ECSE Software Validation Term Project

Part B Story Testing of Rest API

Mihir Binay Kumar - 260794992

mihir.kumar@mail.mcgill.ca

Vassilios Exarhakos - 261051989

vassilios.exarhakos@mail.mcgill.ca

Summary of deliverables

1. TODO

   Amend_todo_list_doneStatus.feature

   It contains the Scenario Outlines for a user who wants to change the done status of their todo items already registered in the API.

   Amend_todo_list_doneStatus.py

   To confirm the validity of the outlines in the Amend_todo_list_doneStatus.feature file, we made assertions for not only the changed doneStatus but also for Titles, description and Todo ID. Each Scenario had at least 2 different todo items amended accordingly.

   Amend_todo_list_title.feature

   It contains the Scenario Outlines for a user who wants to change the title of their todo items already registered in the API.

   Amend_todo_list_title.py

   To confirm the validity of the outlines in the Amend_todo_list_title.feature file, we made assertions for not only the changed titles but also for the done status, description and Todo ID. Each Scenario had at least 2 different todo items amended accordingly.

   Create_new_todo_list_description.feature

   It contains various Scenario Outlines for a user who wants to create a new todo item with a specific description in their mind.

   Create_new_todo_list_description.py

   To confirm the validity of the outlines in the Create_new_todo_list_description.feature file, we made assertions for not only the new description but also the expected done status and title. Each Scenario had at least 2 different todo items created accordingly.

   Create_new_todo_list_item_doneStatus.feature

   It contains various Scenario Outlines for a user who wants to create a new todo item with a specific doneStatus in their mind.

Create_new_todo_list_item_doneStatus.py

To confirm the validity of the outlines in the Create_new_todo_list_item doneStatus.feature file, we made assertions for not only the new doneStatus but also the expected description and title. Each Scenario had at least 2 different todo items created accordingly.

Get_todo_item.feature

It contains various Scenario Outlines for a user who wants to get a todo item based on their known todo ID or just get all of them together.

Get_todo_item.py

To confirm the validity of the outlines in the Get_todo_item.feature file, we made assertions for the todo item description, done status, Title, ID and the number of items returned by the API if necessary.

2. Project

Complete_project.feature

It contains the Scenario Outlines for a user who has completed a project and wants to show that in the database using API.

Complete_project.py

It contains the step definitions for Complete_project.feature.

Get_project.feature

It contains the Scenario Outlines for a user who wants to retrieve a project from the database using the API.

Get_project.py

It contains the step definitions for Get_project.feature.

Post_existing_project.py

It contains the Scenario Outlines for a user who wants to amend an existing project in the database using a POST call.

Post_existing_project.feature

It contains the step definitions for Post_existing_project.feature.

Put_existing_project.feature

It contains the Scenario Outlines for a user who wants to amend an existing project in the database using a PUT call.

Put_existing_project.py

It contains the step definitions for Post_new_project.feature.

Post_new_project.feature

It contains the Scenario Outlines for a user who wants to create a new project in the database using the API.

Post_new_project.py

It contains the step definitions for Post_new_project.feature.

3. Other

Random_testing.mp4

Video showing the test suite being executed in a randomized order

Random.py

Python Script that executes test suite in random order

bug_report.txt

Summary of all bugs found throughout parts A and B of the project.

runTodoManagerRestAPI-1.5.22.jar

Application being tested

Description of the structure of the story test suite

Projects

For the story testing of the projects section of the API, the testing focused on five user stories, with three acceptance tests defined for each story :

Background: RestAPI is running and has no projects in it

Before working on the restAPI runTodoManager, we must ensure that the API's JAR file runs and the projects list contains no entries.

1. **The user wants to post a new project to the database**
    a. **Normal Flow:** The user wants to Post a valid project with all fields declared
    This test sends a Post request to /projects with a body that declares all fields. It then checks that a project was sent back with all the same fields in response and that the same project exists in the database.
    b. **Alternate Flow:** The user wants to Post a project with no fields declared
    This test sends a Post request to /projects with a body that declares no fields. It then checks that a project was sent back with all fields at their default values in response and that the same project exists in the database.
    c. **Error Flow:** The user wants to Post a project with an ID declared
    This test sends a Post request to /projects with a body that declares an id. It then checks that an error was raised in response.

2. **The user wants to retrieve an existing project using Get**
    a. **Normal Flow:** <u>The user wants to retrieve an existing project by using its id</u>
    This test creates a project in the database and then sends a Get request to the id of that project. It then checks that all fields were returned correctly.
    b. **Alternate Flow:** <u>The user wants to retrieve an existing project by getting all</u> projects and looking through them
    This test creates a project in the database and then sends a Get request to /projects. It then checks in the list of projects returned for that project and that all fields were returned correctly.
    c. **Error Flow:** <u>The user wants to get a project by ID from an empty database</u>
    This test sends a get request to an id with no project associated to it and checks that an error was raised in response,
3. **The user wants to amend an existing project using a POST call**
    a. **Normal Flow:** <u>The user wants to amend all fields of an existing project in the database using a POST call</u>
    This test creates a project in the database and then sends a Post request to amend all fields of the project. It then checks that the project was correctly sent back in the response and that the project had its fields updated in the database.
    b. **Alternate Flow:** <u>The user wants to amend no fields of an existing project in the database using a POST call</u>
    This test creates a project in the database and then sends a Post request with no fields declared in the body. It then checks that the project was sent back in the response and that the project had none of its fields changed in the database.
    c. **Error Flow:** <u>The user wants to amend the ID of an existing project in the database using a POST call</u>
    This test creates a project in the database and then sends a Post request with an id declared in the body. It then checks that an error was raised.
4. **The user wants to amend an existing project using a PUT call**
    a. **Normal Flow:** <u>The user wants to amend all fields of an existing project in the database using a PUT call</u>
    This test creates a project in the database and then sends a Put request to amend all fields of the project. It then checks that the project was correctly sent back in the response and that the project had its fields updated in the database.

b. **Alternate Flow:** The user wants to amend no fields of an existing project in the database using a PUT call
This test creates a project in the database and then sends a Put request with no fields declared in the body. It then checks that the project was sent back in the response and that the project had none of its fields changed in the database.
c. **Error Flow:** The user wants to amend the ID of an existing project in the database using a PUT call
This test creates a project in the database and then sends a Put request with an id declared in the body. It then checks that an error was raised.

5. **The user has finished a project and wants the database to show that**
   a. **Normal Flow:** The user wants to mark the project as done in the database
   This test creates a project in the database and then uses a Post call to update the active field to False and the completed field to True. It then checks that the project had its fields updated in the database.
   b. **Alternate Flow:** The user wants to delete the project from the database entirely
   This test creates a project in the database and then uses a Delete call to delete the project from the database. It then checks that the project no longer exists in the database.
   c. **Error Flow:** The user wants to mark a project that has already been deleted as done
   This test creates a project in the database and then uses a Delete call to delete the project from the database. It then tries to update the completed and active fields of the project and checks that an error was raised.

Todo

For the story testing of the todo section of the API, the testing focused on five user stories, with three acceptance tests defined for each story :

Background: RestAPI is running and available

Before working on the restAPI runTodoManager, we must ensure that the API's JAR file runs and the todo list contains todo items that we as a user know and can access by making HTTP-based calls.

1. **As a user, I want to update one of my todo lists with a new done Status**

a) <u>**Normal Flow Scenario:**</u> **As a user, I want to update a todo item with a new done status True**
   As a user, we send in a JSON body to the POST API call '/todos/id' with just a new doneStatus as True for both a todo item with a previous doneStatus as False and another todo item with a doneStatus as True. Based on that, we assert the new doneStatus and check if the other attributes remain the same from the JSON body returned from the restAPI.

b) <u>**Alternative Flow Scenario:**</u> **As a user, I want to update a todo item with a new done status False and description**
   As a user, we send in a JSON body to the POST API call '/todos/id' with a new doneStatus as False for both a todo item with a previous doneStatus as False and another todo item with a doneStatus as True and also change the description of the todo item registered inside the restAPI. Based on that, we assert the new doneStatus and check if the other attributes remain the same from the JSON body returned from the restAPI.

c) <u>**Error Flow Scenario:**</u> **As a user, I want to update a todo item with a new done Status as a non-boolean Status**
   As a user, we send in a JSON body to the POST API call '/todos/id' with just a new doneStatus as 'InProgress' and 'Almost' (Non-Boolean Variables) for both, a todo item with a previous doneStatus as False and another todo item with a doneStatus as True. Based on that, we assert an error message from the rest API saying "Failed Validation: doneStatus should be BOOLEAN".

2. **As a user, I want to change the title of one of my todo list items**

a) <u>**Normal Flow Scenario:**</u> **As a user, I want to update a todo item with a new title**
   As a user, we send in a JSON body to the POST API call '/todos/id' with just a new title. Based on that, we assert the new title and check if the other attributes remain the same from the JSON body returned from the restAPI.

b) <u>**Alternative Flow Scenario:**</u> **As a user, I want to update a todo item with a new title and description.**

As a user, we send in a JSON body to the POST API call '/todos/id' with a new title and a new description. Based on that, we assert the new title and description and check if the other attributes remain the same from the JSON body returned from the restAPI.

c) **<u>Error Flow Scenario:</u> As a user, I want to update a todo item with a blank title**
As a user, we send in a JSON body to the POST API call '/todos/id' with a blank title. Based on that, we assert an error message from the rest API saying "Failed Validation: title: can not be empty".

3. **As a user, I want to create a new todo list item with a specific description in mind**

a) **<u>Normal Flow Scenario</u>: As a user, I want to create a new todo item with a description, title and done Status**
As a user, we send in a JSON body to the POST API call '/todos' with a new title, description and doneStatus. Based on that, we assert all attributes with the attributes sent in from the API call, from the JSON body returned.

b) **<u>Alternative Flow Scenario</u>: As a user, I want to create a new todo item with a description and title**
As a user, we send in a JSON body to the POST API call '/todos' with a new title and description. Based on that, we assert all attributes with the attributes sent in from the API call and assert if the default doneStatus is false from the JSON body returned.

c) **<u>Error Flow Scenario</u>: As a user, I want to create a new todo item with just a description**
As a user, we send in a JSON body to the POST API call '/todos' with a new description. Based on that, we assert an error message from the rest API saying "title: field is mandatory".

4. **As a user, I want to create a new todo item with a specific done status in mind**

a) **<u>Normal Flow Scenario</u>: As a user, I want to create a new todo item with a True done status**

As a user, we send in a JSON body to the POST API call '/todos' with a new title and True doneStatus. Based on that, we assert all attributes with the attributes sent in from the API call, from the JSON body returned.

b) **<u>Alternative Flow Scenario</u>: As a user, I want to create a new todo item with a False done status**

As a user, we send in a JSON body to the POST API call '/todos' with a new title and False doneStatus. Based on that, we assert all attributes with the attributes sent in from the API call, from the JSON body returned.

c) **<u>Error Flow Scenario</u>: As a user, I want to create a new todo item with a done status but without a title**

As a user, we send in a JSON body to the POST API call '/todos' with a new doneStatus. Based on that, we assert an error message from the rest API saying "title: field is mandatory".

5. **As a user, I want to check on one of my todo items**

a) **<u>Normal Flow Scenario</u>: As a user, I want to get a specific single todo item through their integer todo ID**

As a user, we send in a JSON body to the GET API call '/todos/id'. Based on that, we assert all attributes with the attributes initially sent in from the POST API call in the Background section

b) **<u>Alternative Flow Scenario</u>: As a user, I want to find my todo item from a list of all todo items**

As a user, we send in a JSON body to the GET API call '/todos'. Based on that, we assert all attributes and all todo items with the attributes initially sent in from the POST API call in the Background section

c) **<u>Error Flow Scenario</u>: As a user, I want to get a specific single todo item through their string todo ID**

As a user, we send in a JSON body to the GET API call '/todos/id' however the ID is sent in as a String instead of an integer. Based on that, we assert an error message from the rest API saying "Could not find an instance with todos/{str(id)}".

Description of the source code repository

The repository consists of the report file, a video of the story tests being run in a randomized order named Random_testing.mp4, a directory named bug_report and a directory named test. The bug_report directory contains a summary of all bugs found since the beginning of the project, titled bug_report.txt. The test directory contains two subdirectories : TODOs and Projects. Each subdirectory contains the feature and step files for story testing each subsection, the application being tested called runTodoManagerRestAPI-1.5.22 and a Python script random.py which runs the tests in a randomized order.

Findings of story test suite execution

The test suite allowed us to understand the API on a more user level. The test suite execution had two failures, both associated with the behaviour of the PUT call. By using a PUT call, the user technically creates a completely new data point instead of updating their already existing data point. This can cause confusion and loss of data for users which can be considered to be a huge disadvantage of using this API. Aside from that bug, which we had already found during part A of the project, the test suite found no new bugs.