ECSE Software Validation Term Project

Part B Story Testing of Rest API

The following user stories are being tested in the part of the project.

As a student, I categorize tasks as HIGH, MEDIUM or LOW priority, so I can better manage my time.
As a student, I add a task to a course to do list, so I can remember it.
As a student, I mark a task as done on my course to do list, so I can track my accomplishments.
As a student, I remove an unnecessary task from my course to do list, so I can forget about it.
As a student, I create a to do list for a new class I am taking, so I can manage course work.
As a student, I remove a to do list for a class which I am no longer taking, to declutter my schedule.
As a student, I query the incomplete tasks for a class I am taking, to help manage my time.
As a student, I query all incomplete HIGH priority tasks from all my classes, to identity my short-term goals.
As a student, I want to adjust the priority of a task, to help better manage my time.
As a student, I want to change a task description, to better represent the work to do.

Story Test Suite

At least three acceptance tests must be defined for each story. At least one of each of the following types of acceptance tests are required.

- Normal flow
- Alternate flow
- Error flow

Story tests are defined as scenario outline gherkin scripts, in feature files. Each scenario outline gherkin script can be executed many times with different values for variable data.

Story tests should include a background section to set the initial conditions for the gherkin script.

Story Test Automations

Project teams will use the open source story test automation tool cucumber, or any similar tool which can parse and execute gherkin scripts.

Team shall implement step definitions which control the to do list manager rest api studies in Part A of the project.

Step definitions should be built as a library, elements of which can be reused in several different acceptance tests.

Code in step definitions should:

- Use clean, well-structured code follow guidelines of Bob Martin on Clean Code.

Each gherkin script should

- Assess correctness
- Restore the system to the initial state upon completion
- Run in any order

Additional Story Test Considerations

Ensure story tests fail if service is not running.

Additional Bug Summary Considerations

The project team should use the same format as in part A to report any bugs identified. If new bugs are found, please indicate which story they are related to.

Story Test Suite Video

Show video of all story tests running in the teams selected development environment.

Include demonstration of tests run in different orders.

Written Report

Target report size is between 5 and 10 pages.

Summarizes deliverables.

Describes structure of story test suite.

Describes source code repository.

Describe findings of story test suite execution.

Summary of clean code guidelines from Bob Martin.

Understandability tips

- Be consistent. If you do something a certain way, do all similar things in the same way.
- Use explanatory variables.
- Encapsulate boundary conditions. Boundary conditions are hard to keep track of. Put the processing for them in one place.
- Prefer dedicated value objects to primitive type.
- Avoid logical dependency. Don't write methods which works correctly depending on something else in the same class.
- Avoid negative conditionals.

Names rules

- Choose descriptive and unambiguous names.
- Make meaningful distinction.
- Use pronounceable names.
- Use searchable names.
- Replace magic numbers with named constants.
- Avoid encodings. Don't append prefixes or type information.

Functions rules

- Small.
- Do one thing.
- Use descriptive names.
- Prefer fewer arguments.
- Have no side effects.
- Don't use flag arguments. Split method into several independent methods that can be called from the client without the flag.

Comments rules

- Always try to explain yourself in code.
- Don't be redundant.
- Don't add obvious noise.
- Don't use closing brace comments.
- Don't comment out code. Just remove.
- Use as explanation of intent.
- Use as clarification of code.
- Use as warning of consequences.

Source code structure

- Separate concepts vertically.
- Related code should appear vertically dense.
- Declare variables close to their usage.
- Dependent functions should be close.
- Similar functions should be close.
- Place functions in the downward direction.
- Keep lines short.
- Don't use horizontal alignment.
- Use white space to associate related things and disassociate weakly related.
- Don't break indentation.
- Objects and data structures
- Hide internal structure.
- Prefer data structures.
- Avoid hybrids structures (half object and half data).
- Should be small.
- Do one thing.
- Small number of instance variables.
- Base class should know nothing about their derivatives.
- Better to have many functions than to pass some code into a function to select a behavior.
- Prefer non-static methods to static methods.

Tests

- One assert per test.
- Readable.
- Fast.
- Independent.
- Repeatable.

Avoid Code smells

- Rigidity. The software is difficult to change. A small change causes a cascade of subsequent changes.
- Fragility. The software breaks in many places due to a single change.
- Immobility. You cannot reuse parts of the code in other projects because of involved risks and high effort.
- Needless Complexity.
- Needless Repetition.
- Opacity. The code is hard to understand.