
This is the **published version** of the bachelor thesis:

Crisan, David Alexandru. *Desarrollo de un chat bot basado en IA generativa con integración en un sitio web*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/334378>

under the terms of the  license.

Desarrollo de un chat bot basado en IA generativa con integración en un sitio web

David Alexandru Crisan

Resumen— Este trabajo final de grado detalla el diseño, la implementación y el despliegue en producción de una plataforma SaaS (Software as a Service), concebida para la democratización de la inteligencia artificial en entornos empresariales. Aplicando la técnica de Generación Aumentada por Recuperación (RAG), el sistema permite a las organizaciones extraer conocimiento directamente de sus documentos privados (PDF), eludiendo la necesidad de aplicar costosos procesos de reentrenamiento sobre modelos fundacionales genéricos. La solución se materializa a través de un ecosistema que integra un servidor asíncrono en Node.js, persistencia vectorial en Pinecone, y una interfaz reactiva en React. El ciclo de vida del producto culmina con la inyección de un Widget autónomo encapsulado en Vanilla JS. La validación en la nube (AWS EC2) certifica la solidez de la arquitectura, operada de manera segura tras un proxy inverso Nginx, con certificados SSL y políticas CORS adaptativas que solucionan vulnerabilidades críticas de seguridad web como el Contenido Mixto (Mixed Content).

Palabras clave— Inteligencia Artificial, RAG, SaaS, React, Node.js, Nginx, Almacenamiento Vectorial, CORS, Seguridad Web, AWS.

Abstract— This graduation thesis details the design, implementation, and production deployment of a multi-tenant SaaS platform conceived to democratize corporate artificial intelligence. Applying the Retrieval-Augmented Generation (RAG) paradigm, the system allows organizations to extract knowledge from their private structured documents (PDFs), bypassing the need to apply expensive retraining processes on foundational models. The solution is materialized through an ecosystem integrating an asynchronous Node.js server, vector persistence in Pinecone, and a reactive user interface in React. The product lifecycle culminates with the injection of an autonomous Vanilla JS Widget. Production validation within AWS Cloud certifies the solidity of the architecture, operated securely behind an Nginx reverse proxy, featuring SSL certificates and adaptive CORS policies that successfully solve critical web security vulnerabilities such as Mixed Content.

Keywords— Generative AI, RAG, SaaS, React, Node.js, Nginx, Vector Store, CORS, Web Security, AWS.

1 INTRODUCCIÓN

EL rápido avance de la Inteligencia Artificial Generativa y de los Grandes Modelos de Lenguaje (LLMs) ha transformado radicalmente la forma en la que interactuamos con la tecnología. Herramientas como

ChatGPT han demostrado de manera irrefutable una capacidad enorme para procesar lenguaje natural, razonar sobre problemas complejos y mantener conversaciones fluidas. Sin embargo, cuando las empresas intentan adoptar esta tecnología para potenciar sus flujos de trabajo diarios, se encuentran con una barrera técnica muy grande: la falta de conocimiento sobre sus propios datos privados.

Los modelos fundacionales se entrenan con datos públicos extraídos de Internet hasta una fecha de corte concreta. Por lo tanto, desconocen por completo los manuales de operaciones, los catálogos de productos o las bases de datos internas de una empresa en particular. Si a un modelo genérico se le hace una pregunta muy específica sobre el funciona-

- E-mail de contacto: 1671090@uab.cat
- NIU del Estudiante: 1671090
- Menció: Ingenieria en Tecnologías de la Información
- Trabajo tutorizado por: Marc Ortega
- Curso 2025/26

miento de un negocio privado, este suele intentar complacer al usuario inventándose la respuesta, un fallo crítico que técnicamente se conoce como 'alucinación'.

Para solucionar este problema de raíz sin tener que reen-trenar el modelo desde cero (un proceso conocido como *Fine-Tuning*, que resulta demasiado costoso, lento e inviable para la mayoría de las pequeñas y medianas empresas), la industria de la ingeniería de software ha adoptado una técnica llamada Generación Aumentada por Recuperación (RAG).

La técnica RAG consiste en interceptar la pregunta del usuario, buscar la respuesta en los documentos privados indexados de la empresa y enviarle esos fragmentos a la inteligencia artificial como 'contexto' adicional en su consulta. De esta forma, obligamos al sistema a basar sus respuestas estrictamente en los datos reales aportados en tiempo real.

El problema fundamental que motiva este Trabajo de Final de Grado (TFG) es la escasez de herramientas fáciles de usar para el usuario final. Montar un sistema RAG desde cero requiere hoy en día gestionar bases de datos vectoriales, desarrollar conexiones API, asegurar servidores web y programar interfaces de chat a medida. Este proyecto propone agrupar toda esa complejidad técnica creando una plataforma web integral de tipo Software as a Service (SaaS). Esta plataforma permite a cualquier persona registrarse, crear, entrenar y publicar un asistente experto en sus propios datos en cuestión de minutos, todo ello sin la necesidad de escribir código.

2 ESTRUCTURA DEL INFORME

El documento se organiza de la siguiente manera:

En la sección de objetivos se exponen las metas principales y secundarias del proyecto, desglosando los pasos técnicos necesarios para construir el ecosistema de software, desde el servidor de base de datos hasta el despliegue en la nube.

En el estado del arte se analizan las tecnologías actuales y se justifica por qué es necesaria una plataforma unificada frente a las soluciones fragmentadas para desarrolladores que existen hoy en día.

En la metodología y planificación se detalla cómo se ha organizado el trabajo siguiendo un flujo ágil e iterativo. Se explica la decisión estratégica de priorizar de forma absoluta la programación del servidor (enfoque Backend-First).

En el desarrollo del sistema se desgranar las implementaciones técnicas realizadas. Se explica la arquitectura cliente-servidor, las soluciones aplicadas frente a los problemas encontrados con las dependencias de Node.js, la creación del panel de control en React y la ingeniería detrás del Widget autónomo. Aquí se incluyen fragmentos de código fuente esenciales.

En el apartado de despliegue Cloud se documenta a fondo la puesta en producción del sistema en servidores de Amazon Web Services (AWS). Se explica la configuración del Proxy Inverso con Nginx, la seguridad perimetral lograda con cer-

tificados TLS/SSL y cómo se solucionaron los bloqueos de red impuestos por los navegadores modernos (CORS y Contenido Mixto).

Finalmente, en las conclusiones se valora el cumplimiento de los objetivos y se proponen mejoras. También se añade un apartado detallando el uso de la Inteligencia Artificial como herramienta de apoyo durante el transcurso del trabajo.

3 OBJETIVOS

En este apartado se define de manera clara el objetivo general del proyecto y se desglosan las metas técnicas específicas que han guiado cada una de las fases de desarrollo y despliegue del sistema.

3.1 Objetivo General

El objetivo principal de este trabajo es el diseño, la programación y la validación en un entorno real de producción en la nube (Cloud) de una plataforma de Software as a Service (SaaS) completa. Esta herramienta debe permitir crear y entrenar de forma automatizada chatbots basados en IA generativa, facilitando su integración en páginas web de terceros a través de un Widget inyectable de forma totalmente transparente.

3.2 Objetivos Específicos

Para lograr la consolidación del sistema global, el ciclo de vida del proyecto se dividió en los siguientes objetivos específicos:

- **Gestión de Usuarios y Seguridad:** Implementar un sistema de autenticación asíncrono y seguro basado en tokens criptográficos (JWT) que permita a los usuarios registrarse y que garantice que los documentos subidos por cada cliente permanezcan aislados herméticamente.
- **Infraestructura de Ingesta RAG:** Construir el motor algorítmico capaz de leer archivos estructurados en formato PDF, dividir el texto en fragmentos lógicos, convertirlos en vectores matemáticos (*embeddings*) mediante OpenAI y guardarlos en una base de datos vectorial.
- **Interfaz de Usuario (Dashboard):** Desarrollar una aplicación web de página única (SPA) moderna, fluida y reactiva que funcione como el centro de control principal para gestionar los bots, definir su personalidad (System Prompts) y observar el progreso de la subida de los archivos.
- **Desarrollo de Componente Integrable:** Programar una librería inyectable extremadamente ligera, redactada en lenguaje JavaScript puro (Vanilla JS), capaz de incrustar una ventana de chat interactiva en cualquier

página web externa, previniendo herencias de estilo no deseadas.

- **Despliegue e Infraestructura Cloud:** Escalar la aplicación desde el entorno de desarrollo local hacia servidores remotos de AWS, asegurar las comunicaciones con certificados SSL/HTTPS, configurar un proxy inverso para enrutar el tráfico correctamente y solucionar las directivas de seguridad de los navegadores (CORS y Mixed Content).

4 ESTADO DEL ARTE

En el momento actual, el ecosistema de desarrollo de Inteligencia Artificial para empresas se encuentra notablemente fragmentado. Por un lado, tenemos a las corporaciones creadoras de los grandes modelos de lenguaje (tales como OpenAI o Anthropic), las cuales ofrecen el acceso a su IA a través de APIs de pago. Por otro lado, contamos con los proveedores de infraestructura, como las bases de datos vectoriales Pinecone, Milvus o ChromaDB, diseñadas para guardar las representaciones numéricas de los textos.

Para unir estos dos mundos y construir un sistema RAG útil, los programadores se apoyan habitualmente en librerías de código abierto como LangChain. El inconveniente es que estas herramientas están diseñadas exclusivamente para ingenieros de software: no proveen una interfaz visual de usuario, no incluyen sistemas de registro y carecen de un mecanismo para publicar el chatbot final en una web pública de cara al consumidor.

Frente a esta coyuntura de tecnologías dispersas, el presente trabajo plantea una solución arquitectónica unificada. El valor diferencial de la plataforma desarrollada reside en que encapsula todo el arduo proceso técnico subyacente detrás de un panel visual fácil de operar. Se constituye así como un producto terminado que derriba la barrera tecnológica para las empresas.

5 METODOLOGÍA Y PLANIFICACIÓN

Dada la envergadura del sistema y la cantidad de microservicios diferentes que debían comunicarse entre sí, se descartó de plano la utilización de una planificación tradicional en cascada. En su lugar, se adoptó una metodología de desarrollo ágil e iterativa, basando el avance del proyecto en la construcción de prototipos funcionales que se testeaban semana a semana.

5.1 El Enfoque Backend-First

La decisión de trabajo más crítica fue la priorización absoluta del desarrollo del servidor central (enfoque Backend-First). Se impuso como norma estricta del proyecto no diseñar ni maquetar ninguna interfaz gráfica hasta que el motor lógico RAG demostrara ser capaz de procesar un PDF y devolver una respuesta contextual correcta de forma completamente autónoma.

Para validar esta premisa en las etapas tempranas, se interactuó con la API directamente utilizando el software de pruebas HTTP, Thunder Client. Este procedimiento garantizó que el 'cerebro' del sistema operara sin fisuras. Gracias a este aislamiento inicial, cuando surgieron complicaciones más adelante al integrar la interfaz de React, fue inmediato determinar que los fallos provenían de la vista o de la red, y no de la lógica de la IA.

6 DESARROLLO DEL SISTEMA

Esta sección detalla cómo se ha programado el proyecto, describiendo la arquitectura elegida y explicando las resoluciones técnicas aplicadas para superar los obstáculos que surgieron durante la creación del código.

6.1 Arquitectura y Modelado de Datos

La plataforma se rige por una arquitectura Cliente-Servidor. La capa del servidor (Backend), que gestiona toda la carga computacional, se desarrolló con Node.js y el framework Express. Esta elección tecnológica resulta ideal para este tipo de escenarios SaaS, ya que el modelo asíncrono impulsado por eventos permite procesar múltiples peticiones de usuarios simultáneamente sin bloquear el hilo principal.

La persistencia de los datos relacionales (como las cuentas de usuario y la configuración de los agentes virtuales) se gestiona mediante un motor MySQL. Para interactuar eficientemente con esta base de datos desde JavaScript, se integró el ORM (Object-Relational Mapping) Sequelize.

Listing 1: Mapeo y relaciones del modelo de datos de bots usando Sequelize.

```
const Bot = sequelize.define('Bot', {
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  name: {
    type: DataTypes.STRING,
    allowNull: false
  },
  systemPrompt: {
    type: DataTypes.TEXT,
    allowNull: false
  },
  userId: {
    type: DataTypes.INTEGER,
    references: {
      model: User,
      key: 'id'
    }
  }
}, {
  tableName: 'bots',
  freezeTableName: true
});

User.hasMany(Bot, { foreignKey: 'userId' });
Bot.belongsTo(User, { foreignKey: 'userId' });

module.exports = Bot;
```

Como muestra el código anterior, Sequelize permite estructurar y relacionar las tablas (1:N) como objetos de JavaScript. La enorme ventaja de usar este ORM quedó patente a mitad del proyecto, cuando hubo que añadir el campo

'nombre' al registro de usuarios; la tabla física se actualizó mediante el comando de sincronización en caliente, preservando intacta la información previa.

En cuanto a la seguridad de las cuentas, se descartó el almacenamiento de sesiones en memoria y se articuló la protección mediante JSON Web Tokens (JWT).

Listing 2: Middleware para la interceptación y validación de tokens JWT en rutas privadas.

```
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ message: 'No hay token',
      permiso denegado' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = decoded;
    next();
  } catch (error) {
    res.status(403).json({ message: 'Token no válido' });
  }
};
```

Este middleware intercepta todas las peticiones dirigidas a zonas privadas del panel y valida la integridad de la firma en cuestión de milisegundos, manteniendo el servidor ligero y libre de estados.

6.2 El Motor RAG y Fallos de Dependencias

El proceso de Generación Aumentada por Recuperación constituyó el desafío de programación más laborioso. El flujo técnico exigía que el servidor recibiese un archivo binario desde el navegador, extrajese su texto plano, lo dividiese en partes lógicas solapadas, las enviase a OpenAI para calcular sus vectores matemáticos, y finalmente los persistiese en la base de datos espacial Pinecone.

A lo largo de la ejecución de esta fase, el ecosistema sufrió dos fallos críticos debidos a la incompatibilidad del software de terceros. El primer incidente ocurrió al intentar extraer texto de los archivos PDF. El servidor detenía su ejecución lanzando una excepción fatal por aborto del módulo (`AbortException`). Al someter los registros de error a un análisis exhaustivo, se dictaminó que la librería base utilizada, *pdf-parse*, operaba bajo un sistema de empaquetado heredado (CommonJS) que chocaba arquitectónicamente con el moderno cargador de módulos integrado en Node.js v24. La resolución técnica obligó a desinstalar la librería y a sustituirla por una bifurcación comunitaria moderna denominada *pdf-extraction*, restaurando así la funcionalidad del módulo de ingesta.

El segundo fallo resultó ser aún más limitante. El código oficial de conexión remota (SDK) proporcionado por Pinecone albergaba errores internos de validación estricta al ejecutarse en el entorno Node v24, lo que bloqueaba la subida de los vectores procesados a la nube.

Para no paralizar el cronograma, se aplicó ingeniería inversa

y se tomó la decisión de crear un 'bypass' estructural. Se prescindió de la librería oficial y se desarrolló un conector directo de bajo nivel mediante peticiones HTTP nativas.

Listing 3: Bypass HTTP nativo para evadir el defecto de validación en el SDK de Pinecone.

```
const response = await fetch('https://${host}/query', {
  method: 'POST',
  headers: {
    'Api-Key': process.env.PINECONE_API_KEY.trim(),
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({
    vector: vectorDeOpenAI,
    topK: 3,
    filter: { botId: { "$eq": botId.toString() } },
    includeMetadata: true
  })
});
```

Como se observa en el Listing 3, esta contramedida técnica requirió construir y formatear manualmente el cuerpo JSON exigido por los protocolos de la API REST del proveedor y enviarlo de forma directa. La adopción de esta estrategia garantizó el éxito del guardado del conocimiento corporativo, salvando los graves defectos del código ajeno.

6.3 Migración a React y UX Defensiva

Originalmente, el planteamiento de este trabajo contemplaba la utilización del framework Vue.js para el panel de administración. Sin embargo, a medida que la plataforma se tornó más robusta, se hizo evidente que la orquestación simultánea de variables en tiempo real (el progreso porcentual de la subida de los PDF, las respuestas de la IA y el control de la vigencia de la sesión) demandaba un paradigma de gestión de estados mucho más estricto. Se procedió a migrar todo el entorno del cliente hacia la librería React, apoyada por la herramienta de construcción Vite.

El uso de los patrones funcionales de React ("Hooks") facilitó la implementación de dinámicas de experiencia de usuario (UX) profesionales. Se implementó una lógica combinada con `useRef` y `useEffect` destinada a desplazar la ventana de visualización hacia abajo automáticamente (*auto-scroll*).

Listing 4: Lógica de React para el desplazamiento automático y reactivo (auto-scroll) del chat.

```
const messagesEndRef = useRef(null);

useEffect(() => {
  messagesEndRef.current?.scrollIntoView({
    behavior: 'smooth'
  });
}, [messages, isLoading]);
```

Paralelamente, se incrustaron técnicas de programación defensiva a nivel de cliente para salvaguardar los recursos del servidor. En el formulario de autenticación, el código intercepta las pulsaciones y evalúa si la clave proporcionada en el campo 'Contraseña' y en la casilla de confirmación difieren. En caso afirmativo, el envío se bloquea instantáneamente y se despacha una alerta visual, evitando peticiones de red innecesarias.

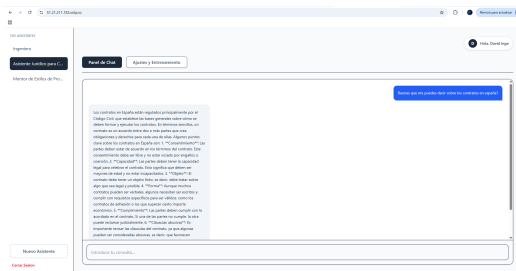


Figura 1: Vista detallada del Dashboard de gestión del Chatbot.

6.4 Ingeniería del Widget Inyectable

Para que el producto de software tuviese aplicación en el mundo real, resultaba imperativo dotar a los clientes de un mecanismo para que pudiesen incrustar a su asistente experto en sus propias plataformas web corporativas. Este artefacto final es el **Widget**.

Para evitar, bajo cualquier circunstancia, que el código de nuestra herramienta fracturase el diseño de la web del cliente, se adoptó una política de aislamiento radical. El Widget se programó exclusivamente en lenguaje JavaScript primigenio (Vanilla JS), absteniéndose del uso de dependencias de empaquetado pesadas. Su funcionamiento se blindó envolviendo la totalidad del flujo conversacional dentro de una Expresión de Función Invocada Inmediatamente (IIFE).

Listing 5: Estructura IIFE del Widget y algoritmo de lectura dinámica de atributos.

```
(function() {
  const currentScript = document.currentScript ||
    document.querySelector('script[src*="widget.js"]');
  const botId = currentScript.getAttribute('data-bot-id');
  const BACKEND_URL = "/api/";

  if (!botId) return console.error("Error: Falta el ID del bot");

  // Inyección de CSS aislado y lógica del chat...
})();
```

En lugar de requerir la importación de archivos de estilo externos (archivos CSS) que corren el riesgo de ser sobrescritos por reglas de mayor especificidad en la web anfitriona, el script computa dinámicamente sus propias reglas visuales y las incrusta directamente en la cabecera del documento ajeno. Además, el Widget identifica de manera autónoma a qué asistente virtual RAG del servidor debe consultar leyendo el atributo `data-bot-id`, garantizando una interconexión transparente.

7 DESPLIEGUE CLOUD Y SEGURIDAD

La culminación práctica de este Trabajo de Final de Grado se materializó en la transición del software desde el entorno de desarrollo local hacia un ecosistema de hardware real, conectado ininterrumpidamente a Internet.

La solución integral se provisionó desplegando una máqui-

na virtual elástica EC2 dentro del entorno Cloud de Amazon Web Services (AWS), armada con el sistema operativo Ubuntu Linux. Con el firme propósito de garantizar que el hilo del programa Node.js persistiera encendido frente a errores y se relanzara de manera autónoma, se orquestó la aplicación mediante el gestor de subprocesos PM2.

7.1 Proxy Inverso (Nginx) y Certificados SSL

Siguiendo estrictas directrices de ciberseguridad, se concluyó que el puerto utilizado por el servidor backend (el puerto 3000) jamás debía estar expuesto de forma directa a la red pública. Para erigir una barrera sólida de mitigación de ataques y ordenación del tráfico, se instaló Nginx, configurado para actuar en modalidad de Proxy Inverso avanzado.

La labor de este proxy es operar como un 'portero': recibe todo el tráfico que llama a los puertos estandarizados de navegación web (el puerto 80 para HTTP y el puerto 443 para HTTPS) y, posteriormente, redirige de forma transparente la solicitud hacia la aplicación recluida. Durante las primeras comprobaciones de este enrutamiento, la aplicación colapsó casi en su totalidad, arrojando devoluciones masivas de códigos de error por ruta no encontrada (404 Not Found).

Al diseccionar los registros de acceso del servidor Nginx, se descubrió la naturaleza de la anomalía. Nginx estaba enviando la cadena de la URL de forma literal. Si el frontend realizaba una consulta de tipo `/api/bots`, Nginx le entregaba al servidor la ruta adherida cruda, obligando a Node.js a intentar resolver controladores redundantes. La solución arquitectónica fue precisa: añadir el carácter de la barra diagonal inclinada al final de la directiva de redirección principal.

Listing 6: Configuración del bloque de Nginx para realizar el truncamiento de prefijos de enrutamiento.

```
location /api/ {
  proxy_pass http://localhost:3000/;
  proxy_http_version 1.1;
  proxy_set_header Upgrade $http_upgrade;
  proxy_set_header Connection 'upgrade';
  proxy_set_header Host $host;
}
```

Tal y como ilustra el Listing 6, la simple colocación del delimitador final activa de inmediato un mecanismo interno de Nginx encargado de 'recortar' del URI el segmento correspondiente a `/api/`, despachando la ruta limpia hacia la cola de eventos de Node.js y restituyendo por completo la interoperabilidad de la herramienta.

Para infundir confianza en los usuarios y salvaguardar su privacidad era mandatorio aplicar una capa de cifrado sobre el canal de comunicaciones (TLS). Empleando el servicio de asignación de dominios dinámicos `sslip.io` atado a la IP pública de AWS, se tramitó, expidió e instaló un certificado de seguridad criptográfica avalado por la autoridad Let's Encrypt mediante la utilidad automatizada Certbot. Merced a este proceso, toda la navegación y el envío de consultas de IA de la plataforma quedaron forzados a transitar enmascarados bajo el protocolo seguro HTTPS.

7.2 Políticas de Red: CORS y Mixed Content

Al enfrentar el proyecto a la prueba de fuego definitiva —la inserción del código del Widget autónomo en el esqueleto de un archivo HTML externo simulando la tienda virtual de un cliente comercial—, los mecanismos de barrera y protección inherentes a los navegadores modernos (como Google Chrome) bloquearon inmediatamente el funcionamiento del servicio de chat, inundando la consola con graves notificaciones de alerta de seguridad.

El primer bloqueo fulminante fue causado por una vulnerabilidad de red categorizada como **Contenido Mixto (Mixed Content)**. Como el entorno de pruebas HTML acababa de ser validado y protegido por el flamante certificado SSL que instalamos horas antes (cargando recursos sobre HTTPS), pero el viejo código estructural del Widget persistía en apuntar estáticamente hacia la dirección IP cruda de AWS valiéndose de un puerto no seguro (`http://...:3000`), el motor del navegador abortó drásticamente la transmisión para salvaguardar a su usuario de posibles vulnerabilidades **Man-in-the-Middle**.

La erradicación de este importante fallo se materializó reescribiendo la capa de red del Widget. Se extirparon todas las referencias absolutas a nombres de dominio e IPs, sustituyéndolas por llamadas a rutas relativas unificadas (`/api/chat`). De este modo, las peticiones conversacionales comenzaron a viajar heredando naturalmente la cobertura de encriptación impuesta por el Proxy Nginx.

El segundo bloqueo sistemático desatado fue originado por la archiconocida **Política CORS (Intercambio de Recursos de Origen Cruzado)**. Por motivos históricos de seguridad web, la totalidad de los navegadores comerciales se rigen por la regla del Mismo Origen (**Same-Origin Policy**), que prohíbe que un documento hipertexto exija datos a un dominio distinto al suyo propio. Dado que la esencia misma de un servicio SaaS es proveer e inyectar recursos en dominios foráneos, resultó imperioso forjar un conducto de autorización a nivel de red para salvar este impedimento.

Listing 7: Modificación en el empaquetador Vite para permitir la permeabilidad de orígenes cruzados.

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

export default defineConfig({
  plugins: [react()],
  server: {
    cors: {
      origin: "*", // Atributo para habilitar conexiones
        web externas
      methods: ["GET", "POST"]
    }
  }
})
```

Para sortear las restricciones, se reconfiguró la aplicación a dos niveles. Primero, se ordenó expresamente al servidor backend que impregnara las cabeceras HTTP salientes con los distintivos formales de aprobación (`Access-Control-Allow-Origin: *`), y se programó el middleware para que brindase una respuesta

afirmativa a las comprobaciones preliminares de seguridad exigidas por los clientes web. Asimismo, tal y como queda reflejado en el Listing 7, se ajustaron drásticamente las directivas de seguridad del compilador Vite para abrir y legitimar las compuertas de origen.

Como ajuste minucioso final de esta extensa etapa de despliegue Cloud, para certificar que el servidor perimetral Nginx dispondría siempre del permiso necesario para distribuir los archivos visuales a los visitantes sin desencadenar indeseables errores 403 Forbidden (Acceso Denegado), se maniobró con los permisos nucleares del sistema operativo en Ubuntu Linux. Se desposeyó del control de la carpeta pública al usuario base administrador de AWS (`ubuntu`), adjudicando los derechos de lectura a favor del usuario nativo encargado de correr los procesos del servidor web (`www-data`) apoyándose en el comando `chown -R www-data:www-data`. Al refinar los ficheros con las máscaras de seguridad Unix recomendadas para la industria (`chmod 755 y 644`), se logró dar por estabilizado el entorno de software en producción.

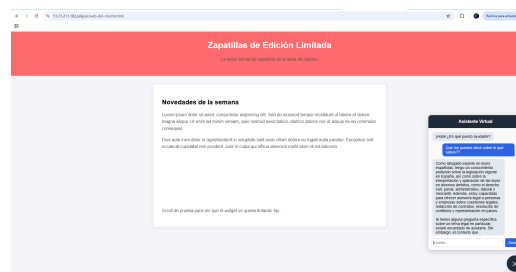


Figura 2: Integración del widget en el entorno de pruebas externo.

8 CONCLUSIONES

La complejidad técnica y el despliegue en la nube de este Trabajo de Final de Grado sirven para certificar con rotundidad el cumplimiento absoluto de todas las metas de diseño esbozadas en los documentos iniciales. A través de un riguroso e intenso proceso de ingeniería de software a gran escala, se ha logrado concebir, implementar de forma resolutiva, y estabilizar en entornos de producción una plataforma informática integral cimentada bajo la exitosa filosofía comercial Software as a Service.

La aportación técnica principal de este proyecto reside explícitamente en el insuperable nivel de abstracción ofrecido. El ecosistema informático diseñado empuja la democratización real del acceso corporativo de las pequeñas entidades a la potentísima Inteligencia Artificial Generativa. Ahora, miles de organizaciones carentes del capital humano y técnico especializado demandado hoy en día, pueden finalmente beneficiarse holgadamente de las ventajas intrínsecas aportadas por el paradigma resolutivo RAG. El cliente final de la plataforma se limita sencillamente a registrarse, subir sus manuales de empresa en formato PDF en un panel visual intuitivo y copiar una humilde línea de script para materializar un valioso activo digital de altísimo funcionamiento resolutivo en su portal.

Las duras resoluciones arquitectónicas tomadas con celeridad a lo largo de las distintas y exigentes fases de la evolución del proyecto denotan un enfoque analítico, reactivo y de gran agilidad. Destaca notoriamente la migración que se acometió hacia el ecosistema de React en busca de un control férreo para blindar estados complejos asíncronos en tiempo real aun habiendo trabajado ya con Vue, así como se ha de destacar la muy notable acción de ingeniería inversa y diseño de bypasses HTTP nativos aplicados para lograr soslayar el temible colapso interno originado por las fallidas dependencias de la librería *pdf-parse* en el moderno entorno Node.js v24. Este tipo de respuestas decididas y con enfoque resiliente ante la habitual incertidumbre técnica se tornan, indudablemente, en características vitales muy valiosas en el exigente ejercicio diario de la ingeniería informática en el mundo empresarial moderno. Los resultados de latencia constatados visualmente tras los análisis operativos observados en los despliegues del servidor virtual provisionado sobre AWS EC2 corroboran de manera solvente la incuestionable fiabilidad del diseño inicial, integrando a la postre un hermético ecosistema perimetral muy vigorosamente blindado frente a las severas normativas de la red web moderna (CORS y Contenido Mixto).

Por otra parte, avanzando a nivel logístico y funcional, resulta también una adición prioritaria perfeccionar el engranaje subyacente del motor RAG original con novísimos módulos autónomos extra para potenciar la fase fundamental de ingesta de conocimiento. Al respecto, y mediante el empleo rutinario de las técnicas orientadas al rastreo web dinámico e indexación asíncrona conocidas como **Web Scraping**. Esta fantástica ampliación técnica permitiría al sistema general tener la vital labor de mantener retroalimentado su contexto base de forma ininterrumpida y persistente, escaneando recurrentemente amplios portales de noticias recientes o bases de documentación online, complementando de forma idílica la ya existente herramienta basada en la carga estrictamente manual de documentos en el formato cerrado y rígido del estándar PDF.

A nivel de producto, se plantea una ruta evolutiva para la consolidación de la plataforma como solución de nivel empresarial. En primer lugar, se ha identificado la necesidad de la transición hacia un paradigma multimodal, integrando modelos de visión computacional para que el motor RAG pueda interpretar diagramas, tablas y contenido gráfico integrado en los documentos, eliminando la actual ceguera del sistema ante elementos no textuales. Paralelamente, se diseñaría la arquitectura para la implementación de un sistema de API Keys cifradas mediante hashing seguro, las cuales actuarán como mecanismo de validación externa, garantizando que la integración del widget en portales de terceros cumpla con los estándares más rigurosos de control de acceso, gestión de asistentes virtuales y trazabilidad en el consumo de recursos.

USO DE LA IA GENERATIVA

Acorde a las directrices de integridad académica establecidas por la institución, la Inteligencia Artificial generativa (específicamente, Gemini y ChatGPT) se ha empleado

de forma complementaria como herramienta de asistencia técnica y de redacción durante el desarrollo, operando en todo momento bajo estricta supervisión humana. Se garantiza formalmente que ninguna decisión sobre la arquitectura central ni sobre el diseño de las bases de datos ha sido delegada a estos modelos; la totalidad de la lógica de negocio y el código estructural han sido razonados e implementados íntegramente por el autor del proyecto.

En el ámbito de la depuración del código (debugging), Gemini desempeñó un papel crucial al analizar trazas de error (stack traces) en el entorno de Node.js. Su intervención permitió identificar rápidamente una incompatibilidad severa entre la librería original *pdf-parse* y el nuevo entorno Node v24, sugiriendo eficazmente su reemplazo por el fork modernizado *pdf-extraction*. Este diagnóstico certero evitó numerosas horas de ensayo y error manual, resolviendo el bloqueo del módulo de extracción de PDFs de forma inmediata.

Por su parte, ChatGPT se utilizó como acelerador en la capa del frontend, agilizando la creación de estructuras base (esqueletos de código HTML) orientadas al despliegue de las interfaces gráficas con React y Tailwind CSS. Adicionalmente, ante las restricciones de seguridad experimentadas durante el despliegue en los servidores Cloud de AWS (específicamente por bloqueos de red derivados de políticas Cross-Origin o Mixed Content), el asistente proporcionó soporte teórico sobre los fundamentos de la Same-Origin Policy de los navegadores. Sobre la base de esta documentación, el autor procedió a programar y habilitar manualmente las reglas permisivas necesarias, tanto en la configuración del compilador de Vite como en las cabeceras de red del proxy inverso gestionado por Nginx.

Finalmente, estas herramientas de Inteligencia Artificial se emplearon de forma auxiliar en la revisión y mejora de la documentación oficial del proyecto, incluyendo los sucesivos informes de seguimiento. Su uso en esta fase se centró en la corrección ortotipográfica, el refinamiento sintáctico y la adecuación del estilo, garantizando en todo momento un tono académico formal, claro y coherente, sin alterar en ningún caso el contenido técnico, las conclusiones, ni la autoría intelectual del documento.

AGRADECIMIENTOS

Deseo expresar mi más profundo y sincero agradecimiento a mi tutor académico. Su incansable labor de orientación técnica, paciencia infinita y rigor constructivo a lo largo de todas y cada una de las exhaustivas revisiones, validaciones y reuniones de seguimiento de este proyecto han resultado indispensables para poder llevar esta iniciativa tecnológica a buen puerto, y para la transformación de un concepto en un producto maduro de alta ingeniería de software.

De idéntica forma, quiero extender una cálida y especial gratitud a mi entorno cercano y familiar por el incommensurable soporte emocional y logístico sostenido de forma constante durante todos estos meses de trabajo y esfuerzo frente a los inevitables bloqueos mentales y técnicos del ciclo de desarrollo. Finalmente, hago mención especial a mis

compañeros de promoción de la Escuela de Ingeniería, con quienes he reído y disfrutado mucho en todos los momentos fáciles y también difíciles durante la carrera universitaria.

REFERENCIAS

- [1] React. (2026). *The Library for Web and Native User Interfaces. Documentation & API Reference*. Disponible en línea en: <https://react.dev/> (último acceso: 14 de Mayo de 2026).
- [2] Tailwind Labs. (2026). *Tailwind CSS v4 Documentation Reference Guide*. Disponible en línea en: <https://tailwindcss.com/docs/> (último acceso: 12 de Mayo de 2026).
- [3] Vite. (2026). *Vite Next Generation Frontend Tooling Reference*. Disponible en línea en: <https://vite.dev/> (último acceso: 14 de Mayo de 2026).
- [4] Node.js. (2026). *Node.js Technical Runtime Documentation v24*. Disponible en línea en: <https://nodejs.org/en/docs/> (último acceso: 14 de Mayo de 2026).
- [5] OpenAI. (2026). *OpenAI Platform API Reference Guidelines for Text Embeddings*. Disponible en línea en: <https://platform.openai.com/docs/> (último acceso: 10 de Mayo de 2026).
- [6] LangChain. (2026). *LangChain JS Orchestrator Official Framework Documentation*. Disponible en línea en: <https://js.langchain.com/> (último acceso: 10 de Mayo de 2026).
- [7] Pinecone. (2026). *Pinecone Vector Store REST API Integration Reference*. Disponible en línea en: <https://docs.pinecone.io/reference/api/introduction> (último acceso: 10 de Mayo de 2026).
- [8] Nginx. (2026). *Nginx Reverse Proxy Administrator Guide & Load Balancing Documentation*. Disponible en línea en: <https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/> (último acceso: 8 de Junio de 2026).

A continuación se representa, a modo de árbol gráfico, la jerarquía definitiva de archivos, variables de entorno encubiertas y estructuración de directorios de las dos grandes macro-aplicaciones, tal y como se hallan implementadas y aisladas de forma segura dentro del ecosistema del sistema operativo (Ubuntu Linux) del servidor de producción final de Amazon Web Services:

```

CHATBOT-BACKEND/
  src/
    config/
      database.js
    controllers/
      authController.js
      botController.js
      chatController.js
    middleware/
      authMiddleware.js
    models/
      Bots.js
      Users.js
    routes/
      apiRoutes.js
    services/
      aiService.js
      vectorStore.js
  .env
  package.json

CHATBOT-FRONTEND/
  public/
    web-del-cliente.html
    widget.js
  src/
    components/
    pages/
      Dashboard.jsx
      Login.jsx
    App.jsx
    main.jsx
  tailwind.config.js
  vite.config.js

```

APÉNDICE

A.1 Estructura Organizativa Extendida del Repositorio de Código en Producción

Con el firme propósito de asegurar la máxima viabilidad durante el mantenimiento y la escalabilidad a largo plazo, y con la intención expresa de facilitar futuros flujos de despliegue automatizado e integración continua, toda la base de código del proyecto ha sido severamente escindida. Se ha aplicado con rigor académico el principio de diseño de **Separación de Responsabilidades** (**Separation of Concerns**).