ECSE 429: Software Validation

Project Part A: Written Report

Group 7

February 18th, 2025

Ryan McGregor – 260868511 – ryan.mcgregor@mail.mcgill.ca

Brenden Trudeau – 260941695 – brenden.trudeau@mail.mcgill.ca

Emmy Song - 261049871- emmy.song@mail.mcgill.ca

Summary of Deliverables

1. Deliverables from exploratory testing sessions

Three sessions were performed, one for each endpoint (todos, projects, and categories). Session notes were taken for each session, each contain:

a) The charter of the session
b) All observations and findings
c) Summary of key observations and findings
d) List of concerns
e) New testing ideas
f) Decisions made during the session
g) Screenshots of bugs found

2. Unit test suite

There is one unit testing script per endpoint, for a total of 3. Each one has at least:

a) One unit test per documented API
b) One unit test per undocumented API found during the exploratory sessions
c) Some unit tests covering both JSON and XML payloads
d) One unit test covering malformed payloads
e) One unit test covering invalid operations

Some of the unit tests will fail because of bugs in the API. These tests will show the bug as an assertion error with a description. There are also comments above these tests that show the expected behaviour.

3. Video

A screen recording that shows the execution of the testing scripts in our environment. The video also shows all the unit tests being executed in a random order.

4. Bug summaries

Each bug found during an exploratory testing session was written down in a bug_summary.txt file which shows:

a) A description of the bug
b) The impact of the bug
c) The steps to reproduce the bug

5. Written report

This document.

Findings of Exploratory Testing Sessions

During the exploratory testing of the REST API for the Todo List Manager, various endpoints were assessed to evaluate their functionality, consistency, and potential issues. The API was successfully launched, and all tested methods—including GET, HEAD, POST, PUT, and DELETE—functioned as expected. However, the OPTIONS method was permitted without proper documentation, indicating a need for clarification in the API reference. API calls were executed using Postman, providing a structured way to validate responses and interactions. While the API supports both JSON and XML output formats, the session for todos and projects capabilities focused on JSON, while the session for categories was completed with XML.

A notable observation was the similarity in behavior between the PUT and POST methods when updating existing records in all 3 endpoints. Further investigation is required to determine their precise differences, as the documentation does not explicitly outline them. Additionally, the lack of confirmation prompts when deleting tasks, categories, or projects poses a risk of unintended deletions and potential duplicate requests.

A major concern across the todos, projects, and categories endpoints is the lack of data persistence. The data is not retained between sessions, making the application impractical for long-term task management. This limitation affects usability, as users would lose their data upon restarting the application, requiring re-entry of all information. Addressing this issue by implementing persistent storage would significantly improve the system's reliability and practicality.

For the categories endpoint, attempts to retrieve projects (/categories/:id/projects) or todos (/categories/:id/todos) with an invalid ID revealed potential data integrity concerns.

Overall, the exploratory testing identified key functional capabilities of the API while highlighting areas where documentation and data handling require improvement. Addressing these concerns will enhance the clarity, reliability, and usability of the system.

Unit Test Suite Description

The Unit Test Suite for the REST API Todo List Manager was developed in Python using the PyTest framework. The suite is structured into three modules—todos, projects, and categories—corresponding to the key APIs identified during exploratory testing. This modular design ensures maintainability, scalability, and clear separation of concerns. Each module verifies documented behaviors, explores undocumented capabilities, and ensures that the API handles both JSON and XML payload formats without introducing unintended side effects. The tests were executed using the pytest -v --random-order command, with the pytest-random-order plugin ensuring test independence by randomizing execution order.

Todos Endpoint (test_todo_api.py)

The todos module extensively tests the functionality of the /todos endpoint, covering core operations and edge cases. Standard HTTP methods such as GET, HEAD, POST, PUT, and DELETE were tested to confirm proper API behavior. For instance, GET /todos and HEAD /todos verify successful retrieval of todos and appropriate response headers. The POST /todos tests cover various payload scenarios, including complete payloads, title-only payloads, and invalid requests missing mandatory fields. These tests ensure that the API returns appropriate HTTP status codes, such as 201 Created for successful operations and 400 Bad Request for invalid payloads. Additionally, the module explores undocumented behaviors, such as attempting a PUT /todos request without specifying an ID, which appropriately returns a 405 Method Not Allowed response. Tests also verify proper error handling for invalid ID operations, such as retrieving or deleting non-existent todos, which return 404 Not Found errors. Edge cases like consecutive deletion requests confirm that the API correctly handles repeated actions without unintended side effects. The todos module also includes tests for relationship management endpoints (/todos/:id/tasksof and /todos/:id/categories), verifying the creation and removal of associations between todos, projects, and categories. To ensure comprehensive format support, additional tests handle XML payloads, checking for correct processing of valid XML and appropriate rejection of malformed XML with 400 Bad Request responses.

Projects Endpoint (test_project_api.py)

The projects module thoroughly tests the /projects endpoint, covering both documented and undocumented behaviors while ensuring data integrity and proper payload handling. It validates essential HTTP methods (GET, HEAD, POST, PUT, and DELETE) for creating, retrieving, updating, and deleting projects. The suite begins by confirming endpoint accessibility and verifying successful retrieval of projects with appropriate headers. Retrieval tests confirm proper response structures and headers, while creation tests (POST /projects) validate the handling of complete and partial payloads, ensuring Boolean fields are stored as strings. Update operations (POST and PUT /projects/:id) confirm project details are amended correctly, and deletion tests (DELETE /projects/:id) ensure proper removal without affecting unrelated data. The module also addresses relationship management, testing the association and dissociation of tasks and categories with projects. Undocumented behaviors, such as PUT /projects/:id/tasks without a request body, were tested and returned the expected 405 Method Not Allowed response. The module concludes with XML payload handling, validating the API's ability to process XML-formatted data for retrieval and creation, including proper content-type handling and error detection for malformed inputs. Overall, this module ensures robust validation of the projects API, confirming expected behaviors, handling of edge cases, and reliable processing of both JSON and XML payloads.

Categories Endpoint (test_categories_api.py)

The categories module provides comprehensive coverage of the /categories endpoint, validating the core functionality, relationships, and payload handling. Key HTTP methods (GET, HEAD, POST, PUT, and DELETE) are tested to confirm proper category creation, retrieval, updating, and deletion. Relationship management between categories, projects, and todos is thoroughly examined to ensure correct associations and dissociations. The module includes tests that explore valid and invalid operations, such as retrieving or deleting non-existent categories, with proper 404 Not Found error handling. Edge cases are addressed by testing known bugs, including scenarios where invalid IDs return incorrect status codes. Undocumented behaviors are highlighted with clear assertions when the API's actual responses differ from expected outcomes. Additionally, XML payload handling is validated through tests for creating and retrieving categories in XML format, ensuring the API properly processes XML content and handles malformed requests with

appropriate 400 Bad Request responses. Overall, this module ensures reliable validation of the categories API by confirming expected behaviors, handling edge cases, and supporting both JSON and XML payload formats consistently and accurately.

Overall, the Unit Test Suite effectively validates the core functionalities of the Todos, Projects, and Categories endpoints. It ensures proper handling of JSON and XML payloads, accurate error responses, and data integrity across related endpoints. By covering both documented behaviors and edge cases, including known bugs and undocumented scenarios, the suite provides robust and reliable API validation. Its modular structure and consistent testing approach make it maintainable and adaptable for future enhancements, ensuring the API performs as expected under various c

Repository Description

All of our deliverables are stored in a repository on GitHub. It includes a README which describes the requirements, how to launch/shutdown the RESTful API, as well as how to run the tests. Additionally, the README breaks down the roles of all team members. The three testing scripts are in the folder pytest_rest_api and are labeled test_*endpoint*_api.py. The session notes are a single excel file with a different tab for each session.

Findings of Test Suite Execution

The test suite execution revealed that the API generally functions as intended, with all core HTTP methods (GET, HEAD, POST, PUT, DELETE) working correctly across the todos, projects, and categories endpoints. The unit tests were able to reproduce specific bugs identified during exploratory testing such as the incorrect 200 OK response for nonexistent category IDs in the /categories/:id/projects endpoint, highlighting potential data integrity concerns. Additionally, the behavior of PUT and POST methods was found to be similar when updating existing records, raising questions about their distinct purposes, as the API documentation does not clearly differentiate between them. This ambiguity could lead to misuse or confusion for developers integrating with the API.

A significant concern confirmed during testing is the lack of data persistence, as data is not retained between sessions. This limitation undermines the API's practicality for long-term task management, as users would need to re-enter all information after each restart. These findings, consistently reproduced in the test suite, emphasize the need for improved documentation, clearer method differentiation, and the implementation of persistent storage to enhance the API's reliability and usability. Addressing these issues would significantly improve the overall user experience and system robustness.