Manifiesto de Codificación para Desarrollos de Software - APIs REST

Principios Generales:

1.Simplicidad y Claridad

El código debe ser fácil de entender, sin sobrecomplicar los procesos. Evitar soluciones complejas cuando una más sencilla pueda ser efectiva.

2.Consistencia

El código debe seguir convenciones y patrones consistentes para que sea más fácil de mantener y comprender por otros desarrolladores. Las herramientas como linters(Cada ide debe tener el sonarLint activo con las reglas globales del sonar qube)y formateadores automáticos son recomendables para mantener la consistencia. Además se debe respetar la estructura MVC donde las peticiones entran al controlador ,van al service y de ahí al component(allí se realiza toda la lógica de negocio).

Los nombres de las clases(entidades,dto's,components,service) deben ser claros y consistentes al proceso que se realiza, por ejemplo: si se mapea una tabla deberá tener la estructura CamelCase esto para dar orden y legibilidad asi:

Nombre de la tabla en BD	Nombramiento de la Entidad	Nombramiento del Component	Nombramiento del service	Nombramiento del Repository
DET_DISPONIBILIDAD	DetalleDisponibilidad	DetalleDisponibilidadComponent	DetalleDisponibilidadServicce	DetalleDisponibilidadRepository

3. Modularidad y Reusabilidad

El código debe ser estructurado en módulos pequeños, reutilizables y fáciles de probar. Las funcionalidades comunes deben ser encapsuladas en servicios o bibliotecas reutilizables.

4.Nombramiento de los proyectos y paquetes.

Al momento de crear los servicios basados en el arquetipo, él nombramiento del GroupId y Package debe ser en minúscula y separado por puntos asi: com.ada.module.presupuesto.ordendepago.

Para el ArtefactId debe tener estructura CamelCase así: OrdenDePago.

Para la Versión debe tener la siguiente estructura x.x.x donde si es un proyecto nuevo deberá quedar así:0.0.1-SNAPSHOT.

Estructura de la API:

1.Versionado de la API

Todas las APIs deben tener un esquema de versionado claro. Se recomienda incluir la versión en la URL, como GET /api/v1/usuarios.

2.Uso de Métodos HTTP

Las APIs deben usar los métodos HTTP estándar de manera coherente:

- GET: Para obtener datos.
- POST: Para crear recursos.
- PUT: Para actualizar recursos.
- DELETE: Para eliminar recursos.

3.Codificación de Respuestas HTTP

Las respuestas deben usar códigos de estado HTTP adecuados:

- 200 OK: Solicitud exitosa.
- 201 Created: Recurso creado.
- 204 No Content: Solicitud exitosa sin contenido.
- 400 Bad Request: Solicitud mal formada.
- 401 Unauthorized: Falta de autenticación.
- 403 Forbidden: Prohibido.
- 404 Not Found: Recurso no encontrado.
- 500 Internal Server Error: Error del servidor.

4.Formato de Datos

Las APIs deben usar JSON como formato de intercambio de datos.

Diseño de Endpoints

1. Convenciones de Nombres

Los nombres de los endpoints deben ser coherentes y reflejar los recursos que representan:

- /usuarios (en plural para listas de recursos).
- /usuarios/{id} (para acceder a un recurso específico).

2. Uso de Parámetros en la URL y Query Parameters

Utiliza parámetros de ruta (/usuarios/{id}) para recursos individuales y parámetros de consulta (/usuarios?edad=30) para filtrar, ordenar y paginar los resultados.

3. Manejo de Errores

Las APIs deben devolver errores de manera estandarizada. La respuesta debe incluir un código de estado HTTP adecuado y un cuerpo de respuesta con un mensaje descriptivo del error:

{

```
 "status": "INTERNAL_SERVER_ERROR",
 "statusCode": 404,
  "message": "Se han generado errores al intentar conectarse al cliente.",
  "errors": [
      "No es posible acceder a este cliente, puede estar bloqueado
temporalmente."
  ],
```

```
  "solutions": [
      "Revise el estado de la conexión."
  ],
  "logProcess": ""}
```

Seguridad:

1. Autenticación y Autorización

Las APIs deben requerir autenticación a través de métodos seguros, como JWT (JSON Web Tokens) o OAuth 2.0. No deben almacenarse credenciales en texto claro.

2. Protección contra Inyecciones

Se deben aplicar medidas de seguridad como la validación de entrada, uso de consultas parametrizadas y escaneo de vulnerabilidades de inyección SQL, XSS, CSRF, entre otros.

Documentación:

1. Documentación Automática de la API

Las APIs deben estar documentadas usando herramientas como Swagger/OpenAPI para generar documentación interactiva y siempre actualizada. Los endpoints, parámetros y respuestas deben estar descritos claramente.

2. Comentarios en el Código

Aunque el código debe ser claro, los comentarios deben usarse para explicar “por qué” se hace algo, no “qué” se hace, ya que el segundo debe ser evidente en un código bien escrito.

Pruebas:

1. Cobertura de Pruebas

Es fundamental que las APIs tengan una buena cobertura de pruebas unitarias, de integración y de aceptación. Las pruebas deben ser automatizadas siempre que sea posible.

Manejo de Logs:

1. Uso de Logs

Las APIs deben registrar eventos importantes para facilitar la depuración y la auditoría. Los logs deben contener información relevante pero no deben incluir información sensible como contraseñas o tokens.

2. Niveles de Logs

Utilizar niveles adecuados para los logs, como:

DEBUG: Información detallada para depuración. INFO: Información sobre el flujo normal de la aplicación. WARN: Advertencias de posibles problemas. ERROR: Errores que ocurren durante la ejecución.

Manejo de Logs:

1. Optimización de Consultas Se deben evitar consultas ineficientes que puedan causar cuellos de botella. El uso de índices en las bases de datos y la paginación en respuestas con grandes volúmenes de datos es fundamental.

2. Escalabilidad

El diseño de la API debe permitir una fácil escalabilidad, tanto horizontal como vertical, utilizando técnicas como la cacheización de resultados y la distribución de carga.

←Regresar