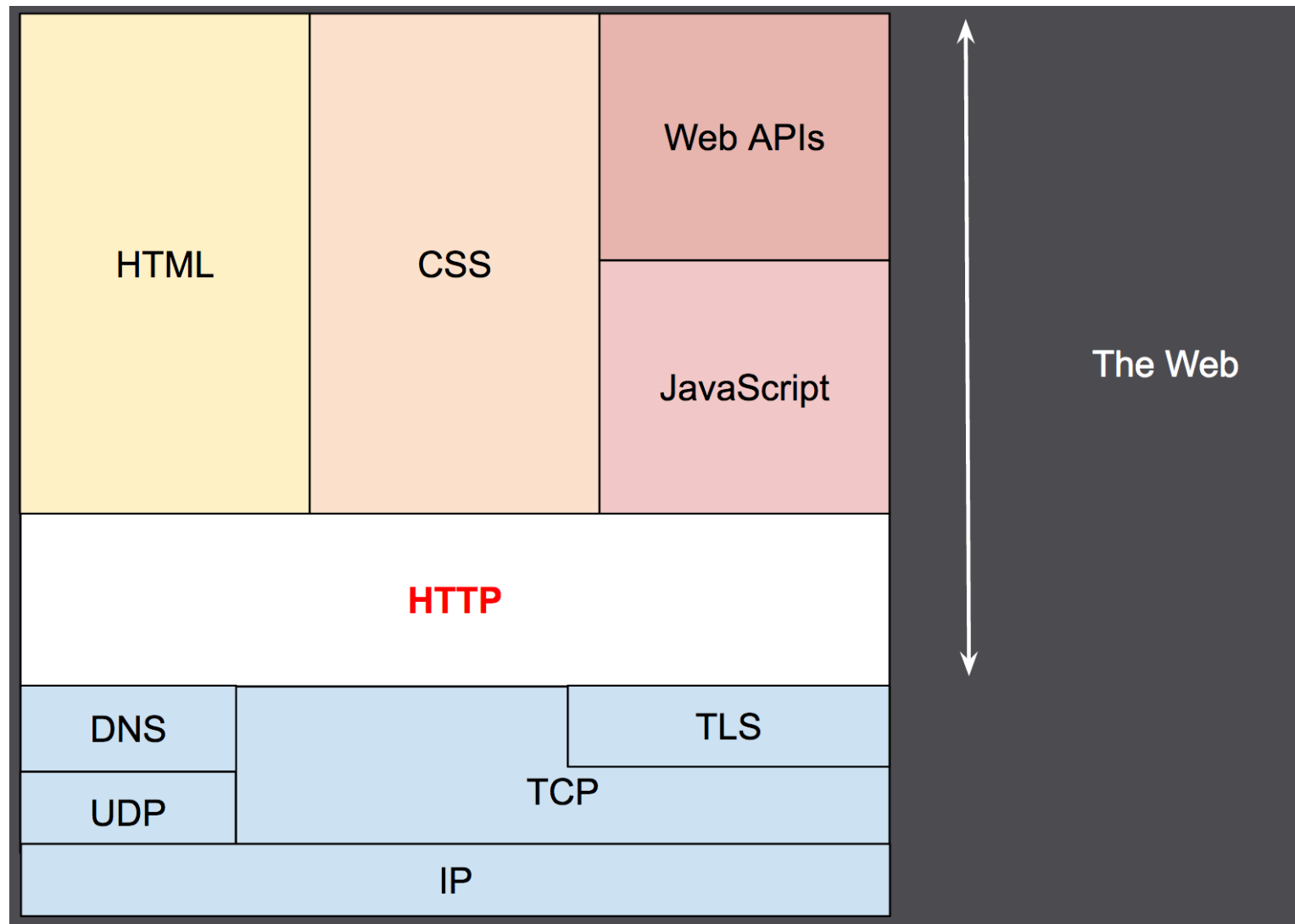


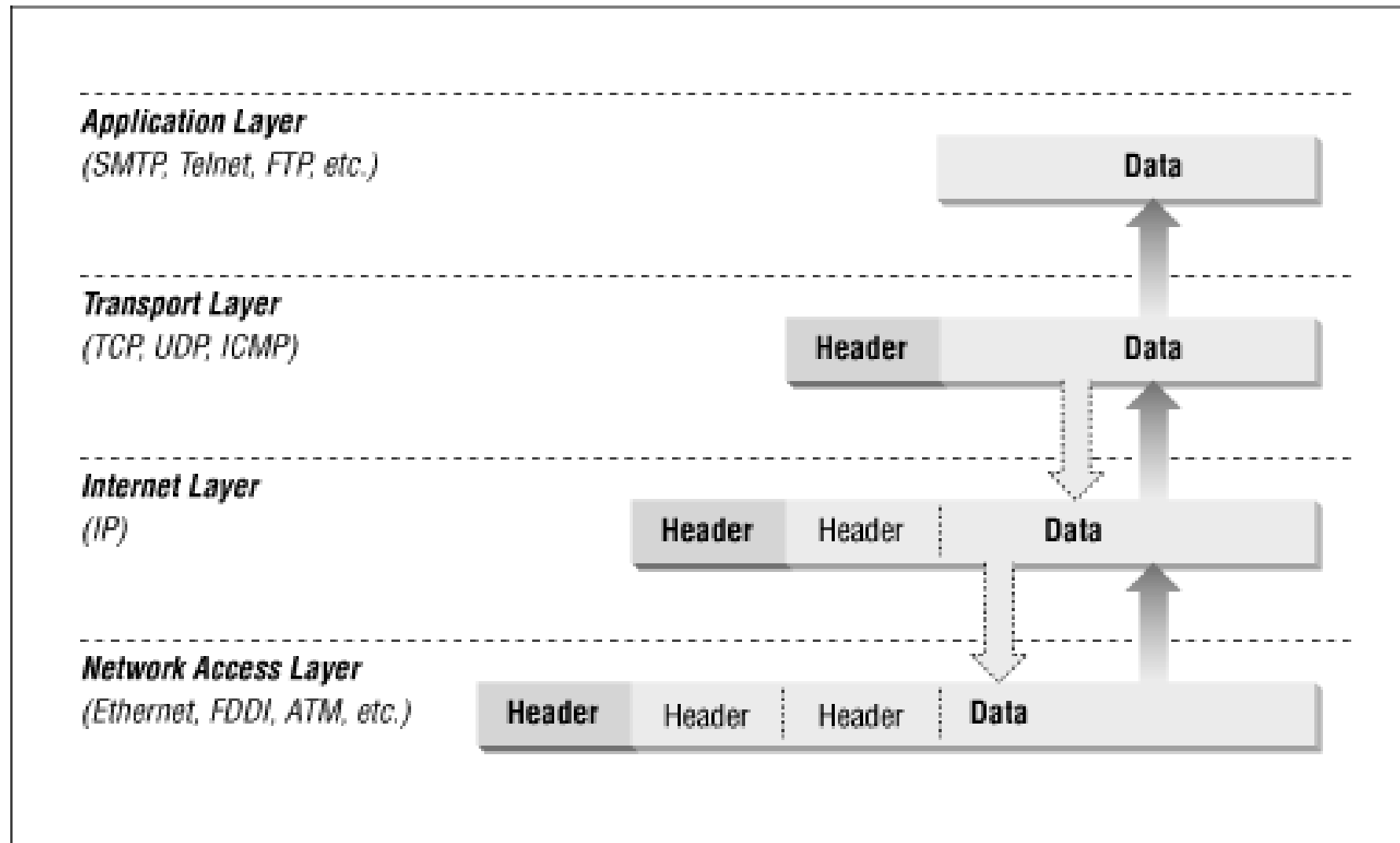
TÓPICO 11 - PROTOCOLO HTTP

Disciplina de Backend - Professor Ramon Venson - SATC 2026.2

Protocolos

- TCP e IP , responsáveis pelo endereçamento e transmissão de pacotes;
- DNS , responsável pela tradução de endereços para nomes de domínio;
- HTTP e HTTPS , responsáveis pelo tráfego de mensagens entre aplicações;





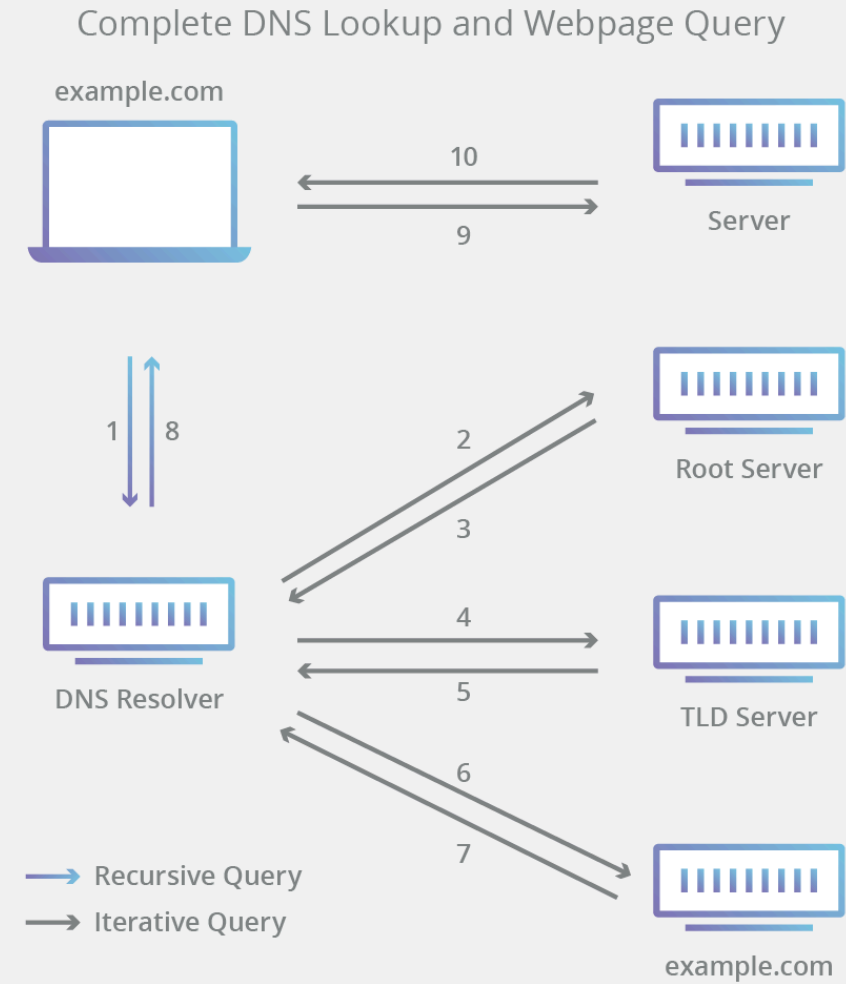
TCP e IP

Os protocolos **TCP** e **IP** são geralmente utilizados como base para todas as trocas de dados em web services

- **TCP**: Garante integridade, ordem e entrega final dos pacotes
- **IP**: roteamento dos pacotes através do endereço IP

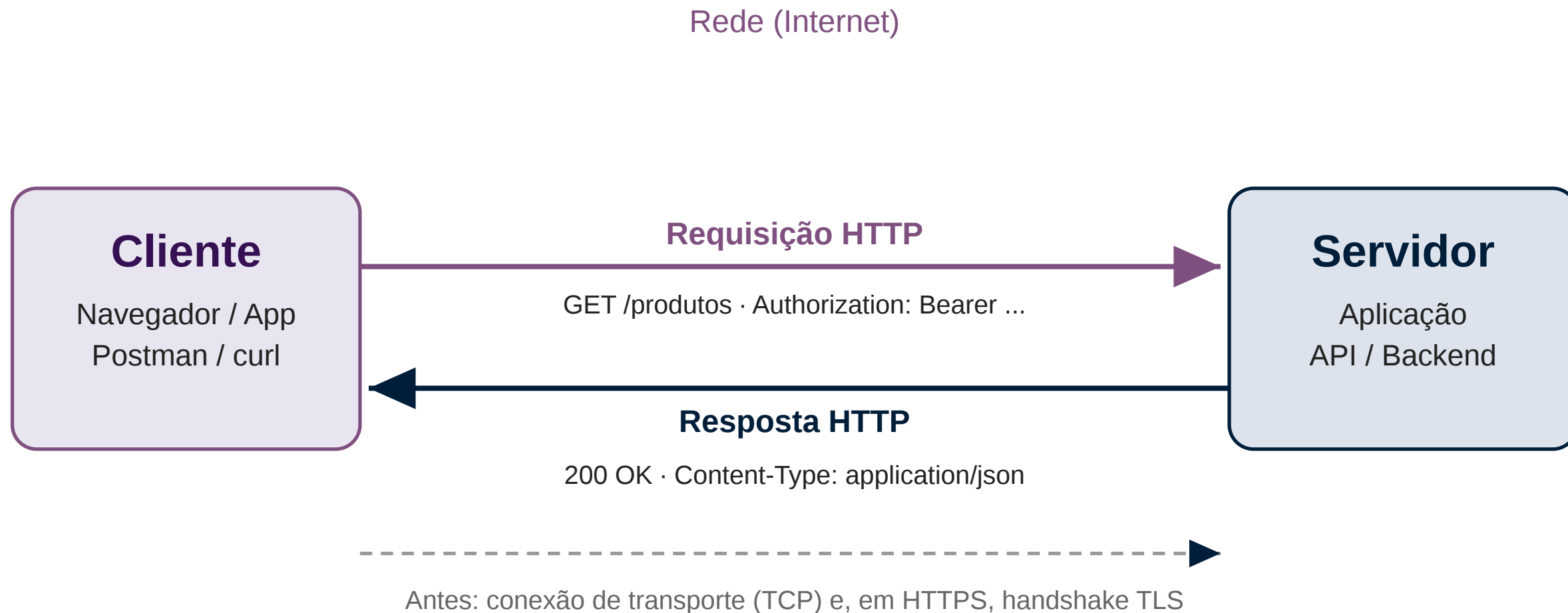
DNS

Domain Name System,
responsável pela tradução
dos nomes de domínio
(ex.: exemplo.com.br) para
um endereço IP (ex.:
20.30.2.1)



Cliente e Servidor

- O **cliente** inicia a comunicação (navegador, app mobile, Postman, `curl`);
- O **servidor** processa a requisição e devolve uma resposta;
- A conversa segue o modelo **requisição/resposta**;
- Antes da mensagem HTTP há uma conexão de transporte (TCP) e, em HTTPS, um handshake TLS.

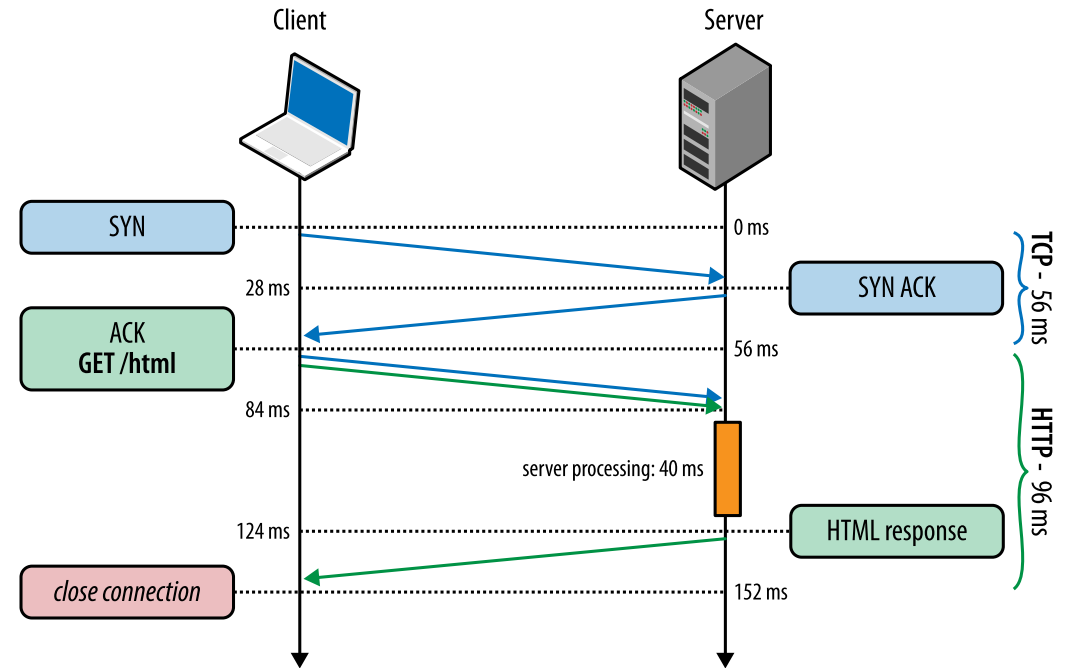


Estabelecendo uma conexão

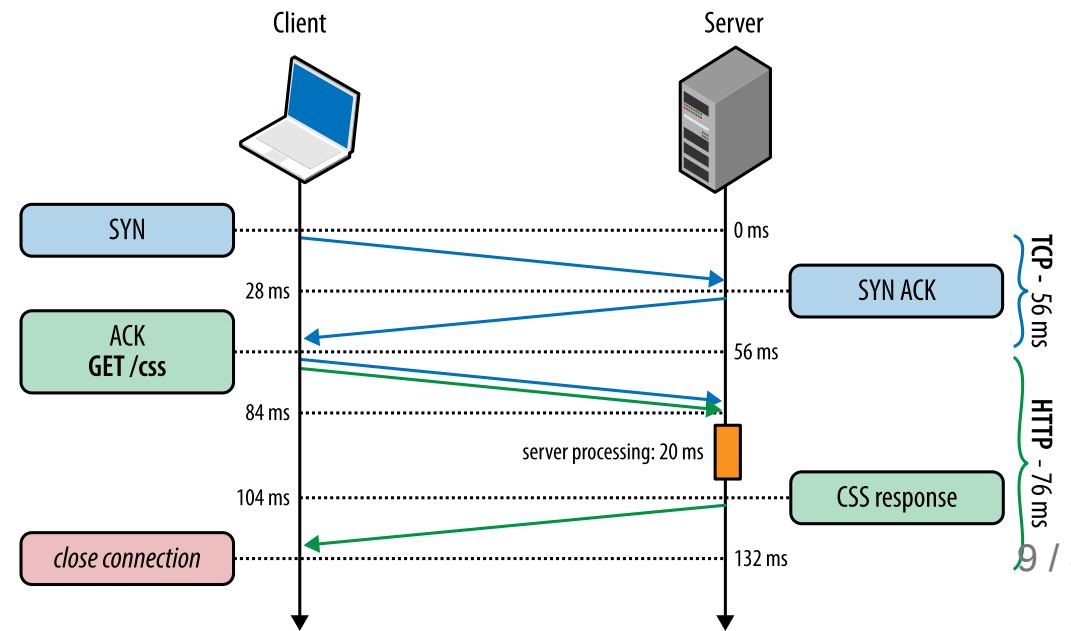
Sempre que uma requisição HTTP é feita, é preciso que as camadas de rede/transporte realizem a conexão com o servidor antes de realizar qualquer requisição.

Conexões do tipo **Keep-Alive** podem ser utilizadas para múltiplas requisições.

TCP connection #1, Request #1: HTML request

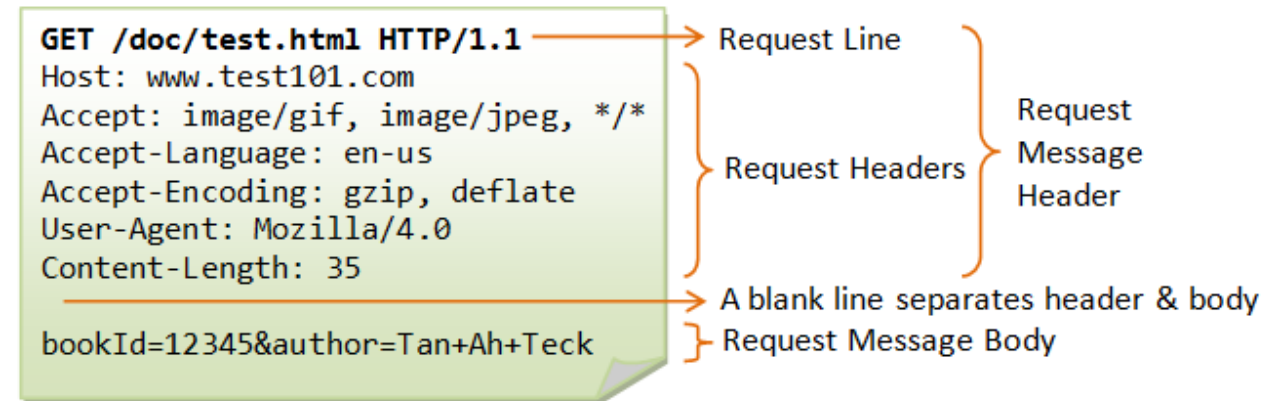


TCP connection #2, Request #2: CSS request



Protocolo HTTP

- Camada de Aplicação
- Requisição/Resposta
- Protocolo sem estado (stateless)
- Porta 80



Exemplo de Requisição HTTP

- **Linha inicial:** método + caminho + versão;
- **Cabeçalhos:** metadados como `Host` e `Content-Type` ;
- **Corpo:** dados enviados (aqui, um JSON).

A conexão acontece nas camadas de rede/transporte, mas em `HTTP/1.1` o `Host` continua sendo importante.

Requisição HTTP

POST `/login HTTP/1.1`

Host: `example.com`

Content-Type: `application/json`

Content-Length: `39`

Accept: `application/json`

```
{"user": "teste",  
  "password": "123456"}
```

Resposta HTTP

HTTP/1.1 200 OK

Content-Type: application/json

Content-Length: 27

Date: Fri, 06 Aug 2030 ...

Cache-Control: no-cache

```
{"token": "abc123",  
  "role": "admin"}
```

Exemplo de Resposta HTTP

- **Linha de status:** versão + código (200 OK);
- **Cabeçalhos:** tipo do corpo, cache, cookies...;
- **Corpo:** recurso devolvido ao cliente.

Métodos HTTP

Método	Descrição	Corpo na requisição	Seguro	Idempotente
GET	Recupera um recurso	Não	Sim	Sim
POST	Cria / processa	Sim	Não	Não
PUT	Substitui um recurso	Sim	Não	Sim
PATCH	Atualiza parcialmente	Sim	Não	Não
DELETE	Remove um recurso	Não	Não	Sim
HEAD	Apenas cabeçalhos	Não	Sim	Sim
OPTIONS	Capacidades do endpoint	Não	Sim	Sim

Métodos seguros e idempotentes

- Métodos **seguros** não devem alterar o estado do servidor (GET , HEAD , OPTIONS)
- Métodos **idempotentes** produzem o mesmo efeito quando repetidos (GET , PUT , DELETE)
- POST normalmente **não** é idempotente

Códigos de Status HTTP

Os códigos indicam o resultado da operação e são divididos em faixas:

- 1xx informativos
- 2xx sucesso (200 OK , 201 Created , 204 No Content)
- 3xx redirecionamento (301 , 304 Not Modified)
- 4xx erro do cliente (400 , 401 , 403 , 404 , 429)
- 5xx erro do servidor (500 , 502 , 503)

Referência: [MDN](#) ou [HTTP Cat](#)

401 x 403

- 401 Unauthorized : cliente não se autenticou corretamente
- 403 Forbidden : cliente está autenticado, mas não tem permissão

Exemplos:

- token inválido ou ausente -> 401
- usuário comum tentando acessar rota administrativa -> 403

Cabeçalhos Comuns

Cabeçalho	Descrição	Exemplo
Host	Nome do servidor	example.com
Content-Length	Tamanho do corpo da mensagem em bytes	73
Content-Type	Tipo do corpo da mensagem	text/html
User-Agent	Identificação do cliente	Mozilla/5.0 (Windows NT 10.0; Win64; x64
Authorization	Credenciais de autenticação	Bearer <token>

Cabeçalho	Descrição	Exemplo
Accept	Tipos de conteúdo aceitos pelo cliente	text/html
Cookie	Dados de sessão do cliente	sessionId=abc123; userId=1
Connection	Controle de conexão	keep-alive ou close
Cache-Control	Controle de cache	no-cache, no-store, must-revalidate
Last-Modified	Data e hora da última modificação do recurso	Wed, 21 Oct 2015 07:28:00 GMT
DNT	Preferência de não rastreamento do usuário	1 (não rastrear) ou 0 (permitir rastrear)

Tipos de Conteúdo

O formato do corpo da mensagem geralmente é identificado usando um cabeçalho chamado `Content-Type`. Exemplos:

- Texto plano genérico (`text/plain`)
- Áudio (`audio/mpeg`), Vídeo (`video/mp4`) e Imagem (`image/png`)
- Binário (`application/octet-stream`)
- HTML (`text/html`)
- Formulário (`multipart/form-data`)
- JSON (`application/json`)

Content-Type x Accept

O `Content-Type` indica o tipo de dado que está sendo enviado na requisição ou resposta, enquanto o `Accept` indica os tipos de dados que o cliente está disposto a receber.

Negociação de conteúdo: Content-Type x Accept

Content-Type

Formato do corpo que ESTOU enviando

POST /produtos

Content-Type: application/json

`{"nome": "Mouse", "preco": 89.9}`

Accept

Formatos que ACEITO receber

GET /produtos

Accept: application/json

Accept: application/xml;q=0.9

O servidor devolve o corpo no formato negociado e indica isso no cabeçalho

Content-Type: application/json

o mesmo recurso pode ter múltiplas representações (HTML, JSON, XML, CSV...)

Cookies e Sessão

- HTTP é **stateless**: cada requisição é independente;
- Cookies permitem reconstruir contexto entre requisições;
- O servidor envia `Set-Cookie` e o cliente reenvia em `Cookie` ;
- Sessões, tokens (ex.: JWT) e estado no servidor cumprem papel semelhante.

HTTP é stateless: cada requisição é independente



Cache HTTP

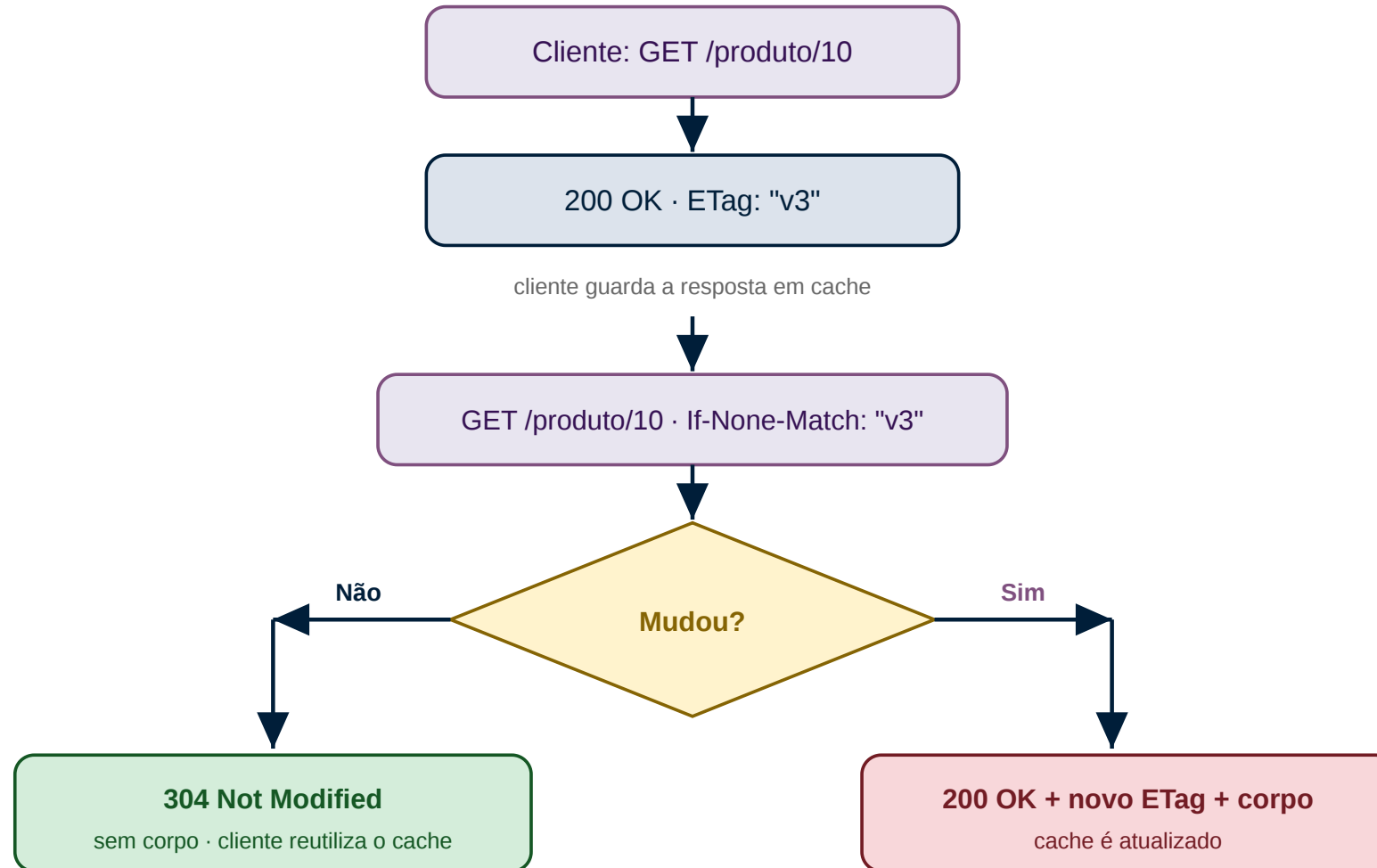
Cabeçalhos de cache ajudam a evitar transferências desnecessárias.

- `Cache-Control` : define políticas de cache
- `ETag` / `Last-Modified` : identificam a versão do recurso
- `If-None-Match` / `If-Modified-Since` : validação condicional pelo cliente

Exemplo:

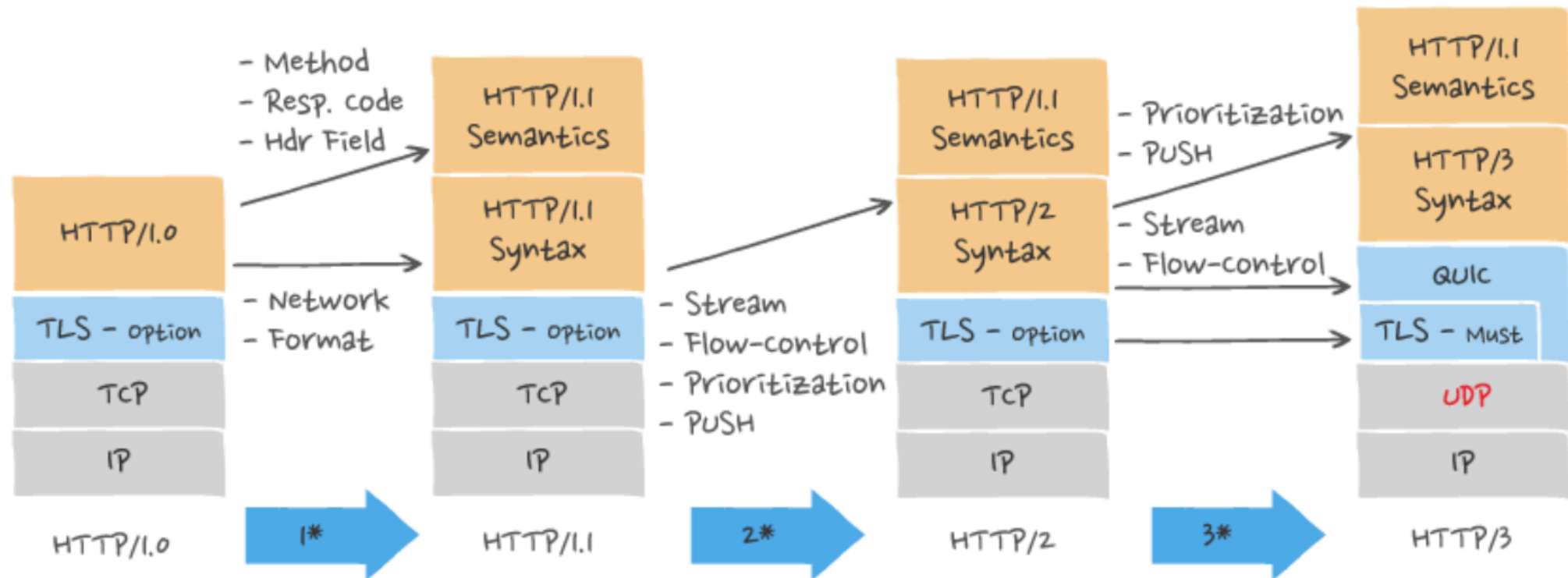
```
HTTP/1.1 200 OK
Cache-Control: max-age=3600
ETag: "produto-10-v3"
```

Cache HTTP com ETag e requisição condicional



Comparação entre versões HTTP

HTTP protocol stack transition and comparison



Evolução do HTTP

HTTP/1.1

1997

- Baseado em texto
- Sobre TCP
- Conexões keep-alive
- Métodos e códigos amplos
- Head-of-line blocking

HTTP/2

2015

- Protocolo binário
- Sobre TCP
- Multiplexação de streams
- Compressão de cabeçalhos
- Sujeito a bloqueio de TCP

HTTP/3

2018

- Sobre UDP + QUIC
- Menor latência
- Sem head-of-line blocking
- TLS 1.3 obrigatório
- Connection ID

- 1.0 : 1996
 - Pouco/Nenhum suporte a compressão
 - Apenas GET , HEAD e POST
- 1.1 (mais utilizado): 1997
 - Novos Métodos e Códigos de Resposta
 - Keep-Alive
 - Compressão aprimorada do corpo da mensagem

- 2.0 : 2015
 - Introduz protocolo binário ao invés de texto
 - Compressão do cabeçalho
 - Resposta e Requisições Multiplexada na camada de aplicação
 - Permite "enfileirar" requisições e respostas
 - Pacotes perdidos na camada de rede/transporte ainda bloqueiam as requisições
 - Server Push existiu na especificação, mas teve adoção prática limitada

- 3.0 : 2018
 - Troca do TCP+TLS para UDP+QUIC
 - Multiplexação não bloqueante na camada de transporte
 - Pacotes perdidos na camada de rede/transporte não bloqueiam outros pacotes
 - Criptografia obrigatória (TLS 1.3)
 - Conexão identificada por `Connection ID` ao invés de apenas endereço IP e porta
 - Menor latência

HTTPS

- Extensão do protocolo HTTP
- Utiliza uma camada de segurança chamada *Transport Layer Security* (TLS)
- Negociação da criptografia usando chaves públicas
- Iniciada durante a fase de conexão
- Porta 443

O TLS é uma versão mais recente do *Secure Sockets Layer* (SSL), termo ainda usado para descrever a mesma camada.

HTTPS: handshake TLS antes dos dados HTTP



HTTP no Spring Web

O Spring Web abstrai a complexidade do protocolo HTTP, permitindo que você se concentre na lógica de negócio. Para isso, podemos usar anotações como `@RestController` , `@RequestMapping` , `@GetMapping` , `@PostMapping` , entre outras.

Exemplo de Controller

```
@RestController
@RequestMapping("/produtos")
public class ProdutoController {

    @GetMapping("/{id}")
    public ResponseEntity<Produto> getProduto(@PathVariable UUID id) {
        Produto produto = produtoRepository.findById(id).get();
        return ResponseEntity.ok(produto);
    }

    @PostMapping
    public ResponseEntity<Produto> createProduto(@RequestBody Produto produto) {
        Produto createdProduto = produtoRepository.save(produto);
        return ResponseEntity.status(HttpStatus.CREATED).body(createdProduto);
    }
}
```

Tipos de Mapeamento

Anotação	Método HTTP	Descrição
@GetMapping	GET	Recupera recursos
@PostMapping	POST	Cria recursos
@PutMapping	PUT	Substitui recursos
@PatchMapping	PATCH	Atualiza parcialmente recursos
@DeleteMapping	DELETE	Remove recursos
@RequestMapping	Qualquer	Mapeamento genérico, pode ser usado com qualquer método HTTP

Cabeçalhos e Parâmetros

Anotação	Descrição
@RequestHeader	Acessa valores de cabeçalhos HTTP
@RequestParam	Acessa parâmetros de query string
@PathVariable	Acessa valores de segmentos da URL
@RequestBody	Desserializa o corpo da requisição em um objeto Java
@CookieValue	Acessa valores de cookies enviados pelo cliente

Códigos de Estado

Podemos controlar o código de estado retornado por um método mapeado usando a anotação `@ResponseStatus` e o enum `HttpStatus` :

```
@ResponseStatus(HttpStatus.CREATED)
@PostMapping
public Produto createProduto(@RequestBody Produto produto) {
    return produtoRepository.save(produto);
}
```

Retorno padrão x `ResponseEntity`

Por padrão, retornamos o objeto diretamente: o Spring responde `200 OK` com o corpo serializado.

```
@GetMapping("/{id}")  
public Produto getProduto(@PathVariable UUID id) {  
    return produtoRepository.findById(id).get();  
}
```

Quando precisamos controlar **status**, **cabeçalhos** ou **corpo**, usamos

`ResponseEntity` :

```
@GetMapping("/{id}")
public ResponseEntity<Produto> getProduto(@PathVariable UUID id) {
    Produto produto = produtoRepository.findById(id).orElse(null);
    if (produto == null) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
    }
    return ResponseEntity.ok(produto);
}
```

Use `ResponseEntity` apenas quando precisar desse controle extra; caso contrário, retornar o objeto é mais simples.

Implementando Cookies no Spring

```
@GetMapping("/{id}/cookie")
public ResponseEntity<Produto> comCookie(@PathVariable UUID id) {
    ResponseCookie cookie = ResponseCookie.from("ultimoProduto", id.toString())
        .httpOnly(true).path("/").maxAge(Duration.ofHours(1)).build();
    return ResponseEntity.ok()
        .header(HttpHeaders.SET_COOKIE, cookie.toString())
        .body(produtRepository.findById(id).get());
}
```

- `ResponseCookie` monta o valor do cabeçalho `Set-Cookie` ;
- o cliente (Postman/navegador) armazena o cookie e o reenvia em `Cookie` nas próximas requisições.

Implementando Cache com ETag no Spring

```
@GetMapping("/{id}/cache")
public ResponseEntity<Produto> comCache(@PathVariable UUID id,
    @RequestHeader(value = "If-None-Match", required = false) String ifNoneMatch) {
    String etag = "\"produto-" + id + "-v1\"";
    if (etag.equals(ifNoneMatch)) {
        return ResponseEntity.status(HttpStatus.NOT_MODIFIED).eTag(etag).build();
    }
    return ResponseEntity.ok().eTag(etag)
        .cacheControl(CacheControl.maxAge(Duration.ofSeconds(60)))
        .body(produtRepository.findById(id).get());
}
```

- 1ª chamada → 200 com ETag e Cache-Control ;
- 2ª chamada com If-None-Match igual ao ETag → 304 Not Modified (sem corpo).

Testando APIs na prática

Ferramentas como **Postman** e **Insomnia** permitem montar e inspecionar requisições HTTP:

- escolher o método (`GET` , `POST` , ...) e a URL;
- definir cabeçalhos (`Accept` , `Content-Type` , `Authorization`);
- enviar corpo em JSON/XML e visualizar a resposta;
- observar **status code**, tempos e cookies;
- salvar coleções de requisições e variáveis de ambiente.

Também é possível usar o `curl` no terminal e a aba *Network* do navegador.

O que aprendemos nessa aula?

- O modelo cliente/servidor e o ciclo requisição/resposta;
- A estrutura de uma mensagem HTTP (linha inicial, cabeçalhos e corpo);
- Métodos HTTP, códigos de status, cabeçalhos, cookies e cache;
- As versões do HTTP e o papel do HTTPS/TLS;
- Como inspecionar requisições com Postman/Insomnia.