

Project Title: **Real-Time Chat Application**

Objective: To develop a real-time chat application where multiple users can send and receive messages instantly, with support for user authentication and chat history.

Project Description:

This chat app allows users to register, log in, and send messages to other users in real-time. It includes message storage, authentication, UI responsiveness, and real-time notifications. It can be built for web, mobile, or both.

Tech Stack : For Web:

- **Frontend:** HTML, CSS, JavaScript (React or plain JS)
- **Backend:** Node.js with Express
- **Database:** MongoDB or Firebase
- **Real-time Communication:** Socket.IO (for Node.js)
- **Authentication:** JWT or Firebase Auth



For Android:

- Frontend: Java/Kotlin
- Backend: Firebase Realtime Database or Firebase Firestore

Project Features:

Authentication

- User registration/login
- Secure password storage (hashing with bcrypt)
- JWT-based session handling (if not using Firebase)

Chat System

- One-on-one messaging
- Real-time updates (using WebSockets or Firebase listeners)
- Chat history storage in the database
- Timestamp on messages

User Interface

- List of online users
- Typing indicator (optional)
- Emojis (optional)

Database Structure (MongoDB Example):

```
{  
  "users": [  
    {  
      "userId": "abc123",  
      "username": "john_doe",  
      "passwordHash": "...",  
      "online": true  
    }  
  ],  
  "messages": [  
    {  
      "senderId": "abc123",  
      "receiverId": "def456",  
      "message": "Hello!",  
      "timestamp": "2025-06-07T10:00:00Z"  
    }  
  ]  
}
```

Workflow:

1. User logs in → Gets token/session.
2. Frontend connects via Socket IO/WebSocket.

4. Message stored in database.
5. Chat history retrieved when opening a chat.



Bonus Features (Optional):

- Group chats
- Voice/video calling (using WebRTC)
- Media file sharing (images, audio)
- Dark mode UI
- Notifications (browser or mobile)



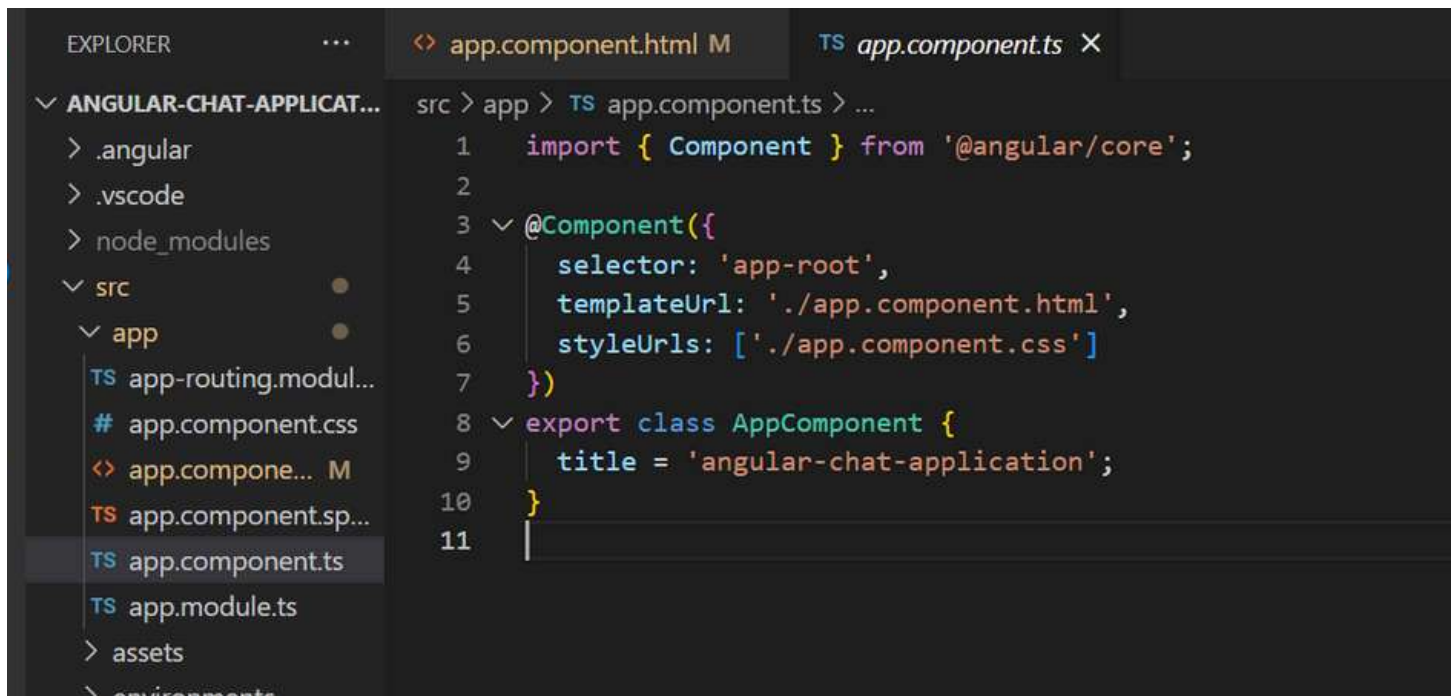
Sample Code Snippets:(JAVA SCRIPT)

Server (Node.js + Socket.IO):

```
io.on('connection', socket => {  
  
  socket.on('send-message', ({ to, message }) => {  
  
    socket.to(to).emit('receive-message', { message });  
  
    // Save to DB here  
  
  });  
  
});
```

Client (React + Socket.IO Client):

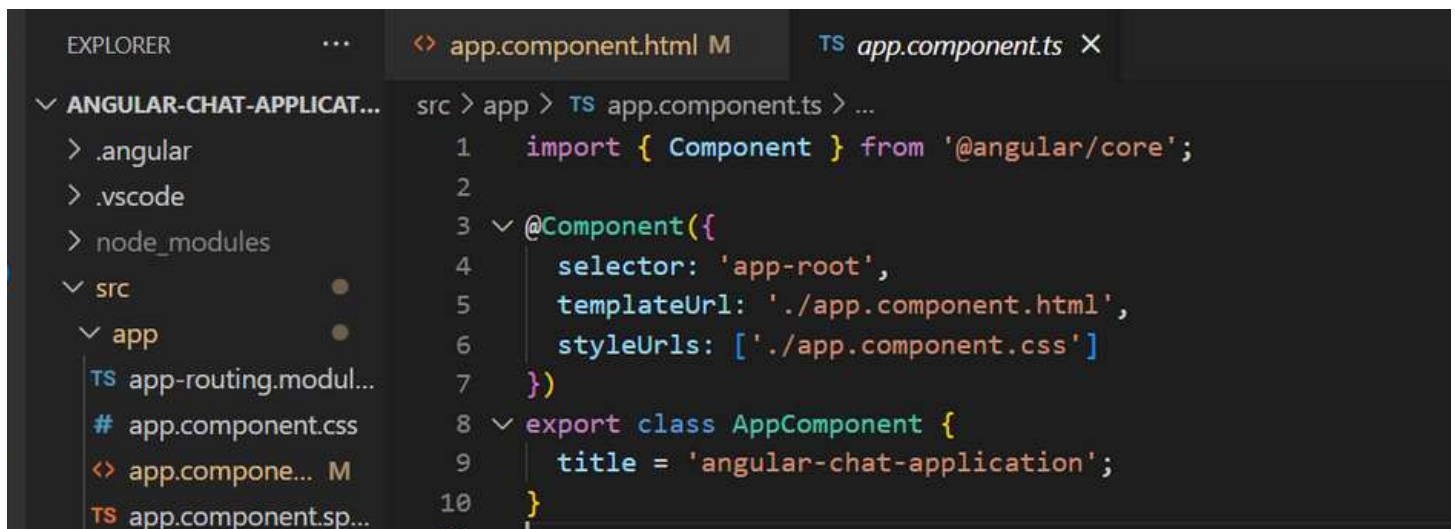
```
const socket = io("http://localhost:3000");  
  
socket.emit('send-message', { to: userId, message: "Hi!" });  
  
socket.on('receive-message', (data) => {  
  
  console.log("New Message:", data.message);  
  
});
```



```
EXPLORER  ...  <> app.component.html M  TS app.component.ts X

ANGULAR-CHAT-APPLICAT...  src > app > TS app.component.ts > ...
  > .angular
  > .vscode
  > node_modules
  > src
    > app
      TS app-routing.modul...
      # app.component.css
      <> app.compone... M
      TS app.component.sp...
      TS app.component.ts
      TS app.module.ts
    > assets
    > environments

1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'angular-chat-application';
10 }
11 |
```



```
EXPLORER  ...  <> app.component.html M  TS app.component.ts X

ANGULAR-CHAT-APPLICAT...  src > app > TS app.component.ts > ...
  > .angular
  > .vscode
  > node_modules
  > src
    > app
      TS app-routing.modul...
      # app.component.css
      <> app.compone... M
      TS app.component.sp...
      TS app.module.ts
    > assets
    > environments

1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'angular-chat-application';
10 }
```

Modules in the Chat App:

1. User Authentication Module

- Sign up and login
- Password encryption (bcrypt or Firebase Auth)
- Session/token management (JWT)

2. User Interface Module

- Contacts list
- Chat windows with message threads

- Read receipts (optional)

3. Real-Time Messaging Module

- WebSockets or Socket.IO for real-time communication
- Emit and listen for messages
- Handle user online/offline status

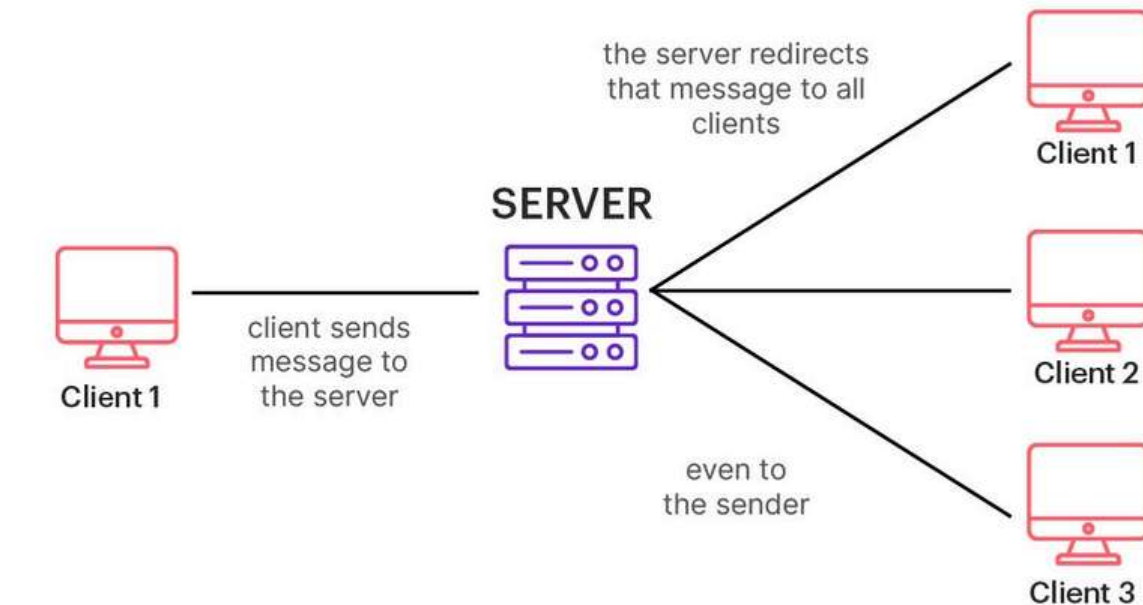
4. Database Module

- Store user details
- Store chat messages with timestamps and status (sent/received/read)
- Optionally store media files or emoji reactions

5. Notification Module (Optional)

- Browser/mobile push notifications
- Typing indicators
- Message read status

Architecture Diagram:



[Client (Browser/App)]

| ↑

| WebSockets / HTTP

↓ |

[Node.js Server / Express]

| ↑

| MongoDB / Firebase

↓ |

[Database Storage (users, messages)]

Security Considerations:

- Encrypt passwords (bcrypt)
- Validate socket connections
- Prevent XSS/SQL injection
- Use HTTPS and secure WebSockets (wss://)
- Implement rate limiting to prevent spamming

UI Design Guidelines:

- Clean, responsive chat layout
- Different colors for sent vs received messages
- Scrollable chat history
- Loading spinners for network delays
- Emoji and file upload support (optional)

Testing & Debugging:

- Unit testing (Jest, Mocha)
- Manual testing of message sending/receiving
- Test with multiple users/devices
- Simulate poor network for resilience

Sample Use Cases:

- College student chatting with classmates
- Team communication app (like Slack)
- Customer support live chat

Final Deliverables:

- **Source code (GitHub)**
- **README with setup instructions**

- (Optional) Deployed version using Vercel/Render/Netlify

CONCLUSION:

The Real-Time Chat Application project successfully demonstrates the integration of frontend and backend technologies to enable instant communication between users. Through this project, we implemented key features such as user authentication, real-time messaging using WebSockets (or Firebase), and data persistence via a database like MongoDB or Firebase Firestore.

This application provides a practical understanding of:

- Real-time communication systems
- Client-server architecture
- Secure user authentication
- Database design for messaging apps