Introdução a Arquitetura REST

Representational State Transfer, ou simplesmente REST, é um estilo de arquitetura para sistemas distribuídos com base em princípios da web, como HTTP, URI e formato de dados como JSON ou XML, para permitir a comunicação entre sistemas distintos de forma simples, escalável e padronizada. Essa arquitetura é amplamente utilizada para a criação de APIs (Application Programming Interface) e serviços web, permitindo que diferentes aplicativos se comuniquem e compartilhem recursos entre si. O REST enfatiza a interoperabilidade entre diferentes sistemas e a separação de preocupações entre servidor e cliente, tornando os sistemas mais flexíveis, fáceis de manter e evoluir.

A arquitetura REST foi criada pelo cientista da computação Roy Fielding durante seu doutorado pela Universidade da Califórnia. Diferentemente de web services SOA (Simple Object Access Protocol), que é um protocolo, REST não é um protocolo e sim uma arquitetura que faz uso do protocolo HTTP como solução para requisições e respostas. É importante destacar que API REST não é a única nomenclatura aceita, embora seja a mais utilizada, podemos ver referências citando API RESTful ou Web Service REST.

O que é API REST?

Uma API REST é um tipo de API baseado na arquitetura REST que permite a comunicação entre diferentes sistemas de softwares. Ela define um conjunto de operações (métodos HTTP) que podem ser usadas para criar, ler, atualizar e excluir recursos (dados) em um servidor web. Uma API REST usa URIs (Identificadores Uniformes de Recursos) para identificar os recursos e o formato JSON ou XML para representar dados transferidos entre os sistemas. Com uma API REST, diferentes aplicativos podem se comunicar e compartilhar recursos entre si de forma padronizada e escalável. Isso permite a criação de soluções de softwares mais flexíveis e integráveis.

– Clientes

Quando você acessa uma API como a do Youtube ou Google Maps, o acesso será realizado por meio de uma aplicação cliente. Essa aplicação pode ser o navegador web do seu laptop, um aplicativo em seu smartphone, tablet ou até mesmo smart tv. As APIs REST podem ser classificadas em três tipos com base em seu nível de acesso: privadas, públicas e híbridas.

API REST privada: é uma API que só pode ser acessada por aplicativos e sistemas específicos, geralmente dentro da mesma organização ou rede. Isso garante que somente aplicativos autorizados possam usar a API e acessar os recursos protegidos.

API REST pública: é uma API que pode ser acessada por qualquer aplicativo por meio da internet. Essas APIs são frequentemente usadas por desenvolvedores para integrar seus aplicativos com serviços populares de terceiros, como Google Maps, Facebook, Twitter, etc.

API REST híbrida: é uma combinação de API pública e privada, onde a API pode ser acessada por qualquer aplicativo, mas alguns recursos e funcionalidades são protegidos e só podem ser acessados por aplicativos autorizados.

A escolha de qual tipo de API REST usar depende do nível de segurança necessário e dos recursos que se deseja disponibilizar para os aplicativos clientes. As APIs privadas são geralmente usadas para proteger dados sensíveis e garantir a segurança dos aplicativos internos. As APIs públicas são usadas para disponibilizar serviços para o público em geral e para promover a integração de aplicativos de terceiros. As APIs híbridas são usadas quando é necessário equilibrar a segurança dos dados com a acessibilidade pública dos recursos.

Uma aplicação cliente poderá ser desenvolvida em qualquer linguagem de programação que faça uso do protocolo HTTP. Essa é uma vantagem para os desenvolvedores de aplicações clientes, já que não ficam obrigados a utilizar uma linguagem de programação que não estão habituados ou mesmo que não tenham o domínio necessário.

– O que são Recursos

Os recursos são os registros que as APIs REST fornecem a seus clientes. Eles podem ser de diversos tipos como textos, imagens, vídeos, etc. Os recursos são representados por URIs (Uniform Resource Identifiers), que são endereços únicos que identificam o recurso a ser acessado. Cada recurso pode ser acessado e manuseado usando um conjunto de métodos HTTP (como GET, POST, PUT, DELETE) que são definidas pela API. Os recursos são a base da arquitetura REST e permitem que os clientes manipulem os dados da API de forma padronizada e intuitiva.

– Documentação

A documentação de uma API REST é formada por um conjunto de informações que descrevem a funcionalidade, os recursos e os endpoints disponíveis na API. Ela é destinada aos desenvolvedores que desejam consumir a API por meio de aplicações clientes e fornece detalhes sobre como acessar e usar seus recursos.

A documentação da API REST também pode incluir exemplos de código que mostram como usar a API em diferentes linguagens de programação e plataformas, bem como informações de autenticação e segurança, limites de taxa de solicitação e outras informações relevantes. Uma boa documentação da API REST é importante para garantir que os desenvolvedores de aplicações cliente possam integrar facilmente a API em seus aplicativos, entender como usar seus recursos e resolver problemas quando eles surgem.

Benefícios de APIs REST

O modelo arquitetural REST se tornou preferível em relação ao SOAP com o passar dos anos. Isso ocorreu por várias razões, algumas delas citadas a seguir:

Escalabilidade: As APIs REST são altamente escaláveis e podem lidar com grandes volumes de solicitações sem comprometer o desempenho ou a disponibilidade. Isso é possível devido ao fato da arquitetura REST ser Stateless, ou seja, não é mantido um estado ou contexto entre a aplicação cliente e o servidor.

Flexibilidade: As APIs REST permitem que os desenvolvedores criem soluções de software flexíveis e modulares, que podem ser facilmente adaptadas para atender às necessidades em constante mudança dos usuários.

Padronização: As APIs REST são baseadas em padrões abertos e amplamente adotados, como HTTP, URIs e JSON ou XML, o que torna mais fácil para os desenvolvedores integrar diferentes sistemas de software.

Reutilização: As APIs REST permitem que os recursos e serviços sejam compartilhados e reutilizados em diferentes aplicativos, o que pode economizar tempo e recursos de desenvolvimento.

Integração: As APIs REST permitem a integração fácil de diferentes aplicativos e sistemas de software, permitindo que as organizações compartilhem dados e informações entre si.

Acessibilidade: As APIs REST podem ser acessadas por diferentes plataformas e dispositivos, incluindo navegadores da web, aplicativos móveis e dispositivos IoT, o que amplia o alcance da solução de software.

Segurança: As APIs REST oferecem recursos de segurança, como autenticação e autorização, para garantir que apenas usuários autorizados possam acessar os recursos protegidos.

Esses benefícios tornam as APIs REST uma solução popular e eficiente para a construção de aplicativos e sistemas de software modernos e escaláveis.

Segurança em APIs REST

Existem diferentes tipos de segurança que podem vir a ser utilizados em APIs REST e assim, proteger os dados e garantir a integridade das solicitações e respostas. Alguns dos tipos de segurança mais aplicados em API REST são:

Autenticação: verificar a identidade do usuário que está solicitando acesso à API. Os métodos de autenticação incluem senhas, tokens de acesso, certificados digitais e autenticação de dois fatores.

Autorização: verificar se o usuário tem permissão para acessar o recurso solicitado na API. Isso é geralmente feito com base em regras de acesso predefinidas, como as funções do usuário.

Criptografia: protege os dados em trânsito, tornando-os inelegíveis para qualquer pessoa que possa interpretá-los. A criptografia pode ser aplicada a diferentes níveis, como TLS/SSL no nível de transporte ou criptografia ponta a ponta para dados sensíveis.

Monitoramento de atividades: rastreia as atividades do usuário na API para detectar comportamentos suspeitos e garantir a conformidade com as políticas de segurança. Isso pode incluir registros de auditoria, monitoramento de eventos e alertas de segurança.

Gerenciamento de tokens: protege os tokens de acesso usados na autenticação, garantindo que apenas usuários autorizados possam acessar os recursos da API. Isso pode incluir a validação de tokens, a expiração de tokens e a revogação de tokens em caso de comprometimento de segurança.

Os diferentes tipos de segurança são de grande valia para garantir que as APIs REST se mantenham seguras e protejam os dados confidenciais dos usuários e das organizações.

API REST - Fundamentos

API REST - Fundamentos

A arquitetura REST trabalha sobre o protocolo HTTP e cada URI funciona como um recurso da API. Portanto, devemos usar substantivos para determinar recursos em vez de verbos.

URI - Uniform Resource Identifier, ou Identificador Uniforme de Recursos

Exemplo: https://mballem.com/cursos/

A URI une o Protocolo (https://);

A localização do recurso (URL - mballem.com);

E o nome do recurso (URN - /cursos/)

Endpoints estilo RPC usam verbos, por exemplo, api/v1/getPersons. Em contrapartida, em REST, esse endpoint deve ser escrito como api/v1/persons. Então, você deve estar se perguntando como poderemos diferenciar as ações executadas em um recurso REST? Pois bem, é agora que faremos uso dos chamados Métodos HTTP (HTTP METHOD). A ideia é fazer com que os métodos HTTP trabalhem como um verbo, por exemplo, GET (recuperar), DELETE (remover), POST (criar), PUT (modificar) e PATCH (atualização parcial), entre outros.

Endpoint

O endpoint REST é um URI exclusivo que representa um recurso REST.

Por exemplo, http://app/api/v1/persons é um endpoint REST. Além disso, /api/v1/persons é o caminho do endpoint e as pessoas são recurso REST.

Métodos HTTP

O protocolo HTTP define um conjunto de métodos ou verbos de requisição responsáveis por indicar a ação a ser executada para um dado recurso. Em APIs REST alguns desses verbos são mais comuns que outros de serem utilizados. Isso acontece não por preferência, mas por conta dos tipos de ações sobre os recursos. Por exemplo, o método GET é utilizado para operações que recuperam um recurso e provavelmente uma API REST terá muito mais ações do tipo GET do que do tipo DELETE (usado para exclusão de recursos).

Um conceito muito falado quando começamos a estudar REST é quais métodos são ou não são idempotentes. Na verdade, esse não seria um conceito específico da arquitetura REST, mas sim, do protocolo HTTP. Porém, por algum motivo, sempre foi muito citado em estudos sobre REST.

Um método HTTP idempotente é um método HTTP que pode ser chamado várias vezes sem alterar os resultados. Não importa se o método é chamado apenas uma vez ou dezenas de vezes. O resultado deve ser o mesmo. Importante destacar, idempotência se aplica ao resultado, não ao recurso. Entretanto, esse assunto pode gerar controvérsias, assim como vemos na postagem O que é ser idempotente em REST? O debate continua do site Infoq.

GET

O GET é usado para recuperar um recurso do servidor. É uma operação segura e idempotente, ou seja, várias solicitações para o mesmo recurso retornarão o mesmo resultado e não modificará o estado do recurso no servidor.

Alguns casos de uso do GET:

Recuperar uma página da web

Buscar a lista de postagens de um blog

Download de arquivo de um servidor

POST

POST é usado para criar um novo recurso no servidor. Não é considerado um método idempotente, o que significa que várias solicitações para criar o mesmo recurso resultará em vários recursos sendo criados. As solicitações POST também podem modificar o estado do recurso no servidor.

Alguns casos de uso do POST:

Criação de uma nova conta de usuário

Carregar uma nova imagem ou arquivo

Enviar um formulário a um servidor para criar um novo recurso

PUT

PUT é usado para atualizar um recurso existente no servidor. É considerado idempotente, o que significa que várias solicitações para atualizar o mesmo recurso resultarão em o mesmo estado do recurso no servidor. O PUT também pode criar um novo recurso se o recurso que está sendo atualizado não existe.

Casos de uso para PUT:

Atualizar o conteúdo de uma postagem de blog

Modificar os detalhes de uma conta de usuário

Substituir um arquivo ou imagem em um servidor por um nova versão

Atualizar todo o endereço de um contato

PATCH

O método PATCH é usado para fazer uma atualização parcial em um recurso existente no servidor. O objetivo é realizar atualizações em campos ou propriedades específicas de um recurso, sem exigir que todo o recurso seja enviado ao servidor.

Casos em que o PATCH é usado:

Atualizar o título ou descrição de uma postagem de blog

Alterar a senha ou endereço de e-mail associado a uma conta de usuário

Fazer uma atualização parcial de um arquivo ou imagem

DELETE

O verbo DELETE é usado para excluir um recurso do servidor. É considerado idempotente, o que significa que várias solicitações para excluir o mesmo recurso resultarão no mesmo estado do servidor. Isso parece estranho e gera muitas dúvidas. Mas na verdade, ao executar a primeira solicitações de DELETE para o recurso, ele será excluído e a resposta será 200 (OK) ou 204 (No Content). As próximas solicitações para o mesmo recurso (já excluído) retornarão 404 (não encontrado). Claramente, a resposta é diferente da primeira solicitação, mas não há mudança de estado para nenhum recurso do lado do servidor porque o recurso original já foi excluído.

Usar casos para DELETE incluem:

Remover uma postagem de blog

Excluir uma conta de usuário

Remover um arquivo ou imagem

Podemos destacar que uma operação de exclusão pode ser física ou lógica. A física é quando o recurso é realmente excluído da base de dados e a lógica quando uma coluna na tabela do recurso indica que ele foi excluído, mesmo ainda estando presente na tabela. A exclusão lógica é muito usada para manter dados históricos.

Demais métodos HTTP

O protocolo HTTP tem mais alguns métodos além dos citados. Para algumas APIs eles poderão vir a ser úteis, mas talvez para outras, nem tanto. Portanto, nossa abordagem será sobre o GET, POST, PUT, PATCH e DELETE. Caso queira conhecer outros métodos, como HEAD, OPTIONS, TRACE, etc, acesse - Métodos de requisição HTTP.

Status HTTP

Toda vez que uma solicitação HTTP é enviada a uma API REST, algum tipo de resposta será retornada de volta ao solicitante. Independentemente se a solicitação foi aceita, gerou erro, se o recurso não foi encontrado ou se teve sucesso na execução.

Essa resposta é chamada de HTTP Status Code e existem cinco categorias de Status Code no protocolo HTTP.

Informational responses (100 – 199)

Successful responses (200 – 299)

Redirects (300 – 399)

Client errors (400 – 499)

Server errors (500 – 599)

Alguns dos mais comuns códigos em respostas REST

200 - OK - Padrão de resposta para solicitações HTTP sucesso. A resposta real dependerá do método de solicitação usado. Em uma solicitação GET, a resposta conterá uma entidade que corresponde ao recurso solicitado. Em uma solicitação POST a resposta conterá a descrição de uma entidade, ou contendo o resultado da ação.

201 - Create - O pedido foi cumprido e resultou em um novo recurso que está sendo criado. Usado em solicitação POST e PUT quando criar um novo recurso.

204 - No Content - O servidor processou a solicitação com sucesso, mas não é necessária nenhuma resposta. Resposta comum na solicitação de um DELETE.

400 - Bad Request - O pedido não pôde ser entregue devido à sintaxe incorreta.

401 - Unauthorized - Basicamente usado quando a solicitação exige que o cliente esteja autenticado, entretanto, o cliente ainda não realizou a autenticação. A resposta deve incluir um cabeçalho do campo www-authenticate contendo um desafio aplicável ao recurso solicitado.

403 - Forbidden - O pedido é reconhecido pelo servidor, mas este recusa-se a executá-lo. Ao contrário da resposta “401”, aqui a autenticação não fará diferença e o pedido não deve ser requisitado novamente.

404 - Not Found - O recurso solicitado não foi encontrado, mas pode ser disponibilizado novamente no futuro. As solicitações subsequentes pelo cliente são permitidas.

405 - Method Not Allowed - Foi feita uma solicitação de um recurso usando um método de pedido que não é compatível com esse recurso, por exemplo, usando GET em um formulário, que exige que os dados sejam apresentados via POST, PUT; ou usados em um recurso somente de leitura.

409 - Conflict - Indica que a solicitação não pôde ser processada por causa do conflito no pedido, como um conflito de edição.

422 - Unprocessable Entity - O pedido foi bem formatado, mas foi incapaz de ser seguido devido a erros de semântica.

500 - Internal Server Error - Indica um erro do servidor ao processar a solicitação.

501 - Not Implemented - O servidor ainda não suporta a funcionalidade ativada.

503 - Service Unavailable - O servidor está em manutenção ou não consegue dar conta dos processamentos de recursos devido à sobrecarga do sistema. Isto deve ser uma condição temporária.

É muito importante que todas as respostas de solicitações possuam um Status Code definido para caso da solicitação ter sido processada ou ter gerado algum tipo de erro. O Status Code é a forma como a aplicação cliente vai saber tratar a resposta da requisição. Por exemplo, se a resposta for 200 o cliente poderá listar os dados recebidos, mas se for 403 poderá informar ao usuário que ele não tem permissão de acesso àquele recurso.

Para saber mais saber mais sobre status code do HTTP acesse - [HTTP response status codes](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status)

A arquitetura REST trabalha sobre o protocolo HTTP e cada URI funciona como um recurso da API. Portanto, devemos usar substantivos para determinar recursos em vez de verbos.

URI - Uniform Resource Identifier, ou Identificador Uniforme de Recursos

Exemplo: https://mballem.com/cursos/

A URI une o Protocolo (https://);

A localização do recurso (URL - mballem.com);

E o nome do recurso (URN - /cursos/)

correção no metodo post para patch???