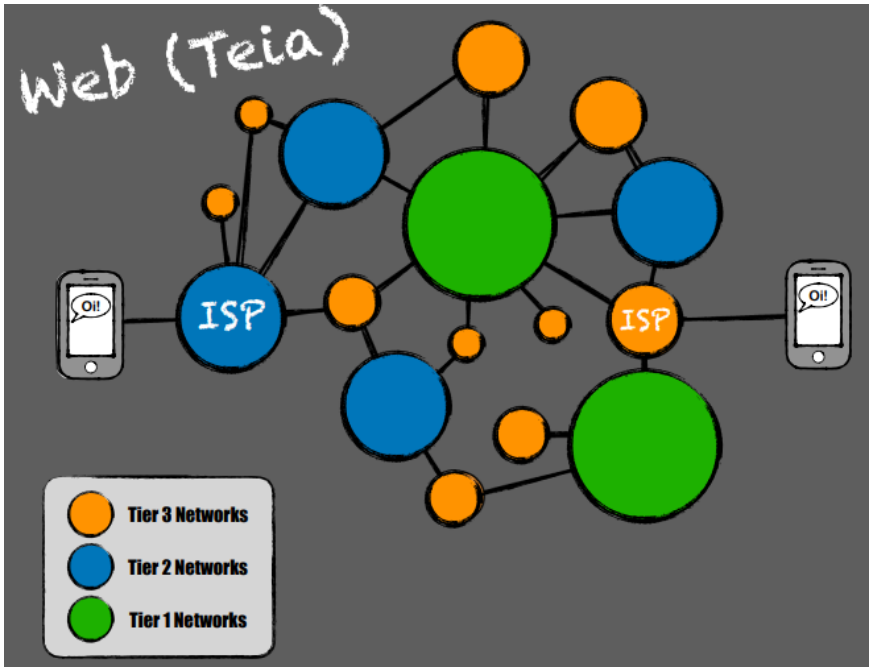
# Conceitos sobre Web

**Topologias:** Centralizada vs Distribuída (a internet é distribuída)




**TCP/IP:** Transmission Control Protocol (orientado a conexão e confiável) / Internet Protocol (roteamento entre redes)

**Camadas do TCP/IP (Padrão Layers):**

**Rede/Física** (Ethernet, WIFI, MAC adress, hardware e enlace(software)) ←**Internet** (IP, Conexão entre redes, transferência de pacotes, etc) ← **Transporte** (Comunicação maquina-maquina= TCP, UDP(melhor esforço), comunicação host-a-host, confiabilidade, integridade, etc) ← **Aplicação** (HTTPS, FTP, SMTP, comunicação processo-a-processo, numero da porta, etc)

**Anatomia do IPv4:**

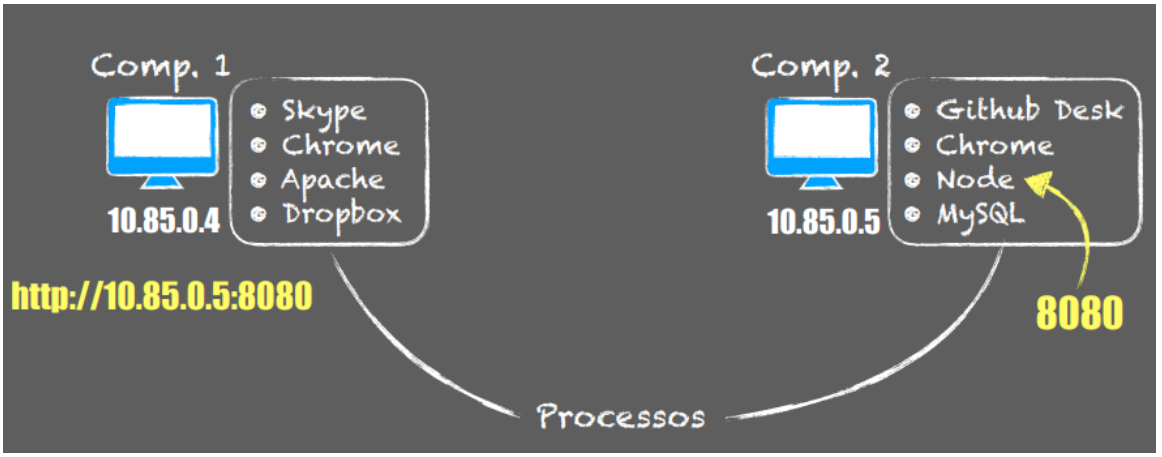
- Parte está destinada para identificar a rede a qual o computador está
- A outra parte está destinada a identificar o computador dentro de uma rede.

**Classe A** → 1 parte Rede e 3 partes Host

**Classe B** → 2 partes Rede e 2 partes Host

**Classe C** → 3 partes Rede e 1 parte Host

**Conceito de porta:** O processo para que haja uma comunicação entre dois computadores é mapeado pelas portas.




**Protocolo HTTP:** Hyper Text Transfer Protocol

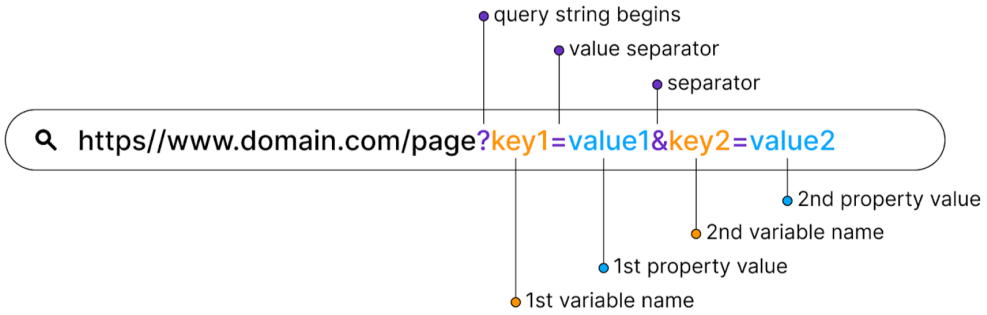
- **Características:** camada de aplicação/ cliente-servidor/ stateless (sempre no mesmo estado)/ TCPIP/ HTML, CSS, JS, Mídias...
- **Fluxo:** Usuário informa a URL → Browser gera a Requisição (composta de URL + Params) ao servidor → Servidor Web gera a Resposta (HTML, CSS, JS, imagens, etc) ao browser → Browser exibe a página ao usuário.

**Requisição via GET:** parâmetros vão diretamente dentro da URL

https://api.example.com/search?query=%s&page=%d

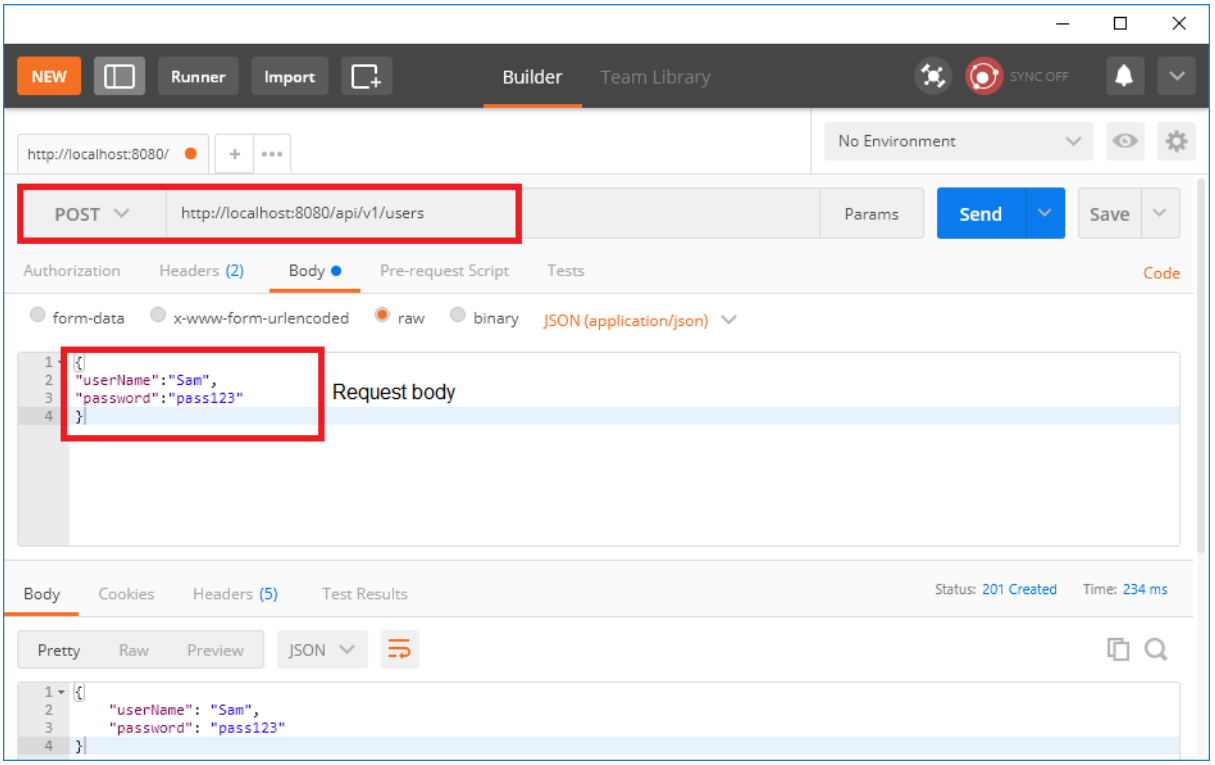
- /search é um endpoint
- se fosse /search/users/4002... → o 4002 é um path param
- **"?"** → tudo que vem depois do ? representa uma query (consulta que farei em cima do serviço search).
- **"&"** → representa a separação dos query params (parâmetros, como a pesquisa ser em inglês ou português, etc)

- os query params são compostos de uma chave e um valor (key=value)

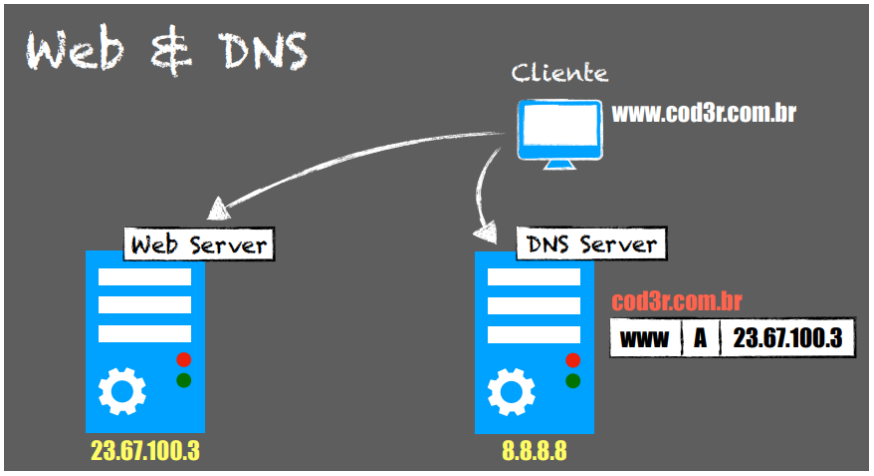

**Requisição via POST:** os parâmetros da uma requisição são enviados no corpo da requisição HTTP, e não na URL.

- POST é considerado mais seguro que GET para enviar dados sensíveis, pois os dados não ficam expostos na URL e nos logs do servidor ou do navegador. No entanto, para proteção real, é necessário usar HTTPS.




**Servidores Web:** Apache HTTP Server, JBoss, Nginx, NodeJS, Tomcat

- Uma mudança feita em um DNS deve ser replicada em todos os DNSs do mundo, por isso demora cerca de 24hrs!




- Dentro de um Servidor é muito comum ter uma estrutura de pastas de arquivos estáticos (imagens, css, javascripts, etc)
- Muitas vezes esses conteúdos estáticos são guardados em um servidor externo (um serviço especializado) pra não sobrecarregar muito seu servidor.

- Web Stack (LAMP = **L**inux **A**pache **M**ysql **P**hp)

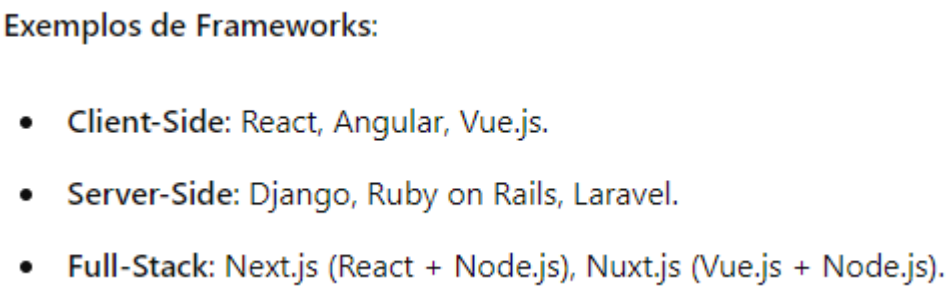

**Aplicação Client Side vs Server Side:**

- Em uma aplicação **client-side**, a maior parte do processamento é feita no navegador do usuário (cliente). A interação com o servidor é mínima, geralmente apenas para buscar ou enviar dados.
- Em uma aplicação **server-side**, a maior parte do processamento (o conteúdo) é feita no servidor. O navegador do usuário apenas exibe a interface e interage com o servidor para cada ação.

### Comparação

| Característica | Client-Side | Server-Side |
|---|---|---|
| **Execução** | Navegador do Cliente | Servidor |
| **Interatividade** | Alta | Média |
| **Linguagens** | JavaScript, HTML, CSS | PHP, Python, Ruby, Java, etc. |
| **Segurança** | Menor | Maior |
| **SEO** | Mais difícil | Mais fácil |
| **Escalabilidade** | Melhor no cliente | Pode sobrecarregar o servidor |
| **Desempenho** | Varia conforme dispositivo | Consistente |
| **Dependência** | Navegador | Servidor |

**Aplicações Híbridas:**
Muitas aplicações modernas combinam abordagens client-side e server-side para tirar proveito dos benefícios de ambos. Um exemplo comum é o uso de `Single Page Applications` (SPAs) que fazem uso de frameworks client-side (como React ou Angular) para a interatividade e de APIs server-side para processamento de dados.



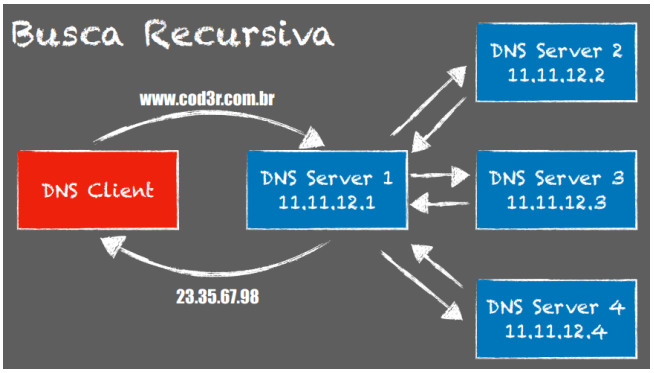

**Conteúdo Estático vs Dinâmico:**

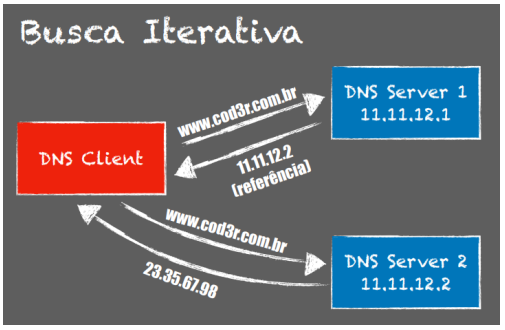
- O conteúdo é estático quando o servidor simplesmente, para renderizar uma página, lê os arquivos estáticos físicos no servidor web (images, css, etc).
- O conteúdo é dinâmico quando ele é gerado a partir de um código (é personalizado, pode mudar a cada vez q for rodado dependendo dos dados fornecidos) (/clientes, /produtos/top, /usuario/12, js, backend, etc).

**Tripé da aplicação web:** HTML, CSS e JS

**Servidor DNS:** Traduz os nomes de domínio em endereço IP (www.cod3r.com.br → 54.69.61.89).

- O DNS funciona sobre o protocolo UDP na porta 53!
- **Funcionamento básico:** usuário informa o endereço no browser → computador envia uma consulta DNS para o servidor local → o servidor local responde com o endereço IP → computador acessa o servidor a partir do IP obtido.
- Busca recursiva vs Iterativa




**HTTP Seguro (HTTPS):**

- Dados criptografados usando os protocolos TLS ou SSL
- Protege contra interceptação (MITM)
- Criptografa todas as informações: URL, cookies e headers
- Usa certificados digitais (emitidos por autoridades certificadoras)
- HTTP + TLS (transport layer security protcol SSL v4)
- HTTP + SSL (secure sockets layer protocol

**Web Service:** Serviços que acontecem em cima da web

- Simple Object Access Protocol (SOAP) → WSDL, XML, RPC/ É burocrático.
- Representational State Transfer (REST) → JSON, HTTP/ É simples.


SOAP

| URL | Método | Descrição |
|---|---|---|
| /clientes | POST | Novo cliente |
| /clientes | GET | Obtém todos |
| /clientes/36 | GET | Obtém cliente 36 |
| /clientes/12 | PUT | Atualiza cliente 12 |
| /clientes/41 | DELETE | Exclui cliente 41 |

REST

**Escolhas arquiteturais:** SOA vs Micro Service

- **SOA** é uma abordagem arquitetural onde os componentes de software são organizados como serviços interoperáveis. Esses serviços são reutilizáveis e podem ser usados em diferentes aplicações.
- **Microservices** é uma abordagem arquitetural que estrutura uma aplicação como uma coleção de pequenos serviços, cada um rodando em seu próprio processo e comunicando-se via protocolos leves, geralmente HTTP/REST.

### Comparação SOA vs Microservices

| Característica | SOA | Microservices |
|---|---|---|
| Granularidade | Serviços maiores e mais abrangentes | Serviços menores e focados |
| Comunicação | Frequentemente através de ESB | Diretamente via APIs RESTful |
| Protocolos | SOAP, REST, outros | Principalmente HTTP/REST, também gRPC |
| Reutilização | Alta reutilização de serviços | Foco menor na reutilização entre aplicações |
| Escalabilidade | Escalabilidade organizacional | Escalabilidade técnica |
| Complexidade | ESB pode adicionar complexidade | Gestão de múltiplos serviços pode ser complexa |
| Desenvolvimento | Monolítica (grande serviço) | Ágil (serviços independentes) |
| Dependências | Maior dependência de um ESB | Independência entre serviços |
| Implementação | Pode envolver diferentes tecnologias | Pode envolver diferentes tecnologias |
| Resiliência | Falha em um serviço pode afetar outros | Falha isolada a um único serviço |

**Computação em Nuvem:**

- Virtualização/ máquinas virtuais
- **Nuvens de Infraestrutura:** Aumenta a infraestrutura de acordo com a demanda do sistema, “adicionando” e desalocando novas máquinas de acordo com a demanda.

**HTTP - 8 métodos:**

- **GET:** Obter/ler informações do servidor
- **POST:** Submeter ou enviar informações ao servidor (cria novos registros)
- **PUT:** Altera os registros totalmente

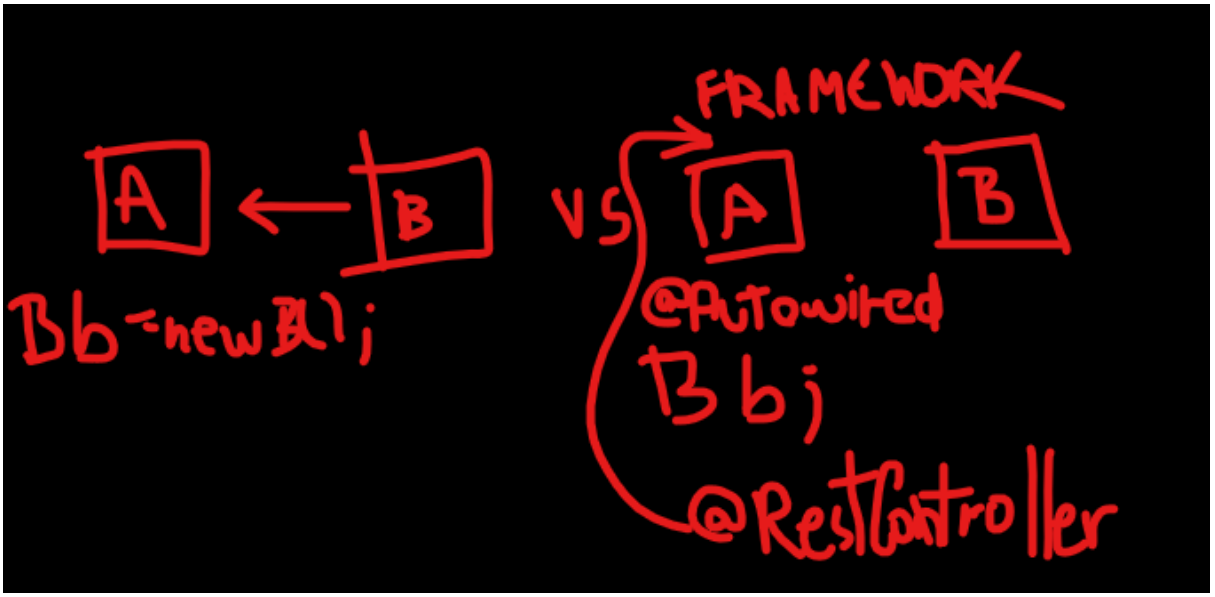
- **PATCH**: Altera os registros parcialmente
- **DELETE:** Deleta os registros
- **OPTIONS:** Retorna quais são os métodos https que suportam essa url (get, post, etc)
- **TRACE:** Requisição de teste
- **HEAD:** Mesma resposta do GET mas sem o conteúdo, só para conferir se ela está funcionando, por ex.

**Padrões de projeto → Padrão MVC (**View ↔ Controller ↔ Model**)**

- **Model:** Coração da sua aplicação. É a camada que se liga ao banco de dados. Nela, terão:
  - Regras de negócio
  - Entidades
  - Camada de acesso à dados
  - **View:** Responsável por renderizar a resposta (relacionado a visualizar a sua aplicação).
  - Javascript
  - CSS
  - HTML
  - Template Engine
- **Controller:** No meio dos dois, é quem coordena o fluxo de dados do model ao view, mandando request da página ao banco de dados, etc. Ele faz o intermédio do fluxo.
  - Browser → Webserver → Aplicação`→ Controller

  **Padrão DAO:** Data Access Object

  **Injeção de Independência:**


A
B
Bb = new B();
VS
FRAMEWORK
A
B
@Autowired
B bj
@RestController