Universidade Estácio de Sá
Campus Santa Cruz

Relatório Discente de Acompanhamento

Nome: Tamires de Souza Alves
Curso: Desenvolvimento Full Stack
Disciplina: Por que não paralelizar
Número da Turma: 9001
Semestre Letivo: 3º Semestre
GitHub: https://github.com/tamiresalves024/MissaoPratica_Nivel5_Mundo3.git

Servidores e clientes baseados em Socket, com uso de Threads tanto no lado cliente quanto no lado servidor, acessando o banco de dados via JPA.

Objetivo da Prática

Criar um servidor baseado em Socket, com acesso ao banco de dados JPA, além de utilizar os recursos nativos do Java para implementação de clientes síncronos e assincronos. As Threads serão usadas tanto no servidor, para viabilizar múltiplos clientes paralelos, quanto no cliente, para implementar a resposta assincrona.

Análise e Conclusão

Como funcionam as classes Socket e ServerSocket?
.O ServerSocket aguarda por conexões entrantes e o Socket é usado para estabelecer a conexão e enviar/receber dados.

Qual a importância das portas para a conexão com servidores?
As portas são como endereços específicos em um servidor que permitem que diferentes tipos de serviços se comuniquem. Elas são essenciais para direcionar o tráfego de rede para os aplicativos corretos em um servidor. As portas ajudam na organização e no gerenciamento de múltiplos serviços em um único servidor, garantindo que os dados sejam entregues aos aplicativos corretos.

Para que servem as classes de entrada e saída ObjectInputStream e ObjectOutputStream, e por que os objetos transmitidos devem ser serializáveis?
ObjectInputStream: Converte objetos Java em bytes, permitindo que sejam transmitidos ou gravados em arquivos.
ObjectOutputStream: Faz o inverso, convertendo bytes de volta para objetos.

Os objetos transmitidos precisam ser serializáveis para serem convertidos em uma sequência de bytes e, assim, transferidos ou armazenados. A serialização é o processo de converter um objeto em uma sequência de bytes para que possa ser recriado posteriormente. Isso é necessário para garantir que os objetos possam ser reconstruídos corretamente após a transmissão ou armazenamento. A inerface ‘Serializable’ em Java é usada para indicar que um objeto pode ser serializado, permitindo que seja convertido em bytes.

Por que, mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados?
As classes de entidades JPA no cliente representam dados e não têm acesso direto ao banco de dados. O acesso ao banco é controlado pelo provedor JPA, que traduz operações em objetos para consultas SQL, mantendo o isolamento ao interagir com o banco de dados. Isso assegura que todas as operações no

banco sejam feitas de maneira controlada e segura.

Como as Threads podem ser utilizadas para o tratameno assincrono das respostas enviadas pelo servidor?
Threads podem ser usadas para lidar com respostas assíncronas do servidor, permitindo que o programa continue executando outras tarefas enquanto aguarda uma resposta. Um exemplo comum é usar uma thread separada para receber e processar as respostas do servidor enquanto a thread principal do programa continua executando outras operações. Isso ajuda a evitar bloqueios, permitindo que o programa seja mais responsivo e continue executando outras tarefas enquanto espera as respostas do servidor.

Para que serve o método invokeLater, da classe SwingUtilities?
É usado para executar uma tarefa na thread de despacho de eventos (Event Dispatch Thread – EDT) do Swing. Ele permite que seja agendado uma operação para ser realizada de forma assíncrona, garantindo que seja executada na thread apropriada para manipulação de interface gráfica, evitando problemas de concorrência e mantendo a interface responsiva.

Como os objetos são enviados e recebidos pelo Socket Java?
Pode-se enviar objetos pelo Socket usando ‘ObjectOutputStream’ e receber usando ‘ObjectInputStream’. Eles lidam com a conversão dos objetos em bytes para envio e vice-versa, permitindo a transferência de objetos pela rede.

Compare a utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java, ressaltando as características relacionadas ao bloqueio do processamento
No comportamento síncrono usando Socket Java, as operações de envio e recebimento são bloqueantes, o que significa que o programa espera até que os dados sejam enviados ou recebidos antes de continuar. Isso pode causar bloqueios no processamento, especialmente se houver atrasos na rede ou no servidor.

Já no comportamento assíncrono, pode-se implementar operações não bloqueantes usando Threads ou APIs assíncronas. Isso permite que o programa continue executando outras tarefas enquanto aguarda as operações de Socket, evitando bloqueios no processamento principal.