ShadowChat Project Report

1. Introduction

ShadowChat is a real-time messaging application designed for secure and seamless communication. It provides users with an intuitive platform to exchange messages instantly while ensuring privacy and security. The project aims to create a user-friendly chat experience leveraging modern web technologies.

2. Features and Detailed Implementation

2.1 Real-time Messaging using WebSockets

How WebSockets Work:

WebSockets provide full-duplex communication channels over a single TCP connection. Unlike traditional HTTP requests, which require repeated polling to fetch updates, WebSockets maintain a persistent connection between the client and the server. This enables real-time message delivery with minimal latency.

Implementation in ShadowChat:

- **Backend:** Socket.io is integrated with the Node.js server to manage real-time communication.
- **Frontend:** The client listens for incoming messages and updates the chat UI dynamically.
- **Steps:**
    1. The server initializes a WebSocket connection using Socket.io.
    2. When a user sends a message, it is emitted to the recipient's socket ID.
    3. The frontend updates the chat window without requiring a page refresh.

Code snippet from server.js:

```
io.on('connection', (socket) => {
  console.log('A user connected');
```

```
  socket.on('sendMessage', (message) => {
    io.emit('receiveMessage', message);
  });
});
```

2.2 User-Friendly Interface

Importance of a Good UI:

A well-designed interface improves user engagement and usability. ShadowChat uses React.js with Tailwind CSS and DaisyUI for a clean and responsive design.

Implementation in ShadowChat:

- **React Components:** Modular components handle chat windows, message input, and notifications.
- **CSS Optimization:** Tailwind CSS ensures lightweight styling with faster loading times.
- **Theme Customization:** DaisyUI is used for theming and UI consistency.

2.3 State Management with Zustand

Why Zustand?

Zustand is a lightweight state management library that simplifies global state handling without the boilerplate of Redux.

Implementation in ShadowChat:

- **Chat State Management:** Zustand stores active conversations and messages.
- **User Authentication:** Zustand maintains user session data across the application.

Code snippet:

```
import create from 'zustand';

const useChatStore = create((set) => ({
  messages: [],
```

addMessage: (message) => set((state) => ({ messages: [...state.messages, message] })),

}));

2.4 Online Status

How Online Status Works:

Tracking online status requires monitoring user activity.

Implementation in ShadowChat:

- **Socket.io tracks when a user connects or disconnects.**
- **The user's status is updated in the database and reflected in the UI.**

2.5 Notifications

Importance of Notifications:

Real-time alerts ensure users don't miss messages.

Implementation in ShadowChat:

- **Visual Alerts:** Browser-based notifications enhance engagement.

3. Tech Stack and Justification

3.1 Frontend: React.js, Tailwind CSS & DaisyUI

- **React.js** was chosen for its component-based architecture, making UI development modular and scalable.
- **Tailwind CSS** provides a utility-first approach, enabling faster styling with a responsive design.
- **DaisyUI** enhances UI development with prebuilt themes and components.

3.2 Backend: Node.js & Express.js

- **Node.js** is efficient for handling multiple concurrent connections, making it ideal for real-time applications.
- **Express.js** simplifies backend development by providing robust routing and middleware support.

3.3 Database: MongoDB

- **MongoDB** is a NoSQL database that offers flexibility and scalability, making it perfect for storing chat logs dynamically.

3.4 State Management: Zustand

- **Zustand** is used for global state management, reducing complexity and improving performance.

3.5 Real-time Communication: Socket.io

- **Socket.io** enables bi-directional real-time communication, crucial for instant messaging applications.

3.6 Deployment: Render

- **Render** offers a hassle-free deployment experience with automatic scaling, making it suitable for hosting the application.

4. Installation & Setup

4.1 Prerequisites

- Node.js (v16 or later)
- MongoDB (local or cloud instance)

4.2 Steps

1. Clone the repository:
2. git clone https://github.com/Luv-valecha/ShadowChat.git
3. cd ShadowChat
4. Install dependencies:
5. npm install
6. Configure the .env file with necessary credentials.
7. Start the backend server:
8. npm run server
9. Start the frontend:
10. cd client
11. npm start

5. Deployment

ShadowChat is deployed on Render and can be accessed via this link. The hosting solution provides seamless scaling and reliability for real-time communication.

6. Conclusion

ShadowChat is a fully functional real-time chat application that emphasizes security and ease of use. Future improvements could include:

- Integration of voice and video calls
- Multi-device synchronization
- AI-powered message summarization

With its modern tech stack and robust architecture, ShadowChat aims to provide a reliable and secure communication platform for users worldwide.