Boston University
Electrical & Computer Engineering
EC464 Capstone Senior Design Project

User's Manual

ConnectBU

Submitted to

Osama Alshaykh
8 St Marys St
Boston, MA 02215
(858) 361-9043
osama@bu.edu

by

Team 3
ConnectBU

Team Members

Hussain Albayat hussainb@bu.edu
Yousuf Baker ybaker@bu.edu
Benjamin Chan chanben@bu.edu
Nadim El Helou nadimh@bu.edu
Damani Phillip djphilip@bu.edu

Submitted: April 13, 2021

User Manual

Table of Contents

Executive Summary

BU students lack personalized academic/professional/extra-curricular advising resources—it's difficult to meet other students to ask for advice. Support is often decentralized and up to the student to parse from distributed resources (clubs, faculty, events, etc). Connect BU is a platform that seeks to democratize access to personalized advising by allowing students to search and connect with each other based on school/major, clubs, classes, research, and on campus employment. The final deliverable will be the minimum viable product (MVP) of a web app that delivers this experience by allowing students to search by these criteria, and to connect through a built-in messaging functionality, all while maintaining security and privacy of student information. This webapp will be built primarily using an agile software development approach where most of the work will be done in one-to-two-week sprints, which allows for continuous testing and integration of the web app features and pages. The main innovative feature is the search functionality, which allows for a granularity in search criteria not often found in general use platforms, and the design of the profile information fields, which allows for storage and display of student specific information.

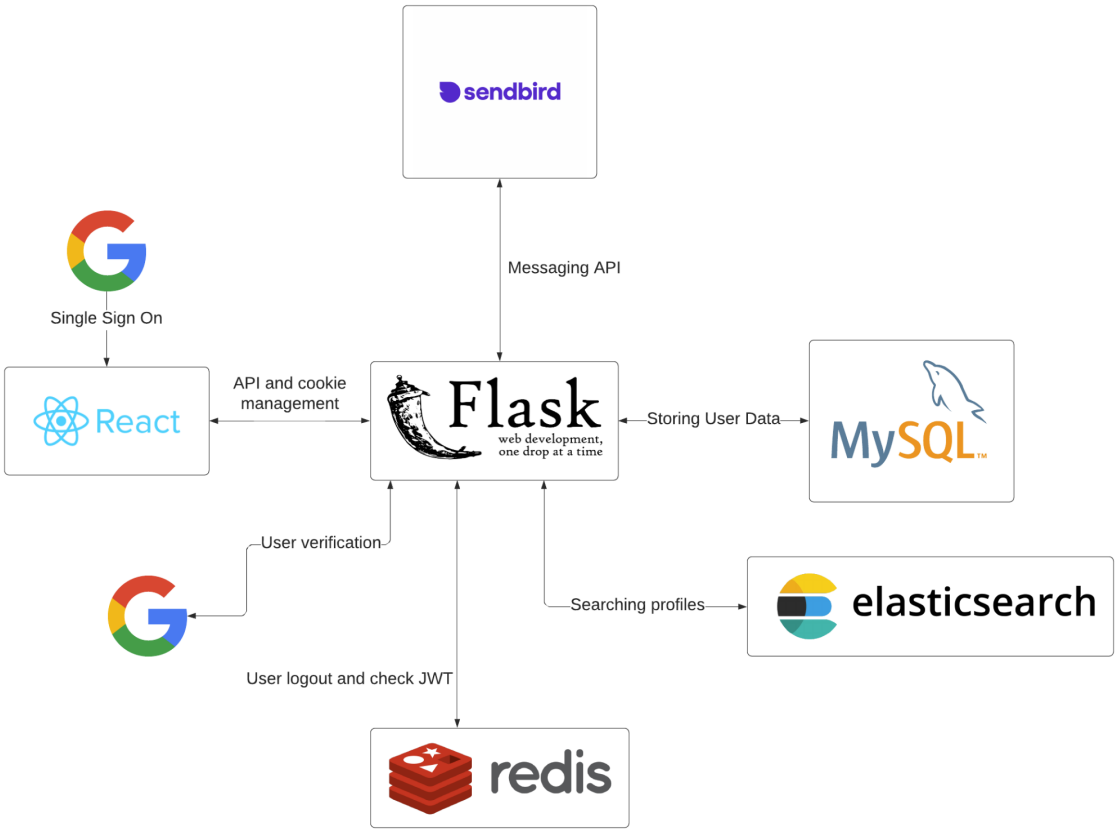
1 Introduction

The main problem our group is focusing on is that BU students lack personalized academic/ professional/ extra-curricular advising resources. It's difficult to meet other students to ask for advice; support is often decentralized and up to the student to parse from distributed resources (clubs, faculty, events, etc.). Though this is endemic in the general college landscape, and tends to exacerbate pre-existing issues in college preparedness of students from different demographics and socioeconomic backgrounds. At Boston University support resources exist, however they are oftentimes distributed and difficult to find and navigate, such as in the case of looking through the 300+ groups at BU, developing a network of upper-classmen, or even gaining the knowledge of what resources and opportunities exist. There are also certain trade-secrets to finding opportunities that are only learned by talking to other students, such as reaching out to the PhD students, as opposed to the P.I. of a lab you would like to work with.

Hence, we have built a unified platform to allow for easier communication between members of the university (mainly students and faculty). This platform is built primarily using agile software development paradigms, and using a cloud based architecture that aggregates a variety of microservices to provide the necessary functionality (search, chat, connecting) for our application. From reading about and observing student experiences at BU and elsewhere, having access to this network of upperclassmen can make the university experience holistically smoother and more efficient, especially for first generation students. **Our platform intends to circumvent this issue by democratizing access to this network**, **helping to decrease the disadvantage faced by students who are not able to access that same network.**

2 System Overview and Installation

2.1 Overview block diagram



2.2 Physical description.

Given that the final product is a website/online platform, there is no physical interface.

2.3 Installation, setup, and support

Local Installation Instructions

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 in a terminal to run the flask app:
   $ export FLASK_APP=app.py
   $ flask run --port=5000
6. Run the following commands in a second terminal to run the search API
   $ export FLASK_APP=search_api.py
   $ flask run --port=4000

7. In the cloned repository, navigate to the frontend folder and install the project's node dependencies by running the following:
   $ npm install
8. Run the following command to run the web application
   $ npm run start
9. In your browser, navigate to "http://localhost:3000" to view the web app.

Remote Installation Instructions

1. Create an AWS account
2. Create a Redis server using ElastiCache and change the ip address of the Redis server in /test_scripts/app.py to the one that was just created
3. Create 4 EC2 Instances (can be the t2.micro size) with Ubuntu Server 20.04 LTS
4. Instructions for each EC2 Instance (assuming that each machine is SSH'd into):
    1. Backend
        i. Copy over all of the /backend files into a directory
        ii. Follow the same instructions for installing the Flask app locally
        iii. To run the Flask app, do the following command:
        ```
        sudo env/bin/python3 app.py
        ```
    2. Frontend
        i. Install nginx and run it:
        ```
        sudo apt install nginx
        sudo systemctl start nginx
        ```
        ii. Add the following lines to /etc/nginx/sites-enabled/defaultabove location /
        ```
        location /api/ {
             proxy_pass https://api.connectbu.com;
        }
        ```
        iii. Copy over all of the /frontend files
        iv. Run the following commands to install the dependencies and build the app:
        ```
        npm install
        npm run build
        ```
        v. Remove all files in /var/www/default and copy over build files with the following:
        ```
        sudo rm -rf /var/www/default/*
        sudo cp -r build/* /var/www/default
        ```
    3. Redis
        i. Copy over /test_scripts
        ii. Install the requirements with the following command:
        ```
        pip install -r requirements.txt
        ```
        iii. Run the server with the following command:
        ```
        python3 app.py
        ```
    4. ElasticSearch
        i. Copy over search_api.app
        ii. Run in the same manner as local but with --host=0.0.0.0 in the flask run

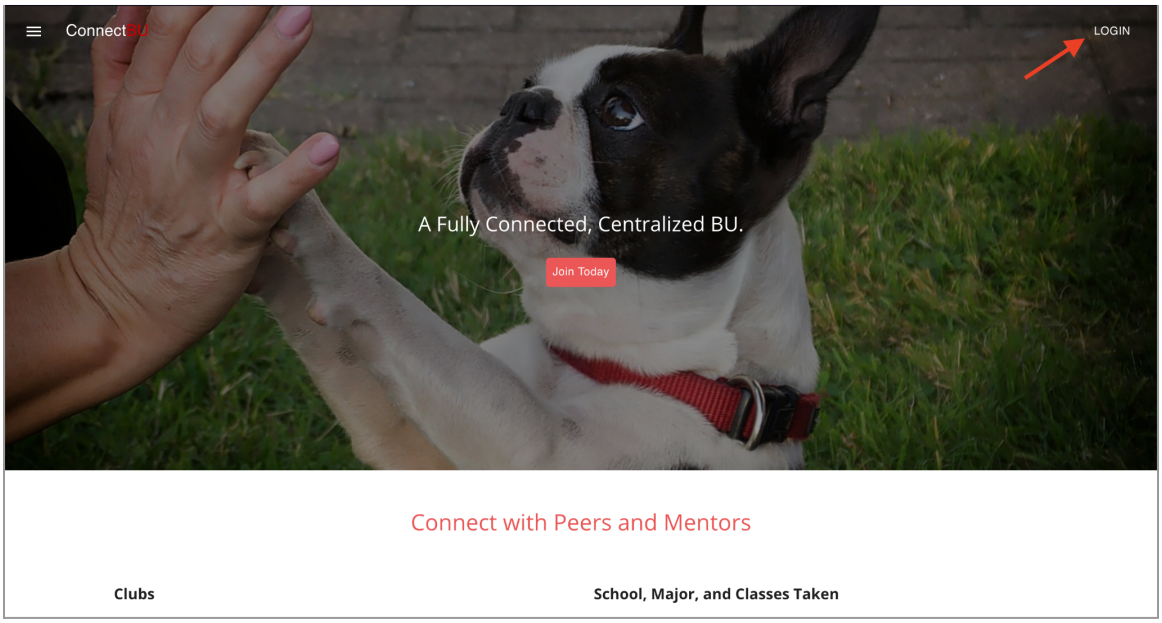
5. Use Route53 as the DNS server for the website (domain can be purchased elsewhere, such as from Namecheap)
6. Register a Public Certificate from Certificate Manager
7. Create two Application Load Balancers (ALBs) and configure them to use the public certificate and HTTPs
8. Create two target groups, one pointing to the backend EC2 instance and one pointing to the frontend EC2 instance
9. Change the ALBs to each point to one target group
10. Then visit the registered domain to access the site

3 Operation of the Project

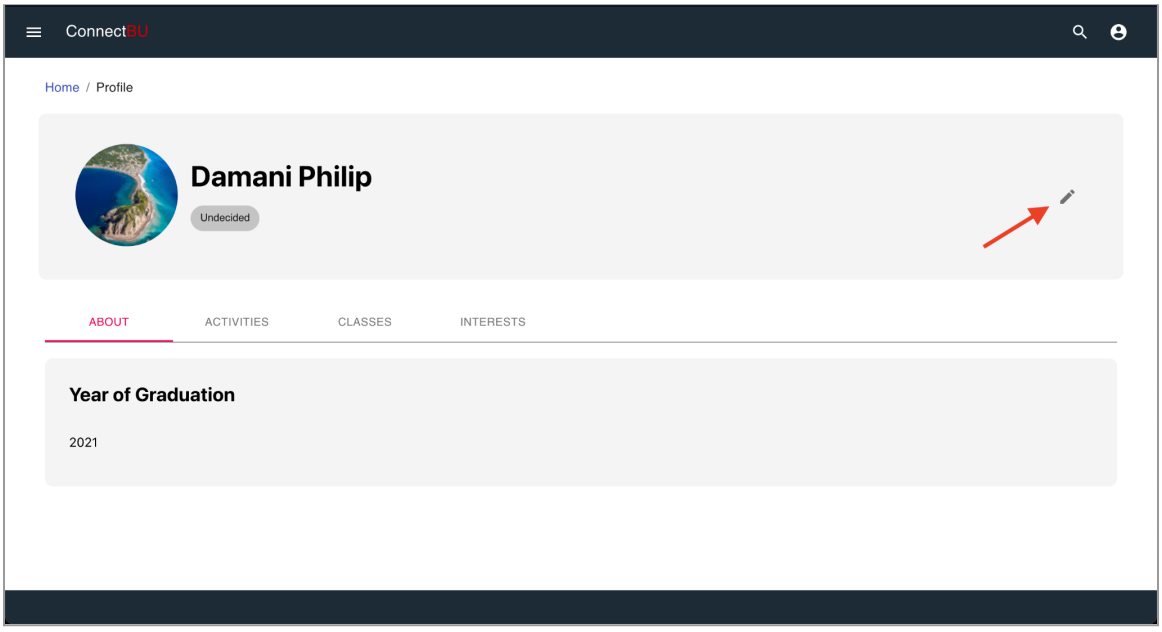
3.1 Operating Mode 1: Normal Operation

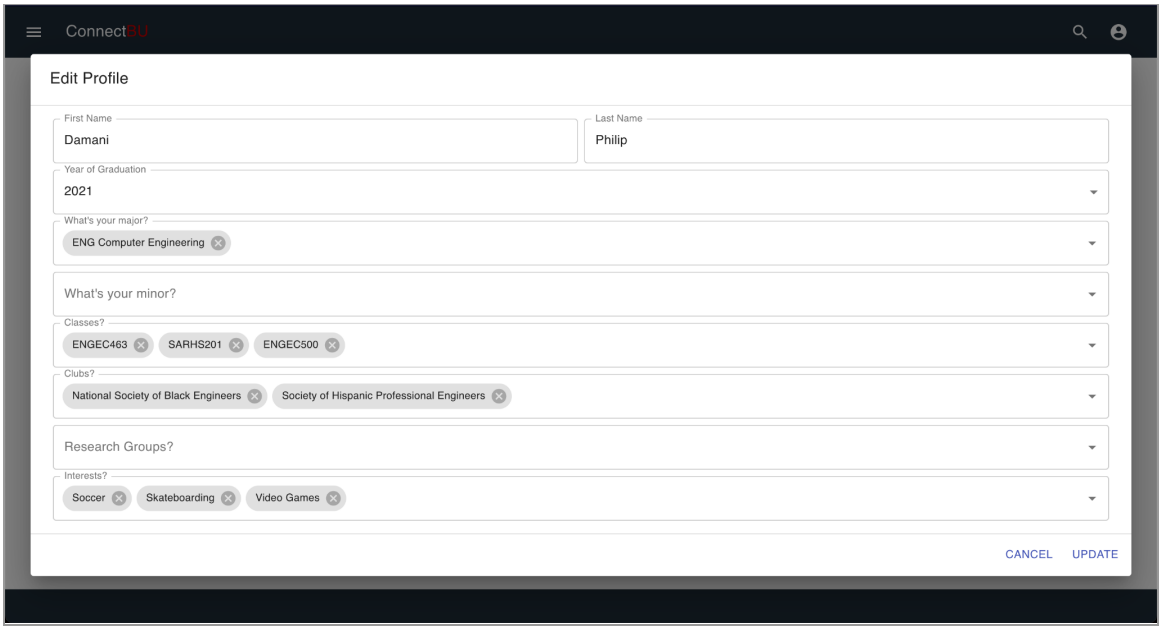
In a normal operating mode, a user will interact with the web application as follows:

1. The user should navigate to https://connectbu.com/
2. The user should login with a BU email address. They will be redirected to their profile page.

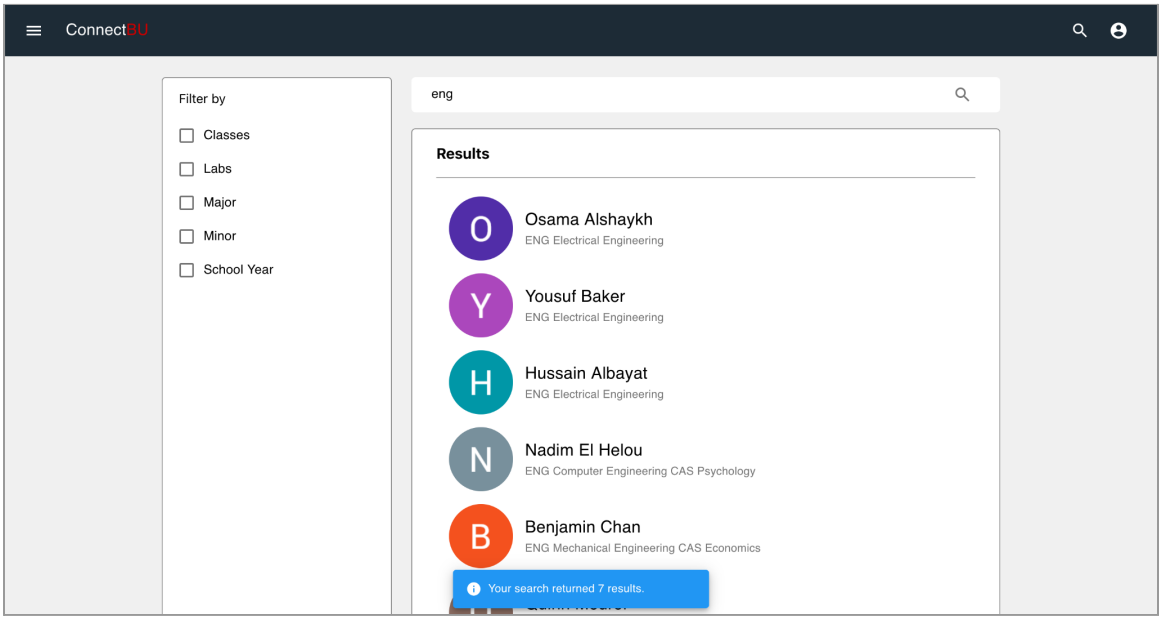


3. The user should click on the pencil icon to edit their profile and add personal, education, and extracurricular information as needed. They will be prompted with a pop-up indicating whether their edit was saved or not.

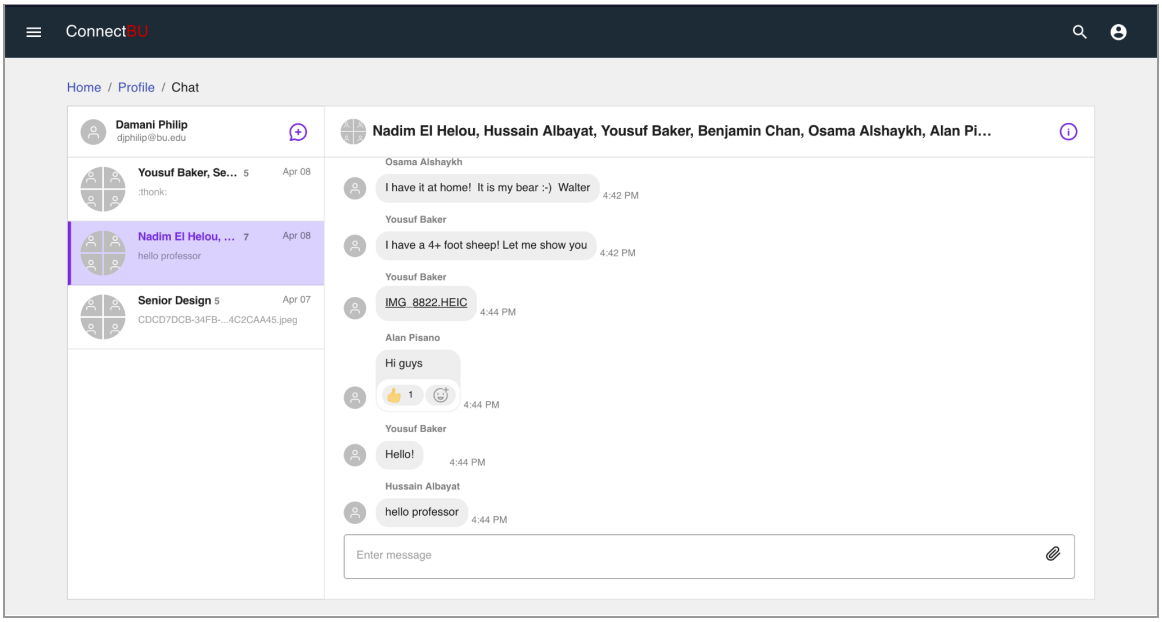




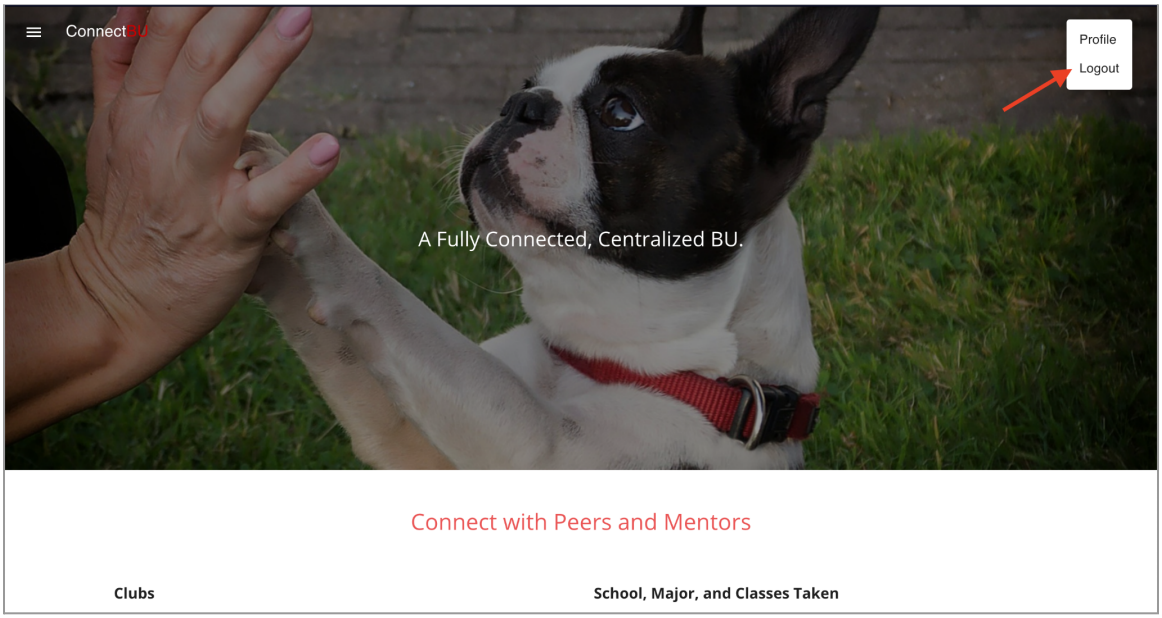
4. To search for another user, the user should navigate to the search page and enter a search query. They may choose to filter the search based on certain criteria by clicking the checkboxes. Upon submitting a search, the web application will generate a pop-up showing the number of results and display any results on the same page.



5. To message another user, the user should navigate to the chat page. There, the user can create individual/group chats and message other users.

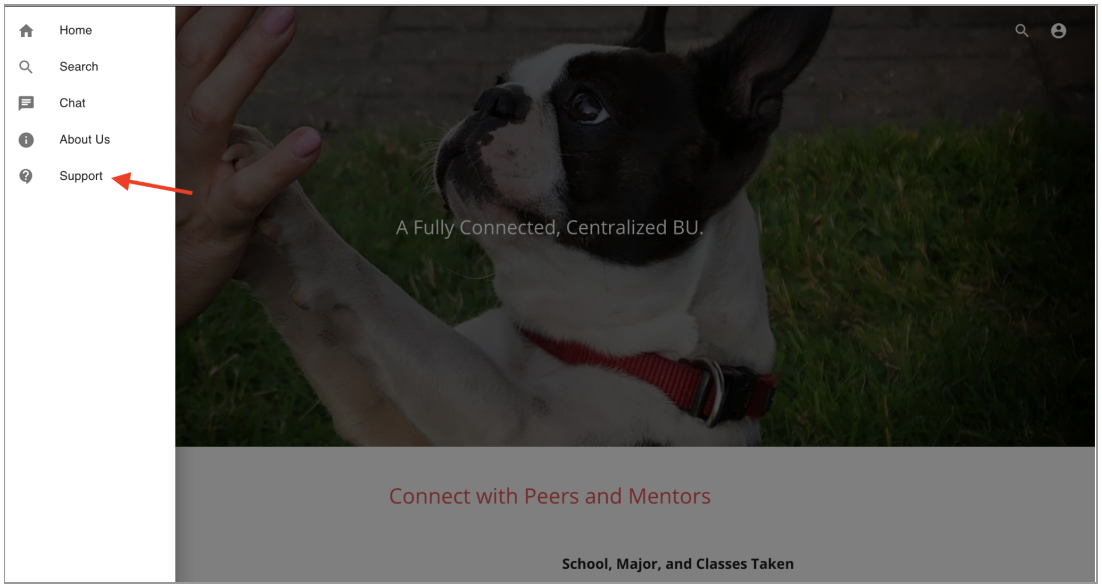


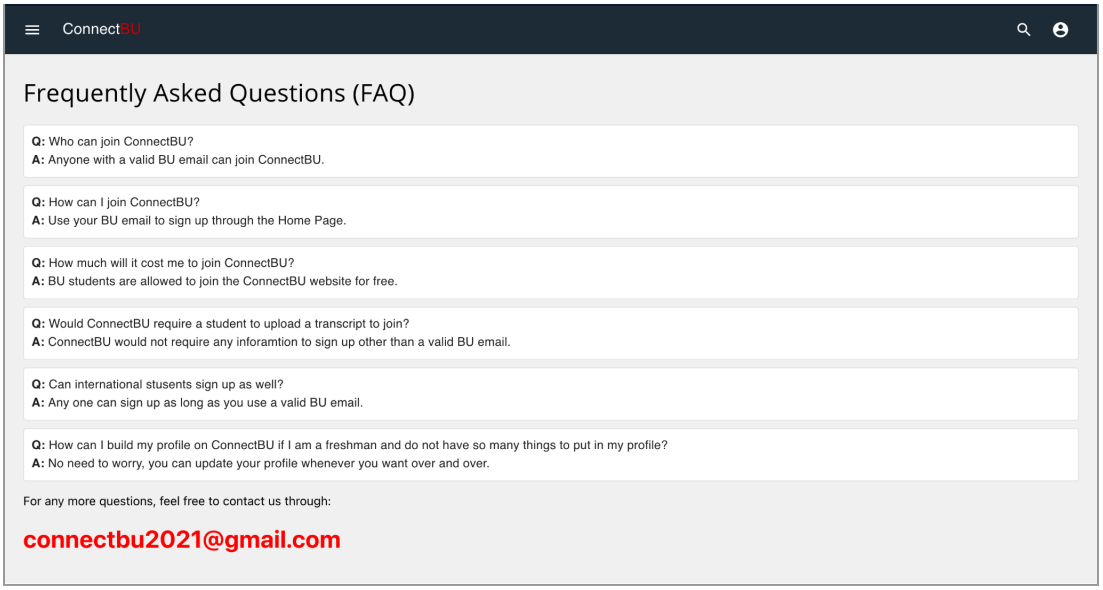
6. Upon closing the web application, the user's login information may be stored for up to 12 hours at a time. To logout, the user should click the profile button and click the logout option.



3.2 Operating Mode 2: Abnormal Operation

Given that the web application is primarily hosted on AWS services, the most likely abnormal states include any service outages or data loss that may occur in our provisioned EC2, RDS, Elasticsearch, or Elasticache instances. Should any of these services fail, the user will be notified that the server is not responding as intended and instructed to try using the application at a later time. In any such scenario or one involving the server producing erratic data (e.g. profile information not being stored correctly in the database), the user can navigate to the support page and message the support email address.

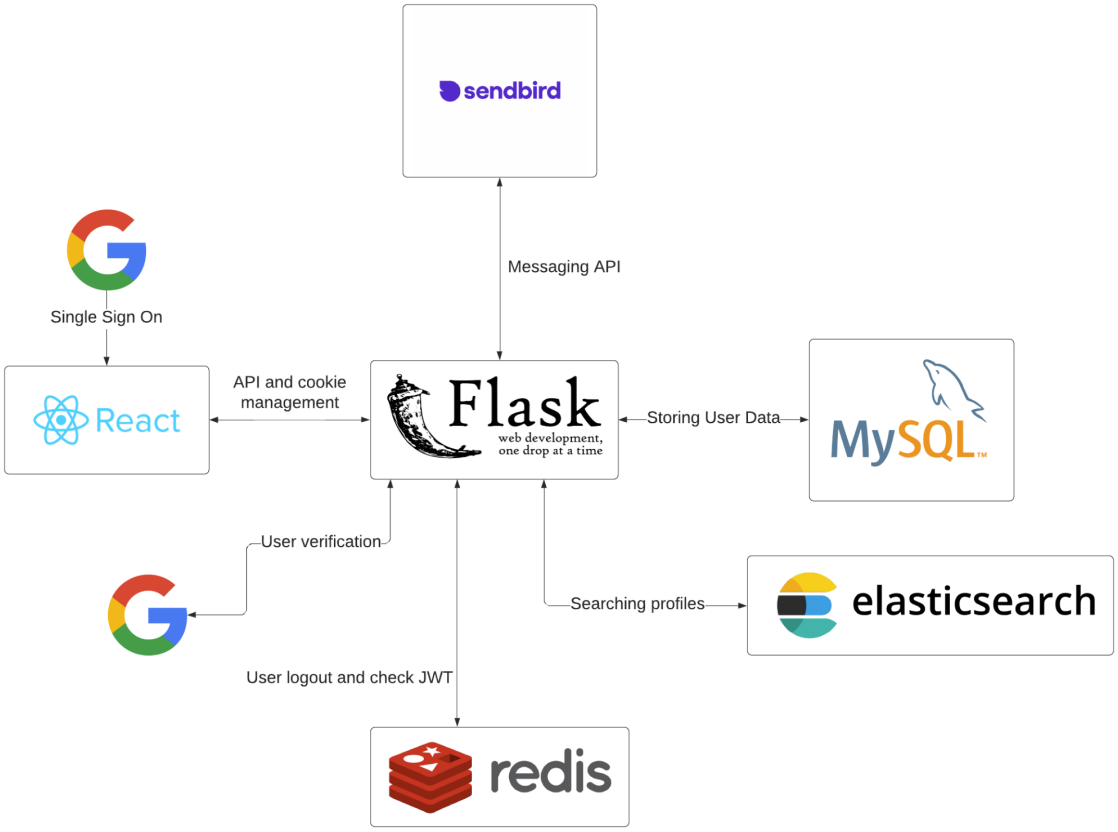




3.3 Safety Issues

The primary safety concerns include the hackability and cyber security of the webapp. Since the web app aggregates a variety of microservices either offered by AWS or Sendbird, which are robust and have built in encryption and cyber security safety measures. Thus, hacking these portions of the webapp would require hacking these third-party service providers. However, since we are making our web app accessible through an endpoint, there is the potential for exploitation through that medium. Although we have limited access to the necessary ports for security, there are not many other obvious preventative measures to take for the robustness and security of our webapp, and we will react to new threats swiftly and accordingly. The backend itself is secured and closed off with JWTs, but this is still potentially vulnerable since the information is being stored client-side and runs the risk of being stolen by adversaries.

4 Technical Background



Our project is built using a modular design approach, where each of the individual components (generally cloud based microservices) are worked on independently, and then connected to the react frontend or flask backend based on their function. The connections between the individual components and the front/backend are shown in the diagram above. Each component in the diagram serves a specific function as explained below:

1. React: React is an open-source, front end, JavaScript library for building user interfaces or UI components. We used React, as well as Material-UI, for the entire front end development.
2. Flask: Flask is a micro web framework written in Python. We use Flask to manage our entire backend. It allows us to query the database, communicate with the messaging service, manage user login and logout, and manage cookies and create API endpoints for communication with the front-end.
3. Sendbird: Sendbird is a third party microservice that provides our chatting services.

4. MySQL: MySQL is an open-source relational database management system. We created a MySQL database for storing all of the users' information. This relational database was modelled using an ER diagram and then translated into SQL code. The database holds information such as their name, major, minor, class year, clubs, interests, and other information.
5. Elasticsearch: Elasticsearch allows you to store, search, and analyze huge volumes of data quickly and in near real-time. It's able to achieve fast search responses because instead of searching the text directly, it searches an index. It is especially optimized for searching through rich text fields, and returning results based on filter choice and ordering them based on relevance to the keyword.
6. Redis: Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker. It allows us to manage user logout and JSON Web Token authentication.
7. AWS: Amazon Web Services is a subsidiary of Amazon providing on-demand cloud computing platforms and APIs. Our entire project is hosted on AWS:
    a. multiple EC2 instances for hosting the website and backend
    b. AWS Elasticsearch Service
    c. AWS Relational Database Server for MySQL

5 Relevant Engineering Standards

Here are all the Engineering standards we followed for our project

1. REST APIs: Our backend is built using the Flask-RESTful library, which conforms to the REST API framework. This consists of our backend taking requests through HTTP, processing the request, and then giving a response with a JSON object. Each request is stateless and no information is transferred between each request.
2. SSO (Single Sign On): SSO is an authentication method that enables users to securely authenticate with multiple applications and websites by using just one set of credentials. Users can securely login to our website with their BU account (Google account) credentials. Our backend then verifies the credentials with Google to complete the authentication process.
3. JSON Web Token (JWT): JWTs are tokens that are used to authenticate users for certain applications. These tokens are signed using a private key by the issuer and prove to the service provider that the user is allowed to use the service. In our application, we use JWTs to make sure that users are allowed to use the endpoint after signing on and will maintain this logged in state for twelve hours.
4. Cookies: Cookies are stored client-side in the user's browser and are used to maintain state for web applications. In our case, we use cookies to store our JWT, which is our method of securing our endpoint. This allows users to use the application without the need to login on every refresh of the page as the cookie is sent on every request.
5. HTML 5: HTML 5 is the latest version of the markup language. This language specifies the structure of the content presented on each page. We ensure that our application is compliant with the current standards by making sure each tag used is supported and not deprecated.
6. TLS 1.3: TLS is used in the security layer of HTTPS, which is an extension of HTTP that encrypts the information transmitted using the protocol. TLS is used to encrypt this information. We use HTTPS to further protect our users' data when interacting with the website. The information sent from the user through the frontend goes over HTTPS.
7. ECMAScript: ECMAScript is a programming language standard for event-driven, client-side scripting that is commonly implemented through the programming language Javascript. ECMAScript ensures that web pages across various types of browsers exchange and use data in conjunction and without restriction. By using Javascript to develop our user interface, we are automatically in compliance with ECMAScript.

6 Cost Breakdown

Consider your EC464 prototype to be the *alpha* version. The next unit made, according to your engineering specifications and design, would be the *beta* version. Later a manufacturing version or release-version would be made.

What would be the cost of your ***beta*** unit when it is created? This should assume market costs, i.e. no donations, no picking through the customer's parts closet.

You can edit the table below to describe the project expenses for the beta version. It is not necessary to provide every detail about parts, labor, and services in your cost breakdown. Decide upon a level of aggregation of investment and group costs accordingly.

Project Costs for Production of Beta Version (Next Unit after Prototype)				
Item	Quantity	Description	Upfront Cost	Monthly Cost
1 Elasticsearch Cache	1	(1 instances of type Redis Standard cache t2.micro OnDemand)	0	12.41
2 RDS for MySQL	1	Storage for each RDS instance (General Purpose SSD (gp2)), Storage amount (30 GB), Quantity (1), Instance type (db.t2.micro), Deployment option (Single-AZ), Pricing strategy (OnDemand)	0	15.86
3 Amazon Elasticsearch Service	1	Storage amount (30 GB), Storage for each Elasticsearch instance (General Purpose SSD (gp2)), Managed storage amount (per UltraWarm instance) (1.5 TB), (1 instances of type c5 large.elasticsearch Compute optimized OnDemand EBS Only), (0 instances of type c4 2xlarge.elasticsearch Compute optimized OnDemand EBS Only), Number of nodes (0), Instance type (ultrawarm1.large.elasticsearch)	0	91.25
4 Amazon EC2	1	Operating system (Linux), Storage amount (30 GB), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month), Workload (Daily, (Workload days: Monday, Tuesday, Wednesday, Thursday, Friday, Baseline: 3, Peak: 4, Duration of peak: 6 Hr 60 Min)), Snapshot Frequency (2x Daily), Amount changed per snapshot (3 GB), Advance EC2 instance (t2.micro), Pricing strategy (On-Demand)	0	41.03
5 Sendbird Monthly Free Trial	1		0	0
		Beta Version-Total Monthly Cost (USD)		160.55

7 Appendices

7.1 Appendix A - Specifications

Technical Measure
User is able to login using BU email (provided by google SSO)
Profile stores information on 7 main fields: school, major, year, classes, labs, clubs, interests Major, minor, classes, clubs, labs, and interest all able to be chosen from preloaded lists Database able to store information for **at least 1000** students
Chat functionality through sendbird, able to create direct and group chats, chat history is able to be saved
Search service provided through elasticsearch, able to filter keyword search based on following fields: classes, labs, major, minor, school year Search time no longer than **2** seconds, search results sorted by relevance to search term

7.2 Appendix B – Team Information

Hussain Albayat is a graduating Electrical Engineer off to pursue a career at Saudi Aramco. He is interested in ham radios and software development.

Yousuf Baker is a graduating Electrical Engineer with interests in all things deep learning, cyber-physical systems, and sustainable AI.

Benjamin Chan is a graduating Computer Engineer with interests in cloud computing, distributed systems, machine learning, and cybersecurity.

Nadim El Helou is graduating with both a Computer Engineering degree and a Psychology degree. He is interested in software engineering, psychology, and *avante-garde* and *art* film.

Damani Phillip is a graduating Computer Engineer with interests in cloud computing, machine learning, and software for social impact. He aspires to the ideal of a societal engineer.