Detailed Test Plan

Organization

Broadly speaking, following will be the structure of our test cases.

qa327_test

| --- frontend

| --- login

| --- test_login.py

| -- register

| --- test_registration.py

| -- misc

| --- test_404.py

……. (so on, as present in actual directories)

| --- backend

| --- login

| --- test_login.py

| -- register

| --- test_registration.py

……. (so on, as present in actual directories)

| --- integration

| --- login

| --- test_login.py

| -- register

| --- test_registration.py

……. (so on, as present in actual directories)

The basic idea is that we will have all test cases segregated firstly by level, i.e. frontend, backend, integration, etc. as separate folders. These folders will have further sub-folders, which will map one-to-one to the requirements and the test cases as we documented in the first assignment. For each of these sub-folders, we will have a requirement specific file, such as

`test_login.py` for front/login, which will store all unit test cases for /login [GET]. For each of the three main folders, we will also have a misc sub-folder, which will be used to store any test cases which do not directly fall under a sub-requirement. The folder structure has been populated with dummy test case files as per this convention, which we will populate for the next assignment in detail.

Ordering & Testing Tools

The testing framework that we will be using is PyTest, a framework that makes it easy to write small tests, and scales to support complex functional testing for applications and libraries.

As evident from our test case organization, we have separated testing into three main components: frontend, backend, and integration. At first, we will only be testing the frontend part, which will primarily be comprising of unit tests written using mocking to create a fake backend. Using PyTest, the test cases can be automated and run directly on GitHub. The test cases will be run once the frontend code is completed. Simultaneously, with the development of the backend, the test cases of the backend can also be directly run on GitHub. Once we are at a stage where our rapid development prototype is at a stage that it can be integrated, we will write accompanying integration test cases, and using PyTest, that too can be automated to run directly on GitHub.

Responsibility

Initially, we had decided to split up the documentations of the test cases by each general requirement, making sure that each of us lead work on at least two requirements. For example, one person was to write all the test cases for the /login and /*, and another was to write all test cases for /register and /logout, etc. However, we ended up sub-dividing these requirements further between group members, because we figured that it'd make the reviewing process of the pull requests a lot easier if more people were to know about requirements in general instead of having dedicated people being responsible for bigger chunks of requirements, in contrast to the approach we took for the first assignment.

For purposes of clarity in documentation, we have now moved the documented test cases from our first assignment into a separate folder within the root folder titled `Test Requirements`. This folder contains our original work for the first assignment, but has just been moved to improve upon the logical code organization within our application.

Budget Management