Cornelius Test Plan

**Test scenarios**. List as many test scenarios as you can think of for each feature.

* Frontend
  + Login page
    - If fields are not filled in properly (left blank, illegal characters), then throw an error message
    - If the password entered is incorrect, prompt the user to enter again
    - If the password was entered incorrectly more than three times, prompt the user to the forget password page
    - Clicking “Login” button with all fields filled (proper character requirements) routes to the landing page
    - Clicking the “Sign up” button leads to Sign Up Page
    - Clicking the “Forget Password” button leads to Forget Password Page
  + Sign Up Page
    - Clicking Sign up leads to landing page
    - Make sure the email or phone number is not in our database already
    - Make sure that new user information is in our database
    - Filling out the text inputs actually saves the data or sends it to the backend
  + User profile page
    - On render, mock an API call to get fake user information
      * Check that the response data is displayed in correct part of page
    - Clicking My Hospitals triggers dropdown to expand
    - Clicking My Insurance Plan triggers dropdown to expand
  + Navigation Bar
    - Clicking Journal Icon triggers Journal timeline to appear
    - Clicking Plus Icon triggers Set Up Recording screen to appear
    - Clicking User Icon triggers User profile page to appear
  + Recording page
    - If we click on “record,” make sure that an alert box pops up asking for audio permissions if permissions aren’t accepted yet
  + General FAQ Page
    - FAQ page shows the current question and answers pair
    - Clicking Question triggers answer to appear in dropdown
* Backend
  + User
    - Create
      * Test that 200 status sent when valid parameters given
      * Test that 500 status sent when missing required parameter
        + Email
        + Password
      * Test that 500 status sent when invalid parameter given
        + Email already exists in database
        + Password too short
    - Read
      * Test that 200 status sent when valid parameters given
      * Test that 500 status sent when incorrect uid given in URL
    - Update
      * Test that 200 status sent when valid parameters given
      * Test that 500 status sent when invalid parameters given
        + Update the primary key uid
        + Change email to an email that already exists in database
        + Password too short
    - Delete
      * Test that 200 status sent when valid parameters given
      * Test that 500 status sent when incorrect uid given in URL
  + Visit Journal
    - Create
      * Test that 200 status sent when valid parameters given
      * Test that 500 status sent when missing required parameter
        + Text from Google Speech-to-Text API missing
      * Test that 500 status sent when invalid parameter given
        + Text from Google Speech-to-Text API formatted incorrectly
    - Read
      * Test that 200 status sent when valid parameters given
      * Test that 500 status sent when incorrect journal\_id given in URL
    - Update
      * Test that 200 status sent when valid parameters given
      * Test that 500 status sent when invalid parameters given
        + Update the primary key journal\_id
    - Delete
      * Test that 200 status sent when valid parameters given
      * Test that 500 status sent when incorrect journal\_id given in URL

**Definition of Done**. How will you verify features are complete and working in a replicable manner? Note, strategies may differ across features (e.g., some may be automated using different tools, but some may need to be manual)?

For testing, our Definition of Done is:

1. Code reviews are completed for all features
2. Testing should be done in production environment
3. All stories in MVP are tested end-to-end
4. The test should be passed for 95% of the cases, with exception to edge cases
5. For integration test, both positive and negative scenarios should be tested
6. Reported defects need to be fixed and then passed to testing again
7. Other Defects in ‘acceptable’ state to the Product Owner
8. Non-functional requirements are not required to meet because of the limitation of time

The unit testing will be done automatically. The integration tests are expected to be done automatically but will be tested manually at first because we need some time to learn the testing tool. UI testing will mostly be done manually and if we have more time, we will explore how to do UI testing in automation.

**Resources**. What do you need to support your tests (i.e., data files, scripts, mock databases, etc.)?

* Data files to mock API calls in the frontend
* Mock user data to test displaying different screens
  + Visit journal, user profile

## Example of Gherkin

As Gherkin is a structured language it follows some syntax let us first see a simple scenario described in gherkin.

***Feature: Search feature for users***

***This feature is very important because it will allow users to filter products***

***Scenario: When a user searches, without spelling mistake, for a product name present in inventory. All the products with similar name should be displayed***

***Given User is on the main page of www.myshopingsite.com***

***When User searches for laptops***

***Then search page should be updated with the lists of laptops***

<https://medium.com/@ronak8036/react-native-testing-tools-f38d715adb57>