CPEN 321 Software Engineering
Winter 2023

M6: Testing (Monday, November 20, 9pm PT)

This is a group assignment. The deliverable for this milestone is **test specification and results** for your project. For the milestone, you will perform:

- Automated testing of your backend, focusing on all APIs exposed by the back-end to the front-end. If you have other externally triggered back-end APIs (e.g., functions called on a certain schedule), you will need to test them as well.
    - To check all erroneous scenarios, you will need to mock (https://jestjs.io/docs/en/mock-functions) all components which you cannot fully control, e.g., databases, filesystem, and APIs of external components.
- Automated end-to-end testing of your project, focusing on frond-end GUI tests for three main use cases.
- Automated testing of two non-functional requirements of your project.
- Manual customer acceptance testing for the app of **your client team.**

You can use assistive AI technology (namely, ChatGPT 3.5) when working on this assignment. No points will be deducted for documented use. However, undocumented use will be considered academic misconduct and will be treated accordingly.

You must document and critically analyze all usages of ChatGPT during the process of working on this assignment in a systematic way described below. This analysis **must be submitted** as part of the assignment and will be graded. Using ChatGPT 4 or any other version / AI technology is **not allowed**, for fairness (so that all students will get the same level of support).

SETUP:

1. Install Jest (https://jestjs.io) – a testing and code coverage framework for JavaScript.
2. Install Espresso (https://developer.android.com/training/testing/espresso) – a testing and code coverage framework for Android.
3. Set up GitHub Actions (https://docs.github.com/en/actions) to run your back-end tests every time code is pushed to the **main** branch of your Git repository.

SUBMISSION. The submission for this milestone will include two parts.

PART I: a PDF file named "M6_Report", which includes the following information in the order specified below:

1. Names and student numbers of all group members
2. The contribution of each group member to the work done for this milestone **[2 points]**
    - 1-2 sentences per member
3. A short description of the project and the up-to-date use case and module dependency diagrams **[3 points]**
    - Note that the final version of your requirement and design artifacts (the use case diagram, requirement specifications, non-functional requirements, design, etc.) will be graded again in M7.

4. Changes made to the project requirements and design (since M2 and M3, respectively) **[3 points]**
   - If no changes were made, say "None".
   - If there are changes, describe the reason for each change. Please make sure the justification is clear and reasonable.

5. Back-end Tests [**30 points]**.

   5.1 Description of tests for **all exposed interfaces** of your back-end.
   - Exposed interfaces generally correspond to APIs exposed to the front-end of your application (as defined in previous milestones). However, they could also include recurrent events which do not expose any API to the front-end (e.g., an operation scheduled to happen every day).
   - For each exposed interface, create a "*describe*" group (https://jestjs.io/docs/api), which will include all individual test cases for the interface. Annotate each test with information about its inputs, expected outputs, and expected behaviors, as in the example below.

   Example:

```
// Interface POST www.myserver.ca/photo
describe("Upload a photo", () => {
// Input: test_photo.png is a correct photo
// Expected status code: 201
// Expected behavior: photo is added to the database
// Expected output: photo_id
   test("Valid Photo", async () => {
    const res = await app.post("/photo/")
        .attach("photo", "test/res/test_photo.png");
    expect(res.status).toStrictEqual(201);
    expect(await Photo.getAllPhotos()
        .toEqual(expect.arrayContaining(["test_photo.png"])));
   });
// Input: no photo
// Expected status code: 401
// Expected behavior: database is unchanged
// Expected output: None
   test("No Photo", async () => {
     //...
   });
```

```
// Input: bad_photo.txt is a not a photo
// Expected status code: 400
// Expected behavior: database is unchanged
// Expected output: None
   test("Bad Photo", async () => {
    const res = await app.post("/photo/")
        .attach("photo", "test/res/bad_photo.txt");
    expect(res.status).toStrictEqual(400);
    expect(await Photo.getAllPhotos()
        .toEqual(expect.not.arrayContaining(
            ["bad_photo.txt"])));
   });

//... more tests …

  });
```

   5.2 The location of your back-end test suites and instructions to run them
   - The hash of the commit on the **main** branch where your tests run. TAs will run the tests on this version. Some tests may still fail at this point, but they will need to pass by M7.
   - A table listing the location of the test for each API and what mocks were implemented to test API, if any. See an example below.
   - Explain how to run the tests. If no clear explanations are provided, points will be deduced for the entire milestone.
   - Make sure you configure the tests to report coverage information for each individual file in your back-end (see *collectCoverageFrom* configuration option in https://jestjs.io/docs/configuration#collectcoveragefrom-array).

Interface	Location in Git	Required Mocks
POST www.myserver.ca/photo	https://github.com/xx/backend/tests/photo.test.js#L3	Lists DB
GET www.myserver.ca/photo	…	Lists DB
…	…	…
…	…	…
…	…	…

5.3 The location .yml files that run all your back-end tests in GitHub Actions.
- Points will be deducted if tests do not run in regression mode, i.e., on every Git push
- TAs will use your configuration files to run all tests on the latest commit in the main branch. If the test is not triggered from the configuration file, we will disregard it and it will not be executed.

5.4 Screenshots of Jest coverage reports for backend tests
- Run all backend tests together and report the coverage for all files in your backend
  - We expect to see high coverage for each back-end file.
  - If the coverage is lower than 100%, provide a well-formed reason not achieving 100% coverage. Marks will be deduced if the justification is opinion-based.

6. Front-End Tests Implementation and Results **[25 points]**.

6.1 The location and description of your front-end test suites
- Explain how the test suite directory is structured and where to find each test

6.2 The design and implementation of front-end UI tests for **three major use cases** of your application:
- Each UI test case should include success and failure scenarios from the formal use case specification (possibly updated from M2).

Example: Given a formal use case specification for "Add Todo Items" use case:

Main Success Scenario

1. The user opens "Add todo items" screen
2. The app shows an input text field and an "Add" button. The Add button is disabled
3. The user inputs a new item for the list. The add button is enabled.
4. The user presses the add button.
5. The screen refreshes and the new item is at the bottom of the todo list.

Failure scenarios

3a. The user inputs an ill-formatted string.
3a1. The app displays an error message prompting the user for expected format.
5a. The list exceeds the maximum todo-list size
5a1. The app displays an error message to inform the user.
5b. The list is not updated due to recurrent network problems
5b1. The app displays an error message prompting the user to try again later.

A test case specification could be:

Scenario steps	Test case steps
1. The user opens “Add Todo Items” screen	Open “Add Todo Items” screen
2. The app shows an input text field and an “Add button”. The add button is disabled	Check text field is present on screen Check button labelled “Add” is present on screen Check button labelled “Add” is disabled
3a. The user inputs an ill-formatted string	Input ““*^*^^OQ#$” in text field
3a1. The app displays an error message prompting the user for expected format.	Check dialog is opened with text: “Please use only alphanumeric characters”
3. The user inputs a new item for the list. The add button is enabled	Input “buy milk” in text field Check button labelled “add” is enabled
4. The user presses the “Add” button	Click button labelled “add”
5. The screen refreshes and the new item is at the bottom of the todo list.	Check text box with text “buy milk” is present on screen Input “buy chocolate” in text field Click button labelled “add” Check two text boxes on the screen with “buy milk” on top and “buy chocolate” at the bottom
5a. The list exceeds the maximum todo-list size	Repeat steps 3 to 5 ten times Check dialog is opened with text: “You have too many items, try completing one first”

Note that for 5b, one would need to mock backend responses, which is not required for this milestone.

6.3 The Espresso execution logs for the automated test runs (with passed/failed status)

7. Non-functional Requirements [**10 points]**.

   For each of the two non-functional requirements that you verify automatically, report:
   - A one-paragraph description of how you verified that the requirement was met
   - The logs of your verification results for each requirement

8. Manual customer acceptance testing for the app of your client **[10 points]**.
   - Report one major fault you found in your client team’s app. The report should contain the buggy execution scenario (sequence of events, with screenshots) and the description of the fault
   - Yes, you will find some major faults – there is no app without faults two weeks before the final deadline
     - If you report that you cannot find any major fault and then a TA does, you will lose marks. Otherwise, you will be given the full mark.
   - The one major fault that your client team found in your app. Provide a short description, no screenshots are needed.
     - If there is a mismatch between what you note in your report and your client team’s, you won’t be given any marks for this section.
     - If your client team did not find any major fault in your code, ask your teammates working on the back-end to test the app. Report the identified faults.
     - If you report that neither your peer-group nor your own team members can find major faults in your app and then a TA does, you will both lose marks. Otherwise, you will be given the full mark.

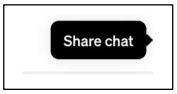
9. Reflections on using/not using ChatGPT 3.5 for this milestone. **[15 points]**

   If you did not use ChatGPT 3.5 when working on the assignment, you must answer the following questions in 1-2 paragraphs each:
   - Why did you decide not to use ChatGPT for this assignment?
   - Show concrete examples of its inadequacy.
   - Did you use any online resources (StackOverflow or similar) instead? If so, name the resources you used, explain how the resource was used, and why this is more beneficial than using ChatGPT.
   - Is there anything else you would like to share about this process?

   If you used ChatGPT 3.5, you must answer the following questions in 1-2 paragraphs each:
   - List all tasks ChatGPT helped you achieve and the reason you decided to use it for each of these tasks.
   - How many test cases did you design and/or implement with the help of ChatGPT? Specify the total number of test cases in your front-end and back-end code and the total number of test cases for which you rely on ChatGPT output.
     - In your test code, you must add a comment before the name of each test to specify whether or not ChatGPT output (possibly refined) was used for implementing the test. The comment should be formatted as follows: "ChatGPT usage: Yes/Partial/No").
   - What are the advantages of using ChatGPT for each of these tasks?
   - What are the disadvantages of using ChatGPT for each of these tasks?
   - Is there anything else you would like to share about this process?

10. Names and a short description of the purpose of all ChatGPT conversations (prompts and replies) used when working on this milestone. Each conversation should be saved and uploaded as an HTML file, as specified in Part II **[2 points]**.

PART II: All ChatGPT conversations (prompts and replies) used or attempted when working on this milestone. Each conversation should be saved and uploaded as an HTML file, using the procedure described below:



- Click on the "Share Chat" icon in the top right corner of the ChatGPT Web window.
- Click on "Copy Link". The link to the chat will be copied to clipboard.
- Open the link in a new tab of a Web browser.
- Save the page by pressing Ctrl + S.
- Name the file using the following schema: T_N.html, where T is the sequential number of the chat used for the milestone. For example, "T_2" is the second chat for test design T.

<u>Assignments that relied on ChatGPT but do not declare the use and do not attach ChatGPT conversations will lose all marks for the milestone.</u>

Good luck!