Team NBA's Deliverable 2: Test Plan

1. Description of Testing Process
    a. Server Tests
        i. Language Test
            1. This test case uses the browser's default language and cookies to detect which language to display in on the GUI. It also uses an open-source software called Mockito, which is a mocking framework that lets you write very neat tests with a clean API.
        ii. Server Computations Test
            1. The way I describe this test case is that it creates a "mock" version of the project and uses the server-side methods to pull information from the project, using "getters."
    b. Shared Tests
        i. Computations Test
            1. This test is one of the longer test cases we are highlighting. This test file references a same-size file in the main project folder of Sigmah. It essentially is a big mathematical tester, because: users can input many different things into the database fields and to avoid errors, we check! Sigmah seems to have a significant French presence as there are some numbers that are inputted as text, in French. These numbers have to be converted and as such, should be tested!
        ii. Decimal Formatage Test
            1. This test uses JUnit's Assert to test the ratio method of the associated class, as well as the truncate method of the same class, which rounds floats to the specified decimal places.
        iii. File Type Test
            1. This test uses JUnit's Assert to return true or false if the accepted file types are recognized
2. Statement of Requirements Tested
    a. The user should be able to access the software and view it in their native language. I.E. The software should encompass multiple languages.
    b. The user should be able to search for specific "header" values of a project in order to search with ease.
    c. The user should be able to input numbers in French, and still be able to perform mathematical operations.
    d. The user should be able to input and receive accurate ratios, as well as have any inputs be rounded to the correct number of decimal points automatically.
    e. The user should be able to upload text-based files, links to webpages, and photos to the software, and they should be accessible by other users.
3. Statement of Tested Items

    a. We tested the ability of the software to encompass multiple languages, each tailored to a specific user and able to be changed depending on who is viewing the information.
    b. We tested the ability of the software to pull specific information out of a project, to aid in searches and ease of access for users.
    c. We tested the ability of the software to convert text-based numerals in French to Integers, and calculate mathematical expressions correctly with these Integers.
    d. We tested the ability of the software to correctly form ratios as well as a method for truncating found in the same class.
    e. We tested to ensure that file uploads from users match our accepted file types and extensions to ensure that each uploaded file is accessible by the project and all users who wish to view them.
4. Procedure for Recording Tests
    a. Most of our test cases use Java's built in class Assert to return a boolean value depending on the outcome of the test. In essence, true if no error, false will throw an exception and elicit an error.
    b. Because Assert is used, the log of outputs from the test can just as easily be outputted to a text file, or if debugging, can be outputted to the console in your IDE to allow the ease of "quick" debugging, to keep from having to re-open the text log.
        i. This helps because when the software is complete, the outputs will be set up to be recorded in an error log, with timestamps, to help developers to fix errors that user's experience.
        ii. However, when still being developed, developers can have the output printed to their IDE's console to allow for quick viewing and editing of the code, in order to more efficiently correct the bug.
5. Testing Requirements
    a. Our requirements for testing, on the surface, were pretty cut and dry. We needed to utilize our software, programmed in Java, on a Linux operating system. However, all three of us have different native OSs. Andrew runs Linux on his OS so he was most prepared for this. Noah runs Windows, and Brian runs macOS. The first obstacle for the latter two was to get a copy of Linux running in order to run the software.
    b. Noah and I both used the latest version of Virtual Box (5.1.28). Once downloaded and installed we then had to actually get a Linux OS up and running.
    c. We are all using Ubuntu 16.04LTS for our OS now, up and running Sigmah.
    d. To run Sigmah, we had to install several APKs to get our "fresh" Linux OSs up to date. After doing so, we used Bash Shell for most of our setup process.
    e. After getting the necessary packages installed on our machine, we then downloaded our IDE, Eclipse. The latest version that we are all using is Eclipse Oxygen.

    f. In Eclipse Oxygen, we are all still using Java SE 8 (we have not updated to the very new SE 9). This is sufficient for running Sigmah and it will not be a problem if you are running an older version of Java.
6. Constraints While Testing
    a. Noah's constraints while testing were not as numerous as Brian's, but still set him back. He suffered an irreparable hard drive failure early on in the project phase and had to buy a new computer and completely reinstall the required software. Additionally, he had to request fewer hours at work because he felt like he did not have enough free time to cover his 300 level CS courses. Since the schedule change and computer upgrade, however, things have been smooth sailing with exception being the hurricane scare.
    b. Brian's constraints while testing were mostly personal and time-based. He has two stepsons, 10 and 8, who play soccer and have an abnormally high volume of homework for kids of their age. They practice soccer Monday, Tuesday, and Thursday every week. His youngest, has the addition of Cub Scouts on Wednesdays. That leaves only Friday, Saturday, and Sunday to work on testing. This is further limited by the flood he experienced in his home recently so he has had to take time to make his home livable, which in turn has left him with minimal time for testing but he has been making an admirable effort and has pulled through for us nonetheless.
7. Test Cases
    a. Test Case #1: Language Test
        i. This is a test case for the language detection system that uses cookies, and the browser's default language to determine the language to be displayed.
        ii. Uses Mockito, which is a mocking framework that lets you write beautiful tests with a clean & simple API. The tests are very readable and they produce clean verification errors.
            1. http://site.mockito.org
    b. Test Case #2: Server Computations Test
        i. This Server Computation test is of a method that will be used often.
            1. This essentially creates a "mock" version of elements that will be used within the project models.
            2. The mock project is then used to call "get" methods to retrieve specific information from the mock project.
        ii. Again, JUnit's class Assert is used, returns an error if false. If true, all is given the green light.
    c. Test Case #3: Computations Test
        i. Imports several shared computation classes from the Sigmah library.
            1. There is a Computations.java file in the Sigmah drive.
        ii. Test several parsable mathematical inputs, from String to Integer.
            1. There are actual test cases for inputting numbers in French, and converting them into integers.

iii. To ensure that in-depth mathematical expressions are parsed correctly and follow the proper order of operations.
    1. Tested to throw errors if they have not.
iv. Ensures that the project is compatible with several different languages and their math counterparts that follow.

d. Test Case #4: Decimal Formatage Test
    i. This test case tests the ratio method from NumberUtils. Very simple and straightforward.
    ii. We tested the ability of the software to correctly form ratios as well as a method for truncating found in the same class.

e. Test Case #5: File Type Test
    i. JUnit class —> Assert
        1. A set of assertion methods useful for writing tests.
        2. Essentially a boolean error checker that throws an error only if false. If true, all is good (no action).
    ii. Output can be displayed to console for quick checking, or outputted to an error log text file.
    iii. In particular, this class is testing the File Types, and the default file extensions for these file types.
    iv. File types supported:
        1. Text: .CSV , .PDF , .XML
        2. Links: .HTML
        3. Images: .JPEG