Faculdade de Tecnologia de Americana “Ministro Ralph Biasi”

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

LARISSA FELTRIN

APIs REST

" O impacto das APIs REST na evolução digital das empresas "

Americana, SP

2023

Faculdade de Tecnologia de Americana "Ministro Ralph Biasi"

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

LARISSA FELTRIN

APIs REST

"O impacto das APIs REST na evolução digital das empresas"

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do Prof. Me. Diógenes de Oliveira

Área de concentração: Programação de computadores.

Americana, SP.

2023

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana Ministro Ralph Biasi- CEETEPS Dados Internacionais de Catalogação-na-fonte

FELTRIN, Larissa

APIs REST: o impacto das APIs REST na evolução digital das empresas. / Larissa Feltrin – Americana, 2023.

44f.

Estudo de caso (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana Ministro Ralph Biasi – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Diógenes de OLIVEIRA

1. Análise de dados 2. Comunicação de dados 3. Sistemas de informação. I. FELTRIN, Larissa II. OLIVEIRA, Diógenes de III. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana Ministro Ralph Biasi

CDU: 681516
681519
681518

Elaborada pelo autor por meio de sistema automático gerador de ficha catalográfica da Fatec de Americana Ministro Ralph Biasi.

Larissa Feltrin

Larissa Feltrin

APIs REST

O impacto das APIs REST na evolução digital das empresas

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas pelo Centro Paula Souza – FATEC Faculdade de Tecnologia de Americana – Ralph Biasi.
Área de concentração: Programação de computadores.

Americana, 28 de novembro de 2023

Banca Examinadora:

Diógenes de Oliveira (Presidente)
Prof. Me.
FATEC Americana

Antônio Alfredo Lacerda (Membro)
Prof. Esp.
FATEC Americana

Jonas Bodê (Membro)
Prof. ~~Esp.~~ Mestre
FATEC Americana

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus e ao meu orientador Prof. Diógenes pelo incentivo, sabedoria e paciência pelas quais conduziu a orientação deste trabalho até o fim. Aos demais professores da Fatec Americana que contribuíram direta ou indiretamente para a melhoria deste trabalho.

DEDICATÓRIA

Dedico esse trabalho a Deus que me guiou e guardou durante todo o curso, quero usar tudo o que aprendi para fazer a vontade Dele, ou seja, o bem. Agradeço aos meus pais pelo incentivo e pela capacidade de acreditarem em mim. Ao meu marido Vinicius, cujo carinho, paciência e apoio constantes me trouxeram forças e coragem para superar as dificuldades encontradas no caminho.

RESUMO

Este trabalho descreve o funcionamento de serviços Web disponíveis através de uma API REST que é um meio muito utilizado atualmente para a interação entre sistemas heterogêneos. O estudo tem perspectiva de facilitar o serviço dos desenvolvedores, e aproveitar aplicativos já disponíveis que forneçam informações necessárias para outros aplicativos, independentemente da plataforma utilizada, com o objetivo de entregar mais rapidamente informações à uma aplicação confiável que soluciona o problema proposto. Para melhor criação e utilização, as APIs são guiadas por boas práticas de desenvolvimento e engenharia de software, seguindo modelo de arquitetura REST. As APIs ainda permitem futuros refinamentos e melhorias com diferentes infraestruturas, se adequando cada vez mais a atender necessidades de intercâmbio de informações entre diferentes empresas e sistemas, trazendo facilidade no dia a dia, economia e acessibilidade aos usuários. Tais conceitos serão revistos ao longo do trabalho.

Palavras Chave: REST, API, Serviços Web.

ABSTRACT

This work describes the functioning of Web services available through a REST API, which is a currently widely used means for interaction between heterogeneous systems. The solution aims to facilitate the service of developers, and take advantage of already available applications that provide necessary information for other applications, regardless of the platform used, with the aim of delivering quickly a reliable application that solves the proposed problem. For better creation and use, APIs are guided by good software development and engineering practices, following the REST architecture model. APIs also allow for future refinements and improvements with different infrastructures, increasingly adapting to meet information exchange needs between different companies and systems. Which provide support in everyday life, savings and accessibility to users. Such concepts will be reviewed throughout the work.

Keywords: *REST, API, Web services.*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API Application Programming Interface.

HATEOAS Hypermedia as the engine of application state.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

JSON JavaScript Object Notation.

OAUTH Autorização Aberta.

REST Representational State Transfer.

SQL Standard Query Language.

URL Uniform Resource Locator.

XML Extensible Markup Language.

1 INTRODUÇÃO

OBJETIVO

Esse trabalho tem como objetivo descrever o funcionamento das APIs e ressaltar sua relevância para as empresas e usuários de sistemas, demonstrando a facilitação nos serviços de comunicação e manipulação dos dados entre sistemas distintos. É citado também o impacto da utilização do REST no desenvolvimento desses sistemas com o objetivo de utilizar boas práticas e convenções de projeto, facilitando o seu uso e provendo as funcionalidades necessárias para quem consumir as APIs. Contudo, pode-se concluir por meio deste, que todos esses benefícios que trazem o uso de APIs, geram economias de tempo e consequentemente economia financeira para as empresas.

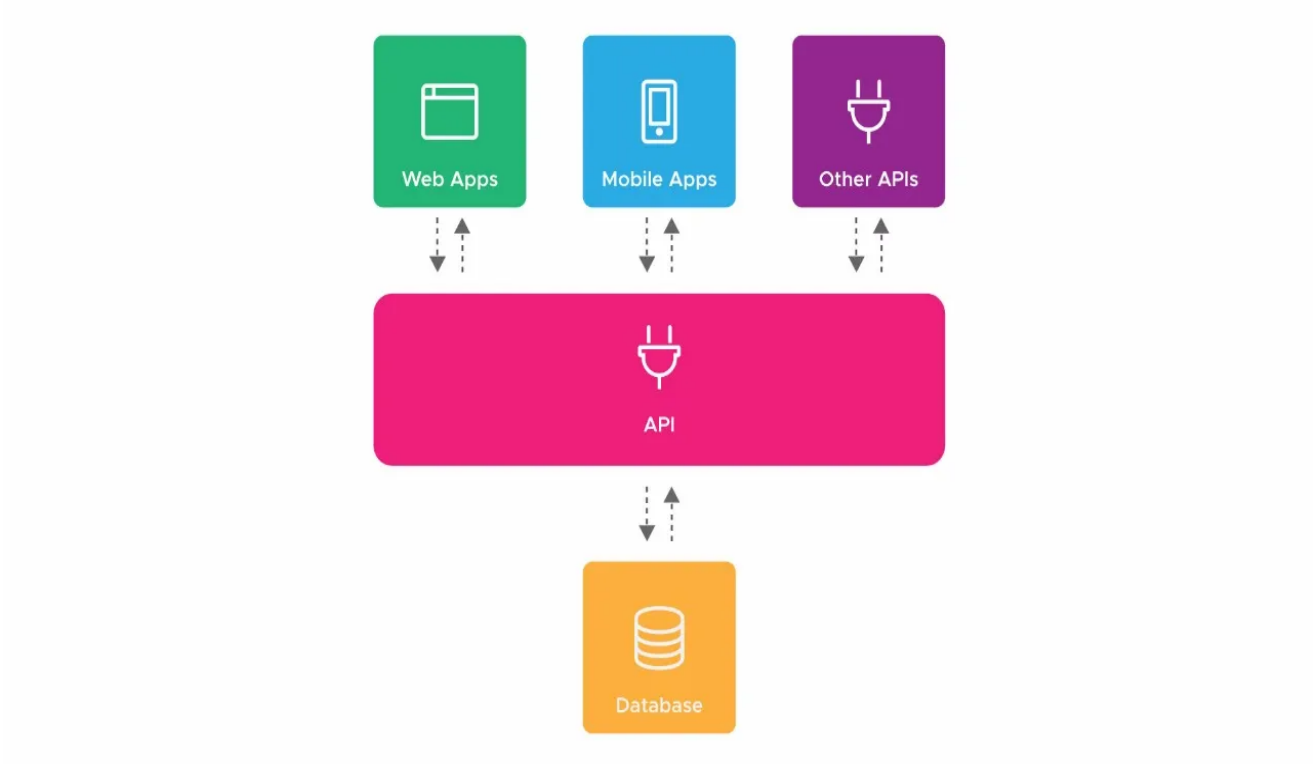
INTRODUÇÃO A API

Com o avanço da tecnologia e o aumento de sistemas em diversas áreas, surgiu o desafio da integração desses sistemas, que por sua vez foi tornando-se cada vez mais complexo.
API (Application Programming Interface) é uma forma que os sistemas utilizam para trocar informações entre si.

Uma maneira fácil de entender como funciona uma API é pensar nos conceitos de requisição (o conteúdo que você deseja) e de resposta (o conteúdo retornado à requisição).
Um exemplo típico é uma solicitação aos serviços do Correio para obtenção do endereço completo a partir do CEP. Nesse caso você envia um CEP específico para consulta e obtém resposta informando o endereço completo que corresponde ao CEP especificado. Este serviço é obtido através de uma chamada a um programa “API”, em outras palavras, as APIs ajudam na comunicação entre sistemas e atuam como intermediárias entre o que um

usuário ou cliente precisa e o que deveria receber. Quando falamos de APIs toda a comunicação dessa interface é feita via web, ou seja, tudo é feito através de uma requisição através do protocolo HTTP a uma URL (endereço de uma API), que por sua vez, envia uma resposta.

Figura 1: Funcionamento de uma API



Fonte: Lima, Thiago: **A anatomia de uma API.** https://thiagolima.blog.br/a-anatomia-de-uma-api-restful-80df2aca158e. acesso em 2023-10-20.

EXEMPLIFICANDO API COM FUNCIONAMENTO DE UM RESTAURANTE

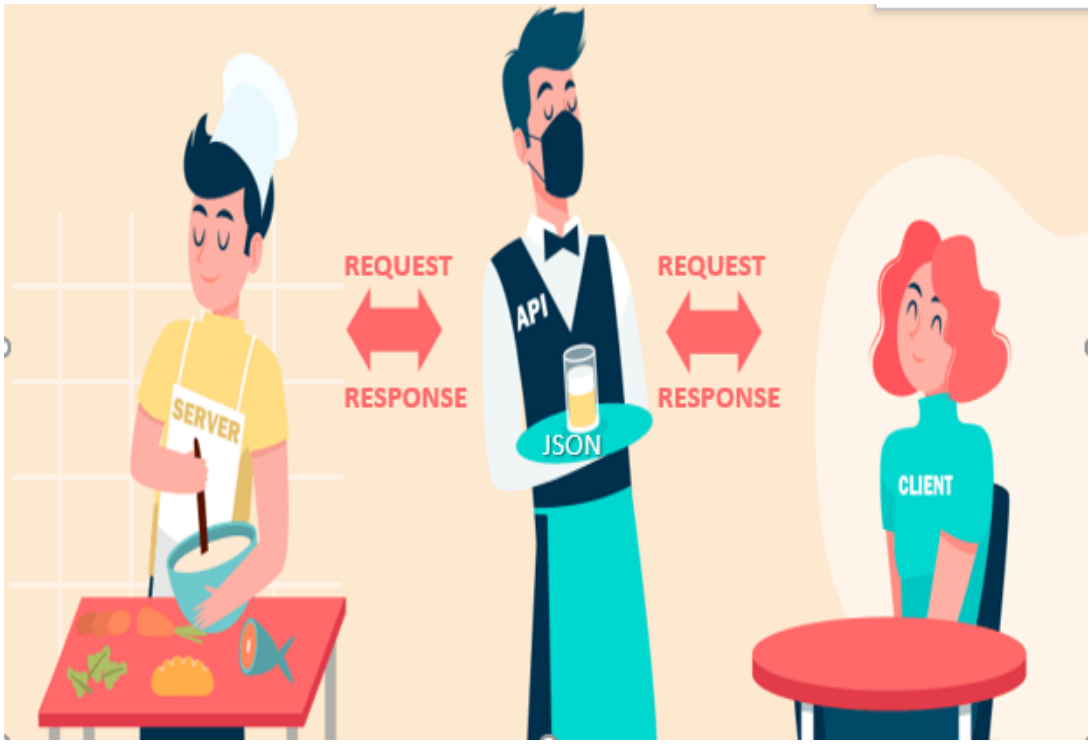
Imagine que você, como cliente, deseja fazer um pedido de comida. O garçom (API) através do cardápio, fornece uma lista de pratos (que são as funções disponíveis).

Daí o cliente solicita o pedido que deseja (request), e tem como retorno da cozinha o prato pronto solicitado (response).

Assim como você não precisa entender o processo de cozimento para desfrutar de uma refeição, os desenvolvedores usam APIs para acessar dados de um software sem precisar entender todo o seu funcionamento interno. É como um canal de comunicação padronizado que facilita a interação entre diferentes sistemas.

No mesmo exemplo do restaurante, REST é como se fosse o garçom eficiente. Quando você pede algo, ele segue regras para trazer exatamente o que você solicitou, sem excessos ou informações desnecessárias. Da mesma forma, o REST é um conjunto de regras para que diferentes sistemas de computadores possam "conversar" entre si na internet de maneira organizada e eficiente.

Figura 2: Exemplificando API com funcionamento de um restaurante



Fonte: Próprio autor

2 INTRODUÇÃO AO REST

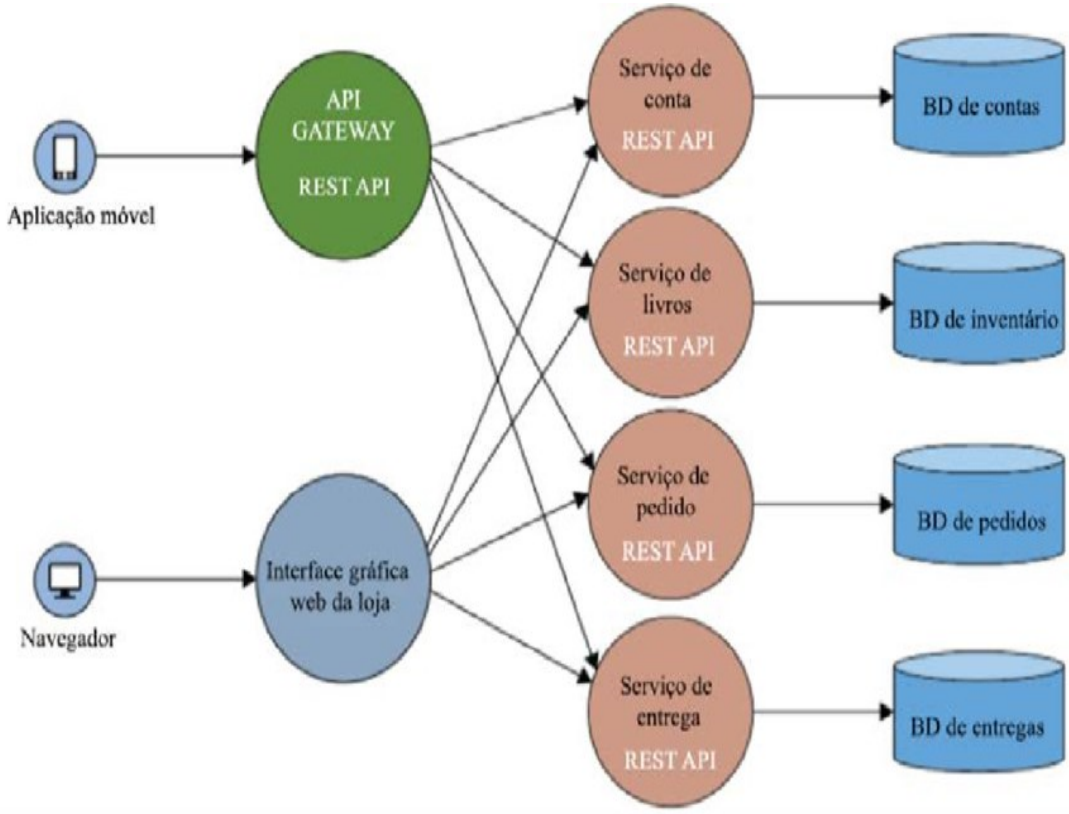
O conceito de REST foi desenvolvido pelo americano Roy Fielding. Roy é também um dos responsáveis pela especificação técnica do protocolo HTTP.

A ideia do REST é utilizar as propriedades do protocolo HTTP de forma mais eficaz e completa, principalmente em termos de semântica. Além do uso mais preciso de protocolos, o resultado é, em última análise, mais eficiência e, portanto, transferência de informações mais rápida.

O REST API facilita para que os consumidores de serviços com o tempo, deixem de ter que conhecer minunciosamente e implementar as particularidades de integração dos outros sistemas e passem a conhecer apenas REST APIs, pois a camada de integração com toda a sua ampla capacidade de conexão, transformação e aplicação de padrões de integração simplificaria todo esse processo.

Em Biehl (2007) [1], se afirma que REST não é uma arquitetura, mas sim uma forma de julgar arquiteturas, onde inclusive se compara com termo “orientado a objeto”. Esta afirmação reforça a falta de regras rígidas de design de uma arquitetura considerada REST.

Figura 3: Design do modelo REST



Fonte: AMARAL, Tatiana. Arquitetura de micros serviços com API REST . https://www.researchgate.net/figure/Figura-1-Arquitetura-de-microsservicos-com-API-REST_fig1_365387501, acesso em 2023-10-19

2.1 DEFINIÇÃO E CARACTERÍSTICAS DO RESTFULL

REST (Representational State Transfer) não é um protocolo ou padrão, mas sim algumas restrições e práticas de arquitetura.
A escolha de criar uma API REST oferece muitos benefícios, incluindo:

-Separação de cliente e servidor;
-Acessibilidade facilitadas aos contratos da aplicação;
-Aplicativo confiável e seguro;
-Escalabilidade;
-Multiplataforma.

Fielding foi o pioneiro a falar sobre Representational State Transfer (REST) em sua tese de Doutorado, ele propõe arquiteturas de software para criar interfaces com maior escalabilidade. Segundo Fielding (2000, p.76) [6], este tipo de arquitetura não é um padrão e sim um estilo híbrido de outras arquiteturas de rede que são combinadas com algumas restrições adicionais definindo uma interface de conexão única.
No capítulo 5 de sua tese, cita seis restrições que governam a escalabilidade da Web e consequentemente compõem o REST, um modelo de design de arquiteturas. Tais restrições foram separadas em 6 grupos: 1. Client-server; 2. Uniform Interface; 3. Layered System; 4. Cache; 5. Stateless; 6. Code-on-demand;

2.2 ARQUITETURA REST

Características da arquitetura REST são:

CLIENTE/SERVIDOR

Separa a responsabilidade da interface do usuário do banco de dados e abstrai suas dependências do lado cliente/servidor, permitindo que esses componentes sejam desenvolvidos sem impactar uns aos outros ou violar contratos.

INTERFACE UNIFORME

Focada na interoperabilidade entre cliente e servidor, pois como cliente e servidor utilizam a mesma interface, é preciso definir algumas regras para realizar esta comunicação. Para conseguir isso, tem que se basear em alguns princípios, que são:

-Reconhecimento de recursos.

-Representação de recursos.

-Mensagens auto descritivas.

-Componente HATEOAS, que ajuda os clientes a consumirem o serviço sem a necessidade de conhecimento prévio profundo da API.

STATELESS

Cada solicitação cliente/servidor deve conter todas as informações necessárias para saber de onde vem a solicitação.

CACHE

O uso de cache em arquiteturas da web melhora a eficácia do sistema, por aceitar que algumas requisições nem sejam processadas pelo servidor. Fielding (2000) [5] diz que cada recurso será identificado como cacheável ou não cacheável. Um recurso presente na cache pode ser retornado diretamente pela cache para os clientes que fazem requisições parecidas, evitando interações entre os componentes do sistema.

CAMADAS

A separação de responsabilidades é um componente fundamental deste modelo arquitetônico. De acordo com os princípios de projeto e boas práticas de design, o ideal é construir camadas independentes que se auto gerenciam, cada camada desconhece as outras e quando ocorrem alterações as outras camadas não são afetadas.

CÓDIGO SOB DEMANDA

Essa característica é opcional e ela se resume na possibilidade de a aplicação pegar códigos executáveis, como o Java script por exemplo, e executar no lado do cliente para satisfazer alguma parte da sua aplicação. Como por exemplo, o cliente pode requisitar a sua API um widget para mostrar na tela.

Talvez seja semelhante a um cliente que pede ao garçom algum prato customizado, diferente do que normalmente o restaurante oferece.

3 PRINCÍPIOS DO DESIGNER RESTFULL

3.1 DEFINIÇÃO E MÉTODOS HTTP

HTTP (HyperText Transfer Protocol) é um protocolo de comunicação de dados para o mundo da Internet (Web). Ele promove e define padrões para comunicação de dados entre navegadores e servidores.

A forma como os recursos são adquiridos e transportados entre clientes e servidores, seguem o padrão request-response. Para exemplificar como o padrão funciona vamos supor que o recurso requerido por um cliente seja uma página Web presente em um site. Neste caso a comunicação seguiria da seguinte forma:

> “Quando um usuário requisita uma página Web (por exemplo, clica sobre um hiperlink), o browser envia ao servidor mensagens de requisição HTTP para os objetos da página. O servidor recebe as requisições e responde com mensagens de requisição HTTP que contêm os objetos” (KUROSE, 2006, p.69).

Esse protocolo deve realizar métodos para autenticar e transportar as informações de requisição e devolução dos dados solicitados, garantindo também vários tipos de dados em cada pacote.

A utilização do HTTP em sistemas com interface web, se encaixa muito bem com o tipo de dados JavaScript Object Notation (JSON), pois ele é mais legível e leve comparado aos outros, e permite facilmente o envio de objetos Javascript com definição de dicionários indexados por chave-valor.

A troca de informações via HTTP vem sendo o meio utilizado mais frequentemente nas APIs Rest, mostrando as entidades do negócio através de URLs e as interações (gravação, leitura, etc) se dão por verbos padrões do protocolo HTTP (POST, GET etc), também existem padrões sobre como comunicar e como usar os códigos de resposta do HTTP.

Os Métodos são as ações permitidas dentro da API, quanto ao código de status, é um valor de retorno padrão para a API. Os dois são características do protocolo HTTP.

O método informa o tipo de ação que está sendo realizada na requisição.
Dentre os principais métodos, temos:

Get (Buscar dados)
Post (Enviar dados)
Put e Patch (Atualizar dados)
Delete (Deletar dados)

Principais métodos utilizados para enviar ou obter dados de uma API:

3.2 MÉTODO GET

Este é o método usado com mais frequência, pois ele busca dados via API. Simplesmente você usa estrutura Query String, que nada mais é do que os parâmetros da requisição, e na consulta envia esses parâmetros e o servidor retorna os dados se for bem-sucedido. Se você não enviar uma Query String na URL, a API retornará todos os dados sem filtro.

Por exemplo se uma API que traz dados sobre indicação dos melhores restaurantes do estado de São Paulo e sua URL base é https://api.melhoresrestaurantes.com.
Podemos buscar os restaurantes com as melhores sobremesas de Americana, utilizando esses parâmetros? estado=saopaulo&cidade=americana e nossa URL ficaria assim:

https://api.melhoresrestaurantes.com/sobremesas?estado=saopaulo&cidade=americana

Como podem ver acima, por se tratar de parâmetros da URL você usa o (?) e caso queira utilizar mais de um parâmetro você utiliza o (&).

OBS: A Query String não é somente utilizada para filtros, ela pode ser utilizada como parâmetros de paginação, versionamento, ordenação, e muito mais.

Outro exemplo seria como desenvolvedor eu gostaria de consultar dados sobre vinhos tintos de até 100 reais, e então a requisição GET ficaria desta forma:

https://api.melhoresrestaurantes.com/vinhos?ate_preco=100&tipo=tinto

Lembrando que a API teria que fornecer os parâmetros da Query String (ate_preco e tipo) em sua documentação para utilização na requisição.

Ou imagine que já tem uma lista de vinhos baseada no pedido anterior e pretende ver mais detalhes de um vinho específico com o identificador único 18. Neste caso, faça a seguinte solicitação, onde 18 é o valor do parâmetro {id} no URL: https://api.melhoresrestaurantes.com/vinhos/18

Nesse caso, {id} é necessário, portanto não utiliza Query String, mas sim um parâmetro de URL.

3.3 MÉTODO POST

Supondo que faça uma busca e não encontre um vinho tinto que procura e deseja registrá-lo nesta API. Use o método POST para enviar esses novos dados do vinho no corpo da requisição da API, como podemos acompanhar abaixo:

POST https://api.melhoresrestaurantes.com/vinhos com o texto:
{
nome: "Los Haroldos Reserva 2020",
uva: "Malbec",
preço: 59,
país: “Argentina",
recomendado: "Massas",
tipo: "Tinto"
}
O seu novo registo de vinho está agora completo.

3.4 MÉTODOS PUT e PATCH

O método PUT é usado em cenários de criação e atualização de dados. Isto significa que se enviar um vinho e este já existe ele simplesmente será atualizado, caso contrário será criado um novo item.
Já o método PATCH é usado para atualizar parcialmente este registro. Então se por exemplo este vinho registado gera o identificador 18 e queremos apenas atualizar o seu preço. A requisição fica assim:

PATCH https://api.melhoresrestaurantes.com/vinhos/18 com o texto:

{"preço": 48}

3.5 MÉTODO DELETE

O método DELETE é responsável por excluir registros.
Então, para apagar por exemplo o vinho que acabou de atualizar, é só realizar a seguinte solicitação:

DELETE https://api.melhoresrestaurantes.com/vinhos/18

3.6 CÓDIGOS DE STATUS HTTP/HTTPS

Segundo Carvalho [7], todo acesso a uma página web retorna um código de status HTTP. O código padrão mais frequente é o 200, que indica sucesso.
Dessa forma, para que o navegador possa se comunicar com o servidor corretamente, existem os códigos de resposta HTTP que mostram o status da sua solicitação.
Os códigos retornam pelo servidor e são usados para notificar o desenvolvedor sobre o que ocorreu depois da solicitação.
Podemos perceber que dentro das categorias podem existir inúmeros tipos de códigos de resposta, mas vamos exemplificar apenas os mais utilizados, onde, segundo Hall (2010, p.179) [13] e Kurose (2006, p.75) [11] os mais comuns são divididos por tipo:

GRUPO 1

Grupo de status que dá respostas informativas, existem poucas e raramente são utilizadas.

GRUPO 2

Grupo de status que indica sucesso na requisição. Os mais recorrentes são:

200 OK – muito usado, ele indica que a requisição foi realizada com sucesso.

201 Created - significa que a requisição deu certo e que um registro foi criado. Esse código geralmente é usado em requisições do método POST.

GRUPO 3

Este grupo contém os códigos utilizados com a intenção de notificar o cliente sobre alterações na requisição e redirecionando para uma nova URL. Os mais conhecidos são:

301 Moved Permanently - Notifica que o recurso A agora é o recurso B, encaminhando o cliente automaticamente de A para B.

304 Not Modified - Retorna em cenários de cache, notificando o cliente que a resposta não foi modificada, e assim, ele pode utilizar a mesma versão em cache da resposta.

307 Temporary redirect - Redireciona de um endereço ao outro, mas apenas temporariamente, ou seja, não é permanente.

GRUPO 4

O grupo retorna erros do cliente, ou seja, se fizer uma requisição de maneira errada, o servidor retornar um erro desse grupo. Os mais conhecidos são:

400 Bad Request - Informa que o servidor não compreendeu sua requisição, porque deve ter feito uma sintaxe ou estrutura inválida.

401 Unauthorized – Esse código significa que há uma camada de segurança no recurso solicitado ao servidor, e que não foi utilizado as credenciais válidas nessa requisição. As credenciais podem ser usuário e senha, token, etc., depende a API que estiver utilizando.

403 Forbidden – Esse erro reconhece as credenciais do cliente, mas está avisando que ele não tem acesso ao recurso.

404 Not Found - Código que notifica quando o servidor não acha o recurso solicitado pelo cliente.

429 Too Many Requests – Esse é mais raro, significa que o cliente ultrapassou o limite de requisições.

GRUPO 5

De grande importância também, esse grupo relata os erros do servidor. Esses erros aparecem quando a requisição foi feita corretamente pelo lado do cliente, mas houve falha de processamento no servidor. Códigos mais recorrentes desse grupo são:

500 Internal Server Erro - Erro mais recorrente que informa que o servidor achou um cenário inesperado de erro e não soube tratar, por isso não retornou uma resposta na requisição do cliente.

503 Service Unavailable - Código avisa que o servidor está fora do ar, em manutenção ou sobrecarregado.

Os códigos citados acima foram apenas alguns dos mais utilizados, porém existem diversos outros para as APIs.

Figura 4: Códigos de status HTTP

Fonte: CARVALHO, Gustavo. **Status HTTP: Tudo sobre os 33 códigos**. Disponível em: https://www.homehost.com.br/blog/internet/status-http-o-que-sao-codigos-de-resposta/, acesso em: 2023-09-18.

3.7 REPRESENTAÇÃO E FORMATO DE DADOS

JSON (JavaScript Object Notation) não é um protocolo de transporte de informações sob o protocolo HTTP/HTTPS. Ele é apenas uma forma muito leve de representação e troca de informações. Seu papel é transferir informações de um lugar para outro. JSON pode ser usado para transportar informações entre mensagens HTTP.

O XML por exemplo, é outra forma de representação de dados, mas não tão utilizada quanto JSON, porque é mais pesada, e sua legibilidade é inferior, como podemos observar nos exemplos abaixo:

Exemplo de um XML:

```
<clientes>
   <cliente>
      <id>1</id>
      <nome>Larissa</nome>
      <idade>31</idade>
   </cliente>
</clientes>
```

O mesmo código em JSON ficaria:

```
"clientes" : [
   {
      "id" : 1,
      "nome" : "Larissa",
      "idade" : 31
   }
]
```

Note que o JSON é mais legível e simplificado do que o XML, além de utilizar muito menos caracteres e ocupar menos bytes dentro de um response, portanto, o download de um response que contenha dados no formato JSON será mais rápido do que no formato XML. Esse é um dos principais motivos para os desenvolvedores preferirem utilizar JSON do que XML para o transporte de informações.

Como é citado no site json.org [8], JSON é um formato de texto completamente independente de linguagem, mas usa convenções que são familiares aos programadores da família de linguagens C, incluindo C, C++, C#, Java, JavaScript, Perl, Python e muitos outros. Essas propriedades tornam o JSON uma linguagem ideal para intercâmbio de dados.

4 GOVERNANÇA DE API

Governança de APIs tem como objetivo elaborar estratégias que possam dar vantagens competitivas às ferramentas de TI da empresa, abrangendo ações, políticas, regras e processos que direcionam as ferramentas. Sugere-se que seja utilizada por toda a área de TI.

Através da governança podemos ter implementações de APIs que possibilite a mesma REST API poder ser utilizada por uma aplicação web, por um aplicativo mobile ou até por algum sistema de proposta diferente que necessite do conteúdo que ela produz. Portanto, cada REST API deve refletir o negócio que ela representa, com eficiência para os sistemas e principalmente seu reuso.

Segundo Richardson (2007, p.235), o versionamento é um dos grandes dilemas dos desenvolvedores APIs, porque as especificações em si atrapalham o seu bom uso.

A melhor maneira de solucionar esse problema é estabelecer técnicas de identificação da versão utilizada, pois assim o versionamento não afeta no uso do consumidor da API.

4.1 VERSIONAMENTO PELA URL

Um grande problema das APIs é quando se atualiza a versão para uma outra diferente da que era utilizada. Existem algumas técnicas de versionamento de APIs que são ótimas soluções para prevenir esse problema, podemos realizar o versionamento por URL de três formas diferentes, que são: subdomínio, path ou query string.

SUBDOMÍNIO

Optando pela opção do subdomínio, cite primeiramente a versão escolhida, assim como no modelo abaixo:

https://api-v1.livros/geografia

Nesse caso iniciou com o "-v1", que específica a versão, assim o desenvolvedor modifica somente o subdomínio na requisição.

PATH
No path, citamos a versão no final da URL:

https://api.livros/v1/geografia

Path é uma opção muito escolhida entre os desenvolvedores, porque proporciona melhor visualização da URL e facilita navegar entre outras versões da API, sendo mais "dev-friendly".

QUERY STRING
A definição de versão via query string já não é mais tão utilizada, pois assim como mostra o exemplo ela não fica muito legível:

https://api.livros/história?version=1.0

4.2 VERSIONAMENTO PELO HEADER

O header seleciona a versão via HTTP content-type. Veja no exemplo abaixo o header accept na requisição:

https://api.livros/história
Accept: application/história.livros.v2+json

Analisando as melhores especificações para versionamento, podemos concluir que essa seria a melhor forma para se utilizar.
Podemos também fazer uso de um header personalizado para especificar a versão desejada, assim como no exemplo que foi criado um nomeado por "api-version" para definir a versão desejada:

https://api.livros.com/história
api-version: 2

Esse modelo torna URL limpa, de fácil visualização e sem a necessidade de modificação, porém não é dev-friendly, já que a requisição requer muito mais atenção.

4.3 DEFINIÇÕES DE CONTROLE DE ACESSO E AUTORIZAÇÃO

Empresas que não conhecem bem o funcionamento de APIs, costumam questionar sobre a segurança no seu uso. As instituições bancárias de grande porte por exemplo, geralmente levam muito a sério sua segurança, e desde 2022 foram obrigadas a expor suas APIs ao mercado, o chamado Open Banking, que ocorreu por uma mudança na legislação.

Por ser uma questão de grande importância a segurança das APIs, abordaremos sobre controle de acesso e autorização.

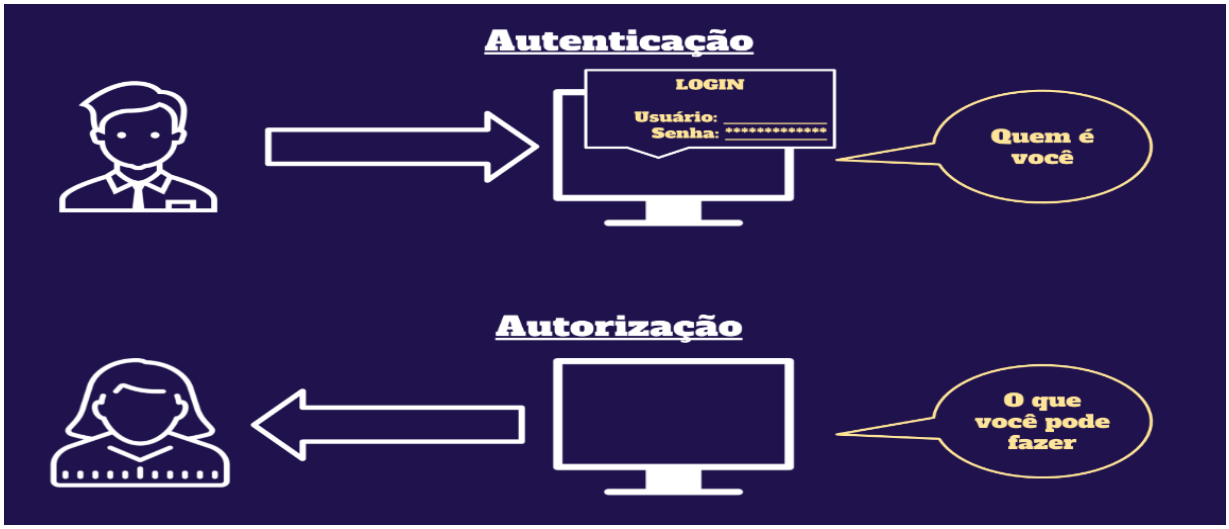
Conforme Muniz cita no livro Jornada API na prática [9], a autenticação e a autorização são partes importantes da segurança de uma API. A autenticação é o processo de verificar a identidade de um usuário, enquanto a autorização é o processo de verificar o que um usuário autenticado tem permissão para fazer. Existem várias maneiras de implementar a autenticação e a autorização em uma API, incluindo tokens, OAuth e JWT (JSON Web Tokens);

4.4 ESPECIFICANDO AUTENTICAÇÃO E AUTORIZAÇÃO

Autenticação é como se você fosse entrar na sua instituição de ensino, e o segurança exigisse suas credenciais como sua carteira de estudante para liberar sua entrada.

Autorização é como se você já estivesse dentro da instituição e quisesse acessar a sala de professores, mas para isso você precisaria ser autorizado pelo segurança.

Figura 5: Autenticação X Autorização



Fonte: SOUZA, Evandro. **Autenticação de usuários com OAuth 2.0 e OpenID Connect**. Disponível em: https://blogdagcom.wordpress.com/2019/09/23/autenticacao-e-autorizacao-de-microservices/, acesso em: 2023-08-28.

4.5 MÉTODOS DE AUTENTICAÇÃO MAIS CONHECIDOS

- BASIC AUTHENTICATION

Basic Authentication é descomplicado e realizado no protocolo HTTP, onde o cliente manda uma requisição com o header Authorization inserindo a palavra Basic, o nome de usuário e a senha, dividido por dois pontos no formato de base64.

Por exemplo, para autorizar o usuário Larissa com senha larissa@123, o cliente escreveria na requisição desta forma:

Authorization: Basic dGhpYWdvOnRoaWFnb0AxMjM=

O formato base64 é muito fácil de ser decodificado, a basic authentication só é indicada quando utiliza outras técnicas de segurança, como o certificado HTTPS.

Existem casos em que se troca usuário e senha por token, mas isso raramente acontece e não é aconselhável.

- API KEYS

No Basic Authentication havia algumas falhas e então o API Keys surgiu para aprimorar o modelo.
Ele funciona com o servidor criando uma chave de acesso única para o cliente, que é necessária em todas as requisições do API.
O modelo de API Keys fez sucesso e foi muito escolhido pelos desenvolvedores por muitos anos, porém ele funciona para autenticação, mas como o usuário com uma Key tem acesso a todas funções da API, podemos dizer que não é eficaz para autorização.
Outro problema nesse método é se a rede não estiver segura, o fato de a API Key realizar todo tipo de requisição HTTP, ela pode se tornar vulnerável a acessos indevidos.

Geralmente a API Key é realizada no header e com a chave recebida de exemplo "lari123", ficaria assim:
Api-key: lari123
Pode ser também via Query String, e não pelo header, mas a falha na segurança aumentaria, pois fica mais visível e suscetível a ataques.

- OAUTH

O OAuth é um método com várias técnicas de segurança, é um padrão projetado para permitir que um site ou aplicativo acesse recursos hospedados por outros aplicativos da web em nome de um usuário.
Ele está sendo considerado atualmente como o modelo mais eficaz para autenticação e autorização.
Algumas das especificações do OAuth são:
- Resource Owner (Proprietário do recurso): Geralmente é o usuário, pois é parte que comanda o acesso aos recursos.
- Resource Server (Servidor de recursos): servidor que contém os recursos e recebe solicitações.
- Authorization Server (Servidor de autorização): um servidor de autorização, gerador de tokens para usuários poderem acessar recursos autorizados pelo proprietário do mesmo.
- Client (Cliente): tem acesso aos recursos do resource server.
Um bom exemplo que podemos dar é que você desenvolveu uma aplicação que utiliza dados do usuário do Instagram, e o funcionamento básico de autenticação do OAuth 2.0 ficaria assim:
No site do usuário tem um botão de "integre ao instagram", portanto, o seu site seria o client.
Ao clicar, o usuário é encaminhado para a tela de login do Instagram (authorization server).
Ele preenche seus dados e o instagram fornece um código de acesso ao client.
Então o client solicita autorização aos recursos (endpoints da API do Instagram) para o resource owner (que é o próprio usuário) enviando o código de acesso recebido anteriormente.

O authorization server confere o código de acesso e se for verdadeiro ele gera um token de acesso para o client.
Assim que o client possuir o token e a autorização, a cada requisição, o resource server (API do Instagram) irá devolver os dados protegidos.
Um formato de token muito seguro que utiliza JSON como base é o JWT.

5 ESTUDO DE CASO API DOS CORREIOS

Os Correios [6] diz que os usuários que querem fazer a integralização, como sites de empresas de logísticas por exemplo, devem fazer o uso do Correios API, onde há uma lista de APIs desenvolvidas em REST e utilizam o protocolo HTTPS. Fornecendo assim uma URL base para cada API e com os verbos HTTP, como os que citamos anteriormente: GET, POST, PUT, DELETE, entre outros.
Essas APIs garantem os serviços de fornecimento dos seguintes dados:

- Dados de agências;
- Endereçamento;
- Disponibilidade de serviço;
- Dados de prazo de entrega;
- Dados de preço do serviço;
- Dados de rastreamento, e;
- Criação de pré-postagem.

Vale ressaltar que para realizar o uso do Correios API é preciso ter contrato com os Correios, pois os acessos as APIs estão relacionadas ao cadastro de APIs restritas no contrato/cartão de postagem.
O acesso as APIs, deve ser realizada pelo catálogo do Correios API. No catálogo o cliente poderá testar e acompanhar as mudanças quando houver.

Os Pré-requisitos para utilizar o Correios API são:

- Ter um cadastro válido no Meu Correios;
- Para acessar a API, utiliza-se uma senha definida para o componente, para criar a senha deve-se acessar o Correios API;
- Ter contrato ativo de prestação de serviços de encomendas ou mensagens com os Correios;
- Empresas que fornecem soluções em TI no segmento de logística.

Existem dois tipos de acesso as APIs dos Correios sendo:
- API Pública: onde todos enxergam a API;

- API Restrita: o acesso é realizado a partir do cadastro do serviço no cartão de postagem.

A API dos Correios é uma arquitetura orientada a micros serviços permitindo desenvolver sistemas que sejam mais flexíveis, escaláveis e com manutenção mais simples.

Figura 6: Micros serviços dos Correios API

Fonte: CORREIOS. **Manual de integração dos Correios API**. Disponível em: https://www.correios.com.br/atendimento/developers/arquivos/manual-para-integracao-correios-api, acesso em: 2023-10-24.

Como o volume de requisições poderá variar de 1 ou centenas de requisições, para obter uma resposta rápida do servidor dos Correios são adotas duas formas de requisição: a forma síncrona e a forma assíncrona.

A diferença entre síncrono e assíncrono:

- Uma requisição síncrona, deve aguardar a finalização do processamento e somente poderá realizar uma nova solicitação após a resposta da requisição, esta ação, pode dar a sensação de congelamento da requisição.

- Uma requisição assíncrona, o cliente envia uma requisição e recebe dados que permitem a consulta do status de processamento. Isso permite realizar novas requisições assíncronas, sem a necessidade de aguardar o fim do processamento das outras requisições. Ao fim poderá requisitar a coleta dos dados que foram processados.

API CEP

Neste exemplo, será utilizado o método que requer o valor para consultar o endereço de um CEP.
A url base é: 'https://apihom.correios.com.br/cep/ ' e que segundo a documentação será um GET : “/v1/endereços/{cep}”.

Para a requisição:

GET 'https://apihom.correios.com.br/cep/v1/enderecos/01001001'

Teremos a resposta:

{
 "cep": "01001001",
 "uf": "SP",
 "localidade": "São Paulo",
 "logradouro": "Praça da Sé",
"tipoLogradouro": "Praça",
 "nomeLogradouro": "da Sé",
 "complemento": "- lado par",
 "abreviatura": "Pç da Sé",
 "bairro": "Sé",
 "numeroLocalidade": 96681,
 "tipoCEP": 2,
 "cepUnidadeOperacional": "01032970",

"lado": "P",
"numeroInicial": 0,
"numeroFinal": 998
}

Em caso de exceção, há o retorno http 400 do servidor, tal como os seguintes exemplos:

EM CASO DE CEP INEXISTENTE:

Figura 7: Mensagem de erro cep inexistente

Fonte: CORREIOS. **Manual de integração dos Correios API**. Disponível em: https://www.correios.com.br/atendimento/developers/arquivos/manual-para-integracao-correios-api, acesso em: 2023-10-24.

EM CASO DE DIGITAR FORA DO PADRÃO:

Neste exemplo foi usado o CEP: 01001-001, com o hífen. Porém o retorno informa a quantidade de caracteres e o formato de exemplo.

Figura 8: Mensagem de erro fora do padrão

Fonte: CORREIOS. **Manual de integração dos Correios API**. Disponível em: https://www.correios.com.br/atendimento/developers/arquivos/manual-para-integracao-correios-api, acesso em: 2023-10-24.

API AGÊNCIA

A API Busca Agência tem a funcionalidade de mostrar as agências por localidade. Há pesquisa por unidades e localidades.

Figura 9: Busca de agências

Fonte: CORREIOS. **Manual de integração dos Correios API**. Disponível em: https://www.correios.com.br/atendimento/developers/arquivos/manual-para-integracao-correios-api, acesso em: 2023-10-24.

Para utilizar a consulta das unidades, é necessário recuperar os dados de tipos de unidades e status. Para a API Agência a base url:

https://apihom.correios.com.br/agencia

e para o exemplo será utilizado um GET: “/v1/unidades” com parâmetros.

Com o retorno:


```
{
 "itens": [
 {
 "id": "00024419",
 "codigoAntigo": "72300019",
 "codigoCadastroGeral": 16439,
 "nome": "AC CENTRAL DE SAO PAULO",
 "ativa": true,
 "status": "2",
 "descStatus": "INSTALADO",
 "tipo": "A",
 "tipoUnidade": {
 "codigo": "09",
 "descricao": "AGENCIA CORREIO",
 "sigla": "AC -TCO"
 },
 "emails": [
 "spmacacp@correios.com.br"
 ],
 "endereco": {
 "cep": "01031970",
 "uf": "SP",
 "localidade": "SAO PAULO",
 "municipio": "SAO PAULO",
 "logradouro": "PRACA DO CORREIO",
 "bairro": "CENTRO",
 "numero": "SN",
 "codigoIbge": "3550308",
 "regiao": "SDE",
 "longitude": "-46.636301",
 "latitude": "-23.544201",
 "fuso": "UTC-03:00",
 "fusoVerao": "UTC-03:00"
 },
 "horarios": {
 "funcionamento": "SEGUNDA À SEXTA",
 "iniExpediente": "09:00",
 "fimExpediente": "18:00",
 "limitePostagemSemana": "17:35"
 },
 "codigoSro": [
 "01009972"
 ],
 "dhAlteracao": "2021-03-16T01:01:22.27"
 },...],
 "page": {
 "size": 50,
 "totalElements": 7209,
 "totalPages": 145,
```

"number": 0 }
}

Nesta pesquisa de exemplo, o atributo size informa quantos elementos por página, foi colocado na requisição um size= 50 ou seja 50 elementos por página, totalizando 145 páginas. Caso a mesma consulta for realizada para informar o conteúdo das demais páginas, basta informar no atributo page o valor de 0..144, se mantiver o size=50.

Além desses dois exemplos de micros serviços das APIs dos Correios, todos os outros podem ser encontrados no manual dos Correios [6].

6 CONSIDERAÇÕES FINAIS

As APIs REST estão sendo utilizadas por praticamente todas as empresas que utilizam sistemas para integração e troca de informação, facilitando suas rotinas de diversas formas.
Os pontos positivos são diversos, incluindo esses citados abaixo:

AGILIDADE E INOVAÇÃO: As APIs REST possibilitam que as empresas sejam mais ágeis na implementação de novos recursos e funcionalidades. Elas permitem a rápida adaptação às mudanças do mercado, facilitando a introdução de inovações e atualizações contínuas.

EFICIÊNCIA OPERACIONAL: As APIs REST simplificam a integração de sistemas internos e externos, melhorando a eficiência operacional. Isso reduz a necessidade de tarefas manuais e processos demorados, economizando tempo e recursos.

EXPANSÃO DE MERCADO: Através das APIs REST, as empresas podem abrir seus serviços e dados para parceiros, clientes e desenvolvedores externos. Isso cria oportunidades para expandir mercados, desenvolver ecossistemas de aplicativos e alcançar novos públicos.

COMPETITIVIDADE: Empresas que adotam APIs REST eficazes podem se destacar no mercado ao oferecer serviços acessíveis, confiáveis e de fácil integração. Isso fortalece a competitividade e a posição no setor.

EXPERIÊNCIA DO CLIENTE: As APIs REST podem aprimorar a experiência do cliente, permitindo a personalização de serviços, integração com outros aplicativos e acesso simplificado a informações e funcionalidades.

ANÁLISE DE DADOS AVANÇADA: APIs REST facilitam a coleta e o compartilhamento de dados, permitindo análises avançadas que impulsionam a tomada de decisões baseadas em dados.

Porém, é de suma importância reparar que a implementação de APIs REST necessita de atenção para alguns pontos, como cuidado com segurança, gerenciamento e governança. Sabendo dessas questões, aconselha-se que essas empresas se utilizem de estratégias eficazes de governança de APIs para assegurar que elas usufruam de todos os benefícios dessas tecnologias com sucesso.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Biehl, Matthias: **RESTful Web Services**, volume 2007. O'Reilly Media, Inc.,1005 Gravenstein Highway North, Sebastopol, CA 95472, 2007.

[2] Fonte: AMARAL*,* Tatiana. **Arquitetura de micros serviços com API REST***.* https://www.researchgate.net/figure/Figura-1-Arquitetura-de-microsservicos-com-API-REST_fig1_365387501, acesso em 2023-10-19.

[3] Lima, Thiago: **A anatomia de uma API***.* Disponível em: https://thiagolima.blog.br/a-anatomia-de-uma-api-restful-80df2aca158e*.* acesso em 2023-10-20*.*

[4] Galhego Cardoso, Marcelo. **HTTP: O mínimo que todo desenvolvedor precisa saber.** *Ebook.* Ed 1, junho 2022.

[5] Fielding, Roy Thomas: **Architectural styles and the design of network-based software architectures**, 2000. Disponível em: https://ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf , acesso em 2023-09-15

[6] Correios. **Manual de integração das APIs dos Correios***, 2023.* Disponível em: https://www.correios.com.br/atendimento/developers/arquivos/manual-para-integracao-correios-api, acesso em 2023-10-24.

[7] CARVALHO, Gustavo. **Status HTTP: Tudo sobre os 33 códigos**. Disponível em: https://www.homehost.com.br/blog/internet/status-http-o-que-sao-codigos-de-resposta/, acesso em: 2023-09-18.

[8] **Introducing JSON***.* Disponível em: https://www.json.org/json-en.html. Acesso em: 2023-10-30

[9] MUNIZ, Antonio. et al. **Jornada API na prática**. 1.Ed. São Paulo: Brasport, 2023.

[10] SOUZA, Evandro. **Autenticação de usuários com OAuth 2.0 e OpenID Connect.** Disponível em: https://blogdagcom.wordpress.com/2019/09/23/autenticacao-e-autorizacao-de-microservices/, acesso em: 2023-08-28.

[11] KUROSE, Ross. **Redes de computadores e a Internet: Uma abordagem top-down.** 3 ed. – São Paulo : Pearson Addison Wesley, 2006

[12] RICHARDSON, L. RUBY, S. **RESTful Web Services**. 1. ed - Gravenstein Highway North, Sebastopol, CA – O'Reilly Media, Inc. Mai. 2007.

[13] HALL. M, BROWN. L. Core Servlet and JavaServer Pages. 2 ed. – Prentice Hall and Sun Microsystems.