*FastAPI - APIs Modernas e Assíncronas com Python:*

*Uma imagem com texto, Tipo de letra, Gráficos, logótipo

Os conteúdos gerados por IA poderão estar incorretos.*

O Fast-Api é um framework web projetado especificadamente para construir aplicativos com API com Python.

*Conceitos essenciais sobre APIs:*

*Representational State Transfer –* ***REST***

RESTfull padrão de criação de APIs

Criação de uma interface de comunicação utilizando puramente HTTP (Protocolo)

**O protocolo HTTP**, que é onde a Internet "roda" é por design, **sem estado**.

Isso significa que toda requisição feita a um servidor é única pois estas requisições não guardam dados (estados) entre uma requisição e outra.

Como se toda vez que você encontrasse um amigo tivesse que se apresentar para ele novamente. Pois nem você nem seu amigo guardam dados (estados) entre vocês.

O REST não muda isso, mas coloca toda a responsabilidade de "lembrar" os dados (estado) da requisição no cliente, que pode ser seu navegador/computador/aplicação.

Isso porque a cada requisição, o servidor responde pela mesma pode ser diferente. Ele pode nunca ter tido contado com o cliente a quem está contactando. Por outro lado, o cliente é o mesmo e ele entende sabe quais dados precisa seja para realizar autenticações ou mesmo para acessar diferentes endpoints.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Entendendo os Endpoints:***

Para criar bons endpoints você precisa saber o que é substantivo e verbo. Pois usamos estes conceitos para criá-los

Em uma API REST, nós temos um elemento chamado ‘resources’ (recursos)

Um ‘resource’ pode ser por exemplo um model na nossa aplicação

Recurso (Substantivo no plural) e Identificador

*HTTP:*

Uma imagem com texto, captura de ecrã, Saturação de cores, design

Os conteúdos gerados por IA poderão estar incorretos.

método ***GET*** no contexto do FastAPI:

1. **Objetivo do GET**: O método GET é utilizado para recuperar dados de um servidor. No caso da API que está sendo construida, ele é usado para retornar informações sobre cursos cadastrados.
2. **Criação de Endpoints**: Você vai criar um endpoint específico que responderá a requisições GET. Isso implica definir como os dados dos cursos serão apresentados.
3. **Modelo de Dados**: É essencial ter um modelo de dados que represente os cursos, incluindo atributos como título, dia e horas.
4. **Validação Automática**: O FastAPI permite definir tipos de dados para os parâmetros, o que ajuda na validação automática das entradas, garantindo que os dados estejam no formato correto.
5. **Testes com Cliente REST**: A execução do código deve ser testada utilizando um cliente REST, permitindo visualizar as respostas das requisições GET. Utilizando o Thunder Client
6. **Busca Específica**: Você pode criar um segundo endpoint que permite buscar um curso específico através de seu identificador, convertendo entradas de string para inteiros.

Uma imagem com texto, captura de ecrã, Tipo de letra

Os conteúdos gerados por IA podem estar incorretos.

método ***POST*** no contexto do FastAPI:

**Introdução ao Método POST**  
O método POST é utilizado para enviar dados ao servidor, tipicamente para a criação de novos recursos. É fundamental na construção de APIs modernas utilizando FastAPI.

**Implementação do Método POST**

* **Endpoint**: A criação de um endpoint para receber dados é o primeiro passo.
* **Modelo de Dados**: Utilize um modelo que define claramente os dados que devem ser enviados, como título e número de aulas.
* **Manipulação de Duplicados**: É essencial implementar lógica para verificar se os dados enviados já existem, utilizando exceções para retornar mensagens apropriadas.

**Estrutura da Requisição**

* **Corpo da Requisição**: Os dados devem ser enviados no corpo da requisição no formato JSON. É importante que campos obrigatórios sejam claramente definidos.

**Tratamento de Erros**

* Respostas de erro devem ser enviadas de forma clara, ajudando o desenvolvedor ou usuário a entender a situação. Isso melhora a experiência do usuário.

**Resposta do Servidor**

* Após a criação de um recurso, o servidor deve retornar um status apropriado (ex: 201 Created) junto com as informações do recurso criado.

**Importância das Boas Práticas**

* Manter um código limpo e bem documentado é crucial para a manutenção futura da API. Testes devem ser realizados para garantir a funcionalidade.

método ***PUT*** no contexto do FastAPI:

**Introdução ao Método PUT**  
O método PUT é utilizado para atualizar um recurso existente no servidor. É comum em APIs para modificar dados de um recurso específico.

**Implementação do Método PUT**

* **Endpoint**: Ao criar um endpoint para o método PUT, deve-se incluir um identificador do recurso que se deseja atualizar, como um ID de curso.
* **Modelo de Dados**: O método espera receber todos os dados do recurso a ser atualizado, incluindo os atributos que não foram alterados. Isso significa que é necessário enviar todos os atributos mesmo que alguns permaneçam inalterados.

**Estrutura da Requisição**

* **Corpo da Requisição**: Os dados devem ser enviados no corpo da requisição no formato JSON. É importante garantir que todos os campos obrigatórios sejam fornecidos.

**Tratamento de Erros**

* Caso o recurso a ser atualizado não exista, o código deve levantar uma exceção HTTP (por exemplo, 404 Not Found).

**Resposta do Servidor**

* Após a atualização, o servidor deve retornar um status apropriado, como 202 Accepted, junto com os dados atualizados do recurso.

**Boas Práticas**

* É recomendável manter o código consistente e bem estruturado, reutilizando lógica que já foi implementada em métodos anteriores, como o GET.

método ***DELETE*** no contexto do FastAPI:

**Introdução ao Método DELETE**  
O método DELETE é utilizado para remover um recurso existente do servidor. Ele é fundamental em operações de gerenciamento de dados em APIs.

**Implementação do Método DELETE**

* **Endpoint**: O endpoint deve incluir um identificador único do recurso a ser excluído, como um ID de curso.
* **Verificação de Existência**: Antes de proceder com a deleção, é importante verificar se o recurso existe. Isso pode ser feito buscando o recurso no armazenamento.

**Estrutura da Requisição**

* **Ausência de Corpo**: Normalmente, o método DELETE não requer um corpo na requisição, uma vez que o recurso a ser excluído é referenciado pelo identificador na URL.

**Tratamento de Erros**

* Se o recurso não existir, o servidor deve retornar uma exceção HTTP adequada, como 404 Not Found, indicando que o recurso não pode ser encontrado.

**Resposta do Servidor**

* Após a deleção bem-sucedida, a resposta do servidor deve retornar um status apropriado como 204 No Content, indicando que a operação foi realizada com sucesso e que não há conteúdo a ser retornado.

**Boas Práticas**

* É recomendado seguir as convenções de status HTTP, mantendo um código claro e organizado. A documentação de cada framework oferece orientações sobre os status retornados.

*Path Parameters:*

Introdução aos Path Parameters  
Os Path Parameters são utilizados para passar informações dinâmicas na URL de uma solicitação. Eles permitem que você identifique recursos específicos em sua API.

Implementação de Path Parameters

Definição de Rotas: Para definir uma rota que utiliza path parameters, use a sintaxe {param\_name} na URL do endpoint. Por exemplo, @app.get("/cursos/{curso\_id}") para acessar um curso específico pelo seu ID.

Uso do Tipo: É importante definir o tipo do parâmetro, utilizando por exemplo curso\_id: int para indicar que o ID deve ser um número inteiro.

Validação de Path Parameters

Validações Adicionais: Quando você define um path parameter, também pode adicionar validações. Isso é feito utilizando os argumentos do Path, como gt (greater than) e lt (less than) para realizar checagens adicionais.

Por exemplo:





**Tratamento de Erros**

* **Exceções**: Se um path parameter não corresponder ao esperado (por exemplo, um ID que não existe), você deve tratar essas exceções utilizando **HTTPException**, retornando um erro adequado, como 404 Not Found.

**Importância das Boas Práticas**

* Documentar suas rotas e utilizar títulos e descrições para path parameters ajuda na clareza da API e na geração automática de documentação, aumentando a usabilidade.

*Documentação FASTAPI:*

<http://localhost:8000/redoc>

<http://localhost:8000/docs#/default/get_curso_cursos__curso_id__get>

*Query* *Parameters:*

**Introdução aos Query Parameters**  
Os Query Parameters são utilizados para filtrar ou personalizar a resposta de uma solicitação em uma API. Eles são adicionados à URL após o símbolo de interrogação **?** e são separados por **&**.

**Implementação de Query Parameters**

* **Definição de Rotas**: Para definir um endpoint que utiliza query parameters, você pode usar argumentos na função do seu endpoint. Por exemplo:

Uma imagem com captura de ecrã

Os conteúdos gerados por IA podem estar incorretos.

* **Valores Padrão**: É possível definir valores padrão para os parâmetros para que, caso não sejam fornecidos na requisição, um valor específico seja utilizado (neste caso, **skip** começa em 0 e **limit** em 10).

**Validação de Query Parameters**

* **Validações Adicionais**: FastAPI permite adicionar validações diretamente aos query parameters, como **gt** (greater than) e **lt** (less than), que ajudam a restringir os valores aceitos.

**Tratamento de Erros**

* **Gestão de Erros**: É crucial tratar possíveis erros de entrada nos parameters, retornando mensagens apropriadas ao usuário caso os parâmetros não atendam às especificações.

**Resposta do Servidor**

* Os parâmetros de consulta podem influenciar a resposta do servidor, permitindo que os dados retornados sejam personalizados conforme o que o usuário requisitar.

**Importância das Boas Práticas**

* Documentar adequadamente os query parameters, incluindo título e descrição, é essencial para o uso adequado da API e aumenta a clareza para quem a utiliza.