REST API Explicado

Na **visão do usuário não existe diferença** entre acessar um **site** ou um **aplicativo**.

Na **visão do desenvolvedor**, existe uma **grande distinção** entre desenvolver um **site** e desenvolver um **aplicativo** para dispositivos móveis

- Arquiteturas Diferentes
  - Desenvolvidas por:
    - Desenvolvedor Web
    - Desenvolvedor Mobile

- Maneira Antiga (sem REST APIs)
  - Mesmo Banco de Dados
    - Mecanismos diferentes de comunicação

- Maneira Antiga (sem REST APIs)
  - Mesmo Banco de Dados
    - Mecanismos diferentes de comunicação
    - Erros segregados

- Maneira Antiga (sem REST APIs)
  - Mesmo Banco de Dados
    - Mecanismos diferentes de comunicação
    - Erros segregados
    - Difícil Manutenção

- Maneira Antiga (sem REST APIs)
  - Mesmo Banco de Dados
    - Mecanismos diferentes de comunicação
    - Erros segregados
    - Difícil Manutenção
    - Problemas de Segurança

- Melhor Maneira
  - REST APIs
    - Desenvolvido por Roy Fielding, em 2000

- Melhor Maneira
  - REST APIs
    - Desenvolvido por Roy Fielding, em 2000
    - Soluciona integração Mobile e Web

- Melhor Maneira
  - REST APIs
    - Desenvolvido por Roy Fielding, em 2000
    - Soluciona integração Mobile e Web
    - Defini-se, Cria-se e Disponibiliza-se Recursos via Endpoints

- Melhor Maneira
  - REST APIs
    - Desenvolvido por Roy Fielding, em 2000
    - Soluciona integração Mobile e Web
    - Defini-se, Cria-se e Disponibiliza-se Recursos via Endpoints
    - Compatível com qualquer arquitetura que faça requisições HTTP

- Melhor Maneira
  - REST APIs
    - Fácil Manutenção

- Melhor Maneira
  - REST APIs
    - Fácil Manutenção
    - Aplicação Segura

- Melhor Maneira
  - REST APIs
    - Fácil Manutenção
    - Aplicação Segura
    - Provê escalabilidade

- Melhor Maneira
  - REST APIs
    - Fácil Manutenção
    - Aplicação Segura
    - Provê escalabilidade
      - Bom para
        - Empresa

- Melhor Maneira
  - REST APIs
    - Fácil Manutenção
    - Aplicação Segura
    - Provê escalabilidade
      - Bom para
        - Empresa
        - Desenvolvedores

- Melhor Maneira
  - REST APIs
    - Fácil Manutenção
    - Aplicação Segura
    - Provê escalabilidade
      - Bom para
        - Empresa
        - Desenvolvedores
        - Usuários

Conceitos de REST

API vs Web Service

APIs

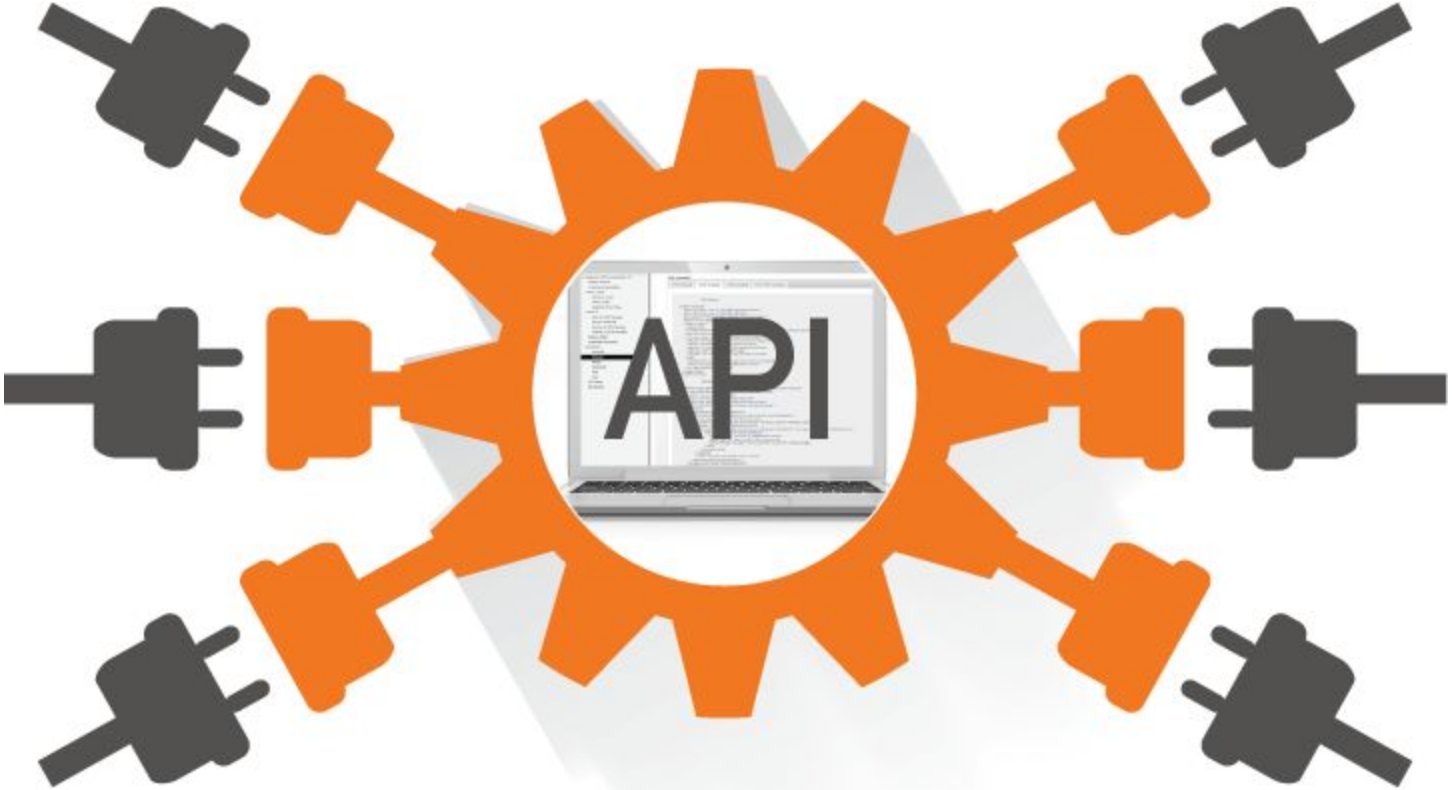
API (Application Programming Interface) – É como uma interface entre dois programas diferentes de modo que eles possam se comunicar um com o outro.

https://fxcosta.wordpress.com/2015/05/31/diferenca-entre-api-e-web-service-de-maneira-simples/

APIs

Ou seja, uma API é a forma que terceiros disponibilizam uma interface de modo que possamos consumir um determinado serviço deles sem nos preocuparmos com a implementação do mesmo.

https://fxcosta.wordpress.com/2015/05/31/diferenca-entre-api-e-web-service-de-maneira-simples/


API

Web Services

Web Service é uma API projetada para se comunicar obrigatoriamente via **rede**. Tipicamente, HTTP é o protocolo mais comumente usado para a comunicação.

https://fxcosta.wordpress.com/2015/05/31/diferenca-entre-api-e-web-service-de-maneira-simples/

Web Services

Web Services utilizam SOAP, REST ou XML-RPC como meio de comunicação. Ou seja, quando uma API precisa enviar dados através da rede, podemos chamar de **Web Services**.

https://fxcosta.wordpress.com/2015/05/31/diferenca-entre-api-e-web-service-de-maneira-simples/

Todos Web Services **são** APIs.
Mas **nem todas** APIs **são** Web Services.

REST e HTTP

HTTP

Hyper **T**ext **T**ransfer **P**rotocol

HTML

Hyper **T**ext **M**arkup **L**anguage

Quando clicamos em um link, na verdade estamos chamando o método **GET do HTTP**

Quando enviamos um formulário, chamamos o método **POST**

GET = Ler
POST = Inserir

PUT = Alterar
DELETE = Remover

Dólar Hoje

US$ 1,00

R$ 3,78

DÓLAR COMERCIAL

SELECIONE A CIDADE...

DÓLAR TURISMO

JSON

```json
{
  "dolarComercial": 3.78,
  "moeda": "BRL"
}
```

XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <dolarComercial>3.78</dolarComercial>
    <moeda>BRL</moeda>
</root>
```

URI

Uniform **R**esource **I**dentifier

(Identificador Uniforme de Recursos)

reservahotel.com/**hotelLookup.do?cities=Rio-de-Janeiro**

hotel**Lookup** ⇒ Pesquisar hoteis
do ⇒ Executar
? ⇒ Query ou Consulta
cities ⇒ Parâmetro chamado cidades

reservahotel.com/hoteis?**cities**=**Rio-de-Janeiro**

URIs Baseados em Recursos

Como estruturar um REST API? Essa parte é negligenciada por muitos, mas **entender os conceitos** de como estruturar um REST API é **fundamental**...

para desenvolver uma aplicação que realmente faça sentido e que seja de **fácil compreensão para os desenvolvedores** que utilizarão seu REST API.

API de Rede Social

Instabook

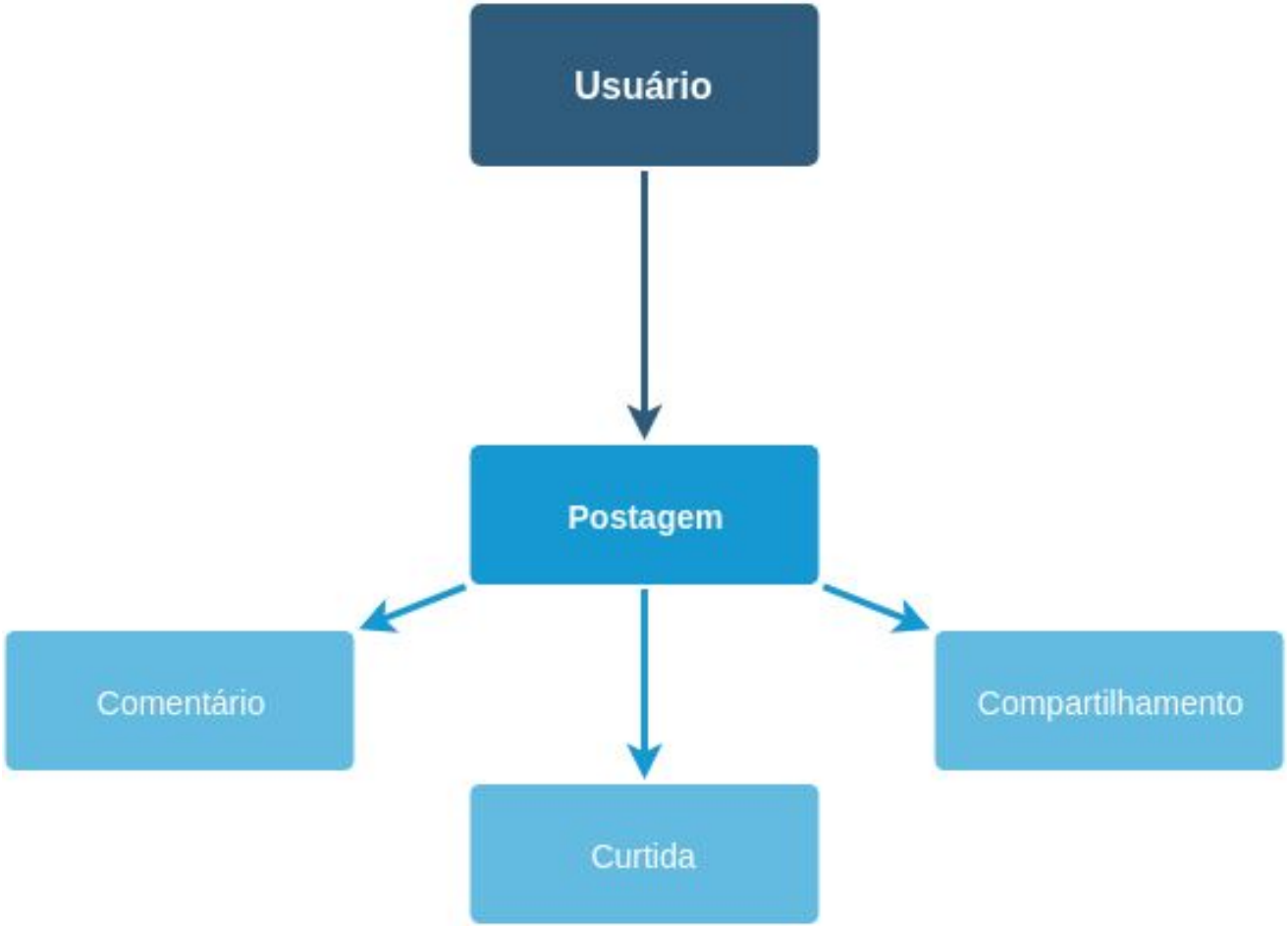
Quais são os recursos que uma API de rede social deve ter?

- Criar ou Postar mensagens
- Comentar em mensagens
- Curtir mensagens
- Compartilhar mensagens
- CRUD de usuários

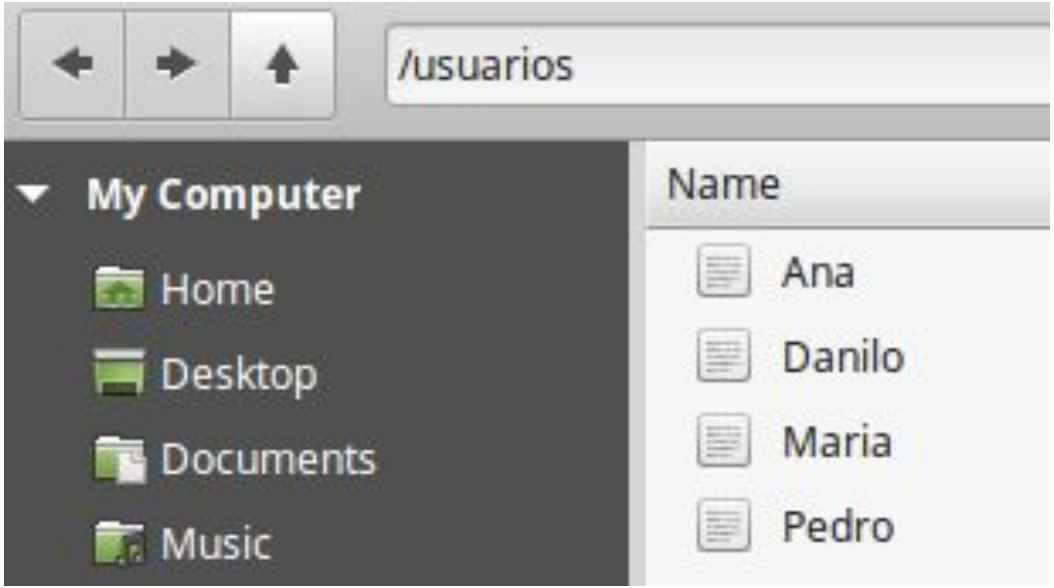
Create (Criar)
Read (Ler)
Update (Alterar)
Delete (Deletar)

CRUD de usuários

- Criar novo usuário
- Ler dados de usuário
- Alterar dados de usuário
- Deletar usuário


Usuário
Postagem
Comentário
Curtida
Compartilhamento

O caminho **/usuarios** me dá acesso
a todos o usuários.



/usuarios/Danilo ⇒ Caminho do usuário Danilo

Logo se eu quiser acessar qualquer usuário especificamente, utilizaria o URI:

/usuarios/{nomeDoUsuario}

Se você quiser definir um URI para todas as postagens do sistema:

Se você quiser definir um URI para todas as postagens do sistema:

/postagens

Se você quiser acessar a postagem que possui o ID 45, como ficaria o URI?

Se você quiser acessar a postagem que possui o ID 45, como ficaria o URI?

/postagens/45

Como ficaria a representação do URI para acessar uma postagem de um ID específico?

Como ficaria a representação do URI para acessar uma postagem de um ID específico?

/postagens/{ID}

Se você quiser definir um URI para as postagens de um determinado usuário:

Se você quiser definir um URI para as postagens de um determinado usuário:

usuarios/{nomeDoUsuario}/postagens

Logo, se eu quiser acessar as postagens do usuário Danilo, o URI seria:

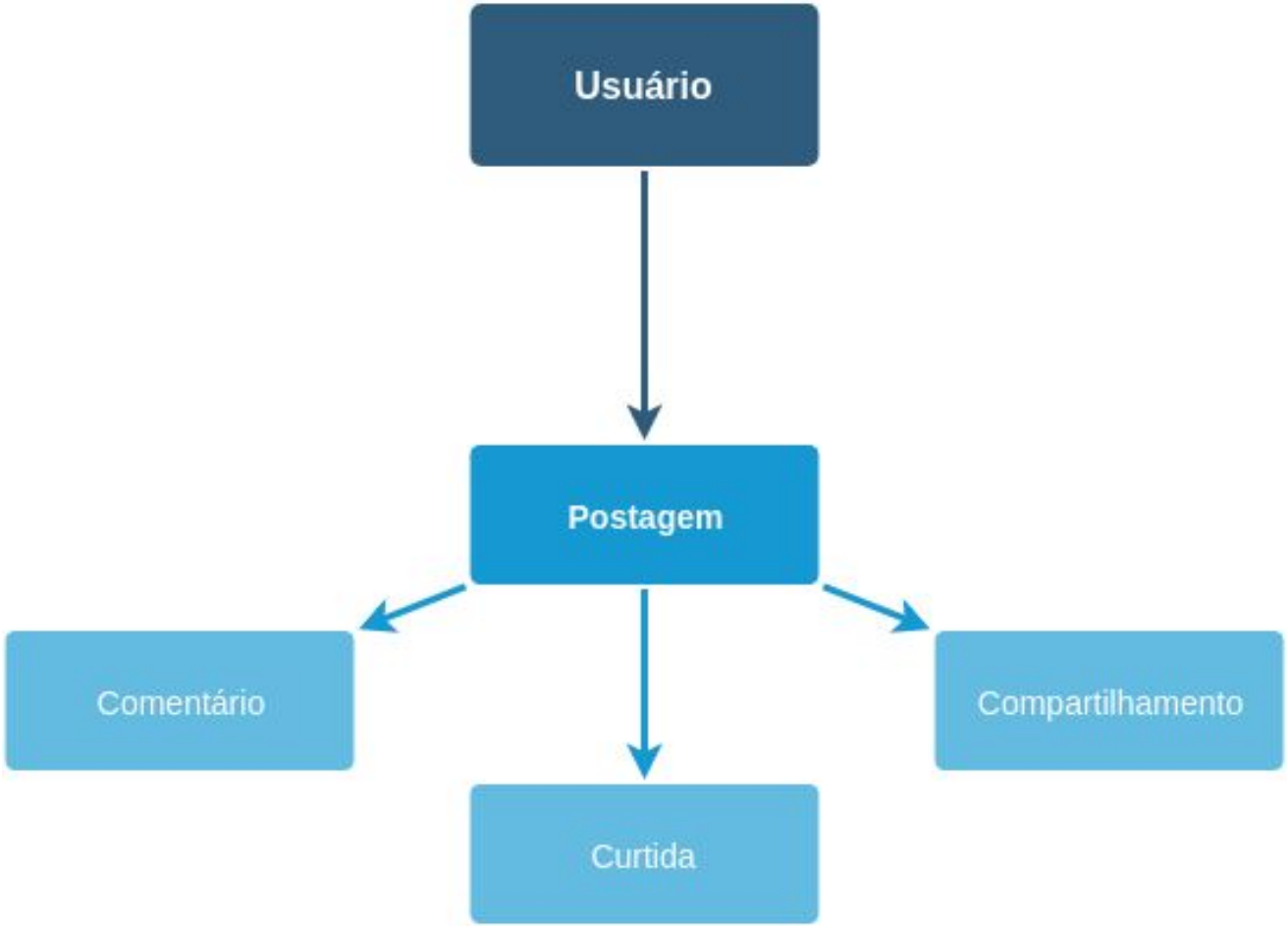
Logo, se eu quiser acessar as postagens do usuário Danilo, o URI seria:

usuarios/Danilo/postagens

E se eu quiser acessar uma postagem específica de um usuário específico?

E se eu quiser acessar uma postagem específica de um usuário específico?

/usuarios/{nomeDoUsuario}/postagens/{ID}


Usuário
Postagem
Comentário
Curtida
Compartilhamento

/postagens/{ID}/comentarios
/postagens/{ID}/curtidas
/postagens/{ID}/compartilhamentos

Não existe certo ou errado. O que existem são boas práticas para nos ajudar a criar REST APIs que sejam de fácil compreensão e intuitivos.

Coleções de Recursos

O URI com **recurso Único** chama-se **Instância** (Instance), e o URI contendo **vários recursos** chama-se **Coleção** (Collection).

O URI com **recurso Único** chama-se **Instância** (Instance), e o URI contendo **vários recursos** chama-se **Coleção** (Collection).

/postagens ⇒ Coleção
/postagens/{ID} ⇒ Instância

Apesar de ser difícil de imaginar **coleções de recursos muito grandes** nos nossos exemplos, devemos estruturar nosso REST API, como se fossem.

Imagine que você deseja solicitar todas as postagens de todos os usuários do Facebook. Com certeza, essa seria uma lista gigantesca.

Por isso, para coleções de recursos, contamos com **Parâmetros de Consulta** (Query Parameters).

Ainda no exemplo das postagens do Facebook. Seria **impossível retornar tudo numa única lista**. Tornando-se necessário utilizar parâmetros de consulta, como **paginação**.

Com paginação, podemos determinar quantos elementos queremos exibir por página, através dos parâmetros de consulta:

limit

e

offset

Limit ⇒ Não mais do que a quantidade limite de elementos serão exibidas (possivelmente menos).

Limit ⇒ Não mais do que a quantidade limite de elementos serão exibidas (possivelmente menos).

Offset ⇒ Quantidade de elementos que serão pulados, de forma que só aparecerão resultados posteriores ao valor de offset.

/postagens?**limit**=10&**offset**=30

limit e offset são **parâmetros padrões** em qualquer aplicação. Porém, podemos criar nossos próprios parâmetros para realizar outros tipos de filtros.

/postagens?**ano**=2018

/postagens?**ano**=2018&**limit**=10&**offset**=30

RECAPITULANDO

- URIs com recurso único são chamados de Instâncias (Instances)

- URIs com vários recursos são chamados de Coleção (Collection)

Uma coleção, idealmente deve conter os nomes no **substantivo** e **plural**

Ex: /postagens, /comentarios, /curtidas

Nunca use verbos no infinitivo: /postar, /comentar, /curtir

Métodos HTTP e CRUD

CRUD	**Métodos HTTP**
Create (Criar)	POST (Criar)
Read (Ler)	GET (Ler)
Update (Alterar)	PUT (Alterar)
Delete (Remover)	DELETE (Remover)

Resposta REST

Status Codes (código de Status) ⇒ São códigos que são enviados juntamente com a resposta de uma requisição HTTP. Esses códigos ajudam o cliente saber se a requisição foi bem sucedida. E se não foi, ajuda-o a entender qual foi a causa provável do erro.

200 - Success

500 - Server error

404 - Not found

AHHHH! 404...

Cockatoo can't find the page,
but she can scream in a cup for you.

Go back home.

Quem coloca essa página 404 geralmente é o desenvolvedor web que está consumindo o REST API em questão. Assim como, toma outras decisões acerca de cada Status Codes.

Por isso, **é muito importante** que quando desenvolvermos os nossos REST APIs, usemos os **códigos apropriados** em cada resposta, sejam de sucesso ou de erro.

É muito fácil programar imaginando que tudo funcionará 100%, e que o usuário não cometerá erros, mas **um bom Web API deve considerar o máximo de cenários** possíveis...

de forma a dar códigos de **sucesso** quando tudo **ocorrer bem**, e também **prever possíveis erros**, e exibir os **códigos que melhor descrevem** a causa do **erro**.

Formato dos Dados

O criador do REST API é quem decide em que formatos ele exibirá os dados como resposta. Como falado anteriormente, os **formatos mais comuns são JSON e XML.**

JSON é relativamente novo em comparação com o XML, mas devido a sua simplicidade, tem crescido muito mais. A maioria dos **REST APIs mais recentes**, **têm JSON como formato padrão** ou principal.

Em **muitos casos**, fornecem **ambos os formatos JSON** e **XML**. Nesse caso, ao fazer a requisição, o cliente pode informar qual o formato que ele deseja receber os dados.

Você pode estar se perguntando agora: como o cliente sabe quais os formatos disponíveis no servidor?

Uma observação: quando digo cliente, estou me referindo ao programador, ou ao desenvolvedor web que **consumirá** o Web API.

Respondendo a pergunta: o cliente descobre não apenas quais formatos de dados, mas quais recursos estão disponíveis através da **documentação do REST API.**

Apesar de não existir nenhuma obrigação de ter documentação, ou até mesmo um manual de como ela deve ser. É altamente **recomendável criar uma documentação**, **e seguir guias de boas práticas**.

E como o servidor identifica que tipo de formato de dados enviar? Através de um parâmetro do **Header**(ou cabeçalho), chamado **Content-Type** (Tipo de Conteúdo).

Junto com a requisição para acessar um Recurso, envia-se um **cabeçalho** contendo as informações necessárias. Essas informações necessárias, são chamadas de **Metadata** (ou metadados em português).

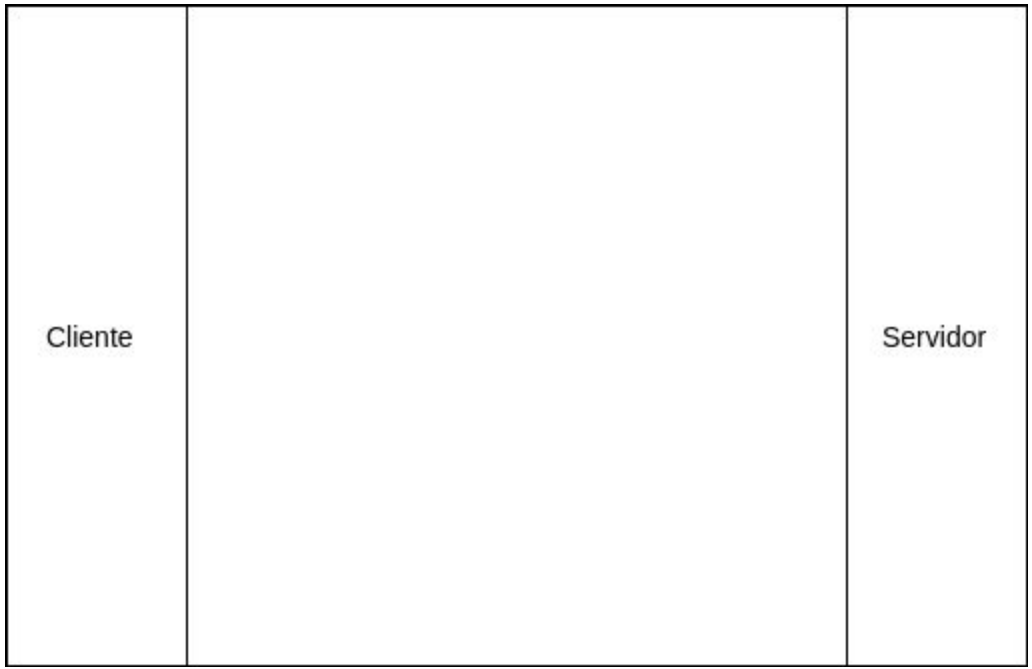
Content-Type é um tipo de Metadata ou Metadados, que indica qual o formato que se espera do servidor. Existem alguns padrões predefinidos de valores do Content-Type.

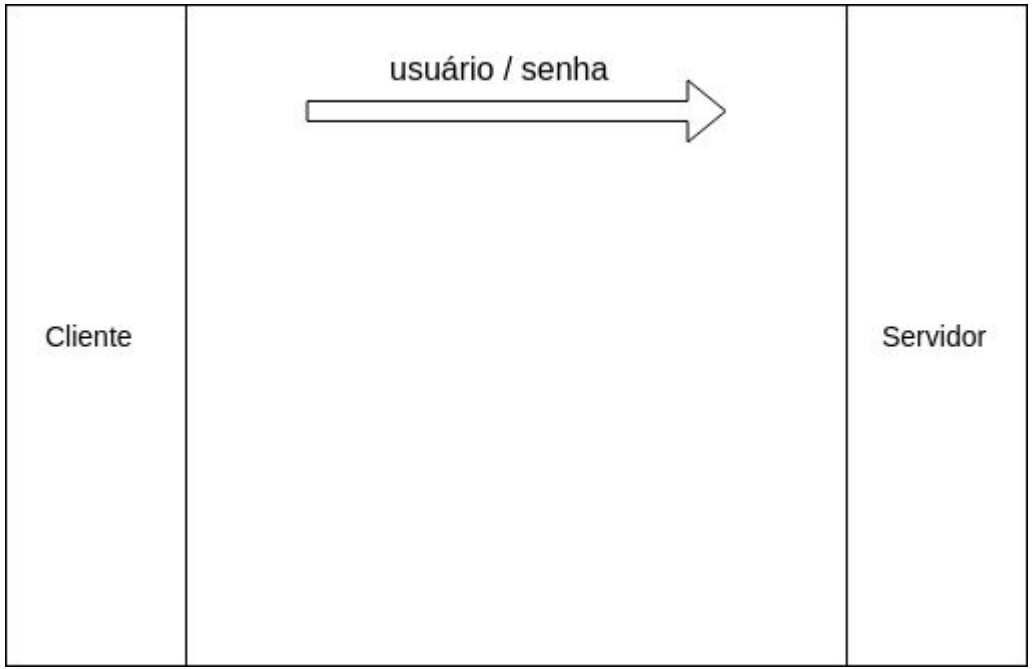
Para receber dados em XML, usa-se **text/xml**.
Para receber dados em JSON, usa-se **application/json**.

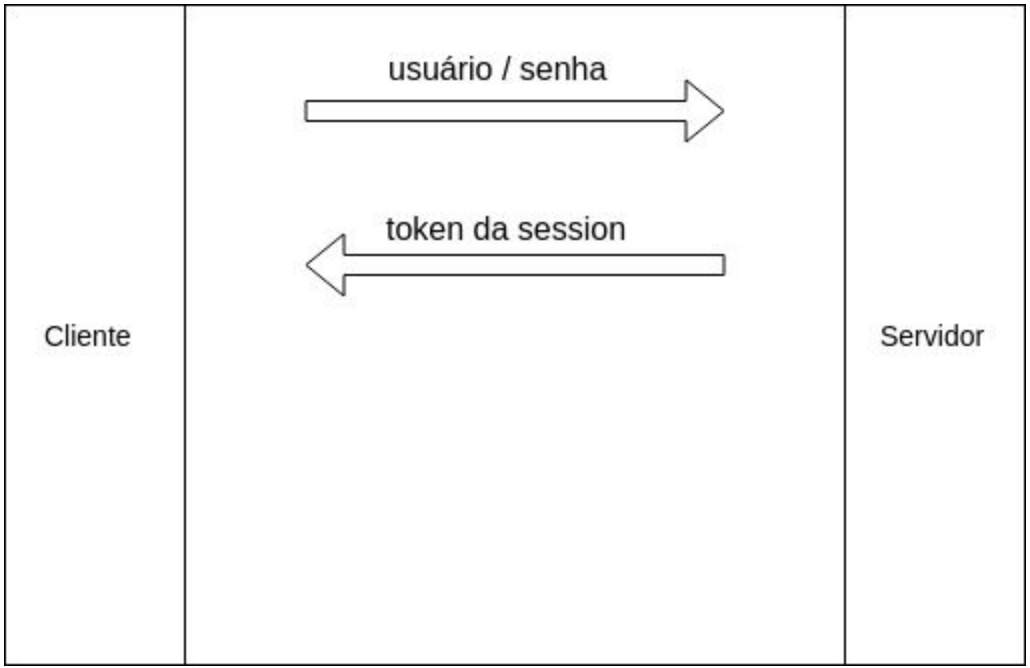
Content-Type	application/json
Header	Value

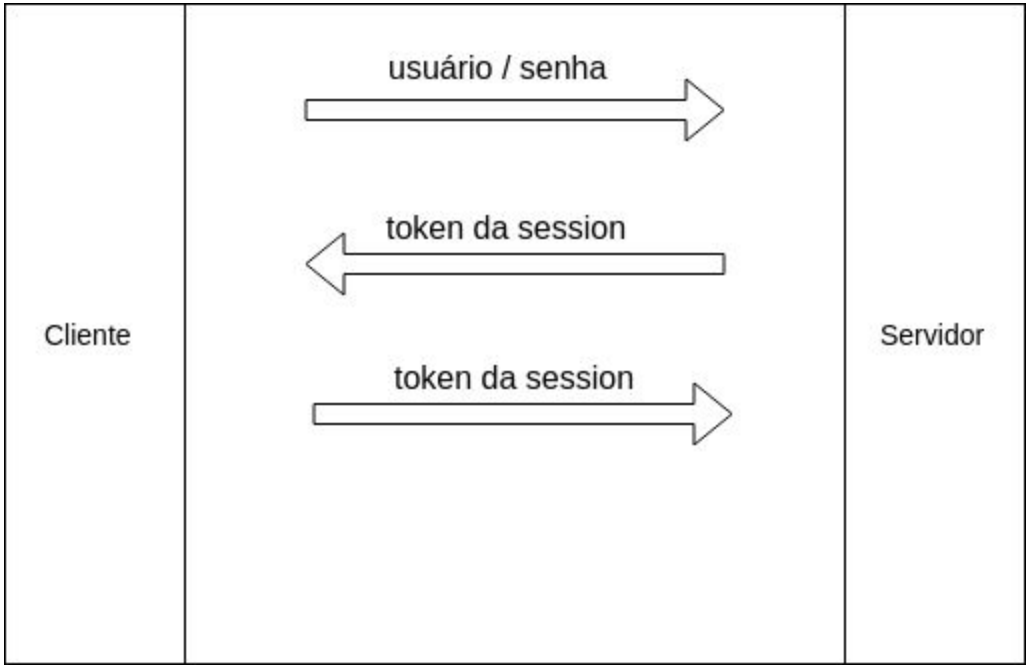
Autenticação REST

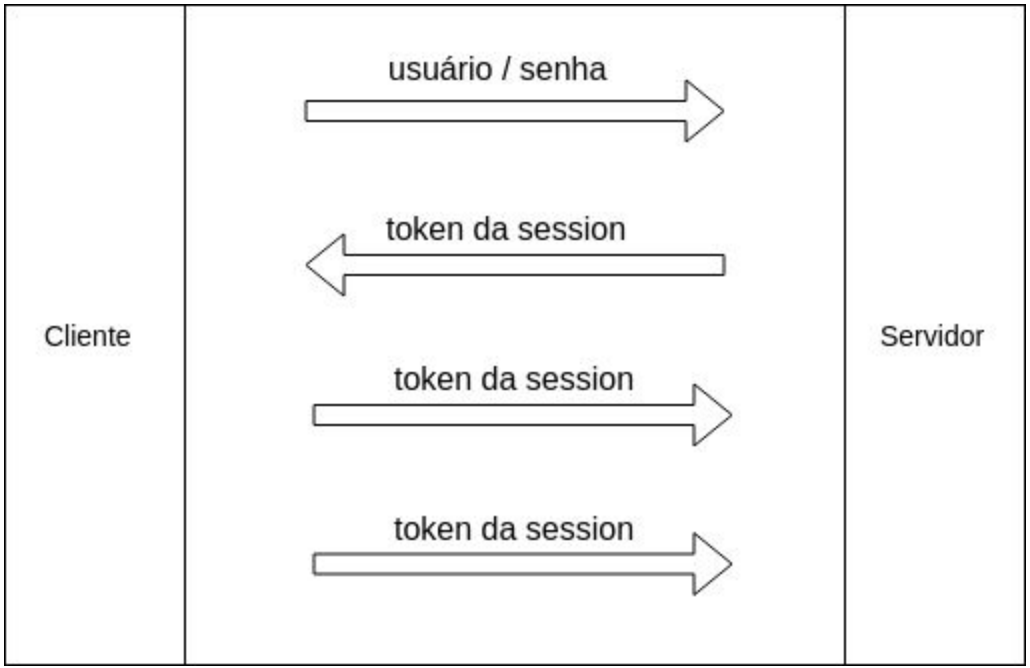
Para entender como funciona a Autenticação de usuários em REST APIs, vamos dar uma olhada em como funciona a autenticação clássica de uma aplicação Web:

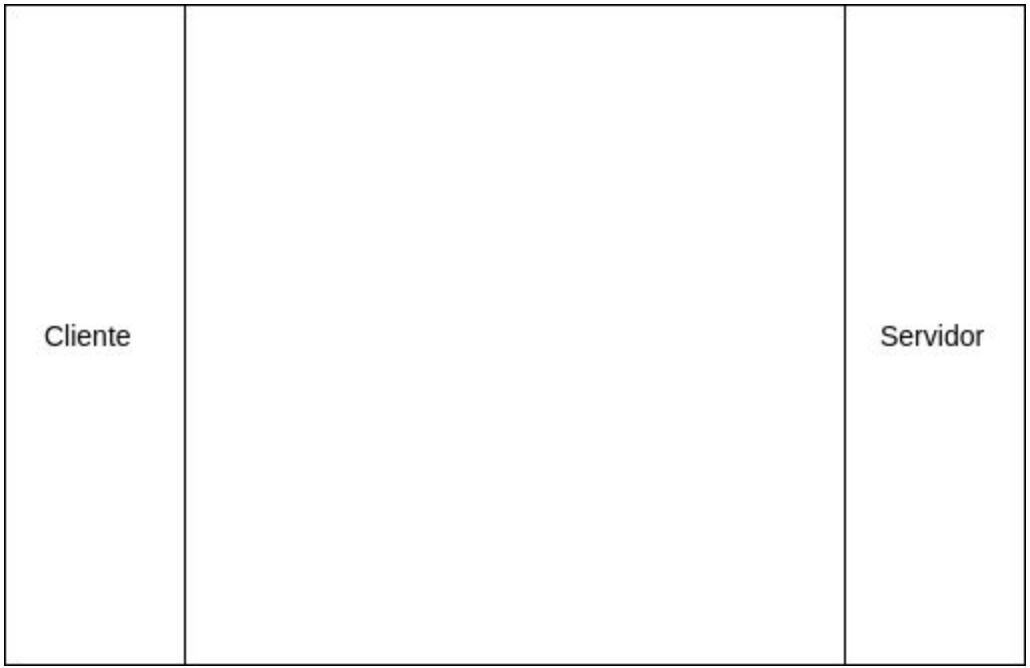

Cliente
Servidor


usuário / senha
Cliente
Servidor


usuário / senha
token da session
Cliente
Servidor


Cliente
usuário / senha
token da session
token da session
Servidor


Cliente
usuário / senha
token da session
token da session
token da session
Servidor

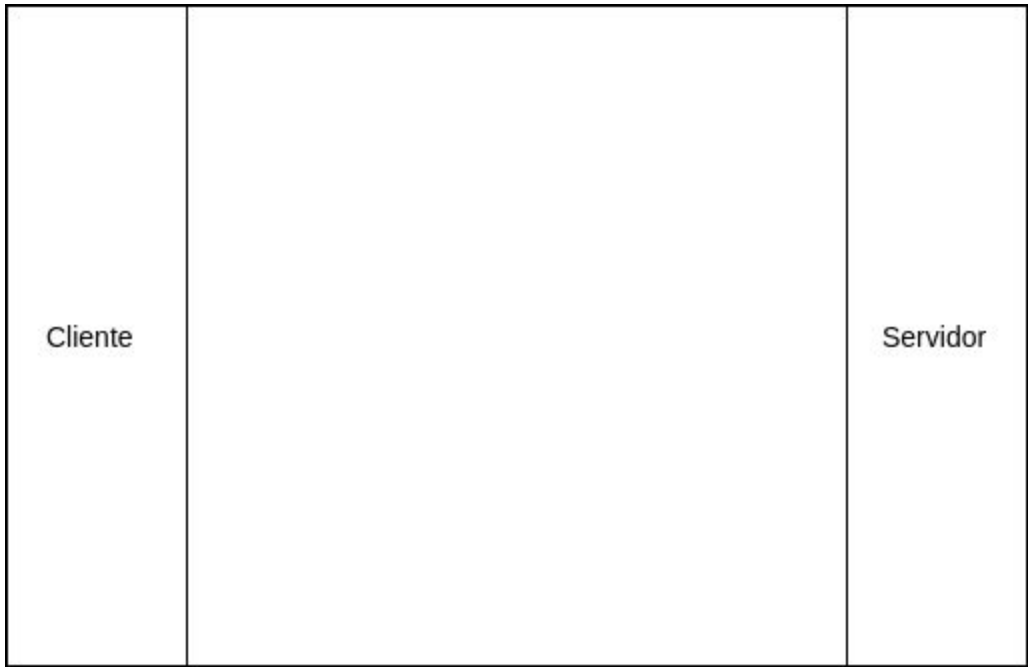

Cliente
Servidor

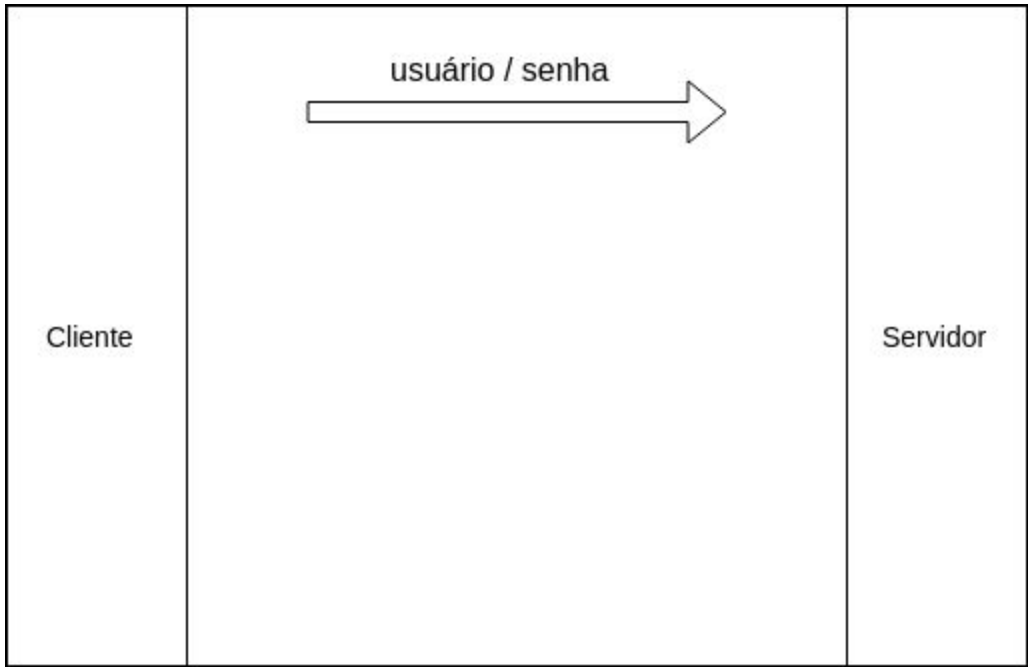
Apesar disso ser relativamente simples. **É um problema quando se trata de REST APIs**, porque REST APIs são stateless.

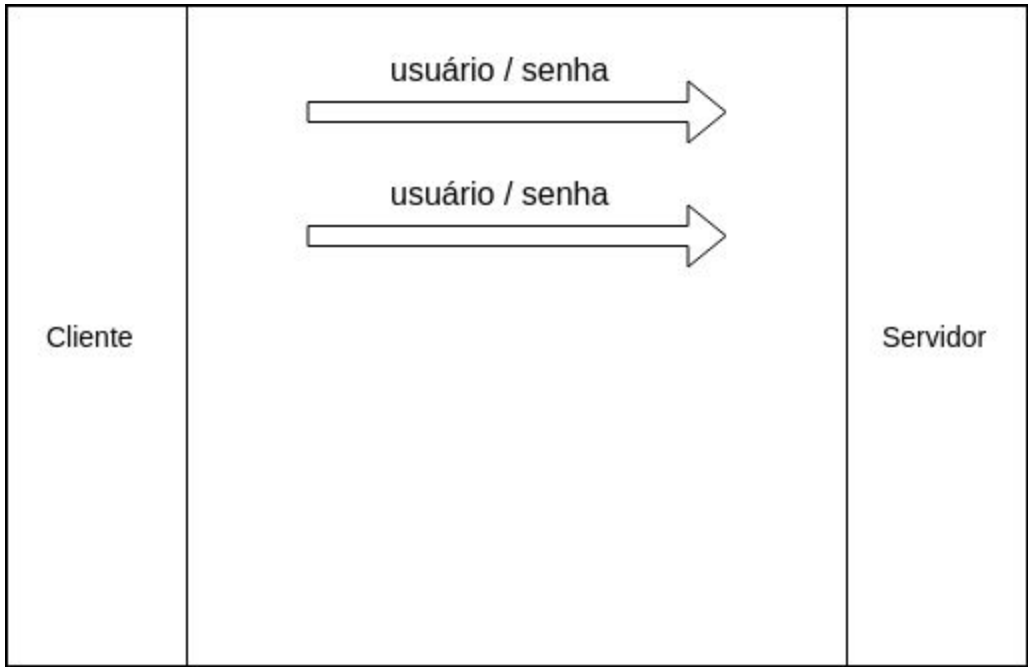
REST APIs são **Stateless** (ou sem estado)

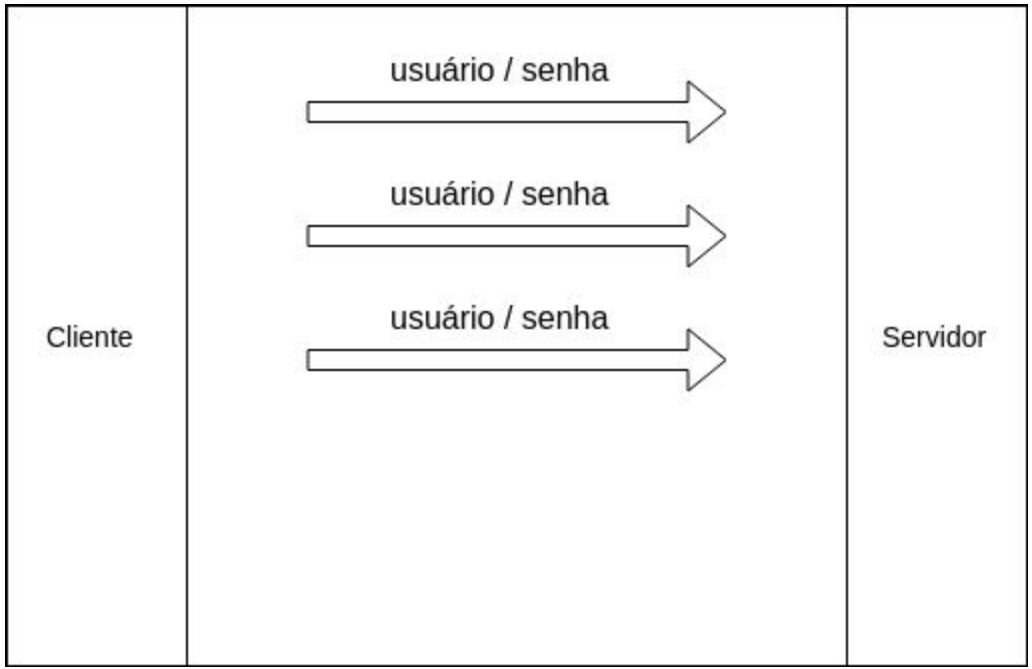
E se é assim que funciona, não é possível fazer autenticações baseadas em “sessions”, pois o servidor não guardará essa informação.

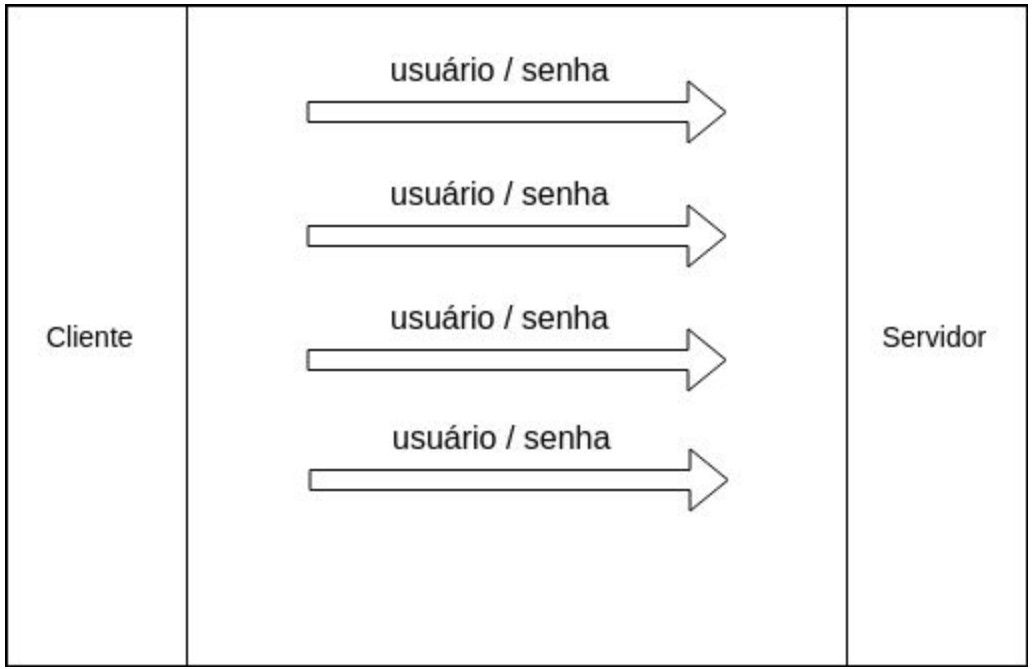
Então como resolvemos esse problema? O mecanismo de autenticação mais básico é um chamado **Basic Auth** (Basic Access Authentication)

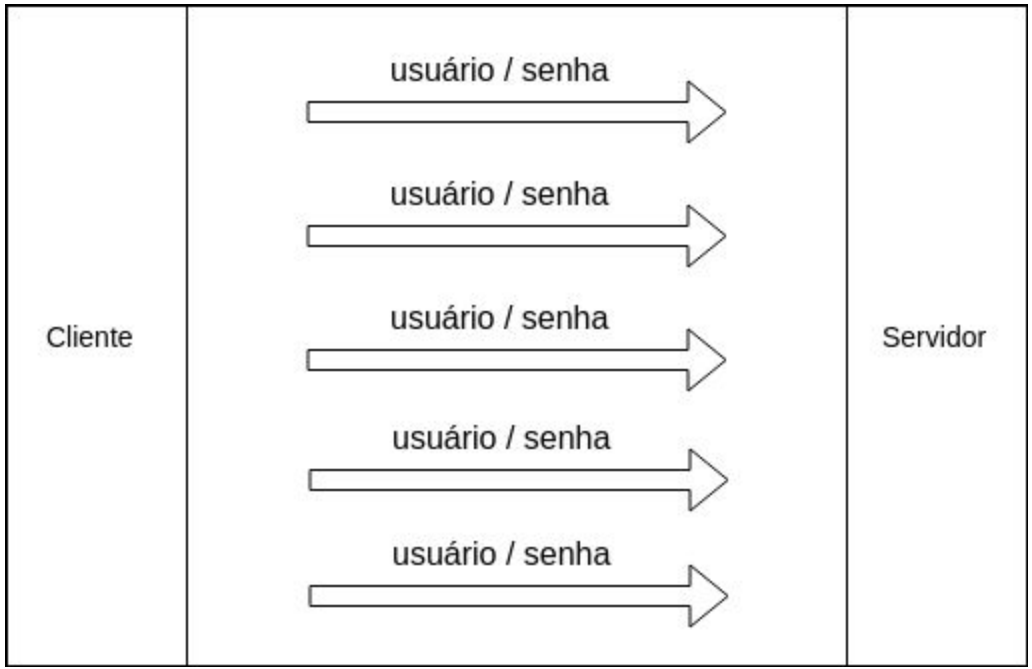

Cliente
Servidor


usuário / senha
Cliente
Servidor


usuário / senha
usuário / senha
Cliente
Servidor

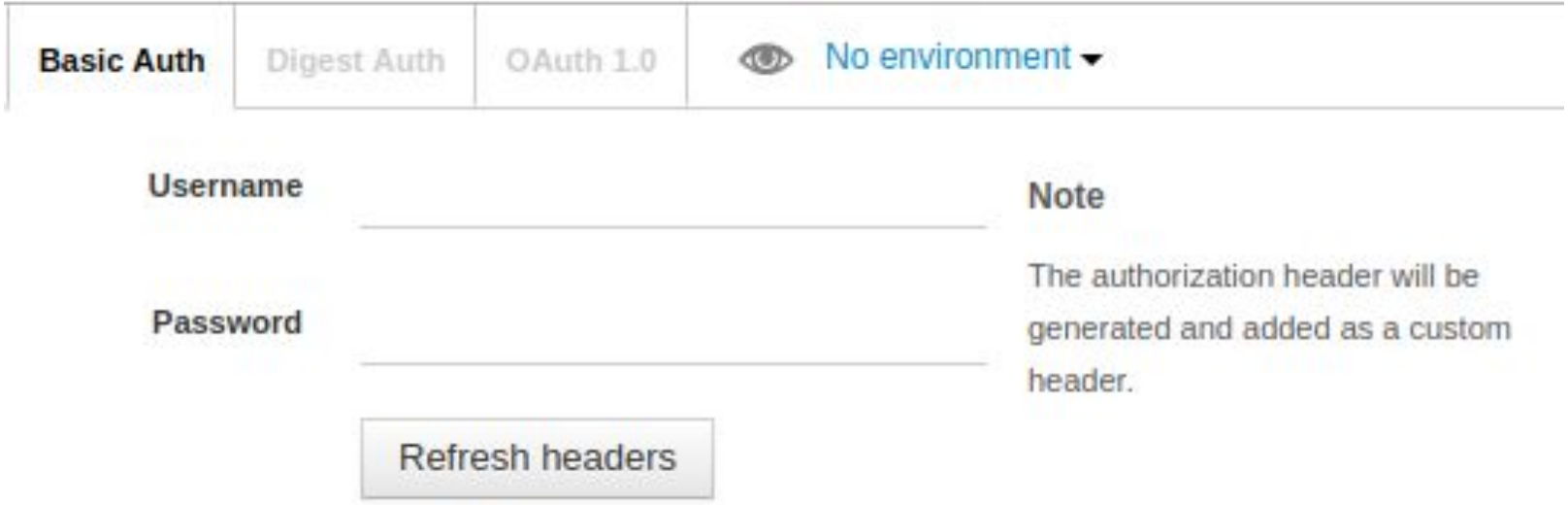

Cliente
usuário / senha
usuário / senha
usuário / senha
Servidor


Cliente
usuário / senha
usuário / senha
usuário / senha
usuário / senha
Servidor


Cliente
usuário / senha
usuário / senha
usuário / senha
usuário / senha
usuário / senha
Servidor

Então não existe nenhum “estado” sendo mantido no servidor, pois toda vez que o cliente precisa fazer uma requisição ao servidor, envia o usuário e senha.

O que não tem problema, pois o cliente acaba se autenticando a cada requisição. Daí o nome: **Basic Auth**, realmente é a forma mais **básica** de **autenticação** possível...


Basic Auth
Digest Auth
OAuth 1.0
No environment
Username
Password
Refresh headers
Note
The authorization header will be generated and added as a custom header.

BASIC AUTH

usuario / senha

codificação Base64

dXN1YXJpbzpzZW5oYQ==

HEADER

Authorization	Basic dXN1YXJpbzpzZW5oYQ==
Header	Value

O **servidor** verifica a primeira palavra chamada “**Basic**” e então sabe que é Basic Auth, logo sabe que a string do lado foi codificada com Base64, e então, faz a **decodificação** Base64, e pega o **usuário e a senha**.

Codificação não é criptografia. Logo, qualquer um que tenha acesso a essa String codificada, pode decodificá-la com Base64. Por isso **não é seguro**.


https://

Você deve estar se perguntando. Se qualquer um pode acessar e decodificar, por que não passar o usuário e senha direto via HTTPS?

A razão pela qual se codifica uma string não é por questão de segurança, mas sim por questão de haver **caracteres** que **não** são **compatíveis com HTTP**, e quando se codifica, elimina-se esse problema.

A razão pela qual se codifica uma string não é por questão de segurança, mas sim por questão de haver **caracteres** que **não** são **compatíveis com HTTP**, e quando se codifica, elimina-se esse problema.

Vantagens

- Simples
- Servidor Stateless
- Suportados por todos os browsers

Desvantagens

- HTTPS apenas
- Sujeito a ataques
- Deslogar é complicado

Melhores Soluções

- Digest Access Authentication
- Asymmetric Cryptography
- OAuth
- **JSON Web Token**