


# 7 ERROS COMUNS AO CRIAR APIS E COMO EVITA-LOS!



# 01 Não Usar Status Codes Corretamente

 O problema: Muitas APIs retornam 200 OK para tudo, mesmo quando há erros. Isso torna difícil identificar falhas no sistema.

 Exemplo ruim:

✓ 200 OK → Para erro de autenticação ✗

✓ Como corrigir:

Use os códigos HTTP corretos:

✓ 200 OK → Requisição bem-sucedida

✓ 201 Created → Recurso criado com sucesso

✓ 400 Bad Request → Erro do cliente

✓ 401 Unauthorized → Falha na autenticação

✓ 404 Not Found → Recurso não encontrado

✓ 500 Internal Server Error → Erro no servidor

## 02 Falta de Validação de Entrada

📌 O problema: Não validar os dados recebidos permite inserção de dados incorretos, insegurança e até falhas na aplicação.

💡 Exemplo ruim:

Um usuário pode enviar um email sem o formato correto ou deixar campos obrigatórios vazios.


✅ Como corrigir:

✓ Use bibliotecas como Joi ou Express Validator para garantir que os dados enviados são válidos.

✓ Defina regras como:

- O campo email precisa ter @ e .com
- O campo senha precisa ter no mínimo 8 caracteres
- ✓ Sempre retorne uma mensagem de erro clara para o usuário.

## 03 Expor Informações Sensíveis

 Exemplo ruim: Retornar erro completo:  
Aqui, o erro expõe a senha do banco!  

```
{"error": "Database connection failed: user=root,  
password=123456"}
```

 Como corrigir:

✓ Nunca exponha detalhes internos do sistema.

✓ Configure mensagens de erro genéricas:

json

✓ Use variáveis de ambiente (.env) para armazenar credenciais.

```
{"error": "Erro interno no servidor"}
```

## 04 Falta de Paginação nas Respostas

📌 O problema: Retornar grandes volumes de dados sem paginação pode sobrecarregar a API, tornando-a lenta e difícil de usar.

🚨 Exemplo ruim:

Uma API retorna 10.000 registros de uma vez! Isso torra o servidor e atrapalha o cliente.

✅ Como corrigir:

✓ Use parâmetros de paginação, como limit e offset:

GET /users?limit=10&offset=20

✓ Retorne informações sobre a paginação, como o total de registros.

# 05 Não Implementar Autenticação e Controle de Acesso

📌 O problema: Muitas APIs não têm autenticação, permitindo que qualquer um acesse os dados!

🚨 Exemplo ruim:

✗ Qualquer pessoa pode chamar `/users` e obter todos os dados sem precisar de um token de autenticação.


✅ Como corrigir:

✓ Use JWT (JSON Web Token) para autenticação segura.

✓ Implemente controle de acesso baseado em permissões de usuário (ex: admin, usuário comum).

✓ Sempre valide o token antes de processar a requisição.

## 06 Não Usar Cache para Melhorar Performance

 O problema: Consultar o banco de dados toda vez que um cliente faz uma requisição pode tornar a API lenta e cara.


 Exemplo ruim:

Cada chamada para /products consulta o banco do zero, mesmo que os dados não tenham mudado.


 Como corrigir:


 Implemente cache com Redis para armazenar consultas frequentes.

 Defina tempo de expiração para o cache (TTL).

 Use ETags para evitar transferências desnecessárias de dados.

# 07 Não Monitorar Erros e Logs

 O problema: Muitas APIs não têm um sistema de logs, tornando difícil detectar falhas.

 Consequência:  
Se um erro acontece, você não sabe o que deu errado até o cliente reclamar.

 Como corrigir:


✓ Use ferramentas como Winston ou Morgan para gerar logs detalhados.

✓ Integre sua API com um serviço de monitoramento como Sentry.

✓ Registre erros críticos e alertas automaticamente.



# Resumo e Checklist

 Agora que você conhece os principais erros, aqui está um checklist rápido para melhorar sua API:

- ✓ Retorne os status codes corretamente
- ✓ Valide todas as entradas do usuário
- ✓ Proteja dados sensíveis
- ✓ Implemente paginação em respostas grandes
- ✓ Use JWT para autenticação
- ✓ Acelere a API com cache
- ✓ Registre logs e monitore erros

**Evitar esses erros tornará sua API mais segura, eficiente e escalável!**

Você já cometeu algum desses erros?

Qual foi o pior problema que já enfrentou ao criar uma API?

Comenta aqui! 🙌🚀

