Prueba Técnica: Desarrollo de Backend con Node.js (TypeScript) / NestJS

Descripción General

El objetivo de esta prueba técnica es evaluar las habilidades de un desarrollador en la creación de un backend utilizando **Node.js con TypeScript** o **NestJS**, implementando buenas prácticas y documentando la API con **Swagger**. El backend servirá para gestionar una lista de **Servicios**, asignarlos a **Usuarios**, y permitirá que un usuario consulte y administre sus servicios. La prueba debe ser completada en **24 horas**.

Objetivos

1. **Creación de una API** que permita gestionar usuarios y servicios.
2. **Implementar autenticación** para proteger los endpoints, generando tokens para el acceso autorizado.
3. **Asociar servicios a usuarios** (similar a "mis servicios").
4. Implementar **documentación Swagger** para los endpoints.
5. Utilizar **seeders** para poblar la base de datos con datos iniciales de usuarios y servicios, asegurando que los datos sean coherentes y estén bien estructurados.
6. Incluir una relación entre usuarios y servicios, donde un usuario puede tener varios servicios asignados.
7. Buenas prácticas de desarrollo con **TypeScript**, uso de **DTOs**, validaciones, y control de acceso.

Requisitos Previos

- El backend debe estar desarrollado en **Node.js con TypeScript** o **NestJS**.
- Utilizar una base de datos como **PostgreSQL** o **MySQL**.
- El proyecto debe estar documentado con **Swagger** para facilitar la interacción con los endpoints.
- El código debe ser subido a un repositorio público en **GitHub**, con un archivo **README.md** que explique cómo ejecutar el proyecto, correr las migraciones y ejecutar los seeders.

Requisitos Técnicos

1. **Entidades y Relación**:
    - **Usuario**:
        - Campos: `id`, `nombre`, `email`, `password`, `rol` (admin, user), `createdAt`, `updatedAt`, `deletedAt`.
        - Un usuario puede tener varios servicios asignados.

    - **Servicio**:
        - Campos: `id`, `nombre`, `descripcion`, `costo`, `categoria`, `createdAt`, `updatedAt`, `deletedAt`.
        - Servicios se asocian a usuarios (relación de muchos a muchos).
2. **Seguridad**:
    - Implementar medidas de seguridad adecuadas para proteger los endpoints que requieren autenticación.
    - Asegurar que las contraseñas sean manejadas de manera segura durante el registro y almacenamiento de usuarios.
3. **Seeders**:
    - Debes crear **seeders** para llenar la base de datos con datos iniciales:
        - Un conjunto de usuarios (ej. 3 usuarios) con contraseñas seguras.
        - Un conjunto de servicios (ej. 5 servicios de categorías como Tecnología, Salud, Hogar).
    - Incluir en el **README.md** los comandos necesarios para correr los seeders.
4. **Documentación con Swagger**:
    - Debes incluir **Swagger** para documentar los endpoints de la API. Swagger debe estar disponible en la ruta `/api`.
5. **Validaciones**:
    - Utilizar **DTOs** y **class-validator** para validar los datos entrantes, asegurando que los campos necesarios estén presentes y tengan el formato adecuado (ej. `email` debe ser un correo válido, `costo` debe ser un número, etc.).

Detalles Técnicos

1. **Tecnologías Obligatorias**:
    - **Node.js con TypeScript** o **NestJS**.
    - **PostgreSQL** o **MySQL** como base de datos.
    - **Documentación Swagger** para la API.
    - **TypeORM** para el manejo de la base de datos y migraciones.
2. **Relaciones**:
    - Un **usuario** puede tener varios **servicios** asociados.
    - Un **servicio** puede estar asignado a varios **usuarios**.
3. **Campos de Servicio**:
    - **nombre**: Nombre del servicio (string).
    - **descripcion**: Descripción del servicio (string).
    - **costo**: Costo del servicio (número).
    - **categoria**: Categoría del servicio (Tecnología, Salud, etc.).
    - **timestamps**: Fechas de creación, actualización y eliminación.
4. **Estructura de Proyecto**:
    - Estructura organizada siguiendo buenas prácticas de **NestJS** o **Node.js** con carpetas para:
        - **Controllers**: Para los controladores de cada entidad.
        - **Services**: Para la lógica de negocio.
        - **Modules**: Para la organización modular del proyecto.

    - **Entities**: Definición de entidades y relaciones de base de datos.
    - **DTOs**: Para manejo de los datos entrantes y salientes.

Entregables

1. **Repositorio Público en GitHub**:
    - El código debe ser subido a un repositorio público en GitHub.
    - El archivo **README.md** debe incluir:
        - Pasos para clonar y ejecutar el proyecto.
        - Comandos para ejecutar las migraciones de la base de datos.
        - Comandos para ejecutar los seeders.
        - Instrucciones para acceder a la documentación de Swagger.
2. **Código de Calidad**:
    - Se espera que el código siga buenas prácticas de desarrollo.
    - Uso adecuado de **TypeScript** y modularización en NestJS o Node.js.
    - Validaciones correctas de datos.
    - Manejo adecuado de eliminación lógica (uso de `deletedAt`).
3. **Tiempo de Entrega**:
    - La prueba debe ser completada en un máximo de **24 horas** desde la recepción de este documento.

Criterios de Evaluación

1. **Cumplimiento de los Requisitos**:
    - Implementación de todos los requisitos solicitados.
    - Uso adecuado de autenticación y manejo seguro de contraseñas.
    - Uso adecuado de seeders para llenar la base de datos con datos iniciales.
    - Documentación completa con Swagger.
2. **Calidad del Código**:
    - Buenas prácticas de desarrollo.
    - Uso adecuado de **TypeScript** y modularización en NestJS o Node.js.
    - Validaciones correctas de datos.
3. **Documentación y README**:
    - El **README.md** debe ser claro y permitir a cualquier desarrollador levantar el proyecto fácilmente.
    - Swagger debe estar correctamente configurado y accesible.