Secure Signup & Login System: Architectural Plan

This document outlines the plan for creating a secure user signup and login system, architecturally similar to a large-scale application like Instagram. We will use PostgreSQL and Cassandra as our databases, a Node.js backend, and a frontend that mimics the Instagram UI.

1. Technology Stack

- **Frontend:** HTML, CSS (Tailwind CSS for styling), JavaScript
- **Backend:** Node.js with Express.js
- **Databases:**
  - **PostgreSQL:** For storing core user data and relational information.
  - **Cassandra:** For storing user activity and other high-volume, non-relational data.
- **Password Hashing:** `bcrypt` library in Node.js

2. Database Schema

PostgreSQL (`users` table)

This table will store the primary user information.

Column	Data Type	Constraints	Description
id	SERIAL	PRIMARY KEY	Unique identifier for each user.
username	VARCHAR(255)	UNIQUE, NOT NULL	The user's unique username.
email	VARCHAR(255)	UNIQUE, NOT NULL	The user's unique email address.
password_hash	VARCHAR(255)	NOT NULL	The hashed and salted password.
full_name	VARCHAR(255)		The user's full name.
created_at	TIMESTAMP	DEFAULT NOW()	Timestamp of when the user account was created.

Cassandra (`user_activity` table)

This table will store user login activity. This is a good use case for Cassandra due to its high write throughput.

Column	Data Type	Key	Description
user_id	INT	Partition Key	The ID of the user from the PostgreSQL `users` table.
activity_time	TIMESTAMP	Clustering Key	The time of the activity.
activity_type	TEXT		The type of activity (e.g., 'login', 'failed_login').
ip_address	TEXT		The IP address from which the activity originated.

3. API Endpoints

The backend will expose the following RESTful API endpoints:

`POST /api/signup`

- **Description:** Creates a new user account.
- **Request Body:**
```
{
  "username": "newuser",
  "email": "newuser@example.com",
  "password": "a_strong_password",
  "fullName": "New User"
}
```
- **Functionality:**
  1. Validate the input (e.g., check for required fields, valid email format).
  2. Check if the `username` or `email` already exists in the PostgreSQL `users` table.
  3. Hash the password using `bcrypt`.
  4. Insert the new user into the PostgreSQL `users` table.
  5. Return a success message.

`POST /api/login`

- **Description:** Authenticates a user and creates a session.

- **Request Body:**
  ```
  {
    "username": "existinguser",
    "password": "user_password"
  }
  ```
- **Functionality:**
  1. Validate the input.
  2. Retrieve the user from the PostgreSQL `users` table by `username`.
  3. If the user exists, compare the provided password with the stored `password_hash` using `bcrypt.compare()`.
  4. If the password is correct, log the login activity in the Cassandra `user_activity` table.
  5. Create a secure session for the user (e.g., using JWT - JSON Web Tokens).
  6. Return a success message and the session token.
  7. If the login fails, log the failed attempt in the Cassandra `user_activity` table.

4. Security Features

We will implement the following security measures:

- **Password Hashing and Salting:** We will use the `bcrypt` library to securely hash and salt user passwords before storing them in the database. This prevents attackers from accessing plain-text passwords if the database is compromised.
- **Input Validation:** All user input will be validated on the backend to prevent common vulnerabilities like SQL injection and Cross-Site Scripting (XSS).
- **Secure Session Management:** We will use JSON Web Tokens (JWT) for session management. JWTs are a secure way to transmit information between parties as a JSON object.
- **HTTPS:** In a production environment, the application would be served over HTTPS to encrypt all communication between the client and the server.
- **Rate Limiting:** To prevent brute-force attacks, we would implement rate limiting on the login endpoint, limiting the number of login attempts from a single IP address.
- **CORS:** We will configure Cross-Origin Resource Sharing (CORS) to restrict which domains can access the API.

5. Frontend Development

The frontend will be a single-page application with two main views:

- **Signup Page:** A form with fields for email, full name, username, and password.
- **Login Page:** A form with fields for username and password.

The design will closely mimic the Instagram login and signup pages. JavaScript will be used to handle form submissions, make API requests to the backend, and manage the user's session.

This plan provides a solid foundation for building a secure and scalable login and signup system. Once you approve this plan, I will proceed with the implementation, starting with the backend and then moving to the frontend.