CollabChat Backend - Product Documentation

CollabChat is a **real-time chat and collaboration platform** backend built with modern web technologies. It provides secure, scalable, and feature-rich messaging capabilities including private chats, group chats, file uploads, end-to-end encryption, and real-time WebSocket communication.

Table of Contents

1. Project Overview
2. Technology Stack
3. Libraries & Dependencies
4. Folder Structure
5. API Reference
6. WebSocket Events
7. Database Models
8. Code Documentation
    - Entry Point (server.js)
    - Socket.IO Setup
    - Configuration
    - Middleware
    - Controllers
    - Routes
    - Models
    - Utilities
9. Security Features
10. Docker Support

Project Overview

CollabChat Backend is a Node.js Express application that powers a real-time chat platform. Key features include:

- **User Authentication**: JWT-based registration and login system
- **Private Messaging**: One-on-one encrypted conversations
- **Group Chats**: Create and manage group conversations with multiple members
- **Real-time Communication**: Socket.IO powered WebSocket connections
- **File Uploads**: Cloudinary integration for media and document sharing
- **Message Encryption**: AES-256-CBC encryption for message security
- **Online Status**: Real-time user presence tracking

Technology Stack

Technology	Purpose
Node.js	JavaScript runtime environment
Express.js	Web application framework
MongoDB	NoSQL database for data persistence
Mongoose	MongoDB object modeling (ODM)
Socket.IO	Real-time bidirectional event-based communication
JWT	JSON Web Tokens for authentication
Cloudinary	Cloud-based media management
Docker	Containerization for deployment

Libraries & Dependencies

Production Dependencies

Package	Version	Purpose
`express`	^4.18.2	Web framework for building REST APIs and HTTP servers
`mongoose`	^7.0.0	MongoDB ODM for data modeling and database operations
`mongodb`	^5.9.2	Official MongoDB driver for Node.js
`socket.io`	^4.6.1	Real-time WebSocket communication library
`jsonwebtoken`	^9.0.0	JWT generation and verification for authentication
`bcryptjs`	^2.4.3	Password hashing using bcrypt algorithm
`cors`	^2.8.5	Cross-Origin Resource Sharing middleware
`dotenv`	^16.0.3	Environment variable management from .env files
`cloudinary`	^1.41.3	Cloud image and video management API
`multer`	^2.0.2	File upload handling middleware

Package	Version	Purpose
`multer-storage-cloudinary`	^4.0.0	Cloudinary storage engine for Multer

Development Dependencies

Package	Version	Purpose
`nodemon`	^2.0.20	Auto-restart server on file changes during development

---

Folder Structure

```
backend/
  server.js                 # Application entry point
  socket-io-setup.js        # WebSocket configuration and handlers
  test-db.js                # Database connection test utility
  package.json              # Project dependencies and scripts
  Dockerfile                # Docker container configuration
  README.md                 # Setup and API documentation
  .env                      # Environment variables (not in repo)

  config/
     cloudinary.js          # Cloudinary configuration and Multer setup

  controllers/
     authController.js      # User registration and login logic
     chatsController.js     # Message sending and retrieval logic
     debugController.js     # Debug and migration utilities
     fileUploadController.js # File upload handling with Cloudinary
     groupsController.js    # Group creation and management
     usersController.js     # User profile and listing logic

  middleware/
     auth.js                # JWT authentication middleware

  models/
     Chat.js                # Chat/conversation schema
     Group.js               # Group schema
     Message.js             # Message schema
     User.js                # User schema with password hashing
```

```
routes/
    auth.js               # Authentication routes (/api/v1/auth)
    chats.js              # Chat routes (/api/v1/chats)
    debug.js              # Debug routes (/api/v1/debug)
    groups.js             # Group routes (/api/v1/groups)
    users.js              # User routes (/api/v1/users)

utils/
    chatUtils.js          # Chat helper functions
    encryption.js         # AES-256-CBC encryption/decryption
```

API Reference

All API endpoints are prefixed with `/api/v1`

Authentication Routes (`/api/v1/auth`)

Method	Endpoint	Description	Auth Required
POST	`/register`	Register a new user	No
POST	`/login`	Login and get JWT token	No

Register User

```
POST /api/v1/auth/register
Content-Type: application/json

{
  "username": "john_doe",
  "password": "securepassword123"
}
```

Response:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "_id": "64abc123...",
    "username": "john_doe",
    "isOnline": false,
    "lastSeen": "2024-01-15T10:30:00.000Z"
  }
}
```

Login User

```
POST /api/v1/auth/login
Content-Type: application/json

{
  "username": "john_doe",
  "password": "securepassword123"
}
```

User Routes (`/api/v1/users`)

Method	Endpoint	Description	Auth Required
GET	`/me`	Get current user profile	Yes
GET	`/`	Get all users (excludes current)	Yes

Get Current User

```
GET /api/v1/users/me
Authorization: Bearer <token>
```

Get All Users

```
GET /api/v1/users?limit=50&offset=0
Authorization: Bearer <token>
```

Chat Routes (`/api/v1/chats`)

Method	Endpoint	Description	Auth Required
POST	`/send`	Send a message	Yes
GET	`/private/:userId`	Get private chat messages	Yes
GET	`/group/:groupId`	Get group chat messages	Yes
POST	`/private/:chatId/upload`	Upload file metadata (private)	Yes
POST	`/group/:chatId/upload`	Upload file metadata (group)	Yes
POST	`/upload/:chatId`	Upload file to Cloudinary	Yes
DELETE	`/private/:userId/clear`	Clear private chat	Yes

Method	Endpoint	Description	Auth Required
DELETE	`/group/:groupId/clear`	Clear group chat	Yes

Send Message

```
POST /api/v1/chats/send
Authorization: Bearer <token>
Content-Type: application/json

{
  "recipientId": "64xyz789...",  // For private chat
  "content": "Hello, how are you?",
  "type": "text"
}
```

Get Private Messages

```
GET /api/v1/chats/private/64abc123/64xyz789?limit=20&offset=0
Authorization: Bearer <token>
```

Upload File

```
POST /api/v1/chats/upload/64chatId123
Authorization: Bearer <token>
Content-Type: multipart/form-data

file: <binary>
caption: "Check this out!"
```

Group Routes (`/api/v1/groups`)

Method	Endpoint	Description	Auth Required
POST	`/`	Create a new group	Yes
GET	`/my`	Get user's groups	Yes
POST	`/:groupId/add`	Add member to group	Yes (Admin only)

Create Group

```
POST /api/v1/groups
Authorization: Bearer <token>
Content-Type: application/json

{
```

```
  "name": "Project Team",
  "members": ["64user1...", "64user2..."]
}
```

Debug Routes (`/api/v1/debug`)

Only available in development mode

Method	Endpoint	Description	Auth Required
GET	`/chats/:userId1/:userId2`	Debug chat history	Yes
POST	`/migrate-group-chatids`	Migrate group chat IDs	Yes

Health Check

`GET /health`

Response:

```
{
  "status": "OK",
  "timestamp": "2024-01-15T10:30:00.000Z"
}
```

WebSocket Events

The application uses Socket.IO for real-time communication. Connect with:

```
const socket = io('http://localhost:3001', {
  auth: { token: 'your-jwt-token' }
});
```

Client → Server Events

Event	Payload	Description
`join`	`{ chatId, recipientId }`	Join a chat room
`send_message`	`{ chatId, message, type, recipientId }`	Send a message
`typing`	`{ chatId, recipientId }`	User is typing

Event	Payload	Description
`stop_typing`	`{ chatId, recipientId }`	User stopped typing

Server → Client Events

Event	Payload	Description
`joined_chat`	`{ chatId, recipientId }`	Confirmation of joining chat
`receive_message`	Message object	New message received
`online_users`	`[userId, ...]`	List of online user IDs
`typing`	`{ userId, username }`	Someone is typing
`stop_typing`	`userId`	Someone stopped typing
`error`	`{ message }`	Error occurred

Database Models

User Model

```
{
  username: String,        // Unique, required
  passwordHash: String,    // Hashed with bcrypt (hidden by default)
  isOnline: Boolean,       // Default: false
  lastSeen: Date,          // Default: now
  createdAt: Date,
  updatedAt: Date
}
```

Chat Model

```
{
  type: String,            // 'private' or 'group'
  members: [ObjectId],     // User references
  lastMessage: String,     // Preview of last message
  lastMessageTime: Date,
  createdAt: Date,
  updatedAt: Date
}
```

Message Model

```
{
  senderId: ObjectId,       // User reference
  chatId: ObjectId,         // Chat reference
  content: String,          // Message content (encrypted)
  type: String,             // 'text' or 'file'
  fileUrl: String,          // Cloudinary URL (for files)
  fileName: String,
  fileSize: Number,
  fileType: String,
  isEncrypted: Boolean,     // Default: false
  readBy: [{                // Read receipts
    userId: ObjectId,
    readAt: Date
  }],
  createdAt: Date,
  updatedAt: Date
}
```

Group Model

```
{
  name: String,             // 2-50 characters
  description: String,      // Max 500 characters
  members: [ObjectId],      // User references
  adminId: ObjectId,        // Group creator/admin
  chatId: ObjectId,         // Associated Chat reference
  profilePicture: String,   // Optional group avatar URL
  createdAt: Date,
  updatedAt: Date
}
```

Code Documentation

Entry Point (server.js)

Purpose: Main application entry point that initializes Express, MongoDB, and Socket.IO.

What it does: 1. **Environment Setup**: Loads environment variables using `dotenv.config()` 2. **Server Creation**: Creates HTTP server with Express and Socket.IO 3. **CORS Configuration**: Configures Cross-Origin Resource Sharing for development/production 4. **Middleware Stack**: - `express.json()` - Parse JSON request bodies - `express.urlencoded()` - Parse URL-encoded bodies - Request logging middleware - CORS middleware

5. **MongoDB Connection**: Connects to MongoDB Atlas using Mongoose 6. **Route Registration**: Mounts all API routes under `/api/v1` 7. **Error Handling**: 404 handler and global error handler 8. **WebSocket Setup**: Initializes Socket.IO with authentication

```
// Key configuration
const PORT = process.env.PORT || 3001;

// Routes mounted:
app.use("/api/v1/auth", require("./routes/auth"));
app.use("/api/v1/users", require("./routes/users"));
app.use("/api/v1/groups", require("./routes/groups"));
app.use("/api/v1/chats", require("./routes/chats"));
app.use("/api/v1/debug", require("./routes/debug"));
```

Socket.IO Setup (socket-io-setup.js)

Purpose: Configures WebSocket server for real-time messaging.

What it does:

1. **Authentication Middleware**:
   - Validates JWT token from `socket.handshake.auth.token`
   - Attaches `userId` and `username` to socket instance
   - Rejects connections without valid tokens
2. **Connection Handler**:
   - Marks user as online in database
   - Broadcasts updated online users list to all clients
3. **Event Handlers**:

   `join` Event:

   - Finds or creates private chat between users
   - Joins the socket to the chat room
   - Emits `joined_chat` confirmation back to client

   `send_message` Event:

   - Validates message content
   - Finds or creates chat (private or group)
   - Encrypts message using AES-256-CBC
   - Saves message to database
   - Updates chat's last message
   - Broadcasts message to all room members

   `typing` / `stop_typing` Events:

- Broadcasts typing indicators to chat room

`disconnect` Event:

- Marks user as offline
- Updates `lastSeen` timestamp
- Broadcasts updated online users list

Configuration

cloudinary.js **Purpose:** Configures Cloudinary for cloud file storage and Multer for file uploads.

What it does: 1. **Cloudinary Setup**: Configures cloud credentials from environment variables 2. **Storage Engine**: Creates CloudinaryStorage for Multer 3. **File Limits**: - Allowed formats: `jpg, jpeg, png, gif, pdf, doc, docx, txt, mp4, mov` - Max file size: 50MB - Auto-detects resource type (image, video, raw)

```
const storage = new CloudinaryStorage({
    cloudinary: cloudinary,
    params: {
        folder: "collabchat",
        allowed_formats: ["jpg", "jpeg", "png", "gif", "pdf", "doc", "docx", "txt", "mp4", '
        resource_type: "auto"
    }
});
```

Middleware

auth.js **Purpose:** JWT authentication middleware for protected routes.

What it does: 1. Extracts Bearer token from `Authorization` header 2. Verifies token using `JWT_SECRET` 3. Attaches `userId` and `username` to request object 4. Returns 401 if token is missing or invalid

```
// Usage in routes
router.get('/protected', authMiddleware, controller);

// After middleware, access authenticated user:
req.userId   // User's MongoDB ObjectId
req.username // User's username
```

Controllers

authController.js **Purpose:** Handles user registration and login.

Functions:

`register(req, res)`: - Validates username and password - Checks for existing username - Creates new user (password auto-hashed by model) - Generates JWT token - Returns token and user object

`login(req, res)`: - Validates credentials - Finds user by username - Compares password using bcrypt - Marks user as online - Generates JWT token - Returns token and user object

---

usersController.js **Purpose:** User profile and listing operations.

Functions:

`getCurrentUser(req, res)`: - Returns authenticated user's profile - Excludes password hash from response

`getAllUsers(req, res)`: - Returns paginated list of all users - Excludes current user from results - Supports `limit` and `offset` query params

---

chatsController.js **Purpose:** Message handling and chat management.

Functions:

`sendMessage(req, res)`: - Handles both private and group messages - Creates chat if it doesn't exist - Encrypts message content - Saves message to database - Updates chat's last message preview

`getPrivateMessages(req, res)`: - Retrieves messages between two users - Creates chat if first interaction - Returns messages in chronological order - Supports pagination with `limit` and `offset`

`getGroupMessages(req, res)`: - Retrieves messages for a group - Resolves group ID to chat ID - Creates group chat atomically if needed - Returns messages in chronological order

`uploadFile(req, res)`: - Creates file message with metadata - Validates user is chat member - Updates chat's last message

`clearPrivateChat(req, res)` / `clearGroupChat(req, res)`: - Deletes all messages in a chat - Resets chat metadata

---

groupsController.js **Purpose:** Group creation and member management.

Functions:

`createGroup(req, res)`: - Creates new group with name and members - Auto-adds creator to members - Creates associated chat document - Links chat to group - Returns populated group object

`getMyGroups(req, res)`: - Returns all groups user is member of - Populates admin and member details

`addGroupMember(req, res)`: - Admin-only operation - Adds user to group and associated chat - Prevents duplicate additions

---

fileUploadController.js **Purpose:** Cloudinary file upload handling.

Functions:

`uploadFileToCloudinary(req, res)`: - Handles multipart file uploads via Multer - Uploads to Cloudinary automatically - Creates file message with URL and metadata - Broadcasts file message via WebSocket - Supports optional caption

---

debugController.js **Purpose:** Development utilities and migrations.

Functions:

`migrateGroupChatIds(req, res)`: - Finds groups without linked chatId - Creates or finds matching chat documents - Links chats to groups - Returns migration statistics

`debugChatHistory(req, res)`: - Finds all chats between two users - Returns message counts and details - Useful for debugging chat issues

---

Routes

auth.js

```
POST /api/v1/auth/register  → authController.register
POST /api/v1/auth/login     → authController.login
```

users.js

```
GET /api/v1/users/me        → usersController.getCurrentUser  [Auth]
GET /api/v1/users           → usersController.getAllUsers     [Auth]
```

chats.js

```
POST   /api/v1/chats/send                            → sendMessage         [Auth]
GET    /api/v1/chats/private/:userId1/:userId2       → getPrivateMessages  [Auth]
GET    /api/v1/chats/group/:groupId                  → getGroupMessages    [Auth]
POST   /api/v1/chats/private/:chatId/messages        → uploadFile          [Auth]
POST   /api/v1/chats/group/:chatId/messages          → uploadFile          [Auth]
POST   /api/v1/chats/upload/:chatId                  → uploadFileToCloudinary [Auth + Multer]
DELETE /api/v1/chats/private/:userId1/:userId2/clear → clearPrivateChat    [Auth]
DELETE /api/v1/chats/group/:groupId/clear            → clearGroupChat      [Auth]
```

groups.js

```
POST /api/v1/groups              → groupsController.createGroup    [Auth]
GET  /api/v1/groups/my           → groupsController.getMyGroups    [Auth]
POST /api/v1/groups/:groupId/add → groupsController.addGroupMember [Auth]
```

debug.js

```
// Only in development mode
GET  /api/v1/debug/chats/:userId1/:userId2  → debugChatHistory    [Auth]
POST /api/v1/debug/migrate-group-chatids    → migrateGroupChatIds [Auth]
```

Models

User.js **Purpose:** User schema with password hashing.

Features: - `username`: Unique identifier, trimmed - `passwordHash`: Auto-hashed using bcrypt pre-save hook - `isOnline`: Real-time online status - `lastSeen`: Last activity timestamp

Methods: - `comparePassword(candidatePassword)`: Bcrypt comparison - `toJSON()`: Removes passwordHash from serialization

```
// Pre-save hook for password hashing
userSchema.pre("save", async function (next) {
    if (!this.isModified("passwordHash")) return next();
    const salt = await bcrypt.genSalt(10);
    this.passwordHash = await bcrypt.hash(this.passwordHash, salt);
    next();
});
```

Chat.js **Purpose:** Conversation container for private and group chats.

Fields: - `type`: 'private' or 'group' - `members`: Array of User references - `lastMessage`: Last message preview (for UI) - `lastMessageTime`: For sorting conversations

---

Message.js **Purpose:** Individual message storage.

Features: - Links to sender and chat - Supports text and file types - File metadata: URL, name, size, MIME type - Encryption flag for encrypted messages - Read receipts array

Indexes: - `{ chatId: 1, createdAt: -1 }` for efficient message retrieval

Group.js **Purpose:** Group metadata and membership.

Fields: - `name`: 2-50 characters - `description`: Optional, max 500 characters - `members`: User references - `adminId`: Group creator with special permissions - `chatId`: Link to associated Chat document - `profilePicture`: Optional group avatar

---

Utilities

chatUtils.js **Purpose:** Chat-related helper functions.

Functions:

`generatePrivateChatId(userId1, userId2)`: - Creates consistent chat ID from two user IDs - Sorts IDs alphabetically before concatenating - Ensures same ID regardless of who initiates

```
// Example
generatePrivateChatId("abc123", "xyz789") // Returns "abc123_xyz789"
generatePrivateChatId("xyz789", "abc123") // Returns "abc123_xyz789" (same)
```

---

encryption.js **Purpose:** AES-256-CBC message encryption/decryption.

Functions:

`generateChatKey(chatId)`: - Creates SHA-256 hash from chatId + secret - Returns 32-byte key for AES-256

`encryptMessage(text, chatId)`: - Generates random 16-byte IV - Encrypts using AES-256-CBC - Returns `"iv:encryptedData"` format

`decryptMessage(encryptedText, chatId)`: - Extracts IV from encrypted string - Decrypts using same key derivation - Returns original plaintext

`isEncrypted(text)`: - Checks if text matches encrypted format

```
// Encryption format: "iv:encryptedData"
// Example: "a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6:f8a9b0c1d2e3f4a5b6c7d8e9f0a1b2c3"
```

Security Features

1. **Password Hashing**: Bcrypt with salt rounds of 10
2. **JWT Authentication**: Secure token-based auth with expiration
3. **Message Encryption**: AES-256-CBC end-to-end encryption
4. **CORS Protection**: Configurable origin whitelist
5. **Input Validation**: Required fields and character limits
6. **Auth Middleware**: Protected routes require valid JWT
7. **Password Exclusion**: `passwordHash` never returned in responses

Docker Support

Dockerfile

```
FROM node:16-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3001
CMD ["npm", "start"]
```

Build and Run

```
# Build image
docker build -t collabchat-backend .

# Run container
docker run -p 3001:3001 --env-file .env collabchat-backend
```

Environment Variables

Variable	Description	Required
`PORT`	Server port (default: 3001)	No

Variable	Description	Required
NODE_ENV	Environment (development/production)	No
MONGODB_URI	MongoDB connection string	Yes
JWT_SECRET	Secret key for JWT signing	Yes
JWT_EXPIRE	Token expiration (e.g., "7d")	Yes
CORS_ORIGIN	Allowed origins (comma-separated)	No
ENCRYPTION_SECRET	32-byte key for message encryption	Yes
CLOUDINARY_CLOUD_NAME	Cloudinary cloud name	Yes*
CLOUDINARY_API_KEY	Cloudinary API key	Yes*
CLOUDINARY_API_SECRET	Cloudinary API secret	Yes*

*Required for file upload functionality

NPM Scripts

```
npm start    # Start production server
npm run dev  # Start development server with nodemon
```

Documentation generated for CollabChat Backend v1.0.0