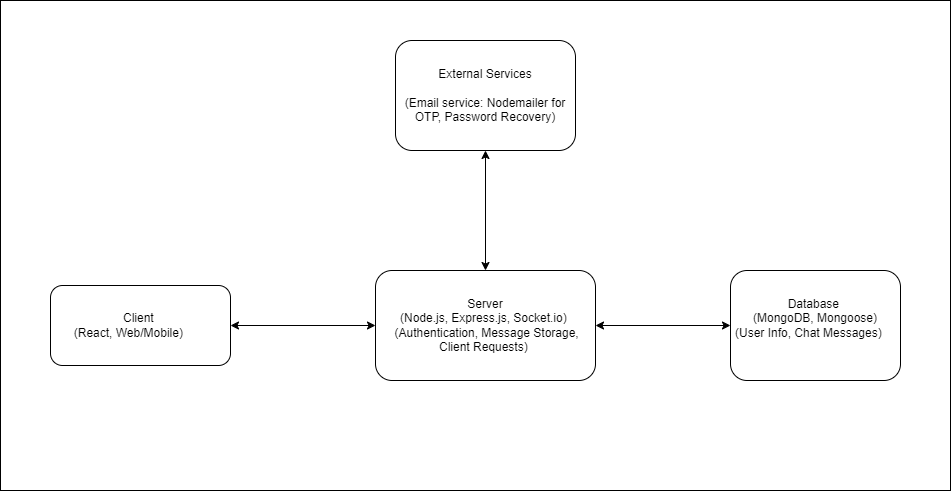
**Technical documentation**

1. Overall architecture design of the Secure Chat Application



|  |  |
| --- | --- |
| **Client** | |
| Technologies | React for web applications and mobile apps |
| Functionality | UI for client interaction with application, which includes registration, login, sending two-way messages and password rest. |
| **Server** | |
| Technologies | Node.js and Express.js for the backend server, socket.io for real-time messaging |
| Functionality | User authentication using JWT, manage client request, stores and retrieve messages from database, moreover have OTP authentication and password recovery. |
| **Database** | |
| Technologies | MongoDB for data storage and Mongoose for object data modeling (ODM) |
| Functionality |  |
| **External Services** | |
| Technologies | Nodemailer for sending emails. |
| Functionality | Sends OTP codes for multi-factor authentication and password recovery instructions, ensuring secure user verification and account recovery. |

A diagram of a software system

Description automatically generated

|  |  |
| --- | --- |
| User Registration | Users initiate the registration process |
| Password Input by User | While users enter the desired passwords, the application suggests guidelines for creating a strong password (e.g., using a mix of uppercase and lowercase letters, numbers, and special characters). |
| Password Strength Validation | Server receives the password input from user, server uses the zxcvbn library to evaluate the password strength, If the password strength is below a certain threshold, the server provides feedback to the client indicating that the password is too weak. |
| Feedback to User | Users are given suggestions for improving their password strength like, your password is too weak. Consider adding more characters, including special characters, and avoiding common words. |

1. Secure Password Storage Diagram

A screenshot of a computer

Description automatically generated

**Description**

1. **User Registration:**
   * Users initiate the registration process via the client (React application, either on web or mobile).
2. **Password Input by User:**
   * Users enter their desired password during the registration process.
3. **Password Strength Validation:**
   * The server, built with Node.js and Express.js, receives the password input from the client.
   * The server uses the zxcvbn library to evaluate the password strength.
   * If the password strength is below a certain threshold, the server provides feedback to the client indicating that the password is too weak.
4. **Password Hashing:**
   * Once a strong password is provided by the user, the server uses the bcrypt library to hash the password.
   * The hashing process includes adding a salt to the password to protect against rainbow table attacks.
5. **Store Hashed Password in Database:**
   * The hashed password, along with the salt, is stored securely in the database (MongoDB) using Mongoose as the ODM (Object Data Modeling).