# **ChatGPT-Style Chat App – Qoder Build Spec (Ready-to-Generate)**

**Objetivo docente**: crear un **frontend de chat** (Next.js 14 + TS + Tailwind + next-intl) que se conecta por HTTP a uno o varios **webhooks de n8n**. El contrato de integración y los textos de UI son **obligatorios** para asegurar que el resultado se vea y funcione como el demo.

## **0) Reglas para el generador (Qoder / Trae / Windsurf / Cursor / etc.)**

* **No sustituyas** Next.js por otros frameworks.
* **Usa estas versiones exactas** (sin rangos):  
   next@14.0.4, react@18.2.0, typescript@5.3.3, prisma@5.7.0, next-auth@4.24.5, next-intl@3.4.0, tailwindcss@3.3.6, pnpm@8.
* **Nombre de la app**: leer desde process.env.NEXT\_PUBLIC\_APP\_NAME y mostrarlo en el header y en el <title>.
* **Streaming obligatorio**: la respuesta al cliente es **SSE** con eventos normalizados (abajo).
* **Mapping n8n obligatorio**: request con **chatInput**; respuesta con **output** (puede incluir sources, usage).
* **Textos de UI obligatorios** en español (es-ES).
* **Atributos de prueba obligatorios**: data-testid="new-conversation|sidebar-search|chat-input|send-button|message-user|message-assistant|view-sources".

## **1) Overview**

Aplicación web tipo ChatGPT con:

* Interfaz moderna de chat.
* **Integración con n8n** para RAG/automatizaciones.
* **Streaming en tiempo real** vía **SSE**.
* Gestión de conversaciones (persistencia mínima para demo; Prisma lista para producción).
* Observabilidad básica.

### **Público objetivo**

* Usuarios hispanohablantes.
* Soporte multilenguaje.
* Admins y desarrolladores que integran con flujos n8n.

### **Idiomas**

* Primario: **es-ES**
* Secundarios: **en-US**, **es-CO**
* Extensible.

## **2) Stack**

### **Frontend**

Next.js 14 (App Router) + TypeScript 5

Tailwind CSS 3 + shadcn/ui + lucide-react + Framer Motion

next-intl (i18n)

### **Backend (en el mismo Next)**

API Route Handlers (Next.js)

Prisma ORM 5 + PostgreSQL 15 (producción)

NextAuth 4 (Credentials + Google opcional)

### **Dev & Calidad**

Vitest, Playwright, ESLint+Prettier, Husky, Docker

**Engines**: node >= 18.19.0, pnpm >= 8.  
 **Instalación**: pnpm i --frozen-lockfile.

## **3) Arquitectura**

### **Diagrama alto nivel**

graph TB

subgraph Client

UI[React UI] --> STREAM[SSE Client]

end

subgraph Next.js

PAGES[App Router]

API[/api/\*]

AUTH[NextAuth]

end

subgraph Data

PRISMA[Prisma]

DB[(PostgreSQL)]

end

subgraph External

N8N[n8n Webhooks]

RAG[RAG/LLM]

end

UI-->PAGES-->API-->N8N-->RAG

API-->PRISMA-->DB

API-->STREAM-->UI

AUTH-->API

### **Capas**

* **Presentación**: Chat, Sidebar, Settings, i18n.
* **Lógica**: ChatService, ConversationService, N8nClient.
* **Datos**: repos de usuario, conversación y mensaje.
* **Infra**: config, logger.

## **4) Frontend**

### **Jerarquía de componentes**

AppLayout

├─ (auth)/login

├─ ChatLayout

│ ├─ ConversationSidebar

│ ├─ ChatArea

│ │ ├─ MessageList → MessageBubble

│ │ └─ ChatInput

│ └─ ChatHeader → SettingsModal

└─ (auth)/register (stub)

### **Estado (tipos guía)**

interface AppState {

user: User | null;

theme: 'light'|'dark';

locale: 'es-ES'|'en-US'|'es-CO';

conversations: Conversation[];

activeConversationId: string | null;

}

interface ChatState {

messages: Message[];

isLoading: boolean;

streamingMessage: string;

error: string | null;

sources: Source[];

}

### **Rutas e i18n**

/ → redirige a /es-ES/chat

/[locale]/chat → interfaz por defecto

/[locale]/chat/[id] → conversación concreta

/api/auth/[...nextauth] → NextAuth

/api/chat/send → endpoint SSE

/api/conversations → CRUD mínimo

**Middleware i18n**

* Locales: ['es-ES','en-US','es-CO']
* defaultLocale: 'es-ES'
* matcher: ['/', '/(es-ES|en-US|es-CO)/:path\*']

### **UI & textos obligatorios**

* Sidebar: **“Nueva Conversación”**, **“Buscar conversaciones”**, **“Conversaciones”**.
* Placeholder input: **“Escribe tu mensaje aquí…”**.
* Botones: **“Enviar”**, **“Regenerar”**, **“Ver Fuentes”**.
* Estados: **“Escribiendo…”**, **“Pensando…”**.

### **data-testid obligatorios**

new-conversation, sidebar-search, chat-input, send-button,  
 message-user, message-assistant, view-sources.

## **5) Backend (Next API)**

### **Contrato SSE ← (obligatorio)**

* **Request**: POST /api/chat/send con JSON:

{

"message": "texto del usuario",

"conversationId": null,

"settings": { "topK": 5, "temperature": 0.7 }

}

* **Respuesta**: Content-Type: text/event-stream con **eventos** (un evento por línea data:):

{"type":"message","data":{"content":"…chunk…"}}

{"type":"sources","data":{"sources":[{"title":"...","url":"...","snippet":"..."}]}}

{"type":"usage","data":{"usage":{"input":0,"output":0}}}

{"type":"complete","data":{"ok":true}}

* En caso de error:

{"type":"error","data":{"message":"detalle","code":"N8N\_SERVICE\_ERROR"}}

Si n8n responde **en bloque**, el servidor **re-emite** SSE troceando output en **600–800 caracteres** por type:"message" hasta completar.

### **Integración n8n (mapping fijo)**

* **ENV**:

N8N\_BASE\_URL="https://hooks.tu-dominio.com"

N8N\_WEBHOOK\_PATH="/webhook/tu-flujo"

N8N\_API\_KEY="" # opcional

* **Request al webhook** (desde el backend):

{

"chatInput": "texto del usuario",

"topK": 5,

"temperature": 0.7,

"history": [

{"role":"USER","content":"..."} // opcional

],

"metadata": {"source":"webapp","appVersion":"1.0.0"}

}

* **Respuesta esperada de n8n**:

{

"output": "respuesta completa en texto",

"sources": [{"title":"...","url":"...","snippet":"..."}],

"usage": {"tokensInput":0,"tokensOutput":0}

}

### **Esqueleto de handler (guía para el generador)**

// app/api/chat/send/route.ts

import { NextRequest } from 'next/server';

export const runtime = 'nodejs'; // SSE estable

export async function POST(req: NextRequest) {

const { message, settings } = await req.json();

const res = await fetch(process.env.N8N\_BASE\_URL + process.env.N8N\_WEBHOOK\_PATH, {

method: 'POST',

headers: {

'Content-Type': 'application/json',

...(process.env.N8N\_API\_KEY ? { Authorization: `Bearer ${process.env.N8N\_API\_KEY}` } : {})

},

body: JSON.stringify({

chatInput: message,

topK: settings?.topK ?? 5,

temperature: settings?.temperature ?? 0.7

})

});

if (!res.ok) {

const text = await res.text();

return new Response(`data: ${JSON.stringify({type:'error',data:{message:text}})}\n\n`, {

headers: {'Content-Type':'text/event-stream'}

});

}

const data = await res.json(); // bloque: { output, sources?, usage? }

const encoder = new TextEncoder();

const stream = new ReadableStream({

async start(controller) {

const push = (obj: any) => controller.enqueue(encoder.encode(`data: ${JSON.stringify(obj)}\n\n`));

// trocear output

const chunks = data.output.match(/[\s\S]{1,750}/g) ?? [];

for (const c of chunks) push({type:'message', data:{content:c}});

if (data.sources) push({type:'sources', data:{sources:data.sources}});

if (data.usage) push({type:'usage', data:{usage:data.usage}});

push({type:'complete', data:{ok:true}});

controller.close();

}

});

return new Response(stream, { headers: { 'Content-Type': 'text/event-stream', 'Cache-Control': 'no-cache' }});

}

## **6) Modelos de datos (Prisma) — versión mínima**

Para el taller basta persistencia en memoria; Prisma queda listo para producción.

model User {

id String @id @default(cuid())

email String @unique

name String?

image String?

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

sessions Session[]

conversations Conversation[]

userRoles UserRole[]

@@map("users")

}

model Conversation {

id String @id @default(cuid())

title String?

settings Json?

userId String

createdAt DateTime @default(now())

updatedAt DateTime @updatedAt

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

messages Message[]

@@index([userId, createdAt])

@@map("conversations")

}

model Message {

id String @id @default(cuid())

content String

role MessageRole

conversationId String

parentId String?

sources Json?

usage Json?

metadata Json?

createdAt DateTime @default(now())

conversation Conversation @relation(fields: [conversationId], references: [id], onDelete: Cascade)

parent Message? @relation("MessageThread", fields: [parentId], references: [id])

children Message[] @relation("MessageThread")

@@index([conversationId, createdAt])

@@map("messages")

}

enum MessageRole { USER ASSISTANT SYSTEM }

model Session {

id String @id @default(cuid())

sessionToken String @unique

userId String

expires DateTime

user User @relation(fields: [userId], references: [id], onDelete: Cascade)

@@map("sessions")

}

## **7) Configuración & Seguridad**

### **.env.example**

# Branding

NEXT\_PUBLIC\_APP\_NAME=MiChat

# NextAuth (demo)

NEXTAUTH\_URL=http://localhost:3000

NEXTAUTH\_SECRET=dev-secret

# n8n

N8N\_BASE\_URL=http://localhost:5678

N8N\_WEBHOOK\_PATH=/webhook/rag-chat

N8N\_API\_KEY=

# i18n

DEFAULT\_LOCALE=es-ES

SUPPORTED\_LOCALES=es-ES,en-US,es-CO

# DB (prod)

DATABASE\_URL=postgresql://user:password@localhost:5432/app

NODE\_ENV=development

LOG\_LEVEL=info

APP\_VERSION=1.0.0

### **CSP (añadir n8n a connectSrc)**

export const securityHeaders = {

contentSecurityPolicy: {

directives: {

defaultSrc: ["'self'"],

scriptSrc: ["'self'", "'unsafe-eval'", "'unsafe-inline'"],

styleSrc: ["'self'", "'unsafe-inline'"],

imgSrc: ["'self'", "data:", "https:"],

connectSrc: ["'self'", process.env.N8N\_BASE\_URL],

fontSrc: ["'self'", "https:"],

objectSrc: ["'none'"],

frameSrc: ["'none'"]

}

}

};

## **8) Estilos (Tailwind)**

Paleta clara/oscura mínima como en tu doc; usar shadcn/ui para inputs, sheets y modal de **“Ver Fuentes”**.

## **9) Tests (Playwright)**

test('flujo de chat', async ({ page }) => {

await page.goto('/es-ES/chat');

await page.getByTestId('new-conversation').click();

await page.getByTestId('chat-input').fill('Hola');

await page.getByTestId('send-button').click();

await expect(page.getByTestId('message-assistant')).toBeVisible();

});

## **10) Docker (producción simple, standalone)**

**Dockerfile**

FROM node:18-alpine AS builder

WORKDIR /app

RUN npm i -g pnpm@8

COPY package.json pnpm-lock.yaml ./

RUN pnpm i --frozen-lockfile

COPY . .

RUN NEXT\_TELEMETRY\_DISABLED=1 pnpm build

FROM node:18-alpine AS runner

WORKDIR /app

ENV NODE\_ENV=production

COPY --from=builder /app/.next/standalone ./

COPY --from=builder /app/.next/static ./.next/static

COPY --from=builder /app/public ./public

EXPOSE 3000

CMD ["node","server.js"]

## **11) n8n – Flujo mínimo (plantilla)**

1. **Webhook (POST)** → recibe { chatInput, topK, temperature }.
2. **Nodo LLM/RAG** → produce answer (string) y opcional sources.
3. **Function** (formatea salida):

const answer = $json.answer || $json.output || '';

return [{ output: answer, sources: $json.sources || [], usage: $json.usage || { tokensInput:0, tokensOutput:0 } }];

1. **Respond to Webhook** → JSON de la Function anterior.

**Open WebUI (opcional)**: crear **PIPE** con:  
 N8N Url=<tu webhook>, Input Field=chatInput, Response Field=output, Emit Interval=2.

## **12) Criterios de aceptación (rúbrica)**

* URL inicial redirige a **/es-ES/chat**.
* Botón **“Nueva Conversación”** crea y selecciona hilo.
* Placeholder exacto **“Escribe tu mensaje aquí…”**.
* **Streaming** visible mediante SSE (message → chunks y complete al final).
* Panel **“Ver Fuentes”** muestra title, url, snippet cuando existan.
* data-testid presentes y test E2E pasa.
* Variables .env completas; build y pnpm dev funcionan.

## **13) Package.json (scripts clave)**

{

"scripts": {

"dev": "next dev",

"build": "next build",

"start": "next start",

"type-check": "tsc --noEmit",

"test:e2e": "playwright test"

},

"engines": { "node": ">=18.19.0", "pnpm": ">=8.0.0" }

}

## **14) Paso a paso para estudiantes**

1. **Elige el nombre** y ponlo en .env.local → NEXT\_PUBLIC\_APP\_NAME=TuNombre.
2. pnpm i --frozen-lockfile → pnpm dev.
3. Crea el **webhook** en n8n (sección 11) y ajusta N8N\_BASE\_URL y N8N\_WEBHOOK\_PATH.
4. Abre /es-ES/chat, crea conversación y envía “Hola”.
5. Verifica streaming y (si hay) **Fuentes**.
6. Corre el test E2E.

### **Apéndice A — Strings i18n (es-ES.json)**

{

"chat": {

"interface": {

"newConversation": "Nueva Conversación",

"placeholder": "Escribe tu mensaje aquí...",

"send": "Enviar",

"regenerate": "Regenerar",

"copy": "Copiar",

"edit": "Editar",

"delete": "Eliminar",

"viewSources": "Ver Fuentes",

"typing": "Escribiendo...",

"thinking": "Pensando..."

},

"sidebar": {

"conversations": "Conversaciones",

"search": "Buscar conversaciones",

"noConversations": "No hay conversaciones aún",

"deleteConfirm": "¿Estás seguro de que quieres eliminar esta conversación?"

},

"settings": {

"title": "Configuración",

"ragSettings": "Configuración de RAG",

"topK": "Número de fuentes (Top-K)",

"temperature": "Creatividad",

"language": "Idioma",

"theme": "Tema",

"save": "Guardar",

"cancel": "Cancelar"

}

},

"common": {

"loading": "Cargando...",

"error": "Error",

"retry": "Reintentar",

"close": "Cerrar",

"save": "Guardar",

"cancel": "Cancelar",

"confirm": "Confirmar",

"yes": "Sí",

"no": "No"

}

}

## **Notas finales (cosas que suelen fallar)**

* Añade **process.env.N8N\_BASE\_URL** a connectSrc en CSP o el fetch/SSE no conecta.
* Asegura Content-Type: text/event-stream y **no comprimir** SSE.
* Si el webhook cambia las claves, **ajusta solo el mapeo**, no el contrato SSE.
* En Windows, la carpeta de mensajes es messages/es-ES.json (con guion).