

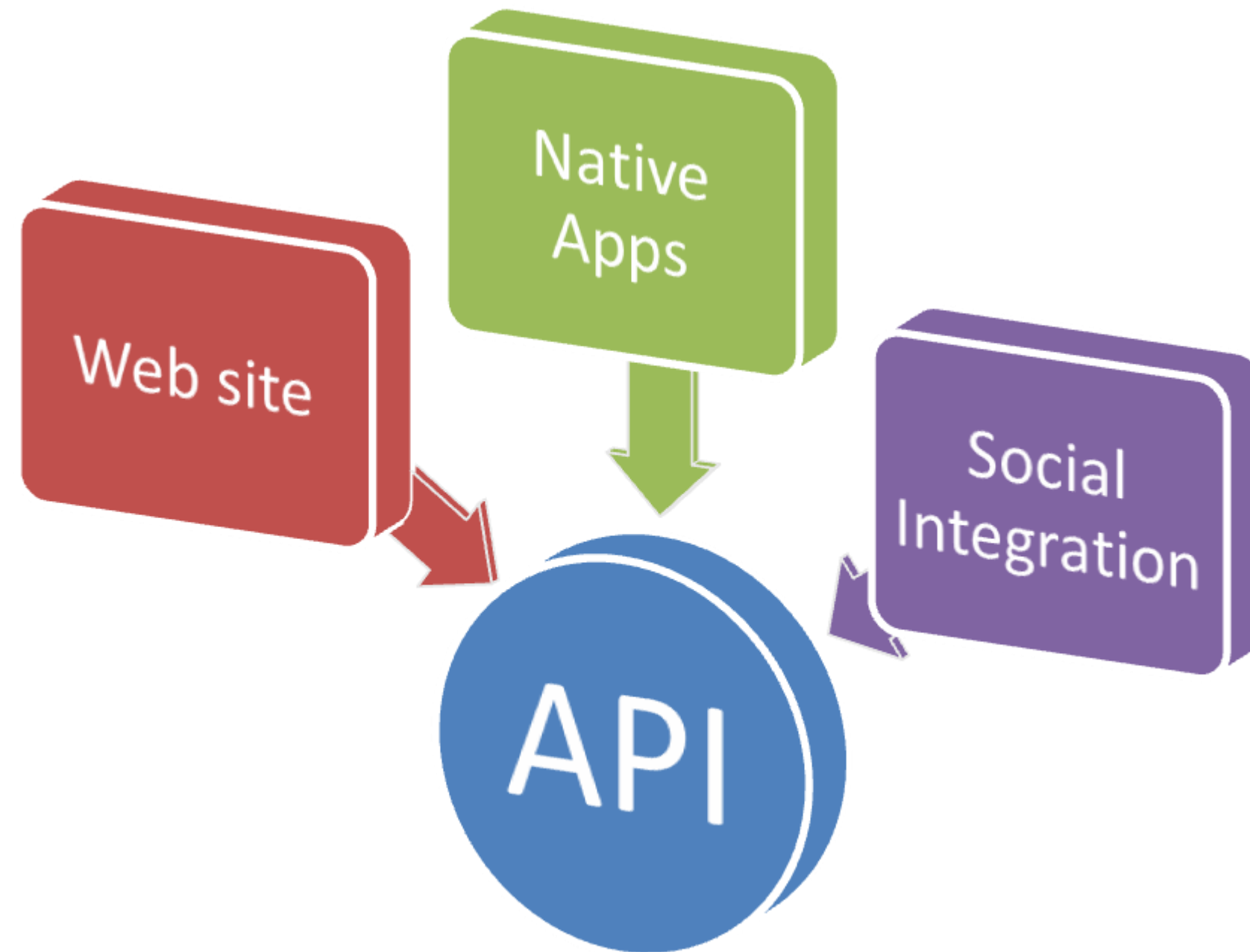
MODELAGEM WEB

(Design de API web)

O que é uma API?

O acrônimo **API - *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.

Modelagem WEB – API




Um exemplo de uso de API.

<https://viacep.com.br/>

viacep.com.br/ws/01001000/json/

<https://viacep.com.br/ws/16200001/json>



viacep.com.br/ws/16200001/json

```
{
  "cep": "16200-001",
  "logradouro": "Rua Barão do Rio Branco",
  "complemento": "",
  "bairro": "Centro",
  "localidade": "Birigüi",
  "uf": "SP",
  "unidade": "",
  "ibge": "3506508",
  "gia": "2148"
}
```

Modelagem WEB – API - Representações

Prof. Sérgio Luiz Tonsig

Agora que já sabemos que **uma API permite a interoperabilidade** entre aplicações, isso reforça ainda mais a *importância de pensarmos em algo padronizado* e, de preferência, *de fácil representação e compreensão por humanos e máquinas*.

De fácil representação e compreensão por humanos e máquinas. Isso pode soar um pouco estranho, mas veja esses três exemplos:

Representação XML	
1	<endereco>
2	<rua>
3	Rua Recife
4	</rua>
5	<cidade>
6	Paulo Afonso
7	</cidade>
8	</endereco>

Representação JSON	
1	{ endereco:
2	{
3	rua: Rua Recife,
4	cidade: Paulo Afonso
5	}
6	}

Representação YAML	
1	endereco:
2	rua: rua Recife
3	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.

Representação XML	
1	<code><endereco></code>
2	<code><rua></code>
3	Rua Recife
4	<code></rua></code>
5	<code><cidade></code>
6	Paulo Afonso
7	<code></cidade></code>
8	<code></endereco></code>

Representação JSON	
1	<code>{ endereco:</code>
2	<code>{</code>
3	<code>rua: Rua Recife,</code>
4	<code>cidade: Paulo Afonso</code>
5	<code>}</code>
6	<code>}</code>

Representação YAML	
1	<code>endereco:</code>
2	<code>rua: rua Recife</code>
3	<code>cidade: Paulo Afonso</code>

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.

Prof. Sérgio Luiz Tossin

Representação XML	
1	<endereco>
2	<rua>
3	Rua Recife
4	</rua>
5	<cidade>
6	Paulo Afonso
7	</cidade>
8	</endereco>

Representação JSON	
1	{ endereco:
2	{
3	rua: Rua Recife,
4	cidade: Paulo Afonso
5	}
6	}

Representação YAML	
1	endereco:
2	rua: rua Recife
3	cidade: Paulo Afonso

Padrões de Representações

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

Modelagens Web **mais atuais** disponibilizam APIs que os softwares clientes podem usar para interagir com um aplicativo. Uma API da Web bem projetada deve observar:

- **Independência de plataforma.** Qualquer cliente deve ser capaz de chamar a API, independentemente de como a API está implementada internamente. Isso requer o uso de protocolos padrão e ter um mecanismo pelo qual o cliente e o serviço Web pode concordar com o formato dos dados a serem trocados.
- **Evolução do serviço.** A API da Web deve ser capaz de evoluir e adicionar funcionalidade independentemente de aplicativos cliente. À medida que a API evolui, aplicativos cliente existentes devem continuar a funcionar sem modificação. Todas as funcionalidades devem ser detectáveis para que os aplicativos cliente possam utilizá-las plenamente.

Arquitetura:

Transferência de Estado Representacional (REST)

- Arquitetura para criar serviços Web.
- REST é um estilo arquitetural para a criação de sistemas distribuídos com base em hipermídia.
- A REST é independente de qualquer protocolo e não está necessariamente ligada a HTTP.
- No entanto, as implementações mais comuns de REST usam HTTP como o protocolo de aplicativo.



Existe uma certa confusão quanto aos termos **REST** e **RESTful**.
Entretanto, *ambos representam os mesmos princípios*.
A diferença é apenas gramatical.

Em outras palavras, *sistemas que utilizam os princípios REST são chamados de RESTful*.

Princípios de design mais importante de APIs RESTful usando HTTP:

- APIs REST são projetadas para *recursos*, que se tratam de qualquer tipo de objeto, dados ou serviço que possa ser acessado pelo cliente.
- Um recurso tem um *identificador*, o qual se trata de um URI que identifica exclusivamente esse recurso. Por exemplo, o URI para um pedido determinada do cliente pode ser:

<https://meusistema.com/pedido/1>

<https://meusistema.com/pedido/1>

Os softwares clientes interagem com um serviço por meio da troca de *representações de recursos*.

Muitas APIs da Web usam JSON como o formato de troca.

Por exemplo, uma solicitação GET para o URI [listado acima](#) poderia retornar este corpo de resposta:

```
{"pedido":1,"valorUnit":99.90,"codProduto":1,"qtde":1}
```


As APIs REST usam uma interface uniforme, o que *ajuda a separar* as implementações de cliente e de serviço.

Para APIs REST baseadas em HTTP, a interface uniforme inclui o **uso de verbos** HTTP padrão para executar operações em recursos. As operações mais comuns são **GET, POST, PUT, PATCH e DELETE**.

As APIs REST *são orientadas por links de hipermídia* contidos na representação.

Por exemplo, a seguir é mostrada uma representação JSON de um pedido. Ela contém links para obter ou atualizar o cliente associado ao pedido.

```
{
  "pedido":3,
  "codProduto":2,
  "qtde":4,
  "vrUnitario":16.60,
  "links": [
    {"rel":"cliente","href":"https://meusistema.com/clientes/3", "action":"GET" },
    {"rel":"cliente","href":"https://meusistema.com/clientes/3", "action":"PUT" }
  ]
}
```

Modelo de Maturidade para APIs da WEB

- **Nível 0:** defina um URI, e todas as operações são solicitações POST para esse URI.
- **Nível 1:** crie URIs separados para recursos individuais.
- **Nível 2:** use métodos HTTP para definir as operações nos recursos.
- **Nível 3:** use hipermídia (HATEOAS - Hypermedia as the Engine of Application State).

Criar API em torno de recursos

Concentre-se nas entidades comerciais que a API da Web se propõe a disponibilizar.

Por exemplo, em um sistema de comércio eletrônico, haveriam clientes, produtos, vendas (carrinho) e recebimento.

<https://meusistema.com/pedidos>



<https://meusistema.com/criarPedido>



Definir operações em termos de métodos HTTP

O protocolo HTTP define vários métodos que atribuem significado semântico a uma solicitação. Os métodos HTTP comuns usados pelas APIs da Web mais RESTful são:

- **GET**, que recupera uma representação do recurso no URI especificado. O corpo da mensagem de resposta contém os detalhes do recurso solicitado.
- **POST**, que cria um novo recurso no URI especificado. O corpo da mensagem de solicitação fornece os detalhes do novo recurso. Observe que POST também pode ser usado para disparar operações que, na verdade, não criam recursos.
- **PUT**, que cria ou substitui o recurso no URI especificado. O corpo da mensagem de solicitação especifica o recurso a ser criado ou atualizado.
- **PATCH**, que realiza uma atualização parcial de um recurso. O corpo da solicitação especifica o conjunto de alterações a ser aplicado ao recurso.
- **DELETE**, que remove o recurso do URI especificado.

Definir operações em termos de métodos HTTP (Exemplos)

Recurso	POST	GET	PUT	DELETE
/clientes	Criar um novo cliente	Obter todos os clientes	Atualização em massa de clientes	Remover todos os clientes
/clientes/1	Erro normalmente	Obter os detalhes do cliente 1	Atualizar os detalhes do cliente 1 se ele existir	Remover cliente 1
/clientes/1/pedidos	Criar um novo pedido para o cliente 1	Obter todos os pedidos do cliente 1	Atualização em massa de pedidos do cliente 1	Remover todos os pedidos do cliente 1

Prof. Sérgio Luiz Tonsig

No protocolo HTTP, formatos são especificados por meio do uso de *tipos de mídia*, também chamados de tipos MIME. Para dados não binários, a maioria das APIs da Web oferecem suporte a JSON (tipo de mídia = application/json) e, possivelmente, a XML (tipo de mídia = application/xml).

O cabeçalho Content-Type em uma solicitação ou resposta especifica o formato da representação. Aqui está um exemplo de uma solicitação POST que inclui dados JSON:

POST https://meusistema.com/clientes HTTP/1.1

Content-Type: application/**json**; charset=utf-8

Content-Length: 64

{"cpf":12345678901,"nome":"Maria Kix","Celular":"(18)989877665"}

HATEOAS - Hipertexto como o Mecanismo de Estado do Aplicativo

```
{
  "codProduto":2,
  "qtde":4,
  "valorUnitario":16.60,
  "links":[
    {
      "rel":"cliente",
      "href":"https://meusistema.com/clientes/3",
      "action":"GET",
      "types":["text/xml","application/json"]
    },
    {
      "rel":"cliente",
      "href":"https://meusistema.com/clientes/3",
      "action":"PUT",
      "types":["application/x-www-form-urlencoded"]
    },
    {
      "rel":"cliente",
      "href":"https://meusistema.com/clientes/3",
      "action":"DELETE",
      "types":[]
    }
  ]
}
```

Controle de Versão

É muito improvável que uma API da Web permaneça estática.

Uma das questões que envolvem o versionamento na WEB é a coexistência de mais de uma versão trabalhando sem problema, ao mesmo tempo.

Controle de Versão

(essa é uma das possibilidades)

GET <https://meusistema.com/clientes/3> HTTP/1.1

Custom-Header: api-version=1



ATIVIDADE

Se seu TCC utiliza alguma API Rest informe os recursos que é disponibilizado através dela e, dê um exemplo de como “consumir” um end point (ou seja, executar um chamado para a sua API).

Se seu TCC não utiliza ainda, faça uma reflexão e nos informe para qual aspecto poderia ser implementado a API Rest e que vantagem isso traria.