

HTTPS e Postman

O que vamos ver hoje?

- Backend
- HTTP e HTTPS
- APIs
- Postman

Backend e APIs

Classificando as Aplicações

- Em projetos de aplicação web, costumamos dividir nosso código em **duas grandes classificações**: Frontend e Backend

Front-End X Back-End

Front

- “Traduz” o design para algo funcional
- Contato direto com o **usuário**
- Interface gráfica
- **“Client-Side”** (lado do cliente)

Back

- Gerencia informações
- Persiste os **dados**
- Cria funcionalidades
- **“Server-side”** (lado do servidor)

Responsabilidades do Backend

- Guardar e fornecer as informações presentes em um **banco de dados**
- Montar toda a estrutura da **lógica de negócio**
- Gerenciar todos os **serviços** utilizados

Protocolos de Comunicação

Contexto

- O Frontend e o Backend precisam se **comunicar**
- Se comunicar significa: **trocar informações/dados** (mensagens) entre si
- Para ter certeza de que todos os sistemas consigam se comunicar, foram criados **protocolos de comunicação**

Protocolos de Comunicação

- Protocolos de comunicação são **um conjunto de regras** que permite que duas ou mais entidades façam a **troca de informações** entre si
- Eles determinam:
 - Os **formatos** das mensagens
 - Os **tipos** possíveis de mensagens
- Existem vários tipos de protocolos

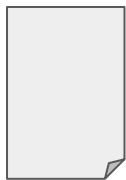
Protocolos - Exemplo 🤝

- Mafalda quer se **comunicar** com Bob enviando uma carta para ele



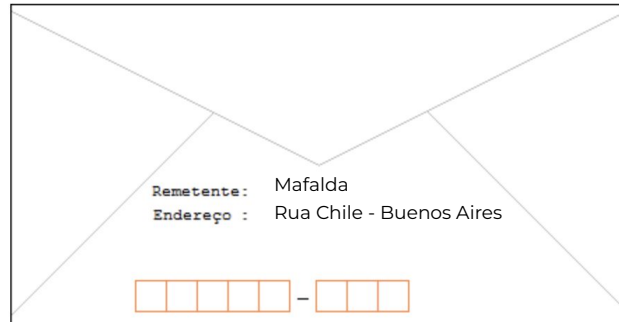
Protocolos - Exemplo 🤝

- O sistema de correios determina o **formato da mensagem e regras** para que ela seja enviada



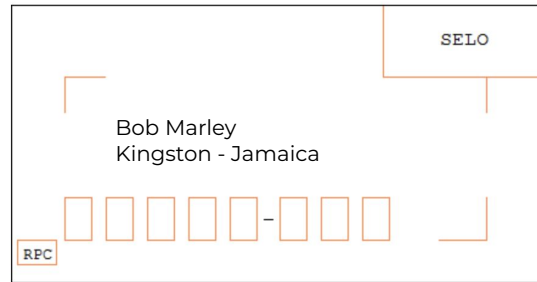
Protocolos - Exemplo 🤝

1) As informações do **remetente** devem estar no verso do envelope



Protocolos - Exemplo 🤝

2) As informações do **destinatário** na frente do envelope



Protocolos - Exemplo 🤝

4) A parte de dentro deve conter o local, data e o **conteúdo** da carta



Local, dd/mm/yyyy

Conteúdo

Assinatura

A diagram of a letter template. It consists of a light gray rectangular box with a folded bottom-right corner. Inside the box, the text "Local, dd/mm/yyyy" is at the top left, "Assinatura" is at the bottom left, and a large, empty rectangular area in the center is labeled "Conteúdo".

Protocolos - Exemplo 🤝

- Então Mafalda preenche com as informações que deseja



Argentina, 01/01/1972

Olá, Bob. Como você está?
Gostaria de saber quantos
discos você já lançou, e quais
são os nomes deles.

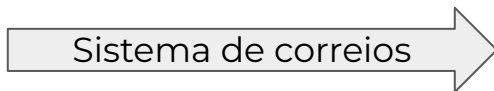
Obrigada,

Mafalda



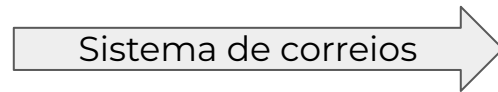
Protocolos - Exemplo 🤝

- Coloca o conteúdo dentro do envelope e envia para Bob pelo sistema de correios



Protocolos - Exemplo 🤝

- Os correios entregam a carta para Bob



Protocolos - Exemplo

- Mafalda seguiu todas as **regras do protocolo**
- O sistema de correios consegue **entregar a mensagem** corretamente
- Bob consegue **entender pedido** de Mafalda
- Bob **responde** Mafalda com os discos que já foram lançados

Protocolos

- **FTP (*File Transfer Protocol*)**
 - Usado, majoritariamente, para **trocar arquivos**
- **IMAP (*Internet Message Access Protocol*)**
 - Protocolo que permite **receber e-mails**
- **SMTP (*Simple Mail Transfer Protocol*)**
 - Protocolo que permite **enviar e-mails**

Protocolos 🤝

- **SSH (Secure Shell)**
 - Acessar uma máquina **remotamente**
 - Exige algum tipo de **login (usuário e senha)**
- **ICMP (Internet Control Message Protocol)**
 - Usado para fazer **sanity-check**
 - **"Cutucar"**



HTTP e HTTPS

HTTP



- **HTTP: Hypertext Transfer Protocol**
- Principal protocolo usado para **comunicação de duas partes na web**
- Permite troca de **diversos tipos de mensagem** como: texto, documentos, arquivos, scripts, etc.

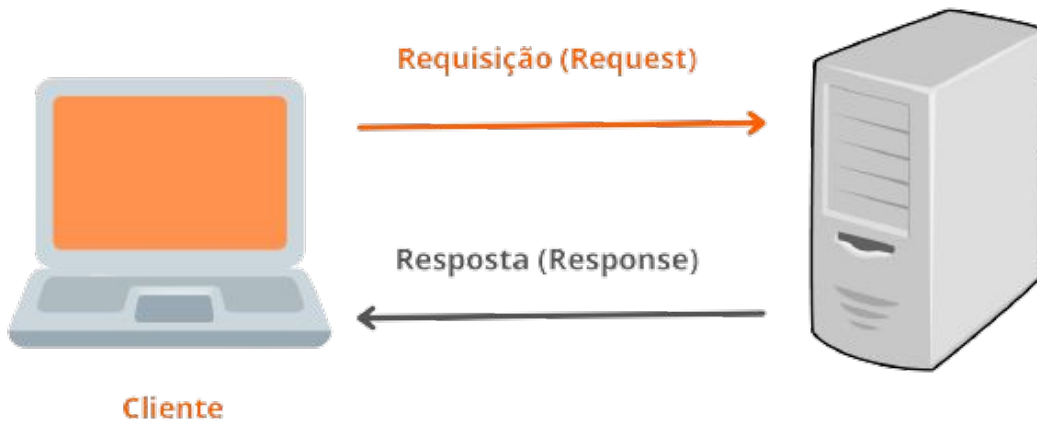


Tim Berners-Lee

Criador do Protocolo HTTP
e da linguagem HTML

HTTP - Entidades

- O protocolo HTTP determina a comunicação entre duas entidades:
 - **client (cliente)** e o **server (servidor)**



Cliente => Inicia a comunicação, fazendo um pedido para o servidor

Servidor => Recebe os pedidos do cliente, e obrigatoriamente fornece uma resposta.

Formato de uma mensagem HTTP

- Assim como no protocolo do sistema de correios, o protocolo HTTP também possui **regras de como devem ser as mensagens**
- Várias informações devem ser passadas para definir o destinatário, o tipo, o formato e outros parâmetros
- Assim, a mensagem pode ser **entregue** e **interpretada** pelo servidor

HTTP - Endereço

- O endereço representa a **localização** do destinatário da mensagem, ou seja, **do servidor**
- Normalmente, este **endereço é uma URL** (Uniform Resource Location)

<http://minha-api.com/>

HTTP - Endereço - Path

- O path é o que vem **depois da primeira barra** no endereço
- Especifica **qual o recurso** que está sendo pedido

`http://minha-api.com/`**usuario**

HTTP - Endereço - Query

- A query é uma parte **adicional** ao endereço
- Permite adicionar parâmetros arbitrários à requisição

`http://minha-api.com/usuario?nome=Mafalda&idade=20`

HTTP - Método

- O método representa o **tipo de operação** que irá ocorrer
- Basicamente, é o tipo de mensagem sendo enviada
- Métodos mais usados:

GET => Método para **pegar** informações

POST => Método para **inserir** informações

PUT => Método para **editar** informações

DELETE => Método para **deletar** informações

HTTP - Headers

- Os headers guardam as **informações sobre a própria requisição** e podem ter qualquer finalidade
- Exemplos:
 - **Formato** do conteúdo
 - **Tipo** do conteúdo
 - **Identificação** do remetente
 - **Tamanho** da mensagem

HTTP - Headers

- Existem alguns headers mais comuns e relevantes:
 - **Authorization:** indica **quem é o remetente** da mensagem
 - **Content-type:** define o **tipo e formato do conteúdo**

HTTP - Body

- O body guarda o **conteúdo** da mensagem, quando ele existir
- Podem seguir vários formatos, como por exemplo: XML, Form-data, Raw e **JSON**
- Atualmente o JSON é o formato padrão, então não há necessidade ser especificado no header o **Content-type** como **application/json**.

HTTP - Body - JSON

- O JSON (***Javascript Object Notation***) é uma maneira de representar informações e é um dos principais padrões para comunicação HTTP
- Muito parecido com **objetos Javascript** e é muito fácil de **converter** de um para outro!
- Garante **estrutura** e **hierarquia** de informações

HTTP - Body - JSON

- A principal diferença é que as chaves devem estar todas **entre aspas**
- Números, strings, arrays, null, undefined ou outros objetos são aceitos como **valores**
- Conversão:
 - **Objeto** ⇒ **JSON**: `JSON.stringify(meuObjeto)`
 - **JSON** ⇒ **Objeto**: `JSON.parse(meuJson)`

Objeto x JSON

Javascript

```
1 const usuario = {
2   nome: "Mafalda",
3   idade: 20,
4   localizacao: {
5     pais: "Argentina",
6     cidade: null
7   },
8   emprego: undefined,
9   amigos: ["Bob", "Carol"],
10 }
```

JSON

```
1 {
2   "nome": "Mafalda",
3   "idade": 20,
4   "localizacao": {
5     "pais": "Argentina",
6     "cidade": null
7   },
8   "emprego": undefined,
9   "amigos": ["Bob", "Carol"],
10 }
```

HTTP - Status

- O status é um parâmetro exclusivo da **resposta**
- Representação **numérica** e **padronizada** do que aconteceu com a requisição



1XX
Informativo

2XX
Sucesso

3XX
Redirecionamento

4XX
Erro Cliente

5XX
Erro Servidor
(não programado)

HTTP - Status



- [Doguinhos](#)
- [Gatinhos](#)
- [Detalhes](#)



400

Bad Request



201

Created



401

Unauthorized



500

Internal Server Error

HTTP - Formato das Mensagens

- Vamos imaginar que a Mafalda é o **client** e o Bob é um **server**

Request

```
GET http://bob-api.com/discos
```

```
Headers:
```

```
Authorization: "mafalda"
```

Response

```
Status: 200
```

```
Headers:
```

```
Content-type: "application/json"
```

```
Body:
```

```
[  
  "The Wailing Wailers",  
  "Soul Rebels"  
  "Soul Revolution"
```

```
...
```

```
]
```

HTTPS

- Versão **mais segura** do HTTP
- Exatamente igual, mas mensagens são **criptografadas**
- Se alguém observa a mensagem no meio, **não consegue entender** o conteúdo



APIs e Postman

API

- Já sabemos que o Backend e o Frontend precisam se **comunicar**
- Isso é feito através de uma **API** (*Application Programming Interface*)

Mas o que é uma API?

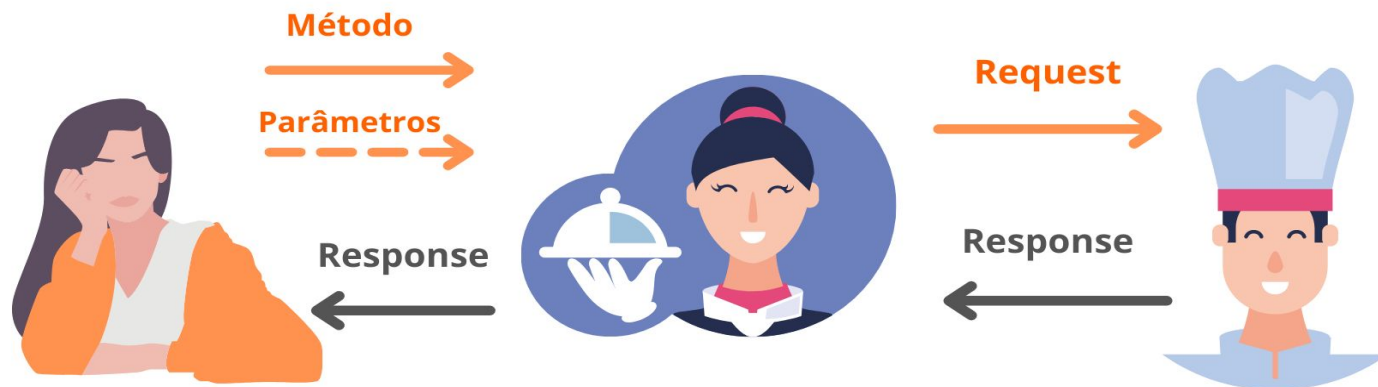
- Conjunto de **funções** disponibilizadas pelo **backend** da aplicação. Portanto, é o **back que constrói a API**.
- Descreve todos os tipos de requisição que podem ser feitos pelos clientes e o formato que devem seguir. Elas são usadas para **disponibilizar, gerenciar e armazenar dados**.
- Cada API (**pública ou privada**) tem uma **documentação** explicitando as funcionalidades existentes (exemplo: [API do Slack](#))

Como funciona uma API?

Amanda

Garçonete

Cheff



USUÁRIO

API

APP

Como usar uma API?

- Toda API deve ter uma **documentação** descrevendo as funcionalidades.
- Usamos **ferramentas** para criar as requisições

Como usar uma API?

- Para **testar e observar o funcionamento de APIs**, vamos usar um programa chamado **Postman**



API REST

- API em que as mensagens seguem o **protocolo de comunicação HTTP**
- Isso impõe um padrão a ser seguido. Esse padrão é chamado de **REST**
- Indica como funcionalidades são organizadas e acessadas

API REST - Estrutura

- O REST se baseia no **método** e no **path** para determinar qual ação será executada com o banco de dados e em relação a qual "recurso".
- Ao conjunto do **método + path**, damos o nome de **endpoint**.
- Um **recurso** é um **"tipo" de dado** que a API possui.
 - Exemplo: usuários, posts, seguidores, grupos



POST CreateUser • POST Login • PUT EditUser • GET Feriados • + ... No Environment

New Collection / Feriados **nome da coleção / nome do endpoint**

Método

GET `https://pokeapi.co/api/v2/pokemon` Path Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Headers 6 hidden

| KEY | VALUE | DESCRIPTION |
|-----|-------|-------------|
| Key | Value | Description |

Onde passará Authorization

Onde passará info do Body

ENDPOINT = Método + Path

Body Cookies Headers (26) Test Results 200 OK 135 ms 2.62 KB Save Response

Pretty Raw Preview Visualize JSON

```
1 {}
2 {"count": 1126,
3  "next": "https://pokeapi.co/api/v2/pokemon?offset=20&limit=20",
4  "previous": null,
5  "results": [
6    {
7      "name": "bulbasaur",
8      "url": "https://pokeapi.co/api/v2/pokemon/1/"
9    },
10   ]
}
```

Resposta da Requisição

Status HTTP

API REST - Estrutura do Endereço

Referência a todos os pokémons

MÉTODO `https://pokeapi.co/docs/v2/pokemon`

Método define o que será
feito com o recurso

Endereço do servidor

Recurso

Referência a um usuário específico

MÉTODO `https://pokeapi.co/docs/v2/pokemon/{id}`

Método define o que será
feito com o recurso

Endereço do servidor

Recurso

Id do
Recurso

API REST - GET

- O método **GET** é usado para **pegar informações**
- Ele **não deve** enviar um **body**
- Podemos usar a estrutura de **query** ou **path parameters** para passar informações extras

API REST - GET: Exemplos

- **Pega todos os pokémons**
 - GET <https://pokeapi.co/api/v2/pokemon/>
- **Pega pokémon com id 1**
 - GET <https://pokeapi.co/api/v2/pokemon/1>
- **Pega pokémons com limite 5**
 - GET <https://pokeapi.co/api/v2/pokemon/?limit=5>

API REST - **POST**

- O método **POST** é usado para **criar novos recursos**
- Informações do recurso criado são enviadas no **body**
- **Exemplo: Cria uma nova playlist**
 - **POST** `http://localhost:3003/playlists`
Body

```
{  
  "name": "Rock"  
}
```

API REST - PUT

- O método **PUT** é usado para **editar recursos**
- Informações do recurso criado são enviadas no **body** e o **id do recurso** como **path parameter**
- **Exemplo: Edita playlist com id 1**
 - `PUT http://localhost:3003/playlists/1`
Body

```
{  
  "nome": "Forróck"  
}
```

API REST - DELETE

- O método **DELETE** é usado para **deletar recursos**
- O **id** do recurso deve ser passado como **path parameter**
- Assim como o GET, **não possui body**
- **Exemplo: Deleta playlist com id 1**
 - **DELETE** <http://localhost:3003/playlists/1>