Prueba técnica: API REST para Procesado de Imágenes y Consulta de Tareas

Descripción General

El objetivo es desarrollar una API REST en **Node.js con TypeScript** que ofrezca:

1. **Procesado de Imágenes**: Generación de variantes en resoluciones específicas a partir de una imagen original.
2. **Consulta de Tareas**: Exposición de información relacionada con las tareas de procesado, incluyendo lógica para consultar el resultado final de las tareas y obtener detalles adicionales, como un "precio" asociado a cada tarea.

La aplicación debe ser organizada y modular, siguiendo las mejores prácticas de desarrollo (API-First, separación de conceptos, gestión de errores y pruebas).

Requisitos Generales

1. **Lenguaje y Herramientas**:

    - Node.js y TypeScript
    - Uso de un framework como **Express** o NestJS para estructurar la API.
    - Base de datos: MongoDB (en local o Docker).
    - Librería para procesar imágenes: **sharp**.
    - Gestión de errores centralizada.
    - Documentación de la API con Swagger/OpenAPI.

2. **Persistencia de Datos**:

    - **Tasks**: Estado de tareas de procesado de imágenes, con un campo adicional `price` que representa el costo asociado a la tarea.
    - **Images**: Información sobre las imágenes originales y sus variantes.

3. **Optimización**:

    - Implementación de consultas eficientes en MongoDB.
    - Uso de índices en campos clave.

4. **Entrega y Buenas Prácticas**:

    - Repositorio Git con historial de commits.
    - **README.md** explicando arquitectura, decisiones tomadas y pasos para ejecutar la aplicación.
    - Tests unitarios e integración.

Requisitos Técnicos

1. Procesado de Imágenes

- Crear un **endpoint REST** para la gestión de tareas de procesado:

Método	**Endpoint**	**Descripción**
POST	/tasks	Crea una tarea de procesado de imagen.
GET	/tasks/:taskId	Devuelve el estado, precio y resultados de una tarea.

Requisitos:

- **POST /tasks**:

    - Recibe: Un path local o URL de la imagen original.

    - Crea una tarea en estado **pendiente** con un precio asignado de forma aleatoria (por ejemplo, entre 5 y 50 unidades monetarias).

    - Almacena el registro en MongoDB y devuelve un `taskId` para consultar su estado.

    - Ejemplo de Respuesta:

```
Unset
{
  "taskId": "65d4a54b89c5e342b2c2c5f6",
  "status": "pending",
  "price": 25.5
```

```
}
```

- **GET /tasks/:taskId**:
    - Devuelve el estado de la tarea (pendiente, completada o fallida), el precio asociado, y los detalles de las variantes procesadas.
    - Si la tarea está completada, se deben incluir los paths de las imágenes generadas.
    - Ejemplo de Respuesta:

```
Unset
{
  "taskId": "65d4a54b89c5e342b2c2c5f6",
  "status": "completed",
  "price": 25.5,
  "images": [
    {
      "resolution": "1024",
      "path": "/output/image1/1024/f322b730b287da77e1c519c7ffef4fc2.jpg"
    },
    {
      "resolution": "800",
      "path": "/output/image1/800/202fd8b3174a774bac24428e8cb230a1.jpg"
    }
  ]
}
```

- Procesamiento de la imagen:
    - Crear variantes de resoluciones **1024px** y **800px** de ancho (manteniendo el aspect ratio).
    - Almacenar las variantes con el formato:

```
Unset
/output/{nombre_original}/{resolucion}/{md5}.{ext}
```

- Persistir en MongoDB:

- **Tasks**: Estado, timestamps, precio y path original.
- **Images**: Path, resolución, md5, timestamp de creación.

---

2. Juego de Pruebas

Implementar pruebas unitarias y de integración para validar el siguiente flujo:

1. **Crear una tarea** usando `POST /tasks`.
    - Validar que se retorna un `taskId`, un estado inicial "pending" y un precio asignado.
2. **Consultar el estado de la tarea** usando `GET /tasks/:taskId`.
    - Si la tarea aún está en estado pendiente, debe devolver solo el precio y el estado.
    - Si la tarea está completada, debe devolver los paths de las variantes generadas y el precio.
3. **Manejo de Errores**:
    - Consultar un `taskId` inexistente debe retornar un error 404.
    - Si ocurre un error en el procesado de la imagen, la tarea debe cambiar a estado "failed" y registrarse el error.

Criterios de Aceptación

1. **Creación de Tareas**:
    - El endpoint POST `/tasks` debe validar la entrada y crear correctamente una tarea con precio asignado.
    - La tarea debe almacenarse en MongoDB.
2. **Consulta de Tareas**:
    - El endpoint GET `/tasks/:taskId` debe retornar correctamente:
        - Estado actual de la tarea.
        - Precio asignado.

    - Paths de las imágenes generadas (si está completada).

3. **Persistencia de Datos**:

    - La base de datos debe contener registros consistentes en la colección `tasks` e `images`.
    - Uso de índices en MongoDB para consultas eficientes.

4. **Gestión de Errores**:

    - Consultas a un `taskId` inexistente deben retornar 404.
    - Errores durante el procesado de imágenes deben reflejarse con estado "failed".

5. **Pruebas**:

    - Pruebas unitarias y de integración deben validar los flujos descritos.

Ejemplo de Datos en MongoDB

Colección `tasks`:

Unset

```
{
  "_id": "65d4a54b89c5e342b2c2c5f6",
  "status": "completed",
  "price": 25.5,
  "createdAt": "2024-06-01T12:00:00",
  "updatedAt": "2024-06-01T12:10:00",
  "originalPath": "/input/image1.jpg",
  "images": [
    { "resolution": "1024", "path": "/output/image1/1024/f322b730b287.jpg" },
    { "resolution": "800", "path": "/output/image1/800/202fd8b3174.jpg" }
  ]
}
```

Entrega

- Repositorio Git con código versionado.

- **README.md** con instrucciones para instalar y ejecutar la aplicación.
- Pruebas unitarias y de integración.
- Scripts para inicialización de la base de datos MongoDB con datos de ejemplo.