Ativ 3 mod 3

|  |
| --- |
| REQUISIÇÃO: |
|  |  |
|  | <?xml versão="1.0"?> |
|  | <sabão:Envelope |
|  | xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"> |
|  | <soap:Body xmlns:m="http://www.empresa.com/dadosclientes"> |
|  | <m:dadosClienteRequest> |
|  | <m:cpf>12885687687</m:cpf> |
|  | </m:dadosClienteRequest> |
|  | </soap:Corpo> |
|  | </soap:Envelope> |
|  |  |
|  | RESPOSTA: |
|  |  |
|  | <?xml versão="1.0"?> |
|  | <sabão:Envelope |
|  | xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"> |
|  | <soap:Body xmlns:m="http://www.empresa.com/dadosclientes"> |
|  | <m:dadosClienteRequest> |
|  | <m:nome> Vanildo</m:nome> |
|  | <m:endereco>rua 28 de novembro, n� 11 </m:endereco> |
|  | <m:telefone>081956789346</m:telefone> |
|  | <sabão:Corpo> |
|  | </soap:Envelope> |
|  |  |
|  |  |
|  | Explicação: O que ocorre é simplesmente uma troca de informações entre aplicativos via HTTP, baseado em XML. |
|  | O Envelope em questão está servindo para definir o que está na mensagem e como processá-la. |
|  | 137 |
|  | ...o HTTP e o XML padrão usado nos serviços Web/Desenvolvimento 2 - WSDL e Métodos HTTP.txt |
|  | @@ -0,0 +1,137 @@ |
|  | Existem 9 métodos HTTP, são eles: GET, HEAD, POST, PUT, PATCH, DELETE, OPTIONS, CONNECT, TRACE. |
|  |  |
|  | Para esta atividade será fornecido um cenário de uma biblioteca de livros simplificada, apenas para o objetivo de exemplificação dos métodos. |
|  | Cenário: em uma biblioteca um funcionrio capaz de consultar livros, através do nome do livro, do autor, editora, editar os dados de livros, saber em que prateleira esse livro se encontra, etc. de um sistema online. Sem ter conhecimento, esta função já faz uso de diversos métodos. |
|  |  |
|  | Método GET: Método que serve para consultar dados. Quando o funcionrio faz uma pesquisa sobre um livro X, o sistema utiliza omtodo GET (Pegue), para retornar a informao solicitada. Este método também é usado ao acessar qualquer página da web, o navegador faz um GET no servidor da aplicação, e retorna a página solicitada. |
|  |  |
|  | Exemplo: |
|  | OBTER | https://biblioteca.com/livros/x |
|  | { |
|  | ID: 1657 |
|  | Nome: �X�, |
|  | Ano: 2001, |
|  | Autor: xxxxx |
|  | } |
|  |  |
|  | Método HEAD: solicita uma resposta de forma indiferente ao método GET, a diferença - que apenas o cabeçolho da requisição - retornado. |
|  |  |
|  | Exemplo: |
|  | CABEÇA | https://biblioteca.com/livros/x |
|  | HTTP/2 200 |
|  | Tipo de conteúdo: application/json |
|  | Controle de cache: sem cache |
|  |  |
|  | Método POST: Método que serve para enviar dados para um aplicativo, como realizar um login ou cadastro de novas informações no servidor, como um novo livro. |
|  |  |
|  | Exemplo: |
|  | Enviando para o servidor um novo livro para ser cadastrado, utilizando om�todo POST na url. |
|  |  |
|  |  |
|  | POST | https://biblioteca.com/livros |
|  | { |
|  | Nome: �Y�, |
|  | Ano: 2002, |
|  | Autor: zzzz |
|  | } |
|  |  |
|  | Resposta: |
|  | Como o POST � um método para enviar dados, esperasse que retorne uma resposta, como no exemplo � um método de cadastro de um novo livro, a resposta foi o livro cadastrado, percebesse que a resposta deu o livro com um �ID� , que é o identificador do livro na base de dados. |
|  |  |
|  | { |
|  | ID: 2498, |
|  | Nome:� Y�, |
|  | Ano:202, |
|  | Autor: zzzz |
|  | } |
|  |  |
|  | O programador pode decidir retornar apenas o status de sucesso que código 201 HTTP, mas não abordará códigos nessa atividade. |
|  |  |
|  | Método PUT: Método que serve para atualizar algum livro na biblioteca. |
|  | Exemplo: suponhamos que o individuo responsavel por registrar o livro Y, digitou o autor com algum erro. Mas o programador disponibilizou apenas o método PUT para atualizações. |
|  |  |
|  | COLOCAR | https://biblioteca.com/livros/2498 |
|  | Observe que ao final da url, está se referindo ao ID do livro Y. Por conversão, omtodo PUT exige que todos os dados sejam reenviados. |
|  |  |
|  | Envio: |
|  | { |
|  | Nome: �Y�, |
|  | Ano: 2002, |
|  | Autor: aaaa <- �nico dado que necessitava de atualização�o |
|  | } |
|  |  |
|  | Resposta: |
|  | { |
|  | ID: 2498, |
|  | Nome: �Y�, |
|  | Ano: 2002, |
|  | Autor: aaaa |
|  | } |
|  |  |
|  | Método PATCH: Outro método que serve para atualização, diferente do método PUT. |
|  | Exemplo: o exemplo abaixo � de como seria enviado a atualiza��o anterior, se fosse utilizado om�todo PATCH |
|  |  |
|  | PATCH | https://biblioteca.com/livros/2498 |
|  | Observe que ao final da url, está se referindo ao ID do livro Y. |
|  |  |
|  |  |
|  | Envio: |
|  | { |
|  | Autor: aaaa <- �nico dado que necessitava de atualização�o |
|  | } |
|  |  |
|  | Resposta: |
|  | { |
|  | ID: 2498, |
|  | Nome: �Y�, |
|  | Ano: 2002, |
|  | Autor: aaaa |
|  | } |
|  |  |
|  | Método DELETE: como o próprio nome sugere, este método é responsável por deletar livros de nossa biblioteca. |
|  | Exemplo: por um motivo adverso, a biblioteca n�o ter� mais o livro Y na sua estante e o livro precisa ser deletado do sistema. |
|  |  |
|  | APAGAR | https://biblioteca.com/livros/2498 |
|  | Observe que ao final da url, está se referindo ao ID do livro Y. |
|  |  |
|  | Neste método o envio de informação é opcional, da mesma forma que a resposta pode conter apenas o código de status. |
|  |  |
|  | Método OPTION: este método é utilizado para consultar as opções permitidas de execução, como quais métodos podem ser executados. |
|  | Exemplo: |
|  | OPÇÃO | https://biblioteca.com/ |
|  |  |
|  | A resposta ao executar este método é um cabeçolho, semelhante ao HEAD, porém com informações apenas de métodos que podem ser utilizados. |
|  | Resposta: |
|  | HTTP/2 200 |
|  | Permitir: GET, POST, PUT, PATCH, DELETE, OPTIONS |
|  | Acesso-Controle-Permitir-Origem: \* |
|  | Métodos de permissão de controle de acesso: GET, POST, PUT, PATCH, DELETE, OPTIONS |
|  | Cabeçalhos de acesso-controle-permissão: tipo de conteúdo |
|  |  |
|  | Método CONNECT: este método é responsável por estabelecer uma conexão ponta a ponta entre dois computadores, normalmente cliente - servidor, através de proxy/firewall. Funciona como uma espécie de canal por onde as comunicações são feitas. |
|  |  |
|  | Exemplo: |
|  | CONECTAR | www.biblioteca.com |
|  |  |
|  | Resposta: |
|  | HTTP/2 200 CONEXÃO ESTABELECIDA |
|  | DADOS: DD/MM/AAAA HH-MM-SS GMT |
|  | SERVIDOR: APACHE/3.1.4(WIN32) |
|  |  |
|  | Método TRACE: este método é usado para a realização de um teste de loopback enviando uma mensagem no destino, e retornando um cabeçolho que será útil para depurações. |
|  |  |
|  | Exemplo: |
|  | RASTREIO | www.biblioteca.com |
|  |  |
|  | Resposta: |
|  | HTTP/2 200 CONEXÃO ESTABELECIDA |
|  | DADOS: DD/MM/AAAA HH-MM-SS GMT |
|  | SERVIDOR: APACHE/3.1.4(WIN32) |
|  | CONEXÃO: FECHAR |
|  |  |
|  | WSDL |
|  | WSDL uma nota XML para descrever um serviço da web. Uma definição WSDL indica a um cliente como compor uma solicitação de serviço da web e descreve uma interface que forneceu pelo provedor de serviços da web. Descrição da retirada do site: ibm.com. |
|  |  |
|  | Imagine que outras empresas serão agregadas a nossa biblioteca, e os programadores dessas novas empresas não têm conhecimento de como todos os serviços foram implementados, suas chamadas, urls, e serviços adicionais, ai que entra no WSDL . |
|  |  |
|  | Um WSDL, que significa Web Services Description Language, é um padrão de mercado para descrever Web Services de forma a eliminar ao máximo a necessidade de comunicação entre as partes envolvidas em uma integração de dados. Ele normalmente é encontrado quando adicionamos "wsdl" ao final da URL onde o Web Service está hospedado. . |
|  | 16 |
|  | ...Desenvolvimento 3 - Dominar o serviço SOAP e saber implementa-lo em JAVA/BancoServer.java |
|  | @@ -0,0 +1,16 @@ |
|  | aplicativo de pacote ; |
|  |  |
|  | importar javax. jws . Serviço de internet ; |
|  | importar javax. jws . WebMethod ; |
|  | importar javax. jws . sabão . SOAPBinding ; |
|  | importar javax. jws . sabão . SOAPBinding . Estilo ; |
|  |  |
|  | @ServiçoWeb |
|  | @ SOAPBinding ( estilo = Estilo . RPC ) |
|  | public interface BancoServer { |
|  | @WebMethod String pegarBoleto ( String codigoBarras ); |
|  | @WebMethod String criarBoleto ( float valor , String cpfCliente ) ; |
|  | @ WebMethod boolean verificarPagamento ( String codigoBarras ); |
|  | @ WebMethod boolean cancelarBoleto ( String codigoBarras ); |
|  |  |
|  | } |
|  | 27 |
|  | ...nvolvimento 3 - Dominar o serviço SOAP e saber implementa-lo em JAVA/BancoServerImpl.java |
|  | @@ -0,0 +1,27 @@ |
|  | aplicativo de pacote ; |
|  |  |
|  | importar java. . \_ Dados ; |
|  | importar javax. jws . Serviço de internet ; |
|  |  |
|  | @ WebService ( endpointInterface = "app.BancoServer" ) |
|  | public class BancoServerImpl implementa BancoServer { |
|  |  |
|  | public String pegarBoleto ( String codigoBarras ) { |
|  | return "Boleto para o codigo" + codigoBarras ; |
|  | } |
|  |  |
|  | public String criarBoleto ( float valor , String cpfCliente ) { |
|  | return "123.456.78910" ; |
|  | } |
|  |  |
|  | public boolean verificarPagamento ( String codigoBarras ) { |
|  | retorna verdadeiro; |
|  |  |
|  | } |
|  |  |
|  | public boolean cancelarBoleto ( String codigoBarras ) { |
|  | retorna falso; |
|  |  |
|  | } |
|  |  |
|  | } |
|  | 12 |
|  | ...rvices/Desenvolvimento 3 - Dominar o serviço SOAP e saber implementa-lo em JAVA/main.java |
|  | @@ -0,0 +1,12 @@ |
|  | aplicativo de pacote ; |
|  |  |
|  | importar javax. xml . ws. ponto final; |
|  |  |
|  | public class BancoServerPublisher { |
|  |  |
|  | public static void main ( String [] args ) |
|  | { |
|  | Ponto final. publicar("http://127.0.0.1:3030/app" , |
|  | novo BancoServerImpl()); |
|  | } |
|  | } |
|  | 18 |
|  | ... os conceitos do modelo REST/Desenvolvimento 4 - Conhecer os conceitos do modelo REST.txt |
|  | @@ -0,0 +1,18 @@ |
|  | 1.Sem estado; |
|  | 2.Cliente-servidor; |
|  | 3.Interface uniforme; |
|  | 4.Armazenamento em cache; |
|  | 5.Sistema em camadas; |
|  | 6.Código sob demanda; |
|  |  |
|  | Sem estado: de acordo com este princípio, cada solicitação enviada do cliente ao servidor deve incluir todas as informações necessárias para compreender a solicitação. As APIs são denominadas �Sem Estado� porque cada chamada � independente umas das outras. |
|  |  |
|  | Cliente-servidor: este princípio trabalha com o conceito de que cliente e servidor devem ser isolados um do outro e ter permissão para desenvolver de forma independente. Ao separar as preocupações de interface de usuário do armazenamento de dados, é possível melhorar a portabilidade através de múltiplas plataformas, simplificar os componentes do servidor, mas principalmente, permitir a evolução de forma independente uma vez que um não tem dependência do outro. |
|  |  |
|  | Interface uniforme: uma característica da arquitetura REST � uma interface uniforme entre o cliente e o servidor. Como o cliente e o servidor monitoram esta interface, deve-se ter um "contrato" bem definido para comunicação entre os lados. Há quatro princípios que devem ser seguidos para obter uma interface uniforme: Identificação dos Recursos, Representação dos recursos, Mensagens auto-descritivas e Hipermídia. |
|  |  |
|  | armazenamento em cache: como uma API sem estado pode aumentar a sobrecarga de solicitação gerenciando grandes cargas de chamadas de entrada e saída, o cache ajuda a melhorar o desempenho, a escalabilidade e eficiência uma vez que reduz o tempo de resposta média quando detectado entre uma série de cliente-servidor confortável. |
|  |  |
|  | Sistema em atletas: a arquitetura REST inclui vrias internacionais que operam juntas para construir uma superioridade que contribui para que a aplicação seja escalável e flexível. É recomendável que o cliente nunca se conecte diretamente no servidor de aplicativo e que uma camada de balanceamento de carga seja adicionada entre cliente-servidor, isso ajuda na segurança a restringir o desempenho do componente. de modo que cada componente não pode "ver" além da camada imediata com a qual está se misturando. |
|  |  |
|  | Código sob demanda: código sob demanda define a possibilidade de serviços baseados em REST responderem como representação de um recurso de informação executável pelo cliente. Esta possibilidade de entrega de funcionalidades, no entanto deve ser suportada pelo cliente criando um ambiente de execução próprio para sua execução. Na maioria das vezes, um servidor retorna representação estética de recursos no formato XML ou JSON. Porém, quando necessário, os servidores podem fornecer código executável ao cliente |
|  | 82 |
|  | ...Desenvolvimento 5 - Definir um modelo REST/Desenvolvimento 5 - Definir um modelo REST.txt |
|  | @@ -0,0 +1,82 @@ |
|  | Para simular um CRUD de uma base de livros, usaremos o dominio: www.biblioteca.com, com os dados abaixo: |
|  |  |
|  | Var livros = [ |
|  | {Nome: livro1, |
|  | Autor: autor1, |
|  | ISBN: 123456-123}, |
|  | {Nome: livro2, |
|  | Autor: autor2, |
|  | ISBN: 123456-456}, |
|  | ] |
|  |  |
|  | Para fazer o CRUD (Created, Read, Update, Delete) é necessário utilizar os métodos HTTP POST, GET, PUT e DELETE e cada um deles � configurado em uma URL diferente, além de que cada um necessitar� de dados para realizar uma operação . |
|  |  |
|  | Para fazer a criação de um novo registro, ou inserir um livro novo, utilize o método POST passando o novo livro no corpo da requisição, da seguinte forma: |
|  |  |
|  | POST | www.biblioteca.com/ |
|  | Corpo da requisição de um objeto livro dentro de um json: |
|  | { |
|  | { �nome�: �livro3�, |
|  | �Autor�: �autor3� |
|  | �isbn�: 123456-789} |
|  | } |
|  |  |
|  | Programação no servidor: |
|  | metodoPost(�/�, pedido, resposta) =>{ |
|  | livros.push(pedido.corpo) |
|  | return response(�Livro adicionado com Sucesso�) |
|  | } |
|  |  |
|  | Para fazer o Read (leitura) de um registro, utiliza omtodo GET passando o que se deseja buscar na url da requisição, da seguinte forma: |
|  | OBTER | www.biblioteca.com/livros/3 |
|  | Livros se referem ao recurso e 3 ao identificador do livro |
|  |  |
|  | Programação no servidor: |
|  | //o 3 se refere ao terceiro livro, mas no array o terceiro tem índice 2 |
|  |  |
|  | metodoGet(�/livros/{id}�, pedido, resposta) =>{ |
|  | índice = request.id - 1 //3 � 1 = índice 2 |
|  | if(índice < 0){ |
|  | índice = 0 |
|  | } |
|  | return response(livros[index]) |
|  | } |
|  |  |
|  | Para fazer o Update (atualização) de um registro, utiliza omtodo PUT passando os dados atualizados no corpo da requisição, da seguinte forma: |
|  | COLOCAR | www.biblioteca.com/livros/3 |
|  | Livros se referem ao recurso e 3 ao identificador do livro |
|  |  |
|  | Corpo da requisição de um objeto livro dentro de um json: |
|  | { |
|  | { �nome�: �livro3.123�, //atualização do nome |
|  | �Autor�: �autor3� |
|  | �isbn�: 123456-789} |
|  | } |
|  | Programação no servidor: |
|  | //o 3 se refere ao terceiro livro, mas no array o terceiro tem índice 2 |
|  |  |
|  | metodoPut(�/livros/{id}�, pedido, resposta) =>{ |
|  | índice = request.id - 1 //3 � 1 = índice 2 |
|  | if(índice < 0){ |
|  | índice = 0 |
|  | } |
|  | livros[index] = request.body |
|  | return response(�Livro atualizado�) |
|  | } |
|  |  |
|  | Para fazer o Delete de um registro, utiliza om�todo Delete passando o index do livro para ser deletado na url, da seguinte forma: |
|  | APAGAR | www.biblioteca.com/livros/3 |
|  | Livros se referem ao recurso e 3 ao identificador do livro |
|  |  |
|  | Programação no servidor: |
|  | //o 3 se refere ao terceiro livro, mas no array o terceiro tem índice 2 |
|  |  |
|  | metodoDelete(�/livros/{id}�, pedido, resposta) =>{ |
|  | índice = request.id - 1 //3 � 1 = índice 2 |
|  | if(índice < 0){ |
|  | índice = 0 |
|  | } |
|  | Livros.splice(index , 1) //remover o item do array |
|  | return response(�Livro Removido�) |
|  | } |
|  |  |
|  | 47 |
|  | Introdução a Web Services/Desenvolvimento 6 - Implementar um modelo REST/index.js |
|  | @@ -0,0 +1,47 @@ |
|  | import express , { json , resposta } de 'express' |
|  | importar statusCode de 'http-status-codes' |
|  |  |
|  | aplicativo const = expresso ( ) ; |
|  |  |
|  | const PORTA = processo . env . PORTO || 3000 ; |
|  |  |
|  | deixar usuários = [ |
|  | { id : 1 , nome : "Arthur" , idade : 27 } , |
|  | { id : 2 , nome : "Luiz" , idade : 35 } , |
|  | { id : 3 , nome : "Silva" , idade : 35 } |
|  | ] |
|  | aplicativo . use ( express . json ( ) ) |
|  | aplicativo . escute ( PORTA , ( ) => { |
|  | console . log( `Servidor rodando na porta ${ PORT } ` ) |
|  | } ) |
|  |  |
|  | aplicativo . get('/users', (req, resp) => { |
|  | retorno resp. enviar (usuários) |
|  | } ) |
|  |  |
|  | aplicativo . post('/users', (req, resp) => { |
|  | const newUser = req . corpo ; |
|  | usuários . push ( novoUsuário ) |
|  |  |
|  | retorno resp. status ( statusCode . CREATED ) . enviar ( novoUsuário ) |
|  | } ) |
|  |  |
|  | aplicativo . put('/users/:userId', (req, resp) => { |
|  | const userId = req. parâmetros . ID do usuário; |
|  | const attUser = req . corpo ; |
|  | attUser . id = userId; |
|  |  |
|  | usuários = usuários. mapa ((usuário) => { |
|  | if ( Número ( userId ) === user . id ) { |
|  | return attUser ; |
|  | } |
|  | retornar usuário ; |
|  | } ) |
|  | retorno resp. enviar (attUser); |
|  | } ) |
|  |  |
|  | aplicativo . delete ('/users/:userId', (req, resp) => { |
|  | const userId = req. parâmetros . ID do usuário; |
|  | usuários = usuários. filtro ( ( usuário ) => usuário . id !== Número ( userId ) ) |
|  | retorno resp. status (statusCode. NO\_CONTENT). enviar ( ) ; |
|  | } ) |
|  | 47 |
|  | ...eb Services/Desenvolvimento 8 - Dominar a implementação de fluxos de requisições/index.js |
|  | @@ -0,0 +1,47 @@ |
|  | import express , { json , resposta } de 'express' |
|  | importar statusCode de 'http-status-codes' |
|  |  |
|  | aplicativo const = expresso ( ) ; |
|  |  |
|  | const PORTA = processo . env . PORTO || 3000 ; |
|  |  |
|  | deixar usuários = [ |
|  | { id : 1 , nome : "Arthur" , idade : 27 } , |
|  | { id : 2 , nome : "Luiz" , idade : 35 } , |
|  | { id : 3 , nome : "Silva" , idade : 35 } |
|  | ] |
|  | aplicativo . use ( express . json ( ) ) |
|  | aplicativo . escute ( PORTA , ( ) => { |
|  | console . log( `Servidor rodando na porta ${ PORT } ` ) |
|  | } ) |
|  |  |
|  | aplicativo . get('/users', (req, resp) => { |
|  | retorno resp. enviar (usuários) |
|  | } ) |
|  |  |
|  | aplicativo . post('/users', (req, resp) => { |
|  | const newUser = req . corpo ; |
|  | usuários . push ( novoUsuário ) |
|  |  |
|  | retorno resp. status ( statusCode . CREATED ) . enviar ( novoUsuário ) |
|  | } ) |
|  |  |
|  | aplicativo . put('/users/:userId', (req, resp) => { |
|  | const userId = req. parâmetros . ID do usuário; |
|  | const attUser = req . corpo ; |
|  | attUser . id = userId; |
|  |  |
|  | usuários = usuários. mapa ((usuário) => { |
|  | if ( Número ( userId ) === user . id ) { |
|  | return attUser ; |
|  | } |
|  | retornar usuário ; |
|  | } ) |
|  | retorno resp. enviar (attUser); |
|  | } ) |
|  |  |
|  | aplicativo . delete ('/users/:userId', (req, resp) => { |
|  | const userId = req. parâmetros . ID do usuário; |
|  | usuários = usuários. filtro ( ( usuário ) => usuário . id !== Número ( userId ) ) |
|  | retorno resp. status (statusCode. NO\_CONTENT). enviar ( ) ; |
|  | } ) |