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 B - STORY TESTING OF REST API

1. SUMMARY OF DELIVERABLES

Firstly, we were tasked to define 10 user stories related to using the API explored in Part A of the project. Our story tests were defined as scenario outline gherkin scripts in feature files. A total of 10 feature files can be found here.

Secondly, we used cucumber to parse, automate and execute our gherkin scripts. We implemented step definitions which control the todo list manager rest api studies in Part A of the project. The step definition 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.

1.1 User Stories

1. As a user, I want to retrieve all projects so that I can view all current project instances.
2. As a user, I want to filter projects by specific attributes so that I can find projects that meet specific criteria.
3. As a user, I want to create a new project without providing an ID so that I can quickly add a new project with the required details only.
4. As a user, I want to update a specific project by ID so that I can modify project details to keep the information accurate and current.
5. As a user, I want to delete a project by ID so that I can remove projects that are no longer relevant or necessary.
6. As a user, I want to create a new todo task so that I can have the task saved in the list of todos for future reference.
7. As a user, I want to retrieve a specific todo task so that I can view its details.
8. As a user, I want to get all todo tasks so that I can retrieve their details.

9. As a user, I want to be able to delete a todo task so that it no longer shows up in the todo list.
10. As a user, I want to update an existing todo task so that I can modify its details

2. STORY TEST SUITE STRUCTURE

The test suite for the **Todo** functionality is designed to:

1. **Initialize and manage the REST API service** required for testing.
2. **Define steps that match Cucumber scenarios** and perform HTTP requests.
3. **Handle different test flows** (normal, alternate, and error flows).
4. **Reuse step definitions** across feature files.
5. **Provide cleanup procedures** to ensure proper shutdown of the service.

Breakdown of Key Components

2. 1 Service Initialization and Management

- **@Given("the service is running")**: This step ensures that the REST API service is running before executing tests. It starts the service if it's not already running.
- **Helper Methods**:
    - isServiceRunning(): Checks if the service is up by sending a request.
    - startService(): Starts the service using a specified command to run the API server.
    - @AfterAll shutdownServer(): A cleanup method to shut down the service after all tests complete, ensuring no leftover processes run post-tests.

These steps and methods are **shared across all feature files** for the **Todo** functionality, as well as potentially other services (e.g., ProjectStepDefinition). This central setup ensures consistency and avoids the need to duplicate service management code in each feature file.

2.2 Common Background Setup

- **@Given("the following todos exist in the system:")**: This step checks if specific todos exist, typically used in background steps to ensure a known initial state for the scenarios. The DataTable input allows the specification of initial todos with predefined attributes (e.g., title, doneStatus).

This step ensures that todos required for a scenario are present in the system, making it possible to perform actions on them or verify their properties.

2.3 Scenario Steps for CRUD Operations

Each scenario in the feature files has corresponding @When and @Then methods in the step definition class. These methods handle CRUD operations like creating, updating, retrieving, and deleting todos.

The suite organizes these CRUD operation steps to support flexible scenario creation across multiple feature files (e.g., CreateTodo.feature, UpdateTodo.feature), making it easy to add and modify tests without redundancy.

2.4 Flow Control: Normal, Alternate, and Error Flows

- The structure uses separate methods to manage **normal**, **alternate**, and **error** flows within each feature.
    - **Normal Flow**: Methods like iSendAPostRequestWithTitleAndDescription() define steps for successful operations, such as creating a todo with all required details.
    - **Alternate Flow**: Methods such as iSendAPostRequestWithTitleOnly() handle scenarios where optional fields (like the description) are missing.
    - **Error Flow**: Methods like theResponseShouldContainErrorMessage() verify that the API returns expected error messages when required fields (like title) are missing.

This approach enables the suite to fully test different behaviors and edge cases within the same feature while keeping the codebase maintainable.

2.5 Reusable Step Definitions and Consistency

- **Shared Steps**: Steps like "the service is running", "the following todos exist in the system", and "I should receive a response status code of XXX" are used across different feature files, enhancing consistency and reducing duplication.
- **Modular Structure**: Methods are organized by feature and functionality, with clear naming conventions. For instance, methods specific to creating todos are grouped, while those related to updating or deleting have their sections, making the test suite easy to read and maintain.

Additionally, similar patterns and structure are employed in **Project Step Definitions** (ProjectStepDefinition), which handle scenarios related to managing projects rather than todos. This ensures that the test suite follows a consistent structure across different functionality, enhancing maintainability and scalability.

2.6 Assertions and Validation

- The suite uses **assertions** (e.g., assertEquals, assertNotNull, assertNull) to verify expected behavior in each step:
    - **Status Code Verification**: iShouldReceiveResponseStatusCode() confirms that the API returns the expected HTTP status code for each request.
    - **Content Verification**: theResponseShouldHaveTodoWithTitleAndDescription() and theResponseShouldContainErrorMessage() ensure that the API response content meets the specified requirements.
- **Error Handling**: Error flows are handled gracefully with specific methods and assertions, verifying that appropriate error messages are returned when input data is invalid.

3. SOURCE CODE REPOSITORY DESCRIPTION

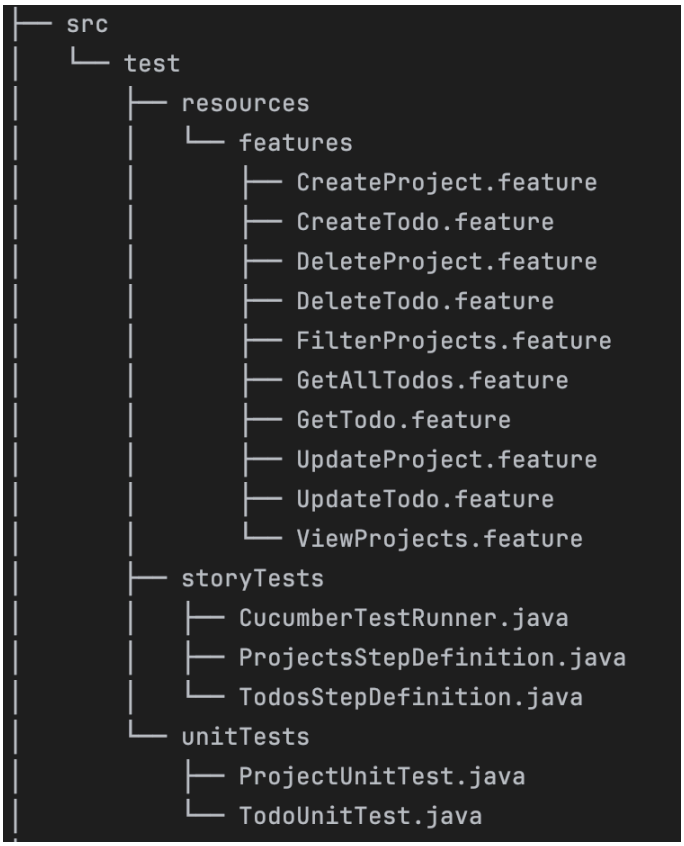

Figure 1. Source Code Repository Structure

Feature files:

The 10 feature files are listed under src/test/resources/features. Every feature file represents a different user story (mentioned in section 1.1), with each file containing multiple scenarios written in Gherkin syntax to test specific behaviors.

StoryTests Directory:

CucumberTestRunner.java: This class is the main entry point for running all Cucumber tests. It is configured to find and execute all the feature files in the features folder. It is annotated with Cucumber and JUnit annotations to integrate Cucumber tests with JUnit. It shuffles all feature files into a random order using a fixed seed for reproducibility. The test runner sets up Cucumber arguments, including glue code location and output plugins, and uses Main.main() to run the tests.

ProjectsStepDefinition.java: This file contains step definitions for the Project-related feature files (CreateProject.feature, DeleteProject.feature, FilterProjects.feature, and UpdateProject.feature, ViewProjects). Each method in this file would map to specific steps in the project-related feature files, providing the underlying logic for each step in the Gherkin scenarios.

TodosStepDefinition.java: This file contains step definitions for the Todo-related feature files (CreateTodo.feature, DeleteTodo.feature, GetAllTodos.feature, GetTodo.feature, and UpdateTodo.feature). Each method in this file would implement the logic corresponding to the steps in the to-do related scenarios.

4. STORY TEST SUITE EXECUTION FINDINGS

4.1 Test Coverage and Feature Verification

The story test suite provides comprehensive coverage of the primary features of the 'Todo Manager REST API.' The features tested include:

- **CreateTodo/CreateProject:** Verifying that todos and projects can be successfully created with specified fields.
- **GetTodo/GetProject:** Validating retrieval operations for todos and projects, including filtering and ensuring expected entities are returned.
- **UpdateTodo/UpdateProject:** Ensuring updates to existing todos and projects are processed correctly with proper status code verification.
- **DeleteTodo/DeleteProject:** Confirming the successful deletion of todos and projects, including cleanup operations to maintain data integrity.

The suite also tests error handling scenarios, such as responses for invalid or missing data, to ensure robust system behavior.

4.2 Additional Bug Summary Considerations

Bug #1: Filtering Projects With an Invalid Parameter

- *Executive Summary*: GET request with invalid filter parameter returns unfiltered results without warning.
- *Description of Bug*: When performing a GET request to /projects with an invalid query parameter (e.g., ?invalidParam=3), the API returns all projects instead of filtering or issuing a warning/error message about the unrecognized parameter. This could mislead users into thinking their request successfully applied the intended filter.
- Potential Impact:
    - Users may misinterpret the results, believing data has been filtered when it is not the case.
    - This behavior compromises the accuracy and reliability of the API for users relying on query parameters for data filtering.
    - Potential issues for applications that depend on precise data queries and may lead to incorrect data analysis.
- Steps to Reproduce the Bug:
    - Send a GET request to http://localhost:4567/projects?invalidParam=3.
    - Observe the response, which returns the full list of projects.
    - Note the absence of any warning or error message indicating the parameter invalid is not recognized.

Bug #2: Filtering Todos With an Invalid Parameter

- *Executive Summary*: GET request with invalid filter parameter returns unfiltered results without warning.
- *Description of Bug*: When performing a GET request to /todos with an invalid query parameter (e.g., ?booleanParam=false), the API returns all todos instead of filtering or issuing a warning/error message about the unrecognized parameter. This could mislead users into thinking their request successfully applied the intended filter.
- Potential Impact:
    - Users may misinterpret the results, believing data has been filtered when it is not the case.
    - This behavior compromises the accuracy and reliability of the API for users relying on query parameters for data filtering.
    - Potential issues for applications that depend on precise data queries and may lead to incorrect data analysis.

- Steps to Reproduce the Bug:
    - Send a GET request to http://localhost:4567/todos?booleanParam=false.
    - Observe the response, which returns the full list of todos.
    - Note the absence of any warning or error message indicating the parameter invalid is not recognized.