(v0.0.1 - 25-07-2024)
Team Leader: Ramiro Grisales

Prueba tecnica – Library API

Ruta: NestJS Advance Route

Caso: Clean Code en NestJs

1. Objetivo

La prueba técnica tiene como objetivo evaluar las habilidades técnicas, creatividad y capacidad de resolución de problemas, y competencias generales del Coders.

2. Condiciones Generales del Assessment

Se permitirá el uso de documentación oficial, ejemplos de código abierto y herramientas de IA como apoyo, siguiendo estrictamente los siguientes lineamientos:

A. **Buen Uso de la IA**

- **Asistencia en Código:** La IA se podrá utilizar para obtener sugerencias y correcciones en el código, pero no para resolver tareas completas de manera autónoma.
- **Aprendizaje y Recursos:** Los Coders podrán utilizar herramientas de IA para acceder a materiales de aprendizaje y documentación relevante.
- **Optimización y Mejora:** Se permite el uso de la IA para optimizar y mejorar el código ya desarrollado por el Coder, siempre y cuando se mantenga la integridad del proceso de aprendizaje.

B. Alcance de la IA

- **Código y Desarrollo:** La IA puede asistir en la escritura, depuración y optimización del código.
- **Investigación:** Los coders podrán utilizar la IA para buscar información y resolver dudas técnicas, puntuales y acotadas.
- **Documentación:** La IA puede ayudar a generar documentación y comentarios en el código.

C. Límites de la IA

- **Prohibido el Plagio:** No se permite la generación de soluciones completas o la copia de código de fuentes externas sin la debida atribución.
- **Autonomía del Coder:** La IA no debe sustituir la capacidad del coder para resolver problemas; debe actuar como una herramienta de apoyo.
- **Evaluación de Habilidades:** La IA no puede ser utilizada para responder a preguntas de evaluación de manera directa, ya que el objetivo es evaluar las competencias del coder.

D. Buen Uso de Internet

- **Investigación y Recursos:** Se permite el uso de internet para buscar información, acceder a documentación y utilizar recursos educativos.
- **Comunicación:** Los coders pueden utilizar internet para comunicarse con el equipo de evaluación o pedir asistencia técnica si es necesario.
- **Desarrollo Colaborativo:** El uso de plataformas de desarrollo colaborativo en línea (como GitHub) está permitido para la gestión de proyectos.

E. Alcance de Internet

- **Acceso a Documentación:** Los coders podrán consultar documentación oficial y recursos educativos en línea.
- **Resolución de Problemas:** Se permite usar foros y comunidades técnicas para resolver dudas específicas y acotadas cuando se referencie.
- **Actualizaciones y Herramientas:** Los coders pueden descargar e instalar herramientas y actualizaciones necesarias para el desarrollo del assessment.

F. Límites de Internet

- **Fuentes Confiables:** Se debe verificar la fiabilidad de las fuentes utilizadas. No se permite el uso de contenido no autorizado o pirata.

- **Prohibido el Fraude:** No se permite buscar o utilizar soluciones completas o respuestas directas a los problemas planteados en el assessment.
- **Seguridad y Privacidad:** Se debe garantizar la seguridad y privacidad de la información personal y de los datos del assessment. No se permite compartir información confidencial.

G. Monitoreo y Cumplimiento

- **Supervisión:** El uso de IA e internet será monitoreado por el TL para asegurar el cumplimiento de los lineamientos.
- **Consecuencias:** Cualquier violación a estos lineamientos resultará en una evaluación adicional y posibles sanciones, incluyendo la descalificación del assessment.

H. Ética y Responsabilidad

- **Uso Ético:** Se espera que todos los coders usen la IA e internet de manera ética y responsable.
- **Responsabilidad Individual:** Cada coder es responsable de su propio trabajo y del cumplimiento de estos lineamientos.

3. Metodología

La prueba técnica se estructurará bajo historias de usuario Scrum con una duración estimada de 1 día por historia. Esta metodología ágil permite simular un entorno de trabajo realista, donde los desarrolladores reciben tareas con plazos definidos y deben gestionar su tiempo y recursos de manera eficiente.

I. Las reglas

a. Tendrás toda la semana para desarrollar esta prueba
b. Debes seguir los lineamientos del Assessment durante la prueba.
c. Debes tener la capacidad de explicar tu código.
d. Se deben cumplir TODOS los criterios de aceptación de la HU.
e. Toda la documentación en inglés.

II. Las metodologías

a. Trabajarás en una historia de usuario similar a un entorno de trabajo real.

b. Demuestra tus conocimientos técnicos y tu capacidad para resolver problemas.
c. ¡Buena suerte y manos a la obra!

4. Historia de usuario

Desafío de Implementación de Clean Code en NestJS con Historias de Usuario

Contexto: Una empresa de software ha solicitado el desarrollo de una API para gestionar la biblioteca de libros de una cadena de librerías. La empresa quiere asegurarse de que el código sea fácil de mantener y escalar, por lo que es esencial aplicar los principios de **Clean Code** y buenas prácticas de desarrollo.

I. Historias de Usuario

- **Historia 1: Crear un libro en la API**

  Como administrador de la biblioteca, **quiero** poder agregar nuevos libros al sistema, **para** gestionar fácilmente las publicaciones que tenemos en el inventario.

  - **Criterios de Aceptación de la historia de usuario:**

    a. La API debe permitir que un libro se cree con los siguientes campos: título, autor, fecha de publicación y género.
    b. Si alguno de los campos obligatorios está vacío, la API debe devolver un error claro y manejado.
    c. Los datos deben almacenarse en una base de datos (PostgreSQL o MongoDB).
    d. Se debe seguir el principio de **Responsabilidad Única**: la lógica de la creación debe estar en el servicio, no en el controlador.

- **Retos Adicionales:**

  a. Cobertura Completa de test

- **Historia 2: Consultar la lista de libros**

  Como visitante de la biblioteca, **quiero** poder ver una lista de los libros disponibles, **para** seleccionar el libro que me interesa.

  - **Criterios de Aceptación de la historia de usuario:**

    a. La API debe devolver una lista de todos los libros disponibles en el sistema.
    b. La lista debe incluir: título, autor, fecha de publicación y género.
    c. Si no hay libros en el sistema, la API debe devolver un mensaje adecuado.
    d. Implementar el principio de **Funciones Pequeñas**: el método del servicio para obtener la lista de libros no debe realizar otras operaciones.

  - **Retos Adicionales:**

    a. La API debe poder retornar los libros filtrados por fecha, autor y genero
    b. Implementar un paginado
    c. Cobertura Completa de test

- **Historia 3: Obtener los detalles de un libro específico**

  Como administrador o visitante de la biblioteca, **quiero** poder ver la información detallada de un libro específico, **para** obtener más detalles sobre él antes de seleccionarlo.

  - **Criterios de Aceptación de la historia de usuario:**

    a. La API debe devolver la información completa de un libro a partir de su id.
    b. Si el libro no se encuentra, la API debe devolver un mensaje de error apropiado.
    c. Se debe aplicar el principio de **Manejo Adecuado de Errores** utilizando los filtros de excepciones de NestJS.

  - **Retos Adicionales:**
    a. Cobertura Completa de test

- **Historia 4: Actualizar la información de un libro**

  Como administrador de la biblioteca, **quiero** poder actualizar la información de un libro existente, **para** mantener los datos actualizados si se cometió un error o cambió alguna información.

  - **Criterios de Aceptación de la historia de usuario:**

a. La API debe permitir la actualización de los campos: título, autor, fecha de publicación y género de un libro.
b. Si el libro no se encuentra, la API debe devolver un mensaje de error.
c. Se deben aplicar **Interfaces Claras**: Utilizar DTOs (Data Transfer Objects) para la actualización de los datos.

Retos Adicionales:

Cobertura Completa de test

- **Historia 5: Eliminar un libro del sistema**

  Como administrador de la biblioteca, **quiero** poder eliminar un libro del sistema, **para** retirar libros obsoletos o dañados del inventario.

  - **Criterios de Aceptación de la historia de usuario:**
    a. La API debe permitir la eliminación de un libro a partir de su id.
    b. Si el libro no se encuentra, debe devolver un error claro.
    c. El borrado debe ser estrictamente lógico
    d. Aplicar el principio de **Modularización**: Cada operación debe estar en un módulo y servicio separados, permitiendo la escalabilidad de la aplicación.

  - **Retos adicionales**
    a. Cobertura total de test

La API debe permitir la eliminación de varios libros a partir de su id

- **Historia 6: Validaciones de Entrada**

  Como desarrollador,

  quiero que la API valide correctamente los datos de entrada,

  para evitar errores al procesar datos incorrectos.

  - **Criterios de Aceptación de la historia de usuario:**

    a. Validar que los campos obligatorios sean enviados y tengan el formato adecuado.
    b. Utilizar decoradores de validación como @IsString, @IsNotEmpty, y @IsDate para validar la entrada de datos.
    c. Si la validación falla, la API debe devolver una respuesta con los errores de validación.
    d. **Testing**: Las pruebas unitarias deben cubrir los controladores y servicios, verificando tanto los casos de éxito como los de error.

- **Historia 7: Pruebas Unitarias**

  Como desarrollador,

  quiero que la API esté completamente cubierta por pruebas unitarias,

  para asegurar que las funcionalidades se comporten correctamente y evitar futuros bugs.

- **Criterios de Aceptación de la historia de usuario:**
  a. Crear pruebas unitarias para los servicios y controladores usando **Jest**.
  b. Las pruebas deben verificar tanto los casos exitosos como los casos de error (como libro no encontrado).
  c. Aplicar el principio de **Pruebas con Valor**: Las pruebas deben centrarse en la funcionalidad clave y en las rutas críticas.

II. Requisitos Técnicos:

a. Utiliza **NestJS**.
b. Implementa **CRUD** (Create, Read, Update, Delete).
c. Almacena los datos en **PostgreSQL** o **MongoDB**.
d. Usa **TypeScript** de manera estricta (strict mode activado).
e. Cubre la API con pruebas unitarias utilizando **Jest**.
f. Implementa y configura un linter

III. Aplicar Clean Code:

a. **Nombres Descriptivos**: Usa nombres que indiquen claramente su propósito para clases, métodos y variables.
b. **Responsabilidad Única**: Los métodos deben tener una única responsabilidad. Evita los métodos que hacen múltiples cosas.
c. **Funciones Pequeñas**: Mantén las funciones y métodos cortos. Idealmente, no más de 20 líneas por función.
d. **Comentarios Útiles**: Los comentarios deben agregar valor (explicar *por qué*, no *qué*), y se debe evitar el uso de comentarios obvios.
e. **Tratamiento de Errores**: Implementa un buen manejo de excepciones utilizando los filtros de NestJS para centralizar el manejo de errores.

f. **Inyección de Dependencias**: Implementa el patrón de inyección de dependencias con los controladores y servicios de NestJS.
g. **Modularización**: Cada módulo debe tener su propio conjunto de controladores, servicios, entidades y DTOs.

- Implementar rutas según RESTful API para las operaciones CRUD de procesamiento de los archivos.

5. Criterios de aceptación generales

- **REST**: Asegurar que las rutas sean intuitivas y sigan las convenciones REST.
- **Controladores:** Cada controlador debe tener un propósito claro y bien definido.
- **Servicios**: Se deben manejar las operaciones de negocio dentro de los servicios y no en los controladores.
- **DTOs (Data Transfer Objects):** Asegúrate de que los DTOs se usen para validar y transformar datos de entrada.
- **Validación**: Aplica la validación de datos utilizando decoradores como @IsString, @IsNotEmpty, etc.
- **Manejo de Excepciones:** Utiliza filtros de excepciones de NestJS para capturar y manejar errores de manera centralizada.
- **Testing**: Las pruebas unitarias deben cubrir los controladores y servicios, verificando tanto los casos de éxito como los de error.
- **Documentación:**
    - Incluir un archivo README que explique cómo configurar y ejecutar el proyecto, en inglés.
    - Utilizar SWAGGER para documentar la API generada.
    - Enlace a PDF en el README con evidencia de diagramas UML (diagrama de arquitectura, diagrama ER, diagrama casos de uso) utilizados en la planeación, capturas del tablero de tareas al inicio y al final donde se evidencia las responsabilidades de cada CODER.
    - Establecer en el READMEel patrón de arquitectura utilizado y su justificación para este caso de uso.

- **Gitflow**:

- Utilizar Gitflow para la gestión de ramas y versiones del proyecto.
- Asegurarse de que los commits sean descriptivos y las ramas sigan la convención de Gitflow.

- **Códigos de Respuesta:** Devolver los códigos de respuesta adecuados para cada operación (**201**, **200**, **400**, **401**, **404**, **500**, etc.).
- **Validaciones APIKEY:** Implementar un validador de x-api-key en los headers para validar que la petición sea confiable.
- **Deploy**: Desplegar en alguna nube.

6. Entregables

a. Código fuente del proyecto en un repositorio GitHub. Adjuntar URL del repositorio y el código en ZIP en Moodle. (solo se revisan ramas con commits que máximo tengan la fecha y hora de finalización de la prueba). Enlace de Moodle para la entrega: aquí
b. URL de la plataforma desplegada en el README.
c. Enlace del PDF con las evidencias de UML, tableros de tareas y demás.
d. Definir en el README mínimo lo siguiente: explicación de la solución propuesta, y la guía paso a paso de la ejecución de la aplicación.

Notas Adicionales:

Se valorará la claridad y organización del código, así como la correcta implementación de las herramientas y prácticas solicitadas.