## Teste para desenvolvedor .net

O teste abaixo visa avaliar o perfil técnico dos candidatos a desenvolvedor backend. Leia atentamente as instruções abaixo e responDa as questões da melhor forma possível.

* O teste será composto de três perguntas técnicas e uma avaliação prática.
* Responda as perguntas com suas palavras, com seu entendimento sobre o tema.
* Publique o teste prático com as respostas em um repositório no git-hub e nos envie.

## Perguntas:

## Explique com suas palavras o que é domain driven design e sua importância na estratégia de desenvolvimento de software.

## Na minha visão em termos básicos o DDD quando Eric Evans criou, ele compilou as melhores práticas de desenvolvimento em uma modelagem no qual O DDD FOCA NO DESENVOLVIMENTO ORIENTADO AO NEGÓCIO, ou seja, NA entidade/domínio, com o objetivo de resolver de maneira mais fácil os complexos processos e regras envolvidos. Essa modelagem é baseada em responsabilidades (camadas): apresentação, aplicação, domínio, serviços e infraestrutura. O processo de desenvolvimento inicial, ou seja, montar toda a estrutura de um projeto em DDD, é mais moroso, mas o torna mais fácil, ágil e flexível com o tempo, para as novas implementações e manutenções.

## Explique com suas palavras o que é e como funciona uma arquitetura baseada em microservices. explique ganhos com este modelo e desafios em sua implementação.

## Micro serviços são pquenos módulos (serviços) que são desenvolvidos, escalonados e executados um independente do outro, assim como seu próprio deploy, ou seja, cada microserviço é uma aplicação separada, fazendo parte de uma aplicação principal. Cada microserviço, tem a responsabilidade de receber, processar e devolver informações através dos protocolos HTTP e/ou AMQP (pode-se utilizar de outros, mas estes, são os principais). Vantagens: os microsserviço costumam ser pequenos e com isso, facilita o desenvolivmento, manutenção (correções mais rápidas), gerenciamento e a curva de aprendizado; por serem independentes, facilita e agiliza a implantação de novas versões; podem ser escalonada horizontalmente sem impactar nas demais; na aplicação principal, pode haver tecnologias diferentes coexistindo pelos microservicos, através dos containers (trabalha bem com docker). Desvantagens: lidar com as falhas dos microserviços e como a falha reflete no sistema e ao usuário, trazendo uma complexidade adicional nos testes e monitoramento. essa desvantagem é minimizada se os microservicos tiverem um baixo nível de acoplamento entre sí e se utilizar ferramentas adequadas de rollback e testes (xunit para teste de integração em webapi). Outra desvantagem, talvez, entenda-se como um desafio, é como particionar uma aplicação de ponta à ponta em microservicos; tomar cuidado com multiplos serviços concorrentes (antigos e novos); requer uma infra mais robusta para execução de todas as threads na aplicação principal; utilizar o microserviço com comunicação sincrona, dependendo da aplicação e o nújero de requests, o tempo de resposta devido ao enfileiramento pode ser muito alto (custos com serialização/deserialização).

## explique qual a diferença entre comunicação sincrona e assincrona e qual o melhor cenário para utilizar uma ou outra.

## Exemplificar da forma mais fácil:

## você pode pedir pizza de duas maneiras, ligando para a pizzaria (ASSINCRONO) ou indo diretamento na pizzaria (SINCRONO). No primeiro caso, depois que você pede a pizza, você continua suas tarefas e aguarda o motoboy fazer a entrega, isto é modo de comunicação assincrono e tem algumas vantagem, tanto para o lado da aplicação por vista do usuário, no qual ele continua a interatividade sem que fique eternamente aguardado um ação do lado do servidor, pois requisição assincrona, libera para a chamada do thread principal, o remetente não fica bloqueado, exmplo: efetuei uma compra, tenho que aguardar a aprovação do cartão de crédito, em quanto isso a aplicação fica liberada para eu interagir, sem ter que ficar X tempo aguardando a respota da operadora do cartão. Ou seja para operações longas como: acesso a banco de dados, sistema de arquivos, serviços externos ao servidor ou mesmo uma melhor interação com o usário, usa-se chamada assincrona. Estas tecnologias reunidas: DOCKER + MICROSERVIÇOS + DDD + TDD + MENSAGERIA (e mais algumas), casam perfeitamente com as duas primeiras perguntas. Agora quando você vai na pizzaria, você fica preso, bloqueado, aguardando a sua mensagem, quero dizer a pizza. Este tipo de requisição é usada quando a próxima execução é dependente da anterior. O próprio w3schools não recomenda mais o uso do AJAX como sincrono, inclusive a classe HttpClient não possuí métodos sincronos.

## Teste prático:

## regra de negócio:

## 

## criar um microservices que, através de um http post efetue uma operação de debito (origem) e credito (destino) nas contas correntes.

## entidades: contacorrente, lancamentos (voce pode incrementar com outras entidades se achar necessário)

## Parâmetros de entrada:

## conta origem

## conta destino

## valor

## Parâmetros de saída:

## http status code

## 

## informações adicionais:

## o método “post” devera receber os parametros no body da requisição em formato json

## UTILIZE Domain Driven Design

## serão avaliados critérios de arquitetura como separação de responsabilidade, clean code, segurança e testes

## tecnologias que você pode utilizar .net core 2.X, c#, xunits (testes)

## no término do projeto, publique o código em um repositório no git-hub