Desenvolvimento de Sistemas

Aula 09 – Arquitetura de Software e Padrões de Projeto

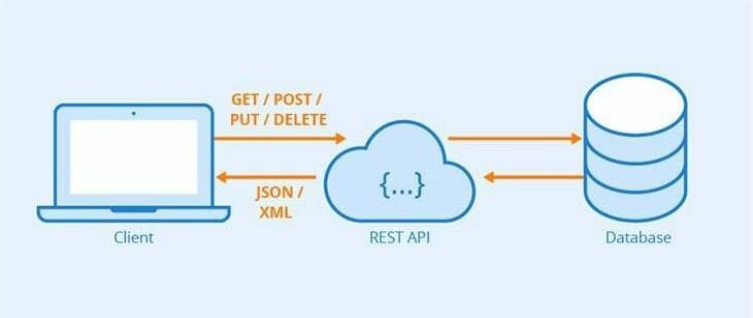
Lorrany B A Marim

SENAI

Introdução a APIs REST

Comunicação eficiente entre sistemas

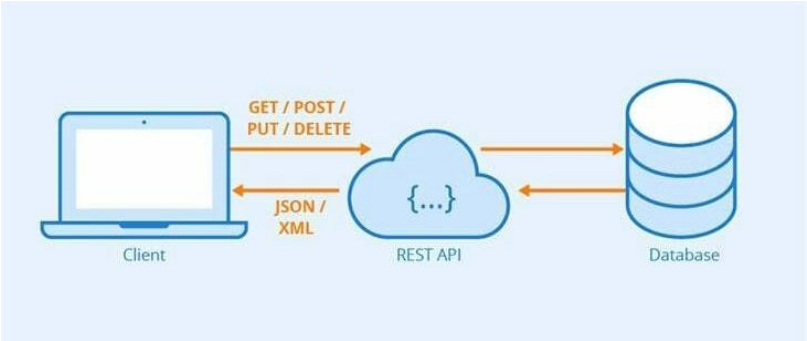
- **O que é REST:** *Representational State Transfer* (Transferência de Estado Representacional). É um estilo de arquitetura para sistemas distribuídos.
- **Stateless (Sem estado):** Uma característica fundamental. O servidor não guarda o estado da sessão do cliente entre as requisições. Cada pedido deve conter todas as informações necessárias para ser processado.



Introdução a APIs REST

Comunicação eficiente entre sistemas

- **Recursos:** Em REST, tudo é um "recurso" (ex: um usuário, um produto, uma nota fiscal), acessado via uma URL única.
- **Objetivo:** Permitir que sistemas diferentes (ex: um app de celular e um servidor) conversem de forma padronizada e escalável.



Verbos HTTP e Endpoints

A gramática da API

- **Endpoints:** São as URLs onde seus serviços estão disponíveis (ex: /api/usuarios).
- **Verbos HTTP:** O REST usa os métodos do protocolo HTTP para definir a ação a ser realizada sobre o recurso:
  - **GET:** Recuperar dados (Ler).
  - **POST:** Criar novos dados.
  - **PUT:** Atualizar dados existentes.
  - **DELETE:** Remover dados.
- **Exemplo:** Um GET em /produtos lista os produtos; um POST em /produtos cria um novo.

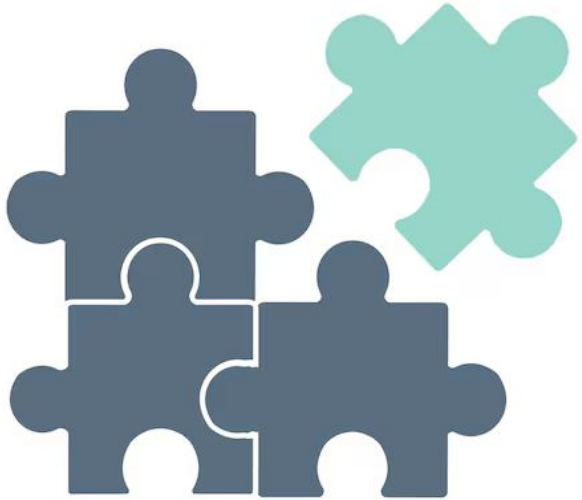
API Endpoint

Boas Práticas de Integração

Construindo APIs profissionais

- **Use JSON:** É o formato padrão para troca de dados em APIs modernas, substituindo o XML por ser mais leve e fácil de ler.

- **Documentação:** Uma API sem documentação é inutilizável. É vital descrever claramente como consumir os serviços.

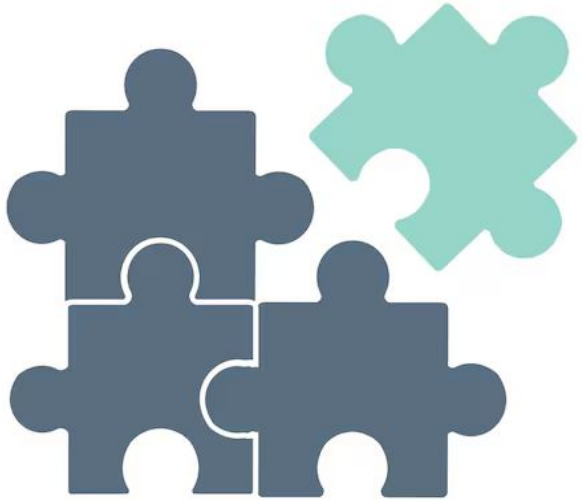


Boas Práticas de Integração

Construindo APIs profissionais

- **Códigos de Status:** Use os códigos HTTP corretos para responder (ex: 200 para sucesso, 404 para não encontrado, 500 para erro do servidor).

- **Segurança:** Proteja os dados validando entradas e garantindo a consistência das informações no banco de dados.



Introdução aos Padrões de Projeto

Não reinvente a roda

- **O que são:** Padrões de Projeto (*Design Patterns*) são soluções típicas e reutilizáveis para problemas comuns no projeto de software.
- **Origem:** Eles capturam a experiência de desenvolvedores experientes, registrando soluções que funcionaram no passado para que não precisemos resolver tudo do zero.

Introdução aos Padrões de Projeto

Não reinvente a roda

- **Objetivo:** Criar sistemas mais flexíveis, modulares e fáceis de manter.

- **Categorias:** Os padrões são geralmente divididos em três grupos: Criacionais (criação de objetos), Estruturais (composição de classes) e Comportamentais (interação entre objetos).

Estrutura de um Padrão

Os 4 elementos essenciais

Para descrever um padrão, usamos quatro partes principais:

1. Nome: Uma referência curta que nos permite falar sobre o problema e a solução (ex: "Use um Singleton aqui").

2. Problema: Descreve *quando* aplicar o padrão e o contexto em que ele é útil.

Estrutura de um Padrão

Os 4 elementos essenciais

Para descrever um padrão, usamos quatro partes principais:

3. Solução: Descreve os elementos (classes e objetos), suas relações e responsabilidades. Não é um código pronto, mas um modelo.

4. Consequências: Os resultados, vantagens e desvantagens de usar o padrão (ex: ganho de flexibilidade vs. complexidade).

Por que usar Design Patterns?

Vantagens para o desenvolvedor

- **Vocabulário Comum:** Facilita a comunicação. Dizer "vamos usar uma Factory" transmite uma ideia complexa rapidamente para a equipe.

- **Reutilização de Experiência:** Permite usar soluções testadas e aprovadas, evitando erros comuns de quem está começando .

Por que usar Design Patterns?

Vantagens para o desenvolvedor

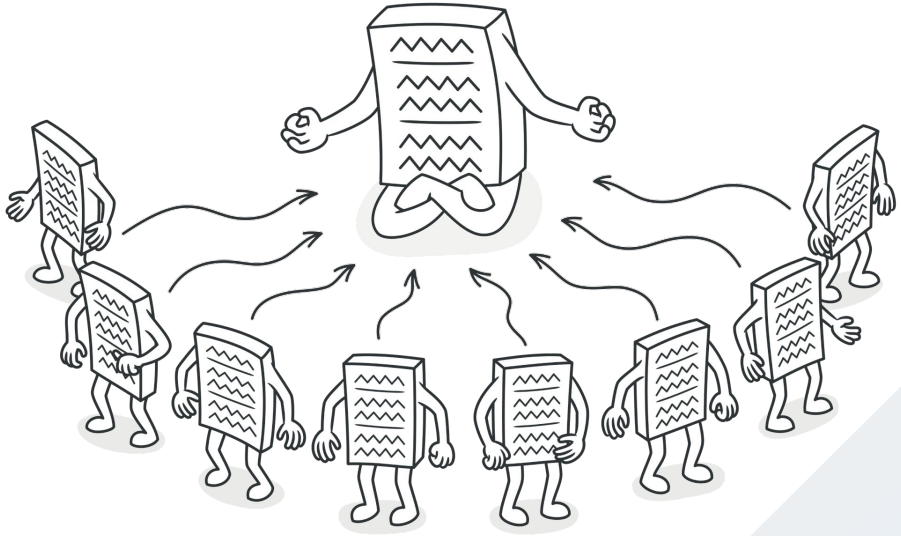
- **Flexibilidade:** Padrões como o *Strategy* ou *Observer* tornam o sistema mais fácil de modificar e estender sem quebrar o código existente.

- **Refatoração:** Ajudam a reorganizar códigos ruins (spaghetti) em estruturas mais limpas e profissionais.

Padrão Singleton

Garantindo a unicidade

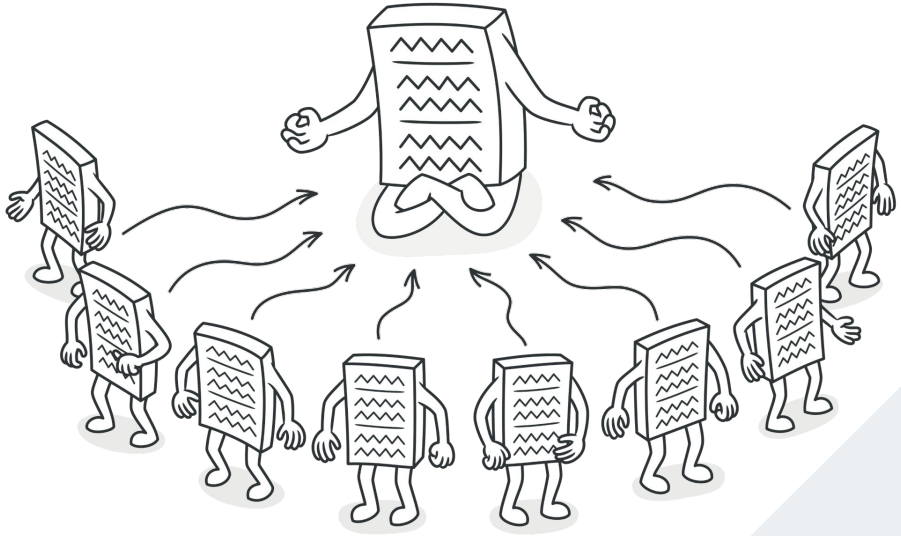
- **Conceito:** O padrão Singleton garante que uma classe tenha apenas **uma única instância** e fornece um ponto global de acesso a ela.
- **Quando usar:** Quando precisamos de apenas um objeto para coordenar ações em todo o sistema.



Padrão Singleton

Garantindo a unicidade

- **Exemplos de uso:**
  - Gerenciador de conexão com Banco de Dados.
  - Sistema de Logs (registro de eventos).
  - Spooler de impressão.
- **Cuidado:** O uso excessivo pode funcionar como variáveis globais, o que dificulta testes.



Padrão Singleton

Exemplo de Código

```python
class DatabaseConnection:
    _instancia = None  # Variável para guardar a única instância

    def __new__(cls):
        # Se a instância ainda não existe, cria uma nova.
        if cls._instancia is None:
            cls._instancia = super(DatabaseConnection, cls).__new__(cls)
            cls._instancia.status = "Conectado"
        # Retorna sempre a mesma instância
        return cls._instancia

# Testando
db1 = DatabaseConnection()
db2 = DatabaseConnection()

print(db1 is db2)  # Saída: True (São o mesmo objeto)
```

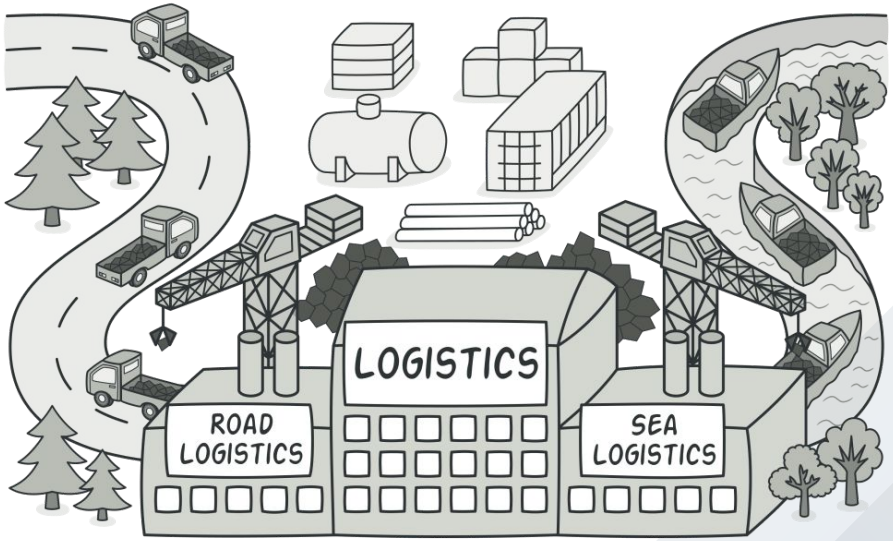
Em Python, usamos o método __new__ para controlar a criação da instância, garantindo que seja única.

Padrão Factory

Terceirizando a criação de objetos

- **Conceito:** Define uma interface para criar um objeto, mas deixa as subclasses (ou uma função específica) decidirem qual classe instanciar.

- **Problema:** Evitar que o seu código fique cheio de `new ClasseEspecifica()`. Se a classe mudar, você teria que alterar o código em vários lugares.

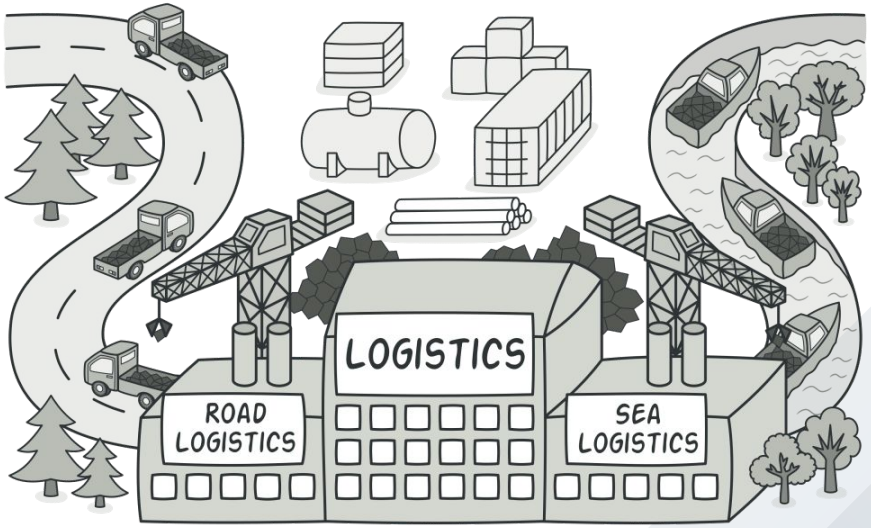


Padrão Factory

Terceirizando a criação de objetos

- **Solução:** Você pede o objeto a uma "Fábrica". A fábrica decide se entrega um objeto do tipo A, B ou C, dependendo da necessidade.

- **Benefício:** O código cliente não precisa saber a classe exata do objeto que vai usar, apenas que ele funciona.



Padrão Factory

Exemplo de Código

```python
class Cachorro:
    def falar(self):
        return "Au Au"

class Gato:
    def falar(self):
        return "Miau"

# Esta é a nossa "Fábrica"
def animal_factory(tipo):
    if tipo == "cao":
        return Cachorro()
    elif tipo == "gato":
        return Gato()
    else:
        raise ValueError("Animal desconhecido")

# O código cliente não precisa saber instanciar as classes
pet = animal_factory("gato")
print(pet.falar())  # Saída: Miau
```

O padrão isola a lógica de criação (os if/else) dentro da função factory, limpando o resto do código.

Fontes Utilizadas

- *Downey, A. B. "Pense em Python".*
- *Gamma, E. et al. "Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos".*
- *Danjou, J. "Python Levado a Sério".*
- *Valente, M. T. "Engenharia de Software Moderna".*
- *Preece, J. et al. "Design de Interação".*


Obrigado!
SENAI