

Programação Móvel

Web Services e REST API

Prof. Dr. Diego R. Antunes

drantunes@utfpr.edu.br

Departamento de Computação
Universidade Tecnológica Federal do Paraná

Uso do Exemplo de Projeto Ionic do Moodle

Usando o Exemplo do Moodle

- Fazer o download;
- Executar `npm install` e `ionic platform add android`
- Alterar `name` em `config.xml`, `package.json` e `ionic.config.json`.

Entrega do Trabalho 1

Entrega do Trabalho 1

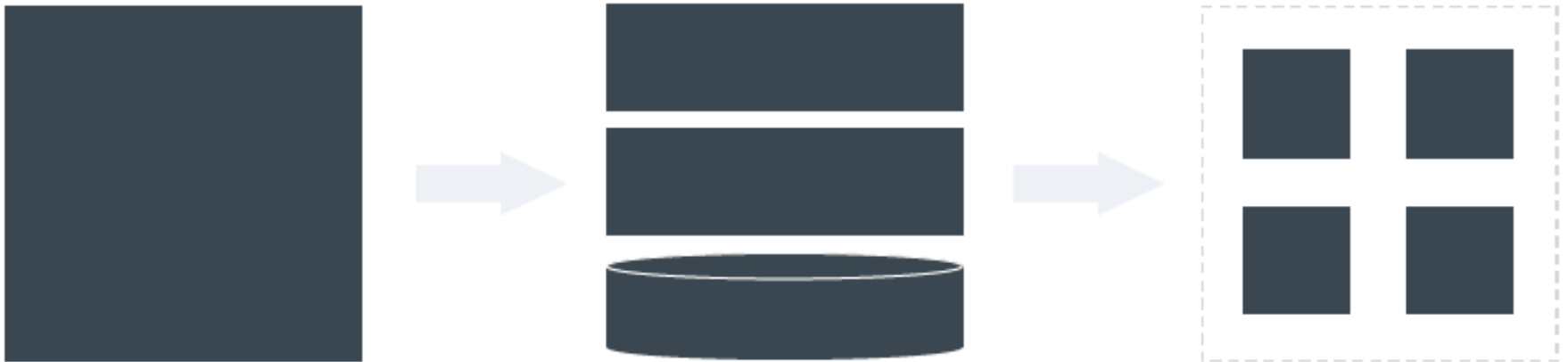
- Envio via Moodle
- Compartilhar no Ionic View, com drantunes@gmail.com
- Padronizar o nome do projeto com Trab1_Nome_do_Aluno
- Alterar em `config.xml`, `package.json` e `ionic.config.json`

Histórico

Evolução do uso dos padrões arquiteturais

Evolução

- Aplicações Monolíticas;
- Aplicações em Camadas;
- Aplicações Orientadas à Serviços;



SOA

Service Oriented Architecture

SOA

- Divide as funcionalidades da aplicação em serviços;
- Pequenos componentes que formam outras aplicações;
- Usada para integração entre sistemas;
- Exemplo: WebService de Previsão no Tempo;

Web Services

- Possibilita o envio de requisição e resposta por URL;
- Utiliza um formato de mensagem: no geral, XML

SOAP

- Simple Object Access Protocol;
- Consiste em um protocolo que suporta os Web Services;
- Trabalha com a idéia de Invocação Remota de Métodos;
- A troca de informações era realizada sob o XML;

WSDL

- Web Services Description Language;
- Linguagem para descrever / mapear os métodos remotos;
- Especificação desenvolvida pela W3C, em XML;

UDDI

- Universal Description, Discovery and Integration;
- “Repositório” para buscar e cadastrar web services;
- Criado pela IBM e Microsoft e disponibiliza:
 - informações gerais da organização, tais como nome, endereço e contatos;
 - informações de organizações e serviços por categorias de negócios;
 - informações técnicas sobre os serviços providenciados pelas organizações.

Desvantagens

- XML pesado para grandes conjuntos de dados;
- Muitos protocolos e padrões;
- WSDL adiciona mais uma camada;
- Chamadas de métodos não natural;
- Problemas na refatoração de código;

RPC

- Remote Procedure Call;
- Chamada de funções em outro endereço (remoto);
- Arquitetura Cliente-Servidor;
- XML-RPC é uma implementação para Web Services;

RPC

- Não exige a criação de WSDL;
- Implementação mais simples que SOAP;
- Seu foco é chamar método remotos;
- Desvantagem: ACOPLAMENTO;
- Se o serviço muda a assinatura do método, cliente quebra.

Exemplos

- www.app.com/webservices/clima/getClima.do?cidade=10
- [/getClientes.do](#)
- [/deleteClientes.do?idCliente=10](#)

Padrão REST

Programando REST APIs

REST

Conceito apresentado por Roy Fielding (2000), em sua dissertação *Architectural Styles and the Design of Network-based Software Architectures*

REST

- REpresentational State Transfer;
- Trabalha com o conceito de transporte de recursos;
- Utiliza o protocolo padrão HTTP;

REST

- Facilita a Integração entre Apps;
- Facilita o uso e a definição de APIs;

Permite então que uma aplicação estenda as suas funcionalidades em uma interface padrão para que novas aplicações possam utilizar suas informações e funções;

Vantagens

- Facilidade de Leitura;
- Usabilidade;
- Protocolo simples e comum: HTTP;
- Interface comum para o uso de uma API;
- Não precisa de uma Linguagem de Descrição (e.g. WSDL);

API

- Application Programming Interface;
- Interface que permite o uso de um serviço web para integrar e obter informações que possam ser utilizadas na composição de outro serviço – via programação.

API - Exemplo

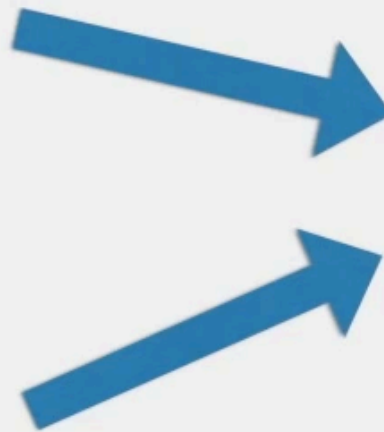
Publicar informações no Facebook de outros Apps?



Apps



Games



REST WebServices

São executados sobre o protocolo HTTP;

REST WebServices



Cliente

The diagram consists of two dark gray rounded rectangular boxes. The box on the left contains the word 'Cliente' in white serif font. The box on the right contains the word 'WebService' in white serif font. There are no lines or arrows connecting the two boxes.

WebService

HTTP Request



HTTP Response



Protocolo de transporte

Ao invés de envelopar as informações com SOAP, o REST usa todos os recursos do HTTP para as requests/responses;

HTTP

HyperText Transfer Protocol

HTTP

- Padrão utilizado pela WWW;
- Utiliza TCP/IP – Atuando na camada de aplicação;
- Utilizado para transmitir recursos (URL);

HTTP

```
> GET / HTTP/1.1  
> User-Agent: Firefox  
> Host: api.facebook.com/friends  
> Accept: */*
```


HTTP

```
< HTTP/1.1 200 OK  
< cache-control: private,...  
< content-encoding:gzip  
< content-type:text/html
```

```
<!doctype html><html itemscope=""  
itemtype="http://schema.org/WebPage"  
lang="pt"><head><meta content="Imagens do  
Google. A pesquisa de imagens mais completa da  
web." name="description">
```

Métodos HTTP

- GET
- POST
- PUT
- DELETE
- PATCH
- HEAD

Métodos HTTP

- Navegador faz o mapeamento entre HTML e HTTP;
- Exemplo: `<form method="GET" ...`

Método GET

- Requisições de leitura / busca;
- Os dados são enviados na URL via query_string
- `www.exemplo.com?nome=joao&cpf=123456`
- `GET ?nome=joao&cpf=123456`

Método POST

- Requisições de escrita para o servidor;
- Os dados são enviados no corpo da requisição;
- Também seguem o query_string, mas ficam protegidos;

Método PUT

- Requisições de atualização (update) para o servidor;
- Os dados são enviados no corpo da requisição;
- Também seguem o query_string, mas ficam protegidos;

Método DELETE

- Requisições para deletar um recurso;
- Os dados são enviados no corpo da requisição;
- Também seguem o query_string, mas ficam protegidos;

Métodos

- **C**reate (POST)
 - **R**ead (GET)
 - **U**ppdate (PUT)
 - **D**elelete (DELETE)
-
- Operações básicas no backend: CRUD

Cabeçalhos (Header)

- Tokens para Autenticação;
- Controle de Cache;
- Controle de Resposta;
- Entre outros.

Status Code

- **1xx** Indica uma mensagem informacional
- **2xx** indica sucesso na requisição
- **3xx** redireciona o cliente para outra URL;
- **4xx** indica erro na requisição no lado do cliente;
- **5xx** indica erros internos no servidor;



error
500
Internal Server Error

API REST

A grande vantagem é o acesso a um conjunto de recursos por meio de uma única URL. Mas como?

SOAP

www.exemplo.com/getClientes

www.exemplo.com/getCliente?idCliente=10

www.exemplo.com/updateCliente?nomeCliente=Joao

REST

GET www.exemplo.com/clientes

GET www.exemplo.com/clientes/10

PUT www.exemplo.com/clientes

Componentes

Componentes de uma API REST

URI para a API

- Geralmente um sub-diretório / sub-domínio da aplicação;
- Pode estar separada da aplicação principal;
- Também conhecido como **endpoint**
- Exemplos:

graph.facebook.com

api.twitter.com

Resource

- Definir quais os recursos a API irá disponibilizar;
 - Exemplos: clientes, pagamentos, produtos
-
- clientes/
 - clientes/{cliente_id}

Routes

- Mapear a localização dos recursos (resources);
- A chamada é feita no padrão: `endpoint/recurso`
- Exemplo: `api.empresa.com/clientes`

Relacionamento de Recursos

- Recursos aninhados;
- Exemplo: um **Cliente** tem muitos **Endereços**;
- Logo, sua API deve possuir um recurso aninhado:
- `/clientes/{cliente_id}/enderecos`

Relacionamento de Recursos

/clientes/1/enderecos

Cliente 1

Endereço 1

Endereço 2

Endereço 3

/clientes/5/enderecos

Cliente 5

Endereço 4

Formatos de Resposta

- JSON ou XML (mais comuns);

```
{  
  "nome": "Diego",  
  "profissao": "Professor"  
}
```

JSON = Array de Objetos JavaScript

Filtros

- `/clientes/10/enderecos?cidade=São Paulo`

Paginação

- `/clientes?offset=30&limit=10`
- Offset é o ponto de início;
- Limite é a quantidade por página;

Autenticação

Técnicas de Autenticação para APIs

Autenticação

Em geral, aplicações modernas utilizam métodos baseados em *token* para fazer a autenticação do usuário.

Autenticação

Dois tipos de autenticação:

- Cliente – API
- Cliente – API – Software Terceiro

Cliente - API

- Exemplo de método: JWT <http://jwt.io>
- JSON Web Token

Características

- Envio de informações pessoais para API;
- Pode ser trafegado via URL (requisito para HTTP REST);
- Pode ser enviado via HEADER na requisição;
- Utiliza criptografia e verificação de assinatura;

Como funciona?

- O cliente possui três informações principais (JSON):
 - Header
 - Payload
 - Signature
- Forma uma string no formato: **xxxxxx.yyyyyy.zzzzzz**

Header

- Descreve o tipo;
- Descreve o algoritmo (e.g. RSA ou SHA256);
- Então esse objeto é codificado para Base64Url –
criptografa para binário e como resultado tem-se um texto;

Payload

- As informações de envio;
- Podem ser de três tipos: *reserved*, *public*, *private*;
- Também faz a codificação para Base64;

Payload

- Reserved:
 - **iss** – quem enviou a solicitação (e.g. site.com.br);
 - **exp** – tempo de expiração;

Payload

- Private:
 - Informações customizadas;
 - Utilizadas para compartilhamento entre partes;
- Public:
 - Informações públicas de acordo com o padrão IANA (e.g. OpenID) -
<http://www.iana.org/assignments/jwt/jwt.xhtml>

Signature

- Cria uma assinatura ao objeto;
- Usa um algoritmo de criptografia;
- Utiliza uma senha (chave);
- E a assinatura é gerada combinando header e payload;

Validação no Servidor

- Pode gerar um novo token e comparar com o enviado;

OAuth 2

Autenticação entre aplicações (APIs);

OAuth 2

- Validação Cliente – API – Software Terceiro;
- Muito utilizado em “Login Social”
- Facebook, Twitter, Github, entre outros, utilizam OAuth 2;

OAuth 2

- Faz a autenticação em uma API em nome de um usuário;
- Exemplo: Do Facebook o usuário publica no Twitter;

Ou seja, um usuário do software **A** quer conectar no software **B** para que **A** acesse suas informações.

Que problema resolve?

- Sem o OAuth, para integrar duas aplicações era necessário que o usuário informasse seu login e senha!
- Se existe uma API genérica, então cada usuário precisaria ir na plataforma de destino, cadastrar uma chave de API e passar ela para a aplicação que deseja acesso.

Que problema resolve?

- O OAuth permite conectar duas aplicações via API;
- O usuário permite ou não acesso aos seus dados;
- Pode escolher quais níveis de informações irá consumir;

Como funciona?

- Antes de tudo, a aplicação cliente precisa se registrar na aplicação destino.
- Exemplo: uma aplicação que envie fotos ao facebook;
- ID do Cliente, Chave de Api, URL de Retorno



Movie List ▾

Dashboard

 Dashboard

Movie List •

Add a New App

Select a platform to get started



iOS



Android



Facebook Canvas



Website

If you're developing on another platform or want to skip this step for now, use the [basic setup](#).

Facebook Login

Active Login Users

Trend

1

☒ Monthly Active Users☒ Weekly Active Users



Start Over

WWW

Quick Start for Website

Skip and Create App ID

TesteApp



Create New Facebook App ID



TesteApp ▾

[Dashboard](#)[Settings](#)[App Review](#)[App Details](#)[Roles](#)[Open Graph](#)[Alerts](#)[Localize](#)[Canvas Payments](#)[Audience Network](#)[Test Apps](#)[Webhooks](#)[Analytics](#)

Dashboard



TesteApp ○

This app is in development mode and can only be used by app admins, developers and testers [?]

API Version [?]

v2.5

App ID

1085473221474627

App Secret

●●●●●●●●

Show

Get Started with the Facebook SDK

Use our quick start guides to set up the Facebook SDK for your iOS or Android app, Canvas game or website.

[Choose a Platform](#)

Facebook Analytics for Apps

Get Analytics and Trends

Use Facebook Analytics for Apps to understand how people are using your iOS, Android or Canvas app.

[Try It Now](#)

Facebook Login

Active Login Users

Trend

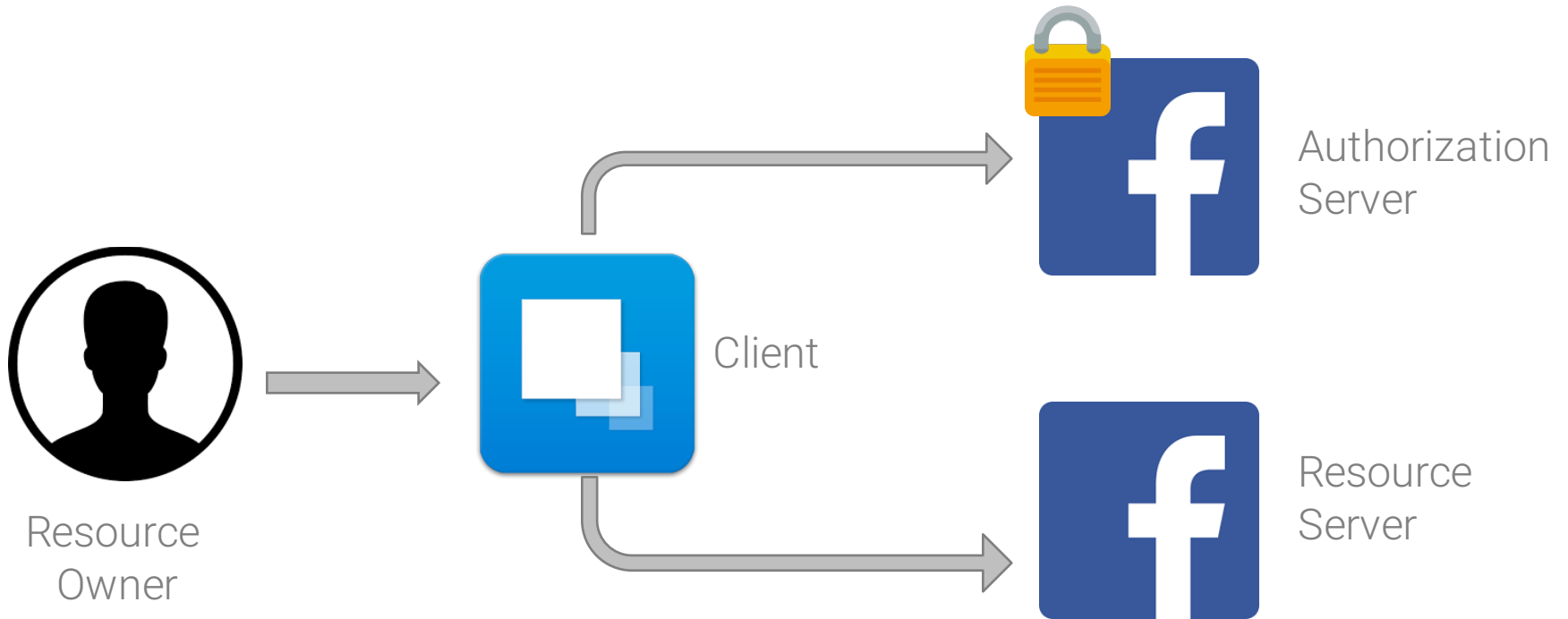
1

☒ Monthly Active Users☒ Weekly Active Users☒ Daily Active Users

Componentes

- Client;
- Authorization Server;
- Resource Server;
- Resource Owner;

Componentes



Autorização



An application would like to
connect to your account

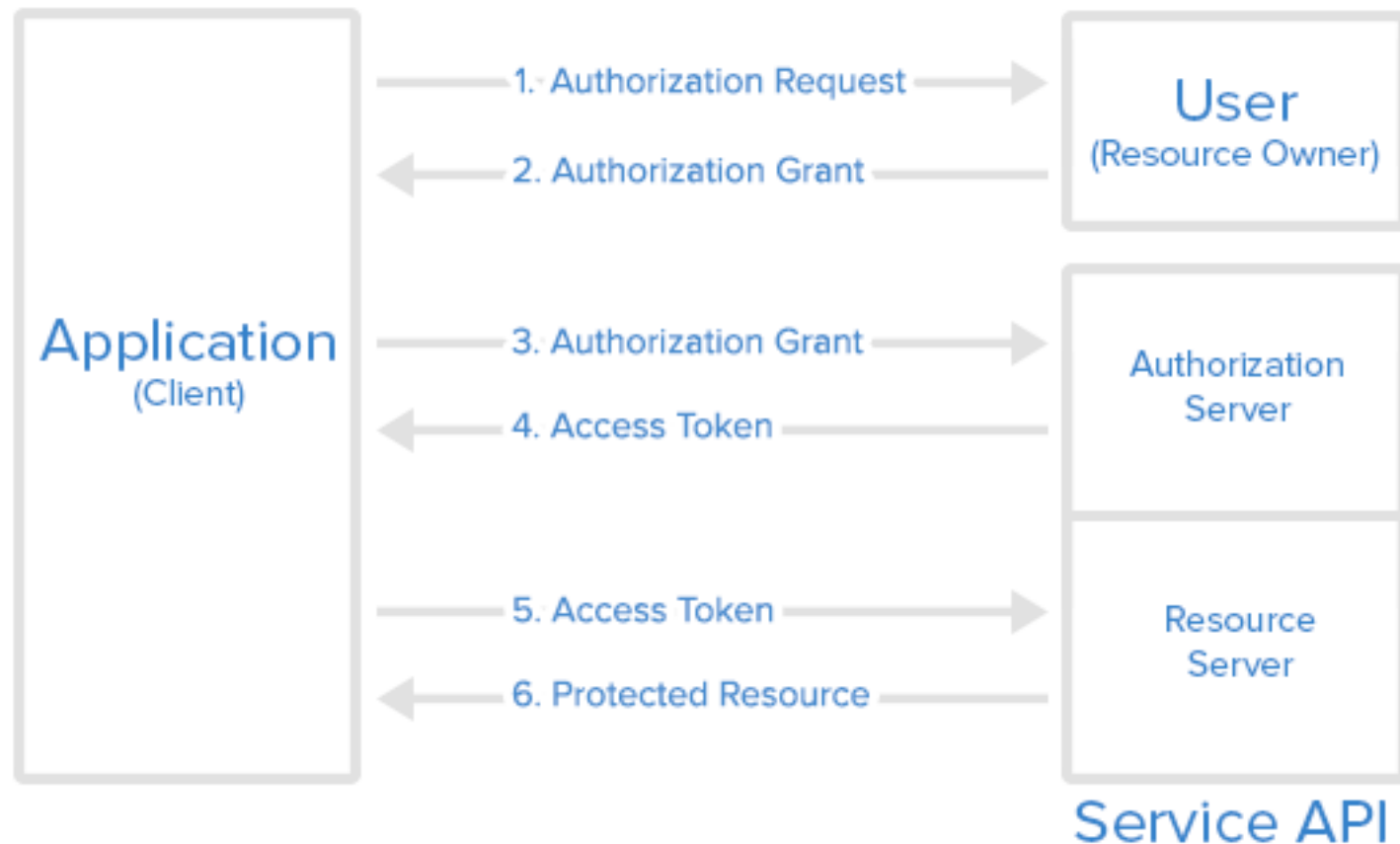
The app **Sample App** by Aaron Parecki would like the
ability to access your basic information and photos.

Allow **Sample App** access?

Deny

Allow

Fluxo Básico



Ionic + HTTP

Consumindo APIs no Ionic

Ionic + HTTP

O Ionic utiliza o módulo HTTP do Angular para realizar chamadas remotas a uma API.

Ionic + HTTP

O módulo HTTP nos permite executar os métodos **GET**, **POST**, **PUT** e **DELETE**, além de possibilitar o envio de parâmetros e cabeçalhos (header) quando necessário.

Exemplo Ionic

Consumindo API de Teste no Ionic

Importar Módulo HTTP

Você deve importar os módulos `Http`, `Response`, `Headers` e `RequestOptions` relacionados ao HTTP do Angular. Além disso, importar o módulo `toPromise`.

TS home.ts

<> home.html

```
1  import { Component } from '@angular/core';
2
3  import { Http, Response, Headers, RequestOptions } from '@angular/http';
4  import 'rxjs/add/operator/toPromise';
5
```

Injetar HTTP no Construtor

```
12  
13     constructor(public http: Http) {}  
14
```

Na classe que você deseja usar o HTTP, você deve injetar o serviço no construtor da classe, conforme o exemplo acima.

Exemplo GET

```
32 metodoGET() {
33     // Parametros que possam ser necessários no Get no formato QyeryString
34     let parametros = 'parametro_1=valor&parametro_2=valor';
35
36     // Cabeçalho, quando a API requiere o envio. Por exemplo, tokens
37     let headers = new Headers({
38         'Cache-Control': 'no-cache'
39     });
40     // Formata o header em um Array
41     let header = { headers: headers };
42     // Qual a URL do Serviço WEB no Método GET com os parâmetros
43     let url = `http://api.servico.com/?${parametros}`;
44     // Executta a requisição remota de forma assíncrona
45     this.http.get(url, header).subscribe(
46         // sucesso.json() retorna um array com os dados do servidor.
47         sucesso => console.log(sucesso.json()),
48         erro => console.log(erro)
49     );
50 }
```



```
32 metodoGET() {
```

```
33     // Parametros que possam ser necessários no Get no formato QyeryString
```

```
34     let parametros = 'parametro_1=valor&parametro_2=valor';
```

```
35  
36     // Cabeçalho, quando a API requiere o envio. Por exemplo, tokens
```

```
37     let headers = new Headers({
```

```
38         'Cache-Control': 'no-cache'
```

```
39     });
```

```
40     // Formata o header em um Array
```

```
41     let header = { headers: headers };
```

```
42     // Qual a URL do Serviço WEB no Método GET com os parâmetros
```

```
43     let url = `http://api.servico.com/?${parametros}`;
```

```
44     // Executta a requisição remota de forma assíncrona
```

Primeiro, definimos os parâmetros da requisição quando necessários. Sempre no formato query_string.

```
50 }
```

```
32 metodoGET() {  
33     // Parametros que possam ser necessários no Get no formato QyeryString  
34     let parametros = 'parametro_1=valor&parametro_2=valor';  
35  
36     // Cabeçalho, quando a API requiere o envio. Por exemplo, tokens  
37     let headers = new Headers({  
38         'Cache-Control': 'no-cache'  
39     });  
40     // Formata o header em um Array  
41     let header = { headers: headers };  
42     // Qual a URL do Serviço WEB no Método GET com os parâmetros  
43     let url = `http://api.servico.com/?${parametros}`;  
44     // Executta a requisição remota de forma assíncrona  
45  
46  
47  
48  
49  
50 }
```

Então, definimos um objeto para o cabeçalho (headers) quando for necessário.

```
32 metodoGET() {
33     // Parametros que possam ser necessários no Get no formato QyeryString
34     let parametros = 'parametro_1=valor&parametro_2=valor';
35
36     // Cabeçalho, quando a API requiere o envio. Por exemplo, tokens
37     let headers = new Headers({
38         'Cache-Control': 'no-cache'
39     });
40     // Formata o header em um Array
41     let header = { headers: headers };
42     // Qual a URL do Serviço WEB no Método GET com os parâmetros
43     let url = `http://api.servico.com/?${parametros}`;
44     // Executta a requisição remota de forma assíncrona
45
46
47
48
49
50 }
```

Em seguida, vamos criar um objeto header, adicionando os parâmetros conforme destacado.

Então especificamos qual a URL (endpoint) da API e adicionamos os parâmetros. Isto funciona apenas para métodos GET.

```
32  metodoGET() {  
33  
34  
35  
36  
37  
38      'Cache-Control': 'no-cache'  
39  });  
40  // Formata o header em um Array  
41  let header = { headers: headers };  
42  // Qual a URL do Serviço WEB no Método GET com os parâmetros  
43  let url = `http://api.servico.com/?${parametros}`;  
44  // Executa a requisição remota de forma assíncrona  
45  this.http.get(url, header).subscribe(  
46      // sucesso.json() retorna um array com os dados do servidor.  
47      sucesso => console.log(sucesso.json()),  
48      erro => console.log(erro)  
49  );  
50  }
```

```
32 metodoGET() {
```

Em seguida, vamos executar a requisição GET.

```
33  
34  
35  
36  
37  
38     'Cache-Control': 'no-cache'  
39 });  
40 // Formata o header em um Array  
41 let header = { headers: headers };  
42 // Qual a URL do Serviço WEB no Método GET com os parâmetros  
43 let url = `http://api.servico.com/?${parametros}`;  
44 // Executa a requisição remota de forma assíncrona  
45 this.http.get(url, header).subscribe(  
46     // sucesso.json() retorna um array com os dados do servidor.  
47     sucesso => console.log(sucesso.json()),  
48     erro => console.log(erro)  
49 );  
50 }
```

```
32 metodoGET() {
```


Sucesso retorna um objeto com a resposta da requisição.
Para ter acesso aos dados, basta utilizar `sucesso.json()`.

```
33  
34  
35  
36  
37  
38     'Cache-Control': 'no-cache'  
39 });  
40 // Formata o header em um Array  
41 let header = { headers: headers };  
42 // Qual a URL do Serviço WEB no Método GET com os parâmetros  
43 let url = `http://api.servico.com/?${parametros}`;  
44 // Executa a requisição remota de forma assíncrona  
45 this.http.get(url, header).subscribe(  
46     // sucesso.json() retorna um array com os dados do servidor.  
47     sucesso => console.log(sucesso.json()),  
48     erro => console.log(erro)  
49 );  
50 }
```

Método POST

Método POST

No método **POST** os parâmetros não são enviados na URL, mas no corpo (body) da requisição. Desta forma, no **POST** o método tem uma pequena mudança:

```
45     let url = `http://api.servico.com/`;
46     // Executa o POST na URL, com parametros no BODY e o header
47      this.http.post(url, parametros, header).subscribe(
48         // sucesso.json() retorna um array com os dados do servidor.
49         sucesso => console.log(sucesso.json()),
50         erro => console.log(erro)
51     );
```


Método PUT

O método **PUT** funciona da mesma maneira que o método **POST**, ou seja, os parâmetros são enviados no corpo da requisição.

Importante

Importante

Ao usar Serviços Web em Aplicativos Móveis, sempre leia com atenção a documentação da API do Serviço, para saber exatamente quais cabeçalhos devem ser enviados, quais as opções de parâmetros, qual a estrutura do JSON de resposta, entre outros.

Exemplo de API

Exemplo

Vamos usar a API do The Guardian, site de notícias internacional que disponibiliza um serviço web gratuito.

<http://open-platform.theguardian.com/>

Exemplo

O exemplo, disponível no Moodle, tem o objetivo de mostrar as novas notícias do The Guardian em um aplicativo móvel Ionic.

Explorar a API

<http://open-platform.theguardian.com/explore/>

theguardian / open platform

[get started](#)

[explore](#)

[documentation](#)

[support](#)

e.g. 'football'

Search content ▾

Add filters...

test

Show All Filters ☐ [Help](#) [Feedback](#)

<http://content.guardianapis.com/search?api-key=test>

{

```
18 // Parâmetros da API
19 let parametros = 'api-key=test';
20 // URL do The Guardian
21 let url = `http://content.guardianapis.com/search?${parametros}`;
22 // Sem headers disponíveis
23 let header = {};
24
25 this.http.get(url, header).subscribe(
26   sucesso => {
27     let news = sucesso.json().response.results;
28     // Para cada notícia, faz um push no array com o título
29     for (let i = 0; i < news.length; i++) {
30       this.noticias.push({
31         titulo: news[i].webTitle,
32         categoria: news[i].sectionName
33       });
34     }
35   },
```


Noticias

Television & radio

Dino and Andy's Skull Juice: big-name guests make this live special

World news

Rescuers battle to reach remote areas of Haiti hit by Hurricane Matthew

Business

Pound hits another 31-year low as chancellor Hammond launches Wall Street charm offensive – business live

Politics

Mike Hookem named as Ukip MEP involved in alleged fight with Steven Woolfe – live

Opinion

Shhh! We're ruining the sex lives of cod | Jules Howard

Exercício

Teste o exemplo disponível no Moodle e explore a API do The Guardian para novos testes.