Group 20

Samy Sabir	261119166
Yonatan Bensimon	261104350

Table of Contents

Summary of Deliverables

The required deliverables for part A of this project are:

- ➢ Two Exploratory Testing Charter-Driven Sessions, including their session notes, a summary of findings, a list of concerns, and a list of new testing ideas.
    - Session1_Samy: exploratory testing of /todos endpoint
    - Session2_Yonatan: exploratory testing of /projects endpoint
- ➢ A complete Unit Test Suite of documented and undocumented APIs, designed with Pytest and composed of two modules:
    - projectEndpoint.py: /projects unit tests
    - todoEndpoint.py: /todos unit tests
- ➢ Bug summaries
- ➢ Unit test Suite video

Findings of Exploratory Testing

I. Session 1 (50 min): TODOS - Samy

The goal of this first session was to explore the capabilities of the /todos endpoint and their potential instabilities under certain circumstances. To demonstrate each capability, we utilized Postman to send HTTP requests documented in the application's local documentation. We also used swagger documentation to validate certain behaviors and uncover certain inconsistencies. Within this 50-minute period, we were only able to explore documented /todos and /todos/:id capabilities with the addition of one: /todos/:id/tasksof.

First, we tested different capabilities for the /todos method. For the most part, it can handle basic CRUD operations. Retrieving (GET) successfully returned all instances of todos in Postman. Creating (POST) also successfully created new todos, generated unique IDs for them, and listed them with all other todos. The OPTIONS method also successfully lists allowed operations in the header. See Session1_Samy.xlsx for input data and responses from Postman. The main issue with /todos is the unclear and sometimes undocumented behaviors of alternate and error flows. This happens, for example, when trying to create (POST) a new todo with the attribute "doneStatus" set to the string "false." The documentation specifies that this field takes a boolean value, but the local and swagger documentation does not specify that the POST method returns a specific message in case of invalid data types. There are multiple recurrences of these undocumented error flows, which could be listed somewhere instead of having a general description of errors. Additionally, there are unclear descriptions for certain operations. For instance, HEAD /todos' behavior is documented as a capability to retrieve headers for all the instances of todo. This brings confusion as to whether this operation should return multiple headers or one header for all todos. Furthermore, restricted operations listed in the swagger documentation (PUT, DELETE, PATCH) successfully returned 405 error codes in Postman.

Second, we explored different capabilities for the /todos/:id method. Yet again, it could handle creating, updating, retrieving, and deleting specific todos with their IDs. However, some behaviors were not as expected under certain conditions. When testing the PUT method type, we tried removing the "tasksof" relationship from todos to projects by putting an empty list for the attribute "tasksof" in our request body, which returned success (200 OK). When retrieving that modified todo, we observed that the "tasksof" was removed. On the other hand, the POST method type for that endpoint has the same description as the PUT method type in the swagger documentation, but it does

not behave the same way. In fact, when performing the same test, we noticed that the POST method type doesn't remove the relationship between the specific todo and its associated project. This is unwanted behavior, as a user could wrongly expect the POST method type to act the same way as the PUT method. Refer to Sess1_Bug_GET_Non_ExistentTodoID_tasksof.docx. Additionally, there were some issues with error message consistency between POST and PUT. When amending a non-existent todo ID with POST, we receive this error message: "No such todo entity instance with GUID or ID 420 found", while with PUT, we get: "Invalid GUID for 420 entity todo." This discrepancy brings confusion in debugging as one method type mentions both GUID and ID in its error message while the other only mentions GUID. Furthermore, OPTIONS and HEAD work as expected, but DELETE, on the other hand, unveils another issue. Swagger documentation declares that the DELETE method type deletes the specific instances of todo upon success but doesn't mention what the response body should contain. Therefore, we've discovered that nothing has been returned in the response body, which is not mentioned in the documentation.

Lastly, we had time to test retrieving the "tasksof" of a specific todos'id. This worked with an existing ID. It successfully (200 OK) returned the associated project items related to that todo in the request's response. However, we uncovered another bug when exploring the alternate flow of retrieving project items of non-existent todo IDs. The method type succeeds with response code 200 when requesting a non-existent ID, which is unwanted behavior. The swagger documentation doesn't mention any error code responses, which is an issue, but the fact that the method type succeeds and returns a body response of a project item is a bug. Sess1_Bug_Remove_tasksof_with_POST_todoID.

II. Session 2 (45 min): PROJECTS - Yonatan

The goal of the second session was to explore the capabilities and areas of potential instability of the /projects endpoint. Over the course of the session, we experimented with the functionalities of the different API relating to the project capability. Also, several areas of potential instability were found over the course of this session.

First, we ran a standard GET request on the /projects endpoint, which successfully returned all existing projects, as expected. Then, we ran that request again, but we also modified the header to request the response in XML, which successfully returned all existing projects in XML.

Next, we tried to run a POST request on /projects, with only a title as an attribute. While a project was indeed created, the title we had indicated was not included. This is when we found our first bug for the project's endpoint. It seemed like the API module did not

receive our title attribute. Once we found this bug, we tried to replicate it with other attributes, and, indeed, none of the attributes we tried worked.

Next, we tried to see what would happen if we ran a DELETE request on /projects. While this is undocumented, it is expected that it would delete all existing projects. Unfortunately, this returned a method not found, and did not delete any project.

We then moved on to the /projects/:id endpoint, where, once again, we first ran a standard GET request on an existing project, which worked great. We then tried to run a GET request on a project that did not exist, which gave an error code 404, as expected. We also tried a HEAD request on an existing project, which also worked fine. Before moving on to the next endpoint, we ran a PUT request, to try and modify the description of an existing project. Unfortunately, this did not work, as, once again, the API module did not seem to receive our attributes, and the API returned the project unmodified. It is important to note that this is a bug similar to the one we found earlier.

The next step was to analyze the /projects/:id/tasks endpoint. We immediately found a bug when we ran a GET request on a project that did not exist. Instead of returning an error code, the API returned tasks that were part of a different project, which was a major bug.

Finally, we attempted to see what would happen if we ran a GET request on /projects/:id/tasks/:id. While this is undocumented, logically, this would return the tasks with that specific id. Unfortunately, this returned a 404 error code.

All in all, this testing session was a success, as we found two major bugs, those being the nonreading of attributes and the return of tasks on a nonexistent project. This session allowed us to write pertinent unit tests to demonstrate those bugs.

Source Code Repository

We used GitHub to host our repository of unit tests and to collaborate on our respective test suites. The repository can be found on GitHub. Under our master directory “Part 1”, we’ve organized our repository into three working directories: /ApplicationBeingTested containing the .jar application, /SessionNotes containing our findings, and /UnitTestSuite containing the aforementioned two unit test modules.

Structure of Unit Test Suite

As mentioned in the previous section, our unit test suite is divided into two modules: the /todos endpoint unit tests and the /projects endpoint unit tests. In each, we’ve incorporated both successful JSON and XML format unit tests as well as multiple error flows. See Figure 1 for the directory tree. Each test method’s name clearly indicates what is being tested, but for clarity, we added textual descriptions that mention which Capability the unit test covers, which error case, if applicable, the unit test is required to correctly identify through a custom error message, and which unit tests are Bugs.

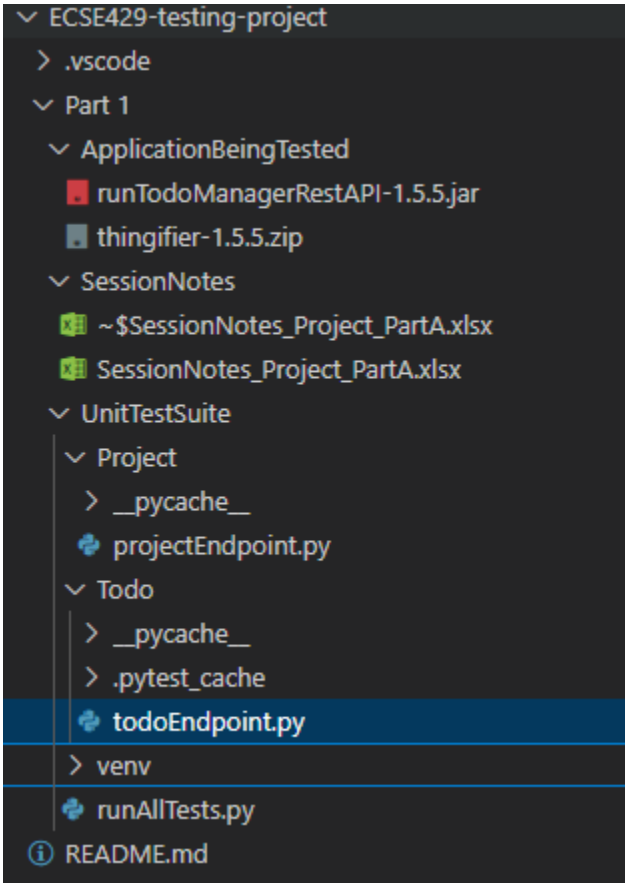

Figure 1. Project A Working Directory Structure

Since our unit test suite was designed with the Pytest framework, we've marked Bugs unit tests with mark.xfail() to ensure these test cases fail and can be addressed later. Refer to our Unit test video to see which test cases are marked XFAIL.

The requirements for running these tests are Python and the Pytest and pytest-randomly packages, which can both be installed using pip install. Pytest-randomly randomly shuffles the order of test items, which ensures that each test is independent. In that regard, a setupAndTeardown method makes sure that the application is running before each test and that it is reset after completing a test.

There are two ways to run our test suite. The first method is to run the command:

pytest <Path-to-test-module>

This method allows the execution of individual test modules. The second method is to run the command:

python <Path-to-run_tests.py>

The run_tests.py script allows the execution of the entire test suite.

Findings of Unit Test Suite Execution

Our unit test suite was designed to reflect our findings in our exploratory sessions. Contrary to our session findings, we added test cases that assert certain methods' correctly formed payloads in XML format. Since our exploratory testing already covered multiple error flows, we just had to reflect these findings in our unit test suite. Many error cases confirmed what was already mentioned in our exploratory testing findings regarding most issues. In fact, we've previously discussed how error messages and outcomes were not documented and, therefore, were unclear. Specifically, for the todos unit tests, certain methods like POST and PUT for the /todos/:id/tasksof endpoint produced different error messages for non-existent IDs, even if they are supposed to have the same behavior. This doesn't reflect best practices of unifying error messages, which is further supported by the results of unit test cases that produce the same error but with different error messages.

Regarding the project unit tests, the unit tests demonstrated each bug that was found during the exploratory session. More specifically, the unit tests demonstrated that the

API Manager did not process the different attributes that were sent in JSON format, which was the main bug we found during the second session of testing. Unit tests were also written to demonstrate the other bugs found during that session. All of our unit tests included the correct run-through of the API module, a test of invalid operations, a test for JSON/XML, and a test for malformed JSON/XML. Also, all of our tests verified if the returned status code was correct.

Lastly, it is important to note that a large portion of our Unit Test Suite is marked with XFAIL because of the bug regarding the API not processing the title attribute. Refer to Sess2_Bug_POST_Projects.docx for the bug summary discovered during /projects exploratory testing. Although this bug was uncovered in the context of creating projects, through our unit tests, it causes many tests to fail as we try to send body requests with changes to the title attribute. Therefore, this project-related bug rippled throughout both modules.