NETS2120 Final Project Report: **Stream Team**
Group Members: Linda Shen, Iain Li, Chanya Thanglerdsumpan, Joanna Fan

Overview: Our group, **Stream Team**, developed **Pennstagram** that showcases our technical skills in building a scalable and feature-rich web application. Throughout NETS2120, we've all acquired invaluable skills that have significantly advanced our software development capabilities. These skills include building cloud-hosted indexers with React and Node.js, creating client-server web applications with DynamoDB, and developing social network applications that utilize multimodal data, vector, and object storage (and more).

Within this project, we make use of powerful tools and services such as Amazon S3, ChromaDB, DynamoDB, EC2, and RDS, all of which serve significant roles in the architecture of our application. We integrate these technologies to support dynamic image sharing, foster user interactions through likes, comments, and group chats, and incorporate a recommendation system employing Kafka, ChromaDB, and NLP techniques. This project has required us to implement features such as user account management, real-time data streaming, and content recommendation, mirroring the functionalities of mainstream social media platforms.

Technical Components:

1. System Architecture and Backend: Pennstagram is hosted on Amazon EC2, utilizing Node.js for backend operations.
2. Database Systems: We employ Amazon RDS for storing our structured data such as our user profiles, posts, group chats, relationships between likes/comments and users, etc. For storing non-relational data (ie. storing large objects like images and videos), we use Amazon S3 to store media from posts. Furthermore, we integrated ChromaDB for image embeddings. When a user registers and uploads a profile picture, we fetch from our ChromaDB embeddings to gather the 5 most similar actors to this user.
3. User Accounts: We implemented our registration system using ChromaDB to handle embedded data within profile photos and actor lists. Users can also dynamically update their profiles, including changing their preferred hashtags, which influence the feed algorithms to customize content delivery. The photos are uploaded to S3. In addition, users can select hashtags and edit the selections in edit profile photo. Logging out, the user's session will be cleared.
4. Social Interaction Features (Chat, Add/Remove Friends, Groups): We offer real-time communication through different chat rooms that use AJAX for message querying, and Axios routing for sending chat room invites. Using these polling methods, we can maintain a persistent connection for live updates and messaging within group chats. We also incorporated a system that will check and display the online status of a user's friends, and if they are active, we can send them a chat invite and they will see that in real-time. For extra credit, we have implemented a similar friend invitation request to

LinkedIn where users can either accept or decline a friend request for a non-friend. Another aspect that we have created is group/communities, where users can search for groups that they're interested in, join them and be able to join a chatroom with all the other group/community members (EC).

5. Chat: Other than what is explained above about chats, users must first send an invite to their friend that they want to create a chat with. Once the other user has accepted the chat invite, a new chat object/room will be created with just the two of them. However, we also implemented an invite option where you can add your other friends into the group chat as well. That friend can then choose to join this existing chat group, see all the previous text history, and also decide to leave whenever. The chat will only be removed from the user's chat interface once they leave.
6. Feed: We employed the SocialRank algorithm to assess user preferences and prioritize what content will be shown onto the user's feed. Although all posts (public posts, friends of users, Twitter/Federated posts) are shown on the feed, SocialRank orders these in terms of prevalence for each user so that they will be able to see more prevalent figures or popular posts first. Furthermore, users are able to publish their own content, which is then sent through Kafka channels to reach other groups and external feeds. Our system supports multimedia posts, which are optimized for quick distribution and rendering (through using S3 and also fetching information from our RDS tables). Our feed relies on our posts table, which contains the author's ID, post ID, content, and a reference to our S3 bucket URL. We also created other tables to keep track of the likes, comments, and hashtags that linked to our main posts table.
7. SocialRank: Social Rank works by employing edge weights to simulate a random walk to the most "relevant" content based on links between users, posts, and hashtags.
8. Search (Natural Language Processing) and Recommendations: We leverage GPT models to implement NLP techniques, enabling users to perform searches using natural language queries. We used OpenAPI Embeddings and Vector Storage to query search results based on our post and user tables. Furthermore, using Apache Spark, we implemented the adsorption algorithm to personalize content and friend suggestions. This system analyzes user interaction data to model preferences and suggest relevant new connections and content.
9. Media Handling: S3 is used for bulk storage of media files with high reliability, while ChromaDB manages the embeddings for advanced image retrieval functionalities.

Design Decisions:

In terms of the database design choices, we wanted all the tables created in the database to have data integrity and abide by the 3NF form. For instance, when we created the **chats** table with the primary key **chat_id**, we created additional tables such as **user_chats** that work as a "link" between the **users** and **chats** table, and we leverage off the foreign key reference concepts for us to be able to trace users back to the chats they're in (same thing with text_id, invite_id, and

several others). We initially wanted to add users as an array of string inside the **chats** table, but we realized that that would require extra parsing on the front/back end and it would also not be possible to foreign key reference an array to an INT (of a singular **user_id** on the **users**) table. Hence, we decided to go with the solution mentioned above. We continue to use this "modular"/normalized structure for all tables including posts, invites, and more.

In terms of our frontend design choices, we chose a minimalistic approach. This project gave us the flexibility to make our own UI/UX choices, so we chose an approach that was user friendly and intuitive.

Lessons:

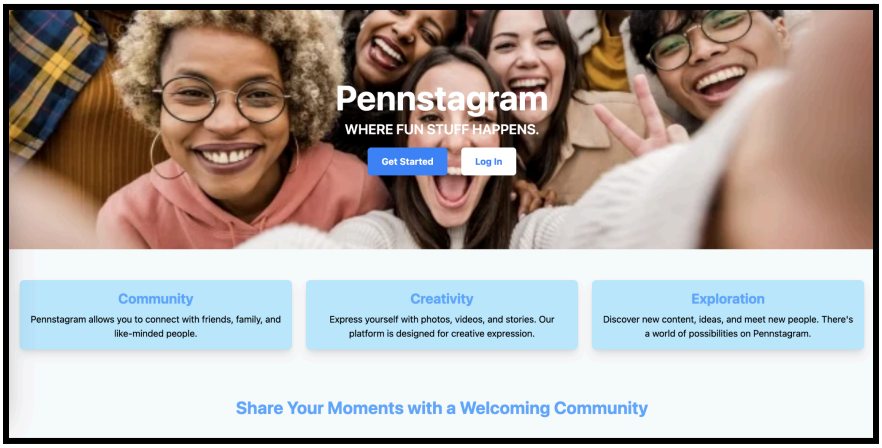
1. Technical Documentation: Given that there were a multitude of features and tech stacks that this project dealt with, we maintained robust technical documentation throughout our project. This included detailed readme's that allowed members to set up more efficiently, as well as commenting and communicating each change we made throughout our code and Github actions.
2. Collaboration: Our group maintained consistent communication throughout the course of this project which significantly streamlined our processes. We were able to delegate tasks while also creating flexible milestone plans in the case that we faced any bottlenecks in our implementation process.

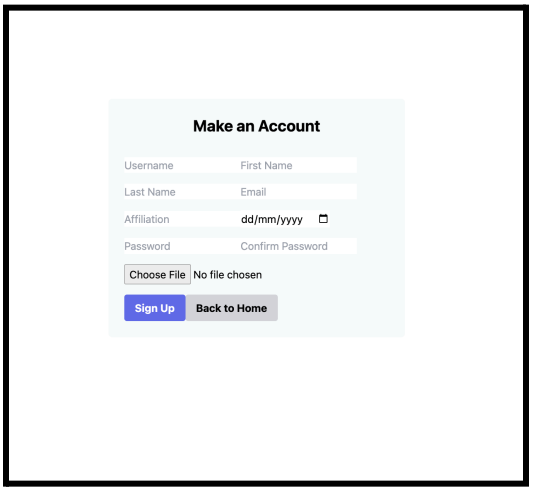
Extra Credit:

1. Friend invites
    a. In order to see who you're friends with, search for new friends, and remove current ones, users can navigate to the "Friends" page. When searching for a new friend to connect with, users can first search for their username and send a friend invite request, which would then be sent to the other user's invite list. The user then either has the choice to accept/decline the request. Once accepted, both users will become friends with each other (and can use friend-only features such as creating a chat together).
2. Groups (similar to Facebook communities)
    a. Users can create and search for "Groups" to join in the "Groups" interface of the app. Very similar logic to how the Friends page is, except that no invitation is required and we also added a feature where they are automatically added to a group chat with other group members.

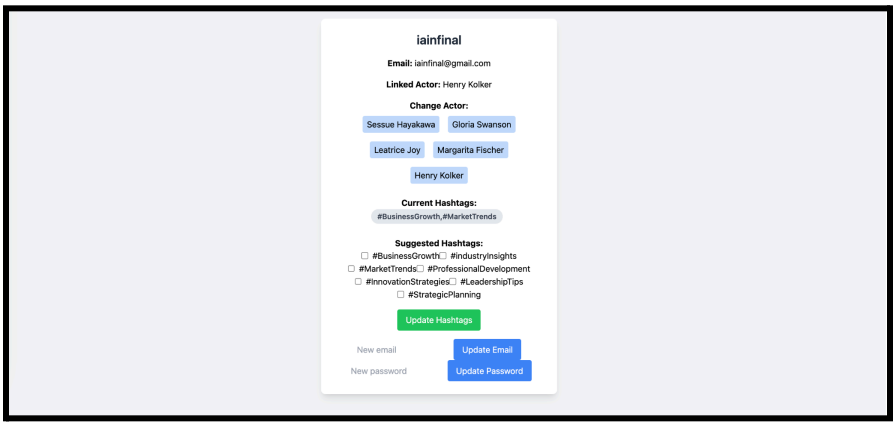
App Highlights:

Login / Registration

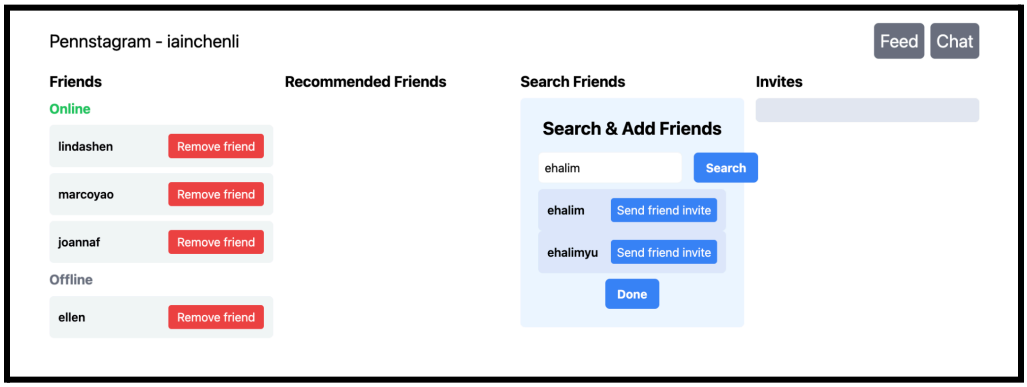


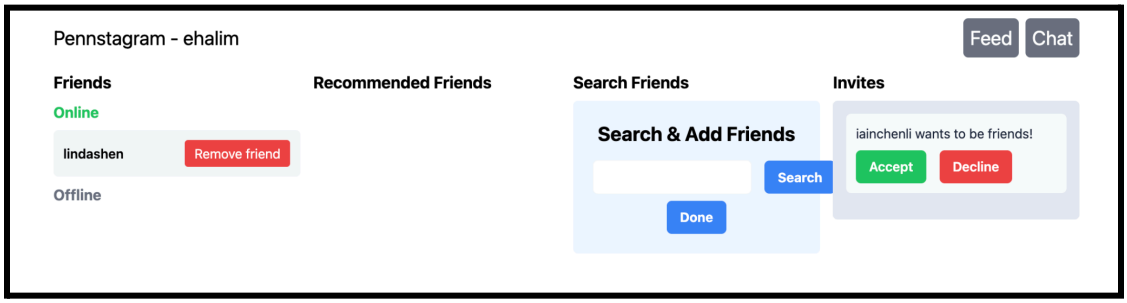


Profile Page/Edit Profile

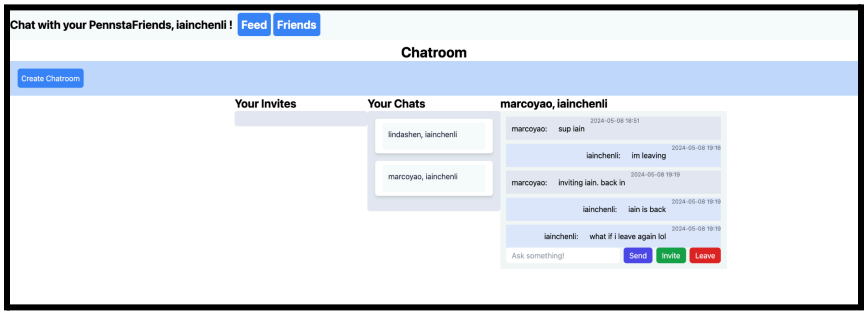


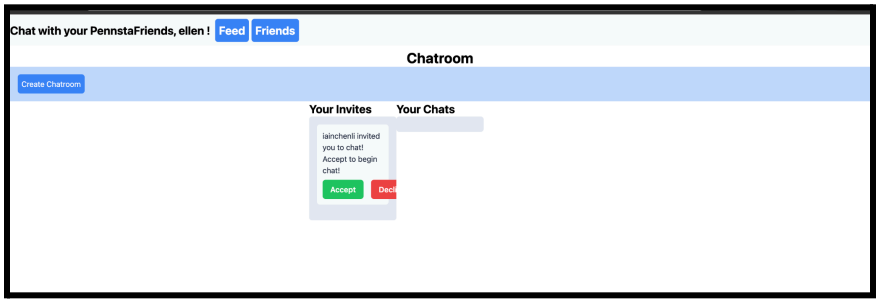
Friend Requests



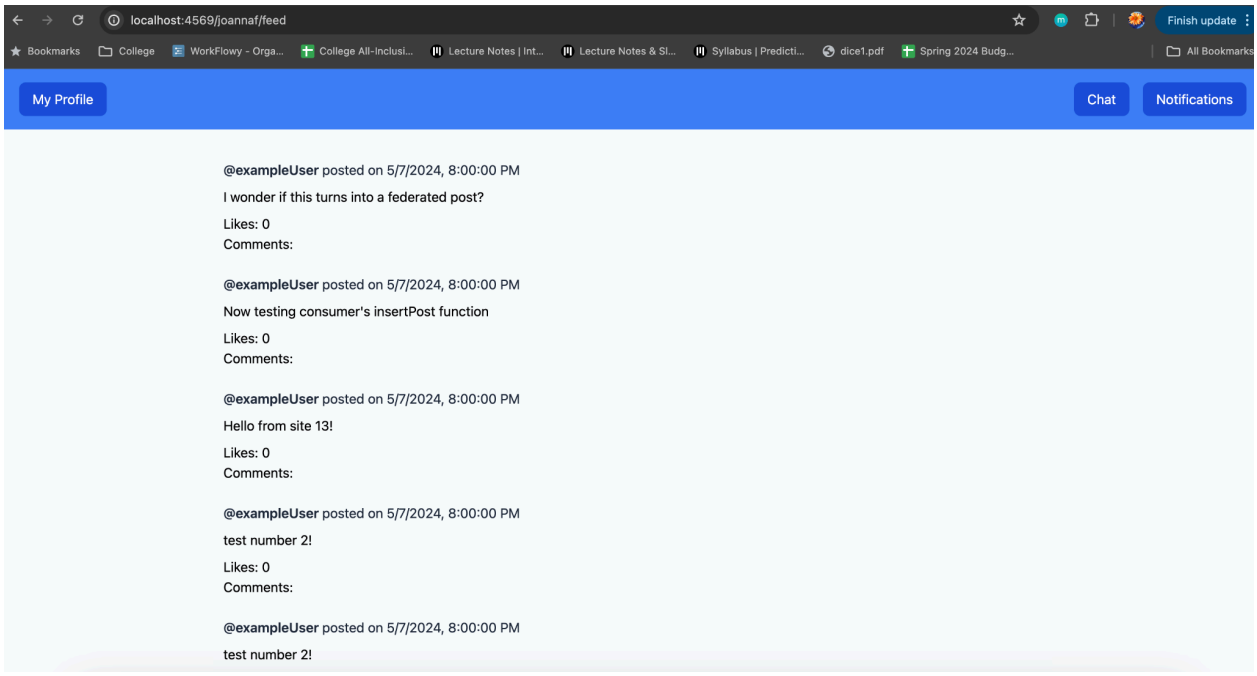


Chat Room





Feed


localhost:4569/joannaf/feed
Finish update
Bookmarks
College
WorkFlowy - Orga...
College All-Inclusi...
Lecture Notes | Int...
Lecture Notes & Sl...
Syllabus | Predicti...
dice1.pdf
Spring 2024 Budg...
All Bookmarks
My Profile
Chat
Notifications
@exampleUser posted on 5/7/2024, 8:00:00 PM
I wonder if this turns into a federated post?
Likes: 0
Comments:
@exampleUser posted on 5/7/2024, 8:00:00 PM
Now testing consumer's insertPost function
Likes: 0
Comments:
@exampleUser posted on 5/7/2024, 8:00:00 PM
Hello from site 13!
Likes: 0
Comments:
@exampleUser posted on 5/7/2024, 8:00:00 PM
test number 2!
Likes: 0
Comments:
@exampleUser posted on 5/7/2024, 8:00:00 PM
test number 2!