FIAP ON

WEBSERVICES &
RESTFUL TECHNOLOGIES

RESTFUL - JSON

EDUARDO GALEGO

1

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE COMANDOS DE PROMPT DO SISTEMA OPERACIONAL

SUMÁRIO

1 RESTFUL - JSON

Iniciamos os nossos estudos em Web Services.

Para um desenvolvedor Full Stack, os conhecimentos de Web Services são fundamentais para a comunicação entre os componentes.

Pode-se encontrar Web Service:

1. Na comunicação de uma aplicação mobile com um *back-end*.
2. Na comunicação entre um front-end e um BFF (*back-end for front-end*).
3. Na comunicação de um dispositivo IOT com um *storage*.
4. Na comunicação de uma aplicação com serviços de IA.

1.1 Introdução à Web Service

Você (com certeza) conhece a Web, mas talvez nunca tenha se indagado: “Por que a Web faz tanto sucesso?” A Web conta hoje com mais de 4,5 milhões de usuários (algo próximo de 60% da população mundial), gastando em média 6 horas e 43 minutos na rede.

Ao desenvolver uma aplicação na Web, normalmente o desenvolvedor opta por utilizar linguagens e protocolos conhecidos como **Open Standards**. Isso garante que sua aplicação funcionará na grande maioria dos navegadores, consequentemente, para a grande maioria dos usuários.

Não importa se o usuário utiliza este ou aquele sistema operacional, se está acessando por um celular ou por um desktop: se utilizar um navegador compatível com as tecnologias padrões da Web, ele irá funcionar!

Linguagens como: HTML, CSS, JavaScript e protocolos como HTTP são utilizados de forma abrangente no mercado, uma vez que eles não pertencem a uma empresa, mas sim a um consórcio.

Dessa forma, os desenvolvedores e empresas fornecedoras de software não correm o risco de algumas dessas linguagens se tornarem proprietárias no futuro e acabar pagando royalties por uso.

O consórcio mais conhecido e que detém grande parte das tecnologias e protolocos é a W3C. Agradeça a ela por utilizar a Web que você conhece hoje! Dentre a lista de protocolos estão: HTML, CSS, XML, URL, DOM, SVG, PNG etc.

Entretanto, a Web foi projetada para os “humanos”, ou seja, o objetivo é compartilhamento de conteúdo visual. Assim, um software que lê páginas Web tem dificuldade de obter dados a partir desse local.

Inúmeras ferramentas foram desenvolvidas para auxiliar os desenvolvedores na construção de softwares que “leem” páginas Web. Eles são conhecidos como *Crawler*, *Spider* ou *Bot*. Alguns até conseguem certo êxito na leitura das páginas, mas é preciso somente uma atualização no site que essas ferramentas acabam por se perder em sua leitura.

Não seria mais interessante que, em vez da Web expor uma página com um conteúdo visual, não apresentasse somente os dados? Dessa forma, um Bot obteria os dados de forma muito mais fácil.

É exatamente esta a proposta de um Web Service: aproveitar-se de todo arcabouço de tecnologias e protocolos já existentes na Web para a comunicação entre aplicações.

Existem inúmeras definições de Web Services na literatura, mas a grande maioria converge para a seguinte: *programa identificado por uma URL, cuja interface é pública e bem definida.*

1.2 Arquitetura

Em um Web Service, um agente conhecido como **Cliente** (ou consumidor) se beneficia do resultado de um processamento que não é executado por ele. O cliente, então, efetua uma chamada para um outro agente, conhecido como **Servidor**, responsável por processar os dados e devolver uma resposta ao cliente. A imagem a seguir ilustra essa dinâmica:

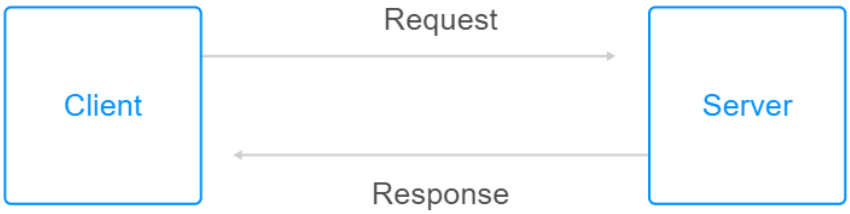

Figura 1.1 - Arquitetura simplificada de agentes de uma chamada Web Service
Fonte: Elaborado pelo autor (2021)

Os conceitos foram colocados no idioma inglês propositalmente, pois a grande maioria dos frameworks que veremos a seguir utiliza essa taxonomia.

Repare que esse é o princípio básico de uma requisição da Web para obtenção de uma página Web, imagem ou qualquer outro recurso. A principal diferença é que, para um Web Services, o que está sendo trafegado são dados normalmente dispostos em um formato como XML ou JSON.

1.3 História

Web Service faz parte de um conceito mais abrangente e antigo da Ciência da Computação, conhecido com Computação Distribuída. Seu principal objetivo é dividir a carga de processamento entre diversos dispositivos.

Já no início dos anos 90, surgiram alguns protocolos específicos de plataforma para comunicação entre aplicações, como o DCE (Distributed Computing Environment) para Unix e MSRPC para Windows.

Logo em seguida, surgiram outras tecnologias que foram mais utilizadas, como: CORBA (Common Object Request Broker Architecture), DCOM (Distributed Common Object Model) e Java RMI (Remote Method Invocation).

O grande problema dessas integrações era a interoperabilidade, assim, uma aplicação, que expunha um serviço desenvolvido em RMI, dificilmente iria ter um consumidor fora das tecnologias Java.

Outro problema eram os tipos de dados. Eles poderiam ser em formato de bytes, e o modo como interpretar esses bytes ficava a cargo de quem estava expondo o serviço. Isso dificultava os consumidores de outras plataformas.

A grande inovação que os Web Services trouxeram vem para solucionar justamente os problemas acima:

- Uma vez utilizando protocolos e linguagens padrões da Web, os serviços agora são verdadeiramente interoperáveis.
- O tráfego padrão dos dados é texto, assim, qualquer linguagem consegue manipular os dados, independentemente da plataforma ou tecnologia do servidor.

Para mais detalhes sobre a História do Web Service, recomendo a leitura do capítulo A Very Short History of Web Services, do livro Java Web Services: Up and Running: A Quick, Practical, and Thorough Introduction.

1.4 Benefícios da Web Service

Ao utilizar Web Service, podemos ter como benefícios:

- Infraestrutura leve e desacoplada de plataforma.
- Abertura de novos caminhos para modelos de desenvolvimento ágil e eficiente.
- Abstração de detalhes de implementação, de forma que quem chama não se preocupa em detalhes como: linguagem, sistema operacional, banco de dados, servidor de aplicação etc.
- Reaproveitamento das tecnologias de transporte disponíveis pela Internet (TCP-IP, HTTP, FTP, SMTP etc.).
- Utilização de padrões de dados muito utilizados na Web (XML e JSON).
- Restante das tecnologias Open Standard: SOAP, WSDL, UDDI, REST etc.

1.5 Tipos de Web Services

As duas principais tecnologias de Web Services são: SOAP e REST.

SOAP (*Simple Object Access Protocol*) é um protocolo estruturado com base em XML (*eXtensible Markup Language*). Foi criado em 1998 e mantido pela W3C. No próximo capítulo vamos detalhar mais esse protocolo.

Já o REST (*REpresentational State Transfer*) foi criado dentro de uma tese de doutorado e se opõe em alguns aspectos ao SOAP.

Segue abaixo um quadro-comparativo entre as duas tecnologias:

Tabela 1.1– Tabela comparativa entre os Web Services SOAP e REST

SOAP	REST
Protocolo de mensagens baseado em XML.	Usa XML ou JSON para enviar e receber dados (não restrito aos dois).
Usa WSDL para estabelecer o contrato entre consumidor e provedor.	Utiliza a própria especificação HTTP e endereços URL para comunicação.
Transfere dados nos protocolos HTTP, SMTP, FTP etc.	Transfere dados apenas no protocolo HTTP.
Mais difícil de implementar.	Fácil de implementar, sobretudo com JavaScript.
Menor performance no tráfego de dados.	Melhor performance no tráfego de dados.

Fonte: Elaborado pelo autor (2021)

1.6 Formato de Dados

Não basta que os dados trafegados sejam textos: eles precisam estar em um formato que facilite a leitura por um software. Para isso, organizamos os dados de forma estruturada, seguindo padrões bem conhecidos e aceitos por toda a comunidade.

Existem vários outros formatos de dados (exemplos: CSV, YAML e RDF), mas vamos nos concentrar nos mais utilizados por Web Services: XML e JSON.

1.6.1 XML

XML é o acrônimo de *eXtensible Markup Language*. Esse formato dispõe os dados utilizando marcadores, atributos e valores. Sua estrutura é em forma de árvore; em outras palavras, o valor de um marcador pode ser outro marcador.

Atualmente a especificação encontra-se na versão 1.1 e é mantido pela W3C.

Segue um exemplo de um arquivo XML válido, contendo dados de veículos:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Insured CustomerID="5">
   <CarPolicy PolicyType="Auto">
```

```
      <Vehicle Category="Sedan">
         <Make>Honda</Make>
         <Model>Accord</Model>
      </Vehicle>
      <Vehicle Category="Sport">
         <Make>Ford</Make>
         <Model>Mustang</Model>
      </Vehicle>
   </CarPolicy>
   <CarPolicy PolicyType="Antique">
      <Vehicle Category="Sport">
         <Make>Triumph</Make>
         <Model>Spitfire</Model>
      </Vehicle>
   </CarPolicy>
</Insured>
```

Comando de prompt 1.1 – Exemplo de um XML de dados de veículos
Fonte: Elaborado pelo autor (2021)

Fique atento aos próximos capítulos, nos quais vamos discutir mais sobre XML, além de conhecer estruturas de dados para validação e manipulação de arquivos XML.

1.6.2 JSON

JSON (acrônimo de *JavaScript Object Notation*) é outro formato de texto simples, similar ao XML. É conhecido por ser menos verboso e mais leve, diminuindo assim o tráfego de rede. A especificação é mantida pela ECMA.

Ele é formado por uma coleção de pares chave-valor e uma lista ordenada de valores. Segue um exemplo:

```
{"menu": {
    "header": "SVG Viewer",
    "items": [
        {"id": "Open"},
        {"id": "OpenNew", "label": "Open New"},
        null,
        {"id": "ZoomIn", "label": "Zoom In"},
        {"id": "ZoomOut", "label": "Zoom Out"},
        {"id": "Help"}
    ]
}}
```

Comando de prompt 1.2 – Exemplo de um JSON de dados de um menu
Adaptado de https://json.org/example.html

Diferente do XML, o JSON especifica chaves em vez de marcadores. O que define o conteúdo são os símbolos de abrir “{“ e fechar chaves “}” (para elementos

completos), os símbolos de colchetes “[“ “]” (para lista de valores) ou o valor diretamente (para elementos simples).

Repare que na linha 6 a palavra reservada **null** é usada no lugar de um elemento. Isso é válido, já que o JSON herdou certas características do JavaScript. Para esse caso, queremos expressar um elemento vazio para a lista.

Seguem os mesmos dados acima, expressos em formato XML:

```
<menu>
   <header>Adobe SVG Viewer</header>
   <item action="Open" id="Open">Open</item>
   <item action="OpenNew" id="OpenNew">Open New</item>
   <separator />
   <item action="ZoomIn" id="ZoomIn">Zoom In</item>
   <item action="ZoomOut" id="ZoomOut">Zoom Out</item>
   <item action="Help" id="Help">Help</item>
</menu>
```

Comando de prompt 1.3 – Mesmos dados de menus expressos em formato XML
Adaptado de https://json.org/example.html

Pode-se observar que:

1. Não existe em XML uma notação que expresse uma lista ou vetor. Nesse caso, o que há é uma repetição de notações (no caso acima, do marcador *item*);
2. Foi criado o marcador *separator* para substituir a palavra reservada null (não existe o conceito de nulidade para XML).

1.7 REST

REST (acrônimo de REpresentational State Transfer) é um modelo alternativo ao SOAP.

Como dito anteriormente, ele foi concebido a partir de uma tese de doutorado, na qual são questionados certos aspectos do SOAP e como são representados os recursos e transferência de estados.

A grande maioria dos Web Services REST utilizam JSON como principal formato de dados, mas outros formatos podem ser usados (inclusive XML).

Diferentemente do SOAP, o REST se fundamenta no HTTP e HTTPS como únicos protocolos de comunicação. Dessa forma, ele se aproveita de alguns dos conceitos já definidos (como *URL*, M*ethods* e *Status Code*, como veremos adiante).

Importante citar: só porque um Web Service disponibiliza dados no formato JSON e utiliza o protocolo HTTP ou HTTPS, não significa que se trata de um Web Service REST.

Para que seu Web Service seja Restful, ou seja, compatível com a especificação REST, precisa seguir as seguintes regras:

- A especificação do recurso no qual se deseja mudar o estado representacional na URL (mais precisamente, no path);
- A utilização correta do método HTTP para cada operação que se deseja fazer (conforme tabela abaixo):

Tabela 1.2— Lista de ações e métodos HTTP correspondentes

Ação	Método HTTP
Busca ou listagem de dados	GET
Criação de um novo recurso	POST ou PUT
Atualização de dados de um recurso existente	PUT ou PATCH
Exclusão de um recurso existente	DELETE

Fonte: Elaborado pelo autor (2021)

- A correta utilização do Status Code de acordo com o resultado obtido.

Na especificação REST, **O QUE** se deseja fazer está especificado no *Method HTTP*. Já **PARA QUEM** deseja aplicar a mudança, está especificado no path da *URL*.

Veja os exemplos a seguir (todos eles consideram que o serviço está disponível no domínio `wsrest.exemplo.com`):

- **Exemplo 1:** Deseja buscar os dados de um cliente, identificado pelo código 1234.

Por se tratar de uma busca, o método correto é o GET. O recurso desejado é cliente. Logo, a chamada Web Service correta seria:

```
GET http://wsrest.exemplo.com/clientes/1234
```

Comando de prompt 1.4 – Chamada REST para buscar dados de cliente por código
Fonte: Elaborado pelo autor (2021)

- **Exemplo 2**: Deseja cadastrar um novo cidadão.

Por se tratar de uma criação, os métodos corretos são POST ou PUT. O recurso desejado é cidadão. Logo, a chamada Web Service correta seria:

```
POST http://wsrest.exemplo.com/cidadaos

{
   "nome": "Frodo Bolseiro",
   "email": "fbolseiro@condado.com.br"
}
```

Comando de prompt 1.5 – Chamada REST para criação de um novo cidadão
Fonte: Elaborado pelo autor (2021)

OBS: A função PUT também pode ser utilizada para a criação de um novo recurso. Para mais detalhes sobre a diferença entre POST e PUT, acesse: <https://restfulapi.net/rest-put-vs-post/>.

- **Exemplo 3:** Deseja atualizar o telefone de um aluno, cujo RA (registro de aluno) é 2323.

Por se tratar de uma atualização, os métodos corretos são PUT ou PATCH. O recurso desejado é aluno. Logo, a chamada Web Service correta seria:

```
PUT http://wsrest.exemplo.com/alunos/2323

{
   "telefone": "11 93585-6588"
}
```

Comando de prompt 1.6 – Chamada REST para atualização de um aluno.
Fonte: Elaborado pelo autor (2021).

Conceitualmente, o PUT substitui todos os dados dos recursos (armazenados anteriormente) que foram enviados na requisição. Para atualizar somente os dados que foram enviados na requisição, mantendo os dados já armazenados, utiliza-se o método PATCH.

Exemplo 4: Deseja-se excluir o produto identificado pelo código 1212.

Nesse caso, o método é o DELETE e o recurso é produto, logo:

```
DELETE http://wsrest.exemplo.com/produtos/1212
```

Exemplo 5: Deseja-se buscar as disciplinas pertencentes a um determinado curso (identificadas por uma sigla).

Trata-se de uma busca, portanto método GET. Entretanto, temos recursos aninhados, assim será necessário especificar o método da seguinte forma:

```
GET http://wsrest.exemplo.com/cursos/SCJ/disciplinas
```

Reforçando: O resultado da consulta acima será uma lista de disciplinas.

Dica: Para buscas mais complexas, que envolvam mais de um recurso aninhado ou múltiplos parâmetros de pesquisa, utilizar *query param* em vez de *path param*. Segue um exemplo:

```
GET http://wsrest.exemplo.com/alunos?ativo=true&curso=SCJ
```

O resultado de uma requisição REST também reaproveita um conceito do HTTP: o Status Code. Trata-se de um código, de uma lista padrão de sucessos e erros dividida em cinco grupos:

- O grupo 1XX indica informativos. Não é utilizado por Web Services REST.
- O grupo 2XX indica que a requisição foi aceita com sucesso. Os mais utilizados são: 200 (OK), 201 (*Created*), 202 (*Accepted*) e 204 (*No Content*).

- O grupo 3XX indica que o cliente precisa tomar alguma ação para obtenção do recurso. Normalmente utilizado para redirecionamento ou cache.
- O grupo 4XX indica erros ocasionados pelo cliente. Dessa forma, é necessário que o cliente tome alguma ação ou corrija algo antes de tentar novamente. Os mais utilizados são: 400 (*Bad Request*), 401 (*Unauthorized*), 403 (*Forbidden*), 404 (*Not Found*), 409 (*Conflict*) e 422 (*Unprocessable Entity*).
- O grupo 5XX indica erros que ocorreram do lado do servidor, impossibilitando o processamento. Nesse caso, o cliente pode efetuar uma nova chamada com os mesmos dados assim que o problema do servidor estiver solucionado. Os mais utilizados: 500 (*Internal Server Error*), 503 (*Service Unavailable*) e 504 (*Gateway Timeout).*

Por vezes, somente o *Status Code* não é suficiente para auxiliar o cliente na resolução de um problema. Assim, podemos utilizar o corpo da mensagem de response para detalhar o erro.

Para exemplificar: podemos utilizar o corpo da mensagem do response para informar quais campos obrigatórios o cliente deixou de enviar em uma requisição de cadastro, ou se não aplicou a máscara correta em um determinado campo de Data e Hora.

Durante o ciclo de vida de um Web Service REST, podem ocorrer mudanças na estrutura dos dados ou nos métodos que forcem o cliente a uma readequação do seu código-fonte. Isso é chamado de **Mudança no Contrato de um Web Service**.

Quando isso ocorre, é interessante manter a versão antiga e a versão nova funcionando simultaneamente durante um tempo, assim os clientes não são pegos de surpresa e terão um período para se readequarem.

Esse conceito é conhecido como **Versionamento de API's**. Seguem alguns exemplos de estratégias para versionamento**:**

- **Por URL:** um número é adicionado na URL, no primeiro path, indicando a versão. Exemplo: GET http://minhaapi.com.br/v1/alunos
- **Por Parâmetro header customizado:** utiliza-se um header customizado para informar a versão. Exemplo: Accept-version: v1.

- **Por Parâmetro header Accept:** neste caso, utiliza-se um header padrão, mas informando no valor o número da versão. Exemplo: Accept: application/vnd.example.v1+json.
- **Por data:** o desenvolvedor do aplicativo consumidor informa, no momento de codificação, a data atual. Dessa forma, o servidor consegue determinar qual a versão da API disponível na data informada e responder à requisição com base nessa versão. Exemplo: API-Version: 2020-10-06 ou http://minhaapi.com.br/2020-10-06/.

Dica: no livro **Web API Design: The Missing Link**, o autor questiona o versionamento de API e propõe uma estratégia de atualização de dados e métodos, de modo a sempre manter compatibilidade com as versões anteriores.

1.8 Finalizando

Neste capítulo, aprendemos o que são Web Services, suas aplicações, história e vantagens. Aprendemos também sobre os dois principais tipos de Web Services: SOAP e REST e nos aprofundamos nos conceitos desse último.

Aprendemos como criar Web Services aplicando as regras de REST: recursos, métodos HTTP e Status Code. E, por último, vimos algumas estratégias para versionamento de API’s.

Agora vamos aprofundar mais o entendimento e utilização de schemas (XSD) e estilos (XSLT).

REFERÊNCIAS

Apigee Google Cloud. **Web API Design: The Missing Link. Best Practices for Crafting Interfaces that Developers Love**. 2018. Disponível em: <https://cloud.google.com/files/apigee/apigee-web-api-design-the-missing-link-ebook.pdf>. Acesso em: 01 fev. 2021.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm>. Acesso em: 01 fev. 2021.

KALIN, Martin. **Java Web Services: Up and Running: A Quick, Practical, and Thorough Introduction**. O'Reilly Media. Second edition. 2013. ISBN: 978-1449365110.

Smartbear. **OpenAPI Specification**. 2020. Disponível em: <https://swagger.io/specification/>. Acesso em: 01 fev. 2021.

GLOSSÁRIO

REST	Acrônimo de REpresentational State Transfer.
JSON	Acrônimo de JavaScript Object Notation.
W3C	O World Wide Web Consortium (W3C) é a principal organização de padronização da World Wide Web.