Orientação a Objetos

Explique o conceito de herança múltipla e como C# aborda esse cenário.

Em C#, uma classe só pode herdar de uma outra classe (herança simples), mas pode implementar várias interfaces. Isso é a forma do C# "contornar" a herança múltipla.
Ou seja, se eu quiser que uma classe tenha comportamentos de várias fontes, deve-se usar interfaces, evitando os problemas de ambiguidade que podem surgir com herança múltipla tradicional.

Explique o polimorfismo em C# e forneça um exemplo prático de como ele pode ser implementado.

Polimorfismo é a ideia de tratar objetos diferentes de forma parecida, desde que eles tenham algo em comum. Por exemplo, se eu tenho uma interface INotification, e classes como EmailNotification e SmsNotification, pode-se tratá-las como INotification e chamar métodos sem me importar com o tipo real. Cada um será executado do seu jeito.

SOLID

Descreva o princípio da Responsabilidade Única (SRP) e como ele se aplica em um contexto de desenvolvimento C#.

Significa que uma classe deve ter um único motivo para mudar. Em C#, isso evita acoplamento excessivo. Por exemplo, uma classe InvoiceService que salva fatura e também envia email está quebrando esse princípio. Melhor separar: uma classe pra salvar fatura, outra pra envio de email.

Como o princípio da inversão de dependência (DIP) pode ser aplicado em umprojeto C# e como isso beneficia a manutenção do código?

É quando o código depende de abstrações, não de implementações concretas.
Em C#, a gente usa interfaces e injeção de dependência. Por exemplo, se eu tenho um IUserRepository, meu serviço depende disso, e não de um UserRepository direto. Isso facilita testes e mudanças futuras.

Entity Framework (EF)

Como o Entity Framework gerencia o mapeamento de objetos para o banco de dados e vice-versa?

O EF faz o "mapeamento" (ORM - Object-Relational Mapping), ou seja, ele transforma classes C# em tabelas e propriedades em colunas. Ele também faz o caminho inverso, carregando dados do banco e transformando em objetos prontos pra usar no código.

Como otimizar consultas no Entity Framework para garantir um desempenho eficiente em grandes conjuntos de dados?

Pra melhorar a performance, é importante usar.

- AsNoTracking() em consultas somente leitura.
- Include() com moderação pra evitar dados grande demais para a necessidade.
- Projeções com Select() pra retornar só o que precisa.
- Evitar carregar todos os dados da tabela sem paginação.

WebSockets

Explique o papel dos WebSockets em uma aplicação C# e como eles se comparam às solicitações HTTP tradicionais.

Os WebSockets permitem comunicação em tempo real, diferente do HTTP tradicional que é requisição/resposta. No C#, com SignalR, é bem comum usar WebSockets pra chats, notificações em tempo real, ou status de pedidos, por exemplo.

Quais são as principais considerações de segurança ao implementar uma comunicação baseada em WebSockets em uma aplicação C#?

É importante proteger com autenticação (ex: token JWT), usar HTTPS pra criptografar os dados, validar o que chega via socket (não confiar na requisição do cliente), e sempre monitorar a conexão, especialmente se o canal fica aberto por muito tempo.

Arquitetura

Descreva a diferença entre arquitetura monolítica e arquitetura de microsserviços. Qual seria sua escolha ao projetar uma aplicação C#?

A arquitetura monolítica consiste em uma única aplicação coesa, onde todos os módulos como autenticação, pedidos, pagamentos, estão agrupados e executados juntos. Isso facilita o desenvolvimento inicial, o deploy e a gestão de infraestrutura, especialmente em projetos menores ou com times reduzidos.

Já a arquitetura de microsserviços divide a aplicação em diversos serviços independentes, cada um com sua responsabilidade clara, podendo ter sua própria base de dados e ser implantado de forma separada. Essa abordagem favorece escalabilidade, flexibilidade e independência entre os squads de desenvolvimento.

Como você escolheria entre a arquitetura de microsserviços e a arquitetura monolítica ao projetar uma aplicação C# que precisa ser altamente escalável?

Se a aplicação precisa escalar partes específicas (por exemplo, só o módulo de pagamentos), microsserviços ajudam. Também se times diferentes cuidam de áreas diferentes do sistema, essa separação facilita. Mas dá mais trabalho: orquestração, deploys, observabilidade...
Então, se o projeto justifica, vale a pena. Senão, prefiro evoluir um monolito bem estruturado.