🌍 NomadMatch - Sistema RAG de ciudades europeas

https://img.shields.io/badge/version-1.0.0-blue
https://img.shields.io/badge/license-MIT-green
https://img.shields.io/badge/langflow-1.7.1-purple
https://img.shields.io/badge/OpenAI-Embeddings-412991

Un sistema de generación aumentada por recuperación (RAG) listo para su producción, creado con Langflow, ChromaDB y OpenAI. Encuentra y recomienda ciudades europeas para nómadas digitales a través de conversaciones en lenguaje natural.

🏷️ Índice

☐ Descripción

NomadMatch es una solución RAG (Retrieval-Augmented Generation) completa que ayuda a los nómadas digitales a encontrar su ciudad europea ideal. Combina:

- **Langflow** para la orquestación visual de flujos
- **ChromaDB** para el almacenamiento vectorial y la búsqueda por similitud
- **OpenAI Embeddings** para la comprensión semántica
- **FastAPI** para servicios backend robustos

- **Vanilla JavaScript** para un frontend ligero

El sistema procesa datos CSV sobre ciudades europeas, crea incrustaciones y ofrece recomendaciones inteligentes basadas en consultas en lenguaje natural sobre el coste de la vida, la velocidad de Internet, el clima y las comunidades de nómadas digitales.

🏗️ Arquitectura

texto

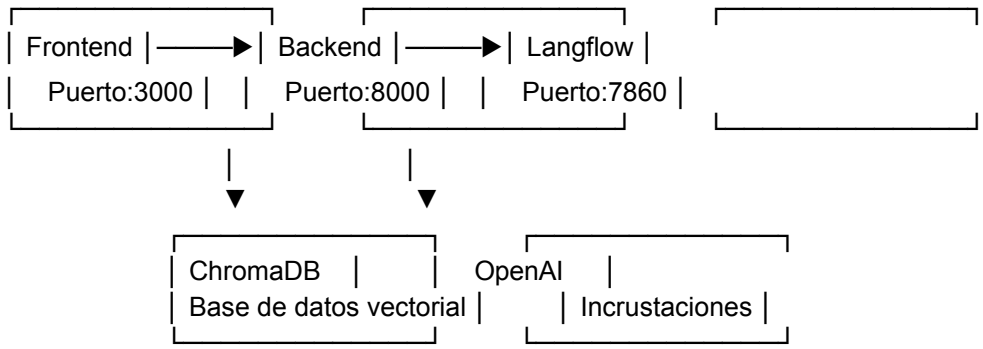
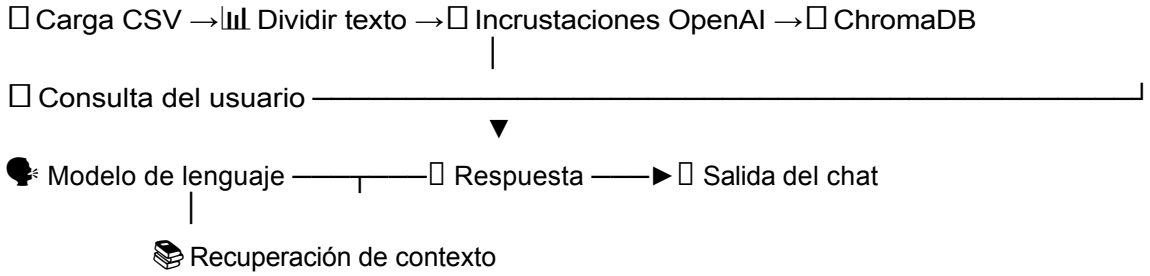


Diagrama de flujo

texto



☐ Características

Características principales

- ☐ **Canalización RAG inteligente**: búsqueda semántica con respuestas sensibles al contexto
- 📊 **Procesamiento de datos CSV**: Fragmentación automática y generación de incrustaciones
- 🔍 **Búsqueda por similitud vectorial**: recuperación rápida y precisa mediante ChromaDB
- ☐ **Chat interactivo**: interfaz de lenguaje natural con gestión de sesiones
- ☐ **Carga mediante arrastrar y soltar**: fácil ingestión de datos a través de la interfaz de usuario

Características técnicas

- ☐ **Docker Compose**: contenedorización completa para una implementación sencilla
- ☐ **API RESTful**: puntos finales completos para todas las operaciones
- 📚 **Documentación generada automáticamente**: documentación OpenAPI/Swagger
- ☐ **Recarga en caliente**: modo de desarrollo con recarga automática
- ☐ **Configuración del entorno**: gestión segura de credenciales

- 📈 **Comprobaciones de estado**: puntos finales de supervisión integrados

Características de los datos

- **Más de 50 ciudades europeas**: conjunto de datos preconfigurado incluido
- **Múltiples atributos**: coste de la vida, velocidad de Internet, clima, etc.
- **Esquema escalable**: fácil de ampliar con nuevos atributos de ciudades
- **Almacenamiento persistente**: ChromaDB con persistencia en disco

📋 Requisitos previos

Requisitos

- **Docker** (20.10+) y **Docker Compose** (2.0+)
- **Clave API de OpenAI** (obtenga una aquí)
- **Git** (2.30+)
- **4 GB+ de RAM** (mínimo)
- **10 GB+ de espacio libre en disco**

Opcional (para instalación manual)

- Python 3.11+
- Node.js 18+
- npm 9+

☐ Inicio rápido

1. Clonar y configurar

bash
git clone https://github.com/yourusername/nomadmatch-rag.git cd
nomadmatch-rag

2. Configurar el entorno

bash
cp backend/.env.example backend/.env
Edita backend/.env y añade tu clave API de OpenAI nano
backend/.env

3. Iniciar con Docker

bash
docker-compose up --build

4. Acceder a las aplicaciones

Servicio	URL	Descripción
Frontend	http://localhost:3000	Interfaz de chat
API del backend	http://localhost:8000	API REST
Documentación de la API	http://localhost:8000/docs	Interfaz de usuario Swagger
Langflow	http://localhost:7860	Editor de flujo

☐ Opciones de instalación

Instalación en Docker (recomendada)

Modo de desarrollo

bash
Compilar e iniciar todos los servicios docker-compose up --build

Ejecutar en segundo plano docker-compose up -d

Ver registros
docker-compose logs -f

Detener servicios
docker-compose down

Detener y eliminar volúmenes docker-compose down -v

Modo de producción

bash
Usar archivo compose de producción
docker-compose -f docker-compose.prod.yml up -d

Escalar servicios
docker-compose -f docker-compose.prod.yml up -d --scale backend=3

Instalación manual

Configuración del backend

bash
cd backend

Crear entorno virtual

```
python -m venv venv
source venv/bin/activate # En Windows: venv\Scripts\activate

# Instalar dependencias
pip install -r requirements.txt

# Establecer variables de entorno
export OPENAI_API_KEY="tu-clave-aquí" export
LANGFLOW_URL="http://localhost:7860"

# Ejecutar el servidor
FastAPI python -m
app.main
```

Configuración del frontend

```
bash
cd frontend

# Instalar dependencias
npm install

# Iniciar el servidor de
desarrollo npm start

# Para la compilación
de producción npm run
build
```

Configuración de Langflow

```
bash
# Instalar Langflow
pip install langflow

# Importar el flujo
langflow --import --file langflow/nomadmatch_langflow.json

# Iniciar el servidor Langflow
langflow --host 0.0.0.0 --port 7860
```

⚙ Configuración

Variables de entorno

Cree un archivo `backend/.env`:

```
env
# Configuración de Langflow
LANGFLOW_URL=http://langflow:7860
LANGFLOW_FLOW_ID=768c1c22-1496-4ccb-8e6f-40a09f44ae3c
```

```
# Configuración de OpenAI
OPENAI_API_KEY=sk-... # Tu clave API de OpenAI
OPENAI_EMBEDDING_MODEL=text-embedding-3-small
OPENAI_CHAT_MODEL=gpt-4o-mini

# Configuración de ChromaDB
CHROMA_PERSIST_DIR=./chroma_data
CHROMA_COLLECTION_NAME=nomadmatch_cities

# Configuración de la API
API_V1_STR=/api/v1
PROJECT_NAME=NomadMatch RAG API
BACKEND_CORS_ORIGINS=["http://localhost:3000", "http://localhost:8000"]

# Entorno
ENVIRONMENT=development
DEBUG=True LOG_LEVEL=INFO
```

Configuración de componentes

Configuración de ChromaDB

- **Colección**: `nomadmatch_cities`
- **Persistencia**: Directorio local con copia de seguridad automática
- **Modelo de incrustación**: `text-embedding-3-small` (1536 dimensiones)
- **Métrica de similitud**: similitud coseno

División de texto

- **Tamaño del fragmento**: 1000 caracteres
- **Superposición de fragmentos**: 0 (garantiza registros limpios de ciudades)
- **Separador**: `\n` (nueva línea)

Modelo de lenguaje

- **Proveedor**: OpenAI
- **Modelo**: `gpt-4o-mini` (optimizado en cuanto a costes)
- **Temperatura**: 0,1 (respuestas coherentes)
- **Transmisión**: Desactivada por defecto

☐ Guía de uso

1. Cargar datos de ciudades

Cargue su archivo CSV con datos de ciudades europeas. El CSV debe incluir columnas como:

- `nombre_ciudad`
- `país`
- `índice_del_costo_de_vida`
- `velocidad_de_internet_mbps`
- `temperatura_media_c`
- `visa_para_nómadas_digitales`
- `espacios_de_trabajo_compartido`
- `nivel_de_inglés`
- `índice_de_seguridad`

Formato CSV de ejemplo:

csv
nombre_de_la_ciudad,país,índice_de_coste_de_vida,velocidad_de_internet_mbps,visa_para_nómadas_digitales,espacios_de_coworking
_espacios Lisboa,Portugal,45,3,180,sí,45
Barcelona,España,52,1,250,sí,78
Budapest,Hungría,38,7,120,sí,32

2. Hacer preguntas

El sistema puede responder a varios tipos de consultas:

Recomendaciones sobre ciudades

- «¿Qué ciudades tienen el mejor índice de coste de vida por debajo de 50?»
- «Muéstrame ciudades costeras con buena velocidad de Internet».
- «Recomiéndame ciudades del sur de Europa con visados para nómadas digitales».

Comparaciones

- «Compara Barcelona y Lisboa para trabajadores remotos».
- «¿Qué ciudad tiene mejor infraestructura: Budapest o Praga?».
- «¿Cuál es la relación entre el coste y la velocidad de Internet en diferentes ciudades?».

Consultas específicas

- «¿Cuál es la velocidad media de Internet en Varsovia?»
- «Muéstrame ciudades con espacios de coworking y clima templado».
- «¿Qué ciudades tienen el coste de vida más bajo?»

3. Interpreta las respuestas

El sistema proporciona:

- **Respuestas directas** a sus preguntas
- **Contexto de la base de datos** que muestra qué ciudades se han tenido en cuenta
- **Puntuaciones de confianza** de la búsqueda por similitud

- **Atribución de la fuente** que muestra qué registros informaron la respuesta

📚 Documentación de la API

URL base

texto
http://localhost:8000/api/v1

Puntos finales

Comprobación de estado

http
GET /health

Respuesta:

json
{
«status»: «healthy»,
«langflow_connected»: true,
«chroma_configured»: true
}

Chat

http
POST /chat
Tipo de contenido: application/json

{
«message»: «¿Qué ciudad tiene la conexión a Internet más
rápida?», «session_id»: «session_1234567890»
}

Respuesta:

json
{
«respuesta»: «Según los datos disponibles, Barcelona tiene la conexión a Internet más rápida, con velocidades de hasta 250 Mbps...»,
«session_id»: «session_1234567890»,
«fuentes»: [
{
«content»: «ciudad: Barcelona | país: España | velocidad_internet_mbps: 250...»,
«metadata»: {
«fuente»:
«ciudades_europeas.csv»,
«índice_de_fila»: 2
},

```
      «puntuación_similaridad»: 0,89
    }
  ]
}
```

Subir documento

http
POST /upload
Tipo de contenido: multipart/form-data

archivo: [CSV_FILE]

Respuesta:

json

```
{
  «mensaje»: «Documento cargado y procesado correctamente»,
  «nombre de archivo»: «ciudades_europeas.csv»,
  "éxito": verdadero,
  "fragmentos_procesados":
  50
}
```

Búsqueda vectorial

http
POST /query
Tipo de contenido: application/json

```
{
  «query»: «ciudades con buena conexión a
  Internet», «num_results»: 5
}
```

Respuesta:

json

```
{
  «resultados»: [...],
  "query": "ciudades con buena conexión
  a Internet", "count": 5
}
```

Colecciones

http
GET /colecciones

Respuesta:

json

```
{
```

```
  «colecciones»: [«nomadmatch_cities»]
}
```

☐ Estructura del proyecto

texto

```
nomadmatch-rag/
├──☐ .github/
│   └── workflows/
│       └── deploy.yml            # Canalización CI/CD
├──☐ backend/
│   ├──☐ app/
│   │   ├──☐ api/
│   │   │   └── routes.py         # Puntos finales de la API
│   │   ├──☐ core/
│   │   │   ├── config.py         # Configuración
│   │   │   └── langflow_client.py
│   │   ├──☐ models/
│   │   │   └── schemas.py        # Modelos Pydantic
│   │   ├──☐ utils/
│   │   │   └── chroma_utils.py # Utilidades de base de datos vectorial
│   │   └── main.py               # Aplicación FastAPI
│   ├── requirements.txt          # Dependencias de Python
│   ├── .env.example              # Plantilla de entorno
│   └── Dockerfile                # Contenedor backend
├──☐ frontend/
│   ├──☐ public/
│   │   ├── index.html            # HTML principal
│   │   └── styles.css            # Estilos CSS
│   ├──☐ src/
│   │   └── app.js                # Lógica frontend
│   ├── package.json              # Dependencias de
│                                   Node
│   └── Dockerfile                # Contenedor frontend
├──☐ langflow/
│   └── nomadmatch_langflow.json # Exportación de Langflow
├──☐ data/
│   └── sample_cities.csv     # Conjunto de datos de muestra
├── docker-compose.yml            # Orquestación de múltiples contenedores
├── .gitignore
├── README.md
└── LICENSE
```

☐ Flujo Langflow

La visualización de Langflow (`nomadmatch_langflow.json`) contiene dos flujos interconectados:

Flujo 1: Ingestión de datos 📚

texto
Archivo (CSV) → Dividir texto → Incrustaciones OpenAI → Chroma DB

- **Objetivo**: Procesar y almacenar datos de ciudades.
- **Desencadenante**: Ejecución manual tras la carga del archivo
- **Salida**: Incrustaciones vectoriales en ChromaDB

Flujo 2: Consulta y respuesta 🐕‍🦺

texto
Entrada de chat → Chroma DB (búsqueda) → Modelo de lenguaje → Salida de chat

- **Objetivo**: Responder a las consultas de los usuarios con RAG
- **Desencadenante**: Mensajes de chat del usuario
- **Salida**: Respuestas contextuales

Componentes clave:

- **Componente de archivo**: analizador CSV con fragmentación
- **Texto dividido**: fragmentos de 1000 caracteres, 0 superposición
- **Incrustaciones de OpenAI**: modelo text-embedding-3-small
- **Chroma DB**: almacenamiento vectorial persistente
- **Modelo de lenguaje**: gpt-4o-mini con temperatura 0,1
- **Entrada/salida de chat**: componentes de interacción con la interfaz de usuario

💻 Desarrollo

Configuración del entorno de desarrollo

1. **Clonar e instalar ganchos pre-commit**

bash
pip install pre-commit
pre-commit install

2. **Desarrollo del backend**

bash
cd backend
pip install -r requisitos-dev.txt pytest
pruebas/ -v

3. **Desarrollo frontend**

bash
cd frontend npm
run dev

Ejecución de pruebas

bash
Pruebas del
backend cd
backend
pytest --cov=app tests/

Pruebas
frontend cd
frontend npm test

Estilo de código

- **Python**: Black, Flake8, mypy
- **JavaScript**: ESLint, Prettier
- **Mensajes de confirmación**: confirmaciones convencionales

☐ Implementación

Opciones de implementación en producción

Opción 1: Docker Swarm

bash
docker swarm init
docker stack deploy -c docker-compose.prod.yml nomadmatch

Opción 2: Kubernetes

bash
kubectl apply -f k8s/

Opción 3: Plataformas en la nube

- **AWS**: ECS con Fargate
- **GCP**: Cloud Run
- **Azure**: Aplicaciones de contenedores
- **Heroku**: Registro de contenedores

Configuración específica del entorno

Configuración de producción

env
ENVIRONMENT=production
DEBUG=false
LOG_LEVEL=WARNING
BACKEND_CORS_ORIGINS=["https://yourdomain.com"]

Consideraciones de seguridad

- ☐ Nunca confirme archivos `.env`
- ☐ Utiliza la gestión de secretos (HashiCorp Vault, AWS Secrets Manager)
- ☐ Habilite HTTPS con Let's Encrypt
- ☐ Implemente la limitación de velocidad
- ☐ Añadir autenticación para producción

🔍 Solución de problemas

Problemas comunes y soluciones

Problemas con la clave API de OpenAI

Problema: `se requiere una clave API de OpenAI para utilizar el proveedor OpenAI`
Solución:

1. Comprueba que tu archivo `.env` tiene la clave API correcta
2. Verifique que la clave esté activa en el panel de control de OpenAI.
3. Asegúrese de que dispone de créditos disponibles

Persistencia de ChromaDB

Problema: El almacén vectorial no persiste entre reinicios.
Solución:

1. Comprueba que la ruta `persist_directory` sea absoluta o relativa al directorio de trabajo
2. Asegúrese de que el volumen de Docker esté correctamente montado
3. Verifique los permisos de escritura en el directorio

Conexión Langflow

Problema: el backend no puede conectarse a Langflow.
Solución:

1. Compruebe que Langflow se está ejecutando: `http://localhost:7860/health`
2. Compruebe la variable de entorno `LANGFLOW_URL`
3. Asegúrese de que el ID del flujo coincida con el flujo importado

Problemas de memoria

Problema: Los contenedores Docker se bloquean con errores de memoria. **Solución**:

yaml
Añadir a docker-compose.yml
servicios:
  backend:
    mem_limit: 1g
    mem_reservation: 512m

Registros y depuración

bash
Ver todos los registros
docker-compose logs -f

Servicio específico
docker-compose logs -f backend

Supervisión en tiempo real
de las estadísticas de Docker

□ Contribuir

¡Agradecemos cualquier contribución! Así es como puedes ayudar:

Proceso de desarrollo

1. **Bifurca el repositorio**
2. **Crea una rama de características**
   bash
   git checkout -b feature/amazing-feature
3. **Confirmar los cambios**
   bash
   git commit -m 'feat: añadir función increíble'
4. **Enviar a la rama**
   bash
   git push origin feature/amazing-feature
5. **Abrir una solicitud de extracción**

Directrices de contribución

- **Estilo del código**: sigue los patrones existentes
- □ **Pruebas**: añadir pruebas para las nuevas funciones
- 📚 **Documentación**: actualizar README y cadenas de documentación
- **Corrección de errores**: incluir pruebas de regresión
- **Cambios en la interfaz de usuario**: garantizar un diseño adaptativo

Prioridades de desarrollo

- Añadir más ciudades europeas al conjunto de datos
- Implementar una capa de almacenamiento en caché para consultas frecuentes
- Añadir autenticación de usuario
- Crear un panel de control con estadísticas de las ciudades
- Admite varios idiomas
- Añadir visualización comparativa de ciudades
- Implementar un bucle de retroalimentación para la calidad de la respuesta

☐ Licencia

Este proyecto está licenciado bajo la licencia MIT. Consulte el archivo LICENSE para obtener más detalles.

texto
Licencia MIT

☐ Equipo

- **Jefe de proyecto**: Su nombre
- **Ingeniero de backend**: Nombre
- **Desarrollador frontend**: nombre
- **Ingeniero de ML**: nombre
- **Diseñador UI/UX**: Nombre

☐ Agradecimientos

- **Equipo de Langflow** por el increíble marco visual
- **OpenAI** por las incrustaciones y los modelos de lenguaje
- **ChromaDB** por la tecnología de base de datos vectorial
- **Comunidad europea de nómadas digitales** por la inspiración y los comentarios
- **Todos los colaboradores** que ayudan a mejorar este proyecto

☐ Contacto y asistencia

- **Problemas de GitHub**: informar de errores
- **Debates**: Únete a las conversaciones
- **Correo electrónico**: your.email@example.com
- **Twitter**: @tu_nombre_de_usuario

📊 Estado del proyecto

Componente	Estado	Versión
API backend	☐ Estable	v1.0.0
Interfaz de usuario frontend	☐ Estable	v1.0.0

Integración con Langflow	☐ Estable	v1.7.1
ChromaDB	☐ Estable	v1.4.0
Integración con OpenAI	☐ Estable	v0.3.35
Configuración de Docker	☐ Estable	Compose v3.8

☐ Hoja de ruta

Primer trimestre de 2026

- ☐ Implementación básica de RAG
- ☐ Carga y procesamiento de CSV
- ☐ Interfaz de chat
- ☐ Contenedorización Docker

Segundo trimestre de 2026

- ☐ Función de comparación entre varias ciudades
- ☐ Resaltado de fuentes de respuesta
- ☐ Optimización del rendimiento
- ☐ Recopilación de comentarios de los usuarios

Tercer trimestre de 2026

- ☐ Sistema de autenticación
- ☐ Recomendaciones personalizadas
- ☐ Limitación de la tasa de API
- ☐ Opciones de filtrado avanzadas

Cuarto trimestre de 2026

- ☐ Aplicación móvil
- ☐ Actualizaciones de datos de la ciudad en tiempo real
- ☐ Sistema de reseñas de la comunidad
- ☐ Expansión global de ciudades

☐ Historia de estrellas

https://api.star-history.com/svg?repos=yourusername/nomadmatch-rag&type=Date

---

Creado con ❤ para los nómadas digitales de todo el mundo🌍 ✈

Última actualización: 11 de febrero de 2026
Versión: 1.0.0