



PROF. RENÊ XAVIER

DESENVOLVIMENTO PARA IOS 11 COM SWIFT 4

ONDE ENCONTRAR O MATERIAL?

[HTTPS://GITHUB.COM/RENEFX/IOS-2018-01](https://github.com/renefx/ios-2018-01)

GITHUB
ALÉM DO TRADICIONAL BLACKBOARD DO IESB

O QUE VAMOS FAZER HOJE?

AGENDA

- ▶ Introdução
- ▶ Postman
- ▶ Mockoon
- ▶ Comunicação com o servidor





INTRODUÇÃO

INTRODUÇÃO

- ▶ O que é HTTP?
 - ▶ Hypertext Transfer Protocol (HTTP)
 - ▶ É um protocolo, assim como no Swift, um protocolo é um acordo
 - ▶ No caso do Swift é um acordo entre classes (lembre-se Delegate)
 - ▶ No caso do HTTP é um acordo entre cliente e servidor
 - ▶ O cliente pede dados ao servidor de uma forma específica (request)
 - ▶ O servidor envia eles de uma forma específica (response)
 - ▶ Servidor - Aquele local na rede que tem as informações que queremos
 - ▶ Cliente - Tudo que tenta acessar o servidor (App, Web, Desktop)

INTRODUÇÃO

- ▶ A **request** do cliente contém:
 - ▶ Uma URL
 - ▶ Método (forma) da requisição
 - ▶ Header (cabeçalho)
 - ▶ Body (corpo) - dependendo do método possui um Body ou não
- ▶ A **response** do servidor contém:
 - ▶ Status code - Código informando como que se deu a requisição
 - ▶ Header (cabeçalho)
 - ▶ Body (corpo) - dependendo do método possui um Body ou não

INTRODUÇÃO

- ▶ A URL da **request** não tem segredo, é o link de acesso para o recurso que você quer obter
- ▶ Dependendo do servidor, ele pode aceitar o envio de parâmetros na URL mesmo
 - ▶ O que não é muito seguro
 - ▶ Segue o padrão:

```
www.site.com/api.php?senha=123&usuario=usuario123
```

- ▶ A forma mais correta é enviar parâmetros no body

INTRODUÇÃO

- ▶ No protocolo (acordo) HTTP foi estabelecido que qualquer um que quisesse se comunicar com um servidor poderia fazê-lo com os seguintes **métodos**:
- ▶ GET
- ▶ POST
- ▶ PUT
- ▶ HEAD
- ▶ DELETE
- ▶ OPTIONS
- ▶ PATCH, TRACE, CONNECT, COPY, LINK, UNLINK, PURGE

INTRODUÇÃO

- ▶ No protocolo (acordo) HTTP foi estabelecido que qualquer um que quisesse se comunicar com um servidor poderia fazê-lo com os seguintes métodos:
 - ▶ GET - Requisitar dados do servidor
 - ▶ POST - Enviar dados para o servidor podendo alterar o dado anterior
 - ▶ PUT - Enviar dados ao servidor sempre sobreescrevendo o dado anterior
 - ▶ HEAD - Semelhante ao Get, mas não retorna os dados, só o cabeçalho
 - ▶ DELETE - Deleta um dado do servidor
 - ▶ OPTIONS - Retorna os métodos HTTP suportados para aquela URL

INTRODUÇÃO

- ▶ O header contém informações pertinentes de quem está enviando o header
- ▶ O tipo do conteúdo que está sendo enviado
- ▶ A linguagem aceita
- ▶ O agente que está enviando esse header
- ▶ Outras informações

INTRODUÇÃO

- ▶ O body é onde são enviados e recebidos os dados que nos interessam
- ▶ O formato do body é determinado pelo header **Content-Type**
 - ▶ O que vimos que envia na url é o **application/x-www-form-urlencoded**
 - ▶ Para comunicação por XML **application/xml** ou **text/xml**
 - ▶ Para comunicação por HTML **text/html**
 - ▶ Para comunicação por Javascript **application/javascript**
 - ▶ Para comunicação por texto puro **text/plain**
 - ▶ Para comunicação por JSON **application/json**
 - ▶ A forma mais comum para Apps é por JSON

INTRODUÇÃO

```
{  
  "artists" : [  
    {  
      "artistname" : "Leonard Cohen",  
      "born" : "1934"  
    },  
    {  
      "artistname" : "Joe Satriani",  
      "born" : "1956"  
    },  
    {  
      "artistname" : "Snoop Dogg",  
      "born" : "1971"  
    }  
  ]  
}
```

- ▶ { } - indicam objetos
- ▶ [] - indicam arrays
- ▶ [{}, {}] - arrays de objetos
- ▶ Conceito chave valor
 - ▶ Para a chave Artistas
 - ▶ Tem um array de objetos
 - ▶ Assim que objetos são trafegados na rede

INTRODUÇÃO

HTTP Method ↴	RFC ↴	Request Has Body ↴	Response Has Body ↴	Safe ↴	Idempotent ↴	Cacheable ↴
GET	RFC 7231 ↗	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 7231 ↗	No	No	Yes	Yes	Yes
POST	RFC 7231 ↗	Yes	Yes	No	No	Yes
PUT	RFC 7231 ↗	Yes	Yes	No	Yes	No
DELETE	RFC 7231 ↗	No	Yes	No	Yes	No
CONNECT	RFC 7231 ↗	Yes	Yes	No	No	No
OPTIONS	RFC 7231 ↗	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231 ↗	No	Yes	Yes	Yes	No
PATCH	RFC 5789 ↗	Yes	Yes	No	No	No

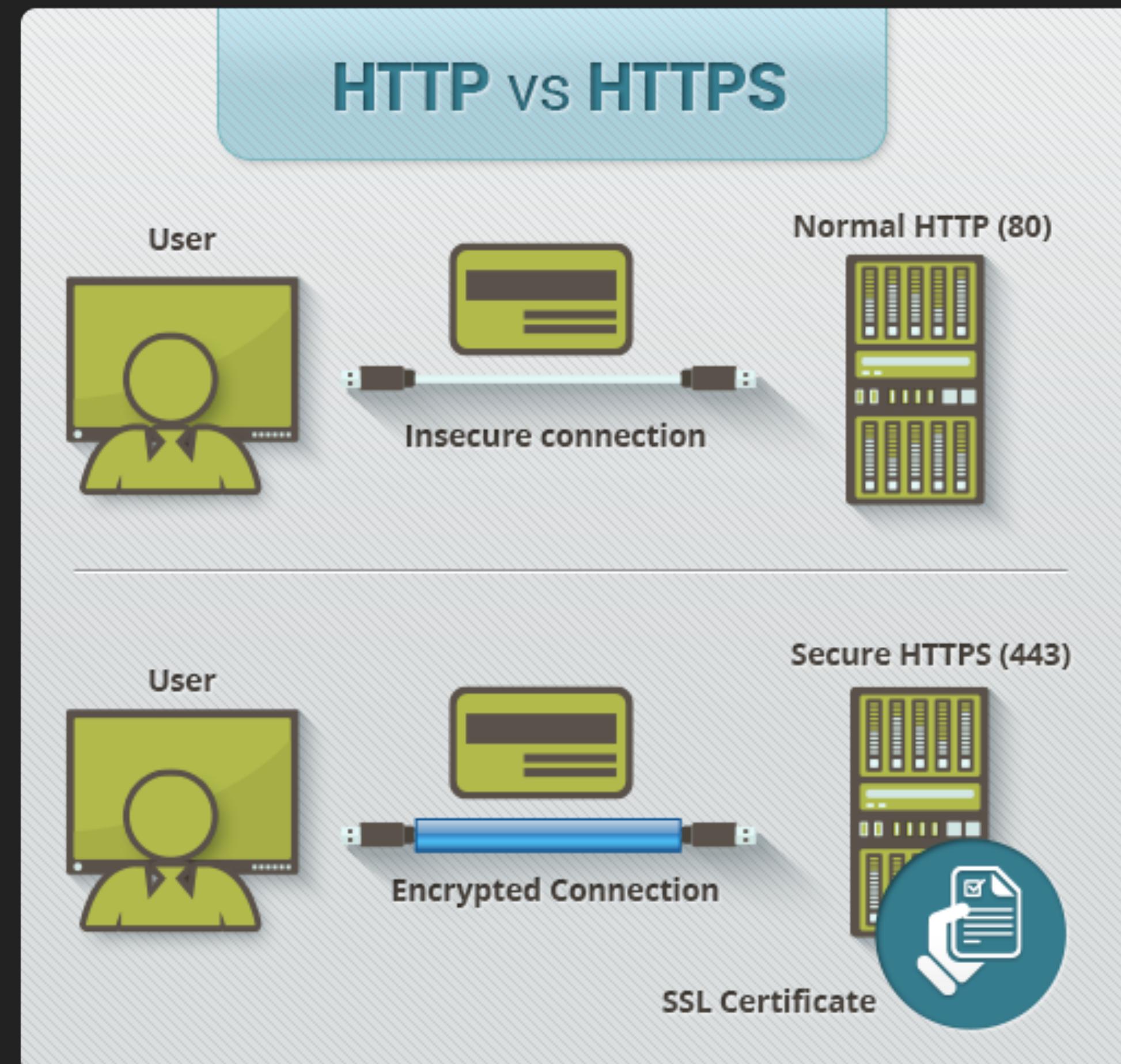
- ▶ **Idempotent** - Ao executar mais de uma vez a mesma request, o server permanece no mesmo estado
- ▶ **Safe** - Ao executar a request, ela não altera o server
- ▶ **Cacheable** - Requisições que podem ficar na memória para um uso futuro

INTRODUÇÃO

- ▶ A response do servidor tem um código:
 - ▶ Informational 1XX
 - ▶ Trazem uma informação
 - ▶ Successful 2XX
 - ▶ Informam sucesso, sendo o 200 sucesso completo sem ressalvas
 - ▶ Redirection 3XX
 - ▶ Informam um redirecionamento. Ex.: 301 - a página foi mudada para outra URL
 - ▶ Client Error 4XX
 - ▶ Erros no cliente, em outras palavras, erro na requisição feita. Seja por falta de usuário e senha (401), envio de um formato de media que não é suportado (415) ou qualquer outro erro na requisição será notificado.
 - ▶ Server Error 5XX
 - ▶ O servidor pode estar fora do ar (503) ou até mesmo ter ocorrido um erro de processamento da sua requisição (503)

INTRODUÇÃO

- ▶ Uma requisição HTTP é insegura, é possível ver tudo que passa nela usando um programa simples como o Wireshark
- ▶ O HTTPS adiciona uma camada de segurança por meio de um certificado SSL



INTRODUÇÃO

No.	Time	Source	Destination	Protocol	Info
8	2011-12-08 10:33:28.279582	127.0.0.1	127.0.0.1	TLSv1	Client Key Exchange, Change Cipher Spec, Finished
9	2011-12-08 10:33:28.291037	127.0.0.1	127.0.0.1	TLSv1	Change Cipher Spec
10	2011-12-08 10:33:28.291107	127.0.0.1	127.0.0.1	TLSv1	Finished
11	2011-12-08 10:33:28.291183	127.0.0.1	127.0.0.1	TCP	37204 > 9094 [ACK] Seq=329 Ack=1724 Win=49408 Len:
12	2011-12-08 10:33:28.291547	127.0.0.1	127.0.0.1	HTTP	GET /ibm/console/login.do?action=secure HTTP/1.1
13	2011-12-08 10:33:28.292688	127.0.0.1	127.0.0.1	HTTP	HTTP/1.1 302 Found
14	2011-12-08 10:33:28.295413	127.0.0.1	127.0.0.1	HTTP	GET /ibm/console/logon.jsp HTTP/1.1
15	2011-12-08 10:33:28.296726	127.0.0.1	127.0.0.1	HTTP	HTTP/1.1 200 OK (text/html)
16	2011-12-08 10:33:28.336734	127.0.0.1	127.0.0.1	TCP	37204 > 9094 [ACK] Seq=1434 Ack=6852 Win=49408 Len:
17	2011-12-08 10:33:30.267266	127.0.0.1	127.0.0.1	HTTP	POST /ibm/console/j_security_check HTTP/1.1 (app)
18	2011-12-08 10:33:30.292939	127.0.0.1	127.0.0.1	HTTP	HTTP/1.1 302 Found
19	2011-12-08 10:33:30.292972	127.0.0.1	127.0.0.1	TCP	37204 > 9094 [ACK] Seq=2193 Ack=7666 Win=49408 Len:
20	2011-12-08 10:33:30.296191	127.0.0.1	127.0.0.1	HTTP	GET /ibm/console/login.do?action=secure HTTP/1.1
21	2011-12-08 10:33:30.296550	127.0.0.1	127.0.0.1	TCP	9094 > 37204 [ACK] Seq=7666 Ack=2174 Win=40704 Len:

► Frame 17 (827 bytes on wire, 827 bytes captured)

► Linux cooked capture

Frame	Decrypted SSL data (738 bytes)
0290	65 6e 63 6f 64 65 64 0d 0a 43 6f 6e 74 65 6e 74 encoded. .Content
02a0	2d 4c 65 6e 67 74 68 3a 20 35 31 0d 0a 0d 0a 6a -Length: 51....j
02b0	5f 75 73 65 72 6e 61 6d 65 3d 77 73 61 64 6d 69 _username=wadmin
02c0	6e 26 6a 5f 70 61 73 73 77 6f 72 64 3d 77 73 61 n&j_pass word=wadmin&act ion=Log+in
02d0	64 6d 69 6e 26 61 63 74 69 6f 6e 3d 4c 6f 67 2b
02e0	69 6e

INTRODUÇÃO

*Wi-Fi [Wireshark 1.12.3 (v1.12.3-0-gbb3e9a0 from master-1.12)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: ip.addr == jimshaver.net Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
18957	117.562181000	jimshaver.net	192.168.0.104	TCP	1514	[TCP segment of a reassembled PDU]
18958	117.562183000	jimshaver.net	192.168.0.104	TCP	1514	[TCP segment of a reassembled PDU]
18959	117.562184000	jimshaver.net	192.168.0.104	TCP	1514	[TCP segment of a reassembled PDU]
18960	117.562185000	jimshaver.net	192.168.0.104	TLSv1.2	580	[SSL segment of a reassembled PDU]
18963	117.562559000	192.168.0.104	jimshaver.net	TCP	54	56336-https [ACK] Seq=675 Ack=18083 win=65536 Len=0
19013	117.648256000	192.168.0.104	jimshaver.net	SSL	427	[SSL segment of a reassembled PDU]
19084	117.762578000	192.168.0.104	jimshaver.net	TCP	66	56339-https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK
19086	117.763383000	192.168.0.104	jimshaver.net	TCP	66	56340-https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK
19087	117.764263000	192.168.0.104	jimshaver.net	TCP	66	56341-https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK
19099	117.764950000	192.168.0.104	jimshaver.net	TCP	66	56342-https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK
19107	117.765639000	192.168.0.104	jimshaver.net	TCP	66	56343-https [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK
19110	117.766342000	jimshaver.net	192.168.0.104	TCP	1514	[TCP segment of a reassembled PDU]
19111	117.766344000	jimshaver.net	192.168.0.104	TCP	1514	[TCP segment of a reassembled PDU]
19112	117.766346000	jimshaver.net	192.168.0.104	TCP	1514	[TCP segment of a reassembled PDU]
19113	117.766347000	jimshaver.net	192.168.0.104	TCP	1514	[TCP segment of a reassembled PDU]
19114	117.766348000	jimshaver.net	192.168.0.104	TCP	1514	[TCP segment of a reassembled PDU]
19115	117.766349000	jimshaver.net	192.168.0.104	TCP	1514	[TCP segment of a reassembled PDU]

Frame 18960: 580 bytes on wire (4640 bits), 580 bytes captured (4640 bits) on interface 0

Ethernet II, Src: 192.168.0.1 (a0:f3:c1:85:ee:90), Dst: AsustekC_93:64:3a (40:16:7e:93:64:3a)

Internet Protocol version 4, Src: jimshaver.net (192.241.223.52), Dst: 192.168.0.104 (192.168.0.104)

Transmission Control Protocol, Src Port: https (443), Dst Port: 56336 (56336), Seq: 17557, Ack: 675, Len: 526

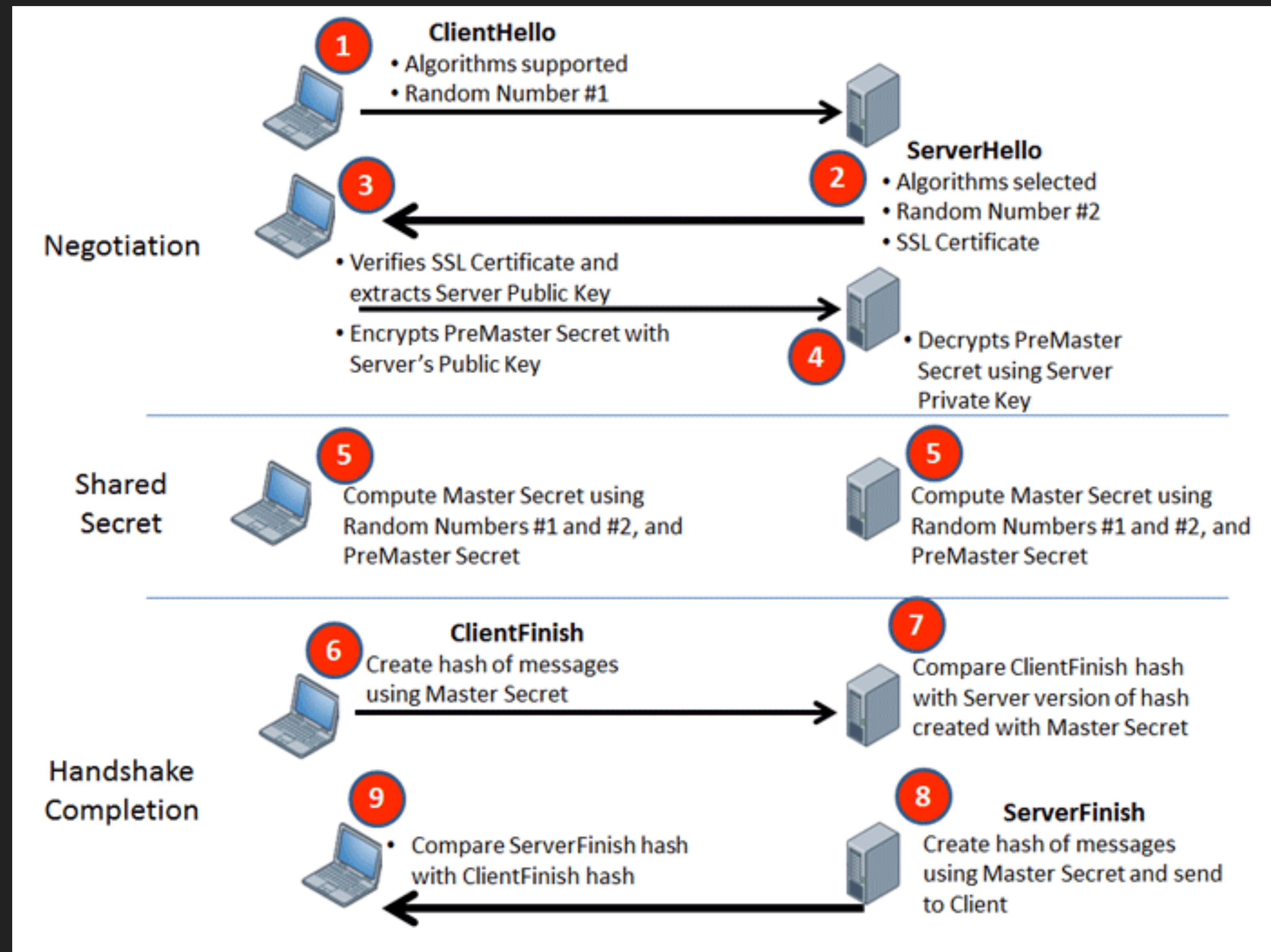
[10 Reassembled TCP Segments (13666 bytes): #18950(1460), #18951(1460), #18953(1460), #18954(1460), #18955(1460), #18956(1460), #18957(1460), #18958(1460), #18959(1460)]

Secure Sockets Layer

Frame (580 bytes) Reassembled TCP (13666 bytes) Decrypted SSL data (13637 bytes)

File: "C:\Users\elitest\AppData\Local\Temp\..." Packets: 22624 · Displayed: 2264 (10.0%) · Dropped: 0 (0.0%) Profile: Default

INTRODUÇÃO



MAIS SOBRE
SSL EM
SWIFT

INTRODUÇÃO

- ▶ WebSocket
 - ▶ Outro tipo de protocolo
 - ▶ Caso o server possua uma informação nova, não é necessário esperar o cliente perguntar, já pode enviar
 - ▶ Funciona como um telefonema
 - ▶ A fundação para real-time mobile e web Applications

INTRODUÇÃO

► REST vs WebSocket - Server

► REST

- A cada requisição é aberta uma conexão com o servidor e ao responder ela é fechada
- O servidor age de forma reativa, somente quando é requisitado

► WebSocket

- Uma conexão é aberta com o servidor e só é fechada quando o servidor ou o cliente requisitar
- No meio tempo podem ser trafagados dados
- O servidor age de forma pró-ativa, informando qualquer atualização ao cliente

INTRODUÇÃO

- ▶ REST vs WebSocket - Server
- ▶ REST
 - ▶ Usados para dados que não são atualizados sem a ação do usuário
 - ▶ Dados que não tem uma alteração constante
 - ▶ E-Commerce
 - ▶ Buscas/Pedidos
- ▶ WebSocket
 - ▶ Usados para dados atualizados **sem** a ação do usuário
 - ▶ Dados em constante alteração
 - ▶ Jogos
 - ▶ Chats
 - ▶ Playback de música/vídeo
 - ▶ Mercado financeiro

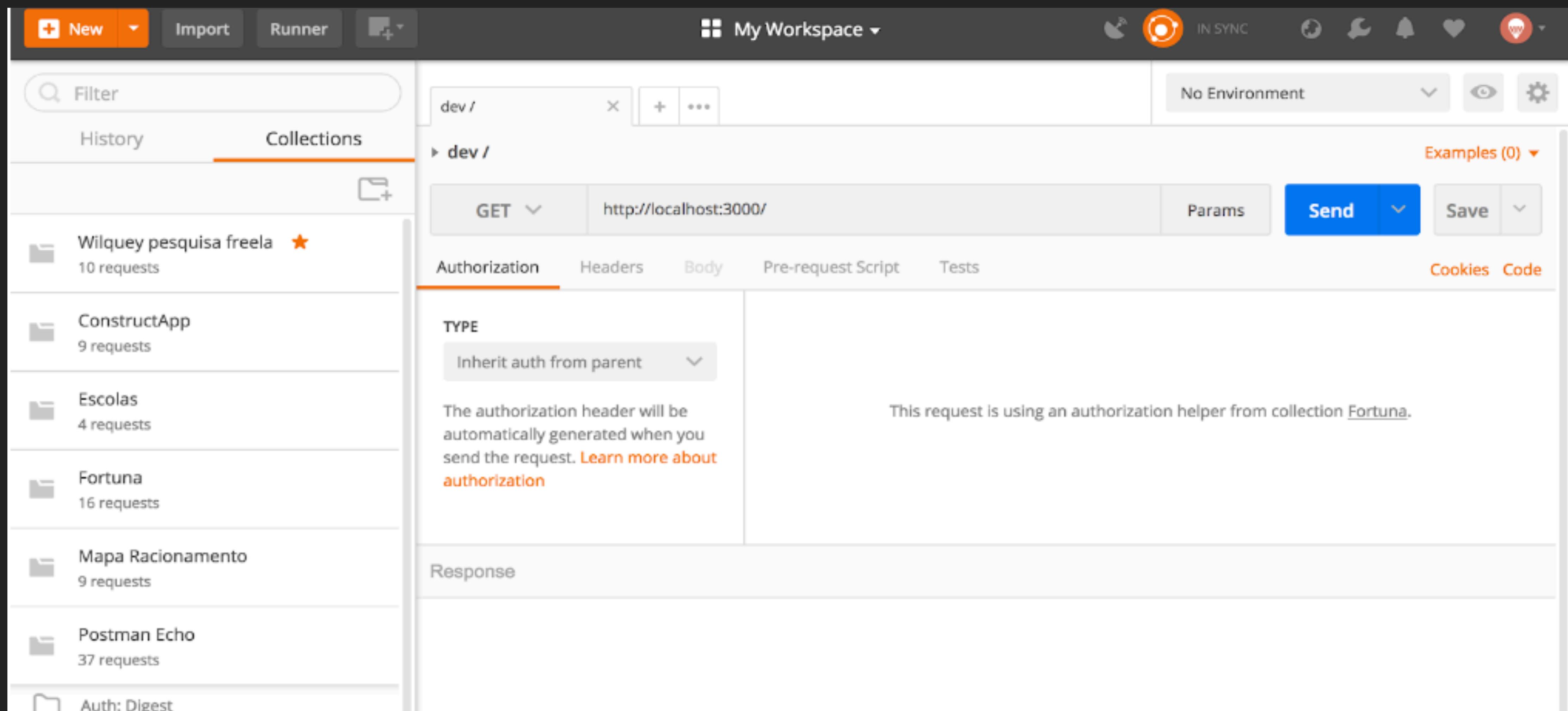
MAIS SOBRE
WEBSOCKETS EM
SWIFT



POSTMAN

POSTMAN

- ▶ Postman é uma ferramenta para testar requisições a um servidor já existente
- ▶ É ótimo e muito usado para saber se o problema está no app ou no servidor



POSTMAN

- ▶ O Postman permite:
 - ▶ Executar requisições HTTP dos diversos tipos (POST, GET, PUT, DELETE...);
 - ▶ Enviar parâmetros no Header da requisição;
 - ▶ Passar argumentos no Body da requisição;
 - ▶ Passar argumentos na própria URL;
 - ▶ Gerenciar a forma de autenticação (nenhuma, OAuth 1.0, OAuth 2.0...);
 - ▶ Gerenciar a forma de cache da requisição;
 - ▶ Executar testes nas requisições;
 - ▶ Executar uma requisição de tempos em tempos para verificar a performance e o retorno.

POSTMAN

- ▶ Para executar um GET geralmente não é preciso de muita coisa além da URL

The screenshot shows the Postman application interface. At the top, there is a header bar with a dropdown menu set to "GET", a URL input field containing "https://postman-echo.com/get?test=123", and a "Send" button. The "Send" button and the "GET" dropdown are both highlighted with red boxes.

Below the header, there are tabs for "Authorization", "Headers", "Body", "Pre-request Script", and "Tests". The "Headers" tab is currently selected and highlighted with an orange bar. The "Body" tab is also visible below it.

In the main content area, there is a table for managing environment variables. The columns are "Key", "Value", and "Description". A new row is being added, with "New key" in the Key column and "Value" in the Value column. There is also a "Description" column and a "Bulk Edit" button.

At the bottom of the interface, there are tabs for "Body", "Cookies (1)", "Headers (9)", and "Test Results (6/6)". The "Body" tab is selected. To the right of these tabs, the status is shown as "Status: 200 OK" and "Time: 179 ms".

The bottom section displays the JSON response from the API call. The response is a single object with the following structure:

```
1 {  
2   "args": {  
3     "test": "123"  
4   },  
5   "headers": {  
6     "host": "postman-echo.com",  
7     "accept": "*/*",  
8     "accept-encoding": "gzip, deflate",  
9     "cache-control": "no-cache",  
10    "cookie": "sails.sid=s%3Anue561L2K39HwcU0uJj0RaWI0SgK0MaX.dJA%2B7x0%2F%2BkbJuBQG4yRNCd7K2da0S5IiYC13yc06KfU",  
11    "postman-token": "272ef7a8-e7a2-485e-8d3c-af5c5b73a66b",  
12    "user-agent": "PostmanRuntime/7.1.1",  
13    "x-forwarded-port": "443",  
14    "x-forwarded-proto": "https"  
15  },  
16  "url": "https://postman-echo.com/get?test=123"  
17}
```

POSTMAN

- ▶ Para executar um GET geralmente não é preciso de muita coisa além da URL
- ▶ Nesse caso, ele até exibe a imagem que recebemos na resposta

The screenshot shows the Postman interface with a successful API call. The request method is set to 'GET' (highlighted with a red box), and the URL is 'https://cdn.pixabay.com/user/2013/11/05/02-10-23-764_250x250.jpg'. The 'Headers' tab is selected. The response body displays a vibrant yellow and white plumeria flower against a black background.

Key	Value	Description
New key	Value	Description

Below the table, the status is shown as 'Status: 200 OK'.

Request Headers:

- Content-Type: application/json
- Accept: */*

Response Headers:

- Content-Type: image/jpeg
- Content-Length: 123456
- Date: Mon, 05 Nov 2018 02:10:23 GMT
- Server: Apache
- X-Pixabay-Cache-Status: HIT
- X-Pixabay-Image-Size: 250x250
- X-Pixabay-Image-URL: https://cdn.pixabay.com/user/2013/11/05/02-10-23-764_250x250.jpg

POSTMAN

- ▶ Aquela requisição com parâmetros na URL pode ser testada preenchendo a URL
- ▶ Clicando em Params
- ▶ Preenchendo os parâmetros
- ▶ Clicando em Send

The screenshot shows the Postman interface. At the top, there is a dropdown menu set to "GET" and a URL input field containing "www.site.com/api.php?senha=123&usuario=usuario123". To the right of the URL field is a "Params" button. Below this, there is a table with three rows. The first row has columns for "Key", "Value", and "Description". The second row contains the key "senha" with a checked checkbox, the value "123", and an empty description column. The third row contains the key "usuario" with a checked checkbox, the value "usuario123", and an empty description column.

Key	Value	Description
<input checked="" type="checkbox"/> senha	123	
<input checked="" type="checkbox"/> usuario	usuario123	

POSTMAN

► O Postman também permite usar de Autorização quando o servidor requisitar

The screenshot shows the 'Authorization' tab in Postman's configuration interface. The 'TYPE' dropdown is set to 'Digest Auth'. A note in the sidebar says: 'The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)'. Below this, there's a checkbox 'Yes, disable retrying the request' which is unchecked. A large orange 'Preview Request' button is at the bottom left. The main panel has fields for 'Username' (postman), 'Password' (*****), and a 'Show Password' checkbox which is unchecked. Under 'ADVANCED' settings, the 'Realm' field contains '{{echo_digest_realm}}', the 'Nonce' field contains '{{echo_digest_nonce}}', the 'Algorithm' field is set to 'MD5', and the 'qop' field contains 'e.g. auth-int'.

Authorization ● Headers (2) Body Pre-request Script Tests ● Cookies Code

TYPE

Digest Auth

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Yes, disable retrying the request

Preview Request

● Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Username postman

Password *****

Show Password

▼ ADVANCED

These are advanced configuration options. They are optional. Postman will auto generate values for some fields if left blank.

Realm {{echo_digest_realm}}

Nonce {{echo_digest_nonce}}

Algorithm MD5

qop e.g. auth-int

POSTMAN

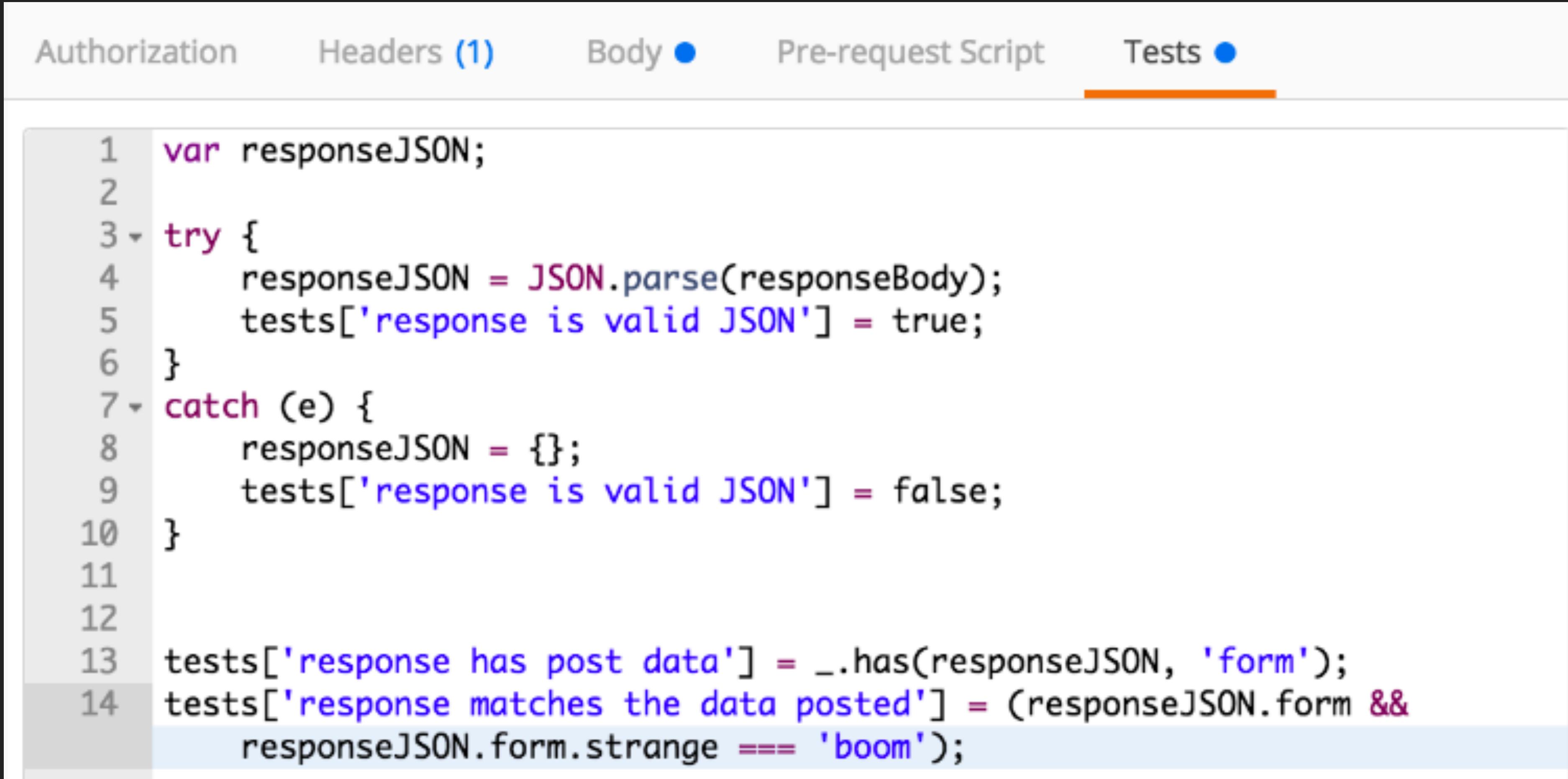
- ▶ Para fazer um POST enviando um JSON
 - ▶ Selecionamos o método POST
 - ▶ Selecionamos o Body
 - ▶ Selecionamos raw e JSON (em laranja)
 - ▶ Preenchemos o JSON com os parâmetros que iremos enviar

The screenshot shows the Postman application interface. At the top, there is a header bar with a 'POST' button (highlighted with a red box), a URL input field containing 'https://postman-echo.com/post', a 'Params' button, and a 'Send' button. Below the header, there are tabs for 'Authorization', 'Headers (1)', 'Body' (which has a blue dot indicating it is selected), 'Pre-request Script', and 'Tests'. Under the 'Body' tab, there are four radio buttons: 'form-data', 'x-www-form-urlencoded', 'raw' (highlighted with a red box), and 'binary'. To the right of these buttons is a dropdown menu set to 'JSON (application/json)' (also highlighted with a red box). At the bottom of the interface, there is a code editor window displaying the JSON payload:

```
1 {  
2   "strange": "boom"  
3 }
```

POSTMAN

- ▶ O Postman permite que façamos testes para verificar se a resposta veio da forma que queríamos
- ▶ Os testes devem ser escritos em JavaScript na aba Tests



```
1 var responseJSON;
2
3 try {
4     responseJSON = JSON.parse(responseBody);
5     tests['response is valid JSON'] = true;
6 }
7 catch (e) {
8     responseJSON = {};
9     tests['response is valid JSON'] = false;
10 }
11
12
13 tests['response has post data'] = _.has(responseJSON, 'form');
14 tests['response matches the data posted'] = (responseJSON.form &&
    responseJSON.form.strange === 'boom');
```

POSTMAN

- Após enviar, conseguimos ver os testes que passaram

The screenshot shows the 'Test Results' tab in Postman, which has a total of 3/3 tests passed. The results are displayed in a list format:

- PASS** response is valid JSON
- PASS** response has post data
- PASS** response matches the data posted

Below the results, there are tabs for Body, Cookies (1), Headers (9), and Test Results (3/3). A secondary navigation bar below the tabs includes All, Passed, Skipped, and Failed.

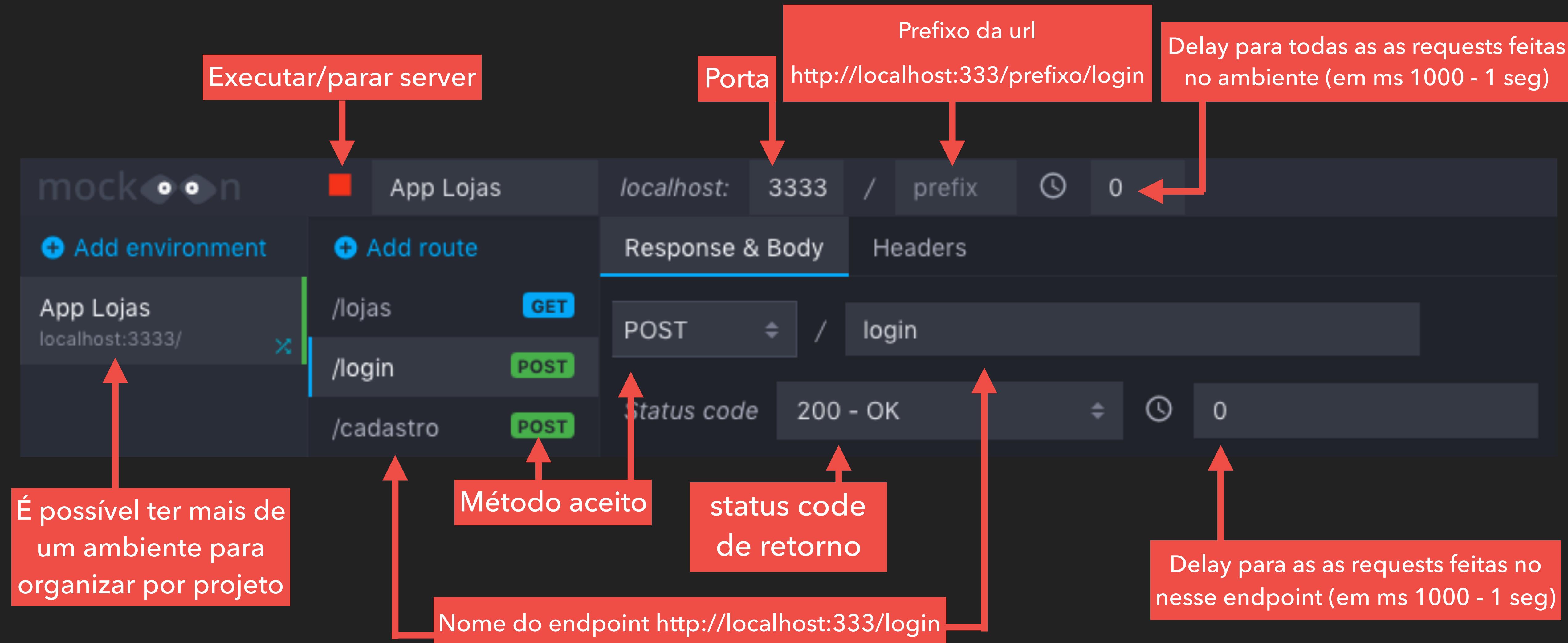


mock◊on

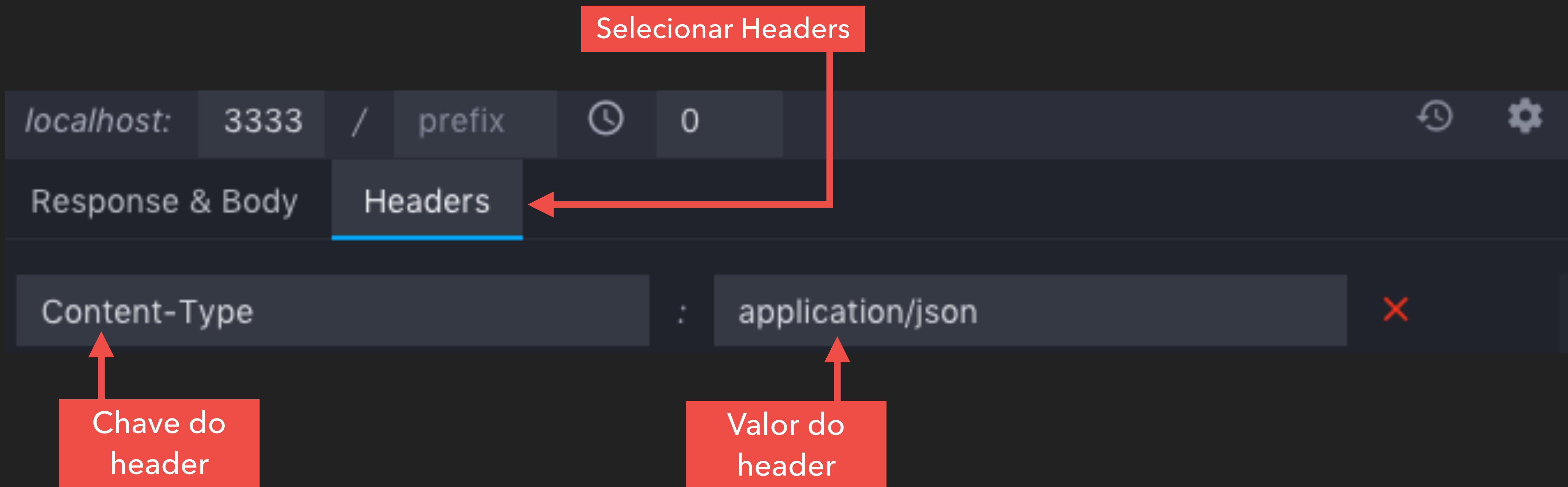
MOCKOON

MOCKOON

► Se o Postman era para testar o servidor, o Mockoon é feito para simular um servidor



- ▶ Podemos colocar Headers
- ▶ Ao colocar nosso tipo de conteúdo como JSON que temos possibilidades interessantes



- ▶ Podemos preencher o Body com um valor estático de retorno
- ▶ Como configuramos para JSON conseguimos usar um template

The screenshot shows a JSON configuration interface. On the left, there's a code editor with numbered lines (1 to 17) containing a JSON object. Lines 1 through 9 define a single object with properties like nome, iconePequeno, iconeGrande, favorita, vendeComputador, vendeJogos, and produtos. Lines 10 through 14 define an array of products under the produtos key, with each product having a titulo, imagem, and descricao. Lines 15 through 17 close the main object and the products array. On the right, there's a header labeled "Content-Type application/json - Supports templating" with a blue information icon. The entire "Content-Type" section is highlighted with a red box.

```
1 [  
2 {  
3     "nome": "Casas Bahia",  
4     "iconePequeno": "header-bahianinho",  
5     "iconeGrande": "Casas_Bahia_logo",  
6     "favorita": true,  
7     "vendeComputador": true,  
8     "vendeJogos": false,  
9     "produtos": [  
10        {  
11            "titulo": "Earpods",  
12            "imagem": "iphonex",  
13            "descricao": "Fone bluetooth da Apple"  
14        }  
15    ]  
16 }]  
17 ]
```

- ▶ Com templates podemos usar de textos, valores, nomes, booleans aleatórios
- ▶ Para indicar um template usa-se {{ }}

```
1  {
2      "nome": "{{firstName}} {{lastName}}",
3      "foto": "user-logged",
4      "email": "{{body 'email' 'erro'}}",
5      "senha": "{{body 'senha' 'erro'}}",
6      "telefone": "{{phone "(xx) XXXXX-XXXX"}}",
7      "idade": "{{int 16 45}}"
8  }
```

Um nome seguido de um sobrenome no formato de String

Valor fixo

O elemento email do body. Caso não tenha sido enviado erro

O elemento email do body. Caso não tenha sido enviado erro

Um telefone com essa máscara no formato de String

Inteiro entre 16 e 45

- ▶ Para evitar de escrever as mesmas estruturas repetidas vezes, podemos usar o `repeat`
- ▶ Nesse caso, gera 8 estruturas como essa, cada uma com 9 produtos

```
{{#repeat 8}}
{
  "nome": "{{company}}",
  "iconePequeno": "exit",
  "iconeGrande": "exit",
  "favorita": false,
  "vendeComputador": {{boolean}},
  "vendeJogos": {{boolean}},
  "produtos": [
    {{#repeat 9}}
    {
      "titulo": "{{lorem 2}}",
      "imagem": "exit",
      "descricao": "{{lorem 5}}"
    }
    {{/repeat}}
  ]
}
{{/repeat}}
```

- ▶ Ainda usamos textos aleatórios com
`{{lorem númeroDePalavras}}`

- É possível fazer um switch para ter um retorno específico para um valor enviado

```
"userName":  
  {{#switch urlParam 'id'}}  
    {{#case "1"}}"John"{{/case}}  
    {{#case "2"}}"Jack"{{/case}}  
    {{#default}}"Peter"{{/default}}  
  {{/switch}}
```

- ▶ Caso usemos variáveis que se relacionam, ele envia elas no mesmo contexto

```
"email": "{{email}}"          // michael.turner@unilogic.com
"firstName": "{{firstName}}", // Michael
"lastName": "{{lastName}}",   // Turner
```

- ▶ Para usar seus próprios dados ou criar novos helpers

LEIA A DOCUMENTAÇÃO

MOCKOON

- ▶ Podemos selecionar para no nosso Header conter uma credencial Basic
- ▶ Como podemos retornar o que vem na requisição. Podemos retornar o que vem no Header no Authorization e verificar se estamos implementando corretamente

Response & Body Headers

Content-Type : application/json X

Authorization : Basic +

```
"nome": "{{firstName}} {{lastName}}",
"foto": "user-logged",
"email": "{{body 'email' 'erro'}}",
"senha": "{{body 'senha' 'erro'}}",
"telefone": "{{phone "(xx) xxxxx-xxxx"}}",
"idade": {{int 16 45}},
"login-header": "{{header 'Authorization' 'erro'}}"
```

- ▶ No Postman podemos fazer a chamada desse endpoint passando a credencial Basic

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** http://localhost:3333/login
- Params:** (button)
- Send:** (button)
- Save:** (button)
- TYPE:** Basic Auth (selected, highlighted with a red box)
- Authorization Warning:** Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)
- Username:** renefx
- Password:** 54321
- Show Password:** (checkbox checked)
- Preview Request:** (button)

MOCKOON

- ▶ O retorno veio meio estranho....
- ▶ Isso ocorre, pois está codificado em Base64 (essa é a segurança do Basic)

```
{  
  "nome": "Layla Mercer",  
  "foto": "user-logged",  
  "email": "renefx@gmail.com",  
  "senha": "321",  
  "telefone": "(34) 99011-7593",  
  "idade": 22,  
  "login-header": "Basic cmVuZWZ40jU0MzIx"  
}
```

<HTTPS://WWW.BASE64DECODE.ORG/>

Decode from Base64 format

Simply use the form below

cmVuZWZ40jU0MzIx

< DECODE >

renefx:54321

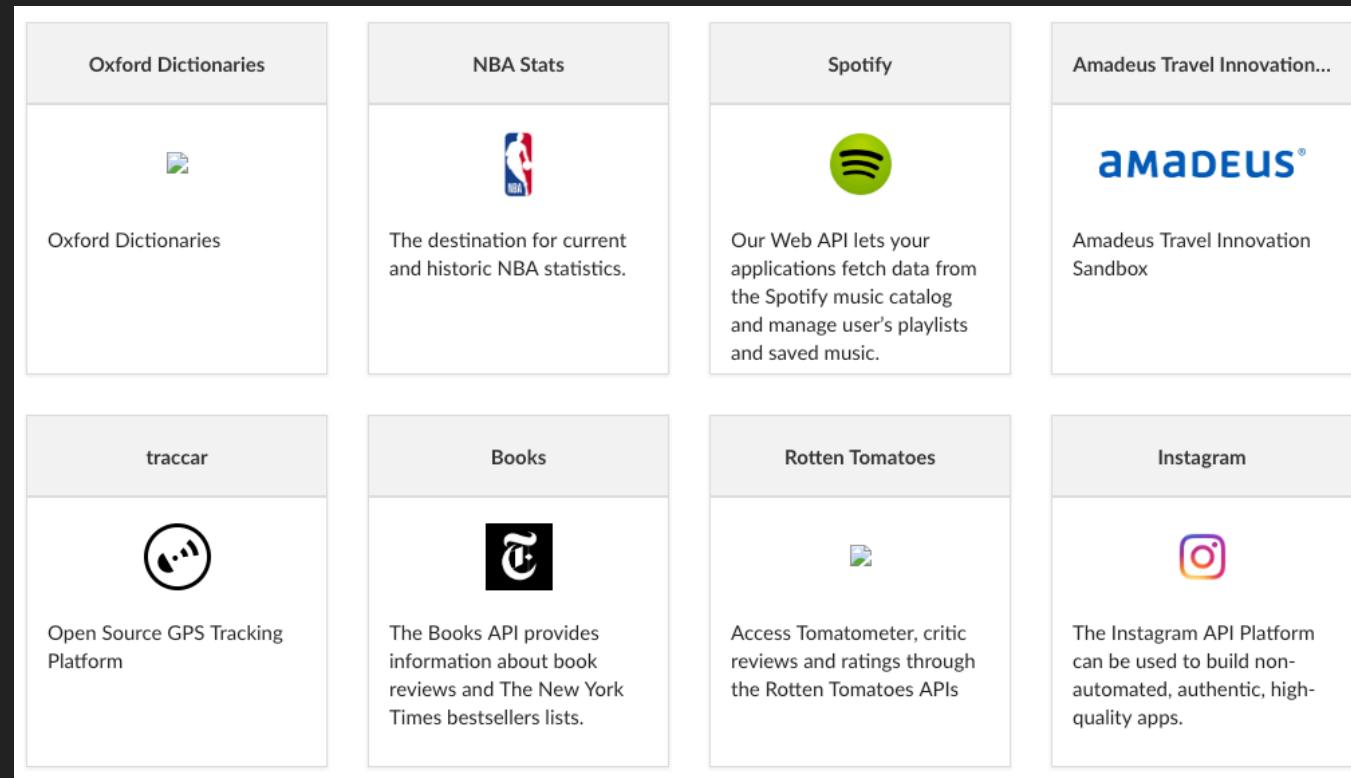
- ▶ É claro, estamos criando um Mock em nosso localhost
- ▶ Isso irá funcionar para todas aplicações que são executadas no nosso computador (Web, Postman, Emulador - Android e iOS)
- ▶ Isso não irá funcionar testando no device a menos que o computador esteja compartilhando a rede para o device
 - ▶ No 3g ou na wifi que liga diretamente ao roteador o localhost do computador não será encontrado



REQUESTS SWIFT

**PRIMEIRO,
PRECISAMOS
DE UMA URL**

PUBLIC APIs



ANY-API

Index

- Animals
- Anime
- Anti-Malware
- Art & Design
- Books
- Business
- Calendar

Animals

API	Description	Auth	HTTPS	CORS
Cats	Pictures of cats from Tumblr	No	Yes	Unknown
Dogs	Based on the Stanford Dogs Dataset	No	Yes	Yes
HTTPCat	Cat for every HTTP Status	No	Yes	Unknown
IUCN	IUCN Red List of Threatened Species	apiKey	No	Unknown
Movebank	Movement and Migration data of animals	No	Yes	Unknown

LEARN ABOUT APIs | WHAT IS AN API ? | API NEWS | API DIRECTORY

APIS (153)

Promoted Listings

	Mashups	Followers
ASCII Art	The ASCII Art API lets developers create various Ascii products from their...	0 6
Art Clinic	Art Clinic is a member of the International Society for Education through...	1 4

PUBLIC-APIS

PROGRAMMABLE WEB

REQUESTS

- Vamos utilizar uma API aberta de Crypto moedas

<https://chasing-coins.com/api>

- Vamos tentar fazer a seguinte requisição:

<https://chasing-coins.com/api/v1/std/coin/BTC>

Coin Stats

Get price, price change and more details for a specific coin.

<https://chasing-coins.com/api/v1/std/coin/BTC>

```
{  
  "change": {  
    "hour": "-0.38",  
    "day": "16.75"  
  },  
  "price": "11744.7",  
  "coinheat": 50  
}
```

REQUESTS

- Em uma ViewController nova, no método ViewDidLoad, vamos criar nossa URL:

<https://chasing-coins.com/api/v1/std/coin/BTC>

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    guard let url = URL(string: "https://chasing-coins.com/api/v1/std/coin/BTC") else {  
        return  
    }
```

TEMOS NOSSA URL!

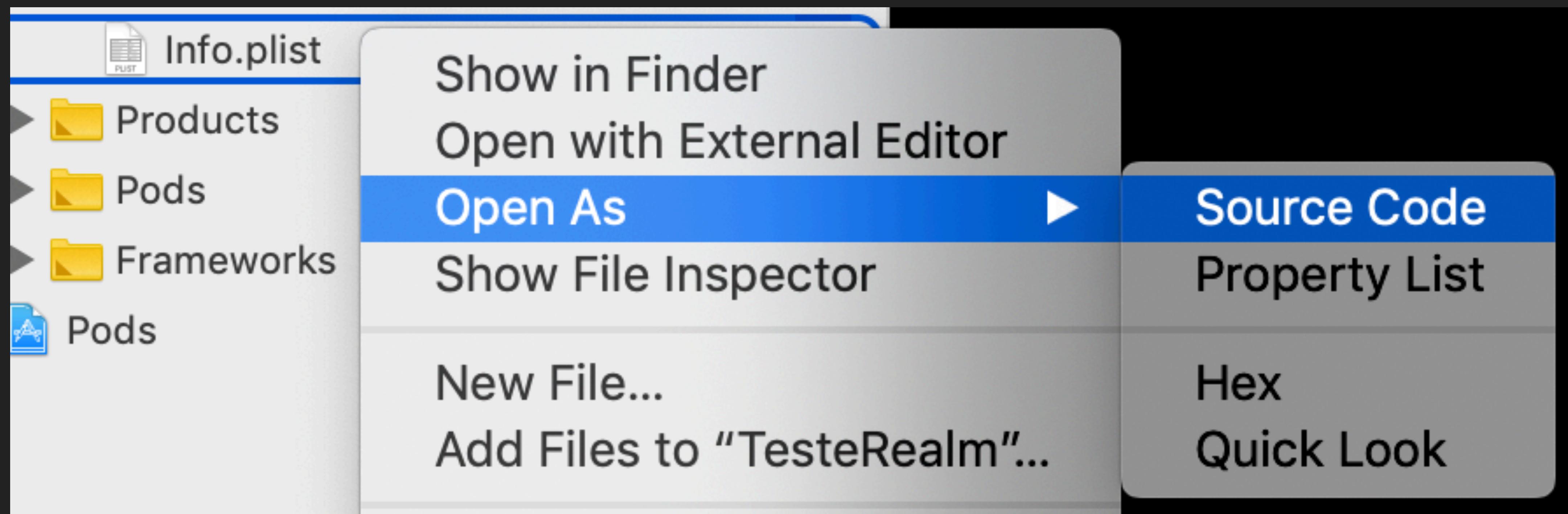
REQUESTS

UM

DETAILS

REQUESTS

- ▶ Toda conexão HTTP é **barrada** por padrão pela Apple
- ▶ **Lembrando que o Mockoon utiliza HTTP para disponibilizar o localhost**
- ▶ Como discutido, o HTTP não tem a camada de segurança do HTTPS
- ▶ Ainda assim é possível se conectar via HTTP
- ▶ Primeiro clique com o botão direito no Info.plist e selecione para abrir como Source Code



REQUESTS

- ▶ Estamos especificando que queremos acessar uma URL em específico: localhost:3333
- ▶ OBS: As duas linhas selecionadas são as duas últimas linhas já presentes no Info.plist

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <false/>
    <key>NSExceptionDomains</key>
    <dict>
        <key>localhost:3333</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
            <true/>
            <key>NSTemporaryExceptionMinimumTLSVersion</key>
            <string>TLSv1.1</string>
        </dict>
    </dict>
</dict>
</plist>
```

REQUESTS

- ▶ Podemos colocar mais de uma URL como exceção

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <false/>
    <key>NSEExceptionDomains</key>
    <dict>
        <key>randomfox.ca</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
            <true/>
            <key>NSTemporaryExceptionMinimumTLSVersion</key>
            <string>TLSv1.1</string>
        </dict>
        <key>localhost:3333</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
            <true/>
            <key>NSTemporaryExceptionMinimumTLSVersion</key>
            <string>TLSv1.1</string>
        </dict>
    </dict>
</dict>
</plist>
```

REQUESTS

VOLTANDO

REQUESTS

► Com nossa URL em mãos, precisamos agora de três coisas:

► **URLSession**

► Determina políticas de cache, timeout...

► **Task**

► Faz parte da URLSession

► Dita como aquela conexão será feita. Background, download, upload...

► **URLRequest**

► É a nossa request que fazíamos no Postman

► Ela irá receber nossa URL

► Possui todos aqueles elementos já falados

URLSESSION

REQUESTS

URLSession:

- ▶ É possível ter várias instâncias da URLSession
- ▶ Cada instância tem a sua configuração independente da outra
- ▶ São thread-safe
- ▶ Na configuração é definido:
 - ▶ Tempo de timeout
 - ▶ Política de cache
 - ▶ Determinar se a conexão deve ou não ser feita por celular
 - ▶ Outros

REQUESTS

- ▶ Para conexões simples, sem configurações, podemos acessar o Singleton da URLSession
URLSession.shared
- ▶ Para outros tipos, é possível usar uma das três configurações:
 - ▶ .default: semelhante ao Singleton, mas permite buscar dados incrementalmente usando um delegate
 - ▶ .ephemeral: Executa com cookies ou credenciais, mas não escreve no disco. Conexão segura para dados sensíveis.
 - ▶ .background(withIdentifier:): Permite fazer download e upload enquanto o App não está executando
 - ▶ https://developer.apple.com/documentation/foundation/url_loading_system/downloading_files_in_the_background
 - ▶ Customizada: crie uma própria com as suas personalizações

REQUESTS

```
let task = URLSession.shared.dataTask
```

```
let session = URLSession(configuration: .)
```

- M URLSessionConfiguration **background(withIdentifier: String)**
- M URLSessionConfiguration ~~backgroundSessionConfiguration(identifier: String)~~
- URLSessionConfiguration default
- URLSessionConfiguration ephemeral
- M URLSessionConfiguration init()

Returns a session configuration object that allows HTTP and HTTPS uploads or downloads to be performed in the background.

```
let config = URLSessionConfiguration.default
config.allowsCellularAccess = false
```

```
let session = URLSession(configuration: config)
```

REQUESTS

- ▶ Como nosso caso é somente um GET simples, vamos usar o Singleton já disponível
- ▶ Vamos entender o que é esse dataTask

```
let task = URLSession.shared.dataTask
```

REQUESTS

TASKS

REQUESTS

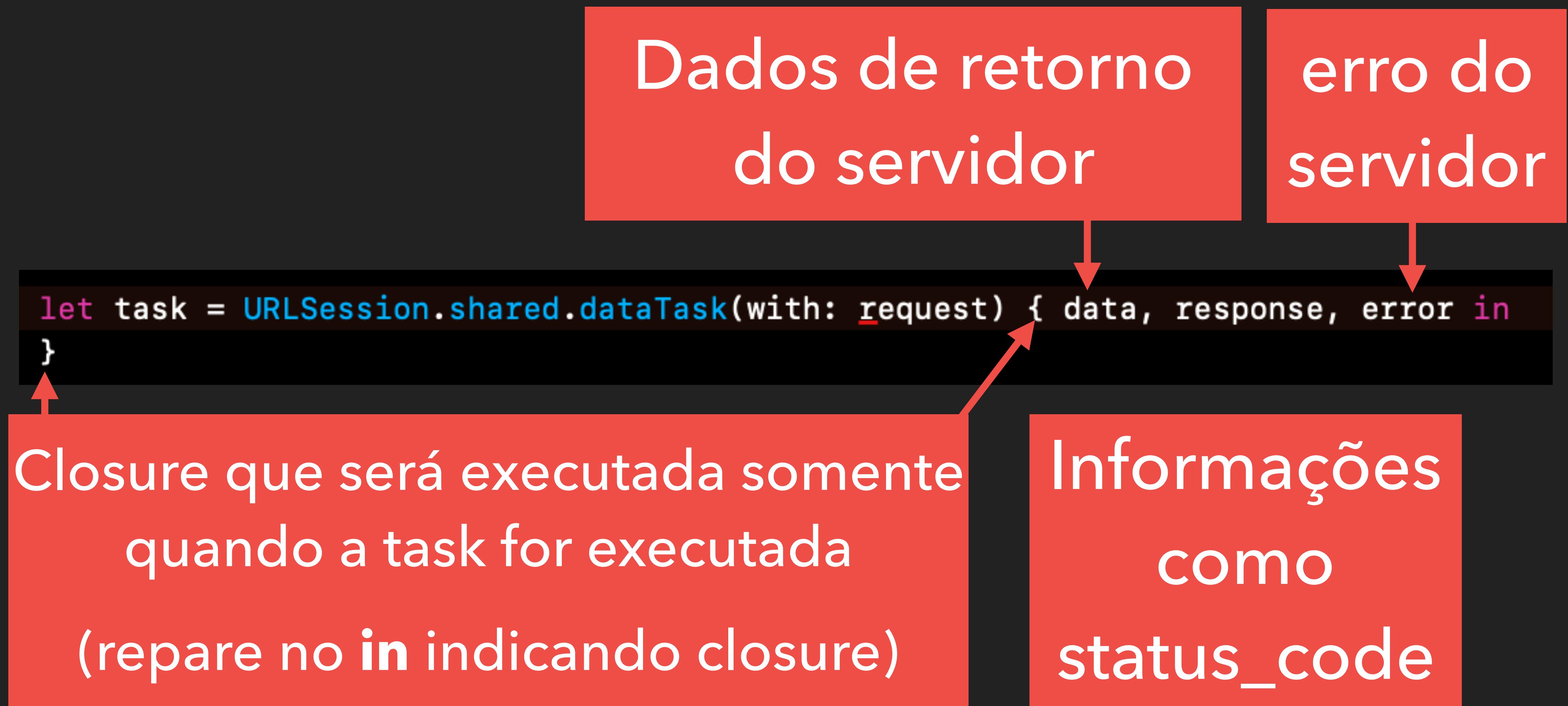
- ▶ Sessões possuem **Tasks**:
- ▶ A Task representa a ação de um recurso sendo carregado/descarregado
- ▶ SessionTasks contém toda a informação necessária para se conectar com um servidor
- ▶ Elas ditam a forma de como a conexão será feita e o que será feito após ela
- ▶ Tasks são instanciadas em um estado suspenso - elas não são executadas imediatamente
- ▶ É necessário executar o método `resume()` delas

REQUESTS

- ▶ Temos três tipos de Tasks:
 - ▶ Data task:
 - ▶ Executa os métodos que falamos (GET, POST, PUT...)
 - ▶ Upload Tasks:
 - ▶ Envia dados, principalmente como um arquivo
 - ▶ Download Tasks:
 - ▶ Recebe dados, principalmente como um arquivo
- ▶ Como as tasks de Upload e Download podem transferir arquivos muito grandes, para elas há suporte para execução em background

REQUESTS

- ▶ Abaixo vemos a nossa task criada



REQUESTS

- ▶ Ao ver ela por completo, vemos que faltam duas coisas
- ▶ Executar o método `resume()`
 - ▶ Só estamos atribuindo a uma variável `task`, para que a chamada ao servidor seja feita é necessário a execução do `task.resume()`
- ▶ Outra coisa que falta é a conexão entre a nossa **URL** e a **Task**

```
guard let url = URL(string: "https://chasing-coins.com/api/v1/std/coin/BTC") else {  
    return  
}  
  
let task = URLSession.shared.dataTask(with: request) { data, response, error in  
}
```

REQUESTS

- ▶ Com isso ajustamos a questão do resume
- ▶ Lembre-se: ao colocar uma Task em uma variável, é possível executá-la em outro momento somente executando o método resume dessa variável
- ▶ Vamos conectar nossa **URL** com nossa **Task**

```
guard let url = URL(string: "https://chasing-coins.com/api/v1/std/coin/BTC") else {  
    return  
}  
  
let task = URLSession.shared.dataTask(with: request) { data, response, error in  
}  
task.resume()
```

REQUESTS

URLREQUESTS

INTRODUÇÃO

- ▶ A **URLRequest** agrupa os elementos que falamos:
 - ▶ Uma URL - <https://chasing-coins.com/api/v1/std/coin/BTC>
 - ▶ Método (forma) da requisição - GET
 - ▶ Header (cabeçalho) - nesse caso não é necessário
 - ▶ Body (corpo) - nesse caso não é necessário

```
var request = URLRequest(url: url)
request.httpMethod = "GET"
```

INTRODUÇÃO

- ▶ A **URLRequest** também faz a conexão entre a URL, a Session e a Task
- ▶ Mas o que fazer com o retorno da Task?

```
guard let url = URL(string: "https://chasing-coins.com/api/v1/std/coin/BTC") else {  
    return  
}  
  
var request = URLRequest(url: url)  
request.httpMethod = "GET"  
let task = URLSession.shared.dataTask(with: request) { data, response, error in  
}  
task.resume()
```

INTRODUÇÃO

- Queremos obter esses dados:

```
1 {  
2   "change": {  
3     "hour": "-0.2",  
4     "day": "-1.25"  
5   },  
6   "price": "6630.17515387",  
7   "coinheat": 40  
8 }
```

INTRODUÇÃO

- ▶ Após executar a chamada ao servidor e voltar, queremos que os dados buscados não sejam nulos.
- ▶ Também queremos que não hajam erros
- ▶ Caso contrário, escrevemos o erro

```
let task = URLSession.shared.dataTask(with: request) { data, response, error in
    guard let data = data, error == nil else {
        print("error=\(error.debugDescription)")
        return
    }
}
task.resume()
```

INTRODUÇÃO

- ▶ Somente lembrando da forma que funciona a execução de uma Closure:

```
print("primeiro")
let task = URLSession.shared.dataTask(with: request) { data, response, error in
    print("terceiro")
    guard let data = data, error == nil else {
        print("error=\(error.debugDescription)")
        return
    }
    task.resume()
    print("segundo")
```

primeiro
segundo
terceiro

INTRODUÇÃO

- ▶ Não somente queremos que não tenha erro
- ▶ Queremos que o Server nos informe que tudo ocorreu da melhor forma possível - Status Code 200 - OK
- ▶ Caso contrário, nos informe o que deu errado

```
let task = URLSession.shared.dataTask(with: request) { data, response, error in
    guard let data = data, error == nil else {
        print("error=\(error.debugDescription)")
        return
    }

    if let httpStatus = response as? HTTPURLResponse, httpStatus.statusCode != 200 {
        print("\(httpStatus.statusCode)")
        print("Mensagem = \(HTTPURLResponse.localizedString(forStatusCode: httpStatus.statusCode))")
    }
}
task.resume()
```

A screenshot of Xcode code editor showing Swift code for making an HTTP request. A yellow warning icon with the text 'Immutable value 'data' was mutated' is visible next to the variable 'data'.

INTRODUÇÃO

```
let task = URLSession.shared.dataTask(with: request) { data, response, error in
    guard let data = data, error == nil else {
        print("error=\(error.debugDescription)")
        return
    }

    if let httpStatus = response as? HTTPURLResponse, httpStatus.statusCode != 200 {
        print("\(httpStatus.statusCode)")
        print("Mensagem = \(HTTPURLResponse.localizedString(forStatusCode: httpStatus.statusCode))")
    }

    do {
        let json = try JSONSerialization.jsonObject(with: data, options: .allowFragments) ⚠ Initialization
    } catch let error {
        print(error.localizedDescription)
    }
}
task.resume()
```

INTRODUÇÃO

- ▶ Como recebemos um JSON, vamos tratar ele como tal
- ▶ Não vamos mais usar ele como Data

```
do {  
    let json = try JSONSerialization.jsonObject(with: data, options: .allowFragments)  
} catch let error {  
    print(error.localizedDescription)  
}
```

INTRODUÇÃO

- ▶ Agora podemos transformá-lo em um Dicionário e acessar as suas propriedades pelo nome das chaves

```
do {  
    let json = try JSONSerialization.jsonObject(with: data, options: .allowFragments)  
    if let dionarioJson = json as? Dictionary<String, Any> {  
        print(dionarioJson)  
    }  
} catch let error {  
    print(error.localizedDescription)  
}
```

INTRODUÇÃO

- ▶ Agora podemos transformá-lo em um Dicionário e acessar as suas propriedades pelo nome das chaves

```
do {  
    let json = try JSONSerialization.jsonObject(with: data, options: .allowFragments)  
    if let dicionarioJson = json as? Dictionary<String, Any> {  
        print(dicionarioJson["price"])  
        print(dicionarioJson["coinheat"])  
        print(dicionarioJson["change"])  
    }  
} catch let error {  
    print(error.localizedDescription)  
}
```

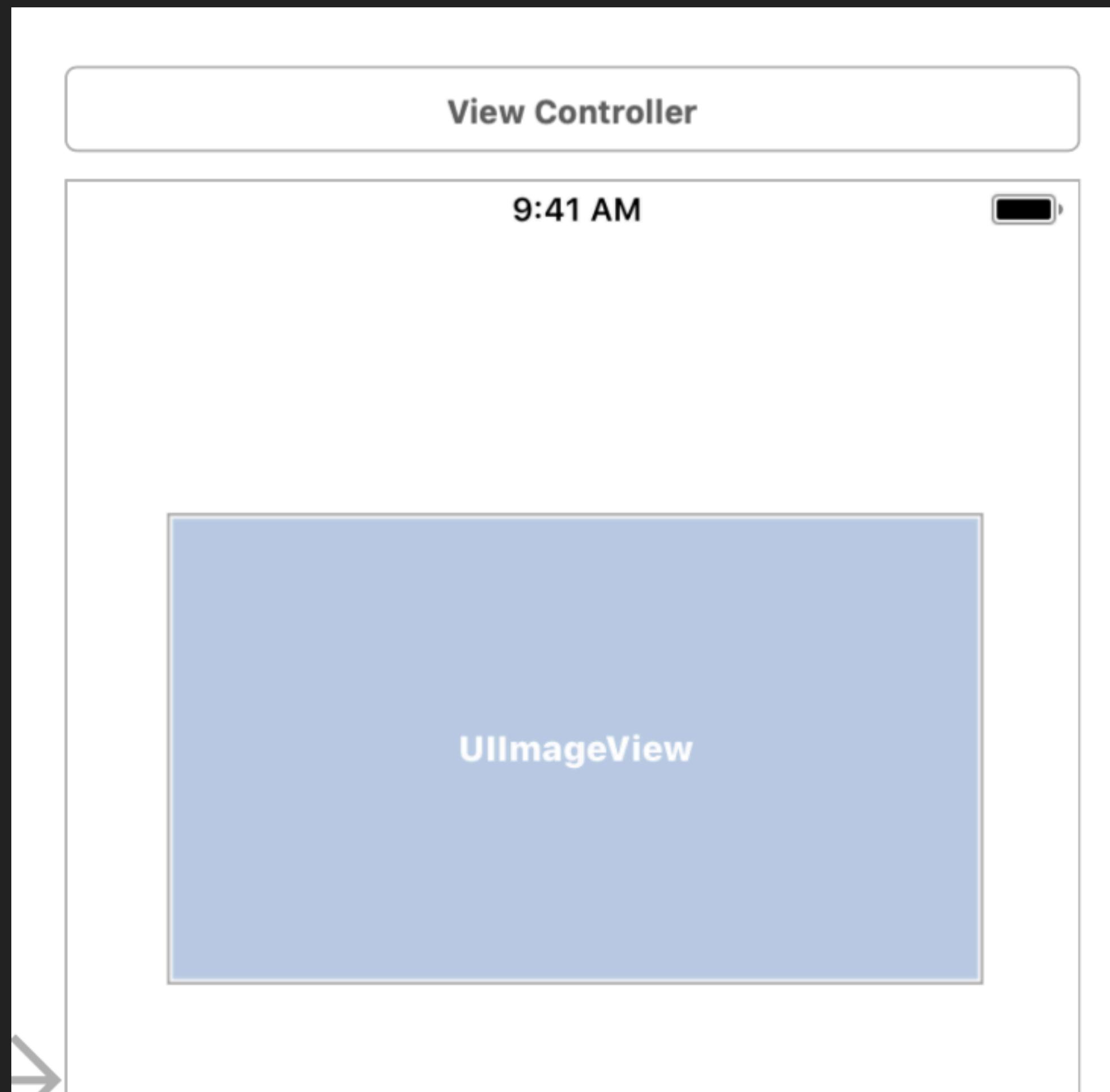
```
Optional(6611.22568309)  
Optional(22)  
Optional({  
    day = "-0.34";  
    hour = "-0.35";  
})
```



TRABALHANDO COM
IMAGENS

REQUESTS IMAGENS

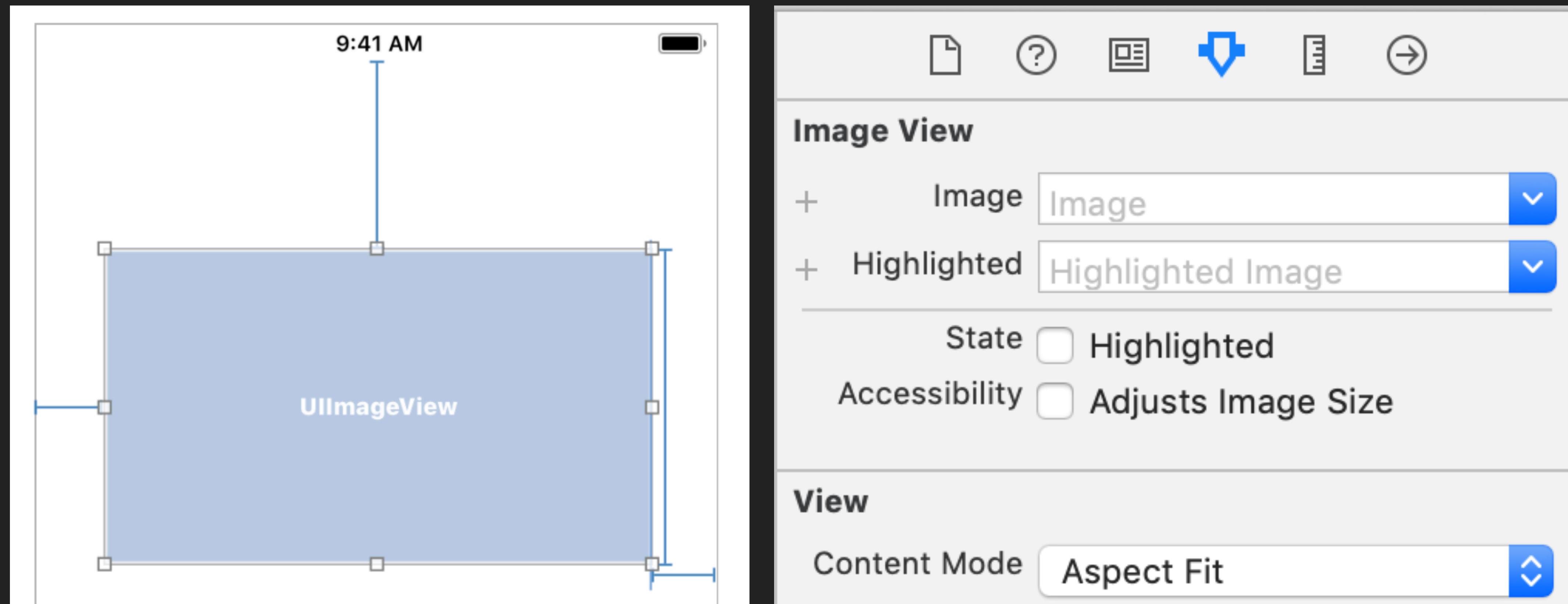
- ▶ Para trabalhar com imagens, vamos criar uma ViewController com uma UIImageView
- ▶ Vamos criar uma IBOutlet para essa UIImageView que queremos preencher



```
2 // ViewController.swift
3 // TesteRealm
4 //
5 // Created by Rene X on 26/09/18.
6 // Copyright © 2018 Rene X. All rights reserved.
7 //
8
9 import UIKit
10 import RealmSwift
11
12 class ViewController: UIViewController {
13     @IBOutlet weak var foto: UIImageView!
14
15     override func viewDidLoad() {
16         super.viewDidLoad()
17     }
18
19 }
20
21
22
23 |
```

REQUESTS IMAGENS

- ▶ Coloque as Constraints
- ▶ Selecione Aspect Fit



REQUESTS IMAGENS

- ▶ Uma imagem, um site (HTML) nada mais é do que uma requisição GET
 - ▶ No caso de um site, o retorno no body é um HTML
 - ▶ No caso de uma imagem, o retorno são os bytes que compõem a imagem
- ▶ Vamos pegar uma URL que será a imagem ao lado
- ▶ Lembre-se de adicionar randomfox.ca no info.plist



```
guard let url = URL(string: "http://randomfox.ca/images/32.jpg") else {  
    return  
}
```

REQUESTS IMAGENS

- Vamos fazer uma requisição GET como já fizemos dentro do ViewDidLoad

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    guard let url = URL(string: "http://randomfox.ca/images/32.jpg") else {  
        return  
    }  
  
    var request = URLRequest(url: url)  
    request.httpMethod = "GET"  
    let task = URLSession.shared.dataTask(with: request) { data, response, error in  
        guard let data = data, error == nil else {  
            print("error=\(String(describing: error))")  
            return  
        }  
  
        if let httpStatus = response as? HTTPURLResponse, httpStatus.statusCode != 200 {  
            print("\(httpStatus.statusCode)")  
            print("response = \(String(describing: response))")  
        }  
    }  
    task.resume()  
}
```

REQUESTS IMAGENS

- Vamos fazer uma requisição GET como já fizemos dentro do ViewDidLoad

```
override func viewDidLoad() {  
    super.viewDidLoad()  
  
    guard let url = URL(string: "http://randomfox.ca/images/32.jpg") else {  
        return  
    }  
  
    var request = URLRequest(url: url)  
    request.httpMethod = "GET"  
    let task = URLSession.shared.dataTask(with: request) { data, response, error in  
        guard let data = data, error == nil else {  
            print("error=\(String(describing: error))")  
            return  
        }  
  
        if let httpStatus = response as? HTTPURLResponse, httpStatus.statusCode != 200 {  
            print("\(httpStatus.statusCode)")  
            print("response = \(String(describing: response))")  
        }  
    }  
    task.resume()  
}
```

Se esse data não for nulo, ele já trará os bytes da nossa imagem

REQUESTS IMAGENS

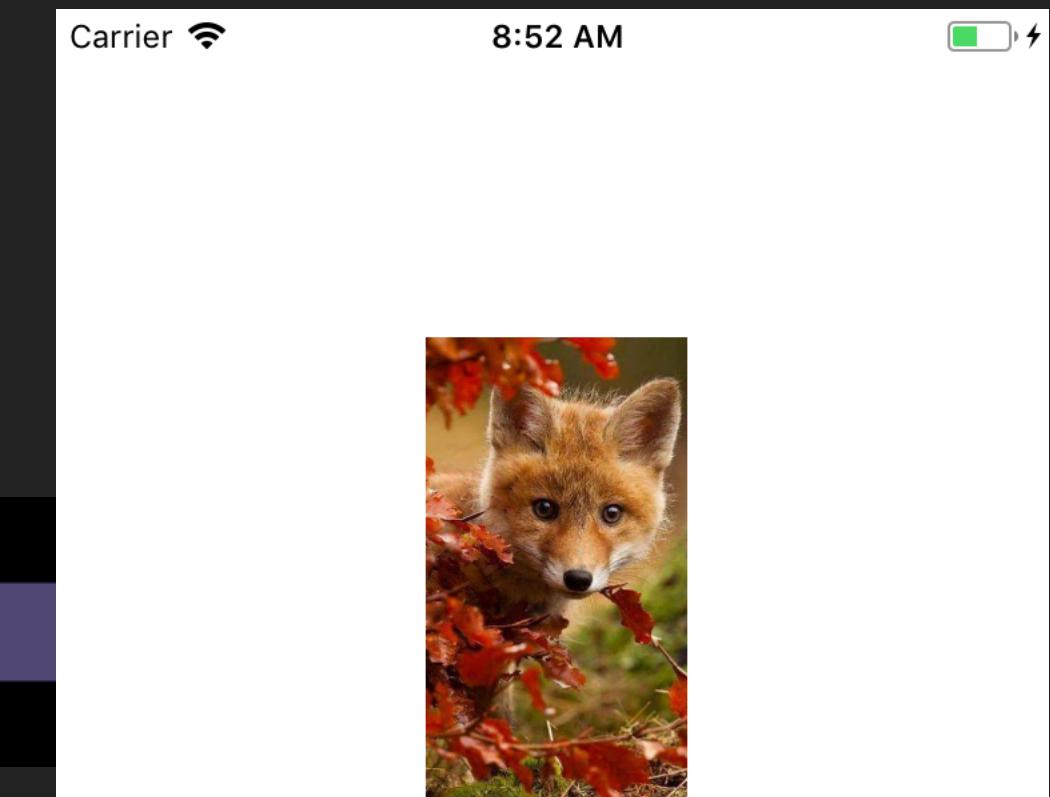
- A UIImage já tem um método preparado para receber um tipo Data

```
self.foto.image = UIImage(data: data)
```

- Ao fazer isso:

- Mas...

```
self.foto.image = UIImage(data: data) ! UIImageView.image must be used from main thread only
```



```
2018-10-15 08:44:49.106705-0300 TesteRealm[25226:6728814] This application is modifying the
autolayout engine from a background thread after the engine was accessed from the main thread.
This can lead to engine corruption and weird crashes.
```

```
Stack:(
```

```
    0  Foundation                      0x000000010dc46df7
AssertAutolayoutOnAllowedThreadsOnly + 77
```

REQUESTS IMAGENS

- ▶ Nossa Task é executada em uma Thread em Background
 - ▶ Operações trabalhosas e de tempo indeterminado devem ser feitas em background para não interferir no uso do App
- ▶ Alterações na UI só são recomendadas na Main Thread
- ▶ É complicado mudar o contexto para a Main Thread?
- ▶ Não

```
DispatchQueue.main.async {  
    self.foto.image = UIImage(data: data)  
}
```

REQUESTS IMAGENS

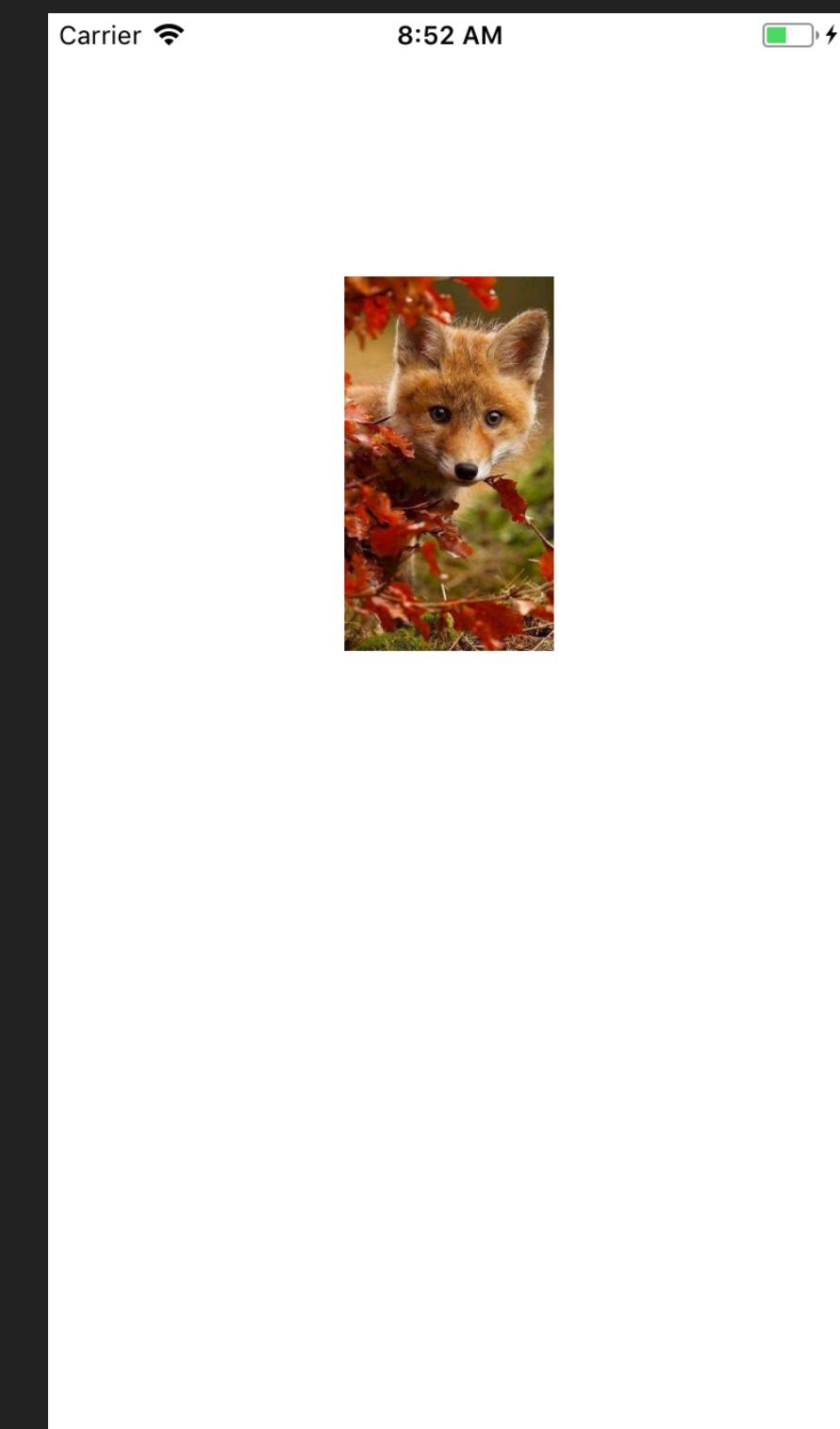
- ▶ Tudo que estiver entre as chaves {} será executado na Thread Main
 - ▶ Lembrando que é como uma closure
- ▶ Devemos lembrar que essa execução não é sequencial

```
DispatchQueue.main.async {  
    print("main")  
    self.foto.image = UIImage(data: data)  
}  
print("fora")
```

fora
main

REQUESTS IMAGENS

- ▶ Ao executar dessa forma conseguimos exibir a imagem sem nenhum erro nem alerta



```
DispatchQueue.main.async {  
    self.foto.image = UIImage(data: data)  
}
```

REQUESTS IMAGENS

- ▶ Nós baixamos uma imagem em específico
- ▶ O problema é que quando tratamos com API's, geralmente, elas não nos retornam o Data da imagem no JSON
 - ▶ O JSON ficaria muito grande
- ▶ Esse mesmo site da raposa tem uma API sem credenciais:

<https://randomfox.ca/floof/>



```
guard let url = URL(string: "http://randomfox.ca/images/32.jpg") else {  
    return  
}
```

REQUESTS IMAGENS

- ▶ A cada vez que executamos um GET nessa API recebemos uma imagem aleatória de uma raposa

<https://randomfox.ca/floof/>

- ▶ Vemos que o JSON nos retorna uma URL
- ▶ Como resolver?



```
1 {  
2   "image": "http://randomfox.ca/images/4.jpg",  
3   "link": "http://randomfox.ca/?i=4"  
4 }
```

REQUESTS IMAGENS

- É possível obter os bytes da imagem só passando a URL:

```
guard let url = URL(string: "http://randomfox.ca/images/32.jpg") else {  
    return  
}  
  
let data = Data(contentsOf: url)
```

- Porém devemos tratar a exceção com um try/catch:

```
guard let url = URL(string: "http://randomfox.ca/images/32.jpg") else {  
    return  
}  
  
do {  
    let data = try Data(contentsOf: url) ⚠ Initialization of im  
} catch {}
```

REQUESTS IMAGENS

- ▶ Logo em seguida preenchemos a imagem com o nosso Data:

```
guard let url = URL(string: "http://randomfox.ca/images/32.jpg") else {  
    return  
}  
  
do {  
    let data = try Data(contentsOf: url)  
    self.foto.image = UIImage(data: data)  
} catch {}
```

- ▶ No entanto, essa forma "trava" a interação do usuário, pois está sendo executada na main thread

REQUESTS IMAGENS

- ▶ Para evitar de ter de codificar várias chamada e uma gerência de cache, especialmente em células da TableView, recomendo usar um pod:

<https://github.com/rs/SDWebImage>



- ▶ É um pod bem documentado, funciona tanto em Objective-C quanto Swift

REQUESTS IMAGENS

- ▶ Simples o uso:

```
import SDWebImage
...
func tableView(tableView: UITableView!, cellForRowAtIndexPath indexPath: NSIndexPath!) -> UITableViewCell! {
    static let myIdentifier = "MyIdentifier"
    let cell = tableView.dequeueReusableCellWithIdentifier(myIdentifier, forIndexPath: indexPath) as UITableViewCell

    cell.imageView.sd_setImageWithURL(imageUrl, placeholderImage:placeholderImage)
    return cell
}
```

CONCLUSÃO

- ▶ Esse código está ficando muito grande para a UIViewController
- ▶ Imagine uma ViewController que faz mais de uma requisição
- ▶ Próxima aula:
 - ▶ Organizando a requisição no MVC
 - ▶ Preenchendo objetos da nossa Model
 - ▶ Como ajustar as requisições ao ciclo de vida da ViewController
 - ▶ Nosso projeto