# Chat Web Application Documentation

## Table of Contents

* [Overview](#overview)
* [Architecture](#architecture)
* [Setup Instructions](#setup-instructions)
* [Backend Details](#backend-details)
* [Frontend Details](#frontend-details)
* [API Documentation](#api-documentation)
* [Data Models](#data-models)
* [Troubleshooting](#troubleshooting)

## Overview

This project is a full-stack real-time chat application with user authentication, media sharing, and a modern, responsive UI. It is built using Node.js, Express, MongoDB, Socket.io, and vanilla JavaScript.

## Architecture

* **Backend:** Node.js, Express, MongoDB (Mongoose), Socket.io, Multer, JWT, bcryptjs
* **Frontend:** HTML5, CSS3, JavaScript (ES6+), Bootstrap
* **Communication:** REST API for authentication, user, and chat management; Socket.io for real-time messaging
* **Authentication:** JWT-based, with secure password hashing

### Folder Structure

Chat web/  
├── backend/  
│ ├── models/ # Mongoose models (User, Chat)  
│ ├── routes/ # Express API routes (auth, users)  
│ ├── uploads/ # Uploaded files (avatars, media)  
│ ├── server.js # Main backend server (Express + Socket.io)  
│ ├── package.json # Backend dependencies  
├── frontend/  
│ ├── index.html # Main chat UI  
│ ├── dashboard.html # User dashboard  
│ ├── login.html # Login page  
│ ├── register.html # Registration page  
│ ├── home.html # Landing page  
│ ├── \*.js, \*.css # Frontend scripts and styles  
├── package.json # (root) Project-level dependencies

## Setup Instructions

### Prerequisites

* Node.js (v16+ recommended)
* MongoDB (local or Atlas)

### Backend Setup

1. Navigate to the backend folder:

* cd "Chat web/backend"

1. Install dependencies:

* npm install

1. Create a .env file in backend/ with:

* MONGO\_URI=your\_mongodb\_connection\_string  
  JWT\_SECRET=your\_jwt\_secret  
  PORT=5000

1. Start the backend server:

* node server.js

### Frontend Setup

No build step required. The backend serves the frontend static files. Simply open home.html in your browser, or access via http://localhost:5000/ after starting the backend.

## Backend Details

* **Express** serves API endpoints and static frontend files.
* **Socket.io** enables real-time messaging between users.
* **Multer** handles file uploads (media sharing, avatars).
* **JWT** is used for authentication; tokens are required for protected routes.
* **MongoDB** stores users and chat messages.

### Key Files

* server.js: Main server file, sets up Express, Socket.io, routes, and MongoDB connection.
* models/User.js: User schema/model (username, email, password, status, bio, avatar).
* models/Chat.js: Chat schema/model (sender, receiver, message, type, fileUrl, readBy, timestamp).
* routes/auth.js: Authentication endpoints (register, login, logout).
* routes/users.js: User and chat endpoints.

## Frontend Details

* **index.html**: Main chat interface, connects to Socket.io, handles messaging and file sharing.
* **dashboard.html**: User dashboard, shows user list and unread message counts.
* **login.html/register.html**: Authentication forms.
* **home.html**: Landing page.
* **script.js/dashboard.js**: Main frontend logic for chat and dashboard.
* **styles.css/dashboard.css**: Styling for chat and dashboard.

## API Documentation

### Authentication

* POST /api/auth/register — Register a new user
  + Body: { username, email, password }
  + Response: { token, username }
* POST /api/auth/login — Login
  + Body: { email, password }
  + Response: { token, username }
* POST /api/auth/logout — Logout
  + Body: { username }
  + Response: { message }

### Users & Chats

* GET /api/users — List all users except the current user (requires JWT)
* GET /api/users/chats?user1=alice&user2=bob — Get chat history between two users
* POST /api/users/chats/mark-read — Mark messages as read
  + Body: { user1, user2, username }
* GET /api/users/chats/unread-count?username=alice — Get unread message counts for a user

### File Upload

* POST /api/upload — Upload a file (multipart/form-data, field: file)
  + Response: { filename, url }

## Data Models

### User

{  
 username: String, // unique  
 email: String, // unique  
 password: String, // hashed  
 status: String, // user status message  
 bio: String, // user bio  
 avatar: String // avatar file path or URL  
}

### Chat

{  
 sender: String, // username  
 receiver: String, // username  
 message: String, // text message (optional for file messages)  
 type: String, // 'text' or 'file'  
 fileUrl: String, // file URL (if type is 'file')  
 fileName: String, // original file name  
 fileType: String, // MIME type  
 readBy: [String], // usernames who have read the message  
 sentByUsername: String, // sender's username  
 timestamp: Date  
}

## Troubleshooting

* **MongoDB connection errors:**
  + Check your MONGO\_URI in .env.
  + Ensure MongoDB is running locally or your Atlas cluster is accessible.
* **CORS issues:**
  + The backend enables CORS for all origins. If you need to restrict it, update the CORS settings in server.js.
* **Socket.io not connecting:**
  + Ensure the backend server is running and accessible at the expected port.
* **File uploads not working:**
  + Ensure the uploads/ directory exists and is writable.
* **JWT errors:**
  + Make sure the frontend sends the token in the Authorization header as Bearer <token> for protected routes.

For further questions or issues, please refer to the code comments or open an issue in your project repository.