**Pergunta 1: Arquitetura e Design**

Tendo em vista os requisitos não funcionais de 300k/users/hour, temos a concorrência de 5k/users/hour estimado (sem consideração de picos em horários, sem considerar limitações por outros processos, sem considerar nenhuma outra informação relevante que não está presente na questão). Com isto, devemos trabalhar com a ideia de micro aplicações utilizando micro serviços conforme diagrama abaixo:



Características da visão lógica:

1. Implementação de aplicação React, com conceitos de reducers para a criação de componentes independentes para colocação em qualquer engine de portal ou webserver.
2. Aplicação de micro serviços baseados em NodeJS com componentes React/Redux + Routes para criação de micro serviços para cada componente de tela e cada componente de negócio necessário.
3. Implementação de layer de persistencia com framework de Mutex para controles de concorrência e paralelismo controlados;

Nas questões de implementação, teríamos que pensar num sistema escalável, logo de baixo acoplamento e dependências, podendo ser multiplicado em instâncias diferentes, com possibilidade de vendors diferentes, mas gerenciado e controlado de forma escalar e modular;

Visão de Implementação



Características da Visão de implementação:

1. Sistemas Online e Offline integrados por meio de mensagerias, provenientes de quaisquer um dos lados;
2. Sistemas reativos com integração direta aos dados via modelo de persistência ou por mensageria;
3. Altamente acoplável a qualquer infraestrutura, independente de tecnologia, on premisses ou cloud;

Assim sendo, podemos concluir que nas questões relativas a infraestrutura e segurança podemos ter:



Assim, temos as seguintes características:

1. Log de sessões de “quem/quando/o que/como fez” integrados;
2. Garantia de segurança na borda (cloud infrastructure) e na aplicação application authorization & authentication)
3. Escalabilidade por se tratar de framework escalável;

**2) Pergunta 2: Limitação Backoffice**

Dado o problema de limitação, a solução dada na questão anterior proporciona a limitação e o paralelismo através do pattern de Mutex, onde sua característica é a limitação de processamento de determinado tipo de componente, ou instancia de determinados tipos de componentes sejam limitados; Além disto, pela implementação da mensageria, nenhuma transação seria “perdida”, já que, antes da persistencia ou processamento em banco, antes do processamento dos componentes de banco, mesmo implementados com mutex, teríamos ainda os componentes de queue para a gestão destes componentes.

Visão da solução



Desta forma:

1. Integrações provenientes dos sistemas se dão por microserviços (routes) que mandam os dados para as filas de processamento(QueueSenders) e são armazenados nas filas (Queues) para serem processadas assincronamente pelos processadores (QueueReaders); Estes por sua vez disparam os componentes com configurações de de paralelismo (Mutex Components) que por fim, caso estejam aptos a rodar naquele instante, efetuam as persistencias/acessos desejados;

**Pergunta 3:Um pouco além do MicroServiço**

* que é Micro Serviço?

São serviços contextualizados que respondem a uma ação ou processo individual, relacionado em muitos casos a uma necessidade sistêmica pontual ou para resolução de um negócio específico. São conhecidos por tratarem informações em nivel atômico e não aglomerada, ou seja, não tratam de grandes volumes de lógicas de processamento encadeados em objetos complexos, mas em grandes volumes de informações provenientes/destinadas a uma ação especifica;

* Porque devemos utilizar?

Devem ser utilizados para resolução pontual de negócios, como sistemas atômicos, ligados por uma grande rede de informações maiores e que precisam ser acessados de forma escalar, performática e organizada. Devem ser agentes de informações (dados) e não de negócios (regras);

* Quais São os Prós e Contras.

Prós: Por se tratarem de serviços atômicos, cada um dos serviços são autônomos, baseados em contextos, ou seja, não dependem nem geram dependência uns dos outros; Podem ser escalados, simplesmente por serem colocados em múltiplas instâncias do mesmo serviço em máquinas diferentes;

Contras: Precisam de um maior cuidado nas questões de segurança e acessos, já que são completamente autônomos e tem pouca dependência dos demais (componentes contextualizados); Devem ser pensados em patterns de concorrência e singularidade de instancia (singletons) para que não haja estouro de processamento por chamadas de D.O.S. por exemplo;

* Como escalaria este micro service?

Em ambientes de Cloud pela criação de novas máquinas virtuais com os mesmos ambientes de deployment, colocando a aplicação em múltiplas instâncias dentro de um CLuster (cenário 1), ou em máquinas balanceadas (cenário 2);

* Como resolveria a questão do "S.E.P" (Single entry point) ?

No caso de um Cluster, isto se resolveria pelo DNS apontando para o endereço do cluster, fazendo a associação de entrada da aplicação pelo DNS (application name); No caso de balancer, da mesma forma pelo DNS, mas implementando regras de “cookie-over-cookie” para persistência de chamadas e rotas e agregação do novo nó;

**Pergunta 4: Latência de Rede**

* que é Latência de REDE?

Latência de rede é o nome que se dá para atrasos na comunicação entre o cliente e o destino da sua chamada. Este problema pode ocorrer por causa de fatores relacionados à: a) o meio físico utilizado; b) a infraestrutura de equipamentos encontrados pelo caminho; c) pela arquitetura de rede implementada; d) configuração de dispositivos e portas de rede; e) tamanho e organização de pacotes utilizados pelo protocolo; e outros fatores que devem ser investigado, como variações de voltagem, temperatura e conectividade que podem gerar tais problemas;

* Qual estratégia deve ser implementada para diminuir essa latência?

Em primeiro caso, deve-se efetuar um estudo na camada de software, verificando disponibilidade de servidores, processamento médio, filas, recursos, etc. para melhor entendimento e proposição de solução; Caso não haja identificação da causa nos meios lógicos, deve-se implementar a mesma verificação nos meios físicos para correção de problemas;

Para fins de objetividade neste teste, analisaria os componentes de banco e de aplicação e efetuaria um tuning fino para melhorar o desempenho;

**Pergunta 5: Concorrência**

Este é um problema que precisa de maiores informações para a resolução das questões abaixo:

* Desenhar uma solução que resolva o problema de venda de estoque zerado;
* A solução deve se atentar a situações de estorno/devolução de compra;
* Deve ser performática e escalável, pensando em situações de sazonalidade;

Vou colocar minhas dúvidas e presumir algumas informações para dar a solução já que não tenho acesso a estas questões:

1. Entendo que o sistema de loja e backoffice tem banco de dados diferentes para estoque, inclusive o sistema fiscal (emissão de notas fiscais) deve estar no backoffice… Assumo isto como verdade
2. Entendo que os sistemas não têm conexão entre si, que devem estar sendo atualizado via batch… Assumo isto como verdade
3. Que os sistemas estão em localidades diferentes. Estou assumindo isto como verdade…

Assim sendo:



**Pergunta 6: Dada uma stream, encontre o primeiro caracter Vogal, após uma consoante, onde a mesma é antecessora a uma vogal e que não se repita no resto da stream. O termino da leitura da stream deve ser garantido através do método hasNext(), ou seja, retorna falso para o termino da leitura da stream. Voce tera acesso a leitura da stream através dos métodos de interface fornecidos ao termino do enunciado.**

Entendi que a solução não era para ser apenas algumas clausulas de filter no Stream, mas que voces queriam ver a capacide de logica como coder… Assim sendo, implementei a mão a logica para não retornar uma classe com 2 linhas de código.

import java.util.\*;

import java.io.\*;

public class StreamTest {

ArrayList<String> vogais = new ArrayList<String>();

ArrayList<String> consoantesRepetidas = new ArrayList<String>();

ArrayList<String> resultados = new ArrayList<String>();

String ultimaConsoante = "";

public String firstChar(Stream teste) {

vogais.add("a");vogais.add("e");vogais.add("i");vogais.add("o");vogais.add("u");

vogais.add("A");vogais.add("E");vogais.add("I");vogais.add("O");vogais.add("U");

String testString = tesString theString = IOUtils.toString(teste.getInputStream(), "UTF-8");

for(int f=0; f<testString.length();f++) {

// pego o caracter da vez

String charAtual = String.valueOf(testString.charAt(f));

//vejo se ele é uma vogal

if(vogais.contains(charAtual)) {

String charAnterior = (f>=1?String.valueOf(testString.charAt(f-1)):"");

System.out.println("e vogal "+charAtual);

// vejo se o cacter em questão tem uma consoante antes dele

if(!vogais.contains(charAnterior)) {

// verifico se existe uma repetição desta consoante

System.out.println("e consoante a anterior "+charAnterior);

if(!consoantesRepetidas.contains(charAnterior)) {

System.out.println("consoante não repete "+charAnterior);

String resultado = (f>=2?String.valueOf(testString.charAt(f-2)):"");

if(!resultado.equals(charAtual)) {

System.out.println("Resultado Encontrado: " + charAtual);

resultados.add(charAtual);

}

}

}

} else {

if(ultimaConsoante.toUpperCase().equals(charAtual.toUpperCase())) {

consoantesRepetidas.add(charAtual);

}

ultimaConsoante = charAtual;

}

}

return resultados.get[0];

}

public boolean hasNext() {

return this.resultados.hasNext();

}

public String next() {

return this.resultados.next();

}

}

**Pergunta 7: Quando você digita a URL de um site (http://www.netshoes.com.br) no browser e pressiona enter, explique da forma que preferir, o que ocorre nesse processo do protocolo HTTP entre o Client e o Server**

1. Abre-se a conexão a porta 80 (caso de HTTP) no servidor
2. Servidor reconhece o pedido e retorna status 200 - Connection Opened
3. Client manda o GET da url / para este serviço
4. O Servidor HTTP buca a pagina home e retorna status 200 - OK
5. Client inicia processo de recebimento da chamada
6. Servidor envia conteudo
7. Servidor manda fim de conteudo
8. Client encerra a conexão