Aula 7 – Códigos de Status HTTP: Semântica e Uso Correto

Imagine que você está em um restaurante movimentado, fazendo um pedido. Cada resposta que o garçom lhe dá – "Pedido anotado!", "Desculpe, esse prato acabou", "Seu pedido está a caminho" – é uma forma de comunicação que informa o status da sua solicitação. No mundo das APIs e microserviços, essa comunicação é feita através dos Códigos de Status HTTP. Eles são a linguagem universal que servidores e clientes usam para se entender, indicando se uma requisição foi bem-sucedida, se algo deu errado, ou se alguma ação adicional é necessária.

Compreender e aplicar corretamente esses códigos não é apenas uma boa prática; é fundamental para construir sistemas robustos, intuitivos e fáceis de depurar. Uma API que retorna códigos de status ambíguos ou incorretos pode levar a frustrações para os desenvolvedores que a consomem e a falhas silenciosas em aplicações. É como ter um garçom que sempre responde "Ok" para tudo, sem realmente informar o que aconteceu.

Nesta aula, vamos desvendar a semântica por trás de cada categoria de código de status HTTP, focando nos mais importantes e frequentemente utilizados. Você aprenderá a identificar a mensagem correta para cada cenário, aprimorando a comunicação entre seus serviços e garantindo que suas APIs sejam tão claras e eficientes quanto possível. Ao final, você será capaz de projetar respostas de API que não apenas funcionam, mas que também guiam o consumidor da API de forma eficaz, otimizando o desenvolvimento e a manutenção de sistemas distribuídos.

A Linguagem Silenciosa das APIs: Categorias de Códigos HTTP

No universo das APIs, a comunicação é a chave. Cada vez que uma aplicação cliente (como um navegador ou um aplicativo móvel) envia uma requisição para um servidor, ela espera uma resposta. Essa resposta não é apenas um conjunto de dados, mas também um sinal sobre o que aconteceu com a requisição. Os Códigos de Status HTTP são exatamente isso: sinais padronizados que informam o resultado de uma operação. Eles são como o painel de controle de um avião, onde cada luz indica um estado específico, permitindo que o piloto (o desenvolvedor) saiba exatamente o que está acontecendo.

Esses códigos são divididos em cinco categorias principais, cada uma com um propósito bem definido. Entender essa categorização é o primeiro passo para dominar a arte de projetar APIs eficientes. Pense nelas como os capítulos de um livro, onde cada capítulo aborda um tipo diferente de história: informativas, de sucesso, de redirecionamento, de erro do cliente e de erro do servidor.

Vamos explorar cada uma dessas categorias, começando pelas mais "positivas" e avançando para os cenários de erro, para que você possa construir uma base sólida sobre como as APIs se comunicam e como você pode usar essa comunicação a seu favor.

1xx Informacionais

Respostas provisórias indicando processamento em andamento

2xx Sucesso

Requisição recebida, compreendida e processada com êxito

3xx Redirecionamento

Ação adicional necessária para completar a requisição

4xx Erro do Cliente

Problema na requisição enviada pelo cliente

5xx Erro do Servidor

Falha no processamento pelo servidor

1xx: Informacionais – Apenas um "Estou Pensando..."

A categoria 1xx é a menos comum no dia a dia do desenvolvimento de APIs RESTful, mas é importante conhecê-la. Esses códigos são respostas provisórias que indicam que a requisição foi recebida e compreendida, e que o processamento continua. Eles são como um "aguarde" ou "estou processando" do servidor, antes que uma resposta final seja enviada.

Imagine que você está ligando para um serviço de atendimento e, antes de ser conectado a um atendente, ouve uma mensagem gravada como "Sua chamada está sendo transferida, por favor, aguarde". Essa mensagem não é a solução final para o seu problema, mas informa que sua solicitação foi recebida e está em andamento. Da mesma forma, um código 1xx não encerra a comunicação, mas prepara o cliente para uma resposta subsequente.

100 Continue

Permite que o cliente envie apenas os cabeçalhos da requisição e aguarde confirmação antes de enviar o corpo, economizando largura de banda.

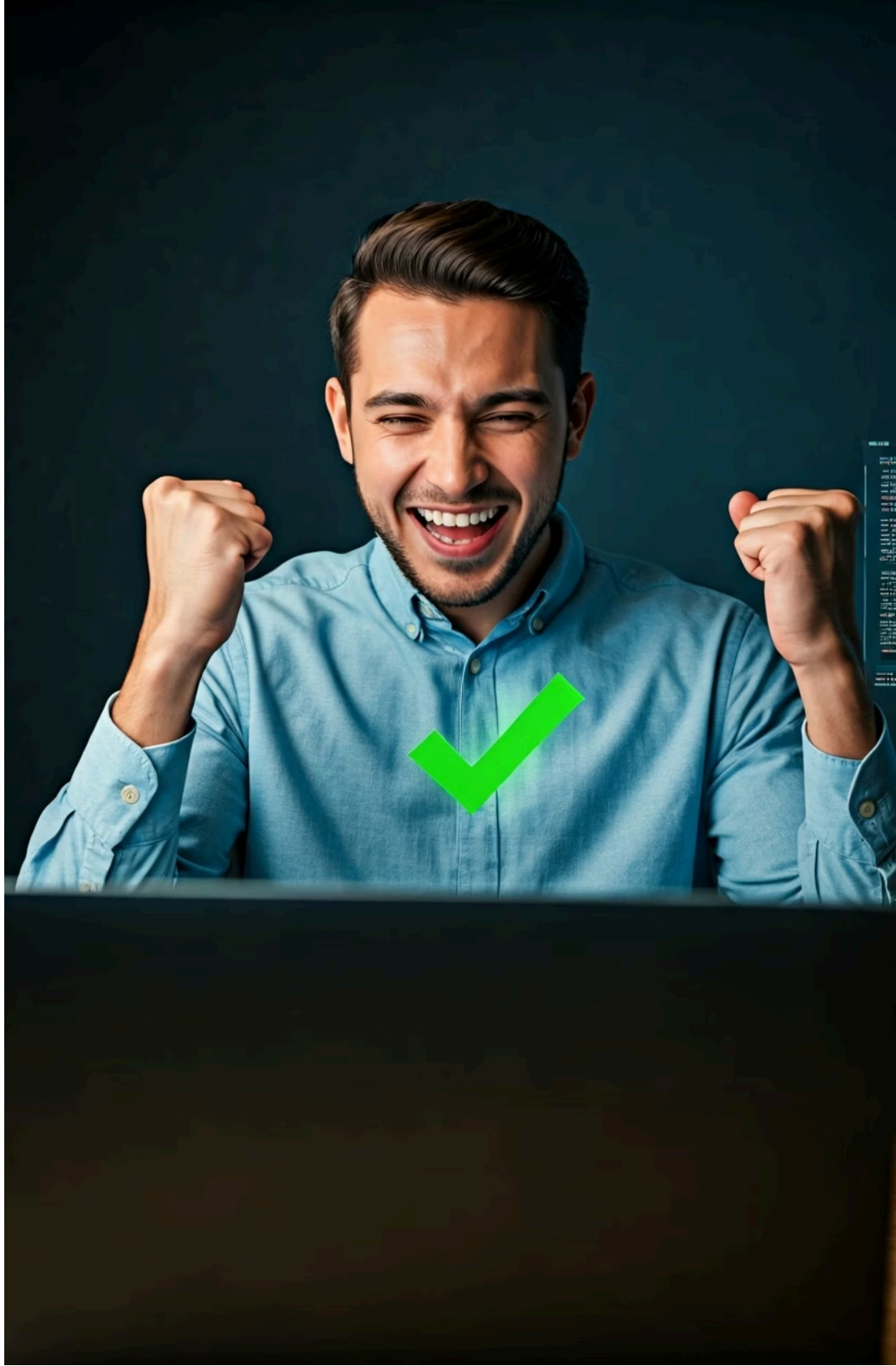
Um exemplo clássico é o 100 Continue, que permite que um cliente envie apenas os cabeçalhos da requisição e aguarde a confirmação do servidor antes de enviar o corpo da requisição, economizando largura de banda em casos onde o corpo é grande e o servidor pode rejeitar a requisição apenas pelos cabeçalhos. Embora não seja frequentemente implementado em APIs REST modernas, sua existência ilustra a capacidade do HTTP de gerenciar fluxos de comunicação complexos.

2xx: Sucesso – "Missão Cumprida!"

Esta é a categoria que todos os desenvolvedores amam ver! Os códigos 2xx indicam que a ação solicitada pelo cliente foi recebida, compreendida, aceita e processada com sucesso. Eles são a confirmação de que tudo ocorreu como esperado, e o servidor entregou o que foi pedido ou realizou a operação com êxito.

Pense em um aplicativo de entrega de comida. Quando você finaliza um pedido e vê a mensagem "Seu pedido foi realizado com sucesso!", acompanhada de um número de rastreamento, essa é a representação de um status 2xx. O sistema recebeu sua solicitação, processou o pagamento e encaminhou o pedido para o restaurante. É a garantia de que sua interação com o serviço foi frutífera.

Dentro dessa categoria, alguns códigos são pilares para qualquer API bem projetada. Eles comunicam não apenas o sucesso, mas também a *natureza* desse sucesso, permitindo que o cliente reaja de forma apropriada. Vamos detalhar os mais importantes, que você usará constantemente ao construir e consumir APIs.



Os Códigos de Sucesso Essenciais

200 OK: O Sucesso Padrão

O 200 OK é o código de status HTTP mais comum e representa o sucesso genérico de uma requisição. Ele significa que a requisição foi bem-sucedida e o corpo da resposta contém os dados solicitados.

Imagine que você está pedindo um extrato bancário online. Quando o sistema exibe seu extrato, isso é um 200 OK. A requisição para obter os dados foi processada, e os dados estão ali, prontos para você. É o "tudo certo, aqui está o que você pediu" do servidor. Em APIs, é o que você espera ao fazer um GET para buscar recursos, ou um PUT/POST que não retorna um novo recurso, mas confirma uma atualização.

201 Created: Um Novo Recurso Nasceu

O 201 Created é usado quando uma requisição (geralmente POST) resulta na criação de um novo recurso no servidor. A resposta deve incluir um cabeçalho Location apontando para a URI do novo recurso e, opcionalmente, o próprio recurso no corpo da resposta.

Pense em criar uma nova conta em uma rede social. Após preencher o formulário e clicar em "Cadastrar", se tudo der certo, o servidor retorna um 201 Created, indicando que seu perfil foi criado e, muitas vezes, já te redireciona para a página do seu novo perfil. Este código é crucial para operações de criação, pois informa ao cliente não apenas que a criação foi bem-sucedida, mas também onde o novo recurso pode ser acessado.

204 No Content: Sucesso Silencioso

O 204 No Content indica que a requisição foi bem-sucedida, mas o servidor não tem nenhuma informação adicional para enviar no corpo da resposta. É frequentemente usado para operações DELETE ou PUT onde o cliente não precisa de uma confirmação de dados, apenas do sucesso da operação.

Considere a ação de excluir um item do seu carrinho de compras online. Você clica em "Remover", o item desaparece, e você não espera que o servidor retorne o carrinho inteiro novamente, apenas a confirmação de que a exclusão foi bem-sucedida. O 204 No Content é perfeito para isso, economizando largura de banda e sinalizando claramente que não há conteúdo para processar na resposta.

3xx: Redirecionamento – "Por Favor, Vá Para Outro Lugar"

Os códigos 3xx informam ao cliente que a requisição precisa de uma ação adicional para ser concluída, geralmente um redirecionamento para outra URL. Eles são como um aviso de "endereço mudou" no mundo digital, guiando o cliente para o local correto do recurso ou serviço.

Imagine que você está procurando um livro em uma biblioteca e descobre que ele foi movido para uma nova seção. O bibliotecário não diz que o livro não existe, mas sim "Ele está na seção de ficção científica, corredor 3". Essa é a essência de um redirecionamento: a informação existe, mas em outro lugar. Em APIs, isso pode acontecer quando um recurso muda de URI ou quando há balanceamento de carga.

Embora menos comuns em interações diretas de API para API (onde o cliente geralmente é programado para saber a URI correta), eles são vitais para a web e podem aparecer em cenários de autenticação ou migração de serviços.

301 Moved Permanently

Mudança Definitiva

O 301 Moved Permanently indica que o recurso solicitado foi permanentemente movido para uma nova URI. O cliente deve atualizar seus links e futuras requisições devem ser feitas para a nova URI.

Se um serviço de API muda a estrutura de suas URLs para um recurso específico, ele pode usar um 301 para guiar os clientes antigos para o novo caminho. Isso é crucial para a manutenção da integridade de links e para o SEO (Search Engine Optimization) em contextos web, garantindo que os motores de busca e os clientes saibam onde encontrar o recurso de forma definitiva.

302 Found

Uma Parada Rápida

O 302 Found (anteriormente Moved Temporarily) indica que o recurso solicitado está temporariamente disponível em uma URI diferente. O cliente não deve atualizar seus links, pois a localização original pode ser restaurada no futuro.

Este código é frequentemente usado em cenários de balanceamento de carga, onde o servidor pode redirecionar uma requisição para uma instância diferente do serviço, ou em fluxos de autenticação, onde após um login bem-sucedido, o usuário é temporariamente redirecionado para uma página de boas-vindas. A chave é a **natureza temporária** do redirecionamento, sinalizando que a URI original ainda é válida para futuras requisições.

4xx: Erro do Cliente – "A Culpa é Sua!"

Chegamos à categoria que indica que algo deu errado por parte do cliente. Os códigos 4xx são essenciais para depurar e para fornecer feedback útil aos consumidores da sua API. Eles são como um "você digitou algo errado" ou "você não tem permissão para fazer isso".

Pense em tentar sacar dinheiro de um caixa eletrônico com um cartão inválido ou um PIN incorreto. O caixa não está quebrado, mas a operação não pode ser concluída por causa de uma falha na sua entrada de dados ou na sua autorização. Em APIs, isso se traduz em requisições malformadas, credenciais inválidas ou tentativas de acessar recursos não permitidos.

A clareza nas mensagens de erro 4xx é um diferencial para qualquer API. Em um ambiente de microserviços, onde diferentes equipes consomem suas APIs, mensagens de erro vagas podem levar a horas de depuração desnecessária. Uma boa mensagem de erro 4xx não apenas informa que algo falhou, mas *por que* falhou e, idealmente, *como corrigir*.

Os Principais Erros do Cliente

400 Bad Request

O Pedido Malfeito

O 400 Bad Request é um código genérico que indica que o servidor não pôde processar a requisição devido a uma sintaxe inválida ou a um formato incorreto do corpo da requisição. É o "não entendi o que você me pediu" do servidor.

Imagine que você está preenchendo um formulário online e, ao tentar enviar, recebe uma mensagem de erro dizendo "Campo 'email' inválido" ou "Data de nascimento fora do formato esperado". Isso é um 400 Bad Request. A requisição chegou ao servidor, mas os dados enviados não estavam de acordo com o que era esperado. Em APIs, isso pode ser um JSON malformado, parâmetros de query ausentes ou com valores inválidos.

401 Unauthorized

Quem é Você?

O 401 Unauthorized indica que a requisição não foi aplicada porque não possui credenciais de autenticação válidas para o recurso alvo. É o "você precisa se identificar" do servidor.

Pense em tentar entrar em um prédio que exige um crachá, mas você não o tem ou seu crachá está vencido. O porteiro (servidor) não te deixa entrar. Em APIs, isso acontece quando um token de autenticação está ausente, é inválido ou expirou. É um pilar da **Segurança "API-First"**, garantindo que apenas usuários autenticados possam interagir com certos endpoints.

---

403 Forbidden

Você Não Pode Entrar Aqui

O 403 Forbidden indica que o servidor entendeu a requisição, mas se recusa a autorizá-la. Diferente do 401, o cliente *se autenticou*, mas não possui as permissões necessárias para acessar o recurso ou executar a ação. É o "eu sei quem você é, mas você não tem permissão para isso" do servidor.

Voltando ao exemplo do prédio, você tem seu crachá (autenticação), mas ele só te dá acesso ao térreo, e você está tentando entrar no escritório do CEO. O porteiro te reconhece, mas te impede. Em APIs, isso pode ocorrer se um usuário autenticado tentar acessar dados de outro usuário, ou tentar realizar uma operação administrativa sem ter o papel de administrador.

404 Not Found

Onde Está o Recurso?

O 404 Not Found é talvez o código de erro mais conhecido da web. Ele indica que o servidor não conseguiu encontrar o recurso solicitado. É o "o que você procura não existe neste endereço" do servidor.

Quando você digita uma URL errada no navegador e vê a famosa página "404", é exatamente isso. Em APIs, significa que o endpoint ou o ID do recurso que você tentou acessar não corresponde a nada que o servidor conheça. É crucial para evitar que clientes tentem interagir com recursos inexistentes, e uma boa prática é garantir que este erro seja retornado apenas quando o recurso *realmente* não existe, e não por outros problemas.

5xx: Erro do Servidor – "A Culpa é Minha!"

Os códigos 5xx são os mais temidos pelos desenvolvedores, pois indicam que o servidor encontrou uma condição inesperada que o impediu de atender à requisição. É o "algo deu muito errado do meu lado" do servidor.

Imagine que você está em um restaurante e, de repente, a cozinha pega fogo ou o sistema de pedidos para de funcionar. Não importa o que você pediu, o restaurante não pode te atender porque há um problema interno grave. Em APIs, isso pode ser um bug no código, uma falha de banco de dados, um serviço dependente indisponível ou um problema de infraestrutura.

Lidar com erros 5xx é um desafio em arquiteturas de microserviços, onde a falha de um único serviço pode ter um efeito cascata. É aqui que conceitos como **Observabilidade** (Logs, Métricas e Tracing) e **Orquestração de Containers** (Kubernetes) se tornam vitais, permitindo que as equipes identifiquem, diagnostiquem e resolvam rapidamente esses problemas, minimizando o tempo de inatividade e garantindo a resiliência do sistema.

Entendendo os Erros do Servidor

500 Internal Server Error

O Erro Genérico do Servidor

O 500 Internal Server Error é o código de erro mais genérico da categoria 5xx. Ele indica que o servidor encontrou uma condição inesperada que o impediu de atender à requisição, e não há um código de status mais específico disponível. É o "não sei o que aconteceu, mas não deu certo" do servidor.

Este erro geralmente aponta para um problema não tratado no código da aplicação, uma exceção não capturada, ou uma falha inesperada em um serviço dependente. Em um ambiente de **Containerização como Padrão** com Docker e Kubernetes, um 500 pode indicar que um contêiner falhou e foi reiniciado, ou que um serviço não conseguiu se comunicar com outro. A chave para lidar com 500s é ter um bom sistema de logs e monitoramento para identificar a causa raiz rapidamente.

Outros Erros 5xx Importantes

Embora o 500 seja o mais comum, existem outros códigos 5xx que podem ser úteis para cenários mais específicos:

- **502 Bad Gateway**: O servidor, atuando como gateway ou proxy, recebeu uma resposta inválida de um servidor upstream (outro serviço ou servidor que ele tentou acessar).
- **503 Service Unavailable**: O servidor está temporariamente incapaz de lidar com a requisição devido a sobrecarga ou manutenção. Geralmente, inclui um cabeçalho Retry-After indicando quando o serviço pode estar disponível novamente.
- **504 Gateway Timeout**: O servidor, atuando como gateway ou proxy, não recebeu uma resposta em tempo hábil de um servidor upstream.

Esses códigos ajudam a refinar o diagnóstico de problemas em arquiteturas distribuídas, onde a comunicação entre múltiplos serviços é constante.

Fornecendo Mensagens de Erro Claras e Úteis

Retornar o código de status HTTP correto é apenas metade da batalha; a outra metade é fornecer uma mensagem de erro que seja clara, útil e acionável para o consumidor da API. Uma mensagem de erro vaga como "Erro interno" (mesmo com um 500) é tão inútil quanto um garçom que apenas balança a cabeça quando o pedido dá errado.

Pense em um aplicativo de banco que, ao tentar fazer uma transferência, simplesmente diz "Erro". Você não sabe se foi falta de saldo, conta inválida, problema de conexão ou um bug. Agora, imagine se ele dissesse: "Saldo insuficiente para a transferência. Seu saldo atual é R$ 150,00." Isso é uma mensagem de erro útil! Ela informa o problema, o contexto e, implicitamente, a solução (adicionar mais dinheiro).

Para APIs, isso significa estruturar as respostas de erro de forma consistente, incluindo detalhes que ajudem o desenvolvedor a entender e corrigir o problema. Uma boa prática é incluir um código de erro específico da aplicação, uma mensagem legível por humanos, e, se possível, detalhes adicionais ou um link para a documentação.

Exemplo de Estrutura de Erro

```
{
  "code": "INVALID_EMAIL_FORMAT",
  "message": "O formato do e-mail fornecido é inválido.",
  "details": "O campo 'email' deve seguir o padrão 'usuario@dominio.com'.",
  "more_info": "https://api.exemplo.com/docs/erros#INVALID_EMAIL_FORMAT"
}
```

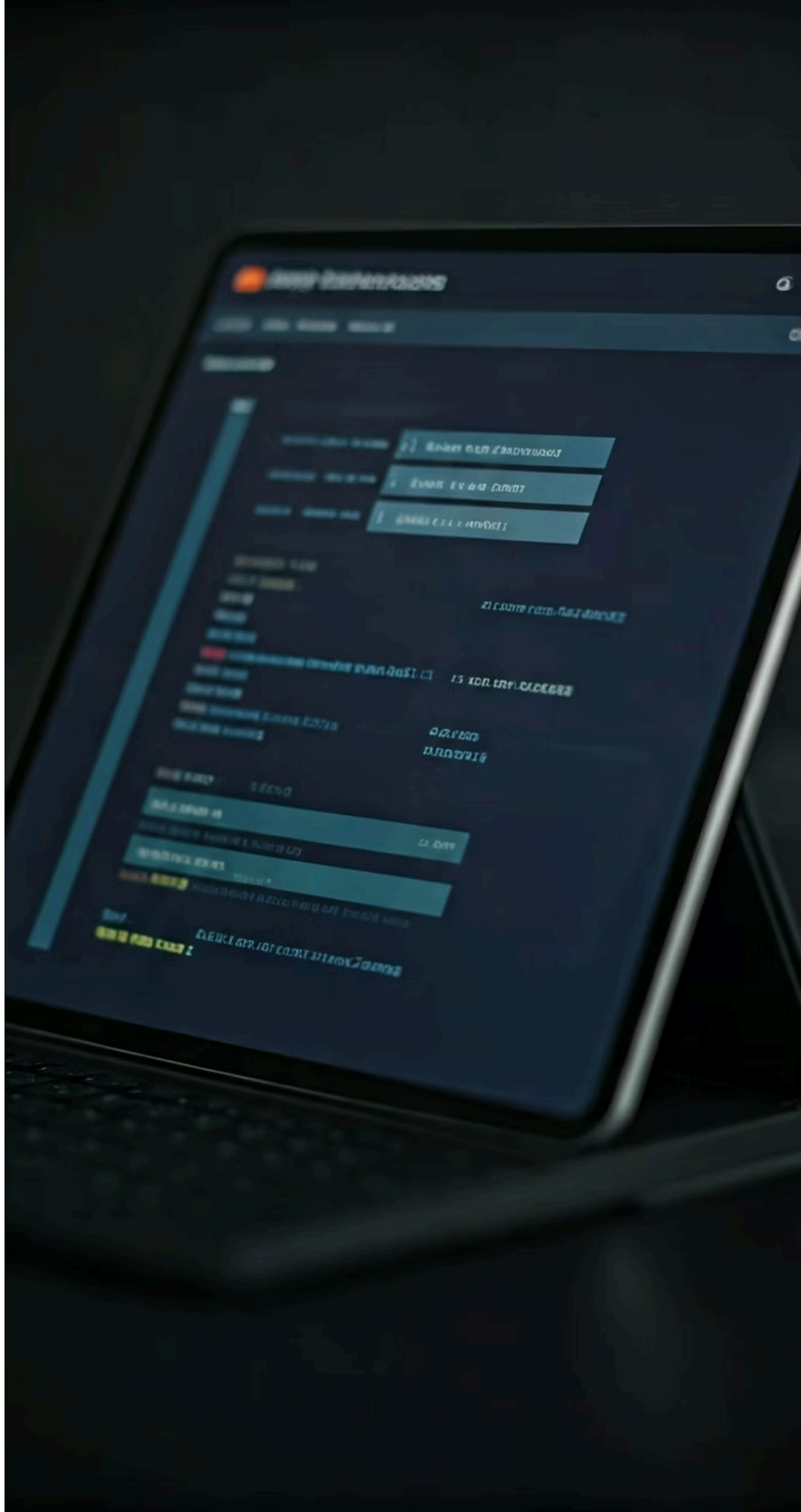
Essa abordagem melhora drasticamente a experiência do desenvolvedor, reduzindo o tempo de depuração e tornando suas APIs mais amigáveis e eficientes.

A Importância da Consistência e da Documentação

A consistência é a espinha dorsal de uma API bem-sucedida. Não basta usar os códigos de status HTTP corretamente; é preciso que essa utilização seja previsível em toda a sua API. Se um 400 Bad Request significa uma coisa em um endpoint e outra em outro, a confusão será inevitável.

Imagine que cada semáforo em uma cidade usasse cores diferentes para "pare" e "siga". O caos seria instantâneo! Da mesma forma, uma API inconsistente gera atrito e frustração. É fundamental definir padrões claros para o uso dos códigos de status e para a estrutura das mensagens de erro, e aderir a eles rigorosamente.

Além da consistência, a documentação é sua melhor amiga. Uma documentação clara e atualizada que detalha quais códigos de status HTTP são esperados para cada endpoint, e quais são os possíveis formatos de mensagens de erro, é um recurso inestimável. Ferramentas como OpenAPI (Swagger) permitem que você defina esses detalhes de forma programática, gerando documentação interativa que serve como um contrato entre sua API e seus consumidores. Isso não só facilita a integração, mas também atua como um guia para a manutenção e evolução da sua API, garantindo que todos falem a mesma "linguagem" HTTP.



Códigos de Status em Microserviços e Observabilidade

Em uma arquitetura de microserviços, onde dezenas ou centenas de serviços se comunicam entre si, a correta utilização dos códigos de status HTTP ganha uma camada extra de importância. Um erro em um serviço pode se propagar, e a capacidade de identificar rapidamente a origem e a natureza desse erro é crucial para a saúde do sistema.

Pense em uma orquestra. Se um músico desafina, o maestro precisa identificar rapidamente qual instrumento e qual músico está causando o problema para corrigir. Em microserviços, os códigos de status HTTP são as "notas" que cada serviço toca. Se um serviço retorna um 500 Internal Server Error, ele está sinalizando um problema interno, que pode ser detectado por ferramentas de **Observabilidade**.

Logs

Podem registrar os códigos de status de cada requisição, permitindo análises detalhadas de padrões de erro e comportamento do sistema.

Métricas

Podem contar a frequência de cada categoria de status (quantos 2xx, 4xx, 5xx), gerando alertas quando os erros 5xx aumentam além de um limite aceitável.

Tracing

Pode seguir uma requisição através de múltiplos serviços, revelando exatamente qual serviço retornou um erro e qual código de status foi gerado em cada etapa.

Essa integração entre códigos de status e observabilidade é fundamental para operar sistemas distribuídos complexos, especialmente em ambientes orquestrados por **Kubernetes (K8s)**, onde a automação da recuperação de falhas depende de sinais claros sobre o estado dos serviços.

Cenários Comuns: Quando Usar Qual Código?

Dominar os códigos de status HTTP é como aprender a usar a ferramenta certa para cada trabalho. Não se trata apenas de memorizar, mas de entender o contexto. Vamos consolidar o conhecimento com alguns cenários práticos que você provavelmente encontrará ao desenvolver APIs.

01

Cenário 1: Criando um Usuário

- **Requisição:** POST /users com dados do novo usuário.
- **Sucesso:** 201 Created (com Location para /users/{id_do_novo_usuario})
- **Erro (dados inválidos):** 400 Bad Request (se o email estiver malformado, por exemplo)
- **Erro (usuário já existe):** 409 Conflict (um código que indica conflito com o estado atual do recurso)

02

Cenário 2: Buscando Detalhes de um Produto

- **Requisição:** GET /products/{id_do_produto}
- **Sucesso:** 200 OK (com os dados do produto)
- **Erro (produto não encontrado):** 404 Not Found
- **Erro (não autorizado):** 401 Unauthorized (se precisar de autenticação para ver produtos)

03

Cenário 3: Atualizando um Pedido

- **Requisição:** PUT /orders/{id_do_pedido} com novos dados.
- **Sucesso:** 200 OK (com os dados atualizados) ou 204 No Content (se não houver dados para retornar)
- **Erro (permissão negada):** 403 Forbidden (se o usuário não for o dono do pedido)
- **Erro (problema no servidor):** 500 Internal Server Error (se o banco de dados falhar)

Esses exemplos mostram como a escolha do código de status reflete a natureza exata da interação e do resultado, fornecendo um feedback preciso e padronizado.

Em Prática: Construindo APIs Mais Robustas

A aplicação correta dos códigos de status HTTP é um pilar para a construção de APIs robustas e amigáveis. Ao longo desta aula, exploramos as categorias 1xx, 2xx, 3xx, 4xx e 5xx, detalhando os códigos mais importantes e seus usos semânticos. Vimos como o 200 OK e 201 Created celebram o sucesso, como o 301 Moved Permanently guia o cliente, e como os 400 Bad Request, 401 Unauthorized, 403 Forbidden e 404 Not Found apontam para problemas do lado do cliente. Finalmente, entendemos que o 500 Internal Server Error e seus irmãos 5xx indicam falhas no servidor, ressaltando a importância da observabilidade em microserviços.

> Para aplicar este conhecimento, sempre se pergunte: "Qual é a *verdadeira* natureza do resultado desta requisição?"

É um sucesso completo? Um sucesso com criação? Um erro do cliente? Um erro do servidor? A resposta guiará você ao código correto. Além disso, lembre-se de que uma mensagem de erro clara e consistente é tão valiosa quanto o próprio código de status, transformando frustração em solução.

Autoavaliação

Questão 1

Qual código de status HTTP deve ser retornado quando uma requisição POST cria um novo recurso com sucesso no servidor?

1. 200 OK
2. 204 No Content
3. 201 Created
4. 301 Moved Permanently

Questão 2

Um cliente tenta acessar um recurso que exige autenticação, mas não fornece um token válido. Qual código de status HTTP o servidor deve retornar?

1. 403 Forbidden
2. 404 Not Found
3. 401 Unauthorized
4. 500 Internal Server Error

Questão 3

Em um cenário de microserviços, se um serviço A tenta se comunicar com um serviço B, mas o serviço B está temporariamente indisponível devido a uma sobrecarga, qual código de status HTTP o serviço B (ou um proxy intermediário) deveria retornar?

1. 400 Bad Request
2. 503 Service Unavailable
3. 200 OK
4. 404 Not Found

Questão 4

Qual das seguintes afirmações sobre o código de status 204 No Content está **correta**?

1. Indica que a requisição falhou devido a um erro de sintaxe.
2. É usado para indicar que um novo recurso foi criado com sucesso.
3. Significa que a requisição foi bem-sucedida, mas não há corpo de resposta para enviar.
4. Sugere que o recurso solicitado foi permanentemente movido para outra URI.

Gabarito

1. c) | **2.** c) | **3.** b) | **4.** c)

Questão Discursiva

Explique a diferença semântica entre os códigos de status HTTP 401 Unauthorized e 403 Forbidden, e forneça um exemplo prático de uso para cada um em um contexto de API REST.

Próximos Passos

Próxima Aula

Aula 8

Paginação, Filtros e Ordenação em APIs REST

Continue sua jornada aprendendo como estruturar endpoints eficientes para grandes volumes de dados.

Recursos Adicionais

- **MDN Web Docs - HTTP Status Codes:** Para uma referência completa e detalhada de todos os códigos de status.
- **RFC 9110 (HTTP Semantics):** Para a especificação oficial e aprofundada dos códigos de status HTTP.
- **Artigos sobre Observabilidade em Microserviços:** Para entender como monitorar e diagnosticar problemas em sistemas distribuídos.

NOTA IMPORTANTE: As informações regulatórias/legais/técnicas desta aula estão atualizadas até 2025. Consulte sempre fontes oficiais para verificar alterações.