BRUNNA CROCHES

PARTE 1

API | REST | SPA

Australian Pelican | Artist Image :Broinowski, Gracius Joseph | Dates:1837-1913

Guia iniciante: API

ABOUT ME

Brunna Croches

Developer Full Stack

Brunna Croches é Dev FullStack, advogada e empreendedora.

Apaixonada por tech, vem adquirido vasto conhecimento na área.

Desenvolveu projetos ricos em diversidade, buscando captar as próximas tendências e necessidades do mercado.

Neste e-book você aprenderá ou recapitulará de forma simplificada e otimizados conceitos de programação feito por ela.

let's share

SUMMARY
REST | API | SPA

1 O que é SPA/ Singe Page Applications?

O que são Single Page Applications

#PraCegoVer - Transcrição dos SlidesVocê sabe o que é SPA?
SPA, ou Single Page Application, é uma aplicação em que todas as funcionalidades estão em uma única página
Ao invés de recarregar a página toda, só uma parte do conteúdo muda

Single Page Applications (SPA) são aplicações cuja funcionalidade está concentrada em uma única página. Ao invés de recarregar toda a página ou redirecionar o usuário para uma página nova, apenas o conteúdo principal é atualizado de forma assíncrona, mantendo toda a estrutura da página estática.

Imagine um dashboard, em que os menus lateral e superior são os mesmos para todas as telas da aplicação. Ao clicar em uma opção como “Cadastro de produtos”, o usuário não precisaria recarregar toda a página para ver que no fim apenas o conteúdo central mudou. Para evitar isso, mantemos os menus fixos e alteramos apenas a parte do meio, em que estarão os formulários, tabelas, etc.

Além de otimizar a performance da aplicação, reduzindo o conteúdo a ser carregado, as SPAs têm foco na experiência do usuário, que lida com uma interface mais rápida.

Exemplos de Single Page Applications

As SPA estão presentes no nosso dia a dia já há algum tempo. Grandes exemplos disso são o Gmail, o Outlook e outras aplicações (web) de e-mail. Na mesma página temos a possibilidade de abrir uma mensagem, excluí-la, respondê-la, etc, sem que toda a estrutura seja recarregada (apenas a parte central muda).

Também há cenários híbridos, ou seja, em partes da aplicação o conceito de SPA é aplicado, enquanto em outros continua a navegação síncrona convencional. Um exemplo disso é o site Airbnb. No primeiro momento fazemos uma busca e somos direcionados para outra página. Nesse segundo ambiente temos o comportamento do tipo SPA: fazemos filtros e apenas os resultados são recarregados. Ao clicar em um dos resultados, porém, somos novamente enviados para outra página, ocorrendo a mudança de contexto.

O que usar para construir Single Page Applications?

Há atualmente no mercado diversos frameworks/bibliotecas que facilitam a criação de aplicações seguindo esse modelo. Entre os principais estão: Angular, React e Vue.js, frameworks JavaScript que trabalham com o conceito de componentes, ilustrado na **Figura 1**.

Figura 1. Conceito de componentes

Nessa imagem podemos ver que a página é formada por vários componentes: menu, barra de navegação, lista de tarefas e cada tarefa são elementos construídos separadamente para funcionar em conjunto.

Quando necessário, apenas uma tarefa ou a lista delas é atualizada, mantendo o restante da página estático e evitando carregamento desnecessário.

Quando não usar Single Page Applications

Quando a navegação representa uma "mudança de contexto", ou seja, quando as características das páginas de origem e destino são muito distintas, não é adequado usar o conceito de SPA. Nesses casos a navegação síncrona convencional faz mais sentido, pois o usuário realmente está saindo de um ambiente e indo para outro.

Um exemplo disso pode ser visto nas imagens abaixo. Note que da página de resultados da busca (**Figura 2**) para a página do curso (**Figura 3**) há muitas diferenças, ou seja, são ambientes realmente distintos. Logo, não seria adequado carregar tudo na mesma página.

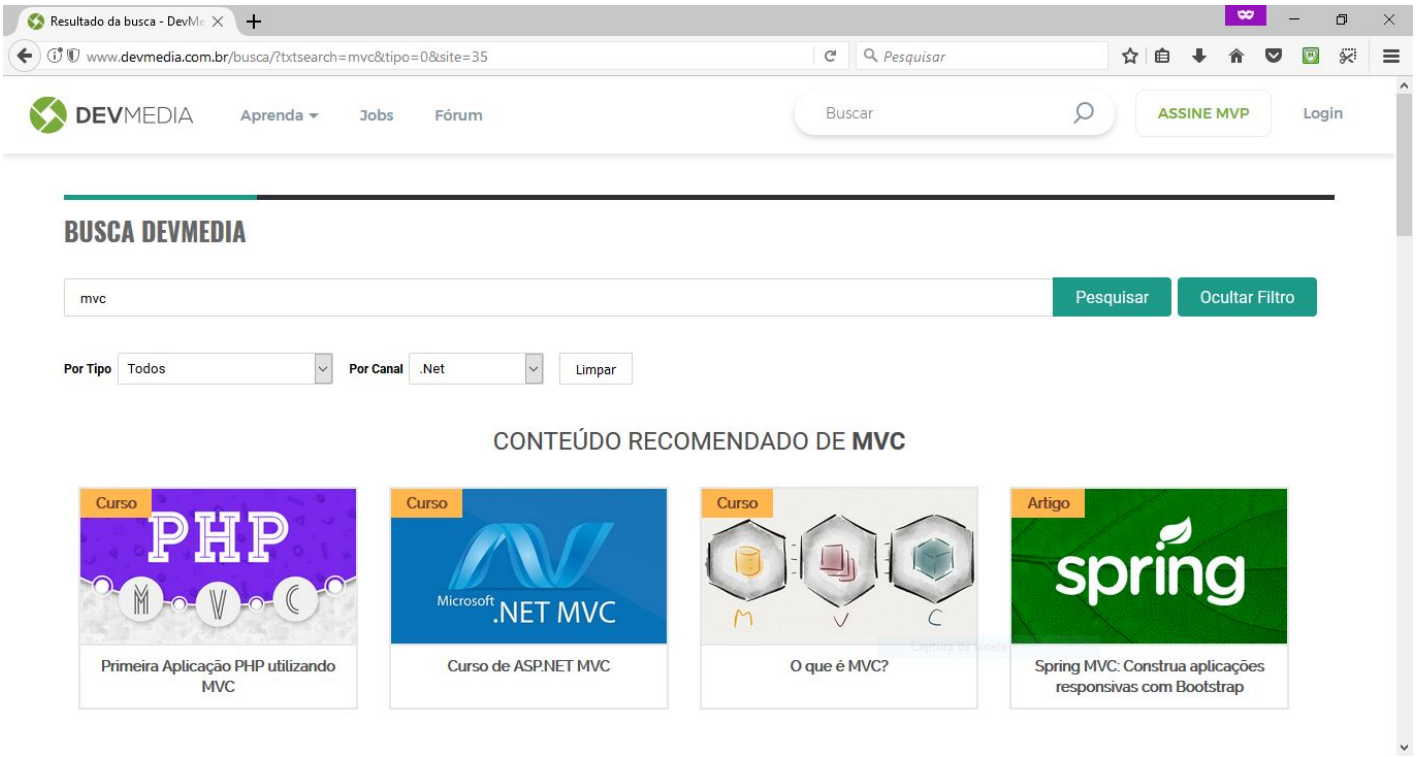


Figura 2. Página de resultados da busca

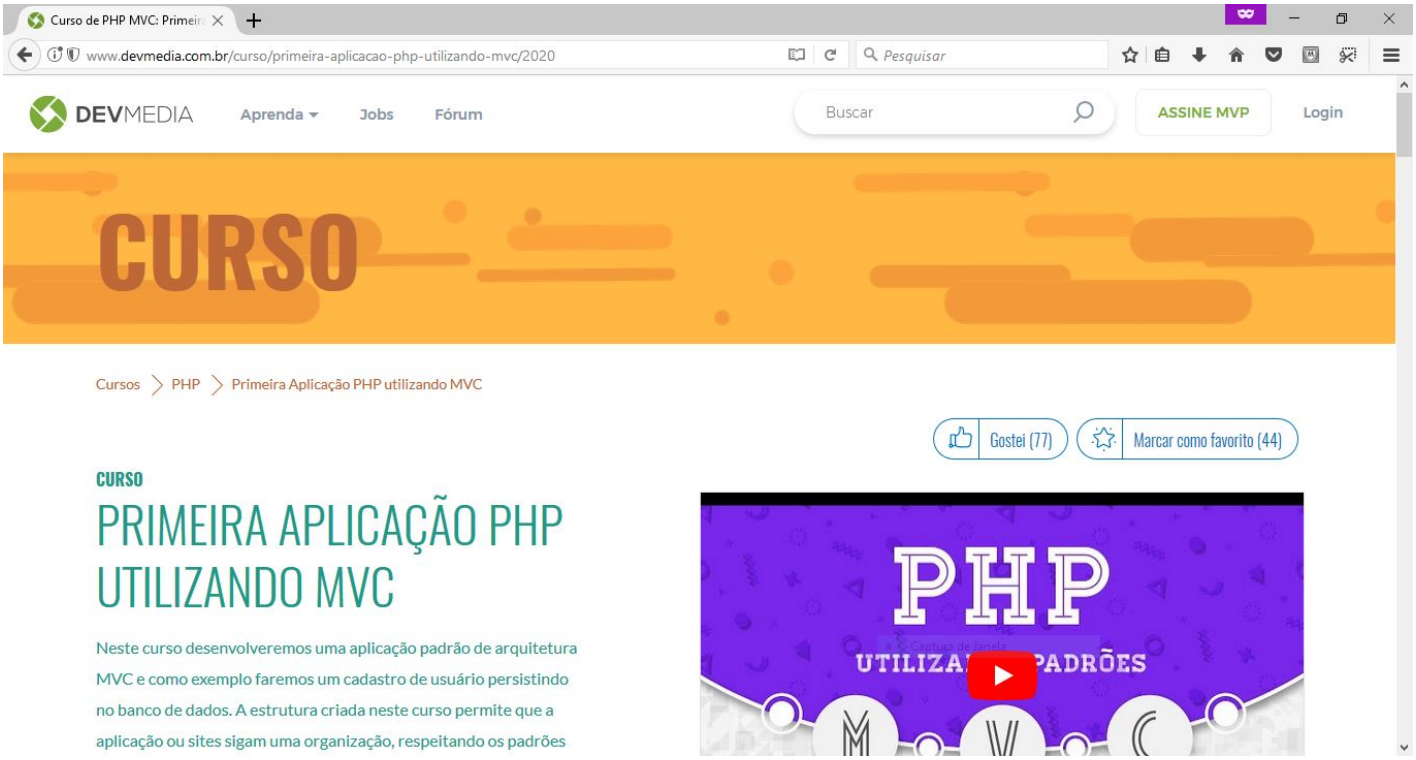


Figura 3. Página inicial do curso

Sugestão de conteúdo

- Curso O que é Angular?
- Curso O que é Vue.js?

2.0 O que é API? - Canaltech

https://www.youtube.com/watch?v=RVIJ...

https://becode.com.br/o-que-e-api-rest-e-restful/

O que é API? REST e RESTful? Conheça as definições e diferenças!

Se o título desse post chamou a sua atenção é por que você é mais uma daquelas pessoas que já tentou aprender e entender as famosas siglas: **API, REST e "RESTful".** Contudo, por algum motivo ou outro, continua com dúvidas no assunto. Talvez você até ache que REST, por exemplo, se resume a simples novos métodos HTTP que, por sua vez, podem ser usados quando são realizadas novas requisições de um cliente a um sevidor na web.

Pensando nisso, nesse post, irei esclarecer alguns pontos sobre o assunto. E, se ao final do artigo, eu conseguir atingir o meu objetivo, você estará se perguntando: **"OMG! onde posso aprender mais sobre o assunto!?"** Então vamos lá!

APIs: problemáticas e origem

É cada dia mais comum termos ***aplicações que funcionem única e exclusivamente pela Internet, sendo consumidas por navegadores*** **em desktops, notebooks ou dispositivos móveis**, isto é, independente de plataforma. ***Por outro lado, grandes empresas precisam alimentar seus softwares de gestão*** **(estoque, contabilidade, ERP e redes sociais)** ***com dados, a todo momento.***

Com essas duas problemáticas em mente (sofwares sendo acessados pela Web e empresas precisando alimentar seus sistemas) começou-se a pensar em uma **solução de sofware que permitisse a conversa entre sistemas e usuários.**

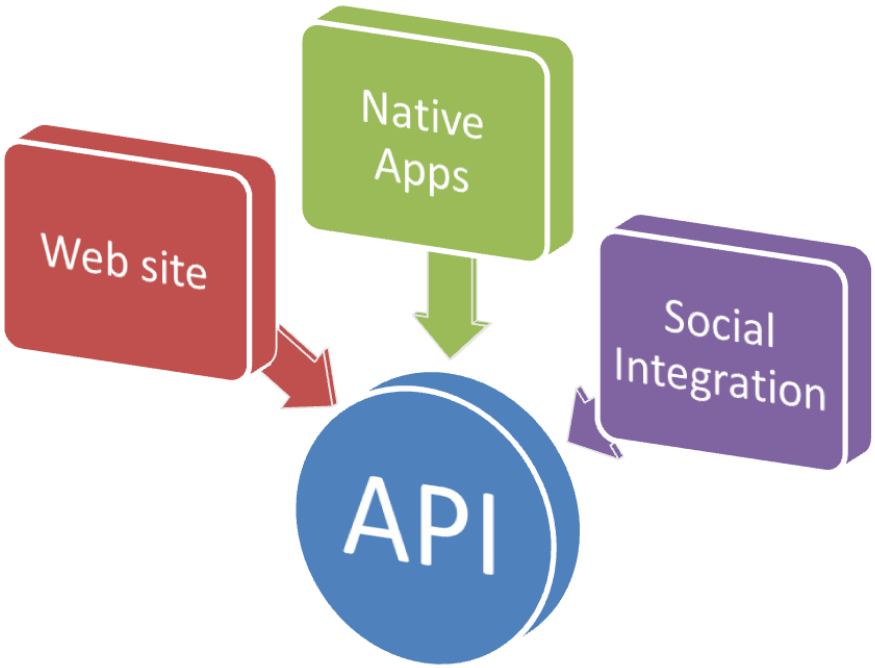
Durante anos, diversas alternativas surgiram e, de uma forma geral, essas aplicações ficaram conhecidas como **APIs.** Basicamente, o funcionamento dessas aplicações baseava-se em fornecer um ponto de acesso entre a aplicação e seu cliente, seja ele um usuário ou uma outra aplicação.

O que é API?

O acrônimo **API** que provém do inglês ***Application Programming Interface*** (Em português, significa Interface de Programação de Aplicações), trata-se de um conjunto de rotinas e padrões estabelecidos e documentados por uma aplicação A, para que outras aplicações consigam utilizar as funcionalidades desta aplicação A, sem precisar conhecer detalhes da implementação do software.

Desta forma, entendemos que as APIs permitem uma **interoperabilidade entre aplicações**. Em outras palavras, **a comunicação entre aplicações e entre os usuários.**

Exemplo de API: Twitter Developers



2) Representações

Agora que já sabemos que uma API permite a interoperabilidade entre usuários e aplicações, isso reforça ainda mais **a importância de pensarmos em algo padronizado** e, de preferência, de fácil representação e compreensão por humanos e máquinas. Isso pode soar um pouco estranho, mas veja esses três exemplos:

Representação XML

```
1
2
3
4
5
6
7
8
```

```
<endereco>
  <rua>
    Rua Recife
  </rua>
  <cidade>
    Paulo Afonso
  </cidade>
</endereco>
```

Representação JSON

```
1
2
3
4
5
6
```

```
{ endereco:
  {
  rua: Rua Recife,
  cidade: Paulo Afonso
  }
}
```

Representação YAML

```
1
2
3
```

```
endereco:
rua: rua Recife
cidade: Paulo Afonso
```

Qual deles você escolheria para informar o endereço em uma carta? Provavelmente o último, por ser de fácil entendimento para humanos, não é mesmo? Contudo, as 3 representações são válidas, pois nosso entendimento final é o mesmo, ou seja, a semântica é a mesma.

Por outro lado, você deve concordar comigo que a primeira representação **(formato XML)** é mais verbosa, exigindo um esforço extra por parte de quem está escrevendo. No segundo exemplo **(formato JSON)** já é algo mais leve de se escrever. Já o último **(formato YAML),** é praticamente como escrevemos no dia a dia.

Sendo assim, esse é o primeiro passo que precisamos dar para permitir a comunicação interoperável. E o mais legal é que essas 3 representações são válidas atualmente, ou seja, homens e máquinas podem ler, escrever e entender esses formatos.

Origem do **REST**

O HTTP é o principal protocolo de comunicação para sistemas Web existente há mais de 20 anos, e em todo esse tempo sofreu algumas atualizações. Nos anos 2000, um dos principais autores do protocolo HTTP, Roy Fielding, sugeriu, dentre outras coisas, o uso de novos métodos HTTP. Estes métodos visavam resolver problemas relacionados a semântica quando requisições HTTP eram feitas.

Estas sugestões permitiram o uso do HTTP de uma forma muito mais próxima da nossa realidade, dando sentido às requisições HTTP. Para melhor compreensão, veja os exemplos abaixo (requisições em formatos fictícios):

- GET http://www.meusite.com/usuarios
- DELETE http://www.meusite.com/usuarios/jackson
- POST http://www.meusite.com/usuarios –data {nome: joaquim}

Pela simples leitura (mesmo o método GET e DELETE sendo em inglês) é possível inferir que no **primeiro caso estamos pegando (GET) todos os usuários do site**, ou seja, teremos uma lista de todos os usuários que estão cadastrados no sistema/site. Já, **no segundo caso, estamos apagando (DELETE) o usuário Jackson**.

No **último exemplo, estamos usando o método POST, em que percebemos o envio de dados extras para cadastrar um novo usuário.**

Veja o quão simples ficou expressar o que desejamos realizar ao acessar um determinado endereço, usando verbos específicos para URLs específicas e usando dados padronizados, quando necessário.

Estes princípios apresentados fazem parte do REST! Em outras palavras, nesses exemplos, **temos: uma representação padronizada, verbos e métodos usados, bem como, URLs**.

O que é **REST**?

REST significa ***Representational State Transfer*.** Em português, **Transferência de Estado Representacional.** Trata-se de uma abstração da arquitetura da Web. Resumidamente, o **REST consiste em princípios/regras/constraints que, quando seguidas, permitem a criação de um projeto com interfaces bem definidas.** Desta forma, permitindo, por exemplo, que aplicações se comuniquem.

E RESTful... qual a diferença?

Existe uma certa confusão quanto aos termos **REST e RESTful.** Entretanto, ambos representam os mesmo princípios. A diferença é apenas gramatical. Em outras palavras, sistemas que utilizam os princípios REST são chamados de RESTful.

- **REST:** conjunto de princípios de arquitetura
- **RESTful:** capacidade de determinado sistema aplicar os princípios de REST.

Como saber mais sobre o assunto?

Apesar de tudo que vimos, aprender e entender REST está muito além de saber apenas qual método usar quando se faz uma determinada requisição. Na verdade, está muito mais ligado a dar semântica às requisições realizadas ao servidor e isso aparentemente parece fácil, mas em um estudo um pouco mais rebuscado, você perceberá que isso é só a ponta do iceberg.

Daí a pergunta que não se quer calar é... por onde começar?

A reposta é até simples. Conceitualmente você pode aprender tudo isso indo direto nas **documentações oficiais (RFC's).** Entretanto, isso pode demandar um bom tempo de aprendizado e é por isso que resolvi fazer um curso que vai te dar esse atalho.

O curso foi desenvolvido para ajudar a todos que estão iniciando no mundo REST, dando uma visão geral e também uma prática necessária ao conhecimento de aplicações REST. Se você se interessou, **é só clicar na imagem abaixo e aproveitar!**

O que é API?

API é um conjunto de rotinas e padrões de programação para acesso a um aplicativo de software ou plataforma baseado na Web. A sigla API refere-se ao termo em inglês "Application Programming Interface" que significa em tradução para o

português "Interface de Programação de Aplicativos".

Uma API é criada quando uma empresa de software tem a intenção de que outros criadores de software desenvolvam produtos associados ao seu serviço. Existem vários deles que disponibilizam seus códigos e instruções para serem usados em outros sites da maneira mais conveniente para seus usuários. O Google Maps é um dos grandes exemplos na área de APIs. Por meio de seu código original, muitos outros sites e aplicações utilizam os dados do Google Maps adaptando-o da melhor forma a fim de utilizar esse serviço.

Quando uma pessoa acessa uma página de um hotel, por exemplo, é possível visualizar dentro do próprio site o mapa do Google Maps para saber a localização do estabelecimento e verificar qual o melhor caminho para chegar até lá. Esse procedimento é realizado por meio de uma API, onde os desenvolvedores do site do hotel utilizam do código do Google Maps para inseri-lo em um determinado local de sua página.

Através das APIs, os aplicativos podem se comunicar uns com os outros sem conhecimento ou intervenção dos usuários. Elas funcionam através da comunicação de diversos códigos, definindo comportamentos específicos de determinado objeto em uma interface. A API liga as diversas funções em um site de maneira que possam ser utilizadas em outras aplicações. Sistemas de pagamento online são um bom exemplo de funcionalidade das APIs que rodam de maneira automática. De modo geral, a API é composta de uma série de funções acessíveis somente por meio de programação.

Recentemente, a utilização das APIs tem se espalhado nos plugins, que complementam a funcionalidade de um determinado programa. Os desenvolvedores de um programa principal criam uma API específica e fornecem a outros criadores, que desenvolvem plugins para aumentar o potencial e as funcionalidades do programa.

Os sistemas operacionais também possuem suas APIs com as mesmas funções descritas acima. Por exemplo, o Windows possui APIs como a Telephony API, Win16 API e Win32 API. Quando um usuário executa um programa que envolva algum processo do sistema operacional, é bem provável que o Windows faça uma conexão entre o software e alguma de suas APIs.

2.1 Critérios para uma API

Restrições para uma API RESTful

Os principais critérios para uma API ser **RESTful** são:

- Uniform Interface;
- Stateless;
- Cacheable;
- Client-Server;
- Layered system.

Sem seguir todas estas restrições, sua API não será RESTful, será apenas mais uma **implementação RPC** em cima do protocolo HTTP.

Dentre todas elas, existe uma restrição que é geralmente menos atendida, a interface uniforme (**Uniform interface**). Mas, o que significa isso?

Alcançar uma interface uniforme significa atingir quatro critérios:

- **Resource-based**: Em contraposição ao comum RPC, REST tenta lidar com recursos invés de métodos. Caso você esteja criando um post chamando */posts/create?title=lorem* você não está seguindo o padrão **REST**, devido ao tratamento de métodos na url. Nesse cenário, o ideal seria fazer uma chamada **POST** para a coleção de */posts*.
- **Manipulation of resources through representations**: O cliente acessa os recursos através de uma representação (JSON, XML, etc.), que contenha informação suficiente para manipular este no servidor, desde que tenha permissão pra isso.
- **Self-descriptive Messages:** As respostas são auto-descritivas, incluindo informação suficiente para que o cliente saiba como utilizá-las. Usando HTTP, por exemplo, é necessário uma propriedade Content-Type incluída no cabeçalho para descrever que tipo de representação é utilizada.
- Hypermedia as the engine of application state (**HATEOAS**).

CONTACT

Brunna Croches

Developer Full Stack

brunnacroches.dev

linkedin.com/brunnacroches

github.com/brunnacroches

@brunnacroches.dev

discord.com/brunnacroches

brunnacroches@gmail.com

let's share