Instalite Technical Report

Ahmed Muharram, Kidus Seyoum, Eleftherios Lazarinis, Annabella Tian

A Brief Overview

Our project is a web application with features and functionality similar to Instagram. To start, users can login and register. This gives them access to the application where they can connect with friends, view posts and updates from friends, see what's trending, chat with friends, and search across the application. Through these features and many more, users can interact with their friends online as well as gain access to a community online through hashtags of similar interests, recommendations, and seeing what's trending.

Technical Components

Login + Signup

When a user logs in, we set a cookie to maintain their session, ensuring they can access the application's features without having to log in repeatedly. To ensure security, we hash the user's password and compare it to the hashed password stored in our database to protect user's privacy and enhance our applications security

Our signup process allows users to create an account with a variety of personal information and upload a profile picture. This profile photo is uploaded to S3 such that we reference the link throughout the application to get the photo. We create an embedding of this photo and use ChromaDB with the IMDB actors as well as the face API to suggest similar-looking actors based on the uploaded picture that users can link to their account.

In case a user forgets their password, we use the Nodemailer module to send them an email with a link to reset their password. We configured Nodemailer to send an email from our unique application email 'kalaproject6@gmail.com' with a link that includes a reset token that is verified on the server side to ensure that the password reset request is legitimate. Using a transporter, the email is sent; this feature adds a layer of security by ensuring that only the legitimate user can reset their password.

The users are stored in a table with columns for: username (primary key), hashed password, birthday, email, hashtags, profile picture, number of friends, linked actor, bio, and affiliation.

Feed

Our feed feature allows users to create and view posts. Posts can be text-only, include images, or use HTML, providing flexibility in how users can share content. To ensure that the most relevant posts are shown to users, we implemented a ranking algorithm that orders posts based on various factors such as recency, engagement, and relevance. To ensure this ranking is always updated, we reload the page every hour so users can see the new top posts. We use SparkJava for this task, leveraging its parallel processing capabilities to efficiently handle large volumes of data and update post rankings.

Users can interact with posts by commenting on them and liking them. These interactions are visible to all users and updated synchronously across the platform for all users. To provide a better browsing

experience, we implemented infinite scrolling, loading more content as the user scrolls down the page, allowing users to discover new content without having to navigate away from the feed.

The posts are stored in a table with columns for: post id (auto-incremented and primary key), parent post (for comments), post type, author, title, content, hashtags, likes, created at timestamp, and origin server (for Kafka posts).

Chat

To enable real-time communication, we set up a server that supports Web Sockets. When a user accepts a chat and opens the chat group, their browser establishes a Web Socket connection to the server that remains open as long as the user is using the chat feature, allowing for real-time communication. When a user sends a message, it is sent over the WebSocket connection to the server. The server then broadcasts the message to all connected clients, including the sender and any other users in the chat, allowing for real-time chats to occur. This is all implemented event handlers on the server to handle events such as when a user sends a message. When a user leaves the chat or closes the application, their WebSocket connection is closed. This ensures that resources are not unnecessarily consumed by maintaining inactive connections.

Implemented in our application, the chat feature allows users to invite others to chat, and the chat history persists between sessions, allowing users to continue their conversations seamlessly.

Profile + Settings

The profile feature fetches user information from the RDS backend and displays it to the user. This includes basic information such as name and profile picture, as well as more detailed information such as friends list and posts. By using cookies, we can determine if the user is viewing their own profile or another user's profile. This allows us to customize the profile view based on the user's identity, optimizing our frontend in terms of efficiently rendering components with less redundancy.

The settings feature allows users to change information about their profile, such as their name, email, and profile picture. We use React and JavaScript to update the page dynamically after the user submits their changes, providing instant feedback to the user.

Requests

Users can send friend requests to other users, who can accept or reject them. We use the backend to connect to the frontend and update the UI synchronously, ensuring that the user's request is processed in real time.

Recommendations

To provide personalized recommendations, we use the "friend of friends" concept, suggesting people to follow based on mutual connections. This makes it easier for users to find others on the platform and increases engagement. We use SparkJava for the algorithm to allow for greater parallelism, allowing us to process large amounts of data efficiently and provide synchronous, updated recommendations to users.

Trending

The trending feature uses a ranking algorithm to find trending hashtags across the platform. We use SparkJava for this task, leveraging its parallel processing capabilities to analyze large volumes of data and identify popular hashtags. The UI displays trending hashtags along with three photos for each hashtag, based on the number of likes. This provides users with a visually appealing and informative overview of the trending topics on the platform.

Search

For search functionality, we generate a string with all relevant information about users and posts then using the OpenAI API to generate embeddings, vector representations of the content. These embeddings are stored in a Chroma collection that is referenced by a VectorStore. We then use VectorStore to be able to efficiently access the embeddings and execute a similarity search for user's queries. Thus, the search finds the top 10 similar results from a given query, ordered by greatest similarity.

Challenges + Lessons Learned

Database

Throughout our project, we learned a great deal about the importance of creating a comprehensive schema for our database tables. Initially, we didn't fully anticipate the complexity of our application's data requirements, which led to several challenges as we progressed as we had to adjust our database schema multiple times. This often required us to drop tables and lose the data they contained, which was frustrating and time-consuming. We found ourselves creating new tables that we hadn't initially planned for. We learned the importance of careful planning and consideration of future requirements and we hope to apply these lessons to future projects to ensure smoother development processes and avoid unnecessary challenges.

Github

Throughout our project, we encountered several challenges related to pushing and pulling on GitHub and resolving merge conflicts. We often faced merge conflicts, which often occurred when multiple team members were working on the same files simultaneously. Manually resolving these conflicts required communication across the group to ensure all needed changes were implemented with each push/pull. At one point, we had to revert to a previous commit due to a critical bug. While this helped us resolve the bug, it hindered our progress as we had to update all the files with our changes again. This process took a considerable amount of time and effort to ensure that all commits were working versions of the code. Overall, these challenges taught us the importance of effective collaboration, communication, and careful management of code changes in a team environment, an especially important skill

Other features

● LinkedIn-style friend requests with confirmation: As explained above, implemented so users can accept or deny friends before a friend connection is made. Shown on the requests page from the sidebar navigation of the application.

● A "forgot password" option: As explained above, uses Nodemailer to send a password reset email to the user's associated email address. Uses reset token for greater security. ● Infinite scrolling: Implemented on the feed so that 10 posts are rendered everytime the user reaches the bottom of the feed. Users will be able to scroll until all posts in the database have been rendered on the feed.

● Groups: Users can join a group with other users so that they can see a curated feed of posts only created by users in that group for that group. Users can create groups and users can join groups.

● Site-wide "what's trending" feature: As explained above, users can look at the trending page to look at popular hashtags as well as the top three posts for each.

● WebSockets for chat: As explained above, each chat creates a websocket that gives a dedicated connection for users in the chat.

● The LLM search always returns valid links for search results: The search takes the top 10 results from the user's query and renders user and post components that always link to users and posts that exist in the platform/database.