CENTRO PAULA SOUZA
ETEC PROF MARIA CRISTINA MEDEIROS
Técnico em Informática para Internet Integrado ao Ensino Médio

Luana Garcia Mathias

Atividade teórica: Conceitos e funcionamento de APIs RESTful em PHP sem uso de frameworks

Ribeirão Pires
2025

Luana Garcia Mathias

Atividade teórica: Conceitos e funcionamento de APIs RESTful em PHP sem uso de frameworks

Trabalho para o curso técnico em Informática para Internet Integrado ao Ensino Médio da ETEC Prof. Maria Cristina Medeiros, orientado pelo Prof. Anderson Vanin, como requisito parcial para obtenção de nota em Informática para Internet.

Ribeirão Pires
2025

1 INTRODUÇÃO

Atividade teórica: Conceitos e funcionamento de APIs RESTful em PHP sem uso de frameworks”

1.1 O que é uma API e para que serve

1.1.1 Definição de API (Application Programming Interface)

É uma interface que permite a comunicação entre dois sistemas (programas, aplicações ou partes de um mesmo sistema).

Fácil de entender da seguinte forma: atua como “garçom”: o cliente faz um “pedido” a API (garçom) processa e devolve uma resposta para o cliente (ex: comida).

1.1.2 Objetivos principais

- Padroniza a troca de dados entre front e o back-end.
- Ocultar detalhes internos de implementação: o cliente não precisa saber como está a “cozinha” ou como o alimento é feito, apenas como pedir a comida, ou seja, o cliente não precisa saber como o servidor armazena ou processa dados, apenas como solicitar recursos
- Facilitar a manutenção e a escalabilidade

1.2 Diferença entre API REST e SOAP

1.2.1 API REST (Representational State Transfer)

- Baseia-se em princípios arquiteturais que usam o protocolo HTTP de forma simples e direta.
- Troca de dados, geralmente, em formato JSON (pode ser também XML, mas JSON é melhor por questões de popularidade).
- Utiliza URLs para representar recursos
- Não possui um padrão rígido de cabeçalhos ou envelopes
- É considerada “stateless”: cada requisição ao servidor deve conter todas as informações necessárias;

1.2.2 API SOAP (Simple Object Access Protocol)

- Padrão mais antigo e complexo, baseado em XML para comunicação.
- Usa envelopes SOAP que envolvem cada mensagem, definindo cabeçalhos (headers) e corpo (body) em XML.
- Costuma empregar WSDL para descrever as operações disponíveis e seus tipos de dados.
- Exige protocolos adicionais como WS-Security, WS-ReliableMessaging, tornando a configuração mais burocrática.
- Muito usado em sistemas corporativos que exigem transações complexas, segurança avançada e padronizações rígidas.

1.3 Principais diferenças resumidas

Formato de dados: *REST → JSON; SOAP → apenas XML*.

Complexidade: REST → mais “leve” e flexível; SOAP → mais verboso e rígido.

Estado: REST → stateless; SOAP → pode manter estado (stateful) dependendo da implementação.

Curva de aprendizado: REST → relativamente simples; SOAP → demanda conhecer WSDL, namespaces XML e padrões WS.

2 Como funcionam as rotas em uma API RESTful

1.4 Conceito de rota (endpoint)

Cada rota é um caminho específico de URL que representa um recurso ou uma coleção de recursos.

Exemplo:

- Listar todos os clientes: `GET /clientes`
- Buscar cliente por ID: `GET /clientes/{id}`
- Criar novo cliente: `POST /clientes`
- Atualizar cliente existente: `PUT /clientes/{id}`
- Deletar cliente: `DELETE /clientes/{id}`

1.5 Mapeamento entre rotas e recursos

Recurso: entidade lógica da aplicação (ex.: cliente, produto, pedido).

Cada rota “aponta” para um recurso específico, identificando-os por um identificador único (ID).

1.6 Fluxo básico de uma requisição

- Cliente faz requisição para uma rota (por exemplo, `GET /produtos/10`).
- Servidor interpreta o método HTTP (GET, POST, PUT, DELETE) e a URL.
- O código da API seleciona a ação apropriada (ex.: buscar produto com ID 10).
- O servidor retorna uma resposta, geralmente em JSON, com os dados ou com a confirmação da operação.

1.7 Boas práticas de rotas RESTful

- Manter URLs sem verbos (ex.: use `/clientes` e não `/getClientes`).
- Usar nomes no plural para coleções (ex.: `/produtos`, `/pedidos`).
- Agrupar funcionalidades semelhantes (ex.: `/clientes/{id}/pedidos` para pedidos de um cliente específico).

- Versionamento da API via URL ou cabeçalho (ex.: `/v1/clientes` ou cabeçalho `Accept: application/vnd.minhaapi.v1+json`).

3 Os principais métodos HTTP (GET, POST, PUT, DELETE)

1.GET

Finalidade: Buscar/recuperar informação de um recurso.

Características:

- Sem corpo na requisição (body vazio).
- Parâmetros podem vir pela query string (ex.: `?page=2&limit=10`) ou pela própria rota (`/produtos/5`).
- É idempotente: várias requisições GET não alteram o estado do recurso.

2. POST

Finalidade: Criar um novo recurso.

Características:

- Envia dados no corpo da requisição, normalmente em JSON (`{ "nome": "Luana", "email": "Luana@ex.com" }`).
- Não é idempotente: duas requisições POST podem criar dois registros diferentes.

3. PUT

Finalidade: Atualizar um recurso existente por completo.

Características:

- Recebe todos os campos necessários para substituir o recurso (exemplo: `{ "id": 5, "nome": "Mathias", "email": "mathias@ex.com" }`).
- É idempotente: chamadas repetidas com os mesmos dados produzem o mesmo resultado.
- Caso o recurso não exista, algumas implementações de REST elegem criar um novo (mas normalmente se usa PATCH para atualização parcial).

4. DELETE

Finalidade: Apagar um recurso.

Características:

- Normalmente não leva corpo na requisição (body vazio).
- É idempotente: se o recurso não existir, a API pode retornar 404 ou 204 sem erro (dependendo da especificação).
- Ao deletar um registro, 204 (No Content) ou 404 (Not Found), conforme necessidade.

4 Estrutura básica de uma API em PHP puro (sem frameworks)

Arquivo único de entrada (ex.: `api.php`)

- Recebe todas as requisições (via `.htaccess` ou configuração do servidor) apontando para ele.
- Verifica em `$_SERVER['REQUEST_METHOD']` qual método HTTP foi utilizado.
- Extrai a rota acessada a partir de `$_SERVER['REQUEST_URI']` (removendo a parte fixa do caminho, se for necessário).

Fluxo de tratamento de rota e método

- Obter a URL solicitada e separar em segmentos (por exemplo: `/clientes/7` → `[“clientes”, “7”]`).
- Identificar qual recurso está sendo requisitado (ex.: “clientes”) e se há um ID (ex.: “7”).
- Baseado no método (GET, POST, PUT, DELETE) e na presença ou não do ID, direcionar para a função correspondente (função de listar, criar, atualizar ou deletar).

  Leitura e envio de dados em JSON

  Para requisições que enviam dados (POST, PUT), usar:

  ```
  php
  $jsonInput = file_get_contents('php://input');
  $dados = json_decode($jsonInput, true);
  ```

  Para responder ao cliente, configurar o header e imprimir JSON:

  ```
  php
  header('Content-Type: application/json');
  echo json_encode($resposta);
  ```

Armazenamento em arquivos JSON (substituindo banco de dados)

- File-based: armazenar cada recurso (ou coleção de recursos) em um arquivo `.json`, por exemplo, `clientes.json`.
- Ao criar um novo cliente:
    1. Ler o arquivo `clientes.json` (com `file_get_contents` e `json_decode`).
    2. Adicionar o novo registro no array de dados.
    3. Salvar de volta usando `file_put_contents` com `json_encode`.
- Similarmente, para editar/deletar: ler o array completo, modificar o elemento desejado e regravar o arquivo.

Exemplo resumido de rota em PHP puro

```
// Inicio do código
        php
          <?php
          // api.php
          header('Content-Type: application/json');
```

```
        // Captura método e rota
          $metodo = $_SERVER['REQUEST_METHOD'];
          $uri = $_SERVER['REQUEST_URI'];
```

```
          // Supondo que a API está em /api/api.php, ajustamos a URI
          $rota = str_replace('/api/api.php/', '', $uri);
          $partes = explode('/', $rota);
          $recurso = $partes[0]; // ex.: “clientes”
          $id = isset($partes[1]) ? intval($partes[1]) : null;
```

```
// Função genérica para ler arquivo JSON
function lerDados($arquivo) {
  if (!file_exists($arquivo)) {
    file_put_contents($arquivo, json_encode([]));
  }
  return json_decode(file_get_contents($arquivo), true);
}
```

```
// Função genérica para salvar dados no arquivo
function salvarDados($arquivo, $dados) {
  file_put_contents($arquivo,json_encode($dados,
JSON_PRETTY_PRINT));
}
if ($recurso === 'clientes') {
  $arquivoClientes = 'clientes.json';
  $clientes = lerDados($arquivoClientes);
  switch ($metodo) {
```

```
    case 'GET':
      if ($id !== null) {
        // Buscar cliente por ID
        foreach ($clientes as $c) {
          if ($c['id'] === $id) {
            echo json_encode($c);
            exit}}
        http_response_code(404);
        echo json_encode(['erro' => 'Cliente não encontrado.']);
      } else {
        // Listar todos os clientes
        echo json_encode($clientes);
      }
      break;
```

```
  case 'POST':
      $input = json_decode(file_get_contents('php://input'), true);
      // Gerar novo ID incremental
      $novoId = count($clientes) > 0 ? end($clientes)['id'] + 1 : 1;
      $input['id'] = $novoId;
      $clientes[] = $input;
      salvarDados($arquivoClientes, $clientes);
      http_response_code(201);
      echo json_encode($input);
      break;
```

```
            case 'PUT':
              if ($id === null) {
                http_response_code(400);
                echo json_encode(['erro' => 'ID é obrigatório para PUT.']);
                exit;
              }
              $input = json_decode(file_get_contents('php://input'), true);
              $atualizado = false;
              foreach ($clientes as &$c) {
                if ($c['id'] === $id) {
                  // Substitui todos os campos (exceto ID) pelo conteúdo de $input
                  $c['nome'] = $input['nome'] ?? $c['nome'];
                  $c['email'] = $input['email'] ?? $c['email'];
                  $atualizado = true;
                  break;
                }
              }
              if ($atualizado) {
                salvarDados($arquivoClientes, $clientes);
                echo json_encode(['mensagem' => 'Cliente atualizado com
sucesso.']);
              } else {
                http_response_code(404);
                echo json_encode(['erro' => 'Cliente não encontrado.']);
              }
              break;
```

```
        case 'DELETE':
            if ($id === null) {
              http_response_code(400);
              echo json_encode(['erro' => 'ID é obrigatório para DELETE.']);
              exit;
            }
            $encontrado = false;
            foreach ($clientes as $index => $c) {
              if ($c['id'] === $id) {
                array_splice($clientes, $index, 1);
                $encontrado = true;
                break;
              }
            }
            if ($encontrado) {
              salvarDados($arquivoClientes, $clientes);
              echo json_encode(['mensagem' => 'Cliente removido com sucesso.']);
            } else {
              http_response_code(404);
              echo json_encode(['erro' => 'Cliente não encontrado.']);
            }
            break;
          default:
            http_response_code(405);
            echo json_encode(['erro' => 'Método não permitido.']);
            break;
        }
      } else {
        http_response_code(404);
        echo json_encode(['erro' => 'Recurso não encontrado.']);
      }
```

5 Vantagens e desvantagens de usar arquivos JSON em vez de banco de dados

1.8 Vantagens de usar JSON (file-based)

1.8.1 Simplicidade de configuração

- Não é necessário instalar, configurar ou manter um SGBD (MySQL, PostgreSQL, etc.).
- Basta ter permissões de leitura e escrita no diretório onde estão os arquivos `.json`.

1.8.2 Portabilidade

- O projeto pode ser transportado facilmente: copiar e colar a pasta que contém os arquivos `.php` e `.json` já funciona.
- Ideal para protótipos, provas de conceito e pequenos projetos.

1.8.3 Formato humano-legível

- É fácil inspecionar manualmente (abrir o `.json` no editor de texto).
- Facilita debugging e alterações pontuais sem ferramentas específicas.

1.9 Desvantagens de usar JSON (file-based)

1.9.1 Escalabilidade limitada

- Ao crescer o volume de dados, o arquivo JSON pode ficar muito grande, gerando lentidão na leitura/escrita.
- Como cada operação de escrita regrava o arquivo inteiro, o consumo de I/O (disco) aumenta muito.

1.9.2 Concorrência e integridade

- Em sistemas com vários usuários simultâneos, há risco de sobrescrever dados: se dois processos lerem o JSON ao mesmo tempo e depois gravarem, um deles sobrescreverá as alterações do outro.
- Não há transações, bloqueio (lock) ou controle de concorrência sofisticado como em bancos de dados.

1.9.3 Limitações em consultas complexas

- Não há índices; buscar ou filtrar dados exige varrer o array inteiro (performance O(n)).
- Difícil executar consultas sofisticadas (ex.: “buscar clientes com mais de 30 anos e que compraram na última semana”) sem carregar tudo na memória e filtrar manualmente.

1.10 Segurança

- Dados sensíveis salvos diretamente em arquivos podem ficar expostos se a pasta não estiver protegida adequadamente.
- Bancos de dados geralmente têm camadas de segurança, permissões de usuário, criptografia em trânsito e em repouso.

REFERÊNCIAS

AGÊNCIA BRASIL. Alfabetização de crianças ainda é desafio para o Brasil. Agência Brasil. Disponível em: https://agenciabrasil.ebc.com.br . Acesso em: 18 mar. 2025.

MDN Web Docs. Métodos HTTP. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Reference/Methods

PHP.net. Manual Oficial do PHP. Disponível em: https://www.php.net/manual/pt_BR/index.php

Stack Overflow. Diferença entre REST e SOAP. Disponível em: https://stackoverflow.com/questions/19884261/difference-between-rest-and-soap

Fielding, R. T. Dissertation (2000) –Architectural Styles and the Design of Network-based Software Architectures. Disponível em: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm