Faculdade IMPACTA

Alex Silva de Sousa <alex.ssousa@aluno.faculdadeimpacta.com.br>

AC2 - Desenvolvimento de APIs e Microsserviços

1 mensagem

Formulários Google <forms-receipts-noreply@google.com> 2 de novembro de 2024 às 17:38
Para: alex.ssousa@aluno.faculdadeimpacta.com.br

Agradecemos o preenchimento de AC2 - Desenvolvimento de APIs e Microsserviços

Veja as respostas enviadas.

AC2 - Desenvolvimento de APIs e Microsserviços

As questões contidas nessa atividade estão relacionadas aos conteúdos das partes 04, 05 e 06.

Seu e-mail (**alex.ssousa@aluno.faculdadeimpacta.com.br**) foi registrado quando você enviou este formulário.

Qual dos seguintes verbos HTTP são idempotentes ou seguros? *

- (●) HEAD e GET são idempotentes e seguros. PUT e DELETE são idempotentes, mas não seguros. POST não é nem idempotente e nem seguro.
- ( ) POST e GET são idempotentes e seguros. PUT e HEAD são idempotentes, mas não seguros. DELETE não é nem idempotente e nem seguro.
- ( ) PUT e DELETE são idempotentes e seguros. GET e POST são idempotentes, mas não seguros. HEAD não é nem idempotente e nem seguro.
- ( ) POST e GET são idempotentes e seguros. PUT eHEAD são idempotentes, mas não seguros. DELETE não é nem idempotente e nem seguro.
- ( ) SELECT é idempotente e seguro. INSERT e UPDATE são idempotentes, mas não seguros. DELETE e DROP não é nem idempotente e nem seguro.

Dentre os verbos HTTP POST, GET, PUT, HEAD e DELETE, quais tem corpo na requisição e quais tem corpo na resposta? *

- (●) Apenas PUT e POST têm corpo na requisição. Apenas GET, PUT, POST e DELETE têm corpo na resposta.
- ( ) Apenas POST e GET têm corpo na requisição. Apenas PUT e POST têm corpo na resposta.
- ( ) Apenas HEAD, GET e DELETE têm corpo na requisição. Apenas HEAD têm corpo na resposta.
- ( ) Apenas GET, PUT, POST e DELETE têm corpo na requisição. Apenas PUT e POST têm corpo na resposta.
- ( ) Apenas SEND, REQUEST e TRY têm corpo na requisição. Apenas RECEIVE, DROP e SELECT têm corpo na resposta.

Marque a alternativa que descreve corretamente a diferença GET, POST, HEAD e DELETE. *

- (●) O GET serve para fazer o download de um recurso. O HEAD serve para fazer o download apenas dos cabeçalhos referentes a um recurso. O POST serve para enviar informações ao servidor. O DELETE serve para destruir/apagar um recurso.
- ( ) O HEAD serve para fazer o download de um recurso. O GET serve para fazer o download apenas dos cabeçalhos referentes a um recurso. O POST serve para enviar informações ao servidor. O DELETE serve para destruir/apagar um recurso.
- ( ) O HEAD serve para fazer o upload de um recurso. O GET serve para fazer o upload apenas dos cabeçalhos referentes a um recurso. O POST serve para receber informações do servidor. O DELETE serve para destruir/apagar um recurso.
- ( ) O GET serve para fazer o apagamento de um recurso. O POST serve para fazer o apagamento apenas dos cabeçalhos referentes a um recurso. O POST serve para apagar informações no servidor. O DELETE não serve para destruir/apagar um recurso.
- ( ) O POST serve para fazer o download de um recurso. O DELETE serve para fazer o download apenas dos cabeçalhos referentes a um recurso. O GET serve para enviar informações ao servidor. O HEAD serve para destruir/apagar um recurso.

Qual é a ordem correta das 7 camadas do modelo OSI? *

- (●) Física, Enlace, Rede, Transporte, Sessão, Apresentação e Aplicação
- ( ) Física, Enlace, Transporte, Rede, Apresentação, Sessão e Aplicação

- Física, Enlace, Rede, Transporte, Aplicação, Sessão e Apresentação
- Pão, Hambúrguer, Queijo, Tomate, Alface, Pão e Gergelim
- Aplicação, Sessão, Apresentação, Rede, Física, Enlace e Transporte.

Sobre a biblioteca requests, qual é a alternativa correta? *

- (●) A biblioteca requests serve para realizar requisições HTTP a outros hosts.
- A biblioteca requests serve para servir conteúdo por meio do protocolo HTTP.
- A biblioteca requests serve para armazenar dados acerca de avaliações de filmes.
- A biblioteca requests serve para diferenciar diferentes tipos de requisições HTTP quanto a idempotência e segurança.
- A biblioteca requests serve para converter dicionários em formato JSON e vice-versa.

Em relação aos verbos HTTP, qual é a melhor definição de idempotente e seguro? *

- (●) Idempotente é quando múltiplas requisições idênticas surtem o mesmo efeito de uma só requisição. Seguro é quando a requisição não deve produzir efeitos colaterais potencialmente danosos.
- Idempotente é quando a requisição não deve produzir efeitos colaterais potencialmente danosos. Seguro é quando múltiplas requisições idênticas surtem o mesmo efeito de uma só requisição.
- Idempotente é quando múltiplas requisições simultâneas surtem o mesmo efeito de uma só requisição. Seguro é quando a requisição é autenticada contra efeitos colaterais não danosos.
- Idempotente é quando a requisição é autenticada contra efeitos colaterais não danosos. Seguro é quando múltiplas requisições simultâneas surtem o mesmo efeito de uma só requisição.
- Idempotente é quando múltiplas requisições simultâneas não são seguras. Seguro é quando a requisição não deve produzir efeitos colaterais potencialmente idempotentes.

Considere a URL http://example.com:5000/a/b/c?x=1&y=2#corpo - Qual é a resposta que corretamente identifica seus componentes? *

- (●) Protocolo = http:// ; Domínio = example.com ; Porta = :5000 ; Caminho = /a/b/c ; Query-string = ?x=1&y=2 ; Fragmento = #corpo
- ( ) Domínio = http:// ; Caminho = example.com ; Protocolo = :5000 ; Query-string = /a/b/c ; Porta = ?x=1&y=2 ; Fragmento = #corpo
- ( ) Porta = http:// ; Núcleo = example.com ; Valor = :5000 ; Pasta = /a/b/c ; Código = ?x=1&y=2 ; Hashtag #corpo
- ( ) Protocolo = http:// ; Domínio = example.com ; Porta = :5000 ; Query-string = /a/b/c ; Cabeçalhos = ?x=1&y=2 ; Link = #corpo
- ( ) Indicação = http:// ; Localização = example.com ; Entrada = :5000 ; Pasta = /a/b/c ; Pesquisa = ?x=1&y=2 ; Fragmento = #corpo

Qual desses códigos é o mais apropriado para para baixar um recurso de uma URL (sem query string) em formato JSON? *

- (●) requests.get("http://example.com").json()
- ( ) requests.post("http://example.com", json = {"a": 1, "b": 2}).json()
- ( ) requests.get("http://example.com", json = {"a": 1, "b": 2}).json()
- ( ) requests.post("http://example.com").json()
- ( ) requests.head("http://example.com").json()

Qual desses códigos é o mais apropriado para enviar um recurso em formato JSON a um servidor e obter o código de status da resposta HTTP? *

- ( ) status = requests.get("http://example.com").status_code
- (●) status = requests.post("http://example.com", json = {"a": 1, "b": 2}).status_code
- ( ) status = requests.get("http://example.com", json = {"a": 1, "b": 2}).status_code
- ( ) status = requests.post("http://example.com").json().status_code
- ( ) status = requests.head("http://example.com").json({"a": 1, "b": 2}).status_code

Mostre qual a afirmação sobre os códigos de status HTTP é a correta. *

- (●) 2xx representa uma requisição bem sucedida. 3xx representa redirecionamento. 4xx representa um erro do cliente. 5xx representa um erro do servidor.
- ( ) 1xx representa uma requisição bem sucedida. 2xx representa redirecionamento. 3xx representa um erro do cliente. 4xx representa um erro do servidor.
- ( ) 2xx representa uma requisição bem sucedida. 3xx representa um erro de conexão. 4xx representa um erro do servidor. 5xx representa um erro do cliente.
- ( ) 2xx representa uma requisição eviada. 3xx representa redirecionamento. 4xx representa um erro de configuração. 5xx representa um erro de programação.
- ( ) 2xx representa redirecionamento. 3xx representa erro do cliente. 4xx representa um erro do servidor. 5xx representa redirecionamento.
- ( ) 1xx representa espera. 2xx representa requisição enviada. 3xx representa transferência. 4xx representa espera. 5xx representa congestionamento. 6xx representam erros graves.