Name	Student ID	Email Address
Anastasiia Nemyrovska	261114550	anastasiia.nemyrovska@mail.mcgill.ca
Mohammed Elsayed	261053266	mohammed.elsayed2@mail.mcgill.ca

ECSE 429 SOFTWARE VALIDATION TERM PROJECT

PART A - EXPLORATORY TESTING OF REST API

1. SUMMARY OF DELIVERABLES

Firstly, we were tasked to perform exploratory testing for the “rest api todo list manager”. In two different sessions, performed by different testers, we learned several of its capabilities, potential instabilities, and found ideas for testing in the future. The sessions were performed using Postman. Complete session notes can be found here.

Secondly, we were tasked to design unit tests for the api. We decided to use JUnit as our testing framework. The test suite is divided into two files (one for Projects and one for Todos) and can be found here.

Thereafter, a video of the tests being executed in random order was taken and the video can be found here.

2. EXPLORATORY SESSION NOTES

Capabilities under examination: todos and projects.
The 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 Notes: Part 1 (todos)

The testing scripts, along with the screenshots, are all provided in the Session #1.pdf file, along with the summary of charter, session findings, concerns, and test ideas.

Files

Session #1.pdf

Session Findings

- OPTIONS:
    - Successfully validated documented options for both /todos and /todos/:id. The provided options appear to be exhaustive.
- GET:
    - Successfully retrieved todos by ID, title, description, and doneStatus.
    - Filtering todos by a word from the description does not work as expected. The full description is required for filtering (raises a question about filtering by inclusion).
    - Successfully retrieved todo data in both JSON and XML formats.

- HEAD:
    - Received expected headers in both JSON and XML formats for all tested requests.

- POST:
    - Handled valid input. It also correctly handled invalid input (missing title field, by raising an error and sending a 400 status. It also successfully created an instance of todo when a title field was provided.
    - Posted the object with the existing id, while modifying the fields that were passed. It also handled invalid input correctly. The API also correctly raised an error and sent a 404 status when given an id with no associated todos.
- PUT:
    - Replaced entire todo objects by specifying partial fields; fields not provided defaulted to initial values. This behavior is not documented.
    - Posting with nonexistent IDs correctly failed, as expected.
- DELETE
    - Successfully deleted todos by ID
    - Attempted retrieval after deletion correctly failed

Concerns

- Filtering by Description: Filtering todos by partial description is not supported. This raises the question of whether inclusion-based filtering should be implemented, as it would provide a more user friendly experience.
- POST/PUT Behavior: While attempting to POST or PUT with an existing or nonexistent ID behaves as expected, the behavior of PUT (completely replacing the

todo) is not documented, especially the resetting of unspecified fields to default values. This may cause unexpected data loss and should be documented or handled differently.
- Posting Empty Todos: The error message "title: field mandatory", indicates that the empty todo was not created because the title field is mandatory and missing. It seems that if the title field was not mandatory, the empty todo would have been created which would have been a bug.
- ID increases even though post fails (newly created ID will be 2 higher than previous object instead of 1). This may lead to data loss and loss of data integrity. App can be shutdown by anyone as it is a simple get request without any authentication or validation

New Testing Ideas

- Inclusion-Based Filtering: Implement tests to explore more flexible filtering methods, such as filtering by partial descriptions, to confirm if this feature can be supported or if it requires future development.
- PUT/POST Validation: Write test cases to explore different use cases for PUT and POST, focusing on documenting and clarifying the default behavior of unspecified fields. Consider testing PATCH as a more suitable alternative for partial updates.
- Test with IDs that are negative

Session Notes: Part 2 (projects)

The testing scripts, along with the screenshots, are all provided in the Session #2..xlsx file, along with the summary of charter, session findings, concerns, and test ideas.

Files

Session #2.xlsx

Session Findings

- OPTIONS:
    - Successfully validated documented options for both /projects and /projects/:id. The provided options appear to be exhaustive.
- GET:
    - Successfully retrieved projects by ID, title, active status, and completed status.
    - Filtering projects by a word from the description does not work as expected. The full description is required for filtering (raises a question about filtering by inclusion).
    - Successfully retrieved project data in both JSON and XML formats.

- HEAD:
    - Received expected headers in both JSON and XML formats for all tested requests.
- POST:
    - Posting an empty project raises questions about whether this should be allowed.
    - Attempting to post with an id correctly triggered a validation failure, preventing the operation.
    - Successfully posted projects with identical titles and descriptions using both JSON and XML, raising concerns about allowing duplicate projects.
    - Successfully posted projects with completed status and active status, even though this combination of states might not make sense.
- PUT:
    - Successfully replaced entire project objects by specifying partial fields; fields not provided defaulted to initial values. This behavior is not documented.
    - Posting with nonexistent IDs correctly failed, as expected.
- DELETE:
    - Successfully deleted projects by ID.
    - Attempted retrieval after deletion correctly failed.

Concerns

- Filtering by Description: Filtering projects by partial description is not supported. This raises the question of whether inclusion-based filtering should be implemented, as it would provide a more user-friendly experience.
- Duplicate Projects: Currently, the system allows duplicate project entries with identical titles and descriptions. There should be clarification on whether this behavior is intended or if duplicates should be prevented.
- Posting with Completed and Active Status: It is currently possible to create projects with conflicting states (i.e., both "completed" and "active"). This may not align with the intended business logic and should be further investigated.
- POST/PUT Behavior in /projects/:id: While attempting to POST or PUT with an existing or nonexistent ID behaves as expected, the behavior of PUT (completely replacing the project) is not documented, especially the resetting of unspecified fields to default values. This may cause unexpected data loss and should be documented or handled differently. Additionally, the task of amending existing projects makes more sense through the usage of PATCH request as opposed to POST, typically reserved for an instance creation.
- Posting Empty Projects: There is ambiguity about whether empty project creation should be allowed. The system accepts it, but this seems potentially problematic.

New Testing Ideas

- Inclusion-Based Filtering: Implement tests to explore more flexible filtering methods, such as filtering by partial descriptions, to confirm if this feature can be supported or if it requires future development.
- Duplicate Prevention: Create test cases to handle and prevent duplicate project titles and descriptions if necessary. Test different scenarios for project uniqueness validation.
- Status Logic Testing: Further explore the logical combination of project statuses. Test various combinations (e.g., completed but active) to validate whether such states are allowed by design or need refinement.
- PUT/POST Validation: Write test cases to explore different use cases for PUT and POST, focusing on documenting and clarifying the default behavior of unspecified fields. Consider testing PATCH as a more suitable alternative for partial updates.
- Empty Project Creation: Investigate the handling of empty projects by creating test cases that validate whether an empty project is useful or should be restricted.

3. UNIT TEST SUITE STRUCTURE

Unit tests can be found here.

1. **TodoUnitTest.java**: The `TodoUnitTest.java` file is focused on testing the CRUD operations (Create, Read, Update, Delete) for "Todo" instances. The tests are designed to verify correct functionality by ensuring that the API returns expected response codes and that the created, updated, and deleted objects behave as expected.
    - **Setup**: The `@BeforeAll` annotation is used to start the API server and ensure it is up and running before any tests are executed. This is followed by setting the base URI for all HTTP requests.
    - **Test Execution**:
        - The `@BeforeEach` method creates a new "Todo" object before each test, verifying that the object is correctly created with the expected fields (title, description, done status).
        - Various tests verify CRUD operations:
            - Creation of a "Todo" with valid inputs is checked, ensuring proper HTTP response codes and correct data in the response.
            - Update operations test whether modifying fields like title or description works as expected.
            - Deletion of a "Todo" ensures proper handling of removing existing entries. Additional tests check if trying to delete an

already deleted "Todo" or an invalid/non-existent ID results in the expected error responses.
    - **Error Handling**: Special attention is given to testing invalid inputs or operations (e.g., malformed input or deleting non-existent entries), ensuring that the application returns proper error codes and messages.
2. **ProjectUnitTest.java**: The `ProjectUnitTest.java` file focuses on managing "Project" objects. It follows a similar structure as the `TodoUnitTest.java`, ensuring proper setup, CRUD functionality, and error handling.
    - **Setup**: The API is launched before tests, and all requests are targeted at a predefined base URI.
    - **Test Execution**:
        - Similar to `TodoUnitTest.java`, the tests here ensure that projects are correctly created, read, updated, and deleted. The tests validate that return codes and application logic, such as field updates, work as expected.
        - These tests also check how the system behaves when given invalid input, such as trying to create a project with missing required fields or updating a project that does not exist.
    - **Error Handling**: The file includes tests that simulate cases where a non-existent project is updated or deleted, ensuring appropriate error messages are returned.

In both files, the structure is designed to ensure that each type of API object has its own set of specific tests, focusing on both functionality and handling of edge cases.

4. SOURCE CODE REPOSITORY DESCRIPTION

Repository can be found here.

The source code repository for this project is hosted on GitHub, allowing for version control, collaborative development, and code review workflows.
Repository Structure: The repository is structured to support organized development, testing, and documentation. Key directories include:

```
/idea                    // IDE-specific configuration files
/Exploratory Tests       // Documentation and session notes for exploratory testing
/src/test                // Unit test suite for the project
```

Exploratory Tests: Contains detailed documentation from various test sessions, including session notes in PDF, text, and spreadsheet formats.
src/test/unitTests: This directory houses the unit test files for the project, ensuring the functionality of critical components. For example:

ProjectUnitTest.java: Tests specific to project-related functionality.
TodoUnitTest.java: Tests that cover the "To-do" functionality.
.gitignore: Specifies files and directories to be excluded from version control.
README.md: Provides a general structure of the project.
Branching Strategy: Each different API is explored on a different branch to keep the work isolated from the master branch.
Merging and Rebasing: The team uses a combination of merges and rebases. Merging is typically performed when integrating feature branches into the main master branch, while rebasing may be used during development to keep branches up-to-date with the latest changes.
Commit Message Guidelines: To maintain clarity in the development process, the meaningful commit messages are used. Each commit describes the change concisely, helping developers and reviewers understand the purpose of the code modifications.
Collaboration and Code Review: Before any code is merged into the master branch, it undergoes a peer review process. This ensures that the code is up to the project's standards and helps catch potential issues before integration.

5. UNIT TEST SUITE EXECUTION FINDINGS

Bug summary (api)

Bug #1: Missing End Tag in API Docs Response

- Executive summary: The XML response from the API docs is malformed due to a missing end tag.
- Description of bug: The XML payload returned by the `/docs` endpoint contains a `<link>` element without a matching closing `</link>` tag, causing the XML parser to fail. This issue prevents the correct parsing and validation of the XML response, leading to errors in automated testing or when the payload is processed by XML parsers.
- Potential impact of bug on operation of system:
    - Automated systems relying on XML parsers will fail when processing this response.
    - Unit tests and any integration that expects valid XML will break.
    - It may prevent API consumers from programmatically reading documentation content, potentially affecting CI pipelines or other automated workflows.
- Steps to reproduce the bug:
    1. Send a GET request to `http://localhost:4567/docs`.
    2. Observe the XML response with the `<link>` element.
    3. Attempt to validate the XML using an XML parser.

4. Observe the error regarding the missing end tag for `<link>`.

Bug summary (todos)

Bug #2: Amend todo with PUT

- Executive summary: PUT request replaces the entire todo instance instead of amending it.
- Description of bug: The documentation states that PUT should amend an existing todo, but instead, it replaces the entire instance with specified fields and defaults unspecified fields, leading to potential data loss.
- Potential impact of bug on operation of system: Amendments to a todo may overwrite important data, leading to data loss and inconsistencies.
- Steps to reproduce the bug:
    1) Send a PUT request with some fields to amend a todo.
    2) Observe that instead of partial changes, the entire todo is replaced with the specified fields and default values for unspecified fields.

Bug #3: Amend todo with POST

- Executive summary: POST request is being used to amend todos instead of PATCH.
- Description of bug: POST is being used to amend existing todos, but this operation should ideally be done with PATCH, which is more suited for partial updates.
- Potential impact of bug on operation of system: Misuse of POST for amendments could lead to confusion and inconsistency in REST API standards.
- Steps to reproduce the bug:
    1) Send a POST request to amend an existing todo.
    2) Observe that POST is used for amending, though PATCH should be the correct method for this operation.

Bug summary (projects)

Bug #4: Posting empty projects

- Executive summary: Possible to post projects with no title or description.
- Description of bug: The API allows users to post projects with no title or description, which leads to incomplete data entries and makes it hard to distinguish or manage projects in the system.
- Potential impact of bug on operation of system: Incomplete data integrity, leading to confusion and reduced project traceability.
- Steps to reproduce the bug:
    1) Send a POST request with an empty title and description.
    2) Observe that the project is created with these empty fields.

Bug #5: Posting identical projects

- Executive summary: Possible to post projects with identical fields.
- Description of bug: The API allows users to post projects with identical fields (same title, description, etc.), resulting in duplicates in the system without any validation to prevent this.
- Potential impact of bug on operation of system: Duplicated project entries, causing clutter, confusion, and difficulty in tracking projects.
- Steps to reproduce the bug:
    1) Send two POST requests with identical project data (title, description, etc.).
    2) Observe that both projects are created without any validation errors.

Bug #6: Posting a completed project

- Executive summary: Possible to post a project marked as completed.
- Description of bug: Users can post a project with a status of "completed" right from creation, which could misrepresent the workflow and project lifecycle.
- Potential impact of bug on operation of system: Projects may appear as completed before any real progress is made, disrupting task management and reporting.
- Steps to reproduce the bug:
    1) Send a POST request for a project with status marked as "completed."
    2) Observe that the project is created with this status.

Bug #7: Posting a completed project with active status

- Executive summary: Possible to post a completed project with status marked as active.
- Description of bug: The API allows posting projects with conflicting statuses (marked as both completed and active), which leads to confusion over the project's actual state.
- Potential impact of bug on operation of system: Misleading project statuses, potentially causing errors in workflow or project management.
- Steps to reproduce the bug:
    1) Send a POST request for a project marked as "completed" but with the status set to "active."
    2) Observe the conflicting status in the response.

Bug #8: Amend projects with PUT

- Executive summary: PUT request replaces the entire project instead of amending it.
- Description of bug: The documentation states that PUT should amend an existing project, but instead, it replaces the entire instance with specified fields and defaults unspecified fields, leading to potential data loss.
- Potential impact of bug on operation of system: Amendments to a project may overwrite important data, leading to data loss and inconsistencies.

- Steps to reproduce the bug:
    3) Send a PUT request with some fields to amend a project.
    4) Observe that instead of partial changes, the entire project is replaced with the specified fields and default values for unspecified fields.

Bug #9: Amend project with POST

- Executive summary: POST request is being used to amend projects instead of PATCH.
- Description of bug: POST is being used to amend existing projects, but this operation should ideally be done with PATCH, which is more suited for partial updates.
- Potential impact of bug on operation of system: Misuse of POST for amendments could lead to confusion and inconsistency in REST API standards.
- Steps to reproduce the bug:
    3) Send a POST request to amend an existing project.
    4) Observe that POST is used for amending, though PATCH should be the correct method for this operation.

Bug #10: Returning projects based on description

- Executive summary: Projects are only returned based on full description match.
- Description of bug: The API only returns projects that match the full description provided in the query, making it hard for users to search for projects using partial descriptions.
- Potential impact of bug on operation of system: Reduced search usability, as users need to enter the exact full description to retrieve relevant projects, limiting flexibility.
- Steps to reproduce the bug:
    1) Send a search query filtering projects by partial description match.
    2) Observe that only projects with the full matching description are returned, not those containing partial matches.