CS 387 Project: NetWorks

Final Report

Pandurang Deore (200050096)
Sarthak Mittal (200050129)
Shikhar Agrawal (200070076)
Mayank Jain (20d070050)

Contents

1 Summary

We have built a functioning clone of 'LinkedIn' as an online job hub cum social media web application. We have used the PERN stack (PostgreSQL, Express, React, Node.js) to implement the same and have incorporated useful features such as posts, likes, comments, jobs and connections. We keep track of two types of users, recruiters and applicants, and provide different functionalities for each. The app intends to act as a virtual medium to connect job aspirants and prospective employers.

2 Overview

2.1 Domain of our Web App

We have developed the app to serve as a part of online professional networks and to be used as a third-party employment portal. Keeping LinkedIn as the projected aim, we have incorporated all of its prominent features while maintaining simplicity and enabling easy navigation of our interface.

2.2 Intended Use

Our application currently supports features that we expect to be used for:

- building an online portfolio, to be used as a reference by other professionals
- keeping up to date with recent job openings and professional opportunities
- searching for suitable employment online and to look for skills that are in demand
- enabling better communication with people in the professional network
- connecting with other applicants with similar interests
- recruiting people for one's organisation and advertising job openings

3 Users

Our application is built keeping two different kinds of users in mind, and this can be understood as two different 'roles' for our application:

3.1 Role of an Applicant

As an applicant, the application allows you to:

- to look for different jobs, check their own eligibility and apply accordingly.

- find other applicants with similar interests and looking for similar job opportunities
- build a social network and chat with people in it
- generate posts with images, captions and hashtags
- like and comment on other's posts

3.2 Role of a Recruiter

As a recruiter, the application provides a bigger set of capabilities. Specifically, a recruiter can also:

- generate job openings and release them into the network
- look for eligible applicants and contact them through the chat interface

4 Database: the back-end

4.1 DDL

Given below is the DDL we have used for our entire backend:

```
CREATE DATABASE networks;

CREATE TABLE users(
    user_id INT NOT NULL,
    username VARCHAR(64) NOT NULL,
    email VARCHAR(64) NOT NULL,
    encrypted_password VARCHAR(80) NOT NULL,
    photo VARCHAR(128),
    place VARCHAR(64) NOT NULL,
    description VARCHAR(256) NOT NULL,
    currentwork_id INT,
    applicant_or_recruiter BOOLEAN NOT NULL,
    primary key (user_id)
);

CREATE TABLE application(
    job_id INT NOT NULL,
    applicant_id INT NOT NULL,
    primary key (job_id, applicant_id)
);

CREATE TABLE skill(
    user_id INT NOT NULL,
    skillname VARCHAR(64) NOT NULL,
    primary key (user_id, skillname)
);

```

```sql
CREATE TABLE connection(
    user1 INT NOT NULL,
    user2 INT NOT NULL,
    primary key (user1, user2)
);

CREATE TABLE conn_invite(
    user1 INT NOT NULL,
    user2 INT NOT NULL,
    primary key (user1, user2)
);

CREATE TABLE likes(
    liker INT NOT NULL,
    post_id INT NOT NULL,
    primary key (liker, post_id)
);

CREATE TABLE comment(
    comment_id INT NOT NULL,
    commenter_id INT NOT NULL,
    post_id INT NOT NULL,
    comment VARCHAR(64) NOT NULL,
    comment_time TIMESTAMP NOT NULL,
    primary key(comment_id)
);

CREATE TABLE post(
    post_id INT NOT NULL,
    user_id INT NOT NULL,
    photo VARCHAR(128),
    caption VARCHAR(64) NOT NULL,
    post_time TIMESTAMP NOT NULL,
    primary key (post_id)
);

CREATE TABLE hashtag_post(
    post_id INT NOT NULL,
    hashtag VARCHAR(32) NOT NULL,
    primary key (post_id, hashtag)
);

CREATE TABLE work(
    work_id INT NOT NULL,
    user_id INT NOT NULL,
    companyname VARCHAR(64) NOT NULL,
    start_time TIMESTAMP NOT NULL,
    end_time TIMESTAMP NOT NULL,
    position VARCHAR(64) NOT NULL,
    primary key (work_id)
);

```

```sql
CREATE TABLE message(
    user1 INT NOT NULL,
    user2 INT NOT NULL,
    sent_time TIMESTAMP NOT NULL,
    mesg_text VARCHAR(80) NOT NULL,
    primary key (user1, user2, mesg_text)
);

CREATE TABLE chat(
    user1 INT NOT NULL,
    user2 INT NOT NULL,
    primary key (user1, user2)
);

CREATE TABLE education(
    edu_id INT NOT NULL,
    user_id INT NOT NULL,
    institutename VARCHAR(64) NOT NULL,
    start_time TIMESTAMP NOT NULL,
    end_time TIMESTAMP NOT NULL,
    primary key (edu_id)
);

CREATE TABLE requirement(
    job_id INT NOT NULL,
    skillname VARCHAR(64) NOT NULL,
    primary key (job_id, skillname)
);

CREATE TABLE job(
    job_id INT NOT NULL,
    company VARCHAR(64) NOT NULL,
    place_of_posting VARCHAR(64) NOT NULL,
    time_launched TIMESTAMP NOT NULL,
    deadline TIMESTAMP NOT NULL,
    full_part BOOLEAN NOT NULL,
    skill_level VARCHAR(64) NOT NULL,
    company_desc VARCHAR(64) NOT NULL,
    job_desc VARCHAR(64) NOT NULL,
    launched_by INT NOT NULL,
    primary key (job_id)
);

CREATE TABLE numusers(
    num_us INT,
    num_edu INT,
    num_job INT,
    num_work INT,
    num_post INT,
    num_comment INT
);
```

```sql
INSERT into numusers values (0,0,0,0,0);

ALTER TABLE users
    ADD CONSTRAINT fk_currentwork_id FOREIGN KEY (currentwork_id) REFERENCES work(work_id)
    ON DELETE SET NULL ON UPDATE CASCADE;

ALTER TABLE application
    ADD CONSTRAINT fk_job_id FOREIGN KEY (job_id) REFERENCES job(job_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE application
    ADD CONSTRAINT fk_applicant_id FOREIGN KEY (applicant_id) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE skill
    ADD CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE conn_invite
    ADD CONSTRAINT fk_user1 FOREIGN KEY (user1) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE conn_invite
    ADD CONSTRAINT fk_user2 FOREIGN KEY (user2) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE connection
    ADD CONSTRAINT fk_user1 FOREIGN KEY (user1) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE connection
    ADD CONSTRAINT fk_user2 FOREIGN KEY (user2) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE comment
    ADD CONSTRAINT fk_commenter_id FOREIGN KEY (commenter_id) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE comment
    ADD CONSTRAINT fk_post_id FOREIGN KEY (post_id) REFERENCES post(post_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE likes
    ADD CONSTRAINT fk_liker_id FOREIGN KEY (liker) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

```

```sql
ALTER TABLE likes
    ADD CONSTRAINT fk_post_id FOREIGN KEY (post_id) REFERENCES post(post_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE post
    ADD CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE hashtag_post
    ADD CONSTRAINT fk_post_id FOREIGN KEY (post_id) REFERENCES post(post_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE chat
    ADD CONSTRAINT fk_user1 FOREIGN KEY (user1) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE chat
    ADD CONSTRAINT fk_user2 FOREIGN KEY (user2) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE work
    ADD CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE message
    ADD CONSTRAINT fk_users FOREIGN KEY(user1, user2) REFERENCES chat(user1, user2)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE education
    ADD CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE requirement
    ADD CONSTRAINT fk_job_id FOREIGN KEY (job_id) REFERENCES job(job_id)
    ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE job
    ADD CONSTRAINT fk_launched_by FOREIGN KEY (launched_by) REFERENCES users(user_id)
    ON DELETE CASCADE ON UPDATE CASCADE;
```

4.2 the API

4.2.1 /Q0 : authorization

The backend uses the cookie sent, if any, to deduce if a user is logged in for authorization purposes.

4.2.2 /Q1 : login

The backend expects the email and corresponding correct password in the request body. The response object signifies success or failure, and has a cookie in the headers on success that is used by express for session management.

4.2.3 /Q2 : sign up

The backend expects the request body to have the user_name, email, password and a boolean flag for whether the person is a recruiter or an applicant. The response object signifies success or failure.

4.2.4 /Q3 : user list

The backend respons with a JSON object having a list of corresponding username, email and userids.

4.2.5 /Q4 : invitation accept/cancel

The backend expects the request body to contain the sender whose request was responded to, and a flag to indicate whether the connection request was accepted or cancelled.

4.2.6 /Q5 : logout

The backend logs out the corresponding user and ends their session.

4.2.7 /Q6 : profile photo

The backend receives a file for the corresponding user's new profile photo.

4.2.8 /Q7 : upload post

The backend expects the request body to contain the caption, image and a list of hashtags.

4.2.9 /Q8 : username and hashed password

The backend returns the current username and corresponding hashed password.

4.2.10 /Q9 : get profile photo

The backend returns the path to the profile photo of the user. The frontend can use this as a hyperlink to display the image in HTML.

4.2.11 /Q10 : invites to user

The backend returns a list of all 'unresponded' connection invites to the logged in user as a list of corresponding username and user ids.

4.2.12 /Q11 : sent invite cancel

The backend expects the id of the user to whom the cancelled request was sent.

4.2.13 /Q12 : invites from user

The backed return all the 'unresponded' connection invites from the logged in user as a list of usernames.

4.2.14 /Q13 : all connections

The backend returns all the connections of the logged in user as a list of usernames.

4.2.15 /Q14 : all available jobs

The backend returns a list of all available jobs in the form of a list of job_ids, their corresponding companies, and other relevant stuff.

4.2.16 /Q15 : cancelling applied job

The backend expects a job_id in the request of a job for which the current user has applied, and removes his application in the backend.

4.2.17 /Q16 : all applied jobs

The backend returns a list of all jobs applied for by the current user in the form of a list of job_ids, their corresponding companies, and other relevant stuff.

4.2.18 /Q17 : job info

Given a job_id in the request, the backend retrieves all the information related to the job and returns it as a JSON object response.

4.2.19 /Q18 : update profile

The backend receives the new place and description along with a new profile photo and updates it in the database.

4.2.20 /Q19 : get posts

The backend expects an integer parameter `x` in the request and responds with the `x` latest posts' ids from the database.

4.2.21 /Q20 : get post info

The backend gets the post_id from the request and returns as a JSON object the contents of the post, and other relevant information such as the post owner.

4.2.22 /Q21 : get comments

The backend gets the post_id from the request and returns as a list of JSON objects all the comments on that post.

4.2.23 /Q22 : new post

The backend collects relevant data from the request JSON and adds the post to the database along with the time of posting.

4.2.24 /Q23 : send connection invite

The backend expects a user_id in the request JSON object and "sends" them a connection request by adding a relevant tuple in the database.

4.2.25 /Q24 : create job

The backend extracts information from the request JSON object and populates a tuple in the database to create a new job opening. This can only be done if the logged in user is a recruiter.

4.2.26 /Q25 : apply for job

Using the job_id from the request object, the backend add an application from the logged in user for the job into the database.

4.2.27 /Q26 : update job details

The backend expects the new company name, place and/or start time and updates them as the field for current work for the currently logged in user.

4.2.28 /Q27 : update education details

The backend unpacks from the request object a list of JSON objects each having an institute name, start time and end time and adds these to the education profile of the currently logged in user.

4.2.29 /Q28 : profile photo for username

The backend expects a username in the request and send the corresponding profile photo to the frontend.

4.2.30 /Q29 : comment on post

The backend expects the content of the comment and the post_id in the request and updates database appropriately.

4.2.31 /Q30 : liked/unliked a post

The backend expects a post_id in the request JSON object and adds or deletes the corresponding tuple for the logged in user into the table for post-likes.

4.2.32 /Q31 : remove a job

The backend expects the request to have a job_id, which it removes from the database.

4.2.33 /Q32 : all jobs from this user

The backend returns a list of JSON objects with corresponding job_ids, company-names and relevant details that have been created by the currently logged in user.

4.2.34 /Q33 : all job applicants

The backend expects a job_id inn the request object and returns a list of JSON objects with corresponding applicant usernames, user_ids and their paths to their resume.

4.2.35 /Q34 : recruiter/applicant

The backend responds with a single boolean relating to whether the currently logged in user is a recruiter or an applicant.

4.2.36 /Q35 : close a job

The backend closes the job specified by the job_id in the request.

4.2.37 /Q36 : profile of other user

The backend expects a username and returns a well built JSON object that can be readily used to display all the data of that user on his profile page by the frontend.

4.2.38 /Q37 : connected or not

The frontend uses this query to toggle between the connect/cancel buttons.

4.2.39 /Q38 : all hashtags

The backend responds with a list of all hashtags present in any post.

4.2.40 /Q39 : post_ids from hashtag

The backend expects a hashtag from the request and return all the post_ids whose posts have that hashtag.

4.2.41 /Q40 : check application

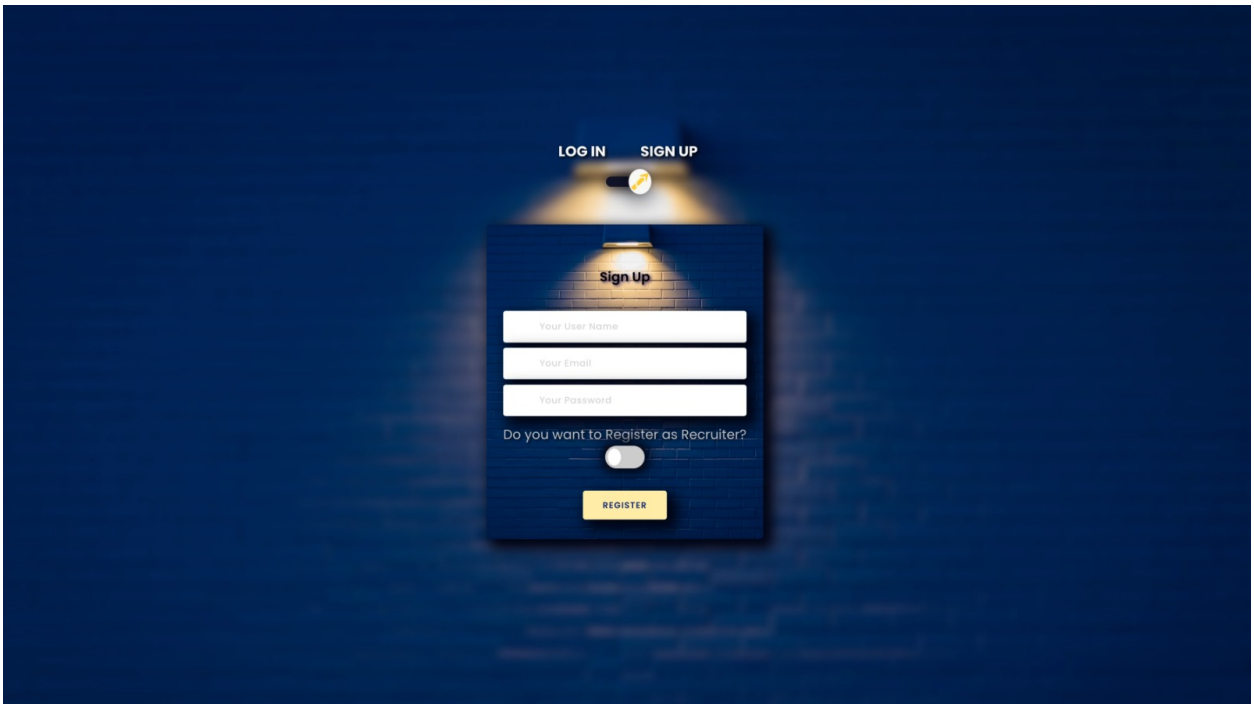
The backend expects a job_id and returns a boolean indicating whether the currently logged in user has already applied for that job or not.

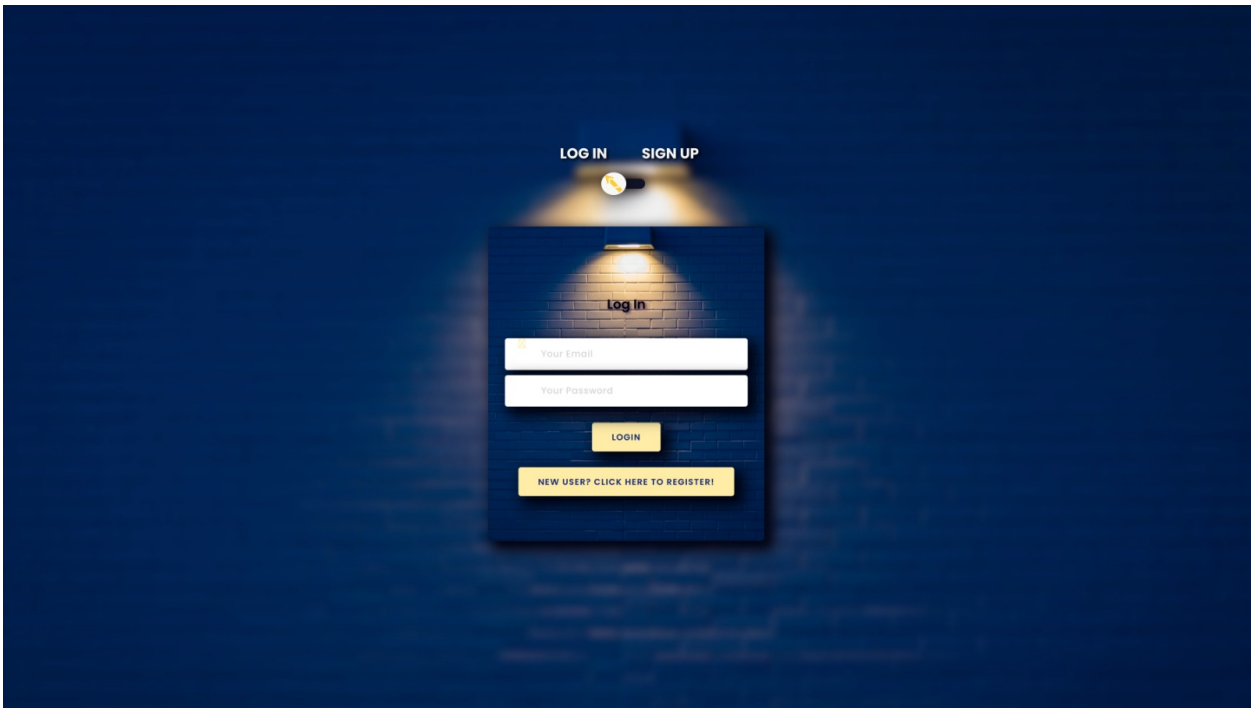
5 WebApp: the front-end

5.1 the API

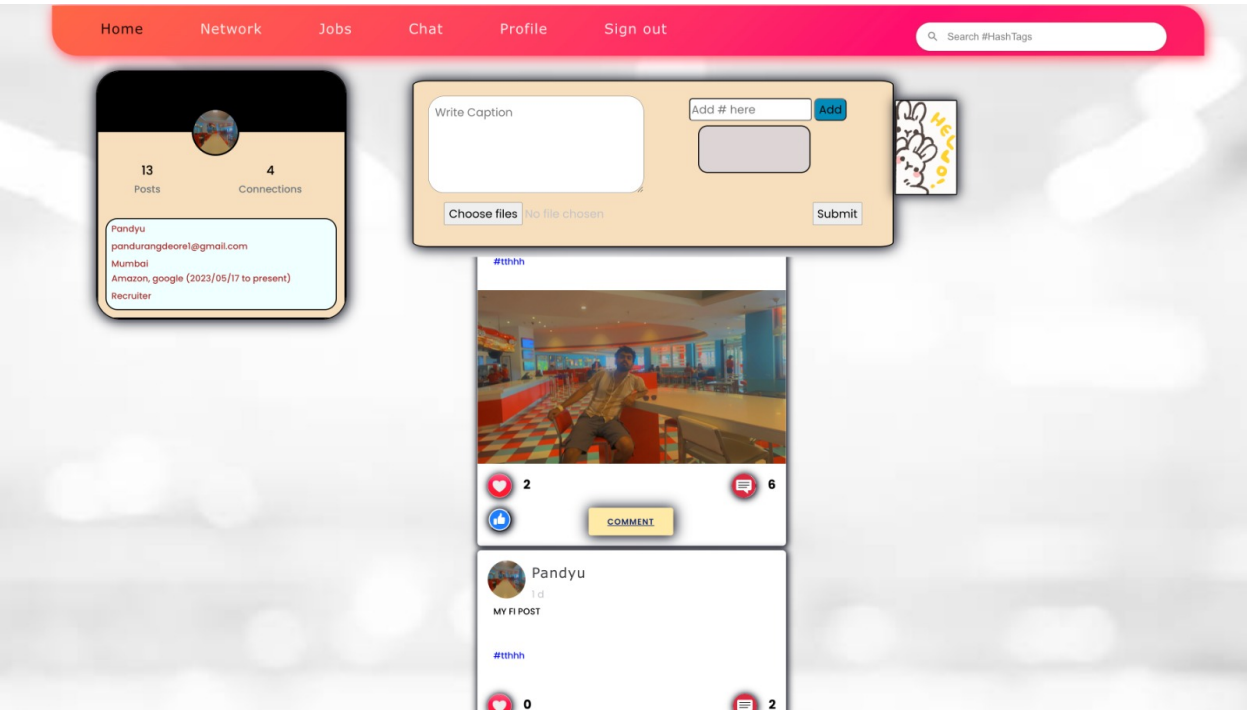
5.1.1 / : the landing page

5.1.2 /login : Login and Signup

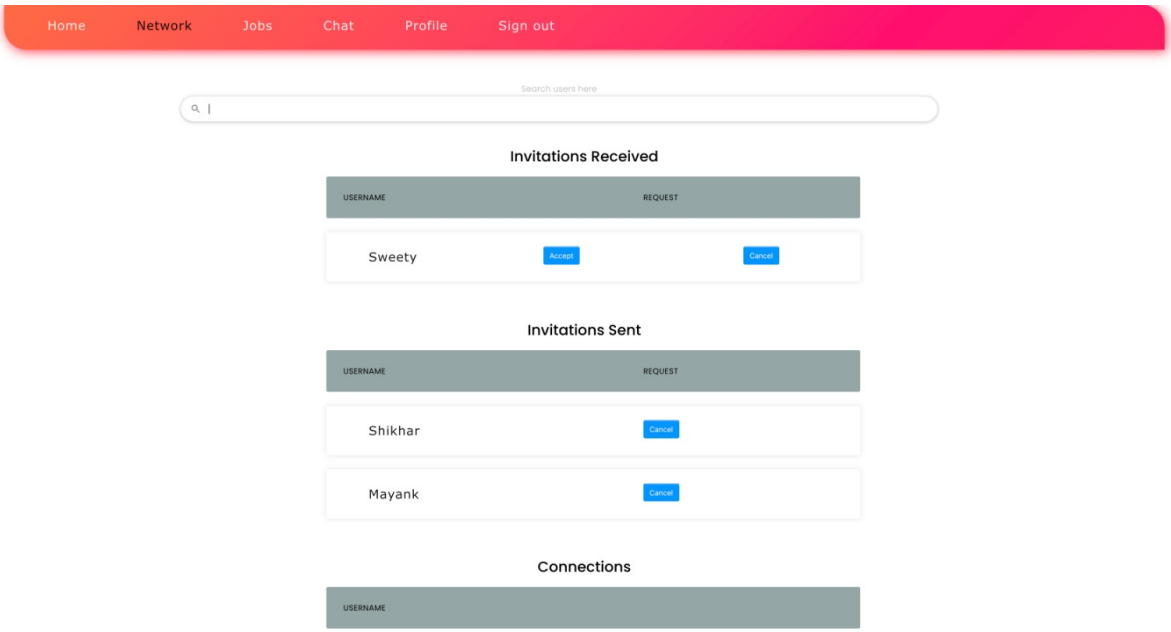




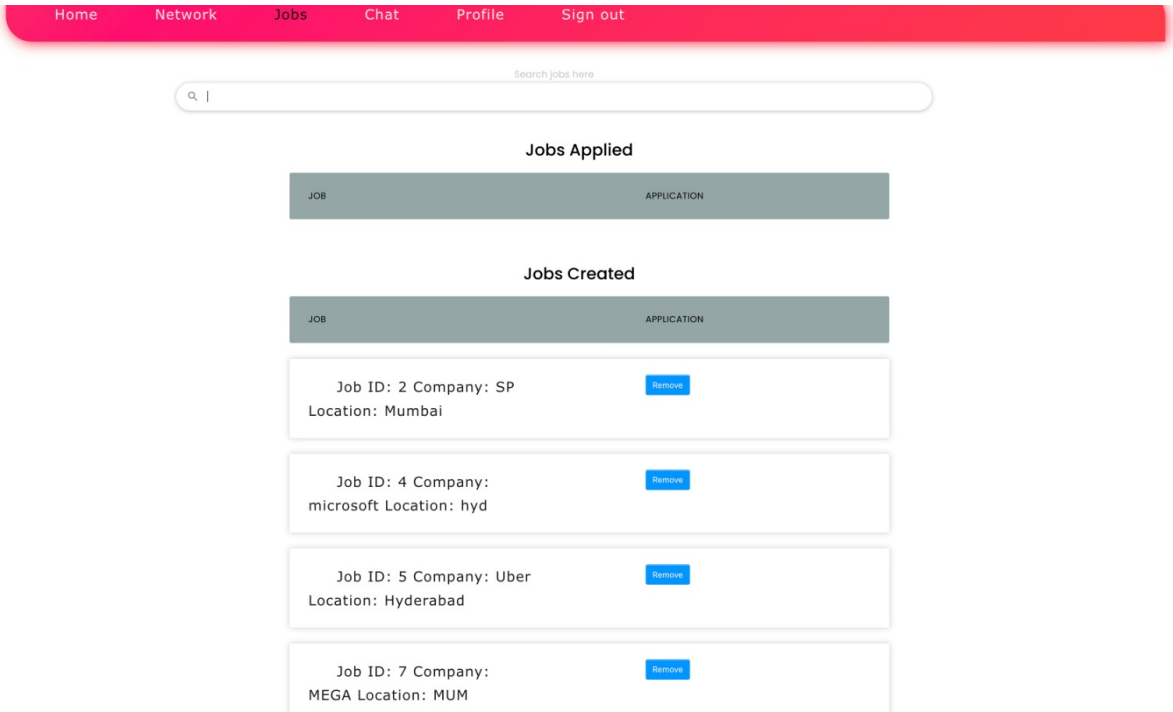
5.1.3 /home : Dashboard and Feed



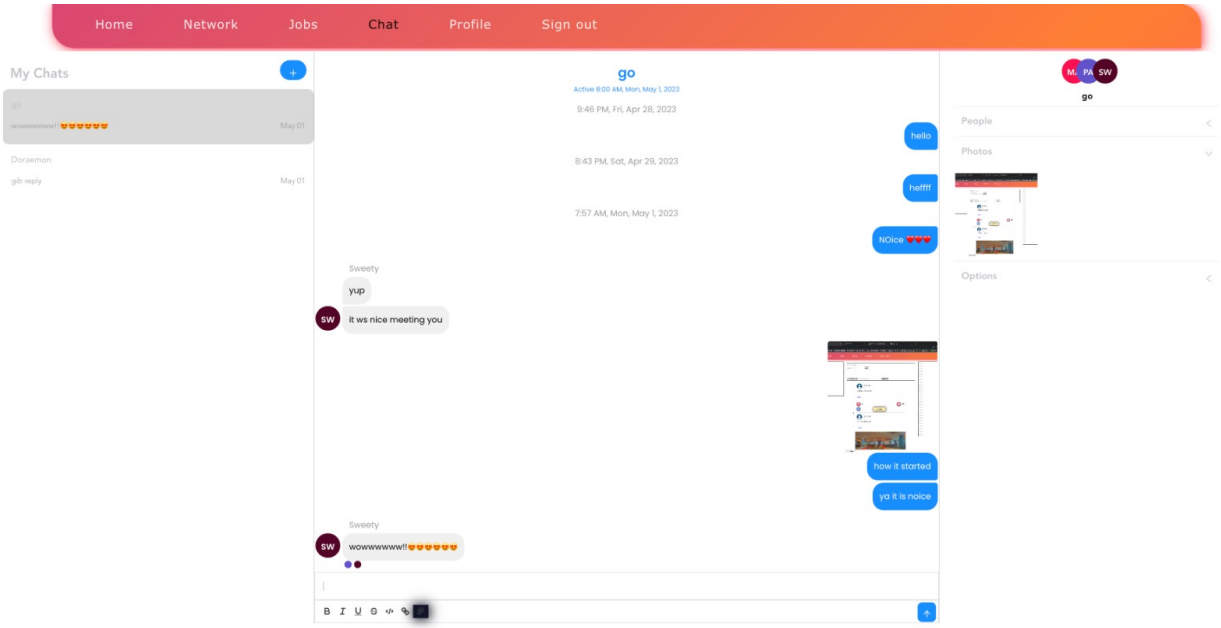
5.1.4 /network : Professional Network



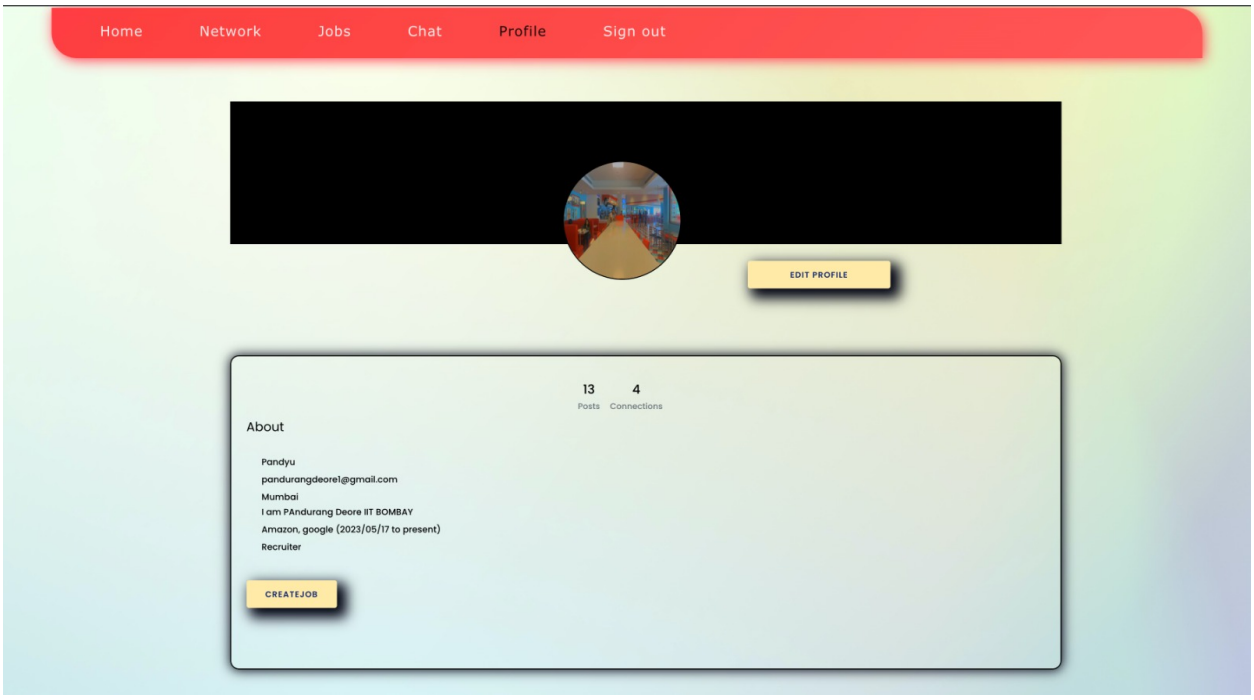
5.1.5 /jobs : View and Apply for Jobs



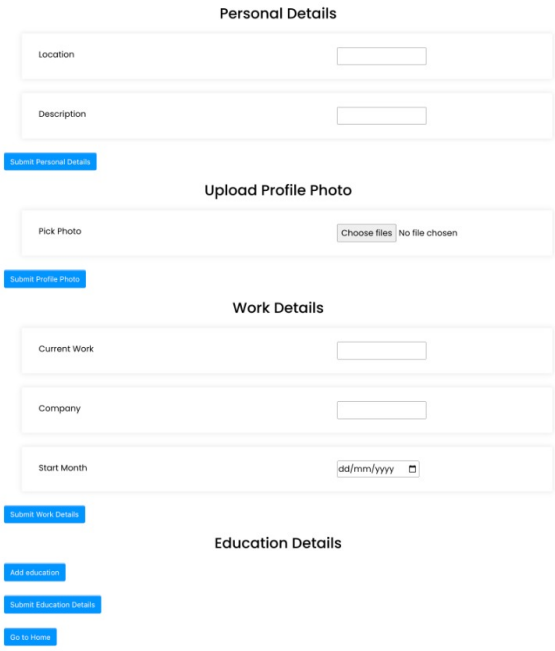
5.1.6 /chat : Chat Interface



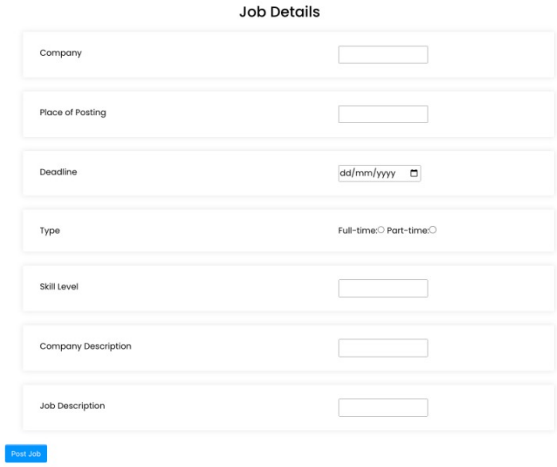
5.1.7 /profile : Viewing self and Other's Profiles



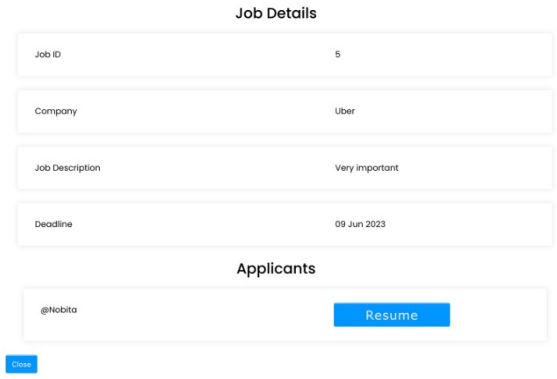
5.1.8 /fill-profile : Updating profile info



5.1.9 /new-job :Creating a new Job



5.1.10 /jobs/details/$: details about a particular job



5.2 security

We are using express sessions coordinated between backend and frontend for security. This enables scalable security without the use of JWT, and simplifies the code's nature itself.

6 Future Work

Our current version of the LinkedIn clone is well functioning and robust as is. If we were given more time, we would be able to add the following features:

- Adding notifications on job application to recruiter.
- Making comment page live with respect to others.
- Making it scalable and distributed.
- Adding previous work to profile.
- Reposting others' posts.
- Filtering jobs.

7 Our Learnings

During the course of this project, we learnt many new concepts:

- How to send and receive media like photos to and from frontend and backend.
- How to deal with documents such as pdf for dealing with resumes.
- How to create an auto scrolling feed.
- How to toggle buttons instantly.
- JavaScript concepts such as async-await.
- How to make a secure authenticating and authorizing frontend.