Questão 1 — O que você entende por arquitetura MVC (em linguagem de leigo)

A arquitetura **MVC** (Model-View-Controller) é como dividir um sistema em três partes para ele ficar mais organizado e fácil de mexer:

- **Model (Modelo):** cuida das informações e regras. Por exemplo, os dados de um cliente ou produto.
- **View (Visualização):** é o que aparece na tela para o usuário — botões, formulários, listas, etc.
- **Controller (Controlador):** interpreta os cliques e comandos do usuário e diz ao sistema o que fazer.

Separando assim, é possível mudar a aparência sem bagunçar os dados, ou alterar regras sem afetar o visual. Isso torna o sistema mais limpo, flexível e fácil de manter.

Questão 2 — Escolha de padrão de projeto: SOLID

Conceito

SOLID é um conjunto de cinco princípios que ajudam a escrever códigos organizados, flexíveis e fáceis de manter. Ele guia como criar classes e funções que funcionam bem separadas umas das outras.

Objetivos

- Evitar que uma classe faça várias coisas ao mesmo tempo
- Permitir que novas funcionalidades sejam adicionadas sem quebrar o que já funciona
- Usar interfaces e abstrações para tornar o código mais reutilizável

Princípios SOLID

Sigla	Princípio	Explicação simples
S	Single Responsibility	Uma classe deve ter apenas uma responsabilidade
O	Open/Closed	Deve estar aberta para extensão, mas fechada para alteração
L	Liskov Substitution	Classes derivadas podem substituir as classes base sem problemas
I	Interface Segregation	Não forçar a implementação de métodos que a classe não precisa
D	Dependency Inversion	Depender de abstrações em vez de detalhes concretos

Exemplo prático — Sistema de Pagamentos

Imagine um sistema que precisa lidar com diferentes formas de pagamento: boleto, cartão de crédito e Pix. Aplicando os princípios SOLID, o sistema poderia ser montado assim:

- Cria-se uma **interface** chamada `IMetodoPagamento` com o método `RealizarPagamento()`.
- Cada tipo de pagamento é representado por uma classe: `PagamentoPorCartao`, `PagamentoPorPix`, etc.
- Essas classes seguem a interface e têm suas próprias implementações.
- Se surgir um novo método, como PayPal, basta criar uma nova classe sem mexer nas anteriores.

Esse exemplo usa todos os princípios SOLID de forma prática, promovendo manutenção fácil, código limpo e crescimento seguro.