



API

INTERFACE DE PROGRAMAÇÃO DE APLICAÇÕES (API) E WEB SERVICES

Arthur Gonçalves Ferreira

INTERFACE DE PROGRAMAÇÃO DE APLICAÇÕES (API) E WEB SERVICES

1ª edição

São Paulo
Platos Soluções Educacionais S.A
2021

© 2021 por Platos Soluções Educacionais S.A.

Todos os direitos reservados. Nenhuma parte desta publicação poderá ser reproduzida ou transmitida de qualquer modo ou por qualquer outro meio, eletrônico ou mecânico, incluindo fotocópia, gravação ou qualquer outro tipo de sistema de armazenamento e transmissão de informação, sem prévia autorização, por escrito, da Platos Soluções Educacionais S.A.

Diretor Presidente Platos Soluções Educacionais S.A

Paulo de Tarso Pires de Moraes

Conselho Acadêmico

Carlos Roberto Pagani Junior
Camila Braga de Oliveira Higa
Camila Turchetti Bacan Gabiatti
Giani Vendramel de Oliveira
Gislaine Denisale Ferreira
Henrique Salustiano Silva
Mariana Gerardi Mello
Nirse Ruscheinsky Breternitz
Priscila Pereira Silva
Tayra Carolina Nascimento Aleixo

Coordenador

Henrique Salustiano Silva

Revisor

Rennan Martini Rodrigues

Editorial

Alessandra Cristina Fahl
Beatriz Meloni Montefusco
Carolina Yaly
Mariana de Campos Barroso
Paola Andressa Machado Leal

Dados Internacionais de Catalogação na Publicação (CIP)

F383i Ferreira, Arthur Gonçalves
Interface de programação de aplicações (API) e web services /
Arthur Gonçalves Ferreira. – São Paulo:
Platos Soluções Educacionais S.A., 2021.
44 p.
ISBN 978-65-5356-033-8

1. API's e web services. 2. Arquiteturas de API's.
3. Soluções do front end. I. Título.

CDD 004

Evelyn Moraes – CRB: 8 010289

2021
Platos Soluções Educacionais S.A
Alameda Santos, nº 960 – Cerqueira César
CEP: 01418-002 — São Paulo — SP
Homepage: <https://www.platosedu.com.br/>

INTERFACE DE PROGRAMAÇÃO DE APLICAÇÕES (API) E WEB SERVICES

SUMÁRIO

Caracterizando as APIs dos Web Services _____	05
Análise de projetos de arquiteturas de APIs e Web Services e seleção de tecnologias para implementação _____	19
Consumo das APIs e Web Services por soluções do <i>front-end</i> ____	30
Decisões tecnológicas para o desenvolvimento e testes de APIs e Web Services _____	43

Caracterizando as APIs dos Web Services

Autoria: Arthur Gonçalves Ferreira

Leitura crítica: Rennan Martini Rodrigues



Objetivos

- Ensinar os conceitos e as características gerais de Web Services.
- Conhecer os fundamentos de APIs.
- Estabelecer a diferença entre Web Services e APIs.



1. Introdução a Web Services

Olá, aluno! Iniciaremos o nosso estudo desta unidade tratando sobre um assunto muito importante para a sua carreira profissional, os Web Services, que também podem ser chamados de Serviços da Web. Sendo assim, neste tópico, você compreenderá quais são as principais características, funcionalidades, aplicabilidades e conceitos relacionados a eles.

Você terá a oportunidade de aprender sobre o surgimento dos Web Services, sua evolução, o que são, as suas funções e os seus serviços, além de saber o que são arquiteturas de serviços. Dessa forma, todo o conteúdo abordado ajudará você a compreender a importância dos Web Services.

1.1 Características e conceitos gerais dos Web Services

Os Web Services são uma tecnologia que surgiu na década de 1990 e reúne diversas soluções na tentativa de incorporar sistemas e, dessa forma, realizar a comunicação deles com aplicações através de tecnologias e protocolos. Em outras palavras, nada mais são do que serviços da web, sendo uma interface construída para realizar comunicação na rede.

Web Services são conjuntos de programas que podem ser publicados, buscados e chamados por meio da internet. Esses programas podem realizar um simples processo de troca de mensagens, como também complexas transações comerciais ou industriais, por exemplo, um processo de compra de produtos. Uma vez que um Web Service é publicado em um servidor web, vários programas e até mesmo outros Web Services podem acessá-lo e chamá-lo, tanto para obtenção de dados como para a interação com serviços que uma organização oferece (TAMAE, 2004).

Desta forma, pode-se definir um Web Service como um programa que oferece diversos tipos de funcionalidades, as quais, necessariamente, são úteis a outros programas. É importante que você saiba que um Web Service fica armazenado em um servidor web e pode ser acessado a qualquer momento, geralmente, pela internet.

1.2 Criando um Web Service

De que forma um Web Service é criado? Toda vez que alguém publica o *back-end* de uma aplicação em um servidor web para que ele possa ser acessado por outros sistemas através de protocolos, ele se torna um Web Service. Segundo Robbins:

Back-end refere-se aos programas e scripts que funcionam no servidor nos bastidores para tornar as páginas da web dinâmicas e interativas. Em geral, o desenvolvimento da web de back-end são muito utilizados por programadores experientes, mas é bom para todos os web design familiarizados com a funcionalidade de back-end. (ROBBINS, 2012, p. 9)

Dessa forma, as informações de uma determinada aplicação são disponibilizadas em um servidor web, e o aplicativo se torna um Web Service. Essas informações armazenadas no servidor web são:

- Processamento de formulários.
- Estrutura de programação de um banco de dados.
- Sistemas de gerenciamento de conteúdo.
- Outros aplicativos da web do lado do servidor usando PHP, JSP, Ruby, ASP.NET, Java e outras linguagens de programação.

Para facilitar o acesso a esses serviços, são utilizadas arquiteturas e padrões, que definem o mecanismo de interação com eles e o formato da interação recebida.

1.3 Modelos de arquiteturas e padrões de Web Services

Em Web Services, existem os modelos de arquiteturas e padrões, que possuem, basicamente, o objetivo de detalhar e proteger as informações importantes de uma arquitetura. Existem quatro modelos: modelo orientado a mensagem, modelo orientado a serviço, modelo orientado a recursos e modelo de policiamento. Entenderemos o funcionamento de cada um deles.

- **Modelo orientado a mensagem:** também conhecido como *message oriented model*, tem como objetivo priorizar detalhes importantes da arquitetura, os quais, necessariamente, estão relacionados com o processamento de mensagens, por exemplo, sua estrutura, o modo de transporte, entre outros aspectos relevantes (MEDYK, 2006).
- **Modelo orientado a serviço:** também conhecido como *service oriented model*, este modelo prioriza detalhes importantes relacionados ao serviço e à ação da arquitetura.
- **Modelo orientado a recursos:** também conhecido como *resource oriented model*, este modelo prioriza detalhes importantes relacionados aos recursos da arquitetura.
- **Modelo de policiamento:** também conhecido como *policy model*, este modelo prioriza detalhes importantes relacionados ao comportamento dos agentes da arquitetura, sendo aplicado sobre agentes que necessitam acessar diretamente os recursos da arquitetura. O modelo de policiamento se relaciona com outros aspectos, tais como políticas de segurança, gerenciamento e aplicações.

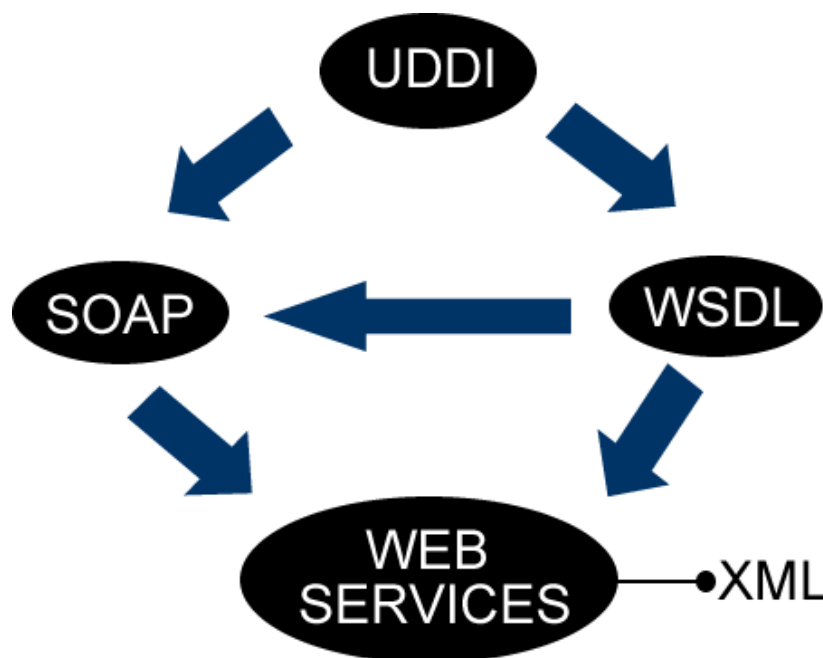
1.4 Tecnologias utilizadas em Web Services

Para o bom funcionamento de um Web Service, são utilizadas algumas ferramentas, as quais auxiliam no processo de segurança, comunicação e administração. São exemplos delas: **HTTP**, **XML**, **WSDL** e **UDDI**.

- **HTTP:** protocolo responsável por transportar dados.
- **XML:** linguagem de marcação que facilita o compartilhamento de informações em Web Services.
- **WSDL:** descreve as interfaces de um Web Service e como as mensagens são formatadas quando são usados os protocolos. É baseada em XML.
- **UDDI:** permite que os comerciantes possam encontrar Web Services de forma rápida, fácil e dinâmica, além de interagir uns com os outros.
- **SOAP:** é uma arquitetura de Web Services.
- **REST:** é uma arquitetura de Web Services.

A Figura 1 mostra um exemplo da utilização das tecnologias WSDL e UDDI e da arquitetura SOAP em um determinado Web Service.

Figura 1 - Tecnologias e arquitetura de Web Services



Fonte: elaborada pelo autor.

1.5 Entidades de arquitetura Web Service

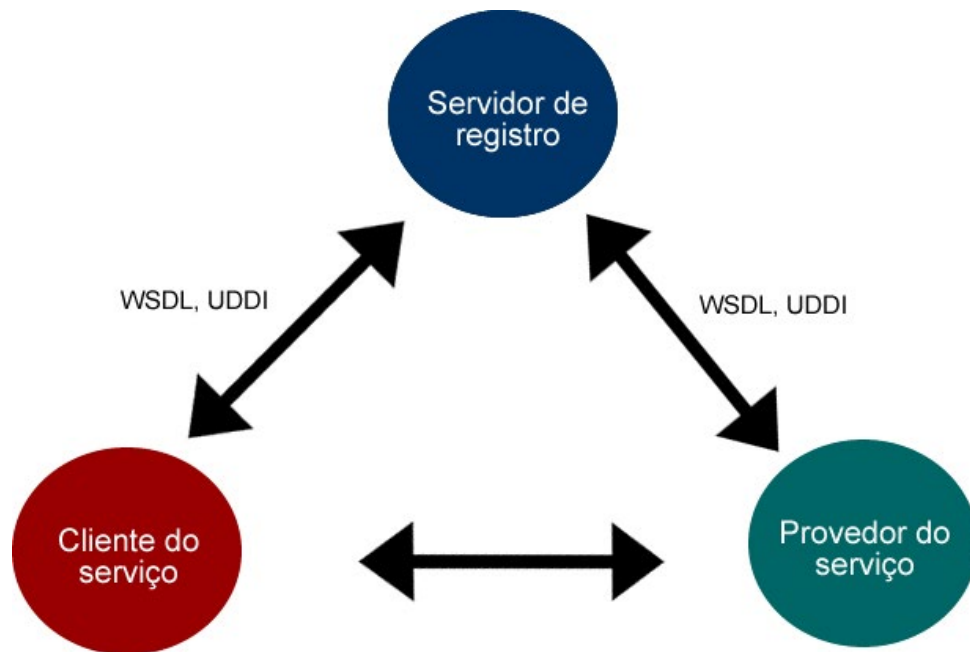
As arquiteturas de Web Services possuem entidades que interagem entre si para realizar uma publicação, uma busca e execuções de operações de Web Services. Basicamente, existem três entidades interagindo entre si: o provedor do serviço (*service provider*), o cliente do serviço (*service requestor*) e o servidor de registro (*service registry*).

Dessa forma, temos:

- **Provedor do serviço (*service provider*):** é uma plataforma que realiza a hospedagem do Web Service. Através dela, é possível o acesso ao serviço do Web Service.
- **Cliente do serviço (*service requestor*):** é a aplicação que está procurando, invocando ou iniciando uma interação com o Web Service. O cliente do serviço pode ser uma pessoa acessando através de um *browser* ou uma aplicação, realizando uma invocação aos métodos descritos na interface do Web Service.
- **Servidor de registro (*service registry*):** é o servidor que armazena, registra e serve como busca de Web Services. Possui arquivos de descrição de serviços que foram publicados pelo provedor de serviço.

A Figura 2 mostra um exemplo de arquitetura de um Web Service com interações das entidades: provedor do serviço (*service provider*), cliente de serviço (*service requestor*) e servidor de registro (*service registry*).

Figura 2 - Interação de entidades Web Services



Fonte: elaborada pelo autor.

Assim, de forma sintetizada, temos que o provedor do serviço publica o Web Service em um servidor, e os aplicativos clientes consomem o Web Service publicado. Para acessar o serviço, o aplicativo cliente usa a localização do Web Service, a qual é o seu URL, dessa maneira, o aplicativo cliente acessa o Web Service e envia parâmetros. Em troca, recebe as informações retornadas, geralmente, como uma estrutura em formato XML ou JASON.

1.6 Arquiteturas e padrões Web Services

Os Web Services podem ser desenvolvidos seguindo arquiteturas e padrões diferentes, sendo **SOAP** e **REST** os mais comuns. Cada padrão define como as informações dos serviços disponíveis da web devem ser publicadas. Conheceremos os principais padrões existentes.

- **SOAP:**

De acordo com Roy (2001), o ***service-oriented architecture (or application) protocol*** (protocolo de arquitetura orientada a serviços),

mais conhecido como SOAP, é um protocolo leve para a troca de dados XML. Este protocolo é construído para chamar programas e aplicações de maneira remota, através de **RPC**, técnica utilizada para desenvolvimento de aplicações distribuídas que segue o modelo cliente-servidor em um ambiente independente de plataforma e linguagem de programação.

São características do SOAP: é um protocolo embasado em XML para realizar troca de mensagens em sistemas distribuídos. Atualmente, é o padrão de arquitetura em Web Services, utilizando, normalmente, o protocolo HTTP como protocolo de transporte. A mensagem enviada pelo SOAP possui os elementos **envelope**, **header** (cabeçalho), **body** (corpo), **payload** (documento) e **fault** (falha).

O envelope SOAP armazena todos os outros elementos e é obrigatório em uma mensagem SOAP. Ele precisa de informações fundamentais do protocolo HTTP para enviar uma mensagem, sendo assim, é comum que, no cabeçalho da mensagem, seja utilizado o SOAP *action*, que indica o endereço da mensagem.

O cabeçalho não é um elemento obrigatório, mas, se utilizado, deve ser o primeiro elemento do envelope. Ele contém informações importantes, como o SOAP *action* citado anteriormente, além de dados informando se a mensagem deve ser processada por um intermediário.

Já o elemento corpo é obrigatório estar no envelope SOAP e deve vir logo após o cabeçalho, caso este esteja presente. O corpo do envelope armazena informações, como métodos e parâmetros, que serão reproduzidos e processados.

O documento pode possuir um conteúdo de requisição, no qual armazenará informações sobre a chamada de um método, ou um conteúdo de resposta, no qual armazenará informações do resultado da chamada do método.

Por último, temos o *fault*, que, basicamente, faz referência ao tratamento de falhas e exceções. Toda falha deve retornar ao consumidor, então cabe ao protocolo HTTP notificá-la, caso ocorra no seu envio e o erro aconteça na execução, ou seja, o próprio SOAP notifica a falha.

- **REST:**

Criado por **Roy Fielding**, o ***Representational State Transfer***, mais conhecido como REST, é um conjunto coordenado de restrições de arquitetura que tenta minimizar a dependência e a escalabilidade das implementações dos componentes (FIELDING, 2000). O REST foi criado com o objetivo de manter o protocolo HTTP para realizar transações na internet sem o auxílio de outro protocolo, recebendo ajuda apenas de seus métodos internos.

A arquitetura REST realiza as transações de inserção, atualização, exclusão e recuperação de informação na internet. Os sistemas baseados na arquitetura REST são conhecidos como RESTful e sempre seguem os princípios descritos por Roy Fielding. É uma arquitetura mais simples comparada à arquitetura SOAP, por isso, analisaremos as diferenças entre a arquitetura SOAP e REST no Quadro 1.

Quadro 1 - REST versus SOAP

SOAP	REST
Arquitetura de comunicação	Arquitetura de software
Complexa	Simples
Possui muitas especificações	Não é burocrática
Utiliza o protocolo HTTP apenas como meio de transporte	Utiliza o protocolo HTTP como único protocolo a realizar transações na internet
Possui muitos métodos customizados	Possui métodos mais uniformes
Possui suporte a XML	Possui suporte a XML, JSON, YAML e outros
Lê cache	Utiliza o método RPC

Fonte: elaborado pelo autor.



2. Introdução aos APIs

APIs são ferramentas muito utilizadas atualmente, geralmente em aplicações web, mas vão muito além disso. É possível encontrar a atuação de APIs em tudo que possamos imaginar, desde aplicações web a aplicações *desktops*, de transações on-line à troca de mensagens. Dessa forma, as APIs são muito mais comuns do que você possa imaginar.

A verdade é que não é de hoje que a tecnologia API é utilizada. Há muito tempo, fala-se em API, se constrói API e se utiliza API. Esta ideia nasceu, praticamente, junto aos computadores comerciais, conhecidos como *desktops* ou pessoais, quando se verificou a possibilidade de um computador que tivesse a capacidade de executar vários programas que não necessariamente tivessem sido desenvolvidos por uma única empresa.

As APIs desenvolvem diversas funcionalidades e são muito importantes para a comunicação entre aplicações. Estudaremos e conheceremos suas características, aplicabilidades e funcionalidades, verificando a diferença, dentro do contexto, entre APIs e Web Services.

2.1 Características e conceitos gerais de APIs

A *Application Programming Interface* (Interface de Programação de Aplicativos), mais conhecida como API, é classificada como um conjunto de rotinas e padrões de programação que possuem o objetivo de acessar aplicativos de software ou plataformas baseados na web. De forma direta, APIs são várias ferramentas, podendo ser métodos de desenvolvimento e protocolos, as quais, juntas, facilitam as comunicações de programas e aplicações.

A API pode ser conceituada também como um conjunto de normas e ferramentas que permitem a comunicação entre aplicações, programas e plataformas, sem o conhecimento ou a intervenção do usuário. Dessa forma, uma API pode disponibilizar diversas funções de um determinado site, para que possam ser usadas em outras aplicações.

Segundo o site RedHat:

Uma API permite que sua solução ou serviço se comunique com outros produtos e serviços sem precisar saber como eles foram implementados. Isso simplifica o desenvolvimento de aplicações, gerando economia de tempo e dinheiro. Ao desenvolver novas ferramentas e soluções (ou ao gerenciar aquelas já existentes), as APIs oferecem a flexibilidade necessária para simplificar o design, a administração e o uso, além de fornecer oportunidades de inovação. (INTERFACE..., 2021, [s. p.])

Desta forma podemos apontar como características das APIs:

- Pode ser de acesso público, restrito ou privado.
- É baseada em arquitetura cliente-servidor.
- Baseada em protocolo HTTP.
- Baseada na disponibilização de serviços.
- Conecta aplicações, programas e plataformas.
- São padrões disponibilizados para desenvolvedores.

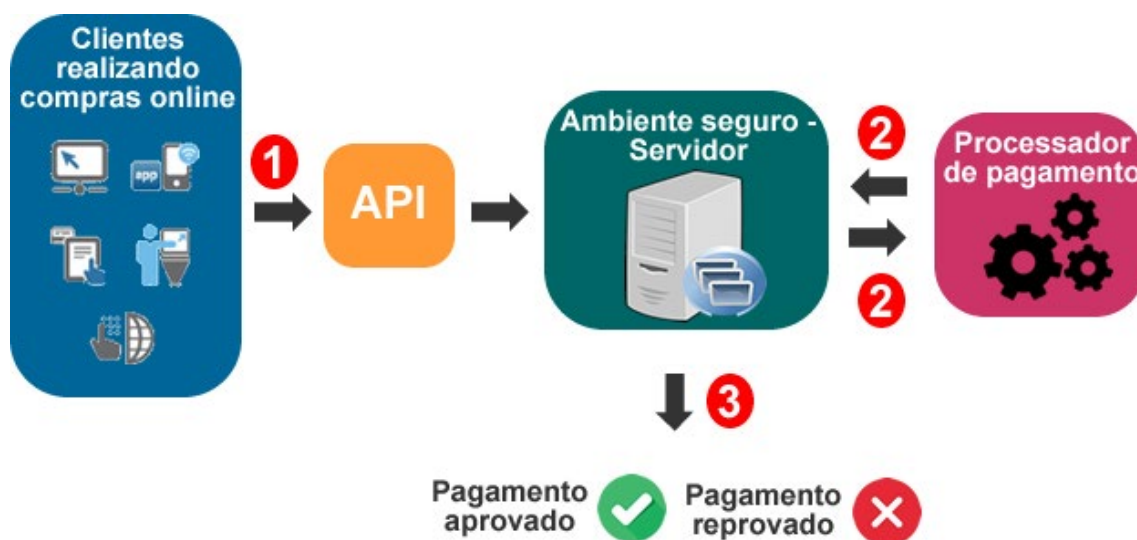
2.2 Exemplos e aplicabilidades de APIs

Para que você possa entender melhor, é possível que já saiba o que é uma API e já a tenha utilizado, visto que ela é muito mais comum do que você imagina, pois faz parte do nosso cotidiano. Imagine que você precisa realizar um pagamento on-line. Nesse contexto, qual é a

sua primeira ideia? Provavelmente, você lembra de algumas formas de pagamento, como PagSeguro, Cielo ou PayPal, certo? Caso tenha pensado nisso, a explicação está justamente no conceito de uma API.

A API é criada para oferecer produtos e funcionalidades a um determinado serviço que já existe. Por que você lembra de PagSeguro, Cielo ou PayPal quando falamos de pagamentos on-line? Porque são aplicações e programas que se comunicam através de APIs integradas a serviços já existentes, os quais você utiliza para realizar pagamentos on-line. Neste caso, você navega em um determinado site de vendas e, para finalizar sua compra, é redirecionado para um desses produtos associados, que proporciona ao site de vendas que o cliente realize seu pagamento por eles. A Figura 3 mostra a ideia de uma API.

Figura 3 - *Application Programming Interface*



Fonte: elaborada pelo autor.

É importante ressaltar que a API é aplicada em vários outros contextos, ou seja, vai muito além da comunicação entre aplicações que realizam transações on-line. Podemos verificar a utilização dela no Google Maps, no qual qualquer empresa pode utilizar os dados dele, adaptá-lo e disponibilizar em seu site, ou seja, não foi preciso a empresa

desenvolver uma aplicação do zero, pois utilizou um padrão já existente e reaproveitou em seu site.

Podemos citar vários outros exemplos da atuação de API: um aplicativo de fotos desenvolvido para celular, no qual o usuário teria acesso à câmera através de uma API já existente, sem necessidade de criar uma interface de câmera; o aplicativo de mensagens instantâneas WhatsApp, que consegue integrar os contatos do telefone através da API; a rede social Instagram, que utiliza uma API para postar a mesma publicação no Facebook.

Para que uma API possa ser criada, é necessário possuir um sistema de fácil integração, segura, estável e com bom desempenho.

2.3 Diferença entre APIs e Web Services

É correto afirmar que todo Web Service é uma API, mas nem toda API é um Web Service, porque tanto o Web Service quanto as APIs realizam a comunicação entre aplicações, porém a forma como são utilizados é totalmente diferente. O Quadro 2 mostra a diferença entre essas duas ferramentas.

Quadro 2 - APIs versus Web Services

APIs	Web Services
Pode realizar qualquer tipo de comunicação	Pode utilizar serviços de comunicação SOAP, REST e XML
Não precisa de uma rede para funcionar	Sempre precisa de uma rede para funcionar
É uma interface direta com um aplicativo	É uma aplicação
Utilizada de forma mais eficaz em comunicações de aplicações para aplicações	Utilizada de forma mais eficaz em comunicações de máquina para máquina

Fonte: elaborado pelo autor.

Aluno, esta unidade tratou sobre o estudo de APIs e Web Services. Você aprendeu todos os conceitos, aplicabilidades e características desses dois temas importantíssimos. Vimos que um Web Service é uma API, mas nem toda API é um Web Service, assim como estudamos a estrutura de um Web Service, verificamos a sua aplicabilidade e diferenciamos da estrutura e aplicabilidade de um API. Com certeza, esse estudo dará a você a oportunidade de agregar conhecimento e crescer profissionalmente. Não deixe de estudar os outros materiais desta unidade. Bons estudos!



Referências

FIELDING, R. T. *et al.* **REST APIs must be hypertext-driven**. Disponível em: <https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>. Acesso em: 15 ago. 2021.

INTERFACE de programação de aplicações: o que é API? **RedHat**, 2021. Disponível em: <https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces>. Acesso em: 15 ago. 2021.

MEDYK, S. **Web Services em Gerência de Redes**. Material de Apoio. Paraná, 2006. Disponível em: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. Acesso em: 15 ago. 2021.

O QUE É API? **CanalTech**, [s. d.]. Disponível em: <https://canaltech.com.br/software/o-que-e-api/>. Acesso em: 15 ago. 2021.

ROBBINS, J. N. **Learning web design: a beginner's guide to HTML, CSS, JavaScript, and web graphics**. 4. ed. Cambridge: O'Reilly Media, 2012.

ROY, J.; RAMANUJAN, A. **Understanding web services**. **IEEE Internet Computing**, v. 3, n. 6, p. 69 –73, 2001.

TAMAE, R. Y. **SISPRODIMEX** – Sistema de Processamento Distribuído de Imagens. 2004. Dissertação (Mestrado em Ciência da Computação) – Fundação de Ensino Eurípides Soares da Rocha, Marília, 2004.

TAMAE, R. Y.; LIMA, P. R. Web Services: uma nova visão da arquitetura de aplicações. **Revista Científica Eletrônica de Sistemas de Informação**, ano I, n. 2, p. 1-3, fev. 2005.

Análise de projetos de arquiteturas de APIs e Web Services e seleção de tecnologias para implementação

Autoria: Arthur Gonçalves Ferreira

Leitura crítica: Rennan Martini Rodrigues



Objetivos

- Analisar, planejar e projetar arquiteturas de APIs.
- Analisar, planejar e projetar arquiteturas de Web Services.
- Apresentar tecnologias para construção de APIs e Web Services.



1. Analisando projetos de arquiteturas de API Web

O desenvolvimento de qualquer tipo de Web Service deve ser muito bem estruturado, para que ele possa funcionar da maneira correta. Temos que lembrar que nem sempre todo Web Service é uma API, neste caso, uma API web é um Web Service.

A construção de um projeto de arquitetura de uma API web não se limita à preocupação da utilização de URLs no protocolo de transporte HTTP, na transmissão de dados ou em cabeçalhos. Realizar a implementação de uma API web é ter a preocupação com o seu funcionamento, pois são muitos processos e elementos que precisam ser levados em consideração.

Por isso, neste tópico, analisaremos de que forma projetos de arquiteturas de APIs web são projetados. Tomaremos como exemplo a ideia de um desenvolvimento de uma API web eficaz, para que você tenha a ideia da importância da construção dela.

1.1 Identificando o projeto de arquitetura de API web

Se você deseja construir uma API web, a primeira coisa que deve fazer é identificar qual seria a sua finalidade. Este processo é muito importante para que seu projeto possa ser implementado da maneira correta. Visualizando a construção de um projeto de arquitetura de uma API Web, podemos verificar, por exemplo, que, além do protocolo de transmissão de HTTP, é possível utilizar outros protocolos de comunicação, como o WebSockets, o XMPP e o MQTT. Para utilizar a API web, torna-se necessário lembrar do conceito de Web Service:

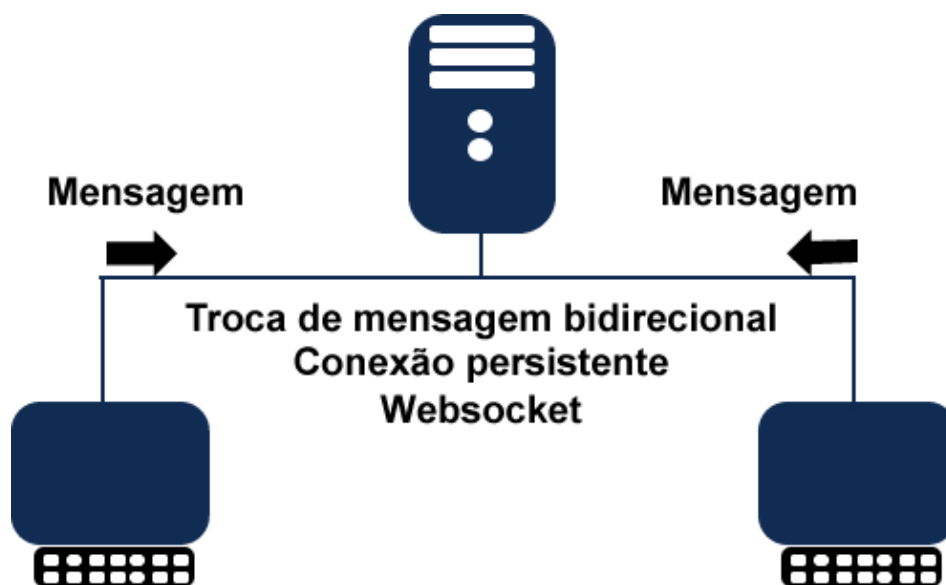
Um Web service é um componente de negócio que fornece uma funcionalidade reutilizável para clientes, ou consumidores, e pode ser pensado como um componente com acessibilidade verdadeiramente global – se houver direitos de acesso apropriados. (SHARP, 2011, p. 716)

Caso você não defina bem o seu projeto de estrutura de API web, pode ser que tenha problemas no andamento dele, porque cada protocolo é utilizado para um tipo de implementação. O protocolo HTTP é um dos principais protocolos de comunicação no meio web, porém, em algumas aplicações, ele não desempenha uma função eficaz, porque não realiza uma comunicação simultânea, o que acaba sobrecarregando um servidor web.

1.2 Projetos de arquitetura de API web com WebSockets

O protocolo WebSockets atua tanto em navegadores quanto em servidores web, realizando uma comunicação bidirecional, ou seja, existe uma comunicação, um transporte de mensagem bilateral, realizado em tempo real. Este tipo de protocolo é muito utilizado em aplicações de jogos on-line, nos quais existem vários jogadores interagindo em tempo real, websites ou aplicativos de chat, links de conteúdos esportivos, reuniões e outros eventos que ocorrem ao vivo, assim como em aplicações que realizam atualizações de informações em tempo real, como redes sociais. A Figura 1 mostra o funcionamento do protocolo WebSocket.

Figura 1 - WebSocket



Fonte: elaborada pelo autor.

1.3 Projetos de arquitetura de API web com XMPP

O protocolo XMPP é baseado em XML, sendo muito utilizado em aplicativos que realizam a comunicação de troca de mensagens instantâneas. Este protocolo tem como recursos a detecção de presença, realizando o controle de status de um usuário, verificando se ele está on-line ou off-line; permite que o próprio usuário defina seu status; permite chamada de vídeos e voz. A maior parte dos aplicativos de mensagens instantâneas que existem atualmente utiliza o protocolo XMPP, por exemplo, WebEx, GoToMeeting, Facebook Messenger, WhatsApp e Telegram. A Figura 2 mostra uma das arquiteturas do protocolo XMPP.

Figura 2 - Ponto a ponto XMPP

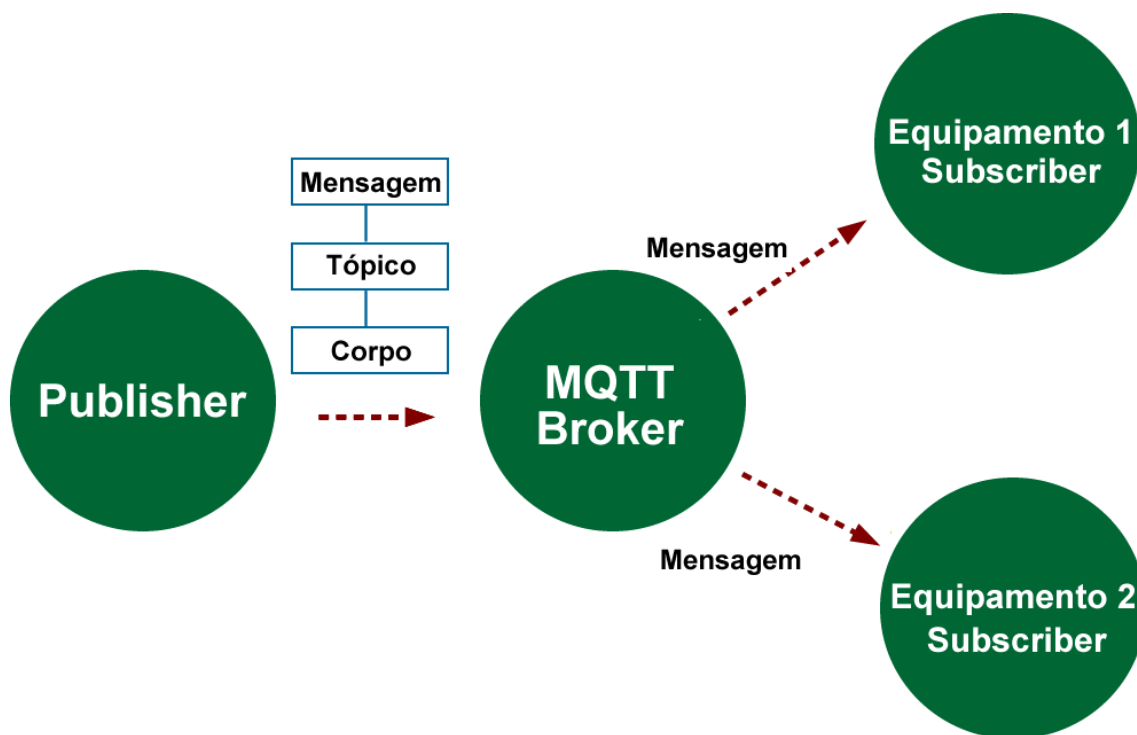


Fonte: elaborada pelo autor.

1.4 Projetos de arquitetura de API web com MQTT

MQTT é um protocolo de troca de mensagens entre máquinas muito utilizado na construção de APIs web que realizam a troca de pequenas mensagens. Logo, suas características principais são: protocolo muito leve e de baixo consumo de hardware. Ele utiliza a arquitetura *publish-subscribe* (publicação e assinatura). Este tipo de protocolo é muito utilizado em aplicações voltadas para IoT (internet das coisas), Arduino (plataforma que armazena componentes elétricos e programas) e na comunicação via bluetooth. A Figura 3 mostra a arquitetura do protocolo MQTT.

Figura 3 - Arquitetura protocolo MQTT



Fonte: elaborada pelo autor.

Dessa forma, fica clara a necessidade de identificar o tipo de projeto que você construirá, para poder utilizar uma arquitetura de API web que funcione para a finalidade que deseja. Se você deseja utilizar uma API web, pode ser que precise utilizar um desses protocolos citados, por isso, verifique as possibilidades da utilização deles em diferentes tipos de projetos de arquiteturas de APIs.



2. Tecnologias para a implementação de uma API Web

Você deve entender que arquiteturas e protocolos são considerados tecnologias, por tanto, temos REST, SOAP, HTTP, WebSockets, XMPP, MQTT, WSDL e UDDI como tecnologias utilizadas para a implementação de um Web Service. Essas tecnologias são muito importantes para que o processo de implementação e funcionamento de um Web Service seja eficiente.

Existem outras tecnologias que são utilizadas na implementação de um Web Service, as quais estão ligadas a equipamentos e programas. Os Web Services são disponibilizados em servidores, computadores específicos que contêm banco de dados e proteção específica para que o Web Service possa ser disponibilizado de forma segura.

2.1 API web e seu funcionamento

Um Web Service passa a existir quando o *back-end* de uma aplicação é disponibilizado no servidor web, no qual o *back-end* seria a estrutura dessa aplicação e está ligado à linguagem de programa em que ela foi escrita e estruturada. Dessa forma, existem programas que auxiliam nessa codificação e que são considerados tecnologias utilizadas para a implementação de um Web Service.

As linguagens de programas utilizadas na implementação de um Web Service variam muito, sendo Java, JavaScript, Python, C, C++, C#, PHP e NodeJS as mais utilizadas.

Após ter realizado a análise do tipo de projeto que trabalhará e verificar qual protocolo se adaptará ao seu projeto, você deve escolher o formato de dados que será utilizado na troca de mensagens entre cliente e servidor. Nós sabemos que a característica do Web Service é a

transferência de dados, neste caso, essa transferência deve ocorrer no formato de documentos padronizados.

Um Web Service utiliza o formato XML na arquitetura **SOAP**, a qual é protocolo simples de acesso a objetos. Já o formato JSON é utilizado na arquitetura **REST**, que possui uma arquitetura diferente da SOAP. É importante escolher a arquitetura do projeto do Web Service que você construirá, para que possa escolher o formato de dados adequado para o protocolo utilizado pelo Web Service.

3. Analisando projetos de arquiteturas com API RESTFUL

Atualmente, as APIs RESTful são mundialmente utilizadas para a implementação de APIs. Por isso, torna-se extremamente necessário saber realizar o processo de reconhecimento relacionado à verificação do que é necessário para este tipo de API. Dessa forma, você entenderá que existem alguns princípios que auxiliam na estrutura de projetos de arquiteturas de APIs, por exemplo, os princípios relacionados à engenharia de software, que auxiliam no processo do estilo da arquitetura da API.

Você já sabe que uma API não é um programa, mas, sim, uma interface, a qual, como costumamos dizer, será consumida (utilizada, usada, executada, etc.) por uma aplicação, um programa, e não por um usuário. Entenda que o consumo de uma API faz com que as funcionalidades de uma interface possam ser utilizadas, neste caso, a aplicação, o programa, não tem conhecimento desta utilização da API, porque, originalmente, ela não faz parte de sua implementação.

Vamos conhecer como deve ser a estrutura de um projeto de arquitetura de uma API RESTful, verificando todas as suas características, funcionalidades e aplicabilidade.

3.1 Características gerais de projetos de arquiteturas de APIs RESTful

De maneira geral, você deve entender que uma API possui três tipos de público, por isso, elas podem ser: APIs privadas, que são aquelas pertencentes a alguma organização; APIs públicas, as quais, nesse caso, estão disponíveis para qualquer pessoa; APIs parceiras, que são cedidas para alguma pessoa de confiança.

Cada tipo de API possui um projeto de arquitetura e pode ser utilizado para um determinado projeto. Quando falamos em projetos de arquitetura de APIs, é necessário levar em consideração alguns requisitos, para que a implantação possa ocorrer da maneira correta. Seja qual for o tipo de API que você optar, sempre considere informações básicas, porém muito importantes, como: segurança, documentação, requisitos de acesso e de gerenciamento. Todas elas auxiliam no processo final de implementação de uma API e, se bem construídas, podem tornar sua implementação bem eficaz.

O REST é uma arquitetura que, inicialmente, foi utilizada somente para Web Services. Possui o protocolo HTTP como referência de comunicação e transferência de informações. Além disso, nesta arquitetura, os padrões são tecnologias HTML, XML e JSON. Uma API REST é definida como:

Uma API REST não deve ser dependente de nenhum protocolo de comunicação único, embora seu mapeamento bem-sucedido para um determinado protocolo possa depender da disponibilidade de metadados, escolha de métodos, etc. Em geral, qualquer elemento de protocolo que usa um URI para identificação deve permitir qualquer esquema de URI a ser usado para fins dessa identificação. (FIELDING, 2008, [s. p.])

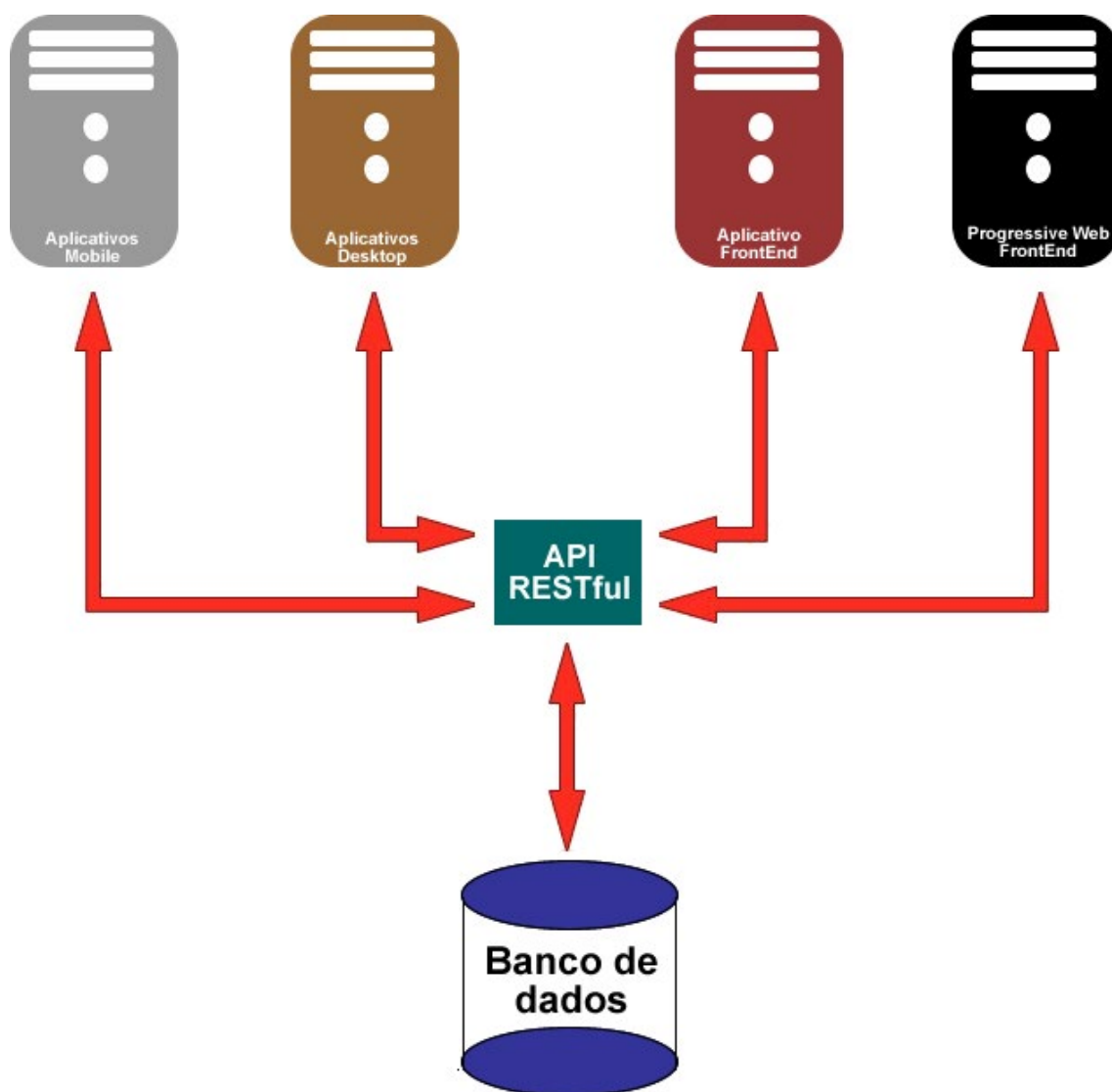
E o que significa API RESTful? O RESTful foi definido por **Roy Fielding** como uma plataforma que utiliza os princípios da API REST. Em outras palavras, ele é uma arquitetura que utiliza um conjunto de regras e

princípios, por exemplo, cliente-servidor, interface uniforme, *stateless*, cache e camadas. Para que você possa entender o RESTful, é necessário conhecer cada um desses princípios:

- **Cliente-servidor:** utiliza o princípio de separar a interface de usuário do armazenamento de dados, dessa forma, existe como resultado a evolução para a não dependência entre cliente-servidor.
- **Interface uniforme:** a interface uniforme entre os componentes cliente e servidor utiliza princípios de identificação e representação de recursos, além de mensagens descritivas, para manter uma boa comunicação entre interfaces de aplicações, mantendo uma interface uniforme.
- **Stateless:** um *stateless* não conhece as aplicações que estão conectadas a ele nem o conteúdo dos dados dessas aplicações ou da forma como elas executam suas funções (BERKENBROCK, 2005). *Stateless* significa sem estado. É um princípio que permite a comunicação entre cliente-servidor, independentemente de estado, em que uma requisição precisa ter toda informação para que possa se tornar compreensível. O *stateless* gera um tráfego alto de dados, o que acaba reduzindo a performance da aplicação, algo que pode ser reduzido pela utilização de cache.
- **Cache:** melhora a performance da aplicação, pois reduz o tempo de resposta de interações cliente-servidor. O cache é controlado pelo servidor com o auxílio do cabeçalho HTTP (*HTTP Header*).
- **Camadas:** as camadas de uma aplicação oferecem menos complexidade e tornam qualquer aplicação sujeita à mudança, o que não é um ponto negativo. Todo projeto de arquitetura de API deve ser desenvolvido através de camadas gerenciadas de forma independente.

A Figura 4 mostra de que forma funciona um projeto de arquitetura de API RESTful.

Figura 4 - API RESTful



Fonte: elaborada pelo autor.

Nesta leitura digital, você teve a oportunidade de conhecer de que forma os projetos de arquiteturas de APIs e Web Services funciona, verificando as tecnologias que são envolvidas neste processo. Dessa forma, é de extrema importância que você sempre faça a análise do projeto que está desenvolvendo, para que seja realizada a escolha correta de todos os detalhes que cercam a utilização de APIs e Web Services.



Referências

BERKENBROCK, C. D. M. **Investigação e implementação de estratégias de notificação de invalidação para coerência de cache em ambientes de computação móvel sem fio**. 2005. Dissertação (Mestrado em Ciências da Computação) – Universidade Federal de Santa Catarina, Florianópolis, 2005.

CEZAR, P. API REST: princípios e boas práticas para serviços RESTful. **Smart TI**, 2018. Disponível em: <https://smarti.blog.br/api-rest-principios-boas-praticas-para-arquiteturas-restful/>. Acesso em: 25 ago. 2021.

FIELDING, R. T. REST API must be hypertext-driven. **Untangled**, 2008. Disponível em: <https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>. Acesso em: 1º nov. 2021.

SHARP, J. **Visual C# 2010: passo a passo**. Porto Alegre, RS: Bookman, 2011.

Consumo das APIs e Web Services por soluções do *front-end*

Autoria: Arthur Gonçalves Ferreira

Leitura crítica: Rennan Martini Rodrigues



Objetivos

- Testar e implementar APIs e Web Services.
- Entender o processo de consumo e conexão com *front-end*.
- Utilizar o programa Node.js.



1. Implementando APIs e Web Services com Node.js

Olá, aluno! Nesta unidade, você aprenderá a implementar e testar APIs e Web Services básicos utilizando o programa Node.js e entenderá como ocorre o processo de consumo e conexão dos APIs e Web Services através de um *front-end*. Conhecer todos os processos e fundamentos do funcionamento de uma implementação e teste de APIs e Web Services ajudará você, de forma prática e dinâmica, a compreender de que forma eles funcionam.

Realizaremos um passo a passo para a criação de um exemplo prático de APIs utilizando o programa Node.js. Segundo Pereira (2016, p. 13):

O Node.js é uma plataforma altamente escalável e de baixo nível. Nele, você vai programar diretamente com diversos protocolos de rede e internet, ou utilizar bibliotecas que acessam diversos recursos do sistema operacional. Para programar em Node.js basta dominar a linguagem JavaScript, isso mesmo, JavaScript! E o *runtime* JavaScript utilizado nesta plataforma é o famoso JavaScript V8, que é usado também no Google Chrome.

Desta forma, você aprenderá algumas configurações e como criar um projeto básico que, ao final, será consumido por um *front-end*. Ao final desta implementação, você será capaz de construir um API, entendendo todas as funcionalidades, características e aplicabilidades que envolvem esse desenvolvimento.

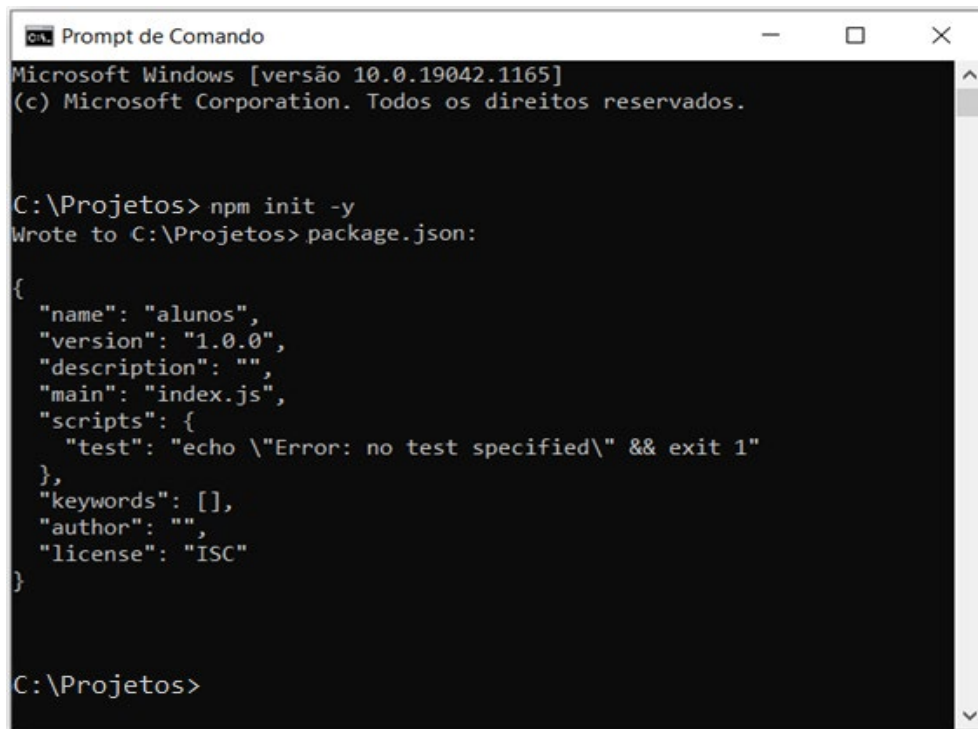
1.1 Instalando o pacote JSON

Antes de qualquer coisa, entenderemos qual API será criada. Neste caso, trata-se de uma bem básica, simples, que servirá apenas para que você possa entender seu funcionamento e, posteriormente, seu consumo. A API que construiremos trata de informações básicas de alunos. A ideia é simular uma consulta a um banco de dados, a qual retornará uma lista de alunos com suas informações.

Para a utilização do Node.js, é necessário realizar algumas configurações, instalação de pacotes e organizar as atividades que serão realizadas, para que tudo possa ser organizado. Primeiro, criaremos uma pasta, a qual guardará a API. Para isso, neste nosso exemplo, criaremos uma pasta no diretório raiz do Windows, cujo caminho é C:\Projetos. Este caminho é muito importante não só para manter uma organização mas também para o passo seguinte, que tratará sobre a instalação de alguns pacotes.

Precisaremos abrir o CMD do Windows para instalar um pacote **package.json**. Este pacote é um arquivo JSON, considerado o coração de qualquer projeto do Node. Ele registra **metadados** importantes sobre um projeto, que são necessários antes de publicar no **NPM**, e define os atributos funcionais de um projeto que o NPM usa para instalar dependências, executar *scripts* e identificar o ponto de entrada para nosso pacote. Conforme Taylor (2003), metadado é a informação estruturada que descreve atributos de recursos informacionais com o propósito de identificação, descoberta e, às vezes, administração. A Figura 1 mostra a instalação do package.json.

Figura 1 - package.json



```
Prompt de Comando
Microsoft Windows [versão 10.0.19042.1165]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Projetos> npm init -y
Wrote to C:\Projetos>package.json:

{
  "name": "alunos",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

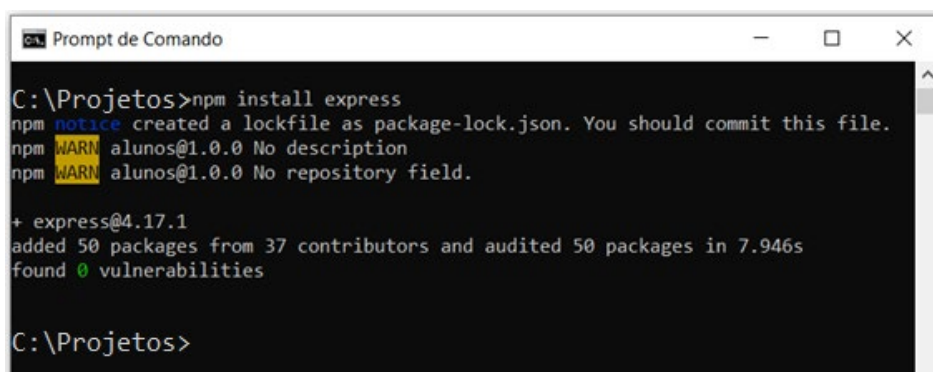
C:\Projetos>
```

Fonte: elaborada pelo autor.

1.2 Instalando o pacote express npm

Após a instalação do package.json, é necessário instalar outro pacote, o express npm. Para isso, você precisa digitar o comando **npm install express**. A Figura 2 mostra a linha de comando npm install express inserida no cmd e o resultado positivo para sua instalação.

Figura 2 - Express npm



```
Prompt de Comando

C:\Projetos>npm install express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN alunos@1.0.0 No description
npm WARN alunos@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 50 packages in 7.946s
found 0 vulnerabilities

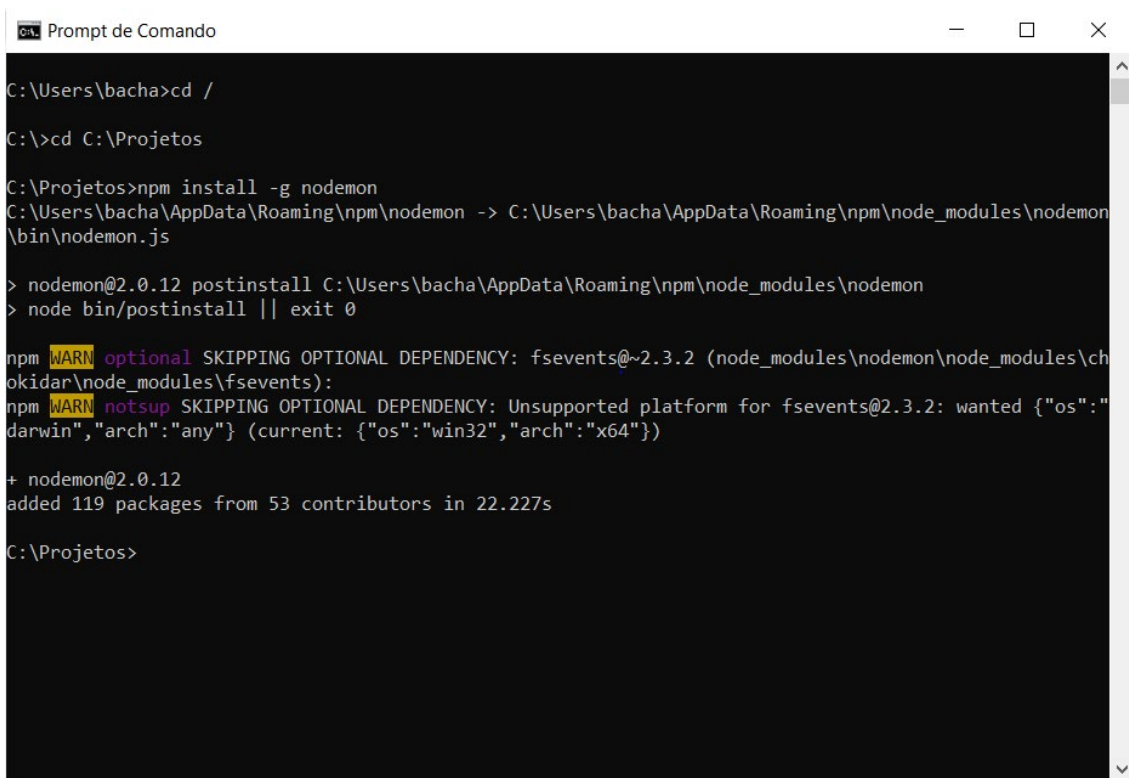
C:\Projetos>
```

Fonte: elaborada pelo autor.

1.3 Instalando o pacote nodemon

Instalaremos outro pacote, conhecido como **nodemon**. Este pacote é uma ferramenta que ajuda a desenvolver aplicativos baseados em Node.js, reiniciando automaticamente o servidor quando mudanças de arquivo no diretório são detectadas. A Figura 3 mostra a instalação do nodemon.

Figura 3 - Instalando o nodemon



```

C:\Users\bacha>cd /

C:\>cd C:\Projetos

C:\Projetos>npm install -g nodemon
C:\Users\bacha\AppData\Roaming\npm\nodemon -> C:\Users\bacha\AppData\Roaming\npm\node_modules\nodemon\bin\nodemon.js

> nodemon@2.0.12 postinstall C:\Users\bacha\AppData\Roaming\npm\node_modules\nodemon
> node bin/postinstall || exit 0

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@~2.3.2 (node_modules\nodemon\node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.3.2: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ nodemon@2.0.12
added 119 packages from 53 contributors in 22.227s

C:\Projetos>
```

Fonte: elaborada pelo autor.

2. Implementando API e Web Services

Para que você possa entender melhor a estrutura de projeto criada até aqui, é importante explicar que, na pasta Projetos, criada na pasta raiz do Windows, em C:, temos a pasta node_modules (criada com o package.json) e o próprio arquivo package.json. Para dar continuidade a essa estrutura, será necessário criar um arquivo .js (JavaScript).

Você pode escolher um editor da sua preferência, mas saiba que pode utilizar um simples bloco de notas para criar um arquivo .js, sendo assim, abra um bloco de notas, vá em “salvar como” e coloque o nome app.js. Agora, temos mais um arquivo na pasta Projetos. Com a estrutura inicial do desenvolvimento, começaremos a implementar códigos no nosso arquivo app.js e entender melhor como express funciona.

2.1 Criando API JSON

Abriremos nosso arquivo app.js, lembrando mais uma vez que você escolher um editor da sua preferência. No nosso exemplo, continuaremos utilizando o bloco de notas. Sendo assim, clique com o botão direito do mouse em cima do arquivo app.js, abra-o com um bloco de notas e insira o seguinte código:

```
const express = require('express')
```

```
const app = express()
```

Na primeira linha, observaremos a importação do pacote express, e na segunda linha, começaremos a instanciar (solicitar) este pacote para que possamos utilizar suas funções. Após inserir essas duas primeiras linhas de códigos, criaremos um *Array* de objetos JavaScript, que terá a função de guardar informações, as quais serão devolvidas para quem solicitar. No nosso exemplo, essas informações são de alunos. A Figura 4 mostra a estrutura de todo o arquivo app.js já com o Array.

Figura 4 - Estrutura app.js



app - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

```
const express = require('express') //importacao do pacote
const app = express() //instanciando express
```

```
const alunos = [
  {
    nome: 'João',
    idade: 12
  },
  {
    nome: 'Maria',
    idade: 13
  },
  {
    nome: 'José',
    idade: 14
  },
  {
    nome: 'Batista',
    idade: 15
  }
]
```

```
app.get('/', function (req, res) {
  res.send(alunos)
})
//localhost:8080
app.listen(8080);
```

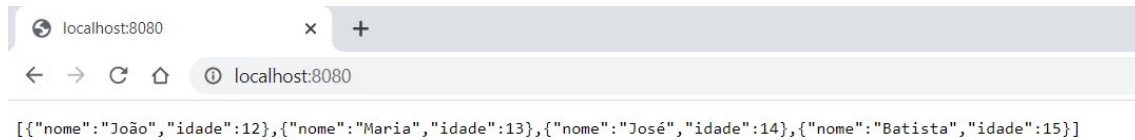
Fonte: elaborada pelo autor.

2.2 Testando API json no navegador

Já podemos verificar no servidor local o resultado do app.js, mas, antes, é necessário conectar o servidor pelo CMD do Windows. No CMD, você deve inserir o comando `nodemon app.js` para poder se conectar. Lembre-se de executar o comando na pasta do projeto, portanto, no nosso exemplo, em C:/Projetos. Após realizar a conexão, abra um

navegador e insira o endereço `http://localhost:8080`. A Figura 5 mostra o resultado.

Figura 5 - Resultado do app.js no localhost



Fonte: elaborada pelo autor

2.3 Entendo a importância dos verbos HTTP na API

Com a API concluída, é possível consumi-la direto de um navegador, de uma aplicação web ou até mesmo de um aplicativo. Dessa forma, precisamos entender de que forma se dará esse consumo, sendo de extrema importância o entendimento do protocolo HTTP e de seus verbos.

Falar do protocolo HTTP é falar sobre a arquitetura REST. Essa arquitetura pode ser entendida como a padronização da arquitetura web, sendo considerada uma abstração do protocolo HTTP, URL e os verbos GET, POST, PUT e DELETE. Uma API REST é reconhecida através de uma URL, e quase sempre o que determina o que ela vai fazer ou não são os verbos utilizados no HTTP. Segundo Fielding (2008, [s. p.]):

Uma API REST não deve ser dependente de nenhum protocolo de comunicação único, embora seu mapeamento bem-sucedido para um determinado protocolo possa depender da disponibilidade de metadados, escolha de métodos, etc. Em geral, qualquer elemento de protocolo que usa um URI para identificação deve permitir qualquer esquema de URI a ser usado para fins dessa identificação.

O Quadro 1 mostra os principais verbos existentes e suas funções.

Quadro 1 | Verbos HTTP

Verbo	Função
GET	Solicita representações de recursos que retornam apenas dados.
HEAD	Solicita resposta de um recurso sem o corpo da resposta.
POST	Causa mudança no estado do recurso, modifica.
PUT	Utiliza cargas de dados de requisição, insere uma informação.
DELETE	Deleta o recurso especificado.
CONNECT	Estabelece via de conexão com o servidor identificado pelo recurso de destino.
OPTIONS	Descreve meios de comunicação com o recurso de destino.
TRACE	Auxilia o recurso de destino, realizando testes de chamadas.
PATCH	Realiza modificações em parciais em um recurso.

Fonte: elaborado pelo autor.

O verbo GET pode ser utilizado acessando o localhost:8080 e recuperar uma lista de alunos; o verbo PUT pode acessar o localhost:8080 para modificar o estado de um aluno; o verbo DELETE pode acessar o localhost:8080, excluindo um cadastro de um aluno; o verbo POST pode acessar o localhost:8080 para inserir um novo aluno.

3. Consumindo API e Web Services

Após a construção da API, podemos consumi-la através de uma página web ou de uma aplicação. Considerando o significado de forma literal, consumir uma API ou um Web Service significa utilizar ou gastar. Este termo é conceituado no desenvolvimento como o ato de utilizar ao menos uma das funcionalidades que a aplicação oferece, ou seja, consumir uma API ou um Web Service significa utilizar, no mínimo, uma de suas funcionalidades.

Mesmo que uma API seja simples, ela oferece funcionalidades e pode ser consumida, sendo que uma das formas mais fáceis de consumir uma API é através da utilização de um *framework*. Só o fato de você poder acessar a URL em que está hospedada a API já significa consumi-la. O *framework* é um programa no qual podemos realizar diversas ações, como usar os verbos HTTP para consumir uma API.

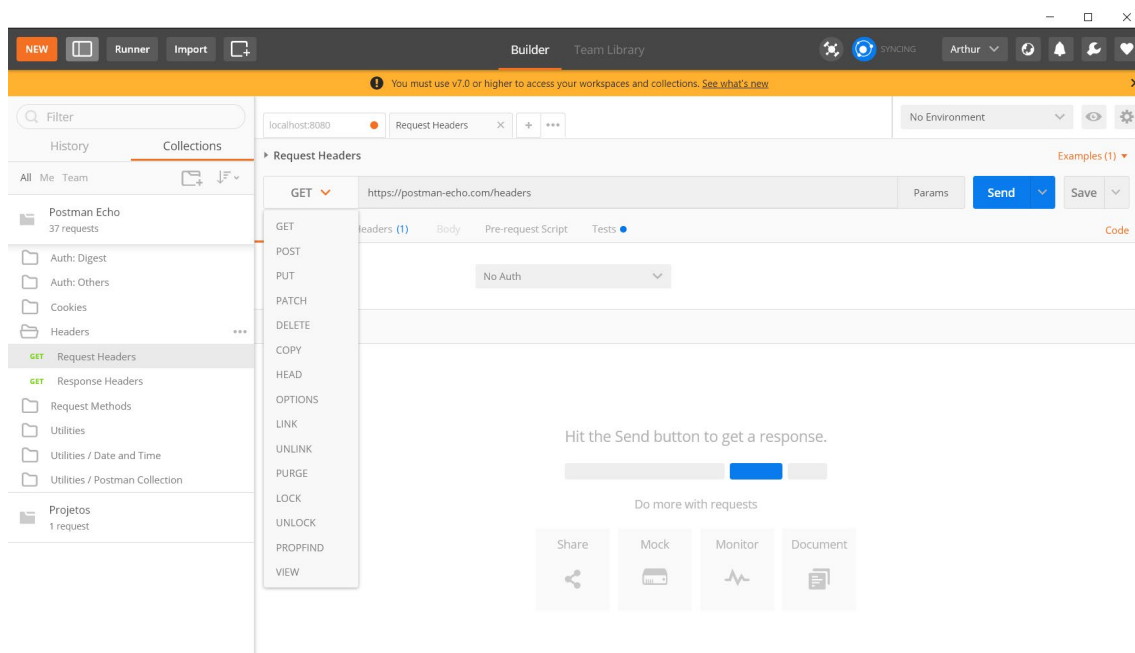
3.1 *Framework* Postman

O *framework* Postman é uma plataforma de API para construir e usar APIs. Este *framework* simplifica todas as etapas do ciclo de vida da API e agiliza a colaboração, para que você possa criar APIs com qualidade e agilidade. A plataforma API Postman apresenta um conjunto de configurações e ferramentas que auxiliam no desenvolvimento do design, na realização de testes, na construção de documentação e no consumo de APIs.

Postman é uma ferramenta *open-source*, multiplataforma, sendo possível utilizá-la baixando o programa e instalando-o no computador, ou baixando e incorporando-o no próprio navegador Google Chrome, sem a necessidade de instalar no computador, para isso, é necessário baixar o programa pela Google Store. Para incorporar o Postman no navegador Google Chrome, acesse a Chrome Web Store. Após baixar e instalar, é necessário realizar um cadastro, para que possa utilizar a aplicação.

Através do Postman, conseguimos enviar diversas requisições em vários formatos de arquivos, como XML e JSON. É possível também verificar o resultado de uma requisição, o tempo de processamento de uma requisição com o seu status, além de enviar requisições através de vários verbos do HTTP. Com o Postman, podemos, ainda, armazenar e compartilhar testes entre plataformas, desde que o usuário realize login com uma conta na plataforma. A Figura 6 mostra a interface da plataforma Postman.

Figura 6 - Plataforma Postman



Fonte: elaborada pelo autor.

3.2 Criando coleções e pastas no Postman

Consumiremos a API que criamos na plataforma Postman, para isso, é necessário criarmos uma coleção. Você deve clicar na opção *Collections*, que fica do lado esquerdo, ao lado da opção *History*. Ao fazer isso, você deve clicar no ícone de uma pasta e, em seguida, dar um nome para a coleção e clicar em *Create*.

Após criar uma coleção, você deve criar uma pasta dentro dela, conforme o seu critério. Para isso, selecione a coleção criada, clique na reticência para abrir o submenu da coleção e clique em *Add Folder* para adicionar uma nova pasta. Uma coleção é como se fosse um projeto novo que você está iniciando, então criar pastas dentro dela dá a ideia de organização, portanto fique à vontade para criar quantas pastas quiser, desde que elas possuam uma função dentro da coleção.

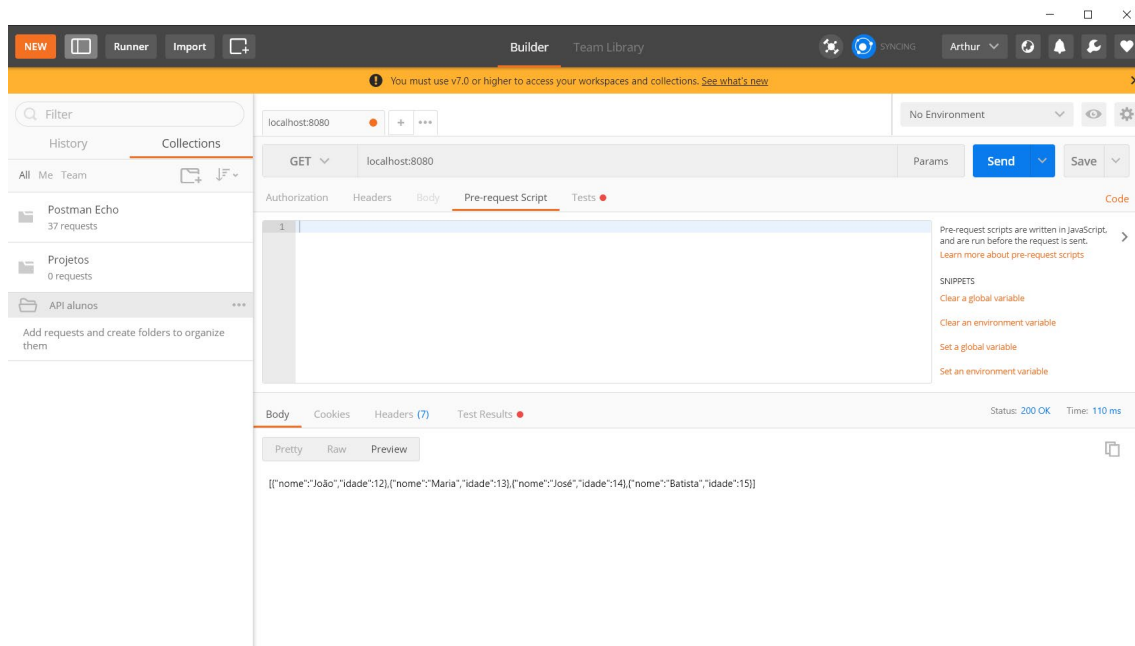
Para o consumo da API *app.js*, que simula uma consulta a um banco de dados com informações de alunos, foi criada uma coleção com o nome

Projetos na plataforma Postman e, dentro dela, uma pasta chamada API alunos. Dessa forma, realizaremos requisições na API app.js dentro da pasta API alunos.

3.3 Consumindo API app.js com requisições de verbos HTTP

Para realizar uma requisição com o verbo GET na plataforma Postman, você deve copiar todo o código que criamos no arquivo app.js, acessar a plataforma e a pasta API alunos que criamos anteriormente e colar o código na área destinada à aba *Tests*. Informe o link do servidor local (localhost:8080) e clique no botão *Send*. A Figura 7 mostra o resultado da requisição GET.

Figura 7 - Requisição get



Fonte: elaborada pelo autor.

Nesta leitura digital, você pode aprender um pouco como funciona a implementação e o teste de APIs e Web Services utilizando o node.js e consumindo as APIs com a plataforma Postman. Desta forma, você poderá construir suas APIs, começando pelas mais básicas

e desenvolvendo-as até que comece a dominar e utilizar todas as ferramentas necessárias.



Referências

CAVALCANTE, I. REST API com Node.js: Back-end e Front-end. **Medium**, 2018. Disponível em: <https://iagoangelimc.medium.com/rest-api-com-node-js-back-end-e-front-end-2d6604eed890>. Acesso em: 9 set. 2021.

FIELDING, R. T. REST APIs must be hypertext-driven. **Untangled**, 2008. Disponível em: <https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>. Acesso em: 1º nov. 2021.

PEREIRA, C. R. **Construindo APIS REST com node.js**. São Paulo: Casa do Código, 2016.

TAYLOR, A. G. **The organization of the information Westport**. London: Libraries Unlimited, 2003.

Decisões tecnológicas para o desenvolvimento e testes de APIs e Web Services

Autoria: Arthur Gonçalves Ferreira

Leitura crítica: Rennan Martini Rodrigues



Objetivos

- Aprender a utilizar ferramentas para o desenvolvimento de Web Services e APIs.
- Conhecer ferramentas para a realização de testes de Web Services e APIs.
- Compreender boas práticas de testes de APIs.



1. Decisões tecnológicas para o desenvolvimento e teste de Web Services

As tecnologias Web Services são aplicações de rede muito eficazes em comunicações de máquina para máquina. Dessa forma, afirma-se que elas são utilizadas mundialmente com esse intuito. Para que isso ocorra, são constituídas de tecnologias, como SOAP, REST, XML, HTTP, entre outras.

Podemos construir e desenvolver um Web Service utilizando ferramentas ou de forma manual. A não utilização de ferramentas acarreta um desenvolvimento mais complicado, sendo assim, a forma mais fácil de desenvolver e testar Web Services é através da utilização de ferramentas que auxiliam em todo o seu processo. Neste tópico, verificaremos de que forma ocorre o desenvolvimento e teste de Web Services sem ferramentas e, posteriormente, analisaremos as que podem ser utilizadas.

1.1 Desenvolvendo e testando tecnologias Web Services sem ferramentas

Um conhecimento prévio que você já deve ter até aqui é que qualquer tecnologia utilizada para a criação de Web Services está fundamentada na utilização de documentos no formato XML. Segundo Magalhães (2020, [s. p.]):

O XML, sigla para eXtensible Markup Language, é um tipo de linguagem de marcação que define regras para codificar diferentes documentos. É muito utilizado para a criação de Notas Fiscais Eletrônicas, também chamadas de NF-e, por armazená-las e ainda garantir uma assinatura digital.

Para que você possa realizar a leitura de um arquivo do tipo XML, é indicada a utilização de um programa que execute esta ação, o qual seja capaz de realizar não só a leitura do documento mas também

qualquer operação. Mesmo que existam programas para manipulação de XML, é possível manipular esses tipos de arquivos de forma manual, realizando a leitura, a gravação e a navegação dos componentes em um determinado documento.

Neste contexto de utilização de tecnologias Web Services sem ferramentas, é importante entender sobre a função de um documento WSDL. Segundo Tayt-Son ([s. d.], [s. p.]):

WSDL é uma linguagem baseada em XML que descreve um Web Service e a maneira de acessá-lo. WSDL é a tecnologia que vai definir todas as informações que um cliente qualquer precisa conhecer para se integrar e utilizar o serviço. Um documento WSDL nada mais é do que um documento do tipo XML como os dados referentes ao serviço a ser utilizado.

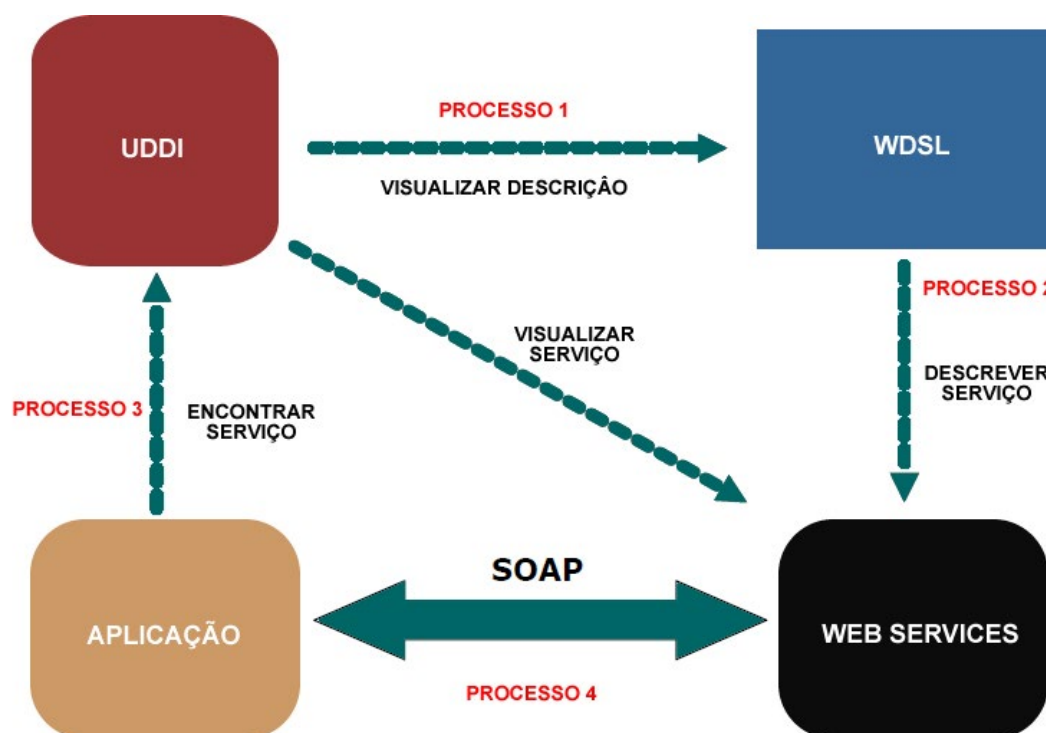
Desta forma, podemos imaginar um exemplo em que um Web Service é utilizado em um sistema desenvolvido em linguagem Java. Neste contexto, o WSDL desempenha sua função de acordo com os métodos utilizados no sistema desenvolvido em Java, sendo que, necessariamente, esses métodos são definidos nas classes utilizadas em sua estrutura. Os dados que serão transportados também devem ser levados em consideração, por exemplo, os *endpoints* (URL do serviço que será acessado) e os pontos de acesso ao serviço.

O protocolo e a arquitetura SOAP de uma tecnologia Web Service realizam a função de enviar e receber mensagens, sendo que o responsável por esse processo é o protocolo HTTP. A comunicação realizada pelo HTTP pode acontecer sem um auxílio de ferramentas, porém isso torna o trabalho do desenvolvedor muito complicado.

Outro exemplo em que podemos realizar o trabalho manual da utilização de tecnologia Web Service sem o auxílio de ferramentas está ligado à publicação de um serviço. Toda publicação Web Service utiliza registros UDDI e é feita no *service provider* (provedor de serviço), que é a mesma coisa que Web Service.

A figura a seguir mostra a importância da utilização de ferramentas no desenvolvimento e teste de Web Services.

Figura 1 - Automação de tecnologias Web Services



Fonte: elaborada pelo autor.

É possível observar que a utilização de ferramentas auxilia no desenvolvimento e no teste de Web Services, tornando seus processos e suas requisições mais simples de serem executados. Sendo assim, a não utilização de ferramentas no desenvolvimento de Web Services acaba limitando um sistema e dificultando um desenvolvimento apropriado e seguro.

1.2 Desenvolvendo e testando tecnologias Web Services com ferramentas

Existem diversas ferramentas utilizadas para o desenvolvimento e teste de Web Services. Grandes empresas investem nesse mercado, por exemplo, a Sun Microsystems e a IBM. É importante ressaltar que

empresas de menor porte também atuam nesse mercado e oferecem ferramentas tão boas quanto as grandes empresas citadas.

Antes de escolher uma ferramenta para o desenvolvimento e teste de uma Web Service, você deve verificar alguns fatores que podem influenciar na escolha, como:

- Fase em que o projeto de desenvolvimento da Web Service se encontra.
- Adaptação em ambientes de desenvolvimento integrado (IDEs).
- Capacidade de crescimento do sistema, sem perder a qualidade (escalabilidade). Dessa forma, é realizada a implementação do *server-side* e do *stateless*.
- Possibilidade de documentar o sistema.

Dentre as ferramentas mais conhecidas para o desenvolvimento e teste de Web Services estão framework.net, Java Apache Axis, JAXRPC, IBM Web service Toolkit e Java Web Service Developer Pack. Verificaremos suas características.

• **IBM web service toolkit**

O IBM web service toolkit fornece recursos confiáveis e seguros para montar e implementar serviços da web utilizando o Java. Este pacote foi projetado para IBM WebSphere Application Server, versão 6.1, estendido com o pacote de recursos do WebSphere Application Server para serviços da web. Suporta o desenvolvimento de serviços da web a partir de Java anotado ou de Web Services Description Language (WSDL) usando qualquer esquema XML.

O Quadro 1 mostra funções e componentes do IBM web service toolkit.

Quadro 1 - Funções e componentes do IBM web service toolkit

UDDI Java API (UDDI4J)	Utiliza banco de dados DB2 para realizar manutenções de registros UDD públicos ou privados.
Java2WSDL	Utilitário do IBM web service toolkit que gera documento WSDL na linguagem Java.
Apache AXIS	Realiza o envio e o recebimento de mensagens SOAP.
Xerces	É um analisador sintático (<i>parser</i>) XML para Java. Dessa forma, transforma o código XML legível para Java.
WebSphere Application Server	É o servidor de aplicação do IBM web service toolkit.

Fonte: elaborado pelo autor.

O IBM web service toolkit possui uma interface gráfica, desenvolvida com Java Swing, sendo considerado multiplataforma, de fácil utilização, tornando simples a utilização de todos os recursos disponíveis neste ambiente.

A tecnologia de Web Services evoluiu muito com o passar dos anos, e a IBM criou vários requisitos de segurança, identificação, gerenciamento e notificação.

• Java Web Service Developer Pack

O Java Web Services Development Pack é um kit de desenvolvimento de software gratuito para o desenvolvimento de serviços web, aplicativos web e aplicativos Java com as mais recentes tecnologias para Java. Ele reúne várias tecnologias, as quais são utilizadas pelos Web Services e integram todo o ciclo de desenvolvimento.

O API Java do Java Web Service Developer Pack auxilia o desenvolvedor realizando a geração de documentos WSDL, usa o **RPC** para realizar chamadas de métodos em serviços disponíveis, usa o SOAP para envio e recebimento de mensagens e publica um serviço no *service providers*. Um RPC, segundo Gandarella:

A parte 'RPC' significa 'Remote Procedure Call', e é essencialmente o mesmo que chamar uma função em JavaScript, PHP ou Python, e assim por diante, utilizando um nome de método e argumentos. Como o XML não é facilmente digerível para todos, uma API RPC pode usar o protocolo JSON-RPC (1000x mais simples) ou pode rolar uma API baseada em JSON personalizada, como o Slack fez com a API da Web. (GANDARELLA, 2017, [s. p.])

• .NET Framework

O .NET Framework oferece automatização de tarefas, realizando a geração de classes para o desenvolvimento e teste de Web Services. Diversas ferramentas auxiliam a integração de um Web Service, dessa forma, com o .NET Framework, é possível desenvolver programas que interagem de maneira simples com as ferramentas do ambiente, além de publicá-los em um servidor com um esforço minimizado.

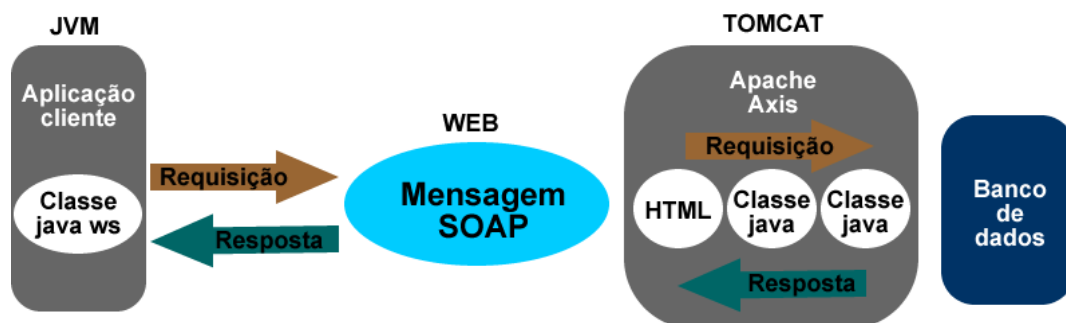
O .NET Framework oferece ao desenvolvedor a simplicidade de poder construir sistemas sem se preocupar em construir códigos relacionados aos detalhes de estrutura relativos ao transporte de informações, à organização e padronização de documentos e às requisições realizadas ao servidor. Todo código relacionado à estrutura da disponibilização, da criação e dos testes dos Web Services é definido e configurado de forma automática por todas as classes deste *framework*.

• Apache Axis

O *framework* Axis foi desenvolvido pela Apache Software Foundation e possui uma estrutura de serviço da web de código aberto baseada em XML. Ele consiste em uma implementação em Java e C++ do servidor SOAP, além de vários utilitários e APIs, para gerar e implantar aplicativos de serviço da web. Foi desenvolvido na linguagem Java e utiliza XML com o protocolo SOAP. É possível desenvolver softwares distribuídos utilizando a linguagem Java ou C++.

A Figura 2 mostra o funcionamento do *framework* Apache Axis.

Figura 2 | Funcionamento do *framework* Apache Axis



Fonte: elaborada pelo autor.

Existem outras ferramentas que auxiliam no desenvolvimento e teste de uma Web Service, por exemplo, o JAX-RPC, que é um acrônimo de Java API for XML-based RPC (API Java para RPC baseado em XML). Essa API faz com que uma aplicação Java chame por um Web Service estruturado em Java, consistente e com sua descrição no WSDL.

Não existe uma ferramenta padrão, pois cada desenvolvedor deverá escolher uma ferramenta adequada ao seu projeto, levando em consideração que a ferramenta escolhida deve possuir comunicação HTTP e saiba manipular dados em XML com a implementação Web Service.



2. Decisões tecnológicos para o desenvolvimento e teste de APIs

A tecnologia API é extremamente importante para a comunicação entre interfaces de aplicações. As APIs são classificadas e conceituadas como um conjunto de instruções, códigos bem estruturados e definidos, que são disponibilizados em um servidor. Dessa forma, temos, ainda, que elas são o *back-end* de uma aplicação, que pode ser consumida por vários sistemas.

Um desenvolvedor ou uma equipe de desenvolvedores precisa tomar decisões tecnológicas para o desenvolvimento e testes de APIs, levando em consideração alguns fatores que veremos a seguir. Assim, a escolha de tecnologias para o desenvolvimento e teste de APIs deve ser um dos primeiros passos antes de iniciar um projeto eficaz.

2.1 Ferramentas para o desenvolvendo de tecnologia APIs

A primeira coisa que todo desenvolvedor de algum projeto de API deve saber é que, na criação de uma API, é utilizado o protocolo HTTP, por isso, é necessário conhecer as principais características dele. Além do protocolo HTTP, são utilizados também servidores web e linguagens de programação, PHP, JavaScript, Ruby, Python, Java, entre outras.

Não podemos esquecer do famoso JavaScript, uma linguagem de programação muito utilizada no desenvolvimento de APIs. Existem diversas APIs totalmente desenvolvidas em JavaScript, o que torna a utilização dessa linguagem essencial. É importante ressaltar que, apesar da grande importância do JavaScript para o desenvolvimento de APIs, você não é obrigado a conhecer a linguagem ou até mesmo só desenvolver com ela, pois uma API pode ser desenvolvida com qualquer linguagem que permita uma interface HTTP.

Outro fator extremamente importante para o desenvolvimento de uma API está relacionado ao objetivo dela, definindo o que você deseja alcançar e transmitir na interação entre aplicações. Logo, no desenvolvimento de APIs, deve-se levar em consideração o escopo da aplicação e de que forma os dados dessa aplicação serão fornecidos.

2.2 Ferramentas para testes de tecnologias APIs

Realizar testes de APIs é muito importante para que você possa verificar se as funcionalidades da API que você criou estão realmente

funcionando. Para a realização desses testes, são utilizados softwares, os quais possuem o objetivo exclusivo de diagnosticar se uma API satisfaz algumas características relacionadas à funcionalidade, confiabilidade, performance e segurança da aplicação.

É fundamental utilizar ferramentas para realizar testes de APIs, porque testar APIs vai muito além de controlar a qualidade, a eficácia e a segurança da aplicação, pois é considerado também um ponto significativo no sucesso do desenvolvimento. Podemos citar algumas ferramentas que podem ser utilizadas: SoapUI, Postman, Katalon Studio e Rest-Assured.

SoapUI realiza testes dedicados de APIs REST SOAP, além de Web Services. Com essa ferramenta, você tem acesso ao código-fonte da aplicação e pode implementar novas funcionalidades nessa aplicação, caso seja uma necessidade. São vantagens do SoapUI:

- Utiliza os recursos *drag-and-drop* (arrastar e soltar), que auxiliam no processo de teste de APIs. Neste caso, oferecem ao desenvolvedor a facilidade de selecionar um código, arrastar e soltar na plataforma, sem a necessidade de digitar um código na íntegra.
- Criar códigos com a linguagem de programação Groovy. Essa linguagem é orientada a objetos, multiplataforma e pode ser executada nos principais sistemas operacionais, como Windows, Linux e MacOS.
- Cria simulações de interação com uma API através de dados carregados de ficheiros, bancos de dados ou Excel.
- Possui suporte de testes assíncronos.

O Postman pode ser utilizado como um plugin no Google Chrome nos sistemas operacionais Windows e Mac. São vantagens do Postman:

- Possui uma interface simples de fácil utilização.
- Realiza testes automáticos e testes exploratórios manuais.
- Pode ser executado nos sistemas operacionais Mac, Windows, Linux e Chrome Apps.
- Utiliza Swagger e RAML, que são programas usados para definir, criar, documentar e consumir APIs.

Katalon Studio realiza testes funcionais automáticos de API e Web Services. Essa ferramenta suporta requisições de arquiteturas SOAP e RESTful, podendo realizar os comandos GET, POST, PUT e DELETE. As vantagens do Katalon Studio são:

- Realiza testes com requisições de arquiteturas SOAP e RESTful.
- Compatível com o AssertJ, uma biblioteca de validação de mensagens de erro verdadeiramente úteis, melhora a legibilidade do código de teste e é projetado para ser fácil de usar em qualquer tipo de IDE.
- Suporta abordagens orientadas a dados.
- Pode ser utilizado por pessoas com ou sem conhecimento em desenvolvimento de software.

O Rest-Assured realiza testes de serviços API REST, oferecendo suporte para validar protocolo HTTP e requisições em JSON. Essa ferramenta é *open-source*, e suas principais vantagens são:

- Possui funcionalidades que proporcionam a criação de códigos do zero.
- Não há a necessidade de possuir conhecimentos específicos sobre HTTP.

2.3 Realizando testes bem-sucedidos de tecnologias APIs

Realizar testes de API é extremamente importante, porque é através deles que o funcionamento, o desempenho e a confiabilidade da aplicação são garantidos. Além disso, eles realizam a verificação de possíveis dependências que podem ocorrer entre aplicações.

Os testes são bem-sucedidos quando as APIs são bem **documentadas**, e isso abrange entender o objetivo delas, sua arquitetura, quais são as integrações que podem ser suportadas e quais são os seus recursos e funções. Para documentar uma API, é possível utilizar as ferramentas **Swagger, RAML e API Blueprint**.

Um teste de API pode ser realizado, inicialmente, em **pequenas funções**, por isso, podemos criar pequenos casos, nos quais podemos implementar um código curto de teste de API para verificar se o resultado é esperado ou inesperado.

É muito comum realizar testes manuais de APIs, porém nem todos que estão envolvidos em testes de APIs possuem o conhecimento sobre desenvolvimento, por isso, é uma boa prática realizar **testes automáticos**. Desta forma, é importante introduzir testes de automação para atividades rotineiras de teste de software.

Em muitos casos o teste de API precisa acontecer através de uma pessoa especializada, que tenha domínio sobre o assunto. Diante disso, é comum ter que contratar pessoas responsáveis pelo teste, as quais, geralmente, são conhecidas como engenheiros de testes automáticos.

Nesta leitura digital, você teve a oportunidade de conhecer e estudar sobre ferramentas que auxiliam no processo de desenvolvimento de APIs e Web Services e entender de que forma realizar seus testes (consumo). Além de todos esses conceitos, você estudou princípios relacionados à boa prática de testes de API. Isso ajudará você a

compreender um pouco mais sobre APIs e Web Services. Não deixe de estudar todo o material deste tema. Bons estudos!

Referências

- ANTUNES, M. API Restful: conceito, princípios e como criar. **Hostgator**, 2019. Disponível em: <https://www.hostgator.com.br/blog/api-restful/>. Acesso em: 10 set. 2021.
- GANDARELLA, W. E agora, API? REST ou RPC? Uma tentativa de entender cada arquitetura. **Medium**, 2017. Disponível em: <https://medium.com/lfdev-blog/e-agora-api-rest-ou-rpc-c24664d4755b>. Acesso em: 10 set. 2021.
- MAGALHÃES, A. L. O que é e para que serve o arquivo XML. **Canaltech**, 2020. Disponível em: <https://canaltech.com.br/software/xml-o-que-e/>. Acesso em: 10 set. 2021.
- MENÉNDEZ, A. I. M. **Uma ferramenta de apoio ao desenvolvimento**. 2002, 97 f. Dissertação (Mestrado em Informática) – Universidade Federal de Campina Grande, Campina Grande, 2002.
- TAYT-SON, L. B. D. C. Web Services. **Grupo de Teleinformática e Automação URFJ**, [s. d.]. Disponível em: https://www.gta.ufrj.br/grad/05_1/webservices/wsdl.htm. Acesso em: 14 set. 2021.



BONS ESTUDOS!