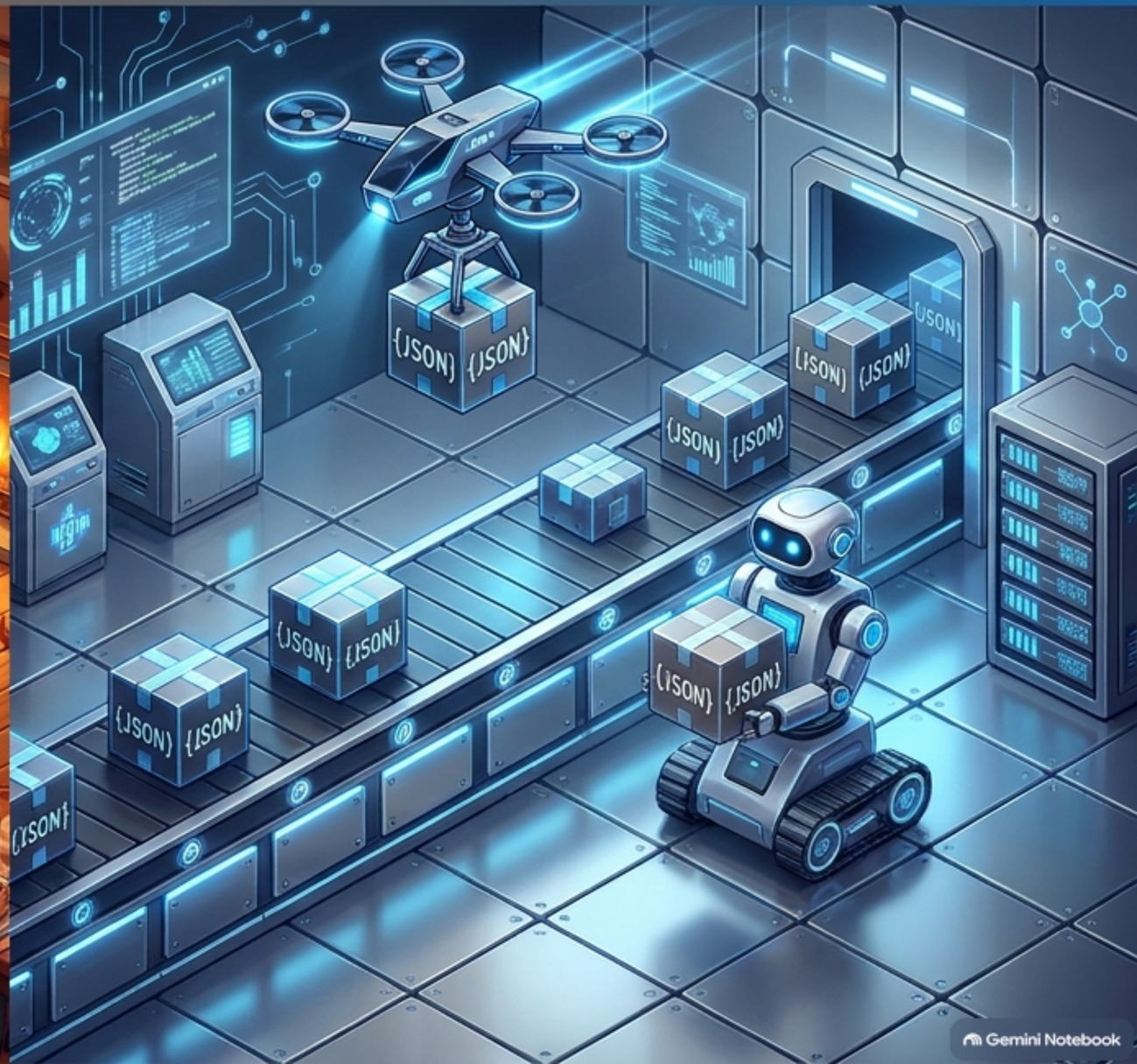
Sua Primeira API com Django REST Framework

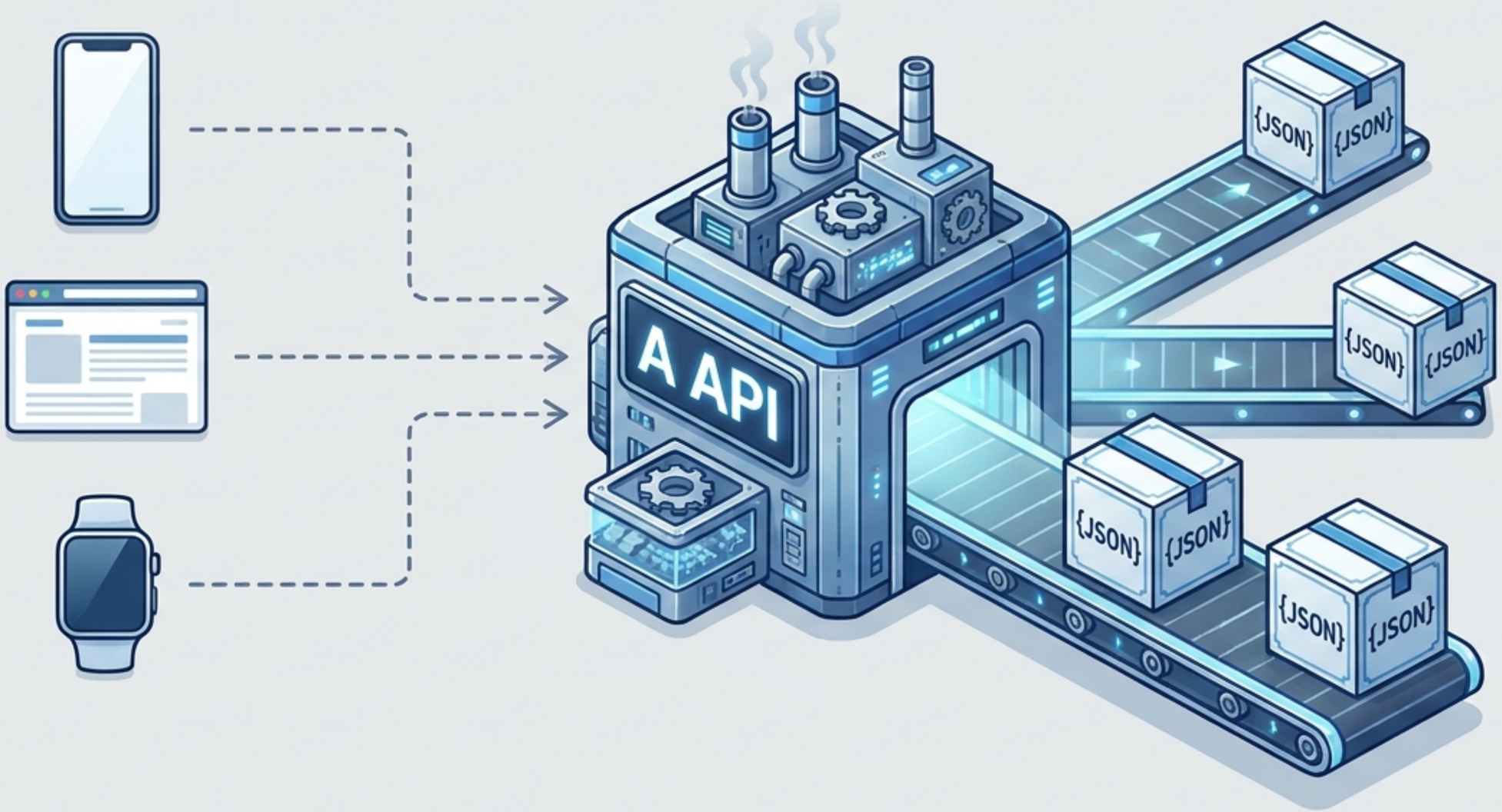
Até agora, a nossa cozinha servia apenas no salão. Chegou a hora de montar o delivery.





O que é uma API REST? O Cardápio com Regras

A API define três coisas fundamentais: o que pode ser pedido, como pedir, e o formato da entrega.
O grande valor da API é o contrato: quem pede não precisa saber como a cozinha funciona.

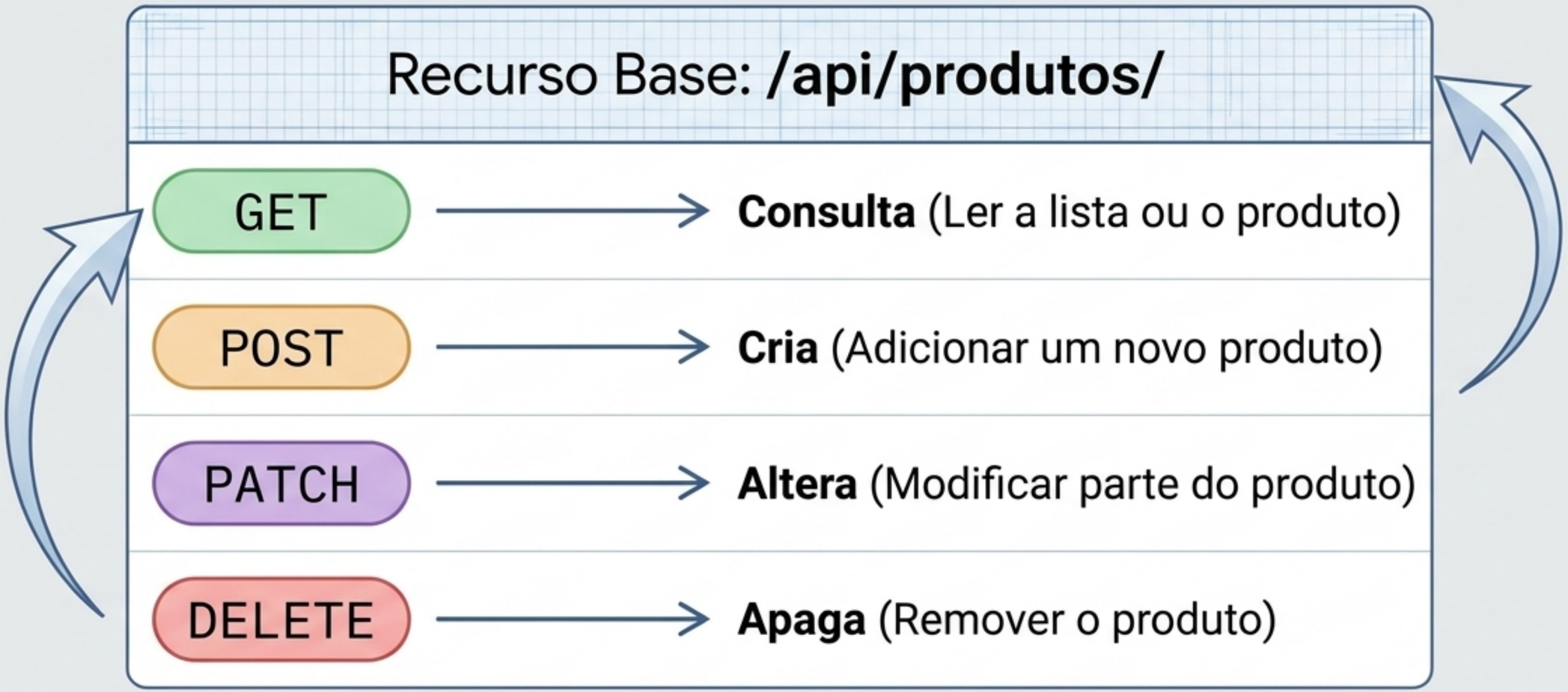


Nota do Professor: Se amanhã você trocar o interior da cozinha ou o banco de dados, o cardápio continua o mesmo e os apps não quebram.

Gemini Notebook

O Vocabulário Padrão (Verbos e Rotas)

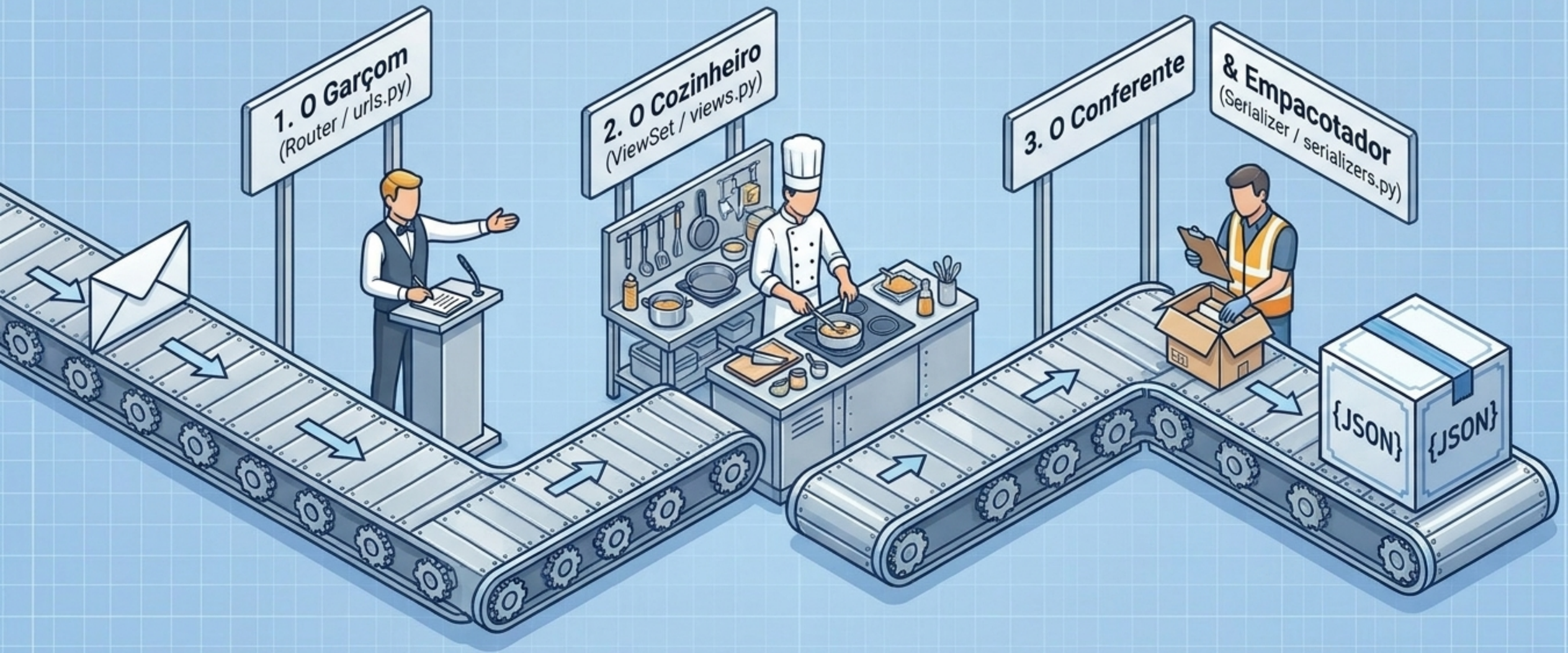
No padrão REST, usamos endereços no plural (substantivos) para as prateleiras e verbos HTTP para a nossa intenção.



Gemini Notebook

A Linha de Montagem do DRF

Para o nosso delivery funcionar perfeitamente, preciamos contratar e configurar três funcionários.



Gemini Notebook

O Garçom: Router (urls.py)

Ele atende na porta. Ao ouvir o endereço desejado (/produtos/) e o verbo da ação (GET ou POST), ele sabe exatamente para qual cozinheiro direcionar o pedido.

```
router = DefaultRouter()
router.register('produtos', ProdutoViewSet)
```



Professor Note: O método include(router.urls) entrega toda a conversa para este garçom gerenciar automaticamente.

Gemini Notebook

O Cozinheiro: ViewSet (views.py)

O ModelViewSet é um chef de cozinha altamente experiente. Ele já sabe listar, criar, detalhar, editar e apagar. Você só precisa dizer a ele qual prateleira usar e quem é o conferente.

Qual prateleira do banco usar.

```python
class ProdutoViewSet(viewsets.ModelViewSet):
    queryset = Produto.objects.all()
    serializer_class = ProdutoSerializer
```

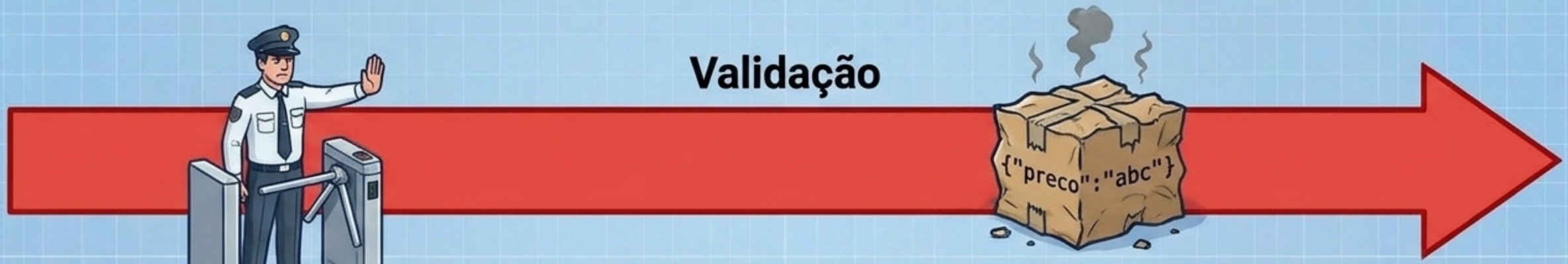
Quem vai embalar/conferir os dados.

Magia do DRF: 4 linhas de código = 5 operações completas na API.

Gemini Notebook

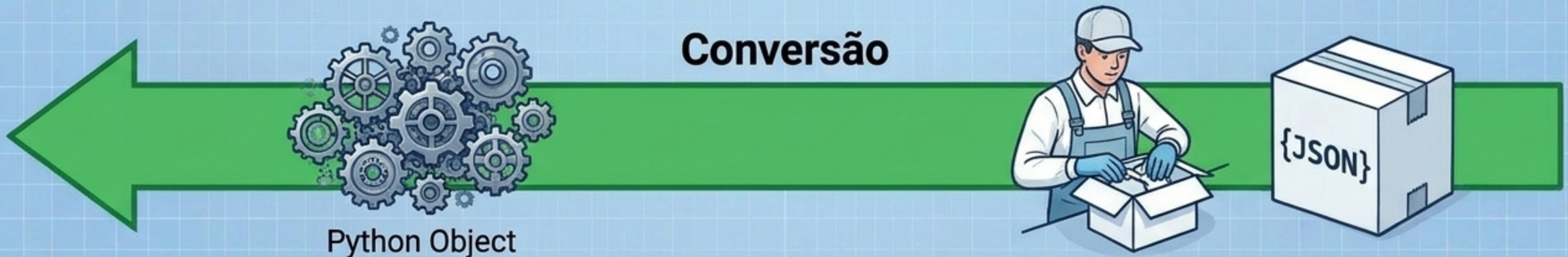
O Conferente & Empacotador: Serializer (serializers.py)

A peça central da segurança da sua API. Como ele lê o model do banco de dados, ele já sabe os limites (ex: nome máximo 120 caracteres). A validação sai de graça.




```
fields = ['id', 'nome', 'preco', 'estoque']
```

O que não está no **fields**, não vaza na saída.



Gemini Notebook

A Matriz de Status (De quem é a culpa?)

A API não fala português, ela fala através de números (Códigos HTTP). O primeiro número revela imediatamente de quem foi a culpa quando algo acontece.

Verde (Deu certo)	**200 (OK):** GET listar/detalhar	**201 (Criado):** POST sucesso
Amarelo (Culpa do Cliente / Pedido Ruim)	**400 (Bad Request):** Dado malformado	**404 (Not Found):** Endereço não existe
	405 (Method Not Allowed): Verbo proibido	**409 (Conflict):** Conflito de estado
Vermelho (Culpa do Servidor)	**500 (Internal Error):** Bug no seu código	

Professor Note: O servidor nunca grava dado ruim.
Se a culpa é do cliente (4xx), a API rejeita e explica o porquê.

Gemini Notebook

Dissecando a Rejeição (400 vs 409)

Por que o aplicativo (frontend) precisa saber a diferença exata do erro?

O 400 (Bad Request)



O 409 (Conflict)



Gemini Notebook

Defesa Ativa (Campos Somente Leitura)

O cliente não dita as regras do sistema. Ele pode ver o estoque, mas nunca alterar o estoque enviando um número diretamente em um POST. O servidor é quem decide.

```
read_only_fields = ['estoque', 'status']
```



Gemini Notebook

Saindo do Básico (A Ação de Comprar)

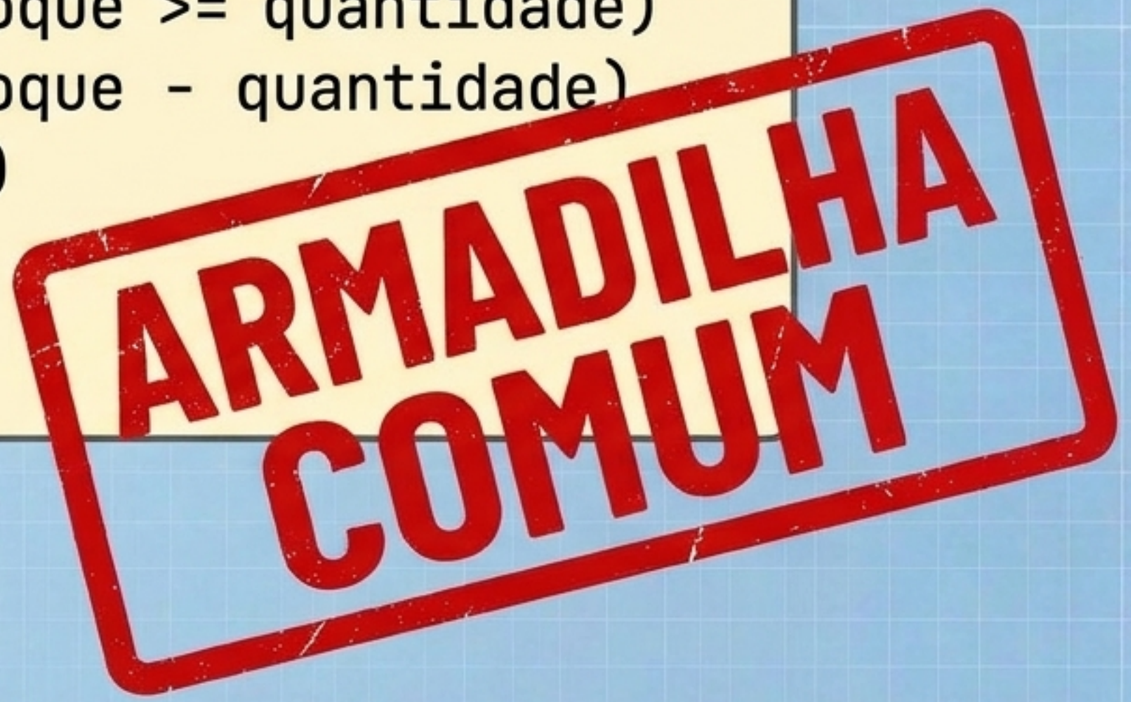
CRUD (Criar, Ler, Atualizar, Apagar) é apenas guardar e mostrar dados. O valor real de um sistema está nas regras de negócio. Quando o cliente compra, o estoque tem que baixar. Qual é a forma segura de programar isso?

O Jeito Óbvio (Pseudocódigo)

```
1. Olhar o estoque (if estoque >= quantidade)
2. Subtrair (estoque = estoque - quantidade)
3. Gravar no banco (save())
```

Professor Note

Isso parece perfeitamente certo, mas vai destruir o seu sistema no mundo real. Vejamos o porquê.



Gemini Notebook

A Armadilha da Concorrência (Condição de Corrida)

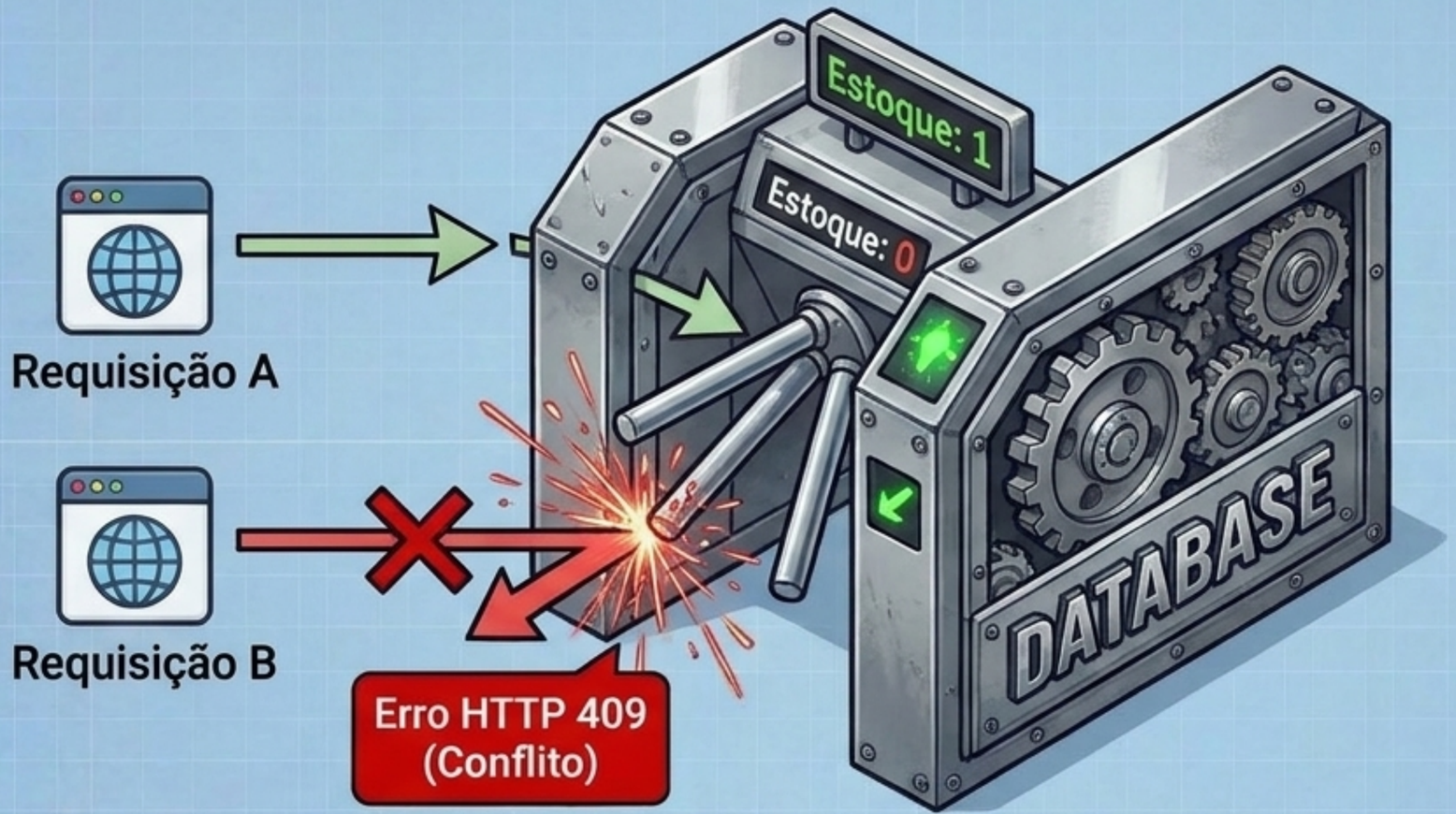
O erro do “jeito óbvio” não está em nenhum passo isolado.
O erro fatal está no intervalo de tempo entre olhar e mexer.



Resultado: **Duas vendas confirmadas. Apenas uma garrafa.**

Gemini Notebook

```
transaction.atomic() e UPDATE... WHERE estoque >= quantidade
```



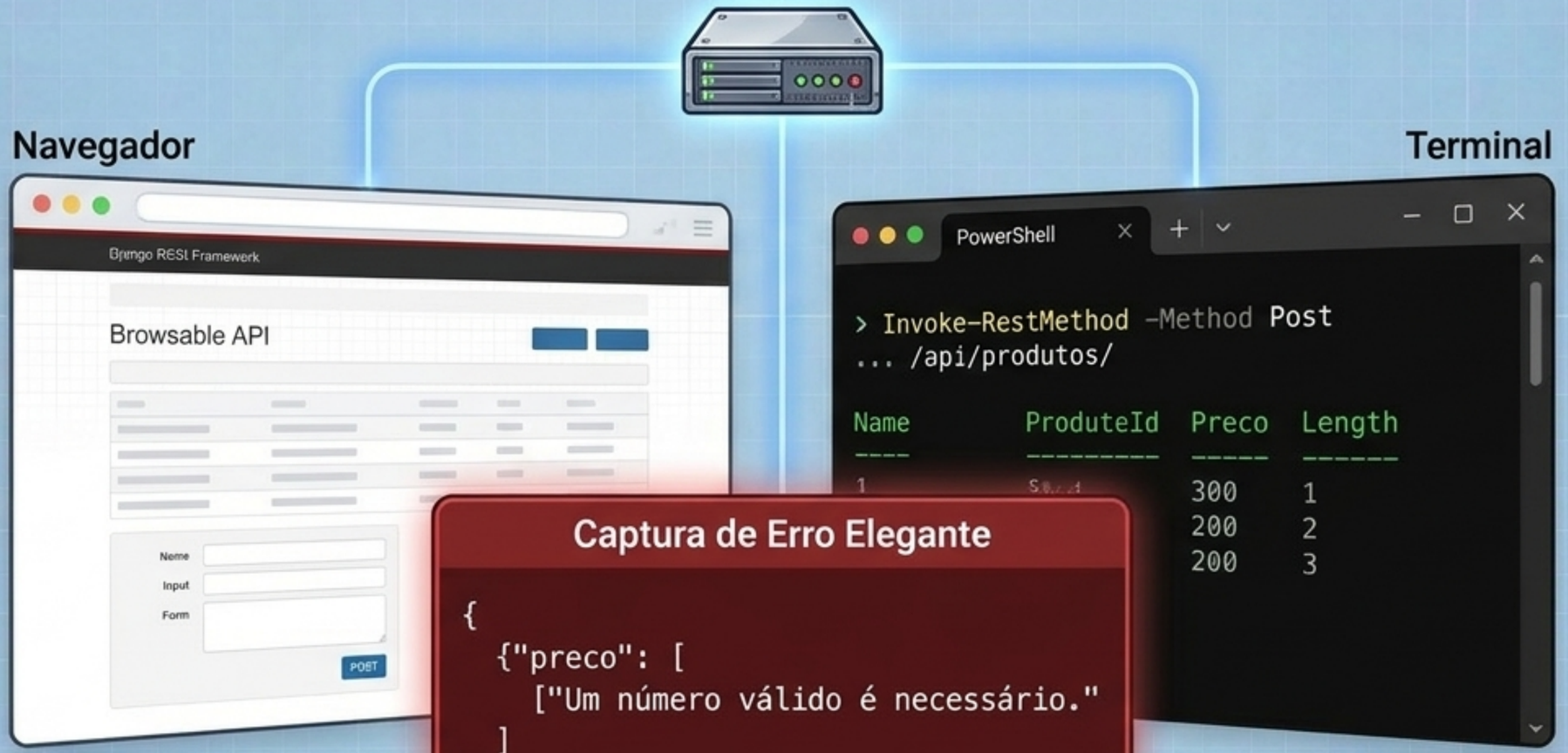
A Solução Atômica (O Banco como Árbitro)

A única solução segura é fazer a verificação e a subtração no mesmo comando, dentro do banco de dados, fechando a porta para a concorrência. É tudo ou nada.

Gemini Notebook

O Contrato em Ação (Testando a API)

Sua API tem múltiplas portas de entrada. Você pode explorá-la visualmente ou automatizar pedidos via código, a resposta subjacente será sempre o mesmo contrato JSON.



Captura de Erro Elegante

```
{
  {"preco": [
    ["Um número válido é necessário."
  ]
}
```

Gemini Notebook

Regras de Ouro da Cozinha de Dados

Domine estes três princípios e você não escreverá apenas código, mas sim engenharia de software real.

1. **A API é um contrato que protege o banco**. Quem consome não precisa (e não deve) conhecer a cozinha.

2. **O Serializer** é o seu **melhor segurança**. Nunca confie em dados do cliente. Valide na entrada e oculte detalhes internos na saída.

3. **Concorrência** exige **soluções atômicas**. Se houver regras financeiras ou de estoque, verifique e mexa no mesmo comando dentro do banco.



Próxima etapa: Provar que tudo isso funciona para sempre com Testes Automáticos.

Gemini Notebook