📚 Postman

Seção 1: Introdução

- O que é o Postman?

O Postman é uma ferramenta popular para testes de APIs, amplamente utilizada por desenvolvedores e equipes de tecnologia. Ele permite realizar requisições HTTP (como GET, POST, PUT, DELETE) de forma simples e organizada, facilitando a integração com APIs e a automação de testes.

Principais recursos do Postman:

- Interface intuitiva para criação de requests.
- Suporte a variáveis internas e globais, evitando o uso de dados fixos (hard coded).
- Permite testes automatizados com JavaScript, incluindo validações e verificações de performance.
- Ideal para uso individual e colaborativo, com recursos de compartilhamento de coleções e workspaces entre membros da equipe.
- Suporte para boas práticas de testes de API, como:
  - Verificação de formato de dados retornado
  - Medição de tempo de resposta (performance)
  - Manutenção de ambientes separados (dev, prod, etc.)

Popularidade:

- Utilizado por mais de 500 mil empresas no mundo.
- 98% das empresas da lista Fortune 500 usam o Postman.

Formas de acesso:

1. Versão local (desktop) – para Windows, macOS e Linux.
2. Versão web – acessível direto do navegador.
3. Extensão para Google Chrome – com recursos limitados.

Integrações:

O Postman pode ser integrado com várias ferramentas, como o Microsoft Power Automate, permitindo que chamadas e testes de API sejam automatizados como parte de fluxos de trabalho maiores.

- O que é uma API?

Uma API (Application Programming Interface, ou Interface de Programação de Aplicativos) é um conjunto de regras e protocolos que permite a comunicação entre diferentes aplicações. APIs são fundamentais no desenvolvimento de sistemas modernos, pois raramente um software funciona de forma completamente isolada.

Exemplo de uso prático:

Ao criar uma conta em um site utilizando o Gmail, o site acessa a API do Google para obter dados como nome e e-mail do usuário, facilitando a integração entre os sistemas.

Função das APIs:

- Estabelecer a comunicação entre frontend e backend.
- Definir como os dados devem ser enviados e recebidos.
- Garantir que ambas as partes "falem a mesma língua".
- Assegurar a segurança da comunicação, especialmente em aplicações web.

Segurança:

APIs geralmente exigem autenticação, muitas vezes por meio de chaves de API (API Keys), para proteger os dados e garantir que apenas usuários autorizados tenham acesso.

Requisições típicas:

- **GET:** para buscar informações.
- **POST:** para enviar ou cadastrar dados.

Documentação da API:

Toda API bem estruturada conta com uma documentação oficial, que descreve:

- Os endpoints disponíveis.
- Os métodos permitidos (GET, POST, etc.).
- As regras de autenticação e segurança.
- Os limites de requisições.
- Exemplos de uso prático.

A documentação facilita o uso correto da API, tanto por desenvolvedores que desejam integrá-la quanto por aqueles que estão criando suas próprias APIs.

Seção 2: Primeiros Passos no Postman

- Primeiros Passos:

Explorando APIs públicas

Na aba Home do Postman, é possível visualizar workspaces, APIs populares e categorias de integração (como redes sociais, pagamentos, inteligência artificial e bancos de dados). Muitas dessas APIs, como as do Twitter, Instagram (Meta) e PayPal, já possuem integração direta com o Postman, facilitando a exploração e testes.

Workspaces

- Workspaces são áreas de trabalho dentro do Postman, ideais para separar projetos distintos.
- Cada workspace pode ter configurações e níveis de acesso diferentes, permitindo colaboração em equipe ou uso individual.
- Exemplo criado: Workspace chamado *"Primeira API"* com acesso pessoal.

Ambientes

- Os ambientes permitem definir variáveis específicas (como URLs, tokens, parâmetros), facilitando a customização das requests.
- É possível criar múltiplos ambientes dentro de um mesmo workspace, permitindo testar diferentes contextos (como desenvolvimento, homologação e produção) de forma isolada.

Collections

- Collections funcionam como pastas que organizam as requisições dentro de um projeto.
- Dentro de uma collection, podem ser criadas múltiplas requests agrupadas logicamente.
- Exemplo criado: Collection chamada *"Requisições - Exemplo 1"*.

Requests

As requests são os testes feitos contra uma API, e podem ser configuradas com:

- Método HTTP (GET, POST, PUT, PATCH, DELETE etc.).
- URL da API.
- Parâmetros de segurança e autenticação.
- Cabeçalhos (Headers), corpo da requisição (Body) e scripts de pré-teste e pós-teste.
- Possibilidade de rodar testes automatizados diretamente após a requisição.

- Request GET:

Realizamos a primeira requisição (request) utilizando o Postman, com o método GET acessando a API pública e aberta chamada ReqRes (`https://reqres.in`), que é amplamente utilizada para testes e simulações de chamadas REST.

Uso da API ReqRes

- A API ReqRes oferece endpoints para teste com dados reais e simulações de erros.
- Os principais métodos disponíveis são:
  - `GET` lista de usuários (`/api/users?page=2`)
  - `GET` usuário específico
  - Simulação de usuário não encontrado
  - Outros métodos como `POST` , `PUT` , `DELETE` também estão disponíveis

Execução no Postman

- O endereço completo (`https://reqres.in/api/users?page=2`) foi inserido na URL da requisição.
- Ao clicar em Send, a API retornou corretamente os dados em formato JSON, listando usuários da página 2.
- O parâmetro de página **(** `page` **)** foi identificado automaticamente pelo Postman e pode ser alterado dinamicamente.

Análise dos componentes da requisição

- **Params:** exibe e permite edição dos parâmetros da URL (ex: `page=2`).
- **Authorization:** neste caso, não foi necessário, pois a API é pública. Porém, em APIs privadas, pode ser exigido API Key ou usuário/senha.
- **Headers:** a requisição simples não exigiu headers adicionais.
- **Body:** não utilizado em métodos `GET` , mas será relevante para métodos como `POST` ou `PUT` .
- **Pre-request Script:** espaço para adicionar scripts em JavaScript executados antes da requisição.
- **Tests:** área onde podem ser criados scripts para verificar respostas, status, tempo, volume de dados e outros critérios (será explorada em aulas futuras).
- **Settings:** permite ajustes de segurança e comportamento padrão da requisição.

Visualização da Resposta

- A resposta foi exibida em formato JSON, com opção de alternar entre visualização bruta (RAW) e formatada (Pretty).
- É possível visualizar também cookies, headers e resultados de testes (caso configurados).

Boas práticas apresentadas

- Nomear as requests adequadamente para manter organização, principalmente em coleções com muitas requisições.
- Exemplo de nomeação: *"GET - ReqRes (Lista de Usuários)"*.
- Utilizar Collections para agrupar requests relacionadas e facilitar o gerenciamento do projeto.

- Request POST:

Criamos uma requisição do tipo POST utilizando o Postman, acessando novamente a API pública de testes ReqRes (`https://reqres.in`). Esse tipo de requisição é utilizado para enviar dados e criar novos registros.

Passo a passo da criação da request:

1. Criada uma nova requisição com o nome "POST - ReqRes".
2. O método foi alterado para POST.
3. Foi utilizado o endpoint `/api/users`, conforme a documentação da API ReqRes.
4. A URL completa da requisição ficou:
   `https://reqres.in/api/users`

Corpo da Requisição (Body)

- Para o método POST, é necessário enviar dados no corpo da requisição.
- Foi utilizado o formato JSON, com os seguintes dados de exemplo:

  ```
  1 json
  ```

  Copiar código

  ```
  {   "name": "test user",   "job": "user" }
  ```

- A aba Body foi configurada como raw, com o tipo JSON selecionado.

Execução e Resposta

- Após enviar a requisição, a resposta retornou com o status 201 (Created), indicando que o recurso foi criado com sucesso.
- A resposta da API incluiu:
  - Os dados enviados: `name` e `job`
  - Um `id` gerado automaticamente
  - A data de criação (`createdAt`)

Considerações adicionais

- O status code 201 indica sucesso na criação do recurso.

- A estrutura da resposta foi apresentada em formato JSON, visualizado em modo formatado (Pretty).
- A criação de requests nomeadas é uma boa prática de organização, especialmente em coleções com múltiplas requisições.

- Request PATCH:

Esse método permite atualizar parcialmente os dados de um recurso já existente, ao contrário do método POST, que cria um novo registro.

Contexto

- O método **GET** é utilizado para buscar dados.
- O método **POST** é utilizado para criar dados.
- O método **PATCH** é utilizado para atualizar dados existentes, sem sobrescrevê-los por completo.

Configuração da request

- Uma nova requisição foi criada com o nome "PATCH - ReqRes".
- Foi selecionado o método PATCH.
- Endpoint utilizado:
  `https://reqres.in/api/users/2`
  (representando a atualização dos dados do usuário com ID 2).

Envio de dados (Body)

- A aba Body foi configurada como raw, com o tipo JSON.
- O conteúdo enviado foi:

  ```
  1 json
  ```

  Copiar código

  `{ "name": "test user", "job": "admin" }`
- Essa informação atualiza apenas os campos indicados (`job` , neste caso), sem remover ou alterar os demais.

Resposta da requisição

- A resposta da API retornou o código 200 (OK), indicando que a atualização foi realizada com sucesso.
- O corpo da resposta incluiu:
  - Os dados atualizados: `name` e `job`

- Um campo `updatedAt`, com a data da modificação

Observações importantes

- O método PATCH é semelhante ao POST na estrutura de envio, mas é utilizado para atualização e não para criação.
- A documentação da API especifica que o status esperado para sucesso com PATCH é 200.
- A requisição foi salva com o nome atualizado e organizada dentro da collection, seguindo as boas práticas de nomenclatura no Postman.

Seção 3: Estudo de Caso - API OpenWeatherMap

Esta seção apresenta um estudo de caso prático utilizando a API OpenWeatherMap, com o objetivo de aplicar os conhecimentos adquiridos sobre métodos HTTP no Postman em uma situação real. O desafio proposto é realizar uma requisição GET para consultar as condições climáticas da cidade de Londres em tempo real, com retorno em português.

Inicialmente, o estudante é orientado a criar uma conta gratuita na plataforma OpenWeatherMap. Após o registro, é necessário gerar uma chave de API (API Key), que será utilizada para autenticação nas requisições. A documentação da API informa que pode haver um pequeno atraso na ativação da chave, o que deve ser considerado no momento dos testes.

Com a conta criada e a chave ativa, o próximo passo é acessar a documentação da API, localizar o endpoint adequado para obtenção de dados climáticos atuais e identificar os parâmetros necessários. A abordagem utilizada envolve a consulta por nome da cidade, utilizando o parâmetro `q=London` combinado com a chave de API (`appid=CHAVE`).

A requisição é criada no Postman dentro de uma collection organizada, utilizando o método GET. Inicialmente, os dados retornados estão em inglês, pois este é o idioma padrão da API. Para que os dados sejam retornados em português, é necessário incluir o parâmetro adicional `lang=pt_br`, conforme descrito na documentação.

Após configurar a URL, os parâmetros e a chave de autenticação corretamente, a requisição é executada com sucesso. A resposta inclui informações como descrição do céu, temperatura, sensação térmica, umidade, visibilidade, vento, entre outros. A descrição do clima é exibida em português, validando a funcionalidade do parâmetro de idioma.

Seção 4: Variáveis no Postman

Nesta seção são abordadas as variáveis no Postman, recurso fundamental para tornar as requisições mais organizadas, reutilizáveis e fáceis de manter. Até este ponto, todas as URLs e parâmetros utilizados estavam escritos diretamente nas requisições. O uso de variáveis permite substituir esses valores fixos por nomes de variáveis, tornando o processo mais eficiente, especialmente em projetos com muitas requests.

Primeiramente, são introduzidas as variáveis locais, que são definidas no escopo de uma coleção específica. Para criá-las, o usuário acessa a aba "Variables" dentro da coleção e define o nome e valor atual da variável (por exemplo, uma URL base). Ao utilizar `{{nomeDaVariável}}` nas requisições, o Postman substitui automaticamente pelo valor definido. Isso permite, por exemplo, que a alteração de uma URL seja feita em apenas um lugar, refletindo em todas as requisições da coleção que utilizam essa variável.

Na sequência, são apresentadas as variáveis globais, que funcionam em todas as coleções dentro do Postman. Elas são criadas acessando a aba de configurações globais e servem para armazenar valores que precisam ser compartilhados em diferentes contextos, como uma mesma URL base utilizada em múltiplas coleções. Assim como nas variáveis locais, é essencial salvar a variável após sua definição para que ela possa ser reconhecida nas requisições.

A aula finaliza explicando quando utilizar cada tipo de variável. As variáveis locais são indicadas para valores específicos de uma coleção, enquanto variáveis globais devem ser usadas quando os valores precisam ser reutilizados em diferentes coleções. A adoção de variáveis é considerada uma boa prática para tornar os testes mais escaláveis e reduzir a duplicação de dados dentro do ambiente de testes.