Apostila Aula 2: Fundamentos de HTTP e Protocolos

Informações Gerais

- **Curso**: APIs REST e SOAP
- **Aula**: 2 de 20
- **Carga Horária**: 4 horas
- **Objetivo**: Entender os fundamentos do protocolo HTTP, base para APIs REST.
- **Conteúdo**:
- Estrutura do protocolo HTTP (métodos, status codes, cabeçalhos).
- Conceitos de cliente-servidor.
- Introdução a URLs, URIs e endpoints.
- Atividade prática: Simulação de requisições HTTP usando ferramentas como Postman.

---

1. Estrutura do Protocolo HTTP

O que é o Protocolo HTTP?

O **HTTP** (*HyperText Transfer Protocol*) é um protocolo de comunicação utilizado para transferir dados na World Wide Web. Ele é a base para a troca de informações entre clientes (como navegadores ou aplicativos) e servidores (que hospedam sites ou APIs). Criado em 1991 por Tim Berners-Lee, o HTTP é um protocolo de camada de aplicação que opera sobre o TCP/IP, garantindo a comunicação confiável entre sistemas distribuídos.

O HTTP é fundamental para APIs REST, pois define como as requisições e respostas são estruturadas. Ele é um protocolo **sem estado** (*stateless*), o que significa que cada requisição é independente e não mantém informações sobre requisições anteriores, a menos que mecanismos como cookies ou tokens sejam usados.

Estrutura de uma Mensagem HTTP

Uma mensagem HTTP, seja uma requisição ou uma resposta, é composta por três partes principais: 1. **Linha inicial**: Define o tipo de mensagem (requisição ou resposta) e informações básicas. 2. **Cabeçalhos (Headers)**: Fornecem metadados

sobre a requisição ou resposta. 3. **Corpo (Body)**: Contém os dados reais, como JSON ou XML, quando aplicável.

Linha Inicial

- **Requisição**: Contém o **método HTTP**, a **URL** e a **versão do protocolo**. Exemplo: `GET /api/usuarios/123 HTTP/1.1`
- **GET**: Método HTTP.
- **/api/usuarios/123**: URL do recurso.
- **HTTP/1.1**: Versão do protocolo.
- **Resposta**: Contém a **versão do protocolo**, o **código de status** e a **mensagem de status**. Exemplo: `HTTP/1.1 200 OK`
- **HTTP/1.1**: Versão do protocolo.
- **200**: Código de status.
- **OK**: Mensagem de status.

Cabeçalhos (Headers)

Os cabeçalhos são pares chave-valor que fornecem informações adicionais sobre a requisição ou resposta. Exemplos: - **Host**: Especifica o domínio do servidor (ex.: `Host: api.exemplo.com`). - **Content-Type**: Define o formato do corpo (ex.: `Content-Type: application/json`). - **Authorization**: Envia credenciais, como tokens (ex.: `Authorization: Bearer abc123`). - **Accept**: Indica os formatos aceitos pelo cliente (ex.: `Accept: application/json`).

Exemplo de cabeçalhos em uma requisição:

```
Host: api.exemplo.com
User-Agent: Mozilla/5.0
Content-Type: application/json
Accept: application/json
```

Corpo (Body)

O corpo contém os dados enviados ou recebidos, como um payload JSON em uma API REST. Nem toda requisição ou resposta possui corpo (ex.: requisições GET geralmente não têm corpo). Exemplo de corpo em uma requisição POST:

```
{
 "nome": "Maria Silva",
```

```
 "email": "maria@exemplo.com"
}
```

Métodos HTTP

Os métodos HTTP definem a ação que o cliente deseja realizar no recurso identificado pela URL. Os principais métodos usados em APIs REST são:

1. **GET**: Recupera dados de um recurso. Exemplo: `GET /api/usuarios/123`
2. Uso: Buscar informações, como a lista de usuários.

   Características: Seguro (não altera dados) e idempotente (múltiplas chamadas produzem o mesmo resultado).

   POST: Cria um novo recurso. Exemplo: `POST /api/usuarios`

5. Uso: Enviar dados para criar um novo usuário.

   Características: Não é seguro nem idempotente.

   PUT: Atualiza um recurso existente ou cria um novo recurso em um local específico. Exemplo: `PUT /api/usuarios/123`

8. Uso: Atualizar os dados de um usuário.

   Características: Idempotente (múltiplas chamadas produzem o mesmo resultado).

   DELETE: Remove um recurso. Exemplo: `DELETE /api/usuarios/123`

11. Uso: Excluir um usuário.

    Características: Idempotente.

    PATCH: Atualiza parcialmente um recurso. Exemplo: `PATCH /api/usuarios/123`

14. Uso: Alterar apenas o email de um usuário.

    Características: Não é necessariamente idempotente.

OPTIONS: Retorna os métodos HTTP suportados pelo servidor para um recurso. Exemplo: `OPTIONS /api/usuarios`

17. Uso: Verificar quais ações são permitidas.

Códigos de Status HTTP

Os códigos de status indicam o resultado de uma requisição. Eles são divididos em cinco classes: 1. **1xx (Informativo)**: Indica que a requisição foi recebida e está sendo processada (raro em APIs). - Exemplo: `100 Continue`. 2. **2xx (Sucesso)**: A requisição foi bem-sucedida. - `200 OK`: Requisição bem-sucedida. - `201 Created`: Recurso criado com sucesso (comum em POST). - `204 No Content`: Requisição bem-sucedida, mas sem corpo na resposta (comum em DELETE). 3. **3xx (Redirecionamento)**: O cliente precisa tomar ações adicionais. - `301 Moved Permanently`: Recurso movido para uma nova URL. - `304 Not Modified`: Recurso não foi alterado (usado com cache). 4. **4xx (Erro do Cliente)**: A requisição contém erros. - `400 Bad Request`: Requisição malformada. - `401 Unauthorized`: Autenticação necessária. - `403 Forbidden`: Acesso negado. - `404 Not Found`: Recurso não encontrado. 5. **5xx (Erro do Servidor)**: O servidor falhou ao processar a requisição. - `500 Internal Server Error`: Erro genérico no servidor. - `503 Service Unavailable`: Servidor temporariamente indisponível.

Exemplo de resposta HTTP:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "id": 123,
 "nome": "João Silva",
 "email": "joao@exemplo.com"
}
```

Versões do HTTP

- **HTTP/1.1**: A versão mais comum, amplamente usada em APIs REST. Suporta conexões persistentes e pipelining.

- **HTTP/2**: Introduzido em 2015, oferece melhor desempenho com multiplexação e compressão de cabeçalhos.
- **HTTP/3**: Baseado em UDP (em vez de TCP), usa QUIC para maior velocidade e segurança. Ainda em adoção em 2025.

HTTPS

O **HTTPS** (*HTTP Secure*) é a versão segura do HTTP, que usa criptografia TLS/SSL para proteger a comunicação. Em APIs, o HTTPS é essencial para garantir a confidencialidade e integridade dos dados, especialmente em aplicações que lidam com informações sensíveis, como dados de pagamento ou pessoais.

---

2. Conceitos de Cliente-Servidor

Modelo Cliente-Servidor

O modelo cliente-servidor é a base da arquitetura da web e das APIs REST. Nesse modelo: - **Cliente**: Inicia a comunicação enviando uma requisição. Exemplos de clientes incluem navegadores, aplicativos móveis ou ferramentas como Postman. - **Servidor**: Responde às requisições, fornecendo dados ou executando ações. Exemplos incluem servidores web (Apache, Nginx) ou servidores de APIs.

A comunicação ocorre em um ciclo: 1. O cliente envia uma requisição HTTP (ex.: GET /api/usuarios). 2. O servidor processa a requisição, acessando bancos de dados ou outros serviços, se necessário. 3. O servidor retorna uma resposta HTTP com os dados solicitados ou um código de status.

Características do Modelo

- **Separação de responsabilidades**: O cliente lida com a interface do usuário, enquanto o servidor gerencia a lógica de negócios e os dados.
- **Escalabilidade**: Servidores podem ser escalados horizontalmente (adicionando mais máquinas) para lidar com mais clientes.
- **Descentralização**: Clientes e servidores podem estar em locais diferentes, conectados pela internet.
- **Sem estado**: Em APIs REST, cada requisição é independente, o que simplifica o design do servidor.

Exemplos Práticos

- **Navegador e Site**: Um navegador (cliente) faz uma requisição GET para um servidor web, que retorna uma página HTML.
- **Aplicativo e API**: Um aplicativo móvel faz uma requisição POST para uma API REST, que cria um novo registro no servidor.
- **IoT**: Um dispositivo inteligente (cliente) envia dados de temperatura para um servidor via API.

Papel em APIs REST

Em APIs REST, o modelo cliente-servidor é implementado usando HTTP: - O cliente (ex.: aplicativo) envia requisições para endpoints específicos. - O servidor (ex.: API hospedada em um servidor cloud) processa a requisição e retorna uma resposta em JSON ou XML. - A arquitetura REST enfatiza a separação clara entre cliente e servidor, permitindo que cada um evolua independentemente.

---

3. Introdução a URLs, URIs e Endpoints

O que é uma URI?

Uma **URI** (*Uniform Resource Identifier*) é uma string que identifica um recurso na internet ou em uma rede. Ela é composta por: - **Esquema**: O protocolo usado (ex.: `http`, `https`). - **Autoridade**: O domínio ou endereço do servidor (ex.: `api.exemplo.com`). - **Caminho**: O caminho para o recurso (ex.: `/usuarios/123`). - **Query (opcional)**: Parâmetros adicionais (ex.: `?id=123`). - **Fragmento (opcional)**: Uma parte específica do recurso (ex.: `#secao1`).

Exemplo de URI:

```
https://api.exemplo.com/usuarios/123?fields=nome,email
```

- **https**: Esquema.
- **api.exemplo.com**: Autoridade.
- **/usuarios/123**: Caminho.
- **?fields=nome,email**: Query.

O que é uma URL?

Uma **URL** (*Uniform Resource Locator*) é um tipo específico de URI que não apenas identifica um recurso, mas também indica como acessá-lo. Todas as URLs são URIs, mas nem todas as URIs são URLs. Em APIs REST, as URLs são usadas para identificar recursos específicos.

Exemplo de URL:

```
https://api.exemplo.com/usuarios/123
```

O que é um Endpoint?

Um **endpoint** é uma URL específica que representa um recurso ou uma ação em uma API. Cada endpoint é associado a um método HTTP e define uma funcionalidade específica. Por exemplo: - `GET /api/usuarios`: Lista todos os usuários. - `POST /api/usuarios`: Cria um novo usuário. - `GET /api/usuarios/123`: Retorna os detalhes do usuário com ID 123.

Boas Práticas para URLs e Endpoints

- **Use substantivos para recursos**: Ex.: `/usuarios` em vez de `/getUsuarios`.
- **Evite verbos nas URLs**: Use métodos HTTP para indicar ações (ex.: `POST /usuarios` para criar).
- **Estruture hierarquicamente**: Ex.: `/usuarios/123/pedidos` para pedidos de um usuário.
- **Use parâmetros de query para filtros**: Ex.: `/usuarios?status=ativo`.
- **Evite extensões de arquivo**: Ex.: `/usuarios` em vez de `/usuarios.json`.

Exemplo de design de endpoints para uma API de e-commerce: - `GET /produtos`: Lista todos os produtos. - `GET /produtos/456`: Detalhes do produto com ID 456. - `POST /pedidos`: Cria um novo pedido. - `PUT /pedidos/789`: Atualiza o pedido com ID 789.

---

4. Atividade Prática: Simulação de Requisições HTTP Usando Ferramentas como Postman

Objetivo da Atividade

Simular requisições HTTP para entender como o protocolo funciona na prática, utilizando a ferramenta Postman para interagir com APIs públicas.

Instruções

1. **Instalação do Postman**:
2. Baixe e instale o Postman (disponível em https://www.postman.com).
3. Crie uma conta ou use a versão offline, se preferir.
4. **Exploração de APIs Públicas**:
5. Escolha uma API pública para testar. Sugestões:
   - **JSONPlaceholder**: Uma API REST falsa para testes (https://jsonplaceholder.typicode.com).
   - **ReqRes**: Outra API REST para simulações (https://reqres.in).
6. Acesse a documentação da API escolhida para entender os endpoints disponíveis.
7. **Realização de Requisições**:

   GET: Faça uma requisição GET para listar recursos. Exemplo: `GET https://jsonplaceholder.typicode.com/users`
   - Analise a resposta, observando o código de status, cabeçalhos e corpo.

   POST: Crie um novo recurso com uma requisição POST. Exemplo: ``` POST https://jsonplaceholder.typicode.com/users Content-Type: application/json

   { “name”: “Ana Costa”, “email”: “ana@exemplo.com” } - Verifique o código de status (esperado: 201 Created). - **PUT**: Atualize um recurso existente. Exemplo: PUT https://jsonplaceholder.typicode.com/users/1 Content-Type: application/json

   { “name”: “Ana Costa Atualizada”, “email”: “ana@exemplo.com” } - **DELETE**: Exclua um recurso. Exemplo: DELETE https://jsonplaceholder.typicode.com/users/1 `` - Verifique o código de status (esperado: 200 OK ou 204 No Content). 4. **Análise de Cabeçalhos**: - Observe os cabeçalhos das respostas, comoContent-Type, ServereDate. - Tente adicionar cabeçalhos personalizados na requisição, comoAuthorization`. 5. **Relatório**: - Documente as requisições realizadas, incluindo: - Método HTTP usado. - URL do endpoint. - Cabeçalhos enviados e recebidos. - Corpo da requisição (se aplicável) e da resposta. - Código de status recebido. - Escreva uma breve

análise (200 palavras) sobre o que foi aprendido, destacando: - Como os métodos HTTP afetam o comportamento da API. - A importância dos códigos de status. - O papel dos cabeçalhos na comunicação.

Exemplo de Requisição no Postman

1. Abra o Postman e crie uma nova requisição.
2. Configure a requisição:
3. Método: GET
4. URL: `https://jsonplaceholder.typicode.com/posts`
5. Cabeçalhos: Adicione `Accept: application/json`.
6. Envie a requisição e analise a resposta: `json [ { "userId": 1, "id": 1, "title": "sunt aut facere repellat provident occaecati...", "body": "quia et suscipit\nsuscipit recusandae consequuntur..." }, ... ]`
7. Observe o código de status (`200 OK`) e os cabeçalhos da resposta.

Entrega

Cada aluno deve entregar um relatório individual com: - Capturas de tela das requisições realizadas no Postman. - Descrição de cada requisição (método, URL, cabeçalhos, corpo, código de status). - Análise de 200 palavras sobre o aprendizado.

Conclusão

Esta aula abordou os fundamentos do protocolo HTTP, que é a base para APIs REST. Compreender a estrutura das mensagens HTTP, os métodos, os códigos de status e os cabeçalhos é essencial para desenvolver e consumir APIs de forma eficaz. O modelo cliente-servidor foi explorado para contextualizar a comunicação entre sistemas, e os conceitos de URLs, URIs e endpoints foram apresentados para esclarecer como os recursos são identificados. A atividade prática com Postman permitiu aos alunos experimentarem requisições HTTP em um ambiente real, reforçando os conceitos aprendidos.

Próximos passos: Na Aula 3, introduziremos os princípios do REST, explorando como o HTTP é aplicado em APIs RESTful e analisando exemplos práticos.