

System Overview

- Frontend – Tailwind, Vite, and React
- Backend – Node.js and Express
- Server – MySQL/RDS databases
- Embeddings, Image Search – ChromaDB
- Image storing – S3
- Streaming incoming posts – Kafka
- Adsorption and social rank algorithm – Apache Spark
- Natural Language Search, RAG – OpenAI
- Security and Sessions – Passport
- Chat – Socket.io

Component Technical Description

Frontend: We use React for each page of our application, and for components such as sidebars and posts, we reuse them throughout different pages. We use React-Router to handle multiple views. We also use the useEffect React Hook to load in data when each component mounts, and Vite and React for the base to make building the app more efficient. For project styling, we utilize Tailwind CSS with the shadcn/ui and radix-ui component libraries.

Profile: After the user registers for a profile, they are directed to a profile page where they can upload a selfie. We use two S3 buckets to store user selfie and actor photos. When the user logs in, we encrypt their password and they are able to view all of their friends. Additionally, we use ChromaDB for photo embeddings so that we can query the top 5 most similar actors. We use MySQL/RDS for storing all of the user information, such as friends, usernames, user hashtags, actors and their id link in S3, etc. We provide top 10 hashtags based off of popularity, and if the user types in a prefix, our frontend displays top 10 hashtags that completes it.

Feed: When users post, our backend routes the data through Kafka. A listener then triggers Spark jobs to rank posts. When users check their feed, the backend retrieves top-ranked posts from the Spark-generated rankings. For the ranking algorithm, we used a modified version from HW3 as detailed in the Project Instructions, and now we have the rankings for each user's perspective and save it to our database. When the user makes a post, we push the post to both our personal Kafka instance to create a message and to the communal Kafka with the federated posts. We then pull information from both Kafka instances to the feed/Kafka side server, which saves it to the database. Our backend reads from the database. Our feed/Spark server reads the database, sends relevant information to Spark to run the adsorption calculations, gets the data back, and sends it to the database. Then our backend queries everything and sends the result back to the user through the frontend.

LLM: We use OpenAI models to perform RAG on our data, and we ask the models to generate queries that we run against the database and post content embeddings and return summarized results.

Friends: We made SQL queries to our database to fetch a list of user's friends, add and remove friends. As for friends who are currently logged into the current system, we implemented this through websockets. Additionally, we used link structure of the graph and activity and hashtag structure of the streaming posts to return a list of recommendations for "who to follow" every day.

Chat: We utilized WebSockets with Socket.io to implement the chat features, such as inviting a friend to chat, creating private and group chats, and leaving chats. Additionally, we created several databases to store data such as invites, messages between users, chat sessions, and chat members. To access or make changes to this data, we made SQL queries to the database.

Nontrivial Design Decisions

Frontend: Some design decisions we made include using a component library, Tailwind in-line styling, and using a Figma wireframe to map out the structure of each page

Profile: We decided to separate user registration and setting up a selfie into two actions. After account creation, we direct users to the profile page where they can set the image.

Feed: We had three different servers that did the following: Server 1 (backend) serves user requests, Server 2 (feed/Kafka) pulls posts and feeds the database, and Server 3 (feed/Spark) runs all spark calculations. We did not want to lump all of our functionalities into the backend so we modularized the app. Additionally, we chose to dockerize for ease in testing locally with different setups and adding or changing services when we migrate.

Chat: For the chat, the logic for invites, private, and group chats influenced the way we set up the database. For instance, we had tables for chat invites, chat sessions, chat members, and messages.

Changes made along the way

Frontend: Initially, we made components from scratch before deciding to switch to a component library for both ease of use and also for styling purposes. We also started off with buttons to navigate to different pages in our application, but as the project progressed we implemented a sidebar to improve user experience and for a cleaner appearance. Furthermore, we originally used the URL to keep track of the username, but we decided to switch to the backend to keep track of the username via cookies.

Profile: We changed the schema quite a few times as we realized what information we wanted to include in our application and how we could optimize for the best structure. We added necessary attributes and created tables for querying hashtags and for pairing up users and hashtags.

Feed: We began by integrating the feed and spark backend into the same server, but it became extremely difficult to manage all of the different requests coming in and out of the server. As a result we switched to a modularized approach where we created separate servers for different functionalities. Furthermore, we dockerized our project because when we were running all the services to test our project, we would have to open up five different terminals which made it difficult to run, and this change made it easier to replicate and test our project.

Chat: We started off implementing chat using basic AJAX requests with polling. However, as the project progressed, we realized that it would make more sense to use WebSockets using socket.io to allow for a persistent connection, which made writing queries for the chat significantly easier to implement.

Lessons Learned

Most of the lessons that we learned along the way of completing this project were taking concepts that we covered in lecture and homework assignments, and building off of these to create our own social media app, Pennstagram. By developing a website application from scratch, we learned how to utilize these systems to work seamlessly together, build a deeper understanding of how all of these technical elements work together, as well as produce a cohesive final deliverable. Additional lessons that we learned include not biting off more than we could chew in terms of feature implementation and multi-tasking. Coming into the project, we were very ambitious in completing all of the extra credit tasks in addition to the core functionalities of the project. However, parts of the project were more difficult than we anticipated and we were only able to implement a portion of the extra credit features. Additionally, integration for the project was challenging as we were all working asynchronously on coding our individual parts, which made connecting the frontend and backend a time-consuming team effort as we discovered new bugs as we progressed.

Extra Credit:

Implementing Chat using WebSockets with socket.io: We utilized socket.io in our chat function to allow users to create private messages and group chats amongst each other, while ensuring persistent connection and exchange data in both directions between the server and client.

Infinite Scrolling: When scrolling through the Pennstagram feed, the user is able to continuously scroll through posts infinitely until they run out of posts to look at.

The LLM search always returns valid links for search results: We ensured that the search results return links so users can add friends, follow, and find interests based on the returned results through the OpenAI models.

System in action (one for each main feature/page)

Login

Pennstagram







Don't have an account? [Sign up](#)

Signup

Pennstagram
Sign up to see photos and videos from your friends.

Already have an account? [Log in](#)


Feed




Discover


I'm not sure what you're asking.

Relevant Posts

**zminsc**



why

 0 likes

[View Comments](#)

Post



New Post

What are you thinking about?

Choose File No file chosen

Create

Friends

Home

Profile

Friends

Messages

Search

Share

Friends

z

zminsc

Steven Chang

zachives

Zachary Ives

zawmin

Zaw Min

Recommendations

My Friends

zminsc

Steven Chang

Profile

Actors

Current Actor

Change Actor

Search for an actor...



Chat

Home

Chats

Invites

Friends

zminsc

Online

zachives

Offline

zminsc

hey who are you?

Thursday 09:36

zawmin:

yo it's ncie to meet you

Thursday 09:37

zminsc:

we are so cool

Thursday 09:37

zawmin:

yeaaaaaaaaaaaa

Thursday 09:37