Vaga analista pleno

Nome : Kauã Marques
Data : 09/02/2026

1. **Diferença entre `ref` e `out`**: No **`ref`**, a variável deve ser inicializada antes de ser passada; o método pode ler ou alterar o valor. No **`out`**, a inicialização prévia é opcional, mas o método é obrigado a atribuir um valor antes de retornar.
2. **Boxing/Unboxing**: **Boxing** é converter um tipo de valor (ex: `int`) para o tipo `object` (referência). **Unboxing** é o processo inverso, extraindo o valor do objeto.
3. **Select vs SelectMany (LINQ)**: O **`Select`** transforma cada elemento em um novo formato (1 para 1). O **`SelectMany`** achata sequências de sequências em uma única lista (1 para N).
4. **Async/Await e Deadlock**: São usados para operações não bloqueantes. Para evitar **deadlocks**, deve-se usar `await` em toda a cadeia (evitar `.Result` ou `.Wait()`) e usar `ConfigureAwait(false)` em bibliotecas.
5. **Moq, Mock e Stub**: **Moq** é um framework para criar objetos simulados. **Stub** fornece respostas prontas para o teste rodar. **Mock** verifica se o código interagiu corretamente com a dependência (ex: se chamou o método "Salvar").
6. **Controller → Service → Repository**: O **Controller** recebe a requisição; o **Service** aplica as regras de negócio; o **Repository** realiza a persistência no banco de dados.
7. **Injeção de Dependência**: É um padrão onde uma classe recebe suas dependências de fora em vez de criá-las. No **.NET Core**, isso é feito via container nativo no `Program.cs`, usando os tempos de vida *Transient*, *Scoped* ou *Singleton*.
8. **Padrões de Projeto**: Sim, utilizo **Repository** para isolar o acesso a dados, **Factory** para criação de objetos complexos e **Strategy** para alternar algoritmos de negócio (como diferentes tipos de cálculo de frete ou desconto) sem alterar a classe principal.
9. Como lida com código legado?

   Meus últimos anos foram sustentando de dando manutenção em código legado, como vb, apsx, .net v- 4.5, usando essas ferramentas percebi que elas não são ruins são mal utilizadas, eu realmente só penso em refatorar ou migrar quando vira gargalho, ou analisando e vendo que no futuro irá prejudicar aplicação, caso do contrário mantenho o monstrinho alimentado por anos.

10. Criar um sistema em c# (versão .NET6) que permita logar e gerenciar pedidos de um e-commerce fictício com estrutura orientada a objetos, regras de negócio claras e boas práticas de código, use o angular versão 18 para o Front end.

Implemente um sistema de pedidos simples com os seguintes requisitos:

Entidades:
Usuario:
Id
Nome

Cliente
Id

Nome
CPF
Produto
Id
Nome
Preço

Pedido
Id
Cliente
Lista de produtos
Data do pedido
Status (Criado, Pago, Cancelado)

Regras de Negócio
Um pedido pode ser criado com um cliente e lista de produtos.
Um pedido pode ser pago (alterar status para "Pago").
Um pedido pode ser cancelado (desde que ainda não esteja pago).
Para inserir e realizar alterações nas entidades deve armazenar o histórico caso precise cria a entidade de histórico.

O sistema deve permitir:
Cadastrar clientes
Cadastrar produtos
Criar pedidos
Listar pedidos por status
Exibir o valor total de um pedido

Requisitos Técnicos
Use orientação a objetos com boas práticas.
Crie ao menos 1 interface e utilize injeção de dependência.
Utilize LINQ onde fizer sentido.
Crie testes unitários básicos (por exemplo, com xUnit).
Separe camadas (pelo menos: Models, Services, Program).