Faculdade Estácio

Campus Belford Roxo - RJ

Desenvolvimento Full Stack

Disciplina - Por Que Não Paralelizar?

Turma 2023.2

Semestre - 3

Cristian da Silva de Macena

Repositório: https://github.com/CristianS34/CadastroServer.git

# Código

```
package cadastroserver;

import cadastroserver.controller.MovimentoJpaController;
import cadastroserver.controller.PessoaJpaController;
```

```
import cadastroserver.controller.ProdutoJpaController;
import cadastroserver.controller.UsuarioJpaController;
import cadastroserver.model.Produto;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.List;
import java.util.logging.Logger;
import java.util.logging.Level;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class CadastroServer {

   private final int PORT = 4321;

   public CadastroServer() {

   }

   private void run() {
       try (ServerSocket serverSocket = new ServerSocket(PORT)) {
           System.out.println("======== SERVIDOR CONECTADO - PORTA
" + PORT + " ========");
           // Inicializa controladores
           EntityManagerFactory emf =
Persistence.createEntityManagerFactory("CadastroServerPU");
           ProdutoJpaController produtoController = new
ProdutoJpaController(emf);
           MovimentoJpaController movimentoController = new
MovimentoJpaController(emf);
           PessoaJpaController pessoaController = new
PessoaJpaController(emf);
           UsuarioJpaController usuarioController = new
UsuarioJpaController(emf);
           while (true) {
               System.out.println("Aguardando conexao de
cliente...");
               Socket socket = serverSocket.accept();
               System.out.println("Cliente conectado.");
               ClientHandler clientHandler = new
ClientHandler(socket, produtoController,
                   movimentoController, pessoaController,
usuarioController);
               Thread thread = new Thread(clientHandler);
```

```
                thread.start();
            }
        } catch (IOException e) {
Logger.getLogger(CadastroServer.class.getName()).log(Level.SEVERE,
null, e);
        }
    }

    public static void main(String[] args) {
        new CadastroServer().run();
    }

    private class ClientHandler implements Runnable {
        private final Socket socket;
        private final ProdutoJpaController produtoController;
        private final UsuarioJpaController usuarioController;

        public ClientHandler(Socket socket, ProdutoJpaController
produtoController,
            MovimentoJpaController movimentoController,
PessoaJpaController pessoaController,
            UsuarioJpaController usuarioController) {
            this.socket = socket;
            this.produtoController = produtoController;
            this.usuarioController = usuarioController;
        }

        @Override
        public void run() {
            try (
                BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                PrintWriter out = new
PrintWriter(socket.getOutputStream(), true)
            ) {
                // Autenticacao
                String username = in.readLine();
                String password = in.readLine();
                if (validateCredentials(username, password)) {
                    out.println("Autenticacao bem-sucedida.
Aguardando comandos...");
                    boolean outerSign = false;
                    while (!outerSign) {
                        String command = in.readLine();
                        if (command != null) {
                            switch (command) {
```

```
                                case "L": sendProductList(out);
break; // Enviar conjunto de produtos do banco de dados
                                case "S": outerSign = true; break;
// Comando para sair
                                default: break;
                            }
                        }
                    }
                } else {
                    try (socket) {
                        out.println("Credenciais invalidas. Conexao
encerrada.");
                    }
                }
            } catch (IOException e) {

Logger.getLogger(ClientHandler.class.getName()).log(Level.SEVERE,
null, e);
            } catch (Exception e) {

Logger.getLogger(ClientHandler.class.getName()).log(Level.SEVERE,
null, e);
            }
        }

        private boolean validateCredentials(String username, String
password) {
            return usuarioController.validarUsuario(username,
password) != null;
        }

        private void sendProductList(PrintWriter out) {
            List<Produto> productList =
produtoController.findProdutoEntities();
            out.println("Conjunto de produtos disponiveis:");
            for (Produto product : productList) {
                out.println(product.getNome());
            }
            out.println();
        }

    }

}
```

# Servidor Completo e Cliente Assíncrono

Threads permitem o tratamento assíncrono das respostas do servidor, garantindo que o cliente continue executando outras tarefas enquanto aguarda a resposta.

# Para que serve o método invokeLater, da classe SwingUtilities?

O método invokeLater da classe SwingUtilities é usado para garantir que um pedaço de código seja executado na Event Dispatch Thread (EDT), que é a thread responsável por gerenciar a interface gráfica no Swing. Isso é essencial para manter a responsividade da interface e evitar problemas de concorrência.

# Como os objetos são enviados e recebidos pelo Socket Java?

Os objetos são enviados e recebidos pelo Socket em Java utilizando fluxos de entrada e saída, especificamente as classes ObjectInputStream e ObjectOutputStream. Para isso, os objetos devem implementar a interface Serializable, permitindo sua conversão em um fluxo de bytes.

# Compare a utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java, ressaltando as características relacionadas ao bloqueio do processamento.

O comportamento assíncrono em clientes com Socket Java evita bloqueios e melhora a responsividade, enquanto o síncrono é mais simples, mas pode causar ineficiências.