UniFil
EDUCAÇÃO A DISTÂNCIA

Unidade 2

Arquitetura cliente/servidor II

Introdução

Com a existência de uma grande demanda por desenvolvimento de softwares web nas empresas, é necessário compreender como melhor desenvolver na web. Seguindo essa visão, nesta unidade vamos conhecer e discutir a arquitetura cliente/servidor e alguns conceitos dentro desse modelo de arquitetura.

Na aula 1, vamos estudar os conceitos e conhecer quais são os elementos que compõem esse modelo de arquitetura para desenvolver sistemas web.

Na aula 2, vamos entrar em detalhes sobre os verbos HTTP, o funcionamento deles e como é possível simular uma estrutura cliente/servidor utilizando softwares disponibilizados no mercado.

Objetivos

- Conhecer os conceitos da arquitetura cliente/servidor.
- Conhecer boas práticas de desenvolvimento web na utilização de verbos HTTP.

Conteúdo programático

Aula 01 — Arquitetura cliente/servidor

Aula 02 — Boas práticas e verbos HTTP

Quer assistir às videoaulas em seu celular? Basta apontar a câmera para os **QR Codes** distribuídos neste conteúdo.

Caso necessário, instale um aplicativo de leitura QR Code no celular e efetue o login na sua conta Gmail.

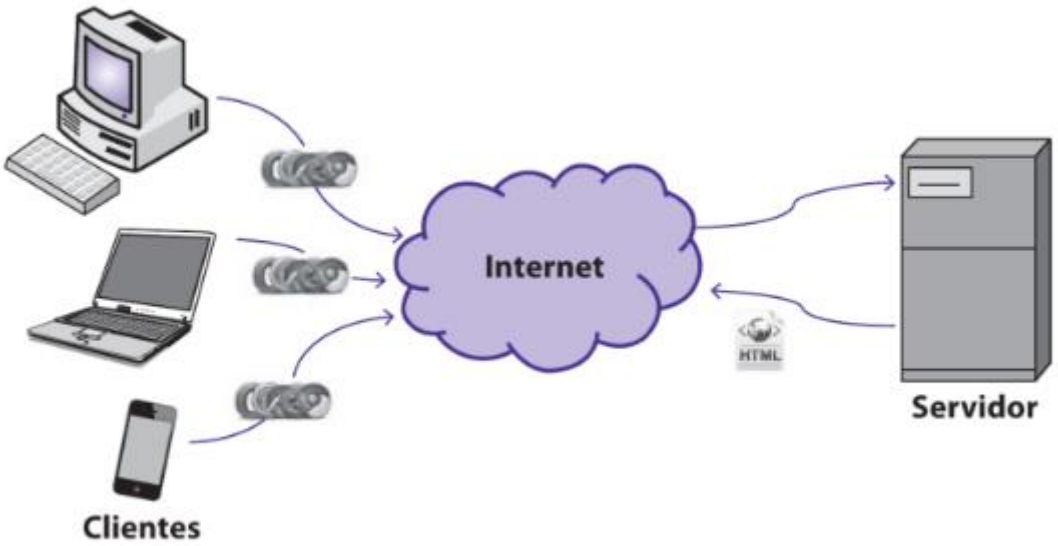
Aula 01 — Arquitetura cliente/servidor

Olá, estudante, como vai? Vamos começar agora a avançar ainda mais no modelo cliente/servidor e em seu funcionamento. Compreender quais são as formas e tratativas dessa arquitetura no funcionamento de um sistema web é essencial para o desenvolvimento correto de softwares web.

A boa aplicação e construção de um serviço web é ótimo para que se possua um sistema consistente e com qualidade; dessa forma, sendo longevo e lucrativo para o cliente, além de satisfazer as necessidades dos usuários.

Antes disso, vale relembrar como é o funcionamento da arquitetura cliente/servidor.

Figura 1 — Estrutura cliente/servidor



Fonte: Miletto e Bertagnolli (2014).

Na figura 1 temos o relato descritivo da estrutura cliente/servidor para sistemas web, em que os dispositivos são os clientes e se comunicam pela internet, utilizando o protocolo HTTP com o servidor que está alocado em um outro local, podendo ser este servidor físico ou não.

Videoaula 1

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

API (Application Programming Interface)

O que é uma API, que em português significa Interface de Programação de Aplicação? Neste caso, a aplicação é qualquer sistema que possua uma função distinta. A API é um mecanismo de comunicação entre dois ou mais componentes de sistema dentro de um projeto web, que utiliza protocolos e definições para que os componentes se comuniquem.

O funcionamento de APIs normalmente está relacionado a uma estrutura cliente/servidor em que a comunicação entre esses dois componentes precisa ser precisa e estabelecida da forma correta.

APIs podem possuir diversos fins e serem criadas de acordo com cada responsabilidade, porém existem quatro grandes grupos de APIs dentro do mercado de tecnologia atualmente:

a) API pública: são APIs abertas a qualquer tipo de público e podem ser encontradas de modo gratuito na internet.
b) API privada: são APIs internas que podem conectar diversas aplicações, normalmente são proprietárias ou possuem custo para utilização.
c) API de parceiros: esse grupo são APIs que possibilitam a comunicação entre uma API privada e uma outra API privada ou pública para utilização de recursos diversos.
d) API composta: são APIs que combinam dois ou mais grupos de APIs para atender a uma necessidade do usuário/cliente.

Uma API pode funcionar de quatro maneiras distintas, tudo depende da demanda do cliente e em qual estrutura web essa API vai ficar alocada para que funcione com qualidade.

APIs SOAP

São APIs que utilizam, na sua comunicação, o SOAP (*Simple Object Access Protocol*); desse modo, o cliente e o servidor trocam mensagens utilizando o XML, modelo que foi amplamente utilizado no passado, porém muito aderido ainda em APIs públicas governamentais por serem de difícil modificação.

APIs RPC

Nesse formato, uma API se comunica de modo a indicar sempre ao servidor qual procedimento executar com os dados, utilizando o RPC (*Remote Procedure Calls*), de modo que o cliente conclui uma operação ou função no servidor e este envia uma saída com as informações necessárias para o cliente.

APIs WebSocket

É um modelo de funcionamento de APIs bem moderno que já utiliza mensagens no formato JSON, oferecendo um formato de comunicação bidirecional entre os dispositivos clientes e o servidor, tornando mais rápida e eficaz a comunicação entre os componentes cliente e servidores.

APIs REST

São o modelo de funcionamento mais flexível e amplamente utilizado no desenvolvimento de aplicações web, o cliente envia para o servidor dados que são processados e tratados da forma correta, e o servidor retorna ao cliente a solicitação de sucesso ou erro.

Existem algumas vantagens na utilização de APIs REST dentro do aspecto de desenvolvimento web, são elas:

a) Integração: a facilidade na integração com outros sistemas da organização que já utilizam do protocolo HTTP para se comunicar ou até mesmo de outros recursos como filas de comunicação.
b) Inovação: com o mercado em mudança diariamente, possuir sistemas web que dão suporte a alterações rápidas e eficazes é superimportante.
c) Expansão: a possibilidade de expandir o funcionamento do sistema web de forma rápida, podendo escalar a aplicação com facilidade, podendo se comunicar com diversas plataformas.

d) Facilidade de manutenção: como uma API REST funciona de acordo com uma fachada de comunicação com outros sistemas, pode ser alterada de modo simples, sem muita dor de cabeça, tornando o sistema web mais longevo.

Por este modelo ser o mais utilizado durante o desenvolvimento de aplicações web, vamos nos aprofundar no seu funcionamento.

Videoaula 2

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

REST

Quando se fala de arquitetura cliente/servidor, é preciso considerar o funcionamento REST (*Representational State Transfer*; em português, Transferência de Estado Representacional), isto é, como são representados os dados dentro desse modelo de comunicação.

O REST não é um protocolo, mas sim um conjunto de definições para construir uma estrutura bem definida de dados, auxiliando na forma em que esses dados irão trafegar dentro do modelo cliente/servidor.

E como representar esses dados para que a comunicação seja efetiva entre o cliente e o servidor? Quando se fala dessa forma, espera-se que os dois se comuniquem de uma maneira que um entenda o outro.

Como no conceito básico de comunicação, existe um transmissor e um receptor e os dois precisam conversar na mesma língua, ou irão precisar de um tradutor. Enfim, o REST é esse conjunto de definições que torna possível que os dois participantes dessa comunicação (cliente e servidor) falem a mesma “língua”.

Esse modelo de “língua” pode ser de diversos formatos:

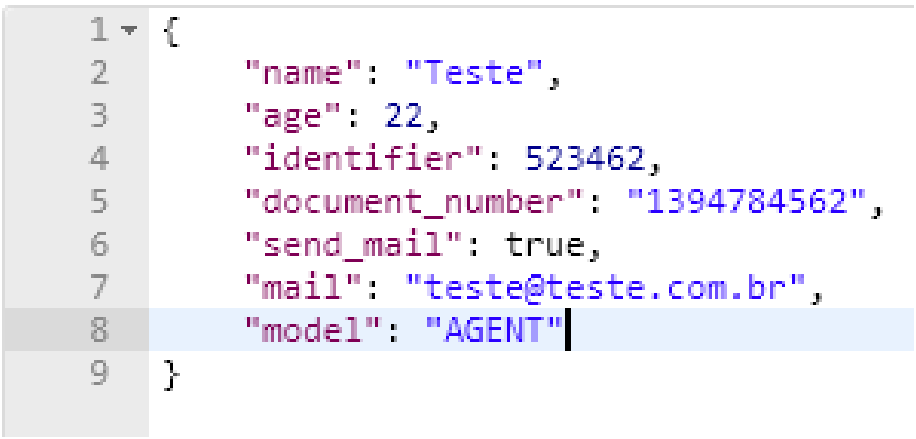
a) Texto sem formatação
b) HTML

c) SOAP

d) JSON

Existem também outros modelos, em sua maioria as APIs desenvolvidas atualmente utilizam o formato JSON (Javascript Object Notation), pois facilita a compreensão tanto da máquina quanto do próprio desenvolvedor.

Figura 2 — Exemplo JSON



Fonte: O autor (2022).

A figura 2 demonstra como é um formato JSON. Em uma requisição HTTP, esses dados são enviados e, para melhor compreendê-los, são chamados de *payload* (em português, carga útil).

Leva esse nome porque os dados enviados para o servidor dentro do payload devem ser úteis para que ele realize alguma operação dentro do servidor e não seja simplesmente uma informação descartável para o sistema.

A estrutura do JSON é composta por um modelo chamado de chave/valor.

Figura 3 — Exemplo JSON (chave/valor)

Fonte: O autor (2022).

Observando a figura 3, pode-se visualizar, dentro do JSON, a perspectiva de chave e valor, sendo a chave indicada pelo quadrado azul; e o valor, pelo quadrado vermelho.

Com isso, pode-se analisar que tanto a chave quanto o valor estão entre aspas, o que indica o início e final do JSON são as “chaves” ({}). A figura 3 relata o JSON de um objeto-pessoa, em que este *payload* é utilizado juntamente com o verbo POST do HTTP para salvar um novo usuário.

Videoaula 3

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

Aula 02 — Boas práticas e verbos HTTP

Olá, aluno, tudo bem? Nesta aula vamos conhecer melhor alguns conceitos de APIs REST e a comunicação de sistemas web dentro da estrutura cliente/servidor, juntamente com um laboratório prático para a visualização do funcionamento de uma API REST.

Durante a construção de um software, pode-se fazer necessário atribuir alguns recursos ao processamento que é necessário realizar.

Videoaula 1

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

REST e Verbos HTTP — Funcionamento

Com a utilização dos verbos HTTP, além de indicar o funcionamento e qual operação o cliente deseja realizar no servidor, é possível também documentar as operações que cada caminho da aplicação realiza.

Para conseguir descrever os exemplos e visualizar o funcionamento da arquitetura cliente/servidor nas práticas, vamos precisar utilizar o software Postman. Durante a realização deste laboratório didático, será utilizado o Postman somente para demonstrar o funcionamento dos verbos HTTP em requisições dentro da arquitetura cliente/servidor e suas características.

Indicação de Leitura

What is Postman? Disponível em: https://www.postman.com/product/what-is-postman/. Acesso em: 27 set. 2022.

Para a demonstração a seguir, foi utilizado o projeto: https://github.com/MarcusVMaziero/ecommerceSystem, que se encontra em domínio público no Github e é de autoria de autor Marcus V. Maziero. O projeto é uma API REST

desenvolvida na linguagem de programação Java, utilizando o framework Spring Boot, com um banco de dados em memória H2.

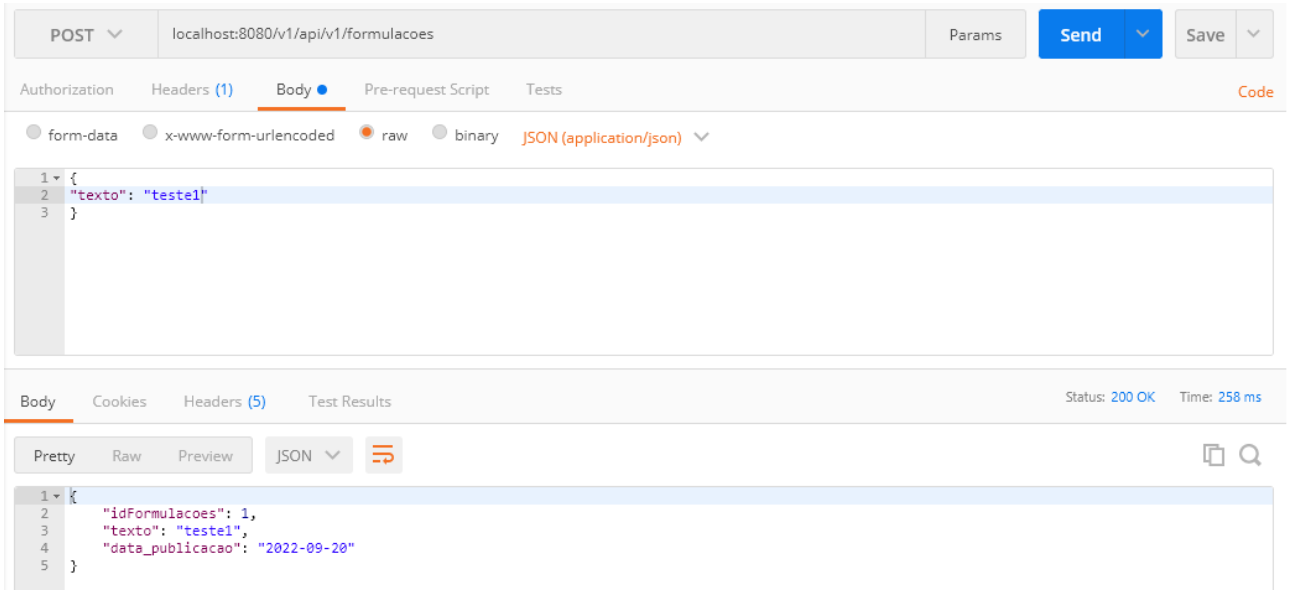
Indicação de Vídeo

REST // Dicionário do Programador. Disponível em: https://www.youtube.com/watch?v=S7MduKwvVGk. Acesso em: 27 set. 2022.

POST

O verbo POST é utilizado para enviar dados ao servidor, dados esses que podem ser processados, validados e armazenados; desse modo, sempre que o cliente precisa enviar dados para serem persistidos ou processados pelo servidor, ele envia um *payload* de dados.

Figura 4 — Exemplo de requisição POST

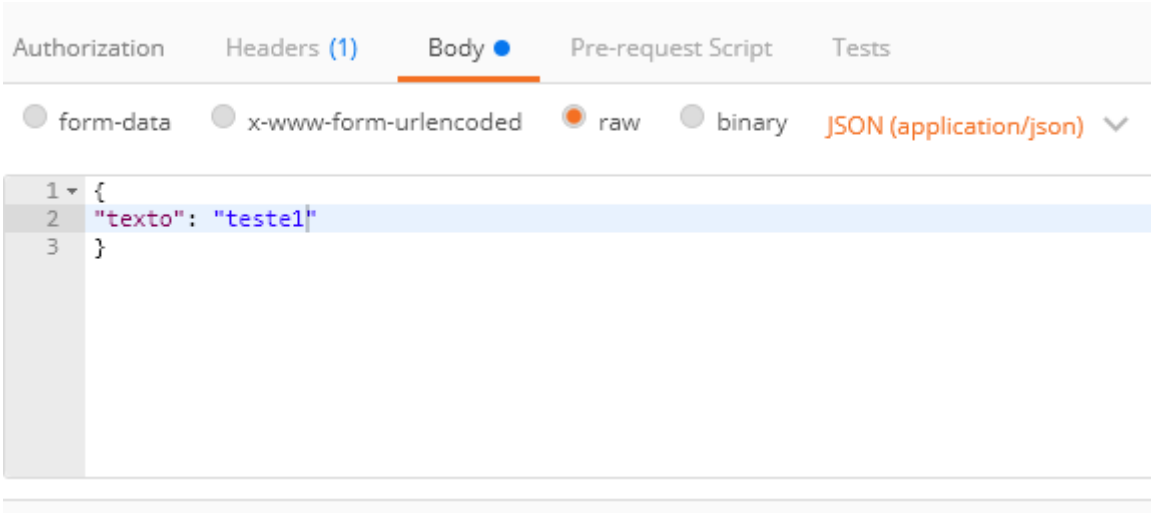


Fonte: O autor (2022).

A figura 4 demonstra a simulação de uma requisição POST para a aplicação anteriormente descrita nesta aula. Observe que é enviada uma requisição para a aplicação que está a ser executada na máquina; desse modo, o software Postman simula o cliente, sendo que o computador simula o servidor.

A aplicação está a ser executada na porta 8080 do computador, por isso o indicativo **localhost:8080**; em seguida vem o caminho da API, que indica para salvar um novo dado.

Figura 5 — Payload: exemplo de requisição POST

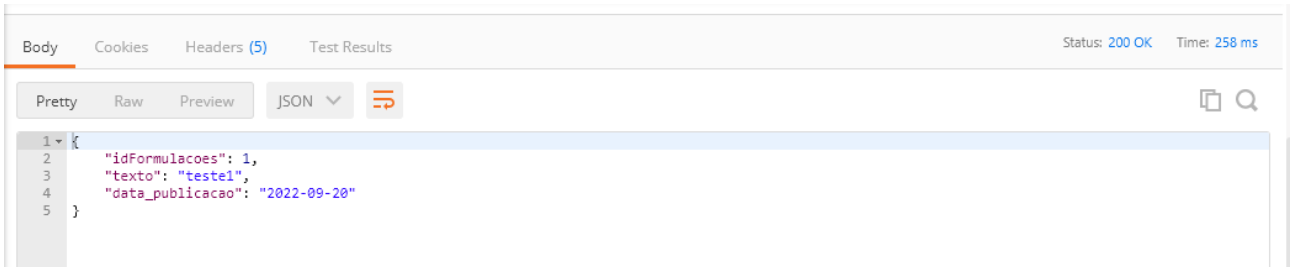


Fonte: O autor (2022).

Observe que, no campo *body*, dentro da figura 5, se apresenta o *payload* enviado para ser salvo na aplicação com os dados dentro de um JSON. Nesse caso, para salvar um novo dado, chamado texto, com o valor “teste1”.

Ao receber essa requisição POST, o servidor realiza o processamento necessário e persiste o dado no banco de dados, e como retorno ele traz um novo payload com mais dados para o cliente.

Figura 6 — Payload: exemplo response POST



Fonte: O autor (2022).

A figura 6 demonstra com mais clareza o retorno do servidor para o cliente com os dados da requisição POST enviados, nesse caso os dados do texto foram salvos com sucesso na base de dados e por isso o payload de retorno (response) possui um dado “idFormulacoes” e “data_publicacao” nesse modelo, indicando a data que foi salva.

Além disso, no canto superior direito da imagem, encontra-se o status HTTP da requisição, no caso 200, que indica sucesso, juntamente com o tempo que o cliente demorou para receber o dado. Nesse exemplo da figura 6, o tempo de resposta do servidor para o cliente foi de 258 milisegundos.

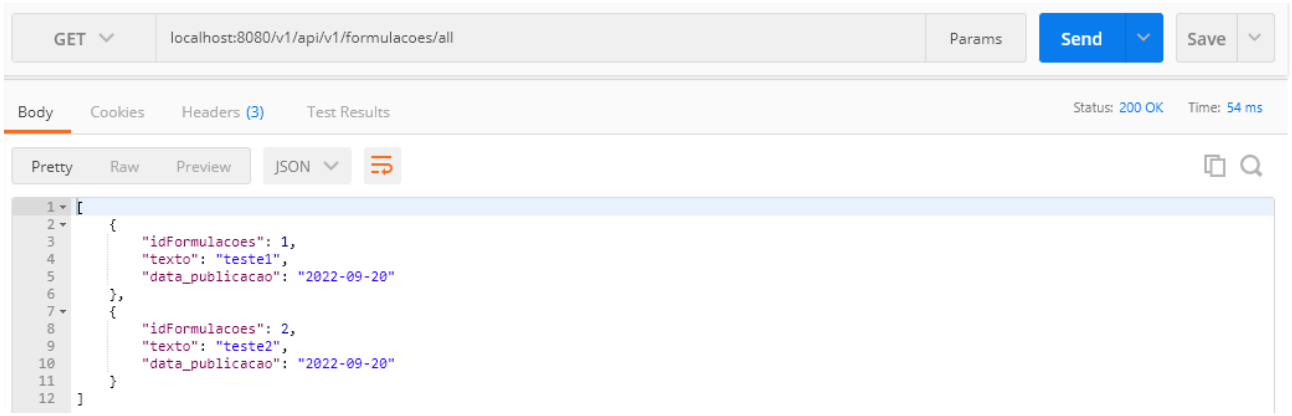
Indicação de Leitura

Códigos de status de respostas HTTP. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status. Acesso em: 27 set. 2022.

GET

A utilização do verbo GET ocorre quando o cliente deseja consultar alguma informação dentro da aplicação que está no servidor.

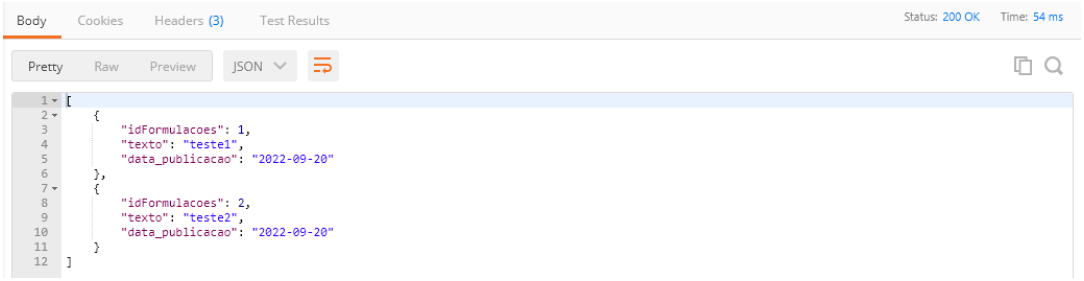
Figura 7 — Exemplo GET all



Fonte: O autor (2022).

A figura 7 demonstra uma requisição GET para o servidor. Observe que o caminho da requisição é diferente da enviada na figura 4, em que existe ainda o indicativo "**/all**", indicando que o cliente deseja saber todas as formulações salvas no banco de dados ao lado do servidor.

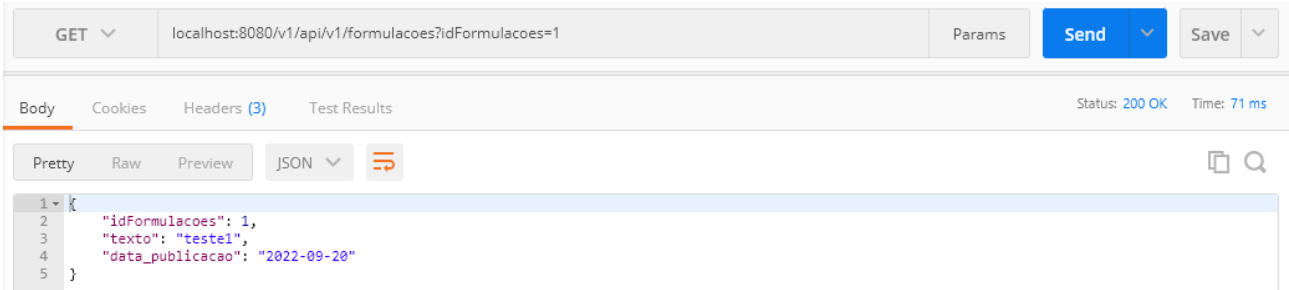
Figura 8 — Exemplo response GET all



Fonte: O autor (2022).

A figura 8 apresenta os aspectos da resposta de uma requisição GET ALL que busca todos os dados de um determinado tipo no banco de dados, nesse caso buscando todas as formulações, observe que existe um contrato de comunicação entre a API e o cliente, pelo qual sempre estão se comunicando por requisições HTTP e utilizando o JSON para enviar dados um para o outro.

Figura 9 — Exemplo response GET one



Fonte: O autor (2022).

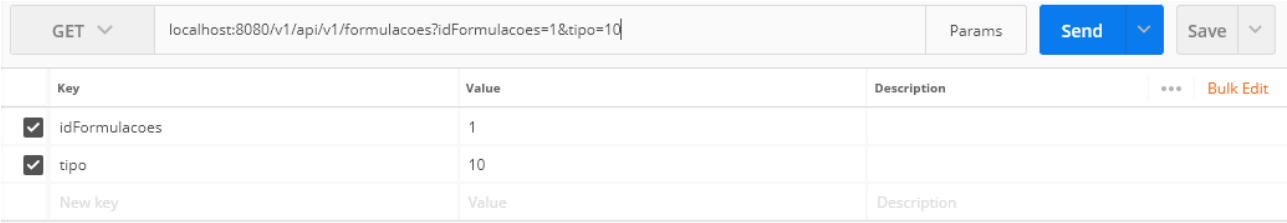
Analisando a figura 9, é possível observar uma requisição GET, porém que retorna somente um dado de formulação, isso ocorre porque o caminho solicitado para o servidor é diferente, nesse caso não existe o ***/all*** e no local é enviado um parâmetro para que o servidor saiba qual dado buscar na base.

Figura 10 — Exemplo response GET one parâmetro

Fonte: O autor (2022).

Observando a figura 10 se analisa o envio de um verbo GET indicando busca, porém de somente um dado o com **idFormulacoes=1** nesta requisição existe um parâmetro demonstrando qual dado o cliente quer que o servidor retorne, por padrão os parâmetros do GET sempre se iniciam com uma **?** indicando que daquele momento para frente são parâmetros e não o caminho(path) da requisição ao servidor.

Figura 11 — Exemplo response GET one, multiparâmetro



Fonte: O autor (2022).

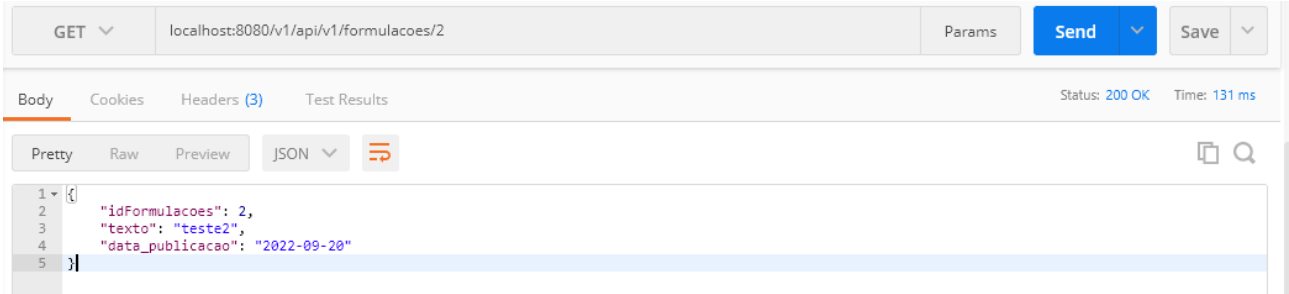
A figura 11 demonstra um exemplo de requisição GET enviando mais de um parâmetro no caminho de chamada ao servidor, caso a API disponibilize e esteja preparada para receber vários parâmetros, esses devem ser enviados com um **&** na frente.

Videoaula 2

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

Figura 12 — Exemplo response GET one, parâmetro path



Fonte: O autor (2022).

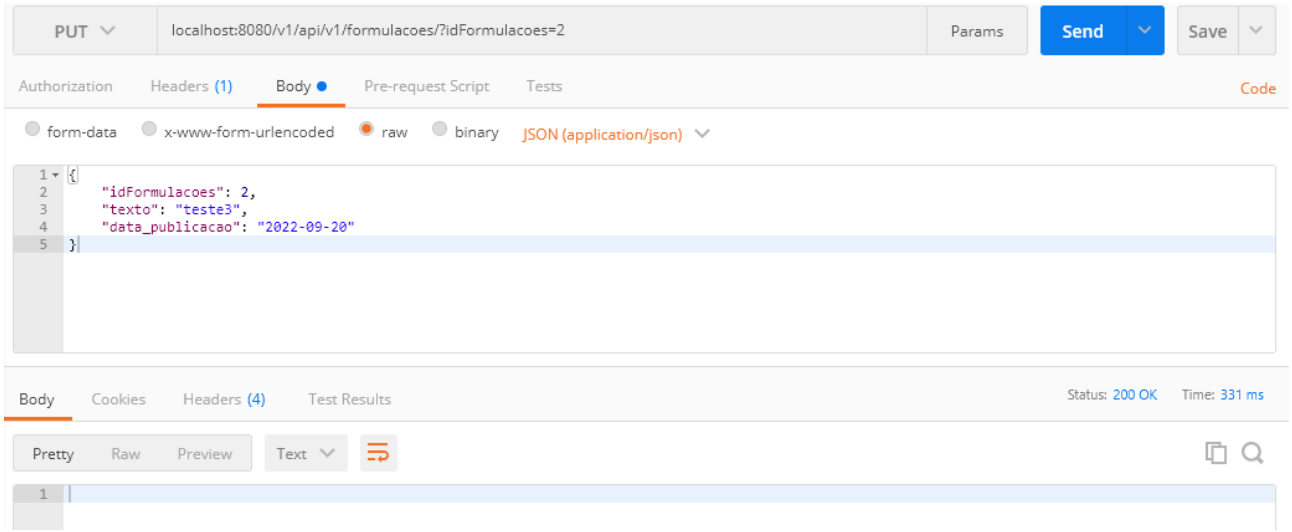
A figura 12 demonstra outra maneira de buscar os dados, enviando o parâmetro de busca dentro do próprio caminho (path). Esse tipo de parâmetro é amplamente utilizado também por alguns desenvolvedores web.

A boa utilização desses recursos favorece a construção de uma API consistente e longeva. Normalmente o envio de parâmetros são padronizados dentro dos projetos e das empresas, ou seja, normalmente uma API utiliza somente parâmetros via path ou somente parâmetros via request, como demonstrado na figura 11.

PUT

Este verbo é utilizado para envio de um payload de atualização de um dado já existente, ou seja, editando um dado na base de dados, dessa forma sempre indica o ID ou identificador do registro que precisa ser atualizado.

Figura 13 — Exemplo response PUT parâmetro



Fonte: O autor (2022).

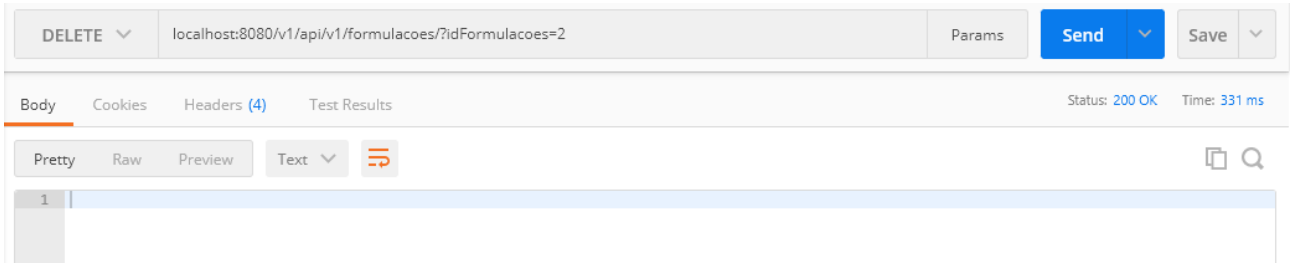
Ao analisar a figura 13, pode-se visualizar uma chamada PUT atualizando os dados do **idFormulacoes 2**, de um modo que é enviado um payload com os dados, porém nessa API é obrigatório informar o id do registro como parâmetro, além de enviar esses dados dentro do corpo (body) da requisição.

Existem outras formas, o verbo PUT não necessariamente precisa ter um id enviado por parâmetro, esse tipo de característica varia de acordo com cada API, lembrando que cada dado apresentado nesta aula não é uma regra, e sim uma possibilidade.

DELETE

De acordo com o próprio nome desse verbo HTTP, ele indica a função de deletar algum tipo de registro, sendo que, nesse caso, assim como no PUT, obrigatoriamente é necessário enviar um tipo de ID ou identificador do dado para que a API saiba qual dados deletar.

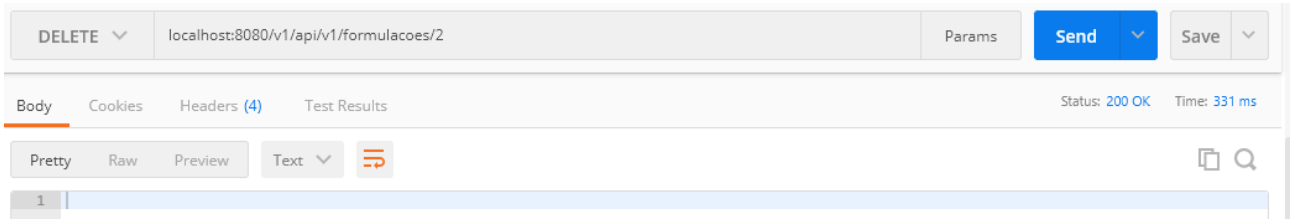
Figura 14 — Exemplo response DELETE parâmetro



Fonte: O autor (2022).

A figura 14 demonstra uma chamada DELETE em que é enviado por parâmetro da requisição (request param) o IdFormulacoes que deve ser excluído da base de dados da aplicação.

Figura 15 — Exemplo response DELETE parâmetro path



Fonte: O autor (2022).

Na figura 15, pode-se visualizar uma requisição DELETE que também envia o identificador do dado que deve ser excluído da base de dados, só que nesse exemplo o identificador é enviado via parâmetro dentro do caminho da chamada ao servidor.

Esse tipo de parâmetro alguns desenvolvedores chamam de path param, em português o parâmetro no caminho. As duas formas são de correta utilização. Tudo depende dos aspectos e das boas práticas consideradas em cada projeto.

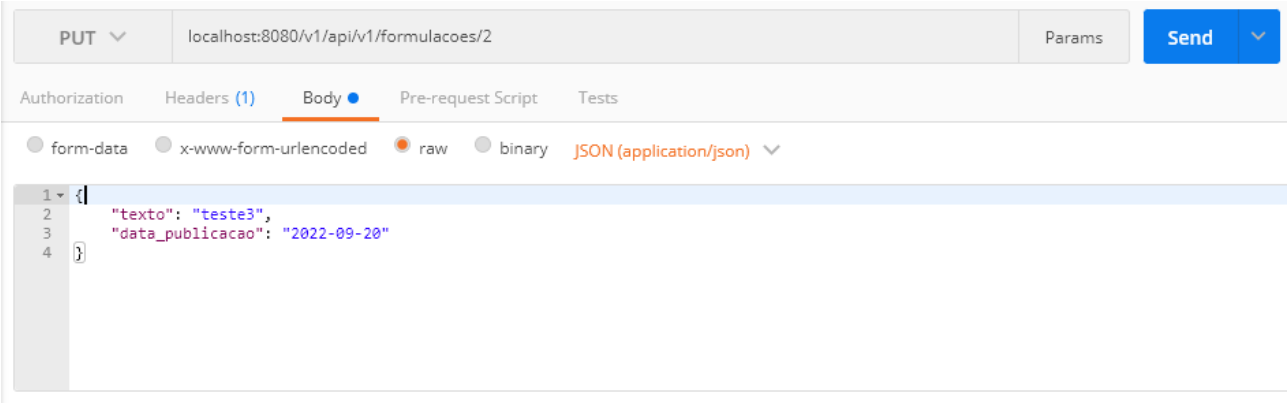
Vale observar que, quando se utiliza do path param, o caminho de requisição pode ser o mesmo tanto para atualizar, quando para deletar ou até mesmo buscar, a principal mudança é o verbo utilizado.

Videoaula 3

Utilize o QR Code para assistir!

Assista ao vídeo para entender melhor o assunto.

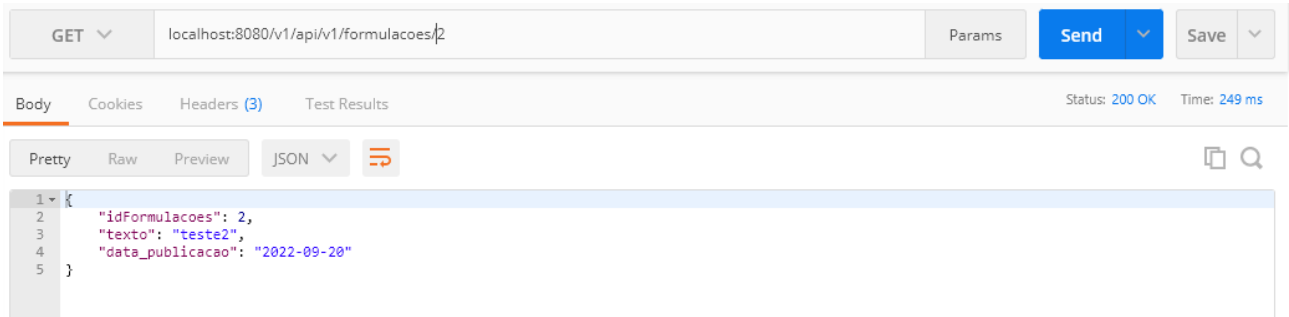
Figura 16 — Exemplo response PUT path param



Fonte: O autor (2022).

A figura 16 apresenta o envio de um verbo PUT em que o identificador do dado está dentro do caminho da requisição. Observe: nesse caso as informações chegam até a aplicação de modo segregado — parte do dado está presente no *payload* dentro do corpo da requisição e outra parte, que é o identificador do dado, está como parâmetro.

Figura 17 — Exemplo response GET path param



Fonte: O autor (2022).

Na figura 17 se apresenta o envio de uma chamada GET para o servidor, em que o identificador do dado está presente no parâmetro de caminho da aplicação.

Observe que não existe uma regra gravada em pedra para onde devem ser enviados os dados, tudo isso depende dos acordos entre o front-end e o back-end; levando em consideração, é claro, cada aspecto do projeto.

Existem outros recursos e conhecimentos relacionados ao HTTP e como funcionam as requisições, além de que os dados podem também ser enviados e transitados dentro de um Header HTTP. Porém existem Headers HTTP que já existem e que vale a pena conhecer.

Indicação de Leitura

HTTP headers. Disponível em: https://www.seobility.net/en/wiki/HTTP_headers. Acesso em: 27 set. 2022.

Os verbos HTTP apresentados nesta aula são somente os básicos e mais utilizados no desenvolvimento web, considerando a estrutura cliente/servidor, desse modo vale ressaltar que existem boas práticas apontadas também dentro dessa aula e que estão vinculadas ao bom uso dos verbos HTTP em requisições dentro de APIs REST.

Por exemplo, é possível que uma requisição POST retorne valores de consulta de todos os dados, isso é uma possibilidade, porém não é considerado uma boa prática; ao realizar atitudes como essa, o desenvolvedor acaba quebrando paradigmas de boas práticas e dificultando o entendimento da aplicação.

Como sabemos, o REST traz um conjunto de definições, e a má aplicação dessas definições acarreta sistemas web mal construídos, com grande demanda de manutenção e com diversos problemas.

Isso não quer dizer que, durante o desenvolvimento web de um projeto, será possível utilizar todos os recursos e todas as boas práticas que foram apresentadas nesta aula e as que ainda iremos conhecer nesta disciplina. Porém, saber que isso impacta diretamente no funcionamento da sua aplicação de forma negativa faz total diferença.

Encerramento da Unidade

Chegamos ao final da nossa segunda unidade, em que você teve contato com os conceitos relacionados à arquitetura cliente/servidor. Provavelmente você já ouviu falar de alguns assuntos abordados nesta unidade.

No mundo dos softwares, há muitas formas de desenvolver sistemas web que podem ser melhores ou piores, no final tudo depende da demanda e do produto que deve ser entregue. Na aula 1, visualizamos alguns conceitos da estrutura cliente/servidor, além de alguns conceitos-base para esse modelo de arquitetura.

Na aula 2, foi discutido e apresentado alguns recursos presentes dentro do desenvolvimento web, como REST e verbos HTTP, além de ferramentas que possibilitam visualizar isso dentro da sua máquina local.

Juntando as duas aulas, foi possível se aproximar mais dos conceitos de algumas boas práticas de desenvolvimento web. Que você possa aplicar esses conhecimentos obtidos em seu cotidiano! Até a próxima!

Bom trabalho!

Referências

O que é API REST? **Red Hat**. Disponível em: https://www.redhat.com/pt-br/topics/api/what-is-a-rest-api/. Acesso em: 22 jun. 2022.

HTTP headers. **Seobility**. Disponível em: https://www.seobility.net/en/wiki/HTTP_headers. Acesso em: 21 jun. 2022.

HTTP Status. **MDN Web Docs**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status. Acesso em: 18 ago. 2022.

What is Api. **AWS Amazon**. Disponível em: https://aws.amazon.com/pt/what-is/api/. Acesso em 18 ago. 2022.

MILETTO, Evandro Manara; BERTAGNOLLI, Silvia de Castro (org). **Desenvolvimento de software II**: introdução ao desenvolvimento web com HTML, CSS, JavaScript e PHP. Porto Alegre: Bookman, 2014.


UniFil
EDUCAÇÃO A DISTÂNCIA
UNIFIL.BR