Overview of the system

Technical description of the components

User Registration

User registration is handled through SignupForm.jsx, which creates a user registration form using React. It utilizes React features and hooks such as useState, useEffect, and useHistory. The form captures essential user information like username, password, name, email, school, birthday, and allows users to upload a profile photo and include hashtags of interest as well. The uploaded photo is then compared to images in the actor database and the top five most similar actor images are displayed for the user to choose from. It also includes functionality for adding and removing hashtags of interest, with a list of top ten hashtags for reference. Upon form submission, the component sends a POST request to a local server endpoint /api/users/signup to register the user. Successful registration redirects the user to the home page using react-router-dom's useNavigate hook. Additionally, it provides a link to log in for existing users.

Login

Login is handled through LoginForm.jsx, which is responsible for rendering a user login form using React. Leveraging React's useState hook, it manages form state for email and password. useUserContext accesses the user context to dispatch login actions upon successful authentication. This component employs useNavigate from react-router-dom to handle redirection upon successful login, directing users to the home page. When the form is submitted, it triggers an asynchronous function to send a POST request to the backend API endpoint /api/users/login for user authentication. If the login attempt succeeds, it stores user information in local storage and updates the user context, followed by redirecting the user to the home page. Additionally, it provides a link to sign up for new users.

Profile

Profile changes are handled through Profile.jsx, which allows for a user profile management interface using React. It employs React's useState hook to manage form state, including fields for updating the user's profile photo, email, password, and hashtags of interest. Users must edit at least 1 field, but can choose which fields they would like to update. Utilizing useParams from react-router-dom, it extracts the userId from route parameters, enabling personalized profile updates. Upon form submission, it executes an asynchronous function to send a PATCH request to the backend API endpoint /api/users/:userId, updating modified profile fields.

Friends

-

Chat

- Used socket.io to implement chat.
- Using socket.io, the server side waits for the client to emit and then the server side while emit to the entire room to send messages
- Must only chat with friends through friendlist if they are online
- Can invite a user that is a friend of the inviter to a group chat or a one on one. They will get a request and have to accept to join the chat

Kafka Feed

- Initialize a consumer for the Twitter-Kafka and FederatedPosts topics
- Subscribe
- Run and add each post into the db
- Used cron to run on a daily basis
- For producer, whenever a user creates a post, it gets added to the FederatedPost topic

Who to Follow

- Find friends of friends.
- Count the # of common friends of friends
- Rank based on occurrence
- If user has no friends, find # of common hashtags of users
- Rank based on occurrence

Adsorption

Adsorption ranking of posts and individuals is performed through Apache Spark and the creation of JavaRDD and JavaPairRDDs through parallelization of the persistent data stored in our RDS to represent the user, hashtag, and post nodes. Edges are then created through another JavaPairRDD, as well as label weights, initialized on each of the users with a weight of 1.0. At the beginning of each cycle, operations such as mapToPair and reduceByKey are performed on the RDDs to combine necessary relationships and propagate label weights in the forward from the users to the other nodes. The label weights are then normalized to sum to 1.0, and the same process is performed to propagate the label weights backwards to the users. The weights are then again normalized, and in order to reduce "wash out" on the label weights, we hard code the weight of the origin vertex to always be 1.0. We repeat this process for 15 iterations or until convergence through a for-loop, and then output the label weights for each hashtag and post. Finally, we use this information to sort through the posts and hashtags for each user in terms of descending user label weight in order to determine the posts to suggest to that user. Posts that have already been suggested are not recommended, and this process is repeated every hour.

S3 Storage

We created an S3 bucket for large object storage. This includes the images that are used for profile pictures. In order to upload from the front-end directly to the S3 bucket, we use Multer as middleware to handle form-data, which includes profile pictures, using a connection through the .env variables to the S3 bucket. Once uploaded, the images can be retrieved through the URL assigned to the images. In order to do so, we wrote a custom bucket policy in S3 to allow access to the objects in the bucket.

Image Search

Image search is implemented through a face-matching algorithm provided by the starter files. We use those files to call on the functions to store and access embeddings in ChromaDB. After the uploading of a profile picture, an embedding is created and compared to those of famous actors. The top 5 are chosen by the algorithm and outputted back on the frontend for the user to choose from.

Nontrivial design decisions

1.
2.

Changes made and lessons learned along the way

1.
2.

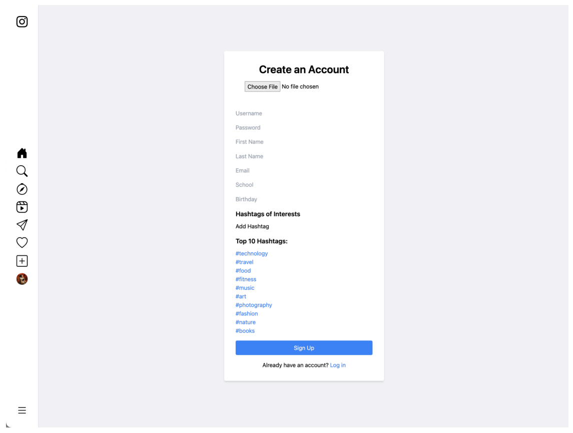
Extra-credit features

1.
2.

Screenshots

User Registration

Login Page


Create an Account

Choose File No file chosen

Username

Password

First Name

Last Name

Email

School

Birthday

Hashtags of Interests

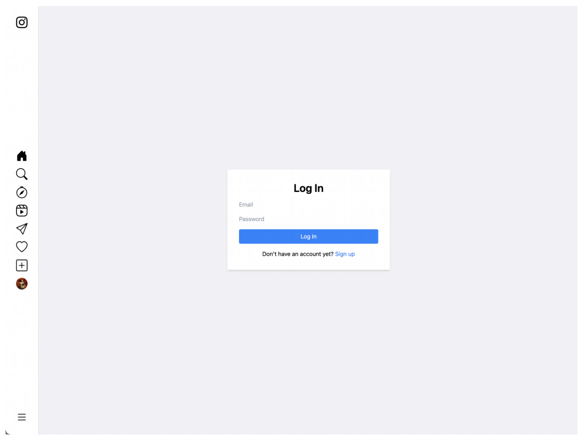
Add Hashtag

Top 10 Hashtags:

#technology
#travel
#food
#fitness
#music
#art
#photography
#fashion
#nature
#books

Sign Up

Already have an account? Log in


Log In

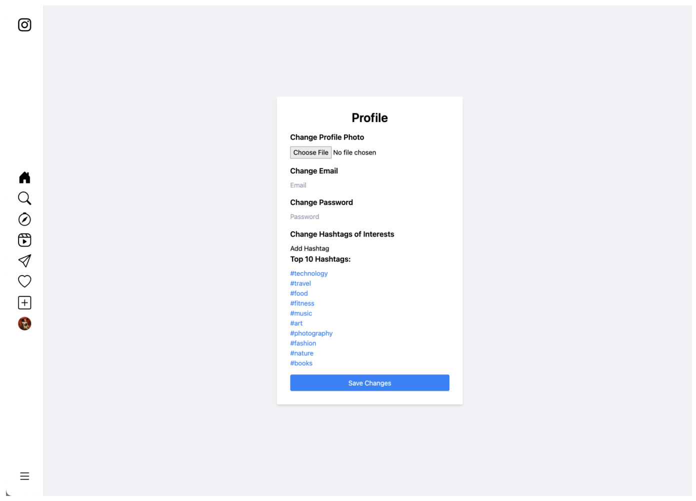
Email

Password

Log In

Don't have an account yet? Sign up

Profile Changes


Profile

Change Profile Photo

Choose File No file chosen

Change Email

Email

Change Password

Password

Change Hashtags of Interests

Add Hashtag

Top 10 Hashtags:

#technology
#travel
#food
#fitness
#music
#art
#photography
#fashion
#nature
#books

Save Changes

MySql Database Schema:

1. users Table
   - `user_id` (Primary Key)
   - `username`
   - `password_hash`
   - `email`
   - `first_name`
   - `last_name`
   - `birthday`
   - `profile_photo_url`
   - `created_at`
   - `updated_at`
2. posts Table
   - `post_id` (Primary Key)
   - `user_id` (Foreign Key linked to users)
   - `image_url` (optional)
   - `video_url` (optional)
   - `text`
   - `created_at`
   - `updated_at`
3. comments Table
   - `comment_id` (Primary Key)
   - `post_id` (Foreign Key linked to posts)
   - `user_id` (Foreign Key linked to users)
   - `text`
   - `created_at`
   - `updated_at`
4. friendships Table
   - `user_id_1` (Foreign Key linked to users)
   - `user_id_2` (Foreign Key linked to users)
   - `status` (e.g., pending, accepted)
   - `created_at`
   - `updated_at`
5. likes Table
   - `like_id` (Primary Key)
   - `post_id` (Foreign Key linked to posts)
   - `user_id` (Foreign Key linked to users)
6. hashtags Table
   - `hashtag_id` (Primary Key)
   - `text`
7. postHashtags Table
   - `post_id` (Foreign Key linked to posts)
   - `hashtag_id` (Foreign Key linked to hashtags)
8. userHashtags Table
   - `user_id` (Foreign Key linked to users)
   - `hashtag_id` (Foreign Key linked to hashtags)
9. chats Table
   - `chat_id` (Primary Key)
   - `name`
   - `created_at`
   - `updated_at`
10. chatUser Table
    - `chat_id (Foreign Key linked to chats)`
    - `user_id (Foreign Key linked to users)`
    - `status (pending, joined)`
11. chatMessages Table
    - `message_id` (Primary Key)
    - `chat_id` (Foreign Key linked to chats)
    - `sender_id` (Foreign Key linked to users)
    - `text`
    - `timestamp`
    - `created_at`
    - `updated_at`
12. groups Table
    - `group_id` (Primary Key)
    - `group_name`
    - `created_at`
    - `updated_at`
13. groupMemberships Table
    - `group_id` (Foreign Key linked to groups)
    - `user_id` (Foreign Key linked to users)
    - `role` (e.g., admin, member)
    - `joined_at`

API Specification

User Management

- Create User: `POST /api/users/register`
- Login User: `POST /api/users/login`
- User Top 5 Similar Actors: `POST /api/users/actors`
- Update User Profile: `PATCH /api/users/{user_id}`
- Delete User: `DELETE /api/users/{user_id}`

Posts Management

- Create Post: `POST /api/posts`
- Get User Posts: `GET /api/posts/user/{user_id}`
- Get Post: `GET /api/posts/{post_id}`
- Delete Post: `DELETE /api/posts/{post_id}`
- Like a Post: `POST /api/posts/{post_id}/likes`

Hashtag Management

- Get Top 10 Hashtags: `GET /api/hashtags`

Comments Management

- Add Comment to Post: `POST /api/posts/{post_id}/comments`
- Get Comments for Post: `GET /api/posts/{post_id}/comments`

Friendships and Interactions

- Get Friendships: `GET /api/friends/{user_id}`
- Create Friendship: `POST /api/friends/{user_id}`
- Update Friendship Status: `PATCH /api/friends/{user_id}`
- Remove friendship: `DELETE /api/friends/{user_id1, user_id2}`
-

Chat and Communication

- Create chat session: `POST /api/chats`
- Accept chat session: `POST /api/chats/requests`
- Send chat session: `POST /api/chats/send`
- Get chat sessions: `GET /api/chats`
- Send message: `POST /api/chats/{chat_id}/messages`
- Fetch chat messages: `GET /api/chats/{chat_id}/messages`

Groups Management

- Create group: `POST /api/groups`
- Add member to group: `POST /api/groups/{group_id}/members`
- Fetch group posts: `GET /api/groups/{group_id}/posts`

Search and Recommendations

- Search Users/Posts: `GET /api/search`
- Fetch trending posts: `GET /api/posts/trending`
- Fetch recommendations: `GET /api/recommendations`

Password Management

- Request password reset: `POST /api/users/forgot-password`
- Reset password: `POST /api/users/reset-password/{token}`

Privacy Settings

- Update post visibility: `PATCH /api/posts/{post_id}/visibility`