FACULDADE ESTÁCIO DE SÁ

CURSO: DESENVOLVIMENTO FULL STACK

3º SEMESTRE – MATRÍCULA 202302595341

Repositório GitHub - alaimalmeida/CadastroServer (github.com)

ALAIM ALMEIDA DE OLIVEIRA

Por que não paralelizar

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

Salvador – BA

2024

1. Introdução

A técnica de desenvolvimento de sistemas distribuídos baseados em sockets é amplamente utilizada para permitir a comunicação entre processos, muitas vezes em máquinas distintas. Este relatório aborda a implementação de um sistema cliente-servidor utilizando sockets e threads, com integração ao banco de dados via Java Persistence API (JPA). O projeto consiste em um servidor que valida credenciais de usuários e possibilita consultas e operações de entrada e saída de produtos, sendo toda a implementação realizada em Java.

2. Objetivo

O presente projeto visa desenvolver uma aplicação cliente-servidor utilizando sockets e threads, proporcionando comunicação síncrona e assíncrona, e integração com um banco de dados SQL por meio do JPA. O sistema deve validar usuários, listar produtos e permitir operações de movimentação de estoque. Este relatório também analisa o funcionamento das classes Socket e ServerSocket, a importância das portas de conexão, o uso de ObjectInputStream e ObjectOutputStream, e a manutenção do isolamento do banco de dados.

3. Desenvolvimento

- Funcionamento das Classes Socket e ServerSocket

As classes Socket e ServerSocket são fundamentais para a comunicação em rede em Java:

- Socket: Representa um endpoint para a comunicação entre duas máquinas. Ao criar um objeto Socket, ele tenta se conectar ao servidor especificado pelo endereço IP e número da porta. Esta classe fornece métodos para enviar e receber dados.

- ServerSocket: Atua como um listener que aguarda conexões de clientes. Quando um cliente tenta se conectar, ServerSocket aceita a conexão e retorna um novo objeto Socket para gerenciar a comunicação com o cliente.

- Importância das Portas para a Conexão com Servidores

As portas são essenciais para a conexão com servidores, pois permitem que múltiplos serviços diferentes operem simultaneamente em um único servidor. Cada porta, numerada de 0 a 65535, representa um canal de comunicação separado. A combinação de um endereço IP e um número de porta define um endpoint de rede único. Por exemplo, o servidor web

padrão utiliza a porta 80 para HTTP e 443 para HTTPS. No nosso projeto, o servidor escuta na porta 4321.

- Utilização de ObjectInputStream e ObjectOutputStream e Serialização de Objetos

ObjectInputStream e ObjectOutputStream são usados para ler e escrever objetos Java entre um cliente e um servidor. Essas classes permitem a transmissão de dados complexos como objetos Java completos, preservando seus estados.

- Serialização: Para que um objeto possa ser transmitido através de um stream, ele deve ser serializável, ou seja, a classe do objeto deve implementar a interface Serializable. A serialização transforma o objeto em um formato que pode ser enviado através de um stream, e a desserialização reconstrói o objeto no outro extremo da conexão.

- Garantia do Isolamento do Acesso ao Banco de Dados com JPA

A utilização de classes de entidades JPA no cliente não implica em acesso direto ao banco de dados a partir do cliente. No projeto, o cliente envia comandos e solicitações ao servidor, que então interage com o banco de dados. Este modelo de arquitetura cliente-servidor garante que o banco de dados só seja acessado pelo servidor, mantendo o isolamento. O servidor realiza as operações de banco de dados e envia os resultados de volta ao cliente. Este design promove segurança, controle centralizado de acesso e facilita a manutenção e a escalabilidade do sistema.

4. Estrutura do projeto

Antes de começar todo o projeto, foi criado as pastas para estruturar todo o projeto, pois é com ele que os arquivos ficam em cada pacote destinado para cada função, segue abaixo a estrutura de pastas do pacote CadastroServer:

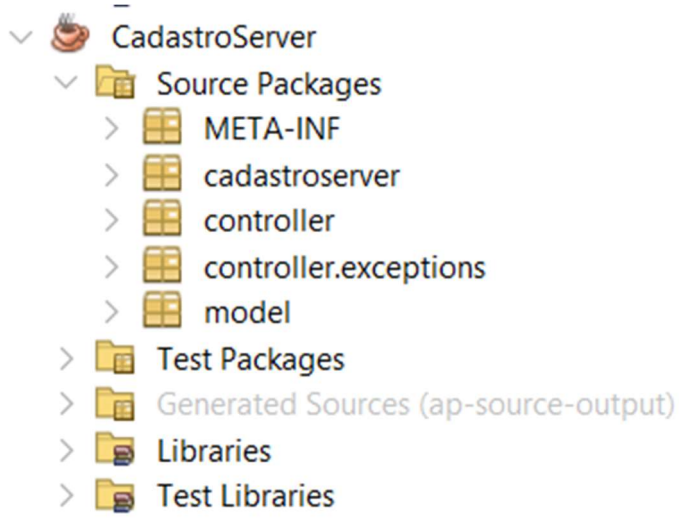

Imagem 01 – Estrutura de pasta

De acordo com os arquivos criados, temos a implementação da conexão com o cliente, onde o mesmo coloca o login e senha para conectar ao sistema. O servidor com a função de validar as credenciais, se as mesmas forem inválidas, ele se desconecta.

Logo em seguida, foi criado uma camada de persistência em CadastroServer, criando um pacote model para implementação das entidades. Utilizando a opção New..Entity Classes from DataBase para poder dar início a criação do arquivo, segue abaixo:

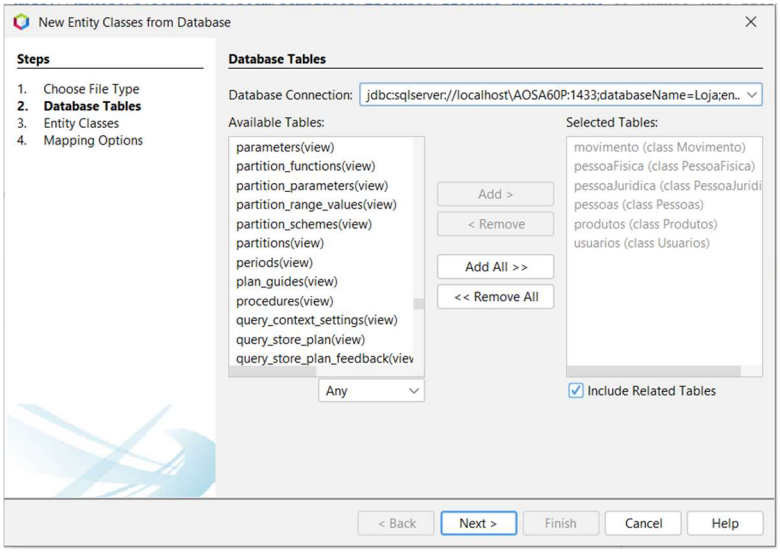

Imagem 02 – Criação de camada de persistência

Depois de criar a camada de persistência, a estrutura ficará dessa forma, com todos os arquivos criados:

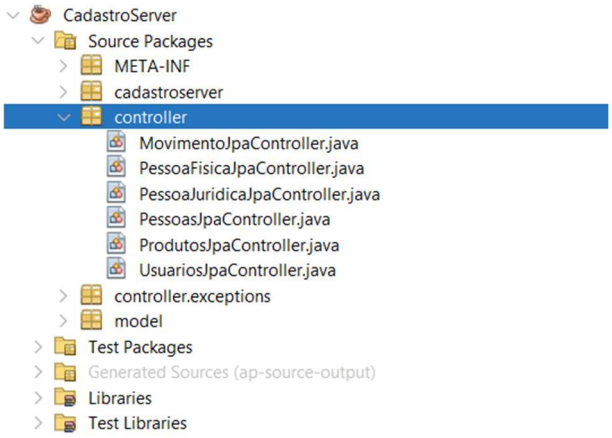

Imagem 04 – Arquivos de persistência

Logo em seguida, foi acrescentado a biblioteca Eclipse Link(JPA 2.1) e o arquivo jar do conector JDBC para o SQL Server:

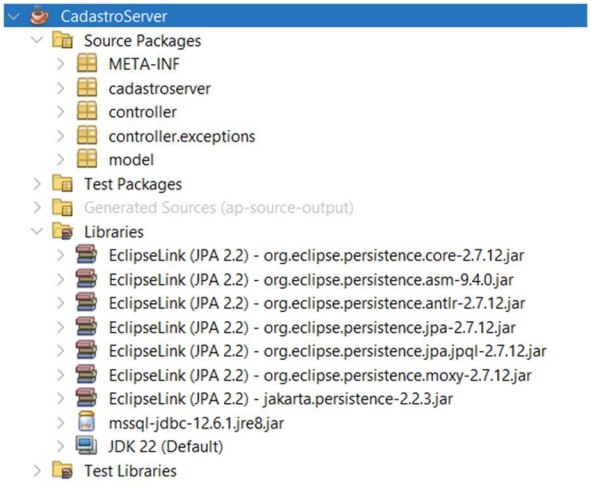

Imagem 05 – biblioteca Eclipse Link(JPA 2.1) e conector JDBC para o SQL Server

Depois de ter adicionado as bibliotecas, foi criado a camada controle, no pacote CadastroServer, utilizando o new.. JPA Controller Classes from Entity Classes e criando o pacote controller:

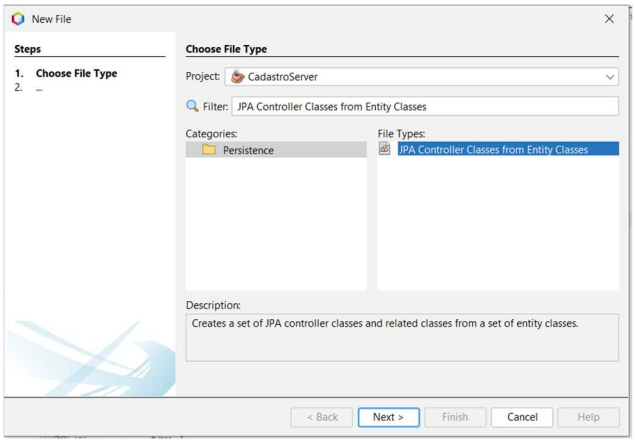

Imagem 06 – Criando arquivo JPA Controller Classes from Entity Classes

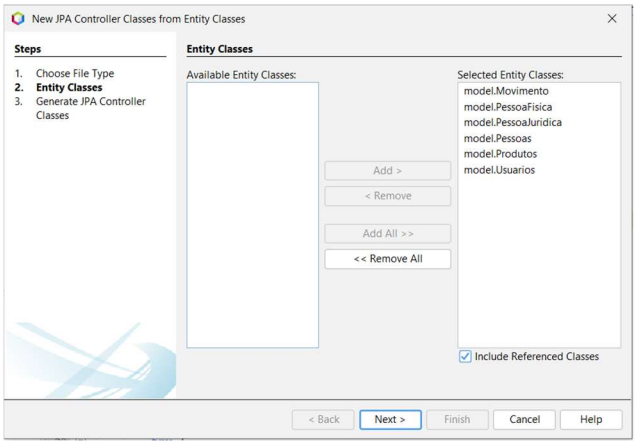

Imagem 07 – Selecionando as entidades

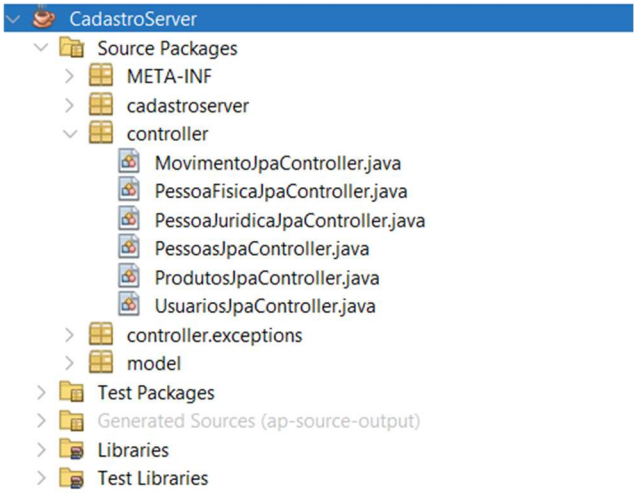

Imagem 08 – Entidades criadas

Depois de ter feito todas essas configurações, foi o momento de executar para dar continuidade com o projeto:

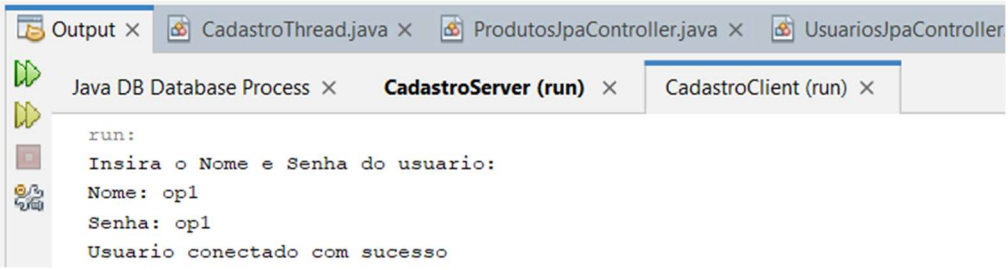

Imagem 09 – Resultado do login bem-sucedido

4. Conclusão

O desenvolvimento deste sistema cliente-servidor utilizando sockets, threads e JPA demonstrou como a comunicação em rede pode ser integrada com operações de banco de dados de maneira eficiente e segura. As classes Socket e ServerSocket permitiram a criação de uma comunicação robusta entre cliente e servidor. As portas de rede garantiram a correta separação dos serviços, e a serialização de objetos possibilitou a transmissão de dados complexos. A arquitetura cliente-servidor manteve o isolamento do banco de dados, com o servidor centralizando todas as operações, assegurando segurança e integridade dos dados.

Esta abordagem fornece uma base sólida para a construção de sistemas distribuídos, podendo ser expandida para incluir mais funcionalidades e suportar um número maior de clientes simultâneos.