Social Network Application

Short Description

A web-based social network application allowing users to create profiles, post updates, manage friendships, and engage in real-time chat. The frontend is built with React, and the backend uses Django REST framework.

Requirements

Software:

- React (Frontend)
- Django REST framework (Backend)
- Docker (for deployment)
- WebSocket (for real-time chat)
- sqlite for database

Libraries and Frameworks:

- Redux (State management)
- Axios (API requests)
- Django Channels (Real-time communication)

Links:

- Frontend GitHub repository
- Backend GitHub repository

Problem it is Solving

The application facilitates social interaction by enabling users to share updates, connect with friends, and communicate in real time. It addresses the need for a centralized platform for social networking with robust features.

Components

- **Frontend:**
  - **Technology:** React, Redux, Axios
  - **Features:** User interface for profiles, posts, comments, friend requests, and chat functionality.
- **Backend:**
  - **Technology:** Django REST framework
  - **Features:** API endpoints for user management, posts, comments, friend requests, and real-time chat.
- **Server:**
  - **Hosting:** AWS (S3 for frontend, ECS/ECR for backend)
  - **Features:** Auto-scaling, load balancing.
- **APIs:**
  - **Internal:** Custom APIs for user profiles, posts, comments, friend requests, and chat messages.

System and Data Flow

Users interact with the React frontend, which sends API requests to the Django backend for user profiles, posts, and comments. The backend processes these requests, updates the database, and manages real-time chat through WebSocket connections using Django Channels. The database stores all the necessary data, ensuring seamless interaction and real-time communication between users.

- **Establishing the Connection**:
  - **Client Side (React)**: The React application establishes a WebSocket connection with the Django server using a WebSocket client library (e.g., `websocket` or `socket.io-client`).
  - **Server Side (Django)**: The Django backend sets up a WebSocket server to handle incoming WebSocket connections from clients.
- **Bi-Directional Communication**:
  - Once the WebSocket connection is established, both the client and server can send messages to each other at any time without waiting for a request.
  - This enables real-time messaging in chat applications where messages are instantly sent and received without the need for continuous polling or repeated requests.

Example Scenario:

- **Sending Messages**: When a user in the React chat interface sends a message, it's transmitted via WebSocket to the Django backend.
- **Broadcasting Messages**: Django then broadcasts the message to all connected clients (other users in the chat).
- **Real-Time Updates**: Each React client receives the message via WebSocket and updates the chat interface in real-time without refreshing the page or making additional HTTP requests.