Hamza Documentation

**Function: edit\_id(id)**

Handle the edit view for a specific recipe.

**Parameters:**

id (int): The ID of the recipe to edit.

**Returns:**

If the request method is 'GET':

Renders the 'edit.html' template with the recipe data.

If the request method is 'POST':

Updates the recipe with the submitted form data and redirects to the view page of the edited recipe.

The edit\_id function handles the editing of a specific recipe identified by its id. It searches for the recipe with the given id in the recipe list. If the recipe is not found, a flash message is set, and the user is redirected to the index page.

If the request method is 'POST', the function retrieves the updated values from the form and assigns them to the corresponding properties of the recipe object. It then updates the ingredients and instructions by iterating over the form fields and extracting the values.

Finally, if the request method is 'GET', the function renders the 'edit.html' template with the recipe data.

**Template: editID.html**

This template is used to render the edit form for a recipe.

**Structure:**

The editID.html file is an HTML template used for editing a specific recipe identified by its ID. It provides a user interface for updating the details of the recipe.

The template consists of the following components:

* Form: The main form element that wraps the input fields and buttons for editing the recipe.
* Recipe Details: Display area for showing the current details of the recipe.
* Input Fields: Input fields for updating the recipe details, such as recipe name, author, prep time, cook time, serving size, ingredients, and instructions.
* Submit Button: A button to submit the updated recipe details.

**Function: editRecipe(self, newRecipe)**

This method is used to edit an existing recipe in the recipe list with the updated information provided in newRecipe.

**Parameters:**

newRecipe: An instance of the Recipe class containing the updated recipe information.

**Behavior:**

* Iterate over each recipe in the data list.
* Check if the ID of the current recipe matches the ID of the newRecipe.
* If a match is found:
  + Update the recipe attributes with the corresponding attributes from newRecipe, if they exist.
  + If an attribute is not provided in newRecipe (i.e., None), the original value of the attribute is preserved.
  + Return from the function.
* If no match is found:
  + Raise a ValueError with the message "Recipe ID not found in the list of recipes."

**Function: viewSpecificRecipe(self, id)**

This method is used to retrieve a specific recipe from the recipe list based on the provided id.

**Parameters:**

id: The ID of the recipe to retrieve.

**Behavior:**

Iterate over each recipe in the data list.

Check if the ID of the current recipe matches the provided id.

If a match is found:

Return the recipe.

If no match is found:

Raise a ValueError with the message "Recipe with ID [id] not found."

**Class: TestRecipeManagement(unittest.TestCase)**

This class defines the unit tests for the RecipeManagement class in the recipeProject module.

**Methods:**

* setUp(self): A method that is executed before each test method. It creates an instance of the RecipeManager class for testing.
* test\_editRecipe\_existingRecipe(self): A test method that verifies the behavior of editing an existing recipe in the RecipeManager class.
* test\_editRecipe\_nonExistingRecipe(self): A test method that verifies the behavior of editing a non-existing recipe in the RecipeManager class.
* test\_editRecipe\_noChanges(self): A test method that verifies the behavior of editing a recipe without making any changes in the RecipeManager class.
* test\_editRecipe\_emptyData(self): A test method that verifies the behavior of editing a recipe when the data is empty in the RecipeManager class.

**Method: setUp(self)**

This method is executed before each test method. It creates an instance of the RecipeManager class to be used for testing.

**Behavior:**

Create an instance of the RecipeManager class and assign it to the recipe\_manager attribute of the test class.

**Method: test\_editRecipe\_existingRecipe(self)**

This test method verifies the behavior of editing an existing recipe in the RecipeManager class.

**Behavior:**

* Create a new recipe object named new\_recipe with the updated recipe information.
* Set the data attribute of the recipe\_manager instance with a list of existing recipes.
* Call the editRecipe method of the recipe\_manager instance, passing new\_recipe as the argument.
* Check if the edited recipe in the data list matches the expected updated attributes.
* Use assertions from the unittest.TestCase class, such as self.assertEqual, to compare the values.

**Method: test\_editRecipe\_nonExistingRecipe(self)**

This test method verifies the behavior of editing a non-existing recipe in the RecipeManager class.

**Behavior:**

* Create a new recipe object named new\_recipe with the updated recipe information.
* Set the data attribute of the recipe\_manager instance with a list of existing recipes.
* Use assertRaises to check if a ValueError is raised when calling the editRecipe method with new\_recipe.
* Check that the data attribute remains unchanged.
* Use assertions from the unittest.TestCase class to perform the checks.

**Method: test\_editRecipe\_noChanges(self)**

This test method verifies the behavior of editing a recipe without making any changes in the RecipeManager class.

**Behavior:**

* Create a new recipe object named new\_recipe that is identical to an existing recipe in the data attribute.
* Set the data attribute of the recipe\_manager instance with a list of existing recipes, including new\_recipe.
* Call the editRecipe method of the recipe\_manager instance, passing new\_recipe as the argument.
* Check that the data attribute remains unchanged, and the new\_recipe object is still at the same position in the list.
* Use assertions from the unittest.TestCase class to perform the checks.

**Method: test\_editRecipe\_emptyData(self)**

This test method verifies the behavior of editing a recipe when the data is empty in the RecipeManager class.

**Behavior:**

* Create a new recipe object named new\_recipe with the updated recipe information.
* Set the data attribute of the recipe\_manager instance to an empty list.
* Use assertRaises to check if a ValueError is raised when calling the editRecipe method with new\_recipe.
* Check that the data attribute remains empty.
* Use assertions from the unittest.TestCase class to perform the checks.

**Execution:**

If the Python script is executed directly (not imported as a module), the unittest.main() function is called to run the tests defined in the TestRecipeManagement class.

**Class: TestRecipeManagerIntegration(unittest.TestCase)**

This class defines the integration tests for the RecipeManager class in the recipeProject module.

**Methods:**

* setUp(self): A method that is executed before each test method. It creates an instance of the RecipeManager class for testing.
* test\_addRecipe\_and\_editRecipe(self): A test method that verifies the integration between the addRecipe and editRecipe methods in the RecipeManager class.

**Method: setUp(self)**

This method is executed before each test method. It creates an instance of the RecipeManager class to be used for testing.

**Behavior:**

Create an instance of the RecipeManager class and assign it to the recipe\_manager attribute of the test class.

**Method: test\_addRecipe\_and\_editRecipe(self)**

This test method verifies the integration between the addRecipe and editRecipe methods in the RecipeManager class.

**Behavior:**

* Create a new recipe object named new\_recipe with the initial recipe information.
* Add the new\_recipe to the recipe\_manager instance by calling the addRecipe method.
* Retrieve the added recipe from the data attribute of the recipe\_manager.
* Check that the added recipe has the correct values by comparing its attributes to the expected values using assertions such as self.assertEqual.
* Create a modified recipe object named modified\_recipe with updated recipe information.
* Edit the recipe in the recipe\_manager instance by calling the editRecipe method with the modified\_recipe.
* Retrieve the modified recipe from the data attribute of the recipe\_manager.
* Check that the modified recipe has the updated values by comparing its attributes to the expected values using assertions such as self.assertEqual.

**Method: tearDown(self)**

This method is executed after each test method. It performs cleanup operations.

**Behavior:**

Reset the recipe\_manager attribute to None.

**Execution:**

If the Python script is executed directly (not imported as a module), the unittest.main() function is called to run the tests defined in the TestRecipeManagerIntegration class.

**Function: do\_edit(self, arg)**

This function is a command implementation for editing a recipe in the Recipe Manager.

**Parameters:**

* arg: A string argument representing the ID of the recipe to be edited.

**Behavior:**

* Check if the arg parameter is provided.
* If arg is provided:
  + Check if it is a numeric value (recipe ID).
  + Retrieve the recipe object with the given ID from the rm (Recipe Manager) instance using the viewSpecificRecipe method.
  + If a recipe is found:
    - Print the details of the current recipe.
    - Prompt the user for new details to update the recipe.
    - Create a deep copy of the recipe using copy.deepcopy() to store the updated details.
    - Prompt the user for each field of the recipe and update it accordingly.
    - If the field is 'ingredients':
      * Prompt the user to enter new ingredients, including name, measurement, and quantity.
      * Validate the ingredient details and append them to the new\_ingredients list.
      * Set the new\_recipe.ingredients attribute to the updated list of ingredients.
    - If the field is 'instructions':
      * Prompt the user to enter new instructions, including instruction number and text.
      * Validate the instruction details and update the new\_instructions dictionary.
      * Set the new\_recipe.instructions attribute to the updated dictionary of instructions.
    - For other fields:
      * Prompt the user for input and update the corresponding attribute in new\_recipe.
      * Perform specific validations for certain fields (e.g., recipe name, preparation time, cook time, serving size).
    - Print the updated details of the recipe.
    - Ask the user if they want to save the changes.
    - If the user confirms:
      * Call the editRecipe method of the rm instance to update the recipe with the new\_recipe.
      * Print a success message indicating that the recipe was updated successfully.
    - If the user does not confirm, print a message indicating that the changes were discarded.
  + If no recipe is found with the given ID, print a message indicating that the recipe was not found.
* If arg is not provided, print a message indicating the correct command format.