**INTERNSHIP PROJECT DOCUMENT**

**ON “Real-Time Chat App”**

**Submitted by**

(Intern)

K Sree Navya

**Submitted to**

**Founder** - Kanduri Abhinay

**CEO & CTO** – Rithin Varma

**1. INTRODUCTION**

The Real-Time Chat Application is a web-based chat system that enables users to communicate in real time. The system utilizes Socket.IO for WebSocket-based instant messaging, while MongoDB is used for message persistence.

The key functionalities of the chat application include:

* User Registration & Authentication using JWT (JSON Web Tokens)
* Real-Time Messaging using Socket.IO
* Message Persistence using MongoDB
* File Sharing for documents and images
* Scalability for handling multiple users

This project was developed as an internship assignment to explore Node.js, Express, MongoDB, and WebSocket technologies.

**2. PROJECT OBJECTIVE**

The primary objective of this project is to develop a real-time messaging system where multiple users can:

* Register/Login using a secure authentication system
* Join a group chat and send/receive messages instantly
* Share files, images, and documents with other users
* Retrieve previous messages stored in the database

The chat system mimics popular messaging applications like WhatsApp and Telegram but is designed for web-based group chat.

**3. SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)**

The project follows the Agile SDLC methodology, ensuring an iterative approach to development.

Phases of SDLC in this Project:

Planning

* Define functional requirements (real-time messaging, authentication, etc.).
* Select technology stack: Node.js, Express, MongoDB, Socket.IO, JWT.
* Identify hardware & software requirements.

Requirement Analysis

* Functional Requirements: Messaging, Authentication, File Uploads.
* Non-Functional Requirements: Performance, Scalability, Security.

System Design

* Backend: Node.js + Express
* Frontend: HTML, CSS, JavaScript
* Database: MongoDB

Implementation

* Develop authentication system (JWT-based).
* Implement real-time chat with WebSockets.
* Store messages in MongoDB.

Testing

* Perform unit, integration, and load testing.
* Debug errors & optimize performance.

Deployment & Maintenance

* Deploy on Heroku/AWS/DigitalOcean.
* Monitor and improve system performance.

**4. TECHNOLOGIES USED**

| **Technology** | Purpose |
| --- | --- |
| **Node.js** | Backend runtime environment |
| **Express.js** | Web framework for handling API requests |
| **MongoDB** | NoSQL database for message storage |
| **Socket.IO** | WebSockets for real-time messaging |
| **Mongoose** | MongoDB ORM (Schema & Query Handling) |
| **Bcrypt.js** | Password hashing for security |
| **JWT (JSON Web Tokens)** | Authentication & session management |
| **HTML, CSS, JavaScript** | Frontend development |
| **Multer** | File upload middleware for handling documents & images |

**5. PROCEDURE AND METHODS USED**

User Registration

* User provides username & password.
* Password hashed with bcrypt and stored in MongoDB.

User Login

* User enters username & password.
* System retrieves hashed password and compares it with bcrypt.
* If correct, system grants access & returns JWT token.

Real-Time Chat Process

* Users connect via WebSockets.
* Messages are sent in real-time using Socket.IO.
* Messages are stored in MongoDB.

File Sharing

* Users upload images & documents.
* Files are converted to base64 & sent via WebSockets.

**6. IMPLEMENTATION DETAILS**

**User Registration**

**1️⃣ Start the program.  
2️⃣ User enters username & password.  
3️⃣ Password hashed using bcrypt.  
4️⃣ User data stored in MongoDB.  
5️⃣ End the process.**

**User Login**

**1️⃣ Start the program.  
2️⃣ User enters username & password.  
3️⃣ Retrieve hashed password from MongoDB.  
4️⃣ Compare input password with stored hash.  
5️⃣ If valid, generate JWT token.  
6️⃣ End the process.**

**Message Sending (WebSocket)**

**1️⃣ User enters chat room.  
2️⃣ User sends message via WebSocket.  
3️⃣ Message is broadcast to all connected users.  
4️⃣ Message is stored in MongoDB.  
5️⃣ Messages persist when users rejoin.**

**7. TESTING AND DEBUGGING**

**✔ Unit Testing: Validate API endpoints and database operations.  
✔ Integration Testing: Verify WebSocket connections.  
✔ Load Testing: Simulate multiple users in the chat room.  
✔ Security Testing: Check for SQL Injection & XSS vulnerabilities.**

**8. FUTURE SCOPE**

**🔹 Multiple Chat Rooms – Allow users to create & join private rooms.  
🔹 Direct Messaging – One-to-one chat functionality.  
🔹 Typing Indicators – Show when a user is typing.  
🔹 Message Reactions – Users can react (like, emoji) to messages.  
🔹 End-to-End Encryption – Secure messaging like WhatsApp.  
🔹 Deployment – Host on AWS, Heroku, or DigitalOcean.**

**9. CHALLENGES AND SOLUTIONS**

* **Handling Multiple Users Simultaneously → Implemented WebSockets for real-time updates**
* **Data Persistence → Used MongoDB to store chat history**
* **Authentication & Security → Implemented JWT for user sessions**

**10. ADVANTAGES**

**✅ Real-Time Communication – Instant messaging with WebSockets.  
✅ Persistent Messages – Messages are saved in MongoDB.  
✅ Secure Authentication – JWT authentication protects users.  
✅ Scalable – Can handle multiple concurrent users.  
✅ Cross-Platform – Works on web browsers & mobile.  
✅ File Sharing – Users can send images & documents.**

**11. CONCLUSION**

**This Real-Time Chat Application successfully enables instant messaging with file sharing, ensuring a secure and scalable chat environment.**

**13. REFERENCES**

1. **Node.js Documentation:** [**https://nodejs.org/en/docs/**](https://nodejs.org/en/docs/)
2. **Express.js Documentation:** [**https://expressjs.com/**](https://expressjs.com/)
3. **Socket.IO Documentation:** [**https://socket.io/docs/**](https://socket.io/docs/)
4. **MongoDB Documentation:** [**https://www.mongodb.com/docs/**](https://www.mongodb.com/docs/)
5. **WebSockets Overview:** [**https://developer.mozilla.org/en-US/docs/Web/API/WebSockets\_API**](https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API)
6. **REST API Best Practices:** [**https://www.freecodecamp.org/news/rest-api-best-practices-rest-endpoint-design/**](https://www.freecodecamp.org/news/rest-api-best-practices-rest-endpoint-design/)
7. **File Uploads with Multer:** [**https://www.npmjs.com/package/multer**](https://www.npmjs.com/package/multer)