|  |  |
| --- | --- |
| **_Brasão DEC** | **PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS**  **PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO**  **DIRETORIA DE EDUCAÇÃO CONTINUADA** |

**­­Curso/Oferta:** Desenvolvimento de Web Full Stack  
**Disciplina:** API e Web Services   
**Professor:** Marco Mendes 

# Atividade Aula 01

**Pré-Requisitos**

* Postman v 7.33 instalado.  
  Sítio para baixar: <https://www.postman.com/downloads/>
* NPM 6.14 e Node 14.9

Sítio para baixar: https://nodejs.org/en/download/current/

**PARTE 1 – Compreensão dos Fundamentos HTTP**

Execute o seguinte tutorial.

Passo 1 - Abra o Postman e examine a sua aba chamada (Launchpad).

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Passo 2 - Vamos criar uma API mínima para compreender alguns fundamentos HTTP. Para isso clique sobre o botão Designing and Mocking API. Ele irá abrir uma nova Aba chamada Bootcamp.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Passo 3 - Clique no botão Designing and Mocking API e na tela posterior depois no botão Design API.

A tela abaixo será mostrada e você agora será para criar uma API simples chamada Designing API.

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

O resultado desse processo deve resultar em uma API simples criada na aba lateral de Collections do lado esquerdo.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Explore livremente os diferentes métodos na interface criada.

Com a API criada, responda agora às seguintes perguntas.

1. Quais os cabeçalhos HTTP enviados na requisição exemplo GET?
   * Cookie, Cache-Control, Postman-Token, Host, User-Agent, Accept, Accept-Encoding, Connection
2. Quais os cabeçalhos HTTP recebidos na resposta exemplo GET?
   * Date, Content-Type, Content-Length, Connection, ETag, Vary, set-cookie
3. Invoque a URL <https://postman-echo.com/get> do seu navegador? Você observa algo diferente?
   * Houveram mais cabeçalhos na requisição e o user-agent mudou para mozilla .
4. Qual a função do cabeçalho User-Agent na chamada dos métodos?
   * Ele identifica o cliente que está fazendo a requisição.
5. As requisições GET operam com otimização de compressão de dados? Justifique.
   * É possível informar tags que otimizam a solicitação de recurso como o gzip e o deflate.
6. Introduza um parâmetro chamado CODIGO\_CLIENTE na requisição POST

Tabela

Descrição gerada automaticamente

O que você observa de diferente na chamada e na resposta?

Na chamada da resposta foi apresentado o código do cliente no json como “args”.

**PARTE 2 – Testes de Contratos HTTP**

Uma boa prática na utilização de HTTP e chamadas de API é realizar testes de contratos sobre essas chamadas. Por exemplo, verificar códigos de resposta, erros, formatos de resposta ou até mesmo estabelecer o tempo máximo de uma requisição.

Passo 1. Vamos montar um exemplo simples. Abra a requisição POST e então a aba de testes e examine a tela mostrada a seguir. Localize o snippet de código do lado direito chamado. *Response time is less than 200ms.*

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente



Clique sobre esse botão e examine o código gerado.

Interface gráfica do usuário, Texto, Aplicativo, Email

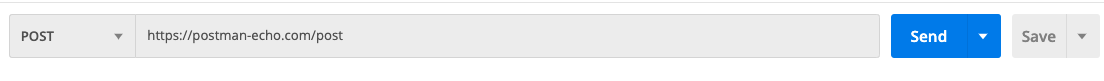
Descrição gerada automaticamente

Rode o código e examine o resultado. Se a sua Internet estiver muito rápida, o seu teste irá passar. Se a requisição demorar mais, o seu teste irá falhar. Um exemplo é mostrado abaixo para um caso de falha. Observe que a requisição demorou 640 ms.

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Para que esse código seja mantido clique no botão Save, conforme mostrado abaixo.





1. Estenda esse código de testes para os outros métodos HTTP com limite de tolerância de 500ms para as operações de escrita e 200ms para as operações de leitura.

Use agora o botão Runner para rodar a sua coleção inteira conforme mostrado abaixo.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Entregável: Foto da execução dos seus testes de contratos.

Interface gráfica do usuário, Texto, Aplicativo, Email, Site

Descrição gerada automaticamente

1. Os seus testes são determinísticos (sempre irão gerar o mesmo resultado)? Justifique.

R: Caso o tempo de resposta seja menor que 200ms para requisição de leitura e 500ms para escrita, o resultado sempre será o mesmo, caso a requisição atrase, recebera uma falha.

**PARTE 3 – Consumo de uma API Real**

Acesse o sítio <https://developer.marvel.com/> e crie uma inscrição gratuita nesse portal.

Esse portal apresenta uma API pública para leitura de informações das revistas da Marvel.

Depois acesse o endereço abaixo - <https://developer.marvel.com/docs>

Examine a documentação da API. Veja que ela construída ao longo de recursos (substantivos). Isso é uma melhor prática REST e será estudada mais a frente na nossa disciplina.

Vamos consumir os dados dessa API. Para isso você irá precisar usar os códigos gerados para a sua conta, disponível aqui - <https://developer.marvel.com/account>

Você precisará usar a sua chave pública e privada conforme explicado nesse endereço

<https://developer.marvel.com/documentation/authorization>

Para gerar o código MD5, use essa ferramenta de auxílio aqui.

<https://www.md5hashgenerator.com/>

Com os códigos gerados, vamos fazer os testes para examinar os dados do Capitão América.

Crie uma nova requisição GET no Postman com os seguintes parâmetros.

Método GET

URL: https://gateway.marvel.com:443/v1/public/characters/1009220

apikey: A sua chave pública gerada no Developer Portal

hash: Código MD5 gerado mais acima.

ts: O valor de timestamp usado para gerar a chave MD5.

Interface gráfica do usuário, Aplicativo, Tabela

Descrição gerada automaticamente

Após realizar os testes básicos, faça um suíte de chamadas para responder os seguintes aspectos

1. Listagem de todos os personagens da Marvel.
2. Listagem das histórias que o capitão América participou.
3. Listagem das revistas que o capitão América participou.
4. Introduza testes de contrato sobre cada uma dessas APIs.
5. Explique o que cada um dos seguintes cabeçalhos de resposta da chamada de uma API da Marvel representa.



**Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente**

Dica: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers>

* Content-Encoding: Usado para especificar o algoritmo de compressão.
* ETag: É um validador, uma string única identificando a versão do recurso;
* Content-type: Indica o tipo de mídia do recurso.
* Date: Informa a data da requisição;
* Connection: Controla se uma conexão de rede continua ou não aberta após o término da transação atual;
* Transfer-Encoding: Especifica a forma de encoding usado para segurança de transferência.

**PARTE 4 – Criação e Consumo de uma API**

Nesse exercício vamos criar uma API com o recurso de servidor Mock Json-Server.   
Para usá-lo, você deve:

Passo 1. Instalar a biblioteca JSON-Server com o comando

npm install -g json-server

Passo 2. Criar um arquivo db.json com os recursos desejados.

{

"posts": [

{ "id": 1, "title": "HTTP-Introducao", "author": "João Silva" },

{ "id": 2, "title": "HTTP-Avançado", "author": "João Silva" }

],

"profile": { "name": "typicode" }

}

Passo 3. Rodar o servidor com o comando.

json-server --watch db.json

Depois teste o seu servidor do seu navegador chamado as URLs.

<http://localhost:3000/posts/1> ou

<http://localhost:3000/posts>

1. Com essa receita crie uma suíte de testes com o seu Postman para as operações GET, POST, PUT e DELETE.

Cada teste deve verificar se:

* O tempo de resposta está abaixo de 100ms e
* O código de resposta correto foi retornado.

Dicas para o exercício:

**Modelo de teste de contrato para o comando get**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Observe que nem todo código de resposta deve ser 200. Veja mais dicas aqui - https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status/200

**Envio de uma solicitação POST**

Uma solicitação POST pode simplesmente ser enviada por meio de um corpo de solicitação.

Por exemplo, o corpo abaixo irá criar um novo objeto de autor em nosso banco de dados.

{"title" : "Aprenda Java em 1001 Noites" , "author" : "Gregory Salis" }

Não se esqueça de enviar a solicitação para a URL correta:

<http://localhost:3000/posts>

Como você pode ver, o corpo da solicitação não precisa passar um ID. O servidor JSON criará automaticamente um novo ID para seu objeto.

Finalmente, não se esqueça de passar os cabeçalhos corretos para a solicitação. Como estamos lidando com uma API JSON, precisamos enviar o cabeçalho.

Content-Type: application/json

**Envio de uma solicitação PUT**

O envio de uma solicitação PUT é muito semelhante ao envio de uma solicitação POST. Mas primeiro devemos especificar o ID do objeto que precisa ser atualizado.

Vamos atualizar o objeto de autor original com ID 1 usando o URL: <http://localhost:3000/posts/1>

O corpo irá modificar o título do livro de código 1

{"title" : "Aprenda Java em 1002 Noites" , "author" : "Gregory Salis" }

**PARTE 5 – Desenho, criação e Consumo de uma API**

1. Projete uma API REST simples para cadastro de cliente com o JSON Server. O seu cadastro deve suportar as seguintes operações:

* Leitura de clientes
* Leitura de um cliente
* Inclusão de um cliente
* Atualização de um cliente
* Atualização parcial de um cliente
* Remoção lógica de um cliente do banco de dados
* Remoção física de um cliente
* Testes de contrato de cada um dessas funções.

O seu esquema de dados do cliente deve suportar:

* Nome
* Sobrenome
* CPF
* Telefone
* Endereço Residencial
* CEP

O seu entregável deve conter:

- Esquema de dados do seu banco de dados

- Rotas

- Código dos testes de contrato de APIs usado no seu Postman