Project B Report - Group 1

Software Validation (ECSE 429)

Abhijeet Praveen (260985492) – abhijeet.praveen@mail.mcgill.ca
Abhigyan Praveen (261047297) – abhigyan.praveen@mail.mcgill.ca
Sebastien Cantin (260979759) – sebastien.cantin@mail.mcgill.ca
Rooshnie Velautham (260985875) – rooshnie.velautham@mail.mcgill.ca

Department of Electrical, Computer and Software Engineering

Contents

1 Summary of Deliverables

For Project B, our team was tasked with developing a comprehensive suite of automated tests for our API, focusing on 20 user stories with an emphasis on the following acceptance test types: Normal flow, Alternate flow, and Error flow.

1.1 User Stories

Here are the 20 user stories our team decided to work with:

- As a user, I want to categorize a todo to better manage and prioritize my tasks.
- As an organizer, I want to create a new category to classify my projects and todos.
- As a project manager, I want to create a new project so that I can organize related tasks under it.
- As a user, I want to add a task to a project, so that I can remember to complete it during the project.
- As a user, I wish to create a new todo in the system so that I can have all my todos in one place.
- As a user, I want to delete a category that is no longer needed to simplify my categorization system.
- As a user, I want to delete a project that is no longer needed to keep my workspace clean.
- As a user, I want to delete a todo from my list so that I can keep the list up-to-date with only relevant tasks.
- As a project manager, I want to view all the categories associated with a specific project, so I can understand the project's diverse areas of focus.
- As a user, I want to view the category a todo is associated with to understand its relevance to different areas.
- As a user, I want to view details of a specific project to understand its scope and status.
- As a user, I want to get all projects under a specific category to see related projects.
- As a user, I want to get a todo that has a certain category so that I can focus on important tasks first.

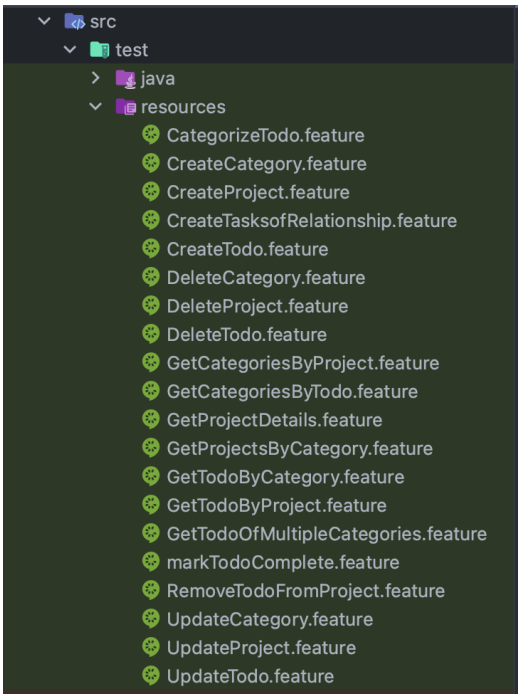
- As a user, I want to get a todo associated with a specific project to understand what needs to be done.
- As a user, I want to view a todo linked to multiple categories, so that I can understand how different tasks overlap across various aspects of my life.
- As a user, I want to mark a todo as complete so that I can track my progress.
- As a user, I want to remove a todo from a project when it's no longer relevant to that project.
- As a user, I want to update the details of a category to correct or modify its information.
- As a team leader, I want to update a project's details to reflect changes in scope or priorities.
- As a user, I wish to update the details of a todo in the system to fix and update any mistakes I may have made earlier.

These user stories were then put into 20 different feature files. Thereafter, the team utilized the Cucumber tool for parsing and executing the Gherkin scripts. Every feature file for a user story had its set of step definitions in a file with the java file having the same name as the feature file. A randomizer was also implemented so that the tests can in a random order, thereby highlighting the independence of each test with the others.

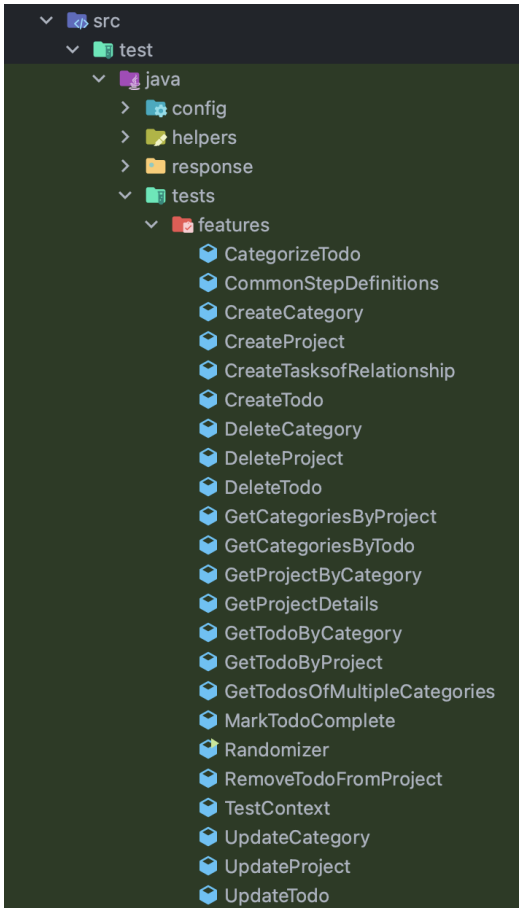
A video of the tests being executed in random order was also taken and the video can be found here

2 Source Code Repository Structure

Our source code repository is structured as follows:



(a) All the feature files under the **src/test/resources** directory



(b) All the step definition code under the **src/test/java/tests/features** directory

Figure 1: Source Code Repository Structure

Every feature file represents a different user story, one of the 20 that were established. There were many step definitions across the feature files that were common. These were added to one java file in order for minimal code repetition. This is implemented in the class called **CommonStepDefinitions**. A randomizer in the class called **Randomizer** runs the tests in a random order. Furthermore, data needed to be shared across different step definitions, this is implemented in **TestContext**. It holds all the data that is needed during the testing and provides that data to all the java files that require it.

3 Structure of Story Tests Suite

The story test suite is organized into multiple files corresponding to the defined user stories, ensuring each test can be run independently and in any order.

For example, the image below is the feature file for the **RemoveTodoFromProject** user story.

```
Feature: Remove Todo from Project
  As a user, I want to remove a todo from a project when it's no longer relevant to that project.

  Background:
    Given the API server is running and available

    Given the following projects exist in the system:
      | projectTitle       | projectDescription | completed | active |
      | Marketing Campaign | New product ads    | false     | true   |

    Given the following todos exist in the system:
      | todoTitle       | todoDescription  | todoDoneStatus |
      | Design Brochure | Pamphlets        | false          |
      | Plan Event      | Book venue       | false          |
      | New Design      | Website homepage | false          |

    Given the following project and todo association exist in the system:
      | projectTitle       | todoTitle       |
      | Marketing Campaign | Design Brochure |
      | Marketing Campaign | Plan Event      |

  Scenario Outline: Normal Flow - Unlink a todo from a project
    When a user attempts to remove the todo "<todoTitle>" from the project "<projectTitle>"
    Then the todo "<todoTitle>" should no longer be linked to the project "<projectTitle>"

    Examples:
      | todoTitle       | projectTitle       |
      | Design Brochure | Marketing Campaign |
      | Plan Event      | Marketing Campaign |

  Scenario Outline: Alternate Flow - Unlink a todo that was not linked to the project
    When a user attempts to remove the todo "<todoTitle>" from the project "<projectTitle>"
    Then the status code returned by the API is "<statusCode>"

    Examples:
      | todoTitle   | projectTitle       | statusCode |
      | New Design  | Marketing Campaign | 404        |

  Scenario Outline: Error Flow - Try to unlink a todo from a non-existent project
    When a user attempts to remove the todo "<todoTitle>" from the non-existent project "<projectTitle>"
    And the status code returned by the API is "<statusCode>"

    Examples:
      | todoTitle       | projectTitle    | statusCode |
      | Design Brochure | Unknown Project | 400        |

  Scenario: Teardown
    Then the system is restored to the original state
```

Figure 2: Gherkin Script for RemoveTodoFromProject

Every Gherkin script follows the same format, where the background section completes the setup that is required for the test. The background checks that the API server is running and available to be tested, this test happens before every scenario is tested. The step definition implementation for this found in **CommonStepDefinitions** as this happens for all feature files. Similarly, any entities that need to be created in the background section are also created in that same file so that the setup ensures a smooth test execution. Afterwards each feature file has three scenarios, a Normal flow, an Alternate Flow and an Error Flow. The implementations of which will be found in the java file dedicated to that Gherkin script. Thereafter, a dedicated teardown mechanism is implemented to maintain system state post-test execution for every scenario, the implementation of which can be found in **CommonStepDefinitions**.

The following is an implementation of the step definition of a **@When** from the **RemoveTodoFromProject**. This is when the action that is required in the step definition is performed.

```java
public class RemoveTodoFromProject {                                                    ⚠ 6 ↑

    11 usages
    private final TestContext testContext;

    /**
     * Attempts to remove a todo from a project.
     *
     * @param todoTitle The title of the todo to be removed.
     * @param projectTitle The title of the project from which the todo is to be removed.
     * @throws IOException if an I/O error occurs during the HTTP request.
     */
    2 usages  abhijeetpraveen
    @When("a user attempts to remove the todo {string} from the project {string}")
    public void aUserAttemptsToRemoveTheTodoFromTheProject(String todoTitle, String projectTitle) throws IOException {
        CloseableHttpClient httpClient = testContext.get( key: "httpClient", CloseableHttpClient.class);
        HashMap<String, Todo> createdTodos = testContext.get( key: "createdTodos", HashMap.class);
        HashMap<String, Project> createdProjects = testContext.get( key: "createdProjects", HashMap.class);

        String todoID = createdTodos.get(todoTitle).getId();
        String projectID = createdProjects.get(projectTitle).getId();

        HttpResponse response = deleteAssociation( associationType: "tasks", projectID, todoID, httpClient);
        testContext.set("statusCode", response.getStatusLine().getStatusCode());
    }
```

Figure 3: Implementation of @When from RemoveTodoFromProject

As seen, a TestContext object is used to retrieve the data that is required to run the test smoothly, allowing the specified action to be performed.

Furthermore, the following is an implementation of the step definition of a **@Then** from the **RemoveTodoFromProject**. This is when the action performed in the **@When** is confirmed to have been executed correctly.

```java
/**
 * Validates that a todo is no longer linked to a project.
 *
 * @param todoTitle The title of the todo.
 * @param projectTitle The title of the project.
 * @throws IOException if an I/O error occurs during the HTTP request.
 */
1 usage  abhijeetpraveen
@Then("the todo {string} should no longer be linked to the project {string}")
public void theTodoShouldNoLongerBeLinkedToTheProject(String todoTitle, String projectTitle) throws IOException {
    CloseableHttpClient httpClient = testContext.get( key: "httpClient", CloseableHttpClient.class);
    HashMap<String, Project> createdProjects = testContext.get( key: "createdProjects", HashMap.class);
    HashMap<String, Todo> createdTodos = testContext.get( key: "createdTodos", HashMap.class);

    String projectID = createdProjects.get(projectTitle).getId();

    HttpResponse response = getAssociation( associationType: "tasks", projectID, httpClient);
    TodoResponse todoResponse = deserialize(response, TodoResponse.class);

    List<String> associatedTodoIds = todoResponse.getTodos().stream() Stream<Todo>
            .map(Todo::getId) Stream<String>
            .toList();

    String todoID = createdTodos.get(todoTitle).getId();

    assertFalse( message: "The Todo should not be linked to the Project anymore", associatedTodoIds.contains(todoID));
}
```

Figure 4: Implementation of @Then from RemoveTodoFromProject

As seen again, a TestContext object is used to retrieve the data that is required to confirm that the action was performed correctly.

4 Findings of Story Tests Suite Execution

The execution of the story test suite highlighted the robustness of the API's core functionalities while also revealing areas requiring improvement. The bugs that were found during Project A were encountered again. However, another bug was found where a project can be created with a blank title successfully without any errors. During our story test execution, when a project with a blank title was created, an error message was being expected to be returned from the API. However, instead of an error message, a project entity having a

blank title was returned. This was witnessed when the following error was thrown in java:

```
Unrecognized field "id" (class response.ResponseError)

not marked as ignorable (one known property: "errorMessages"]) at

[Source: (String)
"{"id":"21","title":"","completed":"false","active":"true","description":"description"}"
```

As seen, a project was returned having a blank title, while the expected return type was an error response with an error message.

This bug from the API was confirmed using Postman as well since creating a project with a blank title returned a successful request as seen in the image below.

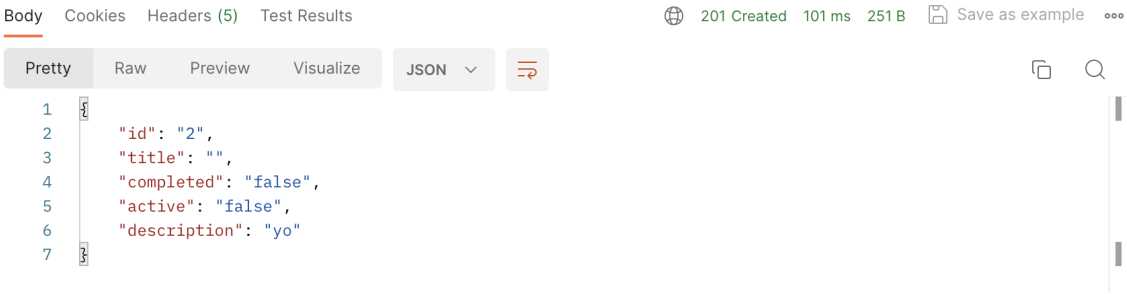


Figure 5: Project with Blank Title