

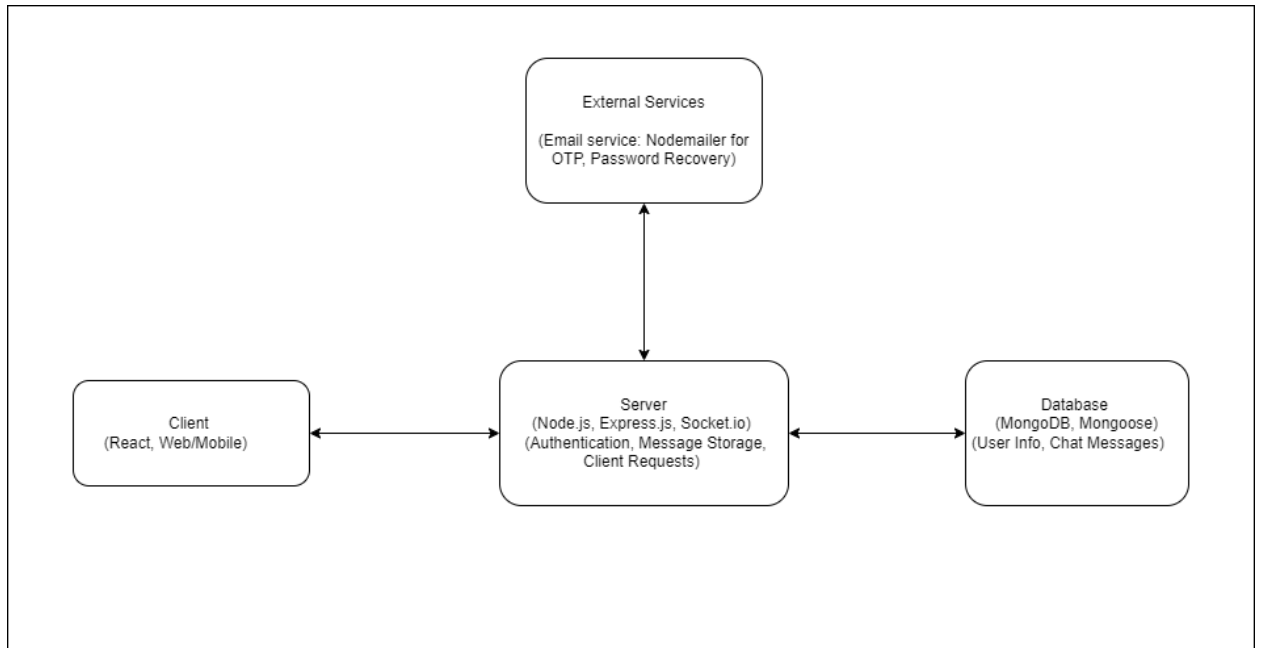
Technical Documentation

Secure Chat Application

Assignment 3 Option 2 Groups 24

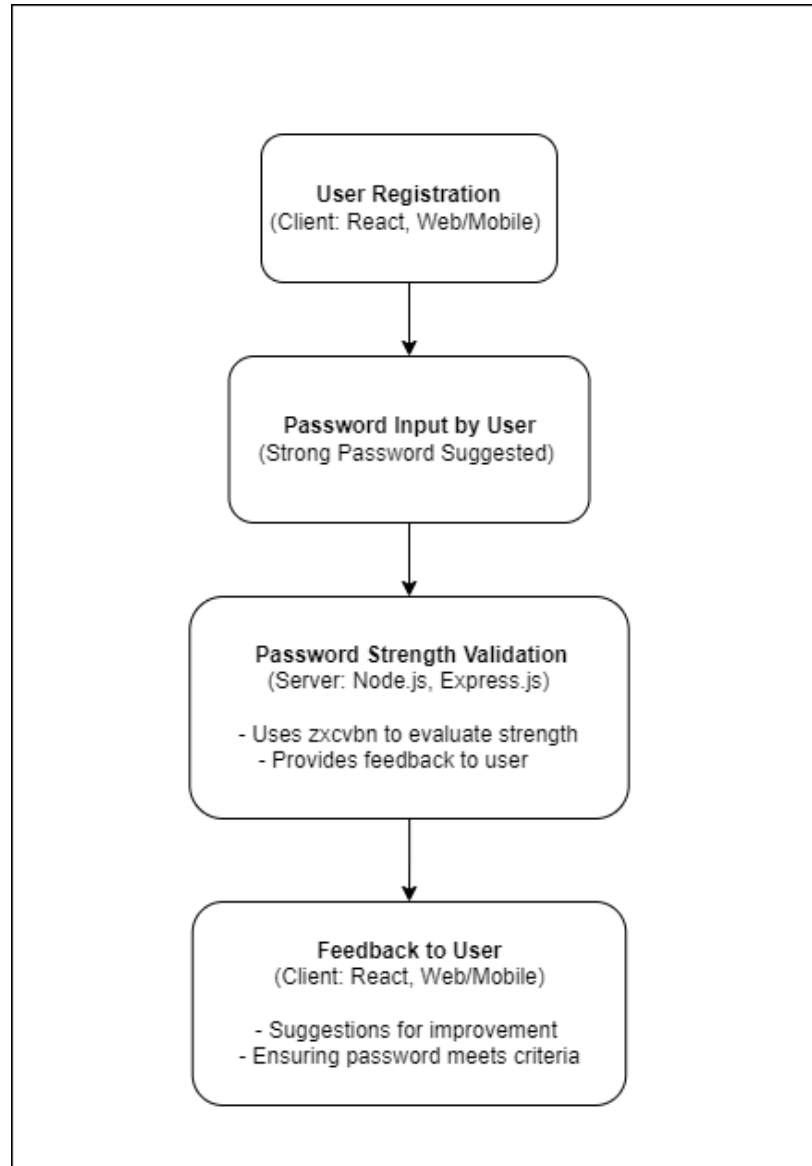
Team Members:	Akshika Wijeratnam s3994077 s3994077@student.rmit.edu.au
	Priyanka Paramananthan s4026308 s4026308@student.rmit.edu.au

(a) Overall architecture design of the Secure Chat Application



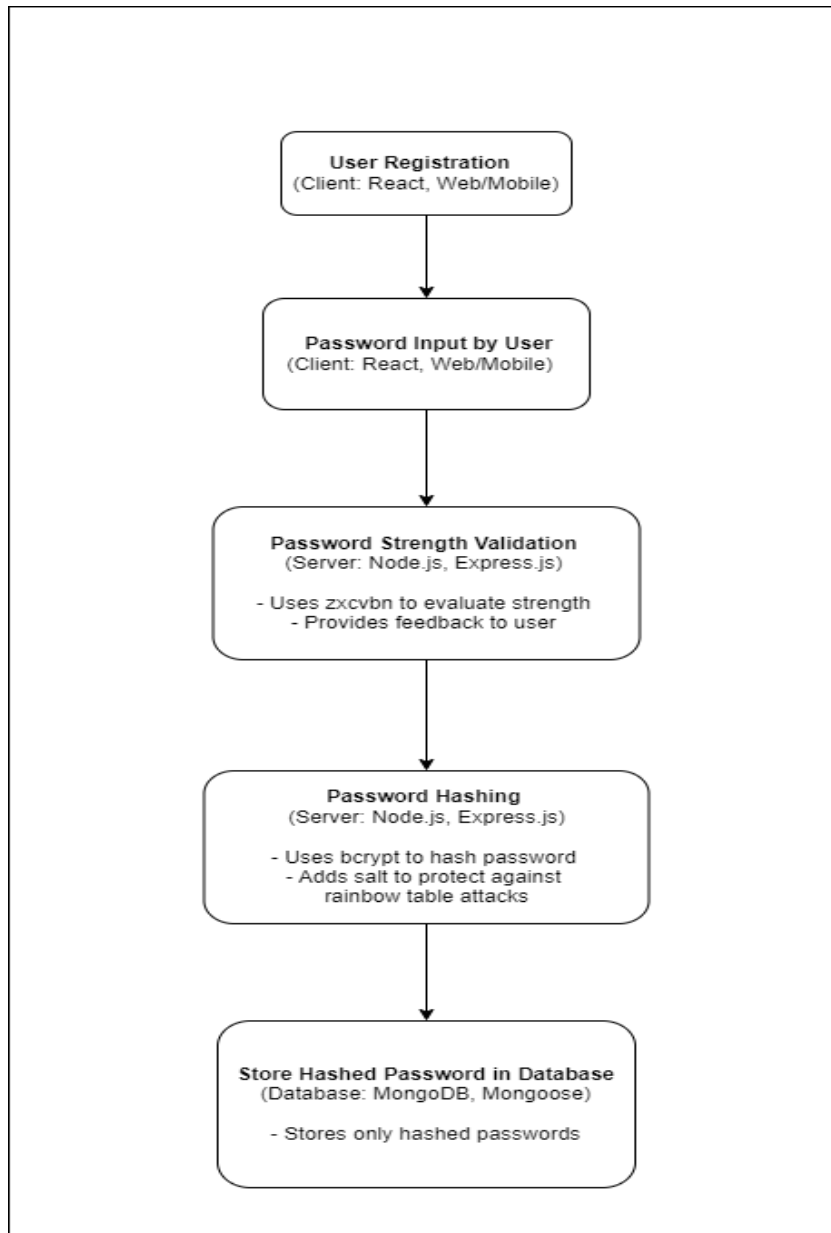
Client	
Technologies	React for web applications and mobile apps
Functionality	UI for client interaction with application, which includes registration, login, sending two-way messages and password rest.
Server	
Technologies	Node.js and Express.js for the backend server, socket.io for real-time messaging
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.

(b)



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.

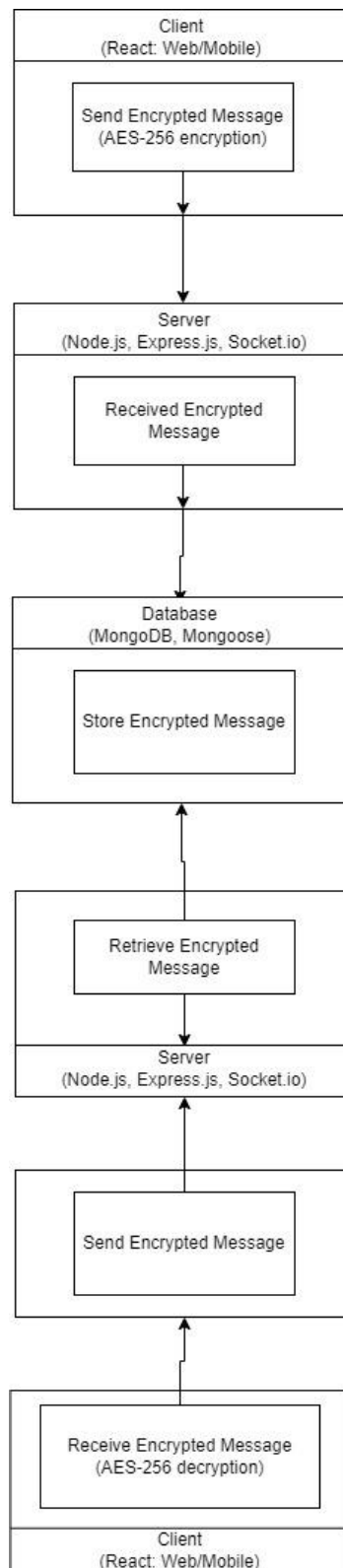
(c) Secure Password Storage Diagram



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).

(d) Secure Communication Diagram



Description:

Client: Through the client, users may send and receive messages (React application, either on web or mobile).

Before messages are delivered to the server, they are encrypted using AES-256 encryption.

Server: The server manages the message transfer between clients and is constructed using Node.js, Express.js, and Socket.io.

The client sends encrypted messages, which are then saved in the database.

The encrypted message is retrieved by the server from the database and sent to the recipient client when a message has to be sent.

Database: MongoDB securely stores encrypted messages with Mongoose serving as the ODM. Data security is guaranteed even in the event that the database is hacked since it only keeps encrypted communications.

Flow of Communication:

AES-256 encryption is used by the client to encrypt the communication.

The server receives the encrypted communication.

The encrypted communication is kept on file by the server in the database.

The encrypted message is retrieved by the server from the database upon a recipient client's request.

The receiver client receives the encrypted message from the server.

The communication is decrypted by the receiving client using AES-256 decryption.

(e) The Diffie-Hellman Protocol

A technique for safely exchanging cryptographic keys over a public channel is the Diffie-Hellman protocol. It makes it possible for the client and the server to create a shared secret key that they may use to encrypt messages in the future. This is a detailed description of the Diffie-Hellman protocol's operation:

Methodical Determination of Public Parameters:

Two huge prime numbers, p (a prime integer) and g (a primitive root modulo p), are agreed upon by both the client and the server. These figures don't have to be kept a secret; they can be disclosed to the public.

The process of creating private keys:

The private key a , which is a secret random number, is chosen by the client.

The private key b is chosen by the server and is likewise a secret random integer.

Compiling Public Keys:

The client uses the following algorithm to determine its public key, A :

$$A = g^a \text{ mod } p$$

The server uses the following algorithm to determine its public key, B :

$$B = g^b \text{ mod } p$$

It is safe to send both A and B via the public channel.

Public Key Exchange:

The client sends its public key A to the server.

The server sends its public key B to the client.

Method for Computing the Shared Secret:

The shared secret S is determined by the client using its own private key, a, and the public key of the server, $S = B^a \text{ mod } p$

The shared secret, S, is determined by the server using the client's public key, A, and its own private key, b: $S = A^b \text{ mod } p$

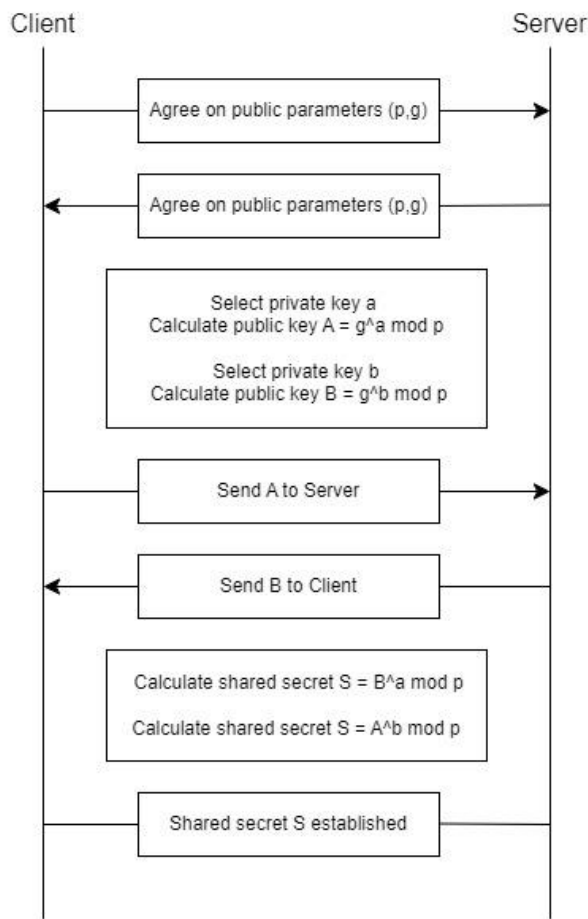
Creation of the Mutual Secret:

The client and the server arrive to the same shared secret, S.

$$S = B^a \text{ mod } p = (g^b \text{ mod } p)^a \text{ mod } p = g^{ba} \text{ mod } p$$

$$S = A^b \text{ mod } p = (g^a \text{ mod } p)^b \text{ mod } p = g^{ab} \text{ mod } p.$$

The shared secret S is the same for the client and the server since $ab=ba$.



Both the client and the server have agreed upon the public parameters (p, g).

Private Keys: Each side chooses at random and keeps them confidential.

Calculated and sent between the client and server are the public keys (A, B).

Shared Secret (S): The same shared secret that is independently determined by the client and server using their respective private keys and the public key that was received.

(f) Novelty and Security of the Developed Secure Chat Application

Novelty

Multi-Factor Authentication (MFA):

Integration: Uses Speakeasy to integrate MFA and generate time-based one-time passwords (TOTP).

Advantage: Offers an additional degree of protection, guaranteeing that unapproved access is stopped without the second authentication factor—even in the event that passwords are stolen.

End-to-End Encryption:

Utilizing AES-256 encryption, this method guarantees that messages are encrypted on the sender's device and only decrypted on the recipient's device during implementation.

Benefit: Preserves the communications' secrecy by making sure that only the designated recipients may read them.

Feedback on Password Strength:

Implementation: Makes use of the zxcvbn library to assess password strength upon registration and gives users immediate feedback.

Advantage: Promotes the creation of strong passwords among users, lowering the possibility of account breach brought on by weak passwords.

Secure Password Storage:

Implementation: Hashes passwords with salt added using bcrypt before saving them in the database for secure password storage.

Benefit: Prevents brute-force attacks and guarantees that passwords in plaintext are kept secret even in the event that the database is hacked.

Real-Time Communication:

Socket.io is implemented to enable real-time client communication.

Benefit: Offers a smooth and quick chat experience, which is necessary for interacting with users.

Rate Limiting:

Implementation: To restrict the amount of login and password reset attempts, use express-rate-limit.

Benefit: Lowers the chance of account breach and stops brute-force assaults.

Features of Security

Strong Encryption: High-level security for data in transit is ensured by AES-256

encryption for communications. Uses Transport Layer Security, or TLS, to protect client-server communication.

Secure Authentication: JWT (JSON Web Tokens) is a safe and stateless method of user authentication. MFA provides improved security while logging in.

Strong Data Protection: Bcrypt is used to salt and hash passwords, safeguarding user credentials that are kept. To make sure the application stays safe from new threats, regular security audits and upgrades are performed.

Input Validation and Sanitization: By validating and sanitizing user inputs, typical vulnerabilities like SQL injection and Cross-Site Scripting (XSS) are avoided.

Beneficiaries

Industries:

Financial Services: To safeguard sensitive customer data, banks and other financial firms must communicate securely.

Healthcare: Respects laws such as HIPAA and guarantees the privacy of patient information.

Businesses:

Secure internal communication is necessary for businesses to safeguard confidential information.

Legal Firms: To discuss important case data, attorneys and legal consultants require private lines of contact.

Educational Institutions:

Universities and schools are examples of educational institutions that offer a safe space for staff and students to interact, exchange materials, and work together on projects.

Users Individually:

Users that are concerned about privacy and security in their personal communications are known as privacy-conscious users.

The created Secure Chat Application offers a strong and intuitive communication platform by fusing cutting-edge security features with creative applications. Through the integration of multi-factor authentication, end-to-end encryption, instantaneous password strength feedback, and safe password storage, the program effectively tackles a range of security issues and guarantees optimal protection for its users.

(g) Resilience Against Common Security Threats

1. Injection of SQL

One code injection method that has the potential to wipe out your database is SQL Injection. It is among the most often used methods of online hacking. This attack involves inserting malicious code via web page input into SQL queries.

Measures of Resilience:

a. Utilizing Mongoose ORM:

Because MongoDB and Mongoose ORM don't utilize SQL for database queries, the application by nature prevents SQL injection. Since Mongoose methods are intended to be resistant to SQL injection, they are used for all database operations.

b. Queries with parameters:

We make sure that any dynamic input in our queries is appropriately parameterized, even if we are utilizing MongoDB. As an instance, while doing a database query:

```
User.findOne({ email: req.body.email })
```

c. Prior to processing, ensure that all user inputs have been validated and sanitized. using libraries to sanitize inputs and stop injection attacks, like validator.

2. Cross-Site Scripting (XSS)

Attackers can exploit an online application to deliver malicious code, usually in the form of a browser side script, to a separate end user. This is known as a cross-site scripting (XSS) attack.

Measures of Resilience:

a. Escaping Output: In order to stop dangerous scripts from running, all user inputs must be escaped before being rendered in the user interface.

Using React's built-in defense against XSS attacks, which involves escaping all data before rendering by default.

b. Policy for Content Security (CSP):

putting in place a content security policy to limit the sources that allow scripts to run.

Example of a CSP header:

```
app.use((req, res, next) => {  
  res.setHeader("Content-Security-Policy", "default-src 'self';  
  script-src 'self'");  
  next();  
});
```

c. HTTP Headers: To defend against frequent online vulnerabilities, use security headers like X-XSS-Protection, X-Content-Type-Options, and X-Frame-Options. Ex:

```
app.use(helmet());
```

d. Input Validation and Sanitization: To sanitize HTML input and stop XSS attacks, use tools like DOMPurify. Example:

```
const DOMPurify = require('dompurify');
const cleanInput = DOMPurify.sanitize(req.body.userInput);
```

An overview of security risks and counter measures

Security Threat	Resilience Measures
SQL Injection	- Use of Mongoose ORM - Parameterized Queries - Input Validation and Sanitization
XSS (Cross-Site Scripting)	- Escaping Output - Content Security Policy (CSP) - Security HTTP Headers - Input Validation and Sanitization

Implementation Specifics

Using Mongoose ORM: Removes the danger of SQL injection by ensuring secure database interactions without the need for direct SQL queries.

Escape Product: To prevent XSS, React automatically escapes any values included in JSX before rendering.

Policy for Content Security: An illustration of how to use CSP in Express.js.

```
app.use((req, res, next) => {
  res.setHeader("Content-Security-Policy", "default-src 'self'; script-src 'self'");
  next();
});
```

Helmet-equipped HTTP Headers: Configuring different HTTP headers for security using Helmet middleware:

```
const helmet = require('helmet');
app.use(helmet());
```

Validation and Sanitization of Input: An illustration of how to validate emails using a validator

```
const validator = require('validator');
if (!validator.isEmail(req.body.email)) {
  return res.status(400).send('Invalid email');
}
```

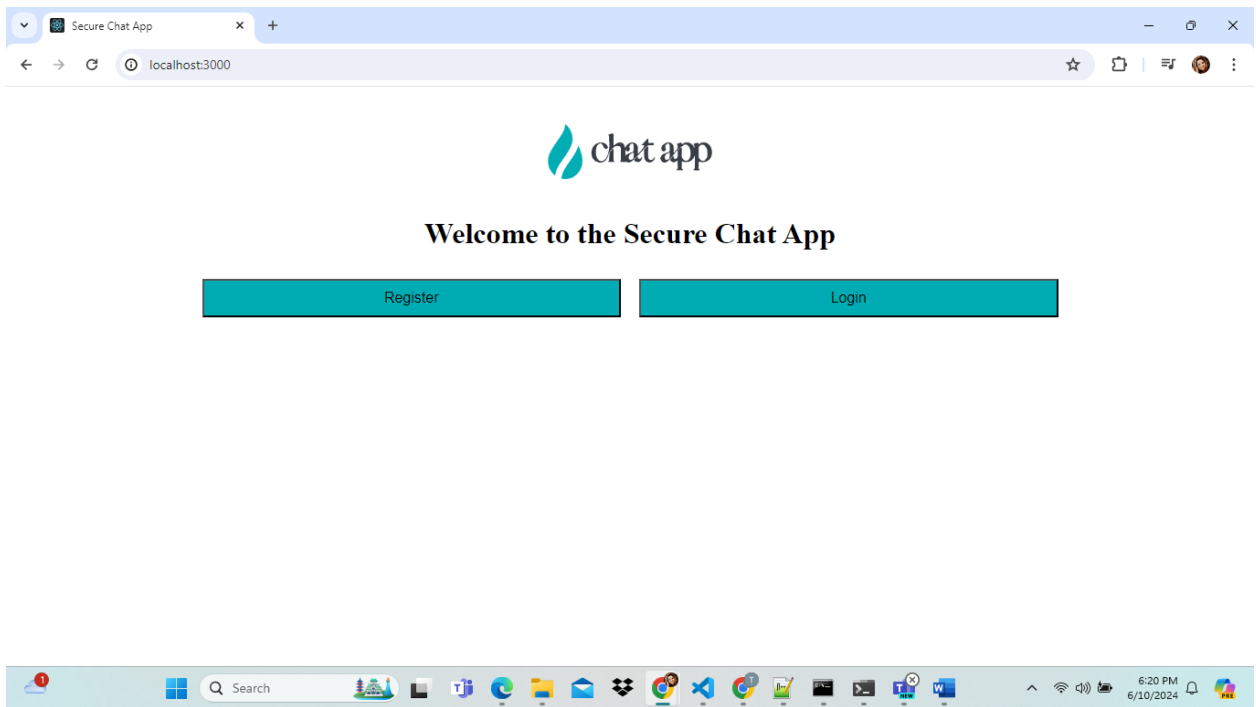
An instance of cleaning user inputs with DOMPurify is as follows:

```
const DOMPurify = require('dompurify');  
const cleanInput = DOMPurify.sanitize(req.body.userInput);
```

By using a safe ORM (Mongoose), proper input validation and sanitization, escape outputs, security header implementation, and content security policies, the developed Secure Chat Application is resistant against common security threats like SQL injection and cross-site scripting (XSS).

Screenshots of the Website,

Welcome Page:



Register Page:

chat app

Register

Sequences like abc or 6543 are easy to guess

- Add another word or two. Uncommon words are better.
- Avoid sequences

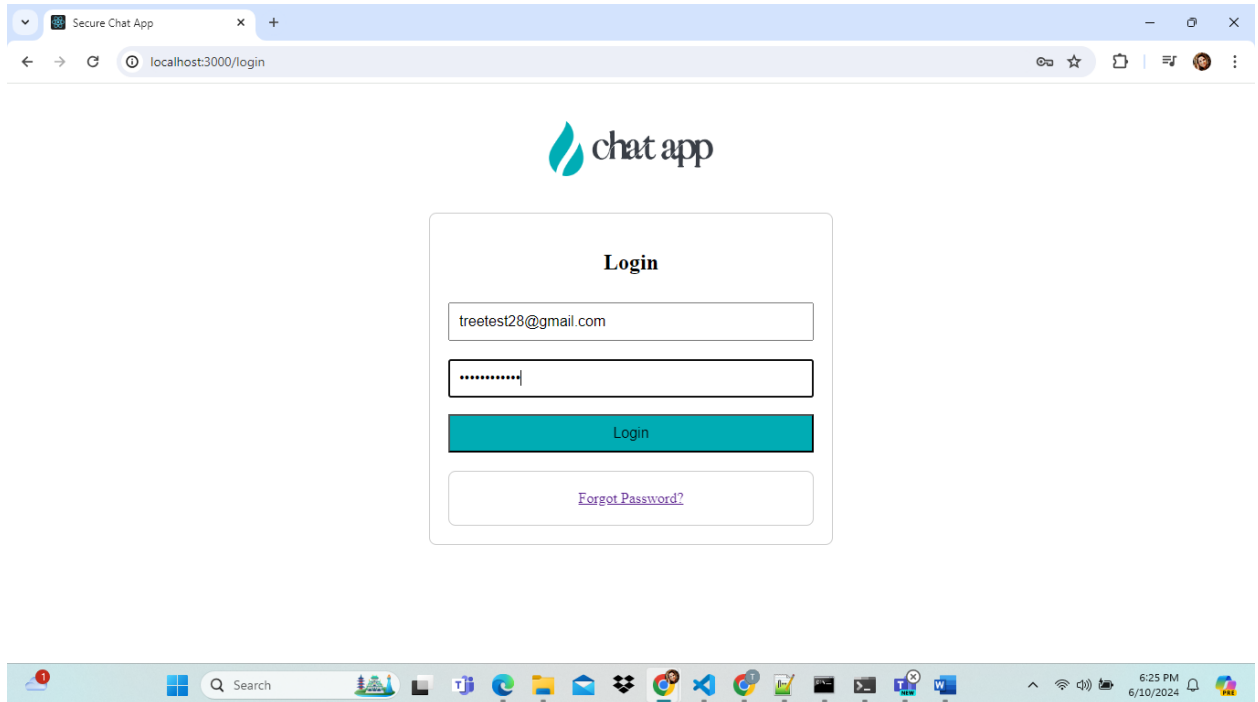
Strong Password Mechanism

chat app

Register

Already have an account? [Login](#)

Login Page:

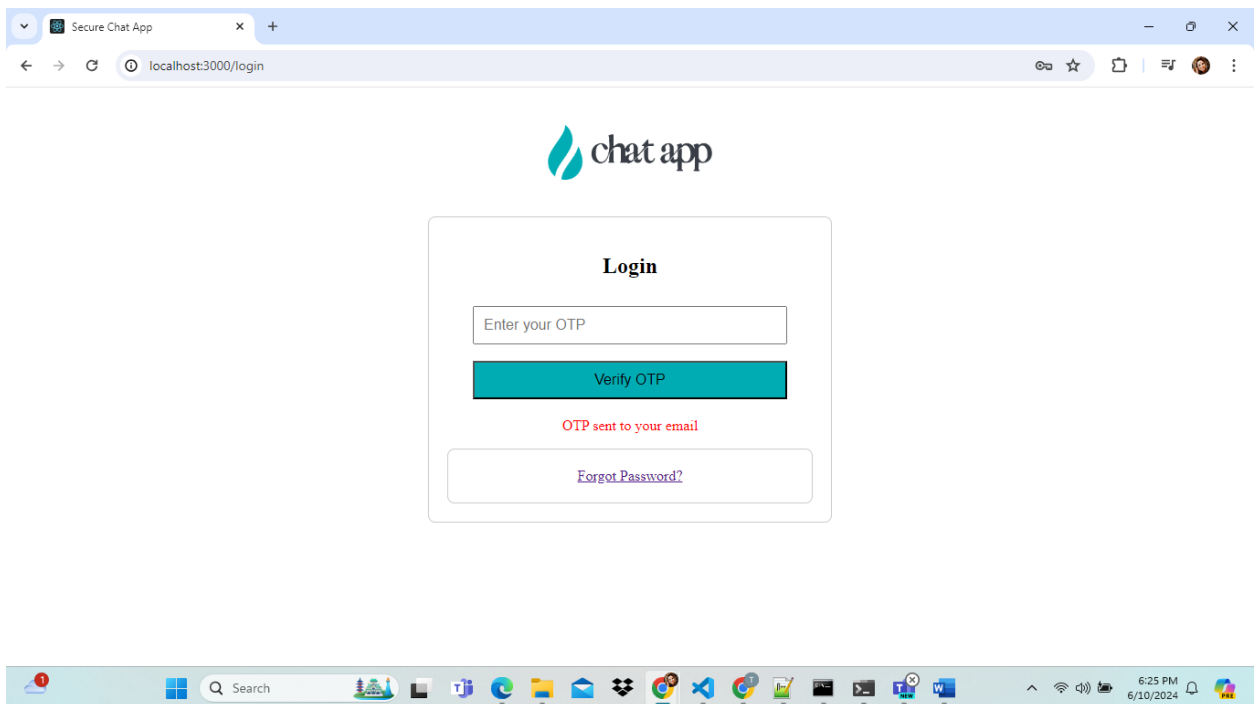


The screenshot shows a web browser window with the title "Secure Chat App" and the address bar displaying "localhost:3000/login". The page features the "chat app" logo at the top. Below the logo is a "Login" form with the following elements:

- A text input field containing the email address "treetest28@gmail.com".
- A password input field with masked characters ".....".
- A teal "Login" button.
- A link labeled "Forgot Password?" below the login button.

The Windows taskbar at the bottom shows the time as 6:25 PM on 6/10/2024.

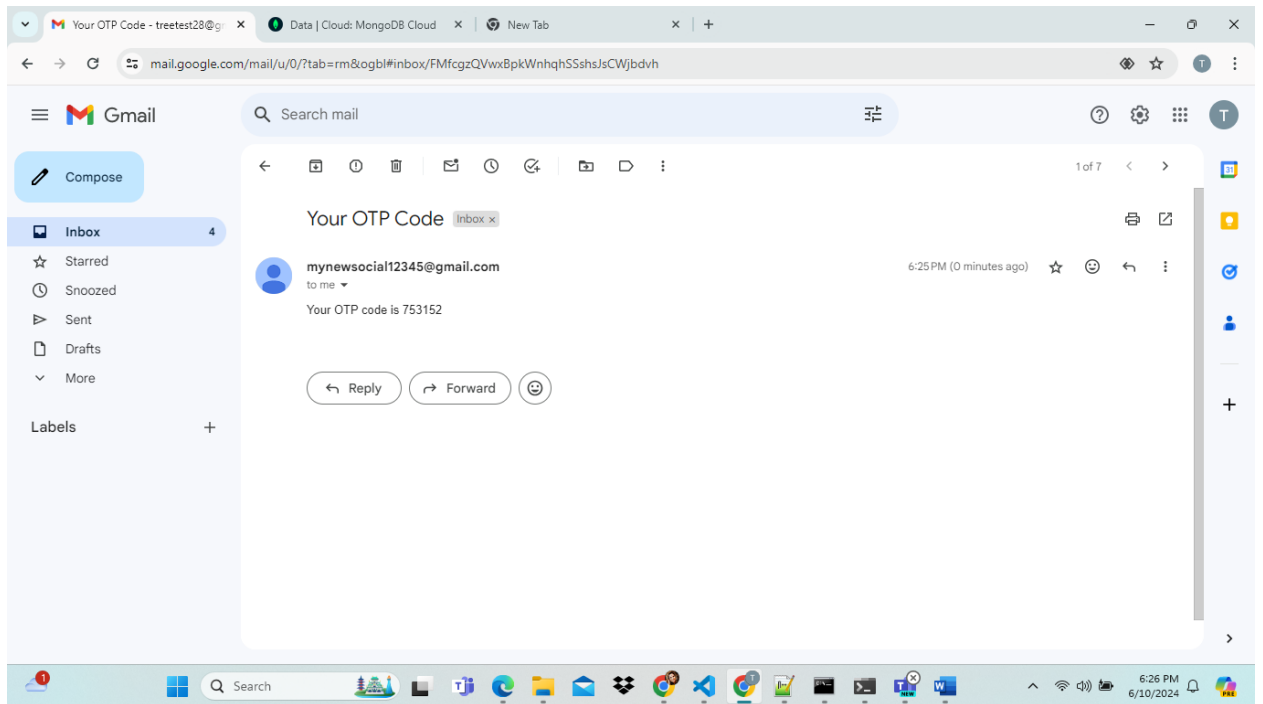
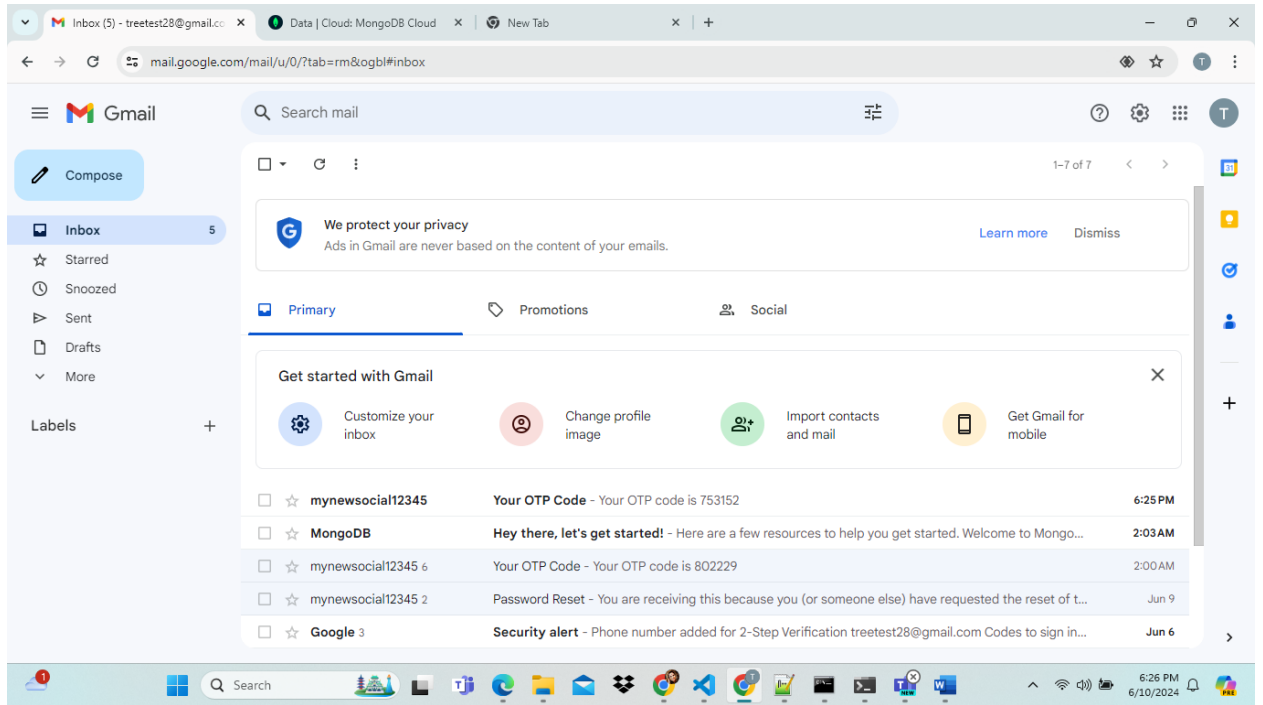
MFA:



The screenshot shows the same web browser window as the login page, but the form is for Multi-Factor Authentication (MFA). The "chat app" logo is at the top. The "MFA" form contains the following elements:

- A text input field with the placeholder text "Enter your OTP".
- A teal "Verify OTP" button.
- A red message "OTP sent to your email" displayed below the button.
- A link labeled "Forgot Password?" at the bottom of the form.

The Windows taskbar at the bottom shows the time as 6:25 PM on 6/10/2024.





Login

Verify OTP

OTP sent to your email

[Forgot Password?](#)



Forgot Password:



Password Recovery

Recover Password



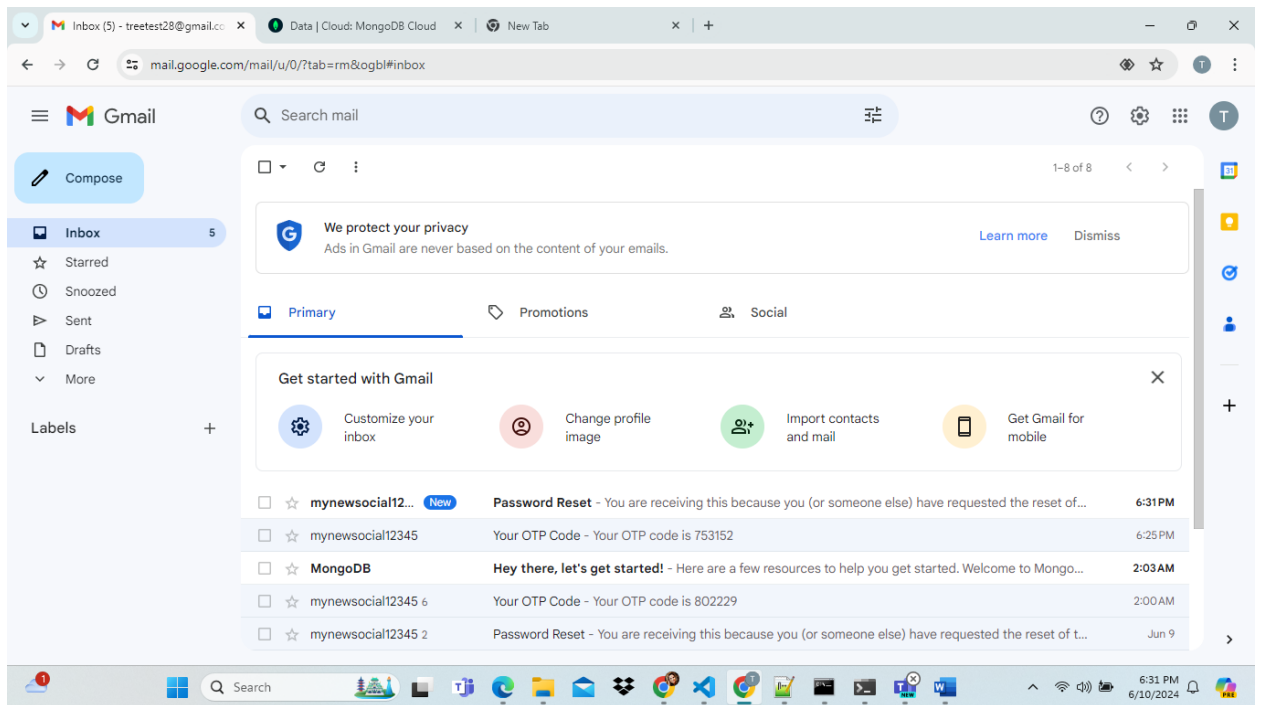


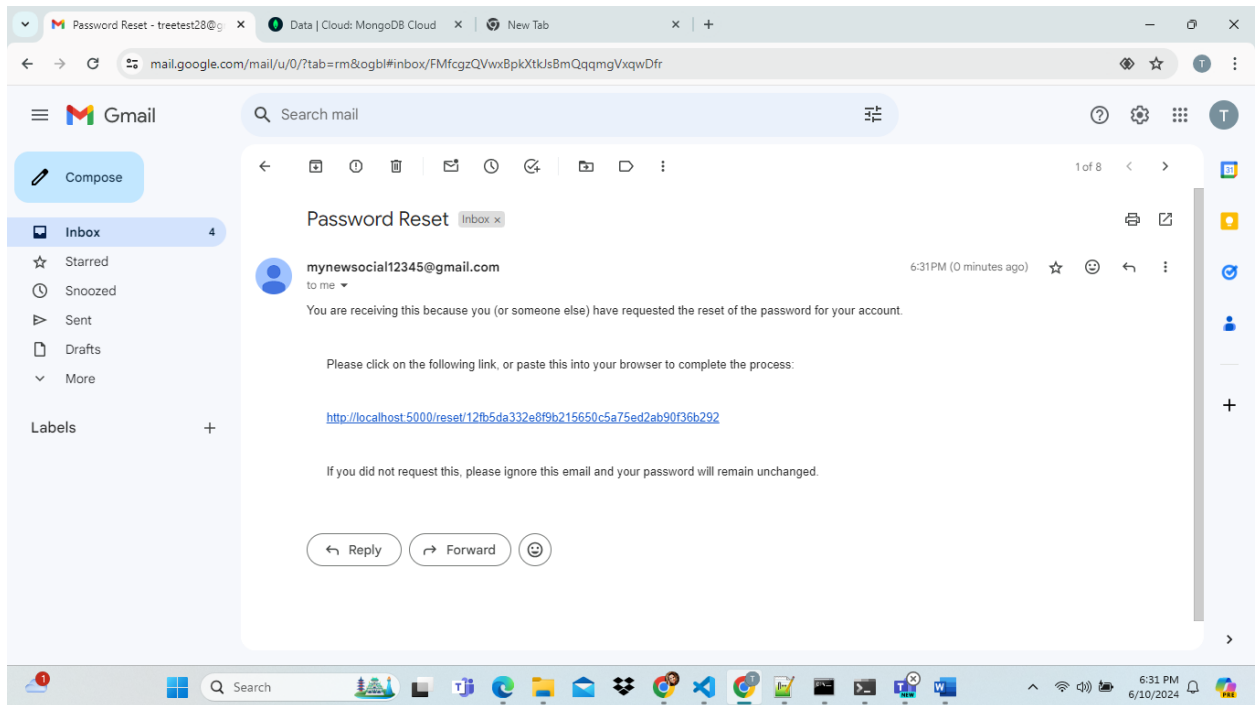
Password Recovery

treetest28@gmail.com

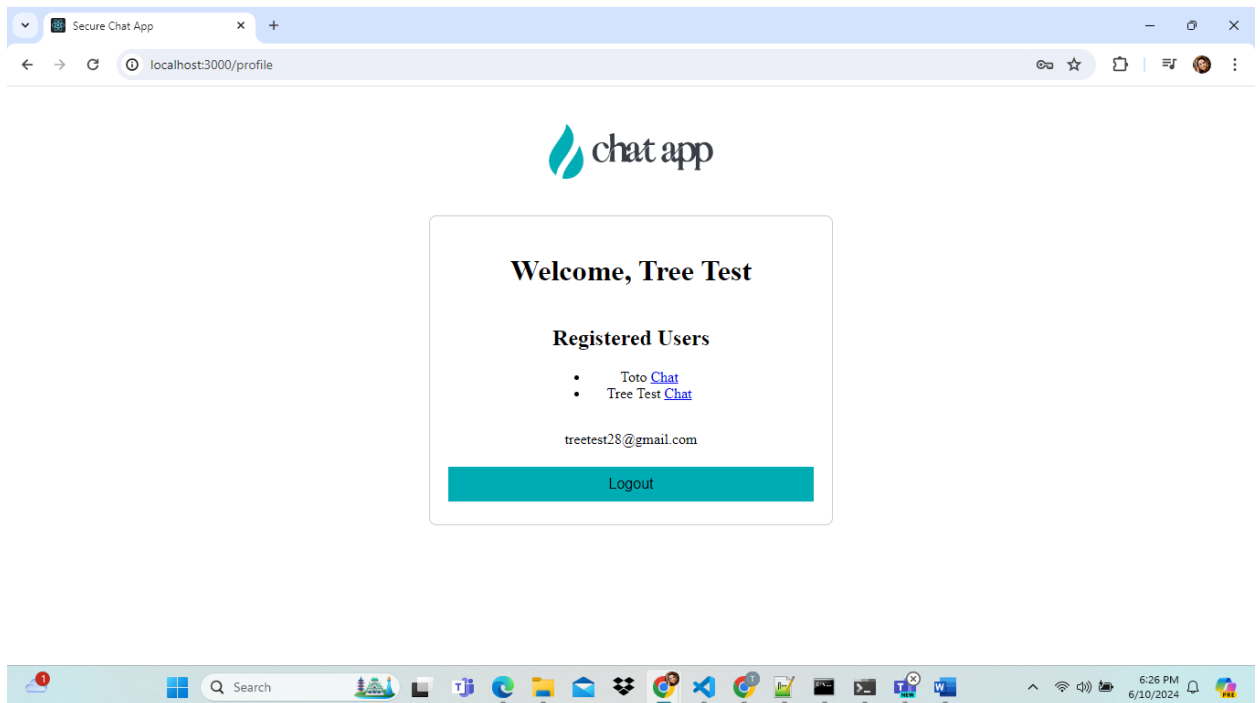
Recover Password

Password reset email sent



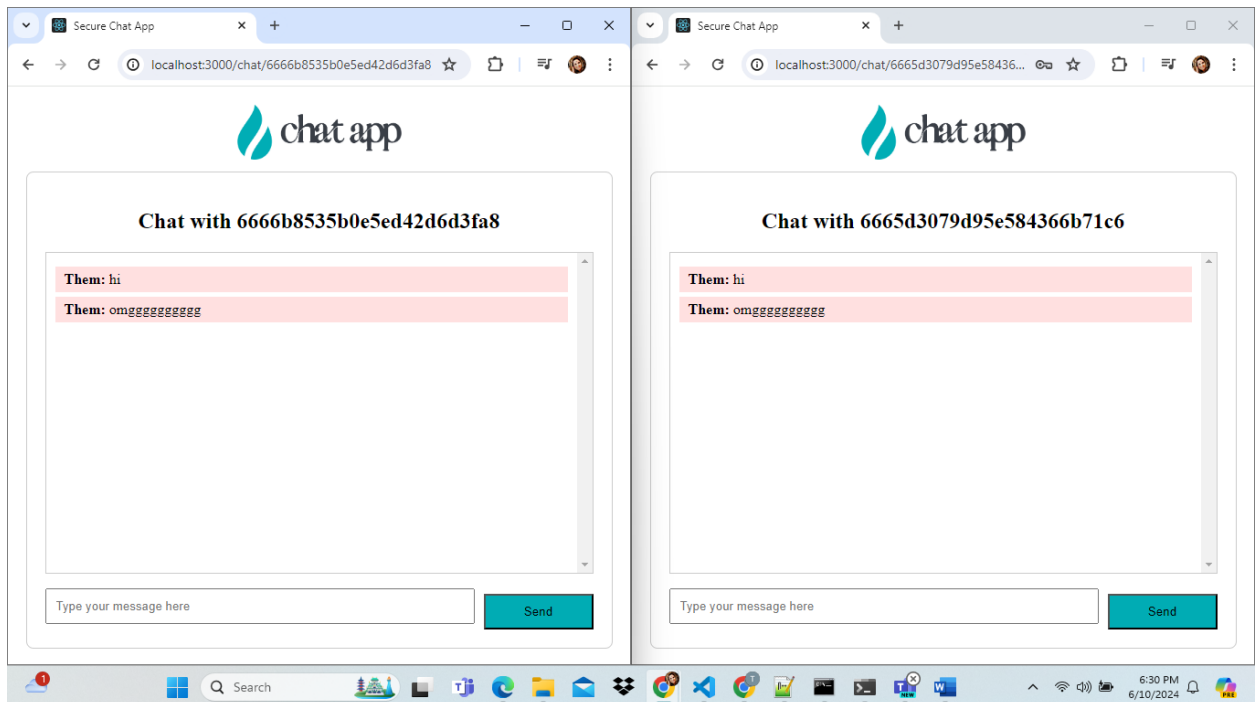
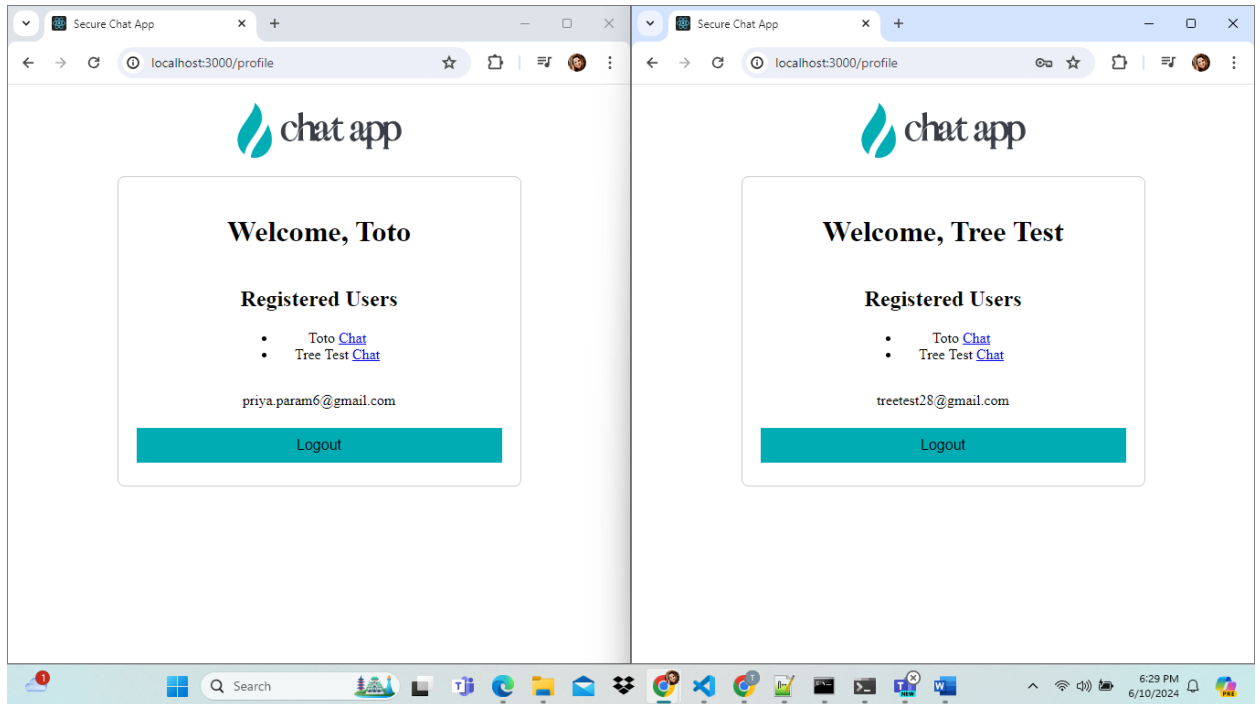


Profile Page:



Chat Page:

Logged in in two different Accounts.



Database (MongoDB)

The screenshot shows the MongoDB Atlas web interface. The browser address bar displays the URL: `cloud.mongodb.com/v2/6665d22daea04347265675f6#/metrics/replicaSet/6665d29e8bb82d7c36389596/explorer/test/users/find`. The interface includes a sidebar with navigation options like Overview, Deployment, Database, Data Lake, Services, and Security. The main content area is titled 'test.users' and shows database metrics: STORAGE SIZE: 36KB, LOGICAL DATA SIZE: 347B, TOTAL DOCUMENTS: 2, INDEXES TOTAL SIZE: 72KB. A search bar is present with the text 'Type a query: { field: 'value' }'. Below the search bar, a JSON document is displayed:

```
{
  "_id": ObjectId('6666b853b0e5ed42d6d3fa8'),
  "username": "Tree Test",
  "email": "treetest28@gmail.com",
  "password": "$2a$10$7jm2s/7ChbWbDAU80EL030jLMhttKC1PsVAFaILV6CW/yr66u5JF0",
  "mfaEnabled": false
}
```

The interface also features buttons for 'Visualize Your Data', 'Refresh', 'Insert Document', and 'Generate queries from natural language in Compass'.

The screenshot shows the MongoDB Atlas web interface. The browser address bar displays the URL: `cloud.mongodb.com/v2/6665d22daea04347265675f6#/metrics/replicaSet/6665d29e8bb82d7c36389596/explorer/test/chats/find`. The interface includes a sidebar with navigation options like Overview, Deployment, Database, Data Lake, Services, and Security. The main content area is titled 'test.chats' and shows database metrics: STORAGE SIZE: 20KB, LOGICAL DATA SIZE: 240B, TOTAL DOCUMENTS: 2, INDEXES TOTAL SIZE: 20KB. A search bar is present with the text 'Type a query: { field: 'value' }'. Below the search bar, a JSON document is displayed:

```
{
  "_id": ObjectId('6666b9915b0e5ed42d6d3fc4'),
  "text": "hi",
  "sender": ObjectId('6665d3079d95e584366b71c6'),
  "recipients": Array (1),
  "timestamp": 2024-06-10T08:30:09.878+00:00
}
```

The interface also features buttons for 'Visualize Your Data', 'Refresh', 'Insert Document', and 'Generate queries from natural language in Compass'.

Summary

The goal of this project is to create a powerful, secure chat application with end-to-end encryption, real-time messaging, user authentication, and secure login. The application leverages contemporary web technologies to provide a smooth user experience while ensuring excellent security through a number of strategies.

Key Features

User Registration and Login:

Password strength checking ensures secure user registration and login.
MFA, or multi-factor authentication, provides increased security.
feature for recovering passwords.

Real-Time Messaging:

Secure end-to-end encrypted communication between users in real-time.
Web sockets for real-time communication.

Security Procedures:

employing robust encryption techniques to protect messages.
User credentials are securely stored using bcrypt hashing.
defense against cross-site scripting (XSS) and SQL injection threats.

Technologies Used

a) Client (React):

Welcome Page: The application's welcome page offers login and registration options.
Register Page: Password strength checking is included in the user registration form.
Login Page: User login form with optional OTP input for MFA is located on the login page.
Profile Page: The user's profile page shows their details and offers a way to log out.
Chat Page: A user-friendly, real-time communications interface.
Enable MFA Page: Multi-factor authentication verification and activating interface.
Password Recovery Page: Form for initiating password recovery.
Password Reset Form: This form allows you to reset your password with a token.

Technologies:

React: Use React to create user interfaces.
Axios: For contacting the server via HTTP requests.
Socket.io-client: For instantaneous correspondence.
zxcvbn: To verify the strength of a password.
Use DOMPurify to clean HTML input.

b) Server (Node.js and Express.js):

Verification Routes: Manages password recovery, MFA, login, and user registration.
Chat Routes: Oversees the exchange of chat messages.

Middleware: Consists of rate limitation, input validation, and JWT authentication.

Socket.io Server: To manage client-to-client real-time communication.

Technology

Runtime environment for server-side code execution is Node.js.

Express.js is the web framework used to construct the server.

Socket.io: For two-way communication in real time.

For hashing passwords, use Bcrypt.

Speakeasy: For creating and confirming MFA TOTP codes.

Nodemailer: For emailing password recovery and OTP requests.

Helmet: To secure the application, specify different HTTP headers.

Express-rate-limit: Used to limit rates and thwart brute-force assaults.

c) Database (MongoDB)

User Model: Hashed passwords and MFA information are stored in a schema for users.

Chat Model: A secure chat message storage schema.

Technology

MongoDB: NoSQL database used to hold message and user data.

Mongoose: An ODM for server-side interaction with MongoDB.

Architecture

Client:

React-built mobile applications or web browsers are used by users to engage with the application.

transmits and receives authentication requests and encrypted communications to and from the server.

Server:

Manages message archiving, user authentication, and client request processing.

uses Socket.io for real-time messaging and JWT for authentication to ensure safe connection.

communicates with outside services (Nodemailer) to send emails for password recovery and OTP.

Database:

Holds chat messages and user data (password hashed, email address, and username).

uses MongoDB to store data in a scalable and adaptable manner.

External Services:

Nodemailer for email password recovery and OTP transmissions.

Using advanced web technology, the Secure Chat Application offers a safe and convenient real-time communications environment. The confidentiality and integrity of user data are guaranteed by the integration of strong security features including multi-factor authentication, end-to-end encryption, and safe password storage. Both individual users who value privacy in their conversations and sectors and corporations that need secure communication routes would benefit from this application.

