Platform Science Software Development Engineer in Test assignment

Author: Aarón Antonio Flores Patricio | antonfpat@gmail.com | +52 442.1.37.49.73

Content

API Testing Approach

The approach followed involves a combination of exploratory testing, manual testing, and automation testing using Codecept.js. The goal is to thoroughly validate the API's behavior, identify bugs, and ensure consistent and reliable results.

1. **Exploratory Testing:** To gain a comprehensive understanding of the Docker Roomba API service, I began with exploratory testing. This involved manually interacting with the API endpoint using Rapid API, a VS Code extension. Through exploratory testing, I aimed to observe the behavior of endpoint responses, understand the input requirements, and identify any anomalies or unexpected behaviors.
2. **Define Test Scenarios and Test Data:** Based on the insights gained from exploratory testing, I defined test scenarios that covered a wide range of positive, negative, and edge cases for each parameter of the API POST request.
3. **Manual Testing:** Using the defined test scenarios and test data, I executed manual testing to validate the Docker Roomba API service. This involved manually sending API requests with the specified inputs and verifying the response against the expected outcome. Through manual testing, I identified most of the bugs in the API service.
4. **Automation Testing:** Based on the results and observations from manual testing, I determine the best approach to structure the automation testing suite.
5. **Test Environment Setup:** Before writing the test scripts, I had to set up the test environment, including installing Codecept.js. I configured the necessary settings, such as specifying the API endpoint URL, headers details, and other relevant configurations.
6. **Test Script Development:** I wrote the test scripts using Codecept.js, leveraging its built-in capabilities for API testing and following the recommendations of the framework website for structuring the Features and Scenarios.
7. **Test Execution:** With the test scripts in place, the automated tests were executed using Codecept.js. The framework orchestrates the execution of the test cases, sends API requests, and compares the actual responses with the expected outcomes.

8. **Results Analysis:** Once the automated tests have completed, I analyzed the test results, comparing them with the results obtained from the manual testing phase. I examined any discrepancies or inconsistencies and investigated errors.

Tests Structure

The test structure is designed to focus on independent test cases for different parameters, including input data verification, connection status, and achievable paths. Additionally, the structure encompasses validation of response payloads and code status, ensuring the expected behavior of the API service.

- **Input Data Verification Tests:** Based on the insights gained from exploratory testing, the best approach was to design independent test cases for verifying the input data of the JSON payload. This includes the parameters roomSize, coords, patches, and instructions. Each test scenario is designed to exclusively test the potential problems associated with each parameter. These tests are organized into separate scripts within the tests folder to ensure modularity and ease of maintenance.
- **API Connection and Key Validation Tests:** In addition to input data verification, two other files were created to test the connection status and keys contained in the payload. The api_connection_test.js file focuses on testing the connectivity to the API service and verifying the presence of the essential keys in the response payload. This ensures that the API service is accessible, and the expected data structure is intact.
- **Achievable Paths and Integrity Testing:** Another file (paths_test.js) is dedicated to assessing achievable paths with no beyond frontiers problems. This test aims to determine the overall integrity of the API endpoint and verify if the final position and cleaned patches are correctly calculated. By simulating various valid scenarios and input combinations, the test ensures that the API service operates as expected and produces accurate results.

In all the test scripts, the focus is on validating the response payloads when valid data is provided. The expected responses are compared against the actual responses received from the API service. This validation ensures that the API service returns the expected data and maintains consistency. Furthermore, the code status is verified based on the expected behavior of the input data. For example, if the input data is invalid or violates specific constraints, the test scripts verify if the API service returns the appropriate error code (e.g., 400 Bad Request). This validation helps ensure that the API service handles invalid inputs correctly and provides meaningful error responses.

Assumptions for requirements

The following assumptions were considered to complement the information given in the requirements:

1. **roomSize:**
    - Even if the grid is very large, computations for the output should be possible.
    - Only positive values are permitted for X and Y coordinates.

- Trigger a 400 bad request error for the following conditions:
    - Negative or zero values for the dimensions of the room (any or both coordinates X-Y).

2. **coords:**
    - Only positive values are permitted for X and Y coordinates.
    - Trigger a 400 bad request error for the following conditions:
        - Negative or beyond the limits of the grid values for the coordinates (any or both coordinates X-Y).
    - Roomba should be capable of starting at the frontiers of the room, including the origin (0, 0).
3. **patches:**
    - Only positive values are permitted for X and Y coordinates.
    - Trigger a 400 bad request error for the following conditions:
        - Negative or beyond the limits of the grid values for the coordinates (any or both coordinates X-Y).
    - Patches can exist at the frontiers of the room, including the origin (0, 0).
4. **instructions:**
    - Trigger a 400 bad request error for the following conditions:
        - Weird characters or letters other than N-E-W-S.
        - Lowercase letters (even if they are the accepted ones).
        - Spaces within the instruction set.
        - Instructions sets that go beyond the grid.
    - The POST request can be done with empty instructions.
    - Roomba can hover on the frontiers of the grid with instructions.

Bugs Report

These bugs have been identified during the testing of the Docker Roomba API service. It is recommended to address these issues to ensure the API service aligns with the specified requirements and behaves as expected.

1. **Room Size:**
    - Negative values are accepted for X and Y coordinates, violating the requirement of only accepting positive values.

- Zero value is accepted for the X coordinate, while it should be prevented, indicating a validation problem for X coordinate.

2. **Coords:**

    - Negative value is accepted for the Y coordinate, while it should be prevented, indicating a validation problem for the Y coordinate.
    - Coordinates out of the grid for both X and Y are allowed, which is inconsistent with the requirements.
    - The X coordinate prevents the assignment of the right frontier value, for example, in a grid of 5x5, assigning 5 to X generates a bad request, violating the expected behavior.

3. **Patches:**

    - Negative value is accepted for the X coordinate, while it should be prevented, indicating a validation problem for the X coordinate.
    - Patches out of the grid for both X and Y are allowed, which is inconsistent with the requirement.
    - The Y coordinate prevents the assignment of the superior frontier value, for example, in a grid of 5x5, assigning 5 to Y generates a bad request, violating the expected behavior.
    - Even if no patches are provided, the output registers the Roomba as having hovered over 1 patch, which is incorrect.
    - Even if no patches are hovered over, the output registers the Roomba as having hovered over 1 patch, which is incorrect.

4. **Instructions:**

    - Going beyond the grid is currently allowed, without any verification of the frontiers, which violates the expected behavior.
    - The Roomba prevents hovering over the North, East, and West frontiers, ignoring the command. However, this behavior doesn't apply to the South frontier, allowing the Roomba to reach it, which is inconsistent.