Testing Report Memo

To: Professor Pisano, Professor Alshaykh, Professor Hirsch

From: ConnectBU (Hussain Albayat, Yousuf Baker, Benjamin Chan, Nadim Elhelou, Damani Phillip)

Team: 3

Date: 02/26/2021

Subject: 2nd Prototype Testing Report

Equipment and Setup

Hardware Equipment:

- Windows or Mac Laptop/Desktop that can run modern web browsers

Software Equipment:

- Front-end development
    - React.js
    - Material UI
    - Redux
    - Google Single Sign On
- Database
    - mySQL
    - Amazon Web Services Relational Database Service
- Back-end development
    - Python
    - Flask
    - Authentication/Authorization (JSON Web Tokens)
    - Messaging API (SendBird)
    - Search API (ElasticSearch)

For the prototype testing, we demonstrated through screen sharing via Zoom. We showed all the web application's functionalities that have been implemented so far, including but not limited to navigation across pages, Google Single Sign On, Redux state management, cookie management, ElasticSearch, etc.

In order to test the prototype from any machine, complete the following setup:

1. Go to ConnectBU's GitHub repository and clone the repository.
2. Install or update npm if you haven't already.
3. Install or update Python (Python 3.8.x) if you haven't already.
4. In the cloned repository, navigate to the backend folder and install the project's Python module dependencies by running the following:

   $ pip install -r requirements.txt
5. Run the following commands to run the backend:

   $ export FLASK_APP=app.py

   $ flask run
6. In the cloned repository, navigate to the frontend folder and install the project's node dependencies by running the following:

   $ npm install
7. Run the following command to run the web application

   $ npm run start
8. In your browser, navigate to "http://localhost:3000" to view the web app.

Functions Tested/Measurable Criteria

The functions tested include

1. Navigation through the different pages of the web application seamlessly and with ease.
    a. Able to switch between pages with minimal loading time, correct page loading, and loading page graphic shows while page is rendering.
2. Signing in with BU email address/Restricting log-in when not with a non-BU email address.
    a. Google SSO correctly logged in the user with the corresponding messaging showing the status. An error message was shown when using a non-BU email and a success message was shown when using a BU email.
3. Seeing a signed-up user's profile information on the profile page.
    a. Name, major, and year of the corresponding user were queried from the database and was rendered on the page.
4. Sending and receiving messages through the chat page.
    a. Messaging API was able to store messages and chats between individuals, and groups, and messages are able to be correctly sent and received.
5. We are able to submit a search query and view the results.
    a. Search term was able to be typed into the search bar, filters selected, and the correct results shown based on the search term and filters/lack-thereof chosen.

Testing Procedure

We employ a rigorous testing protocol in order to demonstrate these functions sequentially, except for the first function (page navigation), which is tested passively throughout the entire testing procedure. We begin by launching the webapp, and showing that the landing page loads correctly. Following this, we navigate to the login page and show that BU emails are able to log in, whereas non BU emails are faced with an error message saying that a BU email is required to login. This login state is also shown to be maintained across page navigation and refreshes until the user explicitly logged out. The third function is tested by navigating to the profile page and showing that the users data that they either filled in during signup, or edited after that, correctly shows on the page. The messaging capability is then tested by navigating to the chat page, and showing that the chat API (sendbird) is correctly integrated into the webapp, with sendbird users created according to the existing users in the database (once a new person signs

up, we automatically create a sendbird profile for them). The chat API was shown to function correctly, with new chats between users (both singular and multiple) able to be created, messages able to be sent and received, files able to be sent and received, and chat history correctly stored. The last function is tested by navigating to the search page, selecting the relevant filters, and searching for the existing profiles using different search terms and showing that the correct results are returned.

Conclusions

From our testing, we conclude that all of the above functions performed at the standard set for this round of testing. By employing the testing protocol outlined above, we adequately demonstrated the functionalities set out for testing. By going through the function testing protocol outlined above, we showed that users can seamlessly navigate through the web app's pages with little-to-no lag time. For page renders that take slightly more time than average, a loading screen graphic is displayed. We also demonstrated that users are limited to registering with a BU email address and status messages are displayed to inform users of whether or not their registration was successful. When a signed-in user navigates to their profile page, their information is queried from the database and displayed on the page via distinct tabs. Our testing showed that users are able to search for other users by filtering based on major, class, lab, etc. and entering a term to find related users. Upon submitting their search query, the search page displays the status of the search process and returns relevant results (if any). We also demonstrated that a signed-in user can navigate to the messaging page and message other individuals or groups. We confirmed that the recipient of the messages could view them when logged in.

However, there are a few caveats to be worked on for future iterations. For example, we need to make sure that all the data (other than the student name, id, and major), are stored in the correct tables in the database. This also includes implementing and testing the endpoints required to add these data to the database, as well as having some form on the profile page to edit this information. Additionally, while the searching page and capability were implemented, the feature was on a separate branch due to workflow, similar to many of our other features. As a result, we still need to merge and connect these to the rest of the application to complete our user flow.