

Semana 6 - Aula 1

HTTPS

FUTURE⁴

Sumário

O que vamos ver hoje?

- Hoje vamos entender um artifício indispensável na vida dos programadores: Protocolos de Comunicação
- **Para isso, nós NÃO VAMOS codar hoje** 

O que vamos ver hoje?

- **Sumário**

- Backend e APIs
- Protocolos de Comunicação
- HTTP e HTTPS
- Postman

Backend

Backend



- Em projetos de aplicações Web costumamos dividir o nosso código em duas grandes classificações: Frontend e Backend
- O Frontend é a parte do projeto que estará em **contato** direto com o **usuário**.
- O Backend é a parte responsável por **gerenciar** as **informações**, persistindo os **dados**

Backend



- Responsabilidades do Backend
 - Guardar e ler as informações de um **banco de dados**
 - Montar toda a estrutura da **Lógica de negócio**
 - **Gerenciar** todos os **serviços** utilizados

Backend



Frontend



Backend



Backend



- API
 - O Backend e o Frontend precisam se comunicar
 - Isso é feito através de uma **API** (*Application Programming Interface*)
 - Isto indica qual será a **interface** pela qual outros projetos conseguem **acessar** o sistema

Backend



- API
 - Normalmente, em aplicações Web e Mobile, isto é feito através de **URL** (Uniform Resource Location)
 - São *links*

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

Protocolos

Protocolos



- O Frontend e o Backend precisam se comunicar
 - Trocar **informações/dados** entre si
- Para ter certeza de que todos os sistemas consigam se comunicar, foram criados **protocolos de comunicação**

Protocolos



- Protocolos de comunicação são, então, **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



- **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**
 - Acessar uma máquina remotamente
 - Exige algum tipo de login (usuário e senha)
- **ICMP (internet control message protocol)**
 - Usado para se fazer sanity-check
 - "Cutucar"



HTTP e HTTPS

HTTP



- **HTTP: *hypertext transfer protocol***

- É o protocolo usado como base para qualquer troca entre entidades do **World Wide Web** (www)
- Permite troca de documentos, imagens, layout, scripts

HTTP



- Nesta comunicação sempre há duas entidades:
 - Clientes (*clients*):
 - Quem **inicia** a comunicação HTTP
 - Ex.: Aplicações Web
 - Servidores (*server*):
 - Quem **recebe** a comunicação
 - Ex.: Backend

HTTP



- As mensagens enviadas pelo cliente são chamadas de **requisições/solicitações (*requests*)**
- As mensagens enviadas pelo servidor são chamadas de **respostas (*responses*)**

Composição 💪

- As requests e as responses são constituídas por:
 - **Endereço**
 - **Body**
 - **Headers**
 - **Path Param**
 - **Query String**
 - **Status**
 - Cookies
 - Cache
 - e Outros

Endereço



- Endereço é uma string que permite identificar a localização da API que queremos "bater" (utilizar)
- Normalmente, este endereço é uma **URL** (*Uniform Resource Location*)

http://minha-api.com/

Body 💪

- O body é uma estrutura que permite que enviamos **informações** na requisição
- Ela pode possuir vários formatos:
 - XML
 - Form-data
 - Raw
 - **JSON**

Body 💪

- JSON (*Javascript Object Notation*) é uma maneira de se representar objetos
- Os atributos do objeto **JSON** tem que estar entre aspas duplas
- Os valores dos atributos podem ser: **string**, **number**, **array**, **boolean**, **null** ou **outro objeto JSON**

Body 💪

- **Body:**

```
31  const user = {  
32    id: "123",  
33    age: 24,  
34    active: true,  
35    likes: ["brawstars", "radiohead", "kitten"],  
36    posts: [  
37      {  
38        image: "url1",  
39        date: "23/10/2019"  
40      },  
41      {  
42        image: "url",  
43        date: "23/10/2019"  
44      },  
45    ]  
46 }
```

```
1  {  
2    "id": "123",  
3    "age": 24,  
4    "active": true,  
5    "likes": ["brawstars", "radiohead", "kitten"],  
6    "posts": [  
7      {  
8        "image": "url1",  
9        "date": "23/10/2019"  
10       },  
11       {  
12         "image": "url2",  
13         "date": "23/10/2019"  
14       }  
15     ]  
16   }
```

Headers 💪

- **Headers** são informações adicionais que passamos na requisição
- Podem ter qualquer finalidade
- **Token de Autenticação**
 - É um header específico que permite identificar qual o usuário em questão

Headers 💪

- **Content-Type:**
 - É um *header* específico que deve ser informado **sempre** que a requisição for um **body**
 - As requisições que formos fazer sempre serão compostos por JSON

Content-Type: application/json

Antes de descansar...



- PONTOS IMPORTANTES
 - O frontend se comunica com o backend a partir da URL de uma API
 - HTTP é o protocolo utilizado para esta comunicação em projetos **WEB**
 - **Body** permite enviar informações. Normalmente usamos **JSON**
 - **Headers** transferem outras informações, como **token de autenticação** e **Content-Type**

Pausa para relaxar 😴

10 min

Antes de descansar...



- PONTOS IMPORTANTES
 - O frontend se comunica com o backend a partir da URL de uma API
 - HTTP é o protocolo utilizado para esta comunicação em projetos **WEB**
 - **Body** permite enviar informações. Normalmente usamos **JSON**
 - **Headers** transferem outras informações, como **token de autenticação** e **Content-Type**

Parâmetros na URL 💪

- Conseguimos, ainda, passar alguns **parâmetros** diretamente na URL
- Esses parâmetros se dividem em: Path Parameter (Param) e Query String

Path Parameter (Param) 💪

- São parâmetros passados na URL separados por /

http://minha-api.com/users/12345

Query String



- São parâmetros passados na URL que possuem uma **chave** e um **valor**, neste formato:

chave=valor

- Devem ser iniciados por um **?** e separado por um **&**

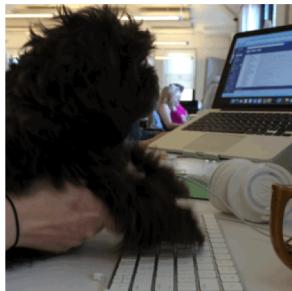
http://minha-api.com/users?name=Joao&age=20

Status

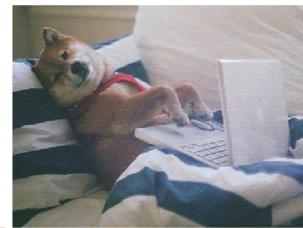


- **Código de Status** é uma maneira que utilizamos para entender **se** uma requisição http foi, ou não, completada com **sucesso**
- Os status são classificados pela centena:
 - **200:** sucesso
 - **300:** redirecionado
 - **400:** erro
 - **500:** erro não programado

Status 💪



Exemplos



FUTURE4

Métodos HTTP →

- **HTTP Methods/Verbs** é o que indica a ação que o servidor deve tomar em cima dos dados recebidos
- Estes métodos podem ser classificados levando em consideração:
 - Objetivo da requisição;
 - Requisição possui um corpo (*body*)
 - Resposta de sucesso possui um corpo (*body*)

Métodos HTTP →

- **Os tipos que vamos ver:**
 - GET
 - POST
 - PUT
 - DELETE

Métodos HTTP →

GET

- Solicita informações sobre um recurso especificado. Devem ser usadas só para **pegar dados**

Request com body	NÃO
Response de sucesso com body	SIM

Métodos HTTP →

PUT

- Envia informações para o servidor com o objeto de **adicionar** ou **alterar informações**.

Request com body	SIM
Response de sucesso com body	NÃO

Métodos HTTP →

POST

- **Envia informações** para o servidor que podem resultar em diversas ações

Request com body	SIM
Response de sucesso com body	SIM

Métodos HTTP →

DELETE

- **Deleta** o recurso especificado

Request com body	NÃO
Response de sucesso com body	NÃO

HTTPS



- A maioria dos protocolos são antigos, então eles **evoluem** bastante
- Vários protocolos tem uma **versão S** (FTPS, IMAPS, **HTTPS**), que significa que tem uma **camada extra de segurança**

HTTPS



- **Encripta** os dados antes de mandar
- **Decriptua** depois de mandar
- Em outras palavras: Imagine que alguém tente monitorar a atividade do seu Wi-Fi. Nas requisições HTTP, ele conseguiria ver todas as **informações** que você envia ou recebe; enquanto no **HTTPS, a maioria delas está criptografada**.

Antes de descansar...



- PONTOS IMPORTANTES

- **Path Param** é passado na **URL** e separado por **/**
- **Query Param** também é passado na **URL**; possui uma **key** (chave), um **value** (valor) e são separados por **&**
- **Status Code** permite que identificamos se houve **sucesso** na requisição.
- **GET, PUT, POST** e **DELETE**

Pausa para relaxar 😴

10 min

Postman

FUTURE⁴

Postman



- Postman é um **programa** que utilizamos muito para testar as APIs
- Ele permite fazer **todos os tipos** de requisição HTTPS
- É uma **interface bem amigável**

Postman



- Vamos utilizar uma API própria que fizemos
- Ela se chama: **future4users**
- E sua documentação se encontra aqui:

<https://documenter.getpostman.com/view/4233568/SVtWw7Mt?version=latest>

Postman



- **future4users:**

- É uma API que permite fazer as seguintes operações:
 - criar usuário
 - editar usuário
 - deletar usuário
 - procurar usuário
 - pegar as informações de um usuário
 - ver todos os usuários



- **Exercício 1: Cadastrando um novo usuário**
 - Vamos criar um usuário com os seguintes dados:
 - nome: Miau
 - email: miau@gmail.com

**Vamos usar o endpoint POST -
/users/createUser**

Postman



- **Exercício 1: Cadastrando um novo usuário**
 - Como sabemos se deu certo?
 - Verificando o status code
 - 200 normalmente representa sucesso
 - Podemos fazer outra chamada para verificar se o usuário foi efetivamente criado

Vamos usar o endpoint GET - /users/getAllUsers

Postman



- **Exercício 2: Alterando as informações**
 - Vamos alterar os dados do nosso usuário:
 - novo email: meowth@gmail.com

Vamos usar o endpoint POST - /users/editUser



- **Exercício 2: Alterando as informações**
 - Para validar:
 - Vamos pegar as informações do nosso usuário.

Vamos usar o endpoint GET - /users/getUser

Postman



- **Exercício 3: Pesquisar o nosso usuário**
 - Vamos procurar o nosso usuário:
 - Utilizando o endpoint de pesquisa. Vamos utilizar tanto o nome como o email

Vamos usar o endpoint GET - /users/searchUsers

Postman



- **Exercício 4: Apagar o nosso usuário**
 - Vamos deletar as informações do nosso usuário

Vamos usar o endpoint DELETE - /users/deleteUser

Postman



- **Exercício 4: Apagar o nosso usuário**
 - Vamos validar de duas formas:
 - 1. Vamos pegar a lista de todos os usuários.

Vamos usar o endpoint GET - /users/getAllUsers

Postman



- **Exercício 4: Apagar o nosso usuário**
 - Vamos validar de duas formas:
 - 2. Vamos tentar pegar as informações do nosso usuário, usando o id que tinhhamos antes.

Vamos usar o endpoint GET - /users/getUser

Resumo

- O frontend se **comunica** com o backend a partir da **URL** de uma **API**
- HTTP é o protocolo utilizado para esta comunicação em projetos **WEB**
- **Body** permite enviar informações. Normalmente usamos **JSON**

Resumo

- **Headers** transferem outras informações, como **token de autenticação** e **Content-Type**
- **Path Param** é passado na **URL** e separado por /
- **Query Param** também é passado na **URL**; possui uma **key** (chave), um **value** (valor) e são separados por &

Resumo

- **Status Code** permite que identificamos se houve **sucesso** na requisição
- **GET, PUT, POST e DELETE**
- **HTTPS** é uma versão do HTTP que possui camadas de segurança
- **Postman** é um programa que utilizamos bastante para testar APIs

Dúvidas?

Obrigado!