# CampusTalk
# CECS 491B (Section 04)
# Binary Bandits

**Emmanuel Rodriguez**
**Edgar Rodriguez-Macias**
**Frank Mancia**
**Daryl Nguyen**
**Kenneth Su**
**Matthew Chung**

CampusTalk - Software Design Document

# Contents

# Section 1 - Project Description

## 1.1 Project Name

**CampusTalk**

## 1.2 Description

CampusTalk is a website application in which students can create different postings and discussions under their own campus domain. CampusTalk promotes interaction and discussion among students attending various California universities. It serves as an opportunity for students to interact with each other and share experiences and questions about student life on a variety of campuses. Students will be able to interact with different posts about their campus, engaging in discussions between the community. The posts made by students can be filtered and voted on, which provides additional functionality for the users.

## 1.3 Updates

CampusTalk has been updated in the Fall 2023 Semester to include new functionalities and UI improvements of the application.
New functionalities

- GIF support
- Image Upload
- Sub comments
- User-Post notifications
- List of supported schools page

UI improvements include:

- Made it easier for the user to create a post by simplifying the UI to a more condensed popup with new functionality (images/gifs).
- Decluttered Views by expanding the contents across the screen and moving the sidebar onto the navbar
- Added hover effects to common places a user might interact with
- Fixed bugs

Backend Improvements:

- Reworked posts component so it can switch views and retain user input once switched back

# Section 2 - Overview

## 2.1 Purpose

The purpose of our web application, *CampusTalk*, is to function as an interconnected community for students and scholars across all college and university campuses in the United States to communicate with one another on a campus-by-campus basis.

## 2.2 Scope

The end-users that we have catered CampusTalk's functions to will be college students with email addresses verified by their education institutions. Each campus will have a forum where verified students of said campus will be registered or subscribed to. Each campus community will have their own subforums based on different aspects of the campus (i.e. Major departments, clubs, organizations, etc.).

## 2.3 Requirements

### 2.3.1 Estimates

| # | Description | Hrs. Est. |
|---|---|---|
| 1 | Create diagrams and system architectures. | 10 Hrs. |
| 2 | Analyze Metrics: (Cost, Defect, Usability) | 10 Hrs. |
| 3 | Establish Database environment & load campus data | 10 Hrs. |
| 4 | Create UI/UX Templates for different components | 15 Hrs. |
| 5 | Back End Development | 180 Hrs. |
| 6 | Front End Development | 120 Hrs. |
| 7 | Testing User Auth. | 50 Hrs. |
| 8 | Testing API/User Comments/User Posts | 50 Hrs. |
| | **TOTAL**: | 445 Hrs. |

### 2.3.2 Traceability Matrix

| SRS Requirement | SDD Module |
|---|---|
| Req 01 - User Registration and Authentication | Module 01 - User Login and Register Page |
| Req 02 - Post-Creation and Management | Module 02 - Create a Post Module |
| Req 03 - Campus Identification | Module 03 - Variety Campus Subdomains |
| Req 04 - Post Tags System | Module 04 - Post Filtration Module |
| Req 05 -Post  Rating System | Module 05 - User Upvote/Downvote Functionality |
| Req 06 - Post Commenting System | Module 06 - Create a Comment Module |

## Section 3 - System Architecture

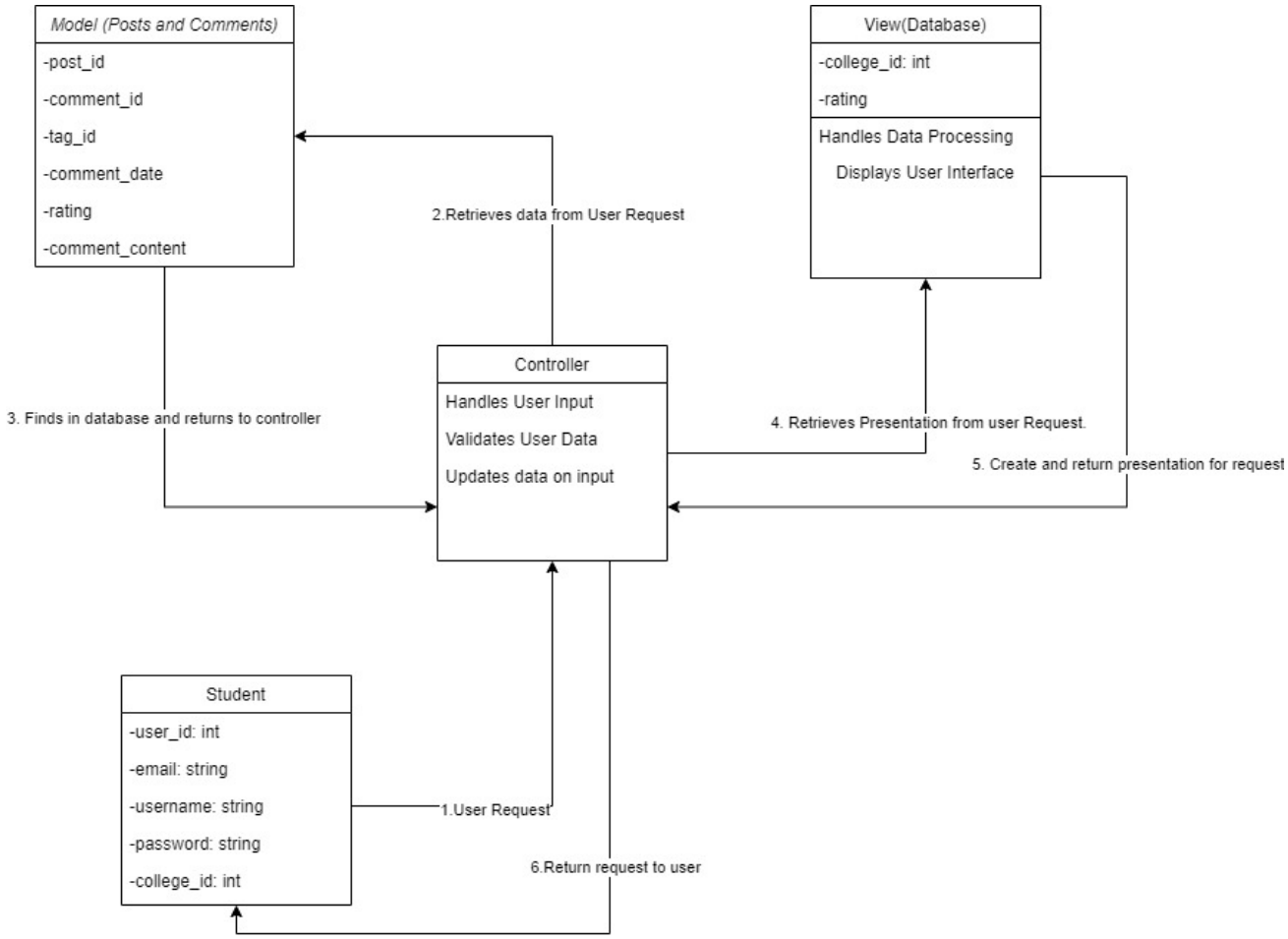
**Model-View-Controller (MVC)** architecture is implemented in CampusTalk. The architectural design pattern referred to as MVC organizes the application logic into three interconnected components: the Model, the View, and the Controller. This organization renders it efficient to create by having a clear logical flow of the architectural behavior. This modular design makes it simpler to develop and maintain the application.

**Model**: The CampusTalk application's data is captured by the model component. It covers the basic functions of user accounts, postings, campus domains, and comments. CampusTalk's model would consist of elements like the User, Post, and Comment classes or modules. The management of data persistence and retrieval from the underlying database or data storage falls under the purview of the model.

**View**: The user interface interactions and data presentation for the user is handled by the View component. The visual components and templates that users interact with when visiting CampusTalk are represented by the View. The client-side backend scripting,React.js and CSS elements that render the user interface and manage user interactions are all included in it.

**Controller**: The Model and the View are linked through the Controller. It handles user interactions, ensures the View presents the updated state of the data, and refreshes the Model in response to user activities. The Controller component of the CampusTalk application would handle user requests, call the appropriate Model functions to retrieve or alter data, and update the View as necessary.

**CAMPUSTALK MVC DIAGRAM**



## Section 4 - Data Dictionary

A brief description of each element in this module or a link to an actual data dictionary

| college_fourm | | |
|---|---|---|
| **Field** | **Notes** | **Type** |
| college_id | Unique Identifier for college | INT |
| college | Name of the college | VARCHAR |
| city | City college resides | VARCHAR |
| state | State college resides | VARCHAR |
| zipcode | Zip Code college resides | INT |
| acronym | Acronym for college name | VARCHAR |
| college_icon_url | Url for the college icon | VARCHAR |
| supported | Binary value if college is supported by our application | BIT |

| comments | | |
|---|---|---|
| **Field** | **Notes** | **Type** |
| post_id | post id that comment was made under | INT |
| comment_id | unique identifier for comment | INT |
| parent_comment_id | id for comment if a comment was replied to it | INT |
| comment_content | comment content | TEXT |
| comment_date | date comment was created | TIMESTAMP |
| user_id | id of the user that created the comment | INT |

| notifications | | |
|---|---|---|
| **Field** | **Notes** | **Type** |
| notification_id | auto generated id for notification | INT |
| post_id | post id where reply was made | INT |
| user_replied | id for user that replied to a post/comment | INT |
| user_op | id for user that will receive notification | INT |
| timestamp | time/date for when reply was made | TIMESTAMP |
| read | dictates if notification should be cleared | INT |

| posts | | |
|---|---|---|
| **Field** | **Notes** | **Type** |
| college_id | college id the post was made under | INT |
| post_id | unique identifier for post | INT |
| user_id | user id for user that created post | INT |
| title | title for post | VARCHAR |

| | | |
|---|---|---|
| content | content for post | TEXT |
| post_date | timestamp of post creation | TIMESTAMP |
| rating | rating for post (likes/dislike) | INT |
| tag_id | id of the tag added to post | INT |
| exclusive | declares if a post should be college exclusive | INT |
| selectedGif | holds url for gif | VARCHAR |
| imageURL | holds path for image upload | VARCHAR |

**tags**

| Field | Notes | Type |
|---|---|---|
| tag_id | unique identifier for tag | INT |
| tag_name | name for the tag | VARCHAR |
| color | hex color code for tag | VARCHAR |

**user_records**

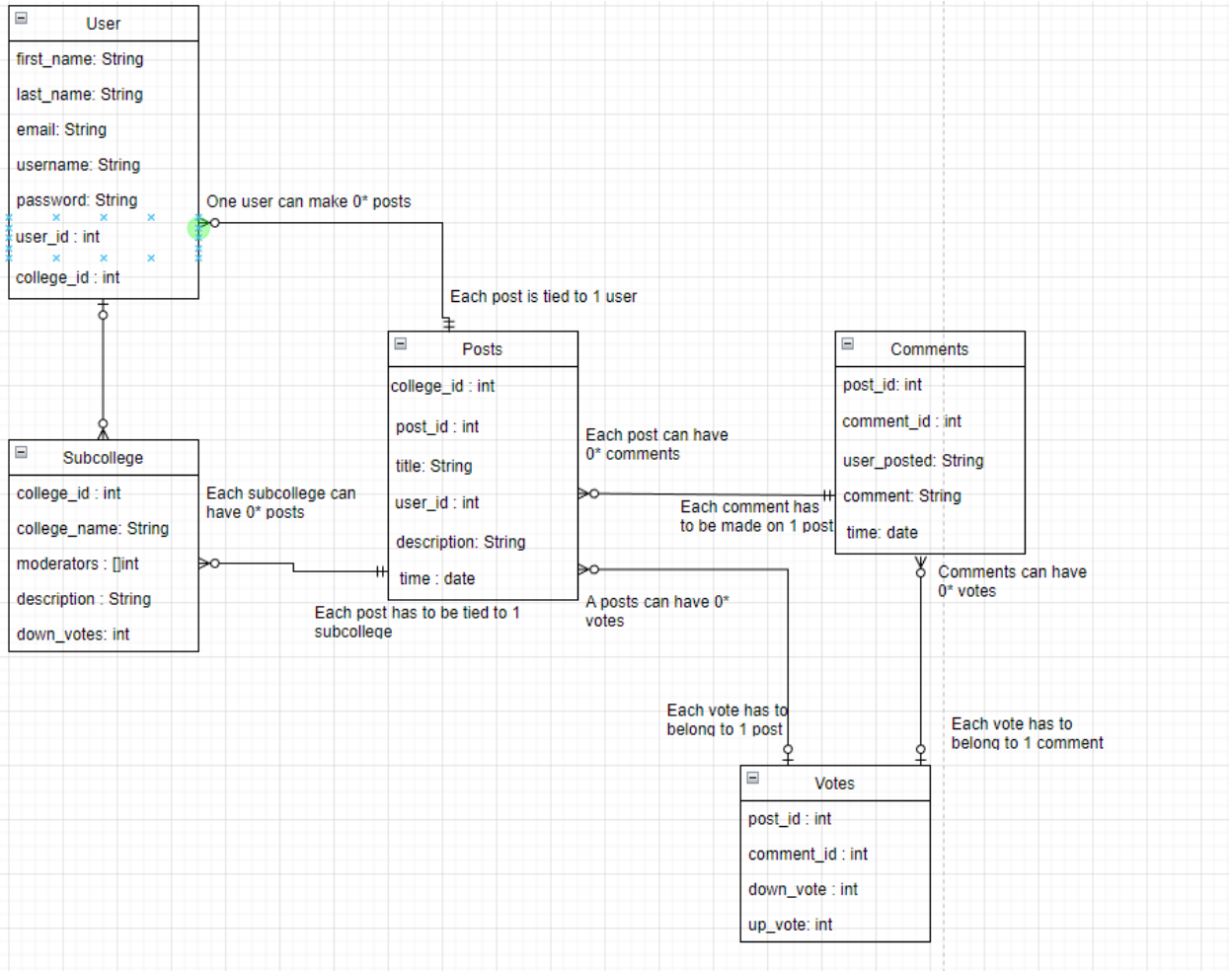
| Field | Notes | Type |
|---|---|---|
| user_id | unique identifier for user | INT |
| full_name | user full name (first and last) | VARCHAR |
| email | user edu address | VARCHAR |
| display_name | user's name that is displayed on site | VARCHAR |
| user_password | user password (hashed) | VARCHAR |
| avatar_url | user avatar picture path | VARCHAR |
| college_id | id for college that user attends | INT |
| join_date | date the user registered | TIMESTAMP |

**votes**

| Field | Notes | Type |
|---|---|---|
| action_id | auto generated value for votes | INT |
| user_id | user id that voted on post | INT |
| content_type | dictates if it is a post or comment | VARCHAR |
| vote | vote value | TINYINT |
| created_at | time/date of vote | TIMESTAMP |
| updated_at | time/date when vote was changed | TIMESTAMP |
| parent_content_id | id of post/comment that was voted on | INT |

# Section 5 - Software Domain Design

## 5.1 Software Application Domain Chart




## 5.2 Software Application Domain

### 5.2.1 Domain Interactions

1. User registration and Authentication: Student registration and accounts require specific school credentials in order to pass authorization to create an account.
2. Forums: Schools are designated and mapped to a specific forum where only students who have that establishment's .edu email are able to access and create posts.
3. Post Creation: Students create posts within their specific .edu forum with titles, post content, and those have designated college ID and post IDs stored in the database, as well as the user ID it is stored to.
4. Post Interactions: Upvotes and Downvotes are identified by int variables stored with the post ID and comment ID.
5. Comments: Comments are identified with strings of the description, the timestamp in which it was created, the post ID that it belongs to, a comment ID that is layered within that post ID, and the user who posted it.
6. Moderation: Our team moderates specific forums and are responsible for adding forums to CampusTalk

# Section 6 – Data Design

## 6.1 Persistent/Static Data

### 6.1.1 Dataset

- Users directly created accounts, posts, and interactions with posts
- Us (The Binary Bandits), as moderators, add new campus forums

### 6.1.2 Static Data

- Campus forums that users are defaulted to joining due to their .edu designation

### 6.1.3 Persisted data

- User accounts
- Posts
- Comments
- Votes on posts
- User settings

## 6.2 GIPHY API

### 6.2.1 Dataset

- Users utilize GifSearch to directly post gifs based on GIPHY's catalog
- GIFs are stored as URLs in the backend based on post_id

### 6.2.2 Persisted Data

- selectedGif stored in backend

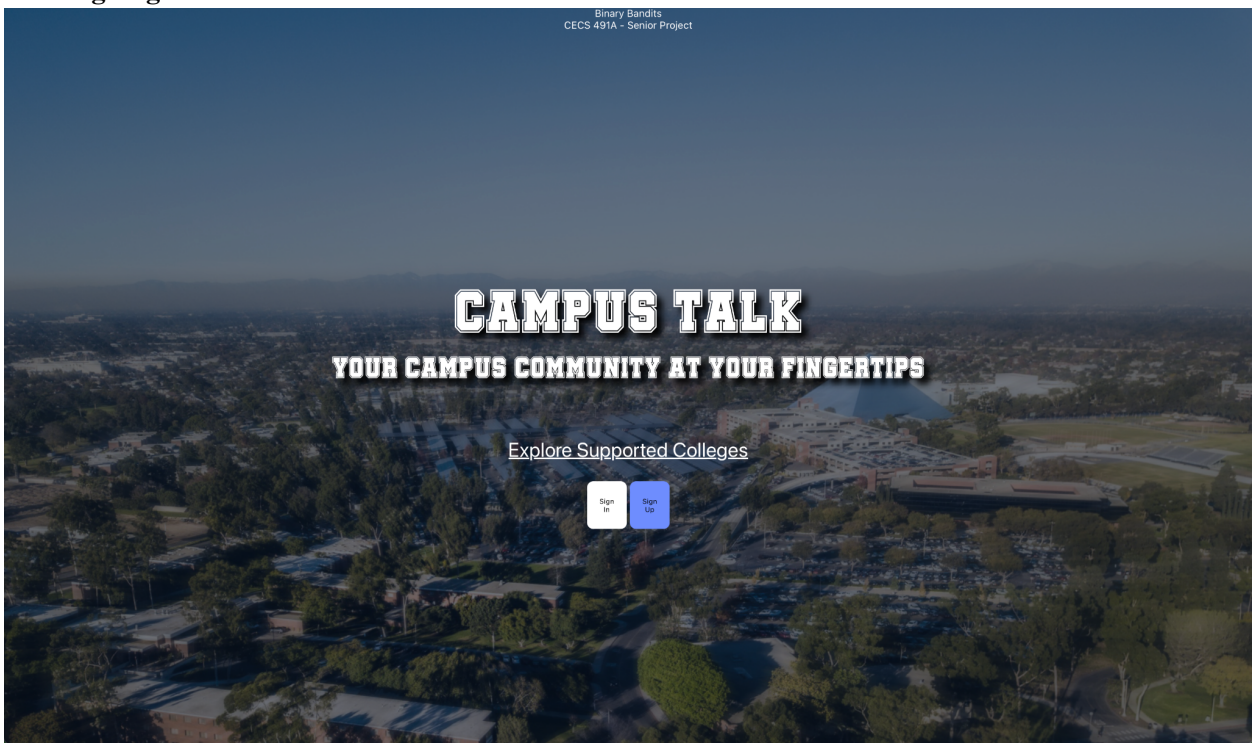
### 6.2.2 Search Endpoint

- CampusTalk has been registered for its own GIPHY Developer Key to grant developers access to using the tool and catalog as part of a media upload option for the web application
- The search enables a catalog of GIF urls in their img form for the user to choose from and this url that is chosen is ported to our database once the user submits their post and uploads the GIF as part of it

# Section 7 - User Interface Design

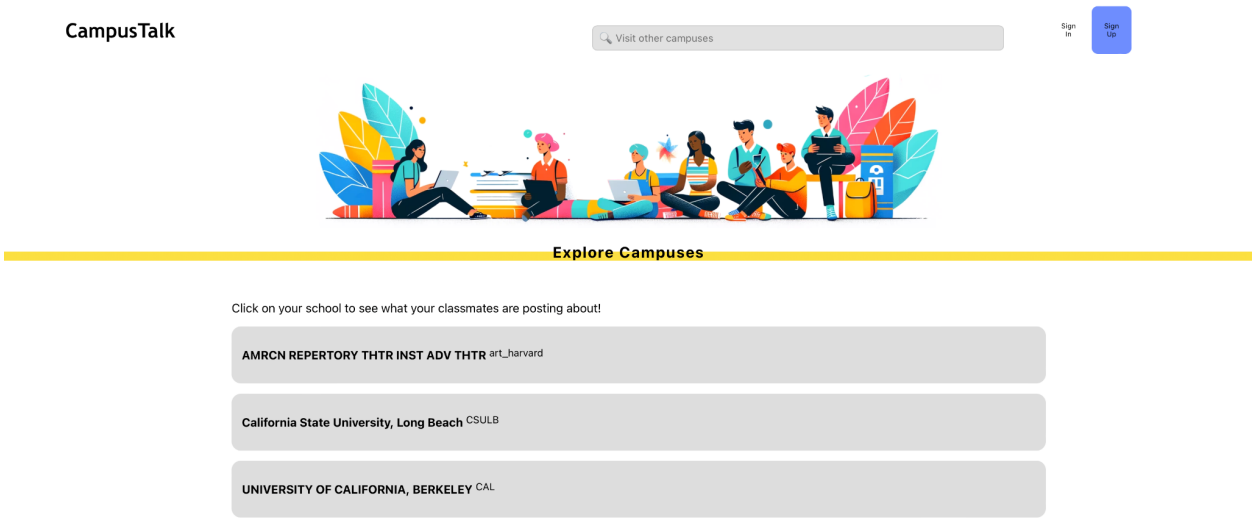
## 7.1 User Interface Design Overview

**Landing Page**



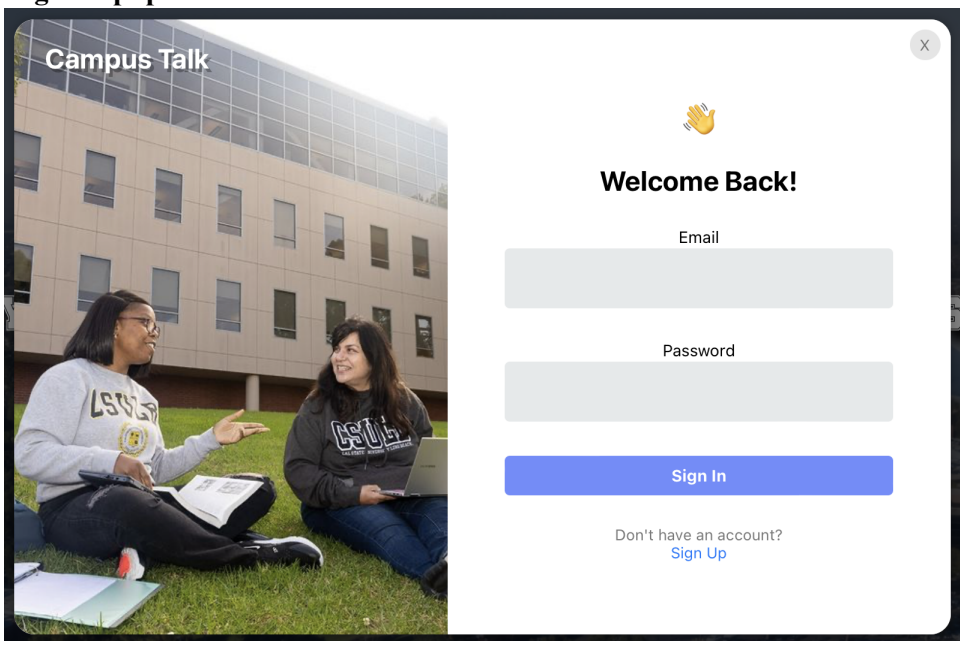

Includes app name, slogan, option to view colleges before signing up, sign in and sign up buttons

### Campuses List Page

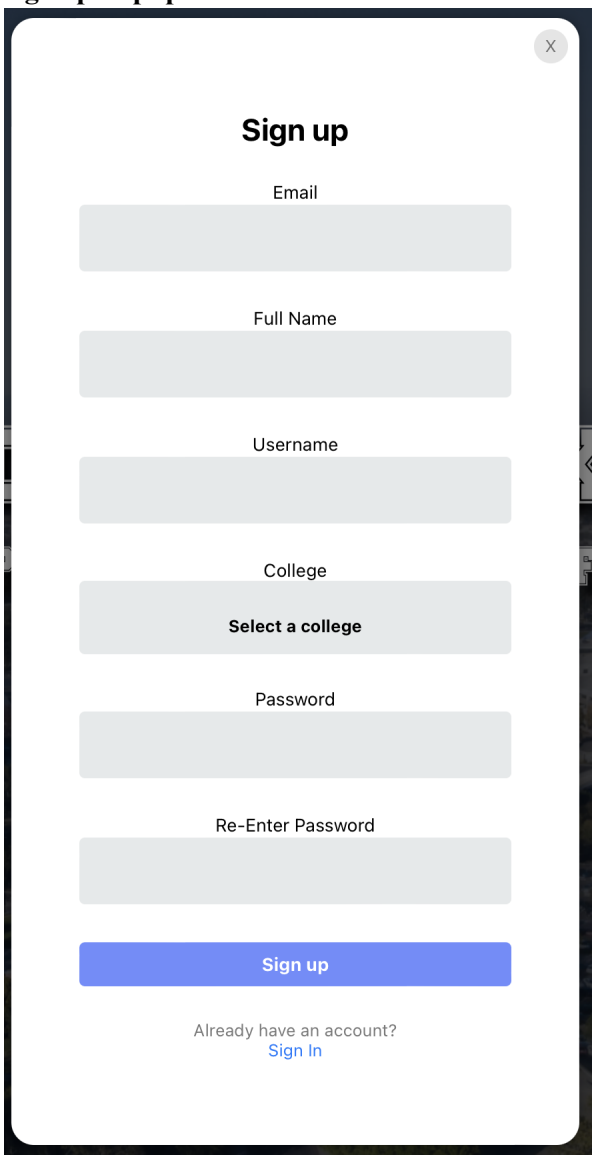



Lists the current campuses on Campus Talk, with a link to view their own feed

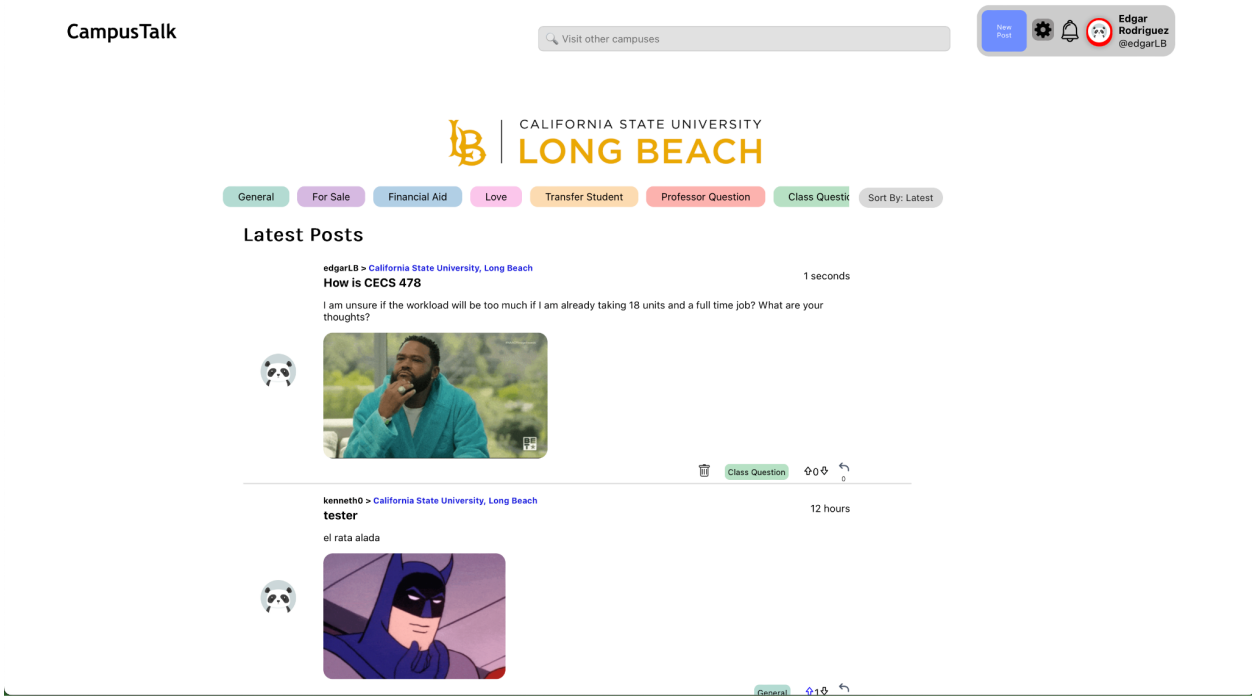
### Login Popup



**SignUp Popup**

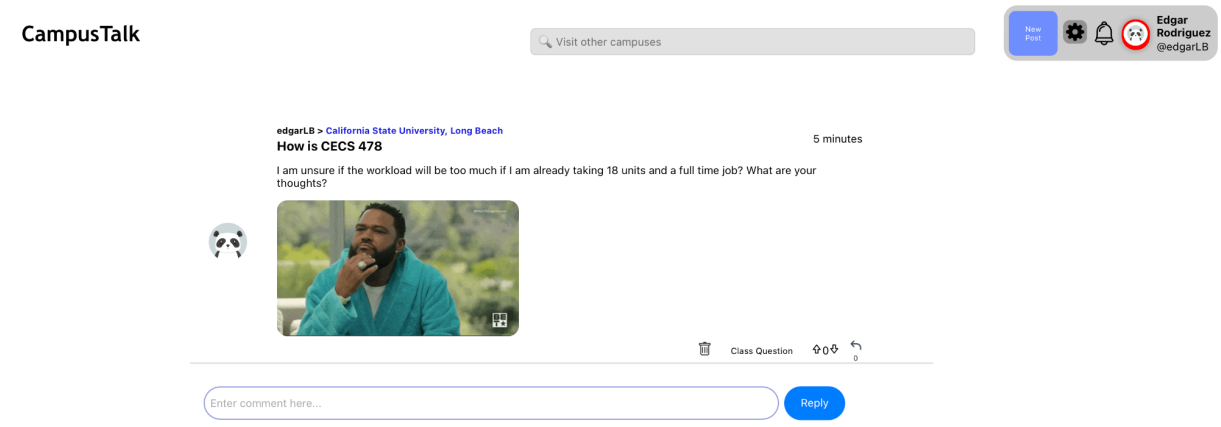


## Campus Feed View



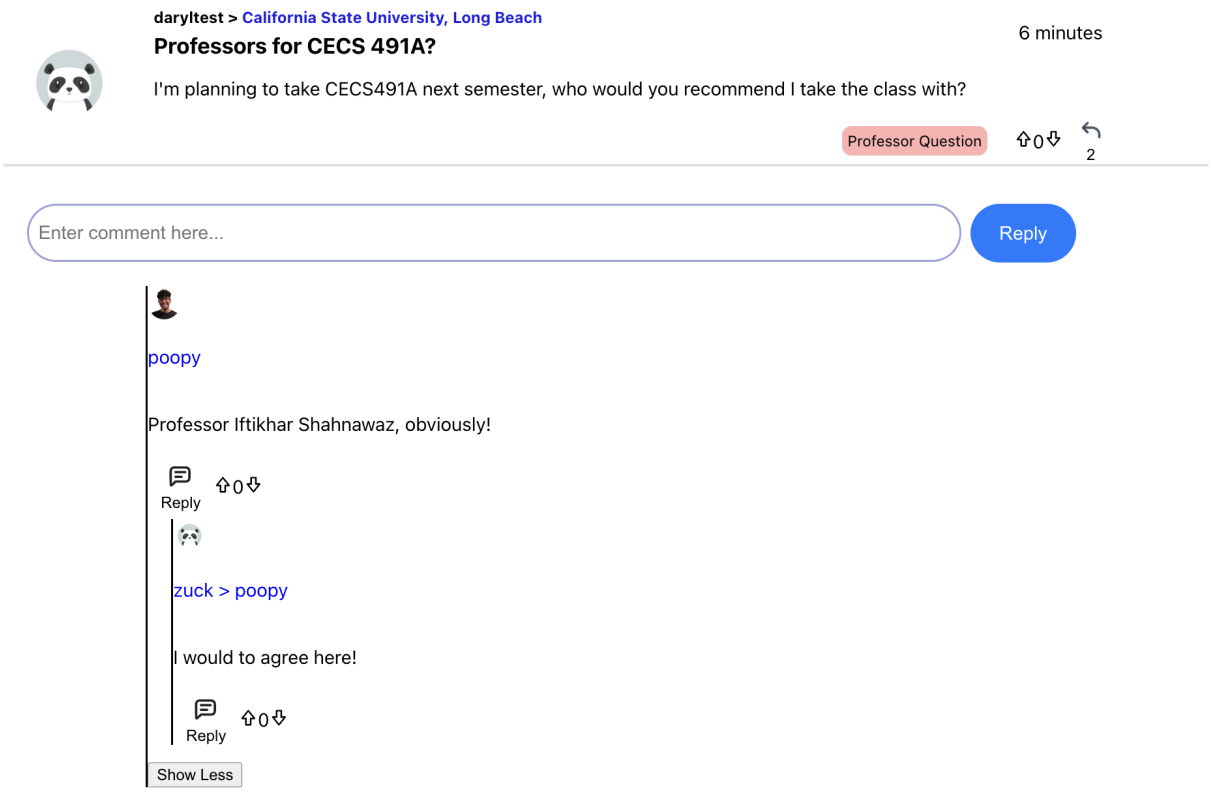

Users can sort the feed utilizing the sort dropdown and filter what type of post is displayed by selecting a tag. Additionally, users can interact with the posts by up or down voting it or clicking on the post. This will redirect users to the single post page. If the post belongs to the user, they will see the option to delete the post.

## Single Post View



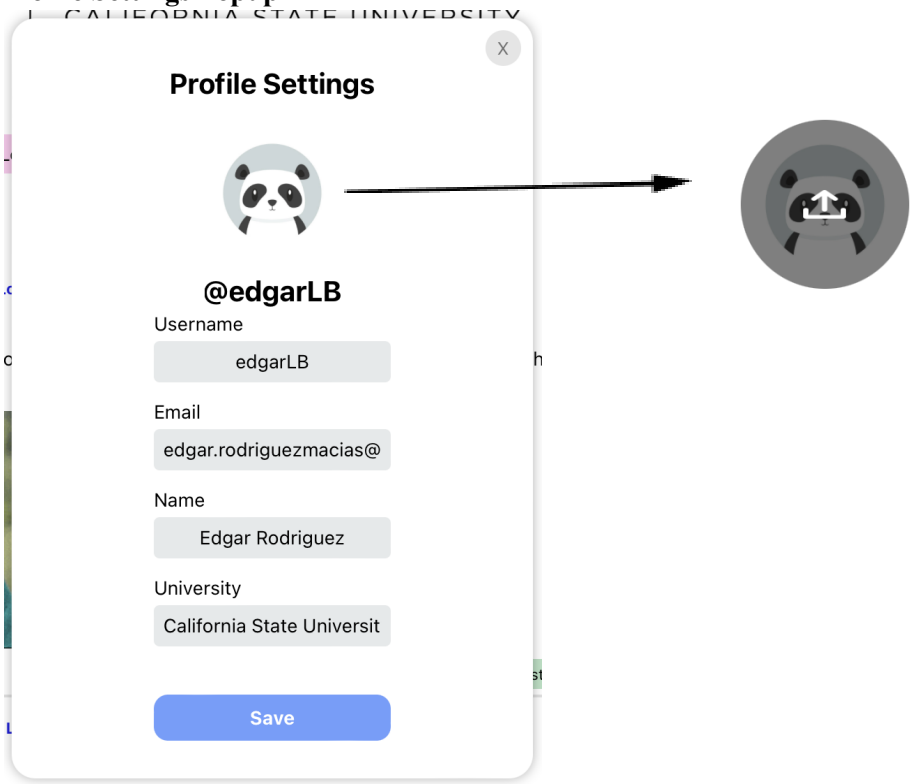

Here, the user will have the same option to interact with posts as in the feed view but will also be able to view and comment on the post.

## Comments



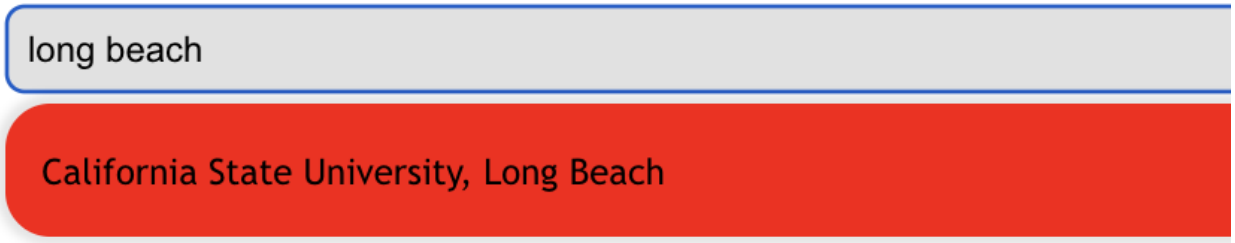

The comments on posts act as a thread where users can engage with others by commenting on comments and also up or downvoting them.

**Profile Settings Popup**

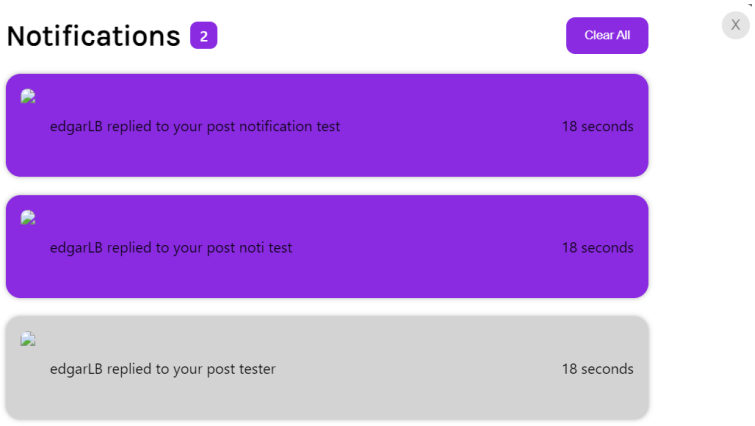


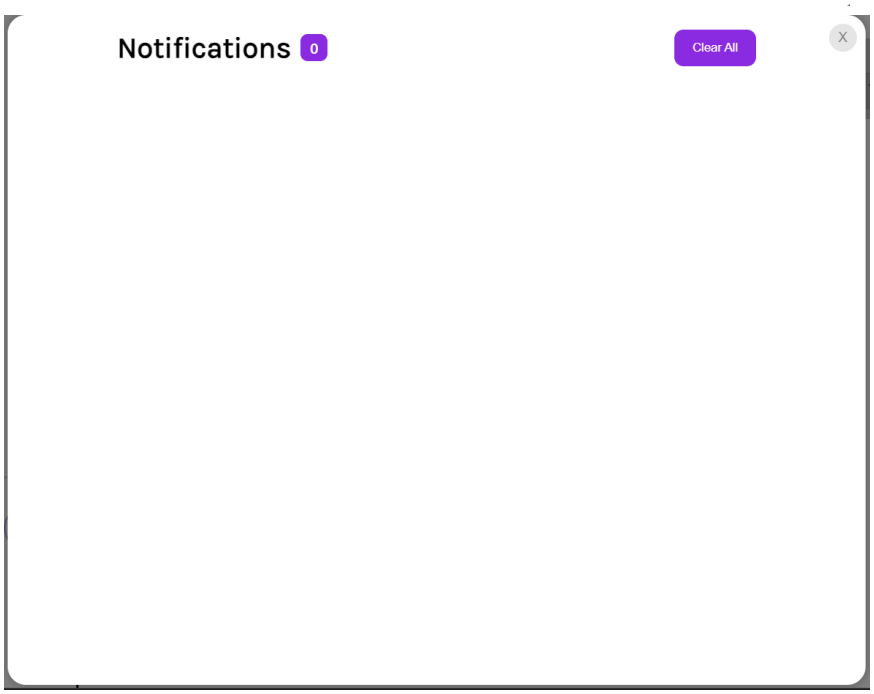

Users can update their profile settings and change their profile picture by clicking on the current picture

**Search Bar**

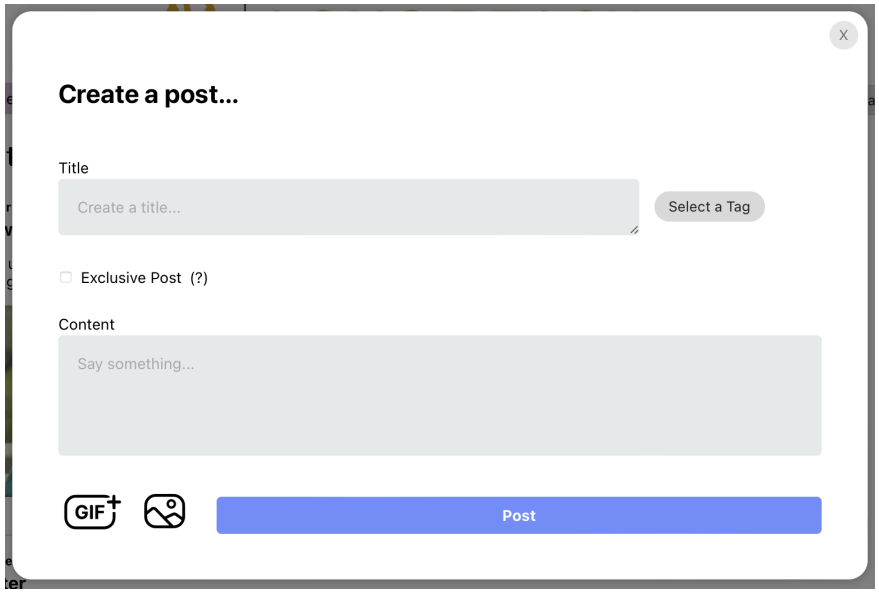
**Notifications**



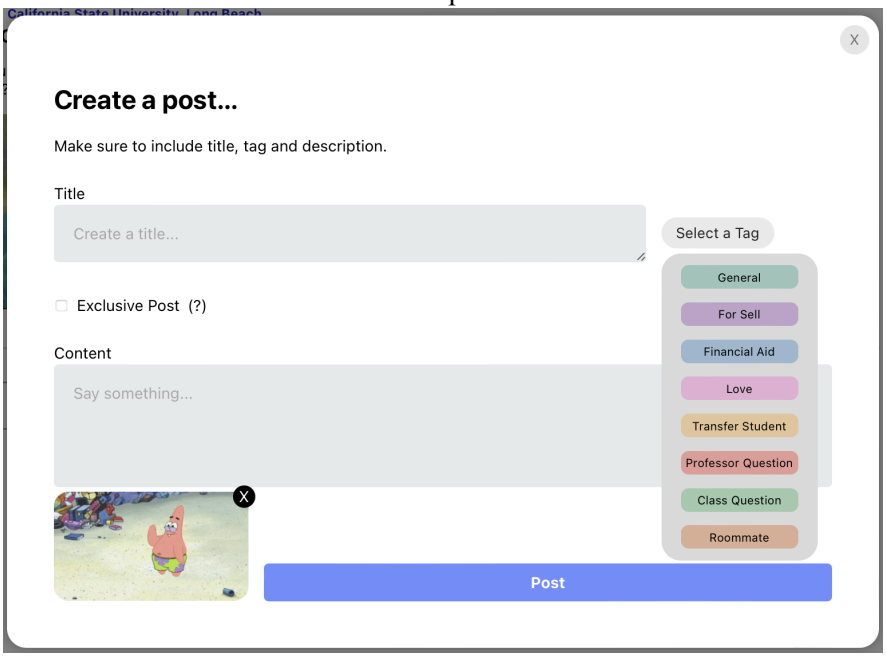



**Create Post Form**

Initial view

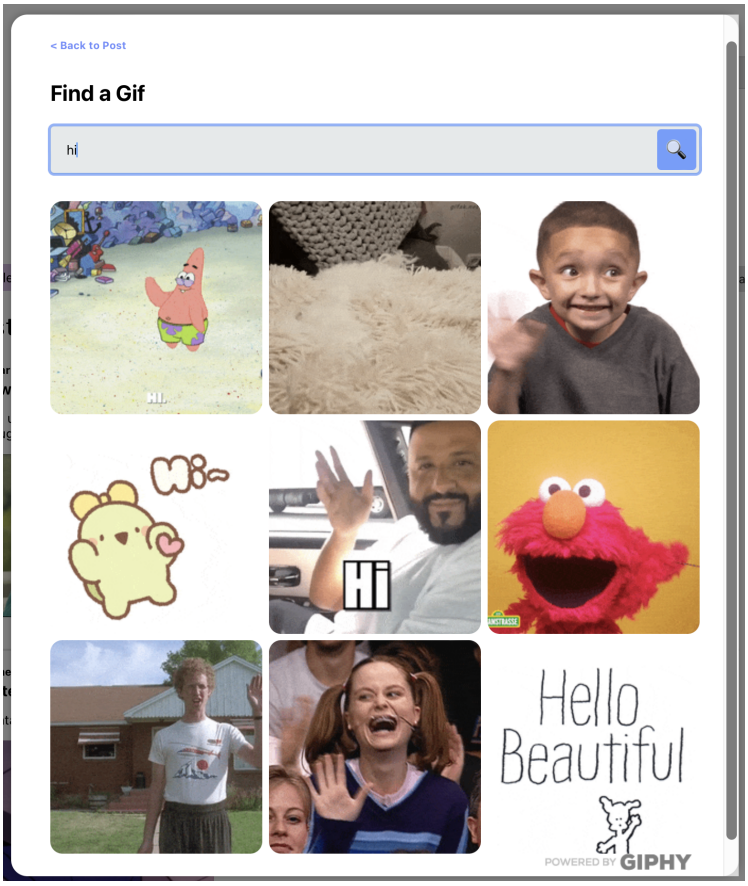



Additional options view




When users create a post, they must include a title, tag, and content; if the user is missing a required field, a reminder will appear, and the user cannot publish a post until all fields are filled. When choosing a tag, the user will be displayed a drop-down with the same color-coded tags from the feed and posts view. They can optionally include media such as a GIF or upload an image. If the user includes media, they will be shown a preview with the option to remove or change media.

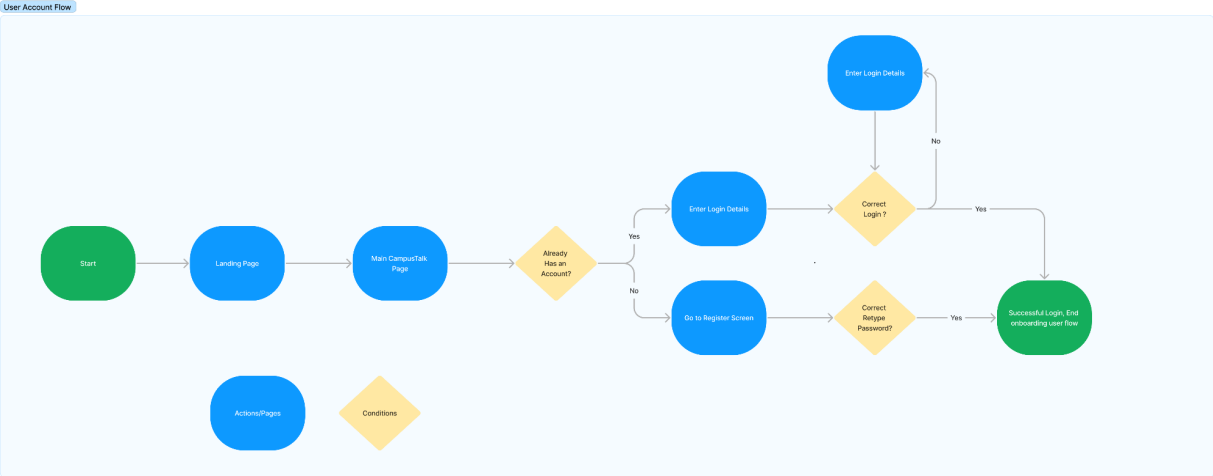
**Gif Search**




The post view will be transformed into a gif search to add a gif. The search is powered by Giphy and displays the user options for a gif to attach. When clicked, the user will be taken back to their post form.

## 7.2 User Interface Navigation Flow

**User Login/Register Flow**




## 7.3 Use Cases / User Function Description

| | |
|---|---|
| **Use Case Name** | User Register/Login |
| **Trigger** | User Selects Register/Login Button on landing page |
| **Actor** | Student(User) |
| **Overview** | The student will sign up by entering the school email, username, and password and will authenticate themself in order to receive access to the web application. |
| **Preconditions** | 1. User has launched the web application<br>2. User not registered or logged in |
| **Flow of Events** | **Events** |
| | 1. Web Application starts up with landing page with two buttons, Registration and Login for the User to select from.<br><br>2. User Selects Register Button<br>A. User inputs username, school email, name, password.<br>B. User submits information by pressing the register button.<br>C. System checks password strength and legitimate school email/username.<br>D. System updates database with account creation.<br>E. System creates a new account for the new user.<br>F. User automatically logs in after successful account creation. |

| | |
|---|---|
| | 3. User Selects Login Button<br>A. User inputs username, and password.<br>B. User presses login button<br>C. System validates User's attempt to Login with authentication<br>D.User Logs into the web application and is granted access to the website. |
| **Termination Outcomes** | **Conditions** |
| | 1. For User Registration<br>A.User's account is successfully created and is redirected into the main webpage.<br><br>2. For User Login<br>A. User's account is validated and User is redirected to the main webpage. |

| | |
|---|---|
| **Use Case Name** | User Creates Post |
| **Trigger** | User Selects Create a Post button |
| **Actor** | Student(User) |
| **Overview** | The student will create a post with a description. The user is sharing their thoughts or opinion about any topic they select on the post. This feature would implement the basis of the web application which facilitates discussion. |
| **Preconditions** | 1. User has launched the web application<br>2. User is logged into CampusTalk<br>3. User is on their own campus domain |
| **Flow of Events** | **Events** |
| | 1.Web Application displays Creates a Post button for user to select<br>2. User presses the Create a Post Button<br>3. User Selects tag for the post to be filterable by topic.<br>4. User inputs post information and submits the Post<br><br>5.System creates a new post in the database with its attributes and creates a unique identifier for the post. |
| **Termination Outcomes** | **Conditions** |
| | 1. The new post is created and displayed onto the web domain alongside with older posts.<br>2. Posts are now selectable to the user to interact with or view. |

| | |
|---|---|
| **Use Case Name** | User Comments |
| **Trigger** | User selects the notifications button on the CampusTalk panel. |
| **Actor** | Student(User) |
| **Overview** | The student interaction would receive live notification on their personal posting. This will enhance the communication aspect of the application based on live interactions with other users. |
| **Preconditions** | 1. User has launched the web application<br>2. User is logged into the web application<br>3. User has created a post.<br>4. Another user has interacted with Student's post. |
| **Flow of Events** | **Events** |
| | 1.User selects another user's post.<br>2.User Interface displays all comments and original post with a textbox for user to comment.<br>3.User enters comment information and submits Post Comment button<br>4.System creates a new comment in the post's database location with its descriptions and creates a unique identifier for that comment.<br><br>5.System updates user interface and displays newly created comments and other comments on the post. |
| **Termination Outcomes** | **Conditions** |
| | 1. The new comment is created and displayed onto the web domain alongside with the original Post.<br>2. Comments are now selectable to the user to interact with or view. |

| | |
|---|---|
| **Use Case Name** | User Creates GIF/Image |
| **Trigger** | User Selects Create “Upload a GIF” button |
| **Actor** | Student(User) |
| **Overview** | The student will create a post through GIF selection. The user is sharing their thoughts or opinion about any topic they select on the post. This feature would implement the basis of the web application which facilitates discussion. |
| **Preconditions** | 1. User has launched the web application<br>2. User is logged into CampusTalk<br>3. User is on their own campus domain |

| **Flow of Events** | **Events** |
|---|---|
| | 1.Web Application displays Creates a Post button for user to select<br>2.User presses the Create a Post Button<br>3.User presses on GIF/Image button.<br>4.User searches for GIF/image through searchbar.<br>5.User selects GIF/Image<br>6.User types caption and title for post and selects post button. |

| **Termination Outcomes** | **Conditions** |
|---|---|
| | 1. The new post is created and displayed onto the web domain alongside with older posts.<br>2. User is able to interact with the post that contains the image or GIF. |

| | |
|---|---|
| **Use Case Name** | User Interacts with Notifications |
| **Trigger** | User Selects Notification Button |
| **Actor** | Student(User) |
| **Overview** | The student is able to interact with their post notifications. The user is able to mark their notifications as read and manage their notifications. This feature would implement live interaction with other people. |
| **Preconditions** | 1. User has launched the web application<br>2. User is logged into CampusTalk<br>3. User is on their own campus domain<br>4. User has created a post<br>5. Another user has interacted with original post, |
| **Flow of Events** | **Events** |
| | 1.Web Application displays Creates a Post button for user to select<br>2. User creates a post<br>3. Another User comments on User's post<br>4.User presses on Notification button.<br>5. The user is allowed to mark notifications read, and clear all notifications |
| **Termination Outcomes** | **Conditions** |
| | 1. The user is able to view notifications and is notified whenever another user is interacting with their post.<br>2. User is able to interact with notifications received from another user. |

## Section 8 - Testing

Our team utilized branch testing.

**Branch Name:** someUI-edgar
**Feature/Fix Description:** Make sure UI components work as intended for the user
**Test Scenarios:**

1. Sign Up as a new user
2. Create posts
3. Personalize account
4. Navigate site

**Issues Found and Resolved**

1. When a tag was clicked on a post, the whole site would crash
2. New features made post clunky to make, needed new UI
3. Comic Sans was used for a title that does not fit the branding
4. Sort By button would appear above any popup
5. Pop-ups that needed to be focused were not, causing them to blend into the background.
6. The image preview was missing when uploading a picture
7. Posts ask for a title but never used it

**Final Status:** All issues resolved. Users can expect the interface to work as intended

**Branch Name:** gif
**Feature/Fix Description:** Allow user to search for GIFs and add them to posts
**Test Scenarios:**

1. User decides to add gif while writing a post
    a. Click on GIF Button under text field
    b. Search for GIF based on search term
    c. Grid of 9 GIFs appears under the search field
    d. User chooses a GIF which attaches to the post form
    e. User may upload this GIF attached to their post
2. GIFs displayed on posts in feed and on Single Post Pages

**Issues Found and Resolved**

1. The GIFs were uploaded but are not displaying when viewing post
2. When switching to and from views, user's post form input was lost
3. User can accidentally choose the wrong GIF

**Final Status:** Selecting, Searching, and Viewing gifs all work as intended. User input data is also persistent when switching views.

**Branch Name:** Comment-Sections
**Feature/Fix Description:** Allow users to comment on comments
**Test Scenarios:**

1. Reply to a post
2. Reply to a comment in post

3. Up and downvote comments

**Issues Found and Resolved**

1. Downvoting after upvoting did not display the correct number of votes.
2. No way to differentiate a sub-comment from a comment

**Final Status:** Reactions to votes, including upvote/downvote functional as well as replying to comments.

**Branch Name:** authentification
**Feature/Fix Description:** Develop authentication for users to register and login
**Test Scenarios:**

1. User can log in
2. User can register
3. User has a session
4. User can perform actions with session
5. User can log out

**Issues Found and Resolved**

1. Issues creating and persisting session
2. Session not being stored

**Final Status:** authentication and session management works as intended.

**Branch Name:** SinglePostPages
**Feature/Fix Description:** Have a page where full post can be view and where user can interact with post
**Test Scenarios:**

1. Click on post and see if it redirects correctly
2. Interact with post

**Issues Found and Resolved**

1. Images uploaded had no max size, so they took up the whole screen

**Final Status:** Post pages are displayed and work as intended

**Branch Name:** clean_up_route_directories
**Feature/Fix Description:** improved file and folder management
**Test Scenarios:**
**Issues Found and Resolved:**

1. Routes in some files were failing and required updates

**Final Status:** file/folder management was improved and routes were more efficiently.

**Branch Name:** mine
**Feature/Fix Description:**
**Test Scenarios:**
**Issues Found and Resolved**
**Final Status:**

**Branch Name:** generic-exclusive-posts
**Feature/Fix Description:** Allow users to share posts across communities outside of their own school
**Test Scenarios:**
1. Create a generic post
2. See if the post is in a separate general feed

**Issues Found and Resolved**
1. All schools share the same post. Making every school a generic feed

**Final Status:** Posts users create are shared in the correct feed they posted it too.

**Branch Name:** landingpage
**Feature/Fix Description:** Make it easy for users to explore website for the first time
**Test Scenarios:**
1. Visit campus talk for the first time
2. Click anything and everything a user might click

**Issues Found and Resolved**
1. Could not tell there were 3 buttons. Looks like two buttons and some text. Need to highlight 3rd click area
2. Floating posts distract from what users can do from the landing page
3. The text blends into the background

**Final Status:** The landing page is now simplified to allow users to get overwhelmed/ lost when visiting for the first time

**Branch Name:** notif
**Feature/Fix Description:** Implement notifications based on User commenting system
**Test Scenarios:**
1. Counter for notifications
2. Clearing all notifications
3. Mark-as-read based on clicking the notifications

**Issues Found and Resolved:**
1. Counter for all notifications would be stuck at 1.
2. Notifications would not clear based on pop-up

**Final Status:** Notifications appear on the original poster when another user comments on it.

**Branch Name:** proper
**Feature/Fix Description:** Fixed component layout and component folders
**Test Scenarios:**
1. Header contains proper components
2. Profile, Notification, and Panel fully functioning
3. Replies are properly handled

**Issues Found and Resolved**
1. Replies not collapsing in show more/less

2. Names in components were improperly labeled.

**Final Status:** The panel, including accessible components, is properly functional after reorganizing headers.

# Section 9 – Deployment

## Docker Compose Configuration

1. ***Defining Services***
    a. nginx: Acts as a reverse proxy, directing incoming requests to appropriate containers based on the requested port.
    b. Backend: Node.js application server handling backend logic
    c. Frontend: React.js application serving User Interface

Deployment is accomplished using Docker to create containers for our services and is hosted on a Google Cloud VM. The docker-compose file is run to establish the containers and the respective firewall rules were added to the Debian VM and Google Cloud virtual cloud network.

```
version: '3'

services:
 nginx:
   image: nginx:latest
   ports:
     - "80:80"
     - "8800:8800"
   volumes:
     - ./nginx/nginx.conf:/etc/nginx/nginx.conf
   depends_on:
     - frontend
     - backend
 backend:
   build:
     context: ./node
```

```
    dockerfile: ./Dockerfile
  expose:
    - "8000"
frontend:
  build:
    context: ./react
    dockerfile: ./Dockerfile
  expose:
    - "3000"
```

This is the docker-compose yaml file that allows all the containers to communicate with each other in the same network. It also allows for an overlook of the ports and number of containers. The first container is an nginx server which will act as a forward proxy, listening for port 80 and 8800 traffic that will be forwarded to the other containers.

## Nginx configuration

1. **Reverse Proxy Setup:**
    a. Configured to listen on ports 80 (HTTP) and 8800 (custom application port).
    b. Port 80 is set up for the front end, Port 8800 is set up for the backend.
2. **Header Management:**
    a. The proxy uses headers including Host, X-Real-IP, X-Forwarded-For, and X-Forwarded-Proto

```
worker_processes 1;

events {
   worker_connections 1024;
}

http {
   server {
       listen 80;
       server_name localhost;

       location / {
           proxy_pass http://frontend:3000;
```

```
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }

    server {
        listen 8800;
        server_name localhost;

        location / {
            proxy_pass http://backend:8000;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
}
```

This is the nginx configuration that tells the server to forward port 80 traffic to the frontend container on port 3000. Additionally, it will forward traffic from port 8800 to the backend container on port 8000.

## Backend Dockerfile

1. **GitHub Access**
    a. Uses CampusTalk's GitHub token to connect from a private repository.
2. **Port Exposure**
    a. Exposes port 8800 for backend service, which is used for nginx proxy
3. **Startup commands**
    a. Uses run npm install, and npm start

```
# Use an official Node runtime as the base image
FROM node:14

# Set the working directory in the container
WORKDIR /app

# Set your GitHub access token as an ARG (pass it during the build)
ARG GITHUB_TOKEN
ENV GITHUB_TOKEN=<REDACTED_GITHUB_TOKEN>

# Download your private GitHub repository (replace with your repository URL)
RUN git clone https://$GITHUB_TOKEN@github.com/raav3n/campustalk && mv
campustalk/campustalk/s>

# Keep only the server files
RUN rm -rf campustalk

# install modules
WORKDIR ./server
RUN npm install

# add new .env file
RUN rm .env
COPY .env /app/server

# Expose the port the app will run on
EXPOSE 8000

# Define the command to start the app
CMD ["npm", "start"]
```

Above is the Dockerfile for the backend server that is hosting node.js with express. It will clone our GitHub repository and only keep the /server files. It will also copy over a .env file stored locally and used to update the host IP and port.

**Frontend Dockerfile**

1. **Build Process**
    a. Removes server-related files focusing on the front-end part.
    b. Sets up serve to host static files
2. **Port exposure**
    a. Port 3000 is exposed, aligning with the nginx configuration for forward trafficking to the front-end.
3. **Startup commands**
    a. Uses RUN npm install, RUN npm run build --production, RUN npm install -g serve

```
# Use an official Node runtime as the base image
FROM node:14

# Set the working directory in the container
WORKDIR /app

# Set your GitHub access token as an ARG (pass it during the build)
ARG GITHUB_TOKEN
ENV GITHUB_TOKEN=<REDACTED_GITHUB_TOKEN>

# Download your private GitHub repository (replace with your repository URL)
RUN git clone https://$GITHUB_TOKEN@github.com/raav3n/campustalk

WORKDIR /app/campustalk/campustalk/

# remove server files and add new .env
RUN rm -rf server/
RUN rm .env
COPY .env /app/campustalk/campustalk

WORKDIR /app/campustalk/campustalk/src

# install and build
RUN npm install
RUN npm run build --production
RUN npm install -g serve

WORKDIR /app/campustalk/campustalk



# Expose the port the app will run on
```

```
EXPOSE 3000


# Define the command to start the app
CMD ["serve", "-s" ,"build"]
```

This is the Dockerfile for the react application. It pulls the code from the GitHub and removes the /server code files. It builds the application and serves it on port 3000.

## Section 10 – References

**Prerequisites:**
Before you can run the app, you need to have the following installed on your system:

- Node.js (version 12 or higher)
- React.js (version 18 or higher)

You also need to set up your development environment for React JS according to the instructions at https://react.dev/learn/installation.

**Installation:**
To install the app, follow these steps:

1. Clone this repository to your local machine.
2. Open a terminal window and navigate to the project directory.
3. Run the following command to install the app's dependencies:

```
npm install
```

4. Run the following command to start the app on your local machine:

```
npm start
```