**Technical Documentation**

**Overview**

The Recipe Management System is a graphical user interface (GUI) application built using the Tkinter library in Python. It allows users to manage a collection of recipes, including adding, viewing, editing, and deleting recipes. Additionally, users can export their recipe data to a JSON file and import recipes from a JSON file.

In design, Azure theme was utilized to enhance the user interface which contributes to improve the overall aesthetics. Without the theme, the user interface would’ve been generic and visual feel good to look at.

Required to download to function:

Install tkinter

Pip install tk

Install Tkinter themes

Pip install ttkthemes

**Classes**

1. RecipeGUI Class

Constructor

- \_\_init\_\_(self, root): Initializes the RecipeGUI class, sets up the main GUI window (root), initializes the recipes dictionary, creates widgets (UI elements), and calls initialize\_data().

Methods

- initialize\_data(self): Reads existing recipe data from a JSON file ("Recipe\_database.json") and updates the recipe list if data exists.

- create\_widgets(self): Defines and packs various widgets for the GUI, such as labels, buttons, and a listbox.

- open\_form(self, recipe=None, edit=False): Opens a form for adding, viewing, or editing a recipe. The form includes fields for recipe name, ingredients, instructions, and rating.

- add\_to\_database(self, res): Adds a recipe (res) to the JSON database file ("Recipe\_database.json").

* add\_recipe(self): Opens the form for adding a new recipe.
* edit\_recipe(self): Opens the form for editing the selected recipe from the list.
* view\_recipe(self): Opens the form for viewing the selected recipe from the list.
* delete\_recipe(self): Deletes the selected recipe after confirming with the user.
* del\_from\_database(self): Updates the JSON database file after deleting a recipe.
* export\_recipes(self): Allows the user to export all recipes to a JSON file.
* import\_recipes(self): Allows the user to import recipes from a JSON file.
* exit\_app(self): Closes the application.
* update\_recipe\_list(self): Updates the recipe list in the GUI.

**2. Main Application**

- if \_\_name\_\_ == "\_\_main\_\_": : Sets up and runs the main GUI application by creating a Tkinter root window, setting the title and geometry, creating an instance of the RecipeGUI class, and starting the Tkinter main loop.

**File Handling**

- The application uses a JSON file ("Recipe\_database.json") to store recipe data persistently. Recipes are loaded from and saved to this file.

**GUI Components**

- The GUI includes labels, buttons, a listbox for displaying recipes, and various entry and text widgets for input and display.

Themes

- The application attempts to set a light theme using an external "azure.tcl" file.

Error Handling

- The application handles potential errors such as file not found, empty data, and missing recipe details by displaying appropriate error messages using Tkinter messagebox.

Limitations

- The application assumes valid input from the user and does not implement extensive error checking for input fields.

**Test Cases**

1. test\_app\_doesnt\_open

Objective:

Ensure that the application window does not open automatically upon initialization.

Steps:

This test case sets up a mocked Tk instance and checks that the mainloop method is not invoked, ensuring that the application window remains closed. You can run this test as part of your continuous integration (CI) process to catch any potential issues related to the Tkinter main loop.

2. test\_add\_recipe

Objective:

Verify the functionality of adding a new recipe to the application.

Steps:

In this test scenario, a new recipe is created with specified details, including the recipe name, ingredients, instructions, category, and rating. The created recipe is then added to the application using the add\_to\_database method. Subsequently, the initialize\_data method is invoked to ensure data persistence by loading existing recipes from the database file ("Recipe\_database.json"). The test validates the successful addition of the recipe to the application by checking if its details exist in the database file. The expected outcome is that the recipe is seamlessly integrated into the application, and its specific details are accurately stored in the database file, affirming the persistence and functionality of the recipe management system.

3. test\_add\_recipe\_to\_database

Objective:

Confirm that a recipe is correctly added to the database.

Steps:

In this test case, a new recipe is mocked and subsequently added to the database. The test verifies the accuracy of this addition by reading the database file and checking if the added recipe matches the expected one. The expected outcome is that the recipe should be successfully added to the database, and the database file should accurately reflect this addition.

4. test\_export\_recipes

Objective:

Validate the export functionality of the application for recipes.

Steps:

In this test case, a sample set of recipes is set up, and the export functionality is triggered. The test ensures that the file dialog is opened and the showinfo function is called during the export process. The expected outcome is that the export process should be completed successfully, with both the file dialog and showinfo functions being invoked as part of the successful export operation.

5. test\_import\_recipes

Objective:

Test the import functionality of the application for recipes.

Steps:

In this test scenario, the application is prepared by setting up existing recipes to simulate a pre-existing state. The import process is then initiated, triggering the opening of a file dialog for the user to select a file containing recipes in JSON format. The test checks whether the file dialog is appropriately opened, facilitating the selection of the import file. Additionally, the showinfo function is called to provide informational feedback to the user. The expected outcome of the test is that the import process effectively integrates the recipes from the selected file into the application. Moreover, the invocation of the file dialog and the showinfo functions validates the smooth execution of the import operation, enhancing the overall usability and feedback mechanisms of the recipe management system.

6. test\_delete\_recipe\_successful

Objective:

Ensure the successful deletion of a recipe from the application.

Steps:

In this test scenario, the application is initialized with a pre-existing set of recipes to establish a starting state. The askyesno function, responsible for obtaining user confirmation, is mockingly simulated to affirm user consent for recipe deletion. Following this, a specific recipe is selected for deletion, and the test ensures that the deletion process is executed successfully. The expected outcome involves the seamless removal of the specified recipe, with the associated methods, such as del\_from\_database, being appropriately invoked. This validates the effective functionality of the recipe deletion process and ensures that all necessary procedures related to the deletion operation are carried out as expected within the recipe management system.

**Extra unit testing(not included in automated circle ci)**

1. test\_view\_recipe

Objective: Validates the functionality to view a recipe.

Steps:

In this test scenario, a specific recipe is added to the application's recipe list, followed by the selection of this added recipe within the GUI list. The 'View Recipe' button is then activated, and the verification step confirms that the 'open\_form' method is appropriately invoked, ensuring the correct passing of recipe parameters for viewing.

1. test\_edit\_recipe

Objective: Tests the editing functionality of a recipe.

Steps:

This test sequence begins by initializing the recipe data and subsequently introduces a new recipe into the database. Following this addition, modifications are made to the details of the added recipe. The test then proceeds by initiating the editing process, selecting the recipe intended for editing. Verification involves confirming that the altered recipe details diverge from the original and have been accurately updated within the application's recipe list. Additionally, it ensures that the 'open\_form' method is appropriately invoked, passing the correct parameters for editing purposes.

**Integration testing**

1. test\_export\_and\_import\_recipes

Objective:

Validate the export functionality import functionality of the application for recipes. of the application for recipes.

Steps:

Trigger the export functionality to save the recipes as a JSON file. Verify that the file dialog opens and the showinfo function is called, confirming successful export completion. it Initiate the import process, prompting the user to select a JSON file containing recipes via a file dialog. Ensure the file dialog opens appropriately and the showinfo function provides informative feedback. Validate the integration of recipes from the selected file into the application during the import process. summaries this don't add anything extra

2 test\_add\_and\_delete\_recipe

Objective:

Confirm the accurate addition of a recipe to the database within the application.

Steps:

The test begins by adding a new recipe (new\_recipe) to the application's database.

It validates the successful addition by checking if the recipe exists in self.app.recipes and if its details match the expected attributes.

Next, it proceeds to delete the added recipe by simulating user input for deletion.

The patch ensures that the application selects the added recipe ("Integration Test Recipe") for deletion.

After the deletion process, it checks whether the count of recipes has changed, confirming that the recipe has been successfully deleted from the application's database.

The test case ensures that recipes can be added and deleted successfully within the application, maintaining data consistency and proper functioning of the deletion mechanism.

In the Integration Testing phase, CircleCI played a crucial role as an automated testing framework. This framework was employed to execute tests automatically whenever changes were pushed to the repository. The integration testing process involved an integration class specifically designed for this purpose.

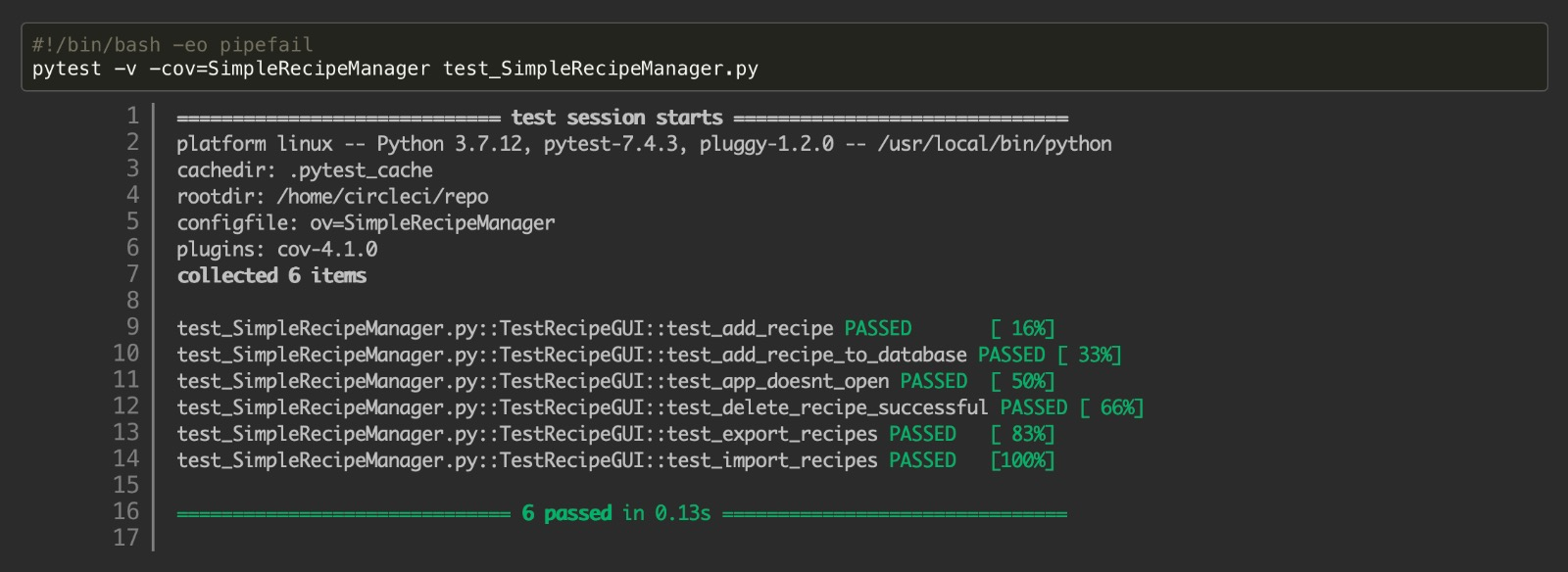
Within the integration class, there were functions crafted to orchestrate the collaboration of various units. These functions essentially acted as coordinators, calling upon the individual functions previously tested in the unit testing phase. By doing so, the team ensured that when different components were brought together, their interactions were seamless and the integrated system operated as expected.

CircleCI, with its automated capabilities, streamlined this integration testing process, allowing for efficient and consistent testing whenever modifications were introduced. The integration class served as a bridge, validating that the units, which were confirmed to work independently during unit testing, harmoniously cooperated when integrated into the larger system. This meticulous approach using CircleCI and integration testing contributed to the overall reliability and stability of the Recipe Program.

**Test Results**

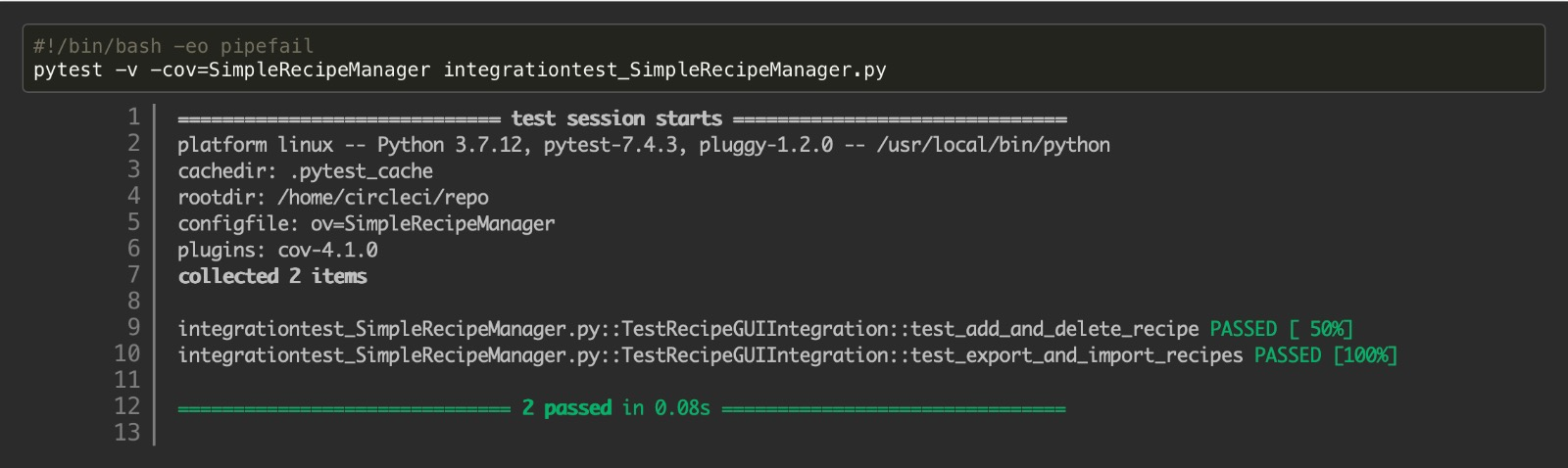
**Unit Testing:**

Successful completion of unit tests, where all tests pass without encountering errors or failures, is a strong indicator of the code's correctness and alignment with specified requirements. In such cases, the tests validate that individual components and functions of the codebase operate as intended, contributing to the overall quality of the software. On the other hand, if specific test cases fail to produce the expected outcomes, the failure details, including the names of the failed tests and associated stack traces, are presented.

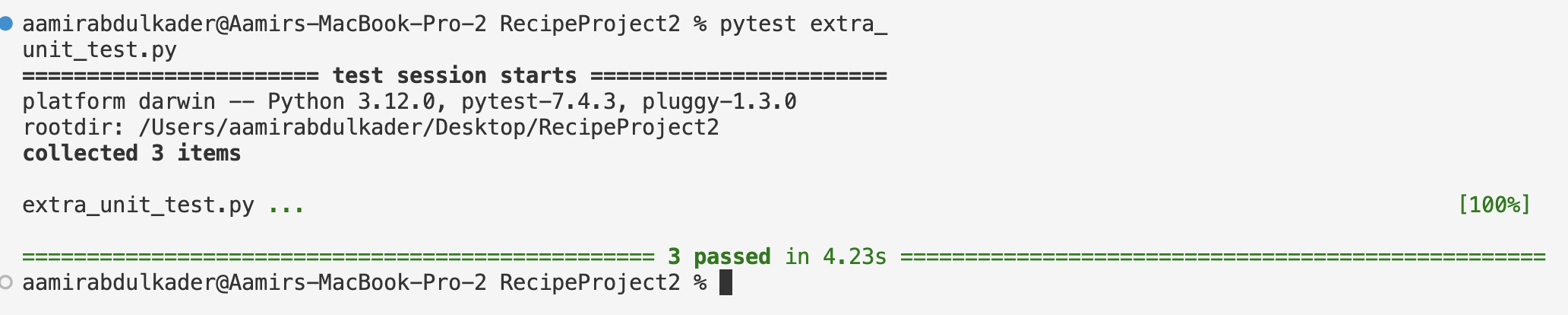


**Integration Testing:**

This testing phase focuses on the collaboration between different parts of the system, ensuring that they integrate smoothly and operate collectively. Passing integration tests provides assurance that the Recipe Management System is not only correct at a unit level but also when components are combined, addressing potential issues related to their interactions.



**Extra unit testing**



CircleCI:

All defined jobs, including unit testing and interface testing, run successfully. The overall build is considered stable and passes.

