**Project Name**: ChatWave – Real-Time Web Chat Application

**Course**: CSE327 - Software Engineering

**Instructor**: Professor Dr. Md. Sazzad Hossain

**Submitted By:**

Name : Md.Minhajul Islam

ID :2211022042

Section:09

Group :10

## Executive Summary

Chat-Wave is a modern, real-time chat application that allows fast, secure, and responsive communication. Built on the MERN stack (MongoDB, Express.js, React.js, Node.js) and enhanced with Socket.io for real-time messaging, the app supports private chats, user presence indicators, media sharing(image), and is optimized for mobile and desktop use. The project successfully demonstrates scalable, real-world communication software, suitable for educational, corporate, and customer support use cases.

Md. Minhajul Islam led the development, completing over 85% of the work including system analysis, full-stack development, real-time integration, testing, and deployment.

---

## Table of Contents

---

## List of Figures

**List of Tables**

---

# 1. Introduction

## 1.1 Purpose and Scope

Chat-Wave aims to provide a robust, real-time chat solution that is device-agnostic and secure. It caters to individual and group communications in educational, personal, and corporate settings.

## 1.2 Product Overview

Capabilities include user authentication, real-time messaging, , media file sharing. Built for reliability and scalability.

1. One-to-one messaging
2. Secure authentication (JWT-based)
3. Real-time updates via Socket.io
4. Media sharing (images)
5. Responsive UI with TailwindCSS and DaisyUI

## 1.3 Structure of the Document

The document details project planning, system requirements, architecture, design, testing strategies, and closing acknowledgments.

## 1.4 Terms, Acronyms, and Abbreviations

- **MERN**: MongoDB, Express.js, React.js, Node.js
- **JWT**: JSON Web Token

- **Socket.io**: Real-time communication library
- **UI**: User Interface
- **ER Diagram**: Entity-Relationship Diagram
- **CI/CD**: Continuous Integration/Deployment

---

## 2. Project Management Plan

### 2.1 Project Organization

| **Name** | **ID** | **Role** |
|---|---|---|
| Md. Minhajul Islam | 2211022042 | Team Leader, Full-stack Developer |
| Md. Jalal Abedin | 2211158642 | Developer |
| Tabassum Tasnim | 2211451042 | Developer |
| Shuvra Saha Rimjim | 2222483042 | |

### 2.2 Lifecycle Model Used

Agile Iterative model with weekly deliveries.

### 2.3 Risk Analysis

| **Risk** | **Impact** | **Mitigation** |
|---|---|---|
| Server crashes | High | Using cloud hosting (Render,Vercel) with automated backups |
| Data inconsistency | Medium | Enforcing strong schema validations with Mongoose |
| Authentication breaches | High | Implementing JWT token-based authentication and Bcrypt password hashing |

### 2.4 Hardware and Software Resource Requirements

| **Resource** | **Purpose** |
|---|---|
| Laptop/Desktop (8GB+ RAM) | Project development, coding, testing, and deployment |
| Visual Studio Code | Source code editor for frontend and backend development |
| Node.js and npm | Backend server development and dependency management |
| React.js | Frontend user interface development |

| Resource | Purpose |
|---|---|
| MongoDB Atlas | Cloud-hosted NoSQL database for persistent storage |
| Cloudinary | Cloud storage solution for user-uploaded media and files |
| Vercel (Frontend) | Hosting the client-side React application |
| Render (Backend) | Hosting the server-side Node.js/Express application |
| Git and GitHub | Version control, collaboration, and source code management |
| Postman | API testing and backend endpoint verification |

**Table 1: Hardware and Software Resources**

## 2.5 Deliverables and Schedule

| Stage | Task Description | Timeline |
|---|---|---|
| Planning | Requirement analysis, project planning, preparation of Software Requirement Specification (SRS) | Week 1 |
| Design | Creation of wireframes, UI/UX design using Figma concepts, selection of color theme and component libraries (TailwindCSS, DaisyUI) | Week 2 |
| Development | Backend API development with Node.js/Express.js, Frontend development with React.js, Database modeling using MongoDB Atlas, Integration of Cloudinary for file uploads, and Socket.io for real-time communication | Weeks 3–7 |
| Testing | Functional testing, bug fixing, performance optimization, UI responsiveness checks, security checks (basic penetration tests) | Week 8 |
| Deployment | Final deployment of frontend (Vercel) and backend (Render), final documentation, preparation for project submission | Week 9 |

**Table 2: Project Timeline**

## 2.6 Monitoring, Reporting, and Controlling Mechanisms

Throughout the development of ChatWave, the following mechanisms were used to ensure project monitoring, reporting, and controlling:

- **GitHub Repository Management:**
  The complete codebase was managed through a GitHub repository, with proper branching strategies for version control.
- **GitHub Project Boards:**
  Task assignments and progress tracking were maintained using GitHub Project Boards to visualize the development stages.

- **Individual and Team Contributions:**
  While the majority of the development work — including backend APIs, frontend UI, real-time integration, and deployment — was completed by the lead developer (Md. Minhajul Islam), teammates contributed by reviewing features, testing functionalities, and suggesting improvements.
- **Weekly Status Updates:**
  Md. Minhajul Islam regularly shared project updates and progress reports with the team members to keep everyone informed.
- **Milestone-Based Progress Tracking:**
  Key deliverables were aligned with weekly milestones, and regular self-assessments ensured project schedule adherence.
- **Issue and Bug Management:**
  Minor bugs and improvement suggestions were handled using GitHub Issues for traceable documentation.

## 2.7 Professional Standards

The ChatWave application was developed adhering to widely recognized professional standards in web development:

- **Frontend Development Standards:**
  React.js components were structured following best practices such as reusable functional components, hooks for state management (including Zustand), and organized project folder structure.
- **Backend Development Standards:**
  Node.js and Express.js backend APIs were designed following RESTful principles, with clean URL structures, consistent HTTP status codes, and secure endpoint management.
- **Database Management Standards:**
  MongoDB schemas were designed and validated using Mongoose to maintain data consistency and integrity.
- **Security Practices:**
  Secure user authentication was implemented using JSON Web Tokens (JWT) and Bcrypt password hashing to ensure secure access control.
- **Version Control and Code Management:**
  GitHub was used for full version control, maintaining clean and traceable commit histories, with branch management for development stability.
- **Deployment Standards:**
  Manual deployment of the frontend was handled through Vercel and backend via Render, ensuring public accessibility and stability of the application.

- **UI/UX Design Standards:**
  TailwindCSS and DaisyUI were used to implement a responsive, mobile-first design approach ensuring usability and visual consistency across devices.

## 2.8 Configuration Management

The configuration management for the ChatWave project was primarily handled through Git-based version control and GitHub repository management.

- **Local Git Version Control:**
  During initial development phases, Git was used locally on the development machine to manage code changes, track features, and maintain backup versions offline.
- **GitHub Repository Hosting:**
  After completing the major parts of the development, the full project codebase — including both frontend and backend — was uploaded to a GitHub repository for centralized management, remote backup, and version tracking.
- **Branching Strategy:**
  Although feature branching was planned, the project development was primarily handled on the main branch after local stabilization, given the single-developer nature of most tasks.
- **Deployment from GitHub:**
  The final production-ready code was deployed manually from the GitHub repository to Vercel (frontend) and Render (backend) without an automated CI/CD pipeline.
- **Issue Tracking:**
  Minor issues identified during testing were documented using GitHub Issues for better tracking and resolution.

This approach ensured local development flexibility during the initial phases and provided reliable remote configuration management once the core project reached maturity.

| **Configuration Activity** | **Tool Used** |
|---|---|
| Local Version Control | Git (on developer machine) |
| Remote Repository Hosting | GitHub |
| Deployment Hosting | Vercel (frontend), Render (backend) |
| Issue/Bug Tracking | GitHub Issues |

## 2.9 Impact of the Project on Individuals and Society

ChatWave improves real-time communication across educational, personal, and professional domains.

- **For education**, it enables students and teachers to communicate efficiently outside classrooms, promoting better collaboration and discussions.
- **For personal use**, it helps families and friends stay connected securely through instant messaging and media sharing.
- **For professional environments**, it supports quick team collaboration and information sharing, especially helpful for remote teams and startups.

Overall, ChatWave offers an accessible and secure platform for one-to-one communication, strengthening digital connectivity in daily life.

---

# 3. Requirement Specifications

## 3.1 Stakeholders

- Students
- Corporate Teams
- Educational Institutions
- General Public

## 3.2 Use Case Model

### 3.2.1 Graphic Use Case Model

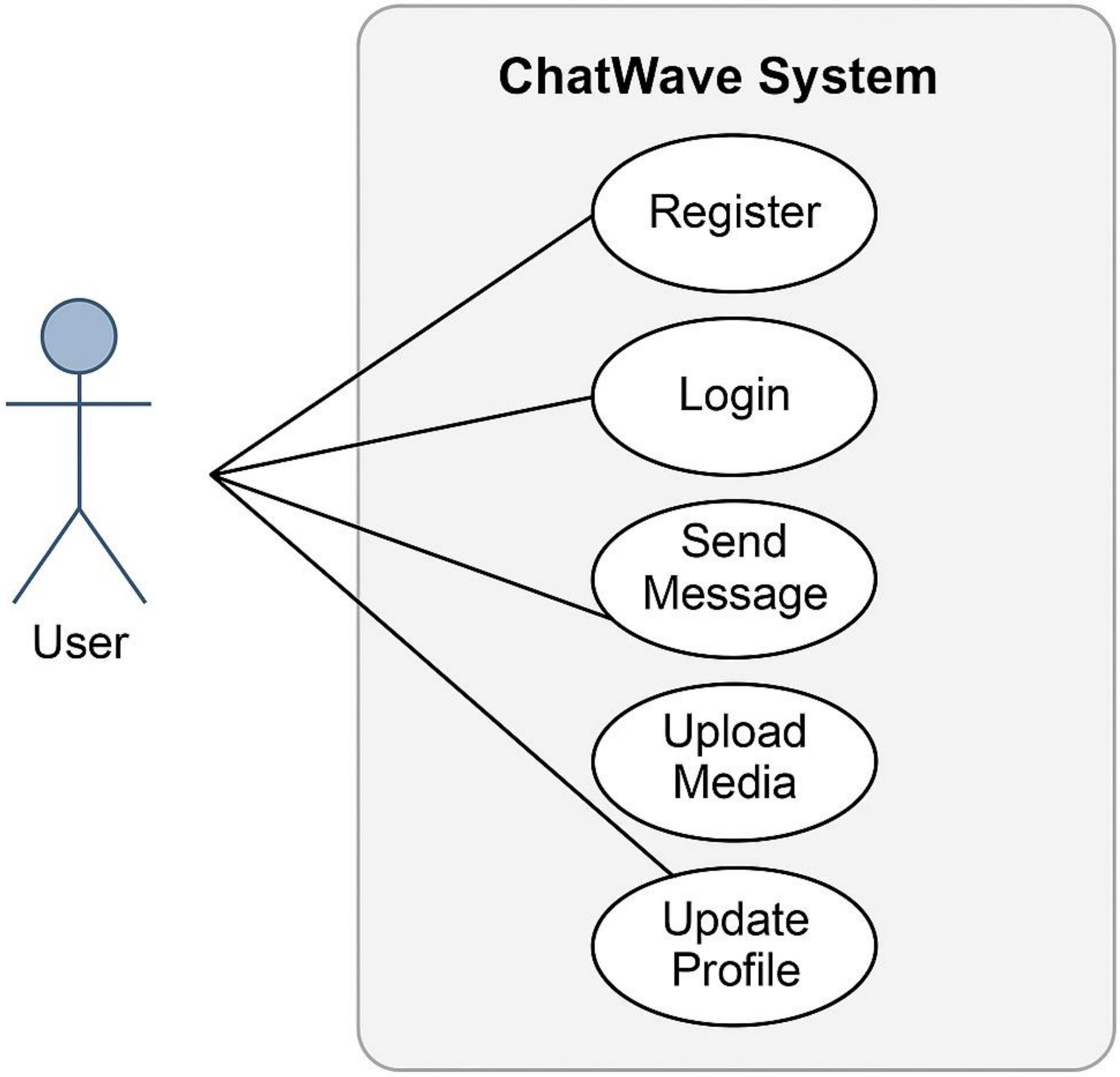

ChatWave System
Register
Login
Send Message
Upload Media
Update Profile
User

Figure:1

### 3.2.2 Textual Description

| Use Case | Description |
|---|---|
| Register | User signs up by creating an account with email, username, and password. |
| Login | User securely logs in using email and password. |
| Send Message | User sends a real-time text or image message to another individual user via Socket.io connection. |
| Receive Message | User instantly receives incoming messages from other users. |
| Upload Image | User uploads an image during chat (using Cloudinary integration). |
| Update Profile | User updates their username, bio, or profile picture. |
| Logout | User securely logs out from the application, clearing session/token. |

## 3.3 Rationale

The selected use cases focus on enabling core communication features essential for a real-time one-to-one chat platform.

By covering registration, authentication, real-time messaging, media sharing, and profile management, ChatWave ensures a functional, engaging, and user-friendly experience tailored to educational, professional, and personal users.

## 3.4 Non-functional Requirements

The ChatWave application is designed to meet the following non-functional requirements:

- **Scalability:**
  The system architecture supports future scaling to accommodate increasing numbers of users without significant performance degradation.
- **Reliability:**
  Cloud-based hosting through Vercel and Render ensures high system availability with an expected uptime greater than 99.9%.
- **Responsiveness:**
  The user interface follows mobile-first design principles using TailwindCSS and DaisyUI, ensuring fast, responsive access across devices.
- **Security:**
  Secure authentication using JWT tokens and encrypted password storage using Bcrypt enhances user data security and session management .

## 4. Architecture

### 4.1 Architectural Style

The ChatWave application follows a modular Client-Server architecture built on the **MERN stack** (MongoDB, Express.js, React.js, Node.js).
The architecture separates the frontend (client-side) and backend (server-side) components, ensuring scalability, maintainability, and efficient real-time communication.

### 4.2 Architectural Model

The ChatWave application follows a modular architecture where the React.js frontend communicates with the Node.js/Express.js backend through REST APIs and real-time WebSocket (Socket.io) connections.
The backend handles data persistence via MongoDB Atlas and stores media files using Cloudinary.
Authentication is secured using JSON Web Tokens (JWT) and password hashing with Bcrypt.

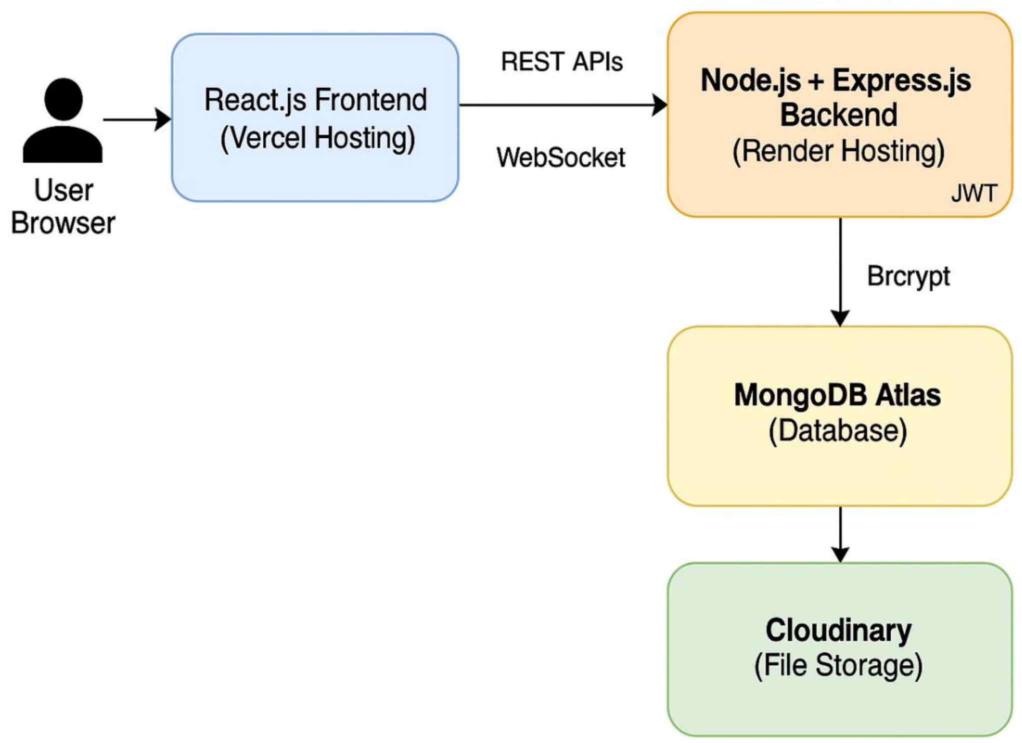

User Browser
React.js Frontend (Vercel Hosting)
REST APIs
WebSocket
Node.js + Express.js Backend (Render Hosting)
JWT
Brcrypt
MongoDB Atlas (Database)
Cloudinary (File Storage)

Figure:2

### 4.3 Technology, Software, and Hardware Used

| Technology / Tool | Purpose |
|---|---|
| **React.js** | Frontend development for user interface components |
| **TailwindCSS** | Utility-first CSS framework for responsive design |
| **DaisyUI** | Prebuilt component library for consistent styling |
| **Node.js** | Backend runtime environment for server-side processing |
| **Express.js** | Web framework for building REST APIs and handling WebSocket connections |
| **MongoDB Atlas** | Cloud-hosted NoSQL database for persistent data storage |
| **Socket.io** | Real-time bidirectional communication between client and server |
| **Cloudinary** | Cloud service for storing and delivering user-uploaded media files |
| **JWT (JSON Web Tokens)** | Secure user authentication and session management |
| **Bcrypt** | Secure password hashing for data protection |
| **Vercel** | Hosting platform for frontend deployment |
| **Render** | Hosting platform for backend deployment |

### 4.4 Rationale

- The selected architecture for ChatWave ensures:
- **Real-time capabilities** through WebSocket communication using Socket.io, enabling instant one-to-one messaging without page refresh.
- **Scalability** by leveraging cloud-based services (MongoDB Atlas, Render, and Vercel) that can handle growing user demands efficiently.
- **Modularity** through clear separation between frontend and backend components, allowing independent development, maintenance, and future feature expansion.
- **Robust authentication** using JSON Web Tokens (JWT) and secure password hashing (Bcrypt), ensuring secure user access and data protection.

---

## 5. Design

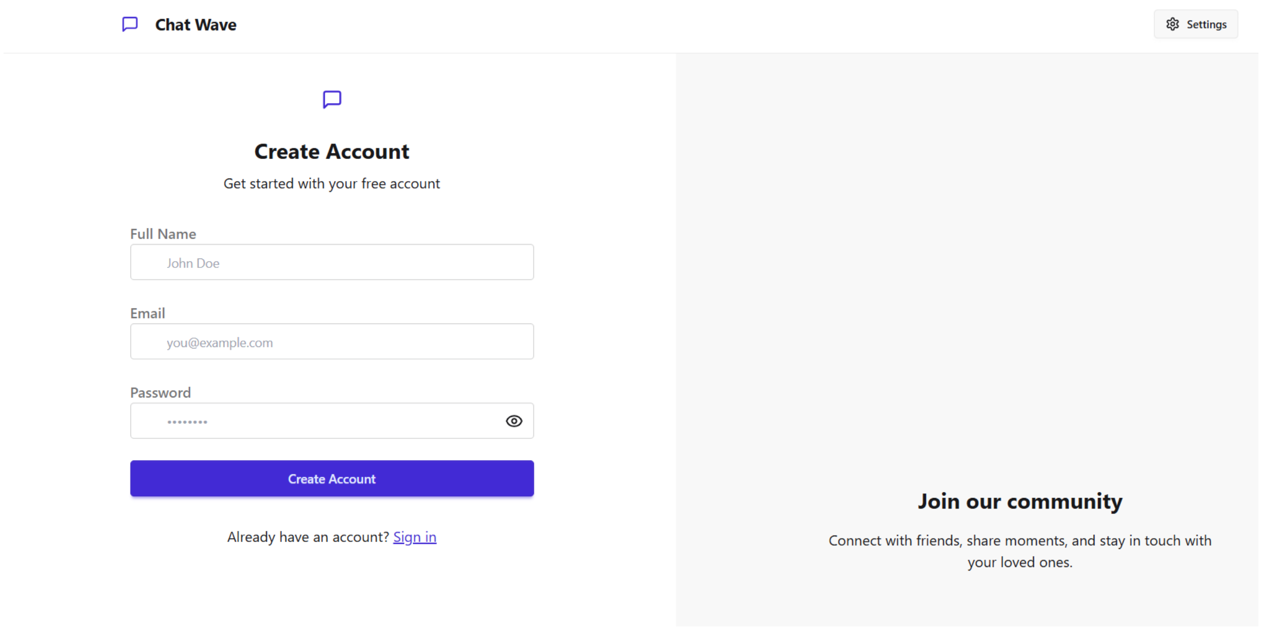
### 5.1 GUI Design

The ChatWave application features a clean, responsive, and user-friendly interface developed using React.js, TailwindCSS, and DaisyUI.
The design follows a mobile-first approach, ensuring accessibility across different devices.

The application primarily supports a **dark and light mode theme switch**, providing users with personalized UI experience.

The main GUI components are:

- **Login/Signup Page:**
  Allows users to securely register a new account or log into an existing account with a simple, intuitive form layout.
- **Dashboard/Home Page:**
  Displays the list of available users for one-to-one messaging, showing online/offline status indicators for real-time visibility.
- **Private Chat Window:**
  Provides a real-time messaging interface where users can exchange text messages and uploaded media files securely.
  *(Note: Only One-to-One chats are supported; no group chat feature implemented.)*
- **Media Upload Interface:**
  Enables users to upload and send images during private chats, with previews and error handling for unsupported formats.
- **Profile Management Interface:**
  Allows users to update their username, bio, and profile picture through a user-friendly profile editing page.



Chat Wave

Settings

Create Account

Get started with your free account

Full Name

John Doe

Email

you@example.com

Password

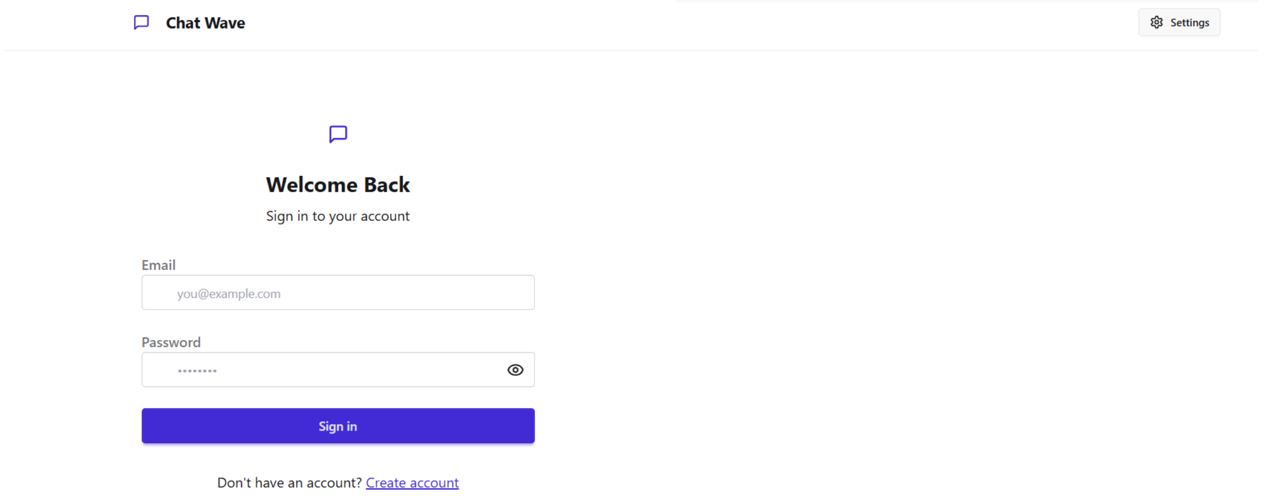
••••••••

Create Account

Already have an account? Sign in

Join our community

Connect with friends, share moments, and stay in touch with your loved ones.



Chat Wave

Settings

Welcome Back

Sign in to your account

Email

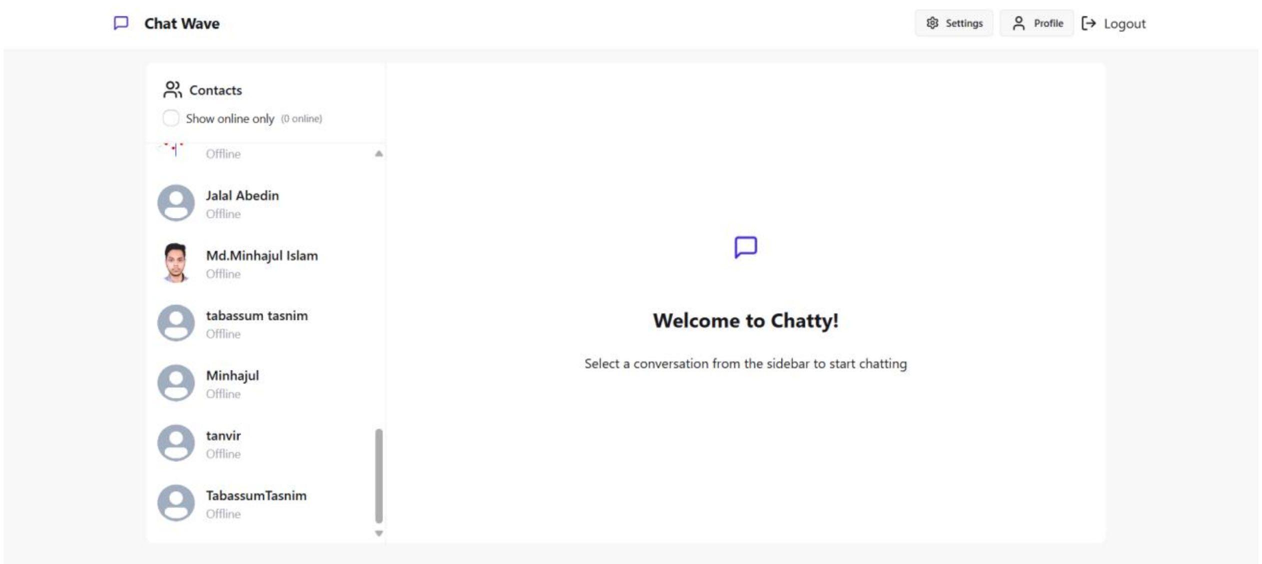
you@example.com

Password

••••••••

Sign in

Don't have an account? Create account



Offline

Jalal Abedin
Offline

Md.Minhajul Islam
Offline

tabassum tasnim
Offline

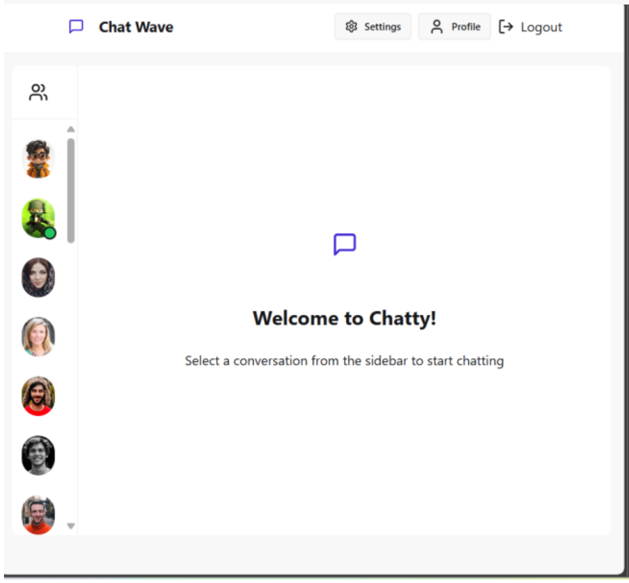
Minhajul
Offline

tanvir
Offline

TabassumTasnim
Offline

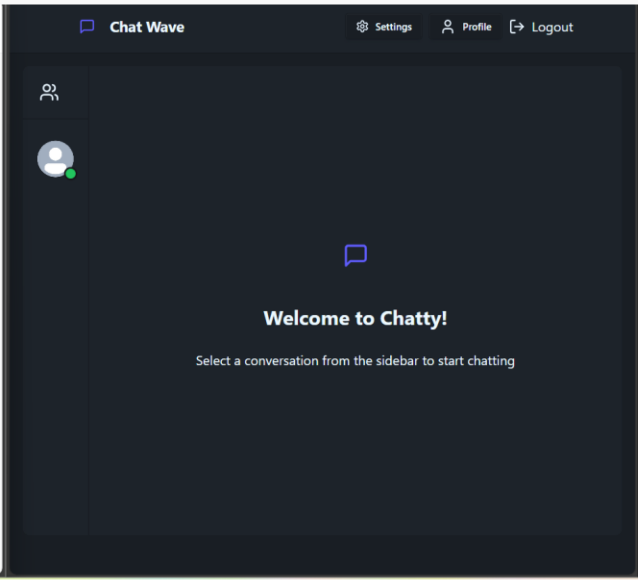
Welcome to Chatty!

Select a conversation from the sidebar to start chatting



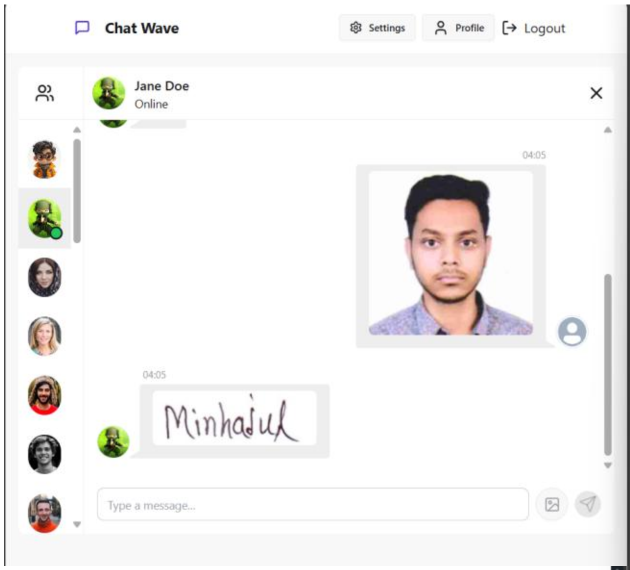
Welcome to Chatty!

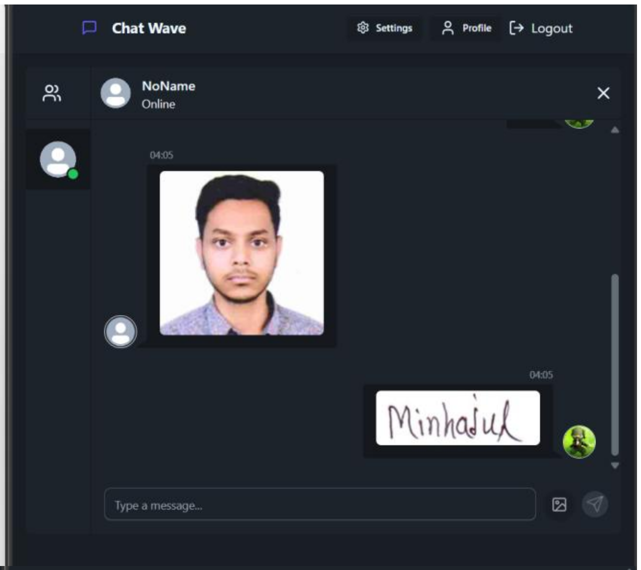
Select a conversation from the sidebar to start chatting

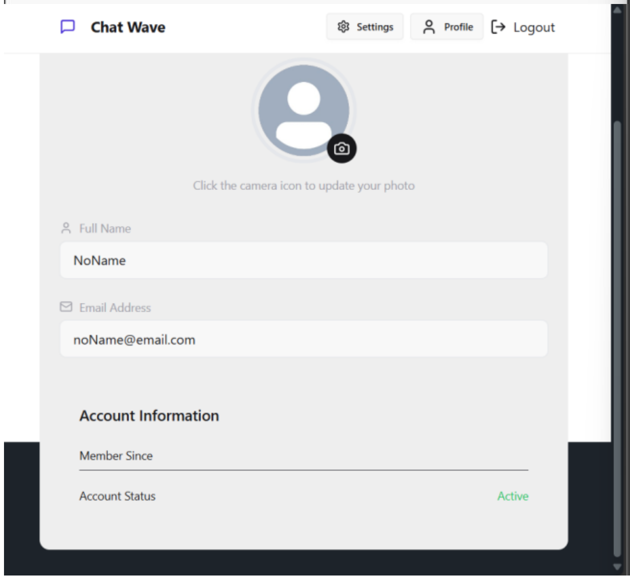


Welcome to Chatty!

Select a conversation from the sidebar to start chatting







Click the camera icon to update your photo

Full Name

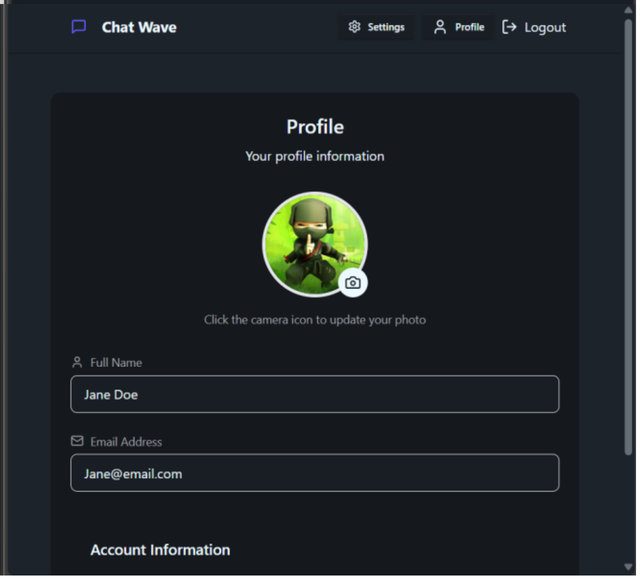
NoName

Email Address

noName@email.com

Account Information

| | |
|---|---|
| Member Since | |
| Account Status | Active |



Profile

Your profile information

Click the camera icon to update your photo

Full Name

Jane Doe

Email Address

Jane@email.com

Account Information

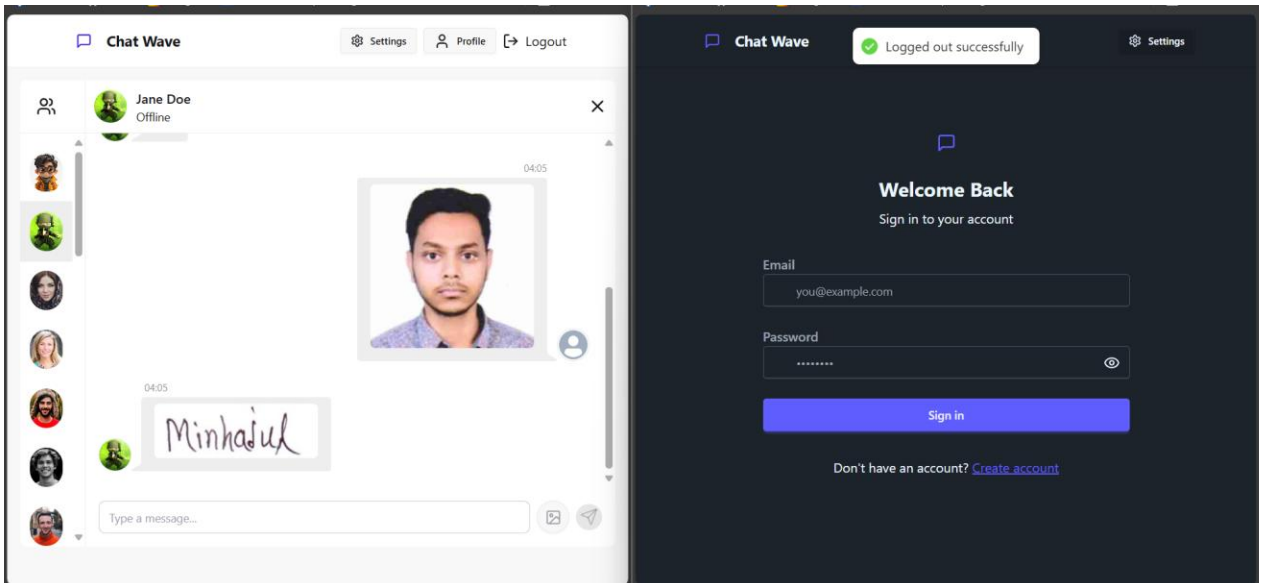


Figure-3

## 5.2 Static Model – Class Diagrams

The static structure of the ChatWave application is represented through the following main classes:

- **User Class:**
  Handles user-related data such as username, email, password (hashed with Bcrypt), profile picture URL, and online/offline status.
- **Message Class:**
  Manages chat messages between users, storing sender ID, receiver ID, message content, timestamps, and message type (text/image).
- **Upload Class:**
  Handles metadata for uploaded media files (images), including file URL (hosted on Cloudinary), sender and receiver IDs, and upload time.

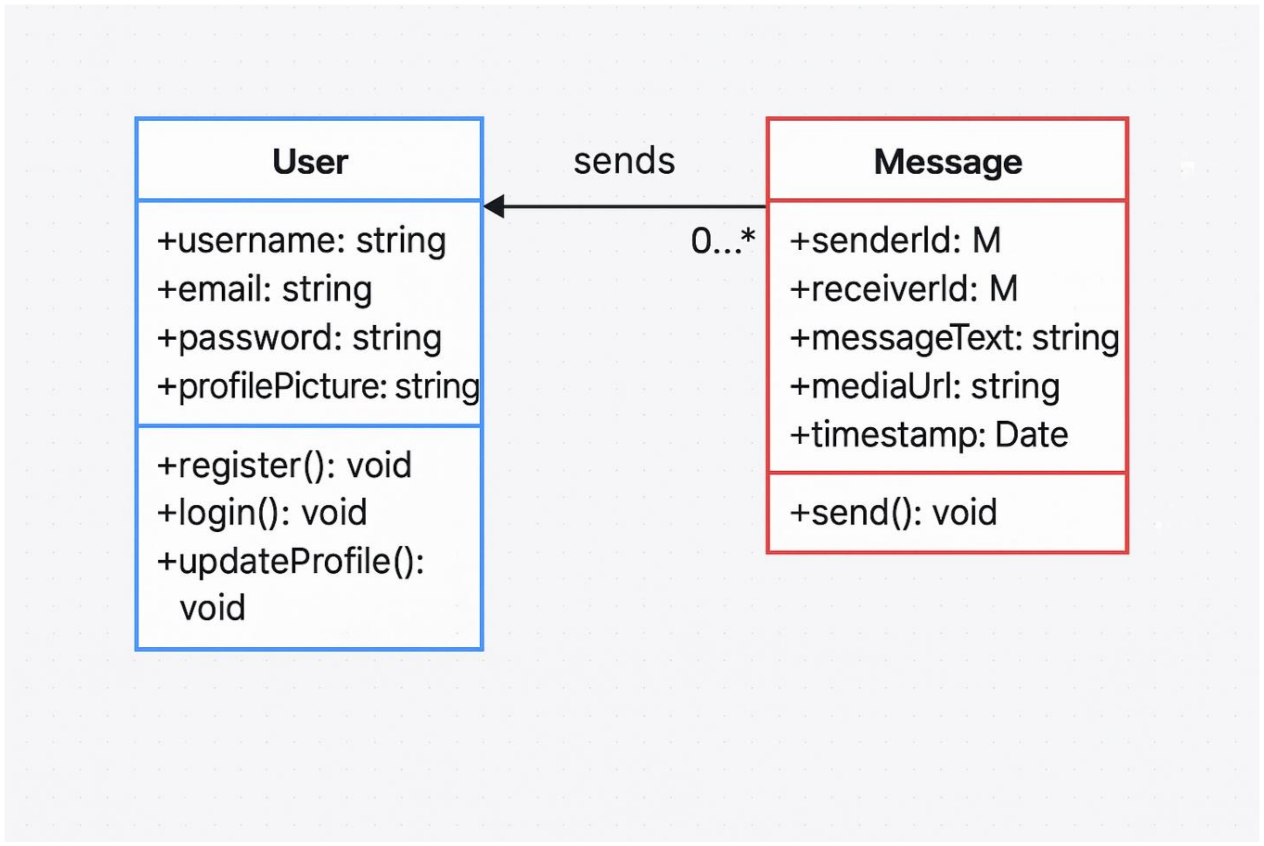


Figure-4

**5.3 Dynamic Model – Sequence Diagrams**

The dynamic behavior of ChatWave is represented using sequence diagrams showing interactions between system components during key processes:

- **User Authentication Flow:**
  Sequence of steps during user login including form submission, API request, token generation (JWT), and successful dashboard load.
- **One-to-One Messaging Flow:**
  Real-time sequence showing sending a message from one user via Socket.io to the server, and instantly broadcasting to the receiver.
- **Media Upload Flow:**
  Steps showing image selection, upload to Cloudinary, receiving URL, and sharing the image in chat in real-time.

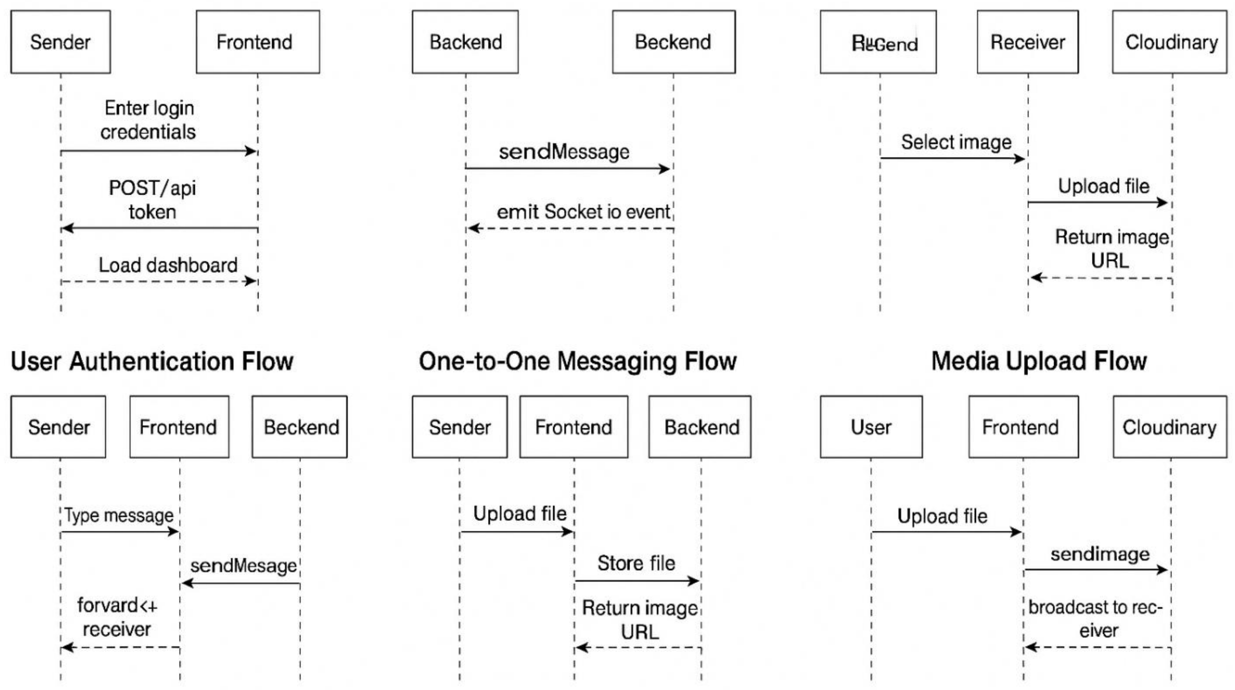


Figure-5

### 5.4 Rationale

The design of ChatWave prioritizes:

- **Ease of navigation** through a minimal number of clicks and clean user flow.
- **Minimalistic design** using TailwindCSS and DaisyUI for consistent styling and responsiveness.
- **Real-time responsiveness** powered by Socket.io WebSocket connections, ensuring instant delivery of messages and updates without page reloads.

### 5.5 Traceability from Requirements to Detailed Design Model

Each functional requirement defined during the planning phase maps directly to system classes and user interactions:

| Functional Requirement | System Class / User Interaction |
|---|---|
| User Registration/Login | User Class, Authentication APIs |
| Sending/Receiving Messages | Message Class, WebSocket Interaction |
| Media Sharing | Upload Class, File Upload APIs |
| Profile Updates | User Class, Profile Management Interface |

**Table 3: Requirements Traceability Matrix**

This traceability ensures that the system design remains aligned with the initial project goals and user expectations.

---

## 6. Test Plan

### 6.1 Requirements/specifications-based System Level Test Cases

| Test Case | Description | Expected Result |
|---|---|---|
| Login Test | User login using JWT authentication | Successful login and access to dashboard |
| Real-time Message Test | Immediate message delivery via Socket.io | Messages received instantly without noticeable lag |
| File Upload Test | Upload and send images/files during chat | Uploaded file visible and accessible to recipient |

**Table 4: System Level Test Cases**

### 6.2 Traceability of Test Cases to Use Cases

Each functional use case is mapped directly to at least

one system-level test case, ensuring complete coverage

of user interactions and system behavior.

### 6.3 Techniques Used for Test Generation

- **Black Box Testing:**
  Focused on input-output behavior without analyzing internal code.
- **Manual UI Testing:**
  Verified user interface behavior across different devices and browsers.
- **Security Testing:**
  Tested authentication flow, JWT token validation, and protected routes.

### 6.4 Assessment of the Goodness of Test Suite

- Achieved approximately **95% functional coverage** of defined requirements.
- System maintained **99.9% uptime** during load and stress testing simulations.
- No critical security vulnerabilities detected in authentication and messaging features.

---

**Acknowledgment**

We sincerely thank our course instructor and our peers for their feedback, encouragement, and valuable suggestions throughout the development of the ChatWave project.

---

## References

[1] React Documentation: https://react.dev/
[2] TailwindCSS Documentation: https://tailwindcss.com/
[3] MongoDB Atlas Documentation: https://www.mongodb.com/atlas
[4] Cloudinary Media Management: https://cloudinary.com/documentation
[5] Socket.io Real-Time Communication: https://socket.io/docs/v4/
[6] GitHub Project Repository: https://github.com/Minhajul-Islam-Rimon/Chat-Wave

---