

Apostila — Material de apoio

# A Comunicação na Web: HTTP e o Modelo Cliente-Servidor

Desenvolvimento Front-End I

**Período:** Semana 2

**Autor:** Cleziel Franzoni da Costa

# Sumário

<b>1</b>	<b>A Comunicação na Web — HTTP e o Modelo Cliente-Servidor</b>	<b>1</b>
1.1	O Modelo Cliente-Servidor . . . . .	1
1.2	A Anatomia de uma Requisição HTTP (o Pedido) . . . . .	2
1.3	A Anatomia de uma Resposta HTTP (a Entrega) . . . . .	3
1.4	Estudo de Caso: Por que o Front-End se Importa com HTTP? . . . . .	4

# Capítulo 1

## A Comunicação na Web — HTTP e o Modelo Cliente-Servidor

### Introdução

Na semana anterior, exploramos a origem da *World Wide Web*. Agora, vamos abrir o *capô* para entender como ela realmente funciona. Toda vez que você clica em um link, assiste a um vídeo ou envia um formulário, uma conversa invisível e ultrarrápida acontece entre o seu dispositivo e um servidor remoto. Essa conversa segue um conjunto de regras estritas, um idioma universal chamado **HTTP** (*HyperText Transfer Protocol*).

Dominar esse idioma é uma habilidade fundamental para um desenvolvedor front-end, pois é a chave para **diagnosticar problemas**, **otimizar desempenho** e **criar aplicações dinâmicas** que vão além de simples páginas estáticas.

### 1.1 O Modelo Cliente-Servidor

Toda a interação na Web é baseada em um modelo simples, mas poderoso: o **modelo cliente-servidor**.

#### Cliente (Client)

É o software ou dispositivo que inicia a comunicação e solicita um serviço ou recurso. No nosso contexto, o cliente é quase sempre o **navegador web** (Google Chrome, Mozilla Firefox, etc.).

#### Servidor (Server)

É um computador (ou um conjunto de computadores) potente e sempre conectado, cujo trabalho é armazenar recursos (páginas HTML, imagens, dados) e *escutar* por solicitações dos clientes. Ao receber uma solicitação, o servidor a processa e envia uma resposta de volta.

## Diagrama de Fluxo

```

+-----+ +-----+
| | 1. Requisição HTTP | |
| Cliente | -----> | Servidor |
| (Navegador) | (Pedido de página) | (Hospedagem) |
| | 2. Resposta HTTP | |
| | <----- | |
+-----+ (Código da página) +-----+

```

A beleza desse modelo está na clareza de papéis: **o cliente sempre pede e o servidor sempre responde**.

## 1.2 A Anatomia de uma Requisição HTTP (o Pedido)

Quando o seu navegador precisa de um recurso, ele monta uma *mensagem de texto* e a envia para o servidor. Essa mensagem é a **requisição HTTP**.

### Exemplo de uma requisição GET real

```

GET /cursos/tads/index.html HTTP/1.1
Host: www.ifes.edu.br
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
, like Gecko) Chrome/108.0.0.0 Safari/537.36
Accept-Language: pt-BR,pt;q=0.9

```

## Partes principais

### Linha de início

- **GET: método HTTP**, a intenção do pedido. GET significa “me dê este recurso”.
- **/cursos/tads/index.html: caminho do recurso** no servidor.
- **HTTP/1.1: versão do protocolo**.

### Cabeçalhos (Headers)

- **Host: www.ifes.edu.br** — para qual servidor o pedido é destinado.
- **User-Agent: ...** — informações sobre o navegador e sistema operacional que fazem o pedido.

### Métodos HTTP principais

**GET** Solicita a representação de um recurso específico (buscar dados).

**POST** Envia dados para o servidor para criar um novo recurso (ex.: enviar formulário de cadastro). Os dados vão no *corpo* da requisição.

**PUT** Atualiza um recurso existente com os dados enviados.

**DELETE** Remove um recurso específico.

## 1.3 A Anatomia de uma Resposta HTTP (a Entrega)

Após receber e processar a requisição, o servidor monta sua própria mensagem de texto: a resposta HTTP.

### Exemplo de uma resposta HTTP real

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2025 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 155
Last-Modified: Wed, 08 Jan 2025 23:11:55 GMT
Server: Apache/2.4.1 (Unix)

<!DOCTYPE html>
<html>
<head>
  <title>Olá, Mundo!</title>
</head>
<body>
  <h1>Página de Exemplo</h1>
</body>
</html>
```

### Partes principais

#### Linha de status

- HTTP/1.1 — versão do protocolo.
- 200 OK — **código de status**. 200 indica sucesso.

#### Cabeçalhos (Headers)

- Date: data e hora da resposta.
- Content-Type: text/html — informa que o conteúdo é um documento HTML.
- Content-Length: tamanho do conteúdo em bytes.

#### Corpo (Body)

Após uma linha em branco, vem o conteúdo do recurso em si. Neste caso, o código HTML da página solicitada, que o navegador irá renderizar.

## Códigos de Status Essenciais

Os códigos são agrupados em classes. Alguns dos mais importantes:

### 2xx — Sucesso

A requisição foi recebida, entendida e aceita com sucesso.

- **200 OK** — requisição bem-sucedida.
- **201 Created** — requisição bem-sucedida e um novo recurso foi criado (comum em POST).

## 4xx — Erro do Cliente

A requisição contém sintaxe inválida ou não pode ser atendida: geralmente é um problema no pedido.

- **400 Bad Request** — o servidor não entendeu a requisição.
- **401 Unauthorized** — autenticação necessária e ausente/falhou.
- **403 Forbidden** — sem permissão para acessar o conteúdo.
- **404 Not Found** — recurso não encontrado.

## 5xx — Erro do Servidor

O servidor falhou em atender uma requisição aparentemente válida.

- **500 Internal Server Error** — erro genérico inesperado no servidor.

## 1.4 Estudo de Caso: Por que o Front-End se Importa com HTTP?

O desenvolvimento front-end moderno não é apenas criar páginas estáticas: é **criar experiências dinâmicas**.

### Cenário: Feed infinito (Instagram/Twitter)

**Ação do usuário:** abrir o aplicativo e rolar para ver posts mais antigos.

**Código front-end (JavaScript):** ao detectar a rolagem no fim da página, *não* recarrega tudo. Em vez disso, envia uma **requisição GET** para uma API (ex.: <https://api.instagram.com/>).

**Servidor (back-end):** recebe o pedido, busca os próximos itens no banco de dados e envia de volta em uma resposta. O corpo frequentemente vem em **JSON** (formato de texto para dados estruturados).

**Código front-end (continuação):** recebe 200 OK com JSON; lê autor, imagem, legenda etc.; cria elementos HTML e **insere dinamicamente** no final da página.

Sem um entendimento claro de como fazer requisições **GET** e interpretar respostas (incluindo lidar com um 500 se o servidor falhar), essa funcionalidade não seria possível.

## Conclusão

O **HTTP** é a linguagem fundamental que permite que a interface visual (front-end) se comunique com a lógica e os dados (back-end). Saber ler e interpretar essa comunicação na aba **Network** do navegador é uma das habilidades de depuração mais poderosas que você vai desenvolver.