celo

CELO HEALTH
QA Testing Practical Task
Test Plan and Cases

Ray Hackshaw

Table of Contents

NB: This document is a skeleton model for a test plan + test case(s) – certain areas, such as revision history, schedule and team members etc., have been left out as they are not suitable/applicable for this exercise.

ray@rayhackshaw.com 02102328715 rayhackshaw.com

1 Introduction

This document serves as an example test plan / test case for Celo Health as part of their recruitment process for their 2020 QA practical challenge.

1.1 Objectives

Celo Health have developed a web/mobile application used to facilitate secure medical correspondence between healthcare professionals. The application streamlines communication and ensures for friendly and intuitive user experiences. As a tester, the goal is to ensure the fundamentals of the application are working as intended, as well as exploring any other functionality and testing for faults.

2 Scope

The exercise will be centered around tests for the basic specified functionality given in the brief: a successful login system, pin code registration/re-entry upon locking the screen and sending existing contacts messages.

3 Test Environment

A Windows/Mac desktop environment with sufficient internet capabilities, as well as access to the following browsers:

Google Chrome version 60 and above.
Microsoft Edge version 16 and above.

An iOS/Android mobile environment with sufficient internet capabilities, with access to the Google Play Store or the Apple App Store.

4 Test Cases

The web-application was tested using a combination of Python and Selenium automated-testing technologies.

The decision to create automated tests was driven by the fact that the web-application could generate/load new content relatively quickly, thus making automated navigation much simpler, effective and time efficient.

Tests operated on certain assumptions that the user would make, however precautions were taken in the form of things such as try/except blocks and timeout exceptions to allow testing to flow smoothly and for all errors to be handled appropriately.

Test Setup

Tests were automated using Python and Selenium, with each fundamental testing component being broken up into its own separate section for the sake of clarity and ease of use.

Each component could be accessed on its own in order to perform individualized/fragmented testing, however a master test script was primarily used in order to perform regression testing of previous components and ensure previous functionality was never compromised when adding additional tests.

All tests were automated in very similar ways; in short, the page source was parsed to identify key elements needed to be filled out/clicked/interacted with in order to advance the application further. Elements were identified using their unique ID values, CSS selectors and/or by their XPATH values.

The tests relied on waiting for the webpage to fully load before parsing through the document to look for elements. This was achieved through implementing small wait times for each test process to ensure elements could be located first.

Login Tests

Login tests were carried out in 2 key stages; the **first** being navigating to the login page and finding the sign-in button and clicking it.

The **second** being identifying the username and password fields necessary for logging in, sending through their respective credentials and clicking submit, and then ensuring the login was successful by searching the page for the existence/lack of an error message.

Pin Code Tests

Pin code tests were automated in the same manner as the login tests; however, 2 different versions of the test were designed in order to accommodate for both scenarios.

In the **first** case, the user must enter their pin number for the first time, as this is the first instance of the user logging in properly to the application. The user must pick a 4-digit code and enter it twice into the application in order to register it to their account. Both fields are located within the page and filled out using the same pin code, and the 'next' button is subsequently pressed.

In the **second** case, the user has already registered their pin number, so they must only enter their old pin code in once in order to login to their account again. This occurs if the user is signing in again and the browser cache holds their other login details already, or if the user locks the application screen and is trying to log back in.

Message Sending Tests

For sending messages, an existing conversation was looked for within the page source and then clicked on. Once selected, the conversation's message box was found and a message was loaded in to be sent.

Sending could be handled either by navigating to the send button to the right of the message box and actioning a click, or by imitating a regular 'Enter' keystroke in the message box.

Other message types such as attachments, images and emojis could all be sent using similar methods to regular messages, only at the cost of a few additional elements to search for and click on.

Bonus/Extra UI Tests

Extra tests included testing interaction with the navigation menu, specifically the profile div located in the top right of the web-application. All areas of the drop-down menu could be navigated in similar ways to the rest of the application, however some sections opened and redirected to new tabs. The application could be returned to by imitating the keystrokes for "ctrl + tab" or by closing the window.

5 Results, feedback and analysis

- When a user has locked their screen and is attempting to regain access to their account, incorrectly entering a pin will enable a modal pop-up to block part of the screen, and only by closing/acknowledging the pop-up are you then able to return to re-enter your pin. This is different to the inline error message received for a weak pin code/non-matching pin code upon first time entry.

- As the page content is generated automatically, a delay is needed when looking for elements within the page source in order to carry out testing, slightly hurting the overall efficiency of testing the application.

- Testing pin codes operates under the assumption that the user will enter a 'strong' pin, however nowhere on the page does it define what constitutes as a 'weak'/'strong' pin combination. This may hurt user experiences if they decide to opt for a pin they wish to remember easily which is then unexpectedly rejected.

- Changing tabs within the extra tests can cause some problems within testing, as switching back to another tab using the "ctrl + tab" method will only look for the next tab, not necessarily the previous one. Closing the newly opened window would ensure a safe return to the application, however this may not always be desirable within testing. Similarly, checking the current window URL to match against the desired URL and then swapping if they do not match may be an arbitrary and time-consuming method.