Qi Chen – 261053991

**Deliverables Summary**

There are four deliverables in the part B of the project.

The first one is the repository URL stored in a text file. The repository contains the feature files, the cucumber test runner and the step definitions.

The second one is a Word document containing the 5 main user stories that were tested during this part of the project, and they follow the following style: As a user I want to complete some operation to get some value.

The third one is a short video file to demonstrate all the story tests with the normal, alternate and error flow running and asserted for correctness. The video is divided into two parts. The first one shows the tests running sequentially and there is a short overview of the content of the generated HTML file that shows the content and results of the tests themselves. The second one make use of the Cucumber CLI to execute the tests in a random order using a random seed to show that each scenario can be executed independently to each other.

The last one is a Word file, and it is the report for part B itself.

Since no new bugs was found from the acceptance tests, no new bug report has been submitted.

**Story Test Suite Structure**

Test Runner

All the gherkin’s scripts are executed from a single runner, “CucumberRunnerTest.java” when the JUnit engine is used to run the story tests instead of the Cucumber CLI. The primary domain model that is tested is “todos”.

This is a class that ensures the tests will run smoothly in a proper environment by ensuring multiple things, first, the proper environment configuration:

* The Cucumber engine is used
* Cucumber knows where the required feature files are located
* Cucumber knows where the glue (step definitions) is
* It ensures that an HTML file is generated at the end that summarizes what features and scenarios were tested and the results.

Then, that the server is available before starting each test:

* It ensures that the server is reachable by sending a dummy HTTP request to the server address. If a connection error happens, the test ends, otherwise the class sets up the test environment.

After that, the proper testing environment is set up for each test by executing the cleanup procedure:

* The class get a list of all the domain instances that are present on the server at that time.
* For each instance, a delete request is sent to the instance endpoint

That way, each test starts in a blank state with no residual instances of the same domain from previous tests.

After the test is executed, whatever the result or if an exception happened, there is a cleanup procedure that is executed so that the server returns to its initial state.

Feature File

The gherkin scripts are grouped into feature files. Each feature file tests a single user story.

In each feature file, there are three execution flows that are tested. Each of them represents the different paths a user may follow when using the service provided by the server.

Normal Flow:

This describes the most straightforward and most expected situation that can happens to the user when following the user story. It is ideal where everything works as planned without any disruptions. There are no special cases, error messages or exceptions. It evaluates how the server can handle usual requests and if it is able to provide the business value.

For example, if a user wishes to create a new todo item in the server, the most straightforward path may be as follow:

* The user wants to create a new todo item.
* The user fills all the todo fields; the title, the doneStatus and the description of the todo instance.
* The user sends their request to the server.
* The user confirms that their todo item was created by seeing the additional todo item in their todos list of all instances.

Alternate Flow:

Compared with the normal flow, the alternate flow describes a different path from the normal flow. Despite the deviation from the normal flow, the alternate flow still leads to a successful outcome.

This flow is used to test some optional steps, special cases or a different behavior a user may wants to engage into that the server will have to accommodate to smoothly process the user request and to be still able to process it successfully. It evaluates how the server can handle flows that diverge from the normal such as edge cases.

For example, still following the same user story that the user wants to create a new todo item, the alternate flow was tested as such:

* The user wants to create a new todo item.
* The user fills some of the todo fields; only the title and the doneStatus of the todo instance.
* The user sends their request to the server.
* The user confirms that their todo item was created by seeing the additional todo item in their todos list of all instances.
* And the description field was automatically filled with a default value, which is an empty string in this case.

As we can see, even when the user followed a different path than the normal one, a path that may be seen as uncommon, the server properly processed the missing non-mandatory field and filled it up with a default value and processed the request successfully instead of sending an error to the user.

Alternate Flow:

Compared to the two other flows, this flow leads directly to an error. It describes a sequence of actions from the user that lead them to an invalid condition or state and prevents them from continuing down to this path. It evaluates how the server can gracefully handle wrong inputs or state and provide the correct error handling like sending the correct notifications and redirecting the user to a correct state.

For example, with the same user story, an error flow was tested as such:

* The user wants to create a new todo item.
* The user fills some fields, but not the mandatory title field.
* The user sends their request to the server.
* The user receives an error message saying the request is invalid.
* And the user sees that no new instance of todo was created, confirming that their request was rejected.

Here, the server was able to properly handle the erroneous request from the user and not only prevent the creation of a bad todo instance, but also send the correct error message to the user to notify their error.

Step Definitions

All the step definitions are stored in “CucumberStepDefinitions.java”.

There are state variables to ensure that information can flow between each step of a test.

For example, the initial number of instances in a list is stored to compare it with the number of instances in a list after a user action, the request body is stored so that the user can change what field and their value will be using familiar language before the request body will be sent to the server and any newly created object instance is stored to keep track of its properties such as its id.

To ensure the highest reusability of each step, the steps in the gherkin scripts are intentionally written to be as general and modular as possible to be able to granularly manipulate the state variables while still retaining the user friendly and familiar language.

**Story Tests Findings**

Compared with the unit tests, the story tests primary tested what the server could provide from a business value perspective, that meant that some technical aspects such as the error codes, weren’t as much as enforced as the unit tests since they already tested on that.

For the 15 scenarios tested across 5 user stories, the actual results were in line with what was expected from a business perspective and no new bug was found during the story testing.

**Source Code Repository**

URL: <https://github.com/qiy-chen/ECSE429SoftwareTesting>

The source code is a Maven project directory with the dependencies needed to properly execute the unit and acceptance tests. The tested application is in the root.

The 5 feature files are store in src/test/resources/com/testproject/ and each one test a single user story.

The Cucumber Runner and Step Definitions are in src/test/java/com/testproject/. In the step definitions file, the definitions are grouped by what keywords they define and the groups are ordered by the order of usage, so the “Given” definitions are the first one, then the “Then” definitions and the “Then" definitions, the ones that assert correctness, are the last.

The deliverables except the video are situated in the “Documents” folder. The report is under the folder “Reports” and the user stories are in the current folder.