Campus: Polo n° 3366 Centro - Garopaba - SC
Curso: Tecnólogo em Desenvolvimento Full Stack
Disciplina: Por que não paralelizar?
Turma: 9001
Nome do Aluno: Luiz Evaldo Pereira
Endereço:

1. **Título**

Desenvolvimento de um Sistema Cliente-Servidor em Java com Autenticação e Listagem de Produtos utilizando Sockets, Threads e JPA

2. **Objetivo**

O objetivo do presente trabalho é construir um sistema completo que permita autenticar clientes, atender comandos (como listagem de produtos) e organizar o código de forma modular e eficiente, aplicando conceitos essenciais de redes, persistência e programação concorrente em Java.

3. **Códigos**

3.1 Java

```
public Usuario findUsuario(String login, String senha) {
   try {
        return em.createQuery("SELECT u FROM Usuario u WHERE u.login = :login AND u.senha = :senha", Usuario.class)
                .setParameter("login", login)
                .setParameter("senha", senha)
                .getSingleResult();
   } catch (NoResultException e) {
       return null;
   }
}
```

3.2CadastroThread.java

```
package cadastroserver;

import controller.*;
import model.Usuario;
import model.Produto;
import java.net.Socket;
import java.io.*;
import java.util.List;

public class CadastroThread extends Thread {
   private ProdutoJpaController ctrl;
   private UsuarioJpaController ctrlUsu;
   private Socket s1;
```

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

    @Override
    public void run() {
        try(ObjectOutputStream out = new ObjectOutputStream(s1.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(s1.getInputStream())) {

            // Recebe login e senha
            String login = (String) in.readObject();
            String senha = (String) in.readObject();

            // Valida usuário
            Usuario usuario = ctrlUsu.findUsuario(login, senha);
            if (usuario == null) {
                s1.close();
                return;
            }

            // Ciclo de resposta
            while (true) {
                String comando = (String) in.readObject();

                if ("L".equalsIgnoreCase(comando)) {
                    List<Produto> produtos = ctrl.findProdutoEntities();
                    out.writeObject(produtos);
                    out.flush();
                } else {
                    break; // Sai do loop em outros casos (pode ajustar conforme necessidade)
                }
            }

            s1.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3.3 ExecuçãoPrincipal.java

```
package cadastroserver;
```

```
import controller.*;
import javax.persistence.*;
import java.net.ServerSocket;
import java.net.Socket;

public class CadastroServer {
    public static void main(String[] args) {
        try {
                                                                EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroPU");
            ProdutoJpaController ctrl = new ProdutoJpaController(emf);
            UsuarioJpaController ctrlUsu = new UsuarioJpaController(emf);

            ServerSocket server = new ServerSocket(4321);
            System.out.println("Servidor aguardando conexão na porta 4321...");

            while (true) {
                Socket clienteSocket = server.accept();
                System.out.println("Cliente conectado: " + clienteSocket.getInetAddress());
                CadastroThread thread = new CadastroThread(ctrl, ctrlUsu, clienteSocket);
                thread.start();
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3.4 CadastroClient.java

```
package cadastroclient;

import model.Produto;
import java.net.Socket;
import java.io.*;
import java.util.List;

public class CadastroClient {
    public static void main(String[] args) {
        try (Socket socket = new Socket("localhost", 4321);
             ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
             ObjectInputStream in = new ObjectInputStream(socket.getInputStream())) {

            // Envia login e senha (exemplo: op1/op1)
            out.writeObject("op1");
            out.writeObject("op1");
```

```
            out.flush();

            // Envia comando L para listar produtos
            out.writeObject("L");
            out.flush();

            // Recebe lista de produtos
            List<Produto> produtos = (List<Produto>) in.readObject();
            System.out.println("Produtos recebidos:");
            for (Produto p : produtos) {
                System.out.println("- " + p.getNome());
            }

            // Fecha conexão automaticamente usando try-with-resources

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4. **Análise e conclusão:**

Como funcionam as classes Socket e ServerSocket?

ServerSocket: fica aguardando conexões na porta designada; quando um cliente conecta, gera um objeto `Socket` representando a conexão aberta.

- **Socket**: representa a conexão cliente-servidor em ambos os lados, permitindo o envio e recebimento de dados.

Qual a importância das portas para a conexão com servidores?

Portas identificam pontos de entrada de serviços em uma máquina, permitindo que múltiplas aplicações utilizem a rede simultaneamente.
Sem portas, não seria possível direcionar pacotes a aplicações específicas.

Para que servem ObjectInputStream e ObjectOutputStream? Por que os objetos transmitidos devem ser serializáveis?

Servem para enviar e receber objetos Java através de streams ligados ao socket. Objetos devem ser serializáveis para que possam ser convertidos em sequências de bytes e reconstruídos no outro lado.

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

O cliente apenas recebe objetos serializados, sem fazer conexão direta ao banco. O servidor é responsável pela consulta e acesso a dados via JPA, garantindo o controle do acesso e isolamento.