Sam_MiniBot Documentation

Project Architecture

|-- .firebaserc |-- .gitignore |-- .nvmrc |-- chat-test.json |-- env.txt |-- firebase.json |-- firestore-debug.log |-- firestore.indexes.json |-- firestore.rules |-- package.json |-- test.json +-- functions | |-- package.json | |-- tsconfig.json | +-- src | | |-- chatbot.ts | | |-- index.ts | | |-- loadClientConfig.ts | | |-- requestMiniBot.ts | | +-- config | | | |-- env.ts | | +-- providers | | | |-- deepseek.provider.ts | | | |-- gemini.provider.ts | | | |-- llm.types.ts | | | |-- openai.provider.ts | | +-- services | | | |-- context.service.ts | | | |-- faq.service.ts | | | |-- llm.service.ts +-- LoadData | |-- data-loader.prod.ps1 | |-- load-client.js | |-- sam-minibot.initial.clean.json | |-- sam-minibot.initial.json +-- public | |-- favicon.ico | |-- favicon512.ico | |-- index.html | |-- manifest.json | |-- robots.txt +-- scripts | |-- load-gpt-config.prod.ps1 | |-- load-gpt-config.ps1 | |-- load-initial-client.prod.ps1 | |-- load-initial-client.ps1 | |-- set-firebase-config.ps1 +-- src | |-- App.css | |-- App.tsx | |-- firebase.ts | |-- index.css | |-- index.tsx | |-- setupTests.ts | +-- components | | |-- ChatBot.tsx | | |-- ChatInput.tsx | | |-- ChatMessage.tsx | | +-- landing | | | |-- Features.tsx | | | |-- Footer.tsx | | | |-- Hero.tsx | | | |-- HowItWorks.tsx | | | |-- MinibotForm.tsx | | | |-- TechStack.tsx | +-- services | | |-- chatService.ts | +-- types | | |-- index.ts

Project Files

File: chat-test.json__.txt

chat-test.json

{ "clientId": "sam-minibot-prototipe", "message": "Hola" }

File: env.txt

Firebase web config

REACT_APP_FIREBASE_API_KEY=xxxxx REACT_APP_FIREBASE_AUTH_DOMAIN=mini-bot-xxxx.firebaseapp.com REACT_APP_FIREBASE_PROJECT_ID=mini-bot-xxxx REACT_APP_FIREBASE_STORAGE_BUCKET=mini-bot-xxxx.firebasestorage.app REACT_APP_FIREBASE_MESSAGING_SENDER_ID=000000000 REACT_APP_FIREBASE_APP_ID=1:0000000:web:xxxxxxx

API base (en local usaremos emuladores)

REACT_APP_API_BASE=http://127.0.0.1:5001/mini-bot-xxxxx/us-centralxxx

API GPT (Opcional)

OPENAI_API_KEY=sk-proj-xxxxxxxxxxxx-xxxxxxxxxxxxxxxx-xx DEEPSEEK_API_KEY=sk-xxxxxxxxxxxxxxxx DEEPSEEK_URL=https://platform.deepseek.com/ GEMINI_API_KEY=xxxxxxx-xxxxxx

MAILER USER

MAIL_USER=xxxxx.xxxxx@xxxxx.com MAIL_PASS=xxxxxx

CLAVE_SUPER_SECRETA=xxxxxxxxxxxx

File: firebase.json__.txt

firebase.json

{ "hosting": { "public": "build", "ignore": ["firebase.json", "**/.*", "**/node_modules/**"], "rewrites": [{ "source": "/chatbot", "function": "chatbot" }, { "source": "**", "destination": "/index.html" }] }, "functions": { "predeploy": ["npm --prefix \"$RESOURCE_DIR\" run lint", "npm --prefix \"$RESOURCE_DIR\" run build"], "source": "functions" }, "emulators": { "auth": { "port": 9099 }, "functions": { "port": 5001 }, "firestore": { "port": 8080 }, "hosting": { "port": 5000 }, "ui": { "enabled": true }, "singleProjectMode": true } }

File: firestore.indexes.json__.txt

firestore.indexes.json

{ // Example (Standard Edition): // // "indexes": [// { // "collectionGroup": "widgets", // "queryScope": "COLLECTION", // "fields": [// { "fieldPath": "foo", "arrayConfig": "CONTAINS" }, // { "fieldPath": "bar", "mode": "DESCENDING" } //] // }, // // "fieldOverrides": [// { // "collectionGroup": "widgets", // "fieldPath": "baz", // "indexes": [// { "order": "ASCENDING", "queryScope": "COLLECTION" } //] // }, //] //] // // Example (Enterprise Edition): // // "indexes": [// { // "collectionGroup": "reviews", // "queryScope": "COLLECTION_GROUP", // "apiScope": "MONGODB_COMPATIBLE_API", // "density": "DENSE", // "multikey": false, // "fields": [// { "fieldPath": "baz", "mode": "ASCENDING" } //] // }, // { // "collectionGroup": "items", // "queryScope": "COLLECTION_GROUP", // "apiScope": "MONGODB_COMPATIBLE_API", // "density": "SPARSE_ANY", // "multikey": true, // "fields": [// { "fieldPath": "baz", "mode": "ASCENDING" } //] // }, //] "indexes": [], "fieldOverrides": [] }

File: functions__package.json__.txt

package.json

{ "name": "functions", "scripts": { "build": "tsc", "lint": "echo \"Lint skipped\"", "build:watch": "tsc --watch", "serve": "npm run build && firebase emulators:start --only functions", "shell": "npm run build && firebase functions:shell", "start": "npm run shell", "deploy": "firebase deploy --only functions", "logs": "firebase functions:log" }, "engines": { "node": "20" }, "main": "lib/index.js", "dependencies": { "@google/generative-ai": "^0.24.1", "cors": "^2.8.5", "dotenv": "^17.2.3", "firebase-admin": "^11.10.1", "firebase-functions": "^4.6.0", "nodemailer": "^7.0.12", "openai": "^6.15.0" }, "devDependencies": { "@types/cors": "^2.8.17", "@types/node":

"^18.19.0", "@types/nodemailer": "^7.0.4", "firebase-functions-test": "^3.1.0", "typescript": "^5.3.3" }, "private": true }

File: functions__src__chatbot.ts__.txt

chatbot.ts

// functions/src/chatbot.ts // VERSION: 3.9.1 — SaaS + FAQ + PDF + Multi-LLM (SAFE)

import * as admin from 'firebase-admin'; import { Request, Response } from 'express';

import { resolveFAQ } from './services/faq.service'; import { getClientContext } from './services/context.service'; import { generateLLMResponse } from './services/llm.service';

// ---------- // Init Firebase Admin // ----------

export function getDB() { return admin.firestore(); };

// ---------- // Types // ---------- interface ChatRequest { clientId: string; message: string; sessionId?: string; }

type ResponseSource = 'faq' | 'default' | 'llm';

// ---------- // Handler // ---------- export async function chatbotHandler(request: Request, response: Response) { try { const { clientId, message, sessionId = 'default' } = request.body as ChatRequest;

```
// ----------
// Validations
// ----------
if (!clientId) {
  return response.status(400).json({ error: 'clientId es requerido' });
}
```

```
if (!message) {
  return response.status(400).json({ error: 'El mensaje es requerido' });
}
const db = getDB();
const clientRef = db.collection('clients').doc(clientId);
const clientSnap = await clientRef.get();

if (!clientSnap.exists) {
  return response.status(404).json({
    error: 'Cliente no encontrado'
  });
}

const clientData = clientSnap.data()!;

let responseText: string;
let source: ResponseSource;
let confidence = 0;

// --------------------
// 1️⃣ FAQ
// --------------------
const faqResult = await resolveFAQ(clientId, message);

if (faqResult && faqResult.confidence >= 0.6) {
  responseText = faqResult.answer;
  confidence = faqResult.confidence;
  source = 'faq';
}

// --------------------
// 2️⃣ LLM (OpenAI / Gemini / DeepSeek)
// --------------------
else if (clientData.llm?.enabled && clientData.llm.provider) {
  const context = await getClientContext(clientId);

  const llmAnswer = await generateLLMResponse(
```

```
    clientData.llm.provider,
    {
      message,
      context: context ?? undefined,
      systemPrompt: clientData.llm.systemPrompt
    }
  );

  responseText = llmAnswer;
  source = 'llm';
  confidence = 1;
}

// --------------------
// 3️⃣ Default del cliente / fallback global
// --------------------
else {
  const defaultConfig = await clientRef
    .collection('chatbot_config')
    .doc('default')
    .get();

  responseText =
    defaultConfig.exists && defaultConfig.data()?.value
      ? defaultConfig.data()!.value
      : 'Lo siento, no he entendido tu pregunta. ¿Podrías reformularla?';

  source = 'default';
  confidence = 0;
}

// --------------------
// Persist conversation
// --------------------
const now = new Date();

await clientRef.collection('chat_conversations').add({
```

```
  sessionId,
  userMessage: message,
  botResponse: responseText,
  source,
  confidence,
  timestamp: now
});

// --------------------
// Response
// --------------------
return response.status(200).json({
  response: responseText,
  sessionId,
  source,
  confidence,
  timestamp: now.toISOString()
});
```

} catch (error) { console.error('[chatbot] Error:', error);

```
return response.status(500).json({
  error:
    'Lo siento, ha ocurrido un error al procesar tu mensaje. Por favor, intént
});
```

}}

File: functions__src__config__env.ts__.txt

env.ts

// functions/src/config/env.ts

import * as functions from 'firebase-functions';

/* * *Configuración centralizada de variables de entorno * (Firebase Functions Config) /*
const config = functions.config();

export const env = { llm: { openaiKey: config.llm?.openai_key ?? '', geminiKey: config.llm?.gemini_key ?? '', deepseekKey: config.llm?.deepseek_key ?? '' },

mail: { user: config.mail?.user ?? '', pass: config.mail?.pass ?? '' },

admin: { superSecret: config.admin?.super_secret ?? '' } };

File: functions__src__index.ts__.txt

index.ts

import * as admin from 'firebase-admin'; import * as functions from 'firebase-functions'; import express from 'express'; import cors from 'cors';

import { chatbotHandler } from './chatbot'; import { loadClientConfig } from './loadClientConfig'; import { requestMiniBot } from './requestMiniBot';

admin.initializeApp();

// ============================ // CORS (abierto por ahora) // ============================ const corsHandler = cors({ origin: true });

// ========== // CHATBOT (API pública) // ========== export const chatbot = functions.https.onRequest((req, res) => { corsHandler(req, res, () => { if (req.method !== 'POST') { res.status(405).json({ error: 'Método no permitido' }); return; } chatbotHandler(req, res); }); });

// ========== // LOAD CLIENT CONFIG (ADMIN) // ========== const loadClientApp = express();

// 🟢 ESTO ES CRÍTICO loadClientApp.use(express.json({ limit: '2mb' }));

// ❌ NO auth // ❌ NO headers // ❌ NO secrets // ❌ NO OAuth loadClientApp.post('/', loadClientConfig);

export const loadClientConfigFn = functions.https.onRequest(loadClientApp);

// ========== // MINI BOT // ========== export { requestMiniBot };

File: functions__src__loadClientConfig.ts__.txt

loadClientConfig.ts

// functions/src/loadClientConfig.ts

import * as admin from 'firebase-admin'; import { Request, Response } from 'express';

// ---------- // Tipos (claridad y robustez) // ---------- type ClientPayload = { clientId: string; name?: string; domain?: string; active?: boolean; llm?: { enabled?: boolean; provider?: string | null; model?: string | null; k: string: any; }; createdAt?: any;

};

type Payload = { client: ClientPayload; chatbot_config?: { default?: { value?: string; k: string: any }; k: string: any; }; chatbot_responses?: Array<{ id: string; question: string;

answer: string; active?: boolean; order?: number; k: string: any; }>; llm?: { enabled?: boolean; provider?: string | null; model?: string | null; k: string: any; }; context?: { pdf?: Record; k: string: any; };

};

/ * Carga o actualiza la configuración completa de un cliente MiniBot * - Cliente (upsert, idempotente) * - Configuración default * - FAQs * - Configuración LLM * - Contexto PDF * * ⚠ Firebase Admin debe estar inicializado en index.ts */* export async function loadClientConfig(request: Request, response: Response): Promise { try { // ---------- // Método permitido // ---------- if (request.method !== 'POST') { response.status(405).json({ error: 'Método no permitido' }); return; }

```
// --------------------
// Validar body
// --------------------
if (!request.is('application/json')) {
  response.status(400).json({
    error: 'Content-Type debe ser application/json'
  });
  return;
}

const payload = request.body as Payload;

if (!payload || typeof payload !== 'object') {
  response.status(400).json({ error: 'Payload inválido' });
  return;
}

// --------------------
// Validaciones críticas
// --------------------
if (!payload.client?.clientId) {
  response.status(400).json({
    error: 'client.clientId es requerido'
  });
  return;
```

```
}

const clientId = String(payload.client.clientId).trim();
if (!clientId) {
  response.status(400).json({
    error: 'client.clientId inválido'
  });
  return;
}

const db = admin.firestore();
const clientRef = db.collection('clients').doc(clientId);
const now = admin.firestore.FieldValue.serverTimestamp();

// --------------------
// 1) CLIENTE (UPSERT idempotente)
// --------------------
await db.runTransaction(async (tx) => {
  const snap = await tx.get(clientRef);

  const existing = snap.exists ? snap.data() : undefined;
  const createdAt =
    existing?.createdAt ??
    payload.client.createdAt ??
    admin.firestore.FieldValue.serverTimestamp();

  tx.set(
    clientRef,
    {
      ...payload.client,
      clientId,
      createdAt,
      updatedAt: now
    },
    { merge: true }
  );
});
```

```
// --------------------
// 2) CONFIGURACIÓN DEFAULT
// --------------------
const defaultValue = payload.chatbot_config?.default?.value;
if (typeof defaultValue === 'string' && defaultValue.trim()) {
  await clientRef
    .collection('chatbot_config')
    .doc('default')
    .set(
      {
        value: defaultValue,
        updatedAt: now
      },
      { merge: true }
    );
}

// --------------------
// 3) FAQs / RESPUESTAS
// --------------------
if (Array.isArray(payload.chatbot_responses)) {
  for (const faq of payload.chatbot_responses) {
    if (!faq?.id || !faq?.question || !faq?.answer) continue;

    const faqId = String(faq.id).trim();
    if (!faqId) continue;

    await clientRef
      .collection('chatbot_responses')
      .doc(faqId)
      .set(
        {
          question: faq.question,
          answer: faq.answer,
          active: faq.active ?? true,
          order: faq.order ?? 0,
```

```
          updatedAt: now
        },
        { merge: true }
      );
  }
}

// --------------------
// 4) CONFIGURACIÓN LLM
// --------------------
const llmToSave =
  payload.client.llm && Object.keys(payload.client.llm).length
    ? payload.client.llm
    : payload.llm && Object.keys(payload.llm).length
    ? payload.llm
    : null;

if (llmToSave) {
  await clientRef.set(
    {
      llm: {
        ...llmToSave,
        updatedAt: now
      },
      updatedAt: now
    },
    { merge: true }
  );
}

// --------------------
// 5) CONTEXTO PDF
// --------------------
if (payload.context?.pdf) {
  await clientRef
    .collection('context')
    .doc('pdf')
```

```
    .set(
      {
        ...payload.context.pdf,
        updatedAt: now
      },
      { merge: true }
    );
}

// --------------------
// RESPUESTA OK
// --------------------
console.log(`[loadClientConfig] Configuración aplicada: ${clientId}`);

response.status(200).json({
  status: 'ok',
  clientId,
  message: 'Configuración aplicada correctamente'
});
```

} catch (error: any) { console.error('[loadClientConfig]', error);

```
response.status(500).json({
  error: 'Error cargando configuración del cliente',
  details: error?.message ?? String(error)
});
```

}}

File: functions__src__providers__deepseek.provider.ts__.txt

deepseek.provider.ts

// functions/src/providers/deepseek.provider.ts

import OpenAI from 'openai'; import { LLMProvider, LLMRequest } from './llm.types';

export class DeepSeekProvider implements LLMProvider { private client: OpenAI;

constructor(apiKey: string) { this.client = new OpenAI({ apiKey, baseURL: 'https://platform.deepseek.com/' }); }

async generate(req: LLMRequest): Promise { const completion = await this.client.chat.completions.create({ model: 'deepseek-chat', messages: [{ role: 'system', content: req.systemPrompt ?? '' }, { role: 'user', content: req.context ? `${req.context}\n\n${req.message}` : req.message }] });

```
return (
  completion.choices[0]?.message?.content ??
  'No se pudo generar respuesta'
);
```

} }

File: functions__src__providers__gemini.provider.ts__.txt

gemini.provider.ts

// functions/src/providers/gemini.provider.ts

import { GoogleGenerativeAI } from '@google/generative-ai'; import { LLMProvider, LLMRequest } from './llm.types';

export class GeminiProvider implements LLMProvider { private client: GoogleGenerativeAI;

constructor(apiKey: string) { this.client = new GoogleGenerativeAI(apiKey); }

async generate(req: LLMRequest): Promise { const model = this.client.getGenerativeModel({ model: 'gemini-pro' });

```
const prompt = `
```

${req.systemPrompt ?? ''}

${req.context ?? ''}

Pregunta: ${req.message} `.trim();

```
const result = await model.generateContent(prompt);
return result.response.text();
```

} }

File: functions__src__providers__llm.types.ts__.txt

llm.types.ts

// functions/src/providers/llm.types.ts

export type LLMVendor = 'openai' | 'gemini' | 'deepseek';

export interface LLMRequest { message: string; systemPrompt?: string; context?: string; }

export interface LLMProvider { generate(req: LLMRequest): Promise; }

File: functions__src__providers__openai.provider.ts__.txt

openai.provider.ts

// functions/src/providers/openai.provider.ts

import OpenAI from 'openai'; import { LLMProvider, LLMRequest } from './llm.types';

export class OpenAIProvider implements LLMProvider { private client: OpenAI;

constructor(apiKey: string) { this.client = new OpenAI({ apiKey }); }

async generate(req: LLMRequest): Promise { const completion = await this.client.chat.completions.create({ model: 'gpt-4o-mini', temperature: 0.4, messages: [{ role: 'system', content: req.systemPrompt ?? 'Eres un asistente profesional, claro y orientado a ayudar.' }, { role: 'user', content: req.context ? `${req.context}\n\nPregunta:\n${req.message}` : req.message }] });

```
return (
  completion.choices[0]?.message?.content ??
  'No fue posible generar respuesta.'
);
```

}}

File: functions__src__requestMiniBot.ts__.txt

requestMiniBot.ts

// functions/src/requestMiniBot.ts

import * as functions from 'firebase-functions'; import * as admin from 'firebase-admin'; import * as nodemailer from 'nodemailer'; import { Request, Response } from 'express';

export function getDB() { return admin.firestore(); };

// ---------- // Transporter (correo) // ---------- const transporter = nodemailer.createTransport({ service: 'gmail', auth: { user: process.env.MAIL_USER, pass: process.env.MAIL_PASS } });

// ---------- // Function // ---------- export const requestMiniBot = functions.https.onRequest(async (req: Request, res: Response): Promise => { try { if (req.method !== 'POST') { res.status(405).json({ error: 'Método no permitido' }); return; }

```
  const { contact, config } = req.body;
  const db = getDB();

  if (!contact?.email || !contact?.siteName) {
```

```
    res.status(400).json({
      error: 'Datos de contacto incompletos'
    });
    return;
  }

  // ------------------------------
  // Guardar solicitud en Firestore
  // ------------------------------
  await db.collection('minibot_requests').add({
    contact,
    config,
    status: 'pending',
    createdAt: new Date()
  });

  // ------------------------------
  // Enviar correo
  // ------------------------------
  await transporter.sendMail({
    from: `"SAM MiniBot" <${process.env.MAIL_USER}>`,
    to: 'luisenguerrero.cm@gmail.com',
    subject: `Solicito un MiniBot para mi Sitio Web ${contact.siteName}`,
    html: `
      <h2>📩 Nueva solicitud de MiniBot</h2>
      <p><strong>Sitio:</strong> ${contact.siteName}</p>
      <p><strong>Contacto:</strong> ${contact.name} (${contact.email})</p>
      <p><strong>Mensaje:</strong> ${contact.message ?? '—'}</p>
      <pre>${JSON.stringify(config, null, 2)}</pre>
    `
  });

  // ------------------------------
  // Respuesta OK
  // ------------------------------
  res.status(200).json({
    status: 'ok',
```

```
    message: 'Solicitud enviada correctamente'
  });
  return;

} catch (error) {
  console.error('[requestMiniBot]', error);
  res.status(500).json({
    error: 'Error procesando la solicitud'
  });
  return;
}
```

});

File: functions__src__services__context.service.ts__.txt

context.service.ts

import * as admin from 'firebase-admin';

export function getDB() { return admin.firestore(); };

/ * Obtiene el contexto base del cliente desde PDF procesado * (texto plano previamente almacenado) /* export async function getClientContext(clientId: string): Promise { const db = getDB(); const doc = await db .collection('clients') .doc(clientId) .collection('context') .doc('pdf') .get();

if (!doc.exists) return null;

const data = doc.data(); return data?.text || null; }

File: functions__src__services__faq.service.ts__.txt

faq.service.ts

import * as admin from 'firebase-admin';

export function getDB() { return admin.firestore(); };

interface FAQMatch { answer: string; confidence: number; }

/* * *Busca una respuesta FAQ por similitud* / export async function resolveFAQ(clientId: string, message: string): Promise { const db = getDB(); const snapshot = await db .collection('clients') .doc(clientId) .collection('chatbot_responses') .where('active', '==', true) .get();

if (snapshot.empty) return null;

const normalizedMessage = normalize(message);

let bestMatch: FAQMatch | null = null;

snapshot.forEach(doc => { const data = doc.data(); if (!data.question || !data.answer) return;

```
const similarity = similarityScore(
  normalizedMessage,
  normalize(data.question)
);

if (!bestMatch || similarity > bestMatch.confidence) {
  bestMatch = {
    answer: data.answer,
    confidence: similarity
  };
}
```

});

return bestMatch; }

/ ---------- *utils* ---------- /

function normalize(text: string): string { return text.toLowerCase().trim().replace(/[^\w\s]/gi, ''); }

function similarityScore(a: string, b: string): number { const longer = a.length > b.length ? a : b; const shorter = a.length > b.length ? b : a; if (longer.length === 0) return 1; return (longer.length - levenshtein(longer, shorter)) / longer.length; }

function levenshtein(a: string, b: string): number { const matrix = Array.from({ length: b.length + 1 }, (_, i) => [i]); for (let j = 0; j <= a.length; j++) matrix[0][j] = j;

for (let i = 1; i <= b.length; i++) { for (let j = 1; j <= a.length; j++) { matrix[i][j] = b[i - 1] === a[j - 1] ? matrix[i - 1][j - 1] : Math.min(matrix[i - 1][j - 1] + 1, matrix[i][j - 1] + 1, matrix[i - 1][j] + 1); } } return matrix[b.length][a.length]; }

File: functions__src__services__llm.service.ts__.txt

llm.service.ts

// functions/src/services/llm.service.ts

import { env } from '../config/env'; import { LLMRequest, LLMVendor } from '../providers/llm.types'; import { OpenAIProvider } from '../providers/openai.provider'; import { GeminiProvider } from '../providers/gemini.provider'; import { DeepSeekProvider } from '../providers/deepseek.provider';

const providers: Record = { openai: new OpenAIProvider(env.llm.openaiKey), gemini: new GeminiProvider(env.llm.geminiKey), deepseek: new DeepSeekProvider(env.llm.deepseekKey) };

export async function generateLLMResponse(vendor: LLMVendor, req: LLMRequest): Promise { const provider = providers[vendor];

if (!provider) { throw new Error(`LLM provider no soportado: ${vendor}`); }

return provider.generate(req); }

File: functions__tsconfig.json__.txt

tsconfig.json

{ "compilerOptions": { / *Plataforma* / "target": "ES2020", "lib": ["ES2020"],

```
/* Módulos */
"module": "commonjs",
"moduleResolution": "node",

/* Estructura del proyecto */
"rootDir": "src",
"outDir": "lib",
"sourceMap": true,

/* Interoperabilidad */
"esModuleInterop": true,
"allowSyntheticDefaultImports": true,

/* Calidad de código */
"strict": true,
"noImplicitReturns": true,
"noUnusedLocals": true,
"forceConsistentCasingInFileNames": true,
```

```
/* 🔑 CLAVE PARA TU ERROR ACTUAL */
"skipLibCheck": true
```

},

"include": ["src"], "exclude": ["node_modules", "lib"] }

File: LoadData__data-loader.prod.ps1__.txt

data-loader.prod.ps1

$ErrorActionPreference = "Stop"

$ScriptDir = Split-Path -Parent $MyInvocation.MyCommand.Path $JsonPath = Join-Path $ScriptDir "sam-minibot.initial.json"

Write-Host "🚀 Cargando configuración del cliente desde sam-minibot.initial.json..."

if (-not (Test-Path $JsonPath)) { Write-Error "❌ No se encuentra el archivo JSON: $JsonPath" exit 1 }

Leer JSON como texto UTF-8 limpio

$jsonRaw = Get-Content $JsonPath -Raw -Encoding UTF8

Validar que sea JSON válido

try { $null = $jsonRaw | ConvertFrom-Json } catch { Write-Error "❌ El archivo JSON no es válido" exit 1 }

Token Firebase

$token = firebase auth:print-access-token

if (-not $token) { Write-Error "❌ No se pudo obtener el token de Firebase" exit 1 }

try { $response = Invoke-RestMethod `-Uri "https://us-central1-mini-bot-7a21d.cloudfunctions.net/loadClientConfigFn"` -Method POST `-Headers @{ "Authorization" = "Bearer $token" "Content-Type" = "application/json; charset=utf-8" }` -Body $jsonRaw

Write-Host "☑ Configuración cargada correctamente:" $response | ConvertTo-Json -Depth 5

} catch { Write-Error "❌ Error cargando configuración:" Write-Error $_.Exception.Message exit 1 }

File: LoadData__load-client.js__.txt

load-client.js

/ * LoadData/load-client.js * ---------- * Carga la configuración inicial de un cliente * en SAM MiniBot (producción). * * ✔ NO usa OAuth * ✔ NO usa service-account * ✔ NO usa secrets * ✔ Funciona igual que curl /*

const fs = require('fs'); const path = require('path'); const fetch = require('node-fetch');

// ========== // CONFIG // ==========
const FUNCTION_URL = 'https://us-central1-mini-bot-7a21d.cloudfunctions.net/loadClientConfigFn';

const JSON_PATH = path.resolve(__dirname, 'sam-minibot.initial.json');

// ═══ // MAIN // ═══
async function main() { console.log('\n🚀 Cargando configuración del cliente MiniBot...\n');

if (!fs.existsSync(JSON_PATH)) { console.error('❌ No se encuentra el archivo JSON:', JSON_PATH); process.exit(1); }

const jsonBody = fs.readFileSync(JSON_PATH, 'utf8');

let parsed; try { parsed = JSON.parse(jsonBody); } catch { console.error('❌ JSON inválido'); process.exit(1); }

if (!parsed.client?.clientId) { console.error('❌ Falta client.clientId'); process.exit(1); }

console.log(`📦 Cliente: ${parsed.client.clientId}`); console.log('📨 Enviando configuración...\n');

const res = await fetch(FUNCTION_URL, { method: 'POST', headers: { 'Content-Type': 'application/json' }, body: jsonBody });

const text = await res.text();

if (!res.ok) { console.error(`❌ Error HTTP ${res.status}`); console.error(text); process.exit(1); }

console.log('☑️ Configuración cargada correctamente:\n'); console.log(text); }

main().catch(err => { console.error('❌ Error inesperado:', err); process.exit(1); });

File: LoadData__sam-minibot.initial.clean.json__.txt

sam-minibot.initial.clean.json

{ "client": { "clientId": "sam-minibot-prototipe", "name": "SAM MiniBot", "domain": "https://sam-minibot.web.app", "active": true, "llm": { "enabled": false, "provider": null, "model": null } },

"chatbot_config": { "default": { "value": "Soy Sam tu asistente virtual. Puedo ayudarte a conocer cÃ³mo funciona SAM MiniBot y cÃ³mo puede integrarse en tu sitio web." } },

"chatbot_responses": [{ "id": "faq_01", "question": "Hola, saludos, buenas,", "answer": "Hola Soy SAM tu asistente virtual. SAM MiniBot es un asistente conversacional SaaS que puedes integrar en tu sitio web para responder preguntas, guiar usuarios y escalar a inteligencia artificial.", "active": true, "order": 1 }, { "id": "faq_02", "question": "Â¿Para quÃ© tipo de negocios sirve SAM MiniBot?", "answer": "SAM MiniBot es ideal para empresas, emprendimientos, instituciones educativas y organizaciones que necesitan atenciÃ³n automatizada y escalable.", "active": true, "order": 2 }, { "id": "faq_03", "question": "Â¿Necesito conocimientos tÃ©cnicos para usarlo?", "answer": "No. SAM MiniBot puede integrarse mediante un script sencillo sin modificar tu backend.", "active": true, "order": 3 }, { "id": "faq_04", "question": "Â¿Puedo personalizar las respuestas?", "answer": "SÃ. Puedes definir tus propias preguntas y respuestas desde Firestore o un panel administrativo.", "active": true, "order": 4 }, { "id": "faq_05", "question": "Â¿SAM MiniBot funciona sin inteligencia artificial?", "answer": "SÃ. Funciona perfectamente con preguntas frecuentes incluso sin GPT.", "active": true, "order": 5 }, { "id": "faq_06", "question": "Â¿QuÃ© ventaja tiene usar GPT con SAM MiniBot?", "answer": "Permite responder preguntas complejas usando documentos, catÃ¡logos o PDFs como contexto.", "active": true, "order": 6 }, { "id": "faq_07", "question": "Â¿Los datos de mi empresa estÃ¡n seguros?", "answer": "SÃ. Cada cliente tiene su propio espacio aislado en la base de datos.", "active": true, "order": 7 }, { "id": "faq_08", "question": "Â¿Puedo usar SAM MiniBot en varias pÃ¡ginas?", "answer": "SÃ. Un mismo bot puede funcionar en mÃºltiples sitios del mismo cliente.", "active": true, "order": 8 }, { "id": "faq_09", "question": "Â¿SAM MiniBot guarda conversaciones?", "answer": "SÃ, de forma opcional, para anÃ¡lisis y mejora del servicio.", "active": true, "order": 9 }, { "id": "faq_10", "question": "Â¿CÃ³mo

empiezo a usar SAM MiniBot?", "answer": "Puedes contactarnos o seguir la guÃa de integraciÃ³n disponible en esta pÃ¡gina.", "active": true, "order": 10 }] }

File: LoadData__sam-minibot.initial.json__.txt

sam-minibot.initial.json

{ "client": { "clientId": "sam-minibot-prototipe", "name": "SAM MiniBot", "domain": "https://mini-bot-7a21d.web.app/", "active": true, "llm": { "enabled": false, "provider": null, "model": null } },

"chatbot_config": { "default": { "value": "Soy Sam tu asistente virtual. Puedo ayudarte a conocer cómo funciona SAM MiniBot y cómo puede integrarse en tu sitio web." } },

"chatbot_responses": [{ "id": "faq_01", "question": "Hola, saludos, buenas,", "answer": "Hola Soy SAM tu asistente virtual. SAM MiniBot es un asistente conversacional SaaS que puedes integrar en tu sitio web para responder preguntas, guiar usuarios y escalar a inteligencia artificial.", "active": true, "order": 1 }, { "id": "faq_02", "question": "¿Para qué tipo de negocios sirve SAM MiniBot?", "answer": "SAM MiniBot es ideal para empresas, emprendimientos, instituciones educativas y organizaciones que necesitan atención automatizada y escalable.", "active": true, "order": 2 }, { "id": "faq_03", "question": "¿Necesito conocimientos técnicos para usarlo?", "answer": "No. SAM MiniBot puede integrarse mediante un script sencillo sin modificar tu backend.", "active": true, "order": 3 }, { "id": "faq_04", "question": "¿Puedo personalizar las respuestas?", "answer": "Sí. Puedes definir tus propias preguntas y respuestas desde Firestore o un panel administrativo.", "active": true, "order": 4 }, { "id": "faq_05", "question": "¿SAM MiniBot funciona sin inteligencia artificial?", "answer": "Sí. Funciona perfectamente con preguntas frecuentes incluso sin GPT.", "active": true, "order": 5 }, { "id": "faq_06", "question": "¿Qué ventaja tiene usar GPT con SAM MiniBot?", "answer": "Permite responder preguntas complejas usando documentos, catálogos o PDFs como contexto.", "active": true, "order": 6 }, { "id": "faq_07", "question": "¿Los datos de mi empresa están seguros?", "answer": "Sí. Cada cliente tiene su propio espacio aislado en la base de datos.", "active": true, "order": 7 }, { "id": "faq_08", "question": "¿Puedo usar SAM MiniBot en varias páginas?", "answer": "Sí. Un mismo

bot puede funcionar en múltiples sitios del mismo cliente.", "active": true, "order": 8 }, { "id": "faq_09", "question": "¿SAM MiniBot guarda conversaciones?", "answer": "Sí, de forma opcional, para análisis y mejora del servicio.", "active": true, "order": 9 }, { "id": "faq_10", "question": "¿Cómo empiezo a usar SAM MiniBot?", "answer": "Puedes contactarnos o seguir la guía de integración disponible en esta página.", "active": true, "order": 10 }] }

File: package.json__.txt

---

package.json

{ "name": "sam_minibot", "version": "0.24.8", "private": true, "dependencies": { "@google/generative-ai": "^0.24.1", "autoprefixer": "^10.4.17", "google-auth-library": "^10.5.0", "node-fetch": "^2.7.0", "postcss": "^8.4.35", "react": "^18.2.0", "react-dom": "^18.2.0", "react-hook-form": "^7.70.0", "react-scripts": "5.0.1", "tailwindcss": "^3.4.1", "web-vitals": "^2.1.4" }, "devDependencies": { "@testing-library/jest-dom": "^6.4.2", "@testing-library/react": "^14.2.1", "@testing-library/user-event": "^14.6.1", "@types/jest": "^29.5.12", "@types/node": "^18.19.0", "@types/react": "^18.2.48", "@types/react-dom": "^18.2.18", "typescript": "4.9.5" }, "scripts": { "start": "react-scripts start", "build": "react-scripts build", "test": "react-scripts test", "eject": "react-scripts eject", "firebase:serve": "npm run build && firebase emulators:start", "firebase:deploy": "npm run build && firebase deploy" }, "eslintConfig": { "extends": ["react-app", "react-app/jest"] }, "browserslist": { "production": [">0.2%", "not dead", "not op_mini all"], "development": ["last 1 chrome version", "last 1 firefox version", "last 1 safari version"] } }

File: public__index.html__.txt

index.html

Necesitas habilitar JavaScript para ejecutar esta aplicación.

File: public__manifest.json__.txt

manifest.json

File: robots.txt

File: scripts__load-gpt-config.prod.ps1__.txt

load-gpt-config.prod.ps1

Write-Host "Activando GPT y contexto PDF para SAM MiniBot..."

Invoke-WebRequest `-Uri https://us-central1-mini-bot-7a21d.cloudfunctions.net/loadClientConfigFn` -Method POST `-Headers @{ "Content-Type" = "application/json" }` -Body (Get-Content .\sam-minibot.gpt.json -Raw)

Write-Host "Configuración GPT aplicada correctamente."

File: scripts__load-gpt-config.ps1__.txt

load-gpt-config.ps1

Write-Host "Activando GPT y contexto PDF para SAM MiniBot..."

Invoke-WebRequest `-Uri http://localhost:5001/mini-bot-7a21d/us-central1/loadClientConfigFn` -Method POST `-Headers @{ "Content-Type" = "application/json" }` -Body (Get-Content .\sam-minibot.gpt.json -Raw)

Write-Host "Configuración GPT aplicada correctamente."

File: scripts__load-initial-client.prod.ps1__.txt

load-initial-client.prod.ps1

param ([string]$JsonFile = "sam-minibot.initial.json")

Write-Host "🚀 Cargando configuración del cliente desde $JsonFile..."

0️⃣ Resolver rutas absolutas

$ScriptDir = Split-Path -Parent $MyInvocation.MyCommand.Path $JsonPath = Join-Path $ScriptDir $JsonFile $CleanJsonPath = Join-Path $ScriptDir "sam-minibot.initial.clean.json"

if (-not (Test-Path $JsonPath)) { Write-Error "❌ No se encuentra el archivo JSON: $JsonPath" exit 1 }

1️⃣ Obtener token Firebase

$token = (firebase auth:print-access-token).Trim()

if (-not $token) { Write-Error "❌ No se pudo obtener token Firebase" exit 1 }

2️⃣ Leer y escribir JSON limpio (UTF-8 sin BOM)

$jsonContent = Get-Content $JsonPath -Raw $utf8NoBom = New-Object System.Text.UTF8Encoding($false)

3️⃣ Ejecutar request con curl

$responseCode = curl.exe `-s` -o "$ScriptDir\response.tmp" `-w "%{http_code}"` -X POST `-H "Authorization: Bearer $token"` -H "Content-Type: application/json" `--data-binary "@$CleanJsonPath"` https://us-central1-mini-bot-7a21d.cloudfunctions.net/loadClientConfigFn

4️⃣ Validar resultado

if ($responseCode -ne "200") { Write-Error "❌ Error cargando configuración (HTTP $responseCode)" Get-Content "$ScriptDir\response.tmp" exit 1 }

Write-Host "☑️ Configuración cargada correctamente:" Get-Content "$ScriptDir\response.tmp"

File: scripts__load-initial-client.ps1__.txt

load-initial-client.ps1

Write-Host "Cargando configuración inicial de SAM MiniBot..."

Invoke-WebRequest `-Uri http://localhost:5001/mini-bot-7a21d/us-central1/loadClientConfigFn` -Method POST `-Headers @{ "Content-Type" = "application/json" }` -Body (Get-Content .\sam-minibot.initial.json -Raw)

Write-Host "Configuración inicial cargada correctamente."

File: scripts__set-firebase-config.ps1__.txt

set-firebase-config.ps1

Write-Host "Sincronizando variables de entorno con Firebase..."

Cargar .env

Get-Content ".env" | ForEach-Object { if ($_ -match "^(.?)=(.)$") { $key = $matches[1] $value = $matches[2]

```
switch ($key) {
  "MAIL_USER" {
    firebase functions:config:set mail.user="$value"
  }
  "MAIL_PASS" {
    firebase functions:config:set mail.pass="$value"
  }
  "OPENAI_API_KEY" {
    firebase functions:config:set llm.openai_key="$value"
  }
  "DEEPSEEK_API_KEY" {
    firebase functions:config:set llm.deepseek_key="$value"
  }
  "GEMINI_API_KEY" {
    firebase functions:config:set llm.gemini_key="$value"
  }
}
```

} }

Write-Host "Variables cargadas en Firebase Config"

firebase functions:config:get

Write-Host "Recuerda ejecutar: firebase deploy --only functions"

File: src__App.css__.txt

App.css

.App { text-align: center; }

.App-header { background-color: #282c34; padding: 20px; color: white; }

File: src__App.tsx__.txt

App.tsx

import React, { useState } from 'react'; import ChatBot from './components/ChatBot.tsx'; import Hero from './components/landing/Hero.tsx'; import Features from './components/landing/Features.tsx'; import HowItWorks from './components/landing/HowItWorks.tsx'; import MinibotForm from './components/landing/MinibotForm.tsx'; import TechStack from './components/landing/TechStack.tsx'; import Footer from './components/landing/Footer.tsx'; import './App.css';

/* * *Componente principal de la aplicación que orquesta la landing page y el chatbot.* / function App() { // Estado para controlar la visibilidad de la ventana de chat, ahora en el padre const [isChatOpen, setIsChatOpen] = useState(false);

return (

{/ *Sección Hero con la llamada a la acción principal* /} setIsChatOpen(true)} />

```
{/* Sección de Características */}
<Features />

{/* Sección de "Cómo Funciona" */}
```

```
      <HowItWorks />

      {/* Sección del Formulario Minibot */}
      <MinibotForm />

      {/* Sección de Stack Tecnológico */}
      <TechStack />

      {/* Footer */}
      <Footer />

      {/* El componente Chatbot ahora recibe el estado y la función para controlar
      <ChatBot isChatOpen={isChatOpen} setIsChatOpen={setIsChatOpen} />
    </div>
```

); }

export default App;

File: src__components__ChatBot.tsx__.txt

ChatBot.tsx

import React, { useState, useEffect, useRef } from 'react'; import ChatMessage from './ChatMessage.tsx'; import ChatInput from './ChatInput.tsx'; import { ChatMessage as ChatMessageType, ChatResponse } from '../types'; import { sendMessage } from '../services/chatService.ts';

interface ChatBotProps { isChatOpen: boolean; setIsChatOpen: React.Dispatch>; }

/* * *Componente principal del chatbot. * Ahora es controlado por el componente padre (App.tsx).* / const ChatBot: React.FC = ({ isChatOpen, setIsChatOpen }) => { const [messages, setMessages] = useState([]); const [isLoading, setIsLoading] = useState(false); const [sessionId, setSessionId] = useState(''); const messagesEndRef = useRef(null);

// Generar un ID de sesión al montar el componente useEffect(() => { const newSessionId = Math.random().toString(36).substring(2, 15) + Math.random().toString(36).substring(2, 15); setSessionId(newSessionId);

```
// Mensaje de bienvenida inicial
const welcomeMessage: ChatMessageType = {
  id: 'welcome',
  text: '¡Hola! Soy un asistente virtual. ¿En qué puedo ayudarte hoy?',
  sender: 'bot',
  timestamp: new Date()
};

setMessages([welcomeMessage]);
```

}, []);

// Desplazarse hacia abajo cuando hay nuevos mensajes useEffect(() => { scrollToBottom(); }, [messages]);

/* * *Envía un mensaje del usuario y obtiene la respuesta del bot * @param userMessage Mensaje del usuario* / const handleSendMessage = async (userMessage: string) => { // Añadir mensaje del usuario al estado const userMsg: ChatMessageType = { id: `user-${Date.now()}`, text: userMessage, sender: 'user', timestamp: new Date() };

```
setMessages(prevMessages => [...prevMessages, userMsg]);
setIsLoading(true);
```

```
try {
  // Enviar mensaje al backend y obtener respuesta
  const response: ChatResponse = await sendMessage(userMessage, sessionId);

  // Añadir respuesta del bot al estado
  const botMsg: ChatMessageType = {
    id: `bot-${Date.now()}`,
    text: response.response,
    sender: 'bot',
    timestamp: new Date()
  };

  setMessages(prevMessages => [...prevMessages, botMsg]);
} catch (error) {
  console.error('Error al obtener respuesta del bot:', error);

  // Mensaje de error
  const errorMsg: ChatMessageType = {
    id: `error-${Date.now()}`,
    text: 'Lo siento, ha ocurrido un error al procesar tu mensaje. Por favor,
    sender: 'bot',
    timestamp: new Date()
  };

  setMessages(prevMessages => [...prevMessages, errorMsg]);
} finally {
  setIsLoading(false);
}
```

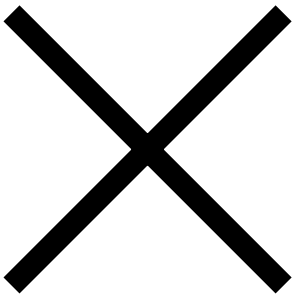
};

/* * *Desplaza la vista de mensajes hacia abajo* / const scrollToBottom = () =>
{ messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' }); };

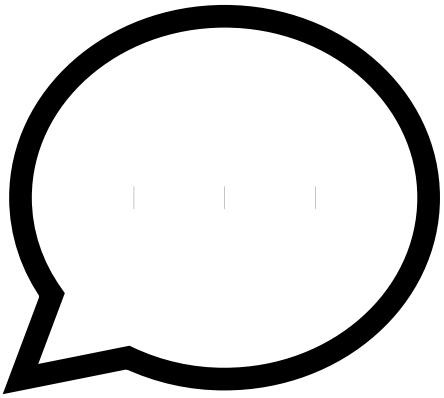
/* * Alterna la visibilidad de la ventana de chat. * Ahora usa la función pasada como prop. / const toggleChat = () => { setIsChatOpen(!isChatOpen); };

return (

{isChatOpen ? (



) : (



)}

{/ Botón para abrir/cerrar el chat /}

```
{/* Ventana de chat */}
{isChatOpen && (
  <div className="absolute bottom-16 right-0 w-80 h-96 bg-white rounded-lg s
    {/* Cabecera del chat */}
    <div className="bg-blue-500 text-white p-4 flex justify-between items-ce
```

```
          <h2 className="text-lg font-semibold">Asistente Virtual</h2>
          <div className="flex items-center">
            <div className="w-2 h-2 bg-green-400 rounded-full mr-2"></div>
            <span className="text-sm">En línea</span>
          </div>
        </div>

        {/* Contenedor de mensajes */}
        <div className="flex-1 p-4 overflow-y-auto bg-gray-50">
          {messages.map((message) => (
            <ChatMessage key={message.id} message={message} />
          ))}

          {/* Indicador de que el bot está escribiendo */}
          {isLoading && (
            <div className="flex justify-start mb-4">
              <div classNa="bg-gray-200 text-gray-800 rounded-lg rounded-bl-nc
                <p className="text-sm typing-indicator">Escribiendo</p>
              </div>
            </div>
          )}

          <div ref={messagesEndRef} />
        </div>

        {/* Input de mensajes */}
        <ChatInput onSendMessage={handleSendMessage} disabled={isLoading} />
      </div>
    )}
  </div>
```

); };

export default ChatBot;

File: src__components__ChatInput.tsx__.txt

ChatInput.tsx

import React, { useState, FormEvent } from 'react';

interface ChatInputProps { onSendMessage: (message: string) => void; disabled?: boolean; }

/ * Componente para el input de mensajes del chat * @param onSendMessage Función para enviar un mensaje * @param disabled Indica si el input está deshabilitado */* const ChatInput: React.FC = ({ onSendMessage, disabled = false }) => { const [message, setMessage] = useState('');

const handleSubmit = (e: FormEvent) => { e.preventDefault(); if (message.trim() && !disabled) { onSendMessage(message); setMessage(''); } };

return (

{message}

setMessage(e.target.value)} placeholder="Escribe tu mensaje..." className="flex-1 px-4 py-2 border border-gray-300 rounded-full focus:outline-none focus:ring-2 focus:ring-blue-500" disabled={disabled} />

ml-2 px-4 py-2 rounded-full ${ message.trim() && !disabled ? 'bg-blue-500 text-white hover:bg-blue-600' : 'bg-gray-300 text-gray-500 cursor-not-allowed' } transition-colors duration-200}

>

Enviar

); };

export default ChatInput;

File: src__components__ChatMessage.tsx__.txt

ChatMessage.tsx

import React from 'react'; import { ChatMessage as ChatMessageType } from '../types';

interface ChatMessageProps { message: ChatMessageType; }

/ * Componente para mostrar un mensaje en el chat * @param message Mensaje a mostrar */* const ChatMessage: React.FC = ({ message }) => { const isUser = message.sender === 'user';

return (

flex ${isUser ? 'justify-end' : 'justify-start'} mb-4}>
max-w-xs lg:max-w-md px-4 py-2 rounded-lg ${ isUser ? 'bg-blue-500 text-white rounded-br-none' : 'bg-gray-200 text-gray-800 rounded-bl-none' }} >

{message.text}

text-xs mt-1 ${isUser ? 'text-blue-100' : 'text-gray-500'}}>
{formatTime(message.timestamp)}

); };

/ * Formatea la fecha y hora del mensaje * @param date Fecha a formatear * @returns Cadena con la fecha y hora formateada */* const formatTime = (date: Date): string => { return new Date(date).toLocaleTimeString('es-ES', { hour: '2-digit', minute: '2-digit' }); };

export default ChatMessage;

File: src__components__landing__Features.tsx__.txt

Features.tsx

import React from 'react';

const Features: React.FC = () => { const features = [{ name: 'Arquitectura Serverless', description: 'Cero infraestructura que gestionar. Escala automáticamente para miles de usuarios sin esfuerzo.', icon: ' ⚡ ', }, { name: 'Respuestas en < 5s', description: 'Optimizado para la velocidad. Procesamiento de consultas y respuestas casi instantáneas.', icon: '🚀', }, { name: 'Seguro y Fiable', description: 'Comunicación segura mediante HTTP y las mejores prácticas de Firebase para proteger tus datos.', icon: ' 🔒 ', }, { name: 'Lógica Propia', description: 'Sin dependencias de plataformas de terceros. Construido desde cero para máxima flexibilidad.', icon: '🧠', },];

return (

Características Principales

Una solución robusta diseñada para ofrecer la mejor experiencia conversacional.

{features.map((feature) => (
{feature.icon}

{feature.name}

{feature.description}

))}
); };

export default Features;

File: src__components__landing__Footer.tsx__.txt

Footer.tsx

import React from 'react';

const Footer: React.FC = () => { return (

© {new Date().getFullYear()} SAM Mini Bot Assistant. Todos los derechos reservados.

Prototipo funcional desarrollado por Luis Enrique Guerrero.

); };

export default Footer;

File: src__components__landing__Hero.tsx__.txt

Hero.tsx

import React from 'react';

interface HeroProps { openChat: () => void; }

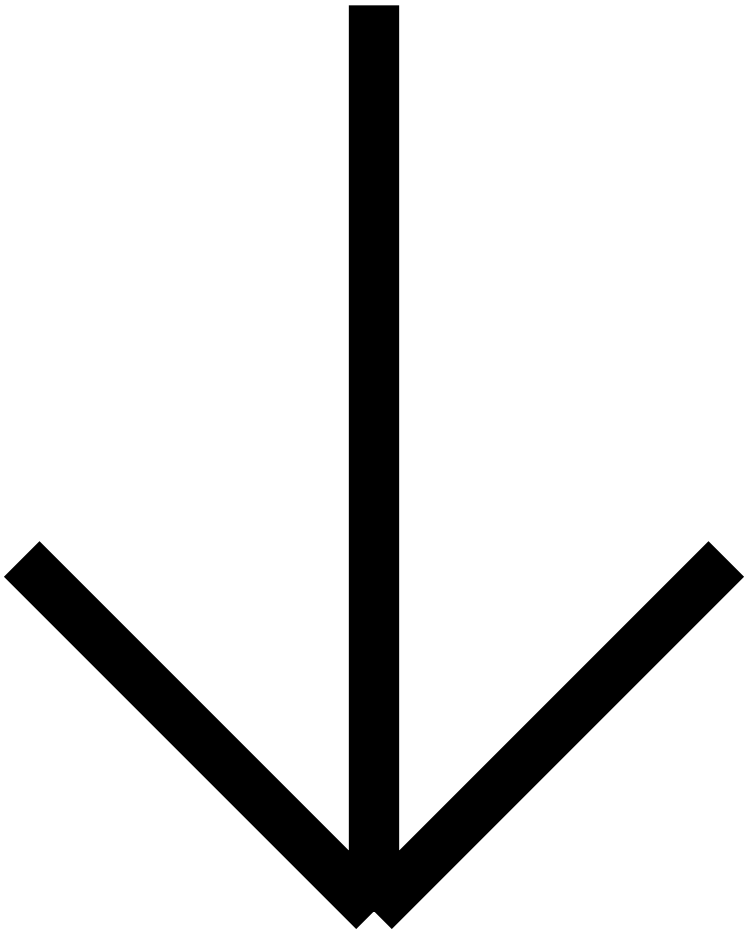
const Hero: React.FC = ({ openChat }) => { return (

Asistente Conversacional Inteligente y Rápido

Experimenta la nueva generación de asistencia automatizada. Construido con una arquitectura serverless para un rendimiento y escalabilidad inigualables.

Chatea ahora

Saber más



); };

export default Hero;

File: src__components__landing__HowItWorks.tsx__.txt

HowItWorks.tsx

import React from 'react';

const HowItWorks: React.FC = () => { const steps = [{ name: 'Escribe tu pregunta', description: 'Usa el chat para hacer lo que necesites.', icon: '💬' }, { name: 'Procesamiento Inteligente', description: 'Nuestro backend analiza tu consulta en tiempo real.', icon: '⚙️' }, { name: 'Recibe una Respuesta', description: 'Obtén la respuesta precisa y rápida en segundos.', icon: '☑️' },];

return (

¿Cómo Funciona?

Un proceso simple en tres pasos para obtener la información que buscas.

{steps.map((step, index) => (
{step.icon}

`{ Paso ${index + 1}: ${step.name} }`

{step.description}

))}
); };

export default HowItWorks;

File: src__components__landing__MinibotForm.tsx__.txt

MinibotForm.tsx

import React, { useState } from 'react';

interface FAQ { id: string; question: string; answer: string; }

const MinibotForm: React.FC = () => { // ---------- // Estado para mostrar/ocultar formulario // ---------- const [showForm, setShowForm] = useState(false);

// ---------- // Datos de contacto // ---------- const [contactName, setContactName] = useState(''); const [contactEmail, setContactEmail] = useState(''); const [company, setCompany] = useState(''); const [website, setWebsite] = useState(''); const [message, setMessage] = useState('');

// ---------- // Configuración del bot // ---------- const [clientId, setClientId] = useState(''); const [botName, setBotName] = useState(''); const [defaultResponse, setDefaultResponse] = useState('Lo siento, no he entendido tu pregunta. ¿Podrías reformularla?');

const [useGPT, setUseGPT] = useState(false);

const [faqs, setFaqs] = useState([{ id: 'faq_01', question: '', answer: '' }]);

const [submitted, setSubmitted] = useState(false); const [loading, setLoading] = useState(false);

// ---------- // Handlers // ---------- const addFAQ = () => { setFaqs([...faqs, { id: `faq_${String(faqs.length + 1).padStart(2, '0')}`, question: '', answer: '' }]); };

const updateFAQ = (index: number, field: 'question' | 'answer', value: string) => { const updated = [...faqs]; updated[index][field] = value; setFaqs(updated); };

const handleSubmit = async () => { setLoading(true);

```
const payload = {
  contact: {
    name: contactName,
    email: contactEmail,
    company,
    website,
    message
  },
  config: {
    client: {
      clientId,
      name: botName,
      domain: website,
      active: true,
      llm: {
        enabled: useGPT
      }
    },
    chatbot_config: {
      default: {
        value: defaultResponse
      }
    },
    chatbot_responses: faqs
      .filter(f => f.question && f.answer)
      .map((f, index) => ({
        id: f.id,
        question: f.question,
        answer: f.answer,
        active: true,
        order: index + 1
      }))
  }
};

try {
```

```
    await fetch('/requestMiniBot', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(payload)
    });

    setSubmitted(true);
  } catch (error) {
    console.error('Error enviando solicitud', error);
    alert('Ocurrió un error al enviar la solicitud.');
  } finally {
    setLoading(false);
  }
```

};

// ---------- // UI // ---------- if (submitted) { return (

¡Solicitud enviada! 🎉

Hemos recibido tu solicitud. Nuestro equipo se pondrá en contacto contigo muy pronto para activar tu MiniBot.

); }

return (

Solicita tu MiniBot para tu Sitio Web

```
    <p className="text-gray-600 text-center mb-8">
      Completa el formulario y nos pondremos en contacto contigo muy pronto!
    </p>

    {/* Botón para mostrar formulario - solo visible cuando showForm es false */
```

```
  {!showForm ? (
    <div className="text-center">
      <button
        onClick={() => setShowForm(true)}
        className="bg-gradient-to-r from-blue-600 to-indigo-600 text-white px-
      >
        Diligenciar Formulario de Solicitud
      </button>
    </div>
  ) : (
    // Formulario con animación de aparición
    <div className="animate-fadeIn space-y-8">
      {/* Contacto */}
      <div>
        <h3 className="text-xl font-semibold mb-4 text-gray-800">Información d
        <div className="grid grid-cols-1 md:grid-cols-2 gap-4 mb-6">
          <div>
            <label className="block text-sm font-medium text-gray-700 mb-1">
              Nombre completo *
            </label>
            <input
              className="border border-gray-300 p-3 rounded-lg w-full focus:ri
              placeholder="Nombre completo"
              value={contactName}
              onChange={e => setContactName(e.target.value)}
              required
            />
          </div>
          <div>
            <label className="block text-sm font-medium text-gray-700 mb-1">
              Correo electrónico *
            </label>
            <input
              className="border border-gray-300 p-3 rounded-lg w-full focus:ri
              placeholder="Correo electrónico"
              type="email"
              value={contactEmail}
```

```
              onChange={e => setContactEmail(e.target.value)}
              required
            />
          </div>
          <div>
            <label className="block text-sm font-medium text-gray-700 mb-1">
              Empresa / Proyecto
            </label>
            <input
              className="border border-gray-300 p-3 rounded-lg w-full focus:ri
              placeholder="Empresa / Proyecto"
              value={company}
              onChange={e => setCompany(e.target.value)}
            />
          </div>
          <div>
            <label className="block text-sm font-medium text-gray-700 mb-1">
              URL del Sitio web donde quieres tu Chat Bot
            </label>
            <input
              className="border border-gray-300 p-3 rounded-lg w-full focus:ri
              placeholder="https://tusitio.com"
              type="url"
              value={website}
              onChange={e => setWebsite(e.target.value)}
            />
          </div>
        </div>

        <div>
          <label className="block text-sm font-medium text-gray-700 mb-1">
            Mensaje adicional (opcional)
          </label>
          <textarea
            className="border border-gray-300 p-3 rounded-lg w-full focus:ring
            rows={3}
            placeholder="Cuéntanos más sobre lo que necesitas..."
```

```
            value={message}
            onChange={e => setMessage(e.target.value)}
          />
        </div>
      </div>

      {/* Configuración del Bot */}
      <div>
        <h3 className="text-xl font-semibold mb-4 text-gray-800">Configuración
        <div className="grid grid-cols-1 md:grid-cols-2 gap-4 mb-6">
          <div>
            <label className="block text-sm font-medium text-gray-700 mb-1">
              Coloca aquí el nombre de tu empresa.
            </label>
            <input
              className="border border-gray-300 p-3 rounded-lg w-full focus:ri
              placeholder="mi-sitio-bot"
              value={clientId}
              onChange={e => setClientId(e.target.value)}
              required
            />
            <p className="text-xs text-gray-500 mt-1">Identificador único para
          </div>
          <div>
            <label className="block text-sm font-medium text-gray-700 mb-1">
              Qué nombre quieres darle a tu Bot?
            </label>
            <input
              className="border border-gray-300 p-3 rounded-lg w-full focus:ri
              placeholder="Ej: Asistente XYZ"
              value={botName}
              onChange={e => setBotName(e.target.value)}
              required
            />
          </div>
        </div>
```

```
        <div className="mb-6">
          <label className="block text-sm font-medium text-gray-700 mb-1">
            Agrega en este espacio el mensaje que quieres que tu Bot responda
          </label>
          <textarea
            className="border border-gray-300 p-3 rounded-lg w-full focus:ring
            rows={3}
            placeholder="Lo siento, no he entendido tu pregunta. ¿Podrías refo
            value={defaultResponse}
            onChange={e => setDefaultResponse(e.target.value)}
            required
          />
        </div>

        <div className="mb-6">
          <label className="flex items-center gap-2 cursor-pointer">
            <input
              type="checkbox"
              checked={useGPT}
              onChange={() => setUseGPT(!useGPT)}
              className="w-4 h-4 text-blue-600 rounded focus:ring-blue-500"
            />
            <span className="text-gray-700">Activar GPT (recomendado para resp
          </label>
        </div>
      </div>

      {/* FAQs */}
      <div>
        <div className="flex justify-between items-center mb-4">
          <h3 className="text-xl font-semibold text-gray-800">Preguntas y resp
          <button
            onClick={addFAQ}
            className="text-blue-600 hover:text-blue-800 font-mediu flex item
          >
            <svg className="w-5 h-5" fill="none" stroke="currentColor" viewBox
              <path strokeLinecap="round" strokeLinejoin="round" strokeWidth="
```

```
            </svg>
            Agrega la posible pregunta que podría hacer tu cliente
          </button>
        </div>

        <div className="space-y-4">
          {faqs.map((faq, index) => (
            <div key={faq.id} className="border border-gray-200 rounded-lg p-4
              <div className="mb-3">
                <label className="block text-sm font-medium text-gray-700 mb-1
                  Pregunta {index + 1}
                </label>
                <input
                  className="border border-gray-300 p-3 rounded-lg w-full focu
                  placeholder="¿Cuál es el horario de atención?"
                  value={faq.question}
                  onChange={e => updateFAQ(index, 'question', e.target.value)}
                />
              </div>
              <div>
                <label className="block text-sm font-medium text-gray-700 mb-1
                  Aquí registra la Respuesta que tu Bot le dará a tu cliente f
                </label>
                <textarea
                  className="border border-gray-300 p-3 rounded-lg w-full focu
                  rows={2}
                  placeholder="Nuestro horario de atención es de lunes a vierr
                  value={faq.answer}
                  onChange={e => updateFAQ(index, 'answer', e.target.value)}
                />
              </div>
            </div>
          ))}
        </div>
      </div>

      {/* Botón de envío */}
```

```
      <div className="pt-6 border-t border-gray-200">
        <div className="flex flex-col sm:flex-row gap-4 justify-end">
          <button
            onClick={() => setShowForm(false)}
            className="px-6 py-3 border border-gray-300 text-gray-700 rounded-
          >
            Cancelar
          </button>
          <button
            disabled={loading}
            onClick={handleSubmit}
            className="bg-gradient-to-r from-green-600 to-emerald-600 text-whi
          >
            {loading ? (
              <span className="flex items-center justify-center gap-2">
                <svg className="animate-spin h-5 w-5 text-white" fill="none" v
                  <circle className="opacity-25" cx="12" cy="12" r="10" stroke
                  <path className="opacity-75" fill="currentColor" d="M4 12a8
                </svg>
                Enviando...
              </span>
            ) : 'Enviar Solicitud de MiniBot'}
          </button>
        </div>
      </div>
    </div>
  )}
</section>
```

); };

export default MinibotForm;

File: src__components__landing__TechStack.tsx__.txt

TechStack.tsx

import React from 'react';

const TechStack: React.FC = () => { const tech = [{ name: 'React', description: 'Una librería para construir interfaces de usuario.' }, { name: 'TypeScript', description: 'JavaScript con tipado estático para un código más robusto.' }, { name: 'Firebase', description: 'Plataforma para desarrollar apps web y móviles de alta calidad.' }, { name: 'Tailwind CSS', description: 'Framework de CSS para crear diseños a medida rápidamente.' },];

return (

Potenciado por Tecnología de Punta

Construido con un stack moderno y eficiente para garantizar el mejor rendimiento.

{tech.map((t) => (
{t.name.charAt(0)}

{t.name}

{t.description}

))}
); };

export default TechStack;

File: src__firebase.ts__.txt

firebase.ts

import { initializeApp } from "firebase/app";

const firebaseConfig = { apiKey: process.env.REACT_APP_FIREBASE_API_KEY!, authDomain: process.env.REACT_APP_FIREBASE_AUTH_DOMAIN!, projectId: process.env.REACT_APP_FIREBASE_PROJECT_ID!, storageBucket: process.env.REACT_APP_FIREBASE_STORAGE_BUCKET!, messagingSenderId: process.env.REACT_APP_FIREBASE_MESSAGING_SENDER_ID!, appId: process.env.REACT_APP_FIREBASE_APP_ID!, };

export const firebaseApp = initializeApp(firebaseConfig);

File: src__index.css__.txt

index.css

@tailwind base; @tailwind components; @tailwind utilities;

body { margin: 0; font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen', 'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue', sans-serif; -webkit-font-smoothing: antialiased; -moz-osx-font-smoothing: grayscale; background-color: #f5f5f5; }

code { font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New', monospace; }

/ *Animación para el indicador de escritura* / @keyframes typing { 0% { content: '.'; } 33% { content: '..'; } 66% { content: '...'; } }

.typing-indicator::after { content: ''; animation: typing 1.5s infinite; }

@keyframes fadeIn { from { opacity: 0; transform: translateY(10px); } to { opacity: 1; transform: translateY(0); } }

.animate-fadeIn { animation: fadeIn 0.5s ease-out; }

File: src__index.tsx__.txt

index.tsx

import React from 'react'; import ReactDOM from 'react-dom/client'; import './index.css'; import App from './App.tsx';

const root = ReactDOM.createRoot(document.getElementById('root') as HTMLElement); root.render( );

File: src__services__chatService.ts__.txt

chatService.ts

// src/services/chatService.ts

import { ChatRequest, ChatResponse, MiniBotRuntimeConfig } from '../types';

// ---------- // Runtime config (inyectada por index.html o widget) // ----------

declare global { interface Window { **SAM_MINIBOT_CONFIG**?: MiniBotRuntimeConfig; } }

const runtimeConfig: MiniBotRuntimeConfig = window.**SAM_MINIBOT_CONFIG** || { clientId: 'sam-minibot-prototipe', };

const CLIENT_ID = runtimeConfig.clientId;

// -- // API BASE RESOLUTION (🔥 CLAVE DEL PROBLEMA) // --

function resolveApiBase(): string { // 1️⃣ Widget / runtime injection (clientes externos) if (runtimeConfig.apiBase) { return runtimeConfig.apiBase; }

// 2️⃣ Build-time env (React) if (process.env.REACT_APP_API_BASE) { return process.env.REACT_APP_API_BASE; }

// 3️⃣ Firebase Hosting (producción) if (typeof window !== 'undefined' && window.location.hostname.endsWith('.web.app')) { return 'https://us-central1-mini-bot-7a21d.cloudfunctions.net'; }

// 4️⃣ Local / fallback (emulador) return 'http://127.0.0.1:5001/mini-bot-7a21d/us-central1'; }

const API_BASE = resolveApiBase(); const CHAT_ENDPOINT = `${API_BASE}/chatbot`;

// -- // Servicio principal // --

export const sendMessage = async (message: string, sessionId?: string): Promise => { try { const requestData: ChatRequest = { clientId: CLIENT_ID, message, sessionId: sessionId || generateSessionId(), channel: 'web', };

```
const response = await fetch(CHAT_ENDPOINT, {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(requestData),
});

if (!response.ok) {
  const errorText = await response.text();
  throw new Error(
    `[ChatService] Error ${response.status}: ${errorText}`
```

```
  );
}

return (await response.json()) as ChatResponse;
```

} catch (error) { console.error('[ChatService] Error al enviar mensaje:', error); throw error; } };

// --- // Utils // ---

const generateSessionId = (): string => Math.random().toString(36).substring(2, 15) + Math.random().toString(36).substring(2, 15);

File: src__setupTests.ts__.txt

setupTests.ts

File: src__types__index.ts__.txt

index.ts

// src/types/index.ts

// ---------------------------- // UI types // ---------------------------- export interface ChatMessage { id: string; text: string; sender: 'user' | 'bot'; timestamp: Date; meta?: { confidence?: number; matched?: boolean; source?: 'faq' | 'llm' | 'system'; }; }

// ---------------------------- // API contract // ---------------------------- export interface ChatRequest { clientId: string; message: string; sessionId?: string; // Futuro: metadata del canal / widget channel?: 'web' | 'widget' | 'api'; locale?: string; // 'es', 'en', etc. }

export interface ChatResponse { response: string; sessionId: string; confidence: number; matched: boolean;

// nuestro backend devuelve ISO string (estable) timestamp: string;

// Futuro: trazabilidad del motor source?: 'faq' | 'llm'; model?: string; // ej: 'gpt-4o-mini' / 'gemini-1.5-flash' }

// ---------------------------- // SaaS runtime config (frontend) // ---------------------------- export interface MiniBotRuntimeConfig { clientId: string; apiBase?: string; // ej: https://tudominio.com (si no es mismo origen) mode?: 'faq' | 'hybrid' | 'llm'; }

File: test.json__.txt

test.json

{ "client": { "clientId": "test" } }
