Coding Exercise: Secure Client-Server Messaging Application

Objective

Design and implement a secure client-server application for encrypted communication between multiple clients. The application should handle authentication, encryption, and message broadcasting while adhering to best practices in security and scalability.

The client should be written in react, and the server in node.js

---

Problem Statement

Your task is to develop a **secure messaging application** with the following requirements:

1. **Server Functionality**:
   - Handles multiple clients concurrently.
   - Authenticates clients using a username-password combination.
   - Uses a **public/private key mechanism** for secure communication.
   - Stores messages in a database for audit purposes (encrypted at rest).
   - Implements message broadcasting: if a client sends a message, it should broadcast to all connected clients**. You are not allowed to use websockets**.
2. **Client Functionality**:
   - Allows a user to register with a username and password.
   - Connects to the server and authenticates using the credentials.
   - Encrypts messages before sending to the server (e.g., using RSA or AES).
   - Receives and decrypts broadcast messages from the server.
3. **Security Requirements**:
   - All communication between clients and the server must be encrypted (e.g., using TLS).
   - Messages stored in the database must be encrypted.
   - Passwords must be hashed securely before storage (e.g., bcrypt, Argon2).
4. **Additional Requirements**:
   - The application must handle at least 10,000 concurrent connections.
   - The server should log all significant events (e.g., logins, message sends) for monitoring purposes.
   - A basic API should be available for querying stored messages (accessible only to authenticated clients).

Deliverables

1. Source code for the client and server.
2. A README explaining:
   - How to set up and run the application.
   - Design choices, including encryption algorithms and frameworks used.
   - Trade-offs or limitations in your implementation.
3. A script for seeding the database with mock user credentials and messages.
4. Unit tests covering at least:
   - User authentication.
   - Message encryption/decryption.
   - Message broadcasting.