



AULA 04

FASTAPI

API REST COM FASTAPI

O QUE VEREMOS HOJE

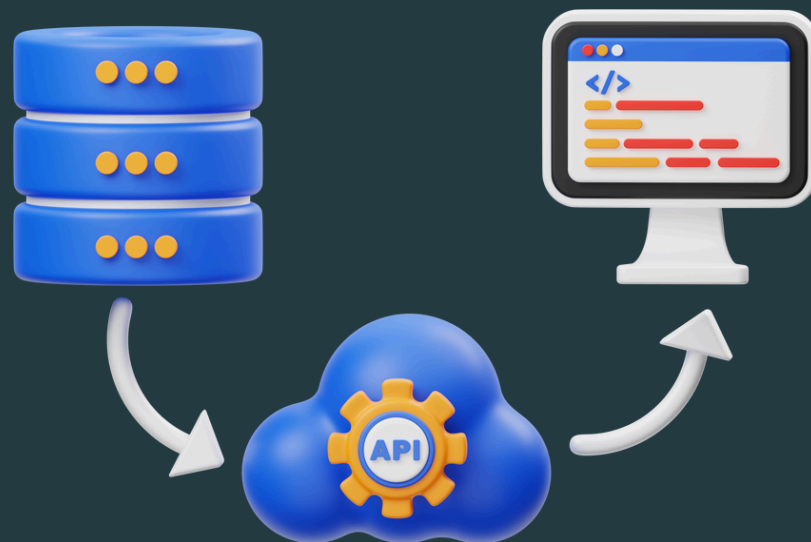
- 01 ARQUITETURA REST
- 02 RECURSOS E URLS
- 03 VERBOS HTTP E STATUS CODE
- 04 MODULARIZAÇÃO DE UMA API REST
- 05 CRUD

ARQUITETURA REST

REST (Representational State Transfer) é uma forma de estruturar e criar uma API que permite que diferentes sistemas se comuniquem de forma eficiente usando a web.

Essa forma define regras de como os dados são enviados e recebidos, usando o protocolo HTTP, para garantir que as requisições sejam simples e que o servidor saiba como responder corretamente.

Cada requisição é independente das outras, contendo todas as informações necessárias para ser processada.



REST - PRINCÍPIOS FUNDAMENTAIS

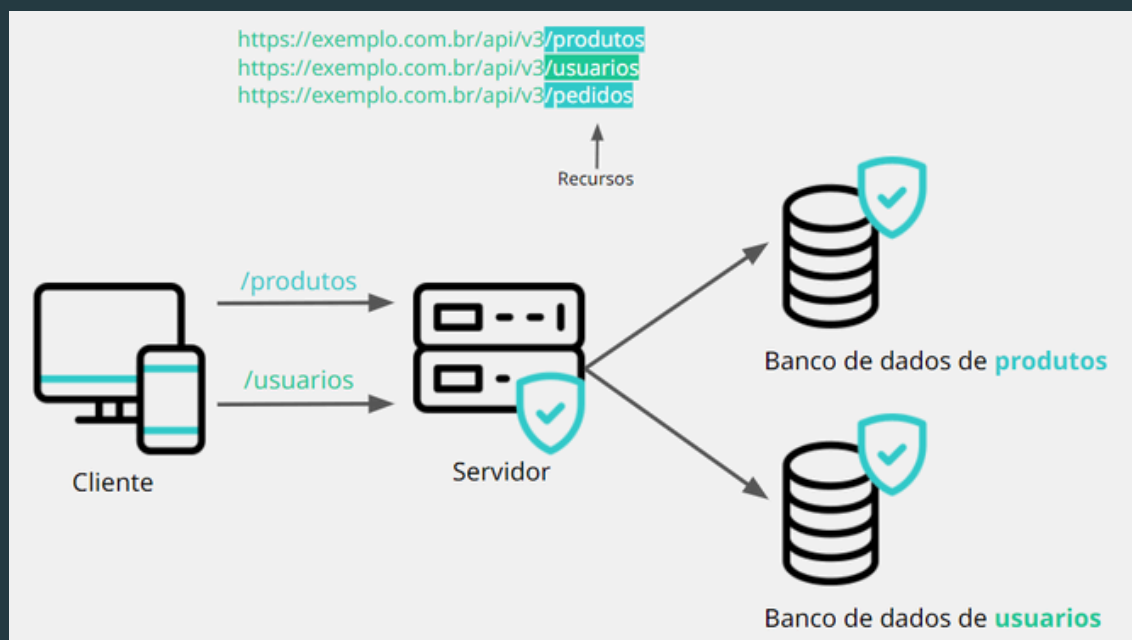
- **Statelessness (Ausência de estado):** Uma API REST deve ser projetada para que cada requisição seja completa por si só. O cliente envia todas as informações necessárias para que o servidor responda de forma independente.
- **Uniform Interface (Interface Uniforme):** Todas as interações entre cliente e servidor seguem padrões uniformes. As URLs, os métodos HTTP (GET, POST, PUT, DELETE), e os formatos de dados (geralmente JSON) são consistentes em toda a API.
- **Client-Server (Cliente - Servidor):** define uma separação clara entre o cliente e o servidor em uma arquitetura REST. O cliente é responsável por enviar requisições HTTP para o servidor, que processa essas requisições e retorna respostas.

RECURSOS E URLS

Uma das principais características da API REST é a forma que ela trata os **recursos**, que nada mais são do que os **objetos ou coleções de objetos retornados para o cliente quando ele faz a solicitação**.

Basicamente tudo que é feito em uma API REST envolve interagir com esses recursos.

Já as **URLs** são os endereços que usamos pra acessar os recursos.



VERBOS HTTP E STATUS CODE

Os verbos HTTP são fundamentais para a comunicação em APIs REST, pois definem as ações que podem ser realizadas sobre os recursos. Os principais são:

- GET (recupera dados),
- POST (cria novos recursos),
- PUT (atualiza os recursos),
- DELETE (remove recursos).

Os Status Codes, por sua vez, indicam o resultado da operação solicitada, permitindo uma comunicação clara entre o cliente e o servidor. Eles são categorizados em:

- 2xx: Sucesso,
- 4xx: Erros do cliente,
- 5xx: Erros do servidor.

MODULARIZAÇÃO DE UMA API REST

A modularização das funcionalidades de uma API REST é crucial para a qualidade da comunicação entre sistemas.

Modularização é a prática de dividir o código em partes menores, independentes e reutilizáveis (módulos). Isso facilita o desenvolvimento, manutenção e escalabilidade do projeto.


```
api/
├── app/
│   ├── main.py          # Ponto de entrada da aplicação
│   ├── routes/          # Define as rotas/endpoints
│   │   ├── users.py     # Rotas relacionadas a usuários
│   │   └── items.py     # Rotas relacionadas a itens
│   ├── controllers/     # Lógica dos controladores
│   │   └── user_controller.py
│   ├── services/        # Lógica de negócios
│   │   └── user_service.py
│   ├── models/          # Definição dos modelos de dados
│   │   └── user_model.py
│   ├── schemas/         # Definição dos esquemas de validação (Pydantic)
│   │   └── user_schema.py
│   ├── db/              # Configuração do banco de dados
│   │   └── database.py
```

ROUTES

As rotas (ou endpoints) são os caminhos pelos quais os clientes interagem com a API.

Cada rota está associada a um caminho URL específico e a uma função que processa a requisição.

As rotas mapeiam verbos HTTP (GET, POST, PUT, DELETE, etc.) para ações em recursos da API.



```
1 from fastapi import APIRouter
2
3 # Definindo um APIRouter específico para o recurso "user"
4 router = APIRouter()
5
6 # Exemplo de rota GET para listar usuários
7 @router.get("/users")
8 def list_users():
9     return {"message": "Listando todos os usuários"}
10
11 # Exemplo de rota POST para criar um novo usuário
12 @router.post("/users")
13 def create_user():
14     return {"message": "Usuário criado com sucesso"}
```


ROUTES

No arquivo principal (main.py) da API é necessário fazer a importação do arquivo de rotas e incluir usando o `include_router`.

O `include_router` é uma função do FastAPI que permite incluir um conjunto de rotas.




```
1 from fastapi import FastAPI
2 from routes import user
3
4 app = FastAPI()
5
6 app.include_router(user.router)
7
8 @app.get('/')
9 def index():
10     return {'message': 'api running'}
11
```

SCHEMAS

São utilizados para definir a estrutura e a validação dos dados que a API recebe e retorna.


No FastAPI, os schemas são implementados com Pydantic, que permite criar modelos de dados com validação automática.



```
1  from pydantic import BaseModel, EmailStr
2  from typing import Optional
3
4  class User(BaseModel):
5      name: str
6      email: EmailStr
7      age: Optional[int] = None
8
```

SCHEMAS

Os schemas são usados nas rotas, por isso, devem ser importados no arquivo routes.



```
1  from fastapi import APIRouter
2  from schemas.user import User
3
4  router = APIRouter()
5
6  users = []
7
8  @router.get('/users')
9  def get_users():
10     return users
11
12  @router.post('/users')
13  def create_user(user: User):
14     users.append(user)
15     return {
16         'message': 'User created successfully',
17         'user': user
18     }
19
```

CRUD

CRUD é um acrônimo que representa as quatro operações básicas que é possível realizar em dados em um sistema:

- **C**reate (Criar)
- **R**ead (Ler/Consultar)
- **U**ppdate (Atualizar)
- **D**elete (Deletar/Excluir)

Essas operações são fundamentais para qualquer aplicação que interaja com dados, como um sistema de gerenciamento de usuários, um sistema de estoque, etc.

Quando se cria uma API RESTful, a maioria dos endpoints estará relacionada a essas operações.

CRUD E OS VERBOS HTTP

- **Create** → **POST**: Usamos o verbo HTTP POST para criar novos recursos.
- **Read** → **GET**: Usamos o verbo HTTP GET para ler ou buscar informações. Essa operação é usada quando queremos consultar dados de um recurso existente, como obter uma lista de usuários ou detalhes sobre um usuário específico.
- **Update** → **PUT**: Usamos os verbos HTTP PUT para atualizar um recurso existente.
- **Delete** → **DELETE**: Usamos o verbo HTTP DELETE para excluir um recurso.

ATIVIDADE PRÁTICA

Crie um CRUD de gerenciamento de livros. Cada livro terá o título, autor, número de páginas e uma categoria. O ID do livro deve ser gerado pela API.

O CRUD deve conter os seguintes endpoints:

- POST /books → Criar um novo livro.
- GET /books → Obter todos os livros.
- GET /books/{book_id} → Obter detalhes de um livro específico.
- PUT /books/{book_id} → Atualizar as informações de um livro existente.
- DELETE /books/{book_id} → Deletar um livro.

Utilize a modularização para incluir as rotas e o schema