
🧩 MODULE 4 — Messages / Chat nội bộ

🎯 1️⃣ Mục tiêu nghiệp vụ

- Cho phép người dùng (trừ reviewer) gửi tin nhắn cá nhân hoặc nhóm (author ↔ editor ↔ chief editor).
- Hỗ trợ realtime (hiện tin nhắn ngay không cần reload).
- Có thông báo (notification badge hoặc popup).
- Lưu lại toàn bộ lịch sử trao đổi (audit).
- Hỗ trợ tìm kiếm người dùng để bắt đầu cuộc trò chuyện.

🧠 2️⃣ Kiến trúc đề xuất

⚙️ Công nghệ dùng

Thành phần	Công nghệ
Database	PostgreSQL (bảng messages, conversations)
Realtime	WebSocket (Next.js + Socket.IO) hoặc Pusher
Backend	Next.js App Router /api/messages/
Frontend	React Client Component + SWR + WebSocket
UI Framework	Shadcn/UI hoặc Ant Design ChatBox

🗄️ 3️⃣ Cấu trúc dữ liệu (Database)

Prisma schema gợi ý:

```
model Conversation {
  id        Int      @id @default(autoincrement())
  createdAt DateTime @default(now())
  users     User[]   @relation("ConversationUsers")
  messages  Message[]
}

model Message {
  id             Int      @id @default(autoincrement())
  senderId       Int
  conversationId  Int
  content        String
  createdAt      DateTime  @default(now())
  conversation    Conversation @relation(fields: [conversationId], references: [id])
  sender         User @relation(fields: [senderId], references: [id])
}
```

🔧 4️⃣ Backend Implementation

File: app/api/messages/route.ts

```
import prisma from "@/lib/db";
import { NextResponse } from "next/server";

// Danh sách hội thoại của user
export async function GET(req) {
  const userId = parseInt(req.headers.get("x-user-id"));
  const conversations = await prisma.conversation.findMany({
    where: { users: { some: { id: userId } } },
    include: {
      users: { select: { id: true, name: true, role: true } },
      messages: { orderBy: { createdAt: "desc" }, take: 1 },
    },
  });
  return NextResponse.json(conversations);
}

// Tạo tin nhắn mới
export async function POST(req) {
  const { conversationId, senderId, content } = await req.json();
```

```
  const message = await prisma.message.create({
    data: { conversationId, senderId, content },
    include: { sender: true },
  });
  // Gửi qua WebSocket
  globalThis.io?.to(`conversation_${conversationId}`).emit("new_message", message);
  return NextResponse.json(message);
}
```

🌐 WebSocket Realtime Layer

Tạo WebSocket server (tích hợp vào Next.js custom server hoặc Node adapter):

```
// lib/socket.js
import { Server } from "socket.io";

export function initSocket(server) {
  const io = new Server(server, { cors: { origin: "*" } });
  globalThis.io = io;

  io.on("connection", (socket) => {
```

```
  socket.on("join_conversation", (id) => socket.join(`conversation_${id}`));
  socket.on("send_message", (msg) => {
    io.to(`conversation_${msg.conversationId}`).emit("new_message", msg);
  });
 });
}
```

(Nếu deploy Vercel: dùng dịch vụ như Pusher hoặc Supabase Realtime thay Socket.io).

⚛️ 5 Frontend Implementation

File: app/dashboard/messages/page.tsx

```
"use client";
import { useState, useEffect } from "react";
import io from "socket.io-client";
import useSWR from "swr";

const fetcher = (url) => fetch(url, { headers: { "x-user-id": "1" } }).then(r => r.json());
const socket = io(); // socket kết nối server

export default function Messages() {
```

```
const { data: conversations, mutate } = useSWR("/api/messages", fetcher);
const [active, setActive] = useState(null);
const [messages, setMessages] = useState([]);
const [input, setInput] = useState("");

useEffect(() => {
  if (active) {
    socket.emit("join_conversation", active.id);
    socket.on("new_message", (msg) => {
      if (msg.conversationId === active.id) setMessages((m) => [...m, msg]);
    });
  }
  return () => socket.off("new_message");
}, [active]);

async function sendMessage() {
  await fetch("/api/messages", {
    method: "POST",
    body: JSON.stringify({
      conversationId: active.id,
```

```
      senderId: 1,
      content: input,
    }),
  });
  setInput("");
 }

 return (
  <div className="flex h-[80vh] border rounded">
    {/* Danh sách hội thoại */}
    <div className="w-1/3 border-r overflow-y-auto">
      {conversations?.map((c) => (
        <div
          key={c.id}
          onClick={() => setActive(c)}
          className={`p-3 cursor-pointer ${active?.id === c.id ? "bg-gray-100" : ""}`}
        >
          {c.users.filter((u) => u.id !== 1).map((u) => u.name).join(", ")}
          <p className="text-xs text-gray-500">
```

```
            {c.messages[0]?.content.slice(0, 30)}
          </p>
        </div>
      ))}
    </div>

    {/* Khung chat */}
    {active && (
     <div className="flex flex-col w-2/3">
      <div className="flex-1 overflow-y-auto p-3">
       {messages.map((m, i) => (
        <div key={i} className="my-1">
         <span className="text-sm font-semibold">{m.sender.name}:</span>{"
"}
         {m.content}
        </div>
       ))}
      </div>
      <div className="flex p-3 border-t">
       <input
```

```
            value={input}
            onChange={(e) => setInput(e.target.value)}
            className="flex-1 border rounded p-2"
            placeholder="Nhập tin nhắn..."
          />
          <button onClick={sendMessage} className="bg-blue-600 text-white p-2 rounded ml-2">
            Gửi
          </button>
        </div>
      </div>
    )}
  </div>
 );
}
```

🎨 6️⃣ Gợi ý giao diện

- Sidebar hiển thị danh sách hội thoại.
- Khung chat trung tâm (scrollable).
- Thanh nhập tin nhắn ở cuối.

- Notification badge 🔔 trên header khi có tin mới (sử dụng Context hoặc Zustand).

🧱 7️⃣ Bảo mật & Phân quyền

Quy tắc	Cách xử lý
Reviewer không được chat	Chặn bằng middleware: if (role === "reviewer") deny()
Người dùng chỉ thấy hội thoại mình tham gia	Prisma query filter theo userId
Lưu lịch sử đầy đủ	Bảng messages có timestamp
Chống spam	Rate limiter (tối đa 5 message/10s)

⚡ 8️⃣ Tối ưu & Mở rộng

Nâng cấp	Mô tả
🧠 Mention	@username trong khung chat
📎 File share	Upload PDF, image kèm tin nhắn
🔔 Notification	Dùng /api/notifications để gửi “Bạn có tin nhắn mới”
📱 Mobile responsive	Flex layout + Tailwind responsive class

🧱 I. MỤC TIÊU NGHIỆP VỤ

Hệ thống trao đổi & bình luận gồm **2 cấp độ giao tiếp**:

Cấp độ	Mô tả	Đối tượng
🧩 **Chat nội bộ (Private Messages)**	Tin nhắn realtime giữa các vai trò có quyền liên lạc trong hệ thống	Tác giả ↔ Biên tập ↔ Tổng biên tập ↔ (Phản biện chỉ nhận từ biên tập)
💬 **Bình luận công khai (Public Comments)**	Độc giả hoặc người dùng bình thường bình luận công khai dưới mỗi bài viết	Độc giả ↔ Bài báo

---

🧩 II. PHÂN QUYỀN CHAT NỘI BỘ

🎯 Quy tắc luồng trao đổi

Người gửi	Có thể chat với	Ghi chú
Tác giả (Author)	Biên tập (Editor), Tổng biên tập (Chief Editor), Tác giả khác	❌ Không chat với phản biện
Biên tập (Editor)	Tác giả, Tổng biên tập, Phản biện	Có thể gửi phản hồi đến phản biện
Phản biện (Reviewer)	Biên tập, Tổng biên tập	❌ Không chat với tác giả
Tổng biên tập (Chief Editor)	Tất cả (trừ độc giả)	Có quyền điều phối mọi hội thoại

Người gửi	Có thể chat với	Ghi chú
Độc giả (Reader)	❌ Không có chat nội bộ	Chỉ được bình luận công khai

---

🧩 III. CẤU TRÚC DỮ LIỆU (Prisma ORM)

```
model Conversation {
  id        Int      @id @default(autoincrement())
  createdAt DateTime @default(now())
  type      String   // "private" | "group" | "system"
  messages  Message[]
  users     User[]   @relation("ConversationUsers")
}

model Message {
  id             Int      @id @default(autoincrement())
  conversationId Int
  senderId       Int
  content        String
  createdAt      DateTime  @default(now())
  conversation   Conversation @relation(fields: [conversationId], references: [id])
```

```
  sender        User @relation(fields: [senderId], references: [id])
}

model Comment {
  id        Int      @id @default(autoincrement())
  articleId Int
  userId    Int?
  content   String
  createdAt DateTime @default(now())
  article   Article @relation(fields: [articleId], references: [id])
  user      User? @relation(fields: [userId], references: [id])
}
```

⚙️ IV. API BACKEND

1️⃣ Chat nội bộ – /api/messages/route.ts

GET – lấy danh sách hội thoại hợp lệ theo quyền

```
export async function GET(req) {
  const userId = parseInt(req.headers.get("x-user-id"));
  const user = await prisma.user.findUnique({ where: { id: userId } });
```

```
const roleMatrix = {
  author: ["editor", "chief_editor", "author"],
  editor: ["author", "reviewer", "chief_editor"],
  reviewer: ["editor", "chief_editor"],
  chief_editor: ["author", "editor", "reviewer"],
};

const allowedRoles = roleMatrix[user.role] || [];

const conversations = await prisma.conversation.findMany({
  where: {
    users: {
      some: {
        role: { in: allowedRoles },
      },
    },
  },
  include: {
    users: { select: { id: true, name: true, role: true } },
    messages: { orderBy: { createdAt: "desc" }, take: 1 },
```

```
    },
  });

  return NextResponse.json(conversations);
}
```

POST – gửi tin nhắn (có kiểm tra quyền)

```
export async function POST(req) {
  const { conversationId, senderId, content } = await req.json();
  const sender = await prisma.user.findUnique({ where: { id: senderId } });
  const conv = await prisma.conversation.findUnique({
    where: { id: conversationId },
    include: { users: true },
  });

  const allowedRoles = {
    author: ["editor", "chief_editor", "author"],
    editor: ["author", "reviewer", "chief_editor"],
    reviewer: ["editor", "chief_editor"],
    chief_editor: ["author", "editor", "reviewer"],
```

```
  };

  // kiểm tra quyền người gửi trong hội thoại
  const valid = conv.users.every((u) =>
    allowedRoles[sender.role]?.includes(u.role)
  );
  if (!valid)
    return NextResponse.json({ success: false, message: "Not allowed" });

  const msg = await prisma.message.create({
    data: { conversationId, senderId, content },
    include: { sender: true },
  });
  globalThis.io?.to(`conversation_${conversationId}`).emit("new_message", msg);
  return NextResponse.json(msg);
}
```

---

2 Bình luận công khai – /api/comments/route.ts

```
export async function GET(req) {
  const articleId = parseInt(req.nextUrl.searchParams.get("articleId"));
```

```
  const comments = await prisma.comment.findMany({
    where: { articleId },
    include: { user: { select: { name: true } } },
    orderBy: { createdAt: "asc" },
  });
  return NextResponse.json(comments);
}

export async function POST(req) {
  const { articleId, userId, content } = await req.json();
  const comment = await prisma.comment.create({
    data: { articleId, userId, content },
    include: { user: { select: { name: true } } },
  });
  globalThis.io?.emit(`comment_${articleId}`, comment);
  return NextResponse.json(comment);
}
```

⚛️ V. FRONTEND TRIỂN KHAI

1️⃣ Giao diện Chat nội bộ

📁 /app/dashboard/messages/page.tsx

```
"use client";
import io from "socket.io-client";
import useSWR from "swr";
import { useState, useEffect } from "react";

const fetcher = (u) => fetch(u, { headers: { "x-user-id": "1" } }).then((r) => r.json());
const socket = io();

export default function Messages() {
  const { data: convs, mutate } = useSWR("/api/messages", fetcher);
  const [active, setActive] = useState(null);
  const [msgs, setMsgs] = useState([]);
  const [input, setInput] = useState("");

  useEffect(() => {
    if (active) {
      socket.emit("join_conversation", active.id);
      socket.on("new_message", (msg) => {
        if (msg.conversationId === active.id)
```

```
        setMsgs((prev) => [...prev, msg]);
    });
  }
  return () => socket.off("new_message");
}, [active]);

async function send() {
  await fetch("/api/messages", {
    method: "POST",
    body: JSON.stringify({
      conversationId: active.id,
      senderId: 1,
      content: input,
    }),
  });
  setInput("");
}

return (
  <div className="flex h-[80vh] border rounded">
```

```
<div className="w-1/3 border-r overflow-y-auto">
  {convs?.map((c) => (
    <div key={c.id} onClick={() => setActive(c)} className="p-3 cursor-pointer border-b hover:bg-gray-100">
      {c.users.filter((u) => u.id !== 1).map((u) => u.name).join(", ")}
    </div>
  ))}
</div>

{active && (
  <div className="flex flex-col w-2/3">
    <div className="flex-1 p-4 overflow-y-auto">
      {msgs.map((m, i) => (
        <div key={i} className="my-1">
          <strong>{m.sender.name}:</strong> {m.content}
        </div>
      ))}
    </div>
    <div className="flex border-t p-2">
      <input
```

```
            className="flex-1 border p-2 rounded"
            placeholder="Nhập tin nhắn..."
            value={input}
            onChange={(e) => setInput(e.target.value)}
          />
          <button onClick={send} className="bg-blue-600 text-white p-2 rounded ml-2">
            Gửi
          </button>
        </div>
      </div>
    )}
  </div>
 );
}
```

2️⃣ Giao diện bình luận công khai

📁 /app/articles/[slug]/page.tsx

```
"use client";
import { useState, useEffect } from "react";
```

```
import io from "socket.io-client";
const socket = io();

export default function ArticleComments({ articleId }) {
  const [comments, setComments] = useState([]);
  const [input, setInput] = useState("");

  useEffect(() => {
    fetch(`/api/comments?articleId=${articleId}`)
      .then((r) => r.json())
      .then(setComments);
    socket.on(`comment_${articleId}`, (c) => setComments((p) => [...p, c]));
    return () => socket.off(`comment_${articleId}`);
  }, []);

  async function sendComment() {
    await fetch("/api/comments", {
      method: "POST",
      body: JSON.stringify({ articleId, userId: 1, content: input }),
    });
```

```
  setInput("");
}

return (
  <div className="mt-8">
    <h3 className="font-semibold text-lg mb-2">Bình luận</h3>
    <div className="mb-4">
      {comments.map((c, i) => (
        <div key={i} className="border-b py-2">
          <p><b>{c.user?.name || "Ẩn danh"}:</b> {c.content}</p>
        </div>
      ))}
    </div>
    <textarea
      value={input}
      onChange={(e) => setInput(e.target.value)}
      className="w-full border p-2 rounded"
      placeholder="Nhập bình luận..."
    />
```

```
    <button onClick={sendComment} className="bg-blue-600 text-white p-2 rounded mt-2">
      Gửi bình luận
    </button>
  </div>
 );
}
```

🔐 VI. KIỂM SOÁT QUYỀN CHAT

Middleware /lib/chat-guard.ts

```
export const roleMatrix = {
 author: ["editor", "chief_editor", "author"],
 editor: ["author", "reviewer", "chief_editor"],
 reviewer: ["editor", "chief_editor"],
 chief_editor: ["author", "editor", "reviewer"],
};

export function canChat(senderRole, receiverRole) {
 return roleMatrix[senderRole]?.includes(receiverRole);
}
```

Trước khi tạo cuộc trò chuyện hoặc gửi tin nhắn, gọi:

```
if (!canChat(sender.role, receiver.role)) throw new Error("Not allowed");
```

🔔 VII. THÔNG BÁO (Notifications)

Mỗi khi có tin nhắn mới hoặc bình luận:

```
await prisma.notification.create({
  data: {
    userId: receiverId,
    type: "message",
    message: `Bạn có tin nhắn mới từ ${sender.name}`,
    link: `/dashboard/messages/${conversationId}`,
  },
});
```

UI có thể hiển thị biểu tượng 🔔 với countUnreadMessages.