SENAI SUMARÉ - I2HNA

Pedro Henrique - HTTP, Vantagens e Desvantagens
João Vitor - Rest x Soap e API
Samuel Baek - Json
Isabela Martinelli - XML

MATERIAL DE TREINAMENTO WEB SERVICES

Este material tem o intuito de treinar e preparar funcionários no aspecto técnico, emocional, profissional e analista sobre o tema Web Services. A partir deste conteúdo os profissionais irão desenvolver habilidades e competências sobre os seguintes tópicos:

- ➔ Definição;
- ➔ REST x SOAP;
- ➔ HTTP;
- ➔ Json;
- ➔ XML;
- ➔ API;
- ➔ Vantagens e desvantagens.

Boa Leitura.

DEFINIÇÃO

Os *WEB SERVICES* (Serviços Web) incluem qualquer aplicação ou serviço web que utiliza modelos de padronização que utilizam a internet para efetuar a comunicação entre servidor e cliente. Atuando na 6º camada OSI (Apresentação), é responsável por criptografar e "traduzir" os dados para que eles possam ser utilizados pelos protocolos HTTP, HTTPS (Navegador WEB, E-mail) e garantindo a interação do aplicativo com o cliente.

https://www.alura.com.br/artigos/http?srsltid=AfmBOori8EUD6Nn92ffIyfVfNpgRX_Iu_AYLGB-_O6zj7R2ldhtB3HC-

https://www.alura.com.br/artigos/conhecendo-o-modelo-osi?srsltid=AfmBOopiHZx_Il0Pil-3_hGhBM8ofm4k8tqXSGipEYIkg0EinugY73Td

REST X SOAP

REST:

-PRINCÍPIOS FUNDAMENTAIS DA REST:

1. STATLESS:

Cada requisição feita ao servidor deve conter todas as informações necessárias para ser processada, sem depender de estados armazenados no servidor. Os processos e aplicações stateless não retém informações sobre interações anteriores do usuário. Cada transação é feita do zero, como se fosse a primeira vez.

2. Client-server:

A API deve separar o cliente(frontend) do servidor (backend), permitindo a evolução independente de ambos. O cliente é um dispositivo qualquer de computação, como celulares, computadores, tablets, que através de protocolos de rede se conectam a servidores para fazer uma solicitação de serviço ou recurso e esperar a resposta

3. Cacheable(cacheável):

As respostas da API devem informar se os dados podem ser armazenados em cache para melhorar o desempenho e reduzir a carga no servidor. O cache é uma estrutura computacional de armazenamento focada em manter cópias de dados que são acessados com frequência. Ele serve para melhorar a latência e o desempenho da API REST

4. Layered system (sistema em camadas):

É um padrão de design de software (design pattern) que organiza um sistema em camadas distintas e independentes. Cada camada possui uma responsabilidade específica e se comunica apenas com as camadas adjacentes, seguindo uma estrutura hierárquica. Essas camadas são:

-Camada de apresentação: exibe a interface do sistema aos usuários finais;

-Camada de negócios: contém lógica de negócios ao sistema;

-Camada de acesso a dados: responsável por realizar operações de leitura e gravação nos bancos de dados ou em outros sistemas de armazenamento de dados;

-Camada de infraestrutura:fornece suporte técnico para as outras camadas.

5. Code on demand:

São soluções de tecnologia contratadas sob demanda, ou seja, conforme as necessidades de uma empresa. Nesse sistema, uma equipe de profissionais parceira ou um sistema tecnológico são alocados para atuar em um determinado empreendimento por um período predefinido.

6. Uniform Interface:

Define o uso correto dos status codes, que são as respostas dadas ao servidor quando determinada ação acontece, como a criação de um novo recurso. Ela permite que o cliente fale com o servidor em uma única linguagem, independente do backend arquitetônico de qualquer um. Ela serve para facilitar o deslocamento entre páginas e processos, encontrar informações, realizar tarefas pelos usuários sem erros

-RECURSOS:

1. O que são?

No geral, são as informações que diferentes aplicações fornecem os seus clientes. Os recursos podem ser imagens, vídeos, textos, números ou qualquer tipo de dado. A máquina que fornece o recurso ao cliente também é chamada de servidor.

2. Como identificar e modelar recursos?

É possível identificar um recurso usando o URN/

Exemplo: coleção de recursos “clientes” e recurso único “cliente”

URN”/clientes” URN”/clientes{idCliente}

-EXEMPLOS EM DIFERENTES CONTEXTOS:

GET:

É usado para obter algum recurso hospedado no servidor

GET / HTTP/1.1

Host: www.devmedia.com.br

...

Post:

Enviar dados para o servidor:

POST /cadastro.php HTTP/1.1

Host: www.devmedia.com.br

Nome=Estevao&sobrenome=Dias....

PUT:

```
PUT http://yourserver/oslc/so/WorkTask/123

{
  "dcterms:taskname": "Check-out Leaking – Modified for Test"
}
```

DELETE:

É utilizado para excluir um recurso.

DELETE /arquivo.html HTTP/1.1

PATCH:

Pode ser utilizado na correção de jogos de computador.

PATCH /file.txt HTTP/1.1

Host: www.example.com

Content-Type: application/example

If-Match: “e0023aa4e”

Content-Length: 100

API SOAP

A API SOAP expõe funções ou operações, enquanto as API REST são orientadas por dados.

Por exemplo: considere uma aplicação com dados de funcionários que outras aplicações podem manipular. A API SOAP da aplicação pode expor uma função chamada CreateEmplyoee.

EXEMPLOS PRÁTICOS DE API RESTFUL:

REDES SOCIAIS:

-Instagram: https://medium.com/@thiagossmarques/api-do-instagram-2dc4195b728

E-COMMERCE:

-AMAZON: https://docs.aws.amazon.com/pt_br/apigateway/

SERVIÇOS:

-Google: https://developers.google.com/apis-explorer?hl=pt-b

SEGURANÇA DE APIs EM RESTFUL:

AUTENTICAÇÃO:

-API key:

Consiste em criar uma chave única, que não expira par autorizar o envio das requisições.

-OAuth 2.0:

Permite que um aplicativo de terceiros consiga acesso limitado a um serviço HTTP em nome do proprietário de um recurso, orquestrando uma interação de aprovação entre o proprietário do recurso e o serviço HTTP ou permitindo que o aplicativo de terceiros recebe acesso em seu próprio nome.

-JWT:

Possibilita a autorização de acesso a aplicativos da web e móveis, sendo uma maneira segura de transmitir informações confidenciais, como credenciais de usuário, entre diferentes sistemas, podendo ser transmitidos por meio de HTTP, HTTPS, WebSocket e outros protocolos da web

AUTORIZAÇÃO:

CONTROLE DE ACESSO:

-É possível definir os níveis de acesso para diferentes usuários e aplicativos, o que garante que cada um tenha acesso somente às informações e recursos que são relevantes para suas necessidades e atribuições.

HTTPS:

-É um protocolo de transferência de hipertexto seguro (HTTPS) é uma versão mais segura do HTTP. Serve para proteger dados sensíveis, como informações pessoais e bancárias, e garantir que terceiros não possam intercepta-las

VALIDAÇÃO DE DADOS:

-É o processo que visa assegurar a integridade das informações inseridas em um sistema on-line. Em outras palavras, significa garantir que os dados necessários para a prestação de serviços estejam devidamente completos, seguros, assertivos e consistentes.

PREVENÇÃO DE ATAQUES:

-SQL INJECTIONS:

É um tipo de ameaça de segurança que se aproveita de vulnerabilidades em sistemas que trabalham com bases de dados realizando ataques com comandos SQL; onde o atacante consegue inserir uma instrução SQL personalizada e indevida através da entrada de dados de uma aplicação, como formulários ou o URL de uma aplicação online.

-CROSS-SITE SCRIPTING - XSS:

Envolve código malicioso sendo injetado em sites de outra forma confiáveis. Essa injeção de código se aproveita de brechas em aplicações web para se apropriar de dados no navegador.

URL REST:

-ESTRUTURA URL:

É seguido por dois pontos e duas barras. Se um número de porta for específico, esse número segue o nome do host, separado por dois pontos. O nome do caminho começa com uma única barra

Exemplo: https://www.google.com.br/?hl=pt-BR

Você pode também utilizar a tag html <base> para especificar o endereço URL utilizado por todos os endereços relativos contidos dentro de um documento

O termo “URL path” se refere à parte de uma URL que indica a localização de um recurso específico dentro de um site

Exemplo:

No endereço: https://www.exemplo.com/produtos/eletronicos/televisores, o URL PATH é /produtos/eletrônicos/televisores

O query parameter é um componente essencial nas URLs que permite a passagem de informações entre o cliente e o servidor

Sintaxe: uma chave e um valor, separados por um sinal de igual

Exemplo: https://www.exemplo.com/produtos?categoria=eletronicos

“categoria” é a chave e “eletronicos” é o valor

BOAS PRÁTICAS DE NOMEAÇÃO URL:

-Palavras simples e descritivas;

-Palavras no idioma do público alvo;

-Utilização da codificação UTF-8 conforme necessário;

-Não utilizar caracteres não ASCII;

-Não utilizar números de ID longos e ilegíveis no URL

EXEMPLOS DE URLs DE API REST

GitHub: https://api.github.com/user/Bard Recupera informações sobre o usuário “Bard”

OpenWeatherMap: https://api.openweathermap.org/data/2.5/weather?q=London obtém dados meteorológicos para Londres

Unsplash: https://api.unsplash.com/photos/random?count=1 recupera uma foto aleatória

Fontes: https://apidog.com/pt/blog/rest-api-url-best-practices-examples/

https://br.hubspot.com/blog/marketing/api-do-instagram

https://aws.amazon.com/pt/what-is/api/#:~:text=APIs%20s%C3%A3o%20mecanismos%20que%20permitem,meteorol%C3%B3gico%20cont%C3%A9m%20dados%20meteorol%C3%B3gicos%20di%C3%A1rios.

HTTP (Estrutura, CRUD, Uso, Código de Status (Já tem))

O HTTP (Hypertext Transfer Protocol) é a base para que o cliente e o servidor se comuniquem, como acesso em páginas web, imagens, arquivos. Tudo isso através de mensagens padronizadas (WEB SERVICES). O HTTP é o principal e o protocolo mais utilizado no uso de APIs.

Sua estrutura é baseada por meio do modelo cliente-servidor. Através do requerimento do cliente (pergunta) o servidor retornará (resposta) ao cliente através de um código de status. Um exemplo é uma pesquisa na internet, utilizando o URL (endereço físico) como "*HTTPS://www.lojadetenis.com.br*" + o requerimento do

cliente (URN) "*/tenis-modelo-um*" o servidor irá devolver uma URI "*HTTPS://www.lojadetenis.com.br/tenis-modelo-um".*
https://www.dltec.com.br/blog/redes/crud-em-redes-entenda-porque-voce-precisa-aprender/

Como o servidor sempre retornará à requisição do cliente ele também retornará erros ou requisições não atendidas. Esses retornos são conhecidos como "Códigos de Status". Basicamente, existem cinco tipos de códigos de status, aqueles começados com o número 1 (1XX), os começados com o número 2 (2XX), os começados com o número 3 (3XX), os começados em número 4 (4XX) e os começados em número 5 (5XX).

1XX – Os códigos de status começados em 1 significam que o requerimento de API do cliente foi recebido e está sendo processado, informando que ele aguarda uma resposta.

- 100 – Indica que o servidor recebeu a solicitação do cliente e que o cliente deve continuar com a solicitação.
- 101 – Sinaliza que o servidor está mudando para um protocolo diferente, conforme o cliente pediu.

2XX – Já os códigos começados em 2 significam que o requerimento do cliente foi bem-sucedido. O servidor informará esse código juntamente com os recursos de API solicitados.

- 200 – Indica que a solicitação foi bem-sucedida.
- 202 – Indica que o servidor analisou a solicitação, ela foi aceita, mas ainda não iniciou ou concluiu o processamento.

3XX – A classe de códigos que se inicia com o número 3 indica que o cliente precisa realizar ações adicionais para que seu API seja atendido. Essa classe geralmente aparece em redirecionamento de *URL's.*

- 301 – Aponta que o redirecionamento não pôde ser concluído por conta da URL requerida não ser daquele navegador.
- 302 – A URL foi encontrada e o endereço atual está redirecionando o cliente para outro endereço.

4XX – Os códigos iniciados com o número 4 sinalizam uma mensagem de erro por parte do cliente através da solicitação da API.

- 400 – Esse código indica uma solicitação inválida que está impossibilitando a API de ser processada por conta de um requerimento errado por parte do cliente. Os erros mais comuns que geram esse código são erros de sintaxe.

- 404 – Informa que a API solicitada não pôde ser encontrada no servidor, mas não informa se o recurso está ausente permanentemente ou apenas temporário, algumas das causas mais comuns são URL digitadas incorretamente.

5XX – Os códigos iniciados com o número 5 indicam uma solicitação com falha devido a um erro de servidor ao processar um requerimento.

- 500 – O servidor não pode atender à solicitação devido a uma condição inesperada. Geralmente um erro do navegador.
- 503 – O serviço não pode lidar com a solicitação porque ficou sem recurso ou está em manutenção. Geralmente enviam esse código devido a um erro temporário

https://www.moesif.com/blog/technical/api-design/Which-HTTP-Status-Code-To-Use-For-Every-CRUD-App/

A partir do código de status podemos reconhecer o erro e utilizando o CRUD (Create, Read, Update, Delete) ou verbos HTTP (Post, Get, Put, Delete) para otimizarmos, deletarmos, solicitarmos ou corrigir algum problema ou algum requerimento para a nossa API. Ou seja, utilizamos o HTTP para solicitarmos informações ao servidor a fim de receber uma resposta.

Alguns exemplos do uso do CRUD no HTTP são, por exemplo, se quisermos atualizar apenas o nome de um usuário, podemos enviar uma requisição PATCH passando apenas o novo nome. Dessa forma, o método PATCH é indicado quando o objetivo é fazer uma atualização parcial de dados de um recurso.

Outro exemplo, se quisermos excluir permanentemente um usuário, podemos enviar uma requisição DELETE informando o ID desse usuário na URL. O método DELETE deve ser utilizado quando o objetivo é remover ou excluir de forma permanente um determinado recurso.

https://doc.magentochina.org/swagger/#/eavAttributeSetRepositoryV1/eavAttributeSetRepositoryV1DeleteByIdDelete

JSON (JavaScript Object Notation)

O JSON é um formato que armazena informações estruturadas e é principalmente usado para transferir dados entre um servidor e um cliente. O arquivo é basicamente uma alternativa simples e mais leve ao XML (*Extensive Markup Language*), que tem funções similares.

Sintaxe: A ideia utilizada pelo JSON para representar informações é simples: para cada valor representado, atribui-se um nome (ou rótulo) que descreve o seu significado. Esta sintaxe é derivada da forma utilizada pelo JavaScript para representar informações.

número real:

"altura": 1.72

string:

"site": “www.devmedia.com.br”

booleano:

"casado": true

Formato: JSON (JavaScript Object Notation) é um formato aberto usado como alternativa ao XML para a transferência de dados estruturados entre um servidor de Web e uma aplicação Web. Sua lógica de organização tem semelhanças com o XML, mas possui notação diferentes

O delimitador “ { ” marca o início de uma seção e o “ } “ marca seu fim

Os pares de valor e atributo são separados por “ : “

e seus valores, quando texto, ficam entre aspas (números, por exemplo, não recebem as aspas).

Valores: Um par nome/valor deve ser representado pelo nome entre aspas duplas, seguido de dois pontos, seguido do valor. Os valores podem possuir apenas 3 tipos básicos:

- numérico (inteiro ou real)
- booleano (False ou true)
- string

Tipos de dados:

- Booleano (true ou false)

{“married”:”false”)

- Array (é uma coleção ordenada de valores. É cercado por colchetes **[]** e cada valor dentro é separado por uma vírgula.)

"students":[
{"firstName":"Tom", "lastName":"Jackson"},
{"firstName":"Linda", "lastName":"Garner"},
{"firstName":"Adam", "lastName":"Cooper"}
]

- Nulo (valor vazio)

{“bloodType”:”null”}

- Número (segue o formato de ponto flutuante de precisão dupla do JavaScript)

{“age”:”30”}

- Objeto (conjunto de pares de nomes ou valores inseridos entre { })

“employees”: {"firstName":"Tom", "lastName":"Jackson”}

- Sequência (é uma sequência definida de zero ou mais caracteres Unicode. É colocado entre duas aspas duplas.)

"firstName":"Tom”

Fontes:
https://www.oracle.com/br/database/what-is-json/
https://ceweb.br/guias/dados-abertos/capitulo-38/
https://www.hostinger.com.br/tutoriais/o-que-e-json

XML (Extensible Markup Language)
Isabela

Sintaxe básica:
Indentation:

- Documentos XML devem conter um elemento raiz que é o pai de todos os outros elementos;

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

- Em XML, não é possível não fechar uma tag. Todas as tags devem ser fechadas.
- As tags em XML são case sensitive. A tag <Letter> é diferente da tag <letter>.
- Elementos XML podem ter atributos em pares de nome/valor como em HTML. Em XML, o valor do atributo de sempre estar entre aspas.

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

Alguns caracteres têm um significado especial em XML. Se você colocar o caractere “<” dentro de um elemento XML, isso vai gerar um erro pois o programa interpreta o caractere como o início de um novo elemento. Para evitar esse erro, o caractere é substituído por uma referência de entidade.

Existem 5 referências de entidade pré-definidas em XML:

&lt;	<	menos que (less than)
&gt;	>	mais que (greater than)
&amp;	&	e comercial (ampersand)
&apos;	‘	apóstrofe (apostrophe)
&quot;	“	aspas (quotation mark)

Obs: Apenas “<” e “&” precisam obrigatoriamente serem substituídos, mas é um bom hábito substituir > com &gt também.

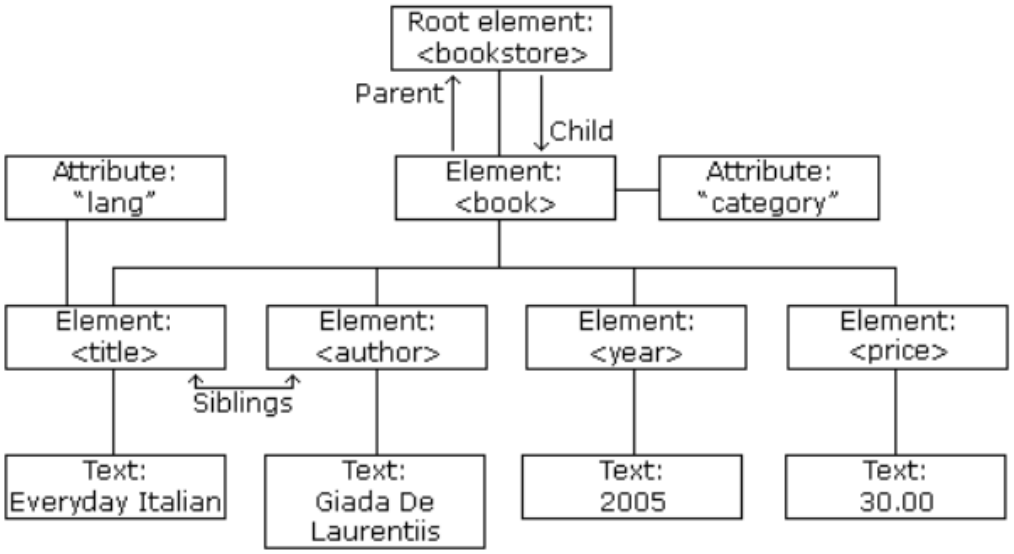
Estrutura de árvore em XML:

- Documentos XML são formados por elementos de árvore;
- Uma árvore XML começa por um elemento raiz e galhos da raiz até os elementos filhos;
- Todos os elementos podem ter sub-elementos (elementos filhos);
- Os termos pais, filho e irmão são usados para descrever as relações entre elementos;
- Todos os elementos podem ter um conteúdo de texto e atributos.

Elementos:

Um elemento pode conter:

- texto;
- atributos;
- outros elementos;
- ou uma junção desses.



No exemplo acima

<title>, <author>, <year>, e <price> tem um conteúdo de texto (como 29.99).

<bookstore> e <book> tem um conteúdo de elementos.

<book> tem um atributo (category="children").

Elementos XML devem seguir essas regras de nomeação:

- Os nomes dos elementos são case sensitive;
- Os nomes dos elementos devem começar com uma letra ou underscore;
- Nomes de elementos não podem começar com xml (ou XML, ou Xml, etc.);
- Nomes de elementos podem conter letras, números, hifens, underscores e pontos;
- Nomes de elementos não podem conter espaços.

Qualquer nome pode ser usado, nenhuma palavra é reservada (exceto xml).

Melhores práticas de nomeação:

- Crie nomes descritivos, como: <pessoa>, <primeironome>, <sobrenome>;

- Crie nomes curtos e simples, com <titulo_livro> em vez de <o_titulo_de_um_livro>
- Evite “-”. Se você nomear algo como “primeiro-nome”, alguns softwares podem achar que você que subtrair “primeiro” de “nome”;
- Evite “.”. Se você nomear algo como “primeiro.nome”, alguns softwares podem achar que “primeiro” é uma propriedade do objeto “nome”;
- Letras de fora do alfabeto da língua inglesa com “éòá” são permitidos em XML, mas tome cuidado com problemas se o seu software não tiver suporte para elas.

Atributos:

Valores de atributos devem sempre estar entre aspas. Podem ser usados aspas simples ou duplas.

Por exemplo para definir o gênero de uma pessoa, o elemento <person> pode ser escrito assim:

```
<person gender="female">
```

Ou assim:

```
<person gender='female'>
```

Se o próprio atributo conter duas aspas você pode usar aspas simples, como no exemplo a seguir:

```
<gangster name='George "Shotgun" Ziegler'>
```

Ou você pode usar entidades de caractere:

```
<gangster name="George &quot;Shotgun&quot; Ziegler">
```

Algumas coisa para considerar quando usar atributos:

- atributos não podem conter diversos valores (elementos podem);
- atributos não podem ter estruturas de árvores (elementos podem);
- atributos não são facilmente expansíveis (para mudanças futuras).

Atributos de XML para Metadados:

Às vezes referências ID são atribuídas a elementos. Esses IDs podem ser usados para identificar mais facilmente elementos XML. Esse exemplo demonstra isso:

```xml
<messages>
  <note id="501">
    <to>Tove</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Tove</from>
    <heading>Re: Reminder</heading>
    <body>I will not</body>
  </note>
</messages>
```

Os atributos ID acima são para identificar diferentes notas. Não parte da nota em si. Metadados (dados sobre dados) devem ser guardados com atributos, e os dados em si devem ser guardados como elementos.

Declaração XML (DTD):

DTD significa Document Type Definition (Definição de tipo do documento).

O propósito de um DTD é definir a estrutura, os elementos e os atributos de um documento XML.

Note.dtd:

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

O DTD acima é interpretado da seguinte forma:

!DOCTYPE note - Define que o elemento raiz do documento é note

!ELEMENT note - Define que o elemento note contém os elementos: “to, from, heading, body”

!ELEMENT to - Define o elemento “to” para ser do tipo “#PCDAT”

!ELEMENT from - Define o elemento “from” para ser do tipo “#PCDATA”

!ELEMENT heading - Define o elemento “heading” para ser do tipo “#PCDATA”

!ELEMENT body - Define o elemento “body” para ser do tipo “#PCDATA”

Obs: #PCDATA significa parseable character data (dados de caracteres utilizáveis).

Usando DTD para declaração de entidade:

Uma declaração DOCTYPE também pode ser usada para definir caracteres ou strings especiais, usados no documento:

Exemplo:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE note [
<!ENTITY nbsp " ">
<!ENTITY writer "Writer: Donald Duck.">
<!ENTITY copyright "Copyright: W3Schools.">
]>

<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
<footer>&writer; &copyright;</footer>
</note>
```

Obs: Uma entidade tem três partes: ela começa com “&”, depois vem o nome da entidade, e termina com “;”

Quando usar DTD:

- Com um DTD, grupos independentes de pessoas podem concordar em usar um DTD padrão para intercâmbio de dados;
- Com um DTD, você pode verificar se os dados que você recebeu de fora são válidos;
- Você também pode usar DTD para verificar seus próprios dados.

Quando não usar um DTD:

- O XML não exige um DTD;
- Quando você está experimentando XML, ou quando você está trabalhando com documentos pequenos de XML, criar DTDs pode ser uma perda de tempo;

- Se você desenvolve aplicações, espere até que as especificações estejam estáveis antes de adicionar um DTD. De outra maneira seu software pode parar de funcionar por conta de erros de validação.

https://www.w3schools.com/xml/

REST X SOAP

REST:

-PRINCÍPIOS FUNDAMENTAIS DA REST:

1. STATLESS:

Cada requisição feita ao servidor deve conter todas as informações necessárias para ser processada, sem depender de estados armazenados no servidor. Os processos e aplicações stateless não retém informações sobre interações anteriores do usuário. Cada transação é feita do zero, como se fosse a primeira vez.

2. Client-server:

A API deve separar o cliente(frontend) do servidor (backend), permitindo a evolução independente de ambos. O cliente é um dispositivo qualquer de computação, como celulares, computadores, tablets, que através de protocolos de rede se conectam a servidores para fazer uma solicitação de serviço ou recurso e esperar a resposta

3. Cacheable(cacheável):

As respostas da API devem informar se os dados podem ser armazenados em cache para melhorar o desempenho e reduzir a carga no servidor. O cache é uma estrutura computacional de armazenamento focada em manter cópias de dados que são acessados com frequência. Ele serve para melhorar a latência e o desempenho da API REST

4. Layered system (sistema em camadas):

É um padrão de design de software (design pattern) que organiza um sistema em camadas distintas e independentes. Cada camada possui uma responsabilidade específica e se comunica apenas com as camadas adjacentes, seguindo uma estrutura hierárquica. Essas camadas são:

-Camada de apresentação: exibe a interface do sistema aos usuários finais;

-Camada de negócios: contém lógica de negócios ao sistema;

-Camada de acesso a dados: responsável por realizar operações de leitura e gravação nos bancos de dados ou em outros sistemas de armazenamento de dados;

-Camada de infraestrutura:fornece suporte técnico para as outras camadas.

5. Code on demand:

São soluções de tecnologia contratadas sob demanda, ou seja, conforme as necessidades de uma empresa. Nesse sistema, uma equipe de profissionais parceira ou um sistema tecnológico são alocados para atuar em um determinado empreendimento por um período predefinido.

6. Uniform Interface:

Define o uso correto dos status codes, que são as respostas dadas ao servidor quando determinada ação acontece, como a criação de um novo recurso. Ela permite que o cliente fale com o servidor em uma única linguagem, independente do backend arquitetônico de qualquer um. Ela serve para facilitar o deslocamento entre páginas e processos, encontrar informações, realizar tarefas pelos usuários sem erros

-RECURSOS:

1. O que são?

No geral, são as informações que diferentes aplicações fornecem os seus clientes. Os recursos podem ser imagens, vídeos, textos, números ou qualquer tipo de dado. A máquina que fornece o recurso ao cliente também é chamada de servidor.

2. Como identificar e modelar recursos?

É possível identificar um recurso usando o URN/

Exemplo: coleção de recursos “clientes” e recurso único “cliente”

URN”/clientes” URN”/clientes{idCliente}

-EXEMPLOS EM DIFERENTES CONTEXTOS:

GET:

É usado para obter algum recurso hospedado no servidor

GET / HTTP/1.1

Host: www.devmedia.com.br

Post:

Enviar dados para o servidor:

POST /cadastro.php HTTP/1.1

Host: www.devmedia.com.br

Nome=Estevao&sobrenome=Dias....

PUT:

```
PUT http://yourserver/oslc/so/WorkTask/123
```

```
{
  "dcterms:taskname": "Check-out Leaking – Modified for Test"
}
```

DELETE:

É utilizado para excluir um recurso.

DELETE /arquivo.html HTTP/1.1

PATCH:

Pode ser utilizado na correção de jogos de computador.

PATCH /file.txt HTTP/1.1

Host: www.example.com

Content-Type: application/example

If-Match: “e0023aa4e”

Content-Length: 100

API SOAP

A API SOAP expõe funções ou operações, enquanto as API REST são orientadas por dados.

Por exemplo: considere uma aplicação com dados de funcionários que outras aplicações podem manipular. A API SOAP da aplicação pode expor uma função chamada CreateEmplyoee.

EXEMPLOS PRÁTICOS DE API RESTFUL:

REDES SOCIAIS:

-Instagram: https://medium.com/@thiagossmarques/api-do-instagram-2dc4195b728

E-COMMERCE:

-AMAZON: https://docs.aws.amazon.com/pt_br/apigateway/

SERVIÇOS:

-Google: https://developers.google.com/apis-explorer?hl=pt-br

SEGURANÇA DE APIs EM RESTFUL:

AUTENTICAÇÃO:

-API key:

Consiste em criar uma chave única, que não expira par autorizar o envio das requisições.

-OAuth 2.0:

Permite que um aplicativo de terceiros consiga acesso limitado a um serviço HTTP em nome do proprietário de um recurso, orquestrando uma interação de aprovação entre o proprietário do recurso e o serviço HTTP ou permitindo que o aplicativo de terceiros recebe acesso em seu próprio nome.

-JWT:

Possibilita a autorização de acesso a aplicativos da web e móveis, sendo uma maneira segura de transmitir informações confidenciais, como credenciais de

usuário, entre diferentes sistemas, podendo ser transmitidos por meio de HTTP, HTTPS, WebSocket e outros protocolos da web

AUTORIZAÇÃO:

CONTROLE DE ACESSO:

-É possível definir os níveis de acesso para diferentes usuários e aplicativos, o que garante que cada um tenha acesso somente às informações e recursos que são relevantes para suas necessidades e atribuições.

HTTPS:

-É um protocolo de transferência de hipertexto seguro (HTTPS) é uma versão mais segura do HTTP. Serve para proteger dados sensíveis, como informações pessoais e bancárias, e garantir que terceiros não possam intercepta-las

VALIDAÇÃO DE DADOS:

-É o processo que visa assegurar a integridade das informações inseridas em um sistema on-line. Em outras palavras, significa garantir que os dados necessários para a prestação de serviços estejam devidamente completos, seguros, assertivos e consistentes.

PREVENÇÃO DE ATAQUES:

-SQL INJECTIONS:

É um tipo de ameaça de segurança que se aproveita de vulnerabilidades em sistemas que trabalham com bases de dados realizando ataques com comandos SQL; onde o atacante consegue inserir uma instrução SQL personalizada e indevida através da entrada de dados de uma aplicação, como formulários ou o URL de uma aplicação online.

-CROSS-SITE SCRIPTING - XSS:

Envolve código malicioso sendo injetado em sites de outra forma confiáveis. Essa injeção de código se aproveita de brechas em aplicações web para se apropriar de dados no navegador.

URL REST:

-ESTRUTURA URL:

É seguido por dois pontos e duas barras. Se um número de porta for específico, esse número segue o nome do host, separado por dois pontos. O nome do caminho começa com uma única barra

Exemplo: https://www.google.com.br/?hl=pt-BR

Você pode também utilizar a tag html <base> para especificar o endereço URL utilizado por todos os endereços relativos contidos dentro de um documento

O termo “URL path” se refere à parte de uma URL que indica a localização de um recurso específico dentro de um site

Exemplo:

No endereço: https://www.exemplo.com/produtos/eletronicos/televisores, o URL PATH é /produtos/eletrônicos/televisores

O query parameter é um componente essencial nas URLs que permite a passagem de informações entre o cliente e o servidor

Sintaxe: uma chave e um valor, separados por um sinal de igual

Exemplo: https://www.exemplo.com/produtos?categoria=eletronicos

“categoria” é a chave e “eletronicos” é o valor

BOAS PRÁTICAS DE NOMEAÇÃO URL:

-Palavras simples e descritivas;

-Palavras no idioma do público alvo;

-Utilização da codificação UTF-8 conforme necessário;

-Não utilizar caracteres não ASCII;

-Não utilizar números de ID longos e ilegíveis no URL

EXEMPLOS DE URLs DE API REST

GitHub: https://api.github.com/user/Bard Recupera informações sobre o usuário "Bard"

OpenWeatherMap: https://api.openweathermap.org/data/2.5/weather?q=London obtém dados meteorológicos para Londres

Unsplash: https://api.unsplash.com/photos/random?count=1 recupera uma foto aleatória

Fontes: https://apidog.com/pt/blog/rest-api-url-best-practices-examples/

https://br.hubspot.com/blog/marketing/api-do-instagram

https://aws.amazon.com/pt/what-is/api/#:~:text=APIs%20s%C3%A3o%20mecanismos%20que%20permitem,meteorol%C3%B3gico%20cont%C3%A9m%20dados%20meteorol%C3%B3gicos%20di%C3%A1rios.

https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods/DELETE

https://www.devmedia.com.br/http-get-e-post/41221#modulo-mvp

VANTAGENS E DESVANTAGENS

VANTAGENS: Soluciona problemas de comunicação para HTTP, XML, e UDDI para que duas plataformas interajam, simplifica a troca de informações e tem custo menos que serviços API, protegidos com padrão de segurança SSL.

DESVANTAGENS: Não é compatível com XMP e AJAX, necessita de máquinas especializada para o desenvolvimento do serviço, a comunicação entre sistemas pode introduzir latência especialmente quando estão localizados geograficamente distantes.

https://blog.engdb.com.br/api-e-web-service/

https://pt.linkedin.com/pulse/vantagens-e-desvantagens-de-web-services-na-cria%C3%A7%C3%A3o-aplica%C3%A7%C3%B5es-pedro-b1v8f