Author: William Wang
Student ID: 261052682

ECSE 429 Project Part A: Report

Testing Sessions

A total of 2 Testing Sessions were performed, each with a duration of 45 minutes as instructed, during the following dates:

- Oct 5, 7:30 PM - 8:15 PM
- Oct 7, 8:00 PM - 8:45 PM

Deliverables Summary

Session Notes

Since two exploratory testing sessions were conducted, for each session, there is are corresponding session notes and summary. All of the session notes and summaries are compiled within a single document: **SessionNodes_PartA.xlsx** and each session notes and summaries are on their own separate pages.

Unit Test Suite

The Unit Test Suite for part A of the project is divided in two sections: **test_todo_module_basic.py** and **test_todo_module_id.py**. The design decision behind that is because the team is comprised of only one member and thus, as per the instructions, the main focus of the testing for the project are capabilities related to **todos**. Thus, the only two methods that are relevant to this project are **/todos** and **/todos/:id**.

Pytest was chosen as the open-source unit test tool and all the unit tests were written in Python. Additionally, the library **requests** was used to send requests to the rest API. Pytest plugin **Pytest-random_order** was also used for this project since the instructions required the capability of running the tests in random order.

It is important to note that for this project, the Unit Test Suite does not restore the system to the initial state due to a bug related to the API id counter not being able to reset. This bug will be mentioned later in the report, in the Unit Test Execution Findings section. However, the unit test suite restores the system the best it can to its initial state and consecutive re-runs of the unit test suite is possible without restarting the server.

For each test case, if relevant to the test scenario, a before state is printed out on the console and an after state is also printed out on the console for comparison. This allows us, the testers, to verify that the capability being tested is indeed behaving as expected. For some other tests, where the before state may not be relevant (such as requests that do no modify the todo list), the response status code from the API is printed out on the console instead to ensure that the request did get processed in an expected manner.

Test cases were also designed around the documentation provided by the API page, where the tests would either:

- Test whether the behavior of a request matches the one described in the documentation
- Test whether undocumented behavior return the proper status code from the API
- Test whether there are edge cases where the behavior of a request matches the one described in the documentation most of the time and in some situations, might fail.

Bug Summary Form

The Bug summary form was written in an Excel sheet file named **Bug_Report.xlsx**. The form contains a collection of all the bugs that were found during the exploratory testing. For each bug, the form outlines the description of the bug, instructions on how to reproduce the bug and impacts that the bug can bring to the user.

Unit Test Suite Video

The Unit Test Suite Video is saved in the project repository as a file called **EC-SE429_ProjectA_VIdeo.mp4**. The video showcases a tester starting up the server, showcasing the initial state of the API page. Then, the user proceeds to go into the testing environment and demonstrates how the test runs are done on the Visual Studio Code terminal, where the test cases are ran in random order. Then, once the tests are finished, the user showcases the output log on the console of each test case and their details. Finally, the user returns to the API page, refreshes it to demonstrate that the Unit Test Suite has indeed restored the system to its initial state before the tests. Any further details regarding the requirements and the initial setup to run the tests on one's local machine are outlined in the Readme.md file inside the Testing folder of the repository.

Written Report

This written report provides a description of all the deliverables for the project and describes the findings of unit test suite execution.

Exploratory Testing Findings

As mentioned above, each session were performed for a duration of 45 minutes
Each testing session was performed with the following charter:

Identify capabilities and areas of potential instability of the "rest API todo list manager".
Identify documented and undocumented "rest API todo list manager" capabilities.
For each capability create a script or small program to demonstrate the capability.
Exercise each capability identified with data typical to the intended use of the application.

Session 1

During session 1, the main focus of the exploratory testing was to test basic functionalities of each method type for the methods **/todos** and **/todos/:id** without considering many edge cases. Firstly, by following the API documentations, the **/todos** method types were initially tested to see if the method types that were allowed could properly send a request to the API Server with a valid body. Then, tests were conducted to verify if the API could properly return the unallowed status code 405 whenever a user/ tester tried to request a method type that was not listed in the documentation under **/ todos**. Then, tests were ran to see if the API could handle requests with bodies that had missing non-required properties, such as id, doneStatus. Some test scripts were also written to verify that the API could block requests that had invalid bodies, such as bodies with a missing title. By following those steps, for method **/todos**, the method types POST, DELETE, GET, HEAD were all tested with success. The exception being PUT, where a bug about the wrong status code returned from the API will be futher discussed in the Unit Test Suite Execution Findings section of the report.

Then, test scripts were written to verify if PUT and POST method types were working as intended per the API documentation in method **/todos/:id**. By peforming those tests, POST was confirmed to behave as intended but PUT behaved differently compared to its description in the documentation. This bug is also outlined in the test findings.

At the end of the session, the main concerns were that although PUT and POST at the same documentation description, they behaved differently and it is unclear whether the intended behavior is the one from the documentation or the one from the implementation. The test ideas that were established at the end revolved around further testing methods for **/todos/:id**, testing the API's behavior with malformed XML/JSON payloads and testing POST requests with invalid bodies.

Session 2

During the second session, the first step was to test malformed XML/JSON payloads in **/todos**. After conducting the tests, the API proved to handle them very well and that it would correctly return a status code of 400, indicating that the request was invalid. Then, the next step was to test POST requests with varying body formats. The tests conducted contained a body with missing title, a body without doneStatus and a body with a title specified as empty. From the 3 scenarios, the API properly blocked the bodies with missing and empty titles, as they are described as mandatory and non empty by the documentation. The API also correctly accepted the request with missing doneStatus and set its default value to false.

As for the tests for **/todos/:id**, the method type GET was first tested to ensure that it could not only retrieve the correct todo instance, given an id value, but it could also correctly identify when the id value provided is not present in the API's list and returns a not found status code 404. Then, PUT and DELETE were also tested to observe the API's behavior when the id speficied for PUT does not exist in the list and for DELETE, the API could properly delete all the existing nodes and reset its id value assignment counter to 1. Upon running the tests, the results showed that the API could indeed identify whether a given id value for a PUT request exists in its list and could return a not found error code 404 in that scenario. However, for the DELETE test, the results showed that while the API could delete all the existing todo instances without issue, it could not decrement nor reset its id assignment value counter. This bug is discussed in the test findings section.

After the testing session, the main concern was that if the API's internal id value counter could never decrease, is there any way for the testers to fully restore the system after testing, including resetting the counter value to its initial value? The future test ideas that came up were to perform additional tests with XML payloads and observe whether there are any discrepancies between XML and JSON payloads. Another testing idea was to find more test edge cases with varying request body formats to see if the API's implementation covers all the request body cases.

Source Code Repository

The source code repository for this project is uploaded as a GitHub repository, which is made accessible with the following link:

https://github.com/williamwang1382/ECSE429_Project_PartA/tree/main

The Source Code Repository contains not only the source code for all the testscripts used during the exploratory testing sessions, but it also contains all the session notes and the bug form documents as well. As explained above, the video demo of a user

running the tests can also be found in the repository. The video, bug form and session notes can all be found in the home directory of the repository while the test scripts are located inside the **Testing** folder

Unit Test Suite Execution Findings

One problematic bug that was discovered during the unit test suite execution is the fact that the API never decrements or resets its id counter. This mainly causes problems for the testing workflow for this project since this means that after the test suites have finished, the system can never be restored to its initial state since the counter value for the id will always be incremented. This meant that each test run would require restarting the server if the desired behavior of the unit test suite is to restore the system to its initial state completely after the tests are finished. However, as explained in the **Unit Test Suite** section, this does not affect the capability of the unit test suite of running multiple times consecutively. The impact that this bug has is mainly on the side of the tester, where the tester designing the test cases must take this into consideration and maintain the assumption that hard-coding an id value in a test case will not guarantee correct results on consecutive runs.

Another concerning bug that was found during exploration testing is the discrepancy between the behavior of PUT request for the method **/todos/:id** and the one described in the API documentation. Instead of updating the already existing todo instance on the Server, the request completely overwrites the previous instance with a brand new one. This means that even though the new body provided in the PUT request may not have modified some non-required fields such as relationships with other tasks or categories, the request is still going to erase all the relationship data that was previously stored in the todo instance. An additional point that makes this even more unusual is the fact that in the API documentation, for the method **/todos/:id**, PUT and POST request have the exact same documentation but the two requests display very different behaviors. This bug makes it difficult for users and testers to clearly understand the intended distinction between the two different requests and it is unclear whether the behavior described in the documentation is the intended one or it is rather the one displayed in its capabilities during usage.

An undocumented bug that was discovered during testing was that the undocumented method type PUT in **/todos** does not return the expected status code of 405. From the API, instead of returning a status code of 405, indicating that the PUT method is not allowed per the documentation, a JSONDecoderError is instead raised. This indicates that the unallowed PUT was not properly implemented by the API and that handling the erroneous use of PUT was neglected. This can affect both a user and a tester as it is creating misleading responses from the API and it can lead to misunderstandings on how the API is intended to work.