**CoffeeMaker API Endpoint Design**

Includes new ingredients endpoints and existing updated endpoints

# Overview

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| RESOURCE | GET | POST | PUT | DELETE |
| /ingredients/ | Fetch all existing ingredients | Create a new  ingredient | Error, unsupported | Error, unsupported. |
| /ingredients/{name} | Fetch ingredient by name if exists. Error otherwise. | Error, unsupported | Update ingredient by name, if exists. Error otherwise. | Error, unsupported. |
| /recipes | Fetch all existing recipes | Create new recipe | Error, unsupported. | Error, unsupported. |
| /recipes/{name} | Get recipe by name  if it exists. Error otherwise. | Error, unsupported. | Edit recipe by name if it exists. Error otherwise. | Delete recipe if it exists. Error otherwise. |
| /inventory | Fetch state of inventory. | Error, unsupported. | Update state of inventory. | Error, unsupported. |
| /users | Error, Unsupported | Create a new user and save to database | Error Unsupported | Error Unsupported |
| /users/{name} | Get the individual user by name. Error if the username is not found | Error, Unsupported | Error, Unsupported | Error, Unsupported |

**Note: Inventory’s endpoints remain unchanged as ingredients will handle the total units for each ingredient. Therefore, inventory’s current implementation can remain.**

Function: **getIngredients()**

Route: **GET /ingredients**

Description: This API endpoints returns a list of ingredients with a status of 200 (OK).

Data Details: Calling the endpoint should result in:

{

‘ingredients’: [

{

‘id’: 1,

‘name’: ‘Flour’,

‘units’: 1

},{

‘id’: 2,

‘name’: ‘Water’,

‘units’: 5

}] }

Function: **createIngredient()**

Route: **PUT /ingredients**

Description: This API endpoint should create the ingredient with the given name and state. If an ingredient with the name already exists, expect a status 400.

Data details: If CoffeeMaker already contains:

{

‘id’: 1,

‘name’: ‘Apple’,

‘units’: 0

}

Calling the endpoint with the data:

{

‘name’: ‘Flour’,

‘units’: 1

}

Should result in a status 200 as flour does not exist. We would expect Ingredients to be updated to {

‘id’: 1,

‘name’: ‘Apple’,

‘units’: 0

},

{

‘id’: 2,

‘name’: ‘Flour’,

‘units’: 1 }

Function: **getIngredient(String name)**

Route: **GET /ingredients/{name}**

Description: This API endpoint should fetch the ingredient that matches the given name. If no match is found, then a status of 404 (NOT FOUND).

Data details: If we have a ingredient with the name Apple and call endpoint GET /ingredients/apple, the API should return status 200 with the following data:

{

‘id’: 1,

‘name’: ‘Flour’,

‘units’: 1

}

If we call endpoint GET /ingredients/idontexist then the API should return status 404 (NOT FOUND) with no response JSON data.

Function: **updateIngredient(String name)**

Route: **PUT /ingredients/{name}**

Description: The API endpoint should update the Ingredient with the matched name to the given units. If there is no match, a status 404 (NOT FOUND) should be expected with no response JSON data.

Data details: If CoffeeMaker already contains:

{

‘id’: 1,

‘name’: ‘Apple’,

‘units’: 0

}

And endpoint **PUT ingredients/apple** is called with the data:

{

‘units’: 5

}

We should expect CoffeeMaker to be updated to

{

‘id’: 1,

‘name’: ‘Apple’,

‘units’: 5

}

And return the same content as a response.

If instead **PUT ingredients/idontexist** is called, expect a status 404 with no response data.

Function: **getRecipes()**

Route: **GET /recipes**

Description: The API endpoint should fetch and return all the recipes that currently exist with status 200 (OK).

Data details: Calling the API endpoint **GET /recipes** should result in status 200 with content such as:

{

‘recipes’: [ {

//Recipes have an id, name, price, and a list of ingredients

‘id’: 1,

‘name’: ‘aRecipe’,

‘price’: 5,

‘ingredients’:[{‘name’: ‘coffee’, ‘quantity’: 1},

{‘name’: ‘milk’, ‘quantity’: 1},

{‘name’: ‘sugar’, ‘quantity’: 1},

{‘name’: ‘chocolate’, ‘quantity’: 1,},

{‘name’: ‘vanilla’, ‘quantity’: 5}]

}] }

Function: **createRecipe(String name)**

Route: **POST /recipes**

Description: The API endpoint should create a recipe with the given name and state. The API shall return status 200 and the created recipe on success. If a recipe already exists with the name, a status 400 is returned with no response data.

Data details: If CoffeeMaker already contains:

{

‘id’: 1,

‘name’: ‘aRecipe’,

‘price’: 5,

‘ingredients’:[{‘name’: ‘coffee’, ‘quantity’: 1},

{‘name’: ‘milk’, ‘quantity’: 1},

{‘name’: ‘sugar’, ‘quantity’: 1},

{‘name’: ‘chocolate’, ‘quantity’: 1,},

{‘name’: ‘vanilla’, ‘quantity’: 5}]

}

And the API endpoint **POST /recipes** is called with the following data:

{

‘name’: ‘newRecipe’,

‘price’: 5,

‘ingredients’:[{‘name’: ‘coffee’, ‘quantity’: 1},

{‘name’: ‘milk’, ‘quantity’: 1},

{‘name’: ‘sugar’, ‘quantity’: 1},

{‘name’: ‘chocolate’, ‘quantity’: 1,},

{‘name’: ‘vanilla’, ‘quantity’: 5}]

}

We should expect a status 200 (OK) and the response to contain the newly created recipe. CoffeeMaker would then contain:

{

‘id’: 1,

‘name’: ‘aRecipe’,

‘price’: 5,

‘ingredients’:[{‘name’: ‘coffee’, ‘quantity’: 1},

{‘name’: ‘milk’, ‘quantity’: 1},

{‘name’: ‘sugar’, ‘quantity’: 1},

{‘name’: ‘chocolate’, ‘quantity’: 1,},

{‘name’: ‘vanilla’, ‘quantity’: 5}]

},

{

‘id’: 2,

‘name’: ‘newRecipe’,

‘price’: 5,

‘ingredients’:[{‘name’: ‘coffee’, ‘quantity’: 1},

{‘name’: ‘milk’, ‘quantity’: 1},

{‘name’: ‘sugar’, ‘quantity’: 1},

{‘name’: ‘chocolate’, ‘quantity’: 1,},

{‘name’: ‘vanilla’, ‘quantity’: 5}]

}

If instead a call to API endpoint **POST /recipes** with data

{

‘name’: ‘aRecipe’,

‘price’: 5,

‘ingredients’:[{‘name’: ‘coffee’, ‘quantity’: 1},

{‘name’: ‘milk’, ‘quantity’: 1},

{‘name’: ‘sugar’, ‘quantity’: 1},

{‘name’: ‘chocolate’, ‘quantity’: 1,},

{‘name’: ‘vanilla’, ‘quantity’: 5}]

}

We should expect a status 400 and no response data, as ‘aRecipe’ already exists.

Function: **getRecipe(String name)**

Route: **GET /recipes/{name}**

Description: The API endpoint should return a status 200 (OK) and the content of the recipe with the matched name. If no match is found, return a status of 404 (NOT FOUND) without response data.

Data details: Calling the endpoint GET /recipes/arecipe should result in status 200 and response data:

{

‘id’: 1,

‘name’: ‘aRecipe’,

‘price’: 5,

‘ingredients’:[{‘name’: ‘coffee’, ‘quantity’: 1},

{‘name’: ‘milk’, ‘quantity’: 1},

{‘name’: ‘sugar’, ‘quantity’: 1},

{‘name’: ‘chocolate’, ‘quantity’: 1,},

{‘name’: ‘vanilla’, ‘quantity’: 5}]

}

Function: **editRecipe(String name)**

Route: **PUT /recipes/{name}**

Description: API endpoint will edit the Recipe object with the corresponding identifier. The inputs will be a recipe with a matching name, updates the Recipe’s ingredients and price, and finally output a status 200 with the updated recipe object. If no recipes exist with the identifier, return an empty response with a status 404 (NOT FOUND). If the identifier and Recipe name do not match, return an empty response with a status 400 (Bad Request).

Data details: If CoffeeMaker already contains:

{

‘id’: 1,

‘name’: ‘Mocha’,

‘price’: 5,

‘ingredients’:[{‘name’: ‘coffee’, ‘quantity’: 1},

{‘name’: ‘milk’, ‘quantity’: 1},

{‘name’: ‘sugar’:, ‘quantity’: 1},

{‘name’: ‘chocolate’, ‘quantity’: 1,},

{‘name’: ‘vanilla’, ‘quantity’: 5}]

}

And the API endpoint **PUT /recipes/Mocha** is called with the following data that changes the price, removes sugar, adds cinnamon, and changes ingredient quantities:

{

‘name’: ‘Mocha’,

‘price’: 50,

‘ingredients’:[{‘name’: ‘coffee’, ‘quantity’: 2},

{‘name’: ‘milk’, ‘quantity’: 1},

{‘name’: ‘chocolate’, ‘quantity’: 1},

{‘name’: ‘vanilla’, ‘quantity’: 15},

{‘name’: ‘cinnamon’, ‘quantity’: 2}]

}

We should expect a status 200 (OK) and the Recipes to be saved to databaseSS. The returned Response entity would contain this message:

{

‘id’: 1,

‘name’: ‘Mocha’,

‘price’: 50,

‘ingredients’:[{‘name’: ‘coffee’, ‘quantity’: 2},

{‘name’: ‘milk’, ‘quantity’: 1},

{‘name’: ‘chocolate’, ‘quantity’: 1,},

{‘name’: ‘vanilla’, ‘quantity’: 15},

{‘name’: ‘cinnamon’, ‘quantity’: 2}]

}

If the call was made using **PUT /recipes/Latte** on this database with only the Mocha recipe, then we would expect an empty response and a status of 404(Not Found) from our ResponseEntity.

If the class was made using **PUT /recipes/Mocha** on this database with only the Mocha recipe, then the name, price, and ingredient amounts would be checked to make sure the Mocha database entry and JSON’s name are the same.

{

‘name’: ‘Mocha’,

‘price’: 50,

‘ingredients’:[{‘name’: ‘coffee’, ‘quantity’: -5},

{‘name’: ‘milk’, ‘quantity’: 1},

{‘name’: ‘chocolate’, ‘quantity’: 1},

{‘name’: ‘vanilla’, ‘quantity’: 15},

{‘name’: ‘cinnamon’, ‘quantity’: 2}]

}

In this case the amount of coffee is negative, so the ResponseEntity would return an empty message and a status of 400(Bad Request). This prevents the recipes from being edited into an invalid state with negative values on quantities and price.

Function: **createUser(String username, String password, int role)**

Route: **POST /users**

Description: This endpoint will add a new user to the database. The new user will have the input username, password, and role as an int. The role is a placeholder for an Enumerated type of RoleType which determines if the user is a Manager, Customer, or Barista. The status 200 will be output and the database will be updated with the user. If no username, password, or role is input for the operation a status of 400 (Bad Request) is returned and no user is added. If the username parameter matches the username of an existing user a status of 409 (Conflict) is returned and no user is added to the database.

Data details: If CoffeeMaker already contains:

{

[‘id’: 1,

‘username’: ‘fname’,

‘RoleType’: ‘CUSTOMER’]

}

And the API endpoint **POST /users** is called with the following data that adds the user to database.

{

‘username’: ‘jsmith’,

‘password’: ‘password’,

‘role’: 2

}

We should expect a status 200 (OK) and the Recipes to be saved to database. The returned Response entity would contain this message:

{

[‘id’: 1,

‘username’: ‘fname’,

‘RoleType’: ‘CUSTOMER’

,

‘id’: 2,

‘username’: ‘jsmith’,

‘RoleType’: ‘BARISTA’]

}

If the call was made using **POST /users** on this database without a username, password, or roletype only the Mocha recipe, then we would expect an empty response and a status of 400(Bad Request) from our ResponseEntity.

If the class was made using **POST /users** on this database when a user with the same username already exists then an empty response and a status of 409 (Conflict) from the ResponseEntity.

Data details: If CoffeeMaker already contains:

{

‘id’: 1,

‘username’: ‘fname’,

‘RoleType’: ‘CUSTOMER’

}

And the API endpoint **POST /users** is called on this database with a user of the same username

{

‘username’: ‘fname’

‘password’: ‘pass’

‘role’: 2

}

In this case a user with a matching username already exists and we cannot create this new user. A 409 (Conflict) is returned and the database is not updated.

Function: **login(String username, String password)**

Route: **GET /users/{name}**

Description: This endpoint will get a user and check that the user exists and the input password matches the user. This endpoint will be used during the login of the page. A status of 200 is returned if a user with the corresponding username exists and the passwords match. If the user is found with a matching username but wrong password a status of 400 (Bad Request) is returned. If no user with the matching username is found a status of 404 (Not Found) is returned.

Data details: If CoffeeMaker already contains:

{

[‘id’: 1,

‘username’: ‘fname’,

‘RoleType’: ‘CUSTOMER’]

}

And the API endpoint **POST /users/fname** if the passwords match, then a ResponseEntity is returned with the user and a status of 200.

{

‘id’: 1

‘username’: ‘fname’,

‘RoleType’: ‘CUSTOMER’

}

If the database contains the original data and **POST /users/fname** is called with an incorrect password, then an empty message is returned with a status of 400 (Bad Request)

If the database contains the original data and **POST /users/nothing** is called then a status of 404 (Not Found) is returned because no user with the username is returned.