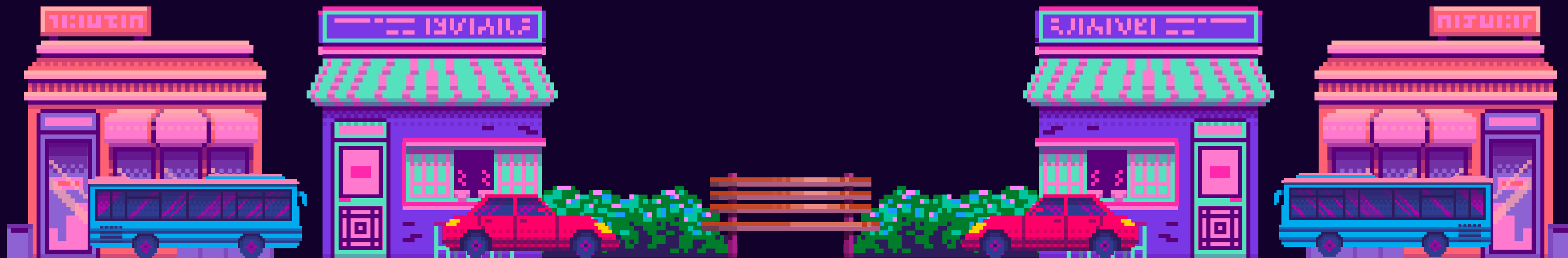




HELLO WORLD



PRIMEIRA API E ESSE MODULES



A HISTÓRIA

Antes do ES6, o Javascript possuía sistemas de controles de módulos como o RequireJS, CommonJS e o sistema de injeção de dependências do Angular. Outras ferramentas como o Browserify e o próprio Webpack resolveram muitos problemas nesse contexto.

Em 2015 tivemos uma primeira implementação do sistema de módulos no Javascript vanilla para Node.js utilizando o require que todos conhecemos. Mas isto ainda não chegou para os browsers.



STRICT MODE

O strict mode faz várias mudanças nas semânticas normais do JavaScript. Primeiro, o strict mode elimina alguns erros silenciosos do JavaScript fazendo-os lançar exceções. Segundo, o strict mode evita equívocos que dificultam que motores JavaScript realizem otimizações: código strict mode pode às vezes ser feito para executar mais rápido que código idêntico não-strict mode. Terceiro, strict mode proíbe algumas sintaxes que provavelmente serão definidas em versões futuras do ECMAScript.

```
> (function() {  
  'use strict'  
  variable = 'hey'  
})();
```

```
✖ ▶ Uncaught ReferenceError: variable is not defined  
   at <anonymous>:3:12  
   at <anonymous>:4:3
```

```
> (() => {  
  'use strict'  
  myname = 'Flavio'  
})();
```

```
✖ ▶ Uncaught ReferenceError: myname is not defined  
   at <anonymous>:3:10  
   at <anonymous>:4:3
```

EXPORT

No modelo CommonJS podemos exportar valores atribuindo eles ao `module.exports` como no snippet abaixo:

```
1  module.exports = 1
2  module.exports = NaN
3  module.exports = 'foo'
4  module.exports = { foo: 'bar' }
5  module.exports = [ 'foo', 'bar' ]
6  module.exports = function foo () {}
7  module.exports = () => {}
```

No ES6, os módulos são arquivos que dão export uma API (basicamente igual ao CommonJS). As declarações, variáveis, funções e qualquer coisa daquele módulo existem apenas nos escopos daquele módulo, o que significa que qualquer variável declarada dentro de um módulo não está disponível para outros módulos (a não ser que eles sejam exportados explicitamente como parte da API, e importados posteriormente no módulo que as usa).

EXPORT PADRÃO

Podemos simular o comportamento do CommonJS basicamente trocando o `module.exports` por `export default`:

```
1  export default = 1
2  export default = NaN
3  export default = 'foo'
4  export default = { foo: 'bar' }
5  export default = [ 'foo', 'bar' ]
6  export default = function foo () {}
7  export default = () => {}
```

Ao contrário dos módulos no CommonJS, declarações `export` só podem ser colocadas no top level do código, e não em qualquer parte dele. Presumimos que essa limitação existe para tornar mais fácil a vida dos interpretadores quando vão identificar os módulos, mas, olhando bem, é uma limitação bem válida, porque não há muitas boas razões para que possamos definir exports dinâmicos dentro das funções da nossa API.

MELHORES PRÁTICAS COM EXPORT

Todas essas possibilidades de exportar um módulo vão introduzir um pouco de confusão na cabeça das pessoas. Na maioria dos casos é encorajado utilizar apenas um export default (e fazer isso só no final do módulo). Então você pode importar a API como o nome do próprio módulo.

```
1  var api = {  
2    foo: 'bar',  
3    baz: 'fooz'  
4  }  
5  
6  export default api
```

Os benefícios são:

- A interface que é exportada se torna bem óbvia, ao invés de termos que ficar procurando aonde exportamos a interface dentro do módulo.
- Você não cria a confusão sobre onde usar um export default ou um export nomeado (ou uma lista de exports nomeados). Tente se manter no export default
- Consistência. No CommonJS é normal usar um único método em um módulo. Fazer isso com exports nomeados é impossível porque você vai estar expondo um objeto com um método dentro, a não ser que você use o as default no export de lista. Já o export default é mais versátil porque você pode exportar só uma coisa
- Usar export default torna o import mais rápido

IMPORTANDO EXPORTS NOMEADOS

É muito parecido com o destructuring assignment que temos na nova especificação.

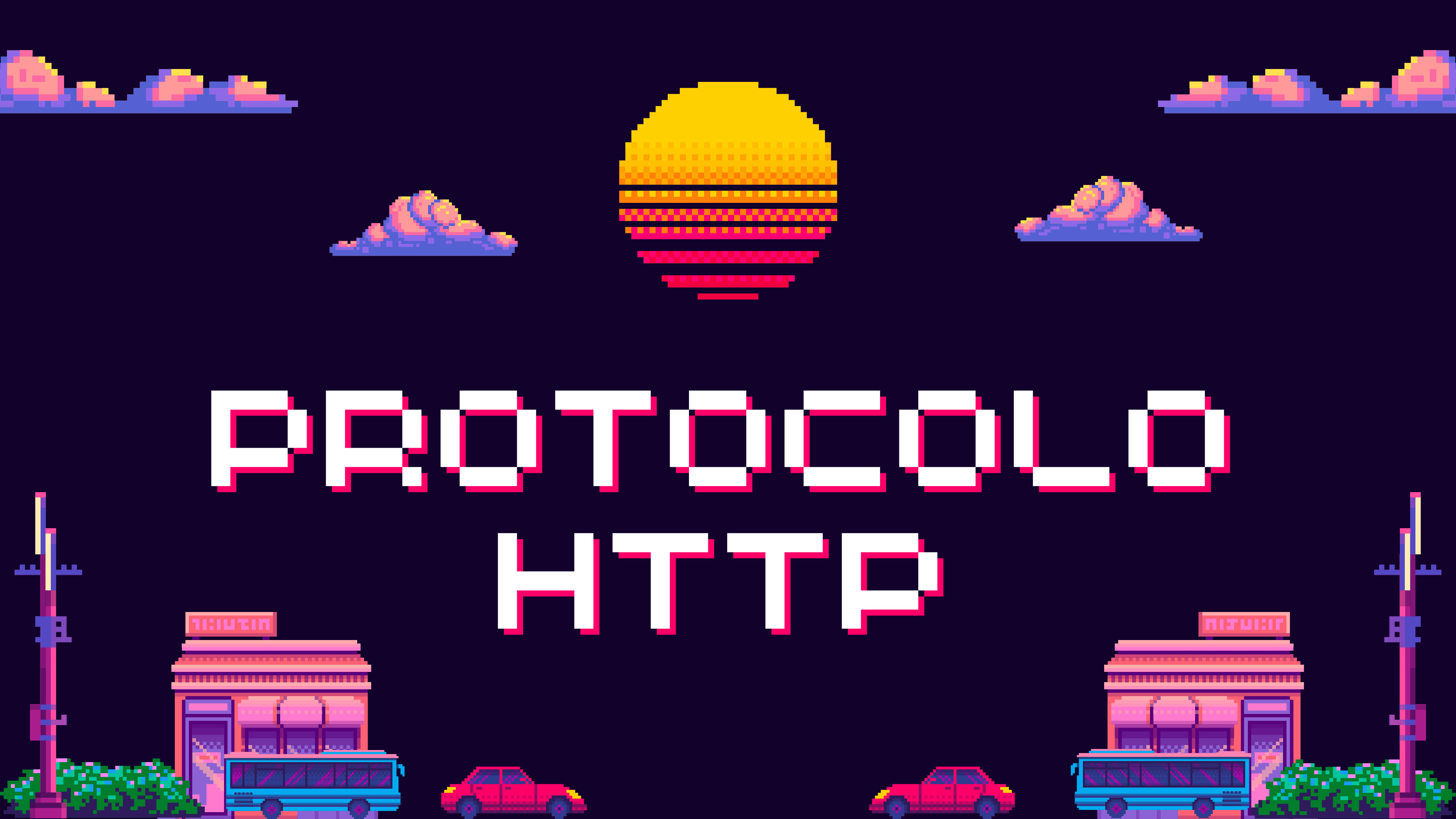
```
import { map, reduce } from 'lodash'
```

Uma outra maneira que podemos também importar os export nomeados, é dar um alias para cada um, ou então apenas para um deles:

```
import { cloneDeep as clone, map } from 'lodash'
```

Você pode misturar os named imports com os exports padrões sem usar as chaves (mas aí você não vai poder dar um alias para eles):

```
1 import { default, map } from 'lodash'  
2 import { default as _, map } from 'lodash'  
3 import _, { map } from 'lodash'
```

PROTOCOLLO

HTTP


A sigla HTTP vem de Hypertext Transfer Protocol. Traduzido para o português, HTTP significa “Protocolo de Transferência de Hipertexto”.

O termo “hipertexto” descreve um sistema de organização de informações em que documentos têm conexões clicáveis, permitindo aos usuários saltar de uma parte do texto para outra de maneira não linear.

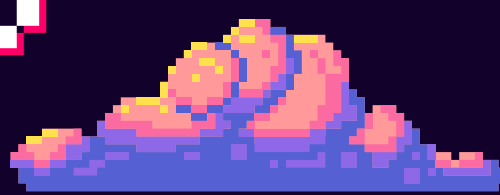
Para começar, HTTP, ou Hypertext Transfer Protocol, é um protocolo de comunicação utilizado para a transferência de informações na World Wide Web (WWW) e em outros sistemas de rede.

O HTTP é a base para que o cliente e um servidor web troquem informações. Ele permite a requisição e a resposta de recursos, como imagens, arquivos e as próprias páginas webs que acessamos, por meio de mensagens padronizadas. Com ele, é possível que um estudante num café em São Paulo leia um artigo que está armazenado em um servidor no Japão.





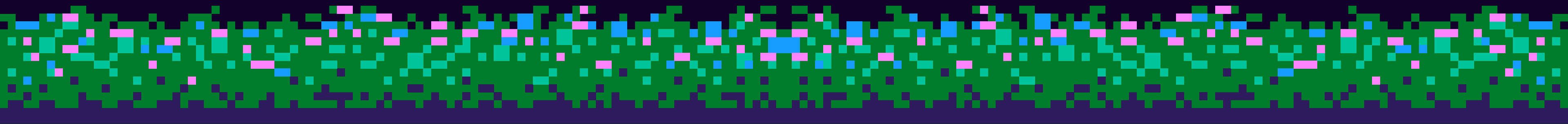
INFRAESTRUTURA DO PROTOCOLO HTTP



Cliente: Um cliente é um dispositivo ou software que interage com recursos na web em nome do usuário. Pode ser um navegador, aplicativo móvel ou software automatizado, que faz solicitações HTTP para acessar informações ou serviços em servidores.

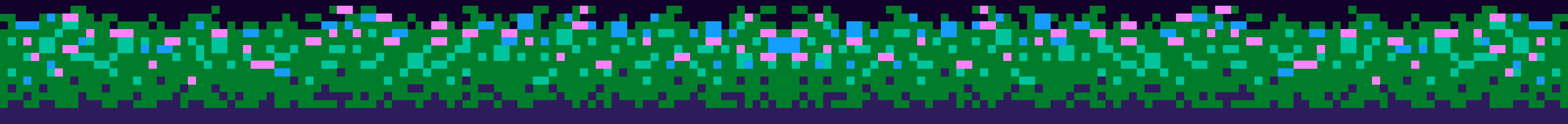
Servidor: O servidor é o dispositivo que hospeda e fornece recursos solicitados pelos clientes. Ele processa as requisições, executa a lógica necessária e retorna as respostas, como páginas web ou dados. A inclusão do cabeçalho Host no HTTP 1.1 permite que múltiplos sites sejam hospedados em um único servidor com o mesmo endereço IP.

Proxies: Proxies são intermediários entre clientes e servidores. Existem dois tipos principais: forward proxies, que agem em nome dos clientes para buscar recursos dos servidores, e reverse proxies, que operam em nome dos servidores para receber solicitações dos clientes. Além de otimizar o tráfego e melhorar a eficiência da rede, proxies oferecem segurança, anonimato, filtragem de conteúdo malicioso, controle de acesso e balanceamento de carga. Eles também podem acelerar o carregamento de páginas e fornecer logging e monitoramento.





ASPECTOS FUNDAMENTAIS DO HTTP NA COMUNICAÇÃO ENTRE CLIENTES E SERVIDORES WEB

- Métodos HTTP O HTTP utiliza métodos, como GET, POST, PUT e DELETE, para indicar a ação desejada na solicitação. Esses métodos definem operações comuns, como obter dados, enviar dados para processamento, atualizar ou excluir recursos.
 - Headers (Cabeçalhos) Os cabeçalhos HTTP contêm informações adicionais sobre a requisição ou a resposta. Eles incluem dados como o tipo de conteúdo, a data da requisição, cookies, e muitos outros.
 - URI (Uniform Resource Identifier) Os recursos na web são identificados por URLs (Uniform Resource Locators) ou URIs. Uma URI é uma sequência de caracteres que identifica um nome ou um recurso na web.
 - Cache O HTTP possui mecanismos de cache para melhorar o desempenho. Os cabeçalhos de controle de cache indicam se o navegador do cliente ou intermediários podem armazenar em cache uma resposta HTTP e, caso possível, por quanto tempo e em quais condições.
 - Tipo de hipermídia
 - O tipo de hipermídia comum no contexto do HTTP é o HTML, utilizado para criar e apresentar documentos na web. Porém, o HTTP suporta uma variedade de tipos de mídia, como XML, JSON, imagens e vídeos, permitindo a transmissão de diversos tipos de dados online.
- 



O PROCESSO DE REQUEST- RESPONSE

LINHA DE SOLICITAÇÃO (REQUEST LINE):

Na linha de solicitação, presente em uma solicitação HTTP, são especificados três elementos principais: o método HTTP, a URI (Uniform Resource Identifier) e a versão do protocolo HTTP.

Exemplo de linha de solicitação:

GET /exemplo/recurso HTTP/1.1

Neste exemplo, "GET" é o método HTTP, "/exemplo/recurso" é a URI, e "HTTP/1.1" é a versão do protocolo.

Uma mensagem inteira de requisição HTTP do servidor poderia se parecer com isto:



```
GET / HTTP/1.1
```

```
Host: www.meusite.com.br
```

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
```

```
Accept: */*
```

```
Accept-Language: en-US,en;q=0.5
```

```
Accept-Encoding: gzip, deflate
```

```
Connection: keep-alive
```

LINHA DE STATUS (STATUS LINE):

Na linha de status, presente em uma resposta HTTP, são fornecidos três elementos principais: a versão do protocolo HTTP, um código de status e uma mensagem de status.

Exemplo de linha de status:

HTTP/1.1 200 OK

Neste exemplo, "HTTP/1.1" é a versão do protocolo, "200" é o código de status indicando sucesso, e "OK" é a mensagem de status associada.

Uma mensagem inteira de resposta HTTP do servidor poderia se parecer com isto:

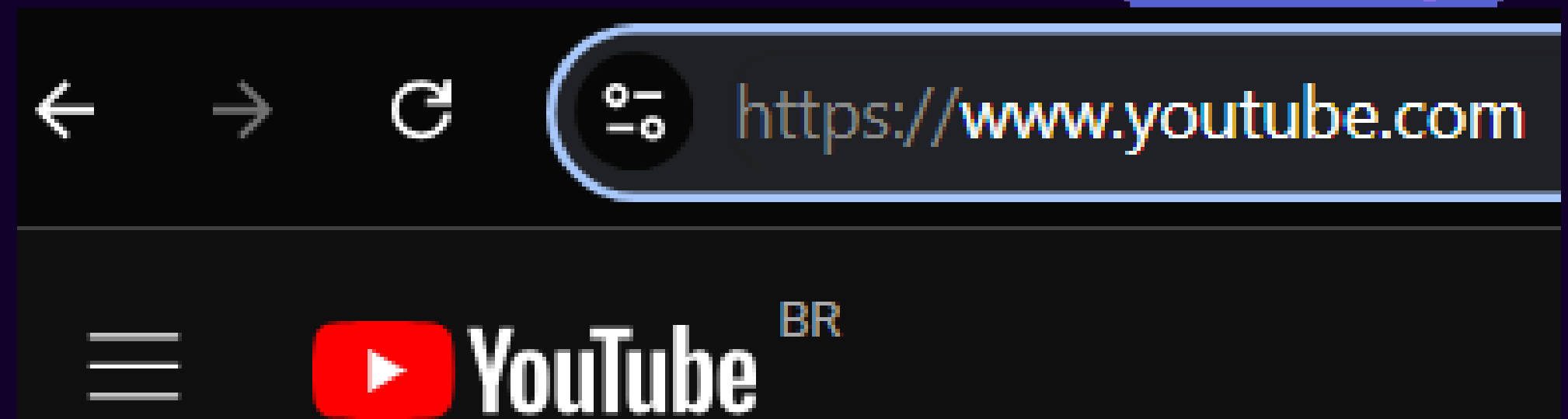
```
HTTP/1.1 200 OK
Date: Sun, 09 Apr 2023 12:28:53 GMT
Server: Apache/2.4.18 (Ubuntu)
Last-Modified: Sat, 08 Apr 2023 23:11:04 GMT
Content-Length: 612
Content-Type: text/html; charset=UTF-8
Connection: closed
```

```
<html>
<head>
<title>Exemplo de Site</title>
</head>
<body>
<h1>Olá, Mundo!</h1>
<p>Este é um exemplo de corpo de resposta HTTP.</p>
</body>
</html>
```

QUAL É A DIFERENÇA ENTRE HTTP E HTTPS?

Compreendemos que o HTTP é um protocolo de comunicação entre cliente e servidor. Mas, você também já pode ter visto em algum site HTTPS, ao invés de HTTP. Afinal, qual a diferença?

Exemplo de uso do HTTPS no site do Youtube.



A sigla HTTPS vem de Hypertext Transfer Protocol Secure, ou Protocolo de Transferência de Hipertexto Seguro. Como o próprio nome já sugere, o HTTPS é uma versão mais segura do HTTP.

O HTTP transmite dados sem criptografia, o que pode torná-los mais suscetíveis a ameaças de terceiros. Já o HTTPS, utiliza uma criptografia SSL/TLS para proteger a integridade de dados, ou seja, se alguém conseguir interromper de alguma forma a comunicação, os dados estão protegidos por uma criptografia.

O HTTPS é crucial para transações online e o manuseio de dados sensíveis, oferecendo uma camada extra de segurança HTTP.

MÉTODOS

- GET: Solicita dados de um recurso específico, geralmente usado para recuperar informações de uma URL, com a opção de passar parâmetros na URL e incluir headers.
- HEAD: Semelhante ao GET, mas retorna apenas os cabeçalhos da resposta, útil para obter metadados sem carregar o conteúdo completo.
- POST: Envia dados ao servidor para processamento, frequentemente usado para formulários ou criação de recursos.
- PUT: Cria ou atualiza um recurso específico, substituindo completamente o recurso existente.
- DELETE: Solicita a remoção de um recurso específico no servidor.
- TRACE: Usado para diagnóstico, reflete a solicitação de volta ao cliente para fins de depuração.
- OPTIONS: Obtém as opções de comunicação permitidas para um recurso ou servidor.
- CONNECT: Estabelece uma conexão de túnel, geralmente para comunicações seguras via proxy HTTP.

GET	/pet/{petId}	Find pet by ID
PUT	/pet	Update an existing pet
DELETE	/pet/{petId}	Deletes a pet
POST	/pet/{petId}/uploadImage	uploads an image



API REST

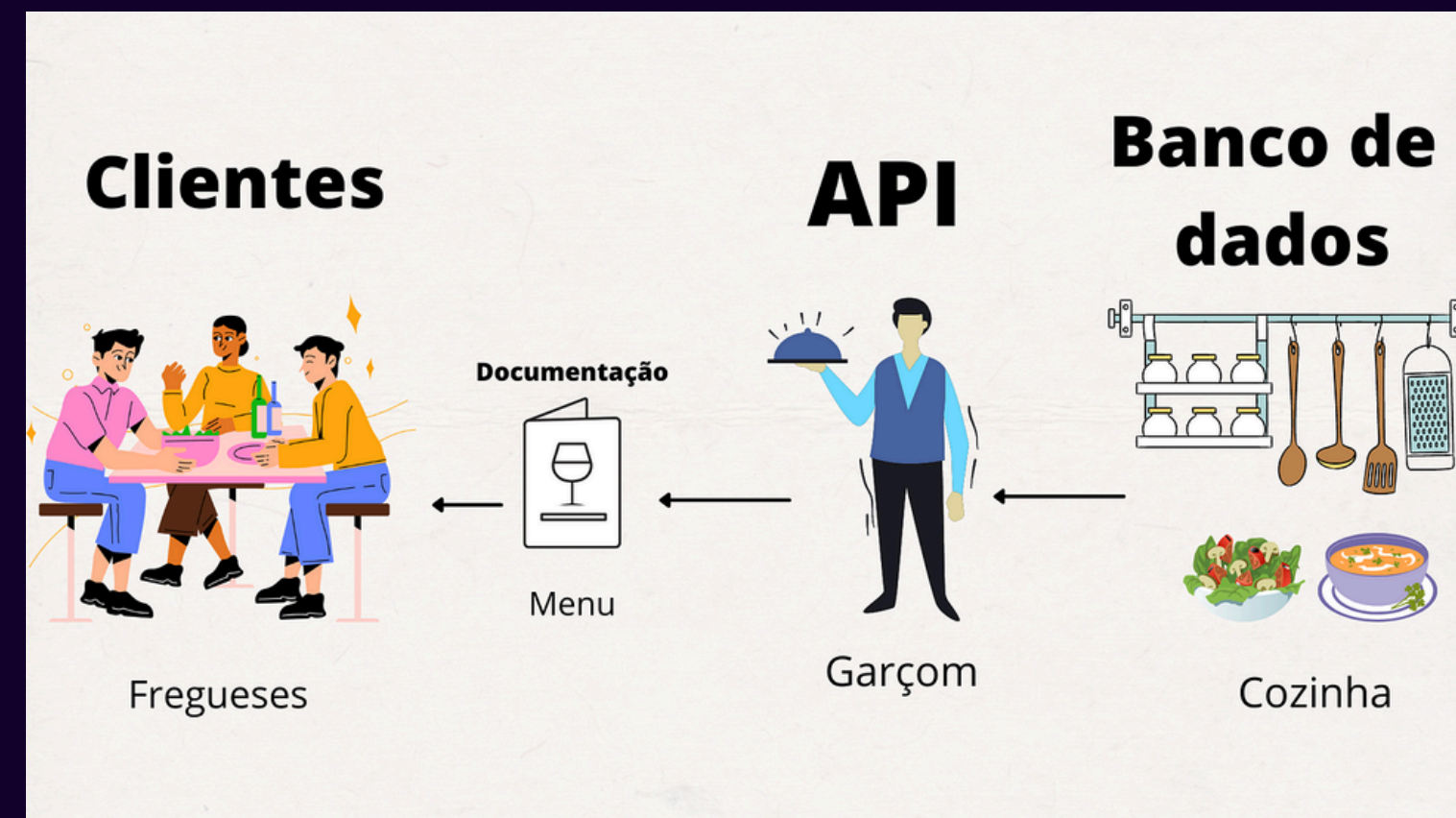
Imagine um restaurante em que o cliente quer fazer seus pedidos com base no que está no cardápio. Mas ele não pode pedir diretamente para o chef de cozinha.

Então, ele interage com o garçom, que é a pessoa que envia as solicitações para a cozinha, para que possam começar a preparar os pedidos.

Assim que o prato estiver pronto, o garçom será responsável por encaminhá-lo até a mesa do cliente, tudo isso de forma organizada.

Essa dinâmica entre cliente, garçom e cozinha reflete o relacionamento essencial em uma API.

Esse relacionamento também define como diferentes componentes de software devem interagir, facilitando a integração entre sistemas e aplicativos, o que pode ser utilizado para diferentes propósitos.



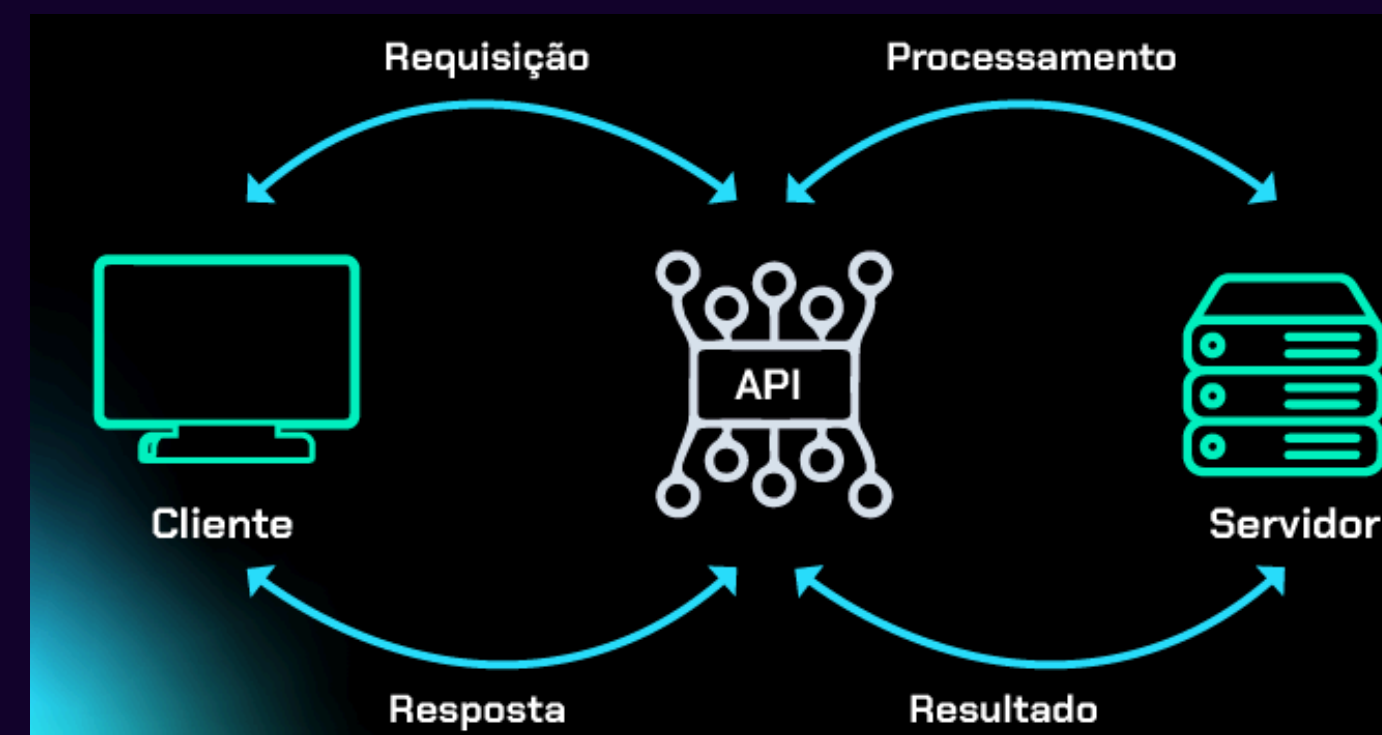
O QUE SIGNIFICA API?

API é um acrônimo em inglês para **Application Programming Interface**. Em português, significa Interface de Programação de Aplicações.

De forma geral, é um conjunto de padrões, ferramentas e protocolos que permite a criação mais simplificada e segura de plataformas, pois permite a integração e a comunicação de softwares e seus componentes.

Como funciona uma API?

As APIs desempenham um papel de conectividade e interação entre diferentes sistemas e aplicações. Existem vários tipos de APIs, cada um servindo a propósitos específicos. Vamos explorar alguns desses tipos?



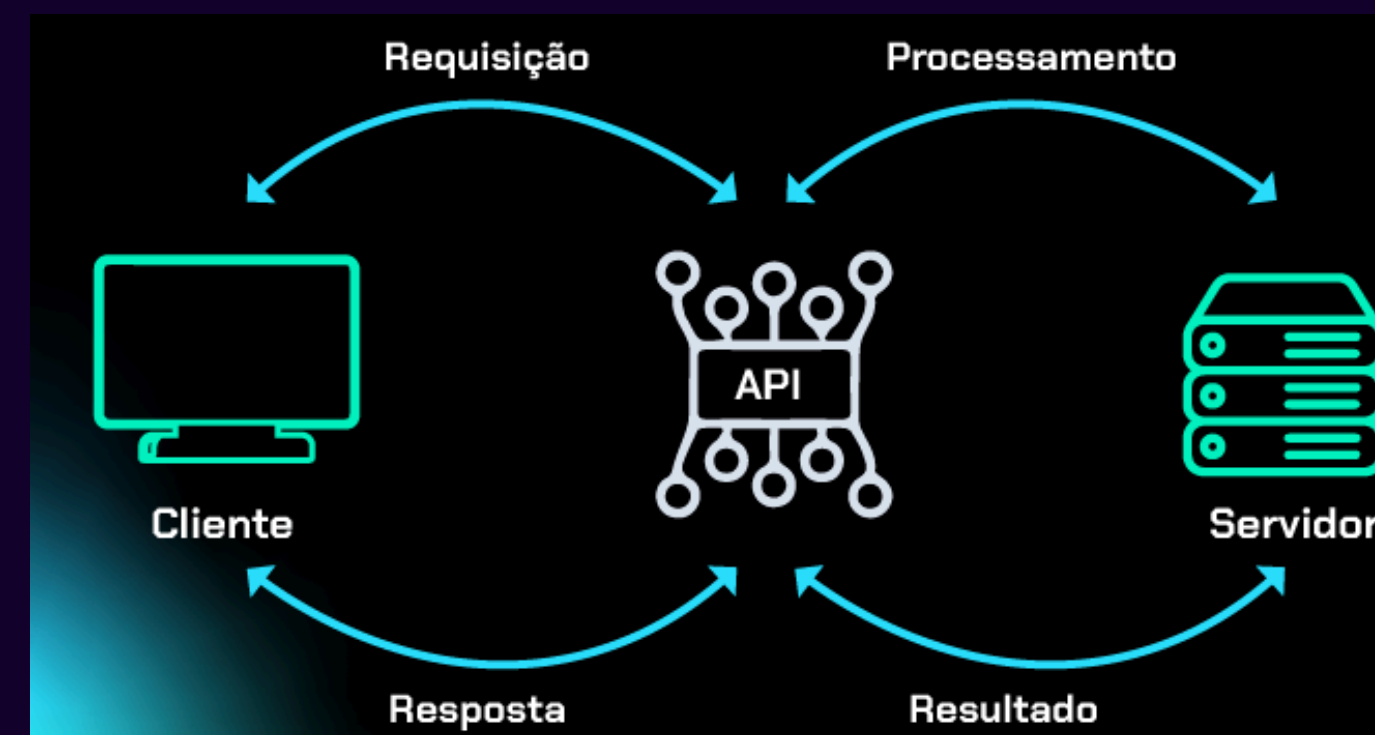
APIS BASEADAS EM WEB

A API baseada em Web Serve como uma ponte eficaz entre o cliente e o servidor. Para isso, utiliza protocolos da web, como **HTTP**.

Assim, permite a comunicação entre sistemas pela internet e, conseqüentemente, que aplicações diferentes interajam e compartilhem dados de forma padronizada, independentemente da tecnologia utilizada em cada extremidade.

Seu funcionamento acontece da seguinte forma:

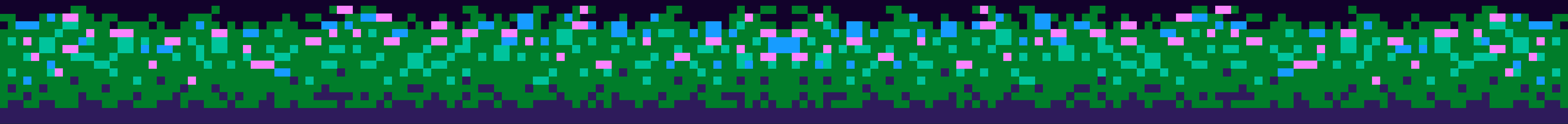
- O servidor aguarda por requisições.
- O cliente envia uma requisição HTTP para o endpoint adequado no servidor (por exemplo, GET /tasks ou POST /tasks).
- O servidor processa a requisição, realiza as operações necessárias (obter ou adicionar tarefas) e retorna uma resposta ao cliente, geralmente em formato JSON.
- O cliente recebe a resposta e pode então processar os dados conforme necessário.





QUAIS SÃO AS VANTAGENS DE USAR UMA API?

APIs têm várias funções e benefícios importantes:

- **Interoperabilidade:** Permitem que diferentes sistemas e aplicativos se comuniquem de forma padronizada, facilitando a integração.
 - **Acesso a funcionalidades:** Desenvolvedores podem usar funcionalidades de um software sem conhecer seus detalhes internos, criando aplicações mais complexas.
 - **Reutilização de código:** APIs permitem que o código seja reutilizado em diferentes contextos, aumentando a eficiência e reduzindo a redundância.
 - **Desenvolvimento rápido:** Ao usar APIs, desenvolvedores podem incorporar funcionalidades existentes, economizando tempo e recursos.
 - **Integração de serviços:** APIs são usadas para integrar serviços de terceiros, como autenticação via API.
 - **Distribuição de dados:** Facilitam o compartilhamento de dados entre diferentes sistemas, permitindo acesso e atualização de informações.
 - **Atualização independente:** Permitem que partes do software sejam atualizadas separadamente, facilitando a manutenção.
 - **Economia de recursos:** Desenvolvedores podem aproveitar serviços existentes, economizando esforço e recursos.
- 

APIs WEB

Considerando que as APIs remotas abrangem um conceito mais amplo, as APIs web se destacam como a comunicação específica realizada pela internet, utilizando os protocolos HTTP/HTTPS.

Esses protocolos estabelecem uma padronização para as mensagens de requisição e resposta. Em grande parte dos casos, essas mensagens são formatadas em XML (Extensible Markup Language) ou JSON (JavaScript Object Notation), como exemplificado abaixo:

Estrutura de mensagens formatadas em JSON:

```
1 {  
2   "lista_pessoas":  
3   {  
4     "pessoas": [  
5       { "nome": "João Silva", "sexo": "M", "idade": "22" },  
6       { "nome": "Maria Eduarda", "sexo": "F", "idade": "21" },  
7       { "nome": "Pedro Gomes", "sexo": "M", "idade": "18" }  
8     ]  
9   }  
10 }
```



APIS REST

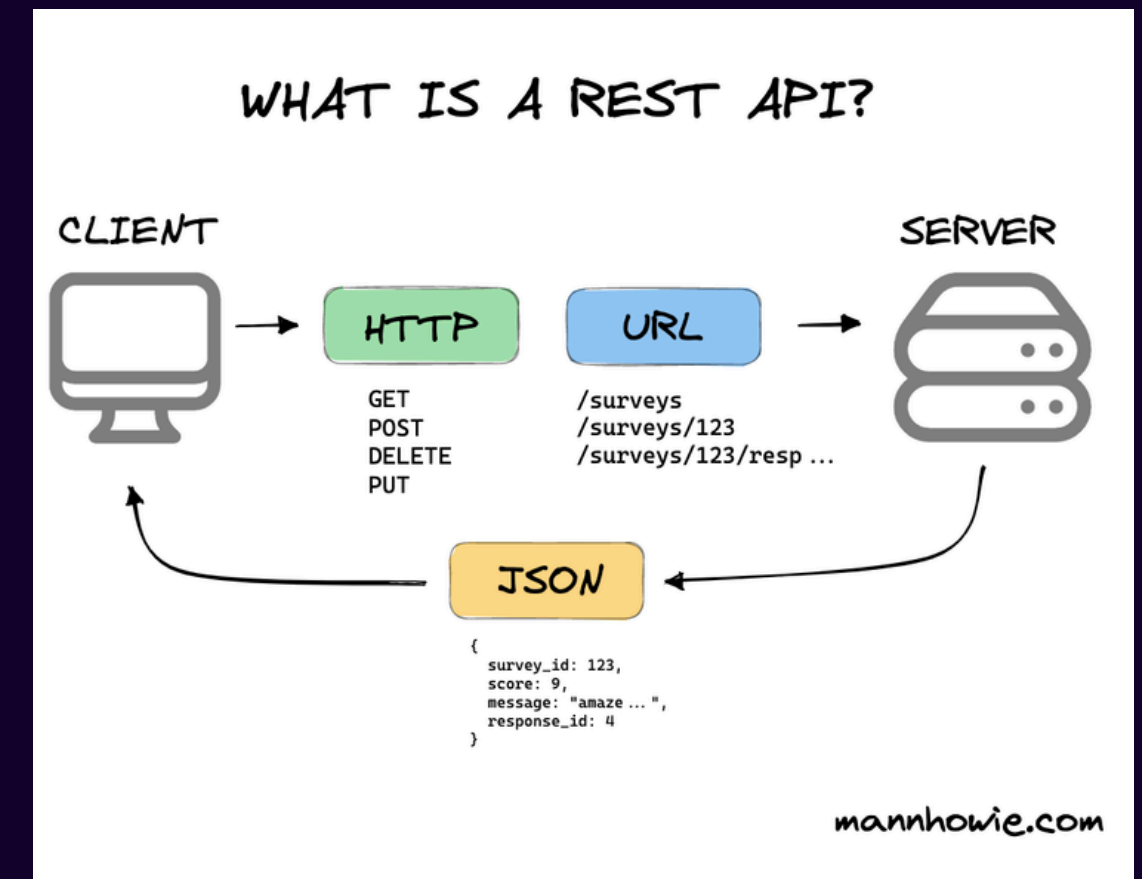
(REPRESENTATIONAL STATE TRANSFER)

As APIs construídas com base na arquitetura REST seguem um estilo que incorpora os princípios fundamentais da web, visando o desenvolvimento de aplicações escaláveis e eficientes.

O formato JSON é comumente adotado para a transmissão de dados e as operações padrão do protocolo HTTP (GET, POST, PUT, PATCH e DELETE) são empregadas para realizar interações com os recursos.

As APIs REST podem ser desenvolvidas utilizando uma grande quantidade de linguagens de programação. No entanto, é crucial que essas APIs estejam alinhadas com os seis princípios de design REST a seguir:

- Interface uniforme: Comunicação consistente entre cliente e servidor usando métodos padronizados, como HTTP e JSON.
- Separação cliente-servidor: Cliente e servidor são independentes, comunicando-se via URI.
- Sem estado: Cada solicitação inclui todas as informações necessárias, sem depender de estados anteriores.
- Cache: Recursos podem ser armazenados temporariamente para respostas mais rápidas, tanto no cliente quanto no servidor.
- Arquitetura em camadas: O sistema é dividido em camadas com funções específicas, facilitando a organização e escalabilidade.
- Código sob demanda (opcional): O servidor pode enviar código para ser executado pelo cliente, estendendo suas capacidades.





EODRA

COODRA

EODRA

EODRA

NOSSOS CONTATOS:



27 99500-7495



<https://beacons.ai/prismatech>



producaoprismatech@gmail.com



Avenida Jerônimo Monteiro 145, Vitória

