Project Report:
Recipedia

CS157A - Team 36
Timothy To, Yiyi Zhang, Sean Zurcher

<u>PROJECT REQUIREMENTS</u>

Project Overview

- ***Goal, motivation, stakeholders, application domain, benefits to users***

Our project is an implementation of a recipe book for casual home use. It allows users to interact with it in much the same way one would interact with a physical cookbook, mainly by looking up recipes that have been stored in a database. Users are able to search for specific recipes by directly searching by name and by filtering them based on whether they are also vegan and/or vegetarian.

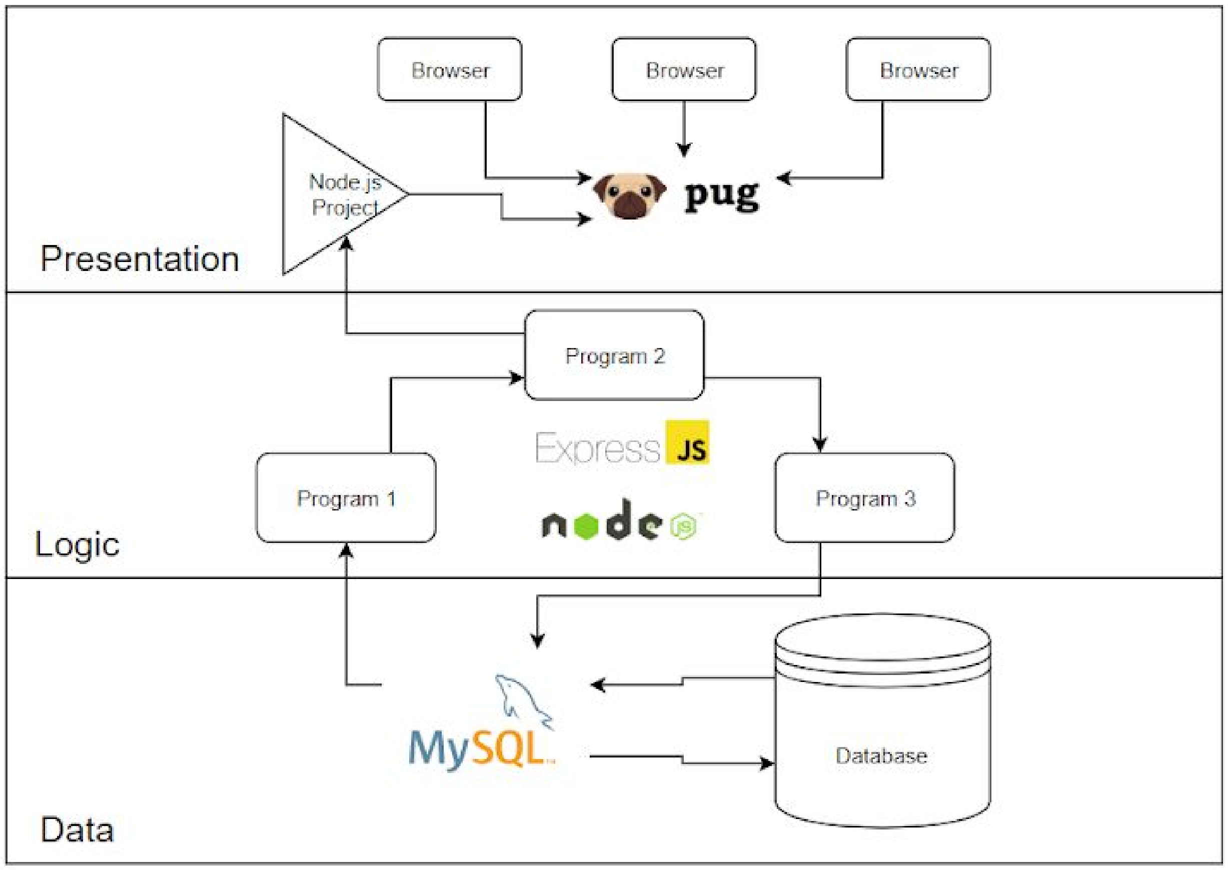
Additionally, users can expand the cookbook by adding their own recipes to the database. Users are able to favorite recipes, which appear on their own separate tab for themselves. Users are also able to rate recipes and on the front page the highest rated recipes should show up. Users are able to register accounts using an email, username and password, and can use this account to write their own recipes and save their favorite recipes.

Lastly, certain users are granted administrative privileges. Users may come across recipes that contain inappropriate content, and may report them for review; users with administrative privileges can later decide to edit these recipes, delete them from the database altogether, or ignore them outright.

Stakeholders in our project include anyone who is interested in cooking either in learning cooking with simple to use recipes or simply veteran with too many recipes to keep track of. This project is an easy tool for home cooks to keep track of their existing recipes, find new popular recipes, and reference them in the future. Our project acts as an entry point for home cooks to broaden their knowledge and to find new and exciting foods to explore.

System Environment

- ***3-tier architecture structure, hardware used, software used, RDBMS version used, application languages***
    - Hardware: We developed Recipedia using Windows 10 machines.
    - Software: Any web browser client should be able to access the front end. We'll be using Apache Tomcat for the server that hosts the servlets and MySQL for the database management system.
    - RDBMS used: We built our database using MySQL, as stipulated by the assignment requirements. While our installations of MySQL were not uniform, we implemented the project using v. 8.0.16 and 8.0.18.
    - Programming/Application languages used: We implemented our backend using Node.js, a dialect of JavaScript, as well as Express.js for designing our application framework. For our front end/presentation level, we used Pug, a templating framework for Node.js that compiles to HTML and makes use of CSS. The server itself is created using Node's built-in server functionalities.



Functional Requirements

- Describe users and how users access the system.

  Users will be able to search for recipes by name, as well as by whether they are vegan or vegetarian. According to user input, the system selects the appropriate recipes from database, and lists them on the web page. Also, users can create their own account through entering their personal information, such as name and email, to access the recipe book. By logging/logging out of their account, they can access more elaborated functionalities like saving their favorite recipes, creating/deleting their own specific recipes, and rating the recipes. These are also recorded in the database.

- Describe each functionality/features, functional processes, and I/O(s).
    - ➔ Register: Users can register by entering their username, password, and email to access the recipe book. This information is obtained through the registration page, and is kept in the database.
    - ➔ Login: The users who have registered may login to access the bulk of the project's functionalities.
    - ➔ Search Recipe: Users are able to search for a recipe by filtering them based on recipe name, as well as by whether the recipes are vegan or vegetarian. Users may search names by entering any individual word contained in the recipe name.
    - ➔ Rate Recipe: Users can rate recipe according to their own tastes. The rating is a user input with a possible score out of five, and the average rating is displayed on the recipe page.
    - ➔ View Recipe: Users may see the recipes posted by other users. They can view the full details of a recipe simply by clicking on it on the webpage.
    - ➔ Create Recipe: Users may add their own specific recipes. The creation of recipe includes entering title, ingredients, preparation time, category and description, instructions and other information. The button used to create a new recipe is easily accessible from the toolbar.
    - ➔ Edit Recipe: Users may edit existing recipes if they wrote them, or if they are an admin with such privileges.

- ➔ Delete Recipe: Users can also delete the recipes if they wrote them, or if they are an admin with such privileges.
- ➔ Add Favorite: Users can mark the recipes as their favorite. There is an easily accessible button on each recipe page that adds the recipe to the user's list of favorites.
- ➔ View Favorites: Users may view all recipes they created/marked as favorite. It should display all the recipes in a list and users should be able to both unfavorite recipes from the list and open up the recipe for viewing.
- ➔ Report Recipe: If a user encounters a recipe that contains inappropriate content, they can flag the recipe to be dealt with by an admin.
- ➔ Resolve Report: Admin can edit, delete, or ignore reported recipes.

Nonfunctional Issues

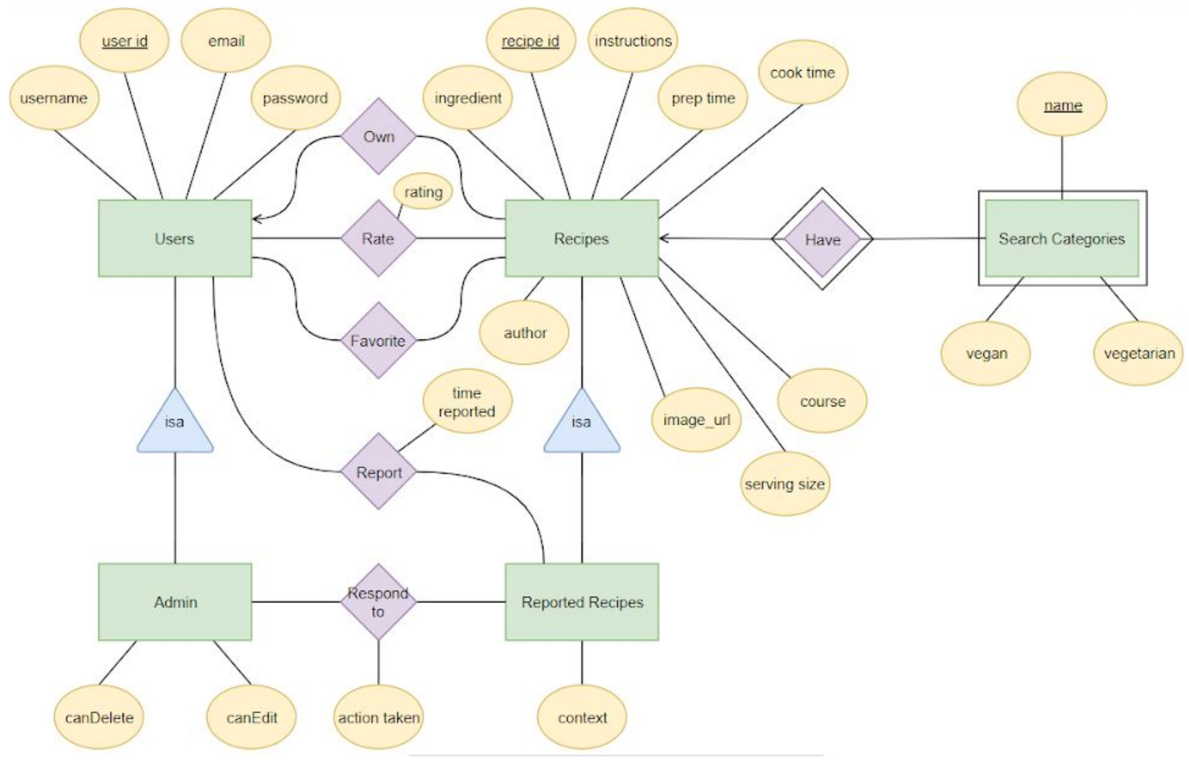
- GUI done using Pug (HTML) and CSS.
    - The application will be easy to use, and has a user-friendly interface which is simple and clear. The interface contains clear navigation with buttons and plain view part helps users understand and find the recipe they want. Users will accomplish their goal easily and quickly with few or no user errors.
    - Our GUI is implemented with Pug and styled with CSS. Pug, in cooperation with our Node.js and Express.js backend, displays and organizes the data to display to the client.

    - Upon launching Recipedia, there will be a login and registration page. Users may log in to access the web application, or register a new account.
    - Once logged in, the homepage of Recipedia will display recipes from the database. This homepage also allows users to view their favorite recipes and their own recipes.
    - By entering a query into the search bar, users will see recipes that have been added into the database, ordered by whichever filter restraints they add to the search.
    - Upon choosing a recipe, the recipe name, serving size, cooking and preparation time, ingredient list, and instructions will be displayed to the user. The recipe can also be rated and favorited on this page.
    - When writing new recipes, users will have fields to enter the recipe name, serving size, cooking and preparation time, ingredient list, and instructions.
- Security
    - Users may not edit other users' recipes unless they have the administrative privileges to do so.
    - Password security, SQL injection attacks, and other cybersecurity attacks were considered out of the scope of this project by Dr. Wu, and were not included in this project.
- Access Control

- Users may create and edit their own recipes, and editing rights are given only to the accounts that created them and users with appropriate administrative privileges..
- Users may rate and favorite recipes, and changing these attributes is restricted to each individual user account.

PROJECT DESIGN

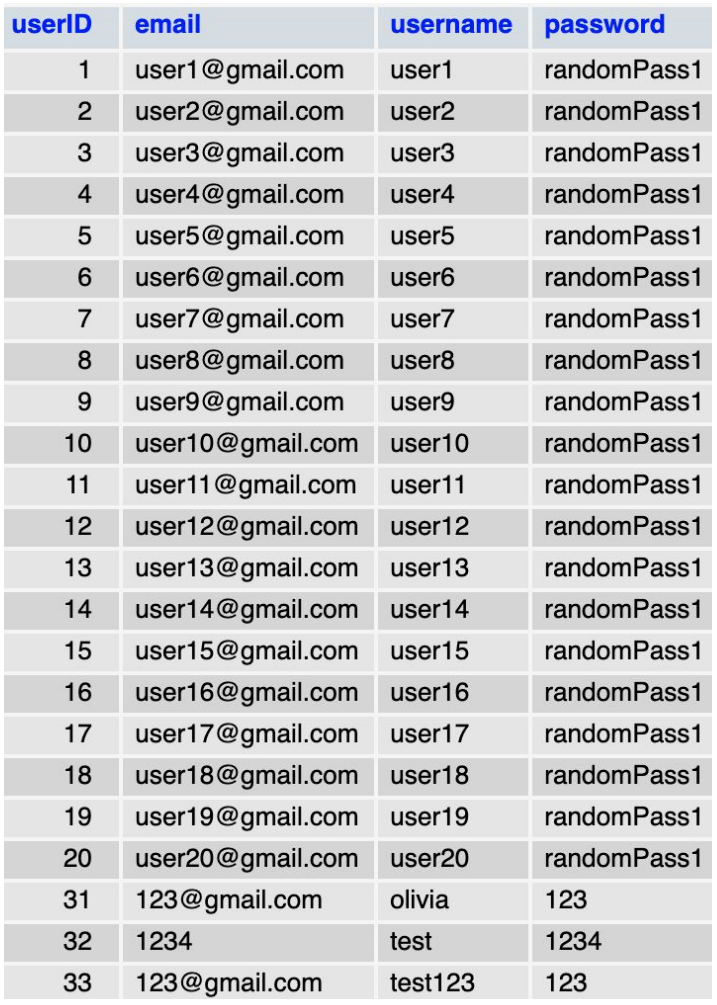
ER Model:



Explanation:

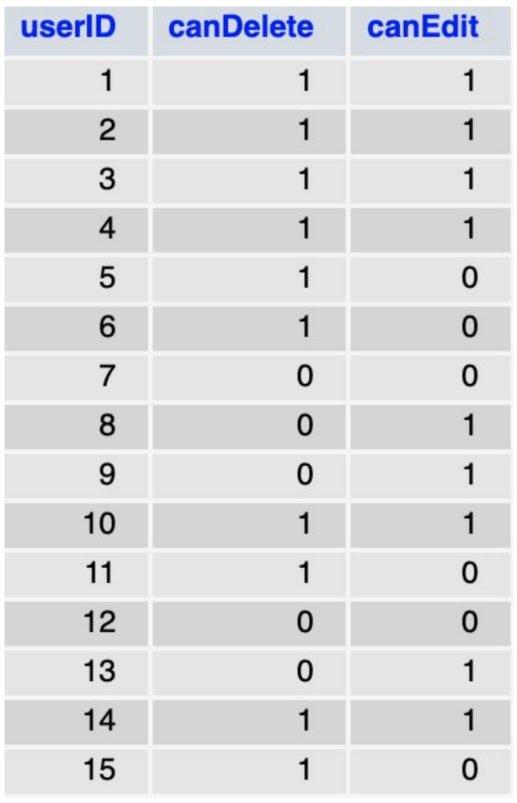
Entity Sets:

- Users
  - Represents visitors to Recipedia that have created their own user accounts.
  - Differentiated by their user id. Attributes include username, password, and email.
  - Schema: Users(user id, username, password, email)



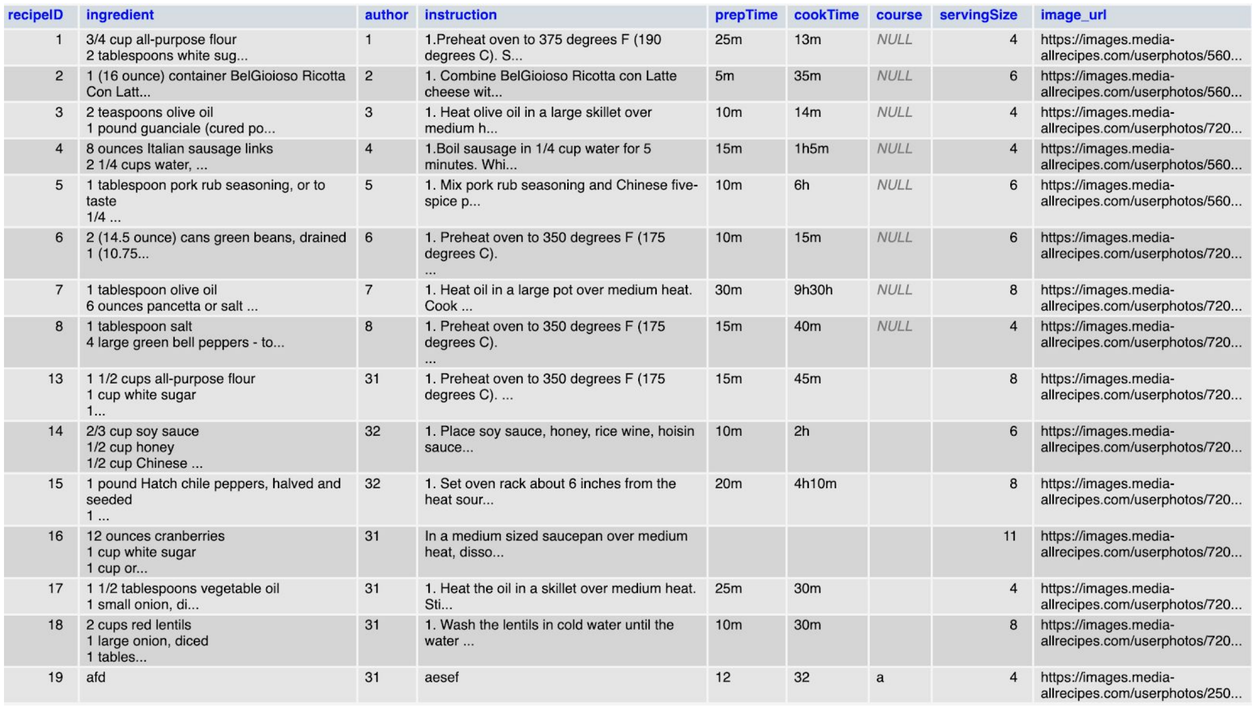
userID	email	username	password
1	user1@gmail.com	user1	randomPass1
2	user2@gmail.com	user2	randomPass1
3	user3@gmail.com	user3	randomPass1
4	user4@gmail.com	user4	randomPass1
5	user5@gmail.com	user5	randomPass1
6	user6@gmail.com	user6	randomPass1
7	user7@gmail.com	user7	randomPass1
8	user8@gmail.com	user8	randomPass1
9	user9@gmail.com	user9	randomPass1
10	user10@gmail.com	user10	randomPass1
11	user11@gmail.com	user11	randomPass1
12	user12@gmail.com	user12	randomPass1
13	user13@gmail.com	user13	randomPass1
14	user14@gmail.com	user14	randomPass1
15	user15@gmail.com	user15	randomPass1
16	user16@gmail.com	user16	randomPass1
17	user17@gmail.com	user17	randomPass1
18	user18@gmail.com	user18	randomPass1
19	user19@gmail.com	user19	randomPass1
20	user20@gmail.com	user20	randomPass1
31	123@gmail.com	olivia	123
32	1234	test	1234
33	123@gmail.com	test123	123

- Admin
  - A subclass of Users. Represents users who have administrative privileges.
  - Primarily responsible for handling Reported Recipes by allowing or striking them.
  - Schema: Admin(<u>user id</u>, canDelete, canEdit)



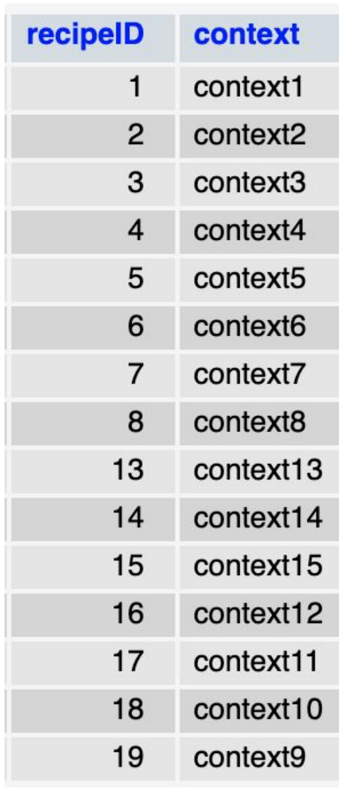
userID	canDelete	canEdit
1	1	1
2	1	1
3	1	1
4	1	1
5	1	0
6	1	0
7	0	0
8	0	1
9	0	1
10	1	1
11	1	0
12	0	0
13	0	1
14	1	1
15	1	0

- Recipes
  - Represents recipes that have been uploaded to Recipedia.
  - Differentiated by their recipe id. Attributes include ingredients, instructions, serving size, prep time, cook time, course (main, side, dessert, etc), and cumulative rating.
  - Schema: Recipes(recipe id, ingredients, instructions, serving size, prep time, cook time, course, rating, author, image_url)



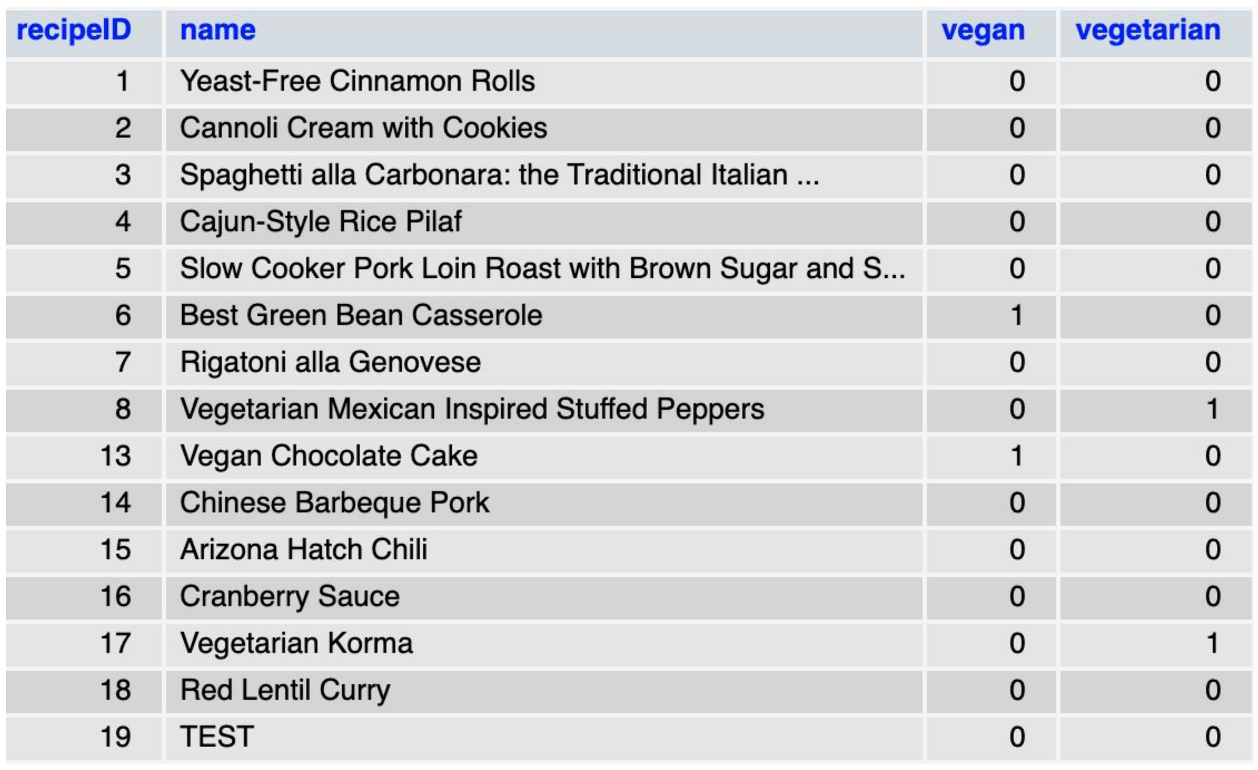
recipeID	ingredient	author	instruction	prepTime	cookTime	course	servingSize	image_url
1	3/4 cup all-purpose flour 2 tablespoons white sug...	1	1.Preheat oven to 375 degrees F (190 degrees C). S...	25m	13m	NULL	4	https://images.media-allrecipes.com/userphotos/560...
2	1 (16 ounce) container BelGioioso Ricotta Con Latt...	2	1. Combine BelGioioso Ricotta con Latte cheese wit...	5m	35m	NULL	6	https://images.media-allrecipes.com/userphotos/560...
3	2 teaspoons olive oil 1 pound guanciale (cured po...	3	1. Heat olive oil in a large skillet over medium h...	10m	14m	NULL	4	https://images.media-allrecipes.com/userphotos/720...
4	8 ounces Italian sausage links 2 1/4 cups water, ...	4	1.Boil sausage in 1/4 cup water for 5 minutes. Whi...	15m	1h5m	NULL	4	https://images.media-allrecipes.com/userphotos/560...
5	1 tablespoon pork rub seasoning, or to taste 1/4 ...	5	1. Mix pork rub seasoning and Chinese five-spice p...	10m	6h	NULL	6	https://images.media-allrecipes.com/userphotos/560...
6	2 (14.5 ounce) cans green beans, drained 1 (10.75...	6	1. Preheat oven to 350 degrees F (175 degrees C). ...	10m	15m	NULL	6	https://images.media-allrecipes.com/userphotos/720...
7	1 tablespoon olive oil 6 ounces pancetta or salt ...	7	1. Heat oil in a large pot over medium heat. Cook ...	30m	9h30h	NULL	8	https://images.media-allrecipes.com/userphotos/720...
8	1 tablespoon salt 4 large green bell peppers - to...	8	1. Preheat oven to 350 degrees F (175 degrees C). ...	15m	40m	NULL	4	https://images.media-allrecipes.com/userphotos/720...
13	1 1/2 cups all-purpose flour 1 cup white sugar 1...	31	1. Preheat oven to 350 degrees F (175 degrees C). ...	15m	45m		8	https://images.media-allrecipes.com/userphotos/720...
14	2/3 cup soy sauce 1/2 cup honey 1/2 cup Chinese ...	32	1. Place soy sauce, honey, rice wine, hoisin sauce...	10m	2h		6	https://images.media-allrecipes.com/userphotos/720...
15	1 pound Hatch chile peppers, halved and seeded 1 ...	32	1. Set oven rack about 6 inches from the heat sour...	20m	4h10m		8	https://images.media-allrecipes.com/userphotos/720...
16	12 ounces cranberries 1 cup white sugar 1 cup or...	31	In a medium sized saucepan over medium heat, disso...				11	https://images.media-allrecipes.com/userphotos/720...
17	1 1/2 tablespoons vegetable oil 1 small onion, di...	31	1. Heat the oil in a skillet over medium heat. Sti...	25m	30m		4	https://images.media-allrecipes.com/userphotos/720...
18	2 cups red lentils 1 large onion, diced 1 tables...	31	1. Wash the lentils in cold water until the water ...	10m	30m		8	https://images.media-allrecipes.com/userphotos/720...
19	afd	31	aesef	12	32	a	4	https://images.media-allrecipes.com/userphotos/250...

- Reported Recipes
    - A subclass of Recipes. Represents recipes that have some sort of inappropriate content in them, and have been flagged for review by an Admin.
    - The time of reporting is documented, as well as the context of the report, detailing why the recipe was reported in the first place.
    - Schema: ReportedRecipes(<u>recipe id</u>, context)



recipeID	context
1	context1
2	context2
3	context3
4	context4
5	context5
6	context6
7	context7
8	context8
13	context13
14	context14
15	context15
16	context12
17	context11
18	context10
19	context9

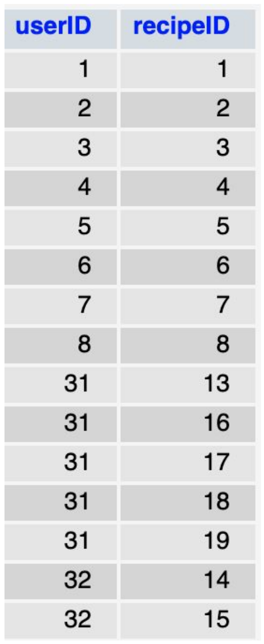
- Search Categories
    - Represents search filters that can be used to narrow down Recipes.
    - Differentiated by their name, but inherits recipe id from Recipes as a primary key.
    - Could potentially be expanded to include additional parameters (keto, paleo, etc)
    - Schema: SearchCategories(recipe id, name, vegan, vegetarian)



recipeID	name	vegan	vegetarian
1	Yeast-Free Cinnamon Rolls	0	0
2	Cannoli Cream with Cookies	0	0
3	Spaghetti alla Carbonara: the Traditional Italian ...	0	0
4	Cajun-Style Rice Pilaf	0	0
5	Slow Cooker Pork Loin Roast with Brown Sugar and S...	0	0
6	Best Green Bean Casserole	1	0
7	Rigatoni alla Genovese	0	0
8	Vegetarian Mexican Inspired Stuffed Peppers	0	1
13	Vegan Chocolate Cake	1	0
14	Chinese Barbeque Pork	0	0
15	Arizona Hatch Chili	0	0
16	Cranberry Sauce	0	0
17	Vegetarian Korma	0	1
18	Red Lentil Curry	0	0
19	TEST	0	0

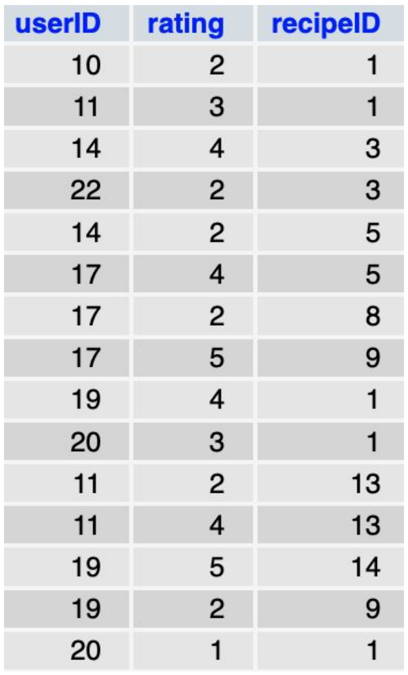
Relationships:

- Own
    - Users may create new Recipes. Once created, they can be managed by their respective Users. Each Recipe can be owned by one User.
    - Schema: Own(user id, recipe id)



userID	recipeID
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
31	13
31	16
31	17
31	18
31	19
32	14
32	15

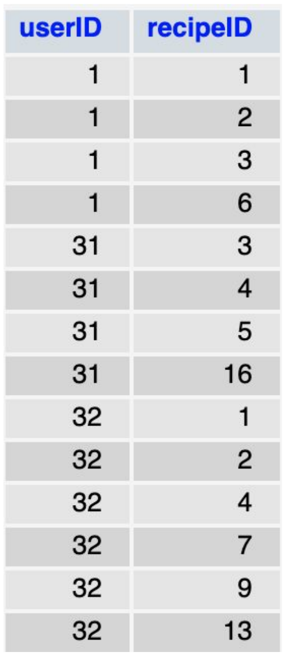
- Rate
    - Users can rate Recipes. Cumulative rating is stored as an attribute of Recipes.
    - Schema: Rate(user id, recipe id, rating)



userID	rating	recipeID
10	2	1
11	3	1
14	4	3
22	2	3
14	2	5
17	4	5
17	2	8
17	5	9
19	4	1
20	3	1
11	2	13
11	4	13
19	5	14
19	2	9
20	1	1

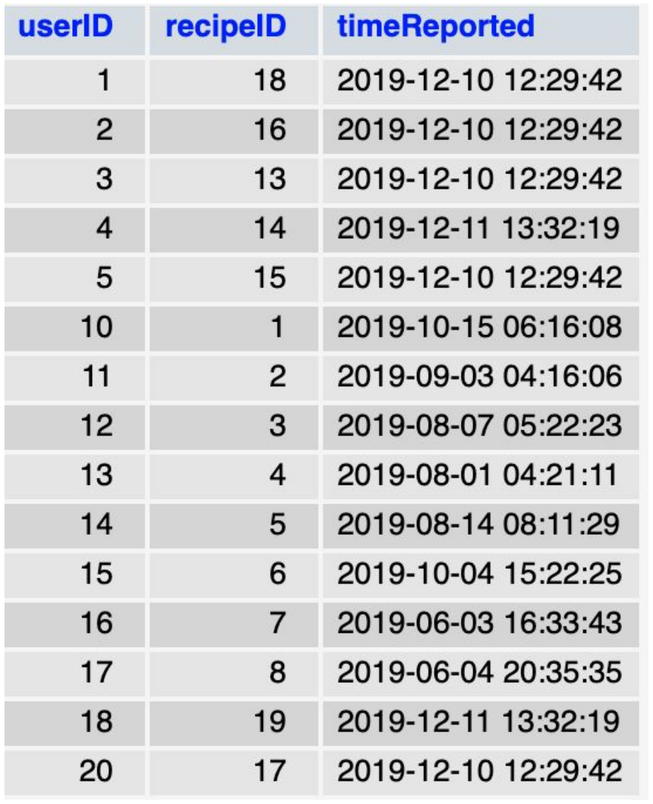
- Favorite

- Users can pick favorite Recipes to save for later.
- Schema: Favorite(<u>user id</u>, <u>recipe id</u>)



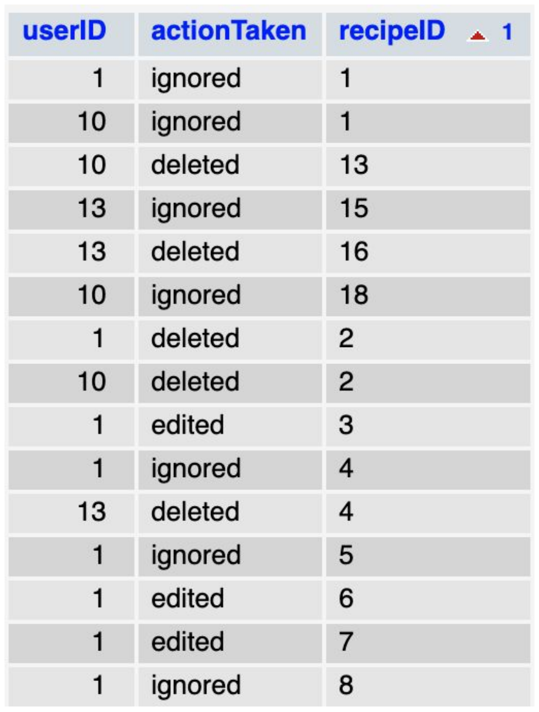
userID	recipeID
1	1
1	2
1	3
1	6
31	3
31	4
31	5
31	16
32	1
32	2
32	4
32	7
32	9
32	13

- Report
  - Users can report Recipes if they find they have inappropriate content in them.
  - Documents the time the report was submitted
  - Schema: Report(<u>user id</u>, <u>recipe id</u>, time reported)



userID	recipeID	timeReported
1	18	2019-12-10 12:29:42
2	16	2019-12-10 12:29:42
3	13	2019-12-10 12:29:42
4	14	2019-12-11 13:32:19
5	15	2019-12-10 12:29:42
10	1	2019-10-15 06:16:08
11	2	2019-09-03 04:16:06
12	3	2019-08-07 05:22:23
13	4	2019-08-01 04:21:11
14	5	2019-08-14 08:11:29
15	6	2019-10-04 15:22:25
16	7	2019-06-03 16:33:43
17	8	2019-06-04 20:35:35
18	19	2019-12-11 13:32:19
20	17	2019-12-10 12:29:42

- Respond to
    - Admin can respond to Reported Recipes to review them and decide whether they should be left alone or taken down.
    - Schema: RespondTo(user id, report id, action taken)



userID	actionTaken	recipeID ▲ 1
1	ignored	1
10	ignored	1
10	deleted	13
13	ignored	15
13	deleted	16
10	ignored	18
1	deleted	2
10	deleted	2
1	edited	3
1	ignored	4
13	deleted	4
1	ignored	5
1	edited	6
1	edited	7
1	ignored	8

- Have
    - Recipes have Search Categories. Visitors to Recipedia can search the database for particular Recipes, and can narrow their search by providing Search Categories that match those that are attached to existing Recipes.
    - Because this is a supporting relationship for the weak entity set Search Categories, this relationship does not have a table, and is represented in the Search Categories table.

Normalization

<u>Functional Dependencies</u>:

- Users: user_id → {username, password, email}

- Recipes: recipe_id → {ingredient, author, instruction, prepTime, cookTime, course, servingSize, image_url}
- searchCategories: recipe_id → {name, vegan, vegetarian}
- Admin: user_id → {canDelete, canEdit}
- reportedRecipes: recipe_id → {context}
- Own: recipe_id → {user_id}
- Rate: user_id, recipe_id → {rating}
- Favorite: N/A

<u>Conversion to BCNF</u>

Each of the functional dependencies listed above is already in BCNF. The left side of each FD is a superkey for the relation it corresponds to.

IMPLEMENTATION

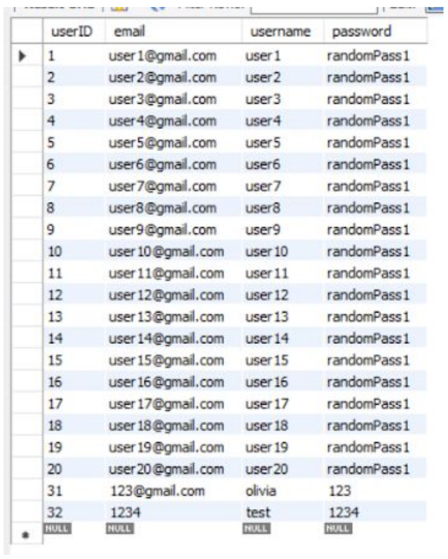
Overview

Our approach to Recipedia relies on the central index.js file. Written in Node.js and routed using Express.js, it is the middleware that handles all communication between user input and the database, as well as all the data manipulation required to present data to the front end. The front end is constructed using Pug templating and CSS, and different Pug files are reached by routing through the Express.js framework. This implementation section will describe the project in order of our functional requirements, while referencing both the source code and the web application itself.

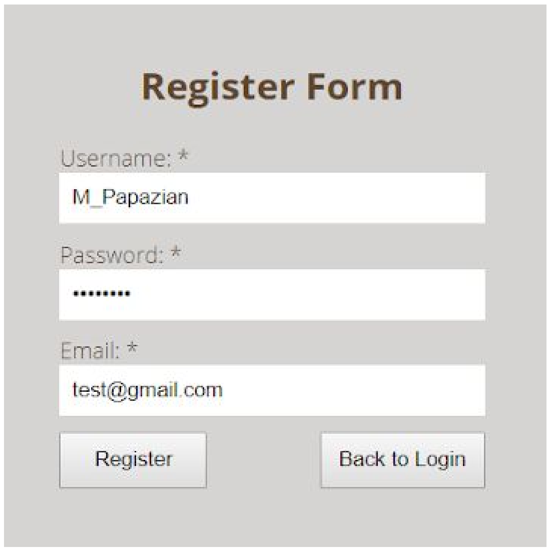
User Registration

In index.js, registration is handled by both app.get('/register') and app.post('/register'), and is displayed using register.pug. Users can enter data into the fields provided. The backend validates their input by making sure their username doesn't contain any special characters, then references the users table to see if the inputted username matches any existing entries in the table with the same username attribute. If not, then the backend inserts the set of new data into the users table.

Before:



userID	email	username	password
1	user1@gmail.com	user1	randomPass1
2	user2@gmail.com	user2	randomPass1
3	user3@gmail.com	user3	randomPass1
4	user4@gmail.com	user4	randomPass1
5	user5@gmail.com	user5	randomPass1
6	user6@gmail.com	user6	randomPass1
7	user7@gmail.com	user7	randomPass1
8	user8@gmail.com	user8	randomPass1
9	user9@gmail.com	user9	randomPass1
10	user10@gmail.com	user10	randomPass1
11	user11@gmail.com	user11	randomPass1
12	user12@gmail.com	user12	randomPass1
13	user13@gmail.com	user13	randomPass1
14	user14@gmail.com	user14	randomPass1
15	user15@gmail.com	user15	randomPass1
16	user16@gmail.com	user16	randomPass1
17	user17@gmail.com	user17	randomPass1
18	user18@gmail.com	user18	randomPass1
19	user19@gmail.com	user19	randomPass1
20	user20@gmail.com	user20	randomPass1
31	123@gmail.com	olivia	123
32	1234	test	1234
NULL	NULL	NULL	NULL



Register Form

Username: *

M_Papazian

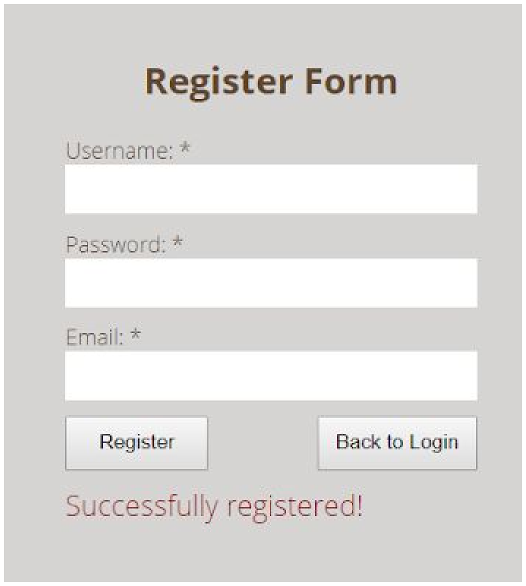
Password: *

••••••••

Email: *

test@gmail.com

After:



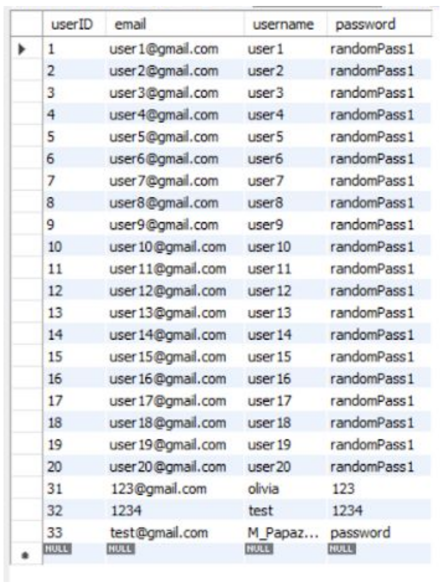
Register Form

Username: *

Password: *

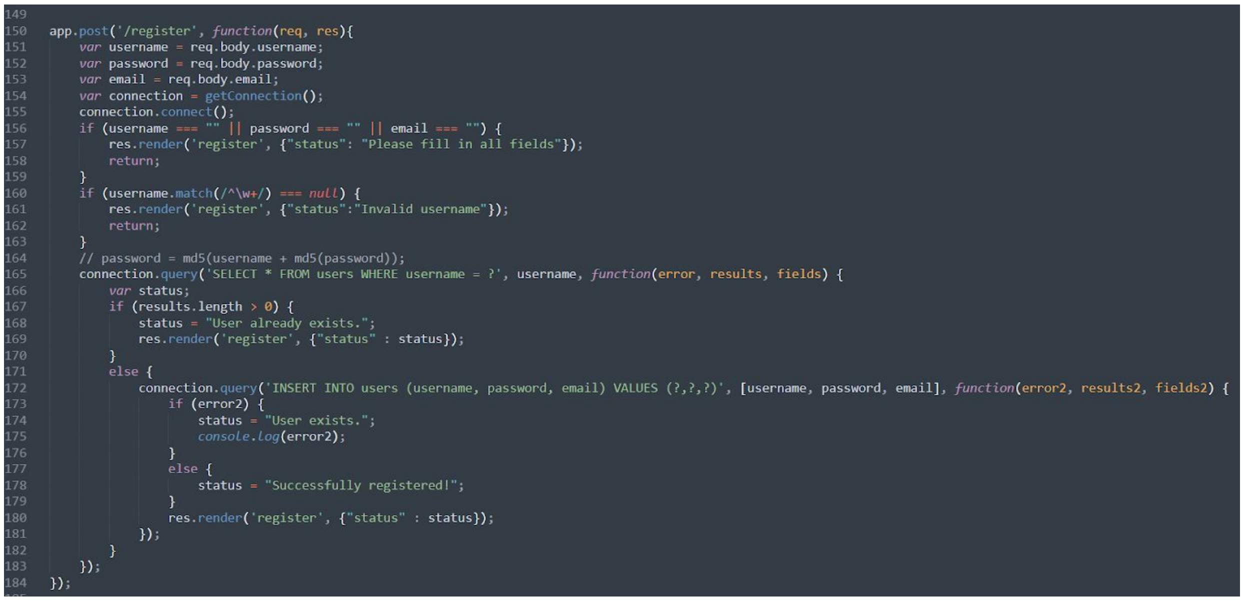
Email: *

Successfully registered!

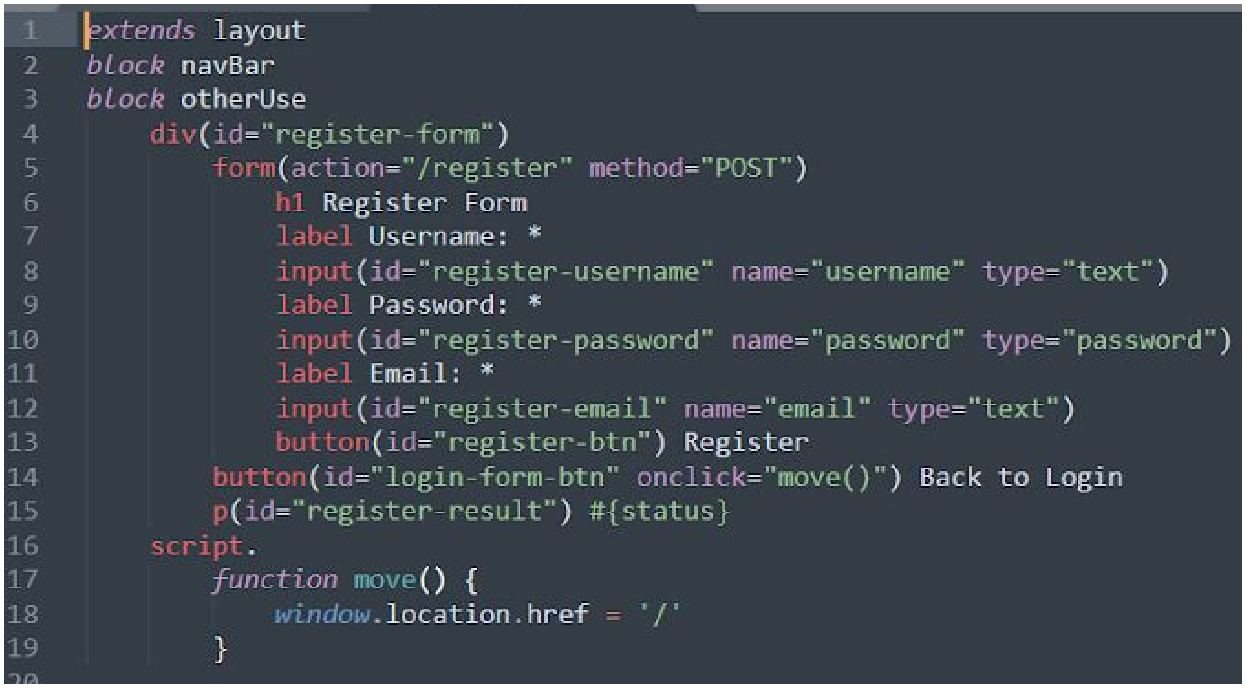


userID	email	username	password
1	user1@gmail.com	user1	randomPass1
2	user2@gmail.com	user2	randomPass1
3	user3@gmail.com	user3	randomPass1
4	user4@gmail.com	user4	randomPass1
5	user5@gmail.com	user5	randomPass1
6	user6@gmail.com	user6	randomPass1
7	user7@gmail.com	user7	randomPass1
8	user8@gmail.com	user8	randomPass1
9	user9@gmail.com	user9	randomPass1
10	user10@gmail.com	user10	randomPass1
11	user11@gmail.com	user11	randomPass1
12	user12@gmail.com	user12	randomPass1
13	user13@gmail.com	user13	randomPass1
14	user14@gmail.com	user14	randomPass1
15	user15@gmail.com	user15	randomPass1
16	user16@gmail.com	user16	randomPass1
17	user17@gmail.com	user17	randomPass1
18	user18@gmail.com	user18	randomPass1
19	user19@gmail.com	user19	randomPass1
20	user20@gmail.com	user20	randomPass1
31	123@gmail.com	olivia	123
32	1234	test	1234
33	test@gmail.com	M_Papaz...	password
NULL	NULL	NULL	NULL

Source:



```
app.post('/register', function(req, res){
    var username = req.body.username;
    var password = req.body.password;
    var email = req.body.email;
    var connection = getConnection();
    connection.connect();
    if (username === "" || password === "" || email === "") {
        res.render('register', {"status": "Please fill in all fields"});
        return;
    }
    if (username.match(/^\w+/) === null) {
        res.render('register', {"status":"Invalid username"});
        return;
    }
    // password = md5(username + md5(password));
    connection.query('SELECT * FROM users WHERE username = ?', username, function(error, results, fields) {
        var status;
        if (results.length > 0) {
            status = "User already exists.";
            res.render('register', {"status" : status});
        }
        else {
            connection.query('INSERT INTO users (username, password, email) VALUES (?,?,?)', [username, password, email], function(error2, results2, fields2) {
                if (error2) {
                    status = "User exists.";
                    console.log(error2);
                }
                else {
                    status = "Successfully registered!";
                }
                res.render('register', {"status" : status});
            });
        }
    });
});
```

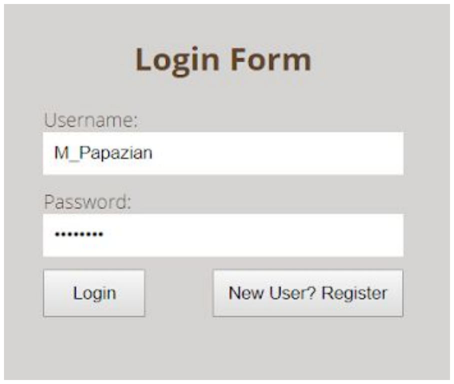


```
extends layout
block navBar
block otherUse
    div(id="register-form")
        form(action="/register" method="POST")
            h1 Register Form
            label Username: *
            input(id="register-username" name="username" type="text")
            label Password: *
            input(id="register-password" name="password" type="password")
            label Email: *
            input(id="register-email" name="email" type="text")
            button(id="register-btn") Register
        button(id="login-form-btn" onclick="move()") Back to Login
        p(id="register-result") #{status}
    script.
        function move() {
            window.location.href = '/'
        }
```

<u>User Login</u>

In index.js, login is handled by app.get('/login'), and is displayed using login.pug. Users may input their login info, and the backend cross references the input data with the users table. If their input data is an exact match for all of the attributes of any row in users, then the route proceeds; if not, then it asks the user to try again. If it's a perfect match, then their userID and username are stored as session variables, and the backend queries the users table again to determine if the user in question has any admin privileges; if they do, then the user's admin status is also stored as a session variable. It then begins loading the home page.

Display:

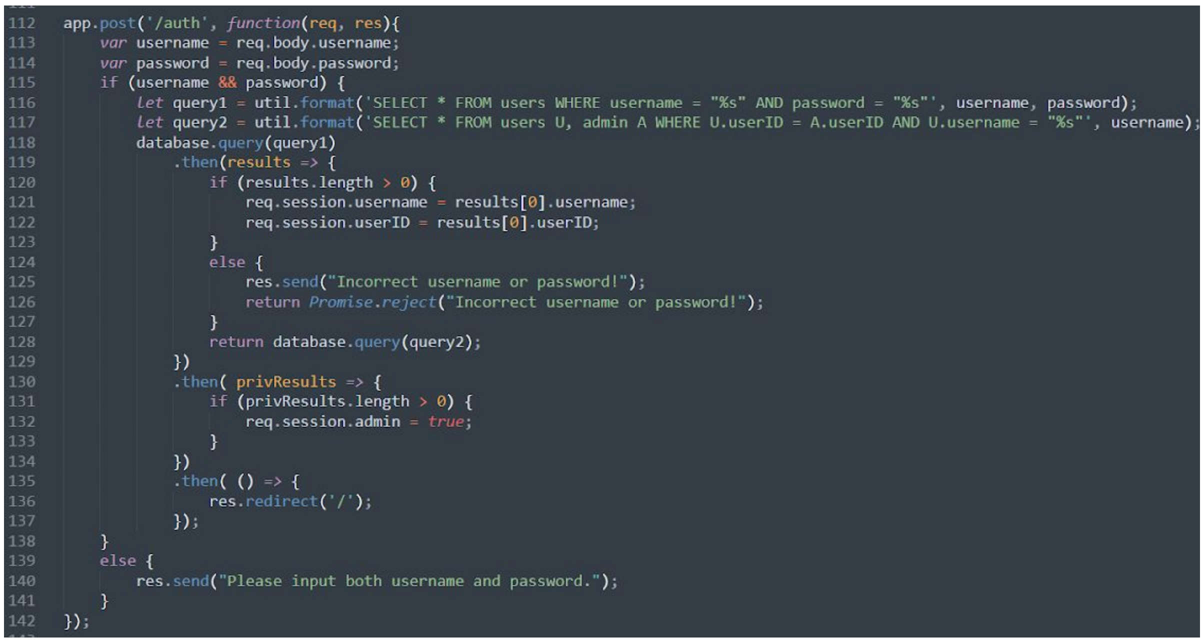


Login Form

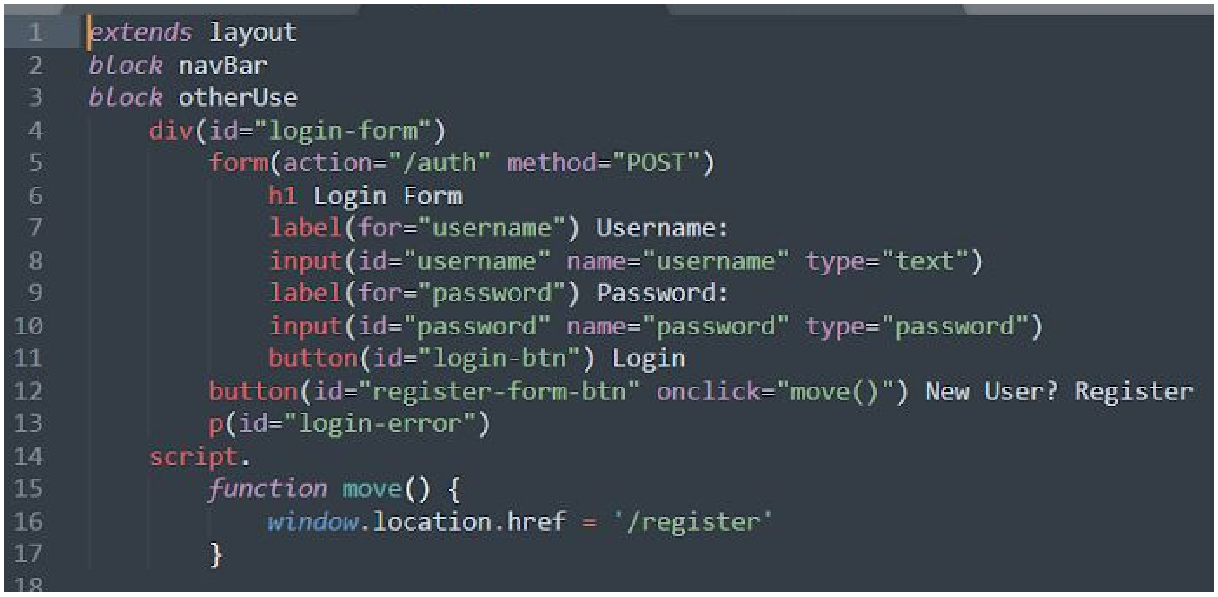
Username: M_Papazian

Password: ••••••••

Source:



```
app.post('/auth', function(req, res){
    var username = req.body.username;
    var password = req.body.password;
    if (username && password) {
        let query1 = util.format('SELECT * FROM users WHERE username = "%s" AND password = "%s"', username, password);
        let query2 = util.format('SELECT * FROM users U, admin A WHERE U.userID = A.userID AND U.username = "%s"', username);
        database.query(query1)
            .then(results => {
                if (results.length > 0) {
                    req.session.username = results[0].username;
                    req.session.userID = results[0].userID;
                }
                else {
                    res.send("Incorrect username or password!");
                    return Promise.reject("Incorrect username or password!");
                }
                return database.query(query2);
            })
            .then( privResults => {
                if (privResults.length > 0) {
                    req.session.admin = true;
                }
            })
            .then( () => {
                res.redirect('/');
            });
    }
    else {
        res.send("Please input both username and password.");
    }
});
```



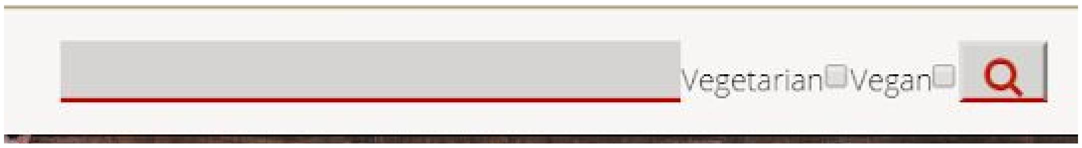
```
extends layout
block navBar
block otherUse
    div(id="login-form")
        form(action="/auth" method="POST")
            h1 Login Form
            label(for="username") Username:
            input(id="username" name="username" type="text")
            label(for="password") Password:
            input(id="password" name="password" type="password")
            button(id="login-btn") Login
        button(id="register-form-btn" onclick="move()") New User? Register
        p(id="login-error")
    script.
        function move() {
            window.location.href = '/register'
        }
```

<u>Search Recipe</u>

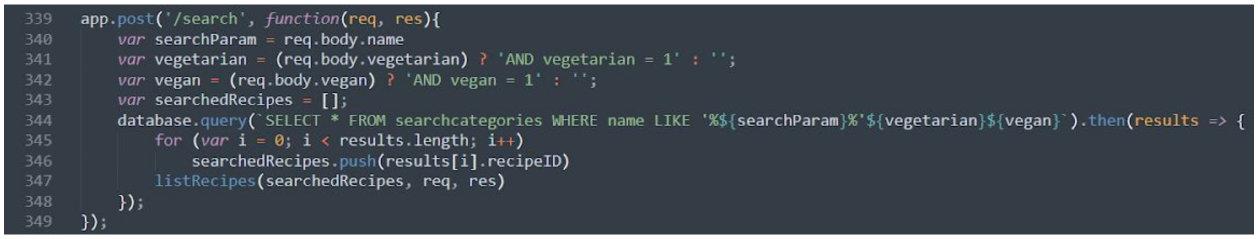
Searching for recipes can be done from any page once logged in to Recipedia. This is handled in index.js by using app.post('/search'), which makes use of index.js' listRecipes() function; and is accessed on the frontend by using layout.pug's navBar block.

Users can type in a query into the search bar. They may also tick the Vegetarian and Vegan boxes. The frontend passes their input to the backend, which attempts to match their query to any entry in the searchCategories' name attribute by using the 'LIKE' keyword, and also checks if these entries' vegan and vegetarian attributes are nonzero, if the boxes were ticked. It compiles a list of recipes that match the given input, and then calls listRecipes() using the array it constructs, and displays the results on the main page again.

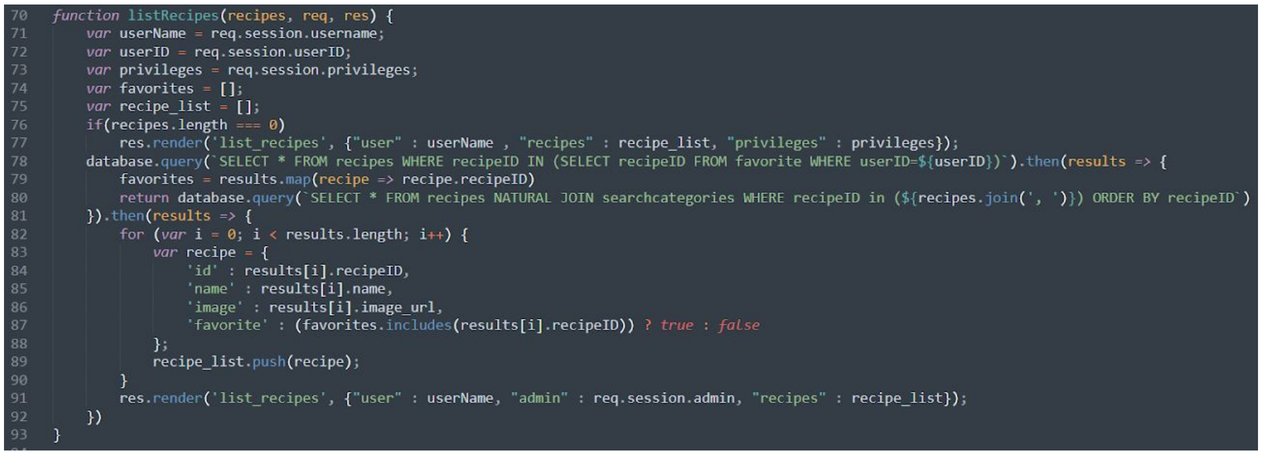
Display:



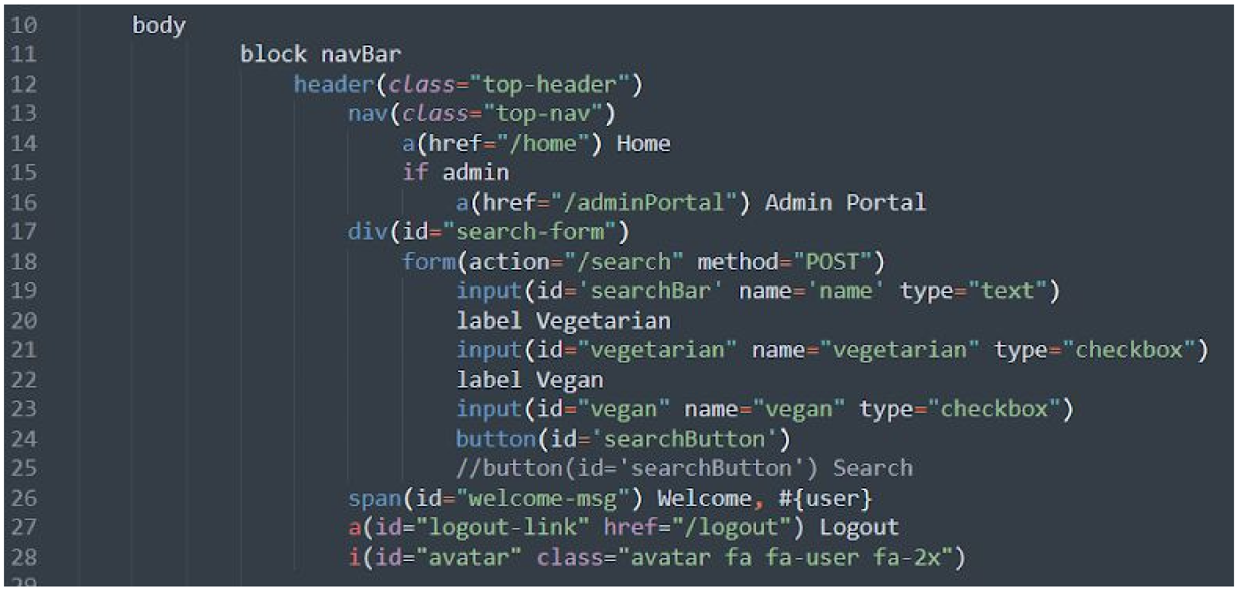
Source:



```
app.post('/search', function(req, res){
    var searchParam = req.body.name
    var vegetarian = (req.body.vegetarian) ? 'AND vegetarian = 1' : '';
    var vegan = (req.body.vegan) ? 'AND vegan = 1' : '';
    var searchedRecipes = [];
    database.query(`SELECT * FROM searchcategories WHERE name LIKE '%${searchParam}%'${vegetarian}${vegan}`).then(results => {
        for (var i = 0; i < results.length; i++)
            searchedRecipes.push(results[i].recipeID)
        listRecipes(searchedRecipes, req, res)
    });
});
```



```
function listRecipes(recipes, req, res) {
    var userName = req.session.username;
    var userID = req.session.userID;
    var privileges = req.session.privileges;
    var favorites = [];
    var recipe_list = [];
    if(recipes.length === 0)
        res.render('list_recipes', {"user" : userName , "recipes" : recipe_list, "privileges" : privileges});
    database.query(`SELECT * FROM recipes WHERE recipeID IN (SELECT recipeID FROM favorite WHERE userID=${userID})`).then(results => {
        favorites = results.map(recipe => recipe.recipeID)
        return database.query(`SELECT * FROM recipes NATURAL JOIN searchcategories WHERE recipeID in (${recipes.join(', ')}) ORDER BY recipeID`)
    }).then(results => {
        for (var i = 0; i < results.length; i++) {
            var recipe = {
                'id' : results[i].recipeID,
                'name' : results[i].name,
                'image' : results[i].image_url,
                'favorite' : (favorites.includes(results[i].recipeID)) ? true : false
            };
            recipe_list.push(recipe);
        }
        res.render('list_recipes', {"user" : userName, "admin" : req.session.admin, "recipes" : recipe_list});
    })
}
```



```
    body
            block navBar
                header(class="top-header")
                    nav(class="top-nav")
                        a(href="/home") Home
                        if admin
                            a(href="/adminPortal") Admin Portal
                    div(id="search-form")
                        form(action="/search" method="POST")
                            input(id='searchBar' name='name' type="text")
                            label Vegetarian
                            input(id="vegetarian" name="vegetarian" type="checkbox")
                            label Vegan
                            input(id="vegan" name="vegan" type="checkbox")
                            button(id='searchButton')
                            //button(id='searchButton') Search
                    span(id="welcome-msg") Welcome, #{user}
                    a(id="logout-link" href="/logout") Logout
                    i(id="avatar" class="avatar fa fa-user fa-2x")
```

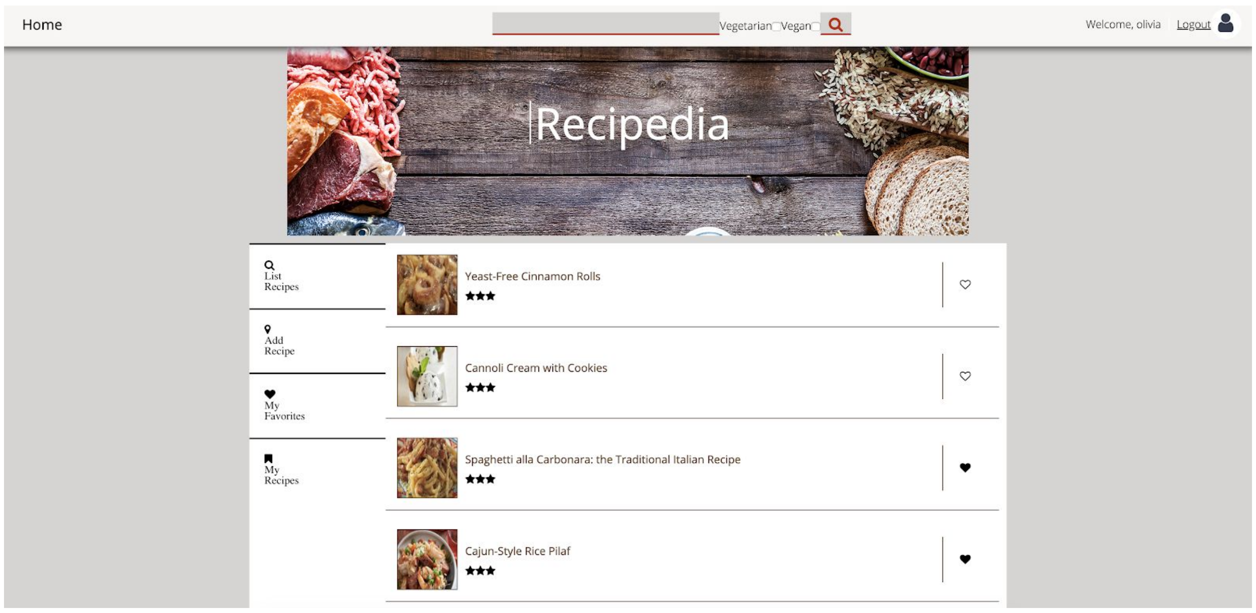
Rate Recipe

TODO

View Recipe

In the homepage (the first view after user login), user can view all the recipes in a list which shows the name and image of the recipe.

Display:



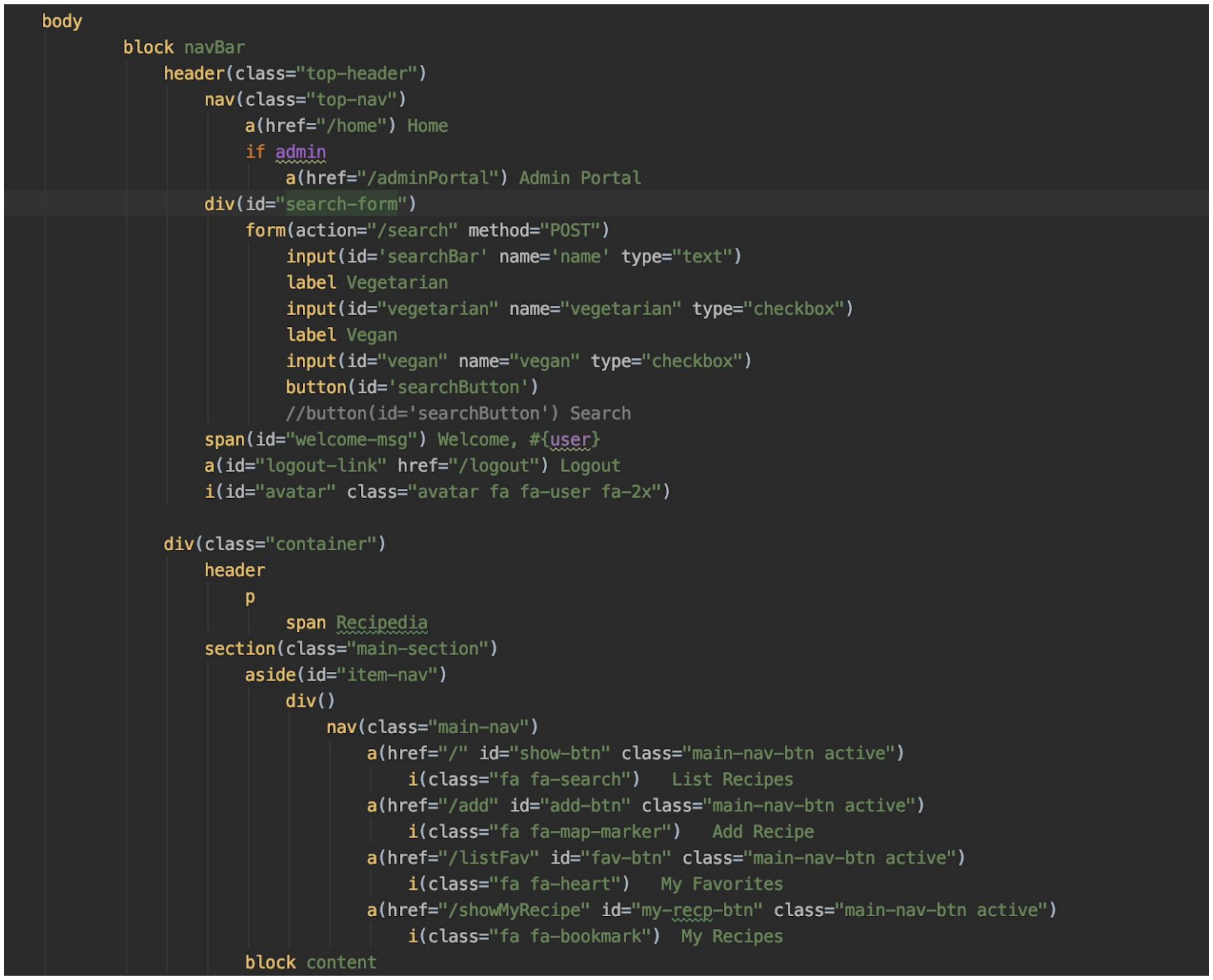
Yeast-Free Cinnamon Rolls

Cannoli Cream with Cookies

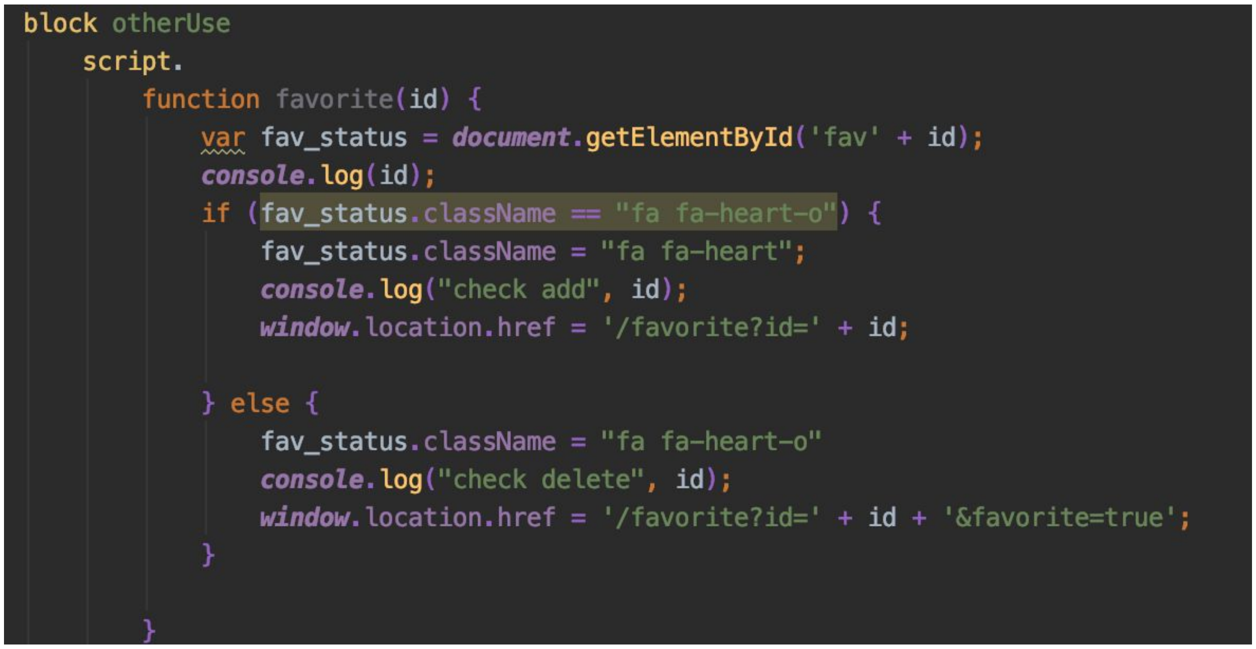
Spaghetti alla Carbonara: the Traditional Italian Recipe

Cajun-Style Rice Pilaf

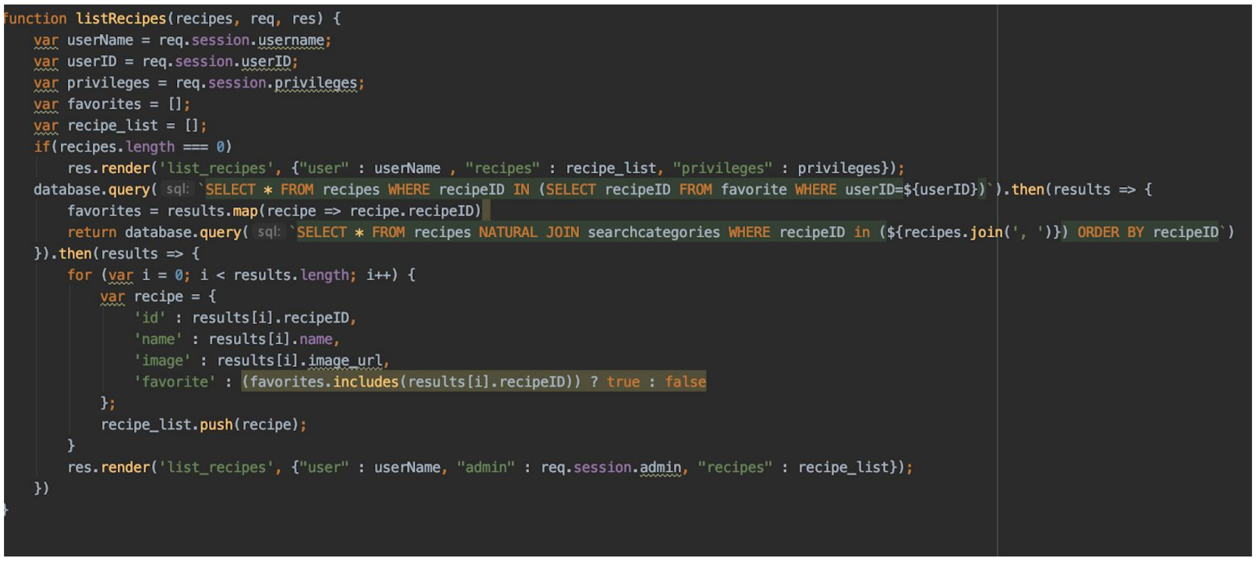
Source:



```
body
        block navBar
            header(class="top-header")
                nav(class="top-nav")
                    a(href="/home") Home
                    if admin
                        a(href="/adminPortal") Admin Portal
                div(id="search-form")
                    form(action="/search" method="POST")
                        input(id='searchBar' name='name' type="text")
                        label Vegetarian
                        input(id="vegetarian" name="vegetarian" type="checkbox")
                        label Vegan
                        input(id="vegan" name="vegan" type="checkbox")
                        button(id='searchButton')
                        //button(id='searchButton') Search
                span(id="welcome-msg") Welcome, #{user}
                a(id="logout-link" href="/logout") Logout
                i(id="avatar" class="avatar fa fa-user fa-2x")

            div(class="container")
                header
                    p
                        span Recipedia
                section(class="main-section")
                    aside(id="item-nav")
                        div()
                            nav(class="main-nav")
                                a(href="/" id="show-btn" class="main-nav-btn active")
                                    i(class="fa fa-search")   List Recipes
                                a(href="/add" id="add-btn" class="main-nav-btn active")
                                    i(class="fa fa-map-marker")   Add Recipe
                                a(href="/listFav" id="fav-btn" class="main-nav-btn active")
                                    i(class="fa fa-heart")   My Favorites
                                a(href="/showMyRecipe" id="my-recp-btn" class="main-nav-btn active")
                                    i(class="fa fa-bookmark")  My Recipes
                    block content
```



```
block otherUse
    script.
        function favorite(id) {
            var fav_status = document.getElementById('fav' + id);
            console.log(id);
            if (fav_status.className == "fa fa-heart-o") {
                fav_status.className = "fa fa-heart";
                console.log("check add", id);
                window.location.href = '/favorite?id=' + id;

            } else {
                fav_status.className = "fa fa-heart-o"
                console.log("check delete", id);
                window.location.href = '/favorite?id=' + id + '&favorite=true';
            }

        }
```



```
function listRecipes(recipes, req, res) {
    var userName = req.session.username;
    var userID = req.session.userID;
    var privileges = req.session.privileges;
    var favorites = [];
    var recipe_list = [];
    if(recipes.length === 0)
        res.render('list_recipes', {"user" : userName , "recipes" : recipe_list, "privileges" : privileges});
    database.query( sql: `SELECT * FROM recipes WHERE recipeID IN (SELECT recipeID FROM favorite WHERE userID=${userID})`).then(results => {
        favorites = results.map(recipe => recipe.recipeID)
        return database.query( sql: `SELECT * FROM recipes NATURAL JOIN searchcategories WHERE recipeID in (${recipes.join(', ')}) ORDER BY recipeID`)
    }).then(results => {
        for (var i = 0; i < results.length; i++) {
            var recipe = {
                'id' : results[i].recipeID,
                'name' : results[i].name,
                'image' : results[i].image_url,
                'favorite' : (favorites.includes(results[i].recipeID)) ? true : false
            };
            recipe_list.push(recipe);
        }
        res.render('list_recipes', {"user" : userName, "admin" : req.session.admin, "recipes" : recipe_list});
    })
}
```

<u>Create Recipe</u>

TODO

<u>Delete Recipe</u>

TODO

View Favorites

TODO

Report Recipe

TOO

Resolve Reports

TODO

PROJECT SETUP

1. Download Node.js, npm, and MySQL. If this is your first time downloading MySQL, configure the root user to your preference.
2. Clone the GitHub repository to your machine.
3. In a text editor, open "{outer_directories}\CS157A-Team36\Node-Express-Pug\index.js". On line 12, change the password to match your root password for MySQL.
4. Open the MySQL workbench, launch the root server.
    a. Under the File dropdown menu, select "Run SQL Script...", navigate to the repository directory, and select "cs157a_testdata.sql". This will generate the schema and tables for the database and populate them with data.
5. In the terminal, navigate to "{outer_directories}\CS157A-Team36\Node-Express-Pug".
6. In the terminal, run "npm install nodemon".
7. Next, run "npm install" to install all the dependencies used in the project.
8. Next, run "nodemon index.js". This will launch the web application.

9. Finally, in a browser, navigate to “localhost:3000”. This will bring you to the login page for Recipedia.
    a. You may register your own new user account, or use any of the test users stored in the ‘users’ table in the database.

<u>**PROJECT CONCLUSION**</u>

<u>Project Takeaways</u>

Timothy's:

During this project, the biggest things I learned were how to use MySQL, the how and why to every table in databases, and learning to use three-tier architecture. Prior to this, I've always worked on a single layer with minimal if any communication to other systems/programs, but this is the first time I've had to connect a backend to middleware to frontend. It's been an extremely interesting if exhausting experience.

Sean's:

This project taught me many new technologies! I did not know how to use Node.js, Express.js, or Pug before working on this project. While I did know some rudimentary MySQL before taking this class, it taught me much more about the language, as well as the underlying design behind constructing a proper database. This includes topics beyond MySQL itself, including ER design, schema design, and the three-tier architecture.

Yiyi's:

From working on the project, I practised how to design the relations of database which I learned from this course. This is the first time I design and use MySQL in a realtime project. In addition, I implemented the project in Node.js and Jade which are the languages I've never tried before. We started our project from requirement decision, ER design and three-tier architecture setting to each function implementing. Although there are many facts we need to improve in the future project, this is an unforgettable experience.

Future Improvements

As our implementation of this project focused mostly on satisfying functional requirements, there are many improvements that can be made beyond the core functionality. The user interface can be improved by expanding on it with CSS, especially the login and registration pages.

Additionally, the core functionalities of Recipedia can be expanded upon. The search function can be made more robust by allowing more unique search parameters, like searching based off of ingredient content, course, serving size, etc. The home page may be improved upon by allowing an easy way to view highly rated recipes, or a better way to sort recipes than the existing implementation. We could also add ways for users to edit their existing information, as in password and email updating.

We may also add additional functionality to the project. This could include adding an ingredient converter, allowing users to make ingredient substitutions if they don’t have one that’s listed in a recipe.