Char(50))

foreign keys: username, rid

candidate key: (username, rid)

primary key: (username, rid)

not null: username, rid

This table holds the rid (recipe id) and the username of the user in order to represent which recipe a user has favorited. By having the username as well as the rid, we are able to retrieve all of a user's favorite recipes using simple queries.

Recipes

```
dbdesignfinal::DATABASE=> \d recipes
      Table "public.recipes"
  Column |          Type          | Collation | Nullable |          Default
-----+-----+-----+-----+-----
  rid    | integer                |           | not null | nextval('recipes_rid_seq'::regclass)
  name   | character varying(50) |           |          |
Indexes:
    "recipes_pkey" PRIMARY KEY, btree (rid)
```

Recipes (Name: Char(50) , rid: Integer)

foreign keys: none

candidate key: rid

primary key: rid

not null: rid, name

The Recipes table holds the names and rid's of each recipe. The rid's are generated sequentially, and the user has no input on what the rid of a recipe is. By doing this, we can ensure that they are all unique, so that multiple recipes of the same name can be generated.

Are of

```
dbdesignfinal::DATABASE=> \d are_of
      Table "public.are_of"
  Column |      Type      | Collation | Nullable |
-----+-----+-----+-----+
  iid    | integer        |           |          |
  rid    | integer        |           |          |
```

Are of (iid: Integer, rid: Integer)

foreign keys: iid, rid

candidate key: none

primary key: (iid, rid)

not null: (iid, rid)

The are_of table relates the ingredients table to the recipe table by using the two foreign keys (iid and rid).

Ingredients

```
dbdesignfinal::DATABASE=> \d ingredients
      Table "public.ingredients"
  Column |          Type          | Collation | Nullable |          Default
-----+-----+-----+-----+-----
  name   | character varying(50) |           |          |
  iid    | integer                |           | not null | nextval('ingredients_iid_seq'::regclass)
Indexes:
    "ingredients_pkey" PRIMARY KEY, btree (iid)
```

Ingredients (iid: Integer, name: Char(50))

foreign keys: none

candidate key: iid

primary key: iid

not null: iid

The ingredients table holds all of the ingredients by name and iid.

iv. SQL Queries

We built a RESTful API which is hosted on Heroku that our service interfaces with in order to make queries to our Postgres database, which is also hosted on Heroku. The API was built using Node.js and Express.js, and it uses our premade queries sort of like a Stored Procedure. The Client(Frontend) will make an HTTP Request to the API, usually with values generated by the user, and the server will determine which query should be used based off of the request.

Screenshots of each individual query can be found below.

Retrieving User Status

```
//User Status
app.get('/api/user/status/:username', (req, res) => {
  if (connected) {
    database.query(`select status from users where username = '${req.params.username}'`)
      .then((result) => {
        // You, a day ago * Getting user status promise fixed
        if (result.rows[0].status === true) {
          res.sendStatus(200);
        } else {
          res.sendStatus(400);
        }
      })
      .catch((err) => {
        res.send(err.stack);
      })
  }
})
```

Retrieving All Recipes

```
//Retrieve all recipes
app.get('/api/recipes/', (req, res) => {
  if (connected) {
    database.query(`select distinct * from recipes;`)
      .then((result) => {
        res.statusCode = 200;
        res.json(result.rows);
      })
  }
})
```

Add Recipe

```
//Add Recipe
app.post('/api/recipes/add/:name/', (req, res) => {
  if (connected) {
    let { name } = req.params;
    database.query(`insert into recipes(name) values('${name}');`)
      .then(() => {
        res.sendStatus(200);
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack);
      })
  }
})
})
```

Add Ingredient to A Recipe

```
//Add Ingredient To Recipe by IID
app.post('/api/recipe/add/ingredientbyiid/:rid/:iid/', (req, res) => {
  if (connected) {
    let { rid, iid } = req.params;
    database.query(`insert into are_of(rid, iid) values(${rid}, ${iid});`)
      .then(() => {
        res.sendStatus(200);
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack);
      })
  }
})
})
```

Delete a Recipe

```
//Delete a recipe
app.delete('/api/recipes/delete/:rid', (req, res) => {
  if (connected) {
    let rid = req.params.rid;
    database.query(`delete from recipes where rid=${rid}`)
      .then(() => {
        database.query(`delete from favorites where rid=${rid}`)
          .then(() => {
            database.query(`delete from are_of where rid=${rid}`)
              .then(() => {
                res.statusCode = 200;
                res.send();
              })
            })
          })
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack);
      })
  }
})
})
```

Add An Ingredient

```
//Add an Ingredient
app.post('/api/ingredients/:name', (req, res) => {
  if (connected) {
    let name = req.params.name;
    console.log(name);
    database.query(`insert into ingredients(name) values('${name}');`)
      .then(() => {
        res.sendStatus(200);
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack);
      })
  }
})
```

Delete an Ingredient

```
//Delete an ingredient
app.delete('/api/ingredients/:iid', (req, res) => {
  if (connected) {
    let { iid } = req.params;
    database.query(`delete from ingredients where iid = ${iid};`)
      .then(() => {
        database.query(`delete from are_of where iid = ${iid};`)
          .then(() => {
            res.status(200);
            res.json();
          })
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack);
      })
  }
})
```

All Ingredients

```
//All Ingredients
app.get('/api/ingredients/', (req, res) => {
  if (connected) {
    database.query(`select distinct iid, name from ingredients`)
      .then((results) => {
        res.status(200);
        res.json(results.rows);
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack);
      })
  }
})
```

All Ingredients in a Recipe

```
app.get('/api/ingredients/byrid/:rid', (req, res) => {
  if (connected) {
    let {rid} = req.params;

    database.query(`select distinct i.iid, i.name from are_of a, ingredients i where a.rid = ${rid} and a.iid = i.iid;`)
      .then((results) => {
        res.status(200);
        res.send(results.rows);
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack);
      })
  }
})
```

Recipe by Ingredient Name

```
//Recipes by Ingredients
app.get('/api/recipes/byingredient/:name', (req, res) => {
  if (connected) {
    let name = req.params.name;
    console.log(name);
    if (typeof(name) !== undefined) {
      database.query(`select r.name, r.rid from recipes r, are_of a, ingredients i where a.iid = i.iid and i.name = '${name}' and r.rid = a.rid;`)
        .then((results) => {
          res.status(200);
          res.json(results.rows);
        })
        .catch((err) => {
          res.status(404);
          res.send(err.stack);
        })
    }
  }
})
```

Recipe by Name


```
//Get Recipes by name
app.get('/api/recipes/byname/:name', (req, res) => {
  if (connected) {
    database.query(`select * from recipes where name = '${req.params.name}'`)
      .then((results) => {
        res.status(200);
        res.send(results.rows);
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack)
      })
  }
})
})
```

Recipe by RID

```
app.get('/api/recipes/byrid/:rid', (req, res) => {
  if (connected) {
    database.query(`select * from recipes where rid = '${req.params.rid}'`)
      .then((results) => {
        res.status(200);
        res.send(results.rows);
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack)
      })
  }
})
})
```

Sign up a User

```
//Sign up a user
app.post('/api/users/signup/:username/:password/:status', (req, res) => {
  if (connected) {
    let { username, password, status } = req.params;
    database.query(`select * from users where username = '${username}'`)
      .then((results) => {
        if (results.rows.length == 0) {
          database.query(`insert into users(username, password, status) values ('${username}', '${password}', '${status}');`)
            .then(() => {
              res.sendStatus(200);
            })
        } else {
          console.log("User already exists");
          res.send(404);
        }
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack);
      })
  }
})
})
```

Log In A User


```
//Log in a user
app.get('/api/users/login/:username/:password/', (req, res) => {
  if (connected) {
    let { username, password } = req.params;
    database.query(`select status from users where username = '${username}' and password = '${password}';`)
      .then((results) => {
        if (results.rows.length) {
          res.status(200);
          res.send(results.rows.status);
        } else {
          res.sendStatus(404);
        }
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack);
      })
  }
})
})
```

All Favorites for a User

```
//Getting all favorites for a user
app.get('/api/users/favorites/:username/', (req, res) => {
  if (connected) {
    let { username } = req.params;
    //No check necessary for if username exists, as this call should only be made when the user is logged in
    database.query(`select * from favorites where username = '${username}';`)
      .then((results) => {
        res.status(200);
        res.send(results.rows);
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack);
      })
  }
})
})
```

Adding a Favorite

```
//Adding Favorite
app.post('/api/favorites/add/:username/:rid/', (req, res) => {
  if (connected) {
    let { username, rid } = req.params;
    //TODO0: Make sure that a user can't favorite something twice
    database.query(`insert into favorites(username, rid) values ('${username}', ${rid});`)
      .then(() => {
        res.sendStatus(200);
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack);
      })
  }
})
})
```

Delete a Favorite

```
//Delete Favorite
app.delete('/api/favorites/delete/:username/:rid/', (req, res) => {
  if (connected) {
    let { username, rid } = req.params;
    database.query(`delete from favorites where username = '${username}' and rid = ${rid};`)
      .then(() => {
        res.sendStatus(200);
      })
      .catch((err) => {
        res.status(404);
        res.send(err.stack);
      })
  }
})
```