**RECI.PY**

**TECHNICAL DOCUMENTATION**

SUBMITTED BY:

Aldam, Raid

Alrifai, Hamza

Esufji, Fahad

Manzano, Josh

William, Samson

SUBMITTED TO:

Bouallegue, Seifeddine

SUBMITTED ON:

June 2023

**Introduction**

Reci.py is a recipe management system that provides users with the ability to add, edit, and delete recipes. Users can import and export data from files for convenient data storage and retrieval, as well as save recipe data in memory. You can effectively manage and organize your recipe collection with Reci.py, ensuring quick access to your culinary creations.

Reci.py is built on Python. We were able to implement essential functionalities with ease thanks to Python's versatility and large library system. During development, we used Git and GitHub to ensure effective version control and collaboration. Git gave us a platform to manage branches and keep track of changes. GitHub not only served our central repository, but also facilitated team tasks and code reviews. Additionally, we automated the testing procedures by integrating CircleCI into our development workflow. This made sure that our code remained stable and that no problems were found and were fixed right away.

**Design**

We used an object-oriented design process to create Reci.py. This greatly helps in maintaining the codebase's modularity. We were able to manage and manipulate recipe-related operations more easily by encapsulating data and functionality into classes using these principles. Furthermore, we used JSON files to store recipes.

**Design: Recipe Class**

Each recipe is represented by an instance of the class “Recipe”. It contains a method to display the recipe in a formatted string.

The constructor initializes a Recipe object with the following parameters:

* id (int) – the unique identifier of the recipe object.
* recipe\_name (str) – the title of the recipe.
* recipe\_author (str) – the name of the recipe's creator.
* prep\_time (int) – the preparation time in minutes.
* cook\_time (int) – the cooking time in minutes.
* serving\_size (int) – how many servings the recipe makes.
* ingredients (list) – the list of ingredient dictionaries, each of which contains the following keys:
  + 'ingredientName' (str) – the name of the ingredient.
  + 'quantity' (int) – the quantity of the ingredient.
  + 'measurement' (str) – the unit for the quantity.
* instructions (dict) – a dictionary containing the recipe's step-by-step directions. The corresponding instructions are represented by the values, while keys stand for the step numbers.

**Design: RecipeManager Class**

The “RecipeManager” class provides methods for viewing, manipulating, importing, and exporting the collection of recipes. The constructor initializes an object with an empty list called “data”, which stores recipes.

1. viewRecipe(self) – prints all recipes in the collection to the screen.
2. addRecipe(self, recipe) – adds a new recipe to the collection. It detects if a recipe with the same ID or name already exists and raises an exception if it does.
3. updateRecipe(self, id, new\_recipe\_name=None, new\_recipe\_author=None, new\_prep\_time=None, new\_cook\_time=None, new\_serving\_size=None) – updates the attributes of an existing recipe using the given id.
4. deleteRecipe(self, id) – removes a recipe from the collection using the given id.
5. exportRecipes(self, filename=”DefaultExportName”) – exports the collection to a JSON file. If an “exports” directory does not already exist, one is created. The file is saved as the given filename.
6. importRecipes(self, filename) – imports recipes from a JSON file. It confirms the file's existence, examines its content, and assigns the data to the RecipeManager object's data attribute.

**Test Plan**

The Python Unittest library is used in our test strategy to verify the functionality of our program. The plan considers a wide range of potential outcomes, including edge cases and potential issues, and ensures that any errors are found and fixed. CircleCI makes it simple to incorporate these tests into our development process, guaranteeing that our code is consistently examined and validated for accuracy.

**Test Plan: viewRecipe**

1. Test Case: Viewing a Recipe List

* Description: This test ensures that the method returns the expected data.
* Expected Outcome: The method should return a string list that is consistent with the expected data.

**Test Plan: addRecipe**

1. Test Case: Adding a Recipe

* Description: This test verifies the basic functionality of the addRecipe method by creating a recipe dictionary.
* Expected Outcome: The recipe will be added to the collection.

1. Test Case: Adding a Recipe with an Existing ID

* Description: This test ensures that the addRecipe method prevents adding recipes with duplicate ID’s.
* Expected Outcome: The method should raise an exception and not add anything.

1. Test Case: Adding a Recipe with an Existing Name

* Description: This test ensures that the addRecipe method prevents adding recipes with duplicate recipe names.
* Expected Outcome: The method should raise an exception and should not add anything.

**Test Plan: updateRecipe**

1. Test Case: Updating a Recipe with an Existing ID

* Description: This test verifies the basic functionality of the updateRecipe method by updating a recipe with a valid ID and checking if the recipe was indeed updated.
* Expected Outcome: The recipe’s attributes must be different after the method is called.

1. Test Case: Updating a Recipe with a Non-Existent ID

* Description: This test ensures that the method ignores updates to recipes with invalid ID’s.
* Expected Outcome: The method should return False and should not update anything.

1. Test Case: Updating a Recipe without Providing Changes

* Description: This test ensures that the updateRecipe method does not change the recipe if the user does not provide any new values.
* Expected Outcome: The recipe should remain untouched, and the method should return True.

**Test Plan: deleteRecipe**

1. Test Case: Deleting a Recipe by ID

* Description: This test verifies the basic functionality of the deleteRecipe method by deleting a recipe with an ID and checking if the recipe is deleted from the collection.
* Expected Outcome: The recipe with the specified ID should be deleted from the recipe manager's data.

1. Test Case: Deleting a Non-Existent ID

* Description: This test ensures that the deleteRecipe method handles the deletion of a non-existent recipe ID correctly.
* Expected Outcome: The method should not delete anything.

1. Test Case: Deleting a Recipe with a String ID

* Description: This test validates the behavior of the deleteRecipe method when a recipe with a string ID is attempted to be deleted.
* Expected Outcome: The method should not delete anything.

**Test Plan: exportRecipes**

1. Test Case: Exporting to JSON

* Description: This test ensures that the data in the JSON files are consistent with the data exported to them.
* Expected Outcome: The data in the files are the same as the data exported.

1. Test Case: Exporting to a Non-Existing File Path

* Description: This test checks if the exportRecipes method does not create a file with no provided name.
* Expected Outcome: The test confirms that the file does not exist.

**Test Plan: importRecipes**

1. Test Case: Importing from JSON

* Description: This test verifies the basic functionality of the importRecipes method by importing a recipe from a JSON file and comparing it to expected data.
* Expected Outcome: The recipe will be consistent with the expected data.

1. Test Case: Importing from a Non-Existent File Path

* Description: This test ensures that the importRecipes method returns an error message when provided with a non-existent JSON filename.
* Expected Outcome: The method should return an error message.