Berikut adalah lanjutan dokumen yang **mulai tepat dari baris yang terpotong** pada conversations.js. Saya sertakan juga berkas backend lain yang relevan (routes/messages, Prisma schema, seed, .env.example, dan README backend) agar proyek langsung jalan.

**Path:** ``

import { Router } from 'express'  
import { PrismaClient } from '@prisma/client'  
import { requireAuth } from '../middleware/auth.js'  
import { upload } from '../app.js'  
  
const prisma = new PrismaClient()  
const router = Router()  
  
// List my conversations with last message preview  
router.get('/', requireAuth, async (req, res) => {  
 const me = req.user.id  
 const parts = await prisma.participant.findMany({  
 where: { userId: me },  
 include: {  
 conversation: {  
 include: {  
 messages: { orderBy: { createdAt: 'desc' }, take: 1 },  
 participants: {  
 include: { user: { select: { id: true, username: true, name: true, avatarUrl: true } } }  
 }  
 }  
 }  
 },  
 orderBy: { conversation: { lastMessageAt: 'desc' } }  
 })  
  
 const items = parts.map((p) => ({  
 id: p.conversationId,  
 isGroup: p.conversation.isGroup,  
 title: p.conversation.title,  
 participants: p.conversation.participants.map((pp) => pp.user),  
 lastMessage: p.conversation.messages[0] || null,  
 updatedAt: p.conversation.updatedAt  
 }))  
  
 res.json(items)  
})  
  
// Create or get 1-1 conversation  
router.post('/start', requireAuth, async (req, res) => {  
 const me = req.user.id  
 const { peerId } = req.body  
 if (!peerId) return res.status(400).json({ error: 'peerId required' })  
 if (peerId === me) return res.status(400).json({ error: "Can't chat with yourself" })  
  
 const existing = await prisma.conversation.findFirst({  
 where: {  
 isGroup: false,  
 AND: [  
 { participants: { some: { userId: me } } },  
 { participants: { some: { userId: peerId } } }  
 ]  
 }  
 })  
 if (existing) return res.json({ id: existing.id })  
  
 const conv = await prisma.conversation.create({  
 data: {  
 isGroup: false,  
 participants: { create: [{ userId: me }, { userId: peerId }] },  
 lastMessageAt: new Date()  
 }  
 })  
 res.json({ id: conv.id })  
})  
  
// (Optional) Create group conversation  
router.post('/group', requireAuth, async (req, res) => {  
 const me = req.user.id  
 const { title, memberIds } = req.body // memberIds: string[] (tanpa me)  
 if (!title || !Array.isArray(memberIds) || memberIds.length === 0) {  
 return res.status(400).json({ error: 'title and memberIds required' })  
 }  
 const allMembers = Array.from(new Set([me, ...memberIds]))  
 const conv = await prisma.conversation.create({  
 data: {  
 isGroup: true,  
 title,  
 participants: { create: allMembers.map((uid) => ({ userId: uid })) },  
 lastMessageAt: new Date()  
 }  
 })  
 res.json({ id: conv.id })  
})  
  
// Upload image for a message (helper): returns URL to include in message  
router.post('/:id/upload', requireAuth, upload.single('image'), async (req, res) => {  
 const conversationId = req.params.id  
 if (!req.file) return res.status(400).json({ error: 'No file' })  
 // Verify participant  
 const member = await prisma.participant.findFirst({ where: { conversationId, userId: req.user.id } })  
 if (!member) return res.status(403).json({ error: 'Forbidden' })  
 const url = `/uploads/${req.file.filename}`  
 res.json({ imageUrl: url, conversationId })  
})  
  
export default router

**Path:** ``

import { Router } from 'express'  
import { PrismaClient } from '@prisma/client'  
import { requireAuth } from '../middleware/auth.js'  
  
const prisma = new PrismaClient()  
const router = Router()  
  
// Get messages with cursor pagination by id (newest first in DB, return ascending)  
router.get('/:conversationId', requireAuth, async (req, res) => {  
 const me = req.user.id  
 const { conversationId } = req.params  
 const limit = Math.min(parseInt(req.query.limit || '20', 10), 50)  
 const cursor = req.query.cursor?.toString() // message id  
  
 // Ensure membership  
 const member = await prisma.participant.findFirst({ where: { conversationId, userId: me } })  
 if (!member) return res.status(403).json({ error: 'Forbidden' })  
  
 const items = await prisma.message.findMany({  
 where: { conversationId },  
 orderBy: { createdAt: 'desc' },  
 take: limit,  
 ...(cursor ? { skip: 1, cursor: { id: cursor } } : {})  
 })  
  
 // Compute next cursor  
 const nextCursor = items.length === limit ? items[items.length - 1].id : null  
 // Return ascending for UI  
 res.json({ items: items.slice().reverse(), nextCursor })  
})  
  
// Create message via REST (Socket also supported)  
router.post('/', requireAuth, async (req, res) => {  
 const me = req.user.id  
 const { conversationId, content, imageUrl } = req.body  
 if (!conversationId || (!content && !imageUrl)) return res.status(400).json({ error: 'Missing fields' })  
  
 // Ensure membership  
 const member = await prisma.participant.findFirst({ where: { conversationId, userId: me } })  
 if (!member) return res.status(403).json({ error: 'Forbidden' })  
  
 const msg = await prisma.message.create({  
 data: { conversationId, senderId: me, content: content || null, imageUrl: imageUrl || null }  
 })  
 await prisma.conversation.update({ where: { id: conversationId }, data: { lastMessageAt: new Date() } })  
 res.json(msg)  
})  
  
export default router

**Path:** ``

// Prisma schema for PostgreSQL  
// Run: npx prisma migrate dev --name init  
  
generator client {  
 provider = "prisma-client-js"  
}  
  
datasource db {  
 provider = "postgresql"  
 url = env("DATABASE\_URL")  
}  
  
model User {  
 id String @id @default(uuid())  
 email String @unique  
 username String @unique  
 name String  
 passwordHash String  
 avatarUrl String?  
 createdAt DateTime @default(now())  
 updatedAt DateTime @updatedAt  
  
 participants Participant[]  
 messages Message[] @relation("UserMessages")  
}  
  
model Conversation {  
 id String @id @default(uuid())  
 isGroup Boolean @default(false)  
 title String?  
 lastMessageAt DateTime? @db.Timestamptz(6)  
 createdAt DateTime @default(now())  
 updatedAt DateTime @updatedAt  
  
 participants Participant[]  
 messages Message[]  
  
 @@index([lastMessageAt])  
}  
  
model Participant {  
 id String @id @default(uuid())  
 userId String  
 conversationId String  
 joinedAt DateTime @default(now())  
  
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
 conversation Conversation @relation(fields: [conversationId], references: [id], onDelete: Cascade)  
  
 @@unique([userId, conversationId])  
 @@index([conversationId])  
 @@index([userId])  
}  
  
model Message {  
 id String @id @default(uuid())  
 conversationId String  
 senderId String  
 content String?  
 imageUrl String?  
 createdAt DateTime @default(now())  
  
 conversation Conversation @relation(fields: [conversationId], references: [id], onDelete: Cascade)  
 sender User @relation("UserMessages", fields: [senderId], references: [id], onDelete: Cascade)  
 statuses MessageStatus[]  
  
 @@index([conversationId, createdAt])  
}  
  
model MessageStatus {  
 id String @id @default(uuid())  
 messageId String  
 userId String  
 status Status @default(SENT)  
 readAt DateTime?  
 deliveredAt DateTime?  
  
 message Message @relation(fields: [messageId], references: [id], onDelete: Cascade)  
 user User @relation(fields: [userId], references: [id], onDelete: Cascade)  
  
 @@unique([messageId, userId])  
 @@index([userId])  
}  
  
enum Status {  
 SENT  
 DELIVERED  
 READ  
}

**Path:** ``

import { PrismaClient } from '@prisma/client'  
import bcrypt from 'bcrypt'  
  
const prisma = new PrismaClient()  
  
async function main() {  
 const alicePass = await bcrypt.hash('password123', 10)  
 const bobPass = await bcrypt.hash('password123', 10)  
  
 const alice = await prisma.user.upsert({  
 where: { email: 'alice@example.com' },  
 update: {},  
 create: {  
 email: 'alice@example.com',  
 username: 'alice',  
 name: 'Alice',  
 passwordHash: alicePass  
 }  
 })  
  
 const bob = await prisma.user.upsert({  
 where: { email: 'bob@example.com' },  
 update: {},  
 create: {  
 email: 'bob@example.com',  
 username: 'bob',  
 name: 'Bob',  
 passwordHash: bobPass  
 }  
 })  
  
 // Create or get 1-1 conversation  
 let conv = await prisma.conversation.findFirst({  
 where: {  
 isGroup: false,  
 AND: [  
 { participants: { some: { userId: alice.id } } },  
 { participants: { some: { userId: bob.id } } }  
 ]  
 }  
 })  
  
 if (!conv) {  
 conv = await prisma.conversation.create({  
 data: {  
 isGroup: false,  
 participants: { create: [{ userId: alice.id }, { userId: bob.id }] },  
 lastMessageAt: new Date()  
 }  
 })  
 }  
  
 // Seed a couple messages  
 await prisma.message.createMany({  
 data: [  
 { conversationId: conv.id, senderId: alice.id, content: 'Hi Bob! 👋' },  
 { conversationId: conv.id, senderId: bob.id, content: 'Hey Alice! Welcome to Chat‑Lite.' }  
 ]  
 })  
  
 console.log('Seed done:', { alice: alice.email, bob: bob.email, conv: conv.id })  
}  
  
main().catch((e) => {  
 console.error(e)  
 process.exit(1)  
}).finally(async () => {  
 await prisma.$disconnect()  
})

**Path:** ``

# PostgreSQL connection string  
# Format: postgresql://USER:PASSWORD@HOST:PORT/DBNAME?schema=public  
DATABASE\_URL="postgresql://postgres:postgres@localhost:5432/chatlite?schema=public"  
  
# API  
PORT=4000  
CORS\_ORIGIN=http://localhost:5173  
  
# Auth  
JWT\_SECRET=replace-with-strong-secret  
  
# Uploads (local folder)  
UPLOAD\_DIR=uploads

**Path:** ``

# Chat‑Lite — Backend  
  
## Setup Cepat  
  
```bash  
cp .env.example .env  
# Edit DATABASE\_URL bila perlu  
pnpm i  
npx prisma migrate dev --name init  
pnpm seed  
pnpm dev

### Endpoint ringkas

* POST /api/auth/register — { email, username, password, name }
* POST /api/auth/login — { emailOrUsername, password }
* GET /api/users/me — butuh Bearer token
* GET /api/users/search?q=alice
* GET /api/conversations — list percakapan saya
* POST /api/conversations/start — { peerId }
* POST /api/conversations/:id/upload — form-data image
* GET /api/messages/:conversationId?cursor=<messageId>&limit=20
* POST /api/messages — { conversationId, content?, imageUrl? }

### WebSocket (Socket.IO)

* auth via `io(aut