Estandarización de Servicios

Estandarización de Servicios

1. Estructura de Respuesta

Todos los servicios deben seguir una estructura de respuesta estándar en formato JSON:

```
{

    "success": false,

    "message": "Mensaje de servicio",

    "data": []

}
```

Reglas para la Respuesta:

- success: Indica si la operación fue exitosa (true o false).
- message: Mensaje claro y descriptivo sobre el resultado de la operación.
- data: Contiene la información devuelta por el servicio. Si no hay datos, debe ser un array vacío ([]).

 Ejemplo de Respuesta Exitosa:

```
{

    "success": true,

    "message": "User retrieved successfully",

    "data": {

        "id": 1,
```

```
        "name": "John Doe",
        "email": "johndoe@example.com"
    }
}
```

 Ejemplo de Respuesta con Error:

```
{
    "success": false,
    "message": "User not found",
    "data": []
}
```

---

2. Generación de Documentación (Swagger)

- Todo servicio debe contar con su respectiva documentación en Swagger.
- La documentación debe incluir:
  - Descripción clara del servicio.
  - Parámetros requeridos y opcionales.
  - Ejemplos de respuestas esperadas.
  - Códigos de estado HTTP posibles.

 Ejemplo de Documentación Swagger en OpenAPI 3.0 (YAML):

paths:

```
  /users/{id}:
    get:
      summary: Get user by ID
```

```
description: Retrieves a user using their unique ID.

parameters:

  - name: id

    in: path

    required: true

    schema:

      type: integer

responses:

  "200":

    description: User retrieved successfully

    content:

      application/json:

        schema:

          type: object

          properties:

            success:

              type: boolean

            message:

              type: string

            data:
```

```
            type: object
            properties:
              id:
                type: integer
              name:
                type: string
              email:
                type: string
"404":
  description: User not found
  content:
    application/json:
      schema:
        type: object
        properties:
          success:
            type: boolean
          message:
            type: string
          data:
```

```
                    type: array

                    items: {}
```

---

3. Códigos de Estado HTTP Estándar

- 200 OK - Solicitud exitosa.
- 500 Internal Server Error - Error interno del servidor.

---

4. Seguridad y Buenas Prácticas

- Validar siempre los datos de entrada antes de procesarlos.
- Evitar exponer información sensible en los mensajes de error.

---

5. Uso de Estructuras de Control

Para mejorar la consistencia y calidad del código, se deben seguir estas reglas al manejar estructuras de control:

Condicionales (if/else)

- Evitar múltiples anidaciones innecesarias.
- Utilizar early return para mejorar la legibilidad.

 Ejemplo Correcto:

```
if not user:

    return {"success": false, "message": "User not found", "data": []}

return {"success": true, "message": "User retrieved", "data": user}
```

 Ejemplo Incorrecto:

```
if user:

    return {"success": true, "message": "User retrieved", "data": user}

else:

    return {"success": false, "message": "User not found", "data": []}
```

Bucles (for, while)

- Utilizar comprensiones de listas cuando sea posible.
- Evitar bucles anidados innecesarios.

 Ejemplo Correcto:

```
even_numbers = [x for x in numbers if x % 2 == 0]
```

 Ejemplo Incorrecto:

```
even_numbers = []

for x in numbers:

    if x % 2 == 0:

        even_numbers.append(x)
```

Revisión #1
Creado 2025-11-14 16:45:36 -05 por Elian
Actualizado 2025-11-14 16:52:03 -05 por Elian