Sprint 1 — Backend Architecture (MERN + Socket.IO + E2EE-ready)

1️⃣ High-Level Overview

[Client (React/Vite)]

|

| HTTPS (REST API calls) + WSS (WebSocket real-time)

v

[Express.js API Server] ---- [Socket.IO Server]

| **|**

| **|**

[Auth Service] **[Real-Time Chat Service]**

| **|**

[MongoDB + Mongoose ORM] <--> [Redis Pub/Sub for scaling]

2️⃣ Core Backend Modules

A. Server Setup

- **Framework: Express.js**
- **Real-time: Socket.IO (running on same server in Sprint 1; separate in future for scaling)**
- **Security: Helmet.js, CORS, Rate limiting**

B. Authentication Module

- **JWT-based auth (short-lived access token, refresh token)**
- **Password hashing: bcrypt**
- **Future-proof for OAuth (Google/Apple sign-in in later sprints)**
- **Middleware:**
  - **authMiddleware → Verifies JWT**
  - **socketAuth → Auth for Socket.IO handshake**

C. User Management

- **MongoDB users collection:**

{

"_id": "ObjectId",

"username": "string",

"email": "string",

"passwordHash": "string",

"profilePic": "string",

"status": "online/offline/away",

"lastSeen": "Date"

}

- **Endpoints:**
  - **POST /auth/register**
  - **POST /auth/login**
  - **GET /user/me (fetch profile)**
  - **PUT /user/status (update online/offline)**

D. Chat Messaging Service

- **MongoDB messages collection:**

{

"_id": "ObjectId",

"chatId": "ObjectId",

"senderId": "ObjectId",

"receiverId": "ObjectId",

"content": "string",

"type": "text/image/file",

"timestamp": "Date",

"read": "boolean"

}

- **Endpoints:**
  - **GET /chat/:chatId → fetch chat history**

- POST /chat/message → send message (fallback to REST if WebSocket fails)

E. Real-time Messaging (Socket.IO)

- Events:
  - message:send → send message
  - message:receive → receive message
  - user:typing → typing indicator
  - user:status → online/offline updates
- Flow:

1. Client emits message:send
2. Server validates & stores in MongoDB
3. Server emits message:receive to target user's socket room

F. Database Layer

- MongoDB + Mongoose → Flexible schema for future upgrades
- Indexes:
  - Compound index on chatId & timestamp for fast retrieval

G. Encryption Setup

- Sprint 1: Store encrypted messages in DB using AES-256 (symmetric)
- Later sprints: Upgrade to E2EE with per-user keys

H. Error Handling & Logging

- Centralized error handler
- Winston/Morgan for logging HTTP & Socket events

3️⃣ Folder Structure

```
backend/
├── src/
│   ├── config/
│   │   ├── db.js          # MongoDB connection
│   │   ├── redis.js       # For scaling in future
```

```
|   ├── models/
|   |   ├── User.js
|   |   ├── Message.js
|   ├── routes/
|   |   ├── authRoutes.js
|   |   ├── userRoutes.js
|   |   ├── chatRoutes.js
|   ├── controllers/
|   |   ├── authController.js
|   |   ├── userController.js
|   |   ├── chatController.js
|   ├── services/
|   |   ├── authService.js
|   |   ├── chatService.js
|   ├── sockets/
|   |   ├── chatSocket.js
|   |   ├── userSocket.js
|   ├── middlewares/
|   |   ├── authMiddleware.js
|   |   ├── errorHandler.js
|   ├── utils/
|   |   ├── encrypt.js
|   ├── app.js            # Express app
|   ├── server.js         # Socket.IO + HTTP server
├── package.json
```

---

4 Tech Stack (Sprint 1 Backend)

- **Node.js + Express.js**
- **Socket.IO — Real-time messaging**
- **MongoDB + Mongoose — Storage**
- **JWT — Auth**
- **Bcrypt — Password hashing**
- **Helmet.js & CORS — Security**
- **Winston/Morgan — Logging**

5️⃣ Expected Time for Sprint 1 Backend

Task	Estimated Time
Project setup & config	1 day
Auth system (JWT + bcrypt)	3 days
User management endpoints	2 days
Chat endpoints	3 days
Socket.IO real-time layer	3 days
Encryption & message storage	1 day
Testing & debugging	1 day
Total	14 days