Faculdade Estácio - Campus - Monte Castelo

Curso: Desenvolvimento Full Stack
Disciplina: Vamos Integrar Sistemas
Número da Turma: RPG0017
Semestre Letivo: 3
Integrante: Lucas de Oliveira dos Santos
Repositório: https://github.com/Lucasph3/mundo3-missao5

1. Título da Prática:

RPG0018 - 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.

2. Objetivos da Prática:

1. Criar servidores Java com base em Sockets.
2. Criar clientes síncronos para servidores com base em Sockets.
3. Criar clientes assíncronos para servidores com base em Sockets.
4. Utilizar Threads para implementação de processos paralelos.
5. No final do exercício, o aluno terá criado um servidor Java baseado em Socket, com acesso ao banco de dados via JPA, além de utilizar os recursos nativos do Java para implementação de clientes síncronos e assíncronos. As Threads serão usadas tanto no servidor, para viabilizar múltiplos clientes paralelos, quanto no cliente, para implementar a resposta assíncrona.

3. Códigos do roteiro:

Arquivo: CadastroServer/src/cadastroserver/CadastroServer.java

```
/*
* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template */
package cadastroserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
/**
*
* @author Mari
*/
public class CadastroServer {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException {
        int serverPort = 4321; // Porta na qual o servidor irá ouvir as conexões
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");
        ProdutoJpaController ctrl = new ProdutoJpaController(emf);
        UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);
        ServerSocket serverSocket = new ServerSocket(serverPort); // Cria um socket de servidor que
escuta na porta especificada por conexões recebidas

        System.out.println("Servidor aguardando conexões...");

        // Loop infinito para continuamente aceitar e processar conexões de clientes recebidas
        while (true) {
            // Aguarda um cliente se conectar e aceita a conexão (chamada bloqueante)
            Socket clienteSocket = serverSocket.accept();
            System.out.println("Cliente conectado: " + clienteSocket.getInetAddress());

            CadastroThread thread = new CadastroThread(ctrl, ctrlUsu, clienteSocket);
            thread.start();
            System.out.println("Aguardando nova conexão...");
        }
    }
}
```

Arquivo: CadastroServer/src/cadastroserver/CadastroThread.java

```java
/*
* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
license * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
*/
package cadastroserver;

import controller.ProdutoJpaController;
import controller.UsuarioJpaController;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import model.Usuario;

/**
*
* @author Mari
*/
public class CadastroThread extends Thread {
    private ProdutoJpaController ctrl;
    private UsuarioJpaController ctrlUsu;
    private Socket s1;
    private ObjectOutputStream out;
    private ObjectInputStream in;

    public CadastroThread(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu, Socket s1)
        { this.ctrl = ctrl;


        this.ctrlUsu = ctrlUsu;
        this.s1 = s1;
    }

    @Override
    public void run() {
        String login = "anonimo";

        try {
            out = new ObjectOutputStream(s1.getOutputStream());
            in = new ObjectInputStream(s1.getInputStream());

            System.out.println("Cliente conectado, aguardando login e senha.");

            login = (String) in.readObject();
            String senha = (String) in.readObject();

            Usuario usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                System.out.println("Usuário inválido. Login="+ login +", Senha="+ senha);
```

```
                out.writeObject("Usuário inválido.");
                return;
            }

            System.out.println("Usuário "+ login +" conectado com sucesso.");
            out.writeObject("Usuário conectado com sucesso.");

            System.out.println("Aguardando comandos...");
            String comando = (String) in.readObject();

            if (comando.equals("L")) {
                System.out.println("Comando recebido, listando produtos.");
                out.writeObject(ctrl.findProdutoEntities());
            }
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        } finally {
            close();
            System.out.println("Conexão com " + login +" finalizada.");
        }
    }

    private void close() {
        try {
            if (out != null) {
                out.close();
            }
            if (in != null) {
                in.close();
            }
            if (s1 != null) {
                s1.close();
            }
        } catch (IOException ex) {
            System.out.println("Falha ao fechar conexão.");
        }
    }
}
```

Arquivo: CadastroClient/src/cadastroclient/CadastroClient.java

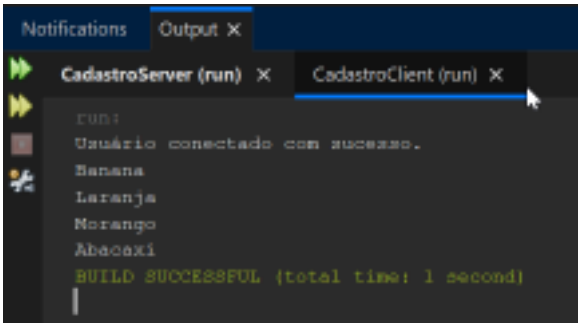
```java
/*
* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
license * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
*/
package cadastroclient;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.util.List;
import model.Produto;

/**
*
* @author Mari
*/
public class CadastroClient {

   /**
    * @param args the command line arguments
    * @throws java.io.IOException
    */
   public static void main(String[] args) throws ClassNotFoundException, IOException { String
       serverAddress = "localhost"; // Endereço do servidor (pode ser substituído pelo IP) int
       serverPort = 4321;
       Socket socket = new Socket(serverAddress, serverPort);
       ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
       ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

       // Login, passando usuário "op1"
       out.writeObject("op1");

       // Senha para o login usando "op1"
       out.writeObject("op1");

       // Lê resultado do login:
       System.out.println((String)in.readObject());

       // Lista produtos:


      out.writeObject("L");

      List<Produto> produtos = (List<Produto>) in.readObject();
      for (Produto produto : produtos) {
          System.out.println(produto.getNome());
      }

      out.close();
      in.close();
```

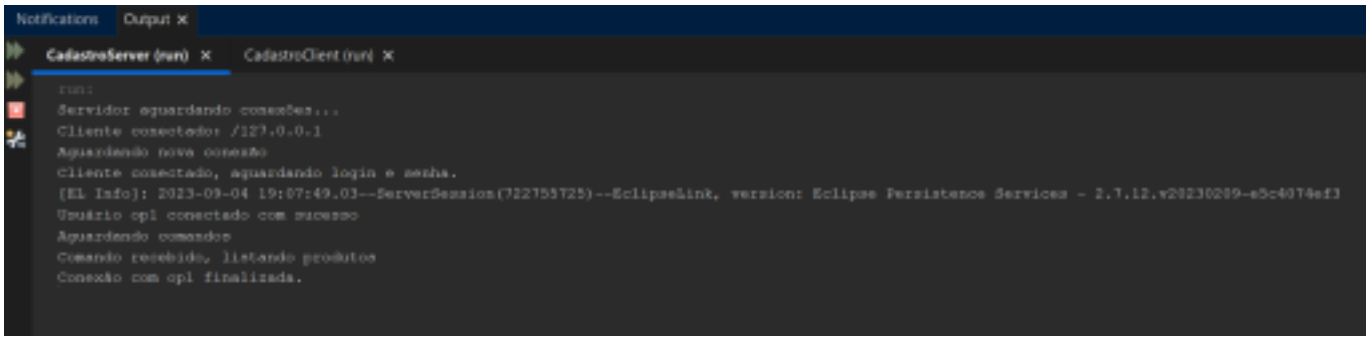
```
            socket.close();
        }
    }
```

Demais arquivos gerados encontram-se no github:

- CadastroServer/src/controller
- CadastroServer/src/model
- CadastroClient/src/model

4. Resultados da execução dos códigos





5. Análise e Conclusão

a) Como funcionam as classes Socket e ServerSocket?

Resposta: A classe ServerSocket é usada para criar um servidor que escuta as solicitações de conexão dos clientes. A classe Socket é usada para criar um soquete do lado do cliente que se conecta ao servidor.

b) Qual a importância das portas para a conexão com servidores?

Resposta: As portas são importantes pois elas permitem que o cliente e o servidor se comuniquem entre si criando um canal identificado, evitando conflitos.

c) Para que servem as classes de entrada e saída ObjectInputStream e ObjectOutputStream, e por que os objetos transmitidos devem ser serializáveis?

Resposta: As classes ObjectInputStream e ObjectOutputStream servem para serializar objetos em convertendo-os em sequências de bytes que podem ser armazenadas ou transmitidas pela rede. Os objetos transmitidos através de soquetes devem ser serializáveis porque a rede só pode transportar dados binários. Assim sendo os objetos são convertidos em sequências de bytes transmitidos e recriados como um objeto idêntico do outro lado da conexão.

d) Por que, mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados?

Resposta: As classes de entidades JPA no cliente não possuem código que acesse o banco de dados, apenas os campos e o modelo básico da representação desses objetos, necessários apenas para a serialização e deserialização dos objetos. A lógica de acesso ao banco de dados fica à cargo das classes Controllers, que neste caso, existem apenas do lado do Servidor, garantindo assim o isolamento do acesso ao banco de dados.