Group 1 Tigers Testing Plan

Login:

1. User acceptance testing
    a. After registering an account successfully, users should be able to use the username and password they created to login to that account.
    b. If the username does not match one of an account that has already been created, then the test fails. Also, if the username matches, but the passed password is not a match for the one in the users table, then the test fails
    c. If either of the fields are left blank, the test case fails.
2. Test Data:
    a. Test data for these test cases will be manually created and inputted into the users table.
        i. There will be a variety of different usernames with different combinations of uppercase and lowercase letters and numbers. Also, there will be a variety of passwords with the same diversity as the usernames with regards to numbers/letters/symbols.
        ii. In order to test this data, we will type data we have inputted into the table into the login fields to ensure that the login page is functioning correctly.
        iii. We will also input username/password combinations that we know do not exist, so that we can ensure that the login page is not letting anyone in who has invalid credentials. Finally, we need to make sure that if the fields are left blank that the login page recognizes this and doesn't let the user into the homepage.
3. Test Environment:
    a. JUnit will be used to run these tests. We chose JUnit because it is a test framework that allows for tests to be automatically run each time the code is run.
        i. This is important for us because it means that every time we implement a feature or change something on the website/program, the login feature will automatically be tested to ensure it was not affected by the new feature.
        ii. By using JUnit, we only have to set up all of the test cases once, and every time we run the code, the login feature will be tested. That way, if the login page breaks, we know what broke it and can fix it because we've been testing every step of the way.
    b. We will use a version of the webpage for the user acceptance testing. This version of the website will be one that is nearly identical to the finished product, minus any improvements needed for the login page discovered during testing.
        i. The users will use this website to register/login and test that their password combinations are working, and that any invalid ones are being denied by the system.
        ii. If any errors are discovered, the webpage can be easily modified and then testing can resume on the modified version.
4. Test Results:

    a. Test Passed: If the test is passed, the user will be redirected from the login page to the home page.
    b. Test Failed: If the test fails the user will **not** be redirected to the login page, and an error message will be displayed on the login page saying that the username or password they inputted did not match any in the system.
5. User Acceptance Testers:
    a. For user acceptance testing, real world people will need to be used for testing. Since the application will be hosted on the CU Boulder campus, other CU students or faculty will be used as testers.
        i. These testers will navigate to the webpage and attempt to login.
            1. If they don't have an account, the test should always fail.
            2. If the tester has created an account, then the test should fail if they intentionally (or accidentally) provide invalid credentials.
        ii. The test should **only** pass if the user provides the username and password that they used to create their account (case sensitive password)

Searching:

1. User Acceptance Testing
    a. Users should be able to search for recipes either by name or keyword.
    b. If no recipes match the search, or if no recipes include the keyword, search fails, telling the user no recipes match their search.
2. Test Data:
    a. Test Data would be an assortment of recipe names we can be certain will not match any recipes in our system.
    b. Other test data would be keywords that wouldn't match any recipe in our system.
3. Test Environment:
    a. JUnit will be used to run these tests. We chose JUnit because it is a test framework that allows for tests to be automatically run each time the code is run.
        i. This is important for us because it means that every time we implement a feature or change something on the website/program, the search feature will automatically be tested to ensure it was not affected by the new feature.
    b. By using JUnit, we only have to set up all of the test cases once, and every time we run the code, the search feature will be tested. That way, if the search page/feature breaks, we know what broke it and can fix it because we've been testing every step of the way.
    c. We will use an unfinalized version for the user acceptance testing. This version will be almost the same as the final product, without the potential additions found in user testing.
4. Test Results:
    a. Passed: If the test is passed, the user will see a list of any recipes that match their search.

    b. Failed: If the test is failed, no recipes will be produced and the user will receive an error stating “no recipes matched search”
5. User Acceptance Testing:
    a. In order to account for a broad spread of search inputs, real people on the CU Boulder campus should be used. This way, we can cover as many diverse cases as possible.
    b. User testers will navigate to the search bar, search anything they might usually search, and
        i. the search should pass if what they searched matches any recipe in our system
        ii. or, the search should fail if it doesn’t match

Register:

1. User Acceptance Testing:
    a. Users should be able to register with a new username.
    b. User authentication fails when the user provides a username that has already been taken.
    c. User authentication fails if the username doesn’t contain at least one uppercase letter, one lowercase letter, and one number.
    d. User authentication fails if the username is too long.
    e. User authentication fails if the username is too short.
    f. User authentication fails if the Register page is left blank, either on the username form, password form, or both.
    g. User authentication fails if the username contains characters that our system does not support.
    h. The form provides the user specific feedback about the error.
2. Description of the Test Data:
    a. The test data we will be using are:
        - Usernames that are too long
        - Usernames that are too short
        - Usernames that do not have at least one uppercase letter, one lowercase letter, and one number
        - Usernames with characters that our system does not support
        - Usernames that aren’t too long
        - Usernames that aren’t too short
        - Usernames that *do* have at least one uppercase letter, one lowercase letter, and one number.
        - Usernames that *do* use characters that our system supports.
        - Edge cases to each of these constraints.

- Usernames that already exist within the system
- Usernames that are left blank
- Passwords that are left blankUsernames that are too long
- Passwords that are too short
- Passwords that do not have at least one uppercase letter, one lowercase letter, and one number
- Passwords with characters that our system does not support
- Passwords that aren't too long
- Passwords that aren't too short
- Passwords that *do* have at least one uppercase letter, one lowercase letter, and one number.
- Passwords that *do* use characters that our system supports.
- Edge cases to each of these constraints.
- Passwords that already exist within the system

3. Description of the Test Environment:
    a. JUnit will be used to run these tests. We chose JUnit because it is a test framework that allows for tests to be automatically run each time the code is run.
        i. This is important for us because it means that every time we implement a feature or change something on the website/program, the register feature will automatically be tested to ensure it was not affected by the new feature.
        ii. By using JUnit, we only have to set up all of the test cases once, and every time we run the code, the register feature will be tested. That way, if the register page breaks, we know what broke it and can fix it because we've been testing every step of the way.
    b. We will use a version of the webpage for the user acceptance testing. This version of the website will be one that is nearly identical to the finished product, minus any improvements needed for the register page discovered during testing.
        i. The users will use our website to test the register page, and check that our system specified username and password combinations are working if correct, and that any invalid ones are being denied by the system.
        ii. If any errors are discovered, the webpage can be easily modified and then testing can resume on the modified version.
4. Description of the Test Results:
    a. Test Passed: If the test is passed, the user will be redirected from the register page to the login page.
    b. Test Failed: If the test fails the user will **not** be redirected to the login page, and an error message will be displayed on the register page saying that the username or password they inputted are (one or multiple of):
        1. Too long
        2. Too short
        3. Already exists
        4. Special character(s) not supported
        5. Left blank

6. Does not have at least one Uppercase letter
7. Does not have at least one Lowercase letter
8. Does not have at least one number.

5. Description of the User Acceptance Testers.
    a. For user acceptance testing, real world people will need to be used for testing. Since the application will be hosted on the CU Boulder campus, other CU students or faculty will be used as testers. Testers will also include the developers of the website.