ECSE Software Validation Term Project

Part A Exploratory Testing of Rest API

Mihir Binay Kumar - 260794992
mihir.kumar@mail.mcgill.ca
Vassilios Exarhakos - 261051989
vassilios.exarhakos@mail.mcgill.ca

Summary of deliverables

TODO_charter.txt
Charter file for the exploratory testing session on the todos section of the API.

TODO_API_calls
Directory with screenshots of API calls during the exploratory testing session on the todos section

Projects_API_calls
Directory with screenshots of API calls during the exploratory testing session on the projects section

Exploratory_testing.mp4
Video of both exploratory testing sessions

Projects_charter.txt
Charter file for the exploratory testing session on the projects section of the API.

test_todo_unit.py
Unit testing module for the todos section of the API

test_project_unit.py
Unit testing module for the projects section of the API

random_testing.py
Python script which runs all unit tests in a randomized order

Bug_Report.txt
Bug report listing the bugs found across both testing sessions.

Unit_Test_Video.mp4
Video of all unit tests being run twice, both times in a randomized order.

runTodoManagerRestAPI-1.5.22.jar
API which is being tested

Findings of exploratory testing

Todos

For our exploratory testing, we used an API software called Insomnia Version - 8.3.0. We tested out all the API calls starting for TODOS except for the relationship ones like:

- /todos/:id/categories
- /todos/:id/categories/:id
- /todos/:id/task-of
- /todos/:id/task-of/:id

Therefore, for each API endpoint, we did a *Valid* and an *Invalid* testing and recorded our findings.

1. **/todos**
    a. **POST /todos**
        i. Valid: After sending in a valid JSON body, with all input attributes (doneStatus, description and title) and the various combinations of the attributes, we noticed the API returned a JSON body with all the attributes including the todo ID and a 201 HTTP status code (CREATED). The default value for doneStatus is False and for description is just an empty string.
        ii. Invalid: For an invalid JSON body, we sent in a POST request without the title attribute and in return got back a JSON body with an error message saying - “title: field is mandatory” and a 400 HTTP status code (BAD REQUEST).

    b. **GET /todos**
        i. Valid: No JSON body was required for this API call. Therefore after calling this API endpoint, we got back all the todo data points as a JSON body with all their attributes and also pre-existing ID 1 and ID 2 data points that already existed in the API

2. **/todos/:id**
    a. **POST /todos/:id**
        i. Valid: With this API endpoint we sent in various JSON bodies-
            1. with all input attributes individually,
            2. with random combinations of attributes and
            3. all attributes together.

        ii. Invalid: We sent in JSON bodies with IDs that we knew did not exist in the API system. Therefore, in return, we got an error message saying - “No such todo entity instance with GUID or ID 56 found” (The invalid ID we sent was 56) and 404 HTTP status code (NOT FOUND).

b. GET /todos/:id

i. Valid: We sent in a valid ID that we knew existed in the API and got back the very same todo data point we had asked for as a JSON body and a 200 HTTP status code (OK).

ii. Invalid: Similar to the Invalid section of the POST API endpoint, we sent an invalid ID and got an error message saying “No such todo entity instance with GUID or ID 89 found” (The invalid ID we sent was 89) also got a 404 HTTP status code (NOT FOUND)

c. PUT /todos/:id

i. Valid: The PUT HTTP call for this link had a BUG that we had discovered. We did the same testing for this API call as we had done for the POST API call since we were expecting the same result. However, when sending in a valid JSON body with just an individual or combination of attributes, the resulting JSON body would make the changes based on the input JSON body but also change the attributes not present inside the input JSON body back to their default values. For example, when we input a JSON body with just a new title for a valid ID, the resulting JSON body would have the new title, but the description would turn into an empty string and the doneStatus if originally True would turn into False. If the input JSON body did not have the title attribute, the output would send back an error message saying “title: field is mandatory”. ##BUG

ii. Invalid: We sent in JSON bodies with IDs that we knew did not exist in the API system. Therefore, in return, we got an error message saying - “No such todo entity instance with GUID or ID 56 found” (The invalid ID we sent was 56) and 404 HTTP status code (NOT FOUND).

d. DELETE /todos/:id

i. Valid: The final HTTP call test for the TODO functionality was the DELETE HTTP call. Therefore, we sent in the ID of a valid data point which resulted back with a 200 HTTP status code (OK). We then checked with the GET HTTP call for the same link and got shown an error message saying - “Could not find an instance with todos/5”. This meant that the DELETE functionality worked exactly as we had expected it.

ii. Invalid: We sent in JSON bodies with IDs that we knew did not exist in the API system. Therefore, in return we got an error message saying - “No such todo entity instance with GUID or ID 56 found” (The invalid ID we sent was 56) and 404 HTTP status code (NOT FOUND).

Project

For our exploratory testing on the projects section of the API, we once again used Insomnia. We tested all API calls which focused on projects except for those which involved interoperability such as:

- /projects/:id/tasks
- /projects/:id/tasks/:id
- /projects/:id/categories
- /projects/:id/categories/:id

We tested each API call using *Valid* and an *Invalid* input and recorded our findings.

3. **/projects**
    a. **POST /projects**
        i. Valid:
        The API correctly created a project and returned a 201 HTTP status for the following JSON bodies:
            1. All fields (Title: String, Description: String, Active: boolean, Completed: boolean) specified with correct typings
            2. Only Title specified with type String
            3. Only Description specified with type String
            4. Only Active specified with type boolean
            5. Only Completed specified with type boolean
        ii. Invalid:
        The API correctly raised an error and returned a 400 HTTP status for the following JSON bodies:
            1. Only Active specified with type String instead of boolean
            2. Only Completed specified with type String instead of boolean
            3. Only Completed specified with a value of null
            4. Only id specified with type int

    b. **GET /projects**
        i. Valid:
        The API correctly returned the JSON bodies of all the previously created project objects as well as a pre-existing project which was already in the database and also returned a 201 HTTP status. Since this call does not take in a JSON body, there is no way to test it with invalid input.

4. **/projects/:id**
    a. **POST /projects/:id**
        i. Valid:
        When specifying an id which had a project in it, The API correctly updated the project with the specified fields amended and returned a 201 HTTP status for the following JSON bodies:
            1. All fields (Title: String, Description: String, Active: boolean, Completed: boolean) specified with correct typings
            2. Only Title specified with type String

3. Only Description specified with type String
4. Only Active specified with type boolean
5. Only Completed specified with type boolean

ii. Invalid:
When specifying an id which had a project in it, the API did not update the project with the specified fields amended and instead correctly raised an error and returned a 400 HTTP status for the following JSON bodies:
1. Only Active specified with type String instead of boolean
2. Only Completed specified with type String instead of boolean
3. Only id specified with type int

When specifying an id which did not have a project in it, the API correctly raised an error and returned a 404 HTTP status.

b. GET /projects/:id

i. Valid:
When specifying an id with a project associated with it, the API returned the body of the project as well as a 200 HTTP status code.

ii. Invalid:
The API correctly raised an error and return a status code of 404 when the id which was requested:
1. Was a positive integer which did not have a project associated with it
2. Was a negative integer

c. PUT /projects/:id

i. Valid:
When specifying an id which had a project in it, the API correctly updated the project with the specified fields amended and returned a 201 HTTP status when all fields were specified.

When there was a field which was not specified, instead of keeping it as the same value it had previously, the PUT call restores the field to its default value. (“” for Strings, False for booleans) ##BUG

ii. Invalid:
When specifying an id which had a project associated with it, the API correctly raised an error and returned a 400 HTTP status when Completed and Active were Strings instead of booleans.

The API also raised an error and return a status code of 404 when the id which was requested:

1. Was an integer which did not have a project associated with it
2. Was text (i.e. /projects/eight)

d. DELETE /projects/:id

i. Valid:
When specifying an id which was associated with a project, the API returned a 200 HTTP status. After sending a get request to the same id, the API returned a 404 HTTP status, meaning that the project is no longer there and deletion was successful.

ii. Invalid:
The API correctly raised an error and returned a status code of 404 when the requested id was:
1. A positive integer with no project associated to it
2. A negative integer
3. Text

Description of the structure of the unit test suite

The unit test suite is divided into 2 Python files. The first file called TODO_Unit_Testing.py takes in all the unit tests for the TODO functionality whereas the second file called Project_Unit_Testing.py has all the unit tests for the Project functionality. Both files have similar setUp and tearDown methods. The setUpClass method gets the current working directory and runs the API and the tearDownClass method terminates the API. There is also a setUp method which runs before each test and clears all data points in the database so that each test can run with the system in the same state. Each file starts with tests assuring that the API correctly handles malformed JSON and XML input. After that, they test their namesake endpoints (/todos or /projects) for both endpoints (POST and GET) with valid as well as invalid unit tests. Furthermore, the second part of each file has the API testing for their respective /:id links (/todos/:id and /projects/:id) for all 4 endpoints (POST, GET, PUT and DELETE) with valid and invalid unit tests. For the bugs found with the PUT requests, they have two unit tests assigned to each buggy API call, one showing the expected behaviour failing and one showing the actual behaviour working.

Description of the source code repository

The source code repository link - https://github.com/MAKS-1211/ECSE429_PROJECT. The name of the repository is ECSE429_PROJECT. The repository consists of a directory called test which contains the 2 Python unit testing files, the JAR file that is processed by the unit test files, the Python file which performs the unit testing in a randomized order and the video of the unit tests being run twice in a randomized order. Another directory inside the repository called the charter_files stores all the 2 charter files for the exploratory testing we had done on the application as well as the video of the exploratory testing session and a folder containing all

screenshots of the API requests from the testing sessions. The bug_report directory contains the bug report file bug_report.txt which lists all bugs found during testing. Finally the report for the project is found at the root of the directory.

Findings of unit test suite execution

The unit test suite execution confirmed the findings of the exploratory testing sessions. The tests showed that the API correctly handles malformed XML and JSON input. The tests also confirm that the API correctly handles valid and invalid GET and POST calls to /todos and /projects. The API also correctly handles valid and invalid GET, POST and DELETE calls to /todos/:id and /projects/:id. The most interesting finding of the unit test suite was for the PUT calls to /projects/:id and/todos/:id where the expected behaviour of the API amending the specified fields and leaving the rest unchanged fails. Instead, the API resets any unspecified fields in the put request to their default values. For each instance this behaviour failed we included a test showing the expected behaviour failing and a test asserting the actual behaviour. This is why there are 7 tests which fail during the expectation of our unit test suite, as they are all the expected behaviour of the PUT request