429 Project Report Summaries - Part 2

Names of testers

Ying Fei Zang (261054638) ying.zang@mail.mcgill.ca

YunShan Nong (261055472) yunshan.nong@mail.mcgill.ca

Yu Tong Hu (261051311) yu.tong.hu@mail.mcgill.ca

Summarizes deliverables

Inside `StepDefinitions.java`:

- The tests are in order of `Todos`, `Categories`, and `Projects`.
- Inside each scenario, the first action of the user always starts with “Given User connects to TodoManager” in order to set the connection. Therefore, only one @Given tag is used in the entire `StepDefinitions` file.
- There is an `@After(@notLast)` method called `closeConnection()` that runs after the execution of each scenario, which closes the HttpURLConnection.
- Every step definition can be run successfully without any errors or failures. However, a connection error, which is unrelated to our tests execution, would appear at the end:

```
//restore initial state upon completion
@AfterAll
public static void tearDown() throws URISyntaxException, IOException, ConnectException {
    URI shutdownUri = new URI(baseURL + "shutdown");
    URL shutdownUrl = shutdownUri.toURL();
    HttpURLConnection shutdownConnection = (HttpURLConnection) shutdownUrl.openConnection();
    shutdownConnection.setRequestMethod(method:"GET");
    int responseCode = shutdownConnection.getResponseCode();
    shutdownConnection.disconnect();
    connection.disconnect();
}
```

```
Scenario: Delete a todo that does not exist     # project2/TodosStories2.feature:148
  Given User connects to TodoManager            # project2.StepDefinitions.setUp()
  When User tries to delete a non-existing todo # project2.StepDefinitions.DeleteNonExistingTodo()
  Then Manager sends back a not found error     # project2.StepDefinitions.managerSendsBackNotFoundError()
      java.net.ConnectException: Connection refused: connect
        at java.base/sun.nio.ch.Net.connect0(Native Method)
        at java.base/sun.nio.ch.Net.connect(Net.java:589)
        at java.base/sun.nio.ch.Net.connect(Net.java:578)
        at java.base/sun.nio.ch.NioSocketImpl.connect(NioSocketImpl.java:583)
        at java.base/java.net.Socket.connect(Socket.java:751)
        at java.base/java.net.Socket.connect(Socket.java:686)
        at java.base/sun.net.NetworkClient.doConnect(NetworkClient.java:183)
        at java.base/sun.net.www.http.HttpClient.openServer(HttpClient.java:531)
        at java.base/sun.net.www.http.HttpClient.openServer(HttpClient.java:636)
        at java.base/sun.net.www.http.HttpClient.parseHTTP(HttpClient.java:782)
        at java.base/sun.net.www.protocol.http.HttpURLConnection.getInputStream0(HttpURLConnection.java:1690)
        at java.base/sun.net.www.protocol.http.HttpURLConnection.getInputStream(HttpURLConnection.java:1599)
        at java.base/java.net.HttpURLConnection.getResponseCode(HttpURLConnection.java:531)
        at project2.StepDefinitions.tearDown(StepDefinitions.java:52)
```

By having the ConnectException: Connection refused: connect, we know that we have

successfully shut down the API. Without this error, we cannot properly disconnect the API server.

- In order to restore the system to the initial state upon completion, we used GET /shutdown API endpoints to disconnect all the connections to the server. By placing it in the `@AfterAll` tag, the server is shut down only after all the scenarios are executed.
- All our tests (scenarios) can run in any order since they are independent from each other. In order for our tests to run in a random order, we chose to configure <runOrder> parameters of the org.apache.maven.plugins:maven-surefire-plugin

This is what we added to our `pom.xml` file:

```xml
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>3.0.0-M8</version>
    <configuration>
        <runOrder>random</runOrder>
    </configuration>
</plugin>
```

This is the printed info when we run the tests with `mvn -test` command BEFORE adding this <configuration>:

```
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ group8 ---
[INFO] skip non existing resourceDirectory C:\Users\yunsh\Documents\ECSE429\group8\src\main\resources
[INFO]
[INFO] --- compiler:3.10.1:compile (default-compile) @ group8 ---
[INFO] No sources to compile
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ group8 ---
[INFO] Copying 3 resources from src\test\resources to target\test-classes
[INFO]
[INFO] --- compiler:3.10.1:testCompile (default-testCompile) @ group8 ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 2 source files to C:\Users\yunsh\Documents\ECSE429\group8\target\test-classes
[INFO]
[INFO] --- surefire:3.0.0-M8:test (default-test) @ group8 ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
```

After adding the configuration in `pom.xml` file, we get a [INFP] line from maven.surefile saying that Test will run in random order.

```
[INFO] --- resources:3.3.1:resources (default-resources) @ group8 ---
[INFO] skip non existing resourceDirectory C:\Users\yunsh\Documents\ECSE429\group8\src\main\resources
[INFO]
[INFO] --- compiler:3.10.1:compile (default-compile) @ group8 ---
[INFO] No sources to compile
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ group8 ---
[INFO] Copying 3 resources from src\test\resources to target\test-classes
[INFO]
[INFO] --- compiler:3.10.1:testCompile (default-testCompile) @ group8 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- surefire:3.0.0-M8:test (default-test) @ group8 ---
[INFO] Tests will run in random order. To reproduce ordering use flag -Dsurefire.runOrder.random.seed=3187947551828900
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
```

However, once we examined the tests more closely, we observed that tests were always run following the order of the scenarios in `.feature` files. Therefore, we created 3 new `.feature` files in which we manually mixed the scenario order of the original `.feature` files. This means each entity (`Todos, Categories, or Projects`) has 2 files: `XXXXXStories.feature` and `XXXXXStories2.feature`. We would refer to `XXXXXStories.feature` to explain the rest of the report.

`Feature: User Interacting with Todos`

I created user stories related to Todos inside `TodosStories.feature`:

- 5 different user stories related to Todos using Gherkin style
- Each story has 3 scenarios (3 different flow): normal, alternate, and error.
    - User story #1 is about GET and PUT with /todos
    - User story #2 is about retrieving a todo using GET /todos/id and POST /todos
    - User story #3 is about GET, POST and DELETE /todos/id
    - User story #4 explores the relation between todos and categories using GET and POST todos/:id/categories
    - User story #5 is explores the relation between todos and projects related to todos/:id/tasksof
- The alternate flow is a modified version of the normal flow, usually contains more steps.
- The error flow is a scenario that expects a failing request (ex: response code to be 404). However, for the sake of smooth running of the cucumber test, I did not throw actual Exception when an error occurred. Instead, I predict the failing in the scenario or I print out the unexpected behaviour to the console with an explanatory message
- All scenarios can be run successfully without any errors or failures
- Scenarios can be runned in any order because they are independent from each other
- Throughout the testing, I found out 3 unexpected behaviours of the todoManager, details of my findings are written inside the last section of this report

---
`Feature: User Interacting with Projects`

Inside `ProjectsStories.feature:`

- 5 different user stories related to Projects using Gherkin style
- Each story has 3 scenarios that correspond to normal, alternate and error flow:
    - Story #1 explores GET projects/
    - Story #2 explores POST projects/
    - Story #3 combines POST projects/, PUT projects/:id and DELETE projects/:id
    - Story #4 explores POST projects/:id/tasks and DELETE projects/:id/tasks/:id
    - Story #5 explores HEAD method
- All stories are written in the format of: “As a user I want to complete some operation to get some value.” They are written inside the .feature file.

`Feature: User Interacting with Categories`

Inside `CategoryStories.feature:`

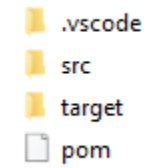
- 5 different user stories related to Categories using Gherkin style
- The first scenario only uses GET as it is the basic command, the 15 other scenarios are more complex.
- Each story has 3 scenarios that correspond to normal, alternate and error flow:
    - Story 1 uses POST categories/ in json and xml
    - Story 2 uses GET and POST together in categories/
    - Story 3 uses GET, DELETE and PUT in categories/:id
    - Story 4 uses GET and PUT and POST in categories/:id/todos
    - Story 5 uses GET and HEAD in categories/:id/projects
- The alternative flow has few steps more than normal flow, because it takes a longer path to reach the result through alternative flow.
- There was one unexpected bug and one expected bug during the testing, they will be explained further in the later part.

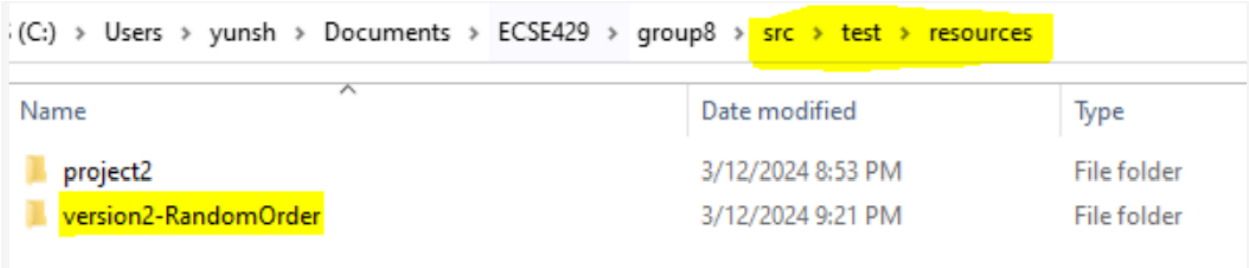
Describes source code repository

—----------

Our story test suite is located inside a folder called `group8`

Our code is written using java inside Visual Studio Code. Our test suite is written using Cucumber, so our project directory is created with the `cucumber-archetype` Maven plugin. It should contain these folders and file.





Inside `version2-RandomOrder` folder, we have all the `XXXXStories2.feature` files mentioned in the above **Summarizes deliverables.**

Inside `group8\src\test\java\project2` folder, we can find `RunCucumberTest.java` and `StepDefinitions.java`, where we implemented the step definitions which control the TodoListManager rest api

Under `group8\src\test\resources\project2` directory
We separated our different types of user stories into 3 `.feature` files

- `TodoStories.feature`
- `CategoryStories.feature`
- `ProjectsStories.feature`

In order to run our story test suite, go inside the `group 8` folder and type `mvn test` in the terminal.

To run our story test suite in a different order (our random approach), replace `XXXXStories.feature` files under `group8\src\test\resources\project2` directory with `XXXXStories2.feature` files under `group8\src\test\resources\version2-RandomOrder` directory.

Describes structure of story test suite

—----------

Inside `StepDefinitions.java` file, there are six global variables, defined at the top of the file, that facilitate the testing.

```
    // Global variables
private static HttpURLConnection connection;
private static String baseURL = "http://localhost:4567/";

// Storing returned values for later testing
private String returnedTodoId;
private String returnedCategoryId;
private String projectIDToBeDeleted;
private String taskIDToBeDeleted;
```

Inside `TodosStories.feature`, we can find:

- 5 different user stories related to Todos
- 16 different scenarios for user stories in total
- each test is marked with one of the three types of flow (Normal, Alternate, Error)
- For each story, there is a normal flow, followed by an alternate flow, and then an error flow at the end.
- All the functions that are used for the 5 user stories are inside `StepDefinitions.java`, at the beginning of the file up to line 446

Inside `CategoriesStories.feature`, we can find:

- 5 user stories related to Categories/
- 16 Scenarios in total (1 basic scenario, and 3 flows for each stories)
- All the functions that are used for the 5 user stories are inside `StepDefinitions.java`, starting at line 448 up to line 860.

Inside `ProjectsStories.feature`, we can find:

- Same structure as `TodosStories.feature`, with 15 different scenarios instead of 16.
- All the functions that are used for the 5 user stories are inside `StepDefinitions.java`, starting at line 860 of the file up to the end of file.

In total, we have 15 user stories, 48 scenarios.

Describe findings of story test suite execution

—----------

For `Todos` (tested by YunShan):

I discovered that when we add a new category to a non-existing todo (invalid ID) using POST /todos/non-existing-id/categories, the server does not return any error code or message, instead we get 200 as response code. Also, I noticed that the category is actually not added to any of the existing todos. I tested this strange behaviour in one of my scenarios (`Retrieve categories related to a non-existing todo`).

On the other hand, when I add a new project to a non-existing todo (invalid ID) using POST /todos/non-existing-id/categories, the server does return the expected error code and error message, telling me that it is NOT FOUND.

Another thing I found out in part B is that posting a todos (POST /todos), we do not need to specify doneStatus, there is no error message. A default value of False would be associated to the created todos automatically. This works for both JSON and XML input. The response code is also normal (201). I noticed this detail when I was trying to write a scenario where User specify their input using JSON:

```
When User posts a new todo with JSON:
        | title | description |
        | Watch 409 recording | watch thursday recording |
```

When I add `| doneStatus | false |`, the server do not accept the input, because this way, false is not a `boolean`, but a `String`. But I cannot specify the type inside `TodosStories.feature` file, so I tried to take off `doneStatus`, and I noticed the request also worked, the todos got created without any error. I always tested with XML input without `<doneStatus>`, and it also worked.

—----------

For `Projects` (tested by Ying Fei):

As I tried to run my step definition about returning the error message, I learned that Java makes distinctions between the methods used for retrieving messages with error code and with success code. So when an error is given back by the server, I should use `connection.getErrorStream()` instead `connection.getInputStream()`.

Another thing I discovered is that in the following image, if I took out the line starting with `int responseCode`, there would be an `java.lang.NullPointerException` when executing `connection.getErrorStream()`, which leads to a test failure. This happens when the Alternate Flow of Story #2 were run.

```java
@Then("Manager sends back an error msg: {string}")
public void managerSendsBackInstanceNotFoundMsg(String msg) throws Exception {
    // Read and print the response from the server
    int responseCode = connection.getResponseCode();
    BufferedReader reader = new BufferedReader(new InputStreamReader(connection.getErrorStream()));
    StringBuilder response = new StringBuilder();
    String line;
    while ((line = reader.readLine()) != null) {
        response.append(line);
    }
    reader.close();
    assertTrue(response.toString().contains(msg));
}
```

The last thing that caught my attention was that I could not retrieve the error using `connection.getErrorStream()` when a bad request is caused by an invalid parameter character such as GET /projects/%1. In Postman, it gives me back an error code of 400. I wanted to catch this error, but did not succeed. So I change this scenario to some other catchable error.

For `Categories` (tested by YuTong):

The unexpected bugs appeared in User Story 4. First one being in the Error Flow. When trying to retrieve the todos associated with an inexistant category ID, I realized that it did give the wrong response code. The expected response code is 404 but instead the response code received is 200. This means that the API was able to get something out of an inexistant category which seems like a bug.

```
Scenario: Retrieve todos related to a category                # cucumber/pickle/example.feature:129
  Given User connects to TodoManager                          # cucumber.pickle.StepDefinitions.setUp()
  When User retrieves todos related to category with ID 100001 # cucumber.pickle.StepDefinitions.get_categories_of_todo_with_ID(int)
!!! The Manager did not report a not found error !!!
The response code is 200
  Then Manager should report a not found error                # cucumber.pickle.StepDefinitions.Manager_expects_a_not_found_error()
```

The second one being in the Alternate Flow 1 and 2. When trying to add todos using PUT vs using POST. In part 1, I realized that no matter if it is about todos or projects, using PUT to add a relationship to a specific category will give an error and the todos or projects won't be added to the category. In the following scenario, I tested adding a relationship with json using PUT and POST. I was able to catch the error for PUT and as expected, PUT didn't work. Similarly to POST, as expected, POST worked completely fine.

PUT

```
Scenario: Add todos to a categories using JSON                # cucumber/pickle/example.feature:111
  Given User connects to TodoManager                          # cucumber.pickle.StepDefinitions.setUp()
  When User adds a todo with PUT JSON data to category with ID 1 # cucumber.pickle.StepDefinitions.Add_a_todo_to_category_with_PUT_ID(int)
405
Error: PUT didn't work
  Then Manager sends back the created todo with the ID        # cucumber.pickle.StepDefinitions.send_back_created_category_with_the_ID()
  When User retrieves todos related to category with ID 1     # cucumber.pickle.StepDefinitions.get_categories_of_todo_with_ID(int)
  Then the request is successful                              # cucumber.pickle.StepDefinitions.check_request_is_successful()
{"todos":[]}
  Then Manager sends back the todos                           # cucumber.pickle.StepDefinitions.sends_back_the_categories()
```

POST

```
Scenario: Add todos to a categories using JSON                  # cucumber/pickle/example.feature:119
  Given User connects to TodoManager                            # cucumber.pickle.StepDefinitions.setUp()
  When User adds a todo with POST JSON data to category with ID 1 # cucumber.pickle.StepDefinitions.Add_a_todo_to_category_with_POST_ID(int
201
{"doneStatus":"false","description":"Test 4","id":"3","title":"Story 4 - Alternative 2"}
  Then Manager sends back the created todo with the ID          # cucumber.pickle.StepDefinitions.send_back_created_category_with_the_ID(
  When User retrieves todos related to category with ID 1       # cucumber.pickle.StepDefinitions.get_categories_of_todo_with_ID(int)
  Then the request is successful                                # cucumber.pickle.StepDefinitions.check_request_is_successful()
{"todos":[{"doneStatus":"false","description":"Test 4","id":"3","title":"Story 4 - Alternative 2"}]}
  Then Manager sends back the todos                             # cucumber.pickle.StepDefinitions.sends_back_the_categories()
```