Prueba Técnica Full-Stack: Mini-Aplicación de Gestión de Personal 🎯

Introducción a la Prueba

Bienvenido/a a nuestra prueba técnica para la posición de Programador/a Full-Stack. Esta prueba está diseñada para evaluar tus habilidades en el diseño y desarrollo de una aplicación pequeña pero completa, cubriendo frontend, backend y base de datos, partiendo desde cero. Buscamos programadores competentes que puedan construir aplicaciones robustas, bien estructuradas y funcionales utilizando un stack tecnológico específico.

Escenario del Proyecto Scenario 🗒️

Imagina que estás comenzando el desarrollo de una nueva sección para una plataforma SaaS de Recursos Humanos. Esta sección permitirá gestionar empleados y visualizar algunas estadísticas básicas sobre ellos.

- **Tecnologías Obligatorias:**
  - **Frontend: Angular 19** (o la última versión LTS estable), TypeScript, HTML, SCSS. Para las gráficas, utiliza **ApexCharts** (integrada mediante `ng-apexcharts`).
  - **Backend: NestJS**, implementando una API REST.
  - **Base de Datos: MongoDB** (se recomienda usar Mongoose como ODM).
  - **Autenticación: JWT** (JSON Web Tokens) para la gestión de sesiones.

Tiempo Estimado: 1-2 días ⏳

Tareas a Desarrollar 🛠️

Deberás construir una pequeña aplicación que incluya las siguientes funcionalidades:

Tarea 1: Autenticación de Usuarios y Estructura Base

- **Descripción:** Implementar un sistema de autenticación básico y la estructura inicial de la aplicación.
- **Requisitos Backend (NestJS):**
  1. **Módulo de Autenticación (`AuthModule`):**

        - Define un modelo/schema simple para `User` en MongoDB (ej. `email`, `password` -asegúrate de almacenar las contraseñas de forma segura usando hashing y salting con `@nestjs/passport` y `bcrypt`-, `name`). Utiliza `@nestjs/mongoose`.
        - Implementa un `AuthService` con lógica para registro y login.
        - Crea un `AuthController` con los siguientes endpoints:
            - `POST /api/v1/auth/register` para crear nuevos usuarios.
            - `POST /api/v1/auth/login` que valide las credenciales y, si son correctas, devuelva un JWT (utiliza `@nestjs/jwt`).
    2. **Protección de Rutas:**
        - Implementa un `JwtAuthGuard` utilizando Passport y `@nestjs/passport` para proteger rutas que requieran autenticación.
- **Requisitos Frontend (Angular 19):**
    1. **Módulo de Autenticación (`AuthModule`):**
        - Componente `LoginComponent`: Un formulario para que los usuarios ingresen `email` y `password`.
        - Componente `RegisterComponent`: Un formulario para que los nuevos usuarios se registren.
    2. **Servicio de Autenticación (`AuthService`):** Un servicio en Angular para manejar el registro, login, logout y almacenamiento/gestión del JWT (ej. en `localStorage`).
    3. **Guardias de Rutas (`AuthGuard`):** Protege las rutas de la aplicación que requieran que el usuario esté autenticado.
    4. **Layout Básico:**
        - Un `LayoutComponent` principal para cuando el usuario esté autenticado (ej. con una barra de navegación simple y un `<router-outlet>`).
        - Un layout para las páginas de autenticación.

Tarea 2: CRUD de Gestión de Personas (Empleados)

- **Descripción:** Implementar la funcionalidad para Crear, Leer, Actualizar y Eliminar (CRUD) registros de personas/empleados. Esta sección debe ser accesible solo para usuarios autenticados.
- **Requisitos Backend (NestJS):**
    1. **Módulo de Personas (`PersonsModule`):**

        - Define un modelo/schema para `Person` en MongoDB (ej. `firstName`, `lastName`, `email`, `position`, `department`, `hireDate`, `salary` - numérico).
        - Implementa un `PersonsService` con la lógica del CRUD.
        - Crea un `PersonsController` protegido con `JwtAuthGuard` y los siguientes endpoints REST:
            - `POST /api/v1/persons` (Crear una nueva persona)
            - `GET /api/v1/persons` (Listar todas las personas, con paginación básica - ej. `?page=1&limit=10`)
            - `GET /api/v1/persons/{id}` (Obtener detalles de una persona)
            - `PUT /api/v1/persons/{id}` (Actualizar una persona)
            - `DELETE /api/v1/persons/{id}` (Eliminar una persona)
    2. **DTOs y Validaciones:** Utiliza DTOs (Data Transfer Objects) con `class-validator` y `class-transformer` para los datos de entrada y salida en el backend (ej. `CreatePersonDto`, `UpdatePersonDto`).
- **Requisitos Frontend (Angular 19):**
    1. **Módulo de Personas (`PersonsModule`):**
        - Componente `PersonListComponent`:
            - Mostrar las personas en una tabla o lista.
            - Incluir opciones para editar y eliminar cada persona.
            - Un botón para "Añadir Nueva Persona" que lleve a un formulario o abra un diálogo modal.
            - Implementar paginación.
        - Componente `PersonFormComponent` (para creación/edición):
            - Un formulario reactivo para ingresar/modificar los datos de una persona.
            - Validaciones en el frontend.
    2. **Servicio de Personas (`PersonsService`):** Un servicio en Angular para interactuar con los endpoints REST del CRUD de personas.

Tarea 3: Dashboard de Visualización de Datos de Personas

- **Descripción:** Crear una página de "Dashboard" (accesible solo para usuarios autenticados) que muestre al menos dos gráficas utilizando **ApexCharts**, basadas en los datos de las personas gestionadas.
- **Requisitos Backend (NestJS - Opcional, pero recomendado):**
    1. Considera si necesitas endpoints específicos en tu `PersonsController` para agregar datos para las gráficas (ej. `GET /api/v1/persons/stats/by-department`). Esto es preferible a

procesar grandes volúmenes de datos en el frontend. Deberás usar el **framework de agregación de MongoDB** si implementas esto.

- **Requisitos Frontend (Angular 19 con `ng-apexcharts`):**
    1. **Módulo de Dashboard (`DashboardModule`):**
        - Componente `DashboardComponent`: Una nueva página/ruta protegida.
    2. **Integración de ApexCharts:** Asegúrate de importar `NgApexchartsModule`.
    3. **Gráfica 1 (Ej. Distribución por Departamento):**
        - Tipo de gráfica: Gráfica de Barras o Torta (Pie Chart).
        - Datos: Mostrar el número de personas por `department`.
        - Utiliza **ApexCharts** para renderizar la gráfica.
    4. **Gráfica 2 (Ej. Distribución de Salarios o Contrataciones por Mes):**
        - Tipo de gráfica: Gráfica de Línea o Histograma.
        - Datos: Mostrar la distribución de `salary` (ej. en rangos) o el número de contrataciones a lo largo del tiempo (ej. por mes del `hireDate`).
        - Utiliza **ApexCharts** para renderizar la gráfica.
    5. **Servicios de Datos:** El dashboard deberá obtener los datos necesarios (ya sea la lista completa de personas o datos agregados desde el backend) para construir las gráficas.

---

3. Entregables Esperados deliverables 📦

El candidato deberá entregar:

1. **Un enlace a un repositorio Git (preferiblemente GitHub, GitLab, o Bitbucket) que contenga:**
    - La aplicación frontend completa (código fuente Angular 19).
    - La aplicación backend completa (código fuente NestJS).
    - Instrucciones claras en el `README.md` sobre cómo configurar (variables de entorno, conexión a MongoDB), instalar dependencias (`npm install` o `yarn install`) y ejecutar ambas aplicaciones (frontend y backend).
    - Incluye un archivo `.env.example` para el backend con las variables de entorno necesarias (ej. `MONGO_URI`, `JWT_SECRET`, `JWT_EXPIRES_IN`).
2. **(Opcional) Un breve video (máximo 5 minutos) demostrando la aplicación funcionando.** 📹

---

4. Criterios de Evaluación Clave ⚖️

Tu solución será evaluada considerando los siguientes aspectos:

- **Funcionalidad (40%):**
  - ¿Todas las tareas se completaron y funcionan según lo esperado? (Login, Registro, CRUD completo, Gráficas funcionales).
  - ¿La autenticación con JWT y protección de rutas funcionan correctamente?
  - ¿Correcta implementación de la paginación?
- **Calidad del Código Backend (NestJS) (20%):**
  - Claridad, organización y legibilidad del código (estructura de módulos, servicios, controladores).
  - Uso de buenas prácticas de NestJS (DTOs, validación con `class-validator`, pipes, guards, inyección de dependencias).
  - Correcto diseño de la API REST (verbos HTTP, códigos de estado, estructura de las respuestas).
  - Seguridad (hashing de contraseñas, implementación de JWT).
  - Interacción con MongoDB (uso de Mongoose, schemas, agregaciones si se implementan).
- **Calidad del Código Frontend (Angular 19) (20%):**
  - Claridad, organización y legibilidad del código Angular.
  - Correcta componentización, uso de servicios, módulos y routing.
  - Gestión del estado y flujo de datos (se valorará el uso de RxJS de forma efectiva).
  - Manejo de formularios reactivos y validaciones.
  - Implementación y configuración de las gráficas con ApexCharts.
  - Diseño responsive básico.
- **Diseño de la Solución y Base de Datos (10%):**
  - Diseño de los schemas de MongoDB.
  - Lógica general de la aplicación y cómo interactúan el frontend y el backend.
- **Calidad del Repositorio y Documentación (10%):**
  - Claridad de los commits (si se usa Git de forma incremental).
  - Calidad de las instrucciones en el `README.md`.
  - Facilidad para poner en marcha el proyecto.

---

5. Consideraciones Adicionales 💡

- **Interfaz de Usuario (UI):** No se espera un diseño UI/UX profesional, pero la interfaz debe ser clara, usable y funcional. Puedes usar Angular Material si lo deseas para acelerar el desarrollo de la UI, pero no es obligatorio.

- **Testing:** Escribir pruebas unitarias (ej. con Jest para NestJS y Karma/Jasmine para Angular) es un **plus muy valorado**. Si las incluyes, enfócate en las partes críticas (ej. lógica de negocio en servicios del backend, servicios de autenticación, lógica compleja en componentes).

Preguntas de Discusión (para la entrevista técnica posterior): 🗣️

1. En tu implementación de NestJS, ¿qué ventajas encontraste al usar DTOs y el sistema de validación integrado?
2. ¿Cómo manejaste el estado de la autenticación en el frontend Angular? ¿Qué alternativas consideraste?
3. Para las gráficas del dashboard con ApexCharts, si los datos de "personas" fueran millones de registros, ¿cómo optimizarías la carga y presentación de las gráficas tanto en backend como en frontend?
4. Si tuvieras que implementar un sistema de roles y permisos más complejo (ej. admin, manager, empleado) sobre tu base actual, ¿cómo lo abordarías utilizando NestJS y JWT?
5. ¿Qué estrategias de manejo de errores implementaste en el backend y cómo se reflejan en el frontend?

¡Mucha suerte con la prueba! Esperamos ver tu solución. 👍