Desafío Técnico – Desarrollador Full-Stack (Nivel Mid)

> **Tiempo máximo sugerido:** 3 horas
> **Idioma de entrega:** Español
> **Entrega:** URL de repositorio público + instrucciones para `docker compose up`

Contexto

Una **startup ficticia** llamada ***Q-Finder*** necesita un prototipo que permita:

1. **Registrar** preguntas frecuentes (FAQs) con su respuesta.
2. **Buscar** la FAQ más similar a una consulta libre del usuario usando un algoritmo de similitud en espacios vectoriales.
3. Mostrar la FAQ encontrada en una página pública con *server-side rendering* (SSR) para buen SEO.

El equipo usa **Docker**, **FastAPI** y **Next.js**; más adelante migrará a AWS Lambda/Amplify, pero por ahora basta con contenedores locales.

Tu misión

Área	Tarea esencial (obligatoria)	Puntos a evaluar
Backend (FastAPI)	• Crea dos endpoints:	
`POST /api/faq` → recibe `{question, answer}` y guarda en memoria o SQLite.		
`POST /api/search` → recibe `{query}` y devuelve la FAQ más similar.		

• Implementa el cálculo de **similitud coseno** sobre vectores de tamaño 128.
• Para "vectorizar" un texto *NO* llames a un modelo externo; genera un vector **determinista** (p. ej. *hashing trick*, TF-IDF reducida o cualquier método ligero). | Diseño limpio de API, claridad del algoritmo, test unitario corto. | | **Frontend (Next.js 14)** | • Página principal (`/`) con un **input controlado** y lista de resultados.
• Usa **React Hooks** propios o de la librería (p. ej. `useSWR`) para:
– Gestionar el estado de búsqueda.
– Mostrar un *loader* con **renderizado condicional**.
• **SSR** para la ruta `/faq/[id]` que muestra la pregunta y respuesta elegida. | Correcta separación de componentes, manejo de estado, SSR/SSG. | | **Docker** | • `Dockerfile` para el backend y para la app Next.js.
• `docker-compose.yml` que levante ambos servicios en puertos diferentes. | Imagen mínima, tiempos de *build* razonables. | | **Agilidad mental** | • Explica en un **README ≤ 150 palabras** por qué tu técnica de vectorización es "suficientemente buena" para un prototipo.
• Añade **2-3 comentarios** en el código donde hayas tomado una decisión de diseño no trivial. | Claridad y criterio. |

> **Opcional (bonus, *NO* obligatorio):**
> • Simulacro de despliegue en AWS con `amplify.yml` o `serverless.yml` .
> • Pequeño agente AI que sugiera correcciones al buscar y no hallar resultados (< 20 líneas).

Reglas y restricciones

1. No incluyas dependencias pesadas (≥ 100 MB).
2. El API **no** debe depender de ningún servicio externo (OpenAI, AWS, etc.).
3. El prototipo debe arrancar con:

```
git clone <tu-repo>
cd <repo>
docker compose up --build
```

5. Aporta, si lo deseas, un pequeño script de *seed* para crear datos de ejemplo.

Entrega

- Repo público (GitHub).
- **README** conciso con: pasos de arranque, tiempo real invertido, descripción corta de tu vectorización.

Criterios de evaluación

Peso	Criterio
35 %	Correctitud funcional (endpoints, SSR, búsqueda coherente).
25 %	Calidad de código (legibilidad, estructura, comentarios decisivos).
15 %	Buen uso de React Hooks y renderizado condicional.
15 %	Dockerización simple y efectiva.
10 %	Claridad del README y justificación técnica.

¡Éxito con el reto y que el código hable!