Escuela Superior Politécnica Del Litoral

Faculty Of Electrical and Computer Engineering

Software Engineering II

Acceptance Test Workshop

José Sebastian Baidal Zambrano

II Term 2024

Repository : https://github.com/jsbaidal/AcceptanceTesting.git

Contenido

Introduction :

The To-Do List Management system is a command-line application built to demonstrate the principles of robust software development through **Behavior-Driven Development (BDD)**. Using Python as the primary programming language, the system's functionality has been rigorously tested with the **Behave** framework, a popular BDD tool. This ensures the application adheres to user requirements while maintaining high-quality standards.

Key technologies and methodologies employed in this project include:

- **Python**: The core programming language used to implement the application logic.
- **Behave**: A BDD framework for Python that allows developers to write human-readable test scenarios using Gherkin syntax.
- **Gherkin Syntax**: Provides a structured and natural language approach to writing feature files, making tests accessible to both technical and non-technical stakeholders.

The BDD methodology emphasizes collaboration between developers, testers, and stakeholders by focusing on defining the behavior of the system in simple terms. Each scenario is written in a Given-When-Then format, ensuring that the expected behavior of the application is clear and testable.

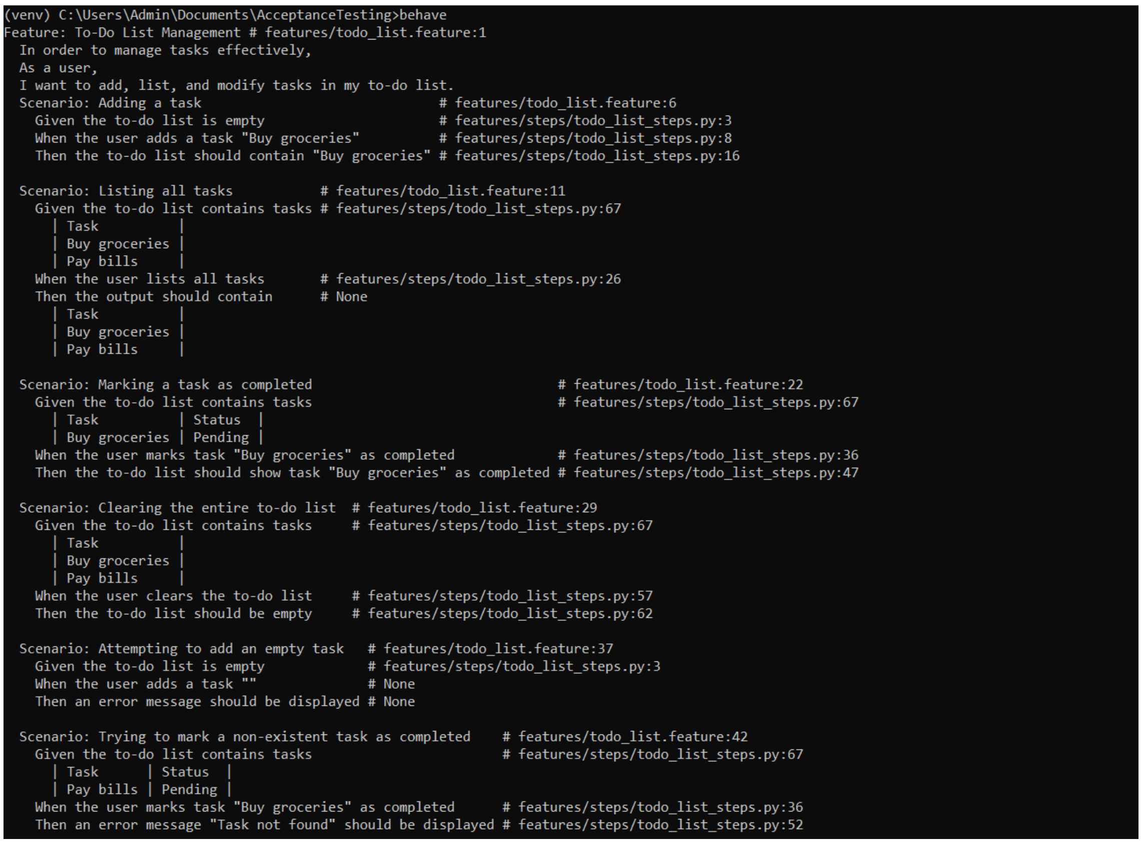
The Behave framework allows for:

- Automated testing of application behavior.
- Easy tracking of features and scenarios through structured `.feature` files.
- Reusable step definitions that simplify test implementation and maintenance.

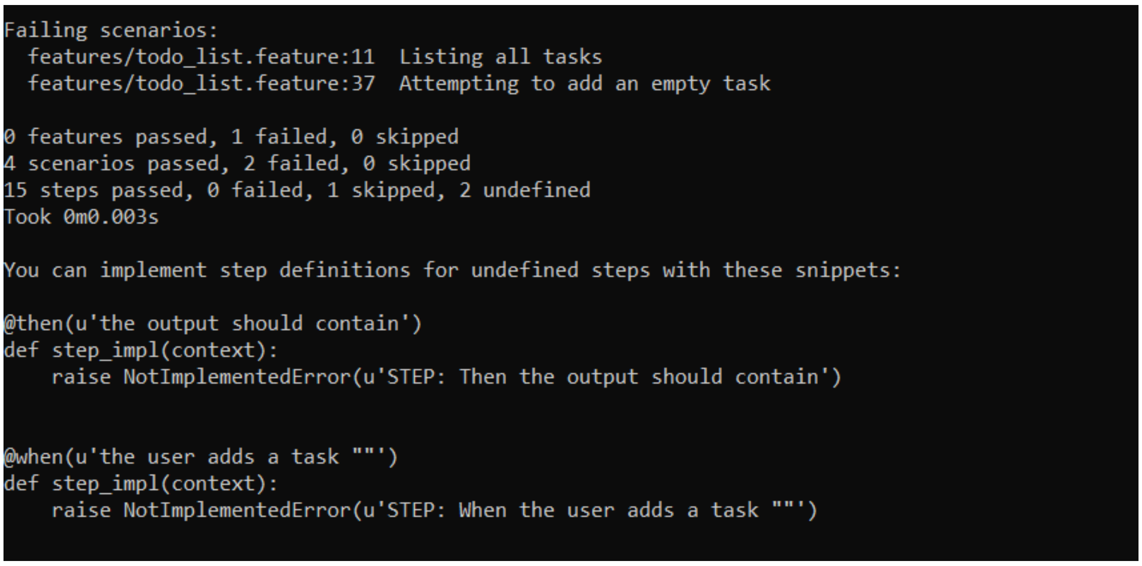
This documentation provides an overview of the test scenarios implemented, their alignment with system requirements, and the technologies used to ensure that the system remains reliable and maintainable.

Development

Initial Errors :

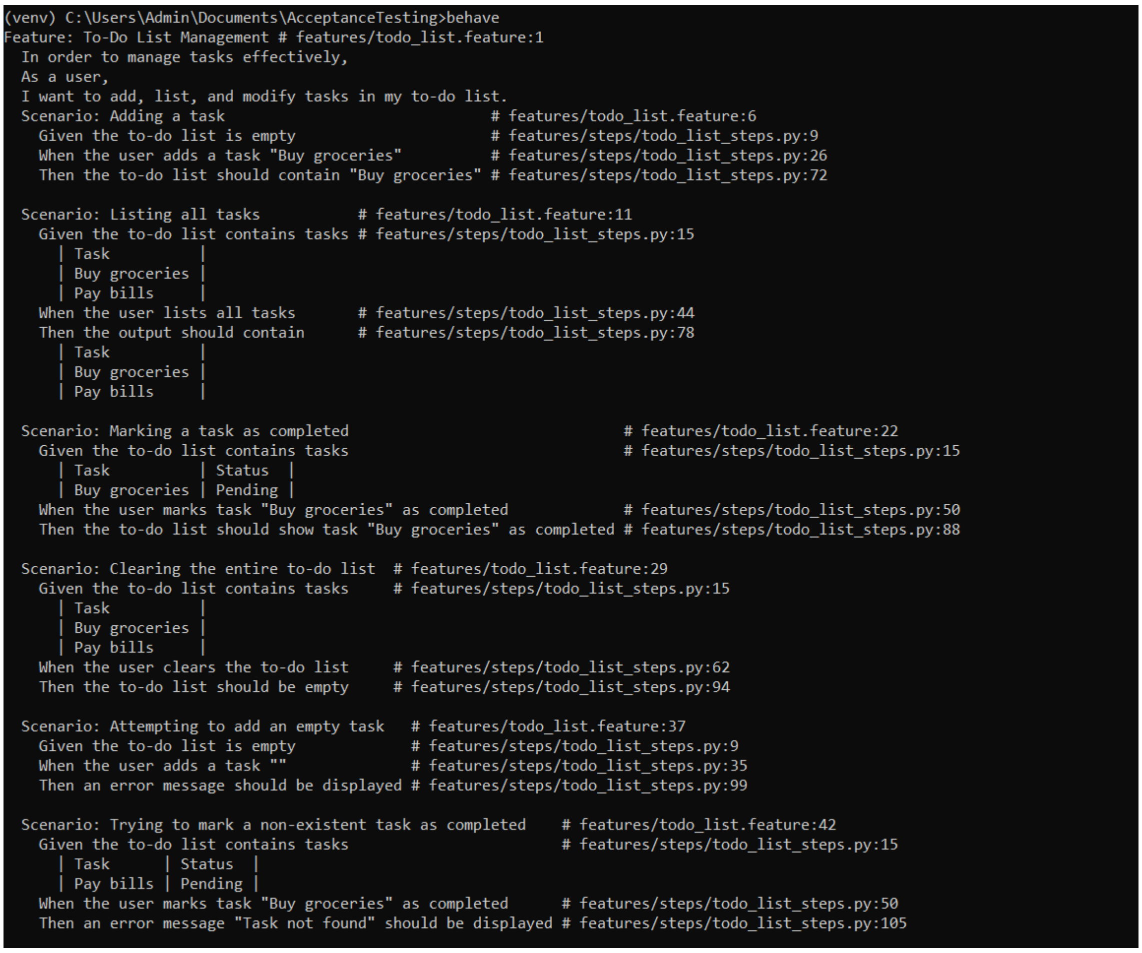


```
(venv) C:\Users\Admin\Documents\AcceptanceTesting>behave
Feature: To-Do List Management # features/todo_list.feature:1
  In order to manage tasks effectively,
  As a user,
  I want to add, list, and modify tasks in my to-do list.
  Scenario: Adding a task                              # features/todo_list.feature:6
    Given the to-do list is empty                      # features/steps/todo_list_steps.py:3
    When the user adds a task "Buy groceries"          # features/steps/todo_list_steps.py:8
    Then the to-do list should contain "Buy groceries" # features/steps/todo_list_steps.py:16

  Scenario: Listing all tasks           # features/todo_list.feature:11
    Given the to-do list contains tasks # features/steps/todo_list_steps.py:67
      | Task          |
      | Buy groceries |
      | Pay bills     |
    When the user lists all tasks       # features/steps/todo_list_steps.py:26
    Then the output should contain      # None
      | Task          |
      | Buy groceries |
      | Pay bills     |

  Scenario: Marking a task as completed                              # features/todo_list.feature:22
    Given the to-do list contains tasks                              # features/steps/todo_list_steps.py:67
      | Task          | Status  |
      | Buy groceries | Pending |
    When the user marks task "Buy groceries" as completed            # features/steps/todo_list_steps.py:36
    Then the to-do list should show task "Buy groceries" as completed # features/steps/todo_list_steps.py:47

  Scenario: Clearing the entire to-do list  # features/todo_list.feature:29
    Given the to-do list contains tasks     # features/steps/todo_list_steps.py:67
      | Task          |
      | Buy groceries |
      | Pay bills     |
    When the user clears the to-do list     # features/steps/todo_list_steps.py:57
    Then the to-do list should be empty     # features/steps/todo_list_steps.py:62

  Scenario: Attempting to add an empty task   # features/todo_list.feature:37
    Given the to-do list is empty             # features/steps/todo_list_steps.py:3
    When the user adds a task ""              # None
    Then an error message should be displayed # None

  Scenario: Trying to mark a non-existent task as completed    # features/todo_list.feature:42
    Given the to-do list contains tasks                        # features/steps/todo_list_steps.py:67
      | Task      | Status  |
      | Pay bills | Pending |
    When the user marks task "Buy groceries" as completed      # features/steps/todo_list_steps.py:36
    Then an error message "Task not found" should be displayed # features/steps/todo_list_steps.py:52
```

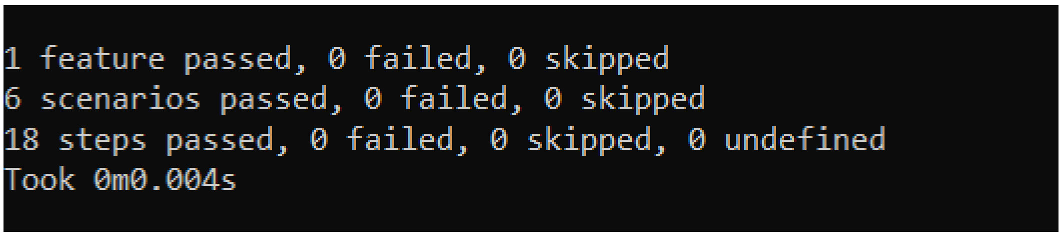


```
Failing scenarios:
  features/todo_list.feature:11  Listing all tasks
  features/todo_list.feature:37  Attempting to add an empty task

0 features passed, 1 failed, 0 skipped
4 scenarios passed, 2 failed, 0 skipped
15 steps passed, 0 failed, 1 skipped, 2 undefined
Took 0m0.003s

You can implement step definitions for undefined steps with these snippets:

@then(u'the output should contain')
def step_impl(context):
    raise NotImplementedError(u'STEP: Then the output should contain')


@when(u'the user adds a task ""')
def step_impl(context):
    raise NotImplementedError(u'STEP: When the user adds a task ""')
```

After corrections:



```
(venv) C:\Users\Admin\Documents\AcceptanceTesting>behave
Feature: To-Do List Management # features/todo_list.feature:1
  In order to manage tasks effectively,
  As a user,
  I want to add, list, and modify tasks in my to-do list.
  Scenario: Adding a task                              # features/todo_list.feature:6
    Given the to-do list is empty                      # features/steps/todo_list_steps.py:9
    When the user adds a task "Buy groceries"          # features/steps/todo_list_steps.py:26
    Then the to-do list should contain "Buy groceries" # features/steps/todo_list_steps.py:72

  Scenario: Listing all tasks          # features/todo_list.feature:11
    Given the to-do list contains tasks # features/steps/todo_list_steps.py:15
      | Task          |
      | Buy groceries |
      | Pay bills     |
    When the user lists all tasks       # features/steps/todo_list_steps.py:44
    Then the output should contain      # features/steps/todo_list_steps.py:78
      | Task          |
      | Buy groceries |
      | Pay bills     |

  Scenario: Marking a task as completed                               # features/todo_list.feature:22
    Given the to-do list contains tasks                               # features/steps/todo_list_steps.py:15
      | Task          | Status  |
      | Buy groceries | Pending |
    When the user marks task "Buy groceries" as completed             # features/steps/todo_list_steps.py:50
    Then the to-do list should show task "Buy groceries" as completed # features/steps/todo_list_steps.py:88

  Scenario: Clearing the entire to-do list  # features/todo_list.feature:29
    Given the to-do list contains tasks     # features/steps/todo_list_steps.py:15
      | Task          |
      | Buy groceries |
      | Pay bills     |
    When the user clears the to-do list     # features/steps/todo_list_steps.py:62
    Then the to-do list should be empty     # features/steps/todo_list_steps.py:94

  Scenario: Attempting to add an empty task  # features/todo_list.feature:37
    Given the to-do list is empty            # features/steps/todo_list_steps.py:9
    When the user adds a task ""             # features/steps/todo_list_steps.py:35
    Then an error message should be displayed # features/steps/todo_list_steps.py:99

  Scenario: Trying to mark a non-existent task as completed    # features/todo_list.feature:42
    Given the to-do list contains tasks                        # features/steps/todo_list_steps.py:15
      | Task      | Status  |
      | Pay bills | Pending |
    When the user marks task "Buy groceries" as completed      # features/steps/todo_list_steps.py:50
    Then an error message "Task not found" should be displayed # features/steps/todo_list_steps.py:105
```



```
1 feature passed, 0 failed, 0 skipped
6 scenarios passed, 0 failed, 0 skipped
18 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.004s
```

Scenarios

Scenario 1: Adding a Task

Description:
The user should be able to add a task to the to-do list.

Steps:

1. **Given**: The to-do list is empty.
2. **When**: The user adds a task "Buy groceries".
3. **Then**: The to-do list should contain "Buy groceries".

Expected Outcome:
The task "Buy groceries" is added to the to-do list.

Scenario 2: Listing All Tasks

Description:
The user should be able to view all tasks in the to-do list.

Steps:

1. **Given**: The to-do list contains tasks:
   - "Buy groceries"
   - "Pay bills"
2. **When**: The user lists all tasks.
3. **Then**: The output should contain:
   - "Buy groceries"
   - "Pay bills".

Expected Outcome:
The output matches the tasks in the to-do list.

Scenario 3: Marking a Task as Completed

Description:
The user should be able to mark a task as completed.

Steps:

1. **Given**: The to-do list contains tasks:
   - "Buy groceries" (Status: Pending).
2. **When**: The user marks the task "Buy groceries" as completed.
3. **Then**: The to-do list should show "Buy groceries" as completed.

Expected Outcome:
The task "Buy groceries" is marked as completed in the to-do list.

Scenario 4: Clearing the Entire To-Do List

Description:
The user should be able to clear all tasks from the to-do list.

Steps:

1. **Given**: The to-do list contains tasks:
   - "Buy groceries"
   - "Pay bills".
2. **When**: The user clears the to-do list.
3. **Then**: The to-do list should be empty.

Expected Outcome:
All tasks are removed from the to-do list.

Scenario 5: Attempting to Add an Empty Task

Description:
The system should handle the case when the user tries to add an empty task.

Steps:

1. **Given**: The to-do list is empty.
2. **When**: The user adds a task "".
3. **Then**: An error message "Task cannot be empty" should be displayed.

Expected Outcome:
An error message is displayed, and no task is added to the list.

Scenario 6: Trying to Mark a Non-Existent Task as Completed

Description:
The system should handle the case when the user tries to mark a task that does not exist as completed.

Steps:

1. **Given**: The to-do list contains tasks:
   - "Pay bills" (Status: Pending).
2. **When**: The user marks the task "Buy groceries" as completed.
3. **Then**: An error message "Task not found" should be displayed.

Expected Outcome:
An error message is displayed, and no changes are made to the list.

Conclusions and Recomendations:

Recommendations

1. Expand test coverage to include edge cases like duplicate tasks and long task names.
2. Automate test execution using CI tools (e.g., GitHub Actions).
3. Improve user feedback with detailed error and confirmation messages.
4. Refactor the code for modularity to simplify maintenance and future extensions.
5. Document the test process and train team members on BDD principles.

Conclusions

1. The project effectively implements BDD with Python and Behave, ensuring reliable task management functionality.
2. Automated tests provide robustness and ease of maintenance for the system.
3. The modular test structure supports scalability and future feature additions.
4. The approach ensures user expectations are met while laying a foundation for further development.