UNISENAC-PELOTAS

FUNDAMENTOS DO BACKEND

PROTOCOLO HTTP: MÉTODOS, STATUS CODES, HEADERS, CONCEITOS DE API REST, ESTRUTURA E MANIPULAÇÃO.

PROF ALINE TIMM

OBJETIVOS DE APRENDIZAGEM:

1. Explicar o que é o protocolo HTTP e a função de seus principais métodos.
2. Diferenciar os códigos de status HTTP mais comuns.
3. Definir o que é uma API e descrever os princípios de uma API REST.
4. Estruturar e interpretar dados no formato JSON.
5. Conectar todos esses conceitos em um exemplo prático.

CENÁRIO CONTÍNUO:

Estamos digitalizando a Biblioteca do SENAC. Nosso sistema, até agora, tem duas partes:

A Interface do Usuário (o site que o bibliotecário usa no navegador) e o Arquivo Central (nosso servidor back-end, onde todos os dados são guardados e gerenciados).
Essas duas partes precisam conversar o tempo todo.
E os registros precisam ficar salvos em algum lugar, o banco de dados.

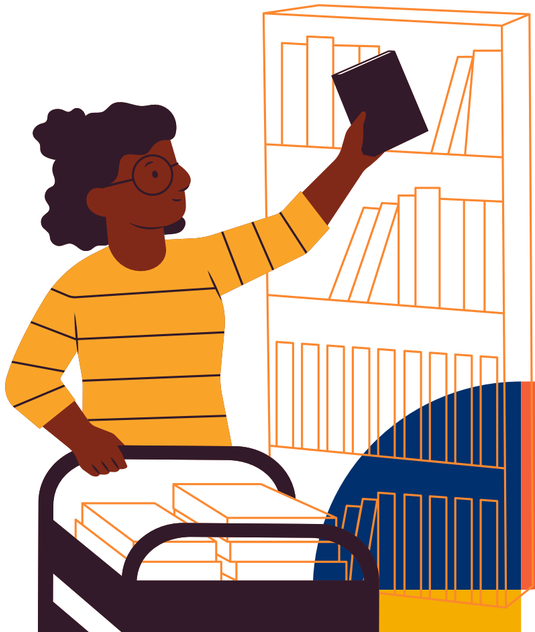
O PROTOCOLO HTTP:

VAMOS PENSAR EM QUALQUER ANALOGIA DIFERENTE DA ANALOGIA DO RESTAURANTE:

Imagine que a Interface do Usuário e o Arquivo Central são dois prédios que precisam trocar pacotes e mensagens constantemente.

Eles não podem simplesmente gritar um com o outro. Eles usam um serviço de entregas ultra-eficiente, o HTTP (Hypertext Transfer Protocol).

O HTTP é como um carteiro digital. Ele pega um pacote na origem (cliente), o leva até o destino (servidor) e depois traz uma resposta de volta.



TIPOS DE ENVIO (MÉTODOS/VERBOS HTTP)

Para cada envio, você precisa preencher um formulário que diz ao carteiro qual é a intenção daquela entrega. Esses formulários são os Métodos HTTP.

Cenário: Gerenciando Livros na Biblioteca SENAC

GET: PEDIDO DE COLETA/BUSCA

- Intenção: "Vá até o Arquivo Central e me traga a ficha de todos os livros."
- O que acontece: É uma ordem para buscar e **LER** informações que já existem.
- Exemplo de Ordem de Serviço: GET /livros
- Outro exemplo: "Preciso apenas da ficha do livro com a etiqueta de identificação nº 123."
- Exemplo de Ordem de Serviço: GET /livros/123

POST: ENVIO PARA ARQUIVAMENTO (CRIAR NOVO)

- Intenção: "Carteiro, leve este novo livro para ser catalogado e guardado no Arquivo Central."
- O que acontece: Você está enviando um pacote com informações novas para serem **CRIADAS** e armazenadas. Os dados do livro vão dentro do pacote.
- Exemplo de Ordem de Serviço: POST /livros

PUT: ENVIO PARA SUBSTITUIÇÃO COMPLETA

- Intenção: "Carteiro, a ficha do livro 123 está totalmente desatualizada. Leve esta nova ficha e substitua a antiga por completo."
- O que acontece: Você envia um pacote para **ATUALIZAR** um item que já existe. O item antigo é descartado e o novo toma o seu lugar.
- Exemplo de Ordem de Serviço: PUT /livros/123

DELETE: ORDEM PARA DESCARTE

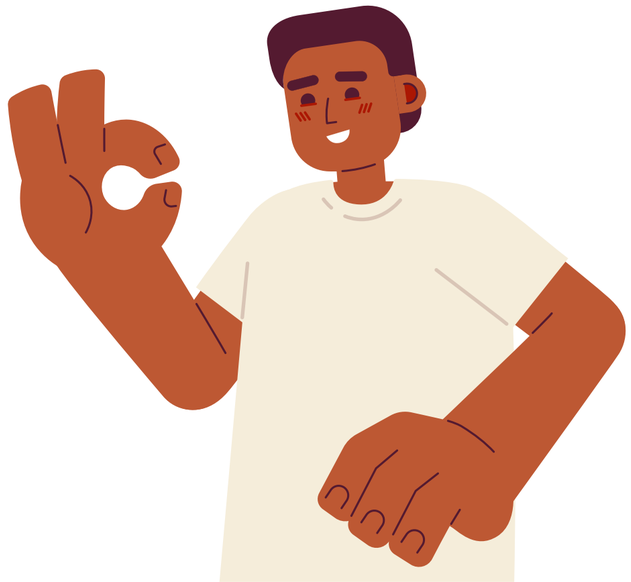
1. Intenção: "Carteiro, por favor, vá ao Arquivo Central e execute esta ordem para descartar permanentemente o livro com etiqueta nº 123."
2. O que acontece: É um comando para **REMOVER** um item do arquivo.
3. Exemplo de Ordem de Serviço: DELETE /livros/123

STATUS CODES OU RECIBO DE ENTREGA

O carteiro sempre volta com um recibo que informa o resultado da entrega. Esses recibos são os Códigos de Status HTTP, carimbos que dizem o que aconteceu.

MENSAGEM DE SUCESSO (2XX):

- 200 OK: "Entrega Realizada e Resposta Anexada." (O carteiro trouxe o que você pediu no GET).
- 201 Created: "Novo Item Arquivado com Sucesso." (Sua ordem POST funcionou).
- 204 No Content: "Ordem Executada. Nada a devolver." (O item foi apagado via DELETE).



MENSAGENS DE ERRO DO REMETENTE (4XX): VOCÊ PREENCHEU O FORMULÁRIO ERRADO.

- 400 Bad Request: "Endereço ou Formulário Inválido." (Você tentou cadastrar um livro sem título).

- 404 Not Found: "Destinatário Não Encontrado." (O livro com o ID que você pediu não existe no arquivo).

MENSAGENS DE ERRO DO SERVIÇO DE ENTREGAS (5XX): DEU PROBLEMA NO ARQUIVO CENTRAL.

- 500 Internal Server Error: "Problema no Arquivo Central." (O servidor teve um erro interno e não conseguiu completar sua solicitação).

API REST E ENDPOINTS

O Arquivo Central (nosso servidor) não é uma bagunça. Ele é extremamente organizado, como um grande arquivo com gavetas e pastas etiquetadas. A API (Application Programming Interface) é o manual de instruções desse arquivo. Ele diz exatamente quais gavetas existem e como você pode interagir com elas.

REST (Representational State Transfer) é um estilo de organização para esse manual, um método que o torna muito lógico e previsível.

ORGANIZANDO O ARQUIVO DA BIBLIOTECA COM REST:

Recursos são os Assuntos das Pastas: No nosso arquivo, livro é um recurso. Já aluno, por exemplo, é outro recurso.

- Endpoints são como as Etiquetas das Gavetas e Pastas: As URLs são os endereços exatos para encontrar cada recurso.

Exemplos:

- https://api.senac.br/livros -> É a etiqueta da gaveta principal que contém todas as pastas de livros.
- https://api.senac.br/livros/123 -> É a etiqueta da pasta específica do livro com ID 123, dentro da gaveta "livros".

MÉTODOS HTTP SÃO AS AÇÕES PERMITIDAS:

O manual da API diz quais formulários (métodos) você pode usar em cada gaveta/pasta.

Endereço (Endpoint)	Formulário (Método)	Ação no Arquivo
/livros	GET	Abrir a gaveta e ler a lista de todas as pastas.
/livros	POST	Colocar uma nova pasta de livro dentro da gaveta.
/livros/123	GET	Puxar e ler o conteúdo da pasta específica nº 123.
/livros/123	PUT	Substituir o conteúdo da pasta nº 123 por um novo.
/livros/123	DELETE	Triturar a pasta nº 123.

FORMATO DE ENVIO DA MENSAGEM (JSON)

A Interface do Usuário e o Arquivo Central precisam falar a mesma língua para se entenderem. A língua oficial da nossa comunicação é o JSON (JavaScript Object Notation).

Pense no JSON como um formato de ficha cadastral universal: simples, organizado e que tanto humanos quanto computadores conseguem ler sem dificuldade.
Estrutura da Ficha JSON:

- Usa pares de etiqueta: informação.
- As etiquetas ("titulo") estão sempre entre aspas.
- As informações podem ser texto ("Arquitetura Limpa"), números (2019), etc.

EXEMPLO DE ARQUIVO JSON


```
{
"id": 123,
"titulo": "Arquitetura Limpa: O Guia do Artesão para
Estrutura e Design de Software",
 "autor": "Robert C. Martin",
 "anoPublicacao": 2019,
"disponivel": true
}
```


JUNTANDO TUDO: O CICLO COMPLETO DE UMA MENSAGEM

Vamos ver o processo de cadastrar um novo livro do início ao fim.

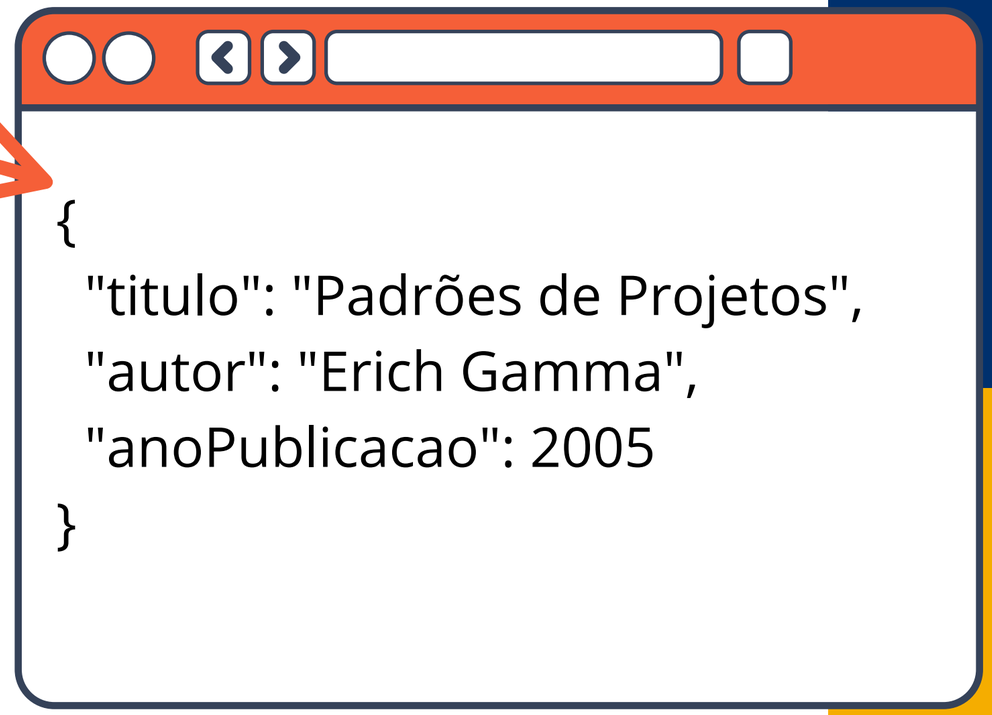
1. Ação:
O bibliotecário preenche o formulário no site com os dados do livro: "Padrões de Projetos" e clica em "Salvar".

2. A preparação (Front-end):

- **Preenche o formulário de envio: POST.**
- **Escreve o endereço de destino: /livros.**
- **Escreve a mensagem interna no idioma oficial (JSON):**
- **Coloca uma observação:**

Content-Type: application/json (Aviso: o conteúdo está em JSON).

- **Entrega a mensagem para o nosso carteiro virtual, o HTTP.**



PROCESSAMENTO NO DESTINO (BACK-END)

- O Arquivo Central (no nosso exemplo é o servidor.js) recebe o pacote.
- Lê o formulário POST e o endereço /livros, e entende: "Preciso arquivar uma nova ficha na gaveta de livros."
- Ele abre o pacote, lê a ficha em JSON, e cria uma nova pasta no arquivo, dando a ela um número de identificação único, por exemplo, 456.

RESPOSTA DO BACK-END

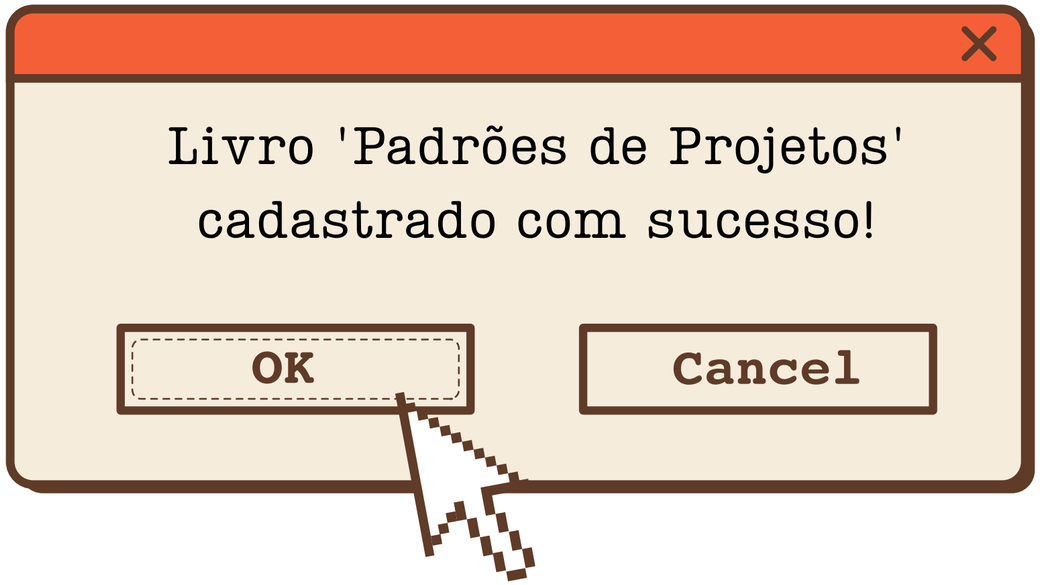
- **O Arquivo Central chama o nosso Carteiro HTTP de volta.**
- **Entrega um recibo junto com a mensagem de sucesso 201 Created.**
- **Anexa ao recibo uma cópia da ficha recém-criada, agora com o ID, para confirmação:**

arquivo.json:

```
{
  "id": 456,
  "titulo": "Padrões de Projetos",
  "autor": "Erich Gamma",
  "anoPublicacao": 2005
}
```

CONFIRMAÇÃO DA ENTREGA (FRONT-END):

- A Interface do Usuário recebe o recibo do carteiro.
- Vê o carimbo 201 Created e sabe que tudo deu certo.
- Lê a ficha anexada para confirmar os dados e exibe na tela: "Livro 'Padrões de Projetos' cadastrado com sucesso!".



RESUMO

1. Front-end: Empacota os dados do formulário em uma "ficha" JSON.

2. Front-end: Monta um "pacote" HTTP com:
Método: POST (a intenção de criar).
Endereço (Endpoint da API REST): /livros (o destino).
Header: Content-Type: application/json (a etiqueta sobre o idioma).
Corpo: A ficha JSON com os dados do livro.

3. Servidor: Recebe o pacote, lê as instruções, processa a ficha JSON e salva o livro.

4. Servidor: Monta uma "resposta" HTTP com:
Status Code: 201 Created (o carimbo de sucesso).
Corpo: A ficha JSON do livro recém-criado, agora com o id.

5. Front-end: Recebe a resposta, vê o status 201, lê os dados de confirmação e exibe a mensagem "Livro cadastrado com sucesso!" para a bibliotecária.

VAMOS VER NO CÓDIGO?

VAMOS VER NO CÓDIGO?

Vamos responder PARCIALMENTE a questão 7 da lista de exercícios 01:
Implementando o Cadastro de Livros: Nosso site atualmente só busca livros. Agora, vamos adicionar a funcionalidade de cadastrar um novo livro.

VAMOS ANALISAR POR PARTES:

```
app.post('/api/livros', async (req, res) => {
    const connection = await pool.getConnection();
    try {
        const { titulo, ano_publicacao, genero, autores_ids } = req.body;

        if (!titulo || !autores_ids || autores_ids.length === 0) {
            return res.status(400).json({ error: 'Título e ao menos um autor são obrigatórios.' });
        }

        await connection.beginTransaction();

        const sqlLivro = 'INSERT INTO LIVRO (titulo, ano_publicacao, genero) VALUES (?, ?, ?)';
        const [resultLivro] = await connection.query(sqlLivro, [titulo, ano_publicacao, genero]);
        const novoLivroId = resultLivro.insertId;

        const sqlLivroAutor = 'INSERT INTO LIVRO_AUTOR (livro_id, autor_id) VALUES ?';
        const valoresLivroAutor = autores_ids.map(autor_id => [novoLivroId, autor_id]);
        await connection.query(sqlLivroAutor, [valoresLivroAutor]);

        await connection.commit();
        res.status(201).json({ id: novoLivroId, message: 'Livro criado com sucesso!' });

    } catch (err) {
        await connection.rollback();
        console.error("ERRO NO BACKEND:", err);
        res.status(500).json({ error: 'Erro ao criar o livro.' });
    } finally {
        connection.release();
    }
});
```

Este trecho de código define o que o servidor deve fazer quando o front-end envia um "pacote" de dados (uma requisição POST) para o endereço /api/livros. O objetivo é pegar as informações que o nosso usuário cadastrou e salvá-las no banco de dados, criando assim um cadastro de um novo livro.

VAMOS ANALISAR POR PARTES:

```
app.post('/api/livros', async (req, res) => {
    const connection = await pool.getConnection();
    try {
        const { titulo, ano_publicacao, genero, autores_ids } = req.body;

        if (!titulo || !autores_ids || autores_ids.length === 0) {
            return res.status(400).json({ error: 'Título e ao menos um autor são obrigatórios.' });
        }

        await connection.beginTransaction();
```

Aqui temos nosso método POST, que está nos dizendo o seguinte: "Se chegar um cadastro com o método POST e o endereço /api/livros, entregue para a função a seguir".
A função a seguir é a função que vai cuidar deste cadastro. Ela é async (assíncrona) porque vai precisar conversar com o banco de dados, uma operação que não é instantânea e precisa esperar por uma resposta.

Pool com await, que significa que nosso código vai esperar até que uma chave seja liberada para nós.
Primeiro, pegamos essa requisição (req.body) e pegamos os dados do livro que o usuário enviou.
Depois, agimos com "segurança". Verificamos se as informações essenciais (título e autores) vieram. Se não, barramos e devolvemos um erro 400 (Bad Request), avisando que o pedido está incompleto.

VAMOS ANALISAR POR PARTES:

```
await connection.beginTransaction();

const sqlLivro = 'INSERT INTO LIVRO (titulo, ano_publicacao, genero) VALUES (?, ?, ?)';
const [resultLivro] = await connection.query(sqlLivro, [titulo, ano_publicacao, genero]);
const novoLivroId = resultLivro.insertId;

const sqlLivroAutor = 'INSERT INTO LIVRO_AUTOR (livro_id, autor_id) VALUES ?';
const valoresLivroAutor = autores_ids.map(autor_id => [novoLivroId, autor_id]);
await connection.query(sqlLivroAutor, [valoresLivroAutor]);

await connection.commit();
res.status(201).json({ id: novoLivroId, message: 'Livro criado com sucesso!' });
```

Esta é a parte mais importante e inteligente do código. Cadastrar um livro envolve duas etapas que não podem ser separadas:

1. Criar o registro principal do livro (na tabela LIVRO).
2. Vincular os autores a esse livro (na tabela LIVRO_AUTOR).

O que acontece se a etapa 1 funciona, mas a etapa 2 falha por algum motivo? Teríamos um livro "órfão" no sistema, sem autor. Isso é um erro de integridade de dados. A transação resolve isso.

await connection.beginTransaction(); Isto diz ao banco: "Atenção! Vou começar uma série de operações. Não salve nada permanentemente ainda. Apenas anote o que estou fazendo."

const sqlLivro = ...; await connection.query(...); Executamos a primeira operação: inserimos os dados na tabela LIVRO. Guardamos o ID do livro recém-criado na variável novoLivroId.

const sqlLivroAutor = ...; await connection.query(...); Executamos a segunda operação: usando o novoLivroId, inserimos os vínculos entre o livro e seus autores na tabela LIVRO_AUTOR.

await connection.commit(); Se chegamos até aqui, significa que as duas operações foram bem-sucedidas. O commit diz ao banco: "Ok, pode salvar tudo permanentemente agora. A operação foi um sucesso!"

res.status(201).json(...) Enviamos de volta o "recibo de sucesso" (201 Created) para o usuário, contendo o ID do novo livro.

VAMOS ANALISAR POR PARTES:

```
    } catch (err) {
        await connection.rollback();
        console.error("ERRO NO BACKEND:", err);
        res.status(500).json({ error: 'Erro ao criar o livro.' });
    } finally {
        connection.release();      You, 12 hours ago • Funcionalida
    }
});
```

Se algo deu errado no try (ex: um ID de autor não existe, o banco caiu), o código pula para cá.
await connection.rollback(); Ele diz ao banco: "Cancele tudo! Esqueça tudo o que eu pedi para fazer desde o beginTransaction. Nada deve ser salvo." Isso garante que não teremos um "livro órfão".
As outras linhas registram o erro para o desenvolvedor e enviam uma resposta de erro genérica (500 Internal Server Error) para o usuário.

Devolve a conexão para o pool: O comando sinaliza para o pool (o gerenciador de conexões que você criou com mysql.createPool) que você terminou de usar aquele objeto de conexão específico para a sua tarefa atual.

AGORA É SUA VEZ!
USE ESTE MATERIAL PARA RESOLVER A LISTA 01 E 02!

OBRIGADA!

Unisenac-Pelotas