RUBY RAILS - CRUD – REST – RESTFULL

🔹 CRUD

CRUD é um acrônimo que representa as quatro operações básicas que podem ser realizadas em um banco de dados ou em uma aplicação de gerenciamento de dados. Essas operações são essenciais para qualquer sistema que armazene, manipule ou recupere dados. O **CRUD** é uma maneira simples de descrever as ações básicas de um sistema.

📌 Explicação Técnica Detalhada:

1. **C - Create (Criar)**:
   A operação de criação permite inserir dados no banco de dados ou em outra forma de armazenamento persistente. No contexto de uma aplicação web, isso geralmente envolve o envio de um formulário com dados para um servidor, que os processa e os salva no banco de dados.
   - **Exemplo**: Criar um novo post em um blog.
   - **Método HTTP**: **POST**.
2. **R - Read (Ler)**:
   A operação de leitura permite consultar dados existentes no banco de dados ou sistema de armazenamento. Isso envolve obter informações para exibição, como por exemplo, listar posts de um blog ou mostrar o conteúdo de um post específico.
   - **Exemplo**: Exibir uma lista de posts ou ver os detalhes de um post.
   - **Método HTTP**: **GET**.
3. **U - Update (Atualizar)**:
   A operação de atualização permite modificar dados existentes. Em uma aplicação, isso pode envolver a edição de um recurso ou dado que já foi criado.
   - **Exemplo**: Editar o conteúdo de um post em um blog.
   - **Método HTTP**: **PUT** ou **PATCH** (o PUT geralmente substitui completamente o recurso, enquanto o PATCH é usado para modificações parciais).
4. **D - Delete (Excluir)**:
   A operação de exclusão remove dados existentes do sistema. Pode ser usado para excluir um recurso completamente.
   - **Exemplo**: Apagar um post de um blog.
   - **Método HTTP**: **DELETE**.

Exemplo de um sistema CRUD:

Imagine uma aplicação de blog onde você possa realizar todas as operações CRUD para gerenciar posts. O fluxo de cada operação seria o seguinte:

- **Criar**: Você cria um novo post por meio de um formulário de submissão que envia uma requisição POST ao servidor.
- **Ler**: Você consulta a lista de posts com uma requisição GET ou visualiza um post específico, acessando a URL `posts/:id`.
- **Atualizar**: Você edita um post existente, enviando uma requisição PUT ou PATCH para atualizar os dados do post.
- **Excluir**: Você exclui um post, enviando uma requisição DELETE ao servidor para removê-lo do banco de dados.

Essas operações se aplicam a qualquer tipo de dado ou recurso em uma aplicação web.

🔹 REST (Representational State Transfer)

REST é um estilo de arquitetura para sistemas distribuídos, principalmente para a construção de APIs (Interfaces de Programação de Aplicações) web. Foi introduzido por Roy Fielding em sua tese de doutorado em 2000 e se baseia em conceitos da web e da arquitetura HTTP.

📌 Explicação Técnica Detalhada:

- **Conceito de REST**:
  REST é uma abordagem arquitetural para construir aplicações distribuídas baseadas em recursos. Um "recurso" é qualquer coisa que você pode manipular e que tem uma representação, como usuários, posts de blog, produtos, etc. Cada recurso tem uma **URL** única e pode ser acessado ou modificado usando as operações CRUD.
- **Princípios do REST**:
  1. **Cliente-Servidor**: A arquitetura REST segue o modelo cliente-servidor, onde o cliente (geralmente o navegador ou outra aplicação) envia requisições HTTP e o servidor processa essas requisições.
  2. **Sem Estado (Stateless)**: Cada requisição é independente. Isso significa que o servidor não mantém estado entre as requisições, e todas as informações necessárias para processar a requisição devem ser enviadas com a requisição (geralmente por meio de parâmetros ou cabeçalhos).
  3. **Cacheável**: As respostas podem ser marcadas como cacheáveis, o que permite que os clientes armazenem as respostas localmente para reduzir a quantidade de chamadas ao servidor.
  4. **Interface Uniforme**: A comunicação entre o cliente e o servidor deve ser simplificada e padronizada. A URL de cada recurso deve ser bem definida, e as operações de CRUD devem ser realizadas usando os métodos HTTP adequados (GET, POST, PUT, DELETE).
  5. **Sistema em Camadas**: O sistema pode ser organizado em camadas hierárquicas, onde diferentes camadas podem ser responsáveis por diferentes tarefas (ex: balanceamento de carga, segurança, etc.).

6. **Código sob demanda (opcional)**: O servidor pode fornecer código executável (como scripts JavaScript) que pode ser utilizado pelo cliente, embora isso não seja essencial.

📌 **Como o REST é usado em APIs:**

- **URLs e Métodos HTTP**: Em REST, os recursos são identificados por URLs e manipulados usando métodos HTTP. Um recurso pode ser qualquer coisa, como um post de blog ou um produto.
    - **GET**: Recupera um recurso ou uma lista de recursos.
    - **POST**: Cria um novo recurso.
    - **PUT**: Atualiza um recurso existente.
    - **DELETE**: Exclui um recurso.
- **Exemplo de API RESTful**:
  Digamos que você tenha uma API para gerenciar posts de blog:
    - **`GET /posts`**: Retorna uma lista de posts.
    - **`GET /posts/:id`**: Retorna um post específico.
    - **`POST /posts`**: Cria um novo post.
    - **`PUT /posts/:id`**: Atualiza um post específico.
    - **`DELETE /posts/:id`**: Exclui um post específico.

REST não define como implementar os recursos, mas sim as boas práticas de como tratar os recursos e usar os métodos HTTP de maneira consistente.

🔹 RESTful

O termo **RESTful** descreve uma implementação de um sistema que segue os princípios do estilo arquitetural **REST** de maneira consistente. Em outras palavras, quando falamos que uma API é **RESTful**, isso significa que ela segue as convenções e princípios do REST de forma adequada.

📌 **Explicação Técnica Detalhada:**

Para que uma API seja considerada **RESTful**, ela deve obedecer a certos princípios e práticas recomendadas:

1. **Uso correto dos métodos HTTP**:
    - **GET** para ler recursos.
    - **POST** para criar recursos.
    - **PUT** para substituir ou atualizar recursos.
    - **DELETE** para excluir recursos.

2. **URLs representando recursos**: Cada URL na API deve representar um recurso ou coleção de recursos. O formato da URL deve ser simples, intuitivo e deve usar substantivos no plural para representar coleções (por exemplo, `/posts` para uma lista de posts).
    - **`GET /posts`**: Retorna todos os posts.
    - **`GET /posts/1`**: Retorna o post com o ID 1.
    - **`POST /posts`**: Cria um novo post.
3. **Semântica clara e consistente**:
    - Cada operação (GET, POST, PUT, DELETE) deve ser mapeada de forma consistente com a ação que ela representa, e cada URL deve ter um propósito claro e específico.
4. **Respostas padronizadas**:
    - As respostas da API devem ser consistentes, usando códigos de status HTTP padrão, como:
        - **200 OK**: Requisição bem-sucedida (geralmente para GET ou PUT).
        - **201 Created**: Recurso criado com sucesso (geralmente para POST).
        - **204 No Content**: Recurso deletado com sucesso (geralmente para DELETE).
        - **400 Bad Request**: Erro devido à requisição malformada.
        - **404 Not Found**: Recurso não encontrado.
        - **500 Internal Server Error**: Erro no servidor.
5. **Representação de recursos**:
    - A resposta de uma requisição geralmente é no formato JSON ou XML, que representa o recurso de forma estruturada. Por exemplo:
        - **GET /posts/1** retorna:

```
json
CopiarEditar
{
  "id": 1,
  "title": "Meu primeiro post",
  "content": "Conteúdo do post aqui"
}
```

6. **HATEOAS (Hypermedia As The Engine of Application State)**:
    - **HATEOAS** é um princípio do REST que sugere que as respostas de uma API contenham links para outras ações possíveis. Isso permite que o cliente navegue por diferentes estados da aplicação sem precisar conhecer todas as rotas de antemão.

Resumo Final:

- **CRUD** é um conjunto básico de operações que uma aplicação pode realizar em dados: **Criar**, **Ler**, **Atualizar** e **Excluir**.

- **REST** é um estilo arquitetural para sistemas distribuídos, especialmente APIs web, baseado em recursos, URLs, e métodos HTTP. Ele é sem estado, com uma interface uniforme, e segue convenções específicas para manipulação de dados.
- **RESTful** é o termo utilizado para descrever APIs que implementam corretamente os princípios de REST de forma consistente e ESTRUTURA.